ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΣΥΓΧΡΟΝΟΥ ΗΛΕΚΤΡΟΝΙΚΟΥ ΚΑΤΑΣΤΗΜΑΤΟΣ ΚΑΙ ΕΛΕΓΧΟΣ ΑΞΙΟΠΙΣΤΙΑΣ ΛΟΓΙΣΜΙΚΟΥ

Κοσμάς Παπαδόπουλος

Α.Μ. 711171179

Επιβλέπων Καθηγητής: Ακριβή Κρούσκα

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΗΞΗ ΣΥΓΧΡΟΝΟΥ ΗΛΕΚΤΡΟΝΙΚΟΥ ΚΑΤΑΣΤΗΜΑΤΟΣ ΚΑΙ ΕΛΕΓΧΟΣ ΑΞΙΟΠΙΣΤΙΑΣ ΛΟΓΙΣΜΙΚΟΥ

ΚΟΣΜΑΣ ΠΑΠΑΔΟΠΟΥΛΟΣ

Α.Μ. 711171179

Εισηγητής:

Ακριβή Κρούσκα, ΕΔΙΠ

Akrivi Krouska
Digitally signed by Akrivi Krouska
Date: 2024.10.09 14:26:41 +03'00'

Εξεταστική Επιτροπή:

Χρήστος Τρούσσας, Επ. Καθ.

Christos Troussas
Digitally signed by Christos Troussas
Date: 2024.10.09 17:00:32 +03'00'

Παναγιώτα Τσελέντη, ΕΔΙΠ

Panagiota Tselenti
Digitally signed by Panagiota Tselenti
Date: 2024.10.09 18:03:46 +03'00'

Ημερομηνία εξέτασης 30/9/2024

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Ο υπογράφων ΚΟΣΜΑΣ ΠΑΠΑΔΟΠΟΥΛΟΣ του ΔΗΜΗΤΡΗ, με αριθμό μητρώου 711171179 φοιτητής του Τμήματος ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ της Σχολής του Πανεπιστημίου Δυτικής Αττικής, δηλώνω υπεύθυνα ότι:

Βεβαιώνω ότι είμαι συγγραφέας αυτής της Διπλωματικής εργασίας και κάθε βοήθεια την οποία είχα για την προετοιμασία της, είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, οι όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών ή λέξεων, είτε ακριβώς είτε παραφρασμένες, αναφέρονται στο σύνολό τους, με πλήρη αναφορά στους συγγραφείς, τον εκδοτικό οίκο ή το περιοδικό, συμπεριλαμβανομένων και των πηγών που ενδεχομένως χρησιμοποιήθηκαν από το διαδίκτυο. Επίσης, βεβαιώνω ότι αυτή η εργασία έχει συγγραφεί από μένα αποκλειστικά και αποτελεί προϊόν πνευματικής ιδιοκτησίας τόσο δικής μου, όσο και του Ιδρύματος. Παράβαση της ανωτέρω ακαδημαϊκής μου ευθύνης αποτελεί ουσιώδη λόγο για την ανάκληση του πτυχίου μου».

«Βεβαιώνω ότι είμαι συγγραφέας της παρούσας διπλωματικής εργασίας και ότι έχω αναφέρει ή παραπέμψει σε αυτή, ρητά και συγκεκριμένα, όλες τις πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, προτάσεων ή λέξεων, είτε αυτές μεταφέρονται επακριβώς (στο πρωτότυπο ή μεταφρασμένες) είτε παραφρασμένες. Επίσης βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά ειδικά για την συγκεκριμένη διπλωματική εργασία»

Ο/Η Δηλών/ούσα

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να εκφράσω τις ειλικρινείς μου ευχαριστίες σε όλους εκείνους που συνέβαλαν στην ολοκλήρωση αυτής της πτυχιακής εργασίας. Πρώτα απ' όλα, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου, Ακριβή Κρούσκα, για την πολύτιμη καθοδήγηση, την αμέριστη υποστήριξη και την εμπιστοσύνη που μου έδειξε καθ' όλη τη διάρκεια αυτής της προσπάθειας. Η καθοδήγησή αυτή υπήρξε καθοριστική για την ανάπτυξη και την ολοκλήρωση της εργασίας αυτής.

Επίσης, θέλω να ευχαριστήσω θερμά τους όλους τους καθηγητές του τμήματος για τις γνώσεις και τις δεξιότητες που μου προσέφεραν καθ' όλη τη διάρκεια των σπουδών μου. Η αφοσίωσή τους στην εκπαίδευση και η δέσμευσή τους για την ακαδημαϊκή πρόοδο μου ήταν καθοριστικοί παράγοντες στην ανάπτυξη των ικανοτήτων μου.

Τέλος, θα ήθελα ιδιαίτερα να ευχαριστήσω την οικογένειά μου για την αδιάκοπη υποστήριξη, την υπομονή και την ενθάρρυνσή τους καθ' όλη τη διάρκεια των σπουδών μου.

ΠΕΡΙΛΗΨΗ

Η εργασία αναλύει και δημιουργεί ένα ηλεκτρονικό κατάστημα, περιέχει μια μικρή περιγραφή για το ηλεκτρονικό εμπόριο και βοηθά στη μελέτη των θεμελιωδών πτυχών του ReactJS και Tailwind CSS. Η εφαρμογή θα επιτρέπει στους χρήστες να αγοράζουν είδη που σχετίζονται με τις ανάγκες τους. Επίσης παρέχει χρήσιμα εργαλεία για να βοηθήσουν τους ιδιοκτήτες να διαχειρίζονται το ηλεκτρονικό τους κατάστημα μέσω της εφαρμογής διαχειριστή προσφέροντας την δυνατότητα των άμεσων δυναμικών εκχωρήσεις προϊόντων και κατηγοριών καθώς και με τον χειρισμό παραγγελιών και πληρωμών.

Άλλοι στόχοι αποτελούν οι αυστηροί έλεγχοι που αφορούν την διεπαφή του χρήστη με την εφαρμογή, καθώς και την σωστή λειτουργία σε διάφορα μέρη του κώδικα όπου χρίζουν ζωτικής σημασίας για την ομαλή λειτουργία του συστήματος. Εξετάζει επιπλέον την αποτελεσματικότητα των διαδικασιών διασφάλισης ποιότητας και παρέχει συστάσεις για την περαιτέρω βελτίωση της απόδοσης και της αξιοπιστίας του ηλεκτρονικού καταστήματος.

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: ΑΝΑΠΤΥΞΗ ΚΑΙ ΕΛΕΓΧΟΣ ΠΟΙΟΤΗΤΑΣ ΕΦΑΡΜΟΓΗΣ

ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: React JS, Tailwind CSS, admin, προϊόντα, προσθήκη, κώδικας, δημιουργία, component, hook, state, store, modal, firebase, storage, quality assurance, testing, library, unit, ui, δεδομένα, mock

ΠΕΡΙΕΧΟΜΕΝΑ

ΕΙΣΑΓΩΓΗ

Η εξέλιξη του ηλεκτρονικού εμπορίου έχει αναδείξει την ανάγκη για αξιόπιστα και αποδοτικά συστήματα ηλεκτρονικών καταστημάτων, τα οποία εξασφαλίζουν άριστη εμπειρία χρήστη και λειτουργική σταθερότητα. Η παρούσα πτυχιακή εργασία εστιάζει στην ανάπτυξη ενός σύγχρονου συστήματος ηλεκτρονικών αγορών, δίνοντας ιδιαίτερη έμφαση στη διασφάλιση της ποιότητας και της απόδοσης του λογισμικού.

Το σύστημα αναπτύχθηκε χρησιμοποιώντας τις τεχνολογίες React.js και Tailwind CSS, που προσφέρουν ευελιξία και ταχύτητα στην ανάπτυξη εφαρμογών με μοντέρνο και αποκριτικό σχεδιασμό. Ένα από τα βασικά χαρακτηριστικά του συστήματος είναι η δυναμική καταχώρηση κατηγοριών και προϊόντων, η οποία επιτρέπει την εύκολη και γρήγορη διαχείριση του περιεχομένου του καταστήματος μέσω ενός ειδικά σχεδιασμένου admin panel. Το ηλεκτρονικό κατάστημα e-shop, αποτελεί τη βιτρίνα του συστήματος και παρέχει μια ολοκληρωμένη εμπειρία αγορών στους χρήστες. Για την αποθήκευση των δεδομένων χρησιμοποιήθηκε το Firebase ως cloud storage, εξασφαλίζοντας την ασφαλή και αξιόπιστη διαχείριση των πληροφοριών. Οι πληρωμές πραγματοποιούνται μέσω της πλατφόρμας Stripe, η οποία προσφέρει αξιόπιστες και ασφαλείς συναλλαγές.

Η διασφάλιση της ποιότητας και της απόδοσης του συστήματος ήταν επίσης κεντρικός στόχος αυτής της εργασίας. Για το λόγο αυτό, διεξήχθησαν εκτενείς έλεγχοι λειτουργιών (UI tests) και μονάδων (Unit tests), οι οποίοι συνέβαλαν στην αναγνώριση και διόρθωση πιθανών προβλημάτων, εξασφαλίζοντας την υψηλή ποιότητα του τελικού προϊόντος.

Η εργασία αυτή αποσκοπεί στην αναλυτική παρουσίαση της διαδικασίας ανάπτυξης του συστήματος, των τεχνολογιών και εργαλείων που χρησιμοποιήθηκαν, καθώς και των μεθόδων διασφάλισης ποιότητας που εφαρμόστηκαν. Παράλληλα, καταγράφει και αναλύει τους κώδικες που υιοθετήθηκαν, παρέχοντας πολύτιμα συμπεράσματα για μελλοντικές βελτιώσεις και επεκτάσεις του συστήματος.

ΚΕΦΑΛΑΙΟ 1: ΗΛΕΚΤΡΟΝΙΚΟ ΕΜΠΟΡΙΟ

1.1 Η ΑΝΑΠΤΥΞΗ ΤΟΥ ΗΛΕΚΤΡΟΝΙΚΟΥ ΕΜΠΟΡΙΟΥ

1.1.1. Τι είναι το ηλεκτρονικό εμπόριο;

Το ηλεκτρονικό εμπόριο είναι η διαδικασία αγοράς και πώλησης αγαθών και υπηρεσιών μέσω του Διαδικτύου. Περιλαμβάνει την ανταλλαγή προϊόντων ή υπηρεσιών μεταξύ επιχειρήσεων, καταναλωτών ή και των δύο. Οι επιχειρήσεις ηλεκτρονικού εμπορίου διευκολύνονται μέσω πλατφορμών όπως ιστότοποι, εφαρμογές για κινητά ή διαδικτυακές αγορές.

Όπου κάποτε το ηλεκτρονικό εμπόριο περιέγραφε μια απλή διαδικασία - μια αγορά καταναλωτή από έναν ιστότοπο ηλεκτρονικού εμπορίου, για παράδειγμα - ο όρος έχει επεκταθεί καθώς οι τεχνολογίες έχουν προχωρήσει. Σήμερα, το ηλεκτρονικό εμπόριο μπορεί να αναφέρεται στο εμπόριο μεταξύ επιχειρήσεων ή σε εσωτερικές επιχειρηματικές συναλλαγές. Εναλλακτικές ονομασίες για το ηλεκτρονικό εμπόριο είναι e-shop, e-store, internet shop, διαδικτυακό κατάστημα, ηλεκτρονικό κατάστημα και εικονικό κατάστημα.

1.1.2. Η Εξέλιξη του Διαδικτυακού Εμπορίου

Το ηλεκτρονικό εμπόριο ανέδειξε μια νέα εποχή στον κόσμο των αγορών, ανοίγοντας νέους ορίζοντες και αλλάζοντας τον τρόπο που οι άνθρωποι αντιλαμβάνονται και προσεγγίζουν το εμπόριο. Η ανάπτυξη νέου λογισμικού και άλλων τεχνολογιών στις αρχές της δεκαετίας του 1990 μετέτρεψε το Διαδίκτυο σε διαφημιστικό μέσο που έχει μεταμορφώσει τις επιχειρήσεις παγκοσμίως. Η ανάπτυξη του ηλεκτρονικού εμπορίου άλλαξε όλες τις επιχειρήσεις σε όλο τον κόσμο. Στις μέρες μας, όλο και περισσότεροι άνθρωποι θέλουν να απλοποιήσουν τη ζωή τους και το ηλεκτρονικό εμπόριο είναι η λύση. Η μετάβαση από το παραδοσιακό εμπόριο στις ψηφιακές πλατφόρμες ήταν μια σταδιακή διαδικασία, η οποία ενισχύθηκε από τη συνεχή βελτίωση των τεχνολογιών πληροφορικής και την αυξημένη προσβασιμότητα στο Διαδίκτυο. Αυτό άνοιξε νέες ευκαιρίες για τους καταναλωτές, προσφέροντας τους τη δυνατότητα να κάνουν αγορές από την άνεση του σπιτιού τους και να έχουν πρόσβαση σε ένα ευρύ φάσμα προϊόντων και υπηρεσιών.

Η εξέλιξη του παγκόσμιου μάρκετινγκ πήρε χρόνο, είναι μια εξελικτική διαδικασία και ισχύει για τις περισσότερες εταιρείες. Το διεθνές μάρκετινγκ έχει εντείνει πολλές πτυχές στην καθημερινή ζωή των πελατών. Για να έχει την ευκαιρία να πετύχει μια εταιρεία πρέπει να ανταποκρίνεται στις απαιτήσεις και τις ανάγκες όχι μόνο της τοπικής αγοράς αλλά και των παγκόσμιων πελατών. Επομένως, Οι δεξιότητες διεθνούς μάρκετινγκ είναι απαραίτητες για τις περισσότερες εταιρείες στον κόσμο. Η διεθνοποιημένη αγορά έχει αλλάξει σε πολύ σύντομο χρονικό διάστημα λόγω αλλαγών στα εμπορικά πρότυπα, τις πρακτικές και τις τεχνικές συναλλαγών. Αυτές οι αλλαγές έχουν επιβληθεί από τις νέες τεχνολογίες και την εξέλιξη των οικονομικών σχέσεων. Τον 20ο αιώνα, η παγκοσμιοποίηση κατέλαβε την παγκόσμια οικονομία, δημιουργώντας νέες έννοιες και ιδέες έχει σχεδιαστεί για να ελέγχει και να

εξηγεί τις συνθήκες που υπάρχουν τώρα για τους επενδυτές. Η παγκόσμια αγορά είναι η κύρια κινητήρια δύναμη δύναμη πίσω από τη διεθνή χρηματοδότηση και το εμπόριο. Ασχολείται με έννοιες τόσο σε μικροοικονομική όσο και σε μακροοικονομική κλίμακα.

1.2. ΙΣΤΟΣΕΛΙΔΕΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΟ ΕΜΠΟΡΙΟ

Οι ιστότοποι είναι αποτελεσματικοί στις πράξεις των επιχειρηματικών εταιρειών για ενεργό προβολή. Οι ιστότοποι είναι πύλες και πυλώνες για τη δημιουργία μιας επιχείρησης μάρκες, προϊόντα και υπηρεσίες κατασκευαστών. Μια εταιρεία που κάνει Δεν ενδιαφέρονται για τις ιστοσελίδες της τελικά θα είναι χρεοκοπία στην επιχείρηση. Οι ιστότοποι που δεν είναι προσαρμοσμένοι είναι σαν μια παλιά και ξεθωριασμένη βιτρίνα μια παραδοσιακή επιχείρηση. Ως εκ τούτου, οι εταιρείες θα πρέπει να έχουν μια ισχυρή ιστορικό και ακριβή προγραμματισμό να ενημερώνουν τακτικά τον ιστότοπό τους ενισχύουν την επιχειρηματική τους στρατηγική. Για πολλούς χρήστες, ο ιστότοπος αναζήτησης Η μηχανή αναζήτησης είναι το σημείο εισόδου στις χρήσεις του διαδικτύου καθώς και α πύλη εισόδου στον κόσμο του ηλεκτρονικού εμπορίου. Για πολλές αλλαγές στο στον κόσμο γύρω μας, κάθε οργανισμός ή οργανισμός πρέπει να συντονίζεται με τις αλλαγές και να ενισχύσει την αποδοτικότητα της εργασίας του.)διαδίκτυο ειδικά οι ιστότοποι μπορούν να αποτελέσουν κατάλληλο μέσο για την επίτευξη αυτού του στόχου.

1.3. ΕΠΙΧΕΙΡΗΜΑΤΙΚΆ ΜΟΝΤΈΛΑ ΗΛΕΚΤΡΟΝΙΚΟΎ ΕΜΠΟΡΊΟΥ

Τα μοντέλα ηλεκτρονικού εμπορίου ποικίλλουν ευρέως και περιλαμβάνουν πολλούς τύπους πωλήσεων. Ακολουθούν οι διαφορετικοί τύποι επιχειρήσεων ηλεκτρονικού εμπορίου:

- Business-to-business (B2B)

Το ηλεκτρονικό εμπόριο B2B αναφέρεται όταν μια εταιρεία αγοράζει αγαθά ή υπηρεσίες ηλεκτρονικά από άλλη επιχείρηση. Μερικά παραδείγματα περιλαμβάνουν ένα εστιατόριο που αγοράζει μια παγομηχανή ή μια δικηγορική εταιρεία που αγοράζει λογιστικό λογισμικό. Το επιχειρηματικό λογισμικό όπως οι πλατφόρμες διαχείρισης σχέσεων πελατών (CRM) και οι εταιρείες επεξεργασίας πληρωμών θεωρούνται επίσης B2B. Οι διαδικτυακές πωλήσεις B2B τείνουν να είναι πιο περίπλοκες από άλλες μορφές ηλεκτρονικού εμπορίου, επειδή βασίζονται σε μεγάλους καταλόγους περίπλοκων προϊόντων για την πώληση.

- Business-to-consumer (B2C)

Το διαδικτυακό λιανικό εμπόριο B2C συμβαίνει όταν ένας καταναλωτής αγοράζει ένα προϊόν μέσω Διαδικτύου για δική του χρήση. Αν και το ηλεκτρονικό εμπόριο B2C φαίνεται πιο σημαντικό, είναι μόνο το μισό περίπου του μεγέθους της παγκόσμιας αγοράς ηλεκτρονικού εμπορίου B2B.

- Καταναλωτής σε καταναλωτή (C2C)

Το C2C λειτουργεί σαν μια ψηφιακή πώληση αυλής ή μια διαδικτυακή δημοπρασία στην οποία μεμονωμένα άτομα πωλούν αγαθά ο ένας στον άλλο. Αυτά μπορεί να είναι προϊόντα που κατασκευάζουν, όπως χειροτεχνήματα ή έργα τέχνης, ή μεταχειρισμένα αντικείμενα που κατέχουν και θέλουν να πουλήσουν.

- Καταναλωτή σε επιχείρηση (C2B)

Όταν ένας καταναλωτής δημιουργεί αξία για μια επιχείρηση, αυτό είναι το εμπόριο C2B. Η δημιουργία αξίας μπορεί να λάβει πολλές μορφές. Για παράδειγμα, το C2B μπορεί να είναι τόσο απλό όσο ένας πελάτης που αφήνει μια θετική κριτική για μια επιχείρηση ή έναν ιστότοπο φωτογραφιών στοκ αγοράζοντας εικόνες από ελεύθερους επαγγελματίες. Επιπλέον, οι επιχειρήσεις που πωλούν μεταχειρισμένα προϊόντα αγοράζουν μερικές φορές εμπορεύματα από ιδιώτες στο διαδίκτυο.

- Business-to-Government (B2G)

Αυτές μερικές φορές ονομάζονται πωλήσεις από επιχείρηση σε διοίκηση (B2A). Συμβαίνουν όταν μια ιδιωτική επιχείρηση ανταλλάσσει αγαθά ή υπηρεσίες με μια δημόσια υπηρεσία. Συνήθως μια επιχείρηση συνάπτει συμβάσεις με έναν δημόσιο οργανισμό για την εκτέλεση μιας εξουσιοδοτημένης υπηρεσίας. Για παράδειγμα, μια εταιρεία φύλαξης μπορεί να υποβάλει προσφορά στο Διαδίκτυο για μια σύμβαση καθαρισμού του δικαστηρίου της κομητείας ή μια εταιρεία πληροφορικής μπορεί να απαντήσει σε μια πρόταση διαχείρισης του υλικού υπολογιστή μιας πόλης.

- Καταναλωτής προς κυβέρνηση (C2G)

Έχετε πληρώσει ποτέ εισιτήριο στάθμευσης μέσω Διαδικτύου; Τότε έχετε βιώσει το C2G. Αυτό το μοντέλο περιλαμβάνει επίσης την πληρωμή φόρων μέσω Διαδικτύου και την αγορά αγαθών από την ηλεκτρονική δημοπρασία μιας κρατικής υπηρεσίας. Κάθε φορά που παραδίδετε χρήματα σε μια δημόσια υπηρεσία χρησιμοποιώντας το Διαδίκτυο, συμμετέχετε στο ηλεκτρονικό εμπόριο C2G.

1.4. ΟΦΈΛΗ ΗΛΕΚΤΡΟΝΙΚΟΎ ΕΜΠΟΡΊΟΥ

Γενικά, ο απώτερος σκοπός ενός διαδικτυακού τόπου, πρέπει να αποτελεί η διαδικασία κατά την οποία θα αποφασιστούν οι τρόποι και τα μέσα που θα τεθούν σε ισχύ, ούτως ώστε να προσελκύσει και να διατηρηθεί το ενδιαφέρον του υποψηφίου πελάτη. Τα θετικά στοιχεία του ηλεκτρονικού εμπορίου είναι πάρα πολλά, αφενός για τις επιχειρήσεις και αφετέρου για τους καταναλωτές.

Χαμηλό οικονομικό κόστος

Ένα από τα οφέλη του ηλεκτρονικού εμπορίου είναι ότι χρειάζεται χαμηλό κόστος για την δημιουργία ενός ηλεκτρονικού καταστήματος. Τα φυσικά καταστήματα γενικά επιβαρύνονται από σημαντικά κόστη, όπως το ενοίκιο που διαθέτουν, την ταμπέλα που πρέπει να προσθέσουν, την διακόσμηση, την αγορά του αποθέματος, τον εξοπλισμό κ.α. Τα φυσικά καταστήματα πώλησης πρέπει επίσης να πληρώσουν το προσωπικό που διαθέτουν, είτε αφορά στο κομμάτι της πώλησης είτε πρόκειται για προσωπικό ασφαλείας, το οποίο μπορεί να χρειάζεται ανάλογα με την αξία των προϊόντων που υπάρχουν στο εκάστοτε κατάστημα.

Το γεγονός ότι σε ένα διαδικτυακό κατάστημά οι υπάλληλοι μπορούν να εργαστούν εξ αποστάσεως, καθίσταται πιο εύκολη η διαδικασία ανεύρεσης του κατάλληλου και αποτελεσματικού ανθρώπινου δυναμικού, το οποίο θα στελεχώσει την επιχείρηση. Ακόμα, στα ηλεκτρονικά καταστήματα, δεν χρειάζεται η αγορά μεγάλων όγκων αποθέματος, χωρίς να επιβαρύνονται με επιπρόσθετα και περαιτέρω έξοδα. Ο σχεδιασμός του λογότυπου από την άλλη, για το ψηφιακό κατάστημα, αποτελεί συνήθως, οικονομικότερη διαδικασία από την δημιουργία μιας ταμπέλας, την οποία χρειάζονται τα φυσικά καταστήματα. Διαπιστώνεται έντονα, ότι τα συνολικά έξοδα είναι γενικά πολύ χαμηλότερα στο ηλεκτρονικό εμπόριο, γεγονός το οποίο αποτελεί το πιο ελκυστικό όφελος για τους νέους επιχειρηματίες που επιθυμούν να κρατήσουν το κόστος τους σε χαμηλά επίπεδα.

Εύκολη πρόσβαση:

Ένα επιπρόσθετο πλεονέκτημα του ηλεκτρονικού εμπορίου είναι ότι τα ψηφιακά καταστήματα είναι πάντα διαθέσιμα ως προς το καταναλωτικό κοινό. Ιδίως, με την δυνατότητα της δημιουργία διαφημιστικών εκστρατειών στα Μέσα Κοινωνικής Δικτύωσης, η προσέλκυση των δυνητικών πελατών μπορεί να διενεργηθεί αποτελεσματικά, οποιαδήποτε ώρα και από οποιοδήποτε μέρος του κόσμου, επιθυμείτε να διενεργηθεί η προσέγγιση τους. Το ότι το ηλεκτρονικό κατάστημα είναι προσβάσιμο στο καταναλωτικό κοινό, ανεξαρτήτως χρονικών ορίων, δίνει την δυνατότητα της προσέλκυσης πελατών, που πιθανότατα να αγόραζαν έναν προϊόν από ένα φυσικό κατάστημα την ώρα όμως που αυτό θα ήταν ανοιχτό.

Για έναν πελάτη που θα παραγγείλει βραδινές ώρες για παράδειγμα, δεν χρειάζεται η ύπαρξη ανθρώπινου δυναμικού το οποίο θα εργάζεται σε νυχτερινή βάρδια, για να εξασφαλιστεί ότι όλες οι παραγγελίες θα υποβληθούν σε επεξεργασία. Το μόνο που χρειάζεται να 20 πραγματοποιηθεί είναι να αυτοματοποιηθεί τα συστήματα παραγγελιών, έτσι ώστε οι πελάτες να λαμβάνουν ένα email επιβεβαίωσης όταν τοποθετούν την παραγγελία τους, γεγονός που θα τους κάνει να νιώσουν σιγουριά και ασφάλεια για το πέρας και την διεκπεραίωση της παραγγελίας τους.

Διεθνείς πωλήσεις:

Ένα ακόμα σημαντικό πλεονέκτημα των ηλεκτρονικών καταστημάτων αποτελεί ότι μπορεί να πουλήσει πιο εύκολα σε πελάτες, τους οποίους μπορεί να προσελκύσει σε παγκόσμια εμβέλεια. Δηλαδή, παρέχεται η δυνατότητα η επιχείρηση να προσελκύσει το ενδιαφερόμενο κοινό της, είτε αυτό βρίσκεται στο Ηνωμένο Βασίλειο, στην Νότια Αμερική ή ακόμα και σε μικρότερες περιοχές εξαιρετικά απομακρυσμένες. Γενικά, οι διεθνείς πωλήσεις αποτελούν ένα μεγάλο επίτευγμα, καθώς ενισχύει σημαντικά την απόκτηση φήμης πιο εύκολα και γρήγορα. Διευρύνοντας την αγορά σε διεθνές επίπεδο μπορεί να γίνει κερδοφόρα μια επιχείρηση πολύ πριν από τους τοπικούς ανταγωνιστές.

Ευκολία εμφάνισης των δημοφιλών προϊόντων:

Η δυνατότητα εύκολης προβολής των δημοφιλών προϊόντων τα οποία έχουν παρουσιάσει το μεγαλύτερο όγκο πωλήσεων, διευκολύνουν την προβολή άρα και την ευκολότερη αγορά αυτών των προϊόντων από τους πελάτες-επισκέπτες της ιστοσελίδας. Παρόλο που σε ένα φυσικό κατάστημα, η προσπάθεια επηρεασμού των εισερχόμενων ατόμων εντός του καταστήματος, ώστε να αγοράσουν μερικά από τα προϊόντα από τους πωλητές θεωρείται εύκολη διαδικασία με την σωστή καθοδήγηση, οι πελάτες γενικά μπορούν πιο εύκολα να βρουν τα προϊόντα με τις καλύτερες πωλήσεις στο ηλεκτρονικό

κατάστημα. Οι καταναλωτές γενικά, επιθυμούν να αγοράσουν προϊόντα με τις καλύτερες πωλήσεις, επειδή έχουν αποδειχθεί ότι είναι τα καλύτερα, επειδή τα προτιμάει έντονα το σύνολο της αγοράς.

Ευκολία εμφάνισης των δημοφιλών προϊόντων:

Η δυνατότητα εύκολης προβολής των δημοφιλών προϊόντων τα οποία έχουν παρουσιάσει το μεγαλύτερο όγκο πωλήσεων, διευκολύνουν την προβολή άρα και την ευκολότερη αγορά αυτών των προϊόντων από τους πελάτες-επισκέπτες της ιστοσελίδας. Παρόλο που σε ένα φυσικό κατάστημα, η προσπάθεια επηρεασμού των εισερχόμενων ατόμων εντός του καταστήματος, ώστε να αγοράσουν μερικά από τα προϊόντα από τους πωλητές θεωρείται εύκολη διαδικασία με την σωστή καθοδήγηση, οι πελάτες γενικά μπορούν πιο εύκολα να βρουν τα προϊόντα με τις καλύτερες πωλήσεις στο ηλεκτρονικό κατάστημα. Οι καταναλωτές γενικά, επιθυμούν να αγοράσουν προϊόντα με τις καλύτερες πωλήσεις, επειδή έχουν αποδειχθεί ότι είναι τα καλύτερα, επειδή τα προτιμάει έντονα το σύνολο της αγοράς

Γρήγορη ανάπτυξη της επιχείρησης

Ένα από τα οφέλη του ηλεκτρονικού εμπορίου είναι η γρήγορη και εύκολη ανάπτυξη της επιχείρησης . Ιδίως στις περιπτώσεις όπου οι ηλεκτρονικές διαφημίσεις, οι οποίες ελέγχουν το πέρας των αποτελεσμάτων που σημειώνουν, πολύ πιο εύκολα και άμεσα από τις παραδοσιακές διαφημίσεις, παρουσιάσουν θετικά αποτελέσματα. Έτσι, η ηλεκτρονική επιχείρηση μπορεί να ενισχύσει την παρουσία των διαφημίσεων της και να τις στηρίξει οικονομικά, με αποτέλεσμα να αυξάνεται ολοένα και περισσότερο η Στα φυσικά καταστήματα από την άλλη, μπορεί να θεωρείται δυσκολότερη η ενίσχυση των κατηγοριών των προϊόντων ή να προστεθούν περισσότεροι εργαζόμενοι εξαιτίας του μικρού χώρου. Εν αντιθέσει με τα ηλεκτρονικά καταστήματα, στα οποία για την προσθήκη νέων προϊόντων δεν υφίσταται χωρικός περιορισμός, αποτελεί σημαντικό γεγονός το οποίο ενισχύει σημαντικά τα επίπεδα εξέλιξης της ηλεκτρονικής επιχείρησης.

1.5. ΤΟ ΜΕΛΛΟΝ ΤΟΥ ΗΛΕΚΤΡΟΝΙΚΟΥ ΕΜΠΟΡΙΟΥ

Οι ειδικοί προβλέπουν ένα πολλά υποσχόμενο και ένδοξο μέλλον του ηλεκτρονικού εμπορίου στον 21ο αιώνα. Στο άμεσο μέλλον το ηλεκτρονικό εμπόριο θα επιβεβαιωθεί περαιτέρω ως σημαντικό εργαλείο πώλησης. Το επιτυχημένο ηλεκτρονικό εμπόριο θα γίνει μια έννοια απολύτως αδιαχώριστη από τον Ιστό, επειδή οι ηλεκτρονικές αγορές γίνονται όλο και πιο δημοφιλείς και συνηθισμένες. Ταυτόχρονα, ο έντονος ανταγωνισμός στον τομέα των υπηρεσιών ηλεκτρονικού εμπορίου θα εντείνει την ανάπτυξη τους. Έτσι, οι μελλοντικές τάσεις του ηλεκτρονικού εμπορίου που επικρατούν θα είναι η αύξηση των πωλήσεων στο Διαδίκτυο και η εξέλιξη [3]. Σύμφωνα με μια νέα μελέτη από το Κέντρο Έρευνας στο Ηλεκτρονικό Εμπόριο, το ηλεκτρονικό εμπόριο θα γίνει η βιομηχανική επανάσταση του 21ου αιώνα [4]. Κάθε χρόνο, ο αριθμός των συμφωνιών ηλεκτρονικού εμπορίου αυξάνεται εξαιρετικά. Οι πωλήσεις των ηλεκτρονικών καταστημάτων είναι κάτι παραπάνω από συγκρίσιμες με αυτές των φυσικών και η τάση θα συνεχιστεί, γιατί πολύς κόσμος «φυλακίζεται» από τις δουλειές και τις δουλειές του σπιτιού, ενώ το Διαδίκτυο εξοικονομεί πολύ χρόνο και δίνει τη δυνατότητα επιλογής αγαθών στο τις καλύτερες τιμές. Η σημερινή έκρηξη των πωλήσεων στο Διαδίκτυο είναι το θεμέλιο για το υπέροχο μέλλον του ηλεκτρονικού εμπορίου.

4 Η τάση «ποσότητα προς ποιότητα» του ηλεκτρονικού εμπορίου γίνεται επίσης ολοένα και πιο εμφανής, καθώς το Διαδίκτυο έχει αποκλείσει γεωγραφικούς παράγοντες από την πώληση ίσα με την προσβασιμότητα. Επομένως, δεν έχει πλέον σημασία αν το κατάστημα σας βρίσκεται στη Νέα Υόρκη ή στο Λονδίνο ή σε μια μικρή πόλη [19]. Για να επιβιώσουν, οι έμποροι θα πρέπει να προσαρμοστούν γρήγορα στις νέες συνθήκες. Για να προσελκύσουν περισσότερους πελάτες, οι ιδιοκτήτες ηλεκτρονικών καταστημάτων όχι μόνο θα πρέπει να αυξήσουν τον αριθμό των διαθέσιμων υπηρεσιών, αλλά και να δώσουν μεγαλύτερη προσοχή σε στοιχεία όπως η ελκυστικότητα των σχεδίων, η φιλικότητα προς τον χρήστη, η ελκυστική παρουσίαση των προϊόντων και ως εκ τούτου, θα πρέπει να εφαρμόσουν σύγχρονες τεχνολογίες ώστε οι επιχειρήσεις τους να γίνουν μέρος του μελλοντικού ηλεκτρονικού εμπορίου [20]. Έχει παρατηρηθεί ότι η επιχείρηση του ηλεκτρονικού εμπορίου θα συνεχίσει να ανθεί, υπό την προϋπόθεση ότι όλα τα οφέλη προσφέρονται στον τυπικό καταναλωτή.

1.6 ΕΠΙΧΕΙΡΗΜΑΤΙΚΑ ΜΟΝΤΕΛΑ

Το ηλεκτρονικό κατάστημα ονοματίζει τις πρωτοβουλίες που επικεντρώνονται σε εφαρμογές για τους καταναλωτές και επιτρέπουν συναλλαγές και αλληλεπίδραση ανάμεσα στην επιχείρηση και τον τελικό καταναλωτή πάνω από το Internet. Χαρακτηριστικά της κατηγορίας αυτής είναι η συσσώρευση περιεχομένου με σκοπό την πώληση αγαθών και την παροχή υπηρεσιών στον καταναλωτή

Το e-shop είναι αδιαμφισβήτητα το πιο διαδεδομένο επιχειρηματικό μοντέλο διαδικτύου και ίσως και το πιο προσιτό ακόμα και σε Μικρό Μεσαίες Επιχειρήσεις. Αποτελεί ίσως το πιο κρίσιμο κριτήριο επιτυχίας της στρατηγικής μάρκετινγκ που επιλέγει μια επιχείρηση όταν στραφεί στο ηλεκτρονικό εμπόριο. Παρότι φαίνεται λόγω της απλότητας στην χρήση τους πολύ απλό να στηθεί ένα ηλεκτρονικό κατάστημα στην πραγματικότητα δεν είναι.

ΚΕΦΑΛΑΙΟ 2: ΔΙΑΣΦΑΛΙΣΗ ΠΟΙΟΤΗΤΑΣ

2.1 ΤΙ ΕΙΝΑΙ Η ΔΙΑΣΦΑΛΙΣΗ ΠΟΙΟΤΗΤΑΣ QA

Η Διασφάλιση Ποιότητας είναι μια προληπτική προσέγγιση για τη διασφάλιση της ποιότητας ενός προϊόντος ή μιας υπηρεσίας. Επικεντρώνεται στην πρόληψη των ελαττωμάτων αντί στην ανίχνευση τους μετά την εμφάνισή τους. Η Διασφάλιση Ποιότητας περιλαμβάνει τη θέσπιση προτύπων, την εφαρμογή διαδικασιών και τη συνεχή παρακολούθηση και βελτίωση αυτών των διαδικασιών για να διασφαλιστεί ότι επιτυγχάνεται το επιθυμητό επίπεδο ποιότητας.

Η μέθοδος Διασφάλισης Ποιότητας περιλαμβάνει πολλά βασικά βήματα. Πρώτον, απαιτεί τη θέσπιση προτύπων ποιότητας και προδιαγραφών που καθορίζουν το επιθυμητό αποτέλεσμα. Αυτά τα πρότυπα χρησιμεύουν ως σημείο αναφοράς βάσει του οποίου αξιολογείται το προϊόν ή η υπηρεσία. Στη

συνέχεια, η Διασφάλιση Ποιότητας περιλαμβάνει την ανάπτυξη και εφαρμογή διαδικασιών και διαδικασιών για να διασφαλιστεί ότι αυτά τα πρότυπα πληρούνται. Αυτό περιλαμβάνει δραστηριότητες όπως ο σχεδιασμός, ο σχεδιασμός, η εφαρμογή και η παρακολούθηση μέτρων ποιοτικού ελέγχου. Τέλος, η Διασφάλιση Ποιότητας απαιτεί συνεχή αξιολόγηση και βελτίωση των διαδικασιών για τη βελτίωση της συνολικής ποιότητας του προϊόντος ή της υπηρεσίας.

2.2 ΓΙΑΤΙ ΕΙΝΑΙ ΣΗΜΑΝΤΙΚΗ Η ΔΙΑΣΦΑΛΙΣΗ ΠΟΙΟΤΗΤΑΣ

Η διασφάλιση ποιότητας μπορεί επίσης να βοηθήσει στον εντοπισμό τυχόν πιθανών προβλημάτων με το προϊόν. Μπορεί να απομονώσει ένα στοιχείο της διαδικασίας παραγωγής που μπορεί να διορθωθεί γρήγορα. Μπορεί επίσης να διαδραματίσει μεγάλο ρόλο στον έλεγχο του κόστους, καθώς αναδεικνύει διαδικασίες που είναι αναποτελεσματικές ή δαπανηρές. Οι αποτελεσματικές πρακτικές QA μπορούν επίσης να ενισχύσουν τη φήμη μιας επωνυμίας, καθώς οι πελάτες έχουν θετικές εμπειρίες με ένα προϊόν. Οι επόπτες ενδέχεται να επικεντρωθούν στην ταχεία αλλαγή των προβλημάτων παραγωγής και να διασφαλίσουν ότι δεν υπάρχουν περαιτέρω προβλήματα. Ελέγχουν επίσης για ζημιές ή μόλυνση. Η διασφάλιση ποιότητας είναι ζωτικής σημασίας για διάφορους λόγους, όπως:

- Εμπιστοσύνη του Χρήστη

Οι χρήστες στο διαδίκτυο έχουν γίνει απαιτητικοί και αναζητούν πάντα την καλύτερη δυνατή εμπειρία. Μια κακή εμπειρία μπορεί να οδηγήσει έναν χρήστη να αποφασίσει να μην επιστρέψει ποτέ ξανά στην ιστοσελίδα ή την εφαρμογή. Η απογοήτευση μπορεί να είναι ένας λόγος για τους πελάτες να επιλέξουν άλλον προμηθευτή ή να μην αγοράσουν ξανά. Όταν οι χρήστες έχουν εμπιστοσύνη στη λειτουργικότητα της πλατφόρμας, είναι πιο πιθανό να τη χρησιμοποιούν συνεχώς.

- Επιστροφή του Χρήστη:

Η ικανοποίηση του πελάτη είναι ουσιώδης για τη διατήρηση της θετικής σχέσης με το κοινό. Όταν μια ιστοσελίδα ή μια εφαρμογή προσφέρει ακριβώς αυτό που ο χρήστης ψάχνει και το κάνει με άνεση και αξιοπιστία, ο χρήστης αισθάνεται ικανοποιημένος και πιθανότατα θα επιστρέψει για περισσότερη χρήση. Για παράδειγμα, αν ένα κατάστημα διαδικτυακών αγορών παρέχει γρήγορη πλοήγηση, εύκολη διαδικασία αγοράς και αποτελεσματική εξυπηρέτηση πελατών, οι πελάτες είναι πιθανότατα να επιλέξουν να αγοράσουν ξανά από εκεί.

- Εξοικονόμηση Χρόνου και Χρημάτων:

Η πρόληψη ελαττωμάτων και σφαλμάτων στα πρώιμα στάδια της ανάπτυξης μπορεί να οδηγήσει σε σημαντική εξοικονόμηση χρόνου και χρημάτων. Για παράδειγμα, αν ένα πρόβλημα ανιχνευθεί και διορθωθεί κατά τη διάρκεια της δοκιμής λογισμικού, αποφεύγεται ο χρόνος και το κόστος που θα απαιτούνταν για τη διόρθωση του αργότερα, αφού το λογισμικό έχει ήδη κυκλοφορήσει στους χρήστες.

- Αποδοτικότητα στην Ανάπτυξη:

Η διαδικασία του quality assurance βελτιώνει την αποδοτικότητα της ομάδας ανάπτυξης επιτρέποντας της να επικεντρωθεί στη δημιουργία νέων λειτουργιών και βελτιώσεων αντί να αφιερώνει πολύτιμο χρόνο στη διόρθωση σφαλμάτων. Με την πρόληψη σφαλμάτων από την αρχή, η ομάδα ανάπτυξης μπορεί να είναι πιο αποτελεσματική και πιο παραγωγική στη δημιουργία νέου λογισμικού.

Με αυτόν τον τρόπο, η διαδικασία του quality assurance δεν είναι μόνο ένα εργαλείο για την ανίχνευση σφαλμάτων, αλλά μια σημαντική διαδικασία που συμβάλλει στη βελτίωση της εμπειρίας του χρήστη, την επίτευξη των στόχων του πελάτη και την αύξηση της αποδοτικότητας της ομάδας ανάπτυξης.

2.3 ΤΡΟΠΟΙ ΔΙΑΣΦΑΛΙΣΗΣ QA

- Δοκιμή συμβατότητας

Αυτός ο τομέας των δοκιμών QA εξετάζει πόσο καλά λειτουργεί ο ιστότοπος σας σε διαφορετικές πλατφόρμες και συσκευές. Αυτό περιλαμβάνει την προβολή του τρόπου με τον οποίο οι χρήστες κινητών τηλεφώνων βλέπουν τον ιστότοπο σας, τον έλεγχο για τη διασφάλιση ότι τα κουμπιά και η πλοήγηση λειτουργούν σε διαφορετικές μορφές και εάν οι χρήστες μπορούν να δουν σωστά εικόνες σε οθόνες διαφορετικού μεγέθους.

- Δοκιμή απόδοσης

Αυτός ο τύπος δοκιμών εξετάζει εάν ο ιστότοπος σας μπορεί να χειριστεί συγκεκριμένα σενάρια, όπως περιόδους με υψηλή επισκεψιμότητα, χωρίς σημαντική πτώση στην απόδοση. Είναι σημαντικό οι ιστοσελίδες σας να φορτώνονται έγκαιρα, διαφορετικά οι πελάτες μπορεί να απογοητευτούν και να αναπηδήσουν από τον ιστότοπο σας επειδή επέλεξαν να έχουν πρόσβαση σε αυτόν σε μια ώρα της ημέρας με υψηλή επισκεψιμότητα. Ωστόσο, η δοκιμή απόδοσης δεν επικεντρώνεται αποκλειστικά στην επισκεψιμότητα του ιστότοπου. Μετρά επίσης πώς λειτουργεί ο ιστότοπος σας όταν οι χρήστες εκτελούν πολλές λειτουργίες ταυτόχρονα, πόσο αξιόπιστες είναι οι λειτουργίες μεμονωμένα και πώς συνεχίζει να συνδέεται όταν οι χρήστες συνεχίζουν να χρησιμοποιούν τις δυνατότητές του για μεγάλο χρονικό διάστημα.

- Δοκιμή περιεχομένου

Αυτό είναι το πιο βασικό τεστ αλλά και το πιο υποκειμενικό κομμάτι του QA. Περιλαμβάνει ο ιστότοπος σας περιεχόμενο σχετικό με τους χρήστες σας και παρουσιάζεται με ελκυστικό τρόπο;

- Δοκιμές Ασφαλείας

Αυτή η πτυχή είναι ιδιαίτερα σημαντική εάν ο ιστότοπος σας είναι ηλεκτρονικός εμπόριο ή με άλλο τρόπο ζητά από τους χρήστες να εισάγουν προσωπικά στοιχεία. Η διασφάλιση ότι οι φόρμες περιέχουν captchas, ότι οι προσωπικές πληροφορίες προστατεύονται μέσω ασφαλών εξουσιοδοτήσεων και η ασφάλεια των κωδικών πρόσβασης είναι μόνο μερικά μέρη της δοκιμής ασφαλείας.

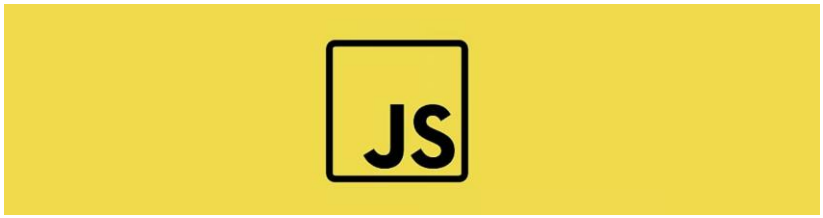
Πολλές επιχειρήσεις δεν δίνουν αρκετή προσοχή στις δοκιμές ασφαλείας και χάνουν τεράστια χρηματικά ποσά από εισβολές ransomware ή μηνύσεις μετά τη διαρροή πληροφοριών χρήστη. Ένα σημαντικό μέρος των δοκιμών ασφαλείας είναι η προσομοίωση μιας εισβολής και η διαπίστωση πώς η εταιρεία σας στο σύνολο της θα αντιδρούσε στην απώλεια σημαντικών δεδομένων. Αυτή η άσκηση δοκιμών μπορεί επίσης να είναι χρήσιμη για άλλους υπαλλήλους καθώς διερευνάτε πώς να ανταποκριθείτε καλύτερα σε μια τέτοια καταστροφή και σκιαγραφείτε ποια βήματα πρέπει να ληφθούν για την αποκατάσταση της ασφάλειας του ιστότοπου μετά από μια εισβολή.

- Λειτουργικός Έλεγχος

Όπως υποδηλώνει το όνομα, εδώ κοιτάζει η ομάδα QA σας για να δει εάν ο ιστότοπος κάνει αυτό που χρειάζεστε. Ανάλογα με το είδος του ιστότοπου που έχετε δημιουργήσει, πιθανότατα θα υπάρχουν διαφορετικές λειτουργίες που όλες πρέπει να λειτουργούν σωστά για να επιτρέπουν στους πελάτες σας να αγοράζουν προϊόντα ή υπηρεσίες. Είναι σημαντικό να έχετε έναν χάρτη του ιστότοπου σας και να περιγράφει πώς πρέπει να λειτουργούν όλες οι λειτουργίες για να συγκρίνετε την απόδοσή τους.

ΚΕΦΑΛΑΙΟ 3: ΤΕΧΝΟΛΟΓΙΕΣ ΑΝΑΠΤΥΞΗΣ ΕΦΑΡΜΟΓΗΣ

3.1 Js



Εικόνα 3.1

Η JavaScript είναι ένα αναπόσπαστο κομμάτι της σύγχρονης ανάπτυξης ιστού, καθορίζοντας την καινοτομία, τη διαδραστικότητα και την απόκριση στο διαδίκτυο. Η JavaScript δίνει τη δυνατότητα στους προγραμματιστές να δημιουργούν καθηλωτικές και ελκυστικές εμπειρίες χρήστη στο διαδίκτυο. Αξιοποιώντας πλαίσια και βιβλιοθήκες JavaScript, όπως το React, το Angular και το Vue.js, οι προγραμματιστές μπορούν να δημιουργήσουν εξαιρετικά ευέλικτες και διαδραστικές διεπαφές. Από δυναμικές ενημερώσεις περιεχομένου έως απρόσκοπτες μεταβάσεις σελίδων, η JavaScript επιτρέπει ένα επίπεδο διαδραστικότητας που ήταν προηγουμένως ανέφικτο με στατικές ιστοσελίδες.

Ένα από τα βασικά πλεονεκτήματα της JavaScript είναι η ικανότητα της να εκτελεί κώδικα απευθείας στο πρόγραμμα περιήγησης του πελάτη. Αυτή η δυνατότητα επεξεργασίας από την πλευρά του πελάτη μειώνει την επιβάρυνση των διακομιστών, μεταφέροντας εργασίες στη συσκευή του χρήστη. Ως αποτέλεσμα, οι διαδικτυακές εφαρμογές μπορούν να παρέχουν ταχύτερους χρόνους απόκρισης και καλύτερες επιδόσεις, ιδίως για εφαρμογές με μεγάλες υπολογιστικές απαιτήσεις ή συχνές αλληλεπιδράσεις.

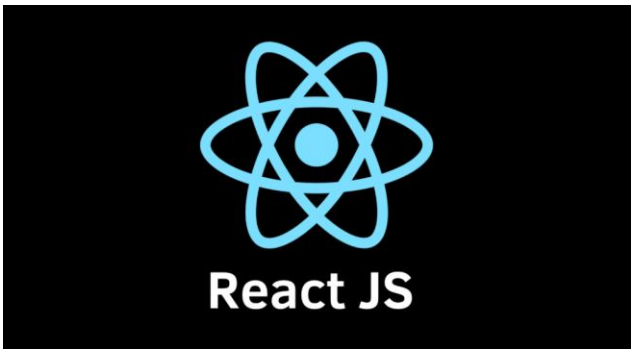
Η υποστήριξη της JavaScript για ασύγχρονο προγραμματισμό είναι καθοριστική για την κατασκευή ευέλικτων και αποδοτικών εφαρμογών ιστού. Μέσω χαρακτηριστικών όπως οι callbacks, οι υποσχέσεις και το async/await, οι προγραμματιστές μπορούν να διαχειρίζονται ασύγχρονες εργασίες, όπως η άντληση δεδομένων από διακομιστές, η επεξεργασία εισόδου χρήστη ή η εκτέλεση χρονοβόρων λειτουργιών χωρίς να μπλοκάρουν το κύριο νήμα εκτέλεσης. Ως αποτέλεσμα, οι εφαρμογές ιστού παραμένουν ευέλικτες και διατηρούν μια ομαλή εμπειρία χρήσης ακόμη και όταν χειρίζονται σύνθετες εργασίες στο παρασκήνιο.

JavaScript είναι εγγενώς διαπλατφορμική, πράγμα που σημαίνει ότι ο κώδικας που είναι γραμμένος σε JavaScript μπορεί να εκτελεστεί σε διάφορα λειτουργικά συστήματα και συσκευές χωρίς τροποποίηση. Αυτή η συμβατότητα πολλαπλών πλατφορμών καθιστά τη JavaScript ιδανική επιλογή για τη δημιουργία εφαρμογών ιστού που πρέπει να απευθύνονται σε ευρύ κοινό σε διαφορετικές συσκευές, προγράμματα περιήγησης και πλατφόρμες. Επιπλέον, πλαίσια όπως το React Native και το Ionic επιτρέπουν στους προγραμματιστές να αξιοποιούν τη JavaScript για την κατασκευή εγγενών εφαρμογών για κινητά, επεκτείνοντας περαιτέρω την εμβέλεια της JavaScript πέρα από τον ιστό.

Το οικοσύστημα της JavaScript διαθέτει ένα τεράστιο φάσμα βιβλιοθηκών, πλαισίων και εργαλείων που βελτιώνουν τη διαδικασία ανάπτυξης και δίνουν τη δυνατότητα στους προγραμματιστές να δημιουργούν γρήγορα εξελιγμένες εφαρμογές ιστού. Από πλαίσια front-end όπως τα React, Angular και Vue.js έως πλαίσια back-end όπως τα Node.js, Express.js και NestJS, υπάρχει ένα εργαλείο για σχεδόν κάθε πτυχή της ανάπτυξης ιστού. Αυτές οι βιβλιοθήκες και τα πλαίσια αφαιρούν κοινές εργασίες, παρέχουν έτοιμες λύσεις σε κοινές προκλήσεις και καλλιεργούν μια ζωντανή κοινότητα προγραμματιστών που συμβάλλουν στην πρόοδο του οικοσυστήματος.

Η ευελιξία της JavaScript επιτρέπει στους προγραμματιστές να επεκτείνουν και να προσαρμόζουν τις εφαρμογές ιστού ώστε να ανταποκρίνονται σε συγκεκριμένες απαιτήσεις και προτιμήσεις. Είτε πρόκειται για την ενσωμάτωση API τρίτου μέρους, είτε για την προσθήκη προσαρμοσμένων κινούμενων εικόνων και εφέ, είτε για την εφαρμογή σύνθετης επιχειρηματικής λογικής, η JavaScript παρέχει τα απαραίτητα δομικά στοιχεία για τη δημιουργία προσαρμοσμένων λύσεων.

3.2 React Js



Εικόνα 3.2

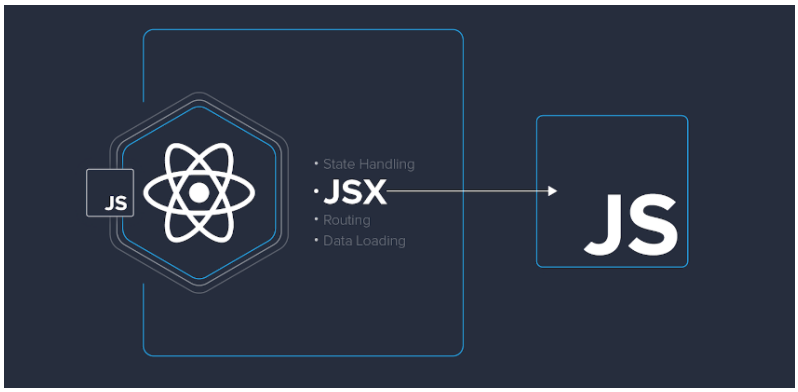
Η React JS αποτελεί μια ισχυρή βιβλιοθήκη JavaScript που έχει αναδειχθεί ως ένα από τα κύρια εργαλεία για την ανάπτυξη δυναμικών εφαρμογών ιστού. Η κύρια της δύναμη εκδηλώνεται μέσω της ευελιξίας και της απλότητας που προσφέρει στους προγραμματιστές. Με την αρχιτεκτονική των στοιχείων, η React επιτρέπει τη δημιουργία εφαρμογών με συνεκτική δομή και ευκολία συντήρησης, καθώς ο κώδικας οργανώνεται γύρω από ανεξάρτητα στοιχεία με συγκεκριμένες λειτουργίες. Αυτό βοηθά στη διαχείριση μεγάλων και πολύπλοκων εφαρμογών με μεγαλύτερη αποτελεσματικότητα και αξιοπιστία. Ένα ακόμη σημαντικό πλεονέκτημα της React είναι η διαχείριση της κατάστασης της εφαρμογής. Η React παρέχει μηχανισμούς για αποδοτική και συνεπή διαχείριση των δεδομένων της εφαρμογής, κάνοντας εύκολη την ενημέρωση του UI ανάλογα με τις αλλαγές στην κατάσταση. Αυτό βοηθά στη δημιουργία εφαρμογών που ανταποκρίνονται άμεσα στις αλλαγές του χρήστη και προσφέρουν μια ευχάριστη και απρόσκοπτη εμπειρία περιήγησης.

Πέραν αυτών, η Node.js επιτρέπει την εύκολη διαχείριση των εξαρτήσεων του έργου μέσω του npm (Node Package Manager), το οποίο παρέχει πρόσβαση σε χιλιάδες βιβλιοθήκες κώδικα JavaScript. Αυτό σημαίνει ότι οι προγραμματιστές μπορούν να αξιοποιήσουν τις έτοιμες λύσεις και τα εργαλεία που προσφέρονται από την κοινότητα του Node.js για την επιτάχυνση της ανάπτυξης των εφαρμογών τους. Επιπλέον, η Node.js είναι ιδανική για την ανάπτυξη microservices αρχιτεκτονικών, καθώς επιτρέπει τη δημιουργία αυτόνομων, ελαφρών και εύκαμπτων υπηρεσιών που μπορούν να επικοινωνούν μεταξύ τους μέσω API. Αυτή η προσέγγιση διαχωρίζει τη λειτουργικότητα της εφαρμογής σε μικρότερα κομμάτια, βελτιώνοντας τη συντήρηση και τη διαθεσιμότητα του συστήματος συνολικά.

Τέλος, η Node.js συνδυάζεται συχνά με βάσεις δεδομένων, λόγω της ευελιξίας και της επεκτασιμότητάς τους. Αυτές οι βάσεις δεδομένων διαχειρίζονται αποδοτικά τα δεδομένα που απαιτούνται από μια σύγχρονη εφαρμογή, όπως αυτές που βασίζονται σε ρεαλιστικά δεδομένα και αντιμετωπίζουν μεγάλες ροές δεδομένων σε πραγματικό χρόνο. Επιτρέπει την επαναχρησιμοποίηση κώδικα μέσω της δομής της και της διαθεσιμότητας πληθώρας βιβλιοθηκών και πακέτων. Αυτό καθιστά την ανάπτυξη γρήγορη και αποτελεσματική, καθώς οι προγραμματιστές μπορούν να επαναχρησιμοποιήσουν ή να ενσωματώσουν έτοιμες λύσεις για κοινές λειτουργίες. Με αυτόν τον τρόπο, η React διευκολύνει την ανάπτυξη εφαρμογών με υψηλή ποιότητα και ταχύτητα, επιτρέποντας τη δημιουργία εφαρμογών που προσφέρουν την καλύτερη δυνατή εμπειρία στους χρήστες.

3.3 React Features

3.3.1 JSX

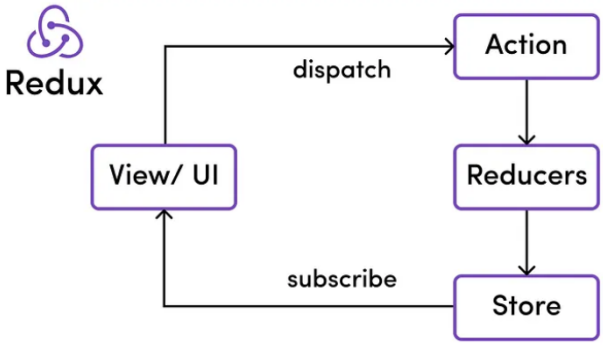


Εικόνα 3.3

Το JSX είναι μια σύνταξη που επιτρέπει στους προγραμματιστές να γράφουν HTML-παρόμοιο κώδικα μέσα σε JavaScript. Αυτός ο ειδικός τρόπος γραφής επιτρέπει την κατασκευή δομικών στοιχείων και διεπαφών χρήστη για εφαρμογές React. Αν και το JSX μοιάζει με HTML, αναγνωρίζεται και μεταγλωττίζεται ως JavaScript κώδικας. Αυτό επιτρέπει στους προγραμματιστές να δημιουργούν δυναμικά στοιχεία και να διαχειρίζονται την εμφάνιση και συμπεριφορά των εφαρμογών τους με μια πιο συμπαγή και ευανάγνωστη σύνταξη. Με αυτόν τον τρόπο, το JSX βοηθά στην επιτάχυνση της ανάπτυξης εφαρμογών React και την απλοποίηση του κώδικα, καθώς επιτρέπει στους προγραμματιστές να οργανώνουν τον κώδικα τους με έναν πιο δομημένο και ευανάγνωστο τρόπο.

Το καλό είναι ότι το React δεν μας υποχρεώνει να χρησιμοποιούμε JSX. Αλλά πολλοί προγραμματιστές, συμπεριλαμβανομένου κι εμένα, το βρίσκουν πολύ χρήσιμο για την διαχείριση των στοιχείων της διεπαφής χρήστη μαζί με τη λογική μας στο JavaScript. Γι' αυτό το JSX είναι αρκετά σημαντικό για το React, γιατί είναι το εργαλείο που χρησιμοποιείται για να δημιουργήθούν αυτά τα φοβερά στοιχεία του React που κάνουν τις ιστοσελίδες να φαίνονται πολύ ανώτερες.

