ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και Ανάπτυξη κατανεμημένου συστήματος ελέγχου

Γεώργιος Χατζηευστρατίου
Α.Μ. 22026

Εισηγητής: Ιωάννης Βογιατζής

(Κενό φύλλο)

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

ΤΙΤΛΟΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Γεώργιος Χατζηευστρατίου
Α.Μ. 22026

Εισηγητής:

Εξεταστική Επιτροπή:

IOANNIS VOYIATZIS
Digitally signed by IOANNIS VOYIATZIS
Date: 2025.01.24 10:05:22 +02'00'

STAVROS FATOUROS
Digitally signed by STAVROS FATOUROS
Date: 2025.01.24 11:14:49 +02'00'

Digitally signed by STYLIANOS VOUTSINAS
Date: 2025.01.24 20:20:19 +02'00'

Ημερομηνία εξέτασης

(Κενό φύλλο)

ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΜΕΤΑΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

«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».

Ο Δηλών

Γ. Χατζηευστρατίου

Γιώργος Χατζηευστρατιου

(Κενό φύλλο)

ΕΥΧΑΡΙΣΤΙΕΣ

Η παρούσα διπλωματική εργασία ολοκληρώθηκε μετά από επίμονες προσπάθειες, σε ένα ενδιαφέρον γνωστικό αντικείμενο, όπως αυτό του σχεδιασμού και ανάπτυξης κατανεμημένου συστήματος έλεγχου. Την προσπάθειά μου αυτή υποστήριξε ο επιβλέπων καθηγητής μου, τον οποίο θα ήθελα να ευχαριστήσω.
Ακόμα θα ήθελα να ευχαριστήσω την οικογένειά μου για τη συμπαράσταση κατά τη διάρκεια των σπουδών μου.

(Κενό φύλλο)

ΕΠΙΣΤΗΜΟΝΙΚΗ ΠΕΡΙΟΧΗ: Μηχανική Λογισμικού για Κατανεμημένα Συστήματα , Ανάπτυξη Εφαρμογών, Αρχιτεκτονική Νέφους, Αρχιτεκτονική Εφαρμογών
ΛΕΞΕΙΣ ΚΛΕΙΔΙΑ: AWS, Java, Angular, Cloud Computing, Cloud Architecture, AWS Architecture, RESTful APIs, Microservices, Διαχείριση Εγγραφών, Συστήματα Αποθήκευσης Δεδομένων, Docker, Kubernetes, Δεδομένα στο cloud, Μηχανική Μάθηση, Agile, DevOps, Continuous Integration

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΑΤΑΛΟΓΟΣ ΕΙΚΟΝΩΝ

Πινακες

ΠΕΡΙΕΧΟΜΕΝΑ ΚΩΔΙΚΑ

ABSTRACT

The shift to digital technologies has created a pressing need for new methods of control and management across academic, business, and banking sectors. This thesis focuses on the design and development of a distributed control system that addresses key concerns like security, efficiency, and flexibility in these diverse fields.

The primary objective of this research is to build a control framework tailored to the specific operational needs of universities, businesses, and banks. By doing so, it aims to improve process efficiency, safeguard data, and enhance the overall management of resources.

The system architecture incorporates modern tools such as AWS for cloud infrastructure, Spring for backend development, Angular for the frontend, Terraform for infrastructure automation, and Docker for containerization. Together, these technologies enable a scalable and secure system, suitable for the demands of today's digital landscape.

Ultimately, the development of this advanced distributed control system is a critical step toward enhancing functionality, security, and sustainability in various sectors, ensuring they remain competitive in the evolving digital age.

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

1.1 Σκοπός της διπλωματικής εργασίας

Σκοπός αυτής της εργασίας είναι ο σχεδιασμός και η ανάπτυξη ενός κατανεμημένου συστήματος ελέγχου, το οποίο θα επιτρέπει στους χρήστες να δημιουργούν, να επεξεργάζονται και να διαχειρίζονται διαφορετικούς τύπους αρχείων μέσω ενός διαδικτυακού περιβάλλοντος. Το σύστημα θα λειτουργεί χωρίς τη χρήση τοπικής εγκατάστασης εξειδικευμένου λογισμικού, απαιτώντας μόνο ένα πρόγραμμα περιήγησης και σύνδεση στο διαδίκτυο. Αντικείμενο της παρούσας διπλωματικής εργασίας είναι η ανάπτυξη ενός κατανεμημένου συστήματος ελέγχου χρησιμοποιώντας AWS Services, Angular και Java. Σκοπός είναι να παρουσιάσω μια λύση, που δείχνει πώς η συνδυασμένη χρήση αυτών των τεχνολογιών μπορεί να βελτιώσει την απόδοση των εφαρμογών σε αυτόν τον τομέα καθώς και πως τα διαφορετικά εργαλεία γλώσσες προγραμματισμού Services κτλ. βοηθούν στην ανάπτυξή, καθώς και τη χρήση κατανεμημένων συστημάτων ελέγχου σαν εργαλείο στην καθημερινή εργασία.

Βήματα Υλοποίησης του Συστήματος

Ανάλυση Τεχνολογιών Συστήματος:

- Μελετάμε και καταγράφουμε τις τεχνολογίες συστήματος, για να κατανοήσουμε τις υπάρχουσες λύσεις και τις δυνατότητες, που προσφέρουν.

Ανάλυση Απαιτήσεων Χρηστών:

- Συλλέγουμε και αναλύουμε τις απαιτήσεις των χρηστών, για να καθορίσουμε τις βασικές και πρόσθετες λειτουργίες, που πρέπει να υποστηρίζει το σύστημα.

Σχεδιασμός Αρχιτεκτονικής Συστήματος:

- Σχεδιάζουμε την αρχιτεκτονική του συστήματος, συμπεριλαμβάνοντας τα κατανεμημένα στοιχεία, τις διασυνδέσεις τους και τα πρωτόκολλα επικοινωνίας.

Σχεδιασμός Οθονών Συστήματος:

- Δημιουργούμε και σχεδιάζουμε τις διεπαφές χρήστη (UI), διασφαλίζοντας ότι η εφαρμογή είναι εύχρηστη και λειτουργική. Για την ανάπτυξη του συστήματος, θα χρησιμοποιηθεί η αντικειμενοστραφή μεθοδολογία ανάπτυξης λογισμικού και θα δημιουργηθούν UML διαγράμματα για την ανάλυση και τεκμηρίωση της εφαρμογής.

Επίσης, θα εξεταστεί η χρήση τεχνολογιών για την υποστήριξη της κατανεμημένης αρχιτεκτονικής. Το σύστημη που θα αναπτυχθεί θα λειτουργεί διαδικτυακά για τη δημιουργία και διαχείριση αρχείων και όχι μόνο. Αν και η αρχική έκδοση του εργαλείου θα έχει περιορισμένο αριθμό λειτουργικών απαιτήσεων λόγω του περιορισμένου χρόνου ανάπτυξης, θα παρέχει τις βασικές δυνατότητες που απαιτούνται για την αποθήκευση και επεξεργασία αρχείων καθώς και τα βασικά για επεξεργασία χρήστη. Στο μέλλον, το σύστημα μπορεί να επεκταθεί με πρόσθετα στοιχεία για την ενίσχυση των λειτουργιών του. Οι χρήστες με σύνδεση στο διαδίκτυο θα έχουν τη δυνατότητα να χρησιμοποιήσουν την εφαρμογή, να δημιουργήσουν λογαριασμούς και να διαχειρίζονται τα αρχεία τους μέσω του συστήματος. Επιπλέον, θα υπάρχει ένας διαχειριστής υπεύθυνος για την καλή λειτουργία της εφαρμογής, την υποστήριξη των χρηστών και την παραμετροποίηση των δυνατοτήτων του συστήματος.

Κεφάλαιο 2 Κατανεμημένα συστήματα ελέγχου

2.1 Ιστορική αναδρομή κατανεμημένα συστήματα ελέγχου

Τα μη κατανεμημένα συστήματα ελέγχου έχουν μια μακρά και σημαντική ιστορία, αποτελώντας τη βάση για τη διαχείριση και τον έλεγχο δεδομένων και λειτουργιών σε πολλούς τομείς.

Πρώιμα Χρόνια: 1940s-1950s

Στα πρώτα χρόνια της μηχανογράφησης, τα συστήματα ελέγχου ήταν απλά και βασίζονταν κυρίως σε μηχανικούς και ηλεκτρομηχανικούς υπολογιστές. Οι υπολογιστές εκείνης της εποχής ήταν τεράστιοι και εξαιρετικά ακριβοί, με περιορισμένες δυνατότητες. Ο έλεγχος και η διαχείριση δεδομένων γινόταν αποκλειστικά από έναν κεντρικό υπολογιστή, που συντόνιζε τα πάντα.

Η Εμφάνιση των Mainframe Υπολογιστών: 1960s-1970s

Με την ανάπτυξη των mainframe υπολογιστών, τα κεντρικά συστήματα ελέγχου έγιναν πιο ισχυρά.
Αυτά τα συστήματα μπορούσαν πλέον να διαχειρίζονται μεγαλύτερους όγκους δεδομένων και πιο πολύπλοκες διαδικασίες. Τα mainframes κυριαρχούσαν στις μεγάλες επιχειρήσεις και τους κυβερνητικούς οργανισμούς, παρέχοντας έναν κεντρικό τρόπο ελέγχου όλων των δεδομένων και λειτουργιών.

Η Άνοδος των Mii Υπολογιστών και της Δικτύωσης: 1970s-1980s

Η έλευση των mini υπολογιστών έφερε επανάσταση στον τρόπο, που λειτουργούσαν οι επιχειρήσεις. Πλέον, και μικρότερες επιχειρήσεις μπορούσαν να υιοθετήσουν κεντρικά συστήματα ελέγχου. Παράλληλα, η δικτύωση των υπολογιστών άρχισε να διαδίδεται, αλλά ο έλεγχος παρέμενε κεντρικός. Αυτή την περίοδο, η αυτοματοποίηση μέσω κεντρικών συστημάτων ελέγχου έγινε κοινός τόπος, ειδικά σε τομείς όπως η παραγωγή και οι τηλεπικοινωνίες.

Οι Προσωπικοί Υπολογιστές και η Client-Server Αρχιτεκτονική: 1980s-1990s

Με την εξάπλωση των προσωπικών υπολογιστών και την ανάπτυξη της αρχιτεκτονικής client-server, τα μη κατανεμημένα συστήματα ελέγχου πέρασαν σε ένα νέο στάδιο. Τώρα, ο κεντρικός έλεγχος μπορούσε να συνδυαστεί με αποκεντρωμένους πελάτες (clients), αλλά η κεντρική βάση δεδομένων εξακολουθούσε να είναι το επίκεντρο του συστήματος. Οι εταιρείες συνέχισαν να βασίζονται σε κεντρικά συστήματα για την ασφάλεια και τη διαχείριση των δεδομένων τους.

Διαδίκτυο και Ψηφιακός Μετασχηματισμός: 1990s-2000s

Η εξάπλωση του Διαδικτύου άλλαξε το παιχνίδι. Οι κεντρικές υποδομές ελέγχου έγιναν ακόμα πιο σημαντικές, ειδικά για τον χειρισμό ψηφιακών υπηρεσιών όπως οι τραπεζιτικές συναλλαγές και το ηλεκτρονικό εμπόριο. Τα συστήματα αυτά έπρεπε να είναι σε θέση να διαχειριστούν μεγαλύτερο όγκο δεδομένων και να προσφέρουν αυξημένη διαθεσιμότητα και ασφάλεια.

Σύγχρονη Εποχή και η Μετάβαση στα Κατανεμημένα Συστήματα: 2000s-σήμερα

Παρόλο που τα κεντρικά συστήματα ελέγχου παραμένουν σημαντικά, η αυξανόμενη ανάγκη για ανθεκτικότητα, ασφάλεια και αποκέντρωση οδήγησε στην ανάπτυξη κατανεμημένων συστημάτων. Οι σύγχρονες αρχιτεκτονικές, όπως το cloud computing και το blockchain, προσφέρουν λύσεις, που ξεπερνούν τους περιορισμούς των παραδοσιακών μη κατανεμημένων συστημάτων.

Τα μη κατανεμημένα συστήματα ελέγχου έχουν παίξει καθοριστικό ρόλο στην ανάπτυξη της πληροφορικής και της μηχανογράφησης. Παρότι η τάση κινείται προς τα κατανεμημένα συστήματα, τα κεντρικά συστήματα παραμένουν κρίσιμα σε πολλούς τομείς, ιδιαίτερα εκεί όπου η κεντρική διαχείριση και ο έλεγχος είναι απαραίτητοι.

2.2 Γιατί πρέπει να προτιμηθεί ενα κατανεμημένο σύστημα ελέγχου

Ένα κατανεμημένο σύστημα ελέγχου μπορεί να προτιμηθεί για πολλους σημαντικούς λόγους, ιδίως σε περιβάλλοντα όπου οι απαιτήσεις διαχείρισης εγγράφων και δεδομένων είναι υψηλές. Εδώ είναι μερικοί από τους κύριους λόγους που καθιστούν επιθυμητή την προτίμηση ενός τέτοιου συστήματος:

Κλιμάκωση και Ευελιξία:

Τα κατανεμημένα συστήματα μπορούν να κλιμακώνονται εύκολα για να καλύψουν τις αυξανόμενες ανάγκες της επιχείρησης. Μπορούν να προστεθούν νέοι πόροι (όπως servers) χωρίς να επηρεαστεί η συνολική απόδοση του συστήματος.

Αξιοπιστία και Διαθεσιμότητα:

Σε περίπτωση που ένα τμήμα του συστήματος αποτύχει ή δεν είναι διαθέσιμο, άλλες περιοχές μπορούν να συνεχίσουν να λειτουργούν κανονικά. Αυτό βελτιώνει την ανθεκτικότητα και τη συνεχιζόμενη λειτουργία του συστήματος.

Ασφάλεια:

Με την κατανομή των δεδομένων και των εφαρμογών σε διάφορες τοποθεσίες ή διακομιστές, μπορεί να ενισχυθεί η ασφάλεια. Επίσης, η ασφάλεια μπορεί να βελτιωθεί μέσω της εφαρμογής διαφορετικών επιπέδων ελέγχου πρόσβασης και κρυπτογράφησης.

Εξοικονόμηση Κόστους:

Τα κατανεμημένα συστήματα μπορούν να είναι πιο οικονομικά αποδοτικά, ειδικά όταν χρησιμοποιούν κοινές υποδομές ή υπηρεσίες cloud, που μειώνουν την ανάγκη για σημαντικές επενδύσεις σε υλικό.

Διαχείριση Μεγάλου Όγκου Δεδομένων:

Είναι ιδανικά για τη διαχείριση μεγάλων όγκων δεδομένων και εγγράφων, καθώς μπορούν να κατανεμηθούν σε πολλούς servers και αποθηκευτικούς χώρους.

Ευκολία Συνεργασίας:

Διευκολύνει τη συνεργασία μεταξύ ομάδων που μπορεί να βρίσκονται σε διαφορετικές γεωγραφικές τοποθεσίες. Οι χρήστες μπορούν να συνεργάζονται και να επεξεργάζονται έγγραφα ταυτόχρονα από διαφορετικά μέρη.

Αποτελεσματική Διαχείριση Ροών Εργασίας:

Ενσωματώνει προηγμένα εργαλεία για τη διαχείριση και αυτοματοποίηση ροών εργασίας, βελτιώνοντας την αποτελεσματικότητα και μειώνοντας την ανθρώπινη παρέμβαση.

Ενοποίηση και Διαλειτουργικότητα:

Μπορεί να ενσωματωθεί με άλλες εφαρμογές και συστήματα, διευκολύνοντας την διαχείριση δεδομένων σε ποικιλία πλατφορμών και εφαρμογών.

Αναφορά και Στατιστικά:

Παρέχει ισχυρές δυνατότητες αναφοράς και ανάλυσης, βοηθώντας στην παρακολούθηση της χρήσης των εγγράφων και των διαδικασιών.

Ανάκτηση από Καταστροφή:

Σε περίπτωση καταστροφής ή απώλειας δεδομένων, η κατανομή των δεδομένων σε διαφορετικές τοποθεσίες μπορεί να διευκολύνει την ανάκτηση και την αποκατάσταση.

Η προτίμηση ενός κατανεμημένου συστήματος ελέγχου είναι συχνά αποτέλεσμα της ανάγκης για υψηλή διαθεσιμότητα, ασφάλεια και ευελιξία, ειδικά σε περιβάλλοντα με σύνθετες απαιτήσεις διαχείρισης δεδομένων.

2.3 Δημοφιλείς υπάρχουσες εφαρμογές

Ορισμένες δημοφιλείς εφαρμογές και πλατφόρμες, που χρησιμοποιούν κατανεμημένα συστήματα ελέγχου περιλαμβάνουν εργαλεία για τη διαχείριση εγγράφων, την παρακολούθηση και τον έλεγχο εκδόσεων. Εδώ είναι μερικά παραδείγματα:

Microsoft SharePoint

Περιγραφή: Ένα ολοκληρωμένο σύστημα διαχείρισης περιεχομένου και συνεργασίας, που επιτρέπει την αποθήκευση, την οργάνωση και την κοινή χρήση εγγράφων και δεδομένων σε κατανεμημένα περιβάλλοντα.

Χαρακτηριστικά: Διαχείριση εγγράφων, συνεργασία σε πραγματικό χρόνο, διαχείριση ροών εργασίας, και ενσωμάτωσή με άλλες εφαρμογές του Microsoft 365.

Google Workspace (πρώην G Suite)

Περιγραφή: Μια σουίτα εργαλείων παραγωγικότητας, που περιλαμβάνει εφαρμογές για τη δημιουργία, αποθήκευση και συνεργασία σε έγγραφα.

Χαρακτηριστικά: Google Docs, Sheets, Slides, Drive για αποθήκευση και κοινή χρήση αρχείων, και ισχυρές δυνατότητες συνεργασίας και σχολιασμού.

Dropbox Business

Περιγραφή: Υπηρεσία αποθήκευσης και συγχρονισμού αρχείων, που προσφέρει δυνατότητες συνεργασίας σε κατανεμημένα συστήματα.

Χαρακτηριστικά: Κοινή χρήση αρχείων, διαχείριση δικαιωμάτων πρόσβασης, συνεργασία σε έγγραφα, και εργαλεία αναφοράς.

GitHub

Περιγραφή: Πλατφόρμα για την παρακολούθηση και διαχείριση κώδικα, πηγαίου κώδικα, χρησιμοποιώντας ένα σύστημα ελέγχου εκδόσεων κατανεμημένου τύπου.

Χαρακτηριστικά: Διαχείριση εκδόσεων, παρακολούθηση αλλαγών, συγχώνευση κώδικα, και συνεργασία σε προγραμματιστικά έργα.

GitLab

Περιγραφή: Πλατφόρμα, που παρέχει δυνατότητες ελέγχου εκδόσεων, συνεχιζόμενης ολοκλήρωσης και ανάπτυξης, και συνεργασίας.

Χαρακτηριστικά: Ενσωματωμένο σύστημα ελέγχου εκδόσεων, CI/CD (Συνεχής Ολοκλήρωση και Συνεχής Ανάπτυξη), και διαχείριση έργων.

Confluence

Περιγραφή: Εργαλείο συνεργασίας και διαχείρισης περιεχομένου, που επιτρέπει τη δημιουργία και οργάνωση εγγράφων και σημειώσεων σε ομάδες.

Χαρακτηριστικά: Δημιουργία σελίδων, συνεργασία σε έγγραφα, ενσωμάτωσή με άλλες πλατφόρμες, και διαχείριση περιεχομένου.

Alfresco

Περιγραφή: Σύστημα διαχείρισης περιεχομένου και επιχειρησιακής διαδικασίας, που προσφέρει λύσεις για τη διαχείριση εγγράφων και περιεχομένου.

Χαρακτηριστικά: Αποθήκευση εγγράφων, διαχείριση ροών εργασίας, έλεγχος εκδόσεων, και συνεργασία.

Αυτά τα εργαλεία και πλατφόρμες χρησιμοποιούν κατανεμημένα συστήματα, για να παρέχουν ισχυρές λύσεις διαχείρισης εγγράφων, συνεργασίας και ελέγχου εκδόσεων, ενισχύοντας την αποτελεσματικότητα και την παραγωγικότητα σε διάφορα επαγγελματικά περιβάλλοντα.

Κεφάλαιο 3 Τεχνολογίες Κατανεμημένων Συστημάτων

3.1 Σύγχρονες Εξελίξεις στις Τεχνολογίες Κατανεμημένων Συστημάτων

Η τεχνολογίες των κατανεμημένων συστημάτων έχουν υποστεί σημαντική εξέλιξη με την πάροδο των χρόνων. Από τις πρώτες καινοτομίες, που έθεσαν τα θεμέλια αυτών των συστημάτων,εως τις σύγχρονες λύσεις, που προσαρμόζονται στις ανάγκες του σημερινού κόσμου. Ας εξετάσουμε μερικές από τις βασικές τεχνολογίες του παρελθόντος και πώς έχουν αντικατασταθεί ή εξελιχθεί σήμερα.

DCE (Distributed Computing Environment): Το DCE υπήρξε μία από τις πρώτες προσπάθειες για την ανάπτυξη κατανεμημένων εφαρμογών. Παρείχε εργαλεία για τη διαχείριση επικοινωνίας και συγχρονισμού μεταξύ διαφορετικών συστημάτων. πραγματοποιείται με τεχνολογίες όπως τα Microservices και το Docker, που προσφέρουν μεγαλύτερη ευελιξία και αποτελεσματικότητα.

DNS (Domain Name System): Το DNS είναι κρίσιμο για τη λειτουργία του Διαδικτύου, καθώς μετατρέπει τα ονόματα τομέων σε IP διευθύνσεις, επιτρέποντας την πλοήγηση και την επικοινωνία στο Διαδίκτυο. Ενώ το DNS παραμένει αμετάβλητο στη βασική του λειτουργία, νέες εξελίξεις όπως το DNS over HTTPS (DoH) έχουν προστεθεί για την ενίσχυση της ασφάλειας των διαδικτυακών αιτημάτων.

X.500: Το X.500 ήταν ένα πρωτότυπο για υπηρεσίες καταλόγου, που διευκόλυνε την αποθήκευση και αναζήτηση πληροφοριών χρηστών σε κατανεμημένα συστήματα. Το X.500 έχει αντικατασταθεί πλέον από το LDAP (Lightweight Directory Access Protocol), που προσφέρει μια πιο ελαφριά και αποδοτική λύση.

NDS (Novell Directory Services): Το NDS από τη Novell πήρξε ένα βασικό εργαλείο για τη διαχείριση χρηστών και πόρων σε δίκτυα. Ωστόσο, σημερα, το Active Directory της Microsoft έχει αναδειχθεί ως η κυρίαρχη λύση για τη διαχείριση καταλόγων και πόρων σε σύγχρονα δίκτυα.

Active Directory: Το Active Directory εξακολουθεί να είναι μια από τις πλέον διαδεδομένες λύσεις για τη διαχείριση δικτύων Windows. Το Azure Active Directory έχει επεκτείνει τις δυνατότητές του στο cloud, προσφέροντας λύσεις για εφαρμογές και υπηρεσίες που βασίζονται στο cloud.

NFS (Network File System): Το NFS χρησιμοποιείται ευρέως για την κοινή χρήση αρχείων σε περιβάλλοντα Unix και Linux. Παρόλο που σύγχρονες λύσεις όπως το SMB (Server Message Block) και οι υπηρεσίες αποθήκευσης στο cloud έχουν διευρύνει τις δυνατότητες, το NFS εξακολουθεί να παραμένει αποτελεσματικό και ευρέως χρησιμοποιούμενο.

AFS (Andrew File System): Το AFS ήταν ένα πρωτοπόριακο κατανεμημένο σύστημα αρχείων για τη διαχείριση και την κοινή χρήση αρχείων σε μεγάλα δίκτυα. Σήμερα, συστήματα όπως το Ceph και το GlusterFS παρέχουν προηγμένα χαρακτηριστικά για κατανεμημένη αποθήκευση και διαχείριση δεδομένων.

Java RMI (Remote Method Invocation): Το Java RMI επέτρεπε την κλήση μεθόδων σε απομακρυσμένα αντικείμενα σε περιβάλλοντα Java. Σήμερα, τεχνολογίες όπως τα RESTful APIs και gRPC έχουν γίνει προτιμώμενες για την επικοινωνία μεταξύ κατανεμημένων συστημάτων, προσφέροντας μεγαλύτερη ευελιξία και υποστήριξη για διαφορετικές γλώσσες προγραμματισμού.

CORBA (Common Object Request Broker Architecture): Το CORBA παρείχε την επικοινωνία μεταξύ αντικειμένων σε κατανεμημένα συστήματα ανεξαρτήτως γλώσσας προγραμματισμού. Στις μέρες μας, το CORBA έχει αντικατασταθεί από λύσεις όπως τα RESTful APIs και SOAP, που προσφέρουν πιο ευέλικτες και σύγχρονες επιλογές για την επικοινωνία μεταξύ υπηρεσιών.

Η παραπάνω ανασκόπηση υπογραμμίζει την πορεία εξέλιξης των τεχνολογιών κατανεμημένων συστημάτων, αναδεικνύοντας πώς οι παλαιότερες τεχνολογίες έχουν είτε αντικατασταθεί είτε προσαρμοστεί για να καλύψουν τις απαιτήσεις του σύγχρονου ψηφιακού περιβάλλοντος.

3.2 Η Τεχνολογία Cloud

Το cloud computing, μια ιδέα που ξεκίνησε να διαμορφώνεται στις δεκαετίες του '60 και του '70, έχει εξελιχθεί σε μία από τις πιο σημαντικές τεχνολογικές εξελίξεις των τελευταίων δεκαετιών. Οι πρώτες προσπάθειες για τη δημιουργία δικτύων υπολογιστών ήταν το θεμέλιο πάνω στο οποίο χτίστηκε η σημερινή πραγματικότητα του cloud. Ωστόσο, η πραγματική έκρηξη στο cloud computing ήρθε στα τέλη της δεκαετίας του '90 και τις αρχές των '00s, με την άνοδο του Διαδικτύου και την ανάπτυξη ψηφιακών υποδομών.

Η διάδοση του cloud computing υποστηρίχθηκε από τη ραγδαία αύξηση της ευρυζωνικότητας, η οποία επέτρεψε στις υπηρεσίες cloud να γίνουν προσβάσιμες από οποιονδήποτε, οπουδήποτε και οποιαδήποτε στιγμή. Η ιδέα της παροχής υπηρεσιών υπολογισμού μέσω απομακρυσμένων servers, όπου οι χρήστες δεν χρειάζεται να επενδύσουν σε δική τους υλική υποδομή, αποδείχθηκε επαναστατική.

Η Amazon υπήρξε πρωτοπόρος σε αυτόν τον τομέα, ιδρύοντας το Amazon Web Services (AWS) το 2006. Η AWS είναι σήμερα μία από τις κορυφαίες πλατφόρμες cloud, προσφέροντας ένα ευρύ φάσμα υπηρεσιών όπως Amazon S3 για αποθήκευση δεδομένων, Amazon EC2 για υπολογιστική ισχύ και Amazon RDS για διαχείριση βάσεων δεδομένων. Με την πάροδο του χρόνου, η AWS έχει αναπτύξει περισσότερες από 200 υπηρεσίες που καλύπτουν κάθε ανάγκη των επιχειρήσεων, από ανάλυση δεδομένων μέχρι τεχνητή νοημοσύνη.

Η ραγδαία ανάπτυξη του cloud computing έχει αλλάξει τον τρόπο που οι επιχειρήσεις διαχειρίζονται τις υπολογιστικές τους υποδομές. Η υιοθέτηση του cloud επιτρέπει σε οργανισμούς να μειώνουν το κόστος, να αυξάνουν την ευελιξία και να κλιμακώνουν τους πόρους τους ανάλογα με τις ανάγκες τους. Επιπλέον, η εξέλιξη του cloud συνεχίζεται, με την ενσωμάτωση νέων τεχνολογιών όπως η edge computing, η οποία επιτρέπει την επεξεργασία και αποθήκευση δεδομένων πιο κοντά στην πηγή τους, αντιμετωπίζοντας προβλήματα περιορισμένης συνδεσιμότητας σε απομακρυσμένες περιοχές.