3.3.2 Redux

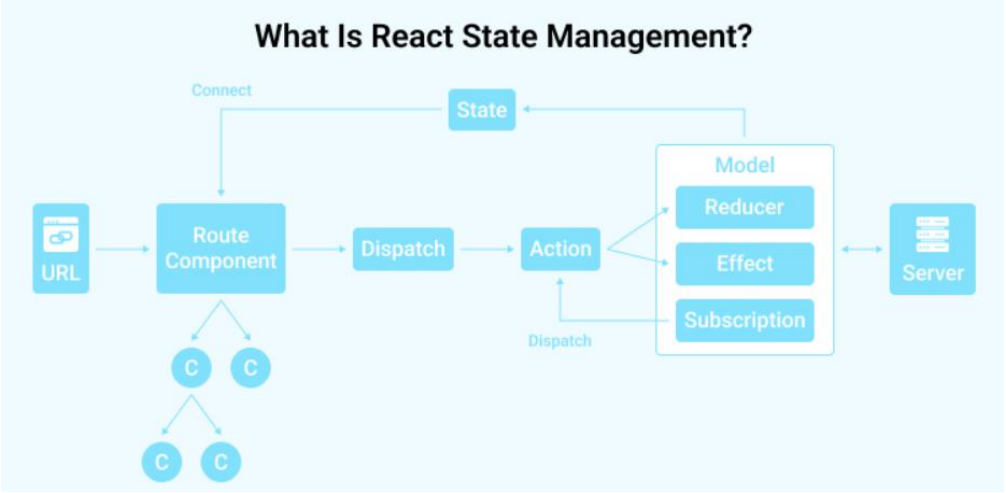


Εικόνα 3.4

Το Redux αναδεικνύεται ως ένα ισχυρό εργαλείο στον κόσμο της διαχείρισης κατάστασης σε JavaScript εφαρμογές. Αυτή η τεχνολογία συχνά συνδυάζεται με βιβλιοθήκες όπως το React ή το Angular, προσφέροντας έναν οργανωμένο και κεντρικό τρόπο διαχείρισης της κατάστασης της εφαρμογής. Ένα από τα κύρια πλεονεκτήματα του Redux είναι η δημιουργία ενός κεντρικού καταστήματος, το οποίο συγκεντρώνει την ολόκληρη κατάσταση της εφαρμογής, επιτρέποντας έτσι εύκολη πρόσβαση και τροποποίηση από οποιοδήποτε συστατικό. Ένα ακόμη σημαντικό χαρακτηριστικό του Redux είναι η προβλεψιμότητα των αλλαγών στην κατάσταση. Μέσω της ανανεώσιμης ανανέωσης των δεδομένων και της αυστηρής διεύθυνσης των δεδομένων, το Redux εξασφαλίζει ότι οι τροποποιήσεις στην κατάσταση είναι προβλέψιμες και ελέγξιμες, βοηθώντας τους προγραμματιστές στην ανίχνευση σφαλμάτων και την αντιμετώπιση τους με ευκολία.

Το Redux προάγει επίσης την αρχή της μητρικής κατάστασης, αποτρέποντας άμεσες μεταλλάξεις κατάστασης και διασφαλίζοντας την προβλεψιμότητα και τη σταθερότητα της κατάστασης της εφαρμογής. Με υποστήριξη για ενδιάμεσες εφαρμογές και προηγμένα εργαλεία ανάπτυξης, όπως το Redux DevTools, οι προγραμματιστές μπορούν να παρακολουθούν και να αναλύουν την κατάσταση της εφαρμογής, επιτρέποντας την αποτελεσματική ανάπτυξη και επίλυση προβλημάτων. Με όλα αυτά, το Redux αποτελεί έναν ισχυρό σύμμαχο στην ανάπτυξη σύγχρονων και πολύπλοκων web εφαρμογών.

3.3.3 State



Εικόνα 3.5

Το state στο React αποτελεί τον πυρήνα της διαχείρισης κατάστασης σε μια εφαρμογή. Αντιπροσωπεύει την τρέχουσα κατάσταση του χρήστη και των δεδομένων, επιτρέποντας την αλληλεπίδραση με το UI και την ενημέρωσή του. Η χρήση του state είναι ουσιώδης για την ανάπτυξη δυναμικών εφαρμογών, καθώς επιτρέπει την αλλαγή του UI σε πραγματικό χρόνο ανάλογα με τις ενέργειες του χρήστη και την εξέλιξη των δεδομένων.

Η αυτόματη αναπαράσταση του UI μετά από κάθε αλλαγή στο state εξασφαλίζει τη συνέπεια και την επαναληπτικότητα της διεπαφής χρήστη, εξαλείφοντας την ανάγκη για μη αποτελεσματικές διαδικασίες ενημέρωσης του UI. Με αυτόν τον τρόπο, το React state διευκολύνει τη δημιουργία ευέλικτων και ανταποκρίσιμων εφαρμογών, επιτρέποντας στους προγραμματιστές να επικεντρωθούν στη λειτουργικότητα της εφαρμογής χωρίς να χάνουν τη συνολική εικόνα της κατάστασης και της αλληλεπίδρασης του UI.

3.3.4 Components

Τα στοιχεία-components του React αναδεικνύουν μια θεμελιώδη διάσταση της ανάπτυξης λογισμικού με την προσέγγιση της επαναχρησιμοποίησης κώδικα και της οργάνωσης της δομής της διεπαφής χρήστη. Καταρχάς, είναι σημαντικό να κατανοήσουμε ότι λειτουργούν σύμφωνα με το ίδιο μοντέλο με τις συναρτήσεις στη JavaScript. Αυτό σημαίνει ότι μπορούν να λαμβάνουν είσοδο και να επιστρέφουν React στοιχεία, καθορίζοντας έτσι το περιεχόμενο που εμφανίζεται στον χρήστη.

Η έννοια της αρχής του διαχωρισμού είναι βασική για την κατανόηση του ρόλου των components. Με τον διαχωρισμό του λογικού τμήματος της εφαρμογής από το γραφικό, δημιουργούμε αυτόνομα, επαναχρησιμοποιήσιμα και ευκολό-χρηστα τμήματα κώδικα. Έτσι, καθένα από αυτά αναλαμβάνει ένα συγκεκριμένο καθήκον, με την εστίαση να είναι στην αντιμετώπιση μιας συγκεκριμένης λειτουργικής ανάγκης της εφαρμογής. Συνεπώς, αντιπροσωπεύουν ένα σημαντικό κομμάτι της αρχιτεκτονικής, επιτρέποντας την ανάπτυξη μιας οργανωμένης, ευέλικτης και εύκολα συντηρήσιμης εφαρμογής.

3.4 NODE JS

Εικόνα 3.6

Η Node.js αποτελεί ένα περιβάλλον εκτέλεσης JavaScript που επιτρέπει τη δημιουργία server-side εφαρμογών με χρήση της γλώσσας προγραμματισμού JavaScript. Ένα από τα βασικά της πλεονεκτήματα είναι η μη-μπλοκαρισμένη (non-blocking) φύση της, η οποία επιτρέπει στον server να εξυπηρετεί αιτήσεις χωρίς να απαιτείται η αναμονή για την ολοκλήρωση κάθε μίας ξεχωριστά. Αυτό οδηγεί σε υψηλή απόδοση και κλιμάκωση των εφαρμογών.

Επιπλέον, η Node.js χρησιμοποιείται ευρέως για τη δημιουργία real-time εφαρμογών όπως τα chat applications και οι streaming υπηρεσίες, καθώς διαθέτει δυνατότητες όπως οι WebSockets και τα συγχρονισμένα γεγονότα. Αυτό επιτρέπει την άμεση ανταπόκριση στις ενέργειες του χρήστη, προσφέροντας μια πιο δυναμική εμπειρία στους χρήστες των εφαρμογών.

3.5 TAILWIND CSS

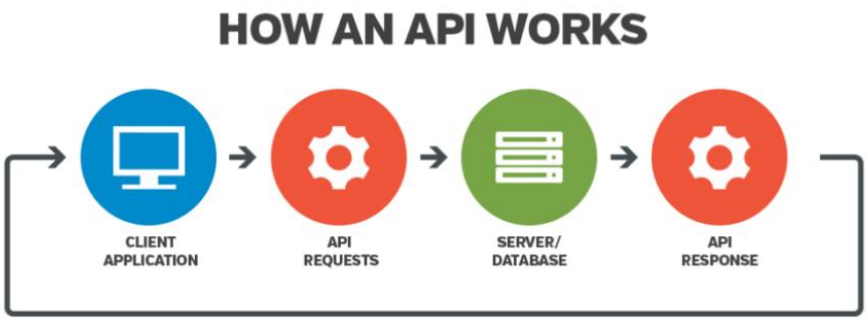
Εικόνα 3.7

Το Tailwind CSS είναι ένα εργαλείο σχεδιασμού UI που έχει κερδίσει σημαντική δημοτικότητα λόγω της προσέγγισης του στη δημιουργία διεπαφών. Η κύρια αξία του Tailwind CSS έγκειται στο γεγονός ότι παρέχει έναν επιπλέον επίπεδο αφαίρεσης στην κατασκευή διεπαφών, καθιστώντας την ανάπτυξη πιο γρήγορη και ευέλικτη.

Μία από τις βασικές προσεγγίσεις του Tailwind CSS είναι η χρήση utility classes, οι οποίες επιτρέπουν την άμεση εφαρμογή στυλ στα στοιχεία της διεπαφής. Αυτό σημαίνει ότι δεν χρειάζεται να γράφετε CSS κανόνες για κάθε στοιχείο ξεχωριστά, αλλά μπορείτε να εφαρμόσετε στυλ απευθείας μέσω των κλάσεων που παρέχει το Tailwind.Η κύρια προτεραιότητα του Tailwind CSS είναι η επιτάχυνση της διαδικασίας ανάπτυξης, καθώς επιτρέπει στους προγραμματιστές να δημιουργούν γρήγορα και εύκολα προσαρμοσμένα στυλ χωρίς την ανάγκη να γράφουν CSS από την αρχή. Αυτό είναι ιδιαίτερα χρήσιμο για μικρούς ή μεσαίους προγραμματιστές που ίσως δεν έχουν τις γνώσεις ή τον χρόνο να δημιουργήσουν προσαρμοσμένα στυλ CSS από το μηδέν.

Παρέχει επίσης ένα εύρος επιλογών σχεδιασμού που καλύπτουν τις περισσότερες ανάγκες ενός έργου UI. Από την ευελιξία στην προσαρμογή των χρωμάτων και των γραμματοσειρών έως τη δυνατότητα δημιουργίας διατάξεων χωρίς περιορισμούς, οι προγραμματιστές μπορούν να προσαρμόσουν εύκολα την εμφάνιση των εφαρμογών τους με βάση τις απαιτήσεις του σχεδιασμού και της λειτουργικότητας. Συνολικά, το Tailwind CSS αποτελεί ένα ισχυρό εργαλείο σχεδιασμού UI που βοηθά τους προγραμματιστές να δημιουργούν εφαρμογές μεγάλης ποιότητας γρήγορα και αποτελεσματικά, εξαλείφοντας πολλά από τα εμπόδια που συνήθως συναντούνται στη διαδικασία ανάπτυξης διεπαφών.

3.6 API'S



Εικόνα 3.8

Το API (Application Programming Interface - Διεπαφή Προγραμματισμού Εφαρμογών) είναι ένα σύνολο κανόνων και προδιαγραφών που επιτρέπουν στις εφαρμογές να επικοινωνούν μεταξύ τους. Μέσω των APIs, διαφορετικά λογισμικά μπορούν να ανταλλάσσουν δεδομένα και να εκτελούν λειτουργίες με αυτόματο και συντονισμένο τρόπο, χωρίς να χρειάζεται ο χρήστης να παρεμβαίνει άμεσα. Με τη βοήθειά τους, οι προγραμματιστές μπορούν να αναπτύξουν λογισμικό που αλληλεπιδρά με άλλα συστήματα, εκμεταλλευόμενοι τις δυνατότητες τους και επεκτείνοντας τις λειτουργίες του. Τα API τείνοθν να διευκολύνουν την ανάπτυξη εφαρμογών με την παροχή μιας καλά καθορισμένης διεπαφής για την ανταλλαγή δεδομένων και την εκτέλεση λειτουργιών. Αυτό έχει ως αποτέλεσμα τη μείωση της πολυπλοκότητας του προγραμματισμού και την αύξηση της αποτελεσματικότητας και της ταχύτητας στην ανάπτυξη λογισμικού.

Επιπρόσθετα,τα APIs προωθούν την ανάπτυξη εφαρμογών υψηλής ποιότητας, καθώς επιτρέπουν την αποσύνδεση των διαφορετικών τμημάτων του λογισμικού. Αυτό σημαίνει ότι μπορούμε να εστιάσουμε σε κάθε τμήμα της εφαρμογής ξεχωριστά, βελτιώνοντας τη συντήρηση και την επέκτασή της. Επίσης τα APIs συμβάλλουν στη βελτίωση της ασφάλειας των συστημάτων, καθώς πολλά από αυτά προσφέρουν μέτρα προστασίας για την αποτροπή της ανεπιθύμητης πρόσβασης και της διαρροής δεδομένων. Μέσω καλά σχεδιασμένων APIs, οι προγραμματιστές μπορούν να ενισχύσουν την ασφάλεια των εφαρμογών και να προστατεύσουν τα δεδομένα των χρηστών.

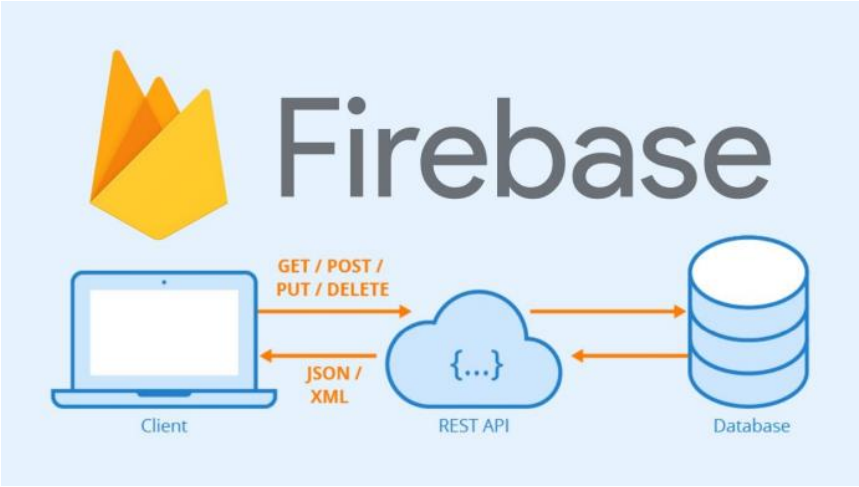
3.7 FIREBASE

3.7.1 Firebase

Το 2011, ο James Tamplin και ο Andrew Lee δημιούργησαν το Firebase υπό το όνομα Envolve. Αρχικά, ο σκοπός του Envolve ήταν να παρέχει στους προγραμματιστές APIs για την ενσωμάτωση λειτουργιών ζωντανής συνομιλίας σε ιστοσελίδες. Ωστόσο, πέρα από απλή συνομιλία, οι χρήστες διαδραμάτισαν

ευρύτερο ρόλο με το Envolve. Οι προγραμματιστές εκμεταλλεύτηκαν το Envolve για τη μεταφορά δεδομένων εφαρμογών, όπως online παιχνίδια, επαφές, ημερολόγια κλπ.

Το Firebase αποτελεί ένα εξαιρετικά ευέλικτο εργαλείο για την ανάπτυξη εφαρμογών στον ιστό, προσφέροντας πληθώρα λειτουργιών που καλύπτουν τις ανάγκες των προγραμματιστών και των εφαρμογών τους. Με χαρακτηριστικά όπως η δυνατότητα αυθεντικοποίησης, η αποθήκευση δεδομένων σε πραγματικό χρόνο, και η αποθήκευση στο cloud, η Firebase αποτελεί βασικό εργαλείο ανάπτυξης λογισμικού.Μεταξύ των πολλών δυνατοτήτων που προσφέρει, η Firebase διαθέτει ένα πλήρες σύστημα αυθεντικοποίησης που επιτρέπει τη διαχείριση των χρηστών και των συνεδριών σύνδεσης. Αυτό αποδεικνύεται εξαιρετικά χρήσιμο για εφαρμογές που απαιτούν πιστοποίηση των χρηστών ή περιορισμό της πρόσβασης σε περιεχόμενο.



Εικόνα 3.9

Επιπλέον, το Firebase προσφέρει μια βάση δεδομένων σε πραγματικό χρόνο, ιδανική για εφαρμογές που απαιτούν άμεση ενημέρωση των δεδομένων σε όλους τους χρήστες. Αυτή η δυνατότητα είναι καίρια για εφαρμογές πραγματικού χρόνου όπως συνομιλία και παιχνίδια. Τέλος, το Firebase προσφέρει εύκολη χρήση και ενσωματώνει πολλές λειτουργίες που επιτρέπουν στους προγραμματιστές να επικεντρωθούν στην κύρια λογική της εφαρμογής τους. Αυτό κάνει την ανάπτυξη ποιοτικών εφαρμογών πιο αποδοτική και γρήγορη.

3.7.2 Firebase authentication

Μια από τις πιο κυρίαρχες δραστηριότητες του Firebase είναι η δημιουργία βήματος πιστοποίησης χρήστη μέσω email, Facebook, Twitter, GitHub ή Google. Επιπλέον, η λειτουργία πιστοποίησης του Firebase υποστηρίζει επίσης την ανώνυμη πιστοποίηση για εφαρμογές. Η πιστοποίηση του Firebase μπορεί να βοηθήσει στη διατήρηση των προσωπικών πληροφοριών των χρηστών με μεγαλύτερη ασφάλεια, διασφαλίζοντας παράλληλα ότι ο λογαριασμός και οι προσωπικές πληροφορίες του χρήστη

δεν θα κλαπούν. Η διεπαφή χρήστη του Firebase παρέχει πρόσβαση τόσο στην ανάκτηση κωδικού πρόσβασης όσο και στην επαλήθευση του χρήστη. Οι πελάτες μπορούν να δημιουργήσουν έναν ανώνυμο προσωρινό λογαριασμό που μπορεί αργότερα να συνδεθεί με παροχέα υπηρεσιών τρίτων χωρίς να αποκαλύψουν την πραγματική τους ταυτότητα. Επιπλέον, διαθέτει μια τεχνική γνωστή ως Smart Lock για την αυτόματη απομνημόνευση και σύνδεση με διαπιστευτήρια. Η διεύθυνση email απαιτείται ως διαπιστευτήριο σύνδεσης από το SDK πιστοποίησης του Firebase.

3.7.3 Firabase Cloud storage

Το Firebase Cloud Storage αποτελεί μια σημαντική υπηρεσία που προσφέρει έναν εξαιρετικά λειτουργικό τρόπο αποθήκευσης και διαχείρισης αρχείων στο cloud. Μέσω αυτής της υπηρεσίας, οι προγραμματιστές έχουν τη δυνατότητα να αποθηκεύουν διάφορα είδη δεδομένων όπως εικόνες, βίντεο, ήχους και άλλα πολυμεσικά αρχεία σε έναν ασφαλή και αξιόπιστο χώρο αποθήκευσης στο cloud της Google.

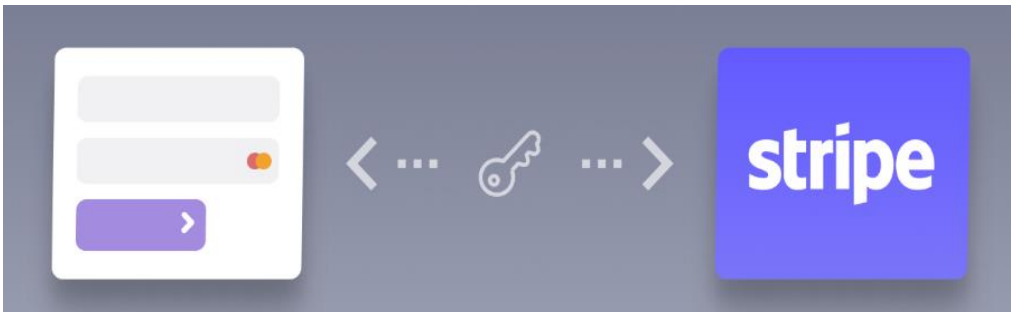
Η βασική λειτουργία του Firebase Cloud Storage είναι η διασφάλιση ότι οι προγραμματιστές μπορούν να αποθηκεύουν και να ανακτούν δεδομένα αρχείων με ευκολία και αποτελεσματικότητα, χωρίς την ανάγκη για πολύπλοκες διαδικασίες ή διαχείριση δικτύου. Επιπλέον, η υπηρεσία παρέχει προηγμένες δυνατότητες όπως ορισμός δικαιωμάτων πρόσβασης και εφαρμογή πολιτικών ασφαλείας, που εξασφαλίζουν την προστασία των δεδομένων από ανεπιθύμητη πρόσβαση ή κακόβουλη χρήση. Οι προγραμματιστές μπορούν να προσαρμόσουν αυτές τις ρυθμίσεις σύμφωνα με τις ανάγκες της εφαρμογής τους, εξασφαλίζοντας έτσι την αποτελεσματική διαχείριση και ασφάλεια των αρχείων τους.

Η Firebase υποστηρίζει διάφορες μεθόδους ανάκτησης δεδομένων, όπως η ανάκτηση αρχείων μέσω URL ή API calls, που διευκολύνουν την ενσωμάτωση των αρχείων στις εφαρμογές και τις υπηρεσίες τους. Με αυτόν τον τρόπο, οι προγραμματιστές μπορούν να δημιουργήσουν εφαρμογές που προσφέρουν υψηλή απόκριση και εξυπηρέτηση στους χρήστες τους, επιτρέποντας τη γρήγορη και αξιόπιστη πρόσβαση στα αρχεία και τα δεδομένα που αποθηκεύονται στο cloud.

Το Firebase Hosting αποτελεί μια πλατφόρμα API που λειτουργεί ως πραγματική υπηρεσία φιλοξενίας βάσεων δεδομένων στην πλατφόρμα του Cloud computing. Ενσωματώνεται και χρησιμοποιείται ταυτόχρονα με το ισχυρό σύστημα διακομιστών της Google. Η κύρια λειτουργία του Firebase Hosting είναι να βοηθά τους προγραμματιστές να απλοποιήσουν πολλές λειτουργίες βάσης δεδομένων κατά τη διάρθρωση του ιστότοπου. Με την ενσωμάτωση του Firebase Hosting στο σύστημα, μπορείτε να εξοικονομήσετε πολύ χρόνο και προσπάθεια για τον σχεδιασμό του πίσω μέρους του ιστότοπου σας. Ειδικότερα, το Firebase γενικά και το Firebase Hosting θεωρούνται επίσης μια ευέλικτη υπηρεσία αποθήκευσης βάσεων δεδομένων με εξαιρετικές δυνατότητες ασφάλειας δεδομένων. Για παράδειγμα, όταν επιθυμείτε να αναπτύξετε μια λειτουργία άμεσης συγχρονισμένης online συνομιλίας όπως η συνομιλία στο Facebook, ο προγραμματιστής χρειάζεται μόνο να αναπτύξει την πλευρά του χρήστη και τις υπόλοιπες λειτουργίες όταν ενσωματωθεί στον ιστότοπο. Χρειάζεται απλώς να καλέσει τις ιδιότητες της API και μπορεί να συνδεθεί και να τη χρησιμοποιήσει αμέσως.

3.8 STRIPE

3.8.1 STRIPE



Εικόνα 3.10

Το Stripe αποτελεί μια αναγνωρισμένη πλατφόρμα πληρωμών που εξυπηρετεί ιστοσελίδες και εφαρμογές σε όλο το φάσμα της επιχειρηματικότητας. Η δημοφιλία του οφείλεται στην αξιοπιστία και την ασφάλεια που παρέχει στην επεξεργασία των πληρωμών, καθώς και στην ευκολία ενσωμάτωσης του στις εφαρμογές. Έχει καθιερωθεί ως μια αξιόπιστη επιλογή για επιχειρήσεις κάθε μεγέθους, προσφέροντας τη δυνατότητα αποδοχής πληρωμών με ασφάλεια και αποτελεσματικότητα.

Το Stripe διαθέτει εξελιγμένες δυνατότητες ασφάλειας, που εξασφαλίζουν την προστασία των προσωπικών δεδομένων και των πληρωμών των πελατών. Με πλήρη συμμόρφωση προς τους κανονισμούς PCI, η πλατφόρμα προσφέρει ένα αξιόπιστο περιβάλλον για τις συναλλαγές, ενώ η ανώδυνη διαδικασία ενσωμάτωσης της στις εφαρμογές εξασφαλίζει την ευκολία χρήσης για τους προγραμματιστές. Τέλος, το Stripe παρέχει μια εκτενή σουίτα λειτουργιών για τη διαχείριση και ανάλυση των πληρωμών. Με δυνατότητες όπως η διαχείριση επιστροφών, η δημιουργία και παρακολούθηση τιμολογίων και η υποστήριξη συνδρομών, οι χρήστες έχουν τη δυνατότητα να διαχειρίζονται τις πληρωμές τους με ακρίβεια και αποτελεσματικότητα. Με όλες αυτές τις δυνατότητες, το Stripe προσφέρει ένα ολοκληρωμένο και ευέλικτο περιβάλλον για τη διαχείριση των πληρωμών σε κάθε είδους επιχείρηση.

Γιατί τα APIs είναι σημαντικά στις πληρωμές;

Ασφάλεια: Η χρήση API για πληρωμές ενσωματώνει σύνθετα μέτρα ασφάλειας που προστατεύουν τα δεδομένα του χρήστη και τις συναλλαγές. Παραδείγματος χάριν, οι APIs της Stripe διαχειρίζονται την κρυπτογράφηση και την επεξεργασία των δεδομένων των καρτών χωρίς να τα αποθηκεύουν στον διακομιστή του εμπόρου, μειώνοντας έτσι τον κίνδυνο διαρροής δεδομένων.

Ευελιξία και Κλιμάκωση: APIs επιτρέπουν στις επιχειρήσεις να προσαρμόζουν τις λύσεις πληρωμών τους στις ανάγκες τους και να κλιμακώνουν αποτελεσματικά τις υπηρεσίες τους καθώς αυξάνεται η ζήτηση. Τα APIs διευκολύνουν την προσθήκη ή την αλλαγή τύπων πληρωμών, νομισμάτων ή γεωγραφικών περιοχών με μικρές προσαρμογές στον κώδικα.

Διαλειτουργικότητα: Χρησιμοποιώντας APIs, οι εφαρμογές μπορούν να επικοινωνούν με πολλαπλά συστήματα και υπηρεσίες παγκοσμίως, παρέχοντας μια ενιαία, ομοιόμορφη εμπειρία για τους χρήστες ανεξαρτήτως της τοποθεσίας τους ή των τοπικών υποδομών.

Συνοψίζοντας, τα APIs στο πεδίο των πληρωμών διαδραματίζουν ένα κρίσιμο ρόλο στην ενσωμάτωση, ασφάλεια, ευελιξία και διαλειτουργικότητα των πληρωματικών λύσεων. Αυτό επιτρέπει στις επιχειρήσεις να προσφέρουν εύκολες, ασφαλείς και προσαρμοσμένες λύσεις πληρωμών στους πελάτες τους.

ΚΕΦΑΛΑΙΟ 4: ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΚΑΙ ΠΕΡΙΓΡΑΦΗ ΛΕΙΤΟΥΡΓΙΩΝ ΕΦΑΡΜΟΓΗΣ

4.1 Contact layer

Εικόνα 4.1

Αυτό το React στοιχείο ονομάζεται ContactLayer και χρησιμοποιείται για την εμφάνιση πληροφοριών επικοινωνίας. Αυτό το στοιχείο σχεδιάστηκε για να λειτουργεί ως διεπαφή χρήστη σε μια ιστοσελίδα, με σκοπό την παρουσίαση πληροφοριών επικοινωνίας. Δημιουργεί δύο τμήματα πληροφοριών επικοινωνίας: ένα που περιέχει τις ώρες λειτουργίας και ένα που περιέχει το τηλέφωνο και το email. Εισάγει δύο εικονίδια από τις βιβλιοθήκες react-icons, συγκεκριμένα τα FaPhone και MdEmail, που χρησιμοποιούνται για την εμφάνιση εικονιδίων τηλεφώνου και email αντίστοιχα.

```
import React from "react";
import { FaPhone } from "react-icons/fa";
import { MdEmail } from "react-icons/md";

export default function ContactLayer() {
    return (
        <div className="w-full flex  justify-between sm:px-4 sm:py-2 z-20
relative bg-gray-700">
            <div className="flex items-center justify-start p-2 sm:p-0 gap-x-3
text-[8px] sm:text-[13px] text-white sm:font-extrabold">
                <h6 className="flex items-center cursor-pointer gap-x-1
hover:underline">
                    Open Monday to Friday 10:00 – 18:00
                </h6>
            </div>
```

```
            <div className="flex items-center justify-end p-1 gap-x-3 text-[8px]
sm:text-[13px] text-white sm:font-extrabold">
                <h6 className="flex items-center cursor-pointer gap-x-1
hover:underline">
                    <FaPhone size={12} className="" />: 210 93XXXXX
                </h6>
                <h6 className="flex items-center cursor-pointer gap-x-1
hover:underline">
                    <MdEmail size={14} className="" />: kosmasXXXX@gmail.com
                </h6>
            </div>
        </div>
    );
}
```

Το πρώτο τμήμα περιλαμβάνει το ωράριο λειτουργίας, ενώ το δεύτερο περιλαμβάνει τα στοιχεία επικοινωνίας, όπως το τηλέφωνο και το email. Η μορφοποίηση του κειμένου και η προσθήκη διαδραστικών χαρακτηριστικών, όπως το cursor-pointer και το hover:underline, επιτρέπουν στον χρήστη να αλληλεπιδρά με τις πληροφορίες επικοινωνίας με μια πιο φυσική εμπειρία.

4.2 HEADER

4.2.1 Επικεφαλίδα της εφαρμογής

Ο κώδικας αυτός περιλαμβάνει λειτουργίες που διαχειρίζονται την αλληλεπίδραση του χρήστη με την εφαρμογή. Αυτές οι λειτουργίες περιλαμβάνουν την εναλλαγή της εμφάνισης μενού και, τον έλεγχο της κατάστασης σύνδεσης του χρήστη καθώς και την διαχείρηση της κατάστασης της εφαρμογής.

Εικόνα 4.2

```
// Data for signup

    const handleOpen1 = (event) => {
        event.stopPropagation();
        setIsOpen1(!isOpen1);
        setIsOpen2(false);
```

```
    };
    const handleOpen2 = (event) => {
        event.stopPropagation();
        setIsOpen2(!isOpen2);
        setIsOpen1(false);
    };
    useEffect(() => {
        const handleClickOutside = (event) => {
            if (menuRef.current && !menuRef.current.contains(event.target)) {
                setMenuToggle(false);
            }
            if (
                accountRef.current &&
                !accountRef.current.contains(event.target)
            ) {
                setAccount(false);
            }
        };

        document.addEventListener("click", handleClickOutside);
        return () => {
            document.removeEventListener("click", handleClickOutside);
        };
    }, []);

    const handleMenu = (event) => {
        event.stopPropagation();
        setMenuToggle(!menuToggle);
    };
    const handleAccount = (event) => {
        event.stopPropagation();
        if (uid) {
            // If the user is logged in, log them out
            dispatch(logout());
            alert("User is Logout!");
        } else {
            setAccount(!account);
        }
    };

    useEffect(() => {
        setCartCounter(cartItems.items.length);

        console.log(cartItems.items.length, "cartItems");
    }, [cartItems.items]);
```

Η λειτουργία handleOpen1 και η λειτουργία handleOpen2 είναι υπεύθυνες για τη διαχείριση της εμφάνισης και απόκρυψης δύο διαφορετικών μενού στην εφαρμογή. Εάν ένα από τα μενού είναι ήδη ανοικτό, η αντίστοιχη λειτουργία κλείνει αυτό το μενού και ανοίγει το άλλο, προσφέροντας μια ελεγχόμενη εμφάνιση για τον χρήστη.

handleOpen1 & handleOpen2: Αυτές οι λειτουργίες εναλλάσσουν την εμφάνιση των μενού ή των υπομενού, διασφαλίζοντας ότι μόνο ένα από αυτά μπορεί να είναι ορατό ταυτόχρονα.

handleMenu & handleAccount: Αυτές οι λειτουργίες διαχειρίζονται την κατάσταση των μενού πλοήγησης και των μενού λογαριασμού. Επιπλέον, η λειτουργία handleAccount προβλέπει τη δυνατότητα αποσύνδεσης του χρήστη, εάν είναι συνδεδεμένος.

Από την άλλη πλευρά, οι λειτουργίες handleMenu και handleAccount διαχειρίζονται την κατάσταση δύο διαφορετικών στοιχείων της εφαρμογής. Η λειτουργία handleMenu ελέγχει την κατάσταση του κύριου μενού, ενώ η λειτουργία handleAccount διαχειρίζεται την κατάσταση του λογαριασμού του χρήστη. Αυτές οι λειτουργίες προσφέρουν μια ομαλή και ευέλικτη διαχείριση των στοιχείων, ανταποκρινόμενες στις διάφορες ενέργειες του χρήστη. Η λειτουργία αυτή περιλαμβάνει τα εξής βήματα:

Ελέγχει εάν ο χρήστης έκανε κλικ εκτός των καθορισμένων περιοχών (menuRef και accountRef). Αν ναι, αλλάζει τις καταστάσεις setMenuToggle και setAccount σε false, κλείνοντας τυχόν ανοιχτά μενού ή παράθυρα διαλόγου.

4.2.2 Διαχείριση Αναζήτησης και Πλοήγηση σε Αποτελέσματα

Εικόνα 4.3

Ο κώδικας παρακάτω αφορά τη λειτουργικότητα της γραμμής αναζήτησης στην εφαρμογή. Ο συνδυασμός hooks από το React και Redux χρησιμοποιείται για να παρέχει δυναμική ανατροφοδότηση στον χρήστη και να κατευθύνει τον χρήστη στη σελίδα αποτελεσμάτων με βάση την αναζήτησή του.

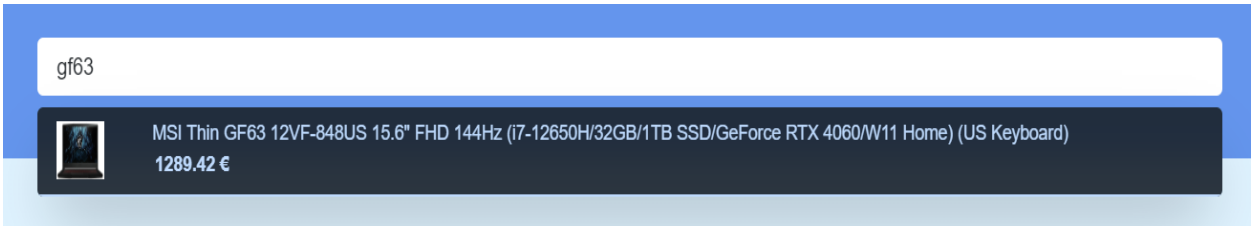
```
// updated search bar
    const history = useHistory();
    const [searchData, setSearchData] = useState([]);
    const [suggestions, setSuggestions] = useState(true);
    const { allProducts } = useSelector((state) => state.products);
    useEffect(() => {
```

```
        let uniqueIds = new Set();
        let searchedData = allProducts?.filter((product) => {
            // console.log(product.title, "product");
            const lowerCaseTitle = product.title?.toLowerCase();
            if (
                !uniqueIds.has(product.id) &&
                lowerCaseTitle.includes(searchValue?.toLowerCase())
            ) {
                uniqueIds.add(product.id);
                return true;
            }
            return false;
        });

        console.log(searchedData, "searchData");
        setSearchData(searchedData);
    }, [searchValue, allProducts]);
    // handle results

    const handleResults = () => {
        setSuggestions(false);
        history.push(`/result/${searchValue}`);
    };

    const [totalPrice, setTotalPrice] = useState(0);
    const [row, setRow] = useState([]);

    const ccyFormat = (num) => {
        return parseFloat(num).toFixed(2);
```

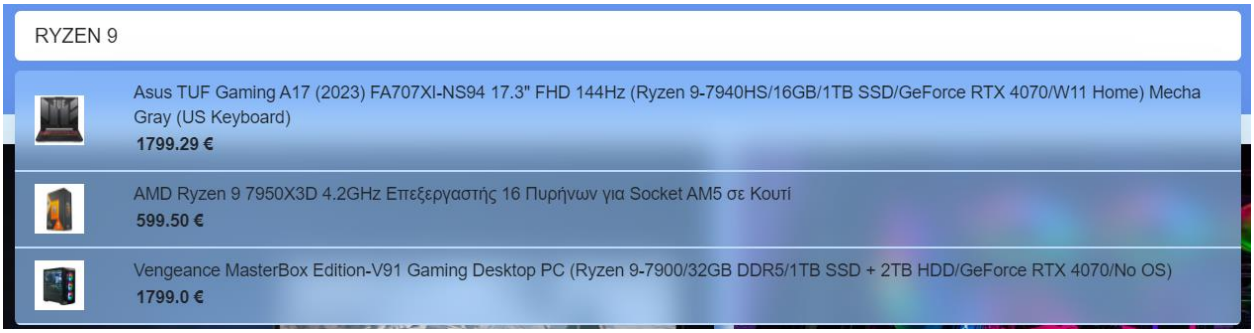
Με τη χρήση του useEffect, παρακολουθείται η αλλαγή στην αναζήτηση (searchValue) και τη λίστα των προϊόντων (allProducts). Κάθε φορά που αυτά τα δεδομένα αλλάζουν, φιλτράρεται η λίστα των προϊόντων με βάση την αναζήτηση και ανανεώνεται η searchData.

Η συνάρτηση handleResults() καλείται για να διαχειριστεί τα αποτελέσματα της αναζήτησης. Εδώ, η μεταβλητή suggestions ορίζεται ως false για να σταματήσουν οι προτάσεις κατά την πλοήγηση στη σελίδα αποτελεσμάτων, και μετά ο χρήστης κατευθύνεται στη σελίδα αποτελεσμάτων με τη χρήση της συνάρτησης history.push().

4.2.3 Αντίδραση στην Τοποθεσία URL και Ενημέρωση Κατάστασης Αναζήτησης



Εικόνα 4.4



Εικόνα 4.5

Η παρακάτω λειτουργία διαχειρίζεται τις αλλαγές στην τοποθεσία URL για να ελέγχει αν ο χρήστης βρίσκεται στη σελίδα αποτελεσμάτων αναζήτησης και να ενημερώνει αντίστοιχα την κατάσταση των αντικειμένων που έχουν αναζητηθεί.

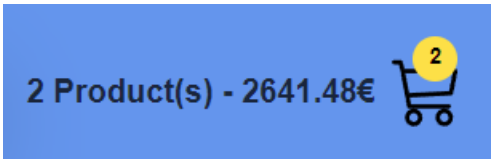
```
const location = useLocation();
    useEffect(() => {
        if (location.pathname.includes("result")) {
            console.log("this is result", location.pathname);
        } else {
            dispatch(getSearchedItem(null));
            // console.log("this is", location.pathname);
        }
    }, [location.pathname]);
```

Χρησιμοποιώντας το useLocation() από το react-router-dom, βλέπει αλλαγές στην τοποθεσία URL.

Το useEffect ενεργοποιείται κάθε φορά που αλλάζει η (pathname) της τοποθεσίας, ελέγχοντας αν η διαδρομή περιλαμβάνει τον όρο "result".

Αν ναι, καταγράφει την παρουσία στη σελίδα αποτελεσμάτων. Αν όχι, εκτελεί την ενέργεια getSearchedItem(null) για να καθαρίσει τυχόν προηγούμενα αποτελέσματα αναζήτησης.

4.2.3 Διαχείριση Καλαθιού Αγορών και Υπολογισμός Συνολικής Τιμής



Εικόνα 4.6

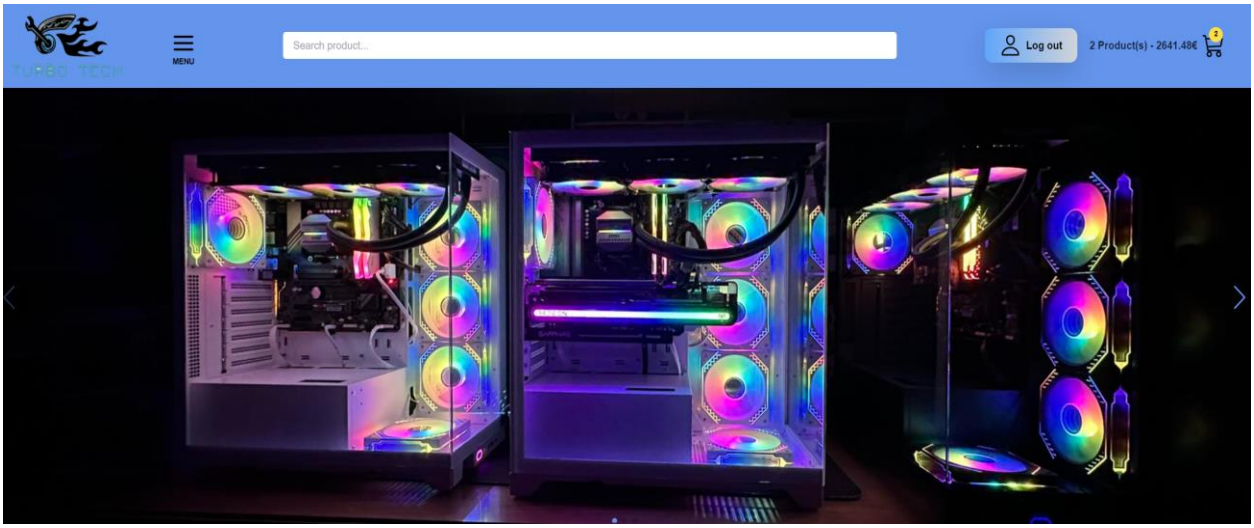
Αυτή η λειτουργία αναλαμβάνει τη διαχείριση των αντικειμένων στο καλάθι αγορών και τον υπολογισμό της συνολικής τιμής αυτών των αντικειμένων.

```
useEffect(() => {
        if (Array.isArray(cartItems.items) && cartItems.items.length > 0) {
            const formattedRows = cartItems.items.map((item, index) => {
                const discount = parseFloat(item.discount) || 0;
                const price = parseFloat(item.price) || 0;

                let discountedPrice = price;
                if (discount) {
                    const discountAmount = (discount / 100) * price;
                    discountedPrice = price - discountAmount;
                }

                return {
                    id: item?.id,
                    sl: index + 1,
                    img: item.images[0],
                    title: item.title,
                    price: discountedPrice,
                    quantity: item?.quantity || 1,
                };
            });
            const totalPrice = formattedRows.reduce(
                (total, row) => total + row.price * row.quantity,
                0
            );

            console.log("Formatted Rows:", formattedRows);
            setRow(formattedRows);
            setTotalPrice(totalPrice);
        } else {
            setTotalPrice(0);
            setRow([]);
        }
    }, [cartItems]);
```

```
    useEffect(() => {
        const calculatedTotalPrice = row?.reduce(
            (total, item) =>
                total + Number(item.price) * (Number(item.quantity) || 1),
            0
        );
        setTotalPrice(calculatedTotalPrice);
    }, [row]);
```

Η ενημέρωση του καλαθιού γίνεται με τη χρήση του useEffect οπου διαπιστώνει αν υπάρχουν αντικείμενα στο καλάθι (cartItems.items). Στη συνέχεια, κάθε αντικείμενο στο καλάθι διαμορφώνεται σε έναν πίνακα όπου κάθε στοιχείο περιλαμβάνει τις λεπτομέρειες του όπως την ταυτότητα, την τιμή μετά την εφαρμογή εκπτώσεων και την ποσότητα. Η τιμή έκπτωσης υπολογίζεται με βάση το ποσοστό της έκπτωσης και την αρχική τιμή του προϊόντος. Μετά τη δημιουργία του ανανεωμένου πίνακα, η συνολική τιμή υπολογίζεται με τη συνάρτηση reduce. Έτσι προσφέρει στον χρήστη απεικόνιση του συνολικού κόστους των αγορών του.

4.2.4 Σταθεροποίηση της Κεφαλίδας κατά την Κύλιση



Εικόνα 4.7

Η επόμενη λειτουργία παρακολουθεί τη θέση κύλισης της επικεφαλίδας και την σταθεροποιεί στο πάνω μέρος της σελίδας όταν ο χρήστης κυλίσει προς τα κάτω, ή αντίστοιχα προς τα πάνω. Με αυτόν τον τρόπο προσθέτει μια πινελιά στην εμφάνιση κατά την περιήγηση απο τον χρήστη.

```
const [scrollPosition, setScrollPosition] = useState(0);
```

```
const [headerFixed, setHeaderFixed] = useState(false);

useEffect(() => {
    const handleScroll = () => {
        setScrollPosition(window.pageYOffset);
    };

    window.addEventListener("scroll", handleScroll);
    return () => {
        window.removeEventListener("scroll", handleScroll);
    };
}, []);
useEffect(() => {
    if (scrollPosition > 190) {
        setHeaderFixed(true);
        // console.log(scrollPosition, "scroll position");
    } else {
        setHeaderFixed(false);
    }
}, [scrollPosition]);
const handleScrollToTop = () => {
    window.scrollTo({
        top: 0,
        behavior: "smooth",
    });
};
```

Το πρώτο useEffect παρακολουθεί τη θέση κύλισης της σελίδας και ενημερώνει τη μεταβλητή scrollPosition κάθε φορά που ο χρήστης κάνει κύλιση. Το δεύτερο useEffect αναλαμβάνει να ελέγχει την τιμή της scrollPosition και να αλλάζει την τιμή της headerFixed ανάλογα. Όταν η θέση κύλισης είναι μεγαλύτερη από 190 pixels, η επικεφαλίδα σταθεροποιείται, αλλιώς όχι. Η συνάρτηση handleScrollToTop καλείται για να μετακινήσει το παράθυρο προβολής στην κορυφή της σελίδας με ομαλό τρόπο, χρησιμοποιώντας την ιδιότητα behavior: "smooth".

4.2.5 Διαχείριση Εμφάνισης Μενού

Εικόνα 4.8

Ο παρακάτω κώδικας αποτελεί μέρος της επικεφαλίδας της εφαρμογής, όπου ρυθμίζεται η εμφάνιση του κουμπιού για την εναλλαγή του μενού. Προσφέρει μια διαδραστική λειτουργία για τον χρήστη να ανοίγει ή να κλείνει το μενού πλοήγησης.

```
<div className="lg:mb-3.5">
                                {menuToggle ? (
                                    <RxCross1
                                        onClick={(event) =>
                                            handleMenu(event)
                                        }
                                        size={38}
                                        className="text-black rounded-full
cursor-pointer "
                                    />
                                ) : (
                                    <FiMenu
                                        onClick={(event) =>
                                            handleMenu(event)
                                        }
                                        size={38}
                                        className="text-black rounded-full
cursor-pointer "
                                    />
                                )}
                                <p className="relative text-[12px] font-bold
text-center text-black bottom--10">
                                    MENU
                                </p>
                            </div>
```

Κατάσταση menuToggle: Το menuToggle είναι ένα Boolean state που διαχειρίζεται αν το μενού είναι ανοικτό ή κλειστό. Ανάλογα με την κατάσταση του, αλλάζει το εικονίδιο που εμφανίζεται. Όταν το menuToggle είναι true εμφανίζεται το εικονίδιο RxCross1, το οποίο υποδηλώνει ότι το μενού είναι ανοιχτό και προσφέρει τη δυνατότητα κλεισίματος του μενού, ενώ όταν το menuToggle είναι false, εμφανίζεται το εικονίδιο FiMenu, το οποίο υποδηλώνει ότι το μενού είναι κλειστό και προσφέρει τη δυνατότητα ανοίγματος του μενού.

Και τα δύο εικονίδια έχουν τον ίδιο event handler onClick που καλεί τη συνάρτηση handleMenu(event). Αυτή η συνάρτηση αντιστρέφει την τιμή του menuToggle, επιτρέποντας έτσι την εναλλαγή μεταξύ της ανοιχτής και κλειστής κατάστασης του μενού.

4.2.6 Είσοδος Αναζήτησης Προϊόντος

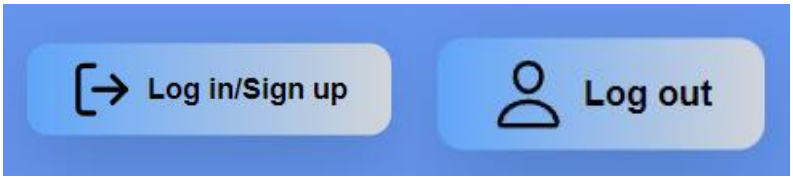
Εικόνα 4.9

Παρακάτω εμφανίζεται το πεδίο εισόδου για την αναζήτηση προϊόντων, με ενσωματωμένη λειτουργικότητα για την ενεργοποίηση της αναζήτησης με βάση την αναζήτηση του χρήστη.

```
<input
                                type="text"
                                placeholder="Search product..."
                                // disabled={!searchDisabled}
                                value={searchValue}
                                onChange={(e) => {
                                    setsearchValue(e.target.value);
                                    dispatch(
                                        getSearchedItem(e.target.value)
                                    );
                                }}
                                onKeyDown={(e) => {
                                    if (e.key === "Enter") handleResults();
                                }}
                                className="focus:outline-none text-[12px] w-
[100%] sm:text-base rounded-lg sm:rounded-xl px-3 py-2 sm:py-3"
                            />
```

Το πεδίο εισόδου χρησιμοποιείται για την εισαγωγή κειμένου από τον χρήστη, στοχεύοντας στην αναζήτηση προϊόντων. Το (placeholder) παρέχει έναν προεπιλεγμένο κείμενο, ενθαρρύνοντας τους χρήστες να εισάγουν το επιθυμητό κριτήριο αναζήτησης. Σύμφωνα τώρα με τις ενέργειες του χρήστη λειτουργούν τα εξής: Η ενέργεια onChange καταγράφει κάθε αλλαγή στην τιμή της εισόδου και ενημερώνει το searchValue μέσω της συνάρτησης setsearchValue. Επίσης, καλεί τη συνάρτηση dispatch(getSearchedItem(e.target.value)), η οποία αναθέτει την τιμή αναζήτησης για να ενημερώσει τα σχετικά δεδομένα των προϊόντων που ταιριάζουν με τα κριτήρια αναζήτησης.

4.2.7 Κουμπιά Λογαριασμού: Είσοδος και Έξοδος

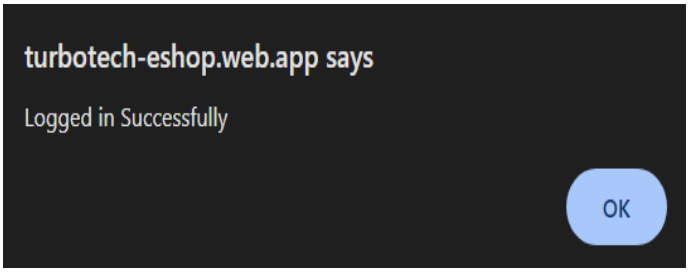


Εικόνα 4.10

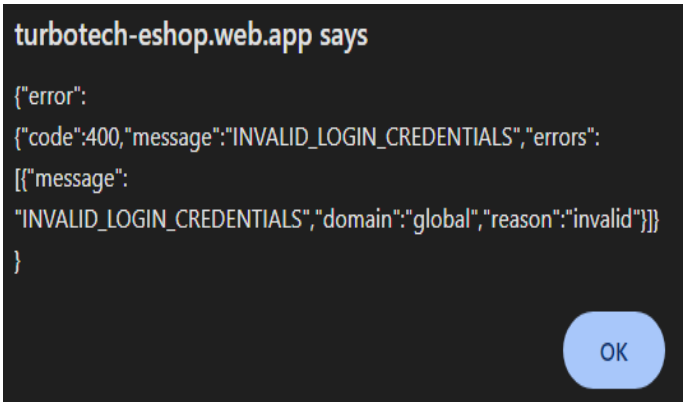
Αυτά τα κουμπιά επιτρέπουν στους χρήστες να διαχειρίζονται την κατάσταση της σύνδεσής τους στην εφαρμογή, είτε θέλουν να συνδεθούν, να αποσυνδεθούν ή να δημιουργήσουν τον λογαριασμό τους.

```
<button
                                    onClick={(event) =>
                                        handleAccount(event)
                                    }
                                    className="flex items-center justify-
center rounded-xl bg-gradient-to-r from-blue-400 to-gray-300 hover:from-gray-300
hover:to-blue-400 px-4 py-2 text-black font-semibold shadow-lg"
                                >
                                    <IoPersonOutline
                                        size={35}
                                        className="mr-2 text-black cursor-
pointer"
                                    />
                                    <span>Log out</span>
                                </button>
<button
                                    onClick={(event) =>
                                        handleAccount(event)
                                    }
                                    className="flex items-center justify-
center rounded-xl bg-gradient-to-r from-blue-400 to-gray-300 hover:from-gray-300
hover:to-blue-400 px-4 py-2 text-black font-semibold shadow-lg"
                                >
                                    <LuLogOut
                                        size={35}
                                        className="mr-2 text-black cursor-
pointer"
                                    />
                                    <span>Log in/Sign up</span>
                                </button>
```

Και τα δύο κουμπιά εφαρμόζουν την ίδια λειτουργία handleAccount όταν γίνεται κλικ, η οποία διαχειρίζεται την είσοδο ή την έξοδο του χρήστη ανάλογα με την τρέχουσα κατάσταση του λογαριασμού (uid). Το εικονίδιο IoPersonOutline χρησιμοποιείται για την ένδειξη της έξοδου (log out), ενώ το LuLogOut για την είσοδο (log in/sign up), παρέχοντας ένα οπτικό σήμα για τη λειτουργία του κουμπιού.

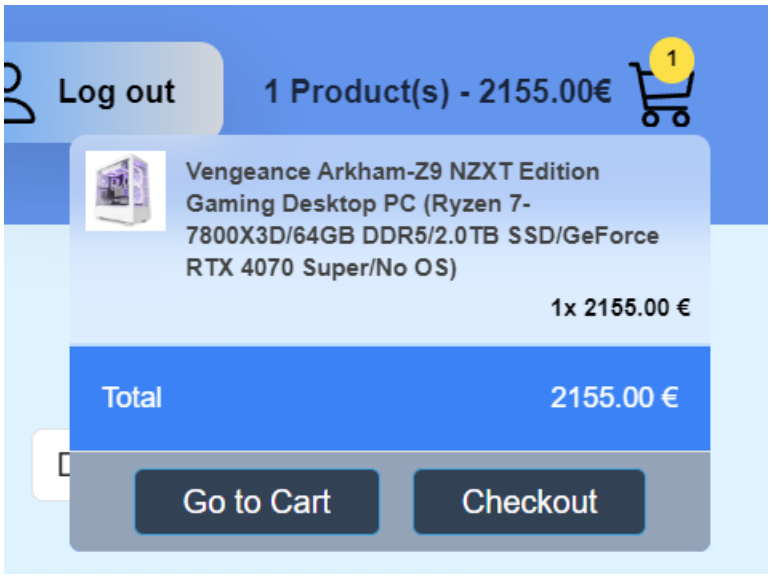


Εικόνα 4.11



Εικόνα 4.12

Σύνδεσμος Προς το Καλάθι Αγορών



Εικόνα 4.13

Ο παρακάτω κώδικας απεικονίζει τον τρόπο σύνδεσης του χρήστη με το καλάθι αγορών της εφαρμογής, παρέχοντας οπτική ενημέρωση για τον αριθμό των προϊόντων και το συνολικό κόστος. Αυτό το στοιχείο ενσωματώνει διαδραστικότητα και οπτική πληροφόρηση για τους χρήστες, επιτρέποντας εύκολη πρόσβαση και διαχείριση των περιεχομένων του καλαθιού αγορών τους.

```
<Link
                                    to="/mycart"
                                    className=" flex items-center hover:no-
underline "
                                >
                                    <span className="text-opacity-75 text-
black mx-2 hidden sm:block">
                                        {cartCounter} Product(s) -{" "}
                                        {ccyFormat(totalPrice)}€{" "}
                                    </span>
                                    <BsCart3
                                        // onClick={() =>
dispatch(handleOpen( ))}
                                        size={35}
                                        className="text-black cursor-pointer"
                                    />

                                    <span
                                        className={`${
                                            cartCounter > 0
                                                ? "block"
                                                : "hidden"
                                        } relative py-[3px] px-[8px] text-
[11px] text-black bg-yellow-300 rounded-full right-6 bottom-4`}
                                    >
                                        {cartCounter}
                                    </span>
                                </Link>
```

Ο σύνδεσμος (Link) οδηγεί τον χρήστη στη σελίδα /mycart, όπου μπορεί να δει τα προϊόντα στο καλάθι του και να προχωρήσει σε αγορά. Εμφανίζει τον αριθμό των προϊόντων και το συνολικό κόστος, παρέχοντας μια άμεση κατανόηση της τρέχουσας κατάστασης του καλαθιού. Η χρήση της συνάρτησης ccyFormat βοηθά στη μορφοποίηση της τιμής σε ευρώ.

Το εικονίδιο BsCart3 αναπαριστά το καλάθι αγορών, διευκολύνοντας την οπτική αναγνώριση της λειτουργίας από τους χρήστες.

4.2.8 Κατάσταση Καλαθιού Αγορών και Επιλογές Πληρωμής

Εικόνα 4.14

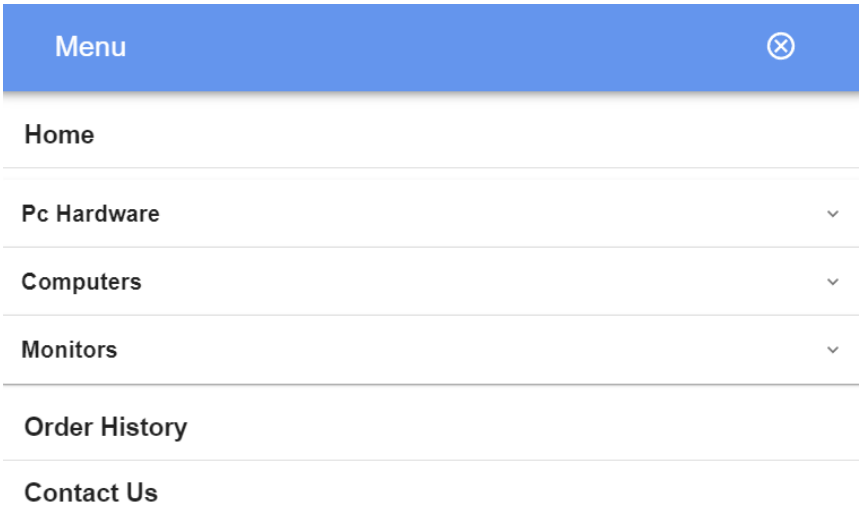
Το απόσπασμα κώδικα που ακολουθεί δείχνει την προβολή των στοιχείων εντός του καλαθιού αγορών και παρέχει δύο κουμπιά για ενέργειες που ο χρήστης μπορεί να επιλέξει: μετάβαση στο καλάθι ή ολοκλήρωση αγοράς. Έτσι ο χρήστης μπορεί άμμεσα να δεί το καλάθι του ή να ανακατευθυνθεί στην σελίδα πληρωμής.

```
<div className="bg-blue-500 rounded-b-md">
                                                <Table>
                                                    <TableRow>
                                                        <TableCell
                                                            sx={{
                                                                fontSize: 14,
                                                            }}
                                                        >
                                                            Total
                                                        </TableCell>
                                                        <TableCell
                                                            align="right"
                                                            sx={{
                                                                fontSize: 14,
                                                            }}
                                                        >
                                                            {ccyFormat(
                                                                totalPrice
                                                            )}{" "}
                                                            €
                                                        </TableCell>
                                                    </TableRow>
                                                </Table>

                                                <div className="d-flex justify-
center bg-slate-400 backdrop-blur-md rounded-b-md">
                                                    {" "}
                                                    <Button
                                                        type="button"
                                                        onClick={() =>
                                                            history.push(
```

```
                                                    "/mycart",
                                                    {
                                                        price:
totalPrice,
                                                        data: row,
                                                    }
                                                )
                                            }
                                            className={` px-4 py-1 m-
2 font-medium rounded-lg bg-slate-700 border-[#40A2D8] hover:bg-slate-900 `}
                                        >
                                            Go to Cart
                                        </Button>
                                        <Button
                                            type="button"
                                            onClick={() =>
                                                history.push(
                                                    "/checkout",
                                                    {
                                                        price:
totalPrice,
                                                        data: row,
                                                    }
                                                )
                                            }
                                            className={` px-4 py-1 m-
2 font-medium  rounded-lg bg-slate-700 border-[#40A2D8] hover:bg-slate-900 `}
                                        >
                                            Checkout
                                        </Button>
                                    </div>
                                </div>
```

Εάν υπάρχουν προϊόντα στο καλάθι, εμφανίζεται ένας πίνακας (Table) με το συνολικό ποσό των αγορών και δύο κουμπιά: ένα για μετάβαση στο καλάθι και ένα για ολοκλήρωση της αγοράς. Το πρώτο κουμπί οδηγεί στη σελίδα του καλαθιού, ενώ το δεύτερο κουμπί οδηγεί απευθείας στη σελίδα ολοκλήρωσης αγοράς (checkout), μεταφέροντας τις απαραίτητες πληροφορίες για τα προϊόντα και το συνολικό κόστος.