Μερικα από τα aws Services

Amazon S3 (Simple Storage Service):

Παρέχει αξιόπιστη και επεκτάσιμη αποθήκευση δεδομένων στο cloud. Οι χρήστες μπορούν να ανεβάζουν, να αποθηκεύουν και να ανακτούν οποιοδήποτε ποσό δεδομένων από οπουδήποτε στον κόσμο. Είναι ιδανικό για αρχεία όπως έγγραφα, εικόνες και βίντεο, και υποστηρίζει τη δημιουργία backup και αρχείων.

Amazon EC2 (Elastic Compute Cloud):

Προσφέρει δυνατότητες υπολογιστικής ισχύος στο cloud. Με τη χρήση EC2, οι χρήστες μπορούν να δημιουργούν και να διαχειρίζονται εικονικές μηχανές (instanes) με διαφορετικές απαιτήσεις επεξεργασίας και μνήμης, επιτρέποντας την εκτέλεση εφαρμογών και την εκτέλεση εργασιών υπολογισμού χωρίς να χρειάζεται φυσικό hardware.

Amazon RDS (Relational Database Service):

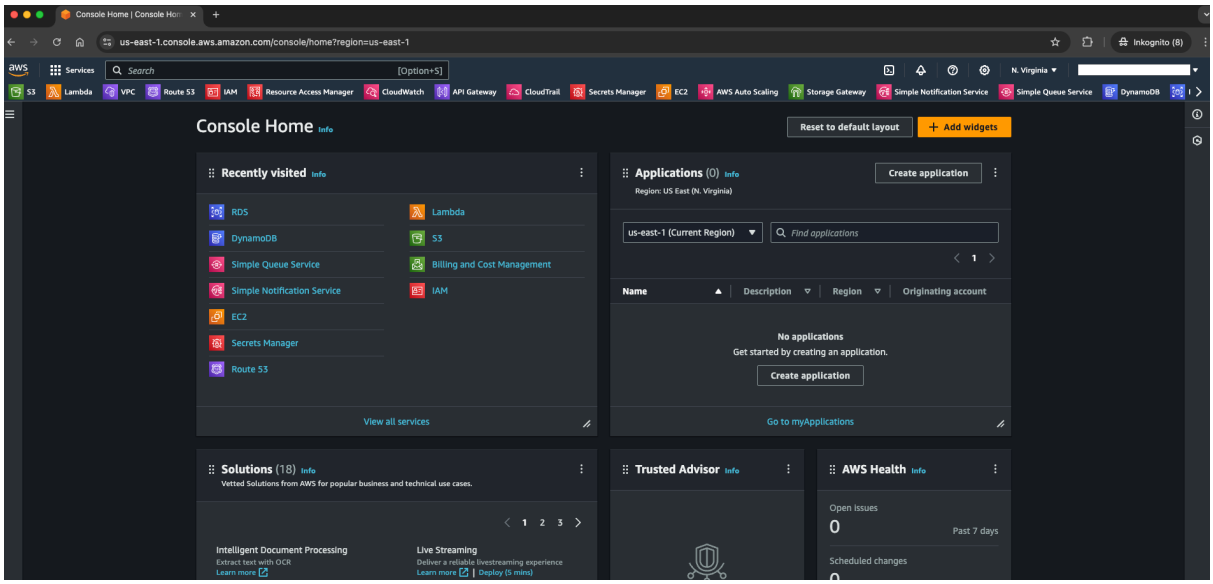
Διευκολύνει τη δημιουργία, τη διαχείριση και την κλιμάκωση σχεσιακών βάσεων δεδομένων στον cloud. Υποστηρίζει διάφορους τύπους βάσεων δεδομένων, όπως MySQL, PostgreSQL, MariaDB, Oracle και SQL Server. Με το RDS, οι χρήστες μπορούν να εστιάσουν στη σχεδίαση και την ανάπτυξη των εφαρμογών τους, χωρίς να ανησυχούν για τη συντήρηση της βάσης δεδομένων.

AWS-Lambda:

Επιτρέπει την εκτέλεση κώδικα χωρίς να χρειάζεται να διαχειρίζεστε servers. Με το Lambda, μπορείτε να γράψετε και να εκτελέσετε κώδικα σε απόκριση σε γεγονότα, όπως αλλαγές σε δεδομένα ή αιτήματα από εφαρμογές, και πληρώνετε μόνο για τον χρόνο εκτέλεσης του κώδικα σας. Είναι χρήσιμο για αυτοματοποίηση διαδικασιών, αναλύσεις δεδομένων και άλλες λειτουργίες που απαιτούν ευελιξία.

Security Groups:

Λειτουργούν ως τείχη προστασίας (firewalls) για τα EC2 instances. Με τα Security Groups, μπορουμε να ελέγχουμε την πρόσβαση και την επικοινωνία μεταξύ των instances και του internet, καθορίζοντας κανόνες που επιτρέπουν ή απαγορεύουν τη ροή δεδομένων ανάλογα με τις IP διευθύνσεις και τις θύρες.



Εικόνα 1 AWS Console

3.3 Η Τεχνολογία Terraform

Το **Terraform** είναι ένα εργαλείο που κάνει τη διαχείριση της υποδομής μιας εταιρείας(servers, Network) πιο εύκολη και αυτοματοποιημένη. Αντί να στήνουμε χειροκίνητα servers, δίκτυα και βάσεις δεδομένων, μπορείς να τα δημιουργησουμε με κώδικα σε απλά αρχεία κειμένου. Με αυτόν τον τρόπο, όταν χρειαστουμε να δημιουργήσουμε ή να αλλάξουμε το Network η τους servers μας, το Terraform αναλαμβάνει να το κάνει αυτόματα για εμας.

Το Terraform μπορει και λειτουργεί με πολλούς διαφορετικούς παρόχους, όπως το **AWS**, το **Azure** ή το **Google Cloud**, πράγμα που σημαίνει ότι μπορουμε να διαχειρίζομαστε τις υποδομες μας ανεξάρτητα από το ποιο cloud χρησιμοποιουμε. Μπορουμε επίσης να δουλεύουμε σε πολλαπλά περιβάλλοντα και να ξέρουμε τι εχει ηδη γινει, αφού κάθε αλλαγή καταγράφεται στον κώδικα. Είναι πολύ χρήσιμο γιατί μας δίνει τη δυνατότητα να φτιάξουμε υποδομές γρήγορα, να κάνουμε επαναχρησημοποιησουμε των ιδιο κωδικα. Έτσι, μπορουμε να εστιάσουμε στην ανάπτυξη του project μας, αφήνοντας τις πολύπλοκες τεχνικές ρυθμίσεις του server στο Terraform.

Μερικά βασικά στοιχεία που πρέπει να ξέρουμε για το **Terraform** είναι:

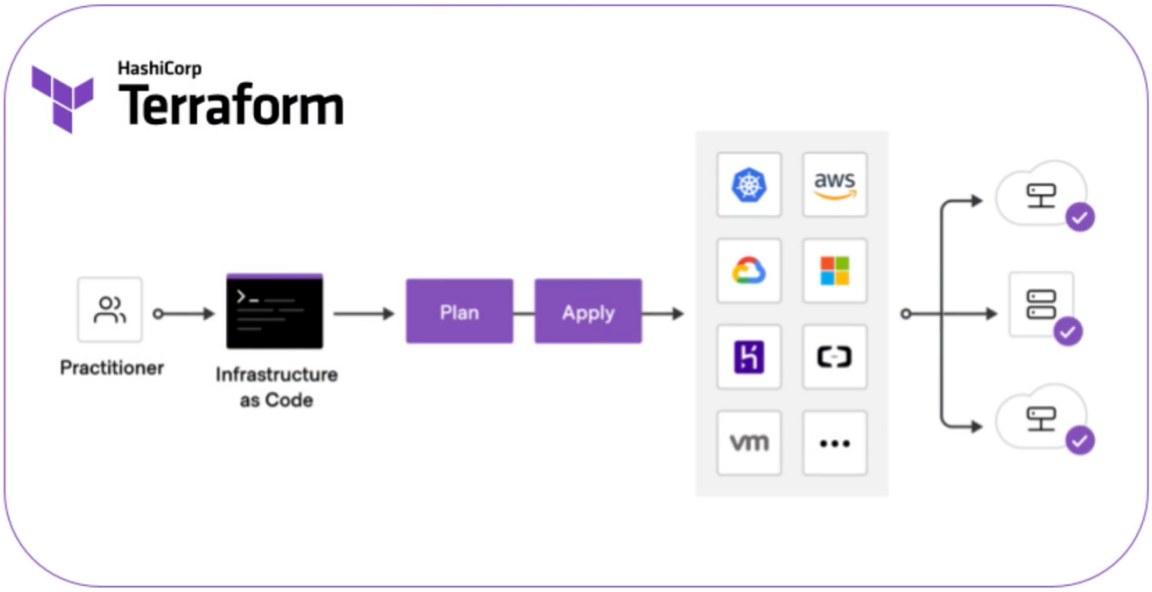
Infrastructure as Code (IaC): Με το Terraform μπορούμε να περιγράψουμε τις υποδομες μας σαν κώδικα. Αυτό μας επιτρέπει να δημιουργούμε, να τροποποιούμε και να αναπαράγουμε τις υποδομες μας εύκολα και με συνέπεια, χωρίς να πρέπει να κάνουμε χειροκίνητες ρυθμίσεις.

Multi-cloud υποστήριξη: Μπορούμε να διαχειριζόμαστε υποδομές σε πολλούς cloud παρόχους ταυτόχρονα, όπως το **AWS**, το **Google Cloud**, το **Azure**, αλλά και σε τοπικά (on-premises) συστήματα. Έτσι, έχουμε ευελιξία και δεν δεσμευόμαστε μόνο σε έναν πάροχο.

Αναπαραγωγιμότητα: Κάθε φορά που γράφουμε ή αλλάζουμε τον κώδικα των υποδομων μας, το Terraform αναλαμβάνει να το εφαρμόσει με τον ίδιο ακριβώς τρόπο. Αυτό μας βοηθά να αποφεύγουμε λάθη, που μπορεί να προκύψουν με χειροκίνητες ρυθμίσεις.

Διαχείριση αλλαγών: Με το Terraform, μπορούμε να δούμε τι αλλαγές θα γίνουν στην υποδομή μας πριν τις εφαρμόσουμε. Έτσι, έχουμε πλήρη έλεγχο για να αποφεύγουμε ανεπιθύμητα αποτελεσματα.

Πολύπλοκη υποδομή: Το Terraform μας επιτρέπει να διαχειριζόμαστε και να συντονίζουμε πολύπλοκες υποδομές, όπως μεγάλα δίκτυα, βάσεις δεδομένων και εφαρμογές, σε πολλά περιβάλλοντα με ένα ενιαίο εργαλείο.



Εικόνα 2 IaC με Terraform

3.4 Δημιουργία υποδομών AWS με την Τεχνολογία Terraform IaC

Η δημιουργία υποδομών στο AWS με τη χρήση της τεχνολογίας **Terraform και** Infrastructure as Code (IaC) επιτρέπει στους χρήστες να αυτοματοποιήσουν τη διαδικασία διαχείρισης των πόρων τους στο AWS. Με το Terraform, μπορούμε να περιγράψουμε όλες τις απαιτούμενες υποδομές, όπως EC2 instances, S3 buckets, VPCs κ.ά., χρησιμοποιώντας αρχεία κώδικα. Έτσι, η δημιουργία, διαμόρφωση και ενημέρωση των υποδομών γίνεται πιο αποτελεσματική και χωρίς λάθη, ενώ διασφαλίζεται η αναπαραγωγιμότητα των ρυθμίσεων.

Η τεχνολογία **IaC** επιτρέπει στις ομάδες να επαναχρησιμοποιούν τις διαμορφώσεις, να συνεργάζονται πιο αποδοτικά και να εφαρμόζουν αλλαγές με συνέπεια και έλεγχο, ακόμα και σε πολύπλοκα περιβάλλοντα AWS.

```
resource "aws_security_group" "instance_sg" {# Security Group for EC2
name        = "instance_sg"
description = "Allow inbound traffic for HTTP and SSH"
vpc_id      = var.vpc_id
ingress {
from_port   = 22
to_port     = 22
protocol    = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}

ingress {
from_port   = 8080
to_port     = 8080
protocol    = "tcp"
cidr_blocks = ["0.0.0.0/0"]
}
}

resource "aws_instance" "app_instance" {
ami                    = "ami-0c55b159cbfafe1f0" # EC2 Instance
instance_type          = "t2.micro"
iam_instance_profile   = aws_iam_instance_profile.ec2_instance_profile.name
vpc_security_group_ids = [aws_security_group.instance_sg.id]
```

```
tags = {
Name = "document_management_system"
}
                                                }
```

Κώδικας 1 Terraform για την δημιουργία EC2 Instance.

Στο συγκεκριμένο αρχείο κωδικα 1 δημιουργούμε

Security Group (aws_security_group)

Αυτό το τμήμα του κώδικα δημιουργεί ένα νέο Security Group με τα εξής χαρακτηριστικά:

- name: Ονομα. "instance_sg".
- description: Περιγραφή που λέει "Allow inbound traffic for HTTP and SSH", μας λεει ότι επιτρέπει την είσοδο για HTTP και SSH.
- vpc_id: Ορίζεται μέσω της μεταβλητής var.vpc_id, η οποία πρέπει να περιέχει την ID του VPC (Virtual Private Cloud) στο οποίο θα δημιουργηθεί το Security Group.

Ingress Rules:

- Πρώτος κανόνας: Επιτρέπει την είσοδο στο port 22 (SSH) από οποιαδήποτε IP διεύθυνση (0.0.0.0/0).
- Δεύτερος κανόνας: Επιτρέπει την είσοδο στο port 8080 (συνήθως χρησιμοποιούμενο για HTTP εφαρμογές) από οποιαδήποτε IP διεύθυνση (0.0.0.0/0).

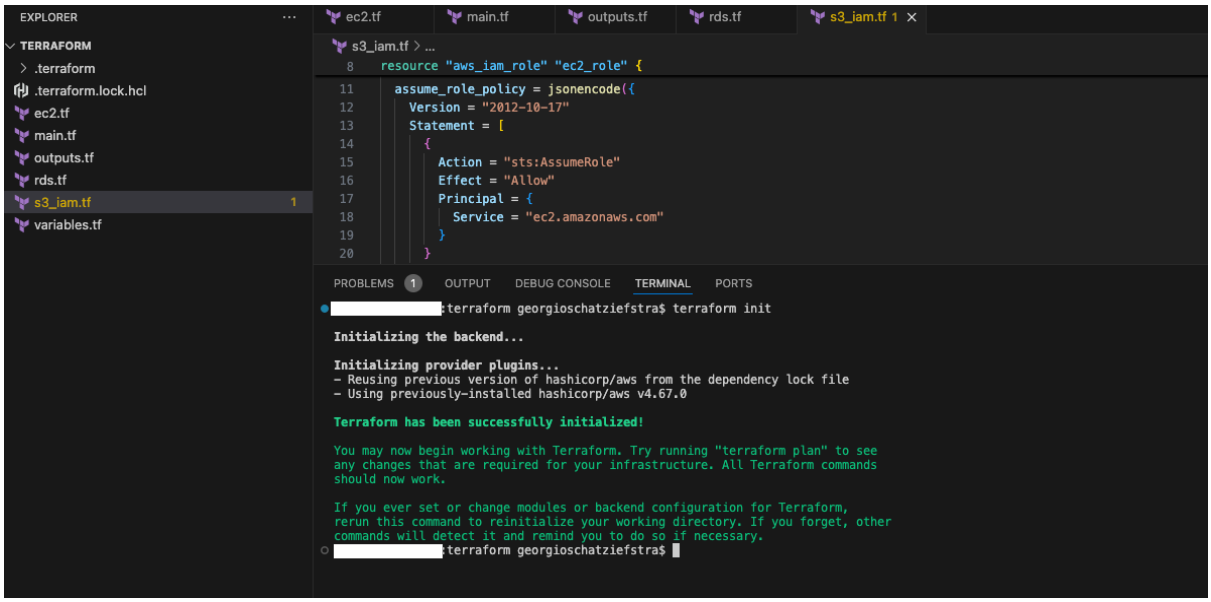
EC2 Instance (aws_instance)

Αυτό το τμήμα του κώδικα δημιουργεί ένα νέο EC2 instance με τα εξής χαρακτηριστικά:

- ami: Ορίζεται μέσω του ID της Amazon Machine Image (AMI). Στην περίπτωση αυτή, χρησιμοποιείται η AMI για Amazon Linux 2 (ami-0c55b159cbfafe1f0).
- instance_type: Ορίζεται σε "t2.micro", που είναι ένας τύπος instance με μικρό κόστος και περιορισμένους πόρους.
- iam_instance_profile: Συνδέει το EC2 instance με ένα IAM instance profile που πρέπει να έχει οριστεί αλλού (aws_iam_instance_profile.ec2_instance_profile.name).
- vpc_security_group_ids: Συνδέει το EC2 instance με το Security Group που δημιουργήθηκε παραπάνω.
- tags: Εφαρμόζει ένα tag με όνομα "Name" και τιμή "document_management_system", για να μπορουμενα αναγνωρίσουμε εύκολα το instance.

Σε συνδυασμό με άλλα αρχεία δημιουργούνται έτσι η δομές που χρειαζόμαστε για το Cloud . Εδώ για το AWS συγκεκριμένα.

Τα αρχεία μας θα πρέπει να έχουν την κατάληξη .tf Δηλαδή rds.tf, ec2.tf, main.tf ,output.tf, s3_iam.tf κοκ.

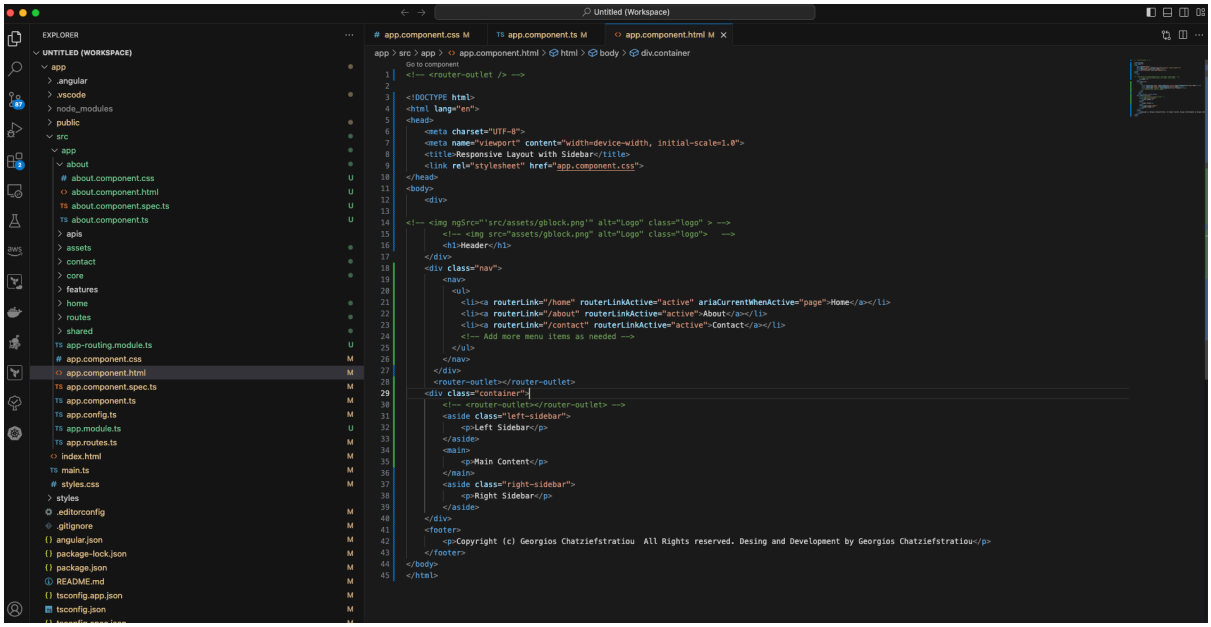


Εικόνα 3 Το αποτέλεσμα μετά από επιτυχή χρήση του terraform init

3.4 Η Τεχνολογία Angular

Το Angular είναι ένα από τα πιο δημοφιλή framework για την ανάπτυξη διαδικτυακών εφαρμογών. Ξεκίνησε ως εσωτερικό project στην Google και έχει μια ενδιαφέρουσα ιστορία που ξεκινά τη δεκαετία του 2000.

Η πρώτη έκδοση Angular ανακοινώθηκε από τον Misko Hevery το 2009 ως AngularJS, ενα JavaScript framework για τη δημιουργία διαδικτυακών εφαρμογών. Απέκτησε ταχεία δημοτικότητα για τη δυνατότητα του να προσφέρει δυνατότητες δικτύωσης δεδομένων (data binding) και τη δυνατότητα να οργανώνει τον κώδικα μέσω της λειτουργίας του MVC (Model-View-Controller).



Εικόνα 4 Η Δομη από ένα Angular Project μεσα από το Visual studio code

Καθώς οι ανάγκες των διαδικτυακών εφαρμογών εξελίσσονταν, η ομάδα ανάπτυξης του Angular ανακοίνωσε το Angular 2 το 2014. Αυτή η νέα έκδοση είχε μια εντελώς ανανεωμένη αρχιτεκτονική, γραμμένη από την αρχή σε TypeScript. Αυτή η επανασχεδιασμένη έκδοση ήταν πολύ πιο αποδοτική, εύκολη στη συντήρηση και επέτρεπε τη δημιουργία πιο σύγχρονων εφαρμογών.

Μερικα από τα στοιχεια που εχει το Angular είναι τα components, Services, Modules .
Πιο κατω βλεπουμε ένα παραδειγμα κώδικα σε Angular για ένα component καθώς βλέπουμε το backend το html και το spec οπου μπορουμε να γραψουμε τα τεστ για ότι κωδικα εχουμε γραψει .

```
import { Component } from '@angular/core';
import { RouterOutlet } from '@angular/router';
@Component({
  selector: 'app-root',
  standalone: true,
  imports: [RouterOutlet],
  templateUrl: './app.component.html',
  styleUrl: './app.component.css'
})
export class AppComponent {
  title = 'app';
  items: string[] = ['Apple', 'Banana', 'Cherry'];
  newItem: string = '';
  addItem() {
    if (this.newItem.trim()) {
      this.items.push(this.newItem);
      this.newItem = '';
    }
  }

                                                }
```

Κώδικας 2 Παράδειγμα από το app.component.ts του Angular

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Responsive Layout with Sidebar</title>
    <link rel="stylesheet" href="app.component.css">
</head>
<body>
    <div>
              <h1>Header</h1>
    </div>
    <div class="nav">
        <nav>
          <ul>
            <li><a routerLink="/home" routerLinkActive="active"
ariaCurrentWhenActive="page">Home</a></li>
            <li><a routerLink="/about" routerLinkActive="active">About</a></li>
            <li><a routerLink="/contact" routerLinkActive="active">Contact</a></li>

          </ul>
        </nav>
      </div>
      <router-outlet></router-outlet>
    <div class="container">
        <aside class="left-sidebar">
```

```
            <p>Left Sidebar</p>
        </aside>
        <main>
            <p>Main Content</p>
        </main>
        <aside class="right-sidebar">
            <p>Right Sidebar</p>
        </aside>
    </div>
    <footer>
        <p>Copyright (c) Georgios Chatziefstratiou All Rights reserved. Desing and
Development by Georgios Chatziefstratiou</p>
    </footer>
</body>
</html>
```

Κώδικας 3 Παράδειγμα από το app.component.html του Angular

```
import { TestBed } from '@angular/core/testing';
import { AppComponent } from './app.component';

describe('AppComponent', () => {
  beforeEach(async () => {
    await TestBed.configureTestingModule({
      imports: [AppComponent],
    }).compileComponents();
  });

  it('should create the app', () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app).toBeTruthy();
  });

  it(`should have the 'app' title`, () => {
    const fixture = TestBed.createComponent(AppComponent);
    const app = fixture.componentInstance;
    expect(app.title).toEqual('app');
  });

  it('should render title', () => {
    const fixture = TestBed.createComponent(AppComponent);
    fixture.detectChanges();
    const compiled = fixture.nativeElement as HTMLElement;
    expect(compiled.querySelector('h1')?.textContent).toContain('Hello, app');
  });
                                          });
```

Κώδικας4 Παράδειγμα το app.component.spec.ts του Angular

Από τότε, οι εκδόσεις του Angular έχουν συνεχίσει να εξελίσσονται, με την ομάδα ανάπτυξης να προσθέτει νέες λειτουργίες, βελτιώσεις στην απόδοση και εργαλεία για τη διευκόλυνση των προγραμματιστών. Το Angular έχει εξελιχθεί σε ένα ισχυρό framework που χρησιμοποιείται ευρέως για τη δημιουργία μοντέρνων εφαρμογών στον ιστό.

3.5 Η Τεχνολογία Java, Spring, JPA

Η Java ξεκίνησε την πορεία της το 1991 από τον James Gosling. Η αρχική της ονομασία ήταν Oak, αλλά στη συνέχεια μετονομάστηκε σε Java, εμπνευσμένη από τον καφέ Java. Η Java έγινε δημοφιλής για τη δυνατότητά της να εκτελείται σε διαφορετικές πλατφόρμες (platform-independent) λόγω της εικονικής μηχανής Java (Java Virtual Machine - JVM).

Η ανάπτυξη της Java έφτασε στο σημείο που η Sun Microsystems ανακοίνωσε τη Java Development Kit (JDK) το 1995, παρέχοντας ένα ολοκληρωμένο σύνολο εργαλείων για την ανάπτυξη λογισμικού σε Java.

Εικόνα 5 https://dev.java/

Ένα από τα πιο σημαντικά γεγονότα ήταν η εισαγωγή των Java Enterprise Edition (EE) και Java Micro Edition (ME) που διευκόλυναν την ανάπτυξη εφαρμογών για επιχειρηματικό περιβάλλον και για συσκευές με περιορισμένους πόρους αντίστοιχα.

Όσον αφορά τα frameworks, η Java έχει παράγει μια ποικιλία από αυτά που επιτρέπουν στους προγραμματιστές να αναπτύσσουν γρήγορα και αποτελεσματικά εφαρμογές. Κάποια από τα πιο γνωστά Java frameworks είναι:

Spring Framework Το Spring ξεκίνησε το 2003 και έγινε ένα από τα πλέον δημοφιλή frameworks για την ανάπτυξη εφαρμογών Java. Προσφέρει λύσεις για τη διαχείριση εξαρτήσεων, την απλοποίηση της ανάπτυξης RESTful υπηρεσιών, τη διαχείριση του κύκλου ζωής των αντικειμένων και άλλα.

Πιο κατω βλέπουμε μια κλάση Java με την χρήση annotations του spring @RestController , @RequestMpping ,@GetMapping, @PostMapping κτλ.