4.2.9 Πλευρικό Μενού Πλοήγησης (Sidebar)



Εικόνα 4.15

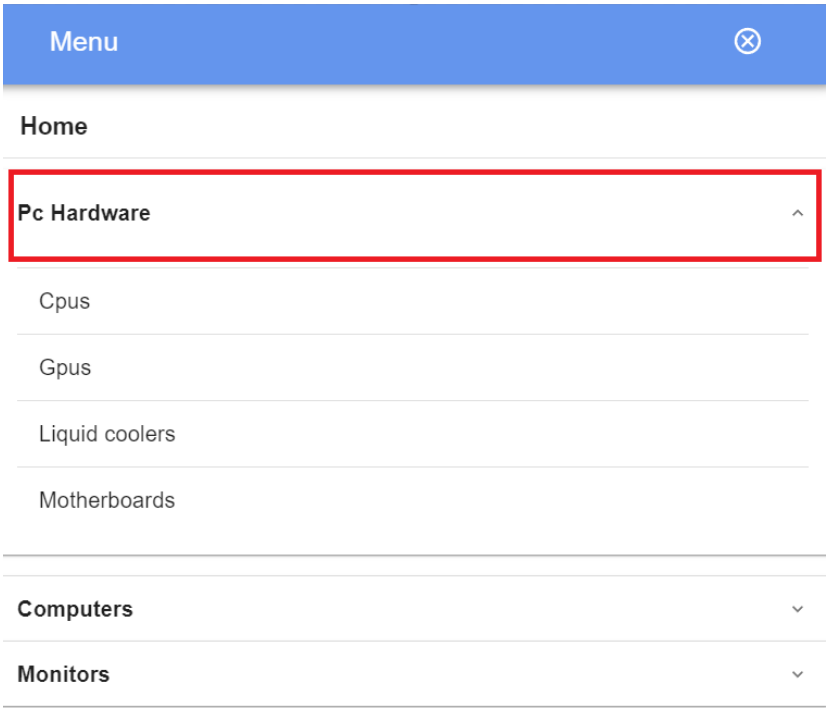
Το παρακάτω απόσπασμα κώδικα δημιουργεί ένα πλευρικό μενού πλοήγησης (sidebar) στην εφαρμογή, το οποίο εμφανίζεται ή εξαφανίζεται ανάλογα με την κατάσταση της μεταβλητής menuToggle. Ουσιαστικά είναι ένα πολύ βασικό κομμάτι της εφαρμογής καθώς από εκεί μπορούν να περιηγηθούν οι χρήστες στις διάφορες κατηγορίες προϊόντων που διαθέτει η εφαρμογή.

```
{/* Menu Sidebar */}
            <div
                ref={menuRef}
                className={`${
                    menuToggle ? "left-0" : "-left-[100vh]"
                }  overflow-hidden  py-10 border-r-2 border-[#6495ED] bg-white
fixed z-10 top-[100px] transition-all duration-300 w-[300px] px-3 h-full  flex-
col gap-y-24`}
            >
```

Δυναμική Τοποθέτηση:

Το className χρησιμοποιεί template literals για να διαχειριστεί τη θέση του πλευρικού μενού. Αν το menuToggle είναι true, το μενού εμφανίζεται στο left: 0, κάνοντας το πλήρως ορατό. Αν είναι false, το μενού κινείται αριστερά έξω από την οθόνη (-left-[100vh]), κρύβοντας το πλήρως από τον χρήστη.

4.2.10 Σύνδεσμοι Κατηγοριών Προϊόντων στο Πλευρικό Μενού



Εικόνα 4.16

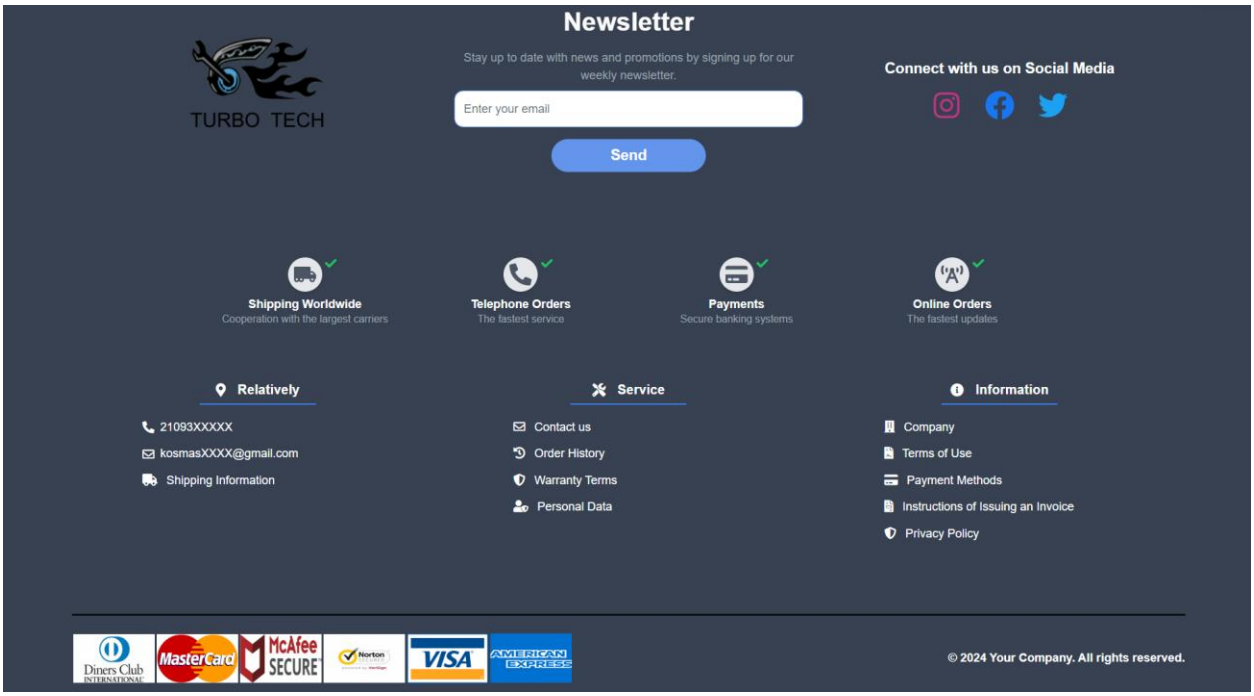
Αποτελεί μέρος του μενού που παρέχει πρόσβαση σε διάφορες κατηγορίες υλικού υπολογιστή. Κάθε σύνδεσμος περιλαμβάνει μια ενέργεια για την απόκρυψη του μενού μετά την επιλογή. Περιλαμβάνονται επιπλέον σύνδεσμοι οι οποίοι δεν έχουν προστεθεί εδώ για τους προφανής λόγους.

```
<div className="flex flex-col p-2 mt-2 text-sm gap-y-5">
                            <Link
                                to="/sort/pc-hardware/motherboards"
                                className="hover:no-underline"
                                onClick={() => setMenuToggle(false)}
                            >
                                <h6 className="font-semibold cursor-pointer
hover:text-[#6495ED]">
                                    Motherboards
                                </h6>
                            </Link>
                            <Link
                                to="/sort/pc-hardware/CPUs"
                                className="hover:no-underline"
                                onClick={() => setMenuToggle(false)}
                            >
                                {" "}
                                <h6 className="font-semibold cursor-pointer
hover:text-[#6495ED]">
```

```
                        CPUs
                    </h6>
                </Link>
```

Οι σύνδεσμοι (Link) διαθέτουν την ενέργεια onClick που καλεί τη συνάρτηση setMenuToggle(false), ώστε να κλείσει το μενού κατά την επιλογή μιας κατηγορίας, βελτιώνοντας την εμπειρία χρήσης και ελαχιστοποιώντας την ανάγκη για περαιτέρω κλικ. Η κλάση hover:no-underline αφαιρεί την υπογράμμιση από τους συνδέσμους κατά τη διάρκεια της αιώρησης, κάνοντας την εμφάνιση πιο καθαρή.

4.3 FOOTER



Εικόνα 4.17

4.3.1 Εμφάνιση Λογοτύπου σε Στήλη Διάταξης

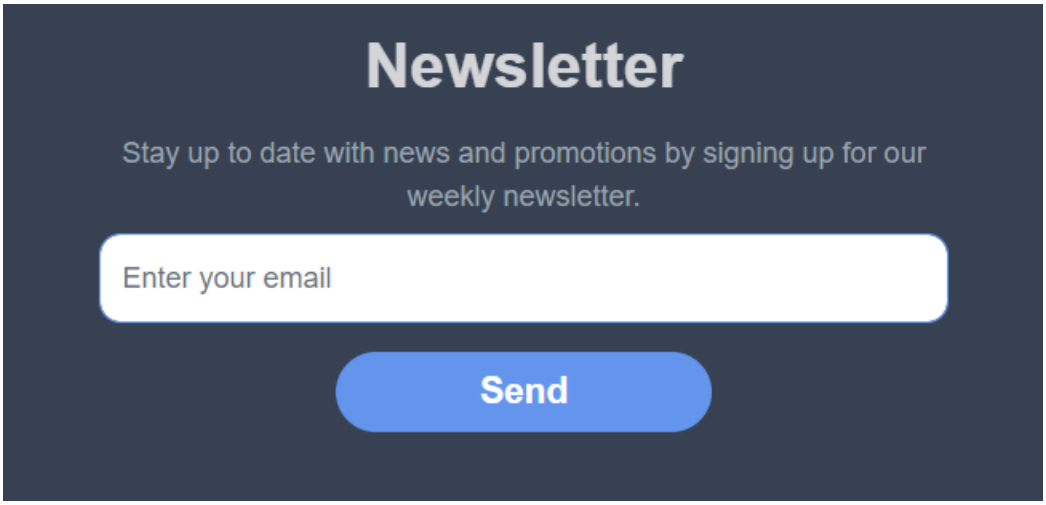
Εικόνα 4.18

Το παρακάτω κομμάτι κώδικα αποτελεί ένα τμήμα της διεπαφής χρήστη που διαχειρίζεται την εμφάνιση του λογοτύπου.

```
{/* Logo */}
                    <Col md={4} className="mb-4">
                        <img
                            src={logo}
                            alt="Logo"
                            className="img-fluid w-[200px] mx-auto"
                        />
                    </Col>
```

src={logo}: Ορίζει την πηγή της εικόνας λογοτύπου.

4.3.2 Φόρμα Εγγραφής σε Newsletter



Εικόνα 4.19

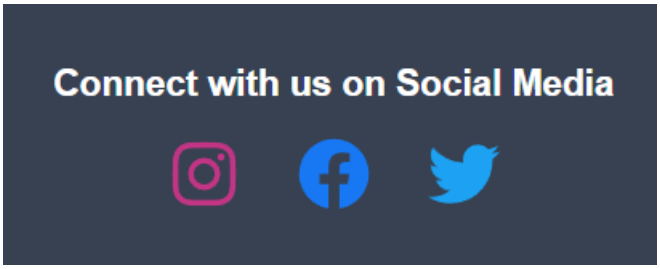
Εδώ προσφέρεται η διεπαφή του χρήστη και διαχειριστή που προσφέρει στους χρήστες τη δυνατότητα να εγγραφούν στο newsletter. Το στοιχείο Col από το Bootstrap χρησιμοποιείται για τη δημιουργία ενός απομονωμένου τμήματος σε μορφή στήλης.

```
<Col md={4} className="mb-3 text-center">
                        <form>
                            <h3 className="font-bold text-[35px] mb-3 mt-2
animate-pulse">
                                Newsletter
                            </h3>
                            <p class="text-gray-400 mb-2">
                                Stay up to date with news and promotions by
                                signing up for our weekly newsletter.
                            </p>
                            <Input
                                type="email"
                                placeholder="Enter your email"
                                value={email}
                                onChange={(e) => setEmail(e.target.value)}
                                className="mb-2 py-6 border-[#6495ED] rounded-xl"
                            />
                            <div className="flex justify-between mt-3">
                                <button
                                    className="bg-[#6495ED] mx-auto font-bold px-
[80px] py-2 md:py-3 text-white rounded-full text-xl hover:cursor-pointer"
                                    onClick={handleSubmit}
                                >
                                    Send
                                </button>
                            </div>
                        </form>
                    </Col>
```

Input: Το στοιχείο εισαγωγής τύπου email με την ιδιότητα (placeholder) που προτρέπει τον χρήστη να εισάγει το email του.

button: Το κουμπί που εκτελεί την ενέργεια υποβολής της φόρμας με το onClick={handleSubmit} για την επεξεργασία των δεδομένων εγγραφής.

4.3.3. Σύνδεσμοι Κοινωνικών Μέσων



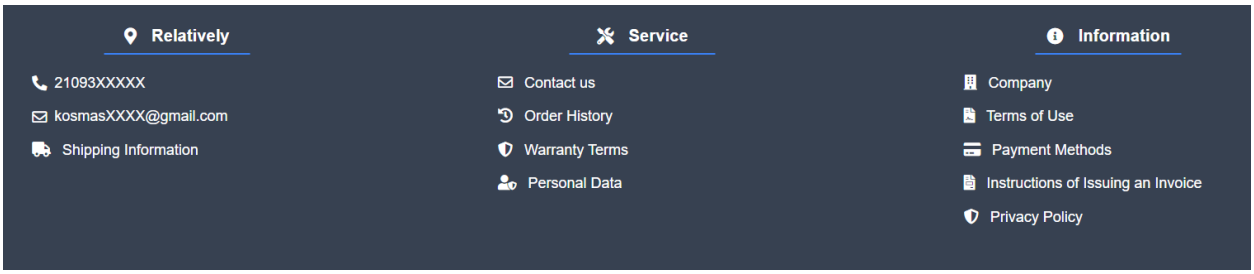
Εικόνα 4.20

Εδώ δημιουργείτε ένα μέρος με εικονίδια για συνδέσμους στα κοινωνικά δίκτυα στο κέντρο της εφαρμογής, επιτρέποντας την άμμεση επικοινωνία με την εταιρεία μέσω των κοινωνικών πλατφορμών.

```
{/* Social Media Links */}
                    <Col md={4} className="mb-3 gap-x-3">
                        <p className="mt-2 mb-3 text-xl font-bold text-center">
                            Connect with us on Social Media
                        </p>
                        <div className="flex justify-center items-center">
                            <FaInstagram
                                size={40}
                                className="mx-3 text-[#C13584] hover:text-
[#E4405F] transition-colors duration-300 hover:cursor-pointer"
                                title="Instagram"
                            />
                            <FaFacebook
                                size={40}
                                className="mx-3 text-[#1877F2] hover:text-
[#4267B2] transition-colors duration-300 hover:cursor-pointer"
                                title="Facebook"
                            />
                            <FaTwitter
                                size={40}
                                className="mx-3 text-[#1DA1F2] hover:text-
[#36C5F0] transition-colors duration-300 hover:cursor-pointer"
                                title="Twitter"
                            />
                        </div>
                    </Col>
```

Τα εικονίδια FaInstagram, FaFacebook, και FaTwitter από τη βιβλιοθήκη FontAwesome χρησιμοποιούνται για να δημιουργήσουν οπτικά διακριτικά για την πρόσβαση στα αντίστοιχα κοινωνικά δίκτυα.

4.3.4 Ενότητα Πληροφοριών



Εικόνα 4.21

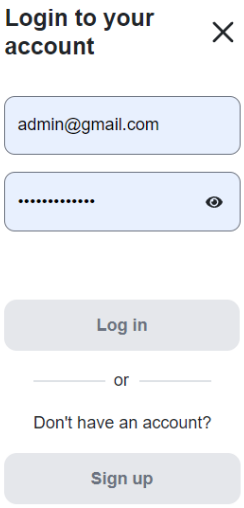
Το ακόλουθο κομμάτι κώδικα παρουσιάζει μια ενότητα πληροφοριών μέσα σε μια στήλη του Bootstrap, προσφέροντας στον χρήστη γρήγορη πρόσβαση σε διάφορες σημαντικές σελίδες όπως πληροφορίες εταιρίας, όροι χρήσης, μέθοδοι πληρωμής, οδηγίες για την έκδοση τιμολογίου και πολιτική απορρήτου.

```
<li className="text-base flex items-center">
                                    <Link to="/terms-of-use">
                                        <i className="fas fa-file-contract mr-
2"></i>{" "}
                                        Terms of Use
                                    </Link>
                                </li>
                                <li className="text-base flex items-center">
                                    <Link to="/payment-methods">
                                        <i className="fas fa-credit-card mr-
2"></i>{" "}
                                        Payment Methods
                                    </Link>
                                </li>
```

Κάθε σύνδεσμος παρέχει γρήγορη πρόσβαση σε σημαντικές σελίδες εντός της ιστοσελίδας, με κάθε στοιχείο της λίστας να έχει ένα σχετικό εικονίδιο που αυξάνει την αναγνωρισιμότητα της συνδεδεμένης λειτουργίας.

4.4 LOGIN/REGISTER

4.4.1 Ανάλυση Κώδικα για Κεφαλίδα Φόρμας Σύνδεσης



Εικόνα 4.22

Η σύνδεση (Login) που παρουσιάζετε χρησιμοποιείται για τη διαχείριση της διαδικασίας σύνδεσης χρηστών σε μια εφαρμογή.

```
const Login = ({handleAccount, handleShowLogin, setAccount, reference}) => {

    const { authLoading } = useSelector((state) => state.authUser);
    const dispatch = useDispatch();
    const [showPassword, setShowPassword] = useState(false);
    const [formData, setFormData] = useState({
      email: "",
      password: "",
  });

  const handleChange = (e) => {
      const { name, value } = e.target;
      setFormData({
          ...formData,
          [name]: value,
      });
  };


  const toggleShowPassword = (event) => {
    event.stopPropagation();
    setShowPassword(!showPassword);
};
```

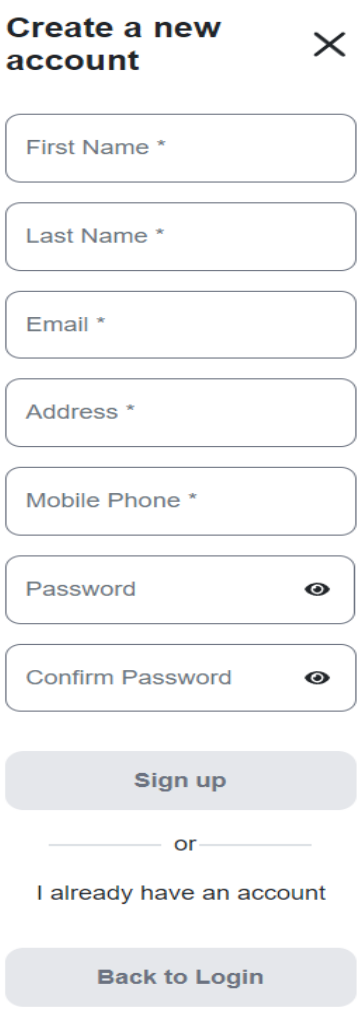
authLoading: Αντλείται από το Redux store και χρησιμοποιείται για να ενημερώσει την UI για την κατάσταση φόρτωσης της διαδικασίας ελέγχου ταυτότητας.

formData: Κρατά τις τρέχουσες τιμές των πεδίων της φόρμας σύνδεσης, περιλαμβάνοντας το email και τον κωδικό πρόσβασης.

showPassword: Ένα boolean state που διαχειρίζεται την εμφάνιση του κωδικού πρόσβασης (κρυφό ή ορατό).

```
                <div className="flex items-center justify-between">
                    <h1 className="font-bold text-[18px] md:text-[22px]">
                        Login to your account
                    </h1>
                    <RxCross2
                        onClick={handleAccount}
                        size={40}
                        className="cursor-pointer"
                    />
                </div>
```

4.4.2 Ανάλυση Συνάρτησης Υποβολής Εγγραφής Χρήστη σε React



Εικόνα 4.23

Η παρακάτω συνάρτηση handleSubmitSignup αποτελεί τη λογική που διαχειρίζεται την υποβολή φόρμας εγγραφής

```
const handleSubmitSignup = (e) => {
        e.preventDefault();
        console.log({ formDataSignup });
        dispatch(
            registerUser(formDataSignup, () => {
                setFormDataSignup({
                    firstName: "",
                    lastName: "",
                    mobileNo: "",
                    email: "",
                    password: "",
```

```
                    confirmPassword: "",
                    address: "", // Reset address field after submission
                });
                setShowLogin(false);
            })
        );
    };
```

console.log({ formDataSignup }): Καταγράφει τα δεδομένα της φόρμας στην κονσόλα, παρέχοντας επιβεβαίωση των στοιχείων που εισήχθησαν από τον χρήστη πριν την επεξεργασία.

dispatch(registerUser(formDataSignup, callback)): Αποστέλλει μια ενέργεια στο Redux store για την εγγραφή του χρήστη. Η ενέργεια περιλαμβάνει τα δεδομένα φόρμας και μια callback συνάρτηση που εκτελείται μετά την επιτυχή εκτέλεση της εγγραφής.

4.5 ΜΕΝΟΥ-ΚΕΝΤΡΙΚΗ ΣΕΛΙΔΑ

4.5.1 Ανάλυση Συνιστώσας Home για Ηλεκτρονικό Κατάστημα

Το Home που παρουσιάζετε αποτελεί την κύρια σελίδα ενός ηλεκτρονικού καταστήματος. Χρησιμοποιεί το React, μαζί με Redux για τη διαχείριση της κατάστασης. Εδώ παρέχεται μια επισκόπηση της λειτουργικότητας και της δομής της συνιστώσας.

```
const Home = () => {
    const history = useHistory();
    const [products, setProducts] = useState([]);
    const dispatch = useDispatch();
    const { allProducts, productsLoading, favoritesProducts, searchValue } =
        useSelector((state) => state.products);
    const { user } = useSelector((state) => state.authUser);
    const [discountCards, setDiscountCards] = useState([]);
    const [allCategoriesProduct, setAllCategoriesProduct] = useState([]);
    console.log(allCategoriesProduct, "categoriis");
    const [searchData, setSearchData] = useState([]);
    const handleAddToCart = (product) => {
        dispatch(addToCart(product));
    };
```

useHistory: Χρησιμοποιείται για τη διαχείριση της ιστορίας πλοήγησης.

useState: Δημιουργία τοπικής κατάστασης για products, discountCards, allCategoriesProduct, και searchData.

useDispatch και useSelector: αποστολή ενεργειών και την επιλογή δεδομένων από το global state.

allProducts, productsLoading, favoritesProducts, searchValue: Μεταβλητές από το Redux store που χρησιμοποιούνται για να εμφανίσουν προϊόντα, να δείχνουν αν τα προϊόντα φορτώνονται, να διαχειριστούν τα αγαπημένα, και τις αναζητήσεις.

handleAddToCart: Μια συνάρτηση που διαχειρίζεται την προσθήκη προϊόντων στο καλάθι αγορών. Καλεί τη δράση addToCart με το Redux dispatch.

4.5.2 Ανάλυση useEffect για Ενημέρωση Προϊόντων και Κατηγοριών

Η χρήση του useEffect χειρίζεται την ενημέρωση των προϊόντων με βάση τα αγαπημένα και την ταξινόμηση τους με κριτήριο την έκπτωση. Επίσης, διαχειρίζεται την ενημέρωση των κατηγοριών προϊόντων.

```
useEffect(() => {
        if (allProducts?.length > 0) {
            const updatedProducts = allProducts.map((product) => {
                const isFavorite = favoritesProducts.some(
                    (favProduct) => favProduct.productID === product.id
                );
                return { ...product, isFav: isFavorite };
            });
            const sortedProducts = updatedProducts.sort(
                (a, b) => b.discount - a.discount
            );
            setDiscountCards(sortedProducts.slice(0, 3));
            setProducts(updatedProducts);
            const categories = [
                ...sortedProducts.slice(0, 3),
                ...updatedProducts,
            ];
            setAllCategoriesProduct(categories);
        } else {
            setDiscountCards([]);
            setProducts([]);
        }
    }, [allProducts, favoritesProducts]);
```

Ενημέρωση Αγαπημένων: Διατρέχει τα allProducts και ενημερώνει κάθε προϊόν αν ανήκει στα αγαπημένα (favoritesProducts), προσθέτοντας μια νέα ιδιότητα isFav για το κάθε προϊόν.

Ταξινόμηση και Επιλογή Προϊόντων: Ταξινομεί τα updatedProducts με βάση την έκπτωση (discount) από το μεγαλύτερο στο μικρότερο. Επιλέγει τα τρία πρώτα προϊόντα με την μεγαλύτερη έκπτωση και τα αποθηκεύει στο setDiscountCards.

Ενημέρωση Κατηγοριών Προϊόντων: Δημιουργεί ένα σύνολο από κατηγορίες προϊόντων από τα επιλεγμένα και τα ανανεωμένα προϊόντα και τα αποθηκεύει χρησιμοποιώντας το setAllCategoriesProduct.

4.5.3 Ανάλυση Χρήσης useEffect για Φιλτράρισμα Προϊόντων Βάσει Αναζήτησης

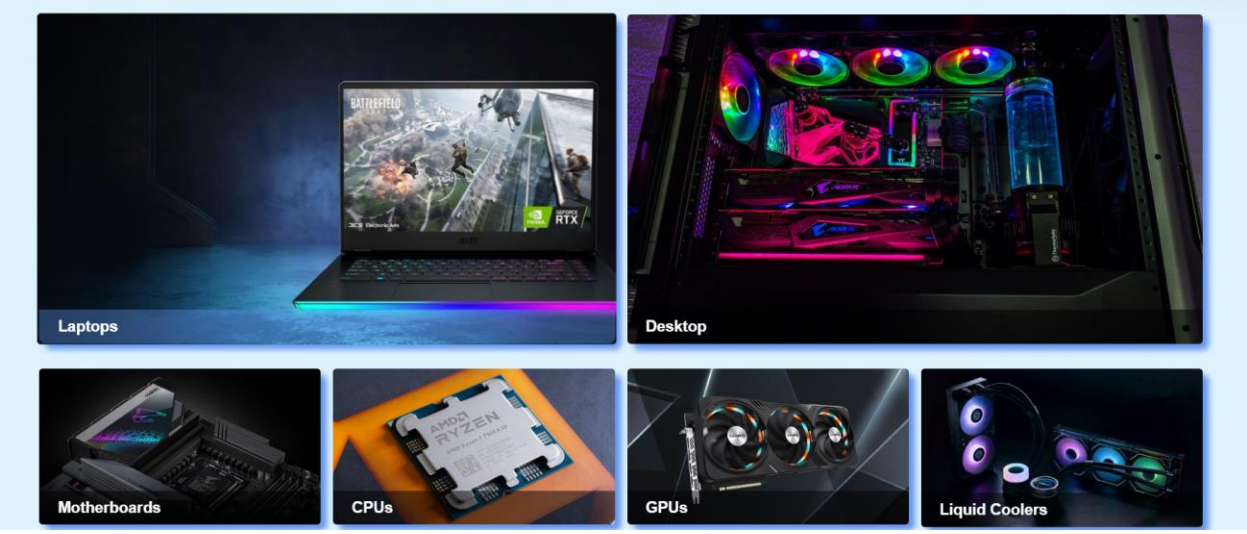
Ενημερώνει δυναμικά τα αποτελέσματα αναζήτησης προϊόντων με βάση την εισαγωγή αναζήτησης του χρήστη. Αυτή η διαδικασία βοηθά στη διασφάλιση ότι μόνο τα σχετικά προϊόντα εμφανίζονται στη διεπαφή χρήστη, βελτιώνοντας την αποδοτικότητα της αναζήτησης.

```
useEffect(() => {
        let uniqueIds = new Set();
        let searchedData = allCategoriesProduct?.filter((product) => {
            console.log(product.title, "product");
            const lowerCaseTitle = product.title?.toLowerCase();
            if (
                !uniqueIds.has(product.id) &&
                lowerCaseTitle.includes(searchValue?.toLowerCase())
            ) {
                uniqueIds.add(product.id);
                return true;
            }
            return false;
        });

        console.log(searchedData, "searchData");
        setSearchData(searchedData);
    }, [searchValue, allCategoriesProduct]);
```

Εκτελείται ένα φιλτράρισμα στη λίστα allCategoriesProduct για να βρεθούν προϊόντα των οποίων οι τίτλοι περιέχουν την αναζητούμενη λέξη, μετατρεπόμενη σε πεζά γράμματα για να αποφευχθούν οι διακρίσεις κεφαλαίων/πεζών.

Ενημέρωση Κατάστασης Αναζήτησης: setSearchData χρησιμοποιείται για να ενημερώσει την κατάσταση με τα φιλτραρισμένα δεδομένα, εξασφαλίζοντας ότι μόνο τα προϊόντα που ανταποκρίνονται στην αναζήτηση θα εμφανίζονται.

4.5.4 Εικόνες Ανακατεύθυνσης



Εικόνα 4.24

```
{
            img: RED_LAPTOP,
            name: "Laptops",
            value: "/sort/computers/laptops",
        },
        {
            img: RED_DESTOP,
            name: "Desktop",
            value: "/sort/computers/desktop-pcs",
        },
```

Διαχειρίζεται την προβολή των δύο πρώτων εικόνων από μια λίστα σε μορφή καρτών με επικάλυψη. Η δομή χρησιμοποιεί το grid system για να παρέχει μια ευέλικτη και αποκρισιμότητα στον σχεδιασμό.

```
{/* First two images */}
                            <div className="grid grid-cols-1 md:grid-cols-2 gap-4
mb-8 mt-4">
                                {cards.slice(0, 2).map((item, idx) => (
                                    <div key={idx} className="relative">
                                        <div className="card-wrapper">
                                            <img
                                                src={item.img}
                                                alt={item.name}
                                                className="rounded-lg cursor-
pointer transition duration-300 transform hover:scale-105"
                                                onClick={() =>
                                                    history.push(item.value)
                                                }
                                            />
```

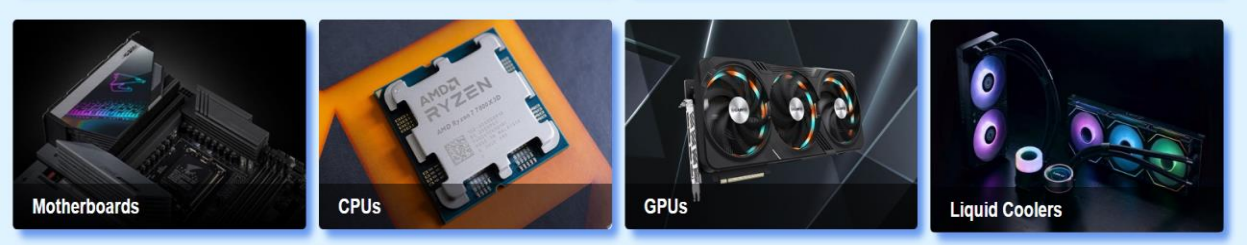
```
                                        <div className="overlay absolute
bottom-0 left-0 right-0 bg-black bg-opacity-40 px-4 py-2 rounded-b-lg">
                                            <h3 className="text-white text-xl
font-bold">
                                                {item.name}
                                            </h3>
                                        </div>
                                    </div>
                                </div>
                            ))}
                        </div>
```

cards.slice(0, 2): Επιστρέφει τις δύο πρώτες κάρτες από τον πίνακα cards.

map((item, idx) => ...): Κάθε στοιχείο και η αντίστοιχη θέση του στον πίνακα χρησιμοποιούνται για να δημιουργήσουν δυναμικά τις απαραίτητες κάρτες.

onClick={() => history.push(item.value)}: Η εικόνα υποστηρίζει μια δράση κλικ που οδηγεί τον χρήστη σε μια νέα διαδρομή, διαφορετική για κάθε κάρτα, βάσει της ιδιότητας item.value.

Ύστερα οι επόμενες 4 σελίδες ανακατεύθυνσης λειτουργούν με τον ίδιο τρόπο.



Εικόνα 4.25

4.5.5 Carousel

Εικόνα 4.26

Ο κώδικας παρουσιάζει τη χρήση του Swiper, μιας δημοφιλούς JavaScript βιβλιοθήκης για τη δημιουργία αποκρισιμών και ελκυστικών sliders. Αυτή η εφαρμογή είναι σχεδιασμένη για να εμφανίζει διαφάνειες (slides) με περιεχόμενο με εναλλασσόμενο τρόπο.

```
<Swiper
                                spaceBetween={30}
                                centeredSlides={true}
                                autoplay={{
                                    delay: 4000,
                                    disableOnInteraction: false,
                                }}
                                pagination={{
                                    clickable: true,
                                }}
                                navigation={true}
                                modules={[Autoplay, Pagination, Navigation]}
                                className="flex items-center justify-center  h-
[700px] max-w-[100%]"
                            >
```

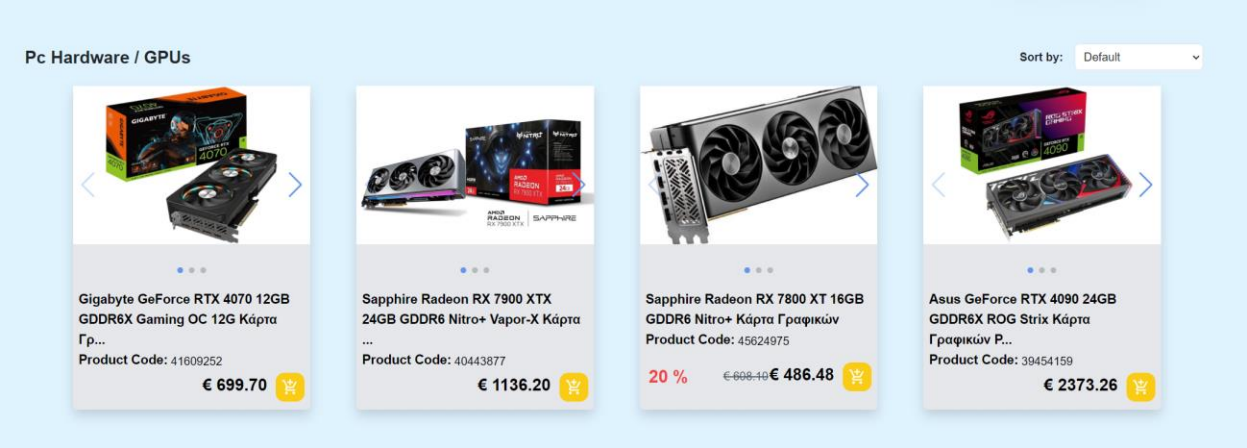
autoplay: Καθορίζει τον αυτόματο κύκλο εμφάνισης των slides με καθυστέρηση 4000ms (4 δευτερόλεπτα) και επιτρέπει τη συνέχιση του autoplay ακόμη και όταν ο χρήστης αλληλεπιδρά με τα slides.
pagination: Προσθέτει σελιδοποίηση με κλικαριστά στοιχεία, επιτρέποντας στους χρήστες να μεταβαίνουν άμεσα σε οποιοδήποτε slide.

navigation={true}: Ενεργοποιεί τα πλήκτρα πλοήγησης (επόμενο/προηγούμενο), προσφέροντας περισσότερες επιλογές για χειροκίνητη αλλαγή των slides.

modules={[Autoplay, Pagination, Navigation]}: Περιλαμβάνει τα απαραίτητα modules που επιτρέπουν τη λειτουργία των προηγούμενων χαρακτηριστικών.

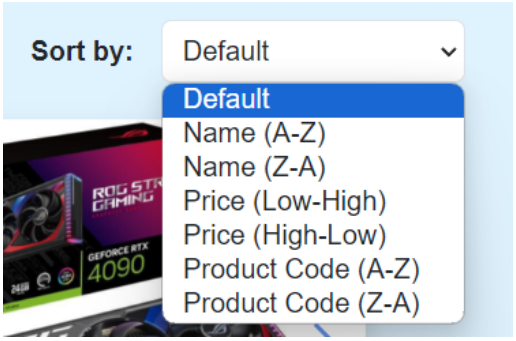
4.6 ΣΕΛΙΔΕΣ ΠΡΟΙΟΝΤΩΝ

4.6.1 ΕΜΑΦΝΙΣΗ ΠΡΟΙΟΝΤΩΝ ΣΕ ΚΑΤΗΓΟΡΙΕΣ



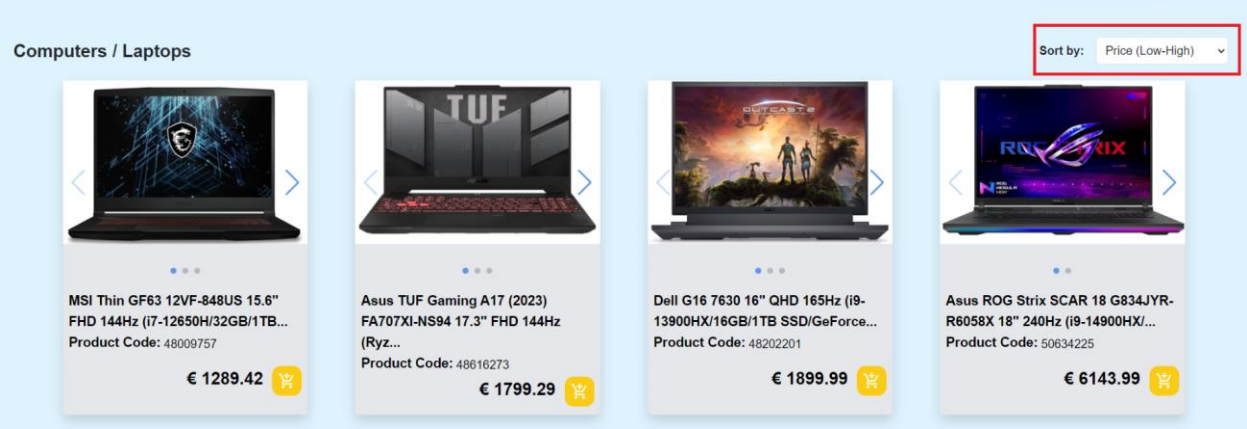
Εικόνα 4.27

Αυτή η λειτουργία επιτρέπει στους χρήστες να διαλέξουν μεταξύ αύξουσας ή φθίνουσας ταξινόμησης των προϊόντων βάσει της τιμής τους.



Εικόνα 4.28

Φιλτράρει και ταξινομεί σε λίστα τα προϊόντα της εφαρμογής με βάση της τιμής αναζήτησης και άλλων επιλεγμένων κριτηρίων ταξινόμησης



Εικόνα 4.29

```
case "price-asc":
                    sortedProducts = updatedProducts.sort(
                        (a, b) => a.price - b.price
                    );
                    break;
                case "price-desc":
                    sortedProducts = updatedProducts.sort(
                        (a, b) => b.price - a.price
                    );
                    break;
```

Όταν επιλεγεί η ταξινόμηση "price-asc", η λειτουργία sort καλείται με ένα callback που συγκρίνει τα πεδία (price) δύο αντικειμένων a και b. Η σύγκριση a.price - b.price οδηγεί σε ταξινόμηση των προϊόντων από το φθηνότερο προς το ακριβότερο.

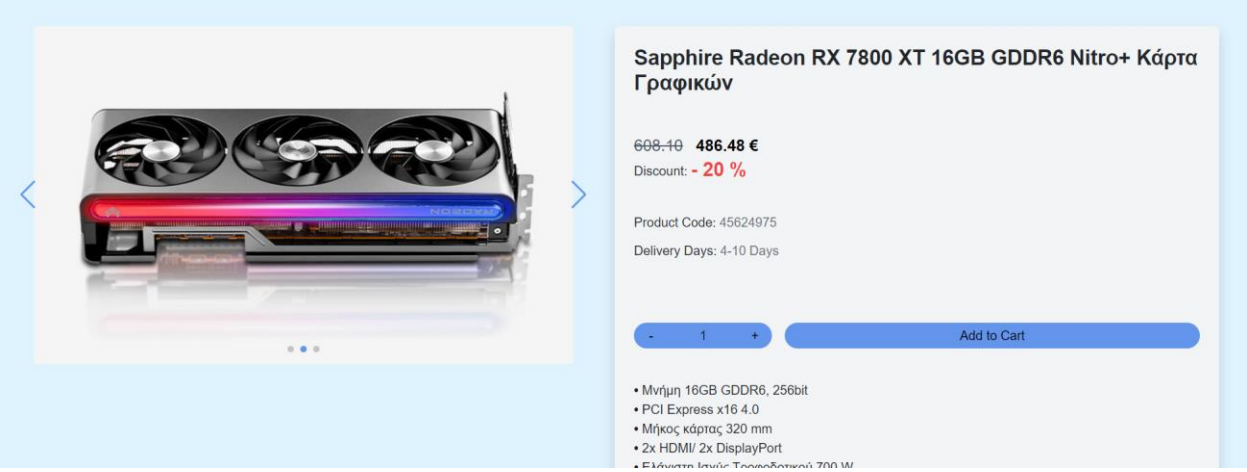
```
// Custom sorting function for product codes
    const compareProductCodes = (a, b, sortOrder) => {
        const codeA = a.productCode.toLowerCase();
        const codeB = b.productCode.toLowerCase();
        if (codeA < codeB) {
            return sortOrder === "asc" ? -1 : 1;
        }
        if (codeA > codeB) {
            return sortOrder === "asc" ? 1 : -1;
        }
```

Η δυνατότητα ταξινόμησης ανά τιμή επιτρέπει στους χρήστες να προσαρμόσουν την προβολή των προϊόντων σύμφωνα με τις προτιμήσεις ή τις ανάγκες τους.

```
// Apply sorting to searchedData
            switch (sort) {
                case "name-asc":
                    searchedData.sort((a, b) => a.title.localeCompare(b.title));
```

```
            break;
        case "name-desc":
            searchedData.sort((a, b) => b.title.localeCompare(a.title));
            break;
        case "price-asc":
            searchedData.sort((a, b) => a.price - b.price);
            break;
        case "price-desc":
            searchedData.sort((a, b) => b.price - a.price);
            break;
        case "code-asc":
            searchedData.sort((a, b) =>
                compareProductCodes(a, b, "asc")
            );
            break;
        case "code-desc":
            searchedData.sort((a, b) =>
                compareProductCodes(a, b, "desc")
            );
            break;
        default:
            // No sorting
            break;
    }
```

4.7 ΚΑΡΤΕΛΑ ΠΡΟΙΟΝΤΩΝ



Εικόνα 4.30

4.7.1 Ανάλυση Κώδικα για Προβολή Κάρτας Προϊόντος στο React

Οι πληροφορίες των προϊόντων κατα την προβολή τους.

```
<Link
                                        to={`/product/${card?.id}`}
                                        className="hover:no-underline"
                                    >
                                        <div
                                            key={card.id}
                                            className="flex flex-col items-
center shadow text-black rounded-md  h-[450px]  w-[330px] bg-gray-200"
                                        >
```

Το Link δημιουργεί έναν υπερσύνδεσμο που οδηγεί στη σελίδα λεπτομερειών του προϊόντος. Η διεύθυνση URL παράγεται δυναμικά χρησιμοποιώντας το id του προϊόντος (card?.id).

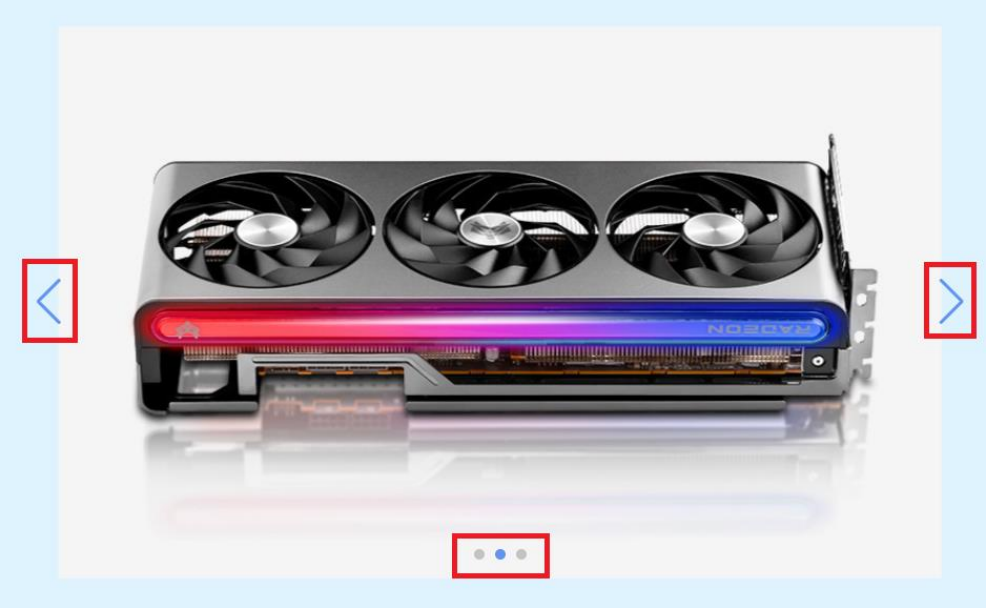
4.7.2 Προσθήκη στα Αγαπημένα

Ο κώδικας που παραθέτετε χρησιμοποιείται για την εμφάνιση ενός εικονιδίου το οποίο διαχειρίζεται τη λειτουργία προσθήκης ενός προϊόντος στα αγαπημένα του χρήστη.

```
GoHeartFill
                                                                    onClick={() =>
                                                                        dispatch(
                                                                            addToFavo
rite(
                                                                                user?
.id,
                                                                                card?
.id
                                                                            )
                                                                        )
                                                                    }
                                                                    size={30}
                                                                    className="text-
yellow-300 cursor-pointer"
                                                                />
```

Κατά το κλικ στο εικονίδιο, καλείται η συνάρτηση dispatch του Redux για την εκτέλεση της ενέργειας addToFavorite. Αυτή η ενέργεια περνάει ως παραμέτρους τον ταυτότητα (ID) του χρήστη (user?.id) και την ταυτότητα του προϊόντος (card?.id).

4.7.3 Παρουσίασης Εικόνων Swiper



Εικόνα 4.31

Προσφέρει προηγμένες λειτουργίες για τη δημιουργία διαδραστικών sliders.

```
<Swiper
                                                    slidesPerView={1}
                                                    pagination={{
                                                        clickable: true,
                                                    }}
                                                    navigation={{
                                                        clickable: true,
                                                    }}
                                                    modules={[
                                                        Navigation,
                                                        Pagination,
                                                        Mousewheel,
                                                        Keyboard,
                                                    ]}
                                                    className="flex items-center
justify-center w-full h-[750px] "
```

```
                                                                >
```

slidesPerView={1}: Καθορίζει ότι θα εμφανίζεται μόνο μία διαφάνεια τη φορά.

pagination και navigation: Επιτρέπουν στους χρήστες να πλοηγούνται μεταξύ των διαφανειών με κλικ, προσθέτοντας διαδραστικότητα.

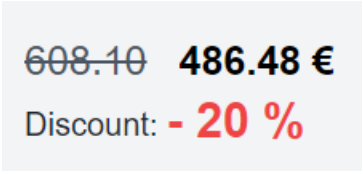
4.7.4 Πληροφοριες προϊοντος

εμφανίζει τον κωδικό ενός προϊόντος. Αυτός ο τρόπος εμφάνισης είναι χρήσιμος για την αναγνώριση και ταξινόμηση προϊόντων μέσα στο σύστημα.

```
<div className="flex">
                                                                {card?.discount >
                                                                    0 && (
                                                                    <>
                                                                         
                                                                        <h1
className="text-2xl font-semibold text-red-500">
                                                                            {
                                                                                card.
discount
                                                                            }{" "}
                                                                            <span
className="font-semibold text-red-500">
                                                                                %
                                                                            </span>
                                                                        </h1>
                                                                    </>
                                                                )}
                                                            </div>
```

Το κείμενο εντός του span αξιολογεί αν υπάρχει κωδικός προϊόντος (card?.productCode). Εάν υπάρχει, εμφανίζεται ο κωδικός, διαφορετικά εμφανίζεται "0".

4.7.5 Εμφάνισης Τιμών και Εκπτώσεων



Εικόνα 4.31

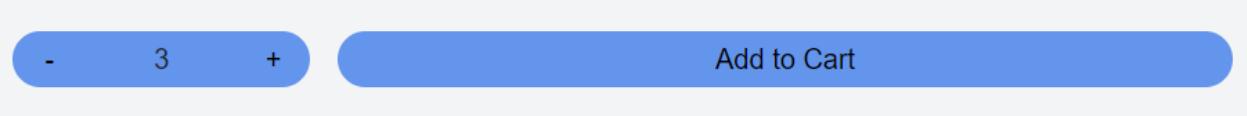
Ο παρακάτω κώδικας αναλύει την παρουσίαση τιμών προϊόντων με και χωρίς έκπτωση, καθώς και την ενδεικτική εμφάνιση του ποσοστού έκπτωσης.

```
<div className="flex">
                                                    {card?.discount >
                                                        0 && (
                                                        <>
                                                             
                                                            <h1
className="text-2xl font-semibold text-red-500">
                                                                {
                                                                    card.
discount
                                                                }{" "}
                                                                <span
className="font-semibold text-red-500">
                                                                    %
                                                                </span>
                                                            </h1>
                                                        </>
                                                    )}
                                                </div>
                                                <div className="flex gap-
x-3">
                                                    {!card?.discount ? (
                                                        <p
className="text-[23px] font-semibold ">
                                                            €{" "}
                                                            {
                                                                card?.pri
ce
                                                            }
                                                        </p>
                                                    )
```

Ο πρώτος div ελέγχει αν το card?.discount είναι μεγαλύτερο από 0. Αν ναι, εμφανίζει το ποσοστό έκπτωσης με ένα διακριτικό κόκκινο χρώμα, υποδηλώνοντας έτσι την ειδική προσφορά στο προϊόν.

Ο δεύτερος div εμφανίζει την κανονική τιμή του προϊόντος αν δεν υπάρχει έκπτωση. Αυτό γίνεται με τη χρήση ενός απλού ελέγχου (!card?.discount) που διαπιστώνει αν το προϊόν δεν έχει εκπτώσεις και στη συνέχεια εμφανίζει την τιμή του.

4.7.6 Προσθήκη Προϊόντος στο Καλάθι Αγορών



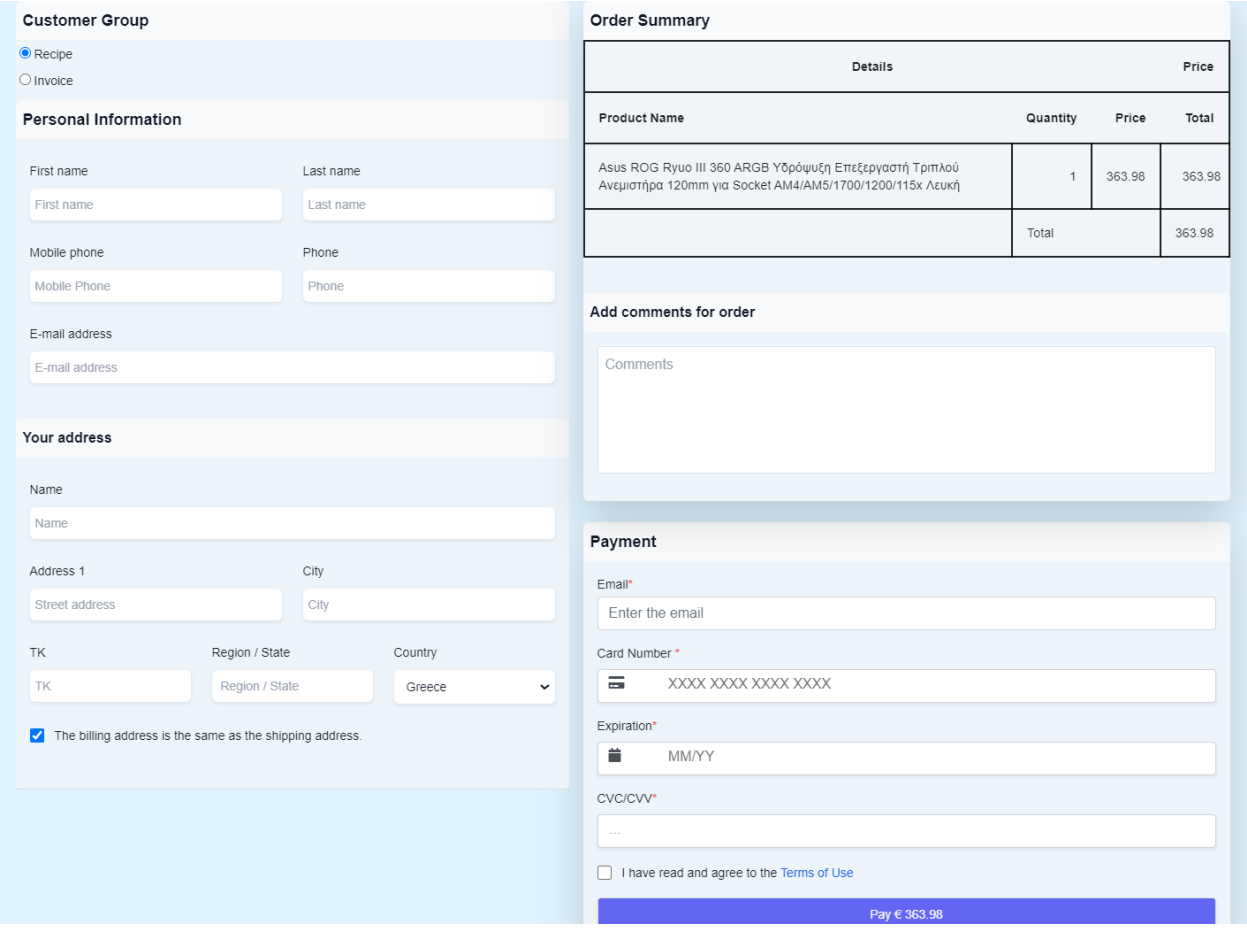
Εικόνα 4.32

```
                                                                <MdAddShoppingCar
t

                                                                    onClick={()
=> {

                                                                        if (
                                                                            user?
.id

                                                                        ) {
                                                                            handl
eAddToCart(

                                                                                {

  ...card,

  quantity: 1,

                                                                                }
                                                                            );
                                                                        } else {
                                                                            alert
(

                                                                                "
Please Login First to Add this Item to cart"

                                                                            );
                                                                        }
                                                                    }}
                                                                    size={40}
                                                                    className="p-
2 text-white bg-yellow-400 cursor-pointer rounded-xl"
                                                                />
```

Διαχείριση Προσθήκης στο Καλάθι: Χρησιμοποιείται ένα εικονίδιο καλαθιού αγορών (MdAddShoppingCart) το οποίο λειτουργεί ως κουμπί για την προσθήκη του προϊόντος στο καλάθι. Αν ο χρήστης είναι συνδεδεμένος, το προϊόν προστίθεται στο καλάθι με τη συνάρτηση handleAddToCart. Αν δεν είναι συνδεδεμένος, εμφανίζεται ένα μήνυμα που τον προτρέπει να συνδεθεί πρώτα

4.8 CART PAGE

4.9 CHECKOUT



Εικόνα 4.32

```
const handleOptionClick = (option) => {
        // Set the selected option
        setOption(option);
    };

    const handlePayment = (response) => {
        console.log({ response });
    };
```

```
// mathematical calculations
const ccyFormat = (num) => {
    return parseFloat(num).toFixed(2);
};

const courier = 5;

useEffect(() => {
    if (!uid) {
        history.push("/");
    }
}, []);

useEffect(() => {
    // Calculate total price when state.data changes
    if (state?.data?.length > 0) {
        const total = state.data.reduce((acc, item) => acc + item.price, 0);
        setTotalPrice(total);
        console.log(state.data, total);
    } else {
        setTotalPrice(0);
    }
}, [state?.data]);

// Add an empty row to create space before the total price row
const rowsWithTotal = state?.data ? [...state.data] : [];
```

handleOptionClick: Η συνάρτηση handleOptionClick αποθηκεύει την επιλογή του χρήστη. Χρησιμοποιείται για να ενημερώσει την κατάσταση της εφαρμογής με την επιλογή που έκανε ο χρήστης.

handlePayment: Καταγράφει την απόκριση από μια προσπάθεια πληρωμής. Αυτό μπορεί να χρησιμοποιηθεί για την ανάλυση της απόκρισης της πληρωμής, για λόγους αποσφαλμάτωσης ή επιβεβαίωση

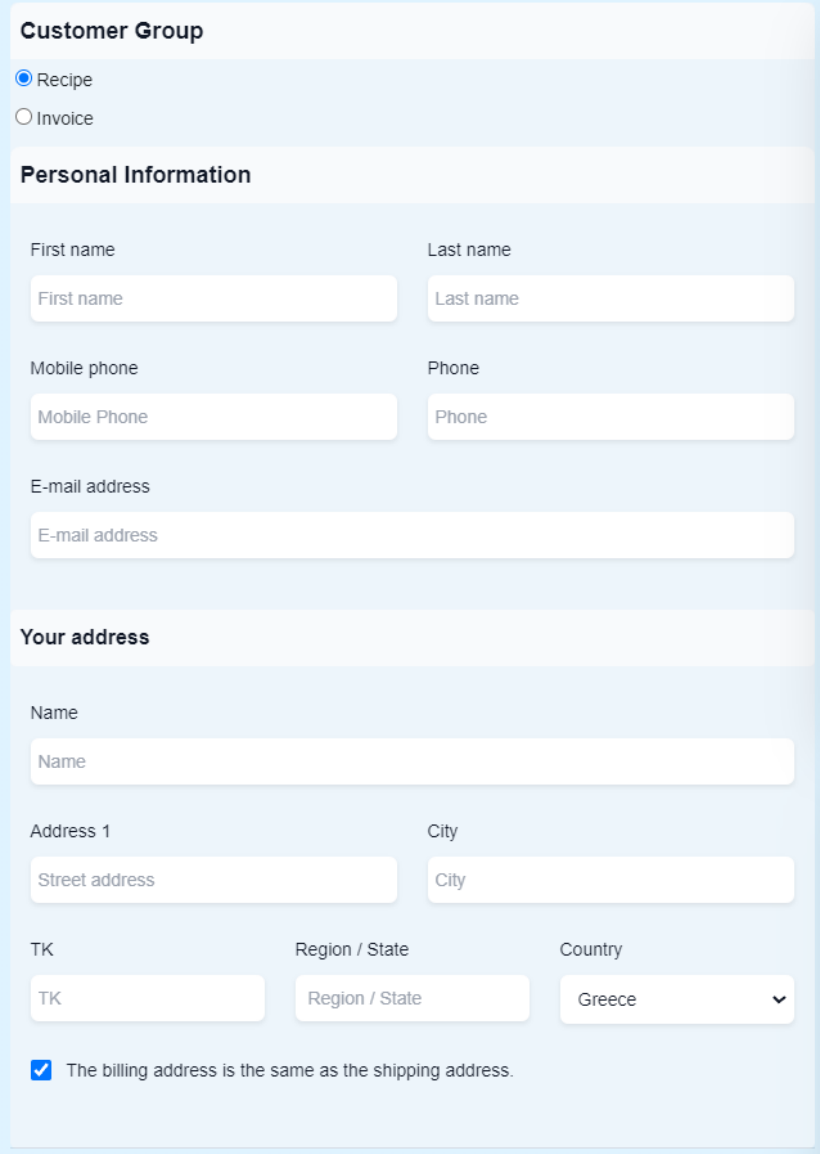
ccyFormat: Μετατρέπει έναν αριθμό σε μορφή νομίσματος με δύο δεκαδικά ψηφία. Χρησιμοποιείται για την εμφάνιση των τιμών προϊόντων ή συνολικών ποσών σε σταθερή και κατανοητή μορφή.

useEffect για Έλεγχο Αυθεντικοποίησης: Ελέγχει αν ο χρήστης έχει συνδεθεί (uid). Αν δεν υπάρχει uid, δηλαδή ο χρήστης δεν είναι συνδεδεμένος, τον ανακατευθύνει στην αρχική σελίδα.

useEffect για Υπολογισμό Συνολικής Τιμής: Αξιοποιεί τα δεδομένα από το state.data για να υπολογίσει τη συνολική τιμή των προϊόντων που περιέχονται στη λίστα. Χρησιμοποιεί τη μέθοδο reduce για να αθροίσει τις τιμές των προϊόντων. Αν δεν υπάρχουν δεδομένα, ορίζει την συνολική τιμή σε 0.

4.9.1 Επιλογή κατηγορίας πελάτη

Περιλαμβάνει ένα στοιχείο διεπαφής χρήστη που παρέχει στον χρήστη τη δυνατότητα να επιλέξει μεταξύ δύο επιλογών Απόδειξης ή Τιμολογίου. (Recipe ή Invoice)



Εικόνα 4.33

```
<div className="border-b border-gray-900/10 pb-4">
                            <h2 className="text-lg font-semibold leading-7 text-
gray-900 bg-slate-50 p-2 rounded-t-lg">
                                Customer Group
                            </h2>
                            <div>
```

```
                        <div className="p-1">
                            <div>
                                {" "}
                                <input
                                    type="radio"
                                    id="Recipe"
                                    name="customer"
                                    checked={
                                        option == "RECIPE"
                                            ? true
                                            : false
                                    }
                                    onClick={() =>
                                        handleOptionClick("RECIPE")
                                    }
                                />
                                <label
                                    htmlFor="Recipe"
                                    className="mx-1 text-sm"
                                >
                                    Recipe
                                </label>
                            </div>
                            <div>
                                {" "}
                                <input
                                    name="customer"
                                    type="radio"
                                    id="Invoice"
                                    onClick={() =>
                                        handleOptionClick("INVOICE")
                                    }
                                />
                                <label
                                    htmlFor="Invoice"
                                    className="mx-1 text-sm"
                                >
                                    Invoice
                                </label>
                            </div>
```

Επιλογή "Recipe":

Input: Περιλαμβάνει ένα ραδιοφωνικό κουμπί (input type="radio") με id="Recipe" και name="customer".

Checked Condition: Ελέγχεται αυτόματα αν η τρέχουσα επιλογή (option) είναι "RECIPE".

Event Handler: Όταν το ραδιοφωνικό κουμπί κλικαριστεί, καλείται η συνάρτηση handleOptionClick με την παράμετρο "RECIPE" για να ενημερώσει την επιλογή.

Επιλογή "Invoice":

Input: Αντίστοιχα, περιλαμβάνει ένα ραδιοφωνικό κουμπί με id="Invoice" και το ίδιο name="customer".

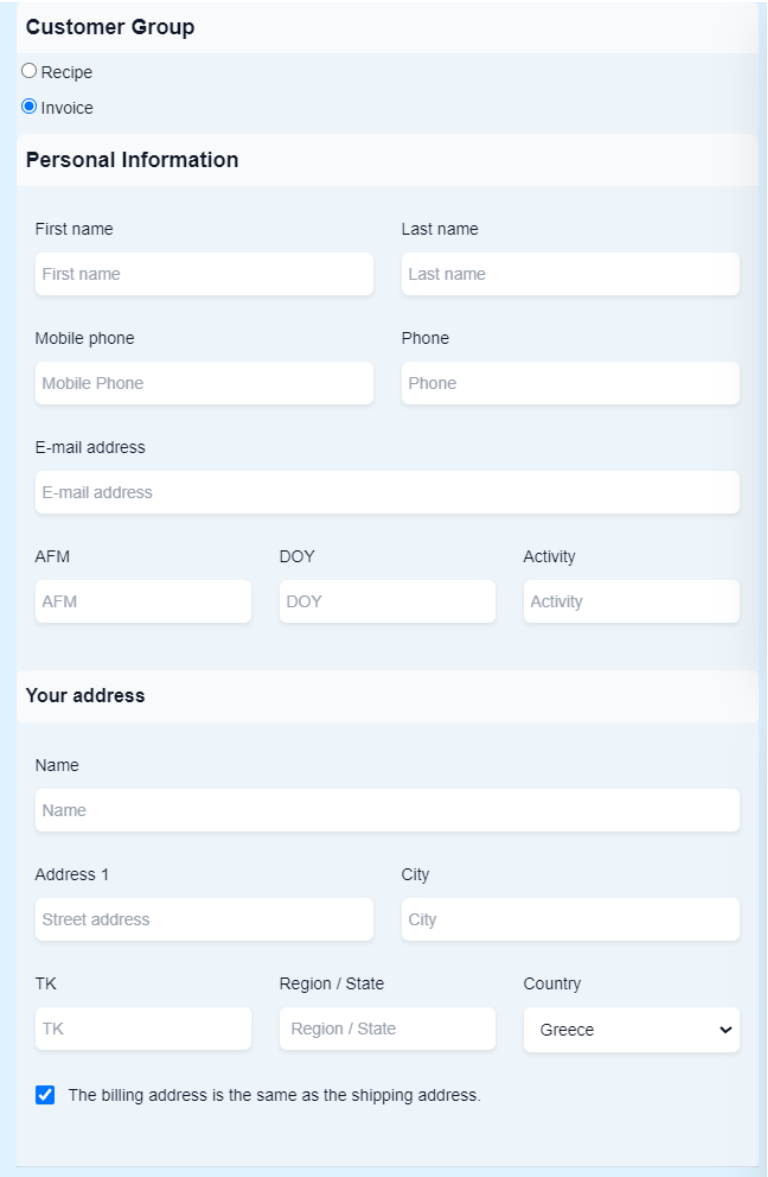
Event Handler: Συνδέεται επίσης με τη συνάρτηση handleOptionClick, αλλά περνάει την παράμετρο "INVOICE" για ενημέρωση.

```
<div className="sm:col-span-3 ">
                                    <label
                                        htmlFor="first-name"
                                        className="block text-sm font-medium
leading-6 text-gray-900"
                                    >
                                        First name
                                    </label>
                                    <div className="mt-2">
                                        <input
                                            type="text"
                                            name="first-name"
                                            id="first-name"
                                            autoComplete="given-name"
                                            placeholder="First name"
                                            className="block w-full rounded-md
border-0 p-1.5 text-gray-900 shadow-sm ring-1 ring-inset ring-gray-300
placeholder:text-gray-400 focus:ring-2 focus:ring-inset focus:ring-indigo-600
sm:text-sm sm:leading-6"
                                        />
                                    </div>
                                </div>
```

```
<div className="sm:col-span-3">
                                    <label
                                        htmlFor="last-name"
                                        className="block text-sm font-medium
leading-6 text-gray-900"
                                    >
                                        Last name
                                    </label>
                                    <div className="mt-2">
                                        <input
                                            type="text"
                                            name="last-name"
                                            id="last-name"
```

```
                                        autoComplete="family-name"
                                        placeholder="Last name"
                                        className="block w-full rounded-md
border-0 p-1.5 text-gray-900 shadow-sm ring-1 ring-inset ring-gray-300
placeholder:text-gray-400 focus:ring-2 focus:ring-inset focus:ring-indigo-600
sm:text-sm sm:leading-6"
                                    />
                                </div>
                            </div>
```

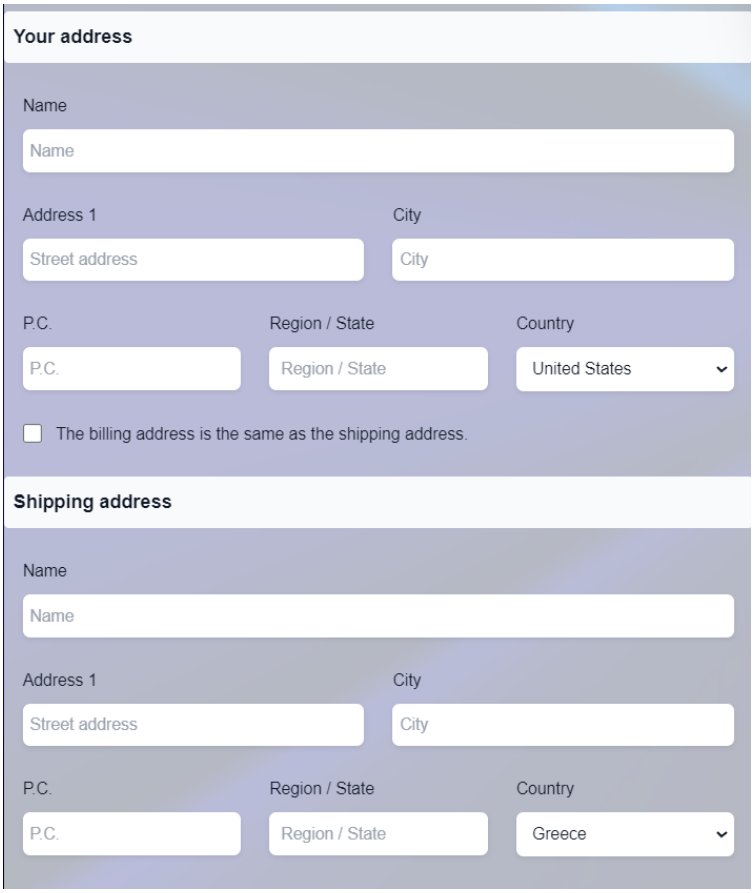
Εαν όμως η επιλογή ειναι INVOICE:



Εικόνα 4.34

```
{option === "INVOICE" && (
                                        <>
                                            <div className="sm:col-span-3 md:col-
span-2">
                                                <label
                                                    htmlFor="VAT"
                                                    className="block text-sm font-
medium leading-6 text-gray-900"
                                                >
                                                    VAT Number
                                                </label>
                                                <div className="mt-2">
                                                    <input
                                                        id="VAT"
                                                        name="VAT"
                                                        type="VAT"
                                                        autoComplete="VAT"
                                                        placeholder="VAT"
                                                        className="block w-full
rounded-md border-0 p-1.5 text-gray-900 shadow-sm ring-1 ring-inset ring-gray-300
placeholder:text-gray-400 focus:ring-2 focus:ring-inset focus:ring-indigo-600
sm:text-sm sm:leading-6"
                                                    />
                                                </div>
                                            </div>
```

4.9.2 Λειτουργία Εμφάνισης Στοιχείων Διεύθυνσης



Εικόνα 4.35

Η φόρμα εμφανίζεται μόνο εάν η μεταβλητή isChecked δεν είναι ενεργοποιημένη (!isChecked). Αυτό υποδηλώνει ότι η φόρμα γίνεται ορατή με βάση ορισμένες συνθήκες ή επιλογές χρήστη που αλλάζουν την κατάσταση της isChecked. Η λειτουργία αυτή είναι χρήσιμη όταν η διεύθυνση χρέωσης είναι διαφορετική από την διεύθυνση αποστολής.

```
{!isChecked ? (
                                <>
                                    <div className="bg-slate-100 sm:col-span-
6  rounded">
                                        <h2 className="text-base font-semibold
leading-7 text-gray-900 bg-slate-50 p-2 rounded-lg ">
                                            Shipping address
                                        </h2>
                                    </div>
```

4.9.3 Σύνοψη παραγγελίας

Order Summary

Details			Price
Product Name	**Quantity**	**Price**	**Total**
Asus TUF Gaming A17 (2023) FA707XI-NS94 17.3" FHD 144Hz (Ryzen 9-7940HS/16GB/1TB SSD/GeForce RTX 4070/W11 Home) Mecha Gray (US Keyboard)	1	1799.29	1799.29
	Total		1799.29

Εικόνα 4.36

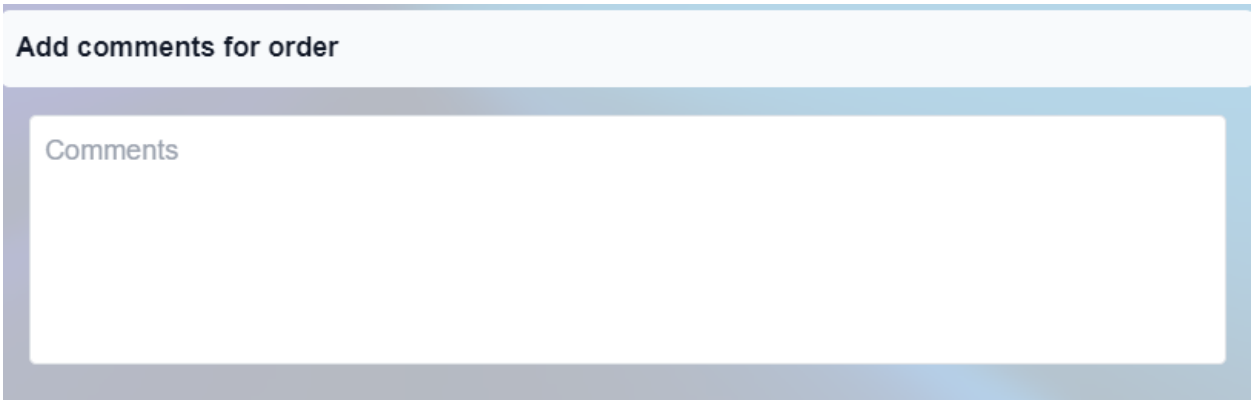
Εμφάνιση Δεδομένων: Χρησιμοποιείται ένας πίνακας (Table) για την εμφάνιση των λεπτομερειών της παραγγελίας, εφόσον υπάρχουν δεδομένα (state?.data?.length > 0).

Κεφαλίδες Πίνακα: Περιλαμβάνονται σειρές κεφαλίδων, όπως Όνομα Προϊόντος, Ποσότητα, Τιμή και Συνολικό Κόστος.

```
{state?.data?.length > 0 ? (
                            <>
                                <Table className="bg-slate-100 bg-opacity-80
backdrop-blur-md mb-10 ">
                                    <TableHead sx={{ border: 2 }}>
                                        <TableRow sx={{ border: 2 }}>
                                            <TableCell
                                                align="center"
                                                colSpan={3}
                                            >
                                                <b>Details</b>
                                            </TableCell>
                                            <TableCell align="right">
                                                <b>Price</b>
                                            </TableCell>
                                        </TableRow>
{rowsWithTotal.map((row) => (
                                            <TableRow
                                                key={row.id}
                                                sx={{ border: 2 }}
                                            >
                                                <TableCell>
```

```
                                    {row.title}
                                </TableCell>
                                <TableCell
                                    align="right"
                                    sx={{ border: 2 }}
                                >
                                    {row.quantity}
                                </TableCell>
                                <TableCell align="right">
                                    {row.price}
                                </TableCell>
                                <TableCell
                                    align="right"
                                    className="px-2"
                                    sx={{ border: 2 }}
                                >
                                    {ccyFormat(
                                        row.quantity * row.price
                                    )}
                                </TableCell>
                            </TableRow>
                        ))}
```

4.9.4 Σχόλια παραγγελίας



Εικόνα 4.37

Ένας χώρος για σχόλια προστίθεται στο τέλος, προσφέροντας στον χρήστη τη δυνατότητα να καταχωρήσει πρόσθετα σχόλια για την παραγγελία.

```
<div className="mt-1 ">
```

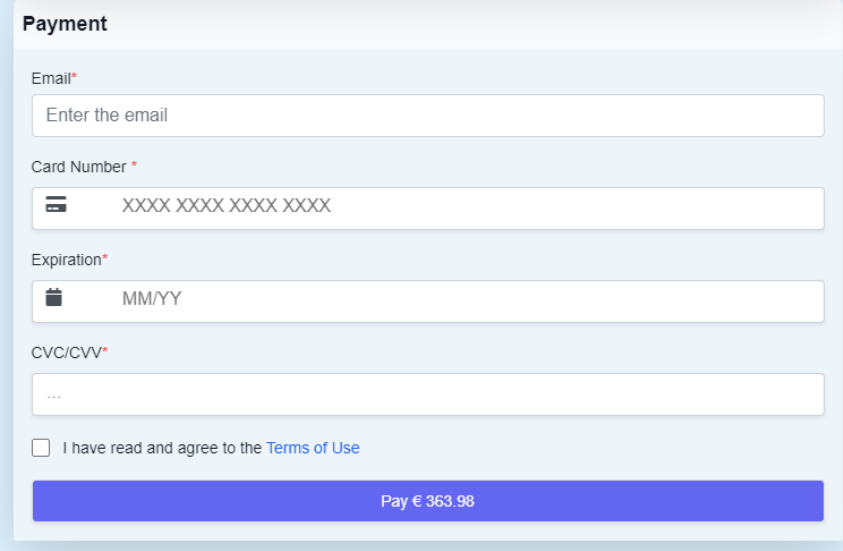
```
                                    <h2 className="text-base font-semibold
leading-7 text-gray-900 bg-slate-50 p-2   rounded-lg ">
                                        Add comments for order
                                    </h2>
                                    <div className="px-3">
                                        {" "}
                                        <textarea
                                            placeholder="Comments"
                                            className="w-full p-2 mt-3 border
border-gray-300 rounded min-h-36 resize-none"
                                        ></textarea>
                                    </div>
                                </div>
```

4.9.5 Εναλλακτική Εμφάνιση

Εάν δεν υπάρχουν δεδομένα για εμφάνιση (άδειο καλάθι), εμφανίζεται μια εικόνα, δίνοντας έναν οπτικά ευχάριστο τρόπο για να δείξει ότι το καλάθι είναι άδειο.

Εικόνα 4.38

4.9.6 Διαχείριση Πληρωμής



Εικόνα 4.39

```
<div className=" shadow-lg shadow-white rounded-md mt-4 ">
                        <h5 className="text-lg font-semibold leading-7 text-gray-
900  p-2 bg-slate-50 rounded-t-md ">
                            Payment
                        </h5>
                        <div className="p-3 bg-slate-100 bg-opacity-75 backdrop-
blur-md ">
                            {state?.price ? (
                                <Elements stripe={stripePromise}>
                                    <StripeCheckout
                                        handlePayment={handlePayment}
                                        bill={state?.price}
                                    />
                                </Elements>
                            ) : (
                                <div className="py-10">
                                    No Payment Initiated
                                </div>
                            )}
                        </div>
                    </div>
```

Έλεγχος Τιμής: Πριν από την ενσωμάτωση του στοιχείου Stripe, ελέγχεται αν υπάρχει μια διαθέσιμη τιμή (state?.price). Αυτό εξασφαλίζει ότι η διαδικασία πληρωμής προχωρά μόνο αν υπάρχει ένα ποσό προς πληρωμή.

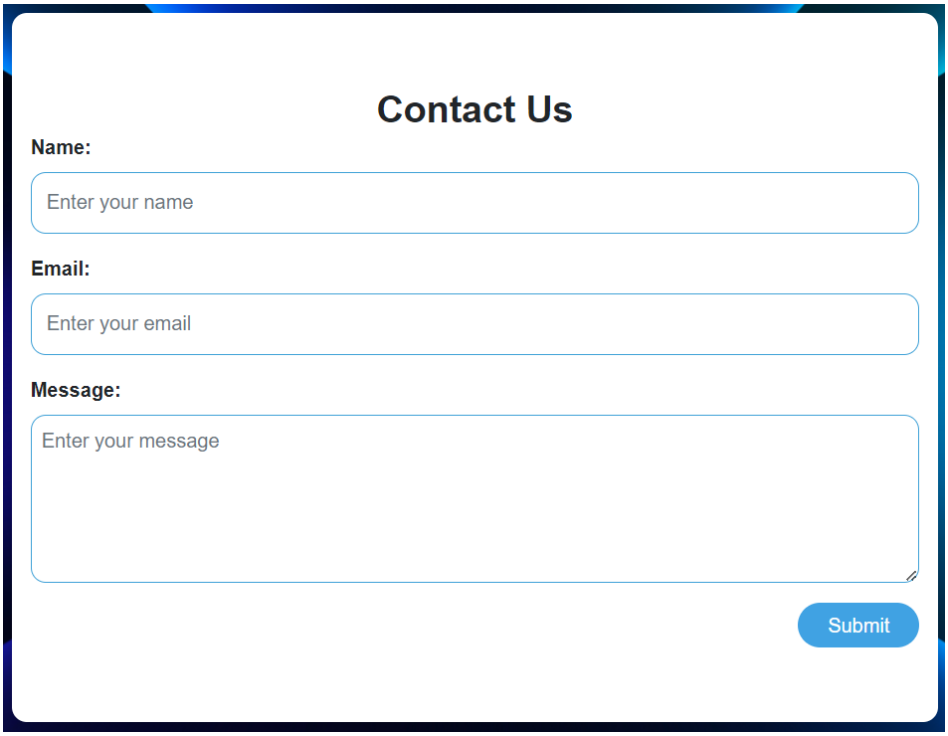
Ενσωμάτωση Stripe: Χρησιμοποιείται το στοιχείο Elements της Stripe, το οποίο περιβάλλει το StripeCheckout. Το stripePromise αναφέρεται πιθανότατα σε μια ασύγχρονη προετοιμασία του Stripe API client που πρέπει να ολοκληρωθεί πριν από τη φόρτωση του στοιχείου.

Στοιχείο StripeCheckout: Το StripeCheckout διαχειρίζεται τη λεπτομερή εφαρμογή της πληρωμής, δέχεται ως props τη συνάρτηση handlePayment για την αντιμετώπιση των αποκρίσεων πληρωμής και την τιμή bill που αντιστοιχεί στην τιμή που πρέπει να πληρωθεί.

4.10 CONTACT US

4.10.1 Φόρμα Επικοινωνίας

Η αποστολή πληροφοριών προς τη διαχείριση της εφαρμογής παρέχει έναν αποτελεσματικό μηχανισμό για τους χρήστες ώστε να επικοινωνούν με τους διαχειριστές της εφαρμογής.Είναι σημαντική για την υποστήριξη πελατών και την επικοινωνία.



Εικόνα 4.40

```
export default function ContactUs() {
```

```
    const dispatch = useDispatch();
    const [formData, setFormData] = useState({
        name: "",
        email: "",
        message: "",
    });

    const handleInputChange = (e) => {
        const { name, value } = e.target;
        setFormData({
            ...formData,
            [name]: value,
        });
    };

    const handleSubmit = (e) => {
        e.preventDefault();

        dispatch(
            contactAdmin(formData, () => {
                setFormData({
                    name: "",
                    email: "",
                    message: "",
                });
            })
        );
    };
```

Χρησιμοποιείται useState για να δημιουργήσει και να διατηρήσει την κατάσταση των δεδομένων που εισάγονται από τον χρήστη (name, email, message).

Για την υποβολή δεδομένων της φόρμας, η συνάρτηση handleSubmit διαχειρίζεται την υποβολή της φόρμας. Αποτρέπει την προεπιλεγμένη υποβολή φόρμας του προγράμματος περιήγησης (e.preventDefault()), και στη συνέχεια χρησιμοποιεί την ενέργεια dispatch του Redux για να στείλει τα δεδομένα στον διαχειριστή. Ύστερα, υπάρχει και η επαναφορά δεδομένων φόρμας μετά την υποβολή οπου μετά την αποστολή των δεδομένων, τα πεδία της φόρμας επαναφέρονται σε κενά, ετοιμάζοντας τη φόρμα για μια νέα χρήση, αν χρειαστεί.

```
<FormGroup>
                        <Label className="font-bold" for="name">
                            Name:
                        </Label>
                        <Input
                            type="text"
                            name="name"
                            id="name"
```

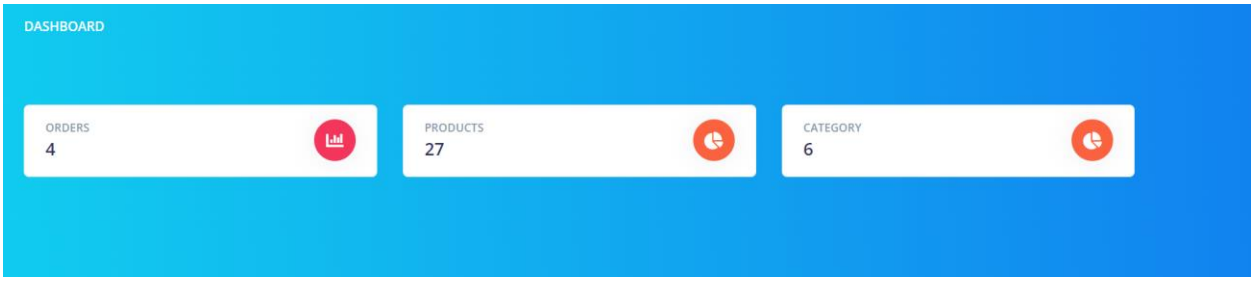
```
                                placeholder="Enter your name"
                                required
                                className="border-1 border-[#40A2D8] py-4
rounded-xl"
                                value={formData.name}
                                onChange={handleInputChange}
                            />
                        </FormGroup>
```

Η συνάρτηση handleInputChange ενημερώνει τα δεδομένα της φόρμας με κάθε αλλαγή που πραγματοποιεί ο χρήστης στα πεδία εισόδου.

ΚΕΦΑΛΑΙΟ 5: ΑΡΧΙΤΕΚΤΟΝΙΚΗ ΚΑΙ ΠΕΡΙΓΡΑΦΗ ADMIN PANEL

5.1 HEADER

5.1.1 Καρτέλες Κατηγοριών



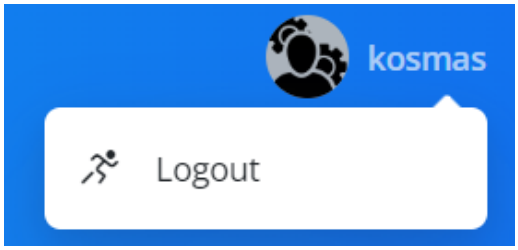
Εικόνα 5.1

Αυτό το τμήμα κώδικα ανήκει σε μια διεπαφή χρήστη και χρησιμοποιείται για να εμφανίσει καρτέλες που παρουσιάζουν στατιστικά σχετικά με τις παραγγελίες τα προϊόντα κλπ. Οι κάρτες αυτές έχουν έναν τίτλο που δείχνουν τον αριθμό των παραγγελιών, των προϊόντων κλπ. Επιπλέον, περιλαμβάνει ένα εικονίδιο με έντονη οπτική παρουσία για να προσελκύσει την προσοχή.

```
<Col lg="6" xl="3">
                <Card className="card-stats mb-4 mb-xl-0">
                  <CardBody>
                    <Row>
                      <div className="col">
                        <CardTitle
                          tag="h5"
```

```
                        className="text-uppercase text-muted mb-0"
                      >
                        Orders
                      </CardTitle>
                      <span className="h2 font-weight-bold mb-0">
                        {allOrders?.length}
                      </span>
                    </div>
                    <Col className="col-auto">
                      <div className="icon icon-shape bg-danger text-white
rounded-circle shadow">
                        <i className="fas fa-chart-bar" />
                      </div>
                    </Col>
                  </Row>
                </CardBody>
              </Card>
            </Col>
```

5.1.2 Εικονίδιο Admin



Εικόνα 5.2

Αυτό το κομμάτι κώδικα είναι σημαντικό για τη δημιουργία ενός κουμπιού που όταν το πατήσει ο χρήστης, εμφανίζει μια λίστα επιλογών. Εμφανίζει την εικόνα του χρήστη και το όνομα του. Αυτό βοηθάει γιατί βλέπουν αμέσως το προφίλ τους και μπορούν να προχωρήσουν σε ενέργειες όπως η αποσύνδεση, χωρίς να χρειάζεται να ψάξουν για το πού βρίσκονται αυτές οι επιλογές. Επίσης, ο κώδικας προσαρμόζει την εμφάνιση του κουμπιού ανάλογα με το μέγεθος της οθόνης, για να φαίνεται πάντα καλά στον χρήστη.

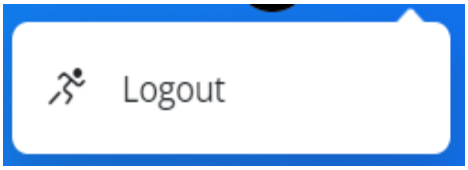
```
<DropdownToggle className="pr-0" nav>
                <Media className="align-items-center">
                  <span className="avatar avatar-sm rounded-circle">
                    <img
                      alt="..."
                      src={
```

```
                    require("../../assets/img/theme/AA.png")
                      .default
                  }
                />
              </span>
              <Media className="ml-2 d-none d-lg-block">
                <span className="mb-0 text-sm font-weight-bold">
                  {user?.username}
                </span>
              </Media>
            </Media>
          </DropdownToggle>
```

<DropdownToggle className="pr-0" nav>: Δημιουργεί ένα πτυσσόμενο κουμπί. Η προσθήκη της κλάσης "pr-0" ρυθμίζει το εξωτερικό περιθώριο (padding-right) σε μηδέν, έτσι ώστε να διασφαλίζεται ότι δεν υπάρχει κενό στο δεξί μέρος του κουμπιού. Η ιδιότητα "nav" υποδηλώνει ότι το στοιχείο αυτό ανήκει στην πλοήγηση (navigation), βοηθώντας στη συνοχή του στυλ και της λειτουργίας του μενού.

<Media className="align-items-center">: Αυτό το τμήμα κώδικα περιλαμβάνει άλλα πολυμέσα (όπως εικόνες ή κείμενο) και ευθυγραμμίζει κάθετα όλα τα στοιχεία στο κέντρο του πλαισίου. Η κλάση "align-items-center" βοηθά στη διατήρηση μιας καθαρής και ευχάριστης διάταξης στον χώρο που περιλαμβάνει τα πολυμέσα.

5.1.3 Κουμπί Logout



Εικόνα 5.3

Αυτός ο κώδικας δημιουργεί ένα μενού πτυσσόμενων στοιχείων (dropdown menu) που περιλαμβάνει την επιλογή "Logout". Κατά την επιλογή του "Logout", το σύστημα αποσυνδέει τον χρήστη από την εφαρμογή και τον ανακατευθύνει στη σελίδα σύνδεσης. Η διαδικασία αποσύνδεσης είναι ελεγχόμενη μέσω ενός κλικ, ακυρώνοντας την προεπιλεγμένη λειτουργία του συνδέσμου και καλώντας μια λειτουργία αποσύνδεσης.

```
<DropdownToggle className="pr-0" nav>
            <Media className="align-items-center">
              <span className="avatar avatar-sm rounded-circle">
                <img
```

```
                    alt="..."
                    src={
                      require("../../assets/img/theme/AA.png")
                        .default
                    }
                  />
                </span>
                <Media className="ml-2 d-none d-lg-block">
                  <span className="mb-0 text-sm font-weight-bold">
                    {user?.username}
                  </span>
                </Media>
              </Media>
            </DropdownToggle>
```

<DropdownMenu className="dropdown-menu-arrow" right>: Αυτή η ετικέτα δημιουργεί το πτυσσόμενο μενού με την κλάση dropdown-menu-arrow, πιθανώς προσδίδοντας ένα στυλ με βέλος στο μενού. Η ιδιότητα right υποδεικνύει ότι το μενού θα ανοίγει ή θα ευθυγραμμίζεται προς τα δεξιά, βοηθώντας στην τοποθέτηση του μενού στον χώρο της διεπαφής.

<DropdownItem href="#pablo" onClick={(e) => {: Αυτή η ετικέτα δημιουργεί ένα αντικείμενο μέσα στο πτυσσόμενο μενού. Ο σύνδεσμος (href="#pablo") είναι αρχικά ένας dummy link που απλά χρησιμοποιείται για σκοπούς αναπαράστασης. Το onClick παρέχει μια συνάρτηση ελέγχου για την αντίδραση στο κλικ, όπου προλαμβάνει την προεπιλεγμένη δράση του συνδέσμου (ανακατεύθυνση) και καλεί μια λειτουργία που χειρίζεται την αποσύνδεση του χρήστη.

5.2 CATEGORY

Αυτός ο κώδικας αποτελεί ένα React component που διαχειρίζεται τις κατηγορίες σε μια εφαρμογή, χρησιμοποιώντας διάφορα μοντέλα δεδομένων και UI στοιχεία για την προσθήκη, αφαίρεση και ενημέρωση των κατηγοριών και υποκατηγοριών. Το component χρησιμοποιεί hooks για τη διαχείριση της κατάστασης (όπως modals για την ενεργοποίηση διαφόρων UI διαλόγων), καθώς και Redux για την ανάκτηση και την αποθήκευση δεδομένων σχετικά με τις κατηγορίες.

```
export default function Category() {
  const dispatch = useDispatch();
  const { loading, allCategories, getLoading } = useSelector(
    (state) => state.category
  );
  const StripedDataGrid = styled(DataGrid)(() => ({
    [`& .${gridClasses.row}.even`]: {
      backgroundColor: "#EEEE",
    },
  }));
```

```
const [addModal, setAddModal] = useState(false);
const addToggle = () => setAddModal(!addModal);
const [subCategoryModal, setSubCategoryModal] = useState(false);
const categoryToggle = () => setSubCategoryModal(!subCategoryModal);
const [title, setTitle] = useState("");
const [category, setCategory] = useState([{ subCategory: "" }]);
const [confirmModal, setConfirmModal] = useState(false);
const [selectedId, setSelectedId] = useState("");
const [subCategories, setSubCategories] = useState([]);
const confirmtoggle = () => setConfirmModal(!confirmModal);
const handleAddCategory = () => {
  setCategory([
    ...category,
    {
      subCategory: "",
    },
  ]);
};
const handleRemoveCategory = (index) => {
  const list = [...category];
  list.splice(index, 1);
  setCategory(list);
};
const handleCategoryChange = (e, index) => {
  const { name, value } = e.target;
  const list = [...category];
  list[index][name] = value;
  setCategory(list);
};
```

const { loading, allCategories, getLoading } = useSelector((state) => state.category);: Αυτή η γραμμή χρησιμοποιεί το hook useSelector του Redux για να ανακτήσει συγκεκριμένες τιμές από το Redux. Εδώ ανακτάται η κατάσταση φόρτωσης, η λίστα όλων των κατηγοριών και η κατάσταση φόρτωσης κατάτην ανάκτηση των κατηγοριών, παρέχοντας ένα μέσο για την εμφάνιση δεδομένων ή τον χειρισμό του UI ανάλογα με τις καταστάσεις αυτές.

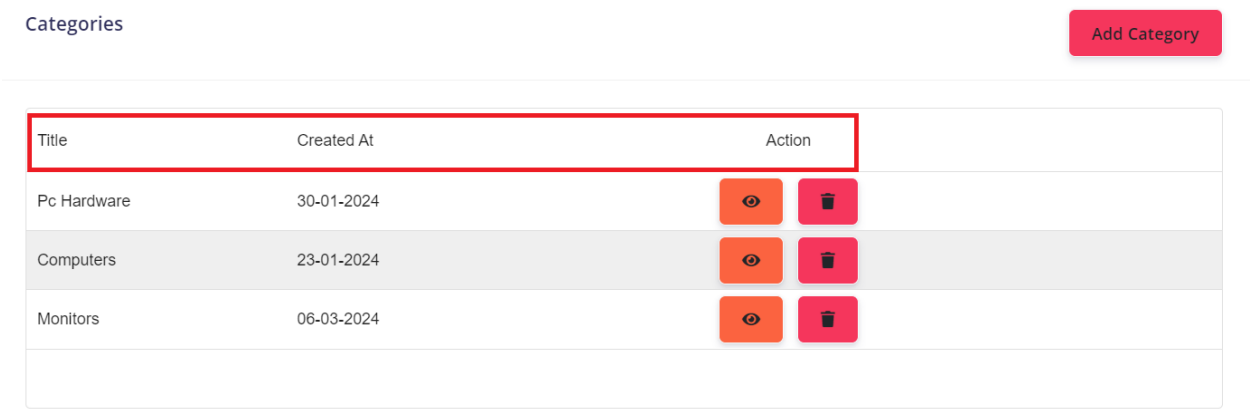
const [addModal, setAddModal] = useState(false);: Αυτό το hook της κατάστασης δημιουργεί μια μεταβλητή addModal και μια συνάρτηση setAddModal για τον χειρισμό της εμφάνισης ή απόκρυψης ενός modal παραθύρου για την προσθήκη νέας κατηγορίας. Αρχικά ορίζεται ως false, σημαίνοντας ότι το modal δεν εμφανίζεται.

const addToggle = () => setAddModal(!addModal);: Αυτή η συνάρτηση ενεργοποιεί ή απενεργοποιεί το modal για την προσθήκη κατηγορίας, εναλλάσσοντας την τιμή της addModal μεταξύ true και false. Αυτό επιτρέπει την εύκολη εναλλαγή της ορατότητας του modal μέσω UI δραστηριοτήτων.

const [category, setCategory] = useState([{ subCategory: "" }]);: Αυτό το hook διαχειρίζεται μια λίστα κατηγοριών, όπου κάθε κατηγορία μπορεί να περιέχει υποκατηγορίες. Αρχικά, η λίστα έχει ένα αντικείμενο με μια κενή υποκατηγορία.

const handleAddCategory = () => {: Αυτή η συνάρτηση επιτρέπει την προσθήκη μιας νέας υποκατηγορίας στη λίστα category. Χρησιμοποιείται για να επεκτείνει τη λίστα με ένα νέο κενό στοιχείο, διευκολύνοντας την εισαγωγή μιας νέας υποκατηγορίας.

5.2.1 Ορισμός Στηλών για Πίνακα Δεδομένων



Εικόνα 5.4

Αυτό το τμήμα κώδικα αντιπροσωπεύει τον ορισμό των στηλών που απαιτούνται για τη δημιουργία ενός πίνακα δεδομένων. Οι στήλες περιλαμβάνουν πληροφορίες όπως κατηγορία, ημερομηνία δημιουργίας και ενέργειες άλλες που μπορεί να εκτελέσει ο χρήστης, όπως προβολή λεπτομερειών και διαγραφή κατηγορίας.

```
const columns = [
    {
      field: "category",
      headerName: "Title",
      width: 230,
    },
    {
      field: "createdAt",
      headerName: "Created At",
      width: 330,
      renderCell: (params) => {
        return (
          <>
            <span>
```

```
            {moment
              .unix(params?.row?.createdAt?.seconds)
              .format("DD-MM-YYYY")}
          </span>
        </>
      );
    },
  },
  {
    field: "actions",
    type: "actions",
    headerName: "Action",
    width: 230,
    renderCell: (params) => {
      return (
        <>
          <Button
            className="bg-warning"
            onClick={() => {
              setSubCategories(params.row.subCategories);
              categoryToggle();
            }}
          >
            <i className="fas fa-eye text-white"></i>
          </Button>
          <Button
            className="bg-danger"
            onClick={() => {
              confirmtoggle();
              setSelectedId(params.row.id);
            }}
          >
            <i className="fas fa-trash text-white"></i>
          </Button>
        </>
      );
    },
  },
];
```

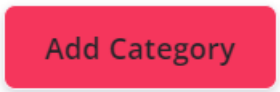
Κάθε αντικείμενο στον πίνακα columns αντιστοιχεί σε μια στήλη στον πίνακα δεδομένων.

Στήλη "Title": Η στήλη αυτή ορίζει το πεδίο category ως πηγή δεδομένων για τον τίτλο της κατηγορίας. Η ιδιότητα headerName καθορίζει τον τίτλο της στήλης στον πίνακα, ενώ το width καθορίζει το πλάτος της στήλης σε pixels.

Στήλη "Created At": Η στήλη αυτή αντιπροσωπεύει την ημερομηνία δημιουργίας. Χρησιμοποιείται η ιδιότητα renderCell για να παρέχει ειδική μορφοποίηση στην εμφάνιση της ημερομηνίας, με χρήση της βιβλιοθήκης moment για τη μετατροπή της σε επιθυμητή μορφή.

Στήλη "Action": Αυτή η στήλη περιλαμβάνει κουμπιά για ενέργειες σχετικά με κάθε κατηγορία. Το πεδίο actions τύπου "actions" καθορίζει την τύπο της στήλης. Κάθε κουμπί στην στήλη ενεργοποιεί συγκεκριμένες λειτουργίες, όπως το άνοιγμα ενός modal ή τη διαγραφή μιας κατηγορίας. Η ιδιότητα renderCell χρησιμοποιείται για την προσαρμογή του εμφανιζόμενου κελιού με ειδική λογική, όπως τοποθέτηση κουμπιών και εκτέλεση συγκεκριμένων λειτουργιών κλικ.

5.2.2 Κουμπί προσθήκης κατηγορίας



Εικόνα 5.5

Ο κώδικας αυτός αποτελεί μέρος μιας διεπαφής χρήστη React που χρησιμοποιείται για την παρουσίαση και διαχείριση κατηγοριών σε ένα σύστημα. Αποτελείται από ένα κάρτα (Card) με μια επικεφαλίδα (CardHeader) που περιλαμβάνει έναν τίτλο και ένα κουμπί για την προσθήκη νέων κατηγοριών. Στο σώμα της κάρτας (CardBody), υπάρχει ένας φορτωτής που εμφανίζεται κατά τη φόρτωση των δεδομένων και ένας πίνακας δεδομένων (StripedDataGrid) που εμφανίζει τις κατηγορίες.

```
<Card>
              <CardHeader>
                <h3 className="mb-0 d-inline-block">Categories</h3>
                <Button
                  className="float-right bg-red text-white"
                  onClick={addToggle}
                >
                  Add Category
                </Button>
              </CardHeader>
              <CardBody>
                {" "}
                <LoadingOverlay
                  active={getLoading}
                  spinner
                  text="Categories Loading...."
                >
                  <StripedDataGrid
                    autoHeight
```

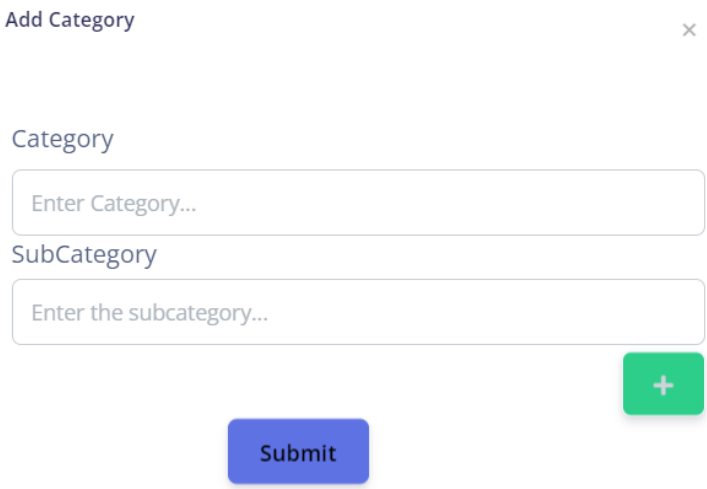
```
                autoWidth
                columns={columns}
                rows={allCategories}
                disableSelectionOnClick={false}
                getRowClassName={(params) =>
                  params.indexRelativeToCurrentPage % 2 === 0
                    ? "odd"
                    : "even"
                }
                hideFooterPagination={true}
              />
            </LoadingOverlay>
          </CardBody>
        </Card>
```

<CardHeader>: Χρησιμοποιείται για την παρουσίαση της επικεφαλίδας της κάρτας. Περιλαμβάνει έναν τίτλο για την ενότητα των κατηγοριών και ένα κουμπί για την προσθήκη νέας κατηγορίας. Το κουμπί αυτό είναι δεξιά τοποθετημένο και χρησιμοποιείται για την ενεργοποίηση της λειτουργίας προσθήκης κατηγορίας.

<CardBody>: Περιέχει το κύριο περιεχόμενο της κάρτας. Εδώ περιλαμβάνεται το LoadingOverlay, το οποίο εμφανίζει έναν φορτωτή κατά τη διάρκεια της φόρτωσης των δεδομένων. Το LoadingOverlay ενεργοποιείται βάσει της κατάστασης getLoading και εμφανίζει ένα γραφικό loading με κείμενο που ενημερώνει τον χρήστη για τη φόρτωση των κατηγοριών.

<StripedDataGrid>: Προσαρμοσμένο DataGrid που εμφανίζει τις κατηγορίες σε έναν πίνακα. Το DataGrid διαμορφώνεται με στήλες και γραμμές που προσδιορίζονται από την μεταβλητή columns και allCategories αντίστοιχα. Επιπλέον, υποστηρίζει τη δυνατότητα να αλλάζει την κλάση των γραμμών βάσει της θέσης τους στην τρέχουσα σελίδα, εναλλάσσοντας μεταξύ 'odd' και 'even' για ευκολότερη ανάγνωση των δεδομένων.

5.2.3 Δυναμική προσθήκη κατηγοριών



Εικόνα 5.6

Αυτός ο κώδικας είναι μέρος ενός React component που επιτρέπει στους χρήστες να διαχειρίζονται δυναμικά τις υποκατηγορίες μιας κατηγορίας. Ο κώδικας διατρέχει μια λίστα category, και για κάθε στοιχείο, εμφανίζει μια φόρμα εισαγωγής για την επεξεργασία του ονόματος της υποκατηγορίας. Παρέχει επίσης κουμπιά για την προσθήκη νέας υποκατηγορίας ή τη διαγραφή της υπάρχουσας.

```
<ModalBody>
              <Label>Category</Label>
              <Input
                type="text"
                name="category"
                placeholder="Enter Category..."
                value={title}
                onChange={(e) => {
                  setTitle(e.target.value);
                }}
                required
              />
              {category.map((item, index) => {
                return (
                  <>
                    <Label className="mb-0">SubCategory</Label>
                    <Input
                      // required
                      type="text"
                      placeholder="Enter the subcategory..."
                      name="subCategory"
                      min="0"
```

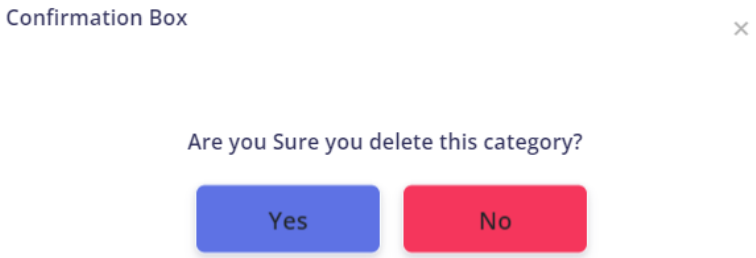
```
                  value={item.length}
                  onChange={(e) => handleCategoryChange(e, index)}
                  className="form-control mt-1"
                />

                {category.length - 1 === index && (
                  <Button
                    className="bg-success text-light my-1 float-right"
                    onClick={() => {
                      handleAddCategory();
                    }}
                  >
                    <i className="fas fa-plus" />
                  </Button>
                )}
                {category.length !== 1 && (
                  <Button
                    className="bg-danger text-light my-1 float-right"
                    onClick={() => {
                      handleRemoveCategory(index);
                    }}
                  >
                    <i className="fas fa-minus" />
                  </Button>
                )}
              </>
            );
          })}
        </ModalBody>
```

Ενεργοποιεί τη συνάρτηση handleCategoryChange, η οποία ανανεώνει την τιμή της υποκατηγορίας με βάση την εισαγωγή του χρήστη. Για την αλλαγή των τιμών των υποκατηγοριών,στο πεδίο εισόδου, η ιδιότητα onChange είναι συνδεδεμένη με την συνάρτηση handleCategoryChange, η οποία καλείται κάθε φορά που ο χρήστης πληκτρολογεί ή αλλάζει το περιεχόμενο του πεδίου. Αυτή η συνάρτηση παίρνει το στοιχείο εισόδου (event) και το δείκτη (index) της υποκατηγορίας, ενημερώνει την τιμή στο αντίστοιχο στοιχείο της λίστας category, διασφαλίζοντας ότι οι αλλαγές αντανακλούν την τρέχουσα κατάσταση των δεδομένων.

Τα κουμπιά που εμφανίζονται στο τέλος κάθε στοιχείου υποκατηγορίας επιτρέπουν τη δυνατότητα διαχείρισης της λίστας. Η προσθήκη ενός νέου στοιχείου υποκατηγορίας γίνεται δυνατή μέσω του κουμπιού προσθήκης, το οποίο εμφανίζεται μόνο δίπλα στο τελευταίο στοιχείο της λίστας. Κατά το πάτημα του κουμπιού, εκτελείται η συνάρτηση handleAddCategory, η οποία προσθέτει ένα νέο, κενό στοιχείο στη λίστα κατηγοριών. Από την άλλη πλευρά, το κουμπί αφαίρεσης εμφανίζεται σε όλα τα στοιχεία εκτός από το πρώτο, επιτρέποντας τη διαγραφή του εκάστοτε στοιχείου υποκατηγορίας. Η συνάρτηση handleRemoveCategory, που καλείται όταν πατηθεί το εν λόγω κουμπί, αφαιρεί το αντίστοιχο στοιχείο από τη λίστα, βάσει του δείκτη του. Η παραπάνω δυναμική λειτουργικότητα

παρέχει μια εξαιρετικά προσαρμοστική και αλληλεπιδραστική διεπαφή χρήστη, η οποία συμβάλλει στην αποτελεσματική διαχείριση των υποκατηγοριών σε μια εφαρμογή.

5.2.4 Διαγραφή κατηγορίας



Εικόνα 5.7

Αυτός ο κώδικας αντιπροσωπεύει ένα σημαντικό κομμάτι μιας αναδυόμενης φόρμας (modal), όπου ο χρήστης δέχεται την ερώτηση αν είναι σίγουρος για τη διαγραφή μιας κατηγορίας. Περιλαμβάνει μια ερώτηση επιβεβαίωσης και δύο κουμπιά: ένα για την επιβεβαίωση της διαγραφής, το οποίο εμφανίζει έναν δείκτη φόρτωσης αν υπάρχει εν εξελίξει διαδικασία φόρτωσης, και ένα για την απόρριψη της ερώτησης.

```
<ModalBody>
            <h5 className="text-center ">
              Are you Sure you delete this category?
            </h5>

            <div className="d-flex justify-content-center align-items-center my-
3">
              <Button
                className="bg-primary text-white w-25 mx-2"
                onClick={() => {
                  dispatch(
                    deleteCategory(selectedId, () => {
                      confirmtoggle();
                    })
                  );
                }}
              >
                {" "}
```

```
              {loading ? <Spinner size="md" /> : "Yes"}
            </Button>

            <Button
              className="bg-danger text-white w-25"
              onClick={confirmtoggle}
            >
              No
            </Button>
          </div>
        </ModalBody>
```

Ερώτηση Επιβεβαίωσης (<h5> Tag): Η επικεφαλίδα μέσα στο ModalBody χρησιμοποιείται για να παρουσιάσει την κρίσιμη ερώτηση "Are you sure you want to delete this category?". Αυτό είναι το κεντρικό μήνυμα που βλέπει ο χρήστης και το οποίο υποδεικνύει την σοβαρότητα της ενέργειας που πρόκειται να εκτελέσει. Το ύφος και η ταξινόμηση του κειμένου (text-center) βοηθούν στην εστίαση της προσοχής του χρήστη στο μήνυμα.

Διαχείριση Επιβεβαίωσης με Κουμπιά (<Button> Tags): Υπάρχουν δύο κουμπιά στον κώδικα, κάθε ένα με διαφορετικό ρόλο:

Το πρώτο κουμπί, το κουμπί επιβεβαίωσης είναι υπεύθυνο για την έναρξη της διαδικασίας διαγραφής της κατηγορίας. Χρησιμοποιεί την action dispatch(deleteCategory(selectedId, ...)) για να εκτελέσει τη διαγραφή με βάση το επιλεγμένο ID. Ενδιαφέρον στοιχείο εδώ είναι η χρήση του loading state, που υποδηλώνει αν η διαδικασία είναι σε εξέλιξη. Αν είναι, εμφανίζεται ένας δείκτης φόρτωσης (<Spinner size="md" />), διαφορετικά, το κείμενο "Yes" δείχνει πως ο χρήστης μπορεί να επιβεβαιώσει τη διαγραφή. Αντίστοιχα, το κουμπί απόρριψης επιτρέπει στον χρήστη να απορρίψει τη διαδικασία διαγραφής και να κλείσει την αναδυόμενη φόρμα (modal). Αυτό επιτυγχάνεται μέσω της συνάρτησης confirmtoggle(), η οποία εναλλάσσει την κατάσταση εμφάνισης του modal, κλείνοντας το.

5.3 ΠΡΟΙΟΝΤΑ

Η χρήση της συνάρτησης useDispatch() από το Redux επιτρέπει την αποστολή ενεργειών προς το store της εφαρμογής για την ενημέρωση του κατάλληλου state. Αντίστοιχα, το hook useSelector() χρησιμοποιείται για την ανάκτηση καταστάσεων από το Redux store για τα δεδομένα των κατηγοριών και των προϊόντων.

Οι υπόλοιπες δηλώσεις περιλαμβάνουν την αρχικοποίηση των local states με τη χρήση του useState(). Αυτά τα states περιλαμβάνουν λίστες για τις εικόνες των επιλεγμένων προϊόντων, διαφορετικά modal states για την προβολή ή την επεξεργασία πληροφοριών, και ένα πιο περίπλοκο state για τη διατήρηση λεπτομερειών προϊόντος που περιλαμβάνει τίτλο, περιγραφή, κωδικό προϊόντος, τιμή, εικόνες, κατηγορία, υποκατηγορία, ημέρες παράδοσης και το επιλεγμένο προϊόν.

```
const dispatch = useDispatch();
```

```
  const { allCategories } = useSelector((state) => state.category);
  const { loading, allProducts } = useSelector((state) => state.product);
  const StripedDataGrid = styled(DataGrid)(() => ({
    [`& .${gridClasses.row}.even`]: {
      backgroundColor: "#EEEE",
    },
  }));
  const [selectedItemImages, setSelectedItemImages] = useState([]);
  const [imgModal, setImgModal] = useState(false);
  const imagesToggle = () => setImgModal(!imgModal);
  const [addModal, setAddModal] = useState(false);
  const addToggle = () => setAddModal(!addModal);
  const [editModal, setEditModal] = useState(false);
  const editToggle = () => setEditModal(!editModal);
  const [categories, setCategories] = useState([]);
  const [subCategories, setSubCategories] = useState([]);
  const [selectedId, setSelectedId] = useState("");
  const [confirmModal, setConfirmModal] = useState(false);
  const confirmtoggle = () => setConfirmModal(!confirmModal);
  const [productDetail, setProductDetail] = useState({
    title: "",
    description: "",
    productCode: "",
    price: 0,
    images: [],
    category: "",
    subCategory: "",
    deliveryDays: 1,
    selectedProduct: [],
  });
```

Ο κώδικας περιλαμβάνει τη διαχείριση των λεπτομερειών προϊόντων μέσω της ενημέρωσης ενός state σε component. Χρησιμοποιεί διάφορες συναρτήσεις για να χειριστεί την αλλαγή των πεδίων στοιχείων του προϊόντος, όπως κατηγορία, υποκατηγορία και επιλεγμένο προϊόν. Οι λειτουργίες αυτές είναι βασικές για τη διαμόρφωση και την ενημέρωση των στοιχείων του προϊόντος πριν από την τελική υποβολή των δεδομένων.

```
const [recommendedProduct, setRecommendedProduct] = useState([]);
  const handleChange = (e) => {
    e.preventDefault();
    const { name, value } = e.target;
    const updatedAboutDetails = {
      ...productDetail,
      [name]: value,
    };
```

```
    setProductDetail(updatedAboutDetails);
  };
  const handleSelect = (opt) => {
    setProductDetail((prevDetail) => {
      return {
        ...prevDetail,
        category: opt,
      };
    });
  };
  const handleRecommendedSelect = (opt) => {
    setProductDetail((prevDetail) => {
      return {
        ...prevDetail,
        selectedProduct: opt,
      };
    });
  };
  const handleSubcategorySelect = (opt) => {
    setProductDetail((prevDetail) => {
      return {
        ...prevDetail,
        subCategory: opt,
      };
    });
  };
```

Η συνάρτηση handleChange αντιμετωπίζει τις αλλαγές στα εισόδους χρήστη, αποθηκεύοντας κάθε αλλαγή στο αντίστοιχο πεδίο μέσω του αντικειμένου event. Ενημερώνει το productDetail state με τις νέες τιμές, διασφαλίζοντας ότι κάθε αλλαγή αντικατοπτρίζεται αμέσως στο UI. Οι συναρτήσεις handleSelect, handleRecommendedSelect, και handleSubcategorySelect διαχειρίζονται επιλογές από dropdown menus, ενημερώνοντας το state productDetail με τις επιλεγμένες τιμές για κατηγορία, επιλεγμένο προϊόν και υποκατηγορία αντίστοιχα. Αυτό επιτρέπει την ευέλικτη ενημέρωση των στοιχείων προϊόντος με βάση τις επιλογές του χρήστη, ενισχύοντας τη διαδραστικότητα και τη λειτουργικότητα της φόρμας.

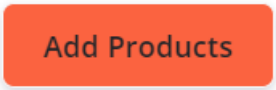
```
useEffect(() => {
    const categoryObjects = allCategories?.map((item) => ({
      value: item.category,
      label: item.category,
    }));
    setCategories(categoryObjects);
  }, [allCategories]);
```

```
  useEffect(() => {
    if (productDetail.category) {
      const matchingCategory = allCategories?.find(
        (item) => item?.category === productDetail?.category.value
      );
      console.log(matchingCategory);
      const matchingSubCategories = matchingCategory
        ? matchingCategory?.subCategories?.map((subCategory) => ({
            value: subCategory?.subCategory,
            label: subCategory?.subCategory,
          }))
        : [];
      setSubCategories(matchingSubCategories);
    }
  }, [productDetail.category, allCategories]);
  useEffect(() => {
    const transformedProducts = allProducts.map((product) => ({
      label: product.title,
      value: product.id,
    }));
    setRecommendedProduct(transformedProducts);
  }, [allProducts]);
  useEffect(() => {
    dispatch(getCategories());
    dispatch(getProduct());
  }, []);
  const [page, setPage] = useState(1);
  const [pageSize, setPageSize] = useState(100); // Set the number of products
per page
  const handlePageChange = (pageNumber) => {
    setPage(pageNumber);
  };
```

Το πρώτο useEffect αναλαμβάνει να μετατρέψει τις κατηγορίες προϊόντων από έναν πίνακα. Αυτή η διαδικασία εκτελείται κάθε φορά που αλλάζουν οι κατηγορίες προϊόντων.Το δεύτερο useEffect ελέγχει εάν υπάρχει μια επιλεγμένη κατηγορία προϊόντος και, αν ναι, βρίσκει τις αντίστοιχες υποκατηγορίες της. Αυτό επιτυγχάνεται μέσω της σύγκρισης της επιλεγμένης κατηγορίας με τις κατηγορίες που είναι διαθέσιμες στο σύστημα και την εύρεση των αντίστοιχων υποκατηγοριών, αν υπάρχουν. Το τρίτο useEffect μετατρέπει τα προϊόντα σε μια μορφή που απαιτείται για την εμφάνισή τους σε ένα dropdown ή σε άλλα UI στοιχεία. Αυτό εκτελείται κάθε φορά που αλλάζουν τα προϊόντα που είναι διαθέσιμα στο σύστημα

5.3.1 Σελίδα προϊόντων-απλή εμφάνιση

Αυτό το τμήμα κώδικα αφορά την εμφάνιση του τίτλου "Products" σε ένα κεφαλίδας κάρτας (Card Header) μαζί με ένα κουμπί για προσθήκη νέων προϊόντων. Το κουμπί αυτό ενεργοποιεί ένα modal για την προσθήκη νέων προϊόντων, ενώ ταυτόχρονα επαναφέρει τις τιμές του productDetail state σε προεπιλεγμένες τιμές.