```
@RestController
@RequestMapping("/api/document/")
public class DocumentController {

    private static final Logger logger =
LoggerFactory.getLogger(MethodHandles.lookup().lookupClass());

    @Autowired
    private final DocumentService documentService;


    @Autowired
    public DocumentController(DocumentService documentService) {
        this.documentService = documentService;
    }

@PostMapping("/upload")
```

```
public ResponseEntity<UploadResponse> uploadDocument(@RequestParam("file")
MultipartFile file,@RequestParam String userId) {
        try {
            UploadResponse response = documentService.uploadDocument(file ,
userId);
            return new ResponseEntity<>(response,
HttpStatus.valueOf(response.getStatusCode()));
        } catch (IOException e) {
            e.printStackTrace();
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }


@GetMapping("/download/{s3Key}")
public ResponseEntity<byte[]> downloadDocument(@PathVariable String s3Key) {
        try (InputStream inputStream = documentService.downloadDocument(s3Key)) {
            byte[] content = inputStream.readAllBytes();
            HttpHeaders headers = new HttpHeaders();
            headers.add(HttpHeaders.CONTENT_DISPOSITION, "attachment; filename=" +
s3Key);
            return new ResponseEntity<>(content, headers, HttpStatus.OK);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

@DeleteMapping("/deleteDocument")
public ResponseEntity<String> deleteDocument(
            @RequestParam String organizationId,
            @RequestParam String userId,
            @RequestParam String fileName) {
        try {
            documentService.deleteDocument(organizationId, userId, fileName);
            return new ResponseEntity<>("Document deleted successfully.",
HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>("Failed to delete document.",
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
```

Κώδικας 5 Παράδειγμα Documentcontroller java κλασης με τα annotations

```


@Service
public class DocumentService {

    private static final Logger logger =
LoggerFactory.getLogger(MethodHandles.lookup().lookupClass());

    @Autowired
    private S3Client s3Client;
    @Autowired
    private DocumentRepository documentRepository;

    @Autowired
    private DocumentMetadataRepository metadataRepository;

    private  String bucketName;
    @Autowired

public DocumentService(S3Client s3Client, @Value("${aws.bucketName}") String
bucketName) {
        this.s3Client = s3Client;
         this.bucketName = bucketName;
     }

    public MediaType getMediaType(String mimeType) {
        switch (mimeType) {
```

```
            case "application/pdf":
                return MediaType.APPLICATION_PDF;
            case "image/jpeg":
                return MediaType.IMAGE_JPEG;
            case "image/png":
                return MediaType.IMAGE_PNG;
            default:
                return MediaType.APPLICATION_OCTET_STREAM
        }
    }

public UploadResponse uploadDocument(MultipartFile file ,String userId) throws
IOException {
        Document lastDocument = documentRepository.findTopByOrderByIdDesc();
        logger.info("Starting the process for upload to s3 send sns  etc..");
        String s3Key =   UUID.randomUUID().toString();
        String filename = file.getOriginalFilename();
        String mimeType = file.getContentType();
        if (mimeType == null) {
            mimeType = "application/octet-stream";
        }
        MediaType mediaType = MediaTypeUtil.getMediaType(mimeType);
        try (InputStream fileInputStream = file.getInputStream()) {
            PutObjectRequest putObjectRequest = PutObjectRequest.builder()
                    .bucket(bucketName)
                    .key(s3Key)
                    .contentType(mediaType.toString())
                    .build();

            s3Client.putObject(putObjectRequest,
RequestBody.fromInputStream(fileInputStream, file.getSize()));

            String fileUrl = String.format("https://%s.s3.amazonaws.com/%s",
bucketName, s3Key);
            logger.info("success uploading file adding metadata to database  ");
            DocumentMetadata metadata = new DocumentMetadata();
            metadata.setFilename(file.getOriginalFilename());
            metadata.setS3Key(s3Key);
            metadata.setFilename(filename);
            metadata.setSize(file.getSize());

            metadataRepository.save(metadata);
            String previousHash = lastDocument != null ? lastDocument.getHash() :
"0";
            String currentHash = HashUtil.generateHash(new String("") +
previousHash);
            Document doc = new Document();
            doc.setDocumentMetadata(metadata);
            doc.setDocumentName(file.getOriginalFilename());
            doc.setApproved(true);
            doc.setHash(currentHash);
            doc.setPreviousHash(previousHash);
            doc.setTimestamp(LocalDateTime.now());

            documentRepository.save(doc);
            logger.info(" metadata added to database returning response code");
             return new UploadResponse(HttpStatus.OK.value(), fileUrl,
s3Key);
        } catch (S3Exception | NoSuchAlgorithmException e) {
            e.printStackTrace();
            throw new RuntimeException("Error uploading file to S3", e);
        }
    }

public InputStream downloadDocument(String s3Key) {
        try {
            GetObjectRequest getObjectRequest = GetObjectRequest.builder()
                    .bucket(bucketName)
                    .key(s3Key)
                    .build();
```

```
            return s3Client.getObject(getObjectRequest);
        } catch (S3Exception e) {

            logger.info("Error getting object from S3: " +
e.awsErrorDetails().errorMessage());
            e.printStackTrace();
            throw new RuntimeException("Error downloading file from S3", e);
        }
    }


public void deleteDocument(String organizationId, String userId, String fileName) {
        String fileKey = organizationId + "/" + userId + "/" + fileName;
        try {
            DeleteObjectRequest deleteObjectRequest = DeleteObjectRequest.builder()
                    .bucket(bucketName)
                    .key(fileKey)
                    .build();

            s3Client.deleteObject(deleteObjectRequest);
        } catch (S3Exception e) {
            e.printStackTrace();
            throw new RuntimeException("Error deleting file from S3", e);
        }
    }
```

Κώδικας 6 Παράδειγμα service κλαση java με τα annotations

Το Java Persistence API (JPA) είναι ενα Java API για την αποθήκευση, ανάκτηση, ενημέρωση και διαγραφή (CRUD) αντικειμένων σε βάσεις δεδομένων και χρησιμοποιείται μαζί με το Spring Framework. Το JPA παρέχει έναν τυποποιημένο τρόπο για την αλληλεπίδραση με βάσεις δεδομένων χρησιμοποιώντας αντικείμενα Java και εξαλείφει την ανάγκη για τη συγγραφή SQL κώδικα για τις περισσότερες βασικές λειτουργίες.

```
@Repository
public interface DocumentRepository extends JpaRepository<Document, Long> {
    Document findTopByOrderByIdDesc();
}
```

Κώδικας 7 Παράδειγμα από την Repository java κλαση

```
@Entity
@SequenceGenerator(name = "doc_seq", sequenceName = "document_sequence",
allocationSize = 1)
public class Document {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "doc_seq")
    private Long id;

    private String documentName;

    private String hash;

    private String previousHash;

     @Column(columnDefinition = "TIMESTAMP")
     private LocalDateTime timestamp;

    @ManyToOne
    @JoinColumn(name = "document_metadata_id")
    private DocumentMetadata documentMetadata;

    @ManyToOne
```

```
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    public DocumentMetadata getDocumentMetadata() {
        return documentMetadata;
    }

    public void setDocumentMetadata(DocumentMetadata documentMetadata) {
        this.documentMetadata = documentMetadata;
    }

    public Document() {
    }
     public Long getId() {
         return id;
     }

     @PrePersist
     protected void onCreate() {
         timestamp = LocalDateTime.now();
     }

     public String getDocumentName() {
        return documentName;
    }

    public void setDocumentName(String documentName) {
        this.documentName = documentName;
    }

    public String getHash() {
        return hash;
    }

    public void setHash(String hash) {
        this.hash = hash;
    }

    public String getPreviousHash() {
        return previousHash;
    }

    public void setPreviousHash(String previousHash) {
        this.previousHash = previousHash;
    }

    public LocalDateTime getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(LocalDateTime timestamp) {
        this.timestamp = timestamp;
    }

    public void setApproved(boolean b) {
    }
}
```

Κώδικας 8 Παράδειγμα από την Document java κλαση

3.6 Η τεχνολογια Postgresql

Η PostgreSQL είναι μια από τις πιο δημοφιλείς και ισχυρές σχεσιακές βάσεις δεδομένων ανοιχτού κώδικα που έχει σχεδιαστεί για να χειρίζεται μεγάλο όγκο δεδομένων και να παρέχει αξιόπιστες επιδόσεις. Ακολουθούν ορισμένα από τα κύρια χαρακτηριστικά και πλεονεκτήματα της τεχνολογίας PostgreSQL:

Σχεσιακή Δομή Δεδομένων

Η PostgreSQL βασίζεται σε μια σχεσιακή μοντελοποίηση δεδομένων, επιτρέποντας στους χρήστες να δημιουργούν πίνακες με σαφείς σχέσεις μεταξύ τους. Αυτό διευκολύνει τη διαχείριση και την αναζήτηση δεδομένων.

Προηγμένες Δυνατότητες SQL

Υποστηρίζει πλούσιές ερωτήσεις SQL, συμπεριλαμβανομένων πολύπλοκων JOIN, υποερωτήσεων, CTE (Common Table Expressions), και window functions, παρέχοντας ευελιξία στις αναζητήσεις.

Επεκτασιμότητα και Απόδοση

Η PostgreSQL σχεδιάστηκε για να επεξεργάζεται μεγάλους όγκους δεδομένων με υψηλή απόδοση. Υποστηρίζει τεχνικές όπως partitioning και indexing, που βελτιώνουν τη διαχείριση και την αναζήτηση.

Δυνατότητες Γεωχωρικών Δεδομένων

Με την προσθήκη του PostGIS, η PostgreSQL υποστηρίζει γεωχωρικά δεδομένα, επιτρέποντας τη διαχείριση χαρτών και χωρικών αναλύσεων.

Συναλλαγές και Ακεραιότητα Δεδομένων

Υποστηρίζει ACID (Atomicity, Consistency, Isolation, Durability) συναλλαγές, διασφαλίζοντας την ακεραιότητα και την ασφάλεια των δεδομένων.

Ασφάλεια

Διαθέτει προηγμένα μέτρα ασφάλειας, όπως κρυπτογράφηση δεδομένων, έλεγχο πρόσβασης και υποστήριξη για SSL, διασφαλίζοντας ότι τα δεδομένα παραμένουν προστατευμένα.

Πολυγλωσσική Υποστήριξη

Υποστηρίζει διάφορες γλώσσες προγραμματισμού, όπως Python, Java, C, και Ruby, διευκολύνοντας την ενσωμάτωσή της σε διάφορες εφαρμογές.

Εικόνα 6 https://www.postgresql.org/

3.7 Αναπτυξη Βάση Δεδομένων με PostgreSQL

Η ανάπτυξη βάσης δεδομένων με PostgreSQL προσφέρει μια ισχυρή και ευέλικτη λύση για την αποθήκευση και διαχείριση δεδομένων. Η PostgreSQL είναι μια ανοιχτού κώδικα σχεσιακή βάση δεδομένων που υποστηρίζει προηγμένα χαρακτηριστικά, όπως πολύπλοκες ερωτήσεις SQL, συναλλαγές, και υποστήριξη για γεωχωρικά δεδομένα μέσω της επέκτασης PostGIS.

Πλεονεκτήματα της PostgreSQL

Επεκτασιμότητα: Υποστηρίζει μεγάλες ποσότητες δεδομένων και μπορεί να επεκταθεί για να καλύψει τις ανάγκες της επιχείρησης.

Ασφάλεια: Παρέχει προηγμένα μέτρα ασφάλειας, όπως κρυπτογράφηση και δυνατότητες ελέγχου πρόσβασης.

Συμβατότητα με πρότυπα sql: Υποστηρίζει τα πρότυπα SQL και διάφορες γλώσσες προγραμματισμού, όπως Python, Java και PHP.

Δημιουργία αντιγράφων ασφαλείας: Διαθέτει εργαλεία για εύκολη δημιουργία αντιγράφων ασφαλείας και αποκατάστασης δεδομένων.

Κοινότητα: Υπάρχει μια ενεργή κοινότητα που παρέχει υποστήριξη, τεκμηρίωση και επιπλέον επεκτάσεις.

Εφαρμογές: Η PostgreSQL χρησιμοποιείται ευρέως σε εφαρμογές που απαιτούν αυξημένη αξιοπιστία και απόδοση, όπως συστήματα διαχείρισης περιεχομένου, ηλεκτρονικό εμπόριο και εφαρμογές ανάλυσης δεδομένων.

Με την ανάπτυξη βάσεων δεδομένων με PostgreSQL, έχουμε στη διάθεσή μας μια ισχυρή και ευέλικτη πλατφόρμα για τη διαχείριση δεδομένων, με πολλές δυνατότητες που προάγουν την απόδοση και την ασφάλεια.

```
CREATE DATABASE mydatabase;
psql -d mydatabase
CREATE TABLE users (
  id SERIAL PRIMARY KEY,
  name VARCHAR(100),
  email VARCHAR(100) UNIQUE NOT NULL
);
```

Κώδικας 9 Παράδειγμα δημιουργίας βάσης δεδομένων και πίνακα

3.8 Η Τεχνολογία Docker

Το **Docker** είναι μια πλατφόρμα που επιτρέπει στους προγραμματιστές και διαχειριστές συστημάτων να δημιουργούν, να διανέμουν και να εκτελούν εφαρμογές μέσα σε containers. Τα containers είναι ελαφριά, απομονωμένα περιβάλλοντα που περιέχουν όλα τα απαραίτητα αρχεία, εξαρτήσεις και ρυθμίσεις για την εκτέλεση μιας εφαρμογής, ανεξάρτητα από το περιβάλλον στο οποίο θα τρέξει. Τα βασικά του τμήματα είναι:

το dockerfile: script που παρέχει πληροφορίες για το container που θα δημιουργηθεί.

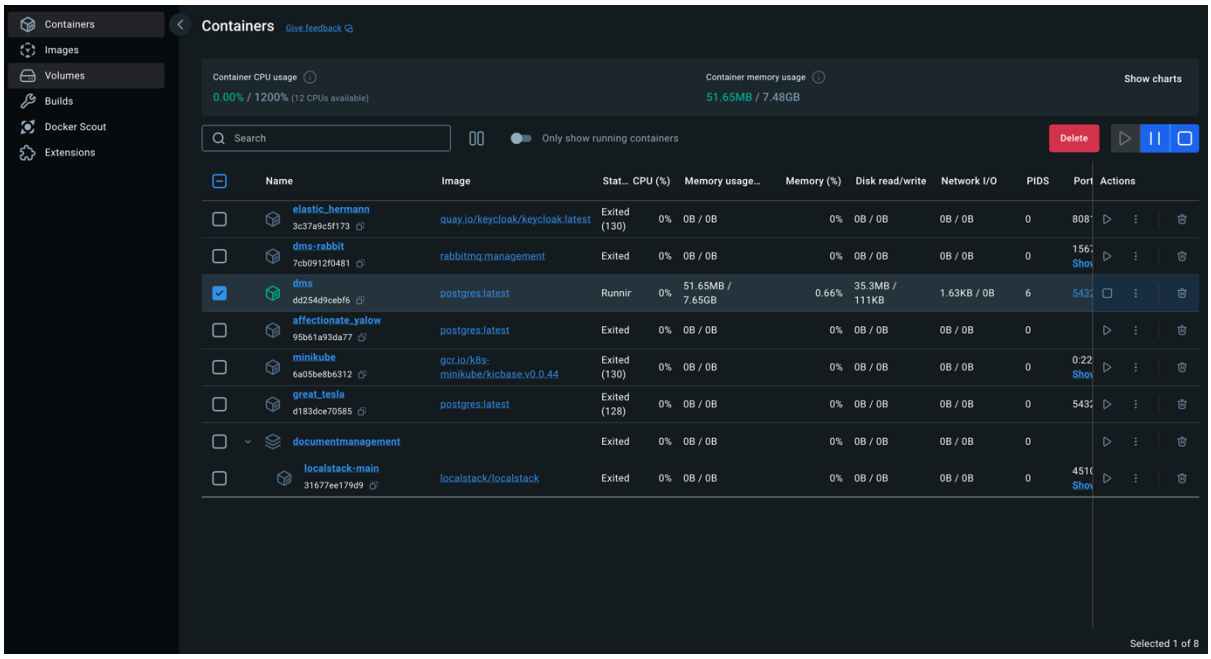
```
FROM openjdk:17-jdk-slim

WORKDIR /app

COPY target/documentmanagementsystem-0.0.1-snapshot.jar documentmanagementsystem-
0.0.1-snapshot.jar


ENTRYPOINT ["java", "-jar", "documentmanagementsystem-0.0.1-snapshot.jar"]