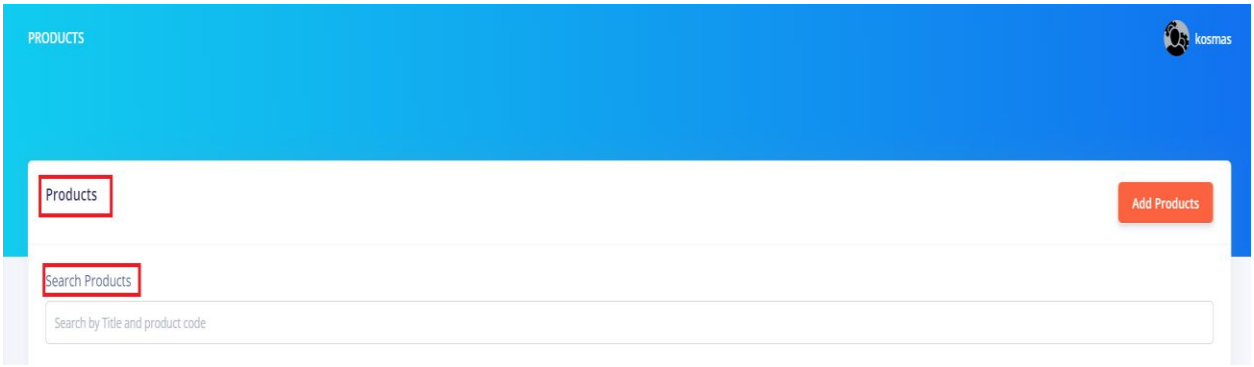


Εικόνα 5.7

```
<CardHeader className="bg-transparent">
                <h3 className="mb-0 d-inline-block">Products</h3>
                <Button
                  className="float-right bg-orange text-white"
                  onClick={() => {
                    addToggle();
                    setProductDetail({
                      title: "",
                      description: "",
                      price: 0,
                      images: [],
                      category: "",
                      subCategory: "",
                      deliveryDays: "",
                      selectedProduct: [],
                    });
                  }}
                >
                  Add Products
                </Button>
              </CardHeader>
```

Το CardHeader καθορίζει την κεφαλίδα της κάρτας όπου εμφανίζονται τα προϊόντα. Ο τίτλος "Products" είναι μια απλή κεφαλίδα χωρίς χρώμα (bg-transparent). Το κουμπί "Add Products" βρίσκεται στη δεξιά πλευρά της κεφαλίδας και έχει χρώμα φόντου πορτοκαλί (bg-orange) και κείμενο λευκό (text-white). Όταν πατιέται το κουμπί, εκτελείται η συνάρτηση addToggle() για να εμφανιστεί το modal για την προσθήκη νέων προϊόντων. Επιπλέον, οι τιμές του productDetail state επαναφέρονται στις προεπιλεγμένες τιμές τους, προετοιμάζοντας τη φόρμα για την προσθήκη νέου προϊόντος.

5.3.2 Καρτέλα και φίλτρο αναζήτησης



Εικόνα 5.8

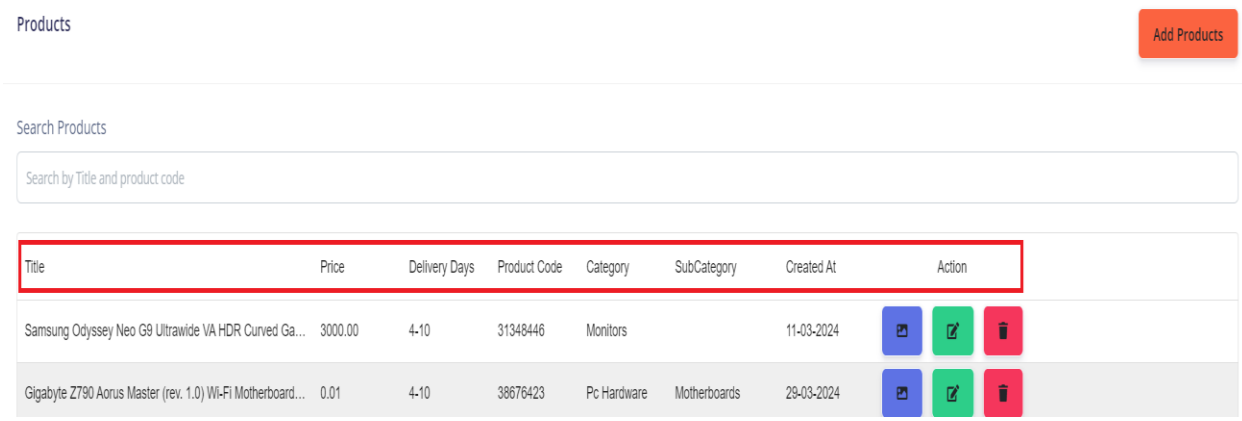
Αυτό το τμήμα κώδικα περιλαμβάνει ένα φίλτρο αναζήτησης για προϊόντα, έναν πίνακα δεδομένων (DataGrid) για την εμφάνιση των προϊόντων και έναν διαχειριστή σελιδοποίησης (Pagination) για την πλοήγηση μεταξύ των σελίδων με προϊόντα.

```
<CardBody>
                {" "}
                <FormGroup>
                  <Label for="search">Search Products</Label>
                  <Input
                    type="text"
                    name="search"
                    id="search"
                    placeholder="Search by Title and product code"
                    value={searchQuery}
                    onChange={handleSearch}
                  />
                </FormGroup>
                <LoadingOverlay
                  active={loading}
                  spinner
                  text="Products Loading...."
                >

                  <StripedDataGrid
                    autoHeight
                    autoWidth
                    columns={columns}
                    rows={filteredProducts.slice(
                      (page - 1) * pageSize,
                      page * pageSize
```

```
                    )}
                    disableSelectionOnClick={false}
                    getRowClassName={(params) =>
                      params.indexRelativeToCurrentPage % 2 === 0
                        ? "odd"
                        : "even"
                    }
                    hideFooterPagination={true}
                    pagination
                    pageSize={pageSize}
                    rowsPerPageOptions={[100]} // Set the number of products per
page
                  />
                  <div className="d-flex justify-content-center mt-3">
                    <Pagination>
                      {Array.from({
                        length: Math.ceil(allProducts.length / pageSize),
                      }).map((_, index) => (
                        <Pagination.Item
                          key={index + 1}
                          active={index + 1 === page}
                          onClick={() => handlePageChange(index + 1)}
                        >
                          {index + 1}
                        </Pagination.Item>
                      ))}
                    </Pagination>
                  </div>
                </LoadingOverlay>
```

Το CardBody περιλαμβάνει ένα φίλτρο αναζήτησης προϊόντων, το οποίο επιτρέπει στον χρήστη να αναζητήσει προϊόντα βάσει του τίτλου και του κωδικού προϊόντος. Οι αλλαγές στο φίλτρο αναζήτησης αντικατοπτρίζονται αμέσως στο state searchQuery μέσω της συνάρτησης handleSearch.Η ετικέτα FormGroup δημιουργεί ένα input πεδίο για την εισαγωγή κειμένου αναζήτησης. Όταν ο χρήστης εισάγει κείμενο στο πεδίο αναζήτησης, το state searchQuery ενημερώνεται με την τρέχουσα τιμή του πεδίου. Το LoadingOverlay χρησιμοποιείται για να εμφανιστεί ένα spinner κατά τη φόρτωση των προϊόντων. Αν το loading είναι true, το spinner εμφανίζεται μέχρι να ολοκληρωθεί η διαδικασία φόρτωσης των προϊόντων.Ο DataGrid χρησιμοποιείται για την εμφάνιση των προϊόντων σε μορφή πίνακα. Τα δεδομένα προϊόντων φιλτράρονται ανάλογα με την αναζήτηση που έχει εφαρμοστεί και στη συνέχεια χωρίζονται σε σελίδες βάσει του pageSize και του page. Ο χρήστης μπορεί να πλοηγηθεί μεταξύ των σελίδων χρησιμοποιώντας τα κουμπιά σελιδοποίησης του Pagination.

Εδώ ορίζονται οι στήλες που θα εμφανίζονται στον πίνακα δεδομένων για κάθε προϊόν.



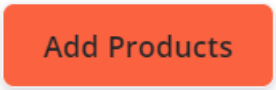
Εικόνα 5.9

```
const columns = [
    {
      field: "title",
      headerName: "Title",
      width: 400,
    },
    {
      field: "price",
      headerName: "Price",
      width: 120,
    },
    {
      field: "deliveryDays",
      headerName: "Delivery Days",
      width: 120,
    },
    {
      field: "productCode",
      headerName: "Product Code",
      width: 120,
    },
    {
      field: "category",
      headerName: "Category",
      width: 120,
      renderCell: (params) => {
        return (
          <>
            <span>{params.row.category.label}</span>
          </>
```

```
        );
      },
    },
```

5.4 ΚΑΤΑΧΩΡΗΣΗ ΕΝΟΣ ΝΕΟΥ ΠΡΟΙΟΝΤΟΣ

5.4.1 Κουμπί προσθήκης προϊόντος



Εικόνα 5.10

Σε αυτό το τμήμα κώδικα ορίζεται η κεφαλίδα του modal για την προσθήκη νέων προϊόντων ("Add Products"). Επιπλέον, ορίζεται μια φόρμα (Form) που θα χρησιμοποιηθεί για την υποβολή των δεδομένων για τη δημιουργία νέου προϊόντος.

```
<Modal isOpen={addModal} toggle={addToggle}>
          <ModalHeader toggle={addToggle}>Add Products</ModalHeader>
          <Form
            onSubmit={(e) => {
              e.preventDefault();
              dispatch(
                createProduct(productDetail, () => {
                  addToggle();
                  setProductDetail({
                    title: "",
                    description: "",
                    price: 0,
                    images: [],
                    category: "",
                    subCategory: "",
                    productCode: "",
                    selectedProduct: [],
                  });
                })
              );
            }}
```

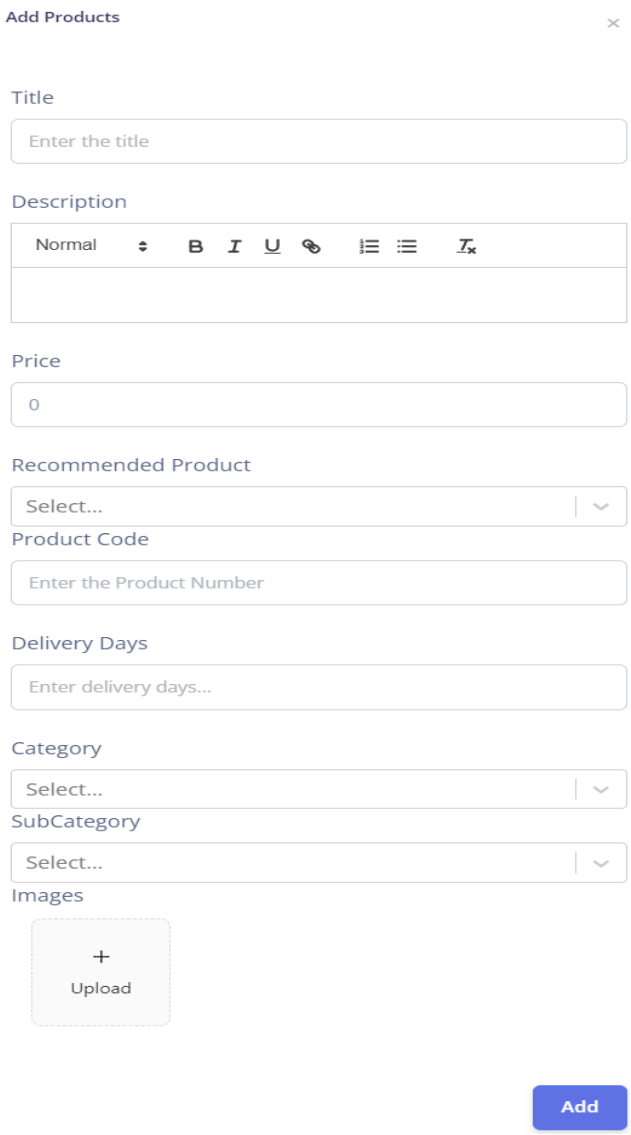
Η κεφαλίδα (ModalHeader) περιέχει τον τίτλο του modal ("Add Products"). Το toggle={addToggle}

επιτρέπει στον χρήστη να κλείσει το modal πατώντας το εικονίδιο "X" στη γωνία του modal ή πατώντας έξω από το modal.

Η φόρμα (Form) καθορίζει τη συμπεριφορά όταν ο χρήστης υποβάλει τα δεδομένα της. Η onSubmit συνάρτηση εκτελείται όταν ο χρήστης υποβάλει τη φόρμα. Εδώ, εμποδίζεται η προεπιλεγμένη συμπεριφορά της φόρμας (η οποία θα επαναφέρει τη σελίδα) με τη χρήση της e.preventDefault().

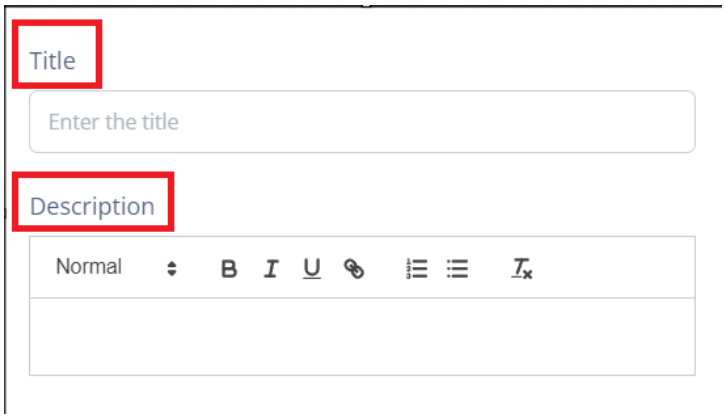
Μετά την υποβολή της φόρμας, καλείται η συνάρτηση createProduct του Redux store για τη δημιουργία του νέου προϊόντος με τα δεδομένα που παρέχονται στο productDetail. Αν η δημιουργία είναι επιτυχής, το modal κλείνει με το addToggle(), και τα δεδομένα στο productDetail επαναφέρονται στις προεπιλεγμένες τιμές.

5.4.2 Προσθήκη περιγραφής προϊόντος



Εικόνα 5.11

Σε αυτό το τμήμα κώδικα ορίζεται το σώμα του modal που περιλαμβάνει τη φόρμα για την προσθήκη νέων προϊόντων. Η φόρμα περιέχει πεδία όπως τίτλος και περιγραφή για το νέο προϊόν.



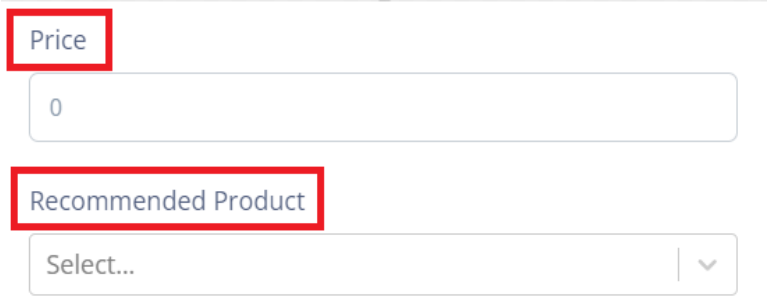
Εικόνα 5.12

```
<FormGroup>
  <Label>Title</Label>
  <Input
    type="text"
    placeholder="Enter the title"
    name="title"
    required
    value={productDetail?.title}
    onChange={(e) => {
      handleChange(e);
    }}
  />
</FormGroup>
<FormGroup>
  <Label>Description</Label>
  <ReactQuill
    theme="snow"
    value={productDetail.description}
    onChange={(value) => {
      setProductDetail((prevDetail) => ({
        ...prevDetail,
        description: value,
      }));
    }}
  />
```

```
                </FormGroup>
```

Η <Form> ορίζει τη φόρμα που θα χρησιμοποιηθεί για την υποβολή των δεδομένων για τη δημιουργία νέου προϊόντος. Το onSubmit εμποδίζει την προεπιλεγμένη συμπεριφορά της φόρμας ώστε να αποτραπεί η ανανέωση της σελίδας μετά την υποβολή της φόρμας. Στη συνέχεια ακολουθούν τα πεδία της φόρμας όπως ο τίτλος και η περιγραφή του προϊόντος. Κάθε πεδίο περιέχει ένα <FormGroup> που περιλαμβάνει ένα <Label> για το όνομα του πεδίου και ένα <Input> ή <ReactQuill> για την είσοδο δεδομένων. Το value και το onChange χρησιμοποιούνται για τη δέσμευση των δεδομένων στο state και την ενημέρωσή τους κατά την είσοδο δεδομένων από τον χρήστη.

Σε αυτό το τμήμα κώδικα ορίζονται περισσότερα πεδία της φόρμας για την προσθήκη νέων προϊόντων, όπως η τιμή και η επιλογή προτεινόμενου προϊόντος.



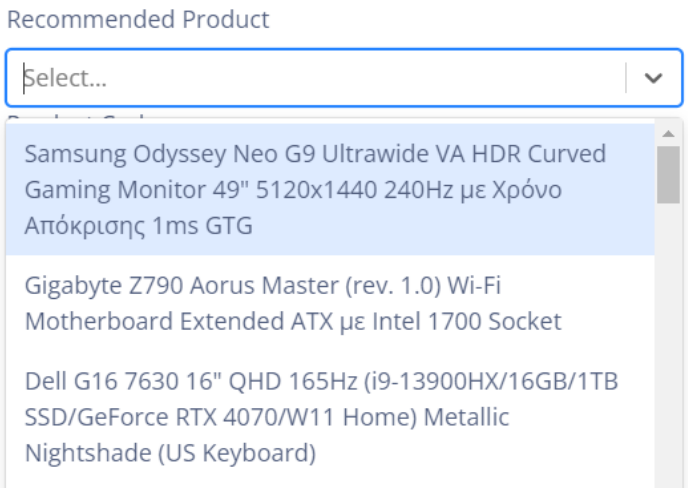
Εικόνα 5.13

```
<FormGroup>
                  <Label>Price</Label>
                  <Input
                    type="text" // Change input type to text
                    placeholder="Enter the price..."
                    name="price"
                    required
                    value={productDetail?.price}
                    onChange={(e) => {
                      const { value } = e.target;
                      // Validate input to allow only numbers and exactly two
decimal places
                      const regex = /^\d+(\.\d{0,2})?$/;
                      if (regex.test(value) || value === "") {
                        handleChange(e);
                      }
                    }}
                  />
                </FormGroup>
```

```
<Label>Recommended Product</Label>
<Select
  isMulti
  className="w-100"
  options={recommendedProduct}
  value={productDetail?.selectedProduct}
  onChange={handleRecommendedSelect}
/>
```

Το πεδίο "Price" χρησιμοποιεί ένα <Input> για την είσοδο της τιμής του προϊόντος. Το type="text" χρησιμοποιείται αντί για type="number" για να επιτραπεί η πληκτρολόγηση ακόμη και των δεκαδικών τιμών.

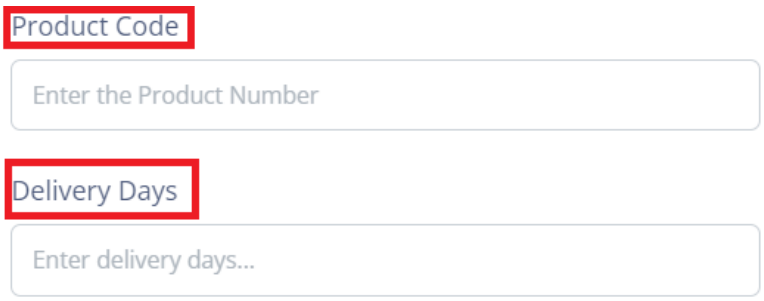
Ένας έλεγχος ακρίβειας δεκαδικών ψηφίων πραγματοποιείται στην onChange λειτουργία του πεδίου "Price". Το πεδίο δέχεται μόνο αριθμητικές τιμές και ακριβώς δύο δεκαδικά ψηφία. Το πεδίο "Recommended Product" χρησιμοποιεί ένα <Select> για την επιλογή προτεινόμενων προϊόντων. Ο χρήστης μπορεί να επιλέξει πολλαπλά προϊόντα από τη λίστα των διαθέσιμων προϊόντων.



Εικόνα 5.14

Το πεδίο αυτό συνδέεται με το state productDetail.selectedProduct και τη συνάρτηση handleRecommendedSelect ώστε να ενημερώνεται το state με τις επιλεγμένες επιλογές του χρήστη.

Εδώ ορίζονται δύο ακόμα πεδία της φόρμας για την προσθήκη νέων προϊόντων: το "Product Code" και οι "Delivery Days"



Εικόνα 5.15

```
<FormGroup>
                  <Label>Product Code</Label>
                  <Input
                    type="text"
                    placeholder="Enter the Product Number"
                    name="productCode"
                    required
                    value={productDetail?.productCode}
                    pattern="[a-zA-Z0-9]+"
                    onChange={(e) => {
                      handleChange(e);
                    }}
                  />
                </FormGroup>
                <FormGroup>
                  <Label>Delivery Days</Label>
                  <Input
                    type="number"
                    placeholder="Enter delivery days..."
                    name="deliveryDays"
                    required
                    value={productDetail?.deliveryDays}
                    onChange={(e) => {
                      handleChange(e);
                    }}
                  />
```

Το πεδίο "Product Code" χρησιμοποιεί ένα <Input> με type="text" για την εισαγωγή του κωδικού του προϊόντος. Το πεδίο είναι υποχρεωτικό και δέχεται μόνο αλφαριθμητικούς χαρακτήρες χρησιμοποιώντας το pattern="[a-zA-Z0-9]+". Το πεδίο "Delivery Days" χρησιμοποιεί ένα <Input> με type="number" για την εισαγωγή των ημερών παράδοσης του προϊόντος. Το πεδίο είναι υποχρεωτικό και δέχεται μόνο αριθμητικές τιμές. Και τα δύο πεδία συνδέονται με τη συνάρτηση handleChange για την ενημέρωση του state κατά την εισαγωγή δεδομένων από τον χρήστη.

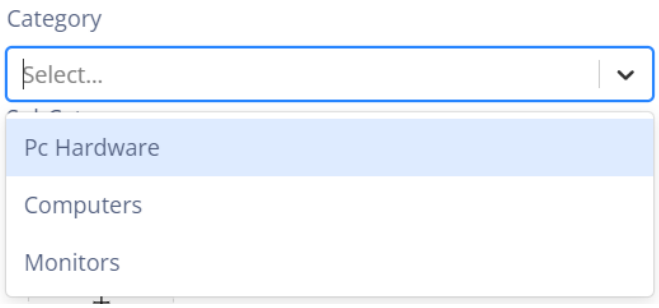
Σε αυτό το τμήμα κώδικα ορίζονται τα πεδία "Category", "SubCategory" και "Images" της φόρμας για την προσθήκη νέων προϊόντων.

Εικόνα 5.16

```
<Label>Category</Label>
                <Select
                  className="w-100"
                  options={categories}
                  value={productDetail?.category}
                  onChange={handleSelect}
                />
                <Label>SubCategory</Label>
                <Select
                  className="w-100"
                  options={subCategories}
                  value={productDetail?.subCategory}
                  onChange={handleSubcategorySelect}
                />
                <Label>Images</Label>
                <PictureWall
                  detail={productDetail}
                  setDetail={setProductDetail}
                />
```

Το πεδίο "Category" χρησιμοποιεί ένα <Select> για την επιλογή της κατηγορίας του προϊόντος. Ο χρήστης μπορεί να επιλέξει μία κατηγορία από τη λίστα των διαθέσιμων κατηγοριών. Το πεδίο αυτό συνδέεται με το state productDetail.category και τη συνάρτηση handleSelect ώστε να ενημερώνεται το state με την επιλεγμένη κατηγορία από τον χρήστη.

Το πεδίο "SubCategory" χρησιμοποιεί επίσης ένα <Select> για την επιλογή της υποκατηγορίας του προϊόντος. Ο χρήστης μπορεί να επιλέξει μία υποκατηγορία από τη λίστα των διαθέσιμων υποκατηγοριών που σχετίζονται με την επιλεγμένη κατηγορία. Το πεδίο αυτό συνδέεται με το state productDetail.subCategory και τη συνάρτηση handleSubcategorySelect ώστε να ενημερώνεται το state με την επιλεγμένη υποκατηγορία από τον χρήστη.

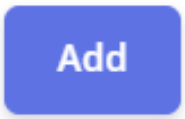


Εικόνα 5.17

Το πεδίο "Images" χρησιμοποιεί έναν custom component <PictureWall> για την επισύναψη εικόνων στο προϊόν. Το custom component αυτό συνδέεται με το state productDetail και τη συνάρτηση setProductDetail ώστε να επεξεργάζεται το state του προϊόντος και να ενημερώνεται με τις επιλεγμένες εικόνες από τον χρήστη.

5.4.3 Προσθήκη εικόνας στα προιόντα

Σε αυτό το τμήμα κώδικα ορίζεται το κουμπί "Add" το οποίο χρησιμοποιείται για την υποβολή της φόρμας προσθήκης νέων προϊόντων.



Εικόνα 5.18

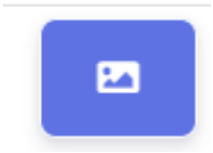
```
<ModalFooter>
              <Button color="primary" type="submit" disabled={loading}>
                {loading ? <Spinner size="md" /> : "Add"}
              </Button>
            </ModalFooter>
```

Το <Button> χρησιμοποιείται για την εμφάνιση του κουμπιού "Add" στο κάτω μέρος του παραθύρου της φόρμας προσθήκης προϊόντων.

Το πεδίο type="submit" σημαίνει ότι το κουμπί θα εκτελέσει τη δράση υποβολής της φόρμας όταν πατηθεί.

Το disabled={loading} χρησιμοποιείται για να απενεργοποιήσει το κουμπί όταν ο φόρμα βρίσκεται σε κατάσταση φόρτωσης (loading), προκειμένου να αποτραπεί η υποβολή πολλαπλών αιτήσεων. Όταν η μεταβλητή loading είναι true, το κουμπί εμφανίζει έναν spinner (<Spinner size="md" />) αντί για τη λέξη "Add", δείχνοντας ότι η διαδικασία φόρτωσης είναι σε εξέλιξη. Αυτό εξασφαλίζει τη συμμόρφωση με τις αρχές της καλής διαχείρισης του UI και την αποφυγή διπλών υποβολών.

5.4.4 Προεπισκόπηση εικόνων των προϊόντων



Εικόνα 5.19

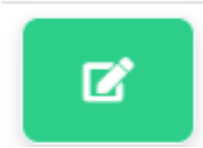
Το παραπάνω τμήμα κώδικα δημιουργεί ένα modal παράθυρο που εμφανίζει εικόνες προϊόντων.

```
<Modal isOpen={imgModal} toggle={imagesToggle}>
          <ModalHeader toggle={imagesToggle} className="border-bottom">
            <h3 className="mb-0">Product Images</h3>
          </ModalHeader>
          <ModalBody>
            <Row md={2} lg={3}>
              {selectedItemImages &&
                selectedItemImages?.map((item) => {
                  return (
                    <Col>
                      <img
                        src={item}
                        width="100%"
                        height="100%"
                        className="mb-3 border shadow-lg"
                      />
                    </Col>
                  );
                })}
            </Row>
          </ModalBody>
        </Modal>
```

Το στοιχείο <Modal> χρησιμοποιείται για να δημιουργήσει ένα επικαλυπτόμενο παράθυρο (modal) που εμφανίζεται όταν η μεταβλητή imgModal είναι true. Ο τίτλος του modal ορίζεται ως "Product Images"

χρησιμοποιώντας το <ModalHeader>, το οποίο περιλαμβάνει επίσης ένα κουμπί "toggle" για το κλείσιμο του modal. Το περιεχόμενο του modal βρίσκεται μέσα στο <ModalBody> και αποτελείται από μια σειρά εικόνων προϊόντων. Αυτές οι εικόνες προϊόντων εμφανίζονται σε μια γραμμή, ή σε πολλαπλές γραμμές ανάλογα με το μέγεθος της οθόνης, χρησιμοποιώντας το <Row> και το <Col>. Κάθε εικόνα προϊόντος παρουσιάζεται μέσα σε ένα <Col>, το οποίο έχει προσαρμοσμένο πλάτος, ύψος και στυλ για τον περιγράφοντα περίγυρο. Με αυτόν τον τρόπο, το modal παρέχει μια ευέλικτη και ευανάγνωστη διεπαφή χρήστη για την προβολή των εικόνων προϊόντων.

5.4.5 Επεξεργασία υπάρχοντος προϊόντος



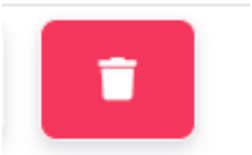
Εικόνα 5.20

Σε αυτό το τμήμα κώδικα ορίζεται το modal που εμφανίζεται για την επεξεργασία προϊόντων. Περιέχει μια φόρμα που επιτρέπει στον χρήστη να ενημερώσει τα στοιχεία ενός προϊόντος.

```
<Modal isOpen={editModal} toggle={editToggle}>
          <ModalHeader toggle={editToggle}>Update Products</ModalHeader>
          <Form
            onSubmit={(e) => {
              e.preventDefault();
              dispatch(
                editProduct(productDetail, selectedId, () => {
                  editToggle();
                  setProductDetail({
                    title: "",
                    description: "",
                    price: 0,
                    images: [],
                    category: "",
                    subCategory: "",
                    productCode: "",
                    deliveryDays: "",
                    selectedProduct: [],
                  });
                })
              );
            }}
```

Το Modal χρησιμοποιείται για τη δημιουργία ενός modal παραθύρου που εμφανίζεται όταν η μεταβλητή editModal είναι true. Ο τίτλος του modal ορίζεται μέσα στο <ModalHeader> χρησιμοποιώντας το κείμενο "Update Products".Η φόρμα που περιλαμβάνεται στο <Form> χρησιμοποιείται για την υποβολή των ενημερωμένων πληροφοριών του προϊόντος. Όταν ο χρήστης υποβάλλει τη φόρμα, εκτελείται η συνάρτηση onSubmit, η οποία αποτρέπει την προεπιλεγμένη συμπεριφορά υποβολής της φόρμας με τη χρήση της e.preventDefault(). Στη συνέχεια, η συνάρτηση dispatch καλείται για να εκτελέσει τη δράση επεξεργασίας του προϊόντος. Μετά την επιτυχή ολοκλήρωση της δράσης επεξεργασίας, το modal κλείνεται με την εκτέλεση της συνάρτησης editToggle() και τα στοιχεία του state productDetail αρχικοποιούνται στις αρχικές τους τιμές.

5.4.6 Διαγραφή υπάρχοντος προιόντος



Εικόνα 5.21

Το παρακάτω τμήμα κώδικα δημιουργεί ένα modal παράθυρο που εμφανίζει ένα πλαίσιο επιβεβαίωσης για τη διαγραφή ενός προϊόντος.

```
<Modal isOpen={confirmModal} toggle={confirmtoggle}>
          <ModalHeader toggle={confirmtoggle}>Confirmation Box</ModalHeader>
          <ModalBody>
            <h5 className="text-center ">
              Are you Sure you delete this product?
            </h5>
            <div className="d-flex justify-content-center align-items-center my-
3">
              <Button
                className="bg-primary text-white w-25 mx-2"
                onClick={() => {
                  dispatch(
                    deleteProduct(selectedId, () => {
                      confirmtoggle();
                    })
                  );
                }}
              >
                {loading ? <Spinner size="md" /> : "Yes"}
              </Button>
```

```
              <Button
                className="bg-danger text-white w-25"
                onClick={confirmtoggle}
              >
                No
              </Button>
            </div>
          </ModalBody>
        </Modal>
```

Το στοιχείο <Modal> χρησιμοποιείται για να δημιουργήσει ένα modal παράθυρο που εμφανίζεται όταν η μεταβλητή confirmModal είναι true. Ο τίτλος του modal ορίζεται ως "Confirmation Box" χρησιμοποιώντας το <ModalHeader>, το οποίο περιλαμβάνει επίσης ένα κουμπί "toggle" για το κλείσιμο του modal. Το περιεχόμενο του modal βρίσκεται μέσα στο <ModalBody> και περιλαμβάνει ένα μήνυμα επιβεβαίωσης διαγραφής προϊόντος. Τα κουμπιά "Yes" και "No" που εμφανίζονται κάτω από το μήνυμα επιβεβαίωσης είναι υπεύθυνα για την επιβεβαίωση ή την ακύρωση της διαγραφής αντίστοιχα. Όταν πατηθεί το κουμπί "Yes", εκτελείται η λειτουργία διαγραφής προϊόντος και το modal κλείνει. Εάν υπάρχει φόρτωση κατά τη διαγραφή, εμφανίζεται ένα spinner αντί για το κείμενο "Yes".

5.5 Εκπτώσεις

5.5.1 Δημιουργία στηλών προϊόντων

Ο παραπάνω κώδικας ορίζει τις στήλες για τον πίνακα προϊόντων που έχουν έκπτωση με τον ίδιο τρόπο που γινόταν και πρίν στα προιόντα.

```
const columns = [
    {
      field: "title",
      headerName: "Title",
      width: 230,
    },
    {
      field: "price",
      headerName: "Price",
      width: 120,
    },
    {
      field: "discount",
      headerName: "Discount Price",
      width: 120,
    },
    {
```

```
      field: "productCode",
      headerName: "Product Code",
      width: 120,
    },
```

Κάθε στήλη ορίζεται με ένα αντικείμενο που περιέχει τα πεδία field, headerName και width.

Το πεδίο field καθορίζει το όνομα του πεδίου του προϊόντος που θα εμφανίζεται στη στήλη.

Το πεδίο headerName καθορίζει τον τίτλο της στήλης που θα εμφανίζεται στον πίνακα.

Το πεδίο width καθορίζει το πλάτος της στήλης στον πίνακα, σε pixels.

5.5.2 Λειτουργίες διαχείρησης σελίδας

Αυτό το τμήμα κώδικα περιλαμβάνει λειτουργίες που σχετίζονται με τη διαχείριση της σελιδοποίησης και της ανανέωσης των προϊόντων.

```
const handleDiscountChange = (e) => {
    const { name, value } = e.target;
    setDetail((prevState) => ({
      ...prevState,
      [name]: value,
    }));
  };
  useEffect(() => {
    dispatch(getProduct());
  }, []);

  const [page, setPage] = useState(1);
  const [pageSize, setPageSize] = useState(10); // Set the number of products per
page
  const handlePageChange = (pageNumber) => {
    setPage(pageNumber);
  };

  const handlePageSizeChange = (event) => {
    setPageSize(event.target.value);
    setPage(1); // Reset page when changing page size
  };
```

Η συνάρτηση handleDiscountChange χειρίζεται την αλλαγή της έκπτωσης ενός προϊόντος. Όταν ο χρήστης εισάγει ή επιλέγει μια νέα τιμή έκπτωσης για ένα προϊόν, αυτή η συνάρτηση καλείται και ενημερώνει την κατάσταση των λεπτομερειών του προϊόντος με τη νέα τιμή έκπτωσης, χρησιμοποιώντας τη συνάρτηση setDetail για να ενημερώσει την κατάσταση των λεπτομερειών του

προϊόντος στη συστατική useState detail. Η useEffect χρησιμοποιείται για να ανακτήσει τα προϊόντα όταν η σελίδα φορτώνεται για πρώτη φορά. Εδώ, η dispatch καλεί τη λειτουργία getProduct() για να ανακτήσει τα προϊόντα από τον διακομιστή.

Το page και το pageSize χρησιμοποιούνται για την αναπαράσταση και τη διαχείριση της σελιδοποίησης των προϊόντων. Η συνάρτηση handlePageChange καλείται κάθε φορά που ο χρήστης αλλάζει σελίδα στη λίστα προϊόντων και ενημερώνει την τρέχουσα σελίδα στην κατάσταση. Επίσης, η συνάρτηση handlePageSizeChange χειρίζεται την αλλαγή του μεγέθους της σελίδας, ενημερώνοντας το pageSize και επαναφέροντας την τρέχουσα σελίδα στην πρώτη, ώστε να εξασφαλιστεί ότι η πρώτη σελίδα εμφανίζεται όταν αλλάζει το μέγεθος της σελίδας.

5.5.3 Καρτέλα προϊόντων με έκπτωση

Title	Price	Discount Price	Product Code	Category	SubCategory	Created At	Action
Samsung Odyssey Neo G9 Ult…	3000.00	10	31348446	Monitors		11-03-2024	%
Gigabyte Z790 Aorus Master (r…	0.01	0	38676423	Pc Hardware	Motherboards	29-03-2024	%
Dell G16 7630 16" QHD 165H…	1899.99	0	48202201	Computers	laptops	28-03-2024	%
Vengeance Oblivion-V91 Raze…	2057.00	0	45016818	Computers	Desktop pcs	29-03-2024	%
Gigabyte GeForce RTX 4070 …	699.70	0	41609252	Pc Hardware	Gpus	29-03-2024	%
Asus TUF Gaming A17 (2023) …	1799.29	0	48616273	Computers	laptops	28-03-2024	%
Corsair ICUE H150i Elite LCD …	338.88	0	40582727	Pc Hardware	Liquid coolers	29-03-2024	%
Asus ROG Maximus Z790 Her…	602.90	10	38442875	Pc Hardware	Motherboards	11-03-2024	%
Test	3000	0	123	Pc Hardware		29-04-2024	%
Asus ROG Ryuo III 360 ARGB…	363.98	0	40114369	Pc Hardware	Liquid coolers	29-03-2024	%

curre 2 3

Εικόνα 5.22

Αυτό το τμήμα κώδικα αφορά την προβολή των προϊόντων με έκπτωση και τη δυνατότητα περιήγησης σε διάφορες σελίδες τους.

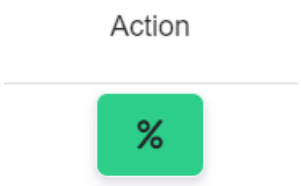
```
            <CardHeader className="bg-transparent">
              <h3 className="mb-0">Discount</h3>
            </CardHeader>
            <CardBody>
              <LoadingOverlay
                active={loading}
```

```
                spinner
                text="Products Loading...."
              >
                <StripedDataGrid
                  autoHeight
                  autoWidth
                  columns={columns}
                  rows={allProducts.slice(
                    (page - 1) * pageSize,
                    page * pageSize
                  )}
                  disableSelectionOnClick={false}
                  getRowClassName={(params) =>
                    params.indexRelativeToCurrentPage % 2 === 0
                      ? "odd"
                      : "even"
                  }
                  hideFooterPagination={true}
                  pagination
                  pageSize={pageSize}
                  rowsPerPageOptions={[10]} // Set the number of products per
page
                />
                <div className="d-flex justify-content-center mt-3">
                  <Pagination>
                    {Array.from({
                      length: Math.ceil(allProducts.length / pageSize),
                    }).map((_, index) => (
                      <Pagination.Item
                        key={index + 1}
                        active={index + 1 === page}
                        onClick={() => handlePageChange(index + 1)}
                      >
                        {index + 1}
                      </Pagination.Item>
                    ))}
                  </Pagination>
                </div>
              </LoadingOverlay>
            </CardBody>
```

Το CardHeader προσδιορίζει τον τίτλο "Discount", προσφέροντας έναν ευδιάκριτο τρόπο για την αναγνώριση του περιεχομένου της κάρτας. Μέσα στο CardBody, ο StripedDataGrid χρησιμοποιείται για την οργάνωση και την προβολή των προϊόντων, προσφέροντας έναν καθαρό και εύκολο τρόπο

προβολής. Η σελιδοποίηση επιτυγχάνεται μέσω του Pagination component, επιτρέποντας στον χρήστη να περιηγηθεί μεταξύ των διαφόρων σελίδων με προϊόντα. Τέλος, οι μεταβλητές pageSize και page προσφέρουν τη δυνατότητα προσαρμογής του μεγέθους της σελίδας και του αριθμού των προϊόντων που εμφανίζονται ανά σελίδα, εξασφαλίζοντας έτσι μια εξατομικευμένη εμπειρία περιήγησης για τον χρήστη.

5.5.4 Κουμπί προσθήκης έκπτωσης



Εικόνα 5.23

Αυτό το τμήμα κώδικα προσθέτει μια στήλη ενεργειών στον πίνακα προϊόντων. Σε αυτή τη στήλη προστίθενται κουμπιά που επιτρέπουν στον χρήστη να εκτελέσει διάφορες ενέργειες, όπως προσθήκη προϊόντος στο καλάθι.

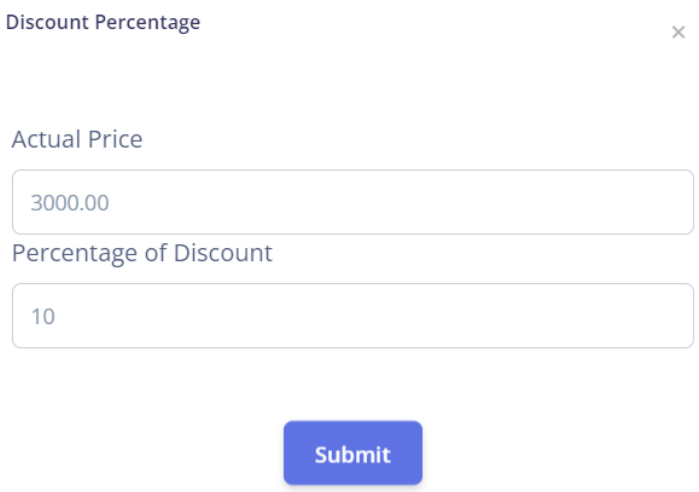
```
{
      field: "actions",
      type: "actions",
      headerName: "Action",
      width: 230,
      renderCell: (params) => {
        return (
          <>
            <Button
              className="bg-success"
              onClick={() => {
                setDetail({
                  actualPrice: params?.row?.price,
                  discountPrice: params?.row.discount
                    ? params?.row.discount
                    : 0,
                });
                addToggle();
                setSelectedId(params.row.id);
              }}
            >
              <i className="fas fa-percent text-black"></i>
```

```
            </Button>
          </>
        );
      },
    },
```

Ορίζει μια στήλη "actions" στον πίνακα προϊόντων. Αυτή η στήλη προβάλλει ένα κουμπί ενέργειας για κάθε προϊόν, επιτρέποντας στον χρήστη να εκτελέσει μια ενέργεια. Η ιδιότητα type: "actions" δηλώνει ότι αυτή η στήλη περιέχει ενέργειες. Ο τίτλος της στήλης ορίζεται ως "Action" μέσω της ιδιότητας headerName. Η ιδιότητα width: 230 καθορίζει το πλάτος της στήλης στον πίνακα.

Η μέθοδος renderCell παρέχει το περιεχόμενο που θα εμφανίζεται σε κάθε κελί της στήλης "actions". Σε αυτήν τη μέθοδο, δημιουργείται ένα κουμπί με την κλάση "bg-success", το οποίο όταν πατηθεί, εκτελεί μια σειρά ενεργειών. Οι ενέργειες περιλαμβάνουν τον προσδιορισμό των λεπτομερειών του επιλεγμένου προϊόντος, όπως η τιμή και η τιμή με έκπτωση, με τη χρήση της συνάρτησης setDetail. Επίσης, η κατάσταση του παραθύρου προσθήκης προϊόντος αλλάζει με τη χρήση της μεθόδου addToggle, ενώ ορίζεται το επιλεγμένο ID του προϊόντος με τη μέθοδο setSelectedId. Έτσι προσφέρει λειτουργικότητα για τη διαχείριση των προϊόντων με έκπτωση, επιτρέποντας στον χρήστη να προβάλει και να επεξεργαστεί τα δεδομένα τους.

5.5.5 Καρτέλα προσθήκης έκπτωσης



Εικόνα 5.24

Αυτό το Modal εμφανίζει τα πεδία επεξεργασίας για το ποσοστό έκπτωσης του προϊόντος και επιτρέπει στον χρήστη να επεξεργαστεί το ποσοστό έκπτωσης ενός από αυτά.

```
<Modal isOpen={addModal} toggle={addToggle}>
          <ModalHeader toggle={addToggle}>Discount Percentage</ModalHeader>
          <Form
            onSubmit={(e) => {
              e.preventDefault();
              let obj = {
                id: selectedId,
                discount: detail?.discountPrice,
              };
              dispatch(
                editDiscount(obj, () => {
                  addToggle();
                })
              );
            }}
          >
            <ModalBody>
              <Label>Actual Price</Label>
              <Input
                type="text"
                value={detail?.actualPrice}
                name="actualPrice"
                onChange={(e) => {
                  handleDiscountChange(e);
                }}
              />
              <Label>Percentage of Discount</Label>
              <Input
                type="number"
                value={detail?.discountPrice}
                name="discountPrice"
                onChange={(e) => {
                  handleDiscountChange(e);
                }}
              />
            </ModalBody>
            <ModalFooter className="d-flex justify-content-center align-items-
center">

              <Button color="primary" type="submit" disabled={loading} style={{
color: 'black' }}>
                {loading ? <Spinner size="md" /> : "Submit"}
              </Button>
            </ModalFooter>
          </Form>
```

```
        </Modal>
```

εστιάζει στη δημιουργία ενός modal παραθύρου που επιτρέπει στον χρήστη να επεξεργαστεί το ποσοστό έκπτωσης για ένα προϊόν. Μέσα στο modal, ο χρήστης μπορεί να εισάγει το νέο ποσοστό έκπτωσης μέσω ενός πεδίου εισαγωγής και να υποβάλει την αλλαγή πατώντας το κουμπί "Submit". Οι μεταβλητές isOpen και toggle ρυθμίζουν την εμφάνιση και τη λειτουργία του modal, ενώ η μεταβλητή loading χρησιμοποιείται για να εμφανιστεί ένα spinner κατά τη διάρκεια της αποστολής των δεδομένων. Μετά την υποβολή της φόρμας, ο κώδικας εκτελεί την απαραίτητη λειτουργία επεξεργασίας του ποσοστού έκπτωσης, όπως για παράδειγμα την ενημέρωση της βάσης δεδομένων. Συνολικά, δημιουργεί διαδικασία για τη διαχείριση του ποσοστού έκπτωσης ενός προϊόντος

ΚΕΦΑΛΑΙΟ 6: FIREBASE

6.1 Login

Σε αυτό το τμήμα του κώδικα πραγματοποιείται ο έλεγχος εισόδου του χρήστη. Χρησιμοποιείται η λειτουργία async/await για την ασύγχρονη εκτέλεση και ελέγχουνται οι παράμετροι εισόδου, όπως το email και ο κωδικός, με τη χρήση του Firebase για την αυθεντικοποίηση και τη βάση δεδομένων Firestore για την ανάκτηση πληροφοριών χρήστη.

```
export const userLogin = (userDetail) => async (dispatch) => {
  dispatch(loginLoader(true));
  try {
    await firebase
      .auth()
      .signInWithEmailAndPassword(userDetail.email, userDetail.password)
      .then((data) => {
        firebase
          .firestore()
          .collection("users")
          .doc(data.user.uid)
          .onSnapshot((doc) => {
            if (doc?.data()?.role == "admin") {
              var tempUser = {};
              tempUser = { id: doc.id, ...doc.data() };
              dispatch(loginLoader(false));
              //console.log(tempUser,'tempuser')
              dispatch({ type: "LOGIN", payload: tempUser });
            } else {
              alert("Sorry only admin can access this panel!");
              dispatch(loginLoader(false));
```

```
            }
          });
      });
  } catch (error) {
    alert(error.message);
    dispatch(loginLoader(false));
  }
};
```

Αρχικά, ενεργοποιείται ένας δείκτης φόρτωσης μέσω της loginLoader για να ενημερωθεί ο χρήστης ότι η είσοδος του επεξεργάζεται. Στη συνέχεια, ξεκινά η απόπειρα εισόδου του χρήστη μέσω της λειτουργίας signInWithEmailAndPassword του Firebase Authentication, χρησιμοποιώντας τα δεδομένα που παρέχονται (email και password). Με τη χρήση του Firestore, ανακτώνται πληροφορίες χρήστη για περαιτέρω επεξεργασία.

Αν ο ρόλος του χρήστη είναι "admin", τότε δημιουργείται ένα αντικείμενο tempUser που περιλαμβάνει τα δεδομένα του χρήστη και στη συνέχεια αποστέλλεται στο Redux store μέσω της δράσης LOGIN. Σε αντίθετη περίπτωση, εμφανίζεται μήνυμα λάθους. Αν υπάρξει σφάλμα κατά τη διαδικασία εισόδου, εμφανίζεται μήνυμα σφάλματος στον χρήστη και απενεργοποιείται ο δείκτης φόρτωσης.

6.2 Log out

Αρχικά, εκκαθαρίζονται τα τοπικά δεδομένα με την localStorage.clear() που μπορεί να έχει αποθηκευμένα η εφαρμογή, όπως πιθανότατα δεδομένα σύνδεσης ή προτιμήσεις του χρήστη. Στη συνέχεια, εκκινείται η διαδικασία αποσύνδεσης από την υπηρεσία πιστοποίησης της Firebase, όπου ο χρήστης έχει συνδεθεί. Αν η διαδικασία αποσύνδεσης είναι επιτυχής, αναγγέλλεται η αλλαγή κατάστασης στο Redux state της εφαρμογής και καλείται μια συνάρτηση επιτυχούς αποσύνδεσης, πιθανότατα για να προβληθεί μήνυμα επιτυχούς αποσύνδεσης στον χρήστη. Αν υπάρξει σφάλμα κατά τη διαδικασία αποσύνδεσης, εμφανίζεται ένα μήνυμα λάθους προς τον χρήστη. Τέλος, υπάρχει μια προσπάθεια περιγραφής του λόγου αποτυχίας σε περίπτωση που η διαδικασία αποσύνδεσης αποτύχει από γενικό σφάλμα.

```
export const doLogOut = (onSuccess) => async (dispatch) => {
  try {
    localStorage.clear();
    firebase
      .auth()
      .signOut()
      .then((data) => {
        dispatch({ type: "LOGOUT", uid: "" });
        onSuccess();
      })
      .catch((err) => {
        alert(err.message);
      });
```

```
  } catch (error) {
    alert(error.message);
  }
};
```

Η γραμμή localStorage.clear(); καθαρίζει τα τοπικά αποθηκευμένα δεδομένα του προγράμματος. Αυτό μπορεί να περιλαμβάνει δεδομένα που έχουν αποθηκευτεί προηγουμένως για τη σύνδεση του χρήστη, όπως tokens πρόσβασης ή άλλες προτιμήσεις.

Ο κώδικας που ακολουθεί εκτελεί τη διαδικασία αποσύνδεσης από την υπηρεσία πιστοποίησης της Firebase. Η firebase.auth().signOut() είναι η μέθοδος που εκτελεί τη διαδικασία αποσύνδεσης. Αυτό αποστέλλει μια αίτηση στην υπηρεσία πιστοποίησης για να αποσυνδεθεί ο τρέχων χρήστης.

Μετά την επιτυχή ή αποτυχημένη αποσύνδεση, υπάρχουν αντίστοιχες ενέργειες. Σε περίπτωση επιτυχίας (then), εκτελείται η συνάρτηση onSuccess(), η οποία μπορεί να είναι μια συνάρτηση που εμφανίζει ένα μήνυμα επιτυχούς αποσύνδεσης στον χρήστη και πιθανώς να προβαίνει σε άλλες ενέργειες. Σε περίπτωση αποτυχίας (catch), εμφανίζεται ένα μήνυμα λάθους που περιέχει το μήνυμα σφάλματος που επέστρεψε η Firebase.

6.3 ΠΡΟΣΘΗΚΗ ΚΑΤΗΓΟΡΙΩΝ

Υλοποιείται η λειτουργία προσθήκης νέας κατηγορίας στη βάση δεδομένων. Όταν ένας χρήστης προσθέτει μια νέα κατηγορία, ο κώδικας αυτός αποστέλλει ένα αίτημα προς τη βάση δεδομένων Firebase για να αποθηκεύσει τα στοιχεία της κατηγορίας. Επίσης, εμφανίζει ένα μήνυμα επιτυχούς προσθήκης κατηγορίας στον χρήστη.

```
export const addCategories =
  ({ categoryDetail, onSuccess }) =>
  async (dispatch) => {
    dispatch(addLoader(true));
    try {
      await firebase
        .firestore()
        .collection("category")
        .add({
          ...categoryDetail,
          createdAt: firebase.firestore.Timestamp.now(),
        })
        .then(() => {
          dispatch(addLoader(false));
          setTimeout(() => {
            alert("category added successfully!");
            onSuccess();
          }, 0);
        });
```

```
    } catch (error) {
      alert(error.message);
      dispatch(addLoader(false));
    }
  };
```

Αρχικά, ενεργοποιείται ο δείκτης φόρτωσης μέσω της συνάρτησης dispatch(addLoader(true)). Στη συνέχεια, ο κώδικας αποστέλλει ένα αίτημα προς τη βάση δεδομένων Firebase για να προσθέσει τα στοιχεία της κατηγορίας. Αν η προσθήκη είναι επιτυχής, ο δείκτης φόρτωσης απενεργοποιείται και εμφανίζεται ένα μήνυμα επιτυχούς προσθήκης κατηγορίας. Έπειτα, καλείται η συνάρτηση onSuccess(), η οποία πιθανότατα πραγματοποιεί περαιτέρω ενέργειες μετά την επιτυχή προσθήκη. Σε περίπτωση αποτυχίας, εμφανίζεται ένα μήνυμα λάθους και ο δείκτης φόρτωσης απενεργοποιείται.

6.4 ΛΗΨΗ ΚΑΤΗΓΟΡΙΩΝ

Χρησιμοποιείται για τη λήψη των κατηγοριών από τη βάση δεδομένων Firebase.

```
export const getCategories = () => async (dispatch) => {
  dispatch(getLoader(true));
  try {
    firebase
      .firestore()
      .collection("category")
      .onSnapshot(async (data) => {
        let tempData = [];
        for (let doc of data.docs) {
          let id = doc.id;
          let data1 = doc.data();
          tempData.push({ id: id, ...data1 });
        }
        dispatch({ type: "GET_ALL_CATEGORIES", payload: tempData });
        dispatch(getLoader(false));
      });
  } catch (error) {
    alert(error.message);
    dispatch(getLoader(false));
  }
};
```

Ενεργοποιείται ο δείκτης φόρτωσης μέσω της συνάρτησης dispatch(getLoader(true)). Η συνάρτηση onSnapshot καλείται για να παρακολουθεί αλλαγές στη συλλογή "category" της βάσης δεδομένων. Κάθε φορά που υπάρχει μια αλλαγή, ένας προσωρινός πίνακας tempData ενημερώνεται με τα νέα δεδομένα κατηγοριών. Τέλος, τα δεδομένα αποστέλλονται στο store μέσω της ενέργειας { type: "GET_ALL_CATEGORIES", payload: tempData }, και ο δείκτης φόρτωσης απενεργοποιείται. Σε περίπτωση λάθους, εμφανίζεται ένα μήνυμα λάθους και ο δείκτης φόρτωσης απενεργοποιείται.

6.5 ΔΙΑΓΡΑΦΗ ΚΑΤΗΓΟΡΙΑΣ

Ο κώδικας ενεργοποιεί ένα δείκτη φόρτωσης πριν την εκτέλεση της διαγραφής και τον απενεργοποιεί μετά την επιτυχή ολοκλήρωση της. Κάθε φορά που μια κατηγορία διαγράφεται με επιτυχία, εμφανίζεται ένα μήνυμα επιτυχίας και εκτελείται μια συγκεκριμένη λειτουργία, η οποία μπορεί να ορίζεται από τον χρήστη.

```
export const deleteCategory = (id, onSuccess) => async (dispatch) => {
  dispatch(addLoader(true));
  try {
    await firebase
      .firestore()
      .collection("category")
      .doc(id)
      .delete()
      .then(() => {
        dispatch(addLoader(false));
        setTimeout(() => {
          alert("Category deleted successfully!");
          onSuccess();
        }, 0);
      });
  } catch (error) {
    alert(error.message);
    dispatch(addLoader(false));
  }
};
```

Η συνάρτηση deleteCategory δέχεται ένα id, που αντιστοιχεί στην ταυτότητα της κατηγορίας που πρόκειται να διαγραφεί, και μια συνάρτηση onSuccess, η οποία καλείται όταν η διαγραφή είναι επιτυχής. Αφού ενεργοποιηθεί ο δείκτης φόρτωσης, η συνάρτηση αποστέλλει ένα αίτημα διαγραφής στη βάση δεδομένων για τη συγκεκριμένη κατηγορία, με βάση το παρεχόμενο id. Όταν η διαγραφή ολοκληρωθεί με επιτυχία, ο δείκτης φόρτωσης απενεργοποιείται και εμφανίζεται ένα μήνυμα επιτυχίας. Τέλος, καλείται η συνάρτηση onSuccess για την εκτέλεση πρόσθετων εργασιών, όπως η ενημέρωση της διεπαφής χρήστη. Σε περίπτωση αποτυχίας, εμφανίζεται ένα μήνυμα λάθους και ο δείκτης φόρτωσης απενεργοποιείται για να αντιμετωπιστεί το σφάλμα

6.6 ΔΗΜΙΟΥΡΓΙΑ ΝΕΟΥ ΠΡΟΙΟΝΤΟΣ

Ο κώδικας αυτός υλοποιεί μια λειτουργία δημιουργίας προϊόντος σε μια εφαρμογή. Αρχικά, ενεργοποιείται ένας δείκτης φόρτωσης για να ενημερώσει τον χρήστη ότι η διαδικασία ξεκίνησε. Στη συνέχεια, ο κώδικας αντιμετωπίζει τις εικόνες που σχετίζονται με το προϊόν. Για κάθε εικόνα,

πραγματοποιείται μια σειρά ενεργειών: πρώτα γίνεται η μετατροπή του ονόματος αρχείου για να προστεθεί ένας τυχαίος αριθμός, στη συνέχεια ανεβάζεται η εικόνα στην αποθήκη Firebase Storage, και τέλος ανακτάται ο σύνδεσμος λήψης της εικόνας. Όλοι οι σύνδεσμοι λήψης αποθηκεύονται σε έναν πίνακα.