EXPOSE 8080
```

Κώδικας 10 Παράδειγμα από ένα dockerfile

το image: αρχείο που χρησιμοποιείται για να εκτελεστεί ο κώδικας του container

το container: η "περιβάλλον" μέσα στο οποίο τρέχουν οι εφαρμογές

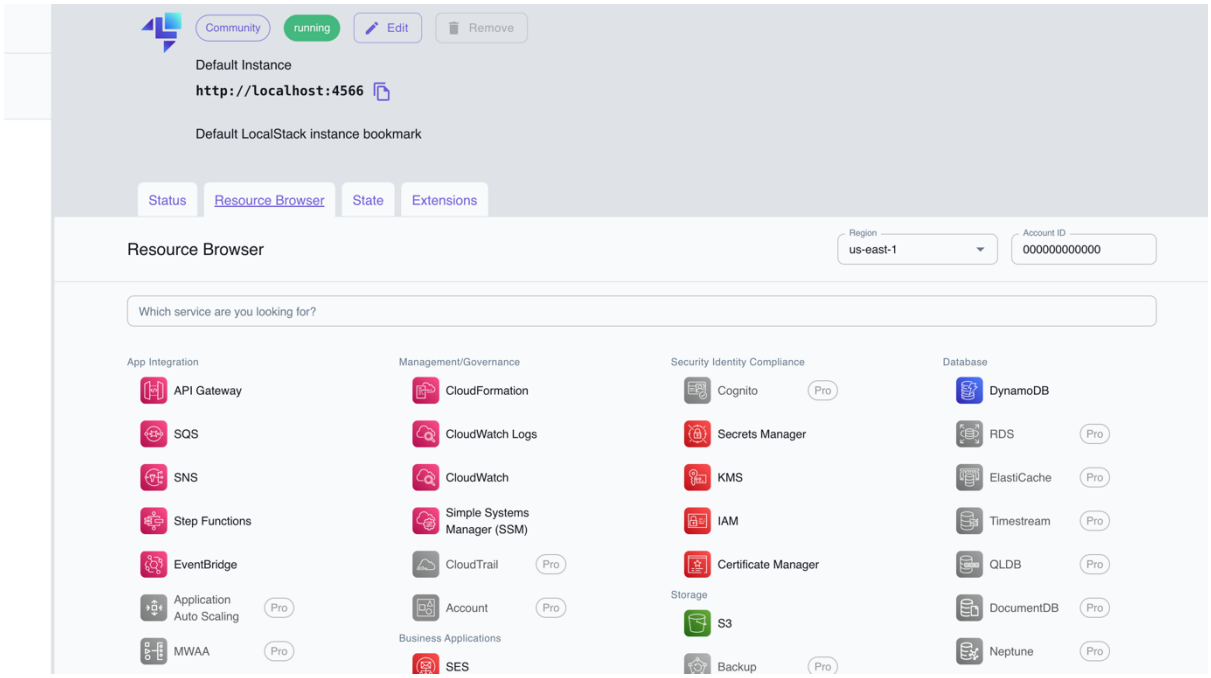
το network: συνδέει τα containers μεταξύ τους



Εικόνα 7 Docker Desktop console

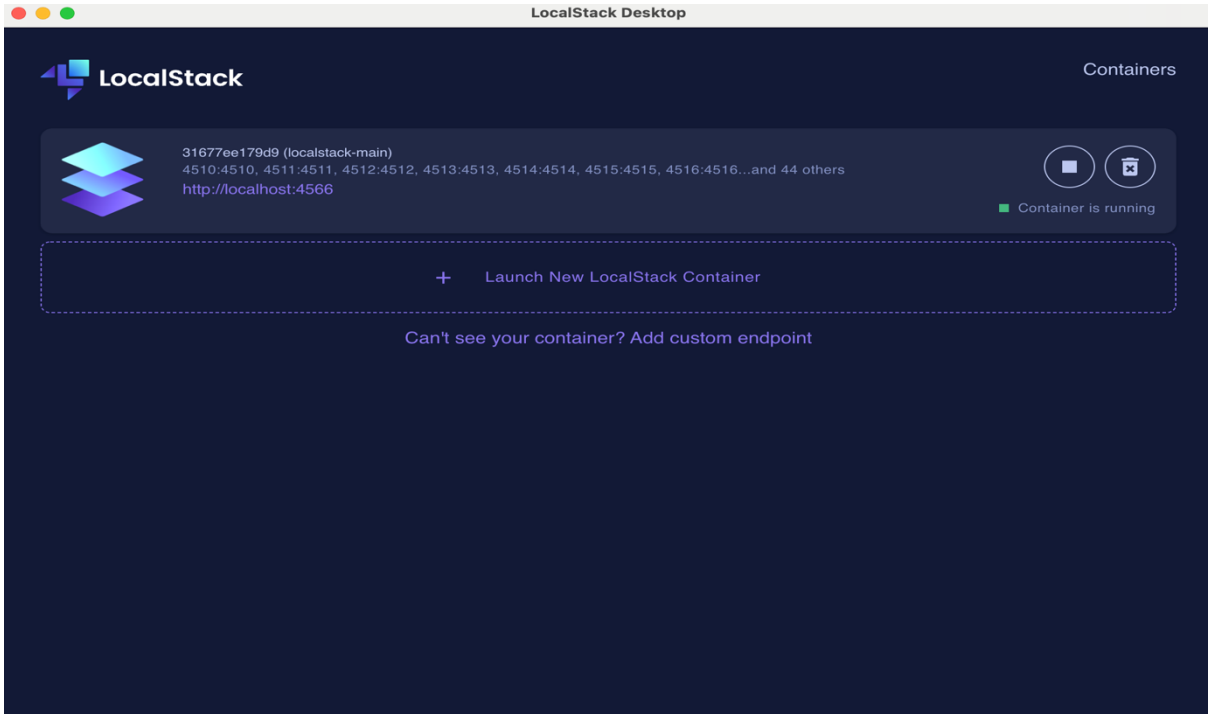
3.9 Η Τεχνολογία Local stack

Το **LocalStack** είναι ένα εργαλείο ανοιχτού κώδικα που επιτρέπει την τοπική ανάπτυξη και δοκιμή εφαρμογών που χρησιμοποιούν τις υπηρεσίες του AWS (Amazon Web Services). Προσομοιώνει πλήρως τις υπηρεσίες του AWS σε έναν τοπικό υπολογιστή, χωρίς να απαιτείται πραγματική σύνδεση με το AWS. Αυτό σημαίνει ότι οι προγραμματιστές μπορούν να δημιουργούν και να δοκιμάζουν τις εφαρμογές τους χωρίς να ανησυχούν για τα κόστη, τις χρεώσεις, ή τις καθυστερήσεις που σχετίζονται με τη χρήση των υπηρεσιών cloud.



Εικόνα 8 Dashboard του localstack Online

Το Localstack desktop μπορει να τρεξει και τοπικα στο υπολογιστημας μεσω docker image για το development environment οπου και θα μπορεσουμε να δοκιμασουμε τοπικα τα aws services που θελουμε να χρησιμοποιησουμε.



Εικόνα 9 Localstack Desktop

ΚΕΦΑΛΑΙΟ 4 Αρχιτεκτονικές Εφαρμογών

4.1 Εισαγωγή στις Αρχιτεκτονικές Λογισμικού

Οι αρχιτεκτονικές λογισμικού είναι οργανωμένοι τρόποι να σχεδιάζουμε και να οργανώνουμε το λογισμικό μας, επικεντρωμένοι στη διαχείριση των πολύπλοκων συστημάτων. Επιτρέπουν τη διάκριση λειτουργιών, τον έλεγχο της διασυνδεσιμότητας, και τη διαχείριση της εξέλιξης του λογισμικού. Συνήθως, κατασκευάζονται γύρω από βασικές αρχές σχεδίασης και προσεγγίσεις.

Η αρχιτεκτονική αναπαριστά ένα θεμέλιο στη δημιουργία κτιρίων, χώρων και συστημάτων που αντικατοπτρίζουν τη λειτουργικότητα, την αισθητική και τις ανάγκες των ανθρώπων. Μέσω της αρχιτεκτονικής, δημιουργούνται λύσεις για προβλήματα σχεδιασμού και δομής, απαιτώντας δημιουργικότητα, τεχνογνωσία και λειτουργική αποδοτικότητα.

Στην επίλυση προβλημάτων με την αρχιτεκτονική, οι αρχιτέκτονες χρησιμοποιούν μια σειρά από τεχνικές, όπως η ανάλυση αναγκών, η δημιουργία λειτουργικών προτύπων και η βελτιστοποίηση δομών για να προσεγγίσουν τα προβλήματα με οργάνωση και δημιουργικότητα.

Οι τρόποι δημιουργίας διαγραμμάτων για την αρχιτεκτονική περιλαμβάνουν τη χρήση γραφικών μοντέλων όπως διαγράμματα UML (Unified Modeling Language), διαγράμματα ροής, διαγράμματα διαδικασιών και δομικά διαγράμματα για να απεικονίσουν τη δομή και τη λειτουργία των συστημάτων.

Η περιγραφη των αρχιτεκτονικών μπορεί να γίνει με μεθόδους όπως:

- **C4 Model (Context, Containers, Components, Code):** Αξιοποιεί τέσσερα επίπεδα διαγραμμάτων για την αναπαράσταση της αρχιτεκτονικής, από το γενικό πλαίσιο μέχρι τον κώδικα.
- **arc42:** Επικεντρώνεται στην τεκμηρίωση αρχιτεκτονικών αποφάσεων και προϋποθέσεων, παρέχοντας ένα δομημένο πρότυπο για την περιγραφή του συστήματος.
- **4+1 View Model:** Περιλαμβάνει πέντε διαφορετικές προβολές (Use Case, Logical, Process, Development, Physical) για την περιγραφή του συστήματος από διαφορετικές οπτικές γωνίες.
- **RM-ODP (Reference Model for Open Distributed Processing):** Χρησιμοποιεί πέντε όψεις (Enterprise, Information, Computational, Engineering, Technology) για την καταγραφή κατανεμημένων συστημάτων.
- **TOGAF (The Open Group Architecture Framework):** Περιλαμβάνει τη διαδικασία ADM (Architecture Development Method) για την ανάπτυξη επιχειρησιακής αρχιτεκτονικής.
- **The Zachman Framework:** Προσφέρει μια δομή που συνδυάζει διαφορετικές προοπτικές και ερωτήσεις για την καταγραφή των αρχιτεκτονικών στοιχείων.
- **UML (Unified Modeling Language):** Χρησιμοποιεί διάφορα διαγράμματα για την καταγραφή διαφόρων πτυχών της αρχιτεκτονικής, όπως class diagrams, sequence diagrams, και component diagrams.
- **IEEE 1471 (ISO/IEC/IEEE 42010):** Εστιάζει στην περιγραφή αρχιτεκτονικών προβολών και των σχέσεων με τα ενδιαφερόμενα μέρη.
- **ICONIX:** Εστιάζει στη σχεδίαση λογισμικού χρησιμοποιώντας UML και παρέχει μια προσαρμοσμένη μέθοδο για την ανάλυση και σχεδίαση του λογισμικού. Περιλαμβάνει στάδια

όπως η ανάλυση απαιτήσεων, η δημιουργία λειτουργικών προτύπων και η λεπτομερής σχεδίαση, με στόχο την απλοποίηση και την κατανόηση της αρχιτεκτονικής του συστήματος.

4.2 Καταγραφή των αρχιτεκτονικών

Για την αποτελεσματική καταγραφή των αρχιτεκτονικών, οι μέθοδοι αυτές συχνά συνοδεύονται από τη χρήση εργαλείων όπως διαγράμματα UML, αρχεία κειμένου, διαγράμματα ροής, κώδικας ή άλλες μορφές τεκμηρίωσης που αντιπροσωπεύουν την αρχιτεκτονική και τις σχετικές αποφάσεις.

Καθώς η τεχνολογία εξελίσσεται, οι μέθοδοι καταγραφής αρχιτεκτονικών επίσης εξελίσσονται για να ανταποκριθούν στις αυξανόμενες απαιτήσεις και να προσφέρουν πιο αποτελεσματικά εργαλεία για τους αρχιτέκτονες και τις ομάδες ανάπτυξης.

Επιπλέον, η διαδικασία περιγραφής αρχιτεκτονικών συχνά εστιάζει σε βασικά στοιχεία όπως:

Συνεχής Επικοινωνία και Αναθεώρηση

Η τεκμηρίωση των αρχιτεκτονικών είναι μια συνεχής διαδικασία που πρέπει να αναθεωρείται και να ενημερώνεται κατά τη διάρκεια της ανάπτυξης και των αλλαγών στο σύστημα.

Αυτοματοποίηση Τεκμηρίωσης

Η χρήση εργαλείων που επιτρέπουν την αυτοματοποιημένη δημιουργία τεκμηρίωσης (όπως σχεδιασμός κώδικα που δημιουργεί αυτόματα διαγράμματα) μπορεί να βελτιώσει τη συνολική ποιότητα και συνέπεια της τεκμηρίωσης.

Ανάπτυξη Προτύπων και Οδηγών.

Η δημιουργία προτύπων για την τεκμηρίωση αρχιτεκτονικών βοηθά στην κατανόηση του πώς πρέπει να δομούνται και να περιγράφονται οι αρχιτεκτονικές αποφάσεις.

Ευελιξία και Ευκολία Χρήσης.

Η τεκμηρίωση πρέπει να είναι εύκολα προσβάσιμη και κατανοητή για τους ενδιαφερόμενους, όπως προγραμματιστές, μηχανικούς, και διαχειριστές συστημάτων.

Τέλος, η διαδικασία καταγραφής αρχιτεκτονικών είναι συχνά μια συνεχής εκτεταμένη πρακτική που απαιτεί συνεργασία και συνεχή βελτίωση για να ανταποκριθεί στις συνεχώς μεταβαλλόμενες ανάγκες ενός συστήματος ή μιας εφαρμογής.

4.3 Αρχιτεκτονική της εφαρμογής

Η εφαρμογή αυτη βασίζεται σε μια σύγχρονη και ανθεκτική αρχιτεκτονική που αξιοποιεί τις υπηρεσίες του AWS για να προσφέρει υψηλή διαθεσιμότητα, επεκτασιμότητα, ανοχή σε σφάλματα και ασφάλεια. Τα κύρια στοιχεία της αρχιτεκτονικής περιλαμβάνουν:

Frontend:

Η διεπαφή χρήστη (GUI) υλοποιείται με **Angular**, προσφέροντας μια διαδραστική εμπειρία χρήστη. Οι χρήστες μπορούν να ανεβάζουν επεξεργάζονται, διαγράφουν, κατεβάζουν αρχεία μέσω του frontend, το οποίο επικοινωνεί με το backend μέσω RESTful APIs στο αρχικό στάδιο τις εφαρμογής.

Backend:

Το backend υλοποιείται με **Spring Boot**, ένα ευέλικτο framework που επιτρέπει την ταχεία ανάπτυξη και διαχείριση API. Το backend διαχειρίζεται τα αιτήματα από το frontend, επιτρέπει την αποθήκευση,επεξεργασια, διαγραφη, αρχείων και μεταδεδομένων και παρέχει λειτουργίες αναζήτησης στο αρχικό στάδιο τις εφαρμογής αργοτερα θα υλοποιηθουν και άλλες επιλογες μαζι με το frontend.

Αποθήκευση Αρχείων:

Τα αρχεία αποθηκεύονται στο **Amazon S3**. Το S3 προσφέρει υψηλή διαθεσιμότητα και ανθεκτικότητα, με τα δεδομένα να αποθηκεύονται αυτόματα σε πολλαπλές ζώνες διαθεσιμότητας (Availability Zones) εντός της ίδιας περιοχής (region). Επιπλέον, το S3 παρέχει δυνατότητες όπως versioning και lifecycle management, για καλύτερη διαχείριση των δεδομένων.

Διαγραφή Αρχείων:

Η διαγραφή αρχείων αποτελεί κρίσιμη λειτουργία για τη διαχείριση του αποθηκευτικού χώρου και την ασφάλεια των δεδομένων. Στην αρχιτεκτονική της εφαρμογής, η διαγραφή αρχείων υποστηρίζεται μέσω του backend, το οποίο αλληλεπιδρά με την υπηρεσία αποθήκευσης στο AWS S3.

Επεξεργασία Αρχείων:

Η επεξεργασία αρχείων επιτρέπει στους χρήστες να τροποποιούν αρχεία που έχουν ήδη ανέβει στο σύστημα. Αυτή η λειτουργία περιλαμβάνει την ενημέρωση των αρχείων και την τροποποίηση των μεταδεδομένων τους.

Κατέβασμα Αρχείων:

Η δυνατότητα κατεβάσματος αρχείων επιτρέπει στους χρήστες να αποκτούν πρόσβαση και να κατεβάζουν αρχεία που έχουν ανεβάσει ή έχουν δικαίωμα πρόσβασης.

Αναζήτησή Αρχείου / Αρχείων:

Η αναζήτηση αρχείων επιτρέπει στους χρήστες να εντοπίζουν αρχεία με βάση συγκεκριμένα κριτήρια αναζήτησης.

Αποθήκευση Μεταδεδομένων:

Τα μεταδεδομένα των αρχείων αποθηκεύονται σε μια βάση δεδομένων **PostgreSQL** που φιλοξενείται στο **Amazon RDS**. Το RDS παρέχει υψηλή διαθεσιμότητα μέσω ανάπτυξης σε πολλαπλές ζώνες διαθεσιμότητας, αυτόματη δημιουργία αντιγράφων ασφαλείας, και διαχείριση αναβαθμίσεων λογισμικού.

Υποδομή:

Η εφαρμογή φιλοξενείται σε μια **EC2 instance** του AWS. Για να εξασφαλιστεί **υψηλή διαθεσιμότητα (High Availability)**, η EC2 instance θα αναπτυχθεί σε **δύο διαφορετικα Availability Zones**. Αυτό διασφαλίζει τη συνεχή λειτουργία της εφαρμογής ακόμα και σε περίπτωση αποτυχίας μιας από τις ζώνες διαθεσιμότητας.

Χρησιμοποιείται ένα **Auto Scaling Group** για την EC2 instance. Το Auto Scaling επιτρέπει την αυτόματη προσαρμογή του αριθμού των EC2 instances ανάλογα με τον φόρτο εργασίας, εξασφαλίζοντας την **επεκτασιμότητα (Scalability)** της εφαρμογής. Έτσι, η εφαρμογή μπορεί να χειριστεί αιχμές στην κίνηση χωρίς προβλήματα απόδοσης.

Elastic Load Balancer (ELB) διαχειρίζεται τη διανομή της κίνησης στα EC2 instances, εξασφαλίζοντας ομοιόμορφη κατανομή των αιτημάτων και **ανοχή σε σφάλματα (Fault Tolerance)**.

Ασφάλεια:

Η εφαρμογή είναι ασφαλισμένη μέσω των ενσωματωμένων υπηρεσιών ασφαλείας της AWS, όπως η κρυπτογράφηση των δεδομένων στο S3 και RDS, η χρήση AWS Identity and Access Management (IAM) για έλεγχο πρόσβασης, και η παρακολούθηση της δραστηριότητας μέσω του AWS CloudTrail.

4.4 Υψηλή διαθεσιμότητα, επεκτασιμότητα, ανοχή σε σφάλματα. High Availability, Scalability και Fault Tolerance

Η εφαρμογή έχει σχεδιαστεί με στόχο την υψηλή διαθεσιμότητα, την επεκτασιμότητα και την ανοχή σε σφάλματα, εξασφαλίζοντας ότι μπορεί να παραμείνει λειτουργική:

- **High Availability**: Η χρήση πολλαπλών Availability Zones τόσο για την EC2 instance όσο και για την RDS PostgreSQL βάση δεδομένων εξασφαλίζει ότι η εφαρμογή θα συνεχίσει να λειτουργεί ακόμα και αν μία από τις ζώνες γίνει μη διαθέσιμη.
- **Scalability**: Το Auto Scaling Group επιτρέπει την αυτόματη κλιμάκωση των EC2 instances ανάλογα με τον φόρτο εργασίας, εξασφαλίζοντας ότι η εφαρμογή μπορεί να διαχειριστεί αυξήσεις στην κίνηση χωρίς προβλήματα.
- **Fault Tolerance**: Ο Elastic Load Balancer και η πολλαπλή ανάπτυξη σε Availability Zones εξασφαλίζουν ότι τα αιτήματα των χρηστών θα κατευθύνονται στα λειτουργικά instances, ακόμα και αν ένα από αυτά αποτύχει.

4.5 Σχέδιο Ανάκαμψης από Καταστροφή Disaster Recovery Plan

Για την εξασφάλιση της συνέχειας της λειτουργίας της εφαρμογής σε περίπτωση καταστροφής, το AWS προτείνει τα εξής:

- **Backup and Restore**: Τα δεδομένα της PostgreSQL βάσης δεδομένων στο RDS πρέπει να υποστηρίζονται τακτικά μέσω αυτόματων αντιγράφων ασφαλείας. Τα backups αποθηκεύονται σε πολλαπλές ζώνες διαθεσιμότητας, επιτρέποντας την αποκατάσταση της βάσης δεδομένων σε περίπτωση αποτυχίας. Στο Αρχικό στάδιο δεν υλοποιείται
- **Cross-Region Replication**: Για το S3, μπορεί να ενεργοποιηθεί η **Cross-Region Replication**, όπου τα δεδομένα αποθηκεύονται σε ένα δεύτερο region για να εξασφαλιστεί η επιβίωση των δεδομένων ακόμα και σε περίπτωση περιφερειακής καταστροφής. Στο Αρχικό στάδιο δεν υλοποιείται.
- **Multi-AZ Deployment**: Το Amazon RDS παρέχει τη δυνατότητα ανάπτυξης σε πολλαπλές ζώνες διαθεσιμότητας (Multi-AZ), εξασφαλίζοντας ότι η βάση δεδομένων θα είναι διαθέσιμη ακόμα και σε περίπτωση αποτυχίας μιας από τις ζώνες. Στο Αρχικό στάδιο δεν υλοποιείται
- **Pilot Light**: Δημιουργία ελάχιστου αναγκαίου συνόλου υποδομών σε δεύτερο region που ενεργοποιούνται σε περίπτωση καταστροφής, μειώνοντας τον χρόνο αποκατάστασης. Στο Αρχικο σταδιο δεν υλοποιείται.
- **Failover and Recovery Testing**: Η δοκιμή των διαδικασιών failover για την EC2 instance και την RDS PostgreSQL βάση δεδομένων, εξασφαλίζοντας ότι το σύστημα μπορεί να μεταβεί σε εναλλακτικά resources με ελάχιστο downtime. Στο Αρχικό στάδιο δεν υλοποιείται.

Και έχουμε το σχήμα για το πως περίπου θα πρέπει να ξεκινήσουμε την υλοποίηση για το έργο μας. Για να καταλάβουμε τι ακριβώς γίνεται σε επίπεδο αρχιτεκτονικής. Χρησιμοποιούμε τα σύμβολα που έχει δημιουργήσει το AWS και έχουμε φτιάξει ένα απλό διάγραμμα για την πρώτη φάση του έργου έτσι ώστε να μπορούμε να το δοκιμάσουμε.

ΚΕΦΑΛΑΙΟ 5 Ανάλυση Απαιτήσεων

5.1 Γενικές απαιτήσεις

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι ο σχεδιασμός και η ανάπτυξη ενός κατανεμημένου συστήματος ελέγχου ανεπτυγμένο στο διαδίκτυο. Το σύστημα αυτό θα διευκολύνει τη γρήγορη διαχείριση , πιστοποίηση και επαλήθευση ηλεκτρονικών αρχείων, εγγράφων (π.χ. Εργασίες, πανεπιστήμιου, βαθμολογίες, έγγραφα, πτυχία , έγγραφα από δημόσιες υπηρεσίες κτλ).
Η εφαρμογή θα φέρει την ονομασία gdms.

5.2 Απαιτήσεις χρηστικότητάς

Το κατανεμημένο σύστημα ελέγχου, που αναπτύσσεται στο διαδίκτυο θα είναι προσβάσιμο από διαφορετικούς χρήστες με διαφορετικούς ρόλους. Γι' αυτό, είναι κρίσιμο να παρέχονται ομοιόμορφες, λειτουργικές και ευέλικτες διεπαφές χρήστη, ώστε να καλύπτονται οι ανάγκες του μεγαλύτερου δυνατού φάσματος χρηστών, ανάλογα με το επίπεδο εξοικείωσης τους με διαδικτυακές εφαρμογές. Οι λειτουργίες της εφαρμογής, που σχετίζονται με τη χρηστικότητα περιγράφονται παρακάτω:

- Η εφαρμογή θα πρέπει, όπου είναι εφικτό, να προσφέρει ένα περιβάλλον εργασίας με την κεντρική οθόνη μεσα από την οποία οι χρήστες θα μπορούν να πλοηγούνται μέσω μενού επιλογών, εργαλειοθηκών (toolbars), κουμπιών πλοήγησης και πλήκτρων συντομεύσεων.
- Εφόσον είναι δυνατό, θα πρέπει να παρέχονται γραφικές ή διαδικτυακές διεπαφές για την προεπισκόπηση των δεδομένων των εφαρμογών.
- Όλες οι εφαρμογές θα πρέπει να διαθέτουν εύχρηστα και ευδιάκριτα μενού πλοήγησης. Δηλαδή μενού με διαφορετικές επιλογές για την αυτοκατεύθυνση στην εφαρμογή, προκειμένου να είναι κατανοητά από τους χρήστες.
- Επιπλέον, είναι επιθυμητή η τυποποίηση όλων των παραπάνω στοιχείων για όλες τις εφαρμογές. Κάθε εφαρμογή θα πρέπει να προσφέρει βοήθεια χωρίς καθυστέρηση στην εκτέλεση ενεργειών, η η εργασιών όπως η καταχώριση αρχείου, η διαγραφή ή η τροποποίηση αρχείου, με τη χρήση βοηθητικών μηνυμάτων.
- Μηχανισμοί ελέγχου για την αποτροπή λάθους και της ποιότητας των δεδομένων, καθώς και διαφορετικά μηνύματα έλεγχου μέσω της οθόνης εργασίας.

5.3 Υποθέσεις - παραδοχές

Κατά τον σχεδιασμό και την ανάπτυξη ενός συστήματος, είναι σημαντικό να καθορίζονται υποθέσεις και παραδοχές. Έχουμε τις λειτουργικές και μη λειτουργικές προδιάγραφες που θα πρέπει να πάρουμε.
Αυτές οι υποθέσεις λειτουργούν ως βάση για τον σχεδιασμό και τη λήψη αποφάσεων κατά τη διάρκεια της διαδικασίας ανάπτυξης. Στο πλαίσιο αυτό, θα εξετάσουμε μερικές βασικές υποθέσεις και παραδοχές που μπορεί να καθορίσουμε για ένα σημαντικο έργο και μετα αναλύουμε τις λειτουργικές και μη λειτουργικές προδιαγραφές:

Υπόθεση Χρήστη

Προσδιορίζουμε ποιοι είναι οι τελικοί χρήστες του συστήματος. Ποιες είναι οι ανάγκες, προσδοκίες και προτεραιότητες τους.

Περιβάλλον Λειτουργίας

Προσδιορίζουμε το περιβάλλον στο οποίο το σύστημα θα λειτουργεί. Ποιες είναι οι εξωτερικές συνθήκες και παράγοντες που μπορεί να επηρεάσουν την απόδοση του.

Υποθέσεις Τεχνολογίας

Καθορίζουμε τις τεχνολογίες που θα χρησιμοποιηθούν και καταθέτουμε υποθέσεις για τη διαθεσιμότητα, τη συμβατότητα και τη σταθερότητά τους.

Περιορισμοί Πόρων

Αναγνωρίζουμε περιορισμούς όπως οι οικονομικοί πόροι, ο χρόνος και οι ανθρώπινοι πόροι που είναι διαθέσιμοι για το έργο.

Υποθέσεις Ασφαλείας

Καθορίζουμε υποθέσεις σχετικά με την ασφάλεια του συστήματος, συμπεριλαμβανομένων των μέτρων ασφαλείας και των αντιμέτρων για την προστασία από απειλές.

Υποθέσεις Αλληλεπίδρασης

Προσδιορίζουμε πώς αλληλοεπιδρούν διάφορα στοιχεία του συστήματος, όπως οι διάφορες υπηρεσίες, οι βάσεις δεδομένων και οι χρήστες.

Οι παραπάνω υποθέσεις και παραδοχές αποτελούν τη βάση για τον σχεδιασμό και την υλοποίηση του συστήματος. Είναι σημαντικό να τις καταγράφουμε και να τις επανεξετάζουμε καθώς το έργο εξελίσσεται, προκειμένου να διασφαλίσουμε τη συνέχεια και την αποτελεσματικότητα του σχεδιασμού μας.

5.5 Λειτουργικές προδιαγραφές

Υποθέτουμε ότι οι χρήστες θα παρέχουν αξιόπιστα προσωπικά στοιχεία κατά τη διαδικασία εγγραφής και πιστοποίησης. Παραδεχόμαστε ότι η ασφάλεια των δεδομένων και η προστασία της ιδιωτικότητας είναι πρωταρχικής σημασίας και θα ληφθούν τα κατάλληλα μέτρα για αυτό. Ο κάθε χρήστης δεν θα πρέπει να μπορεί να δει τι κάνει η εφαρμογή χωρίς να είναι εγγεγραμμένος ή να έχει κάποια ειδικά δικαιώματα . Η εγγραφη του χρηστη θα γινεται στην σελιδα Register οπου ο χρηστης θα εισαγει τα στειχεια του ετσι ώστε να μπορει να δημιουργησει λογαριασμο και να μπορεί να έχει την δυνατότητα να ανεβάσει αρχεία στο σύστημα.

Εγγραφή και Πιστοποίηση Χρηστών

Οι χρήστες θα μπορούν να δημιουργήσουν λογαριασμούς και να επιβεβαιώσουν την ταυτότητά τους.

Αποθήκευσή Αρχείου

Ο χρήστης θα μπορεί να υποβάλει ένα αρχείο στο σύστημα.

Το αρχείο θα πρέπει να περιέχει τα απαραίτητα μεταδεδομένα, όπως τίτλος, περιγραφή και συνημμένα αρχεία.

Διαχείριση Αρχείου

Ο χρήστης μπορεί να διαχειριστεί το Αρχείο

Διαχείριση Αποτελεσμάτων Αναζητήσεις Αρχείου

Ο Χρήστης θα μπορεί να διαχειρίζεται τα αρχεία καθώς και την επεξεργασία αρχείων.

Αναφορές και Ιστορικό

Οι χρήστες θα έχουν πρόσβαση σε στατιστικά δεδομένα και ιστορικό Αρχείου.

Διαχείριση Λογαριασμού

Οι χρήστες θα μπορούν να διαχειρίζονται τα προσωπικά τους στοιχεία και τις ρυθμίσεις του λογαριασμού τους.

Επικοινωνία και Ειδοποιήσεις

Ανακοινώσεις και ειδοποιήσεις προς τους χρήστες για σημαντικά γεγονότα ή αλλαγές στο σύστημα.

Λειτουργικές Παραδοχές για το Frontend

- Εκτέλεση Αιτημάτων Χρηστών: Το Frontend θα εκτελεί αιτήματα χρηστών για την αποστολή και λήψη δεδομένων από το Backend.

- Διαχείριση Χρήστη: Θα παρέχει λειτουργίες διαχείρισης λογαριασμών χρηστών, όπως σύνδεση και αποσύνδεση, ανάκτηση προσωπικών δεδομένων, κλπ.

- Διεπαφή Χρήστη: Το Frontend θα δινει την δυνατοτητα στον χρηστη να βλεπει τα δεδομενα του.

Λειτουργικές Παραδοχές για το Backend

- Επεξεργασία Δεδομένων: Το Backend θα εκτελεί λειτουργίες επεξεργασίας και διαχείρισης των δεδομένων που λαμβάνει από το Frontend.
- Επικοινωνία με Υπηρεσίες: Θα αλληλοεπιδρά με άλλες υπηρεσίες, όπως το AWS, για την αποθήκευση και επεξεργασία δεδομένων.
- Απόδοση Αιτημάτων: Θα ανταποκρίνεται σε αιτήματα από το Frontend με αξιοπιστία και ταχύτητα.

5.6 Μη Λειτουργικές Προδιαγραφές

Ασφάλεια Δεδομένων

Το σύστημα θα πρέπει να παρέχει υψηλό επίπεδο ασφάλειας των δεδομένων, προστατεύοντας τα από ανεπιθύμητη πρόσβαση ή επεξεργασία.

Αξιοπιστία Καταγραφής

Οι καταγραφές στη Βαση Δεδομενων θα πρέπει να είναι αξιόπιστες και αδιαμφισβήτητες, παρέχοντας ακεραιότητα στα δεδομένα.

Ευχρηστία Εφαρμογής

Η εφαρμογή θα πρέπει να είναι εύχρηστη και φιλική προς τον χρήστη, προσφέροντας απλότητα και ευκολία στη χρήση της.

Διαθεσιμότητα

Το σύστημα θα πρέπει να είναι διαθέσιμο και να λειτουργεί με σταθερότητα χωρίς παρεμβολές.

Αυτές οι λειτουργικές και μη λειτουργικές προδιαγραφές είναι ουσιώδεις για την ανάπτυξη και τη λειτουργία ενός αξιόπιστου κατανεμημένου συστήματος ελέγχου.

Μη Λειτουργικές Παραδοχές για το Frontend

- Αισθητική και Ευχρηστία: Το Frontend πρέπει να παρέχει μια ευχάριστη εμπειρία χρήσης με κατάλληλη αισθητική και ευχρηστία.
- Συμβατότητα Περιηγητών: Πρέπει να είναι συμβατό με διάφορους περιηγητές για να υποστηρίζει διαφορετικές πλατφόρμες.
- Απόκριση Χρήστη: Το Frontend θα παρέχει άμεση ανταπόκριση στις ενέργειες των χρηστών χωρίς καθυστερήσεις.

Μη Λειτουργικές Παραδοχές για το Backend

- Ασφάλεια Δεδομένων: Πρέπει να διασφαλίζει την ασφάλεια των δεδομένων κατά την αποθήκευση και τη μεταφορά τους.
- Απόδοση Συστήματος: Πρέπει να είναι αποδοτικό και να ανταποκρίνεται στις απαιτήσεις χρόνου απόκρισης του συστήματος.
- Επεκτασιμότητα: Πρέπει να είναι εύκολα επεκτάσιμο για να υποστηρίζει μελλοντικές λειτουργίες και αυξημένες ανάγκες του συστήματος.

Αυτές οι παραδοχές περιγράφουν τις αναμενόμενες λειτουργίες και τις απαιτήσεις για το Frontend και το Backend, καθώς και τις ανεπίσημες προσδοκίες για την απόδοση και τη συμπεριφορά τους.

ΚΕΦΑΛΑΙΟ 6 Σχεδιασμός Εφαρμογής

6.1 Σχεδιασμός εφαρμογής

Το συστημα που δημιουργειται είναι σε μια μορφη πρωτοτυπου με τις βασικες λειτουργιες και θα μπορει να παρουσιαστει σαν ένα POC (proof of concept). Θα μπορει να γινει επεκταση του συστηματος αναλογα με τις αναγκες. Η εφαρμογη θα ανεβει (deploy) σε ένα cluster στο AWS και από εκει θα μπορει να είναι διαθεσιμη για κάθε χρηστη που εχει μια σταθερη συνδεση στο Internet.
Εφοσον η εφαρμογη είναι ανεβασμενη στο AWS από εκει εχουμε το μοντελο Shared Responsibility οπου θα υπαρχει μια ομαδα που θα είναι υπευθυνη για την διαχειριση από την μερια τις αναπτυξης του εργου για την καλη λειτουργεια τις εφαρμογης, καθως και για τις διαφορες παραμετροποιησεις του συστηματος, και από την άλλη τα υπολοιπα που είναι σχετικα με τον server το cluster τις αδειες για το software που θα χρησιμοποιηθει θα είναι στην ευθυνη του AWS .

6.2 Το Μοντέλο UML

Η Unified Modeling Language (UML) αποτελεί ένα διεθνές πρότυπο γλώσσας μοντελοποίησης που χρησιμοποιείται για τον σχεδιασμό και την ανάλυση συστημάτων. Η χρήση του UML παρέχει πολλαπλά οφέλη για την κατανόηση, τον σχεδιασμό και την υλοποίηση του συστήματος που δημιουργείτε. Παρακάτω αναλύονται μερικά από τα κύρια οφέλη του UML στο πλαίσιο του συστήματός μας.

1. **Κατανόηση του Συστήματος**

   Το UML παρέχει γραφικά μοντέλα που είναι ευανάγνωστα και προσφέρουν μια οπτική αναπαράσταση του συστήματος. Αυτό βοηθά στην καλύτερη κατανόηση των διαφόρων στοιχείων και σχέσεων του.

2. **Σχεδιασμός Συστήματος**

   Το UML παρέχει διαγράμματα όπως τα διαγράμματα κλάσεων, ακολουθίας και διαγράμματα δραστηριοτήτων που επιτρέπουν τον σχεδιασμό και την καθοδήγηση του συστήματος πριν από την υλοποίηση.

3. **Καταγραφή Απαιτήσεων**

   Το UML επιτρέπει τη δημιουργία διαγραμμάτων όπως το διάγραμμα δραστηριοτήτων και το διάγραμμα χρήστη για να καταγραφούν οι απαιτήσεις του χρήστη και οι διαδικασίες.

4. **Ενίσχυση Συνεργασίας Ομάδας**

   Το UML προσφέρει ένα κοινό γλωσσικό πλαίσιο για τους μελετητές, τους σχεδιαστές και τους προγραμματιστές, βελτιώνοντας την επικοινωνία και τη συνεργασία σε ομαδικό επίπεδο.

5. **Συντήρηση και Επέκταση**

   Τα διαγράμματα κλάσεων και αντικειμένων του UML καθιστούν ευκολότερη τη συντήρηση του κώδικα και την προσθήκη νέων χαρακτηριστικών χωρίς να παραβιάζεται η υπάρχουσα λειτουργικότητα.

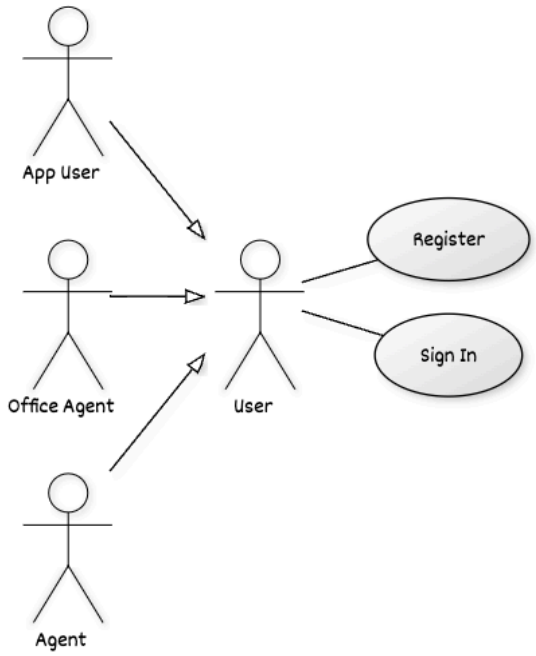
6. **Ανίχνευση Σφαλμάτων**

   Η ανάλυση των διαγραμμάτων ακολουθίας μπορεί να βοηθήσει στον εντοπισμό και τη διόρθωση σφαλμάτων στην επικοινωνία μεταξύ στοιχείων του συστήματος.

Με τη βοήθεια του UML, το σύστημα που δημιουργούμε μπορούμε να σχεδιάσουμε , να αναλύσουμε και να υλοποιήσουμε με περισσότερη αποτελεσματικότητα, ενισχύοντας την κατανόηση του κώδικα και βελτιώνοντας τη συνολική διαδικασία ανάπτυξης.

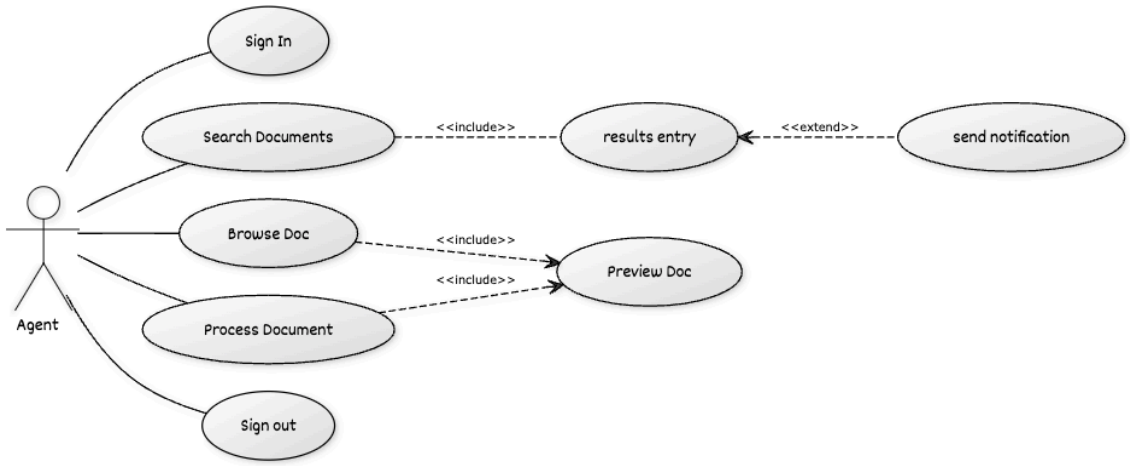
6.3 Περιπτώσεις χρήσης

Στο αρχικο στάδιο της διαδικασίας ανάπτυξης του μοντέλου Use Case (Περιπτώσεων Χρήσης) έναρξης με τον εντοπισμό των χρηστών (actors) του συστήματος. Αναφερόμαστε σε ένα user της εφαρμογης με ένα ρόλο, ο οποίος αλληλοεπιδρά με το σύστημα. Για να εντοπίσουμε τους χειριστές, εξετάζουμε τις απαιτήσεις όπως περιγράφονται στο πρόβλημα. Κάθε χειριστής συμμετέχει σε μία περίπτωση χρήσης, η οποία μπορεί να θεωρηθεί ως μια ακολουθία αλληλεπιδράσεων μεταξύ ενός ή περισσότερων χρηστών. Στην προσπάθεια να προσδιορίσουμε τους χειριστές, είναι σημαντικό να εξασφαλίσουμε ότι πληρούν τις προϋποθέσεις που καλύπτουν όλες τις ανάγκες της εφαρμογής μας.
Επιπλέον, θα υπάρχει η δυνατότητα να μπορούν άλλα συστήματα να επικοινωνούν με την εφαρμογή, διάφορες εξωτερικές εφαρμογές ή διαδικασίες όπως η ληψη back up, ο συνχρονισμός των δεδομένων σε μια δεύτερη βάση δεδομένων που θα μπορεί να χρησιμοποιείται για Read μονο.
Κατά την δημιουργία του μοντέλου Use Case πρεπει να ορισουμε τα εξής σημεια:
- Ποιοι είναι οι χρήστες του συστήματος.
- Με ποιον τρόπο αλληλεπιδρούν με το σύστημα.
Για το έργο αυτό παρουσιάζω μερικές περιπτώσεις χρήσης στις εικόνες.



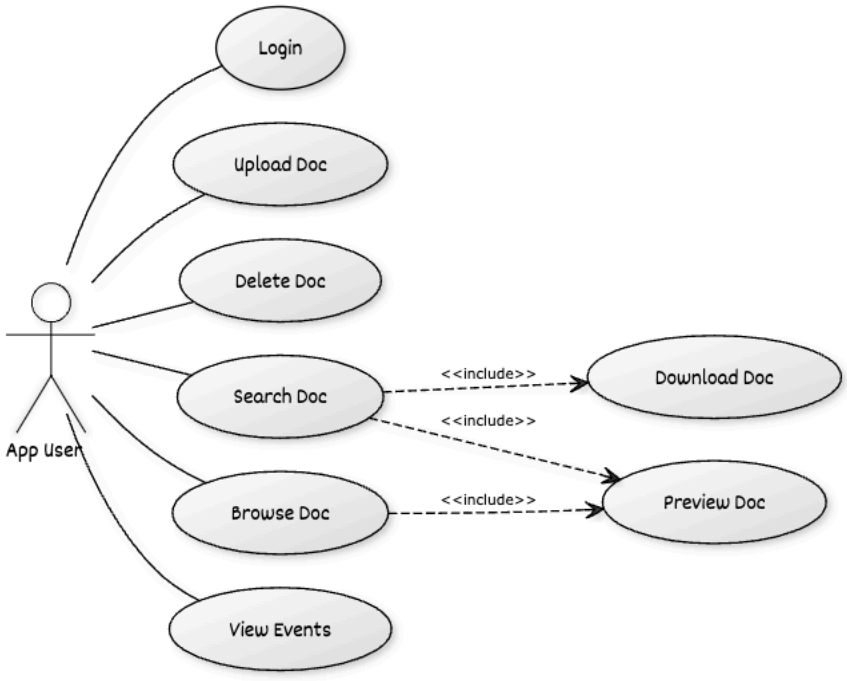
Εικόνα 10 Περίπτωση χρήσης Ομαδες χρηστη

Θα υπάρχουν 3 ομάδες χρηστών, που θα μπορούν να χρησιμοποιούν το σύστημα. Αυτές φαίνονται στην εικόνα 11. Οι χρήστες app user, που θα είναι οι κανονικοι χρήστες, οι χρήστες office agent, που θα έχουν δικαιώματα για τυχον αλλαγές στα έγγραφα ή για διορθώσεις και οι χρήστες agent, όπου θα είναι και οι διαχειριστές του συστήματος και θα εκτελούν τεχνικες εργασίες.



Εικόνα 11 Περιπτώση χρήσης Agent χρήστη

Στην περίπτωση agent χρήστη που βλέπουμε στην εικόνα 12 έχουμε περιληπτικά τις εργασίες που θα μπορεί να κάνει στην εφαρμογή.



Εικόνα 12 περιπτώση χρήσης App user

Χρήστης κάνει Αποθηκευση αρχειου

Ο χρήστης θα ανεβάσει το αρχείο του στο AWS σε ένα Bucket που θα έχει το μονοπάτι μόνο για τον χρήστη . Εφόσον αυτό γίνει και όλα πάνε καλά ο χρήστης θα πάρει μήνυμά επιβεβαίωσης ότι το ανέβασμα είχε επιτυχία.

Το Bucket στο S3 θα το έχουμε δημιουργήσει εμείς για τον χρήστη σε πρώτη φάση έτσι ώστε να μπορούμε να δοκιμάσουμε την λειτουργεία ανεβάσματος αρχείου γρήγορα.

Στην υλοποίηση μελλοντικά ο χρήστης με το που κάνει εγγραφή στην εφαρμογη θα δημιουργει χωρο ένα φακελο στο S3 που θα αποθηκεύονται μόνο τα δικα του έγγραφα. Όταν θα κάνει upload τα έγγραφα του η σε περίπτωση που θα κάνει μια αναζήτηση θα κανει αναζητηση μονο στον φακελο αυτο.

Αποτέλεσμα θα είναι ο μειωμένος χρόνος αναζήτησης για τον χρήστη.

Frontend: Ο χρήστης χρησιμοποιεί το Angular UI για να επιλέξει και να ανεβάσει αρχεία.

Backend: Το Spring Boot backend δέχεται τα αρχεία, τα αποθηκεύει στο Amazon S3 και καταγράφει τα μεταδεδομένα στη βάση δεδομένων PostgreSQL.

Amazon S3: Παρέχει αποθήκευση αρχείων με υψηλή διαθεσιμότητα και ανθεκτικότητα(high availability, scalability, durability) .

Amazon RDS: Αποθηκεύει τα μεταδεδομένα των αρχείων, όπως πληροφορίες όπως το όνομα του αρχείου και την ημερομηνία δημιουργίας. Η βαση δεδομενων postgresql θα είναι ανεβασμενη εκει.

Χρήστης κάνει αναζήτηση αρχείου

Έστω ότι ο χρήστης έχει ανεβάσει το αρχείο κάποια στιγμή και θέλει να δει τι έχει ανεβάσει.

Θα μπορεί μέσω τις διεπαφής να κάνει κάποια αναζήτηση στα αρχεία που έχει ήδη ανεβάσει και να δει το status τους. Η διεπαφη θα κάνει ένα Rest request όπου θα ενεργοποιεί την αναζήτηση σε ένα bucket για το αρχείο που ψάχνει. Θα ενεργοποιεί μια λάμδα Function που θα κάνει αναζήτηση με βάση το όνομα του αρχείου μέσα στο bucket. Σε επιτυχή αναζήτηση θα επιστρέφει το αρχείο για προβολή. Σε μη επιτυχή θα επιστρέφει μήνυμα αποτυχίας εύρεσης αρχείου.

Frontend: Ο χρήστης εισάγει τα κριτήρια αναζήτησης μέσω του Angular UI.

Backend: Το Spring Boot backend εκτελεί την αναζήτηση μέσω των δεδομένων που είναι αποθηκευμένα στη βάση δεδομένων PostgreSQL. Στη συνέχεια, παρέχει τα αποτελέσματα στον χρήστη.

Amazon RDS: Ερωτήματα στη βάση δεδομένων για την αναζήτηση των αρχείων και των σχετικών μεταδεδομένων.

Amazon S3: Η αναζήτηση δεν επηρεάζει άμεσα το S3, αλλά το backend μπορεί να ανακτά πληροφορίες από το S3 αν χρειάζεται.

Χρήστης ξεκινά επεξεργασία αρχείου

Η επεξεργασία αρχείων περιλαμβάνει την τροποποίηση ή αναβάθμιση των αρχείων που έχουν ήδη αποθηκευτεί. Αυτή η διαδικασία μπορεί να περιλαμβάνει την αλλαγή του περιεχομένου, την αναβάθμιση του αρχείου ή την αλλαγή των μεταδεδομένων.Ο χρήστης ξεκινά την επεξεργασία ενός αρχείου που έχει ανεβάσει κάποιος άλλος χρήστης η και ο ίδιος χρήστης στον S3 Bucket.

Frontend: Ο χρήστης φορτώνει την τροποποιημένη έκδοση του αρχείου μέσω του Angular UI.

Backend: Το Spring Boot backend διαχειρίζεται την υποβολή της επεξεργασμένης έκδοσης του αρχείου και την ενημέρωση των σχετικών μεταδεδομένων στη βάση δεδομένων PostgreSQL.

Amazon S3: Ενημερώνει την αποθήκη S3 με την νέα έκδοση του αρχείου, διατηρώντας την προηγούμενη έκδοση εάν είναι ενεργοποιημένο το versioning.

Amazon RDS: Ενημερώνει τα μεταδεδομένα στην βάση δεδομένων για να αντικατοπτρίσει τις αλλαγές.

Χρήστης κανει διαγραφη αρχείου

Η λειτουργία διαγραφής αρχείων επιτρέπει στους χρήστες να αφαιρούν αρχεία από το σύστημα. Όταν ένα αρχείο διαγράφεται, πρέπει να διασφαλίζεται ότι η διαγραφή είναι ολοκληρωμένη τόσο από το frontend όσο και από το backend και ότι τα δεδομένα δεν παραμένουν στο σύστημα, αποφεύγοντας έτσι την κατανάλωση αποθηκευτικού χώρου.

- **Frontend:** Οι χρήστες αλληλεπιδρούν με τη διεπαφή χρήστη Angular για την επιλογή και επιβεβαίωση της διαγραφής ενός αρχείου.
- **Backend:** Το Spring Boot backend διαχειρίζεται τα αιτήματα διαγραφής, αποστέλλει εντολές στο Amazon S3 για την αφαίρεση του αρχείου και ενημερώνει τη βάση δεδομένων PostgreSQL μέσω του Amazon RDS για να αφαιρέσει τα σχετικά μεταδεδομένα.
- **Amazon S3:** Διαγράφει το αρχείο από την αποθήκη S3, διασφαλίζοντας ότι τα δεδομένα δεν παραμένουν.
- **Amazon RDS:** Ενημερώνει την βάση δεδομένων για να αφαιρέσει τις σχετικές εγγραφές μεταδεδομένων.

Αποθήκευσή μεταδεδομενων αρχείου

Η αποθήκευση μεταδεδομένων αφορά την καταγραφή πληροφοριών που σχετίζονται με τα αρχεία, όπως οι περιγραφές, οι ημερομηνίες δημιουργίας, και άλλες σχετικές πληροφορίες.

- **Frontend:** Ο χρήστης μπορεί να προσθέσει ή να τροποποιήσει μεταδεδομένα μέσω του Angular UI.
- **Backend:** Το Spring Boot backend επεξεργάζεται και αποθηκεύει τα μεταδεδομένα στη βάση δεδομένων PostgreSQL.
- **Amazon RDS:** Διατηρεί και διαχειρίζεται τα μεταδεδομένα, διασφαλίζοντας την ακρίβεια και την πλήρη καταγραφή.
- **Amazon S3:** Δεν επηρεάζεται άμεσα, αλλά τα μεταδεδομένα που σχετίζονται με τα αρχεία μπορεί να περιλαμβάνουν πληροφορίες που είναι αποθηκευμένες στο S3.

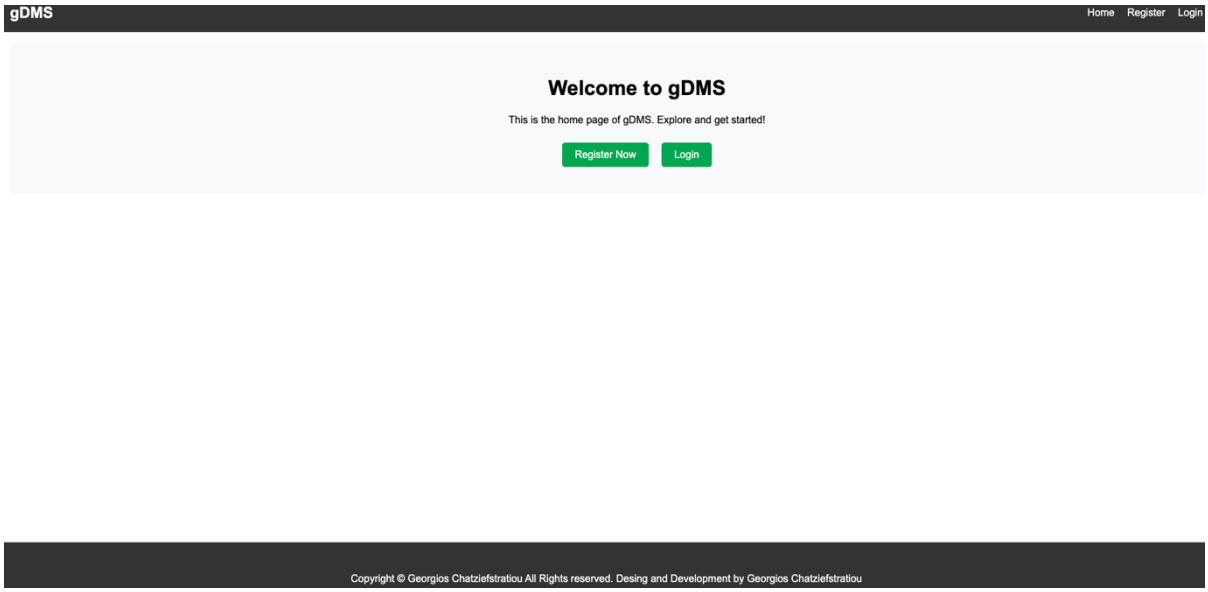
ΚΕΦΑΛΑΙΟ 7 ΣΧΕΔΙΑΣΗ ΟΘΟΝΩΝ ΣΥΣΤΗΜΑΤΟΣ

7.1 Angular Σχεδίαση οθονών συστήματος

Η σχεδίαση οθονών συστήματος σε μια εφαρμογή Angular αποτελεί κρίσιμο βήμα στη διαδικασία ανάπτυξης λογισμικού. Αρχικά, οι σελίδες της εφαρμογής θα είναι απλές και λειτουργικές, με στόχο να κατανοήσουμε τις βασικές λειτουργίες της και πώς αυτές αλληλεπιδρούν με τον χρήστη.
Η προσέγγιση αυτή μας επιτρέπει να εστιάσουμε στην κατανόηση της ροής της εφαρμογής και της αλληλεπίδρασής της με τα δεδομένα, χωρίς να αποσπαστούμε από τις πιο περίπλοκες λεπτομέρειες του UI/UX. Η απλότητα των αρχικών σελίδων θα διευκολύνει την αποτύπωση των βασικών χαρακτηριστικών της εφαρμογής και θα μας βοηθήσει να εντοπίσουμε τυχόν προβλήματα ή περιοχές που χρήζουν βελτίωσης.
Καθώς προχωράμε στην ανάπτυξη, οι σελίδες αυτές θα εξελιχθούν με περισσότερες λειτουργίες και προσαρμογές, ανταγωνιζόμενες τις ανάγκες των χρηστών και τις απαιτήσεις του έργου. Με αυτόν τον τρόπο, θα μπορέσουμε να διασφαλίσουμε ότι η εφαρμογή μας θα είναι όχι μόνο λειτουργική, αλλά και φιλική προς τον χρήστη.

7.2 Αρχική Σελίδα

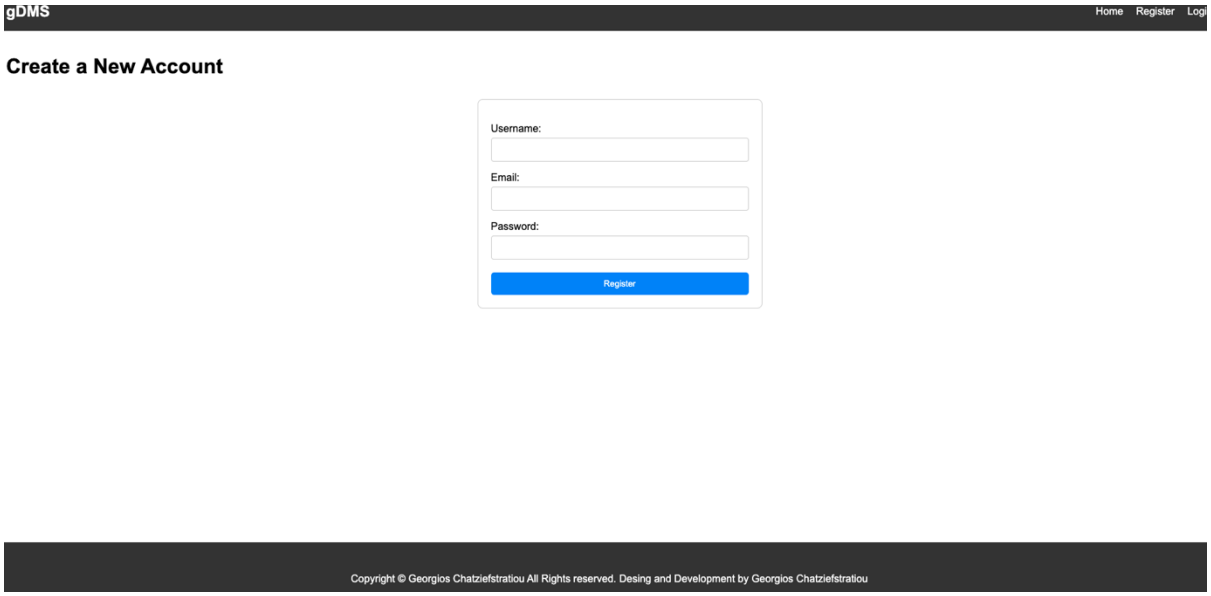
Η αρχική σελίδα λειτουργεί ως πύλη εισόδου για το υπόλοιπο περιεχόμενο του ιστότοπου, προσφέροντας μια συνοπτική επισκόπηση των σημαντικών πληροφοριών και λειτουργιών. Περιλαμβάνει συνδέσμους ή μενού πλοήγησης που καθοδηγούν τον χρήστη σε άλλες ενότητες ή σελίδες του ιστότοπου (σε μελλοντική επέκταση)
Η διάταξή της συχνά περιλαμβάνει γραφικά, κείμενα και διαδραστικά στοιχεία που προσελκύουν την προσοχή του επισκέπτη, ενώ ταυτόχρονα επισημαίνει το σκοπό και την ταυτότητα του ιστότοπου. Η αρχική σελίδα είναι κρίσιμη για τη συνολική εμπειρία του χρήστη, καθώς καθορίζει τις πρώτες εντυπώσεις και συμβάλλει στην εύκολη και γρήγορη πρόσβαση στις πληροφορίες που αναζητά.



Εικόνα 13 Αρχική Σελίδα

7.3 Σελίδα Εγγραφής

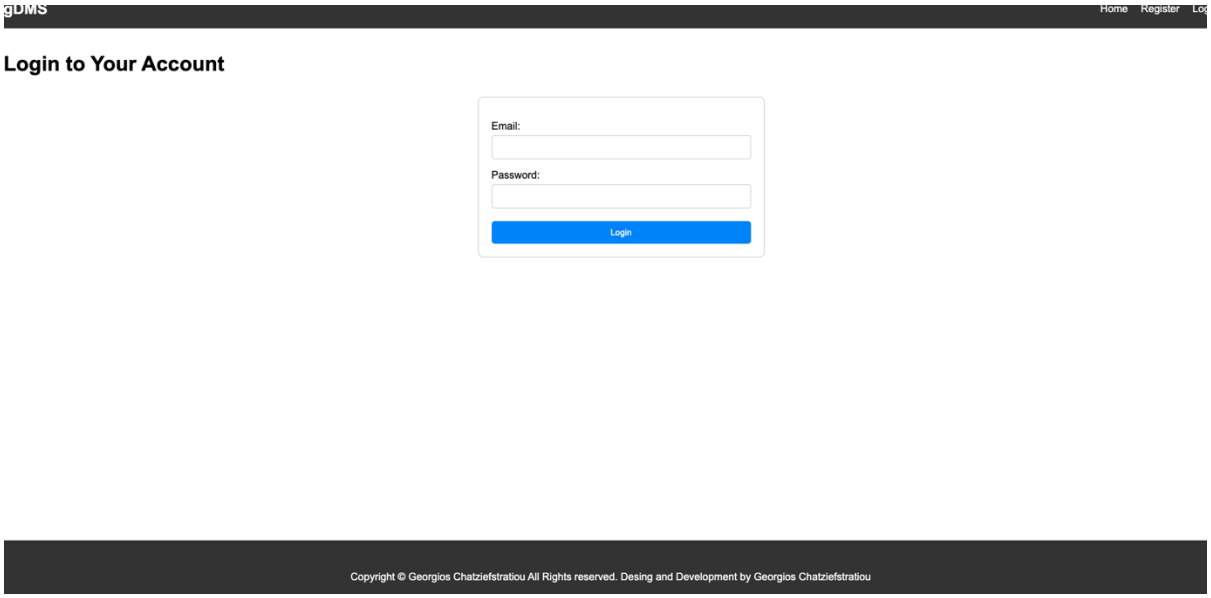
Η σελίδα εγγραφής επιτρέπει στους νέους χρήστες να δημιουργήσουν έναν λογαριασμό στην εφαρμογή. Ο χρήστης θα κληθεί να συμπληρώσει τα απαραίτητα πεδία, όπως το όνομα χρήστη, τη διεύθυνση ηλεκτρονικού ταχυδρομείου και τον κωδικό πρόσβασης. Μόλις ολοκληρωθεί η διαδικασία εγγραφής, ο χρήστης θα λάβει μια επιβεβαίωση μέσω email, διασφαλίζοντας ότι η διαδικασία είναι ασφαλής και αξιόπιστη.



Εικόνα 14 Σελίδα Εγγραφής

7.4 Σύνδεση Χρηστών

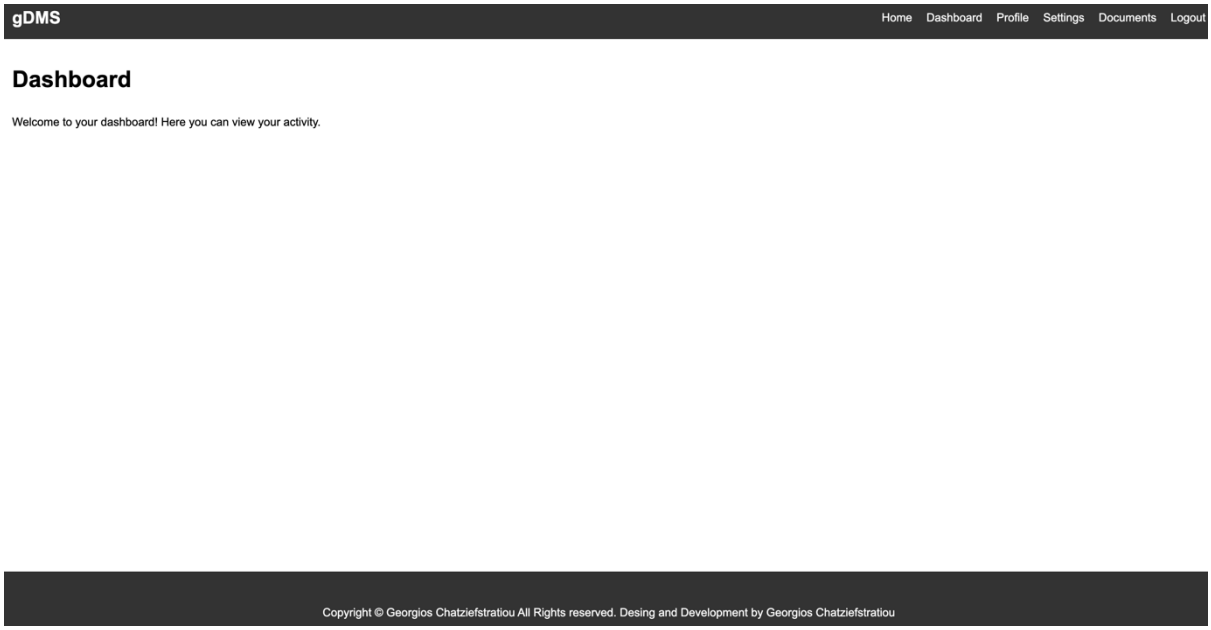
Στη σελίδα σύνδεσης, οι χρήστες μπορούν να εισάγουν τα στοιχεία τους για να αποκτήσουν πρόσβαση στον λογαριασμό τους. Αυτή η σελίδα περιλαμβάνει πεδία για το όνομα χρήστη και τον κωδικό πρόσβασης, καθώς και επιλογές για ανάκτηση του κωδικού πρόσβασης σε περίπτωση που τον έχουν ξεχάσει. Είναι σημαντικό η διαδικασία σύνδεσης να είναι απλή και γρήγορη.



Εικόνα 15 Σύνδεση Χρηστών

7.5 Σελίδα Αποσύνδεσης Χρηστών

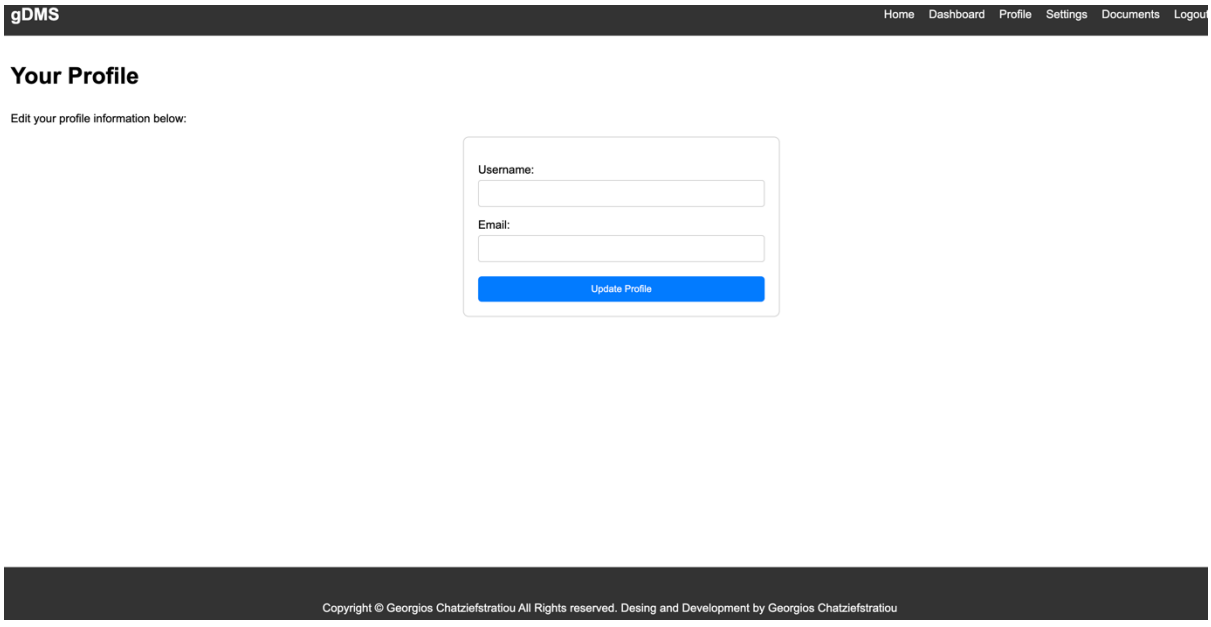
Η σελίδα αποσύνδεσης δίνει τη δυνατότητα στους χρήστες να αποχωρούν από τον λογαριασμό τους. Αυτή η διαδικασία διασφαλίζει ότι τα προσωπικά τους δεδομένα παραμένουν ασφαλή και προστατευμένα, ιδίως σε δημόσιους υπολογιστές. Η αποσύνδεση θα πρέπει να είναι εύκολα προσβάσιμη και να επιβεβαιώνεται από τον χρήστη πριν ολοκληρωθεί.



Εικόνα 16 Σελίδα Αποσύνδεσης Χρηστών

7.6 Προφίλ Χρήστη

Στη σελίδα προφίλ χρήστη, οι χρήστες έχουν τη δυνατότητα να προσαρμόσουν τα προσωπικά τους στοιχεία. Αυτό περιλαμβάνει την αλλαγή του ονόματος χρήστη, του κωδικού πρόσβασης και άλλων ρυθμίσεων. Η δυνατότητα επεξεργασίας των πληροφοριών τους επιτρέπει στους χρήστες να διατηρούν το προφίλ τους ενημερωμένο και ασφαλές.



Εικόνα 17 Προφίλ Χρήστη

7.8 Μενού Αρχείων (Documents)

Στην ενότητα Αρχείων, οι χρήστες μπορούν να διαχειριστούν τα έγγραφά τους με ευκολία. Εδώ μπορούν να ανεβάσουν νέα έγγραφα, να τα επεξεργαστούν ή να διαγράψουν όσα δεν χρειάζονται πια. Αυτή η λειτουργία προσφέρει στους χρήστες την ευχέρεια να οργανώνουν τη δουλειά τους και να διατηρούν τα αρχεία τους τακτοποιημένα.

Εικόνα 18 Μενού Αρχείων (Documents)

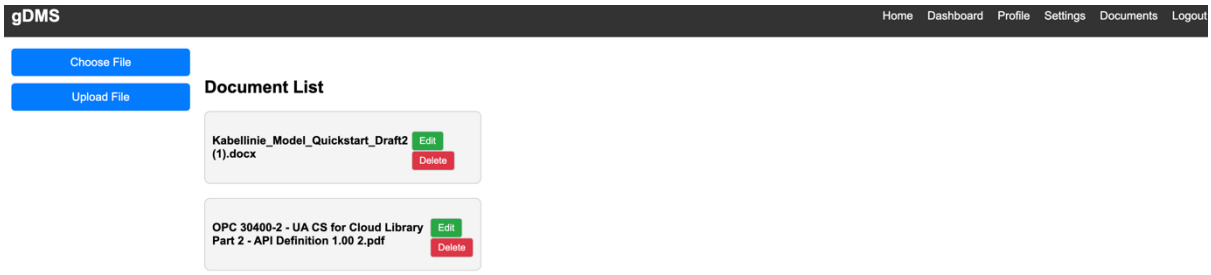
7.9 Ανέβασμα Εγγράφου

Η δυνατότητα ανέβασματος εγγράφων είναι απλή και γρήγορη. Οι χρήστες μπορούν να επιλέξουν τα αρχεία που επιθυμούν να ανεβάσουν από τη συσκευή τους και να τα αποθηκεύσουν στην εφαρμογή. Αυτή η λειτουργία υποστηρίζει διάφορους τύπους αρχείων, προσφέροντας ευελιξία στους χρήστες.

Εικόνα 19 Ανέβασμα Εγγράφου (Error case)

7.10 Επεξεργασία / Διαγραφή Εγγράφου

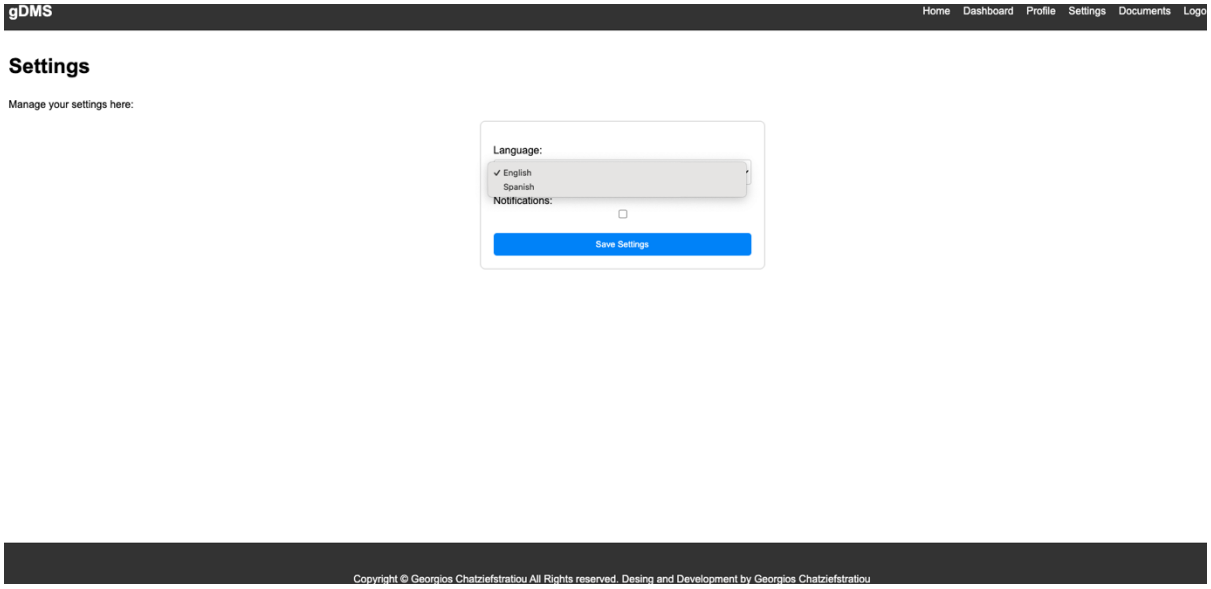
Μέσω της σελίδας επεξεργασίας εγγράφων, οι χρήστες μπορούν να ανοίξουν και να τροποποιήσουν τα έγγραφά τους. Οι εργαλειοθήκες επεξεργασίας είναι φιλικές προς τον χρήστη, επιτρέποντας τη γρήγορη επεξεργασία κειμένου, την προσθήκη εικόνων και άλλων στοιχείων.



Εικόνα 20 Επεξεργασία / Διαγραφή Εγγράφου

7.11 Ρυθμίσεις (Settings)

Στην ενότητα Ρυθμίσεων, οι χρήστες έχουν τη δυνατότητα να προσαρμόσουν την εφαρμογή σύμφωνα με τις προτιμήσεις τους. Μπορούν να αλλάξουν τη γλώσσα της εφαρμογής, να επιλέξουν το χρώμα θέματος και άλλες προσωπικές ρυθμίσεις που βελτιώνουν την εμπειρία χρήσης.



Εικόνα 21 Ρυθμίσεις (Settings)

ΚΕΦΑΛΑΙΟ 8 AWS Υλοποίηση Υποδομών στο Νέφος

8.1 Υλοποίηση Υποδομών με Terraform

Εδώ θα γινει περιγραφη για τις υποδομές που υλοποιούνται στο AWS, όπως EC2 instances, S3 buckets, RDS databases με την χρηση του Terraform

Για την υλοποίηση της εφαρμογής, αρχικά θα χρειαστούμε ένα S3 bucket, όπου κάθε χρήστης θα μπορεί να ανεβάζει το έγγραφό του. Η εφαρμογή που θα αναπτυχθεί με Spring Boot και Angular θα φιλοξενηθεί σε ένα EC2 instance, το οποίο θα δημιουργήσουμε χρησιμοποιώντας το Terraform. Παράλληλα, θα πρέπει να διασφαλίσουμε ότι το S3 bucket και το EC2 instance διαθέτουν τα απαραίτητα δικαιώματα, τα οποία θα ρυθμίσουμε μέσω Terraform, συμπεριλαμβάνοντας και κανόνες που θα καθορίζουν τι μπορεί να εκτελέσει το EC2 instance. Επίσης, δεν πρέπει να παραλείψουμε τη δημιουργία μιας βάσης δεδομένων Postgresql, η οποία θα εγκατασταθεί και αυτή μέσω Terraform στο AWS RDS.

Βλέπουμε ότι με την χρήση του terraform μπορούμε να δημιουργήσουμε τις υποδομές που χρειαζόμαστε χωρίς να χρειαζόμαστε πολλά εργαλεία πάρα μόνο γνώση terraform .
Ακολουθεί ένα δείγμα του κώδικα του terraform για όλα τα αναγκαία services που θα χρειαστούμε.