Ύστερα, ολοκληρώνεται η δημιουργία του προϊόντος όπου οι πληροφορίες του προϊόντος, μαζί με τους συνδέσμους λήψης των εικόνων, αποθηκεύονται στη συλλογή "products" στη βάση δεδομένων Firestore. Όταν η διαδικασία ολοκληρωθεί επιτυχώς, εμφανίζεται ένα μήνυμα επιτυχίας στον χρήστη, ο δείκτης φόρτωσης απενεργοποιείται και η συνάρτηση onSuccess καλείται για ενδεχόμενες επιπρόσθετες ενέργειες.

```
import { v4 as uuidv4 } from "uuid";
export const createProduct = (payload, onSuccess) => async (dispatch) => {
  await dispatch(Loading(true));
  console.log(payload, "payload");
  try {
    const imageURLs = [];
    if (payload.images.length > 0) {
      for await (let item of payload.images) {
        if (item?.originFileObj) {
          let filName1 = item?.originFileObj.name;
          let filName2 = filName1.slice(filName1.lastIndexOf("."));
          let filName3 = uuidv4() + filName2.toLowerCase();
          let storageRef = await firebase
            .storage()
            .ref("Details/" + filName3)
            .put(item?.originFileObj);
          let url = await storageRef.ref.getDownloadURL();
          imageURLs.push(url);
        }
      }
    }
    await firebase
      .firestore()
      .collection("products")
      .add({
        ...payload,
        images: imageURLs,
        isDeleted: false,
        createdAt: firebase.firestore.Timestamp.now(),
        discount: 0,
      })
      .then(() => {
        alert("Product added successfully!");
        dispatch(Loading(false));
        onSuccess();
```

```
      });
  } catch (error) {
    console.log(error.message);
    dispatch(Loading(false));
  }
};
```

Η συνάρτηση createProduct αναλαμβάνει τη δημιουργία ενός προϊόντος στην εφαρμογή. Λαμβάνει ένα payload, το οποίο περιέχει όλες τις απαραίτητες πληροφορίες για το προϊόν, όπως το όνομα, την περιγραφή, την τιμή και τις εικόνες του. Επίσης, δέχεται μια συνάρτηση onSuccess που καλείται όταν η διαδικασία ολοκληρωθεί με επιτυχία. Κατά την εκτέλεση, το πρώτο βήμα είναι η ενεργοποίηση ενός δείκτη φόρτωσης, χρησιμοποιώντας τη συνάρτηση Loading(true). Έπειτα, για κάθε εικόνα που υπάρχει στο payload.images, πραγματοποιούνται οι απαραίτητες ενέργειες για την αποθήκευση και ανάκτηση των εικόνων στο Firebase Storage.

Στη συνέχεια, ολοκληρώνεται η δημιουργία του προϊόντος με την αποθήκευση των πληροφοριών του στη συλλογή "products" στη βάση δεδομένων Firestore. Όταν η διαδικασία ολοκληρωθεί επιτυχώς, εμφανίζεται ένα μήνυμα επιτυχίας, ο δείκτης φόρτωσης απενεργοποιείται και καλείται η συνάρτηση onSuccess. Σε περίπτωση σφάλματος, εμφανίζεται ένα μήνυμα λάθους και ο δείκτης φόρτωσης απενεργοποιείται.

6.7 Λήψη Προϊόντων από τη Βάση Δεδομένων

Η συνάρτηση getProduct χρησιμοποιείται για να ληφθούν όλα τα προϊόντα που δεν έχουν διαγραφεί από τη βάση δεδομένων. Ενεργοποιείται ο δείκτης φόρτωσης πριν από την αναζήτηση των προϊόντων και απενεργοποιείται μετά την ολοκλήρωσή της. Η συλλογή "products" της βάσης δεδομένων Firestore ελέγχεται με τη χρήση της μεθόδου where, ώστε να επιλεγούν μόνο τα προϊόντα που δεν έχουν διαγραφεί. Κάθε φορά που υπάρχει μια αλλαγή στη συλλογή "products", τα νέα δεδομένα επεξεργάζονται και αποθηκεύονται στο Redux store για χρήση στο UI της εφαρμογής.

```
export const getProduct = () => async (dispatch) => {
  dispatch(Loading(true));
  try {
    firebase
      .firestore()
      .collection("products")
      .where("isDeleted", "==", false)
      .onSnapshot(async (data) => {
        let tempData = [];
        for (let doc of data.docs) {
          let id = doc.id;
          let data1 = doc.data();
          tempData.push({ id: id, ...data1 });
        }
        dispatch({ type: "GET_ALL_PRODUCTS", payload: tempData });
```

```
        dispatch(Loading(false));
      });
  } catch (error) {
    alert(error.message);
    dispatch(Loading(false));
  }
};
```

Η παραπάνω συνάρτηση getProduct χρησιμοποιείται για να ληφθούν όλα τα προϊόντα που δεν έχουν διαγραφεί από τη βάση δεδομένων Firestore. Ενεργοποιείται ο δείκτης φόρτωσης πριν από την αναζήτηση των προϊόντων και απενεργοποιείται μετά την ολοκλήρωσή της. Η συλλογή "products" ελέγχεται χρησιμοποιώντας τη μέθοδο where, ώστε να επιλεγούν μόνο τα προϊόντα που δεν έχουν διαγραφεί. Κάθε φορά που υπάρχει μια αλλαγή στη συλλογή "products", τα νέα δεδομένα επεξεργάζονται και αποθηκεύονται στο Redux store για χρήση στο UI της εφαρμογής. Σε περίπτωση σφάλματος, εμφανίζεται ένα μήνυμα λάθους και ο δείκτης φόρτωσης απενεργοποιείται.

6.8 Επεξεργασία Προϊόντος

Αναλαμβάνει την επεξεργασία ενός υπάρχοντος προϊόντος στη βάση δεδομένων. Λαμβάνει ως είσοδο τα νέα δεδομένα του προϊόντος, το αναγνωριστικό του προϊόντος (ID) και μια συνάρτηση επιτυχίας που καλείται όταν η επεξεργασία ολοκληρωθεί με επιτυχία. Κατά τη διάρκεια της επεξεργασίας, ελέγχει εάν υπάρχουν νέες εικόνες για μεταφόρτωση στο Firebase Storage. Στη συνέχεια, ενημερώνει τα δεδομένα του προϊόντος στη συλλογή "products" της βάσης δεδομένων Firestore.

```
export const editProduct = (payload, id, onSuccess) => async (dispatch) => {
  await dispatch(Loading(true));
  try {
    const imageURLs = [];
    if (payload.images.length > 0) {
      for await (let item of payload.images) {
        if (item?.originFileObj) {
          let filName1 = item?.originFileObj.name;
          let filName2 = filName1.slice(filName1.lastIndexOf("."));
          let filName3 = uuidv4() + filName2.toLowerCase();
          let storageRef = await firebase
            .storage()
            .ref("Details/" + filName3)
            .put(item?.originFileObj);
          let url = await storageRef.ref.getDownloadURL();
          imageURLs.push(url);
        } else {
          imageURLs.push(item?.url);
        }
      }
```

```
    }
    await firebase
      .firestore()
      .collection("products")
      .doc(id)
      .set({
        ...payload,
        images: imageURLs,
        updatedAt: firebase.firestore.Timestamp.now(),
      })
      .then(() => {
        dispatch(Loading(false));
        alert("Product updated successfully!");
        onSuccess();
      });
  } catch (error) {
    console.log(error.message);
    dispatch(Loading(false));
  }
};
```

Η συνάρτηση editProduct δέχεται τρία ορίσματα: τα νέα δεδομένα του προϊόντος (payload), το αναγνωριστικό του προϊόντος (id), και μια συνάρτηση onSuccess που καλείται όταν η επεξεργασία ολοκληρωθεί με επιτυχία. Αρχικά, ενεργοποιείται ο δείκτης φόρτωσης. Στη συνέχεια, ελέγχει εάν υπάρχουν νέες εικόνες προς μεταφόρτωση. Αν ναι, κάθε εικόνα μεταφορτώνεται στο Firebase Storage και οι URL τους προστίθενται στον πίνακα imageURLs. Εφόσον ολοκληρωθεί η μεταφόρτωση των εικόνων, ενημερώνεται η εγγραφή του προϊόντος στη βάση δεδομένων Firestore με τα νέα δεδομένα και τα νέα URLs των εικόνων. Τέλος, απενεργοποιείται ο δείκτης φόρτωσης, εμφανίζεται ένα μήνυμα επιτυχίας και καλείται η συνάρτηση onSuccess. Σε περίπτωση σφάλματος, εμφανίζεται το σχετικό μήνυμα λάθους και ο δείκτης φόρτωσης απενεργοποιείται.

6.9 ΔΙΑΓΡΑΦΗ ΠΡΟΙΟΝΤΟΣ

Ο παραπάνω κώδικας αναλαμβάνει τη διαγραφή ενός προϊόντος από τη βάση δεδομένων. Δέχεται ως είσοδο το αναγνωριστικό (ID) του προϊόντος που πρόκειται να διαγραφεί και μια συνάρτηση onSuccess, η οποία καλείται όταν η διαγραφή ολοκληρωθεί επιτυχώς. Αρχικά, ενεργοποιείται ο δείκτης φόρτωσης. Στη συνέχεια, διαγράφεται η εγγραφή του συγκεκριμένου προϊόντος από τη συλλογή "products" της βάσης δεδομένων Firestore. Μετά την επιτυχή ολοκλήρωση της διαγραφής, απενεργοποιείται ο δείκτης φόρτωσης και εμφανίζεται ένα μήνυμα επιτυχίας. Τέλος, καλείται η συνάρτηση onSuccess για να ολοκληρωθούν επιπρόσθετες ενέργειες που είναι απαραίτητες μετά τη διαγραφή ενός προϊόντος.

```
export const deleteProduct = (id, onSuccess) => async (dispatch) => {
  dispatch(Loading(true));
  try {
```

```
    await firebase
      .firestore()
      .collection("products")
      .doc(id)
      .delete();
    dispatch(Loading(false));
    setTimeout(() => {
      alert("Product deleted successfully!");
      onSuccess();
    }, 0);
  } catch (error) {
    console.log(error.message);
    dispatch(Loading(false));
  }
};
```

Η συνάρτηση deleteProduct αποτελείται από μια ανώνυμη async arrow function, η οποία δέχεται δύο ορίσματα: το ID του προϊόντος που θα διαγραφεί και μια συνάρτηση onSuccess. Αρχικά, ενεργοποιείται ο δείκτης φόρτωσης με την κλήση της συνάρτησης Loading(true). Έπειτα, εκτελείται μια ασύγχρονη λειτουργία με τη χρήση του await, όπου διαγράφεται η εγγραφή του προϊόντος από τη συλλογή "products" στη βάση δεδομένων Firestore. Αφού ολοκληρωθεί η διαγραφή, απενεργοποιείται ο δείκτης φόρτωσης και εμφανίζεται ένα μήνυμα επιτυχίας με τη χρήση της συνάρτησης alert(). Τέλος, καλείται η συνάρτηση onSuccess για να ολοκληρωθούν επιπρόσθετες ενέργειες που απαιτούνται μετά τη διαγραφή ενός προϊόντος. Σε περίπτωση σφάλματος, εμφανίζεται το σχετικό μήνυμα λάθους στην κονσόλα και απενεργοποιείται ο δείκτης φόρτωσης.

6.10 ΕΠΕΞΕΡΓΑΣΙΑ ΕΚΠΤΩΣΗΣ

Ο κώδικας παραπάνω υλοποιεί τη λειτουργία επεξεργασίας της έκπτωσης ενός προϊόντος στη βάση δεδομένων. Δέχεται ένα αντικείμενο payload που περιλαμβάνει το ID του προϊόντος και τη νέα τιμή έκπτωσης, καθώς επίσης και μια συνάρτηση onSuccess, η οποία καλείται όταν η διαδικασία ολοκληρωθεί επιτυχώς. Ο δείκτης φόρτωσης ενεργοποιείται πριν την αρχή της διαδικασίας επεξεργασίας. Έπειτα, η συλλογή "products" στη βάση δεδομένων Firestore ενημερώνεται με τη νέα τιμή έκπτωσης του συγκεκριμένου προϊόντος. Όταν η ενημέρωση ολοκληρωθεί επιτυχώς, απενεργοποιείται ο δείκτης φόρτωσης, εμφανίζεται ένα μήνυμα επιτυχίας και καλείται η συνάρτηση onSuccess.

```
export const editDiscount = (payload, onSuccess) => async (dispatch) => {
  await dispatch(Loading(true));
  try {
    await firebase
      .firestore()
      .collection("products")
      .doc(payload.id)
```

```
      .update({
        discount: payload.discount,
      })
      .then(() => {
        dispatch(Loading(false));
        alert("Discount added successfully!");
        onSuccess();
      });
  } catch (error) {
    console.log(error.message);
    dispatch(Loading(false));
  }
};
```

Η συνάρτηση editDiscount αποτελείται από μια ανώνυμη async arrow function, η οποία δέχεται δύο ορίσματα: το αντικείμενο payload, το οποίο περιλαμβάνει το ID του προϊόντος και τη νέα τιμή έκπτωσης, και τη συνάρτηση onSuccess. Αρχικά, ενεργοποιείται ο δείκτης φόρτωσης με την κλήση της συνάρτησης Loading(true). Στη συνέχεια, εκτελείται μια ασύγχρονη λειτουργία, όπου ενημερώνεται η τιμή έκπτωσης του προϊόντος στη βάση δεδομένων Firestore μέσω της συλλογής "products". Μετά την επιτυχή ολοκλήρωση της ενημέρωσης, απενεργοποιείται ο δείκτης φόρτωσης και εμφανίζεται ένα μήνυμα επιτυχίας με τη χρήση της συνάρτησης alert(). Τέλος, καλείται η συνάρτηση onSuccess για να ολοκληρωθούν επιπρόσθετες ενέργειες που απαιτούνται μετά την επεξεργασία της έκπτωσης. Σε περίπτωση σφάλματος, εμφανίζεται το σχετικό μήνυμα λάθους στην κονσόλα και απενεργοποιείται ο δείκτης φόρτωσης.

6.11 ΛΗΨΗ ΠΑΡΑΓΓΕΛΙΩΝ

Ο κώδικας αναφέρεται σε μια λειτουργικότητα που σχετίζεται με τη διαχείριση παραγγελιών σε μια εφαρμογή. Η συνάρτηση getOrders είναι υπεύθυνη για τη λήψη των παραγγελιών από μια βάση δεδομένων. Αρχικά, εμφανίζεται ένα μήνυμα φόρτωσης για να ενημερώσει τον χρήστη ότι η διαδικασία ξεκίνησε. Στη συνέχεια, ο κώδικας ανακτά τις παραγγελίες από τη βάση δεδομένων Firebase. Για κάθε παραγγελία, ανακτούνται οι λεπτομέρειες σχετικά με τα προϊόντα που περιέχει και τον χρήστη που την έχει πραγματοποιήσει.

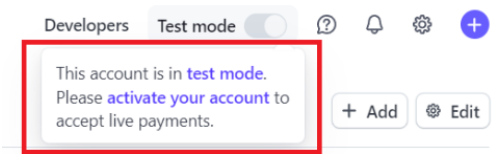
```
import firebase from "../firebase";
export const getOrders = () => async (dispatch) => {
  dispatch(Loading(true));
  try {
    const ordersSnapshot = await firebase
      .firestore()
      .collection("orders")
      .get();
    const orders = [];
    for (const orderDoc of ordersSnapshot.docs) {
```

```
      const orderData = orderDoc.data();
      const productDoc = await firebase
        .firestore()
        .collection("products")
        .doc(orderData.productIds[0])
        .get();
      const productData = productDoc.data();
      const userDoc = await firebase
        .firestore()
        .collection("users")
        .doc(orderData.userId)
        .get();
      const userData = userDoc.data();
      const completeOrder = {
        ...orderData,
        id: orderData?.paymentId,
        productData: productData,
        user: userData,
      };
      orders.push(completeOrder);
    }
    console.log(orders, "orders");
    dispatch({ type: "GET_ORDERS", payload: orders });
    dispatch(Loading(false));
  } catch (error) {
    alert(error.message);
    dispatch(Loading(false));
  }
};
export const Loading = (val) => async (dispatch) => {
  dispatch({ type: "LOADING", payload: val });
};
```

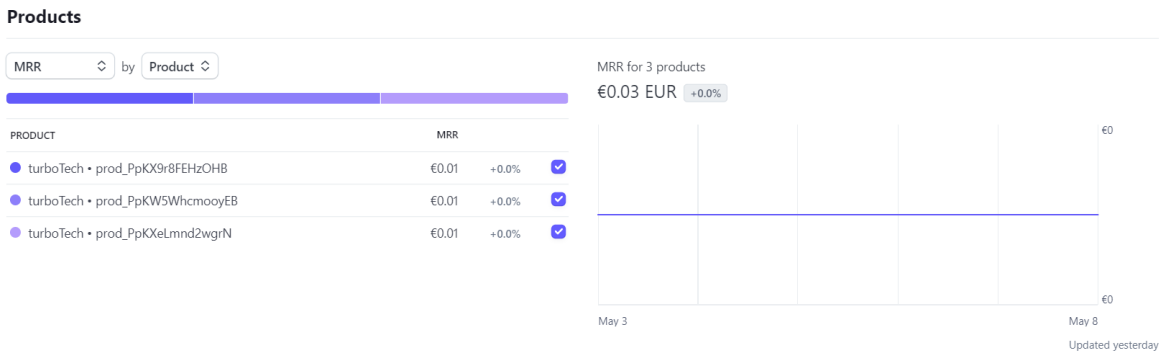
Η συνάρτηση getOrders εκτελεί τις ακόλουθες ενέργειες: Αρχικά, ενεργοποιεί τον δείκτη φόρτωσης. Στη συνέχεια, ανακτά όλες τις παραγγελίες από τη συλλογή "orders" της βάσης δεδομένων. Για κάθε παραγγελία, ανακτώνται τα δεδομένα του προϊόντος και του χρήστη από τις συλλογές "products" και "users" αντίστοιχα. Οι πληροφορίες αυτές συγκεντρώνονται σε ένα αντικείμενο completeOrder το οποίο περιέχει τις πλήρεις πληροφορίες για την παραγγελία, συμπεριλαμβανομένων του προϊόντος και του χρήστη. Τέλος, οι πλήρεις παραγγελίες αποθηκεύονται σε μια λίστα orders, η οποία αποστέλλεται στο store για χρήση από άλλα σημεία της εφαρμογής. Σε περίπτωση λάθους, εμφανίζεται ένα μήνυμα λάθους και ο δείκτης φόρτωσης απενεργοποιείται.

ΚΕΦΑΛΑΙΟ 7: STRIPE SYSTEM

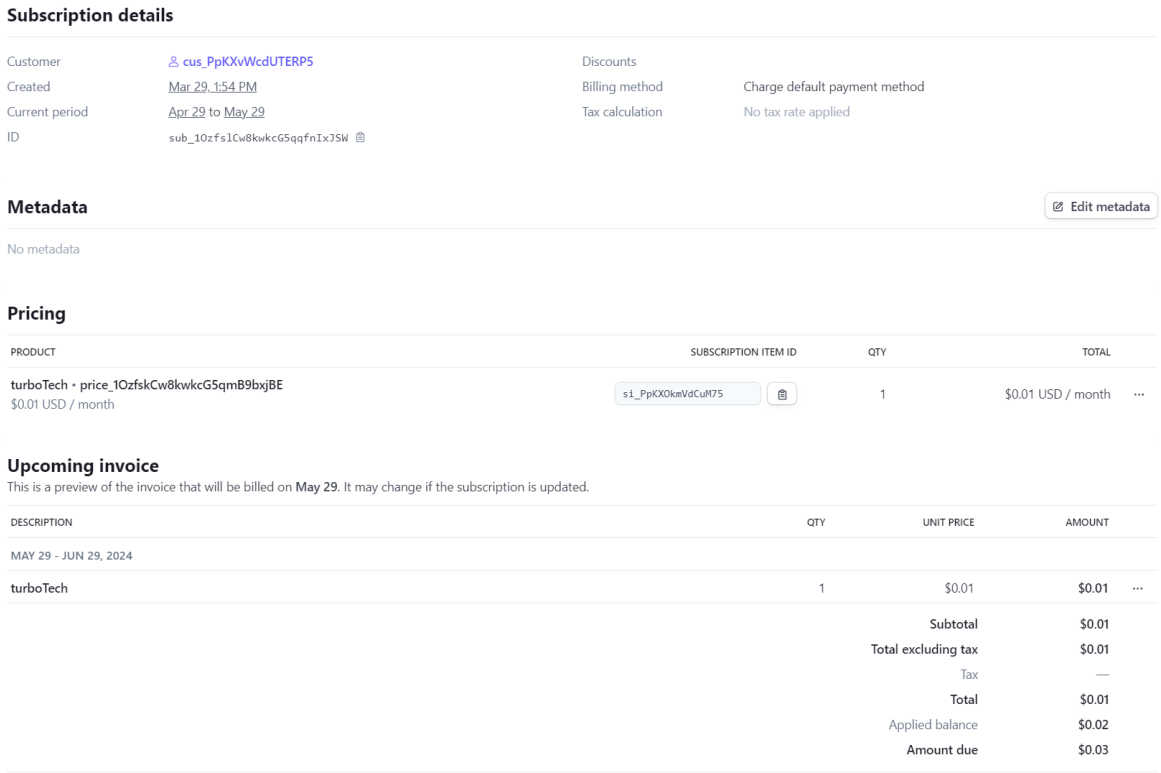
7.1 ΔΗΜΙΟΥΡΓΙΑ ΣΥΝΔΡΟΜΗΣ ΣΤΟ STRIPE



Εικόνα 7.1



Εικόνα 7.2



Subscription details

Customer	cus_PpKXvWcdUTERP5	Discounts	
Created	Mar 29, 1:54 PM	Billing method	Charge default payment method
Current period	Apr 29 to May 29	Tax calculation	No tax rate applied
ID	sub_1OzfslCw8kwkcG5qqfnIxJSW		

Metadata — Edit metadata

No metadata

Pricing

PRODUCT	SUBSCRIPTION ITEM ID	QTY	TOTAL
turboTech • price_1OzfskCw8kwkcG5qmB9bxjBE $0.01 USD / month	si_PpKXOkmVdCuM75	1	$0.01 USD / month

Upcoming invoice

This is a preview of the invoice that will be billed on **May 29**. It may change if the subscription is updated.

DESCRIPTION	QTY	UNIT PRICE	AMOUNT
MAY 29 - JUN 29, 2024			
turboTech	1	$0.01	$0.01
		Subtotal	$0.01
		Total excluding tax	$0.01
		Tax	—
		Total	$0.01
		Applied balance	$0.02
		Amount due	$0.03

Ο παραπάνω κώδικας αντιπροσωπεύει ένα endpoint σε έναν HTTP server, το οποίο λαμβάνει αιτήσεις POST για τη δημιουργία συνδρομών. Όταν λαμβάνει μια αίτηση, εξάγει τα δεδομένα που χρειάζονται για τη δημιουργία της συνδρομής από το σώμα της αίτησης (amountInCents, email, token) και χρησιμοποιεί το Stripe API για να δημιουργήσει ένα προϊόν, έναν τιμοκατάλογο και έναν πελάτη στο Stripe. Στη συνέχεια, χρησιμοποιεί τα στοιχεία αυτά για να δημιουργήσει μια συνδρομή για τον πελάτη.

```
app.post("/create-subscription", async (req, res) => {
    try {
        const { amountInCents, email, token } = req.body;

        // Create a product
        const product = await stripe.products.create({
            name: "turboTech",
            description: "turboTech provides the facility to chat with a bot",
        });

        // Create a price for the product
        const price = await stripe.prices.create({
            unit_amount: amountInCents,
            currency: "usd",
            recurring: {
                interval: "month", // Change as needed
            },
            product: product.id,
        });

        // Create a customer
        const customer = await stripe.customers.create({
```

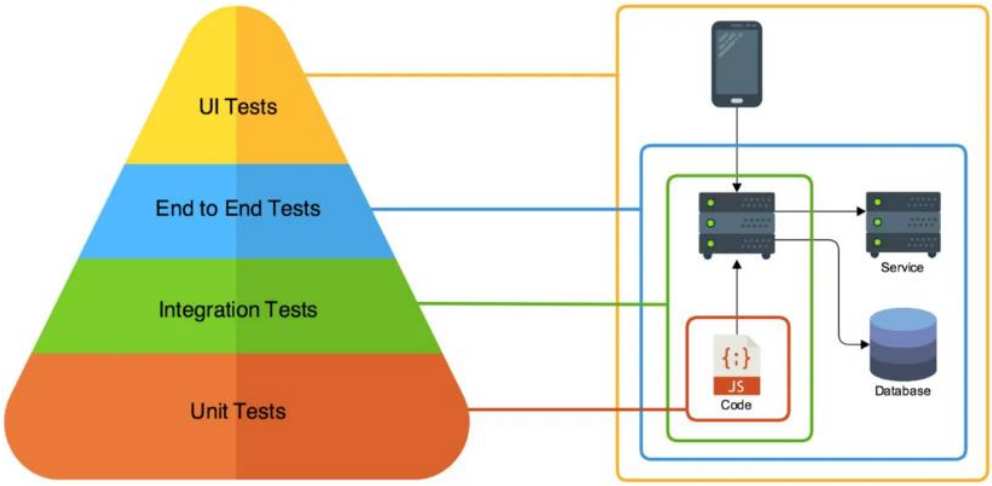
```
            email: email,
            source: token, // This token should come from your client-side Stripe.js
integration
        });
        console.log(customer, "customer");
        console.log(price.id, "priceid");
        // Create a subscription using the customer and price
        const subscription = await stripe.subscriptions.create({
            customer: customer.id,
            items: [{ price: price.id }],
        });
        console.log(subscription, "subscription");
        // Send a success response
        res.json({
            success: true,
            message: "Your subscription successfully done!",
            subscription,
        });
    } catch (error) {
        // Send a failure response
        res.status(500).json({
            success: false,
            message: "Your subscription failed!",
            error: error.message,
        });
    }
});
exports.app = functions.https.onRequest(app);
```

Η συνάρτηση app.post("/create-subscription", async (req, res) => {...}) αποτελεί το σημείο εισόδου για αιτήσεις POST που αφορούν τη δημιουργία συνδρομών. Αρχικά, λαμβάνονται τα απαραίτητα δεδομένα από το σώμα της αίτησης χρησιμοποιώντας το req.body. Στη συνέχεια, χρησιμοποιείται το Stripe API για να δημιουργηθεί ένα προϊόν με το όνομα "turboTech" και μια περιγραφή που παρέχει τη δυνατότητα συνομιλίας με ένα bot. Έπειτα, δημιουργείται ένας τιμοκατάλογος για το προϊόν με τον καθορισμό της τιμής και της νομισματικής μονάδας, καθώς και της επαναλαμβανόμενης περιόδου της συνδρομής. Στη συνέχεια, δημιουργείται ένας πελάτης στο Stripe με τον καθορισμό του email και του πηγαίου κωδικού (token). Αφού δημιουργηθεί ο πελάτης, χρησιμοποιείται η πληροφορία του για να δημιουργηθεί μια συνδρομή με τη χρήση του τιμοκαταλόγου που προηγουμένως δημιουργήθηκε. Τέλος, στέλνεται ένα JSON αντικείμενο ως απάντηση, ενημερώνοντας τον χρήστη για το αποτέλεσμα της διαδικασίας. Σε περίπτωση σφάλματος, επιστρέφεται ένα μήνυμα σφάλματος μαζί με το σχετικό μήνυμα λάθους και ο κατάλληλος κωδικός κατάστασης HTTP.

ΚΕΦΑΛΑΙΟ 8: UNIT/UI TESTING

8.1 Unit Testing

8.1.1 Τί είναι το Unit Testing



Εικόνα 8.1

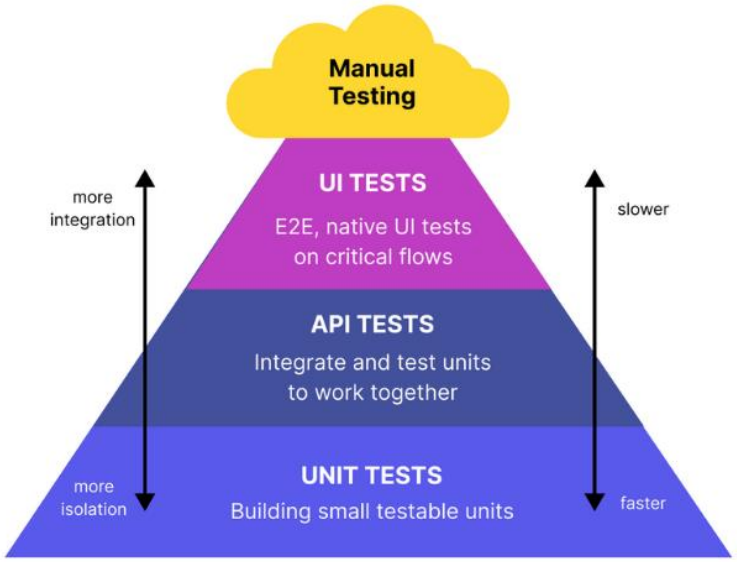
Το Unit Testing ορίζεται ως μια τεχνική διασφάλισης ποιότητας όπου ο κώδικας της εφαρμογής αναλύεται σε δομικά στοιχεία - μαζί με τα δεδομένα, τις διαδικασίες χρήσης και τις λειτουργίες που σχετίζονται με κάθε στοιχείο ή μονάδα - για να διασφαλιστεί ότι κάθε στοιχείο λειτουργεί όπως αναμένεται. Πριν αναπτυχθεί και κυκλοφορήσει οποιοδήποτε λογισμικό, πρέπει να υποβληθεί σε μια σειρά δοκιμών για να διασφαλιστεί η ακρίβεια και η λειτουργικότητα του. Οι δοκιμές λογισμικού ξεκινούν ακόμη και πριν από την ολοκλήρωση της εφαρμογής. Με αυτόν τον τρόπο, τα λάθη και τα σφάλματα εντοπίζονται έγκαιρα πριν χαθούν στους διάφορους κώδικες.

Γενικά, κάθε λογισμικό περνάει από τέσσερα στάδια δοκιμών. Το πρώτο είναι οι έλεγχοι κομματιών κώδικα, ακολουθούμενοι από τη ελέγχους ολοκλήρωσης, τον έλεγχο συστήματος και, τέλος, τον έλεγχο αποδοχής. Οι έλεγχοι αυτοί αποτελούν το θεμέλιο πάνω στο οποίο χτίζονται όλες οι άλλες δοκιμές. Ως εκ τούτου, η ακρίβεια και η διεξοδικότητα αυτών είναι σημαντικοί παράγοντες που επηρεάζουν το πόσο καλά μπορούν να διεξαχθούν οι άλλες δοκιμές και την απόδοση του λογισμικού στο σύνολο του.

Οι διαδικασίες δοκιμής είναι ουσιώδεις για τη διασφάλιση της σταθερότητας και της αξιοπιστίας του λογισμικού καθώς και για τη μείωση του κινδύνου ανεπιθύμητων συμπεριφορών κατά την παραγωγή. Η σωστή εφαρμογή τεχνικών όπως το Unit Testing επιτρέπει στις ομάδες ανάπτυξης να είναι πιο αυτόνομες και εμπιστευτικές στις αλλαγές που επιφέρουν στον κώδικα, γνωρίζοντας ότι τα θεμέλια της λειτουργικότητας παραμένουν ακέραια. Το Unit Testing είναι μια τεχνική δοκιμής λογισμικού όπου ο κώδικας ελέγχεται σε μικρές μονάδες, γνωστές ως "μονάδες". Μια μονάδα μπορεί να είναι ένας συγκεκριμένος συνάρτηση, μια μέθοδος ή ένα τμήμα κώδικα που εκτελεί ένα συγκεκριμένο έργο. Κατά τη διάρκεια του Unit Testing, κάθε μονάδα ελέγχεται ξεχωριστά για να βεβαιωθεί ότι λειτουργεί σωστά.

Τα Unit Tests γράφονται από τους προγραμματιστές και εκτελούνται αυτοματοποιημένα κατά τη διάρκεια της ανάπτυξης του λογισμικού. Τα tests αυτά ελέγχουν τη σωστή λειτουργία των μονάδων κώδικα, συγκρίνοντας τα αποτελέσματα που παράγονται από τις μονάδες με τα αναμενόμενα αποτελέσματα.

8.1.2 Κατηγορίες unit testing



Εικόνα 8.2

- Χειροκίνητη δοκιμή μονάδας

Η μη αυτόματη δοκιμή είναι δοκιμή μονάδας που εκτελείται χωρίς ειδικές εφαρμογές ή προγράμματα. Κάθε βήμα της διαδικασίας δοκιμής πραγματοποιείται από προγραμματιστές. Η χειροκίνητη δοκιμή μονάδας δεν εμφανίζεται συχνά λόγω καλύτερων εναλλακτικών λύσεων και πολλαπλών μειονεκτημάτων. Οι χειροκίνητες δοκιμές μονάδας είναι εντάσεως κόστους, καθώς οι εργαζόμενοι πρέπει να πληρώνονται για το χρόνο που αφιερώνουν κατά τη διάρκεια της διαδικασίας, ειδικά όταν αφορά μη μόνιμο προσωπικό. Η διαδικασία είναι χρονοβόρα, καθώς θα πρέπει να εκτελούνται δοκιμές κάθε φορά που αλλάζει ο κώδικας.

Μπορεί επίσης να είναι δύσκολη η απομόνωση και η δοκιμή ανεξάρτητων μονάδων, καθιστώντας πολύ δύσκολη τη διάκριση της πηγής των σφαλμάτων που προκύπτουν κατά τη διαδικασία δοκιμής. Ο προγραμματιστής εκτελεί συχνά χειροκίνητες δοκιμές και αξιολογεί τη σταθερότητα του λογισμικού μετά την προσθήκη ή την αφαίρεση γραμμών κώδικα.

- Αυτοματοποιημένη δοκιμή μονάδας

Η αυτοματοποιημένη δοκιμή μονάδας είναι ένας τύπος δοκιμής που γίνεται χωρίς ουσιαστική ανθρώπινη συμμετοχή. Τα εργαλεία που χρησιμοποιούνται για την εκτέλεση αυτοματοποιημένων δοκιμών παράγονται τελικά από ανθρώπους. Ωστόσο, μετά τη δημιουργία αυτών των εργαλείων, η εκτέλεση γίνεται αυτόματα. Οι αυτοματοποιημένες δοκιμές πραγματοποιούνται ως μέρος της διαδικασίας κατασκευής του λογισμικού. Όταν χρησιμοποιείτε το αυτοματοποιημένο σύστημα δοκιμής μονάδας, ο προγραμματιστής γράφει μια ενότητα κώδικα στο λογισμικό για να δοκιμάσει τη λειτουργία. Οι προγραμματιστές θα αφαιρέσουν τελικά αυτόν τον κώδικα από ολόκληρο το πρόγραμμα λίγο πριν από την ανάπτυξη.

Επιπλέον, η αυτόματη δοκιμή μονάδας μπορεί να απομονώσει κομμάτια κώδικα για να τους δοκιμάσει ανεξάρτητα. Αυτό αποκαλύπτει αλληλοεξαρτήσεις μεταξύ των μονάδων κώδικα. Οι αυτόματες δοκιμές γίνονται σε μεγάλο βαθμό χρησιμοποιώντας ένα εργαλείο και βιβλιοθήκη ελέγχου. Υπάρχουν πολλά εργαλεία ελέγχου μονάδων διαθέσιμα ως πόροι ανοιχτού κώδικα. Ο προγραμματιστής απλώς κωδικοποιεί τα κριτήρια και συνεχίζει. Το εργαλείο ειδοποιεί επίσης τον ελεγκτή για τα κομμάτια που απέτυχαν στη διαδικασία αυτή. Οι αυτοματοποιημένες δοκιμές μονάδων μπορούν να υπάρχουν ως γραμμές κώδικα ενσωματωμένες στο λογισμικό και θα πρέπει να ελέγχουν ένα μικρό μέρος.

8.1.3 Unit Testing and Quality Assurance Δοκιμές Μονάδων και Διασφάλιση Ποιότητας

Εικόνα 8.3

Τα Unit Testing είναι πολύτιμο όταν χρησιμοποιούνται για αυτοματοποιημένες δοκιμές λογισμικού ως μέρος μιας διαδικασίας διασφάλισης ποιότητας (QA). Σε πολλές ομάδες ανάπτυξης λογισμικού, η διαδικασία QA ξεκινά όταν υποβάλλεται νέος κώδικας, δημιουργείται και ελέγχεται η. Συχνά, περιλαμβάνουν όχι μόνο ελέγχους προγραμματιστή, αλλά και τεστ αποδοχής που σχεδιάστηκαν ή γράφτηκαν από την ομάδα QA. Εάν επιτύχουν όλοι οι έλεγοι της ποιότητας, ο κωδικός γίνεται προσωρινά αποδεκτός και αποστέλλεται σε μηχανικό QA για επιθεώρηση και δοκιμή. Η εκτέλεση της πλήρους σειράς ως το πρώτο βήμα στο QA έχει πολλά οφέλη. Το πιο σημαντικό, οι δοκιμές διασφαλίζουν ότι ο κώδικας είναι σταθερός τη στιγμή που θα φύγει από τα χέρια των προγραμματιστών. Δεν απαιτείται ανθρώπινη παρέμβαση για την εκτέλεση των δοκιμών και την αξιολόγηση των αποτελεσμάτων. Είτε όλοι πετυχαίνουν, είτε υπάρχει αποτυχία. Τέτοια αποτελέσματα Boolean (αληθές/λάθος) είναι ιδανικά επειδή ένα αυτοματοποιημένο σύστημα μπορεί να τα κατανοήσει. Η επιτυχία επιβεβαιώνει ότι οι υποθέσεις των προγραμματιστών είναι έγκυρες και ότι η λειτουργικότητα χαμηλού επιπέδου λειτουργεί σωστά σε ένα επίπεδο ελέγχου που δεν μπορούν ποτέ να επιτύχουν οι λειτουργικές δοκιμές. Όταν πολλοί προγραμματιστές κάνουν αλλαγές ταυτόχρονα, οι έλεγχοι αυτοί παρέχουν βεβαιότητα ότι οι αλλαγές κανενός δεν προκάλεσαν τη διακοπή του κώδικα κάποιου άλλου.

Επίσης βοηθούν στην παροχή λογοδοσίας. Γνωρίζοντας ακριβώς ποιο τεστ αποτυγχάνει συνήθως καθιστά εμφανές ποιος η αλλαγή έσπασε τα πράγματα. Το "σπάσιμο" σήμαινε κάποτε την υποβολή κώδικα που προκάλεσε την αποτυχία μιας μεταγλώττισης, αλλά τώρα συχνά αναφέρεται στην πρόκληση αποτυχίας ενός ελέγχου. Η αποτυχία μιας δοκιμής μονάδας δίνει σαφώς υψηλή προτεραιότητα στην επίλυση του προβλήματος. Εάν το TDD (οι προγραμματιστές δοκιμάζουν πρώτα και μετά χρησιμοποιούν τα αποτελέσματα των δοκιμών για να καθοδηγήσουν την ανάπτυξή τους) ακολουθείται αυστηρά, ο κώδικας δεν πρέπει ποτέ να αφήνεται σε κατάσταση στην οποία αποτυγχάνει. Είναι απολύτως δυνατό να αναπτυχθεί πλήρως δοκιμασμένος σε μονάδες, εντελώς "αλεξίσφαιρος " κώδικας που δεν έχει χρηστικότητα περιλαμβάνουν την "εμφάνιση και αίσθηση" του GUI (γραφικό περιβάλλον διεπαφής χρήστη) , απαντήσεις σε συμβάντα του συστήματος, αλληλεπίδραση με κατανεμημένα στοιχεία εφαρμογής και πολλές άλλες δυνατότητες. Μερικές φορές οι έλεγχοι μπορούν να γραφτούν για την προσομοίωση αυτών των τύπων καταστάσεων, αλλά τελικά, δεν υπάρχει υποκατάστατο της πραγματικότητας ή της αντικειμενικής ανατροφοδότησης ενός χρήστη.

Παρόλο που η χειροκίνητη δοκιμή QA εξακολουθεί να είναι σημαντική, είναι ένα ισχυρό εργαλείο για τη διασφάλιση της ποιότητας. Οι προγραμματιστές που χρησιμοποιούν ανάπτυξη με επίκεντρο τις δοκιμές αναφέρουν δραματικές βελτιώσεις στην ποιότητα του λογισμικού, την ταχύτητα ανάπτυξης και την ικανότητα να κάνουν σημαντικές αλλαγές σχεδιασμού εν κινήσει.

8.1.4 Ποια είναι τα οφέλη του unit testing;

- Μειώνει το κόστος μετά την παραγωγή

Ένα από τα πιο σημαντικά πλεονεκτήματα τείναι η ικανότητά του να μειώνει το κόστος μετά την παραγωγή. Συμπεριλαμβάνοντας τους ελέγχους διαφόρων κομματιών κώδολα στη διαδικασία ανάπτυξης, μπορεί κάποιος να εντοπίσει και να διορθώσει προληπτικά σφάλματα και ζητήματα πριν η εφαρμογή φτάσει στη φάση δοκιμών και διασφάλισης ποιότητας (QA). Αυτή η έγκαιρη ανίχνευση

ελαττωμάτων ελαχιστοποιεί τον χρόνο και τους πόρους που απαιτούνται για τον εντοπισμό σφαλμάτων, με αποτέλεσμα μια πιο ομαλή διαδικασία ανάπτυξης και μικρότερα χρονοδιαγράμματα του έργου.

Επιπλέον, μιας εφαρμογή με λιγότερα σφάλματα και δυσλειτουργίες σημαίνει ότι οι μηχανικοί και οι προγραμματιστές QA έχουν λιγότερη τεκμηρίωση για προετοιμασία και λιγότερες αλλαγές να κάνουν. Αυτός ο φόρτος εργασίας μετά την παραγωγή επιταχύνει το έργο και μειώνει το κόστος του έργου, επιτρέποντας στους διαχειριστές του έργου να προβλέπουν και να προϋπολογιστούν με μεγαλύτερη ακρίβεια και να ξεκινήσουν νέα έργα νωρίτερα.

- Σταθεροποίηση του κώδικα

Η σταθεροποίηση του κώδικα είναι ένα σημαντικό βήμα στη διασφάλιση της αξιοπιστίας και της συνεχόμενης λειτουργίας της εφαρμογής. Στην περίπτωση όπου προσθέτουμε ένα νέο χαρακτηριστικό όπως η δυνατότητα φιλτραρίσματος στα αποτελέσματα αναζήτησης, είναι σημαντικό να διατηρήσουμε την ομαλή λειτουργία του υπάρχοντος κώδικα και να αποφύγουμε την επιπλέον επηρεασμό του. Για παράδειγμα, έχοντας ήδη υλοποιήσει μια λειτουργία καθολικής αναζήτησης ανά όνομα, η οποία έχει δοκιμαστεί και εφαρμοστεί, τώρα θέλουμε να προσθέσουμε τη δυνατότητα φιλτραρίσματος στα αποτελέσματα αναζήτησης. Για να το πετύχουμε αυτό, θα δημιουργήσουμε μια νέα μονάδα για το φιλτράρισμα, η οποία θα πρέπει να ενσωματωθεί ομαλά στον υπάρχοντα κώδικα χωρίς να δημιουργήσει προβλήματα ή σφάλματα.

Είναι σημαντικό να διασφαλίσουμε ότι η νέα μονάδα δεν θα επηρεάσει αρνητικά τη λειτουργία της υπάρχουσας λειτουργίας αναζήτησης. Αυτό μπορεί να επιτευχθεί μέσω των ελέγχων που ελέγχουν τη νέα λειτουργία φιλτραρίσματος και βεβαιώνονται ότι η ενσωμάτωσή της δεν θέτει σε κίνδυνο την υπάρχουσα λειτουργικότητα. Έτσι, με την εφαρμογή των κατάλληλων δοκιμών και την προσεκτική υλοποίηση του νέου χαρακτηριστικού φιλτραρίσματος, μπορούμε να επιτύχουμε τη σταθεροποίηση του κώδικα και να διασφαλίσουμε τη συνεχή και αξιόπιστη λειτουργία της εφαρμογής.

- Βελτιωμένος εντοπισμός σφαλμάτων

Ο εντοπισμός και η διόρθωση της βασικής αιτίας ενός σφάλματος είναι ένα ακόμα σημαντικό πλεονέκτημα του ελέγχου κώδικα σε μονάδες. Αντί να αναζητεί κάποιος τη βασική αιτία ενός σφάλματος σε έναν μεγάλο όγκο κώδικα, ο έλεγχος σε μονάδες επιτρέπει τον εντοπισμό και τη διόρθωση των σφαλμάτων με μεγαλύτερη ακρίβεια και αποτελεσματικότητα.Ας υποθέσουμε ότι η δυνατότητα αναζήτησης δεν λειτουργεί σωστά. Αντί να αναζητήσει κάποιος το σφάλμα σε ολόκληρο τον κώδικα, μπορεί να επανεξετάσει τα αποτελέσματα των δοκιμών μονάδας που περιλαμβάνουν τη λειτουργία αναζήτησης. Μέσω αυτής της προσέγγισης, είναι δυνατόν να εντοπιστεί πιο γρήγορα η πηγή του σφάλματος και να ληφθούν τα αναγκαία μέτρα για τη διόρθωσή του.

Με την εξέταση των αποτελεσμάτων των δοκιμών μονάδας στη συγκεκριμένη λειτουργία της αναζήτησης, οι προγραμματιστές μπορούν να κατανοήσουν τις προκλήσεις και τα σημεία αδυναμίας του κώδικα με μεγαλύτερη σαφήνεια. Αυτό επιτρέπει την ταχύτερη αντιμετώπιση των προβλημάτων και

την εφαρμογή αποτελεσματικών λύσεων, προσφέροντας έτσι μια αποτελεσματική διαδικασία εντοπισμού και διόρθωσης σφαλμάτων.

- Αυξάνει το δυναμικό εσόδων

Η ποιότητα της εφαρμογής επηρεάζει άμεσα την ικανοποίηση των χρηστών και τις δυνατότητες εσόδων. Διασφαλίζετε λοιπόν, ότι η εφαρμογή είναι υψηλής ποιότητας, χωρίς κρίσιμα σφάλματα και προσφέρει μια ομαλή εμπειρία χρήστη. Εάν οι χρήστες προτείνουν την εφαρμογή αφήνοντας θετικές κριτικές στο διαδίκτυο ή μοιράζοντας το με φίλους και συγγενείς, οι δυνατότητες κερδών της εφαρμογής αυξάνονται εκθετικά. Οι ικανοποιημένοι χρήστες είναι πιο πιθανό να προτείνουν την εφαρμογή σε άλλους μέσω θετικών κριτικών στο διαδίκτυο ή μέσω παραπομπών από στόμα σε στόμα. Και ας μην παραβλέπουμε το γεγονός ότι η έλλειψη αρνητικών κριτικών μπορεί επίσης να οδηγήσει σε οικονομική επιτυχία. Ακριβώς όπως υπάρχουν καταναλωτές που αναζητούν ευνοϊκές πληροφορίες, υπάρχουν και εκείνοι που αναζητούν ειδικά κριτικές αξιολογήσεις ως τρόπο αποφυγής ενός ελαττωματικού προϊόντος ή υπηρεσίας. Μπορεί να υποστηριχθεί τελικά ότι, από επιχειρηματική σκοπιά, η συγκέντρωση θετικών κριτικών είναι ωφέλιμη.

- Τεκμηρίωση σε ομαδικά Projects

Σε ομαδικά Projects, συχνά προκύπτουν κενά γνώσης λόγω της κατάτμησης εργασιών και των απομονωμένων ομάδων. Η δοκιμή μονάδων είναι μια αποτελεσματική μορφή τεκμηρίωσης που παρέχει πολύτιμες πληροφορίες σχετικά με τη λειτουργικότητα του κώδικα.

Όταν συμμετέχουν νέοι προγραμματιστές ή όταν τα μέλη της ομάδας αλλάζουν ρόλους, ο έλεγχος των δοκιμών μονάδας τους βοηθά να κατανοήσουν γρήγορα την αρχιτεκτονική της εφαρμογής και τον τρόπο αλληλεπίδρασης των στοιχείων. Αυτές οι δοκιμές χρησιμεύουν ως ένα εκτελέσιμο έγγραφο που παρέχει σημαντικές πληροφορίες που μπορεί να λείπουν από τις γραπτές σημειώσεις. Αυτό προωθεί την καλύτερη συνεργασία μεταξύ των μελών της ομάδας και εξασφαλίζει μια πιο ομαλή διαδικασία παράδοσης.

- Αρχιτεκτονική ευελιξία

Με καλά σχεδιασμένους ελέγχους, οι κώδικες μπορούν να σχεδιαστούν ώστε να επιτρέπουν την προσθήκη ή την αλλαγή λειτουργικότητας χωρίς να διακόπτεται η υπάρχουσα λειτουργικότητα. Ο σαφής διαχωρισμός διασφαλίζει ότι κάθε μονάδα κώδικα υποστηρίζει έναν μόνο σκοπό, καθιστώντας εύκολη την προσθήκη ή την αλλαγή λειτουργιών σε εύλογο χρονικό διάστημα.

- Αποτελεσματική ανακατασκευή

Βοηθάει να επιβεβαιώσουμε ότι λειτουργεί όπως αναμενόταν—ακόμα και αν αναπαράγουμε τη λογική κώδικα ή ενημερώσουμε βιβλιοθήκες τρίτων, για παράδειγμα. Ας υποθέσουμε ότι η λειτουργία λειτουργεί τέλεια, αλλά είναι πάρα πολύ αργή. Για να επιλύσουμε το πρόβλημα ταχύτητας,

εφαρμόζουμε μια πιθανή επιδιόρθωση (π.χ. αντικαθιστούμε τον αλγόριθμο) και δοκιμάζουμε ξανά. Για άλλη μια φορά, μπορούμε να είμαστε σίγουροι ότι δεν έχουμε «σπάσει» το χαρακτηριστικό που έχει δοκιμαστεί στο παρελθόν τέλεια. Το χαρακτηριστικό θα πρέπει ακόμα να περάσει τους ελέγχους ξανά μετά τον ανασχηματισμό ώστε να επιβεβαιωθεί η λειτουργία του.

- Ταχύτερη παράδοση

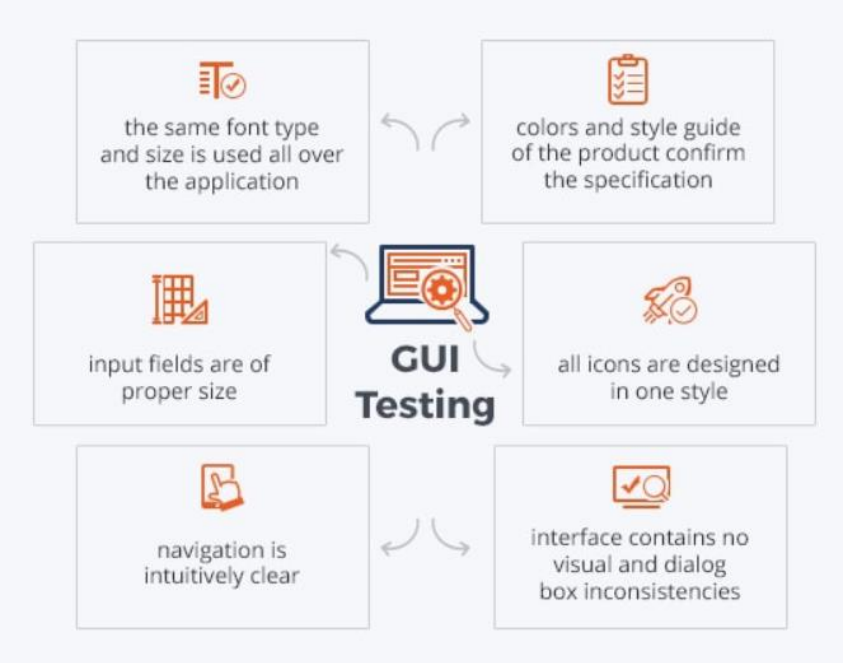
Οι μικρότερες μονάδες κώδικα, που ελέγχονται ανεξάρτητα μέσω δοκιμών μονάδων, μπορούν να χωριστούν σε μικρότερα τμήματα ή ομάδες. Αυτή η αποκέντρωση επιτρέπει την ταχύτερη παράδοση λύσεων, ειδικά όταν εργάζεστε σε έργα μεγάλης κλίμακας.

Συμπερασματικά η διαδικασία επινόησης ελέγχων μικρών μονάδων αναγκάζει να σκεφτεί κανείς τη λειτουργικότητα με βάση τους στόχους του χρήστη. Η ασφάλεια και η ταχύτητα είναι επίσης σημαντικά οφέλη των ελέγχων μονάδων. Όσο αργότερα εντοπιστεί ένα σφάλμα κατά τη διαδικασία ανάπτυξης, τόσο πιο δύσκολο και δαπανηρό μπορεί να είναι να διορθωθεί. Επίσης, επιτρέπει να γίνουν αλλαγές στον κώδικα χωρίς να υπάρχει ανησυχία ότι θα καταστραφούν άλλα τμήματα του κώδικα. Όταν τα σφάλματα εντοπίζονται νωρίς στη διαδικασία ανάπτυξης (πριν φτάσουν στην QA ή την παραγωγή), οι προγραμματιστές μπορούν να κάνουν γρήγορα αλλαγές για να διορθώσουν το πρόβλημα - εξοικονομώντας σε όλους χρόνο και χρήμα.

Αξίζει να αναγνωριστεί ότι η πραγματοποίηση ελέγχων μονάδας δεν εγγυάται ότι ο κώδικάς σας θα είναι εντελώς απαλλαγμένος από σφάλματα. Απλά δεν μπορούν να εντοπίσουν όλα τα σφάλματα στην εφαρμογή. Και ενώ επικυρώνουν τον τρόπο λειτουργίας των ίδιων των μονάδων, δεν θα αποκαλύψουν μεγαλύτερα σφάλματα ολοκλήρωσης, συστήματος ή αποδοχής. Απαιτούν επίσης σημαντική επένδυση χρόνου, αφού πρέπει να εκτελούνται συχνά δοκιμές και να υπάρχει η γνώση ότι γράφονται σωστές και επί της συγκεκριμένης λειτουργίας έλεγχοι σε διαφορετικούς τύπους κώδικα.

8.2 UI TESTING

8.2.1 Τι είναι το UI Testing;



Εικόνα 8.4

Η δοκιμή διεπαφής χρήστη, γνωστή και ως δοκιμή αποδοχής διεπαφής χρήστη, είναι ένας τύπος δοκιμής κατά την οποία δοκιμάζουμε τη διεπαφή χρήστη για την εφαρμογή Ιστού για να διασφαλίσουμε ότι λειτουργεί ομαλά ή εάν υπάρχει κάποιο ελάττωμα που παρεμβαίνει στη συμπεριφορά του χρήστη και δεν πληροί τα καθορισμένα κριτήρια.

Το UI Testing, που αναγνωρίζεται επίσης ως δοκιμή GUI, περιλαμβάνει την εξέταση των στοιχείων διεπαφής χρήστη του λογισμικού για να διασφαλιστεί ότι αποδίδουν όπως αναμένεται. Αυτό συνήθως περιλαμβάνει την αξιολόγηση των οπτικών στοιχείων με τα οποία αλληλεπιδρούν οι χρήστες. Ο πρωταρχικός στόχος είναι να επικυρωθεί όχι μόνο η λειτουργικότητα, αλλά και η απόδοση αυτών των στοιχείων. Το UI Testing ουσιαστικά λειτουργεί ως σημείο ελέγχου ποιότητας, επαληθεύοντας ότι η διεπαφή χρήστη του λογισμικού ευθυγραμμίζεται με τις ανάγκες και τις προσδοκίες του χρήστη, βελτιώνοντας έτσι τη συνολική εμπειρία χρήστη.

Η γνώση του τρόπου με τον οποίο ο χρήστης θα αλληλεπιδράσει με τον ιστότοπο είναι κρίσιμης σημασίας για τη δοκιμή διεπαφής χρήστη. Με άλλα λόγια, κατά την εκτέλεση δοκιμών διεπαφής χρήστη, ο ελεγκτής επιχειρεί να μιμηθεί τη συμπεριφορά του χρήστη για να καθορίσει πώς θα αλληλεπιδράσει ο χρήστης με τον ιστότοπο και εάν ο ιστότοπος λειτουργεί όπως προβλέπεται. Για παράδειγμα, ένα μικρό ελάττωμα διεπαφής χρήστη, όπως ένα πρόβλημα με το κουμπί CTA, μπορεί να εμποδίσει τον επισκέπτη του ιστότοπού σας να συμπληρώσει τη φόρμα δυνητικού πελάτη και, επομένως, να μην πραγματοποιήσει ποτέ μετατροπή. Από την άλλη πλευρά, ποιος ξέρει αν αυτός ήταν ο χρήστης που αύξησε τελικά την απόδοση επένδυσης;

Ένας ιστότοπος περιλαμβάνει πολλά στοιχεία Ιστού γραμμένα σε CSS, JavaScript και άλλες γλώσσες. Η δοκιμή διεπαφής χρήστη καταγράφει αυτά τα στοιχεία για την εκτέλεση δοκιμών και ισχυρισμών σε αυτά. Επικεντρώνεται κυρίως στις δομικές και αισθητικές πτυχές του ιστότοπου, καθώς είναι πιο κρίσιμες για τον χρήστη από το πώς αποθηκεύονται τα δεδομένα στη βάση δεδομένων.

Η δοκιμή διεπαφής χρήστη (User Interface Testing) καλύπτει το διαδραστικό μέρος του ιστότοπου, καθώς το στοιχείο του ιστότοπου μπορεί να συνδεθεί σε οθόνη, πληκτρολόγιο, ποντίκι ή οποιαδήποτε άλλη συσκευή που χρησιμοποιεί ο χρήστης για να αλληλεπιδράσει με τον ιστότοπο.

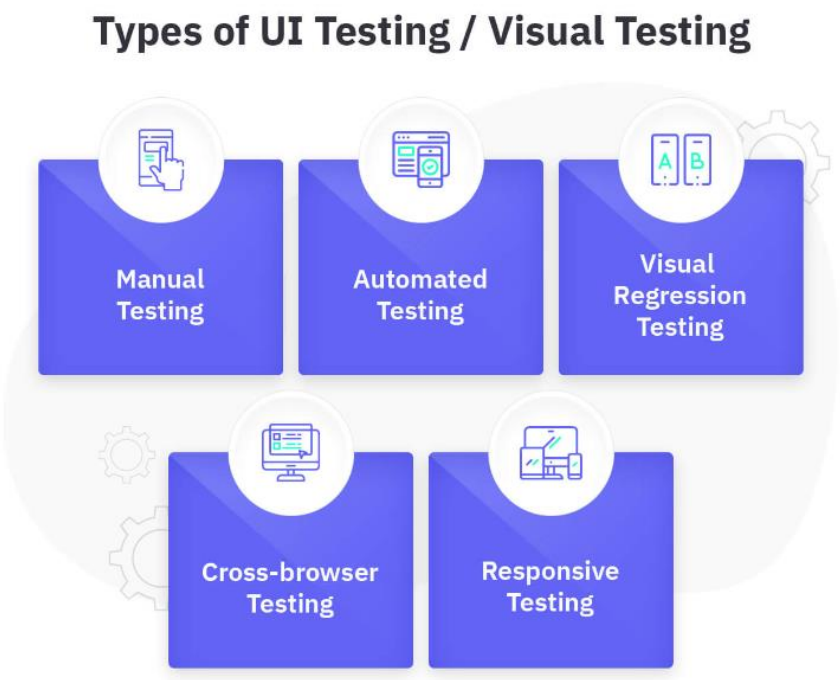
8.2.2 Γιατί είναι σημαντικό το UI Testing;

Το μέσο μέγεθος ενός ιστότοπου έχει αυξηθεί με την πάροδο του χρόνου. Καθώς οι ιστότοποι μεγαλώνουν σε μέγεθος, τώρα περιέχουν εκατοντάδες σελίδες. Επιπλέον, μια σελίδα περιέχει εκατοντάδες στοιχεία που απαιτούνται για τη δημιουργία ενός πλήρους ιστότοπου. Αυτό δημιουργεί ένα τεράστιο φορτίο στον διακομιστή που ανακτά τον ιστότοπο. Ένας αργός ιστότοπος δεν είναι καλό σημάδι για κανέναν προγραμματιστή ιστού. Η δοκιμή αυτού του σεναρίου εμπίπτει επίσης στη δοκιμή διεπαφής χρήστη και μπορεί αναμφίβολα να βελτιώσει την απόδοση.