```
resource "aws_security_group" "instance_sg" {# Security Group for EC2
  name        = "instance_sg"
  description = "Allow inbound traffic for HTTP and SSH"
  vpc_id      = var.vpc_id
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}
resource "aws_instance" "app_instance" {
  ami                    = "ami-0c55b159cbfafe1f0" # EC2 Instance
  instance_type          = "t2.micro"
  iam_instance_profile   = aws_iam_instance_profile.ec2_instance_profile.name
  vpc_security_group_ids = [aws_security_group.instance_sg.id]
  tags = {
    Name = "document_management_system"
  }
}
```

Κώδικας 11 Παράδειγμα Terraform για δημιουργια EC2 στο AWS

Και τώρα θα περιγράψουμε τους πόρους, που δημιουργούνται με terraform για **aws_security_group "instance_sg"**

Αυτό το τμήμα του κώδικα δημιουργεί ένα **Security Group** στο AWS. Ένα Security Group λειτουργεί ως εικονικό τείχος προστασίας για να ελέγχει την εισερχόμενη και εξερχόμενη κίνηση δικτύου σε EC2 instances.

- **name**: Ορίζει το όνομα του Security Group, το οποίο είναι "instance_sg".
- **description**: Περιγραφή του Security Group, που εδώ είναι "Allow inbound traffic for HTTP and SSH" (επιτρέπει εισερχόμενη κίνηση για HTTP και SSH).
- **vpc_id**: Το ID του Virtual Private Cloud (VPC) στο οποίο θα ανήκει το Security Group. Αυτό το ID αναφέρεται μέσω της μεταβλητής var.vpc_id.

Τα **ingress** μπλοκ είναι κανόνες εισερχόμενης κίνησης:

- Ο πρώτος κανόνας:
  - **from_port** και **to_port**: Ορίζει ότι θα επιτρέπεται η κίνηση στο port 22 (που χρησιμοποιείται για SSH).
  - **protocol**: Το πρωτόκολλο είναι "tcp".
  - **cidr_blocks**: Επιτρέπει την πρόσβαση από οποιαδήποτε διεύθυνση IP (0.0.0.0/0).
- Ο δεύτερος κανόνας:
  - **from_port** και **to_port**: Ορίζει ότι θα επιτρέπεται η κίνηση στο port 8080 (συχνά χρησιμοποιείται από web εφαρμογές, π.χ. Spring Boot).
  - **protocol**: Το πρωτόκολλο είναι "tcp".
  - **cidr_blocks**: Επιτρέπει επίσης την πρόσβαση από οποιαδήποτε διεύθυνση IP (0.0.0.0/0).

Και τώρα θα περιγράψουμε τους πόρους, που δημιουργούνται με terraform για **aws_instance "app_instance"**

Αυτό το τμήμα του κώδικα δημιουργεί ένα **EC2 instance** στο AWS.

- **ami**: Το Amazon Machine Image (AMI) ID που θα χρησιμοποιηθεί για τη δημιουργία του EC2 instance. Εδώ το AMI είναι "ami-0c55b159cbfafe1f0", το οποίο είναι πιθανότατα ένα AMI που τρέχει Linux.
- **instance_type**: Ο τύπος του EC2 instance, που εδώ είναι "t2.micro". Αυτός ο τύπος είναι κατάλληλος για ελαφριές εργασίες, όπως μικρές web εφαρμογές.
- **iam_instance_profile**: Αναφέρεται σε ένα IAM instance profile που θα επιτρέψει στο instance να χρησιμοποιεί τα δικαιώματα που έχουν εκχωρηθεί σε αυτό το profile.
- **vpc_security_group_ids**: Συσχετίζει το EC2 instance με το Security Group που δημιουργήθηκε νωρίτερα (aws_security_group.instance_sg.id).
- **tags**: Ορίζει ένα tag με το όνομα "SpringBootAppInstance", το οποίο θα βοηθήσει στην αναγνώριση αυτού του instance.

```
resource "aws_s3_bucket" "documents_bucket" {
  bucket_prefix = var.s3_bucket_prefix
  acl           = "private"
}

resource "aws_iam_role" "ec2_role" {
  name = "ec2_s3_role"
```

```
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
          Service = "ec2.amazonaws.com"
        }
      }
    ]
  })
}
resource "aws_iam_policy" "s3_access_policy" {
  name        = "s3_access_policy"
  description = "Policy to allow EC2 instance to access S3 bucket"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "s3:PutObject",
          "s3:GetObject",
          "s3:ListBucket"
        ]
        Resource = [
          "${aws_s3_bucket.documents_bucket.arn}/*",
          "${aws_s3_bucket.documents_bucket.arn}"
        ]
      }
    ]
  })
}
```

Κώδικας 12 Παράδειγμα Terraform για δημιουργια S3, policy στο AWS

Και τώρα θα περιγράψουμε τους πόρους, που δημιουργούνται με terraform για **aws_s3_bucket "documents_bucket"**

Αυτός ο πόρος δημιουργεί ένα **S3 bucket** στο AWS. Τα S3 buckets χρησιμοποιούνται για την αποθήκευση δεδομένων στο cloud.

- **bucket_prefix**: Ορίζει το πρόθεμα του ονόματος για το S3 bucket. Το πραγματικό όνομα του bucket θα αποτελείται από αυτό το πρόθεμα συν κάποια μοναδική τιμή που θα δημιουργηθεί αυτόματα. Το πρόθεμα παρέχεται μέσω της μεταβλητής var.s3_bucket_prefix.
- **acl**: Ορίζει το επίπεδο πρόσβασης (Access Control List) του bucket. Εδώ, η πρόσβαση είναι ορισμένη ως "private", που σημαίνει ότι μόνο ο κάτοχος του bucket έχει πρόσβαση σε αυτό.

Και τώρα θα περιγράψουμε τους πόρους, που δημιουργούνται με terraform για **aws_iam_role "ec2_role"**

Αυτός ο πόρος δημιουργεί ένα **IAM Role** στο AWS. Ένα IAM Role είναι μια συλλογή δικαιωμάτων που μπορείτε να εκχωρήσετε σε υπηρεσίες AWS ώστε να εκτελούν ενέργειες για λογαριασμό σας.

- **name**: Ορίζει το όνομα του ρόλου, που εδώ είναι "ec2_s3_role".
- **assume_role_policy**: Αυτή η πολιτική ορίζει ποια υπηρεσία μπορεί να "υποδυθεί" (assume) τον ρόλο. Η πολιτική είναι γραμμένη σε JSON και εδώ επιτρέπει στο EC2 (Service = "ec2.amazonaws.com") να αναλάβει αυτόν τον ρόλο. Η πολιτική AssumeRole είναι κρίσιμη για την εξουσιοδότηση της υπηρεσίας EC2 να εκτελεί ενέργειες με τα δικαιώματα που συνδέονται με αυτόν τον ρόλο.

Και τώρα θα περιγράψουμε τους πόρους, που δημιουργούνται με terraform για **aws_iam_policy "s3_access_policy"**

Αυτός ο πόρος δημιουργεί μια **IAM Policy** που επιτρέπει την πρόσβαση σε συγκεκριμένες ενέργειες στο S3 bucket.

- **name**: Το όνομα της πολιτικής είναι "s3_access_policy".
- **description**: Παρέχει μια περιγραφή της πολιτικής, η οποία εδώ δηλώνει ότι η πολιτική επιτρέπει σε ένα EC2 instance να έχει πρόσβαση στο S3 bucket.
- **policy**: Η ίδια η πολιτική είναι γραμμένη σε JSON και περιγράφει τα δικαιώματα που παρέχονται:
    - **Effect**: "Allow" σημαίνει ότι επιτρέπεται η εκτέλεση των καθορισμένων ενεργειών.
    - **Action**: Οι ενέργειες που επιτρέπονται, οι οποίες εδώ περιλαμβάνουν:
        - s3:PutObject: Επιτρέπει την αποστολή αντικειμένων στο bucket.
        - s3:GetObject: Επιτρέπει τη λήψη αντικειμένων από το bucket.
        - s3:ListBucket: Επιτρέπει την καταγραφή των αντικειμένων που βρίσκονται στο bucket.
    - **Resource**: Οι πόροι στους οποίους επιτρέπεται η πρόσβαση. Χρησιμοποιείται το ARN (Amazon Resource Name) του S3 bucket που δημιουργήθηκε προηγουμένως:
        - ${aws_s3_bucket.documents_bucket.arn}/*: Επιτρέπει την πρόσβαση σε όλα τα αντικείμενα εντός του bucket.
        - ${aws_s3_bucket.documents_bucket.arn}: Επιτρέπει την πρόσβαση στο ίδιο το bucket (π.χ., για ενέργειες όπως η καταγραφή).

Αποτέλεσμα με Terraform:

Δημιουργουμε μια υποδομή που περιλαμβάνει:

- Έναν EC2 instance, που θα τρέχει μια εφαρμογή (πιθανώς μια εφαρμογή Spring Boot, όπως υποδηλώνει η ετικέτα).
- Έναν RDS PostgreSQL instance για την αποθήκευση δεδομένων.
- Έναν S3 bucket για την αποθήκευση αρχείων.
- Έναν IAM ρόλο και πολιτική για την ασφαλή πρόσβαση του EC2 instance στον S3 bucket.
- Τα security groups εξασφαλίζουν ότι μόνο οι επιθυμητές θύρες είναι ανοιχτές για πρόσβαση από το διαδίκτυο.

Ο συνδυασμός αυτών των πόρων παρέχει μια ολοκληρωμένη υποδομή για την φιλοξενία μιας εφαρμογής στον AWS με ασφαλή αποθήκευση δεδομένων και πρόσβαση στα απαραίτητα αρχεία.

ΚΕΦΑΛΑΙΟ 9 ΥΛΟΠΟΙΗΣΗ BACKEND

9.1 Υλοποίηση Spring Boot

Δομή Κώδικα και Αρχιτεκτονική:

Η αρχιτεκτονική είναι οργανωμένη γύρω από διάφορά Modules που χειρίζονται διαφορετικές πτυχές της εφαρμογής . Τα κύρια Modules είναι

Controllers: Υπεύθυνο για την επεξεργασία των εισερχόμενων HTTP αιτημάτων και την επιστροφή των κατάλληλων απαντήσεων.

Services: Πραγματοποιεί τη λογική της εφαρμογής και χειρίζονται τις επικοινωνίες μεταξύ των Controllers και των Repositories.

Repositories: Υπεύθυνο για την αλληλεπίδραση με τη βάση δεδομένων, διαχειριζόμενα τις CRUD (Create, Read, Update, Delete) λειτουργίες.

Οι κλάσεις μας οργανώνονται σε υποσυστήματα που εξυπηρετούν συγκεκριμένες λειτουργίες της εφαρμογής μας.

Όνομα υποψήφιας κλάσης	**Σκεπτικό**
HashUtil	Η κλάση αυτή δημιουργεί μοναδικά hashes, εξασφαλίζοντας ότι τα ευαίσθητα δεδομένα αποθηκεύονται με ασφάλεια
LocalStackConfig	Αυτή η κλάση διαχειρίζεται τη διαμόρφωση του LocalStack, διευκολύνοντας την ανάπτυξη και τη δοκιμή υπηρεσιών AWS τοπικά.
MediaTypeUtil	Η κλάση αυτή βοηθά στην αναγνώριση του τύπου αρχείου (π.χ. PDF), εξασφαλίζοντας ότι τα έγγραφα διαχειρίζονται σωστά ανάλογα με το MIME type τους.
S3Config	Διαχειρίζεται τη διαμόρφωση για την αποθήκευση αρχείων στο Amazon S3, περιλαμβάνοντας τις ρυθμίσεις για τη σύνδεση και την ασφάλεια.
DocumentController	Εδώ βρίσκονται οι λειτουργίες που αφορούν τα έγγραφα, όπως η δημιουργία, η ενημέρωση και η διαγραφή εγγράφων από την εφαρμογή.
GlobalExceptionHandler	Αυτή η κλάση χειρίζεται τα exceptions που προκύπτουν κατά την εκτέλεση της εφαρμογής, παρέχοντας ενιαία διαχείριση σφαλμάτων.
UserController	Εδώ διαχειριζόμαστε τις λειτουργίες που σχετίζονται με τον χρήστη, όπως η εγγραφή, η σύνδεση και η διαχείριση των στοιχείων του.
DocumentService	Η κλάση αυτή περιέχει τη λογική της εφαρμογής για τη διαχείριση εγγράφων, συμπεριλαμβανομένων των επιχειρησιακών κανόνων.

DocumentMetadata	Αυτό το μοντέλο περιγράφει τα μεταδεδομένα που σχετίζονται με τα έγγραφα που αποθηκεύουμε, όπως η ημερομηνία δημιουργίας και ο τύπος αρχείου
Document	Αυτή η κλάση αποτελεί το μοντέλο του αρχείου, περιγράφοντας τα χαρακτηριστικά κάθε αρχείου που αποθηκεύεται στην εφαρμογή.
UploadResponse	Η κλάση αυτή διαχειρίζεται την απάντηση που λαμβάνουμε κατά την ανέβασμα αρχείου, παρέχοντας πληροφορίες για την επιτυχία ή αποτυχία της διαδικασίας.
User	Το μοντέλο του χρήστη περιλαμβάνει τα στοιχεία που αφορούν τους εγγεγραμμένους χρήστες της εφαρμογής.
DocumentMetadataRepository	Αυτή η κλάση είναι υπεύθυνη για την αποθήκευση και ανάκτηση μεταδεδομένων εγγράφων από τη βάση δεδομένων.
DocumentRepository	Διαχειρίζεται την αποθήκευση και ανάκτηση εγγράφων στην βάση δεδομένων.

application.properties	Αυτή η διαμόρφωση περιέχει όλες τις ρυθμίσεις της εφαρμογής, όπως οι ρυθμίσεις σύνδεσης στη βάση δεδομένων και άλλες ρυθμίσεις παραμέτρων.
UserService	Η κλάση αυτή περιέχει τη λογική της εφαρμογής για τη διαχείριση χρήστη, συμπεριλαμβανομένων των επιχειρησιακών κανόνων.
UserResponse	Η κλάση αυτή διαχειρίζεται την απάντηση που λαμβάνουμε κατά την δημιουργία χρήστη, παρέχοντας πληροφορίες για την επιτυχία ή αποτυχία της διαδικασίας.
UserRepository	Διαχειρίζεται την αποθήκευση και ανάκτηση στοιχείων των χρηστών στην βάση δεδομένων.
application-local.properties	Αυτή η διαμόρφωση περιέχει όλες τις ρυθμίσεις της εφαρμογής για τοπική χρήση, όπως οι ρυθμίσεις σύνδεσης στη βάση δεδομένων και άλλες ρυθμίσεις παραμέτρων.

Πίνακας 1 Περιγραφή των βασικών κλάσεων

9.2 Τι είναι το RESTful API

Τα RESTful APIs βασίζονται στο πρωτόκολλο HTTP και ακολουθούν μια σειρά από κανόνες που επιτρέπουν την επικοινωνία μεταξύ του πελάτη (client) και του διακομιστή (server). Τα βασικά στοιχεία μιας RESTful αρχιτεκτονικής περιλαμβάνουν:

Χρήση HTTP μεθόδων (GET, POST, PUT, DELETE) για την αλληλεπίδραση με τους πόρους.

Κατάλληλα HTTP status codes για να αναφέρονται τα αποτελέσματα των αιτημάτων.

Συγκεκριμένες και σαφείς διαδρομές (URLs) για την πρόσβαση σε πόρους (resources).

Χρήση JSON ως μορφή ανταλλαγής δεδομένων.

Παράδειγμα Αιτήματος και Απάντησης

Η εφαρμογή μας υλοποιεί ένα API για τη διαχείριση χρηστών (users) και αρχειων (Documents). Θα δείξουμε παραδείγματα για το πώς μπορεί να γίνει ένα αίτημα και η αντίστοιχη απάντηση.

Παράδειγμα Αιτήματος και Απάντησης

1. Αίτημα GET για Λίστα Χρηστών

```
GET /api/users HTTP/1.1
Host: your-api.com
Accept: application/json
```

```
HTTP/1.1 200 OK
Content-Type: application/json

[
  {
    "id": 1,
    "name": "giorgo ch",
    "email": "giorgo.ch@example.com"
  },
  {
    "id": 2,
    "name": "John Wick",
    "email": "John.wick@example.com"
  }
]
```

Κώδικας 13 Παράδειγμά με POST

```
POST /api/users HTTP/1.1
Host: your-api.com
Content-Type: application/json

{
  "name": "giorgo ch",
  "email": "giorgo.ch@example.com"
}
```

Κώδικας 14 Παράδειγμά Απάντησή POST

```
HTTP/1.1 201 Created
Content-Type: application/json

{
  "id": 3,
  "name": "giorgo ch",
  "email": "giorgo.ch@example.com"
}
```

Κώδικας 15 Παράδειγμά Αίτημα PUT για Ενημέρωση Χρήστη

```
PUT /api/users/3 HTTP/1.1
Host: your-api.com
Content-Type: application/json

{
  "name": "giorgo ch",
  "email": "giorgo.ch@example.com"
}
```

Κώδικας 16 Παράδειγμά απαντηση σε PUT