Με την πολυπλοκότητα των σύγχρονων εφαρμογών, το να βασίζεσαι αποκλειστικά σε δοκιμές μονάδων και ολοκλήρωσης δεν είναι βιώσιμο. Οι χρήστες σας εξερευνούν κάθε γωνιά της εφαρμογής σας και έχουν υψηλές προσδοκίες για εύκολη στη χρήση, οπτικά ελκυστική διεπαφή χρήστη. Με τη δοκιμή του UI σας, θα κάνετε πολύ περισσότερα από μικρά σφάλματα, θα βελτιώσετε και το UX σας. Και με ένα θετικό UX, οι χρήστες σας θα είναι πιο πιθανό να επιστρέψουν, ενισχύοντας την ικανοποίηση και τη συνολική αφοσίωσή τους. Αυτό προχωρά περαιτέρω επειδή οι ικανοποιημένοι και πιστοί χρήστες θα τραγουδήσουν υψηλούς επαίνους για το προϊόν σας, ενισχύοντας επίσης μια θετική φήμη επωνυμίας. Επιπλέον, οι πρόσθετες δοκιμές θα εντοπίσουν περισσότερα σφάλματα στο σύστημά σας προτού φτάσει στην παραγωγή. Η διεξοδική δοκιμή της διεπαφής σας θα διασφαλίσει ότι οι χρήστες δεν θα αντιμετωπίσουν σημαντικές δυσλειτουργίες που μπορεί να καταστρέψουν την εμπειρία. Η πιο εμπεριστατωμένη κάλυψη δοκιμών θα βοηθήσει τις ομάδες σας να εξοικονομήσουν χρόνο και χρήμα μακροπρόθεσμα, αντιμετωπίζοντας προβλήματα πολύ νωρίτερα στην ανάπτυξη.

Σύμφωνα με έρευνα της Google, το 53% των χρηστών εγκαταλείπει εργασίες που χρειάζονται περισσότερα από 3 δευτερόλεπτα για να φορτωθούν. Μπορείτε να μειώσετε αυτήν την καθυστέρηση βελτιστοποιώντας το JavaScript και το CSS. Ως εκ τούτου, είναι απαραίτητο να δοκιμάσετε τη διεπαφή χρήστη για τη βελτίωση της απόδοσης του ιστότοπου.

8.2.3 ΤΕΧΝΙΚΕΣ UI TESTING



Εικόνα 8.5

Οι τεχνικές δοκιμής βοηθούν στην εκτέλεση δοκιμών διεπαφής χρήστη (User Interface Testing). Αφού επιλέξουμε μια τεχνική δοκιμής, είναι πολύ πιο εύκολο να ακολουθήσουμε την ιδέα και να λάβουμε τα αποτελέσματα.

- Διερευνητική Δοκιμή

Η διερευνητική δοκιμή δεν απαιτεί προσχεδιασμό. Ο ελεγκτής δημιουργεί δοκιμές με βάση την εμπειρία και άλλους παράγοντες, όπως τα προηγούμενα αποτελέσματα. Ανάλογα με το έργο, αυτές οι παράμετροι μπορεί να διαφέρουν. Στις διερευνητικές δοκιμές, οι δοκιμαστές έχουν ένα ευρύ φάσμα ευελιξίας και ευκαιριών. Για παράδειγμα, η διερευνητική δοκιμή της δοκιμής διεπαφής χρήστη ιστού μπορεί να αποκαλύψει κρυφές περιπτώσεις δοκιμών, καθώς διαφορετικά μηχανήματα μπορεί να χειρίζονται διαφορετικά τη διεπαφή χρήστη. Ο ελεγκτής μπορεί να χρησιμοποιήσει αυτοματισμό για να εκτελέσει τις θήκες σε διαφορετικά σύνολα δεδομένων, ενώ αντιμετωπίζει προκλήσεις διερευνητικών δοκιμών. Μπορείτε να εκτελέσετε τόσο χειροκίνητη όσο και αυτοματοποιημένη διερευνητική δοκιμή.

- Δοκιμή σεναρίου

Η διαδικασία ελέγχου σεναρίου ξεκινά μόλις γράψετε και ορίσετε τις περιπτώσεις δοκιμής. Ο ελεγκτής ορίζει τα σενάρια που περιγράφουν τις καταχωρήσεις του ελεγκτή και τις αναμενόμενες εξόδους. Στη

συνέχεια, τα αποτελέσματα αναλύονται και αναφέρονται. Οι δοκιμαστές μπορούν να εκτελούν χειροκίνητες ή αυτοματοποιημένες δοκιμές με σενάριο με τον ίδιο τρόπο όπως οι διερευνητικές δοκιμές. Λόγω του μεγάλου αριθμού γραμμών κώδικα και της αυξημένης πολυπλοκότητας των σημερινών έργων, συνιστάται η αυτοματοποιημένη δοκιμή δέσμης ενεργειών.

- Δοκιμή εμπειρίας χρήστη

Μπορείτε να χρησιμοποιήσετε την τεχνική δοκιμής εμπειρίας χρήστη παραδίδοντας το ολοκληρωμένο έργο στον τελικό χρήστη. Ο τελικός χρήστης μπορεί στη συνέχεια να χρησιμοποιήσει το ολοκληρωμένο έργο και να στείλει σχόλια στον προγραμματιστή μέσω της ομάδας δοκιμών. Οι οργανισμοί μπορούν επίσης να παρέχουν μια έκδοση beta του προϊόντος στους τελικούς χρήστες τους για να λαμβάνουν σχόλια από διάφορες γεωγραφικές τοποθεσίες. Αυτό οδηγεί σε ένα ιδανικό περιβάλλον δοκιμών.

Δεν είναι δύσκολο να ερμηνευτεί ότι η δοκιμή εμπειρίας χρήστη είναι ένας τύπος διερευνητικής δοκιμής, καθώς οι χρήστες δεν ξέρουν τι να δοκιμάσουν ή πώς να δοκιμάσουν, δηλαδή δεν υπάρχει προκαθορισμένη στρατηγική. Στην πραγματικότητα, γίνεται χειροκίνητα. Μπορείτε να εκτελέσετε δοκιμή εμπειρίας χρήστη σε ένα μερικό προϊόν για να δοκιμάσετε τη διεπαφή χρήστη σε πολλές οθόνες και τοποθεσίες χωρίς να αναπτύξετε ολόκληρο το έργο. Αυτό επιτρέπει στους οργανισμούς να δοκιμάσουν μικρότερα στοιχεία, όπως τη δοκιμή ολόκληρου του έργου. Αυτό τελικά βελτιώνει την ποιότητα του προϊόντος.

- Σενάρια δοκιμής διεπαφής χρήστη

Πριν από τη διεξαγωγή δοκιμών διεπαφής χρήστη, οι ομάδες δοκιμών πρέπει να δημιουργήσουν ένα σχέδιο δοκιμής διεπαφής χρήστη που περιλαμβάνει περιοχές της υπό δοκιμή εφαρμογής, το εύρος μιας εφαρμογής Ιστού, προσεγγίσεις δοκιμών, δραστηριότητες κ.λπ. Ένα καλό σχέδιο δοκιμής διεπαφής χρήστη μπορεί να σας βοηθήσει να δημιουργήσετε έναν δομικό χάρτη της συνολικής σας διαδικασίας, να διαχειριστείτε καλύτερα τον χρόνο σας και να χρησιμεύσει ως οδηγός για να διασφαλίσετε ότι όλοι βρίσκονται σε καλό δρόμο.

8.2.4 ΟΦΕΛΗ UI TESTING

Πλεονεκτήματα του UI / Visual Testing

Το UI / Visual Testing είναι ένα απαραίτητο στοιχείο της σύγχρονης ανάπτυξης λογισμικού, διασφαλίζοντας ότι οι εφαρμογές που απευθύνονται στον χρήστη πληρούν αυστηρά πρότυπα ποιότητας και προσφέρουν μια απρόσκοπτη εμπειρία χρήστη. Καθώς η τεχνολογία συνεχίζει να προοδεύει και οι προσδοκίες των χρηστών εξελίσσονται, η σημασία του UI / Visual Testing δεν μπορεί να υπερεκτιμηθεί. Ακολουθούν ορισμένα βασικά πλεονεκτήματα της ενσωμάτωσης UI / Visual Testing στη διαδικασία ανάπτυξης λογισμικού:

- Βελτιωμένη διασφάλιση ποιότητας: Ένα από τα κύρια πλεονεκτήματα του UI / Visual Testing είναι η ικανότητά του να βελτιώνει τη συνολική ποιότητα των προϊόντων λογισμικού. Εξετάζοντας σχολαστικά τα οπτικά στοιχεία και τα στοιχεία της διεπαφής χρήστη, οι δοκιμαστές μπορούν να εντοπίσουν πιθανά ελαττώματα και ασυνέπειες νωρίς στον κύκλο ανάπτυξης. Αυτή η προληπτική προσέγγιση επιτρέπει στους προγραμματιστές να αντιμετωπίζουν τα ζητήματα εγκαίρως, αποτρέποντάς τους από το να μετατραπούν σε πιο σημαντικά προβλήματα στη γραμμή. Τελικά, αυτό οδηγεί σε ένα τελικό προϊόν υψηλότερης ποιότητας που είναι πιο σταθερό, αξιόπιστο και φιλικό προς το χρήστη.

- Βελτιωμένη εμπειρία χρήστη: Στο σημερινό ανταγωνιστικό ψηφιακό τοπίο, η παροχή ανώτερης εμπειρίας χρήστη είναι πρωταρχικής σημασίας για την επιτυχία οποιασδήποτε εφαρμογής λογισμικού. Το UI / Visual Testing διαδραματίζει κρίσιμο ρόλο στη διασφάλιση ότι οι χρήστες αντιμετωπίζουν μια διαισθητική, οπτικά ελκυστική διεπαφή που ανταποκρίνεται στις προσδοκίες τους. Με την προσομοίωση σεναρίων χρήσης πραγματικού κόσμου και τον έλεγχο κάθε πτυχής της διεπαφής χρήστη, οι υπεύθυνοι δοκιμών. Η σχολαστική προσοχή στη λεπτομέρεια βοηθά στην εξάλειψη προβλημάτων χρηστικότητας, όπως η σύγχυση της πλοήγησης, τα ασυνεπή στοιχεία σχεδίασης ή τα εμπόδια προσβασιμότητας, με αποτέλεσμα τελικά μια πιο ελκυστική και ικανοποιητική εμπειρία χρήστη.

- Εξοικονόμηση κόστους: Αν και ορισμένοι μπορεί να αντιλαμβάνονται τη διεπαφή χρήστη / οπτική δοκιμή ως πρόσθετη δαπάνη, στην πραγματικότητα αποφέρει σημαντική εξοικονόμηση κόστους μακροπρόθεσμα. Εντοπίζοντας ελαττώματα και ασυνέπειες νωρίς στη διαδικασία ανάπτυξης, το UI / Visual Testing συμβάλλει στην ελαχιστοποίηση της ανάγκης για εκτεταμένη επανεξέταση και εντοπισμό σφαλμάτων αργότερα. Αυτή η προληπτική προσέγγιση όχι μόνο μειώνει τον συνολικό χρόνο ανάπτυξης, αλλά μειώνει επίσης τον κίνδυνο δαπανηρών καθυστερήσεων και καθυστερήσεων. Επιπλέον, διασφαλίζοντας ένα τελικό προϊόν υψηλότερης ποιότητας, το UI / Visual Testing συμβάλλει στην ελαχιστοποίηση της πιθανότητας προβλημάτων μετά την κυκλοφορία και δαπανηρών διορθώσεων σφαλμάτων, εξοικονομώντας έτσι χρόνο και πόρους.

- Χρονική αποδοτικότητα: Ο χρόνος είναι ουσιαστικός στην ανάπτυξη λογισμικού και η διεπαφή χρήστη / οπτική δοκιμή μπορεί να βοηθήσει στον εξορθολογισμό της διαδικασίας δοκιμών για τη μεγιστοποίηση της αποτελεσματικότητας. Μέσω της αυτοματοποίησης και της χρήσης εξειδικευμένων εργαλείων δοκιμών, οι δοκιμαστές μπορούν γρήγορα να επικυρώσουν τις οπτικές πτυχές μιας εφαρμογής σε διαφορετικές συσκευές, μεγέθη οθόνης και λειτουργικά συστήματα. Αυτή η αυτοματοποιημένη προσέγγιση όχι μόνο επιταχύνει τον κύκλο δοκιμών, αλλά ελευθερώνει επίσης πολύτιμο χρόνο στους δοκιμαστές ώστε να επικεντρωθούν σε πιο σύνθετες και κρίσιμες πτυχές της διαδικασίας δοκιμών, όπως οι δοκιμές λειτουργικότητας και απόδοσης.

- Βελτιωμένη συνεργασία: Το UI / Visual Testing ενισχύει τη συνεργασία μεταξύ προγραμματιστών, σχεδιαστών και επαγγελματιών QA παρέχοντας ένα κοινό πλαίσιο για την αξιολόγηση των οπτικών πτυχών μιας εφαρμογής. Με τη συμμετοχή όλων των ενδιαφερομένων στη διαδικασία δοκιμών, οι ομάδες μπορούν να αποκτήσουν πολύτιμες γνώσεις και προοπτικές, να εντοπίσουν πιθανά ζητήματα πιο αποτελεσματικά και να συνεργαστούν για λύσεις συλλογικά. Αυτή η διεπιστημονική προσέγγιση όχι μόνο οδηγεί σε ένα πιο εκλεπτυσμένο τελικό προϊόν, αλλά προωθεί επίσης μια κουλτούρα ομαδικής εργασίας και καινοτομίας εντός του οργανισμού.
- Βελτιωμένη φήμη επωνυμίας: Μια οπτικά ελκυστική και φιλική προς τον χρήστη διεπαφή δεν είναι απαραίτητη μόνο για την προσέλκυση και τη διατήρηση των χρηστών, αλλά και για την οικοδόμηση μιας ισχυρής φήμης επωνυμίας. Το UI / Visual Testing συμβάλλει στη διασφάλιση της συνέπειας στο σχεδιασμό, την επωνυμία και την εμπειρία χρήστη σε όλα τα σημεία επαφής, ενισχύοντας την ταυτότητα της επωνυμίας και εμπιστοσύνη στους χρήστες. Παρέχοντας μια απρόσκοπτη και εκλεπτυσμένη εμπειρία χρήστη, οι εταιρείες μπορούν να διαφοροποιηθούν από τους ανταγωνιστές και να καθιερωθούν ως ηγέτες στους αντίστοιχους κλάδους τους.

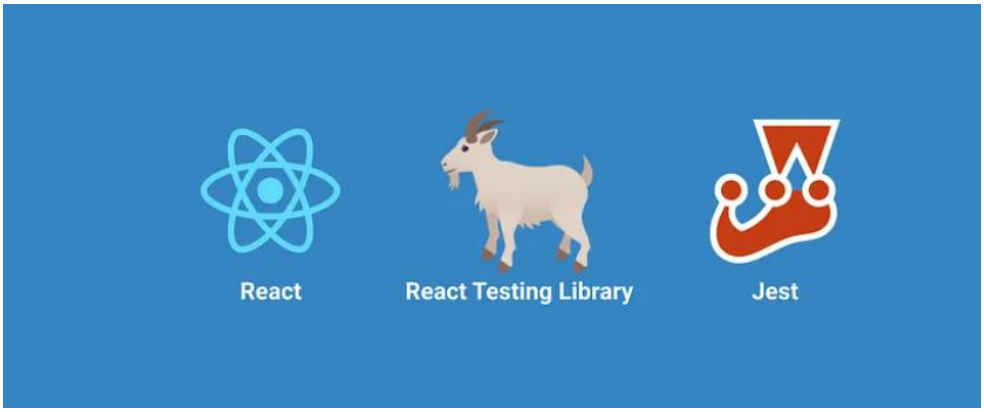
- Συμμόρφωση με Πρότυπα και Κανονισμούς: Στο σημερινό ρυθμιστικό περιβάλλον, η συμμόρφωση με τα πρότυπα και τους κανονισμούς του κλάδου είναι αδιαπραγμάτευτη. Το UI / Visual Testing διαδραματίζει κρίσιμο ρόλο στη διασφάλιση ότι οι εφαρμογές λογισμικού συμμορφώνονται με τα σχετικά πρότυπα, οδηγίες και απαιτήσεις προσβασιμότητας. Εντοπίζοντας πιθανά ζητήματα συμμόρφωσης νωρίς στη διαδικασία ανάπτυξης, οι υπεύθυνοι δοκιμών μπορούν να εργαστούν προληπτικά για την αντιμετώπισή τους, ελαχιστοποιώντας τον κίνδυνο κυρώσεων για μη συμμόρφωση, νομικές ευθύνες και ζημιά στη φήμη.

Συμπερασματικά, το UI / Visual Testing αποτελεί ακρογωνιαίο λίθο της σύγχρονης ανάπτυξης λογισμικού, προσφέροντας ένα ευρύ φάσμα πλεονεκτημάτων που συμβάλλουν στη συνολική επιτυχία ενός έργου. Από τη βελτίωση της ποιότητας και της εμπειρίας χρήστη έως την εξοικονόμηση χρόνου και πόρων, το UI / Visual Testing είναι μια αξιόλογη επένδυση που αποδίδει οφέλη όσον αφορά την ικανοποίηση των πελατών, τη φήμη της επωνυμίας και τη μακροπρόθεσμη βιωσιμότητα. Δίνοντας προτεραιότητα στο UI / Visual Testing και ενσωματώνοντάς το απρόσκοπτα στη διαδικασία ανάπτυξης, οι εταιρείες μπορούν να προσφέρουν εξαιρετικά προϊόντα λογισμικού που ενθουσιάζουν τους χρήστες και αντέχουν στη δοκιμασία του χρόνου.

ΚΕΦΑΛΑΙΟ 9: ΤΕΧΝΙΚΕΣ ΚΑΙ ΕΡΓΑΛΕΙΑ

9.1 JEST

9.1.1 Τι είναι το Jest;



Εικόνα 9.1

Το Jest είναι ένας δοκιμαστικός δρομέας ανοιχτού κώδικα. Το Jest δημιουργείται και διατηρείται από το Facebook (ή το Meta αυτές τις μέρες). Έχει σχεδιαστεί ειδικά για να δοκιμάζει διάφορα είδη δοκιμαστικών περιπτώσεων με τους ισχυρισμούς του. Το Jest είναι μια λύση all-in-one που παρέχει μια ολοκληρωμένη εμπειρία δοκιμών. Μία από τις βασικές αρμοδιότητες ενός δοκιμαστικού δρομέα όπως το Jest είναι να βρίσκει/εκτελεί αυτόματα δοκιμές και να καθορίζει εάν το τεστ είναι επιτυχές ή όχι. Υποστηρίζει επίσης διάφορους τύπους δοκιμών, συμπεριλαμβανομένων δοκιμών μονάδας, δοκιμών ενοποίησης και δοκιμών από άκρο σε άκρο. Το Jest συνδυάζεται πολύ καλά με πολλές γνωστές τεχνολογίες όπως node, typescript, javascript, react, angular, vue κ.λπ. Το Jest υπερηφανεύεται ότι προσφέρει μια ολοκληρωμένη και χωρίς προβλήματα εμπειρία. Η πληρότητα προέρχεται από το γεγονός ότι το Jest δεν βασίζεται σε εργαλεία τρίτων για μεγάλο μέρος της λειτουργικότητας του, όπως κάνουν ορισμένοι ανταγωνιστές. Και το κομμάτι χωρίς προβλήματα οφείλεται στη μηδενική ρύθμιση παραμέτρων του Jest.

Το Jest διαθέτει ένα ευρύ φάσμα βοηθητικών προγραμμάτων δοκιμών και διεκδίκησης που βοηθούν τους προγραμματιστές να δοκιμάσουν εύκολα τις λειτουργίες. Το Jest παρέχει άμεση υποστήριξη για δοκιμές κοροϊδίας και στιγμιότυπων χωρίς καμία μη αυτόματη διαμόρφωση. Το Jest βελτιώνει την εμπειρία προγραμματιστή παρέχοντας λειτουργίες διεκδίκησης και υποστήριξη για κάλυψη κώδικα και δοκιμαστικές σουίτες. Επιπλέον, το Jest έχει τη δυνατότητα να εκτελεί δοκιμές παράλληλα για να βελτιώσει την απόδοση και να βελτιώσει τη συνολική εμπειρία δοκιμών. Δεδομένου ότι είναι αγνωστικιστικό πλαίσιο, έχει μια μεγάλη κοινότητα που σχετίζεται με αυτό.

9.1.2 REACT TESTING LIBRARY

Τι είναι η βιβλιοθήκη δοκιμών React;

Η διαρκώς μεταβαλλόμενη φύση της ανάπτυξης λογισμικού απαιτεί εργαλεία που είναι ευέλικτα και συνεπή. Η βιβλιοθήκη δοκιμών React προσφέρει αυτήν την προσαρμοστικότητα και συνέπεια. Η

ευελιξία της επιτρέπει στους προγραμματιστές να προσαρμόσουν τις δοκιμές τους σύμφωνα με τις ανάγκες τους, ενώ η συνέπεια της εξασφαλίζει ότι οι δοκιμές εκτελούνται αξιόπιστα και αποτελεσματικά κάθε φορά.

Το React Testing Library είναι μια ελαφριά βιβλιοθήκη δοκιμών για εφαρμογές React. Εστιάζει στη δοκιμή της συμπεριφοράς της εφαρμογής μιμούμενος ή προσομοιώνοντας τον τρόπο με τον οποίο ο τελικός χρήστης θα αλληλεπιδρούσε με τα στοιχεία αντί να δοκιμάζει τις λεπτομέρειες υλοποίησης του στοιχείου. Η βιβλιοθήκη παρέχει ένα σύνολο λειτουργιών με τις οποίες μπορείτε να αποδώσετε το στοιχείο σας και να το δοκιμάσετε με οπτικές αλλαγές διεπαφής χρήστη που πραγματοποιούνται στην οθόνη όταν πραγματοποιούμε κάποια αλληλεπίδραση.

Η βιβλιοθήκη δοκιμών React είναι ειδικά προσαρμοσμένη για τη δοκιμή στοιχείων στο οικοσύστημα React. Έχει σχεδιαστεί λαμβάνοντας υπόψη τις καλύτερες συμβάσεις προγραμματιστών. Επιπλέον, το οικοσύστημα ενθαρρύνει τους προγραμματιστές να έχουν υπόψη τους την προσβασιμότητα ενώ γράφουν δοκιμές για στοιχεία. Εφόσον δεν εκτελούμε αυτές τις δοκιμές στο πρόγραμμα περιήγησης, υποστηρίζει την απόδοση των στοιχείων αξιοποιώντας εσωτερικά το Virtual DOM. Ένα από τα καλύτερα πράγματα σχετικά με τη βιβλιοθήκη δοκιμών React είναι ότι παρέχει βοηθητικά προγράμματα αλληλεπίδρασης με τον χρήστη, όπως το fireEvent, τα οποία μπορούν να μιμηθούν την αλληλεπίδραση με τον χρήστη, όπως κλικ, διπλό κλικ, πληκτρολόγηση, αιώρηση κ.λπ.

Ολοκληρώνοντας, η εξερεύνηση της βιβλιοθήκης δοκιμών Jest and React έχει ρίξει φως στο δυναμικό δίδυμο που ενισχύει τις δοκιμές στο οικοσύστημα React. Το React Testing Library είναι μια βιβλιοθήκη που δίνει έμφαση στις δοκιμές με επίκεντρο τον χρήστη, ενώ το Jest είναι ένας δοκιμαστικός δρομέας που ικανοποιεί διάφορες ανάγκες δοκιμών στο React. Τελικά, η επιλογή συνοψίζεται στις απαιτήσεις του έργου και στις προτιμήσεις/φιλοσοφία της ομάδας σας. Ίσως η βέλτιστη στρατηγική να βρίσκεται σε μια συγχώνευση και των δύο κόσμων, αξιοποιώντας τις ολοκληρωμένες δυνατότητες δοκιμών του Jest και συμπληρώνοντάς τες με την προσέγγιση του χρήστη React Testing Library.

ΚΕΦΑΛΑΙΟ 10: ΟΔΗΓΟΣ ΓΙΑ ΔΗΜΙΟΥΡΓΙΑ ΜΟΝΑΔΩΝ ΕΛΕΓΧΟΥ

Σε αυτόν τον οδηγό, θα περιγράψουμε αναλυτικά τα πρώτα βήματα για τη δημιουργία unit tests για το Header component χρησιμοποιώντας React Testing Library και Jest. Η διαδικασία περιλαμβάνει την εγκατάσταση των απαιτούμενων βιβλιοθηκών, τη δημιουργία ενός mock store και την απομόνωση των εξαρτήσεων του component.

10.1 ΑΠΑΙΤΟΥΜΕΝΑ ΒΗΜΑΤΑ

Βήμα 1: Απαιτούμενες Βιβλιοθήκες

Για να ξεκινήσουμε, πρέπει να εγκαταστήσουμε τις απαραίτητες βιβλιοθήκες για τη δοκιμή του React component. Αυτές περιλαμβάνουν το @testing-library/react, @testing-library/jest-dom, redux-mock-store, redux-thunk, react-redux, react-router-dom, και jest.

```
import React from "react";
import { render, screen, fireEvent } from "@testing-library/react";
import { Provider } from "react-redux";
import { MemoryRouter } from "react-router-dom";
import "@testing-library/jest-dom";
```

Βήμα 2: Δημιουργία Mock Store

Ο όρος "mock" αναφέρεται στη διαδικασία δημιουργίας ψεύτικων αντικειμένων ή λειτουργιών που μιμούνται τη συμπεριφορά πραγματικών μερών, συναρτήσεων ή δεδομένων. Ο κύριος σκοπός των mocks είναι να απομονώσουν το κομμάτι του κώδικα που θέλουμε να δοκιμάσουμε, εξαλείφοντας εξωτερικές εξαρτήσεις και επιτρέποντάς μας να εστιάσουμε στη συμπεριφορά της μονάδας που δοκιμάζεται.

Για να δοκιμάσουμε το Header component που χρησιμοποιεί το Redux, πρέπει να δημιουργήσουμε ένα mock store. Το mock store θα μιμηθεί την συμπεριφορά του πραγματικού store χωρίς να έχουμε ανάγκη να χρησιμοποιήσουμε το πλήρες Redux store. Αυτό μας επιτρέπει να απομονώσουμε το component από το πραγματικό περιβάλλον της εφαρμογής και να επικεντρωθούμε στη λειτουργικότητά του.

Εισαγωγή Απαιτούμενων Βιβλιοθηκών:

```
import configureStore from "redux-mock-store";
import thunk from "redux-thunk";
```

Βήμα 3: Δημιουργία Mock Components

Για να απομονώσουμε το Header component από τις εξαρτήσεις του, θα κάνουμε mock τα components και modules που χρησιμοποιεί.

Εικονικά Components από την react-icons βιβλιοθήκη:

```
jest.mock("react-icons/io5", () => ({
    IoMenu: () => <div>IoMenu</div>,
    IoPersonOutline: () => <div>IoPersonOutline</div>,
    IoIosArrowForward: () => <div>IoIosArrowForward</div>,
    IoIosArrowDown: () => <div>IoIosArrowDown</div>,
    IoArrowUpCircleOutline: () => <div>IoArrowUpCircleOutline</div>,
}));
jest.mock("react-icons/bs", () => ({
    BsCart3: () => <div>BsCart3</div>,
}));
jest.mock("react-icons/rx", () => ({
    RxCross1: () => <div>RxCross1</div>,
```

```
}));
jest.mock("react-icons/fi", () => ({
    FiMenu: () => <div>FiMenu</div>,
}));
jest.mock("react-icons/lu", () => ({
    LuLogOut: () => <div>LuLogOut</div>,
}));
jest.mock("react-icons/md", () => ({
    MdDeleteForever: () => <div>MdDeleteForever</div>,
}));
```

Mock για το reactstrap Button Component:

```
jest.mock("reactstrap", () => ({
    Button: ({ children, ...props }) => <button {...props}>{children}</button>,
}));
```

Mock για άλλα Components και Views:

```
jest.mock("../../Global/ContactLayer", () => () => <div>ContactLayer</div>);
jest.mock("../../header/MenuSidebar", () => {
    const React = require("react");
    return React.forwardRef((props, ref) => <div ref={ref}>MenuSidebar</div>);
});
jest.mock("../../../views/Signup", () => () => <div>Signup</div>);
jest.mock("../../../views/Login", () => () => <div>Login</div>);
```

10.1.1 Δημιουργία αρχικών δεδομένων

Στο πλαίσιο της δημιουργίας unit tests για το Header component, είναι απαραίτητο να δημιουργήσουμε ένα mock store που περιέχει τα αρχικά δεδομένα που θα χρησιμοποιηθούν κατά τη διάρκεια των δοκιμών. Τα δεδομένα αυτά θα μιμηθούν την πραγματική κατάσταση της εφαρμογής ώστε να ελέγξουμε σωστά τη συμπεριφορά του Header component. Παρακάτω περιγράφονται αναλυτικά τα βήματα για τη δημιουργία των αρχικών δεδομένων στο mock store.

Αρχικά δεδομένα

Πριν από κάθε τεστ, θα δημιουργούμε ένα αρχικό state για το mock store. Αυτό θα περιλαμβάνει δεδομένα για τα προϊόντα, τον χρήστη και το καλάθι αγορών. Αυτό μας επιτρέπει να διασφαλίσουμε ότι το state του mock store επανέρχεται στην αρχική του κατάσταση πριν από κάθε δοκιμή.

Βήμα 4: beforeEach

Αυτή η συνάρτηση εκτελείται πριν από κάθε τεστ, εξασφαλίζοντας ότι το state του mock store επανέρχεται στην αρχική του κατάσταση πριν από κάθε δοκιμή.

```
beforeEach(() => {
        store = mockStore({
});
    });
```

Βήμα 5: Ορισμός της δομής των δεδομένων

Περιλαμβάνει την κατάσταση φόρτωσης των προϊόντων, την τιμή αναζήτησης, και τη λίστα όλων των προϊόντων. Κάθε προϊόν έχει διάφορα χαρακτηριστικά όπως κατηγορία, ημερομηνίες δημιουργίας και ενημέρωσης, περιγραφή, έκπτωση, ταυτότητα, εικόνες, τιμή, κωδικό προϊόντος, υποκατηγορία, τίτλο, και κατάσταση (π.χ. διαγραφή, αγαπημένο).

```
productsLoading: false,
                searchValue: "",
                allProducts: [
                    {
                        category: {
                            label: "Pc Hardware",
deliveryDays: "4-10",
                        description:
                            '<p>Βασικά χαρακτηριστικά:
price: "602.90",
                        productCode: "38442875",
                        selectedProduct: [],
                        subCategory: {
                            value: "Motherboards",
                            label: "Motherboards",
                        },
                        title: "      Asus ROG Maximus Z790 Hero Wi-Fi
Motherboard ATX με Intel 1700 Socket"
```

Βήμα 6: Χρήστης (authUser):

Αρχικά, ο χρήστης είναι null, δηλαδή δεν είναι συνδεδεμένος. Το αντικείμενο authUser περιέχει τις πληροφορίες του χρήστη. Η ιδιότητα user περιέχει τις λεπτομέρειες του χρήστη. Όταν η τιμή του είναι null, υποδηλώνει ότι ο χρήστης δεν είναι συνδεδεμένος.

```
authUser: {
```

```
                user: null,
            }
```

Βήμα 7: Καλάθι Αγορών (cartReducer):

Περιέχει μια λίστα με τα αντικείμενα στο καλάθι, η οποία αρχικά είναι κενή.

```
cartReducer: {
                items: [],
            }
```

Διαχειρίζεται τα αντικείμενα στο καλάθι αγορών του χρήστη. Η αρχική του κατάσταση είναι μια κενή λίστα, που υποδηλώνει ότι το καλάθι είναι άδειο.

10.2 ΔΗΜΙΟΥΡΓΙΑ ΤΟΥ ΤΕΣΤ

Βήμα 8: Δημιουργία του τεστ

Η συνάρτηση it χρησιμοποιείται για να δημιουργήσει ένα νέο τεστ. Το πρώτο όρισμα είναι η περιγραφή του τεστ ("should render search bar"), και το δεύτερο είναι μια ασύγχρονη συνάρτηση που περιέχει τις εντολές του τεστ. Η συνάρτηση async () => {} ορίζει ότι η συνάρτηση είναι ασύγχρονη, που σημαίνει ότι μπορούμε να χρησιμοποιήσουμε await μέσα σε αυτή.

```
it("should render search bar", async () => {
```

Βήμα 9: Προετοιμασία του αρχικού state του store

Εδώ, αλλάζουμε το state του Redux store για να δηλώσουμε ότι τα προϊόντα φορτώνονται (productsLoading = true). Αυτό μας επιτρέπει να δοκιμάσουμε τη συμπεριφορά του component όταν η κατάσταση φόρτωσης των προϊόντων είναι ενεργή.

```
store.getState().products.productsLoading = true;
```

Βήμα 10: Απόδοση του component

```
render(
            <Provider store={store}>
                <MemoryRouter>
                    <Header />
                </MemoryRouter>
            </Provider>
        );
```

Η συνάρτηση render από τη βιβλιοθήκη @testing-library/react χρησιμοποιείται για να αποδώσει ένα React component σε ένα εικονικό DOM, επιτρέποντας τη δοκιμή του. Αρχικά, τυλίγουμε το Header component με το Provider από το react-redux, χρησιμοποιώντας το Redux store για να παρέχουμε την

απαιτούμενη κατάσταση της εφαρμογής στο component. Στη συνέχεια, χρησιμοποιούμε το MemoryRouter από το react-router-dom για να παρέχουμε ένα router context στο component μας. Αυτό είναι απαραίτητο για components που χρησιμοποιούν React Router για τη διαχείριση της πλοήγησης. Τελικά, αποδίδουμε το Header component που θέλουμε να δοκιμάσουμε. Αυτή η διαδικασία εξασφαλίζει ότι το component μας έχει πρόσβαση σε όλα τα απαραίτητα context και μπορεί να λειτουργήσει κανονικά κατά τη διάρκεια των δοκιμών.

Βήμα 11: Έλεγχος της ύπαρξης της γραμμής αναζήτησης

```
const searchBar = screen.getByPlaceholderText("Search product...");
        expect(searchBar).toBeTruthy();
```

Η συνάρτηση screen.getByPlaceholderText("Search product...") αναζητεί στο εικονικό DOM ένα στοιχείο input που έχει το placeholder κείμενο "Search product...". Αυτό μας επιτρέπει να βρούμε τη γραμμή αναζήτησης στο αποδοθέν component. Στη συνέχεια, η συνάρτηση expect(searchBar).toBeTruthy() ελέγχει ότι το στοιχείο searchBar υπάρχει, δηλαδή ότι η γραμμή αναζήτησης έχει αποδοθεί επιτυχώς. Αν αυτός ο έλεγχος περάσει, σημαίνει ότι η γραμμή αναζήτησης εμφανίζεται όπως αναμένεται.

Βήμα 12: Προσομοίωση αλλαγής στην γραμμή αναζήτησης

```
const searchInput = screen.getByPlaceholderText("Search product...");
        fireEvent.change(searchInput, { target: { value: "Asus" } });
        expect(searchInput.value).toBe("Asus");
```

Η συνάρτηση fireEvent.change(searchInput, { target: { value: "Asus" } }) χρησιμοποιείται για να προσομοιώσει ένα συμβάν αλλαγής στο input πεδίο της γραμμής αναζήτησης. Συγκεκριμένα, αλλάζει την τιμή του input πεδίου σε "Asus". Αυτή η προσομοίωση μιμείται τη συμπεριφορά του χρήστη που πληκτρολογεί τη λέξη "Asus" στη γραμμή αναζήτησης.

Μετά την προσομοίωση, η συνάρτηση expect(searchInput.value).toBe("Asus") ελέγχει ότι η τιμή του input πεδίου έχει όντως αλλάξει σε "Asus". Αυτός ο έλεγχος διασφαλίζει ότι η προσομοίωση της αλλαγής λειτουργεί σωστά και ότι η γραμμή αναζήτησης ενημερώνεται με την αναμενόμενη τιμή. Αν ο έλεγχος περάσει, σημαίνει ότι η τιμή του input πεδίου έχει αλλάξει επιτυχώς και η γραμμή αναζήτησης λειτουργεί όπως αναμένεται όταν ο χρήστης εισάγει κείμενο.

Βήμα 13: Έλεγχος της ύπαρξης συνδέσμου προϊόντος

```
const links = screen.getAllByRole("link");
        const productLink = links[1];
        expect(productLink.textContent.trim()).toContain(
            "Asus ROG Maximus Z790 Hero Wi-Fi Motherboard ATX με Intel 1700
Socket602.90542.61 €"
        );
```

Η συνάρτηση screen.getAllByRole("link") αναζητεί στο εικονικό DOM όλα τα στοιχεία που έχουν ρόλο "link", δηλαδή όλους τους συνδέσμους. Αυτή η συνάρτηση επιστρέφει μια λίστα με όλους τους

συνδέσμους που βρέθηκαν. Ο σύνδεσμος του προϊόντος αποθηκεύεται στην μεταβλητή productLink, που είναι ο δεύτερος σύνδεσμος από τη λίστα (θεωρούμε ότι ο σύνδεσμος του προϊόντος είναι ο δεύτερος στη σειρά). Η συνάρτηση expect(productLink.textContent.trim()).toContain(...) ελέγχει ότι το κείμενο του συνδέσμου περιέχει την αναμενόμενη περιγραφή και τιμή του προϊόντος. Αν ο έλεγχος περάσει, σημαίνει ότι ο σύνδεσμος του προϊόντος εμφανίζεται σωστά με την αναμενόμενη πληροφορία.

10.3 ΕΝΤΟΛΕΣ ΕΚΤΕΛΕΣΗΣ ΕΛΕΓΧΩΝ

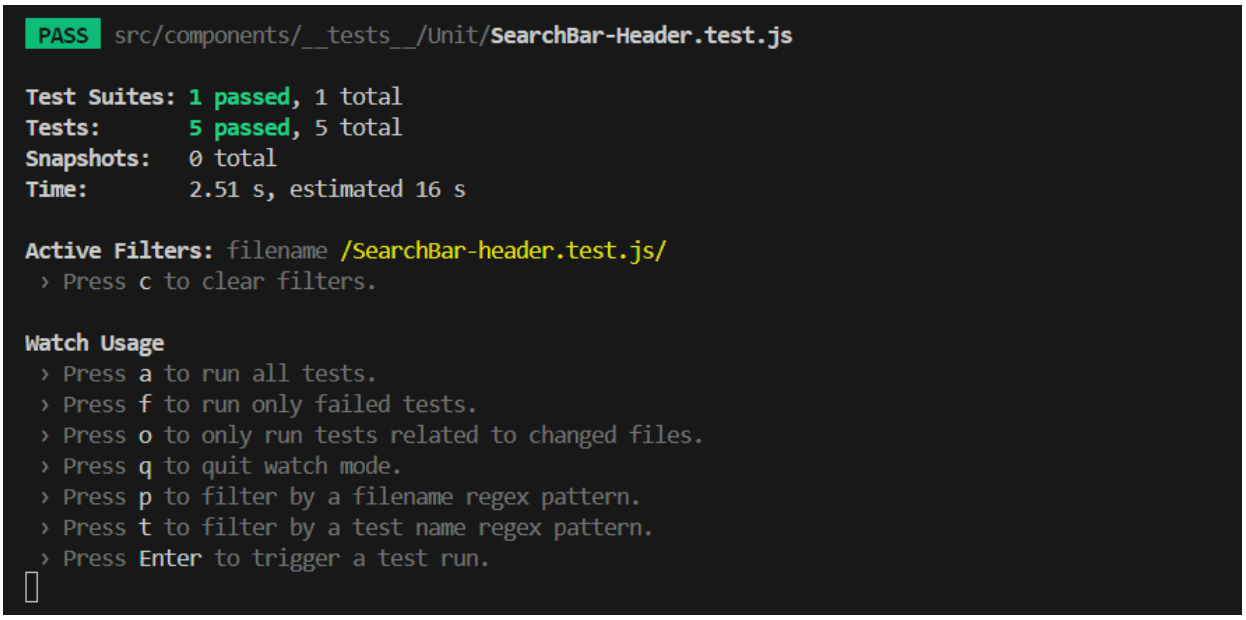
Βήμα 14: Τώρα είμαστε έτοιμοι να εκτελέσουμε τα τεστ. Ανοίξουμε τη γραμμή εντολών (terminal) στο του έργου μας και εκτελούμε την εντολή (αυτή η εντολή θα τρέξει μόνο το συγκεκριμένο αρχείο τεστ):

npm run test SearchBar-header.test.js

Εάν θέλουμε να εκτελέσουμε όλα τα τεστ, απλά χρησιμοποιούμε την εντολή:

npm test

10.4 ΑΝΑΛΥΣΗ ΑΠΟΤΕΛΕΣΜΑΤΩΝ ΕΛΕΓΧΩΝ



Εικόνα 10.1

Όταν εκτελούμε τα tests για ένα component, όπως φαίνεται στην εικόνα που παρείχες, βλέπουμε τα αποτελέσματα των δοκιμών με λεπτομέρεια. Στην περίπτωση του αρχείου SearchBar-header.test.js, τα αποτελέσματα δείχνουν ότι όλες οι δοκιμές εκτελέστηκαν επιτυχώς.

Το αποτέλεσμα "Test Suites: 1 passed, 1 total" σημαίνει ότι μία συλλογή τεστ, η οποία περιλαμβάνει όλες τις δοκιμές για αυτό το αρχείο, πέρασε επιτυχώς. Επιπλέον, το "Tests: 5 passed, 5 total" δείχνει ότι υπήρχαν συνολικά πέντε τεστ και όλα πέρασαν επιτυχώς, υποδεικνύοντας ότι όλες οι επιμέρους δοκιμές στο αρχείο SearchBar-header.test.js εκτελέστηκαν χωρίς προβλήματα.

Η ένδειξη "Snapshots: 0 total" δείχνει ότι δεν χρησιμοποιήθηκαν snapshots για αυτό το τεστ. Τα snapshots είναι χρήσιμα για την παρακολούθηση αλλαγών στο UI components, αλλά στην προκειμένη περίπτωση δεν φαίνεται να έχουν χρησιμοποιηθεί. Ο συνολικός χρόνος εκτέλεσης των τεστ ήταν 2.51 δευτερόλεπτα, ενώ η εκτιμώμενη διάρκεια ήταν 16 δευτερόλεπτα, δείχνοντας ότι τα τεστ εκτελέστηκαν πολύ πιο γρήγορα από το αναμενόμενο.

Το Jest παρέχει διάφορες επιλογές για την εκτέλεση και παρακολούθηση των τεστ. Μπορείτε να εκτελέσετε όλα τα τεστ, μόνο τα αποτυχημένα τεστ, τα τεστ που σχετίζονται με τα αρχεία που έχουν αλλάξει, ή να βγείτε από τη λειτουργία παρακολούθησης. Επίσης, μπορείτε να φιλτράρετε τα τεστ με βάση ένα regex pattern για τα ονόματα αρχείων ή τα ονόματα των τεστ και να εκτελέσετε ξανά τα τεστ με το πάτημα ενός κουμπιού.

ΚΕΦΑΛΑΙΟ 11: UI TESTING

11.1 ΣΥΝΔΕΣΗ ΣΤΗΝ ΕΦΑΡΜΟΓΗ

Οι δοκιμές αυτές εξασφαλίζουν ότι το component λειτουργεί όπως αναμένεται, λαμβάνοντας υπόψη την κατάσταση της εφαρμογής (store).

```
describe("Login component", () => {
    let store;

    beforeEach(() => {
        store = mockStore({
            authUser: {
                authLoading: false, // Assuming initial state for authLoading
            },
        });

        useSelector.mockImplementation((selector) =>
            selector(store.getState())
        );
    });
```

Ο κώδικας χρησιμοποιεί το describe μπλοκ για να ομαδοποιήσει τις δοκιμές που αφορούν το "Login component". Η μεταβλητή store δηλώνεται στο επάνω μέρος του describe και θα χρησιμοποιηθεί για να κρατήσει την κατάσταση του mock store. Η μέθοδος beforeEach εκτελείται πριν από κάθε δοκιμή, ρυθμίζοντας την κατάσταση του store. Η αρχική κατάσταση του store περιλαμβάνει ένα αντικείμενο authUser με την ιδιότητα authLoading αρχικοποιημένη σε false. Η useSelector.mockImplementation αντικαθιστά την πραγματική useSelector με μια mock υλοποίηση που επιστρέφει την κατάσταση από το mock store. Αυτή η δομή επιτρέπει τον έλεγχο του component χωρίς εξωτερικές εξαρτήσεις, διασφαλίζοντας αξιόπιστες και σαφείς δοκιμές.

11.1.1 Έλεγχος για κουμπί σύνδεσης

Έχει στόχο να διασφαλίσει ότι το φόρμα σύνδεσης (login form) εμφανίζεται σωστά στην οθόνη, συμπεριλαμβανομένων των πεδίων για την εισαγωγή email και κωδικού πρόσβασης, καθώς και του κουμπιού σύνδεσης. Είναι ένα βασικό τεστ για την επαλήθευση της αρχικής διεπαφής χρήστη της φόρμας σύνδεσης, εξασφαλίζοντας ότι όλα τα απαραίτητα στοιχεία είναι παρόντα και λειτουργικά.

```
test("renders login form correctly", async () => {
        const { getByPlaceholderText, getByText } = render(
            <Provider store={store}>
                <Login />
            </Provider>
        );

        // Check if email and password fields are rendered
        expect(getByPlaceholderText("Email")).toBeInTheDocument();
        expect(getByPlaceholderText("Password")).toBeInTheDocument();

        // Check if the "Log in" button is rendered
        expect(getByText("Log in")).toBeInTheDocument();
    });
```

Αφού το component έχει render, οι μέθοδοι getByPlaceholderText και getByText χρησιμοποιούνται για να εντοπίσουν τα στοιχεία της φόρμας εισόδου: τα πεδία "Email" και "Password" και το κουμπί "Log in". Οι συναρτήσεις expect ελέγχουν αν αυτά τα στοιχεία είναι παρόντα στο έγγραφο (DOM) χρησιμοποιώντας τη μέθοδο toBeInTheDocument. Αυτό το τεστ διασφαλίζει ότι τα βασικά στοιχεία της φόρμας εισόδου εμφανίζονται σωστά όταν το component Login κάνει render.

11.1.2 Έλεγχος ορατότητας κωδικού

για να ελέγξει ότι η ορατότητα του πεδίου κωδικού (password visibility) αλλάζει σωστά.

```
test("toggles password visibility correctly", async () => {
        const { getByLabelText, getByTestId } = render(
            <Provider store={store}>
                <Login />
            </Provider>
        );

        // Initially, password input type should be "password"
        const passwordInput = getByTestId("password-input");
        expect(passwordInput).toHaveAttribute("type", "password");
    });
```

Αφού το component έχει render, η συνάρτηση getByTestId χρησιμοποιείται για να εντοπίσει το πεδίο κωδικού μέσω του data-testid attribute, το οποίο υποτίθεται ότι είναι "password-input". Η συνάρτηση expect ελέγχει ότι το πεδίο κωδικού έχει το attribute type με τιμή "password", χρησιμοποιώντας τη μέθοδο toHaveAttribute. Αυτό το τεστ επιβεβαιώνει ότι το πεδίο κωδικού ξεκινά με την ορατότητα κωδικού κρυφή, κάτι που είναι μια βασική λειτουργία για τη διασφάλιση της ασφάλειας των χρηστών.

11.1.3 Έλεγχος υποβολής φόρμας εισόδου

για να ελέγξει τη σωστή υποβολή της φόρμας εισόδου. Περιλαμβάνει μια mock υλοποίηση του Firebase authentication.
Αυτό το τεστ επιβεβαιώνει ότι η φόρμα εισόδου λειτουργεί όπως αναμένεται, υποβάλλοντας τα σωστά δεδομένα μέσω της mock υλοποίησης του Firebase authentication.

```
jest.mock("firebase/app", () => ({
        auth: jest.fn(() => ({
            signInWithEmailAndPassword: jest.fn(() => Promise.resolve()),
        })),
    }));

    const middlewares = [thunk];
    const mockStore = configureStore(middlewares);

    test("submits login form correctly", async () => {
        store = mockStore({
            authUser: {
                authLoading: false,
            },
        });
        render(
```

```
            <Provider store={store}>
                <Login />
            </Provider>
        );

        const emailInput = screen.getByPlaceholderText("Email");
        const passwordInput = screen.getByPlaceholderText("Password");
        const submitButton = screen.getByText("Log in");

        fireEvent.change(emailInput, { target: { value: "test@example.com" } });
        fireEvent.change(passwordInput, { target: { value: "password123" } });
        fireEvent.change(submitButton);
    });
```

Αυτό το unit test ελέγχει τη σωστή υποβολή της φόρμας εισόδου στο "Login component". Η jest.mock δημιουργεί μια mock υλοποίηση του Firebase authentication, εξασφαλίζοντας ότι η μέθοδος signInWithEmailAndPassword επιστρέφει ένα επιλυμένο Promise.

Οι middleware ορίζονται με τη χρήση του thunk, και το mockStore δημιουργείται χρησιμοποιώντας το configureStore. Στο test μπλοκ, δημιουργείται το mock store με αρχική κατάσταση όπου το authLoading είναι false.

Τα πεδία email και password εντοπίζονται χρησιμοποιώντας το screen.getByPlaceholderText, και το κουμπί υποβολής εντοπίζεται με το screen.getByText.

11.2 ΕΓΓΡΑΦΗ ΣΤΗΝ ΕΦΑΡΜΟΓΗ

11.2.1 Έλεγχος εμφάνισης κουμπιών

για να ελέγξει ότι όλα τα πεδία εισαγωγής και τα κουμπιά του component εμφανίζονται σωστά. Αυτό το test διασφαλίζει ότι όλα τα απαραίτητα πεδία και κουμπιά του `Signup` component εμφανίζονται σωστά όταν το component κάνει render.

```
describe("Signup Component", () => {
    let store;

    beforeEach(() => {
        store = mockStore({
            authUser: {
                authLoading: false,
                // Add other relevant state properties here if needed
            },
        });
```

```
    });

    test("renders all input fields and buttons", () => {
        const { getByPlaceholderText, getByText } = render(
            <Provider store={store}>
                <Signup />
            </Provider>
        );

        // Check if all input fields are rendered
        expect(getByPlaceholderText("First Name *")).toBeInTheDocument();
        expect(getByPlaceholderText("Last Name *")).toBeInTheDocument();
        expect(getByPlaceholderText("Email *")).toBeInTheDocument();
        expect(getByPlaceholderText("Address *")).toBeInTheDocument();
        expect(getByPlaceholderText("Mobile Phone *")).toBeInTheDocument();
        expect(getByPlaceholderText("Password")).toBeInTheDocument();
        expect(getByPlaceholderText("Confirm Password")).toBeInTheDocument();

        // Check if buttons are rendered
        expect(getByText("Sign up")).toBeInTheDocument();
        expect(getByText("Back to Login")).toBeInTheDocument();
    });
```

Συγκεκριμένα, το test ελέγχει την παρουσία επτά πεδίων εισόδου: "First Name", "Last Name", "Email", "Address", "Mobile Phone", "Password" και "Confirm Password". Επίσης, ελέγχει την παρουσία δύο κουμπιών: "Sign up" και "Back to Login". Η μέθοδος expect επιβεβαιώνει ότι όλα αυτά τα στοιχεία είναι παρόντα στο έγγραφο (DOM) χρησιμοποιώντας τη μέθοδο toBeInTheDocument.

11.2.2 Έλεγχος ορατότητας κωδικού

Αυτό το UI test είναι γραμμένο για να ελέγξει τη λειτουργία ενός εικονιδίου (συνήθως ένα εικονίδιο ματιού) που χρησιμοποιείται για την εναλλαγή της ορατότητας ενός πεδίου εισαγωγής κωδικού πρόσβασης. Όταν το εικονίδιο είναι ενεργοποιημένο, ο κωδικός πρόσβασης θα πρέπει να είναι ορατός (τύπος πεδίου 'text'), και όταν είναι απενεργοποιημένο, ο κωδικός πρόσβασης θα πρέπει να είναι κρυφός (τύπος πεδίου 'password').

```
test("toggles password visibility when eye icon is clicked", () => {
        const { getByPlaceholderText, getByTestId } = render(
            <Provider store={store}>
                <Signup />
            </Provider>
```

```
        );

        const passwordInput = getByPlaceholderText("Password");
        const eyeIcon = getByTestId("password-input").nextSibling;

        fireEvent.click(eyeIcon); // Click eye icon to toggle password visibility

        expect(passwordInput.type).toBe("text"); // Password input type should be
'text' when visible

        fireEvent.click(eyeIcon); // Click eye icon again to toggle password
visibility

        expect(passwordInput.type).toBe("password"); // Password input type
should be 'password' when hidden
    });
});
```

Ρύθμιση του Περιβάλλοντος Δοκιμής: Το τεστ ξεκινάει με το render της Signup φόρμας μέσα σε έναν Provider του Redux store, προκειμένου να εξασφαλιστεί ότι η φόρμα έχει πρόσβαση στο απαραίτητο context.

Ανακάλυψη των Στοιχείων: Χρησιμοποιώντας τις μεθόδους getByPlaceholderText και getByTestId, ανακαλύπτουμε το πεδίο εισαγωγής κωδικού πρόσβασης και το εικονίδιο ματιού.

Πρώτο Κλικ στο Εικονίδιο Ματιού: Με το πρώτο fireEvent.click στο εικονίδιο, το τεστ ελέγχει αν ο τύπος του πεδίου κωδικού πρόσβασης έχει αλλάξει σε 'text', που σημαίνει ότι ο κωδικός πρόσβασης είναι ορατός.

Δεύτερο Κλικ στο Εικονίδιο Ματιού: Με το δεύτερο fireEvent.click, το τεστ ελέγχει αν ο τύπος του πεδίου κωδικού πρόσβασης έχει επιστρέψει σε 'password', που σημαίνει ότι ο κωδικός πρόσβασης είναι ξανά κρυφός.

Αυτό το UI test διασφαλίζει ότι η λειτουργία εναλλαγής ορατότητας του

11.3 ΠΛΗΡΟΦΟΡΙΕΣ ΠΡΟΙΟΝΤΩΝ

11.3.1 Δομή ελέγχου

Ο σκοπός του τεστ είναι να διασφαλίσει ότι το ProductDetails component εμφανίζει σωστά τα δεδομένα του προϊόντος, διαχειρίζεται σωστά την κατάσταση των αγαπημένων προϊόντων, και αλληλεπιδρά με τον χρήστη όπως αναμένεται.

```
describe("ProductDetails Component", () => {
    let store;
```

```
const initialState = {
    products: {
        singleProduct: {
            id: 1,
            title: "Sample Product",
            description: "<p>Sample Description</p>",
            isFav: false,
            price: 100,
            discount: 10,
            productCode: "SP123",
            deliveryDays: 3,
            images: ["sample1.jpg", "sample2.jpg"],
            selectedProduct: [{ value: 2 }],
        },
        favoritesProducts: [],
        allProducts: [
            {
                id: 2,
                title: "Recommended Product",
                images: ["rec1.jpg"],
                price: 50,
                productCode: "RP123",
            },
        ],
    },
    authUser: {
        user: { id: 1 },
    },
};
```

Το initialState περιέχει δύο κύρια αντικείμενα: products και authUser.

Το products περιέχει τα εξής:

singleProduct: Λεπτομέρειες του κύριου προϊόντος που θα προβληθεί στο ProductDetails component.

favoritesProducts: Μια λίστα με τα αγαπημένα προϊόντα του χρήστη, αρχικά κενή.

allProducts: Μια λίστα με όλα τα προϊόντα, περιλαμβάνοντας ένα προτεινόμενο προϊόν.

Το authUser περιέχει ένα αντικείμενο user με τα στοιχεία του χρήστη.

11.3.2 Έλεγχος εμφάνισης δεδομένων

Ο σκοπός αυτού του τεστ είναι να επαληθεύσει ότι το ProductDetails component αποδίδει σωστά τα δεδομένα του προϊόντος στην οθόνη. Συγκεκριμένα, ελέγχει αν ο τίτλος του προϊόντος, η τιμή μετά την έκπτωση, οι ημέρες παράδοσης, και η περιγραφή εμφανίζονται όπως αναμένεται.

```
it("renders ProductDetails component", () => {
        render(
            <Provider store={store}>
                <Router>
                    <ProductDetails />
                </Router>
            </Provider>
        );

        expect(screen.getByText("Sample Product")).toBeInTheDocument();
        expect(screen.getByText(/90/)).toBeInTheDocument(); // Price after
discount
        expect(screen.getByText(/Delivery Days/)).toBeInTheDocument();
        expect(screen.getByText("Sample Description")).toBeInTheDocument();
    });
```

Τίτλος Προϊόντος: Επαληθεύεται ότι το κείμενο "Sample Product" εμφανίζεται στο έγγραφο, επιβεβαιώνοντας την απόδοση του τίτλου του προϊόντος.

Τιμή μετά την Έκπτωση: Επαληθεύεται ότι η τιμή "90" (που προκύπτει μετά την εφαρμογή της έκπτωσης) εμφανίζεται στο έγγραφο. Αν η αρχική τιμή είναι 100 και η έκπτωση είναι 10%, η τελική τιμή θα είναι 90.

Ημέρες Παράδοσης: Επαληθεύεται ότι το κείμενο "Delivery Days" εμφανίζεται στο έγγραφο, επιβεβαιώνοντας την απόδοση των πληροφοριών για τις ημέρες παράδοσης.

Περιγραφή Προϊόντος: Επαληθεύεται ότι το κείμενο "Sample Description" εμφανίζεται στο έγγραφο, επιβεβαιώνοντας την απόδοση της περιγραφής του προϊόντος.

11.3.3 Έλεγχος για κουμπί προσθήκης στο καλάθι

Ο σκοπός αυτού του τεστ είναι να επαληθεύσει ότι όταν ο χρήστης κάνει κλικ στο κουμπί "Add to Cart", το προϊόν προστίθεται στο καλάθι με την κατάλληλη ποσότητα, και ότι η σχετική ενέργεια αποστέλλεται στο Redux store.

```
it("handles Add to Cart click", () => {
        render(
            <Provider store={store}>
                <Router>
                    <ProductDetails />
                </Router>
            </Provider>
```

```
        );

        fireEvent.click(screen.getByText("Add to Cart"));

        expect(store.dispatch).toHaveBeenCalledWith(
            addToCart({ ...initialState.products.singleProduct, quantity: 1 })
        );
    });
```

Προσομοίωση Κλικ στο Κουμπί "Add to Cart"

Η μέθοδος fireEvent.click χρησιμοποιείται για να προσομοιώσει ένα κλικ στο κουμπί "Add to Cart" που εμφανίζεται στην οθόνη. Αυτή η ενέργεια προσομοιώνει τη συμπεριφορά του χρήστη που προσθέτει το προϊόν στο καλάθι αγορών.

Έλεγχος Αποστολής Ενέργειας στο Redux Store

Χρησιμοποιείται η μέθοδος expect(store.dispatch).toHaveBeenCalledWith για να επαληθεύσει ότι η ενέργεια addToCart αποστέλλεται στο Redux store με το σωστό payload. Το payload περιλαμβάνει όλα τα δεδομένα του προϊόντος (singleProduct από το initialState) με προσθήκη της ιδιότητας quantity που ορίζεται σε 1, δηλαδή ότι ένα αντικείμενο του προϊόντος προστίθεται στο καλάθι.