```
HTTP/1.1 200 OK
Content-Type: application/json

{
```

```
  "id": 3,
  "name": "giorgo ch",
  "email": "giorgo.ch@example.com"
}
```

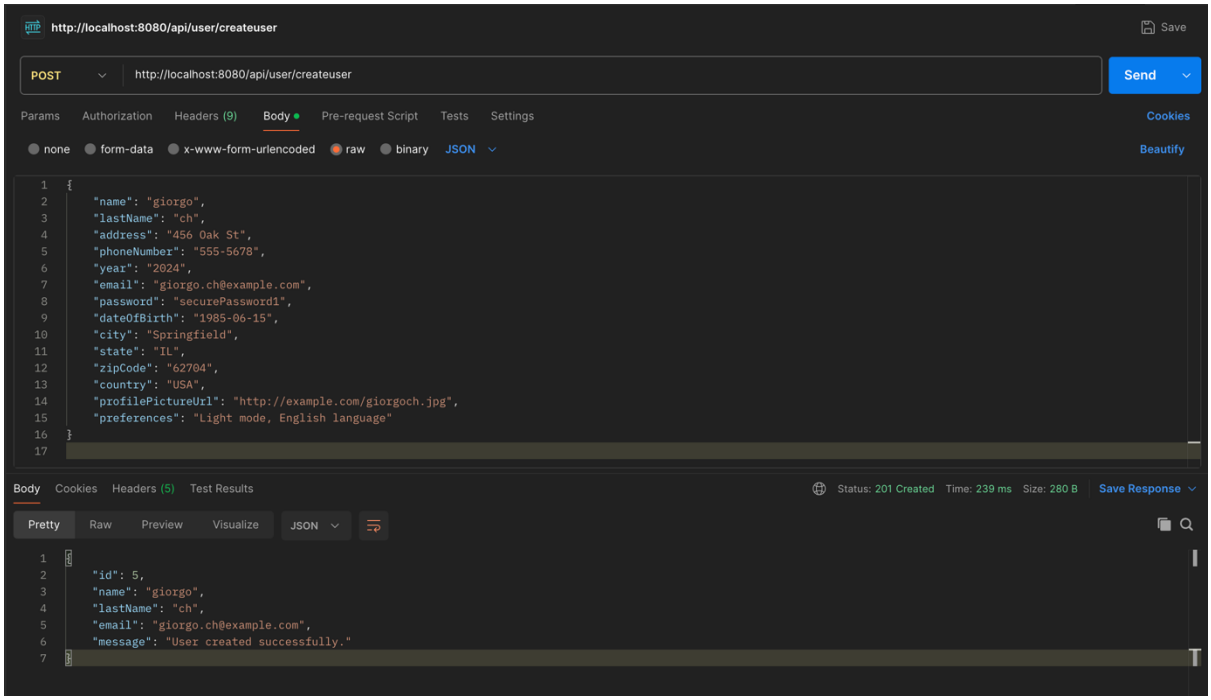
Κώδικας 17 Παράδειγμά Αίτημα DELETE για Διαγραφή Χρήστη

```
DELETE /api/users/3 HTTP/1.1
Host: your-api.com
```

Κώδικας 18 Παράδειγμά απαντηση Delete

```
HTTP/1.1 204 No Content
```

Το αίτημα DELETE αφαιρεί τον χρήστη με το id 3 από το σύστημα. Η απάντηση είναι κενή με κωδικό κατάστασης 204, που σημαίνει επιτυχή διαγραφή.



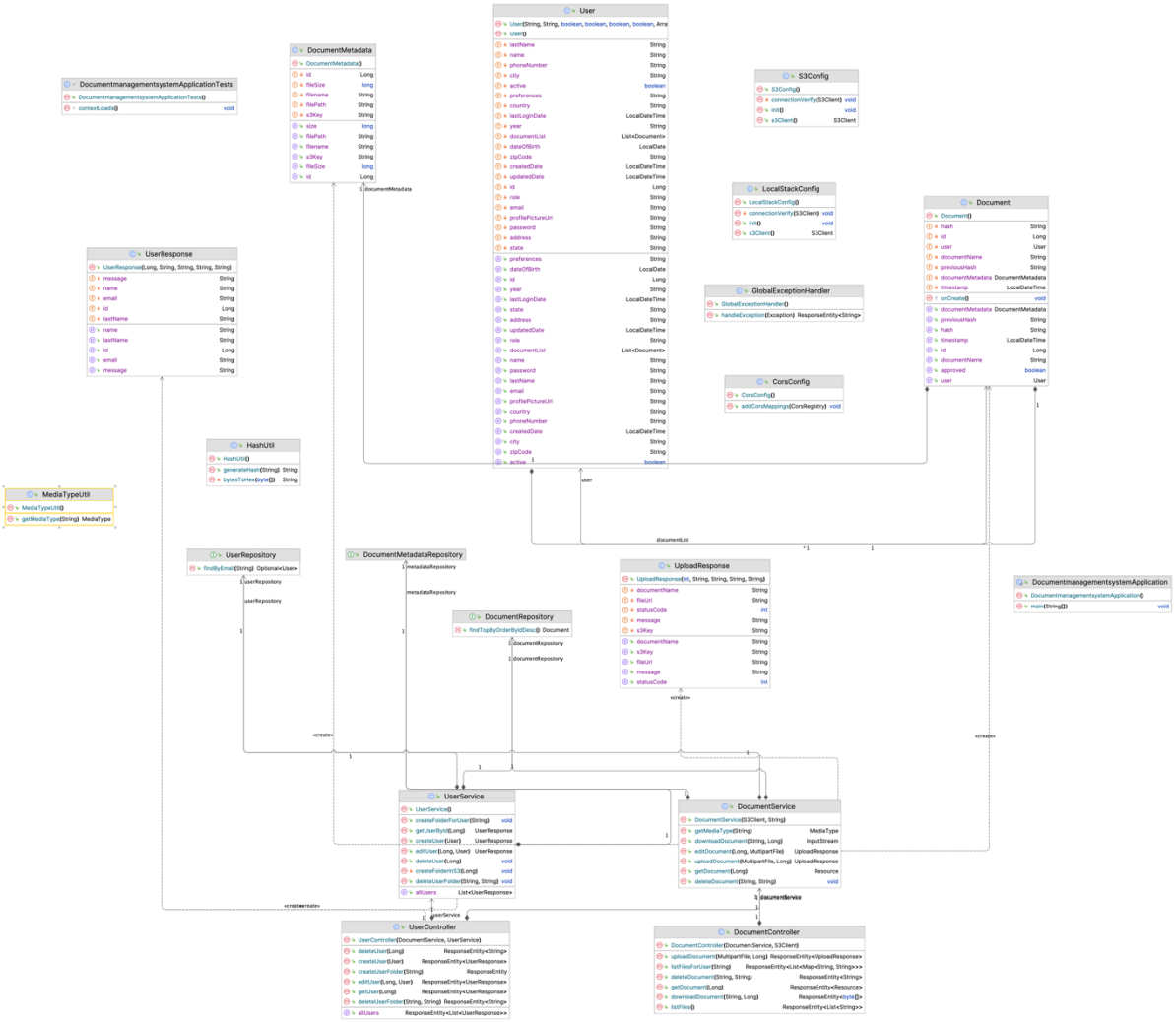
Εικόνα 22 χρήση Postman για αιτημα POST json

9.3 HTTP Status Codes

- **200 OK:** Το αίτημα ήταν επιτυχές και τα δεδομένα επιστράφηκαν.
- **201 Created:** Ένας πόρος δημιουργήθηκε επιτυχώς.
- **204 No Content:** Η ενέργεια ολοκληρώθηκε επιτυχώς, αλλά δεν υπάρχει περιεχόμενο για επιστροφή.
- **400 Bad Request:** Το αίτημα είχε λάθος σύνταξη ή δεδομένα.
- **404 Not Found:** Ο ζητούμενος πόρος δεν βρέθηκε.

9.4 Βέλτιστες Πρακτικές Best Practices

- **Ονοματολογία Διαδρομών (URLs):** Χρησιμοποιούμε ονόματα πόρων στον πληθυντικό (π.χ. /api/users, /api/documents).
- **Χρήση JSON:** Στελνουμε δεδομένα σε μορφή JSON για καλύτερη συμβατότητα.
- **Κωδικοί Κατάστασης:** Χρησιμοποιήσουμε σωστούς κωδικούς κατάστασης για να διευκολύνουμε την κατανόηση των αποτελεσμάτων από τον πελάτη.



Εικόνα 23 UMLclass Diagram

9.5 Χρήση Docker και LocalStack

Για τη διαχείριση και ανάπτυξη της εφαρμογής, χρησιμοποιούμε το **Docker**, το οποίο επιτρέπει την απομόνωση των περιβαλλόντων και την ευκολία στην ανάπτυξη τόσο σε τοπικό επίπεδο όσο και στο cloud. Με τη χρήση Docker, έχουμε δημιουργήσει σταθερά και επαναλήψιμα περιβάλλοντα που διασφαλίζουν ότι η εφαρμογή θα λειτουργεί με τον ίδιο τρόπο ανεξάρτητα από την υποδομή στην οποία αναπτύσσεται. Αυτό μας επιτρέπει να αυτοματοποιούμε τη διαδικασία ανάπτυξης, μειώνοντας την πιθανότητα σφαλμάτων λόγω διαφορές περιβάλλοντος.

Επιπλέον, για τη δοκιμή υπηρεσιών AWS σε τοπικό επίπεδο, χρησιμοποιούμε το **LocalStack**, μια πλατφόρμα που προσομοιώνει τοπικά τις υπηρεσίες του AWS. Μέσω του LocalStack, μπορούμε να αναπτύσσουμε και να δοκιμάζουμε λειτουργίες που σχετίζονται με την υποδομή, όπως S3, Lambda και RDS, χωρίς να απαιτείται σύνδεση σε πραγματικούς πόρους του AWS. Αυτή η πρακτική μειώνει το κόστος και βελτιώνει την ταχύτητα ανάπτυξης, παρέχοντας ένα πλήρως λειτουργικό περιβάλλον για την ενσωμάτωση και τον έλεγχο νέων χαρακτηριστικών πριν από την τελική παραγωγή.

ΚΕΦΑΛΑΙΟ 10 ΠΡΟΤΑΣΕΙΣ ΓΙΑ ΔΕΔΟΜΕΝΑ

10.1 Στρατηγικές Διαχείρισης Δεδομένων

Η διαδικασία συλλογής και αποθήκευσης μεταδεδομένων στην εφαρμογή σχεδιάστηκε με βάση την ανάγκη για υψηλή ακρίβεια και αποτελεσματική διαχείριση δεδομένων. Τα μεταδεδομένα περιλαμβάνουν πληροφορίες σχετικά με τις λειτουργίες, τις συναλλαγές και τους χρήστες της εφαρμογής και καταγράφονται αυτόματα κατά τη διάρκεια της χρήσης της.

Η αποθήκευση των μεταδεδομένων γίνεται στη βάση δεδομένων **PostgreSQL**, η οποία χρησιμοποιείται για την ασφαλή και αποδοτική διαχείριση των δεδομένων. Τα μεταδεδομένα είναι οργανωμένα σε πίνακες και συσχετίζονται με άλλες βασικές πληροφορίες της εφαρμογής, επιτρέποντας την ταχεία ανάκτηση και επεξεργασία τους. Η επιλογή της PostgreSQL προσφέρει σημαντικά πλεονεκτήματα, όπως η υποστήριξη επεκτασιμότητας και η ανθεκτικότητα σε σφάλματα, διασφαλίζοντας την αποθήκευση των δεδομένων σε πραγματικό χρόνο.

10.2 Στρατηγικές Backup και Ανάκτησης

Για την ασφάλεια των δεδομένων έχουμε μια προσέγγιση για την ακεραιότητα και την διαθεσιμότητα των πληροφοριών της εφαρμογής. Καθοριστικό στοιχείο αυτής της προσέγγισης είναι η τακτική δημιουργία αντιγράφων ασφαλείας (backup) της βάσης δεδομένων, με στόχο την προστασία από πιθανές απώλειες δεδομένων. Τα δεδομένα αποθηκεύονται σε ασφαλή αποθετήρια(Repository), ενώ τα αυτόματα backups πραγματοποιούνται σε πολλαπλές ζώνες διαθεσιμότητας, επιτρέποντας την άμεση ανάκτηση (recovery) σε περίπτωση βλάβης ή άλλου απρόοπτου συμβάντος.

Επιπλέον, οι διαδικασίες ανάκτησης δεδομένων περιλαμβάνουν σε τακτά χρονικά διαστηματά δοκιμές, ώστε να διασφαλιστεί ότι η ανάκτηση μπορεί να γίνει αποτελεσματικά και χωρίς καθυστερήσεις. Αυτές οι πρακτικές εξασφαλίζουν ότι τα δεδομένα είναι πάντα προστατευμένα και ότι η εφαρμογή μπορεί να επανέλθει σε πλήρη λειτουργικότητα σε ελάχιστο χρόνο σε περίπτωση αποτυχίας.

10.3 Διαχείριση Μεταδεδομένων

Η ανάλυση των μεταδεδομένων πραγματοποιείται με σκοπό την εξαγωγή χρήσιμων πληροφοριών για τη λειτουργία και την απόδοση της εφαρμογής. Η βάση δεδομένων PostgreSQL παρέχει ισχυρά εργαλεία για την επεξεργασία και ανάλυση των μεταδεδομένων, επιτρέποντας την εκτέλεση σύνθετων ερωτημάτων και την εξαγωγή αναφορών που βοηθούν στη λήψη αποφάσεων.

Για τη δημιουργία αναφορών, χρησιμοποιούνται διαδικασίες ανάλυσης δεδομένων που επιτρέπουν την εξαγωγή σημαντικών στατιστικών και δεικτών από τα μεταδεδομένα. Αυτές οι αναφορές παρουσιάζονται στους υπεύθυνους λήψης αποφάσεων της επιχείρησης και περιλαμβάνουν πληροφορίες σχετικά με την απόδοση της εφαρμογής, τη συμπεριφορά των χρηστών και τις συνολικές τάσεις χρήσης. Μέσω της ανάλυσης αυτής, η εφαρμογή προσαρμόζεται συνεχώς για να βελτιώνει τις λειτουργίες της και να ανταποκρίνεται στις ανάγκες των χρηστών.

10.4 Ανάλυση και Αναφορά

Για την ανάλυση των μεταδεδομένων και τη δημιουργία αναφορών, χρησιμοποιούμε μια συνδυαστική προσέγγιση που βασίζεται σε σύγχρονα εργαλεία επεξεργασίας και ανάλυσης δεδομένων. Αρχικά, τα μεταδεδομένα συλλέγονται αυτόματα από τις διάφορες λειτουργίες της εφαρμογής και αποθηκεύονται στη βάση δεδομένων PostgreSQL, επιτρέποντας την ασφαλή και οργανωμένη αποθήκευσή τους.

Η ανάλυση πραγματοποιείται μέσω εξειδικευμένων ερωτημάτων που εκτελούνται στη βάση δεδομένων, επιτρέποντας την εξαγωγή πληροφοριών σχετικά με τις τάσεις χρήσης, την απόδοση του συστήματος και τη συμπεριφορά των χρηστών. Τα δεδομένα αυτά επεξεργάζονται για να παραχθούν αναφορές, οι οποίες βοηθούν στη λήψη αποφάσεων και στη συνεχή βελτίωση της εφαρμογής. Οι αναφορές αυτές παρέχονται σε μορφή πίνακα ή γραφήματος, διευκολύνοντας την παρουσίαση των αποτελεσμάτων με τρόπο που είναι εύκολα κατανοητός και χρήσιμος για την παρακολούθηση των βασικών δεικτών απόδοσης Key Performance Indicators (KPIs).

Κεφάλαιο 11 ΣΥΜΠΕΡΑΣΜΑΤΑ ΚΑΙ ΠΡΟΟΠΤΙΚΕΣ

11.1 Σύνοψη της διπλωματικής εργασίας

Η παρούσα διπλωματική εργασία επικεντρώθηκε στην ανάπτυξη ενός κατανεμημένου συστήματος διαχείρισης αρχείων με χρήση σύγχρονων τεχνολογιών και βέλτιστων πρακτικών για την ανάπτυξη web εφαρμογών. Το **frontend** της εφαρμογής υλοποιήθηκε με το **Angular framework**, προσφέροντας μια φιλική προς τον χρήστη διεπαφή, ενώ το **backend** βασίστηκε στο **Spring Boot** για τη διαχείριση των λειτουργιών και τη διασύνδεση με την υποδομή στο cloud.

Η εφαρμογή επιτρέπει τη δημιουργία και διαχείριση λογαριασμών χρηστών, την αποθήκευση, την διαγραφή και την προβολή εγγράφων μέσω του **AWS Cloud**, το οποίο εξασφαλίζει υψηλή διαθεσιμότητα και ασφάλεια για τα αποθηκευμένα αρχεία. Η **PostgreSQL** επιλέχθηκε ως βάση δεδομένων για τη διαχείριση των πληροφοριών των χρηστών και των εγγράφων, προσφέροντας αξιοπιστία και κλιμάκωση.

Το σύστημα περιλαμβάνει λειτουργίες όπως:

- Εγγραφή νέων χρηστών και διαχείριση προφίλ.
- Αποθήκευση, προβολή και διαγραφή αρχείων από το cloud.

Σελίδες πλοήγησης, όπως η "Αρχική", "Register", "Documents", "Profile", "Settings" και "Logout", που επιτρέπουν εύκολη πρόσβαση σε όλες τις βασικές λειτουργίες.
Η εφαρμογή έχει κατασκευαστεί με τρόπο που προσφέρει ευελιξία και επεκτασιμότητα, ενώ υποστηρίζεται από μια σύγχρονη υποδομή cloud, καθιστώντας την έτοιμη για μελλοντικές αναβαθμίσεις. Σκοπός της εργασίας ήταν να δημιουργηθεί μια πλήρης και λειτουργική πλατφόρμα διαχείρισης αρχείων που θα καλύπτει τις βασικές ανάγκες των χρηστών και να παρέχει τα θεμέλια για περαιτέρω επεκτάσεις.

11.2 Προοπτικές

Η εφαρμογή που αναπτύχθηκε παρέχει ένα ισχυρό θεμέλιο για μελλοντική επέκταση και βελτίωση. Οι δυνατότητες που έχουν ήδη υλοποιηθεί, όπως η δημιουργία χρηστών και η διαχείριση αρχείων στο AWS Cloud, μπορούν να επεκταθούν με επιπλέον λειτουργίες, ώστε να καλύπτουν πιο εξειδικευμένες και προηγμένες ανάγκες.
Μερικές από τις μελλοντικές προοπτικές για την εφαρμογή περιλαμβάνουν:
Επεξεργασία μεταδεδομένων αρχείων:
Οι χρήστες θα μπορούν να επεξεργάζονται και να προσθέτουν πληροφορίες, όπως τίτλους, περιγραφές και ετικέτες (tags) στα αρχεία τους, διευκολύνοντας την οργάνωση και την αναζήτηση των εγγράφων στο cloud.

Βασική επεξεργασία αρχείων:

Η εφαρμογή θα μπορούσε να επεκταθεί ώστε να επιτρέπει βασικές λειτουργίες επεξεργασίας αρχείων, όπως η αλλαγή μορφής (π.χ., από PDF σε DOCX), η συμπίεση ή η προεπισκόπηση των αρχείων χωρίς την ανάγκη κατεβάσματος.

Αποστολή αρχείων μέσω email:

Οι χρήστες θα μπορούν να στέλνουν απευθείας αρχεία σε άλλους χρήστες μέσω email μέσα από την πλατφόρμα, προσφέροντας έναν εύκολο και άμεσο τρόπο διαμοιρασμού αρχείων.

Ενσωμάτωση πιο προηγμένων εργαλείων ασφάλειας:

Θα μπορούσαν να προστεθούν λειτουργίες όπως κρυπτογράφηση αρχείων ή πρόσβαση με δικαιώματα, ώστε οι χρήστες να μπορούν να ελέγχουν ποιος έχει δικαίωμα προβολής ή επεξεργασίας κάθε αρχείου.

Διαχείριση αρχείων σε ομαδικά περιβάλλοντα:

Η υποστήριξη συνεργασίας μεταξύ πολλών χρηστών πάνω στα ίδια αρχεία θα μπορούσε να βελτιώσει την εμπειρία σε περιβάλλοντα που απαιτούν ομαδική εργασία και συντονισμό.

Δυνατότητες παρακολούθησης και αναφοράς:

Μελλοντικά, η εφαρμογή θα μπορούσε να προσφέρει αναφορές χρήσης, όπως το ποιες ενέργειες εκτελούνται συχνότερα, ποιες κατηγορίες αρχείων ανεβαίνουν και διαγράφονται, ή ποιοι χρήστες είναι πιο ενεργοί.

Τέλος, η ενσωμάτωση API τρίτων θα μπορούσε να επιτρέψει μεγαλύτερη διαλειτουργικότητα με άλλες εφαρμογές ή υπηρεσίες, όπως εργαλεία διαχείρισης έργων, υπηρεσίες cloud από άλλους παρόχους ή εργαλεία ανάλυσης δεδομένων. Οι μελλοντικές επεκτάσεις είναι ρεαλιστικές λόγω της υποδομής του AWS, που προσφέρει κλιμακούμενη υπολογιστική ισχύ και αποθήκευση, ενώ οι δυνατότητες του Spring Boot και του PostgreSQL εξασφαλίζουν την ανθεκτικότητα και την απόδοση της εφαρμογής, ακόμη και σε μεγαλύτερη κλίμακα χρήσης.

Παράρτημα 1 Οδηγός χρήσης

Οδηγός εργαλείων

Frontend

Για την δημιουργία του frontend χρησιμοποιούμε το visual studio code που βρίσκουμε εδώ https://code.visualstudio.com/.

Backend

Για την δημιουργία του backend χρησιμοποιούμε το IntelliJ που βρίσκουμε εδώ https://www.jetbrains.com/idea/download/

Docker

Το docker το βρίσκουμε εδώ https://www.docker.com/get-started/

Node.js

Θα χρειαστούμε το node και το βρίσκουμε εδώ https://nodejs.org/en

Npm

Θα χρειαστούμε το npm και το βρίσκουμε εδώ https://www.npmjs.com/

Angular

Θα χρειαστούμε το angular και το βρίσκουμε εδώ https://angular.dev/tools/cli/setup-local

Terraform

Θα χρειαστούμε το terraform και το βρίσκουμε εδώ
https://developer.hashicorp.com/terraform/tutorials/aws-get-started/install-cli

Οδηγός χρήσης

Εφόσον έχουμε το docker χρειαζόμαστε να τρέξουμε το docker-compose.yml και να κάνουμε docker pull postgres:latest

Στην PostgreSQL θα χρειαστεί να κάνουμε ένα χρήστη και μια βάση δεδομένων.

Εφόσον έχουν γίνει όλα αυτά τρέχουμε τις εφαρμογές και βάζουμε στο browser localhost:4200 για να δούμε το gui.

Για να στήσουμε το infrastructure θα χρειαστεί να κάνουμε ένα AWS λογαριασμό και μετα θα πρέπει να κάνουμε χρήση του terraform με τις εντολές:

Commands στο cmd ή terminal

terraform init θα γίνει initialization του project με terraform

terraform plan θα μας δείξει τι πρόκειται να υλοποιηθεί στο cloud

terraform deploy θα δημιουργήσει τις υποδομές στο cloud

terraform destroy θα καταργήσει τις υποδομές στο cloud.

Παράρτημα 2 Κώδικας

Στο παράρτημα αυτό παρατίθεται ο κώδικας της εφαρμογής.

Ο κώδικας θα είναι διαθέσιμος και στους ακολουθους συνδεσμους GitLab και Github.
https://github.com/giorgoch/thesis_frontend.git
https://github.com/giorgoch/thesis_terraform-.git
https://gitlab.com/giorgoch/backend.git

Κώδικας για Frontend

```
.


body {
    display: flex;
    flex-direction: column;
    min-height: 100vh;
    margin: 0;
    font-family: Arial, sans-serif;
  }

  header {
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 1em;
    background-color: #333;
    color: white;
  }

  .logo {
    margin-right: 10px;
  }

  .nav ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
  }

  .nav ul li {
    display: inline-block;
    margin-left: 20px;
  }

  .nav ul li:first-child {
    margin-left: 0;
  }

  .nav ul li a {
    text-decoration: none;
    color: white;
```

```
}

.nav ul li a:hover {
  color: #ccc;
}

footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 1em;
  margin-top: auto;
}

main {
  flex: 1;
  padding: 2em;
  background-color: #f8f9fa;
}

h1 {
  font-size: 2rem;
  margin-bottom: 1em;
}

/* Form styling */
.form-container {
  display: flex;
  flex-direction: column;
  max-width: 400px;
  margin: auto;
  padding: 20px;
  background-color: white;
  border: 1px solid #ccc;
  border-radius: 8px;
}

.form-container label {
  margin-top: 1em;
}

.form-container input, .form-container select {
  padding: 10px;
  margin-top: 5px;
  border-radius: 4px;
  border: 1px solid #ccc;
}

.form-container button {
  margin-top: 20px;
  padding: 10px;
  background-color: #007bff;
  color: white;
  border: none;
```

```
    border-radius: 5px;
    cursor: pointer;
  }

  .form-container button:hover {
    background-color: #0056b3;
  }


  .action-btn {
    padding: 10px 20px;
    background-color: #28a745;
    color: white;
    text-decoration: none;
    border-radius: 5px;
    transition: background-color 0.3s ease;
  }

  .action-btn:hover {
    background-color: #218838;
  }

  .documents-grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    gap: 20px;
    margin-top: 20px;
  }

  .document-item {
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 8px;
    background-color: #f4f4f4;
  }

  .document-actions {
    margin-top: 10px;
  }

  .document-actions button {
    margin-right: 5px;
  }
```

Κώδικας 19 Το App.component.css

```
import { Component } from '@angular/core';

@Component({
  selector: 'app-dashboard',
  standalone: true,
  imports: [],
  templateUrl: './dashboard.component.html',
  styleUrl: './dashboard.component.css'
```

```
})
export class DashboardComponent {

}
```

Κώδικας 20 Το Dashboardcomponent.ts

```
.container {
  display: grid;
  grid-template-columns: 250px 1fr 600px 1fr;
  height: 100vh;
  width: auto;
}

.left-documents-container {
  padding: 20px;
}

.main-documents-container {
  padding: 20px;
}

.btn {
  padding: 10px 20px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
  text-align: center;
  display: inline-block;
  transition: background-color 0.3s ease;
  font-size: 16px;
}

button {
  padding: 5px 10px;
  border: 1px solid #ccc;
  border-radius: 4px;
  background-color: #fff;
  cursor: pointer;
}

button:hover {
  background-color: #eee;
}

.btn.primary {
  background-color: #007bff;
}

.btn.primary:hover {
  background-color: #0056b3;
```

```
}


.document-actions button.edit {
  background-color: #28a745;
  color: white;
}

.document-actions button.edit:hover {
  background-color: #218838;
}


.document-actions button.delete {
  background-color: #dc3545;
  color: white;
}

.document-actions button.delete:hover {
  background-color: #c82333;
}

.file-upload-container {
  display: flex;
  flex-direction: column;
  gap: 10px;
  margin-bottom: 20px;
}

.custom-file-input {
  display: none;
}

.file-label {
  padding: 10px 20px;
  background-color: #007bff;
  color: white;
  border-radius: 5px;
  cursor: pointer;
  text-align: center;
  display: inline-block;
  transition: background-color 0.3s ease;
}

.file-label:hover {
  background-color: #0056b3;
}

.documents-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  gap: 20px;
  margin-top: 20px;
}
```

```css
.document-item {
  display: flex;
  justify-content: space-between;
  align-items: center;
  padding: 10px;
  border-radius: 8px;
  background-color: #f4f4f4;
  border: 1px solid #ccc;
}

.document-name {
  font-size: 16px;
}

.document-buttons {
  display: flex;
  gap: 10px;
}

.document-actions {
  margin-top: 10px;
}
```

Κώδικας 21 Το Component documents.css

```ts
import { Component } from '@angular/core';
import { DocumentService } from './documentservice';
import { ChangeDetectorRef } from '@angular/core';
import { CommonModule } from '@angular/common';
@Component({
  selector: 'app-documents',
  standalone: true,
  imports: [CommonModule],
  templateUrl: './documents.component.html',
  styleUrl: './documents.component.css'
})
export class DocumentsComponent {


  selectedFile: File | null = null;
   files: { filePath: string, fileName: string }[] = [];
  constructor(private documentService: DocumentService,private cd:
ChangeDetectorRef) { }


  deleteFile(documentname : string): void {
    console.log('Filename is:', documentname);
    this.documentService.deleteDocument(documentname).subscribe({
      next: () => {
        console.log('Document deleted successfully');
      },
      error: (error) => {
        console.error('Error deleting document', error);
      },
```

```
    });
  }

  editFile(documentid: string) {

  }

  ngOnInit(): void {
    this.listFile();
  }

  errorMessage: string = '';

  listFile():void{
    this.documentService.getFiles('1').subscribe({
      next: (data) => {
        console.log('File data from API:', data);
        this.files = data;
      },
      error: (err) => this.errorMessage = err
    });