11.3.4 Έλεγχος προτεινόμενων προϊόντων

Ο σκοπός αυτού του τεστ είναι να επαληθεύσει ότι το `ProductDetails` component εμφανίζει σωστά τη λίστα των προτεινόμενων προϊόντων και ότι τα στοιχεία αυτών των προϊόντων εμφανίζονται όπως αναμένεται.

```
it("displays recommended products", () => {
        render(
            <Provider store={store}>
                <Router>
                    <ProductDetails />
                </Router>
            </Provider>
        );

        expect(screen.getByText("Recommended Products")).toBeInTheDocument();
    });
```

Έλεγχος Απόδοσης Προτεινόμενων Προϊόντων:

Κείμενο "Recommended Products": Επαληθεύεται ότι το κείμενο "Recommended Products" εμφανίζεται στο έγγραφο, επιβεβαιώνοντας ότι η ενότητα των προτεινόμενων προϊόντων αποδίδεται σωστά.

ΚΕΦΑΛΑΙΟ 12: UNIT TESTS

12.1 ΕΠΙΚΟΙΝΩΝΙΑ ΜΕΣΩ EMAIL

12.1.1 Έλεγχος αποστολής μηνύματος επικοινωνίας

Συνολικά, αυτές οι δοκιμές παρέχουν πλήρη κάλυψη για το στοιχείο ContactUs, διασφαλίζοντας ότι λειτουργεί σωστά όταν οι χρήστες υποβάλλουν αιτήματα και ότι η φόρμα επαναφέρεται κατάλληλα μετά από αυτό. Αυτό βοηθά στη διατήρηση της αξιοπιστίας και της χρηστικότητας της δυνατότητας ContactUs εντός της εφαρμογής.

```
describe("ContactUs Component", () => {
  it("submits the form with correct data", async () => {
    const mockDispatch = jest.fn();

    // Mock store with basic implementation of getState and subscribe
    const store = {
      dispatch: mockDispatch,
      getState: () => ({}), // Empty state
      subscribe: () => {}, // Dummy subscribe function
    };

    const { getByLabelText, getByText } = render(
      <Provider store={store}>
        <ContactUs />
      </Provider>
    );

    fireEvent.change(getByLabelText("Name:"), {
      target: { value: "Kosmas" },
    });
    fireEvent.change(getByLabelText("Email:"), {
      target: { value: "Kosmasxxx@gmail.com" },
    });
    fireEvent.change(getByLabelText("Message:"), {
      target: { value: "test message" },
    });
```

```
    fireEvent.click(getByText("Submit"));

    await waitFor(() => {
      expect(mockDispatch).toHaveBeenCalled();
      expect(getByLabelText("Name:")).toBeInTheDocument();
      expect(getByLabelText("Email:")).toBeInTheDocument();
      expect(getByLabelText("Message:")).toBeInTheDocument();
```

Η δοκιμή αυτή επαληθεύει ότι η φόρμα επικοινωνίας (ContactUs) μπορεί να υποβληθεί με επιτυχία χρησιμοποιώντας σωστά δεδομένα. Αρχικά, δημιουργείται μια προσομοιωμένη συνάρτηση dispatch χρησιμοποιώντας το jest.fn(), προσομοιώνοντας την αποστολή ενεργειών στο κατάστημα Redux. Έπειτα, το στοιχείο ContactUs αναπαριστάται μέσω ενός Provider με ένα προσομοιωμένο κατάστημα Redux για τη δοκιμή του.

Στη συνέχεια, προσομοιώνεται η εισαγωγή χρήστη, αλλάζοντας τις τιμές των πεδίων εισαγωγής για το όνομα, το email και το μήνυμα χρησιμοποιώντας το fireEvent.change(). Αφού ολοκληρωθεί η εισαγωγή δεδομένων, εκτελείται ένα γεγονός κλικ στο κουμπί υποβολής και περιμένεται η ολοκλήρωση της ασύγχρονης διαδικασίας υποβολής μέσω του waitFor(). Τέλος, επιβεβαιώνεται ότι η προσομοιωμένη συνάρτηση dispatch έχει κληθεί, ενώ επιπλέον ελέγχεται ότι τα πεδία εισαγωγής για το όνομα, το email και το μήνυμα παραμένουν προσβάσιμα, αποτρέποντας τον πρόωρο καθαρισμό της φόρμας.

12.2 ΜΠΑΡΑ ΑΝΑΖΗΤΗΣΗΣ

12.2.1 Έλεγχος αρχικοποίησης κατάστασης Search bar

Αυτό το UI test εξασφαλίζει ότι το mock store δημιουργείται με συνεπή και σωστά δεδομένα πριν από κάθε δοκιμή. Η αρχικοποίηση του store με προκαθορισμένα δεδομένα επιτρέπει τη διεξαγωγή αξιόπιστων και επαναλήψιμων δοκιμών, παρέχοντας σταθερό πλαίσιο για τη δοκιμή των λειτουργιών που σχετίζονται με τα προϊόντα και τους χρήστες.

```
beforeEach(() => {
        store = mockStore({
            products: {
                productsLoading: false,
                searchValue: "",
                allProducts: [
                    {
                        category: {
                            label: "Pc Hardware",
                            value: "Pc Hardware",
                        },
                        createdAt: {
                            seconds: 1710168237,
```

```
                        nanoseconds: 241000000,
                    },
                    deliveryDays: "4-10",
                    description:
                        '<p>Βασικά χαρακτηριστικά:</p><p><strong
style="color: rgb(31, 31, 31);">• </strong>4 υποδοχές DDR5 Ram έως
7800(O.C.)</p><p><strong style="color:……..,
                    discount: "10",
                    id: "F0NrNaexUBaCIeU7iUmX",
                    images: [
                        "https://firebasestorage.googleapis.com/v0/b/turbot…=
media&token=7721dccf-39ec-4f68-af1d-d5d38b809bf3",
                        "https://firebasestorage.googleapis.com/v0/b/turbot…=
media&token=410d13c3-b9c5-4d7f-baf9-19dff8c42029",
                        "https://firebasestorage.googleapis.com/v0/b/turbot…=
media&token=c3ca13b9-24f9-44bd-aed5-6dd055017600",
                    ],
                    isDeleted: false,
                    isFav: false,
                    price: "602.90",
                    productCode: "38442875",
                    selectedProduct: [],
                    subCategory: {
                        value: "Motherboards",
                        label: "Motherboards",
                    },
                    title: "      Asus ROG Maximus Z790 Hero Wi-Fi
Motherboard ATX με Intel 1700 Socket",
                    updatedAt: {
                        seconds: 1711673121,
                        nanoseconds: 43000000,
                    },
                },
                {
                    label: "Pc Hardware",
                    value: "Pc Hardware",
                    createdAt: {
                        seconds: 1711625703,
                        nanoseconds: 41000000,
                    },
                    deliveryDays: "4-10",
                    description:
                        '<p><strong style="color: rgb(31, 31, 31);">•
</strong>Στροφές ανεμιστήρα: 2200 rpm…….,
                    discount: "10",
```

```
                        id: "Sjve3SoVmxJhGTUqXynU",
                        images: [
                            "https://firebasestorage.googleapis.com/v0/b/turbot…=
media&token=19282588-5f74-45cd-95a9-73f4635adf87",
                            "https://firebasestorage.googleapis.com/v0/b/turbot…=
media&token=bb611853-8acc-4b51-bd51-d30688dbef00",
                            "https://firebasestorage.googleapis.com/v0/b/turbot…=
media&token=8dc8a420-b5cc-47fe-ae30-d1d4d0298513",
                        ],
                        isDeleted: false,
                        isFav: false,
                        price: "399.00",
                        productCode: "45005904",
                        selectedProduct: [],
                        subCategory: {
                            value: "Liquid coolers",
                            label: "Liquid coolers",
                        },
                        title: "Asus ROG Ryujin III 360 ARGB Υδρόψυξη Επεξεργαστή
Τριπλού Ανεμιστήρα 120mm για Socket AM4/AM5/1700/1200/115x",
                        updatedAt: {
                            seconds: 1711626378,
                            nanoseconds: 472000000,
                        },
                    },
                ],
            },
            authUser: {
                user: null,
            },
            cartReducer: {
                items: [],
            },
        });
    });
```

Κάθε προϊόν περιλαμβάνει διάφορα χαρακτηριστικά όπως category, createdAt, deliveryDays, description, discount, id, images, isDeleted, isFav, price, productCode, selectedProduct, subCategory, title, και updatedAt. Αυτές οι πληροφορίες χρησιμοποιούνται στις δοκιμές για να επαληθευτεί η ορθή λειτουργία του κώδικα που σχετίζεται με τα προϊόντα.

Αρχική Κατάσταση Χρήστη:

Η ενότητα authUser περιέχει έναν user ο οποίος είναι null, υποδεικνύοντας ότι δεν υπάρχει χρήστης συνδεδεμένος, ενώ η ενότητα cartReducer περιέχει έναν κενό πίνακα items, υποδεικνύοντας ότι το καλάθι αγορών είναι άδειο.

12.2.2 Έλεγχος εμφάνισης μπάρας αναζήτησης

Αυτό το UI test αποσκοπεί στη δοκιμή της απόδοσης της μπάρας αναζήτησης στο Header component της εφαρμογής. Το τεστ ελέγχει αν η μπάρα αναζήτησης εμφανίζεται σωστά, αν μπορεί να δεχτεί είσοδο από τον χρήστη και αν τα αποτελέσματα της αναζήτησης εμφανίζονται σωστά.

```
it("should render search bar", async () => {
        store.getState().products.productsLoading = true;
        render(
            <Provider store={store}>
                <MemoryRouter>
                    <Header />
                </MemoryRouter>
            </Provider>
        );
        const searchBar = screen.getByPlaceholderText("Search product...");
        expect(searchBar).toBeTruthy();

        const searchInput = screen.getByPlaceholderText("Search product...");
        fireEvent.change(searchInput, { target: { value: "Asus" } });
        expect(searchInput.value).toBe("Asus");

        const links = screen.getAllByRole("link");
        const productLink = links[1];
        expect(productLink.textContent.trim()).toContain(
            "Asus ROG Maximus Z790 Hero Wi-Fi Motherboard ATX με Intel 1700
Socket602.90542.61 €"
        );
    });
```

Η αρχικοποίηση του καταστήματος ξεκινά με την κατάσταση productsLoading να τίθεται σε true για να προσομοιώσει τη φόρτωση προϊόντων. Στη συνέχεια, το Header component αποδίδεται μέσα σε έναν Provider του Redux store και έναν MemoryRouter, παρέχοντας τις απαραίτητες συνθήκες για το τεστ. Η μπάρα αναζήτησης ελέγχεται με τη χρήση του screen.getByPlaceholderText, η οποία εντοπίζει τη μπάρα μέσω του placeholder text "Search product...". Η συνάρτηση expect επιβεβαιώνει ότι η μπάρα αναζήτησης αποδίδεται σωστά στο DOM.

Ακολούθως, ελέγχεται η είσοδος στη μπάρα αναζήτησης. Η screen.getByPlaceholderText χρησιμοποιείται ξανά για να βρεθεί η μπάρα αναζήτησης και η συνάρτηση fireEvent.change προσομοιώνει την αλλαγή της τιμής της μπάρας σε "Asus". Η συνάρτηση expect επιβεβαιώνει ότι η τιμή της μπάρας αναζήτησης ενημερώνεται σωστά σε "Asus".

Τέλος, η απόδοση των αποτελεσμάτων αναζήτησης ελέγχεται με τη χρήση του screen.getAllByRole, η οποία εντοπίζει όλους τους συνδέσμους (links) στο DOM. Ο δεύτερος σύνδεσμος από τη λίστα των

συνδέσμων επιλέγεται και η συνάρτηση expect επιβεβαιώνει ότι το κείμενο του συνδέσμου περιέχει τα αναμενόμενα δεδομένα του προϊόντος που αντιστοιχεί στην αναζήτηση.

12.2.3 Έλεγχος συμπεριφοράς μπάρας αναζήτησης

Αυτό το UI test αποσκοπεί στη δοκιμή της συμπεριφοράς της μπάρας αναζήτησης στο Header component όταν δεν βρίσκονται προϊόντα που να αντιστοιχούν στην αναζήτηση. Το τεστ ελέγχει αν η μπάρα αναζήτησης αποδίδεται σωστά, αν μπορεί να δεχτεί είσοδο από τον χρήστη, και αν δεν εμφανίζονται προϊόντα όταν η αναζήτηση δεν επιστρέφει αποτελέσματα.

```
it("should render nothing when no products are found", async () => {
        render(
            <Provider store={store}>
                <MemoryRouter>
                    <Header />
                </MemoryRouter>
            </Provider>
        );
        const searchBar = screen.getByPlaceholderText("Search product...");
        expect(searchBar).toBeTruthy();

        const searchInput = screen.getByPlaceholderText("Search product...");
        fireEvent.change(searchInput, { target: { value: " NO PRODUCTTTT

" } });
        expect(searchInput.value).toBe("NO PRODUCTTTT

");

        const links = screen.getAllByRole("link");
        const productLink = links[1];
        expect(productLink.textContent).not.toContain("NO PRODUCTTTT

");
    });
```

Έλεγχος Απόδοσης της Μπάρας Αναζήτησης:

Το screen.getByPlaceholderText χρησιμοποιείται για να βρεθεί η μπάρα αναζήτησης στο DOM μέσω του placeholder text "Search product...". Η συνάρτηση expect επιβεβαιώνει ότι η μπάρα αναζήτησης αποδίδεται σωστά.

Έλεγχος Εισόδου στη Μπάρα Αναζήτησης: Τ

ο screen.getByPlaceholderText χρησιμοποιείται ξανά για να βρεθεί η μπάρα αναζήτησης. Η συνάρτηση fireEvent.change χρησιμοποιείται για να προσομοιώσει την αλλαγή της τιμής της μπάρας αναζήτησης

σε " NO PRODUCTTTT". Έπειτε, η συνάρτηση expect επιβεβαιώνει ότι η τιμή της μπάρας αναζήτησης ενημερώνεται σωστά σε " NO PRODUCTTTT".

Έλεγχος Απόδοσης Αποτελεσμάτων Αναζήτησης:

Το screen.getAllByRole χρησιμοποιείται για να βρεθούν όλοι οι σύνδεσμοι (links) στο DOM. Επιλέγεται ο δεύτερος σύνδεσμος από τη λίστα των συνδέσμων. Το expect επιβεβαιώνει ότι το κείμενο του συνδέσμου δεν περιέχει τα δεδομένα της αναζήτησης " NO PRODUCTTTT", επιβεβαιώνοντας ότι δεν εμφανίζονται αποτελέσματα.

12.2.4 Mock κουμπιών

Αυτό το κομμάτι κώδικα χρησιμοποιεί τη βιβλιοθήκη jest για να κάνει mock διάφορες εξαρτήσεις που χρησιμοποιούνται από το Header component. Η τεχνική του mocking χρησιμοποιείται εδώ για να αντικαταστήσει τις πραγματικές εξαρτήσεις με απλά mock components, έτσι ώστε τα τεστ να επικεντρώνονται στη λειτουργικότητα του Header component χωρίς να εξαρτώνται από την υλοποίηση αυτών των εξαρτήσεων.

Ο σκοπός αυτού του mocking είναι:

Να αποφευχθεί η πολυπλοκότητα και οι πιθανές εξωτερικές επιρροές των πραγματικών εξαρτήσεων κατά τη διάρκεια των τεστ. Να βελτιωθεί η ταχύτητα και η αξιοπιστία των τεστ κάνοντας mock τις εξαρτήσεις που δεν είναι άμεσα σχετικές με το αντικείμενο του τεστ. Να εξασφαλιστεί ότι τα τεστ επικεντρώνονται αποκλειστικά στη λειτουργικότητα του Header component.

```
// Mock the dependencies that the Header component relies on
jest.mock("react-icons/io5", () => ({
    IoMenu: () => <div>IoMenu</div>,
    IoPersonOutline: () => <div>IoPersonOutline</div>,
    IoIosArrowForward: () => <div>IoIosArrowForward</div>,
    IoIosArrowDown: () => <div>IoIosArrowDown</div>,
    IoArrowUpCircleOutline: () => <div>IoArrowUpCircleOutline</div>,
}));
jest.mock("react-icons/bs", () => ({
    BsCart3: () => <div>BsCart3</div>,
}));
jest.mock("react-icons/rx", () => ({
    RxCross1: () => <div>RxCross1</div>,
}));
jest.mock("react-icons/fi", () => ({
    FiMenu: () => <div>FiMenu</div>,
}));
jest.mock("react-icons/lu", () => ({
    LuLogOut: () => <div>LuLogOut</div>,
}));
```

```
jest.mock("react-icons/md", () => ({
    MdDeleteForever: () => <div>MdDeleteForever</div>,
}));
jest.mock("reactstrap", () => ({
    Button: ({ children, ...props }) => <button {...props}>{children}</button>,
}));
jest.mock("../../Global/ContactLayer", () => () => <div>ContactLayer</div>);
jest.mock("../../header/MenuSidebar", () => {
    const React = require("react");
    return React.forwardRef((props, ref) => <div ref={ref}>MenuSidebar</div>);
});
jest.mock("../../../views/Signup", () => () => <div>Signup</div>);
jest.mock("../../../views/Login", () => () => <div>Login</div>);
```

Αυτό αποσκοπεί στη διασφάλιση ότι τα τεστ του Header component δεν θα επηρεαστούν από την πολυπλοκότητα ή την υλοποίηση αυτών των custom components.

12.3 ΕΠΙΚΕΦΑΛΙΔΑ ΕΦΑΡΜΟΓΗΣ

12.3.1 Έλεγχος εμφάνισης της επικεφαλίδας

Ο στόχος του τεστ είναι να επαληθεύσει ότι το Header component αποδίδεται σωστά όταν χρησιμοποιείται το αρχικό state της εφαρμογής.

```
test("renders Header component with initial state", () => {
        render(
            <Provider store={store}>
                <MemoryRouter initialEntries={["/"]}>
                    <Header />
                </MemoryRouter>
            </Provider>
        );

        expect(screen.getByAltText("logo")).toBeInTheDocument;

        expect(
            screen.getByPlaceholderText("Search product...")
        ).toBeInTheDocument();

        expect(screen.getByText("MENU")).toBeInTheDocument();

        expect(screen.getByText("ContactLayer")).toBeInTheDocument;
    });
```

Έλεγχος για το Λογότυπο:

expect(screen.getByAltText("logo")).toBeInTheDocument();

Αυτό επαληθεύει ότι υπάρχει ένα στοιχείο με το εναλλακτικό κείμενο "logo" στο Header component.

Έλεγχος για τη Γραμμή Αναζήτησης:

expect(screen.getByPlaceholderText("Search product...")).toBeInTheDocument();

Επαληθεύει ότι υπάρχει ένα στοιχείο γραμμής αναζήτησης με το placeholder text "Search product...".

Έλεγχος για το Κουμπί Μενού:

expect(screen.getByText("MENU")).toBeInTheDocument();

Επαληθεύει ότι υπάρχει ένα στοιχείο με το κείμενο "MENU".

Έλεγχος για το ContactLayer:

expect(screen.getByText("ContactLayer")).toBeInTheDocument();

Επαληθεύει ότι υπάρχει ένα στοιχείο με το κείμενο "ContactLayer", το οποίο είναι το mocked component για το ContactLayer.

12.3.2 Έλεγχος προϊόντων στο καλάθι

Στόχος του τεστ είναι να επιβεβαιώσει ότι το Header component απεικονίζει σωστά τα αντικείμενα του καλαθιού αγορών όταν υπάρχει τουλάχιστον ένα αντικείμενο στο καλάθι.

```
test("renders cart items correctly", () => {
        store = mockStore({
            cartReducer: {
                items: [
                    {
                        id: 1,
                        title: "Product 1",
                        price: "20.00",
                        discount: "10",
                        images: ["image1.jpg"],
                        quantity: 2,
                    },
                ],
            },
            authUser: {
```

```
                uid: null,
            },
            products: {
                allProducts: [],
            },
        });

        const { getByText } = render(
            <Provider store={store}>
                <MemoryRouter initialEntries={["/"]}>
                    <Header />
                </MemoryRouter>
            </Provider>
        );
    });
```

Ρύθμιση του Μοκαρισμένου Καταστήματος (beforeEach):

Η configureStore από το redux-mock-store χρησιμοποιείται για να δημιουργήσει ένα μοκαρισμένο κατάστημα Redux (store) με αρχική κατάσταση που αντικατοπτρίζει την πραγματική δομή του Redux καταστήματος σας. Αυτό περιλαμβάνει ένα αντικείμενο στο cartReducer με λεπτομέρειες όπως id, title, price, discount, images, και quantity.

Απεικόνιση του Component:

Το Header component απεικονίζεται μέσα σε ένα Provider με το μοκαρισμένο store τυλιγμένο σε ένα MemoryRouter (από το react-router-dom). Αυτή η ρύθμιση προσομοιώνει το περιβάλλον όπου το Header component θα υπάρχει στην εφαρμογή σας.

Επιβεβαιώσεις:

Χρησιμοποιώντας τη συνάρτηση getByText από το @testing-library/react, επιβεβαιώνουμε ότι τα στοιχεία που αντιστοιχούν στον τίτλο του προϊόντος, την τιμή και την ποσότητα είναι παρόντα στο απεικονιζόμενο Header component. Προσαρμόστε αυτές τις επιβεβαιώσεις ανάλογα με την πραγματική υλοποίηση και απεικόνιση του Header component σας.

12.4 ΚΑΛΑΘΙ ΕΦΑΡΜΟΓΗΣ

12.4.1 Έλεγχος βασικών ενεργειών καλαθιού

Ο σκοπός αυτών των τεστ είναι να διασφαλιστεί ότι το cartReducer λειτουργεί σωστά για τις βασικές ενέργειες που διαχειρίζονται το καλάθι αγορών, όπως η αρχική κατάσταση, η προσθήκη προϊόντος στο καλάθι, και η αφαίρεση προϊόντος από το καλάθι.

```
describe("cartReducer", () => {

    const initialState = {
        items: [],
    };

    it("should return the initial state", () => {
        expect(cartReducer(undefined, {})).toEqual(initialState);
    });

    it("should handle ADD_TO_CART", () => {
        const product = { id: 1, name: "Product 1" };
        const addAction = { type: "ADD_TO_CART", payload: product };
        const expectedState = {
            items: [product],
        };

        expect(cartReducer(initialState, addAction)).toEqual(expectedState);
    });

    it("should handle REMOVE_FROM_CART", () => {
        const initialStateWithItems = {
            items: [
                { id: 1, name: "Product 1" },
                { id: 2, name: "Product 2" },
            ],
        };
        const removeAction = { type: "REMOVE_FROM_CART", payload: 1 };
        const expectedState = {
            items: [{ id: 2, name: "Product 2" }],
        };

        expect(cartReducer(initialStateWithItems, removeAction)).toEqual(
            expectedState
        );
    });
```

Έλεγχος Αρχικής Κατάστασης:

Η δοκιμή it("should return the initial state") ελέγχει ότι όταν το cartReducer καλείται με undefined κατάσταση και μια κενή ενέργεια, επιστρέφει την προκαθορισμένη αρχική κατάσταση (initialState).

Έλεγχος Προσθήκης στο Καλάθι:

Η δοκιμή it("should handle ADD_TO_CART") ελέγχει ότι όταν το cartReducer λαμβάνει μια ενέργεια τύπου ADD_TO_CART με ένα προϊόν ως payload, επιστρέφει τη νέα κατάσταση του καλαθιού που περιλαμβάνει το προστιθέμενο προϊόν και Ορίζεται ένα προϊόν (product) και μια ενέργεια προσθήκης (addAction) και η αναμενόμενη κατάσταση (expectedState) περιλαμβάνει το προϊόν στη λίστα items. Ύστερα, επαληθεύεται ότι το cartReducer επιστρέφει την αναμενόμενη κατάσταση όταν καλείται με την αρχική κατάσταση και την ενέργεια προσθήκης.

Έλεγχος Αφαίρεσης από το Καλάθι:

Η δοκιμή it("should handle REMOVE_FROM_CART") ελέγχει ότι όταν το cartReducer λαμβάνει μια ενέργεια τύπου REMOVE_FROM_CART με το id ενός προϊόντος ως payload, επιστρέφει τη νέα κατάσταση του καλαθιού που δεν περιλαμβάνει το προϊόν με το συγκεκριμένο id. Ορίζεται μια αρχική κατάσταση με δύο προϊόντα (initialStateWithItems) και Ορίζεται μια ενέργεια αφαίρεσης (removeAction). Η αναμενόμενη κατάσταση (expectedState) περιλαμβάνει μόνο το προϊόν που δεν αφαιρέθηκε.Ύστερα, επαληθεύεται ότι το cartReducer επιστρέφει την αναμενόμενη κατάσταση όταν καλείται με την κατάσταση που περιέχει τα δύο προϊόντα και την ενέργεια αφαίρεσης.

12.4.2 Έλεγχος διαγραφής προϊόντος

Αυτά τα τεστ αφορούν τον έλεγχο πρόσθετων λειτουργιών του cartReducer. Συγκεκριμένα, δοκιμάζονται οι ενέργειες εκκαθάρισης του καλαθιού, διαγραφής προϊόντος από το καλάθι με βάση τη θέση, και αποφυγής διπλότυπων προσθηκών.

```
it("should handle CLEAR_CART", () => {
        const initialStateWithItems = {
            items: [
                { id: 1, name: "Product 1" },
                { id: 2, name: "Product 2" },
            ],
        };
        const clearAction = { type: "CLEAR_CART" };
        const expectedState = {
            items: [],
        };

        expect(cartReducer(initialStateWithItems, clearAction)).toEqual(
            expectedState
        );
    });

    it("should handle DELETE_FROM_CART", () => {
        const initialStateWithItems = {
            items: [
```

```
                { id: 1, name: "Product 1" },
                { id: 2, name: "Product 2" },
            ],
        };
        const deleteAction = {
            type: "DELETE_FROM_CART",
            payload: { index: 0 },
        };
        const expectedState = {
            items: [{ id: 2, name: "Product 2" }],
        };

        expect(cartReducer(initialStateWithItems, deleteAction)).toEqual(
            expectedState
        );
    });

    it("should not add duplicate items on ADD_TO_CART", () => {
        const initialStateWithItems = {
            items: [{ id: 1, name: "Product 1" }],
        };
        const addAction = {
            type: "ADD_TO_CART",
            payload: { id: 1, name: "Product 1" },
        };
        const expectedState = {
            items: [{ id: 1, name: "Product 1" }],
        };

        expect(cartReducer(initialStateWithItems, addAction)).toEqual(
            expectedState
        );
    });
```

Έλεγχος Εκκαθάρισης του Καλαθιού:

Η δοκιμή it("should handle CLEAR_CART") ελέγχει ότι όταν το cartReducer λαμβάνει μια ενέργεια τύπου CLEAR_CART, επιστρέφει μια κατάσταση με άδειο καλάθι. Ορίζεται μια αρχική κατάσταση με δύο προϊόντα (initialStateWithItems) και ορίζεται μια ενέργεια εκκαθάρισης (clearAction). Η αναμενόμενη κατάσταση (expectedState) έχει άδειο το items array. Επαληθεύεται ότι το cartReducer επιστρέφει την αναμενόμενη κατάσταση όταν καλείται με την αρχική κατάσταση και την ενέργεια εκκαθάρισης.

Έλεγχος Διαγραφής από το Καλάθι βάσει Θέσης:

Η δοκιμή it("should handle DELETE_FROM_CART") ελέγχει ότι όταν το cartReducer λαμβάνει μια ενέργεια τύπου DELETE_FROM_CART με ένα index ως payload, επιστρέφει μια νέα κατάσταση που δεν περιλαμβάνει το προϊόν στη συγκεκριμένη θέση.

Έλεγχος Αποφυγής Διπλότυπων Προσθηκών:

Η δοκιμή it("should not add duplicate items on ADD_TO_CART") ελέγχει ότι όταν το cartReducer λαμβάνει μια ενέργεια τύπου ADD_TO_CART με ένα προϊόν που ήδη υπάρχει στο καλάθι, η κατάσταση του καλαθιού δεν αλλάζει και το διπλότυπο προϊόν δεν προστίθεται.

Επαληθεύεται ότι το cartReducer επιστρέφει την αναμενόμενη κατάσταση όταν καλείται με την αρχική κατάσταση και την ενέργεια προσθήκης.

12.5 ΕΝΕΡΓΕΙΕΣ ΠΡΟΙΟΝΤΩΝ

12.5.1 Έλεγχος ανάκτησης προϊόντος

Αυτό το UI test στοχεύει να επαληθεύσει ότι η ενέργεια (action) getProducts αποστέλλει τις σωστές ενέργειες στο Redux store όταν ανακτά τα προϊόντα από μια mock βάση δεδομένων Firebase Firestore. Το τεστ εξασφαλίζει ότι η αλληλεπίδραση με το Firestore είναι σωστή και ότι οι αναμενόμενες ενέργειες αποστέλλονται με τα σωστά δεδομένα.

```
test("getProducts dispatches ALL_PRODUCTS action with products", async () => {
        const products = [
            { id: "1", title: "Product 1" },
            { id: "2", title: "Product 2" },
        ];

        const collectionMock = {
            onSnapshot: jest.fn((callback) => {
                callback({
                    docs: products.map((product) => ({
                        id: product.id,
                        exists: true,
                        data: () => product,
                    })),
                });
            }),
        };

        firebase.firestore.mockReturnValue({
            collection: jest.fn(() => collectionMock),
```

```
        });

        const expectedActions = [
            { type: "PRODUCTS_LOADING", payload: true },
            {
                type: "ALL_PRODUCTS",
                payload: products.map((product) => ({
                    ...product,
                    isFav: false,
                })),
            },
            { type: "PRODUCTS_LOADING", payload: false },
        ];

        await store.dispatch(actions.getProducts());

        expect(store.getActions()).toEqual(expectedActions);
        expect(firebase.firestore().collection).toHaveBeenCalledWith(
            "products"
        );
    });
```

Mock Επιστροφή Firestore:

Το firebase.firestore.mockReturnValue επιστρέφει ένα mock αντικείμενο Firestore που περιλαμβάνει τη συλλογή collectionMock.

Αναμενόμενες Ενέργειες (Expected Actions):

Ορίζεται μια λίστα expectedActions με τις αναμενόμενες ενέργειες που θα αποσταλούν στο Redux store: PRODUCTS_LOADING, ALL_PRODUCTS, PRODUCTS_LOADING

Εκτέλεση και Έλεγχος Ενεργειών:

Καλείται η ενέργεια getProducts με τη χρήση του store.dispatch. Χρησιμοποιείται η expect για να ελέγξει ότι οι ενέργειες που στάλθηκαν (store.getActions()) είναι ίσες με τις expectedActions.Επιπλέον, ελέγχεται ότι η κλήση στη συλλογή Firestore έγινε με το σωστό όνομα ("products").

12.5.2 Έλεγχος ανάκτησης αγαπημένων προιόντων

Αυτό το UI τεστ αποσκοπεί να επαληθεύσει ότι η ενέργεια (action) getFavorites αποστέλλει την σωστή ενέργεια στο Redux store όταν ανακτά τα αγαπημένα προϊόντα ενός συγκεκριμένου χρήστη από μια

mock βάση δεδομένων Firebase Firestore. Το τεστ εξασφαλίζει ότι η αλληλεπίδραση με το Firestore είναι σωστή και ότι οι αναμενόμενες ενέργειες αποστέλλονται με τα σωστά δεδομένα.

```
test("getFavorites dispatches FAVORITE_PRODUCTS action with favorite products",
async () => {
        const uid = "user1";
        const favoriteProducts = [
            { id: "1", title: "Product 1" },
            { id: "2", title: "Product 2" },
        ];

        const collectionMock = {
            where: jest.fn().mockReturnThis(),
            onSnapshot: jest.fn((callback) => {
                callback({
                    docs: favoriteProducts.map((product) => ({
                        id: product.id,
                        exists: true,
                        data: () => product,
                    })),
                });
            }),
        };

        firebase.firestore.mockReturnValue({
            collection: jest.fn(() => collectionMock),
        });

        const expectedActions = [
            { type: "FAVORITE_PRODUCTS", payload: favoriteProducts },
        ];

        await store.dispatch(actions.getFavorites(uid));

        expect(firebase.firestore().collection).toHaveBeenCalledWith(
            "favoriteProducts"
        );
        expect(firebase.firestore().collection().where).toHaveBeenCalledWith(
            "userID",
            "==",
            uid
        );
    });
```

Δημιουργία Mock Δεδομένων:

Ορίζεται το uid του χρήστη και τα mock δεδομένα αγαπημένων προϊόντων (favoriteProducts) με id και τίτλο. Η collectionMock χρησιμοποιείται για να μιμηθεί τη συλλογή Firestore. Περιλαμβάνει τις μεθόδους where και onSnapshot που θα καλούνται κατά την εκτέλεση του κώδικα.

Η onSnapshot καλείται με μια συνάρτηση επανάκλησης που παρέχει τα mock δεδομένα ως έγγραφα.

Εκτέλεση και Έλεγχος Ενεργειών:

Καλείται η ενέργεια getFavorites με τη χρήση του store.dispatch και το uid του χρήστη. Χρησιμοποιείται η expect για να ελέγξει ότι η κλήση στη συλλογή Firestore έγινε με το σωστό όνομα ("favoriteProducts"). Ελέγχεται επίσης ότι η κλήση where στο Firestore έγινε με το σωστό φίλτρο ("userID", "==", uid).

12.5.3 Έλεγχος ανάκτησης προϊόντος με id

Αυτό το UI τεστ αποσκοπεί να επαληθεύσει ότι η ενέργεια (action) getSingleProduct αποστέλλει την σωστή ενέργεια στο Redux store όταν ανακτά ένα συγκεκριμένο προϊόν από μια mock βάση δεδομένων Firebase Firestore. Το τεστ εξασφαλίζει ότι η αλληλεπίδραση με το Firestore είναι σωστή και ότι οι αναμενόμενες ενέργειες αποστέλλονται με τα σωστά δεδομένα.

```
test("getSingleProduct dispatches SINGLE_PRODUCT action with product", async ()
=> {
        const product = { id: "1", title: "Product 1" };

        const docMock = {
            get: jest.fn().mockResolvedValue({
                id: product.id,
                exists: true,
                data: () => product,
            }),
        };

        const collectionMock = {
            doc: jest.fn(() => docMock),
        };

        firebase.firestore.mockReturnValue({
            collection: jest.fn(() => collectionMock),
        });

        const expectedActions = [
            { type: "SINGLE_PRODUCT", payload: { ...product, isFav: false } },
        ];
```

```
        await store.dispatch(actions.getSingleProduct(product.id));

        expect(store.getActions()).toEqual(expectedActions);
        expect(firebase.firestore().collection).toHaveBeenCalledWith(
            "products"
        );
        expect(firebase.firestore().collection().doc).toHaveBeenCalledWith(
            product.id
        );
    });
```

Εκτέλεση και Έλεγχος Ενεργειών:

Καλείται η ενέργεια getSingleProduct με τη χρήση του store.dispatch και το id του προϊόντος. Χρησιμοποιείται η expect για να ελέγξει ότι οι ενέργειες που στάλθηκαν (store.getActions()) είναι ίσες με τις expectedActions. Ελέγχεται επίσης ότι η κλήση στη συλλογή Firestore έγινε με το σωστό όνομα ("products"). Επιπλέον, ελέγχεται ότι η κλήση στο έγγραφο της συλλογής Firestore έγινε με το σωστό id του προϊόντος.

12.5.4 Έλεγχος προσθήσης/αφαίρεσης στα αγαπημένα ενος προϊόντος

Αυτό το UI τεστ αποσκοπεί να επαληθεύσει ότι η ενέργεια (action) addToFavorite προσθέτει ή αφαιρεί σωστά ένα αγαπημένο προϊόν για έναν συγκεκριμένο χρήστη στη βάση δεδομένων Firebase Firestore. Το τεστ εξασφαλίζει ότι η αλληλεπίδραση με το Firestore είναι σωστή και ότι οι σωστές κλήσεις γίνονται ανάλογα με το αν το προϊόν είναι ήδη στα αγαπημένα ή όχι.

```
test("addToFavorite adds/removes favorite product", async () => {
        const uid = "user1";
        const productId = "1";

        // Mocking the Firestore chain calls for add
        const whereMock = jest.fn().mockReturnThis();
        const getMock = jest
            .fn()
            .mockResolvedValueOnce({
                empty: true,
                forEach: jest.fn(),
            })
            .mockResolvedValueOnce({
                empty: false,
                forEach: (callback) => callback({ ref: { delete: jest.fn() } }),
            });

        const collectionMock = {
            where: whereMock,
```

```
            add: jest.fn(),
            get: getMock,
        };

        firebase.firestore.mockReturnValue({
            collection: jest.fn(() => collectionMock),
        });

        await store.dispatch(actions.addToFavorite(uid, productId));

        expect(firebase.firestore().collection).toHaveBeenCalledWith(
            "favoriteProducts"
        );
        expect(firebase.firestore().collection().where).toHaveBeenCalledWith(
            "userID",
            "==",
            uid
        );
        expect(firebase.firestore().collection().where).toHaveBeenCalledWith(
            "productID",
            "==",
            productId
        );
        expect(firebase.firestore().collection().add).toHaveBeenCalledWith({
            userID: uid,
            productID: productId,
        });

        await store.dispatch(actions.addToFavorite(uid, productId));

        expect(
            firebase.firestore().collection().where().get
        ).toHaveBeenCalledTimes(2);
    });
```

Δημιουργία Mock Δεδομένων:

Ορίζεται το uid του χρήστη και το productId του προϊόντος. Το getMock χρησιμοποιείται για να μιμηθεί την κλήση get της συλλογής Firestore, η οποία επιστρέφει δύο διαφορετικές καταστάσεις: Πρώτη φορά: Η συλλογή είναι κενή (empty: true), που σημαίνει ότι το προϊόν δεν είναι στα αγαπημένα. Δεύτερη φορά: Η συλλογή δεν είναι κενή (empty: false), που σημαίνει ότι το προϊόν είναι ήδη στα αγαπημένα και καλείται η μέθοδος delete για να αφαιρεθεί.

Ύστερα, το firebase.firestore.mockReturnValue επιστρέφει ένα mock αντικείμενο Firestore που περιλαμβάνει τη συλλογή collectionMock και τις μεθόδους where, get, και add.

Πρώτη Κλήση:

Καλείται η ενέργεια addToFavorite με τη χρήση του store.dispatch και τα uid και productId. Ελέγχεται ότι η κλήση στη συλλογή Firestore έγινε με το σωστό όνομα ("favoriteProducts") και τα σωστά φίλτρα ("userID", "==", uid και "productID", "==", productId). Ελέγχεται επίσης ότι η μέθοδος add της συλλογής Firestore καλείται με τα σωστά δεδομένα ({ userID: uid, productID: productId }).

Δεύτερη Κλήση:

Καλείται ξανά η ενέργεια addToFavorite με τα ίδια δεδομένα.

Ελέγχεται ότι η μέθοδος get της συλλογής Firestore καλείται δύο φορές συνολικά.

12.6 ΕΜΦΑΝΙΣΗ ΠΡΟΙΟΝΤΩΝ

12.6.1 Έλεγχος ένδειξης φόρτωσης

Το unit test που περιγράφεται έχει ως σκοπό να διαπιστώσει εάν το Product Component εμφανίζει σωστά ένδειξη φόρτωσης ("loading indicator") όταν τα προϊόντα βρίσκονται σε κατάσταση φόρτωσης. Αυτό επιτυγχάνεται με την ρύθμιση της κατάστασης φόρτωσης των προϊόντων σε true και τον έλεγχο για την παρουσία του κατάλληλου στοιχείου UI που δηλώνει φόρτωση στην οθόνη.

```
it('renders loading indicator when products are loading', () => {
    // Mock products loading state to true
    store.getState().products.productsLoading = true;

    render(
      <Provider store={store}>
        <MemoryRouter>
          <Product />
        </MemoryRouter>
      </Provider>
    );

    // Verify that loading indicator is rendered
    expect(screen.getByRole('status')).toBeInTheDocument();
    expect(screen.getByText('Loading...')).toBeInTheDocument();
  });
```

Έλεγχος Παρουσίας Στοιχείου Φόρτωσης: Χρησιμοποιείται η expect σε συνδυασμό με την getByRole για να επιβεβαιώσει ότι το στοιχείο UI με ρόλο 'status' (τυπικά αντιπροσωπεύει τα στοιχεία φόρτωσης) είναι παρόν στο DOM. Επιπλέον, επαληθεύεται με την getByText ότι το κείμενο 'Loading...' εμφανίζεται επίσης στο DOM.

Αυτό το test εξασφαλίζει ότι το component ανταποκρίνεται σωστά σε συνθήκες φόρτωσης, παρέχοντας στον χρήστη ένδειξη ότι δεδομένα φορτώνονται και επιβεβαιώνοντας ότι η εφαρμογή διατηρεί καλή εμπειρία χρήστη ακόμα και κατά τη διάρκεια καθυστερήσεων στη φόρτωση περιεχομένου.

12.6.2 Έλεγχος απεικόνισης προϊόντων

Αυτό το unit test σχεδιάζεται για να διασφαλίσει ότι το Product Component απεικονίζει σωστά τα προϊόντα όταν αυτά είναι διαθέσιμα στο store. Ειδικότερα, το test θα επικεντρωθεί στην επιβεβαίωση της ύπαρξης του στοιχείου με το test ID "product-code" στο DOM, υποδηλώνοντας ότι το προϊόν και οι λεπτομέρειες του φορτώνονται και εμφανίζονται κατάλληλα.

```
it('renders products correctly when products are available', () => {
    store.getState().products.productsLoading = true;

    console.log('geeee')
    console.log(screen, 'sceeen')
    const {getByTestId} = render(
      <Provider store={store}>
        <MemoryRouter>
          <Product />
        </MemoryRouter>
      </Provider>
    );

    expect(screen.getByTestId('product-code')).toBeInTheDocument();
  });
```

Ρύθμιση Κατάστασης Φόρτωσης: Στην αρχή του test, η κατάσταση φόρτωσης των προϊόντων στο Redux store ορίζεται λανθασμένα ως true. Αυτό θα έπρεπε να αλλαχθεί σε false για να μιμηθεί την κατάσταση όπου τα προϊόντα είναι φορτωμένα και έτοιμα για εμφάνιση.

Χρήση Κονσόλας για Debugging: Εισάγονται εντολές console.log για debugging, οι οποίες ίσως δεν είναι απαραίτητες στην τελική μορφή του test και μπορούν να αφαιρεθούν για καθαρότερο κώδικα.

Διεξαγωγή Τεστ: Το component Product απεικονίζεται μέσω του MemoryRouter και του Provider με το Redux store. Στη συνέχεια, χρησιμοποιείται η getByTestId για να αναζητηθεί το στοιχείο με test ID 'product-code'.

Έλεγχος Εμφάνισης Στοιχείου: Η expect συνδυάζεται με getByTestId για να επαληθευτεί ότι το στοιχείο με ID 'product-code' είναι παρόν στο DOM, υποδηλώνοντας την σωστή απεικόνιση των προϊόντων.

12.6.3 Έλεγχος απεικόνισης προϊόντος στις κατηγορίες

Αυτό το unit test σχεδιάζεται για να διασφαλίσει ότι το σύστημα απεικονίζει σωστά τα προϊόντα για μια συγκεκριμένη κατηγορία, με βάση τη διαδρομή που δίνεται στο URL. Συγκεκριμένα, το test προσομοιώνει την πλοήγηση στη διαδρομή /Computers/Laptops για να ελέγξει αν τα συγκεκριμένα προϊόντα φορτώνονται και απεικονίζονται από το σχετικό component.

```
it('renders products for a specific category', () => {
    render(
      <Provider store={store}>
        <MemoryRouter initialEntries={['/Computers/Laptops']}>
          <Switch>
            <Route path="/:cat/:sub">
            </Route>
          </Switch>
        </MemoryRouter>
      </Provider>
    );

  });
```

Ρυθμίσεις Περιβάλλοντος: Το test περιβάλλον στήνεται με τη χρήση του Provider για να παρέχει το Redux store και του MemoryRouter για να προσομοιώσει την πλοήγηση σε ένα συγκεκριμένο URL.

Πλοήγηση URL: Το initialEntries ρυθμίζεται σε ['/Computers/Laptops'] για να προσομοιώσει την πρόσβαση του χρήστη σε αυτή την κατηγορία μέσω του URL.

Διαχείριση Διαδρομής: Χρησιμοποιείται ο Route με path="/:cat/:sub" για να αντιστοιχίσει την URL διαδρομή στις παραμέτρους cat (κατηγορία) και sub (υποκατηγορία), επιτρέποντας την απεικόνιση στοιχείων από το συγκεκριμένο path.

12.7 ΕΙΚΟΝΕΣ ΑΝΑΚΑΤΕΥΘΥΝΣΗΣ

```
const middlewares = [thunk];
const mockStore = configureStore(middlewares);

const initialState = {
    products: {
        allProducts: [
            {
                id: 1,
                name: "Laptop 1",
                img: "path/to/image",
                value: "/sort/computers/laptops",
```

```
                category: { value: "computers" },
                discount: 10,
                isFav: false,
            },
        ],
        productsLoading: false,
        favoritesProducts: [],
        searchValue: "",
    },
    authUser: {
        user: null,
    },
};
```

Ορισμός Middleware:

Το thunk middleware χρησιμοποιείται για να επιτρέπει την αποστολή ασύγχρονων ενέργειων (actions) στον Redux store.

Ορίζοντας το "path/to/image" στην ιδιότητα img του προϊόντος, διασφαλίζουμε ότι οι εικόνες των προϊόντων φορτώνονται σωστά και εμφανίζονται στους χρήστες της εφαρμογής, προσδιορίζοντας τη σωστή διαδρομή πρόσβασης από τις εικόνες στις αντίστοιχες κατηγορίες προϊόντων που έχουμε διαλέξει κατα την διάρκεια συγγραφής του ελέγχου.

12.7.1 Έλεγχος πλοήγησης

Ο σκοπός αυτού του τεστ είναι να ελέγξει ότι η πλοήγηση από την αρχική σελίδα προϊόντων ("Home") σε μια σελίδα κατηγορίας προϊόντων ("computers/laptops") λειτουργεί σωστά κατά την κλικ σε μια κάρτα προϊόντος που αντιστοιχεί σε ένα laptop ("Laptop 1").

```
it("navigates to category pages on card click", async () => {
        render(
            <Provider store={store}>
                <Router history={history}>
                    <Home key={Home} />
                </Router>
            </Provider>
        );

        // Assuming there's an element with the text "Laptop 1" in the Home
component
        const laptopsCard = await screen.findByText("Laptop 1");
        fireEvent.click(laptopsCard);
```

```
        await waitFor(() => {
            expect(history.location.pathname).toBe("/sort/computers/laptops");
        });
    });
```

Κλικ στην Κάρτα "Laptop 1":

Η findByText ψάχνει για το κείμενο "Laptop 1" στον αρχικό κώδικα (Home component) και με τη χρήση της fireEvent.click, κάνει κλικ στην κάρτα που περιέχει το κείμενο "Laptop 1".

Αναμονή για Πλοήγηση και Έλεγχος Διαδρομής:

Η waitFor χρησιμοποιείται για να περιμένει μέχρι να αλλάξει η διαδρομή και γίνεται έλεγχος με τη χρήση του expect για να ελέγξει αν η τρέχουσα διαδρομή (history.location.pathname) είναι αυτή που αναμένεται ("/sort/computers/laptops").

12.8 ΕΛΕΓΧΟΙ FIREBASE

Αυτά τα τεστ διασφαλίζουν ότι η αρχικοποίηση του Firebase πραγματοποιείται σωστά και ότι η αρχικοποιημένη εφαρμογή Firebase είναι διαθέσιμη για χρήση σε όλη την εφαρμογή. Παρέχουν εμπιστοσύνη στην ρύθμιση του Firebase και βοηθούν στην αποτροπή πιθανών προβλημάτων που σχετίζονται με την αρχικοποίηση του Firebase στην εφαρμογή.

```
describe('Firebase Initialization', () => {
  it('initializes Firebase app with the provided configuration', () => {
    expect(firebase).toEqual(firebaseConfig);
  });

  it('exports the initialized Firebase app', () => {
    expect(firebase).toBeDefined();
  });
});
```

επαληθεύει εάν η εφαρμογή Firebase αρχικοποιείται με την παρεχόμενη διαμόρφωση. Συγκρίνει το αρχικοποιημένο αντικείμενο Firebase (firebase) με την αναμενόμενη διαμόρφωση (firebaseConfig) χρησιμοποιώντας το matcher toEqual. Εάν το αντικείμενο Firebase ταιριάζει με την παρεχόμενη διαμόρφωση, υποδεικνύει ότι το Firebase έχει αρχικοποιηθεί σωστά με την αναμενόμενη διαμόρφωση.

Το επόμενο τεστ ελέγχει αν η αρχικοποιημένη εφαρμογή Firebase εξάγεται. Επιβεβαιώνει ότι το αντικείμενο firebase είναι ορισμένο, υποδεικνύοντας ότι η εφαρμογή Firebase έχει αρχικοποιηθεί και εξαχθεί με επιτυχία. Εάν το αντικείμενο firebase είναι ορισμένο, διασφαλίζει ότι η αρχικοποιημένη εφαρμογή Firebase είναι προσβάσιμη για χρήση σε άλλα μέρη της εφαρμογής.

12.9 ΕΛΕΓΧΟΙ STRIPE

12.9.1 Φόρμα πληρωμής

Συνολικά, αυτό το τεστ διασφαλίζει ότι το στοιχείο StripeCheckout αποδίδει σωστά τα στοιχεία της φόρμας, συμπεριλαμβανομένου του πεδίου εισαγωγής email, και ρυθμίζει το περιβάλλον για τη δοκιμή της υποβολής πληρωμής.

```
describe('StripeCheckout', () => {
  test('renders form elements and handles payment submission', async () => {
    // Mock handlePayment function
    const handlePayment = jest.fn();

    const stripePromise = loadStripe('your_stripe_public_key');

    render(
      <Elements stripe={stripePromise}> {/* Wrap StripeCheckout with Elements
provider */}
        <StripeCheckout handlePayment={handlePayment} bill={100} />
      </Elements>
    );


  const emailInput = screen.getByPlaceholderText('Enter the email');
  expect(emailInput).toBeInTheDocument();
  });
```

Επαληθεύει ότι το στοιχείο StripeCheckout αποδίδει σωστά τα στοιχεία της φόρμας και διαχειρίζεται την υποβολή πληρωμής. Ξεκινάει με τη δημιουργία μιας πλαστής λειτουργίας handlePayment χρησιμοποιώντας το jest.fn(). Αυτό μας επιτρέπει να προσομοιώσουμε και να παρακολουθούμε τις κλήσεις σε αυτή τη λειτουργία κατά τη διάρκεια της δοκιμής.

Στη συνέχεια, αρχικοποιεί το Stripe χρησιμοποιώντας τη συνάρτηση loadStripe με μια τιμή-πλασίμπο για το δημόσιο κλειδί του Stripe. Το στοιχείο StripeCheckout αποδίδεται έπειτα μέσα σε έναν πάροχο Elements από το πακέτο @stripe/react-stripe-js. Αυτός ο πάροχος είναι απαραίτητος για τη χρήση των στοιχείων του Stripe μέσα στο στοιχείο.

Μέσα στο αποδοθέν στοιχείο, εντοπίζει το πεδίο εισαγωγής email με το κείμενο του placeholder χρησιμοποιώντας τη μέθοδο screen.getByPlaceholderText που παρέχεται από το πακέτο @testing-library/react. Τέλος, επιβεβαιώνει ότι το πεδίο εισαγωγής email είναι παρόν στο αποδοθέν στοιχείο ελέγχοντας αν είναι στο έγγραφο (toBeInTheDocument()).

12.9.2 Αποτυχία πληρωμής

Διασφαλίζει ότι το στοιχείο StripeCheckout χειρίζεται σωστά τις αποτυχίες πληρωμής εμφανίζοντας ένα μήνυμα σφάλματος. Βοηθά στην επαλήθευση της συμπεριφοράς διαχείρισης σφαλμάτων του στοιχείου και εξασφαλίζει μια ομαλή εμπειρία χρήστη σε περίπτωση προβλημάτων πληρωμής.

```
test('displays error message if payment fails', async () => {
    // Mock handlePayment function to simulate payment failure
    const handlePayment = jest.fn().mockImplementationOnce(() => {
      throw new Error('Payment failed');
    });

    const stripePromise = loadStripe('your_stripe_public_key');

    render(
      <Elements stripe={stripePromise}> {/* Wrap StripeCheckout with Elements
provider */}
        <StripeCheckout handlePayment={handlePayment} bill={100} />
      </Elements>
    );


  const emailInput = screen.getByPlaceholderText('Enter the email');
  expect(emailInput).toBeInTheDocument();
  });
```

Επαληθεύει ότι το στοιχείο StripeCheckout εμφανίζει μήνυμα σφάλματος αν αποτύχει η πληρωμή. Ξεκινάει με τη δημιουργία μιας πλαστής λειτουργίας handlePayment για την προσομοίωση αποτυχίας πληρωμής. Η μέθοδος mockImplementationOnce χρησιμοποιείται για να ρίξει ένα σφάλμα όταν κληθεί η συνάρτηση.

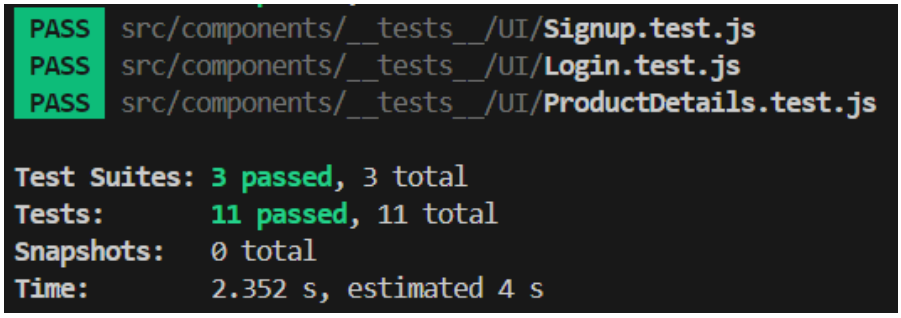
Όπως και στην προηγούμενο τεστ, το Stripe αρχικοποιείται χρησιμοποιώντας το loadStripe με μια τιμή-πλασίμπο για το δημόσιο κλειδί. Το στοιχείο StripeCheckout αποδίδεται μέσα σε έναν πάροχο Elements, όπως και στην προηγούμενη δοκιμή.

Τέλος, εντοπίζει το πεδίο εισαγωγής email μέσα στο αποδοθέν στοιχείο χρησιμοποιώντας τη μέθοδο screen.getByPlaceholderText. Τέλος, επιβεβαιώνει ότι το πεδίο εισαγωγής email είναι παρόν στο αποδοθέν στοιχείο ελέγχοντας αν είναι στο έγγραφο (toBeInTheDocument()).

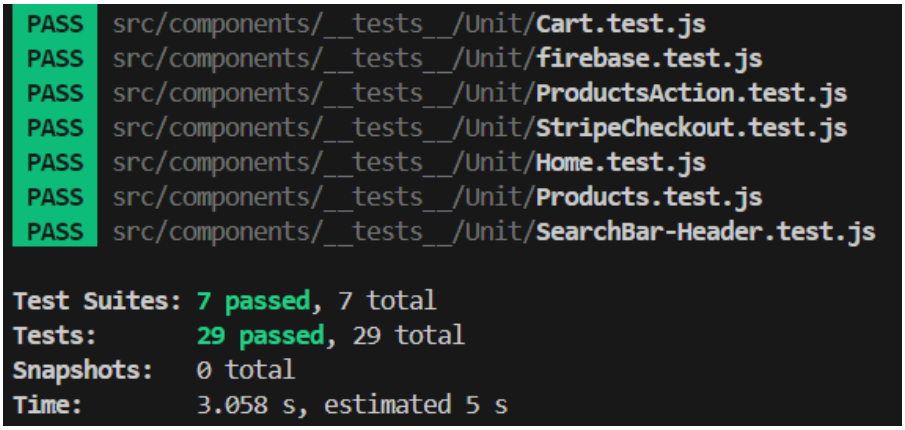
12.10 ΣΥΝΟΛΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ ΕΛΕΓΧΩΝ ΕΦΑΡΜΟΓΗΣ.

Παρακάτω παρατηρούμε τα συνολικά αποτελέσματα όλων των ελέγχων που πραγματοποιήθηκαν.

UI TESTING



UNIT TESTING



ΚΕΦΑΛΑΙΟ 13: ΟΔΗΓΟΣ ΕΚΤΕΛΕΣΗΣ ΕΦΑΡΜΟΓΗΣ

ΕΠΙΛΟΓΗ 1η :

Βήμα 1: Εγκατάσταση Node.js και NVM ακολουθώντας τα βήματα μέσω των επίσημων ιστότοπων.

(Θα χρειαστεί και η εντολή -> npm install --legacy-peer-deps)

Βήμα 2: Χρησιμοποιούμε την version node 16 ή την 20 της node.js με την εντολή -> nvm use 16 ή nvm use

Βήμα 3: Εγκατάσταση της Tailwind Css ακολουθώντας τα βήματα μέσω του επίσημου ιστότοπου.

Βήμα 4: Εκτέλεση της εφαρμογής με την εντολή -> npm start

ΕΠΙΛΟΓΗ 2η :

Μεσω των διαθέσιμων link:

ESHOP https://turbotech-eshop.web.app/

ADMIN PANEL https://turbotech-admin.web.app/

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Ferrera, Cécile; Kessedjian, Eowyne Article Evolution of E-commerce and Global Marketing

[2] INTUIT Mailchimp What is E-commerce?

[3] (Πατσά, 2005; Παυλίδης και Λέανδρος, 2004; Turban et al., 2006; Burshy, 2005; Ραχανιώτου και Ατζάμπου, 1998)

[4] Drew S. Strategic uses of e-commerce by SMEs in the east of England. Eur Manage J 2003;2

[5] Unit Testing Explained: What It Is, Why It's Important, and How to Get Started, April 20, 2023GEEKFLARE

[6] Yash Jain, Crafting Exceptional React.js Code: A Guide to Optimization and Scalability

[7] Juan Carlos Arias Ambriz, Emulating React and JSX in Vanilla JS

[8] React -Redux, Medium, by Raja Rohan Ray

[9] Understanding Tailwind CSS, Medium, by Christo

[10] Gotapi, How Does API Work and How to Use it Effectively?

[11] Stripe API keys – Everything you need to know by wpfullpay

[12] Nathan Peck, Microservice Testing: Unit Tests

[13] Chiradeep BasuMallick, What Is Unit Testing? Types, Tools, and Best Practices

[14] Katalon Unit Testing vs Integration Testing: Key differences

[15] Chiradeep BasuMallick, What Is Unit Testing? Types, Tools, and Best Practices

[16] Unit test Frameworks O.REILLY by Paul Hamill

[17] Kamala Joshi, How Unit Testing Benefits Your Projects: A Comprehensive Analysis

[18] Investing in Unit Testing: Benefits and Approaches by Toptal Developers

[19] Gui testing by UTOR. What, Why, How?

[20] UI Testing: Beginners Guide With Examples and Best Practices by Lambdatest

[21] UI Testing: A Detailed Guide By Shreya Bose, Community Contributor

[22] UI Testing / Visual Testing Definition, Types, and Advantages by resonlio.

[23] UI Testing Fundamentals including UI Test Types, Automating UI Tests, and UI Testing Error Checklist Alastair Wilkes

[24] Jest framework for React unit testing by Lelianto Eko Pradana Lelianto Eko Pradana

[25] Jest Testing: A Helpful, Introductory Tutorial By Testim

[26] React Testing Library vs Jest Adarsh Pandya Simform Engineering by Adarsh Pandya

[27] Fullstack React: The Complete Guide to ReactJS and Friends by, Anthony Accomazzo, Nathaniel Murray, Ari Lerner

[28] Learn React for Free by Scrimba

[29] Tailwind CSS Documentation by Taildind CSS Docs

[30] Testing React Applications with React Testing Library by Kent C. Dodds