  }

  onFileSelected(event: any) {
    const file: File = event.target.files[0];
    if (file) {
      this.selectedFile = file;
      console.log('Selected file:', this.selectedFile);
    }
  }

  private userId = 1;

  uploadFile() {
    if (this.selectedFile) {
      this.documentService.uploadFile(this.selectedFile, this.userId).subscribe({
        next: (response) => {
          console.log('File uploaded successfully!', response);
        },
        error: (error) => {
          console.error('File upload failed!', error);
        }
      });
    } else {
      console.error('No file selected for upload.');
    }
  }

}
```

Κώδικας 22 για Το Documentscomponent .ts

```
import { HttpClient, HttpErrorResponse, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
```

```
import { catchError, map, Observable, switchMap, throwError } from 'rxjs';

@Injectable({
  providedIn: 'root'
})
export class DocumentService {


  constructor(private http: HttpClient) {}


private userId = 1;
private deleteurl = `http://localhost:8080/api/document/deletedocument`;

private baseUrl = `http://localhost:8080/api/document/upload/${this.userId}`;
private listfiles = 'http://localhost:8080/api/document/';

private apiUrl = 'http://localhost:8080/api/document/list-files';

uploadUrl: string = "/upload/";



deleteDocument(documentName: string): Observable<void> {
  console.log( "file apo service : " , documentName)
  const encodedDocumentName = encodeURIComponent(documentName);
  return
this.http.delete<void>(`${this.deleteurl}/${this.userId}/${encodedDocumentName}`).p
ipe(
    catchError((error: HttpErrorResponse) => {
      console.error('Error deleting document:', error);
      return throwError(error);
    })
  );
}


  uploadFile(file: File, userId: number): Observable<any> {
    const formData: FormData = new FormData();
    formData.append('file', file, file.name);

    return this.http.post(this.baseUrl, formData, {
      headers: new HttpHeaders({

      }),
      reportProgress: true,
      observe: 'events'
    });
  }


getFiles(userId: string): Observable<{ filePath: string, fileName: string }[]> {
  return this.http.get<{ filePath: string, fileName: string
}[]>(`${this.apiUrl}/${userId}`)
    .pipe(
```

```
      catchError(this.handleError)
    );
}

  listFiles(): Observable<string[]> {
    return this.http.get<string[]>(`${this.listfiles}/list-files/}`);
  }

  private handleError(error: HttpErrorResponse) {
    let errorMessage = 'An unknown error occurred!';
    if (error.error instanceof ErrorEvent) {

      errorMessage = `Error: ${error.error.message}`;
    } else {

      errorMessage = `Server returned code: ${error.status}, error message is:
${error.message}`;
    }
    return throwError(errorMessage);
  }
}
```

Κώδικας 23 Το Documentservice .ts

```
<footer>
    <p>Copyright &copy; Georgios Chatziefstratiou  All Rights reserved. Desing and
Development by Georgios Chatziefstratiou</p>
</footer>
```

Κώδικας 24 Το Component footer .html

```
<main>
    <h1>Welcome to gDMS</h1>
    <p>This is the home page of gDMS. Explore and get started!</p>
    <div>
      <a class="action-btn" routerLink="/register">Register Now</a>
      <a class="action-btn" routerLink="/login">Login</a>
    </div>

</main>
```

Κώδικας 25 Το Component home.html

```
body {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
  margin: 0;
  font-family: Arial, sans-serif;
  align-items: center;
  justify-content: center;
}
```

```
main {
  padding: 2em;
  background-color: #f8f9fa;
  text-align: center;
}


.action-btn {
  padding: 10px 20px;
  background-color: #28a745;
  color: white;
  text-decoration: none;
  border-radius: 5px;
  transition: background-color 0.3s ease;
  display: inline-block;
  margin: 10px;
}
.action-btn:hover {
  background-color: #218838;
}
```

Κώδικας 26 Το Component home.css

```
body {
    display: flex;
    flex-direction: column;
    min-height: 100vh;
    margin: 0;
    font-family: Arial, sans-serif;
  }

  header {
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 1em;
    background-color: #333;
    color: white;
  }

  .logo {
    margin-right: 10px;
  }

  .nav ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
  }

  .nav ul li {
    display: inline-block;
    margin-left: 20px;
```

```
}

.nav ul li:first-child {
  margin-left: 0;
}

.nav ul li a {
  text-decoration: none;
  color: white;
}

.nav ul li a:hover {
  color: #ccc;
}

footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 1em;
  margin-top: auto;
}

main {
  flex: 1;
  padding: 2em;
  background-color: #f8f9fa;
}

h1 {
  font-size: 2rem;
  margin-bottom: 1em;
}


.form-container {
  display: flex;
  flex-direction: column;
  max-width: 400px;
  margin: auto;
  padding: 20px;
  background-color: white;
  border: 1px solid #ccc;
  border-radius: 8px;
}

.form-container label {
  margin-top: 1em;
}

.form-container input, .form-container select {
  padding: 10px;
  margin-top: 5px;
  border-radius: 4px;
```

```
  border: 1px solid #ccc;
}

.form-container button {
  margin-top: 20px;
  padding: 10px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

.form-container button:hover {
  background-color: #0056b3;
}


.action-btn {
  padding: 10px 20px;
  background-color: #28a745;
  color: white;
  text-decoration: none;
  border-radius: 5px;
  transition: background-color 0.3s ease;
}

.action-btn:hover {
  background-color: #218838;
}


.documents-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  gap: 20px;
  margin-top: 20px;
}

.document-item {
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 8px;
  background-color: #f4f4f4;
}

.document-actions {
  margin-top: 10px;
}

.document-actions button {
  margin-right: 5px;
}
```

Κώδικας27 Το Component login.css

```
<h1>Login to Your Account</h1>
<form class="form-container" (ngSubmit)="login()">
  <label for="email">Email:</label>
  <input type="email" id="email" name="email" [(ngModel)]="email" required />

  <label for="password">Password:</label>
  <input type="password" id="password" name="password" [(ngModel)]="password"
required />

  <button type="submit">Login</button>
</form>
```

Κώδικας 28 Το Component login.html

```
import { Component } from '@angular/core';
import { AuthService } from '../auth.service';
import { Router } from '@angular/router';
import { FormsModule } from '@angular/forms';

@Component({
  selector: 'app-login',
  standalone: true,
  imports: [FormsModule],
  templateUrl: './login.component.html',
  styleUrl: './login.component.css'
})
export class LoginComponent {

  constructor(private authService: AuthService, private router: Router) {}
  private loggedIn = false;
  email: string = '';
  password: string = '';

  login() {

    this.authService.login(this.email, this.password).subscribe(response => {
      if (response.success) {
        this.router.navigate(['/dashboard']);
      } else {

        console.error('Login failed');
      }
    });

    console.log('Email:', this.email);
    console.log('Password:', this.password);
  }

  logout() {
    this.loggedIn = false;
  }
```

```
  isLoggedIn(): boolean {
    return this.loggedIn;
  }


}
```

Κώδικας 29 Το Component login.ts

```
body {
    display: flex;
    flex-direction: column;
    min-height: 100vh;
    margin: 0;
    font-family: Arial, sans-serif;
  }

  header {
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 1em;
    background-color: #333;
    color: white;
  }

  .logo {
    margin-right: 10px;
  }

  .nav ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
  }

  .nav ul li {
    display: inline-block;
    margin-left: 20px;
  }

  .nav ul li:first-child {
    margin-left: 0;
  }

  .nav ul li a {
    text-decoration: none;
    color: white;
  }

  .nav ul li a:hover {
    color: #ccc;
  }
```

```
footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 1em;
  margin-top: auto;
}

main {
  flex: 1;
  padding: 2em;
  background-color: #f8f9fa;
}

h1 {
  font-size: 2rem;
  margin-bottom: 1em;
}


.form-container {
  display: flex;
  flex-direction: column;
  max-width: 400px;
  margin: auto;
  padding: 20px;
  background-color: white;
  border: 1px solid #ccc;
  border-radius: 8px;
}

.form-container label {
  margin-top: 1em;
}

.form-container input, .form-container select {
  padding: 10px;
  margin-top: 5px;
  border-radius: 4px;
  border: 1px solid #ccc;
}

.form-container button {
  margin-top: 20px;
  padding: 10px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

.form-container button:hover {
```

```
    background-color: #0056b3;
  }

  .action-btn {
    padding: 10px 20px;
    background-color: #28a745;
    color: white;
    text-decoration: none;
    border-radius: 5px;
    transition: background-color 0.3s ease;
  }

  .action-btn:hover {
    background-color: #218838;
  }


  .documents-grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    gap: 20px;
    margin-top: 20px;
  }

  .document-item {
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 8px;
    background-color: #f4f4f4;
  }

  .document-actions {
    margin-top: 10px;
  }

  .document-actions button {
    margin-right: 5px;
  }
```

Κώδικας 30 Το Component logout.css

```
<h1>Logout</h1>
<p>You have been logged out successfully.</p>
<a routerLink="/">Go to Home</a>
```

Κώδικας 31 Το Component logout.html

```
import { Component } from '@angular/core';
import { AuthService } from '../auth.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-logout',
```

```
  standalone: true,
  imports: [],
  templateUrl: './logout.component.html',
  styleUrl: './logout.component.css'
})
export class LogoutComponent {


  constructor(private authService: AuthService, private router: Router) {}

  onLogout() {
    this.authService.logout();
    this.router.navigate(['/login']);
  }
}
```

Κώδικας 32 Το Component logout.ts

```
body {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
  margin: 0;
  font-family: Arial, sans-serif;
  background-color: #f8f9fa;
}

header {
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 1em;
  background-color: #333;
  color: white;
}

.logo {
  font-size: 1.5em;
  font-weight: bold;
}

.nav ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

.nav ul li {
  display: inline-block;
  margin-left: 20px;
}
```

```css
.nav ul li:first-child {
  margin-left: 0;
}

.nav ul li a {
  text-decoration: none;
  color: white;
}

.nav ul li a:hover {
  color: #ccc;
}


.wrapper {
  display: flex;
  flex: 1;
  padding: 20px;
}


.sidebar {
  width: 250px;
  background-color: #007bff;
  padding: 20px;
  color: white;
  display: flex;
  flex-direction: column;
  gap: 20px;
}

.sidebar .file-upload-container {
  display: flex;
  flex-direction: column;
  gap: 10px;
  align-items: center;
}

.sidebar label {
  font-weight: bold;
  color: white;
}

.sidebar button {
  background-color: #28a745;
  color: white;
  border: none;
  border-radius: 5px;
  padding: 10px;
  cursor: pointer;
}

.sidebar button:hover {
  background-color: #218838;
```

```
}


.main-content {
  flex: 1;
  padding: 20px;
}

.documents-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  gap: 20px;
}

.document-item {
  padding: 15px;
  border: 1px solid #ccc;
  border-radius: 8px;
  background-color: white;
  display: flex;
  flex-direction: column;
  justify-content: space-between;
}

.document-actions {
  margin-top: 10px;
}

.document-actions button {
  margin-right: 5px;
  padding: 8px;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

.edit-btn {
  background-color: #28a745;
}

.edit-btn:hover {
  background-color: #218838;
}

.delete-btn {
  background-color: #dc3545;
}

.delete-btn:hover {
  background-color: #c82333;
}
```

```
footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 1em;
  margin-top: auto;
}
```

Κώδικας 33 Το Component main-content.css

```
<body>

  <header>
    <div class="logo">gDMS</div>
    <nav class="nav">
      <ul>
        <li><a routerLink="/">Home</a></li>
        <li *ngIf="!authService.isLoggedIn()"><a
routerLink="/register">Register</a></li>
        <li *ngIf="!authService.isLoggedIn()"><a routerLink="/login">Login</a></li>
        <li *ngIf="authService.isLoggedIn()"><a
routerLink="/dashboard">Dashboard</a></li>
        <li *ngIf="authService.isLoggedIn()"><a
routerLink="/profile">Profile</a></li>
        <li *ngIf="authService.isLoggedIn()"><a
routerLink="/settings">Settings</a></li>
        <li *ngIf="authService.isLoggedIn()"><a
routerLink="/documents">Documents</a></li>
        <li *ngIf="authService.isLoggedIn()"><a (click)="logout()">Logout</a></li>
      </ul>
    </nav>
  </header>
  <main>
    <router-outlet></router-outlet>
  </main>
  <footer>
    <app-footer></app-footer>
  </footer>
</body>
```

Κώδικας 34 Το Component main-content.html

```
 import { Component, ElementRef, EventEmitter, Input, Output, ViewChild } from
'@angular/core';
import { AuthService } from '../auth.service';
import { Router } from '@angular/router';

@Component({
  selector: 'app-main-content',
  templateUrl: './main-content.component.html',
  styleUrl: './main-content.component.css'
})
export class MainContentComponent {
```

```
  constructor(public authService: AuthService, private router: Router ) {}


documents: any;

deleteFile(document: any) {

}
editFile(document: any) {

}

errorMessage: string = '';

logout() {
  this.authService.logout();
  this.router.navigate(['/login']);
}

  selectedFile: File | null = null;
  @Input() label: string = 'Select File';
  @Input() acceptedFileTypes: string = '.pdf,.doc,.docx';
  @Output() fileSelected: EventEmitter<File> = new EventEmitter<File>();


  onFileSelected(event: Event): void {
    const inputElement = event.target as HTMLInputElement;
    if (inputElement.files && inputElement.files.length > 0) {
      this.selectedFile = inputElement.files[0];
    }
  }


  clearFileInput(): void {

    this.fileSelected.emit();
  }
}
```

Κώδικας 35 Το Component maincontentcomponent.ts

```
body {
    display: flex;
    flex-direction: column;
    min-height: 100vh;
    margin: 0;
    font-family: Arial, sans-serif;
  }

  header {
```

```
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 1em;
  background-color: #333;
  color: white;
}

.logo {
  margin-right: 10px;
}

.nav ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

.nav ul li {
  display: inline-block;
  margin-left: 20px;
}

.nav ul li:first-child {
  margin-left: 0;
}

.nav ul li a {
  text-decoration: none;
  color: white;
}

.nav ul li a:hover {
  color: #ccc;
}

footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 1em;
  margin-top: auto;
}

main {
  flex: 1;
  padding: 2em;
  background-color: #f8f9fa;
}

h1 {
  font-size: 2rem;
  margin-bottom: 1em;
}
```

```css
.form-container {
  display: flex;
  flex-direction: column;
  max-width: 400px;
  margin: auto;
  padding: 20px;
  background-color: white;
  border: 1px solid #ccc;
  border-radius: 8px;
}

.form-container label {
  margin-top: 1em;
}

.form-container input, .form-container select {
  padding: 10px;
  margin-top: 5px;
  border-radius: 4px;
  border: 1px solid #ccc;
}

.form-container button {
  margin-top: 20px;
  padding: 10px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

.form-container button:hover {
  background-color: #0056b3;
}


.action-btn {
  padding: 10px 20px;
  background-color: #28a745;
  color: white;
  text-decoration: none;
  border-radius: 5px;
  transition: background-color 0.3s ease;
}

.action-btn:hover {
  background-color: #218838;
}


.documents-grid {
  display: grid;
```

```
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    gap: 20px;
    margin-top: 20px;
  }

  .document-item {
    padding: 10px;
    border: 1px solid #ccc;
    border-radius: 8px;
    background-color: #f4f4f4;
  }

  .document-actions {
    margin-top: 10px;
  }

  .document-actions button {
    margin-right: 5px;
  }
```

Κώδικας 36 Το Component profile .css

```
<h1>Your Profile</h1>
<p>Edit your profile information below:</p>
<form class="form-container">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" required />

  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required />

  <button type="submit">Update Profile</button>
</form>
```

Κώδικας 37 Το Component profile .html

```
.dialog-container {
    padding: 20px;
    background-color: white;
    border: 1px solid #ccc;
    border-radius: 8px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    text-align: center;
  }

  .dialog-content {
    margin: 15px 0;
  }

  .dialog-actions {
    display: flex;
    justify-content: center;
  }
```

```
  .action-btn {
    padding: 10px 20px;
    background-color: #007bff;
    color: white;
    border: none;
    border-radius: 5px;
    cursor: pointer;
    transition: background-color 0.3s ease;
  }

  .action-btn:hover {
    background-color: #0056b3;
  }
```

Κώδικας 38 Το Component registermodal.css

```
 <div class="dialog-container">
    <h2>{{ data.success ? 'Success' : 'Error' }}</h2>
    <div class="dialog-content">
      <p>{{ data.message }}</p>
    </div>
    <div class="dialog-actions">
      <button (click)="onClose()">Close</button>
    </div>
  </div>
```

Κώδικας 39 Το Component registermodal .html

```
import { Component } from '@angular/core';
import {   Inject } from '@angular/core';
import { MatDialogRef, MAT_DIALOG_DATA } from '@angular/material/dialog';
import { Router } from '@angular/router';

@Component({
  selector: 'app-register-modal',

  templateUrl: './register-modal.component.html',
  styleUrl: './register-modal.component.css'
})
export class RegisterModalComponent {
  constructor(
    public dialogRef: MatDialogRef<RegisterModalComponent>,
    @Inject(MAT_DIALOG_DATA) public data: { message: string, success: boolean
},private router: Router
  ) {

  }

  ngOnInit() {

    const redirectTimeout = setTimeout(() => {
      this.router.navigate(['/']);
      this.dialogRef.close();
```

```
    }, 3000);


    this.dialogRef.afterClosed().subscribe(() => {
      clearTimeout(redirectTimeout);
    });
  }

  onClose(): void {
    this.dialogRef.close();
    this.router.navigate(['/']);
  }
}
```

Κώδικας 40 Το Component registermodal.ts

```
body {
  display: flex;
  flex-direction: column;
  min-height: 100vh;
  margin: 0;
  font-family: Arial, sans-serif;
}

header {
  display: flex;
  align-items: center;
  justify-content: space-between;
  padding: 1em;
  background-color: #333;
  color: white;
}

.logo {
  margin-right: 10px;
}

.nav ul {
  list-style-type: none;
  margin: 0;
  padding: 0;
}

.nav ul li {
  display: inline-block;
  margin-left: 20px;
}

.nav ul li:first-child {
  margin-left: 0;
}

.nav ul li a {
```

```
  text-decoration: none;
  color: white;
}

.nav ul li a:hover {
  color: #ccc;
}

footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 1em;
  margin-top: auto;
}

main {
  flex: 1;
  padding: 2em;
  background-color: #f8f9fa;
}

h1 {
  font-size: 2rem;
  margin-bottom: 1em;
}


.form-container {
  display: flex;
  flex-direction: column;
  max-width: 400px;
  margin: auto;
  padding: 20px;
  background-color: white;
  border: 1px solid #ccc;
  border-radius: 8px;
}

.form-container label {
  margin-top: 1em;
}

.form-container input, .form-container select {
  padding: 10px;
  margin-top: 5px;
  border-radius: 4px;
  border: 1px solid #ccc;
}

.form-container button {
  margin-top: 20px;
  padding: 10px;
  background-color: #007bff;
```

```
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

.form-container button:hover {
  background-color: #0056b3;
}


.action-btn {
  padding: 10px 20px;
  background-color: #28a745;
  color: white;
  text-decoration: none;
  border-radius: 5px;
  transition: background-color 0.3s ease;
}

.action-btn:hover {
  background-color: #218838;
}


.documents-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  gap: 20px;
  margin-top: 20px;
}

.document-item {
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 8px;
  background-color: #f4f4f4;
}

.document-actions {
  margin-top: 10px;
}

.document-actions button {
  margin-right: 5px;
}
::ng-deep .mat-snack-bar-container {
  display: flex;
  justify-content: center;
  align-items: center;
}


.mat-snack-bar-container {
```

```
  padding: 16px;
  margin: 0 auto;
}
```

Κώδικας 41 Το Component register.css

```
<h1>Create a New Account</h1>
<form class="form-container" (ngSubmit)="onSubmit(registerForm)"
#registerForm="ngForm">
  <label for="username">Username:</label>
  <input type="text" id="name" name="name" required [(ngModel)]="user.name" />

  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required [(ngModel)]="user.email" />

  <label for="password">Password:</label>
  <input type="password" id="password" name="password" required
[(ngModel)]="user.password" />

  <button type="submit">Register</button>
</form>
```

Κώδικας 42 Το Component register .html

```
import { Component } from '@angular/core';
import { User } from '../User';
import { NgForm } from '@angular/forms';
import { RegisterService } from './register.service';
import { MatDialog } from '@angular/material/dialog';
import { Router } from '@angular/router';
import { RegisterModalComponent } from './register-modal/register-modal.component';




@Component({
  selector: 'app-register',
  templateUrl: './register.component.html',
  styleUrl: './register.component.css'
})


export class RegisterComponent {
  user: User = {
    email: '',
    password: '',
    user: '',
    name: '',
    lastname: '',
    phonenumber: '',
    year: '',
    dateOfBirth:  '',
```

```
    city: '',
    state: '',
    zipCode: '',
    country: '',
    active: true,
    role: '',
    profilePictureUrl: '',
    createdDate: '',
    updatedDate: '',
    lastLoginDate: '',
    preferences: '',

  };

  constructor(private registerService: RegisterService,private dialog: MatDialog,
    private router: Router) { }

  successMessage: string = '';
  errorMessage: string = '';
  onSubmit(form: NgForm) {

    console.log('User registered:', this.user);
    if (form.valid) {
      this.registerService.register(this.user).subscribe({
        next: (response) => {
          console.log('User registered successfully:', response);
          this.openModal('User registered successfully!', true);

        },
        error: (error) => {
          console.error('Registration failed:', error);
          this.openModal('Registration failed. Please try again.', false);
        }
      });
    }
  }
  openModal(message: string, success: boolean) {
    this.dialog.open(RegisterModalComponent, {
      data: { message, success },
      width: '400px',
    });

  }


}
```

Κώδικας 43 Το component register .ts

```
import { HttpClient, HttpHeaders } from '@angular/common/http';
import { Injectable } from '@angular/core';
import { Observable } from 'rxjs';
import { User } from '../User';
```

```
@Injectable({
  providedIn: 'root'
})
export class RegisterService {

  private apiUrl = 'http://localhost:8080/api/users/createuser';
  constructor(private http: HttpClient) { }

  register(user: User): Observable<any> {
    const headers = new HttpHeaders({
      'Authorization': 'Basic ' + btoa('user:password'),
      'Content-Type': 'application/json'
    });
    return this.http.post<User>(this.apiUrl, user, { headers });
  }
}
```

Κώδικας 44 Το component register service .ts

```
body {
    display: flex;
    flex-direction: column;
    min-height: 100vh;
    margin: 0;
    font-family: Arial, sans-serif;
  }

  header {
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 1em;
    background-color: #333;
    color: white;
  }

  .logo {
    margin-right: 10px;
  }

  .nav ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
  }

  .nav ul li {
    display: inline-block;
    margin-left: 20px;
  }

  .nav ul li:first-child {
    margin-left: 0;
```

```
}

.nav ul li a {
  text-decoration: none;
  color: white;
}

.nav ul li a:hover {
  color: #ccc;
}

footer {
  background-color: #333;
  color: white;
  text-align: center;
  padding: 1em;
  margin-top: auto;
}

main {
  flex: 1;
  padding: 2em;
  background-color: #f8f9fa;
}

h1 {
  font-size: 2rem;
  margin-bottom: 1em;
}


.form-container {
  display: flex;
  flex-direction: column;
  max-width: 400px;
  margin: auto;
  padding: 20px;
  background-color: white;
  border: 1px solid #ccc;
  border-radius: 8px;
}

.form-container label {
  margin-top: 1em;
}

.form-container input, .form-container select {
  padding: 10px;
  margin-top: 5px;
  border-radius: 4px;
  border: 1px solid #ccc;
}

.form-container button {
```

```
  margin-top: 20px;
  padding: 10px;
  background-color: #007bff;
  color: white;
  border: none;
  border-radius: 5px;
  cursor: pointer;
}

.form-container button:hover {
  background-color: #0056b3;
}


.action-btn {
  padding: 10px 20px;
  background-color: #28a745;
  color: white;
  text-decoration: none;
  border-radius: 5px;
  transition: background-color 0.3s ease;
}

.action-btn:hover {
  background-color: #218838;
}


.documents-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  gap: 20px;
  margin-top: 20px;
}

.document-item {
  padding: 10px;
  border: 1px solid #ccc;
  border-radius: 8px;
  background-color: #f4f4f4;
}

.document-actions {
  margin-top: 10px;
}

.document-actions button {
  margin-right: 5px;
}
```

Κώδικας 45 Το component settings.css

```
<h1>Settings</h1>
<p>Manage your settings here:</p>
<form class="form-container">
  <label for="language">Language:</label>
  <select id="language" name="language">
    <option value="en">English</option>
    <option value="es">Spanish</option>
  </select>

  <label for="notifications">Notifications:</label>
  <input type="checkbox" id="notifications" name="notifications" />

  <button type="submit">Save Settings</button>
</form>
```

Κώδικας 46 Το component settings.html

```
import { NgModule } from '@angular/core';
import { CommonModule } from '@angular/common';
import { ProfileComponent } from './Profile/profile.component';
import { HomeComponent } from './home/home.component';
import { SettingsComponent } from './settings/settings.component';
import { RouterModule, Routes } from '@angular/router';
import { DashboardComponent } from './dashboard/dashboard.component';
import { LoginComponent } from './login/login.component';
import { DocumentsComponent } from './documents/documents.component';
import { RegisterComponent } from './register/register.component';
import { LogoutComponent } from './logout/logout.component';
import { AuthGuard } from './auth.guard';

const routes: Routes = [
  { path: '', component: HomeComponent },
  { path: 'login', component: LoginComponent },
  { path: 'logout', component: LogoutComponent, canActivate: [AuthGuard] },
  { path: 'dashboard', component: DashboardComponent, canActivate: [AuthGuard] },
  { path: 'profile', component: ProfileComponent, canActivate: [AuthGuard] },
  { path: 'settings', component: SettingsComponent, canActivate: [AuthGuard] },
  { path: 'documents', component: DocumentsComponent, canActivate: [AuthGuard] },
  { path: 'register', component: RegisterComponent }
];

@NgModule({
  declarations: [],

  imports: [ CommonModule,RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule {


}
```

Κώδικας 47 Το component app-routing.module.ts

```
import { Injectable } from '@angular/core';
import { ActivatedRouteSnapshot, CanActivate, CanActivateFn, Router,
RouterStateSnapshot } from '@angular/router';
import { AuthService } from './auth.service';


@Injectable({
  providedIn: 'root'
})

export class AuthGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) {}

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot
  ): boolean {
    if (this.authService.isLoggedIn()) {
      return true;
    } else {
      this.router.navigate(['/login']);
      return false;
    }
  }

}
```

Κώδικας 48 Το component authguard.ts

```
import { Injectable } from '@angular/core';
import { of } from 'rxjs';
import { Observable } from 'rxjs/internal/Observable';


@Injectable({
  providedIn: 'root'
})
export class AuthService {
  private loggedIn: boolean = false;

  login(email: string, password: string): Observable<{ success: boolean }> {
    this.loggedIn = true;
    localStorage.setItem('loggedIn', 'true');


    return of({ success: true });
  }

  logout() {
    this.loggedIn = false;
    localStorage.removeItem('loggedIn');
  }
```

```
  isLoggedIn(): boolean {
    return this.loggedIn || localStorage.getItem('loggedIn') === 'true';
  }
}
```

Κώδικας 49 Το component auth.service.ts

```
import { User } from "./User"



export interface Document {

    id: string
    documentname: string
    hash: string
    previousHash: string
    timestamp: string
    //documentMetadata: DocumentMetadata
    user: User

}
```

Κώδικας 50 Το model document .ts

```
export interface FileModel {
    name: string;
    size: number;
    type: string;
    file?:File
  }
```

Κώδικας 51 το model filemodel.ts

```



export interface User {
    user: string;
    email: string;

    password: string;
    name :string;
    lastname :string;
    phonenumber :string;
    year :string;
    dateOfBirth :string
    city :string;
    state: string;
    zipCode :string;
    country:string;
    active: boolean;
    role:string;
    profilePictureUrl: string;
    createdDate:string;
    updatedDate:string
    lastLoginDate:string;
```

```
    preferences:string;
}
```

Κώδικας 52 το model user.ts

```
FROM node:18 AS build

WORKDIR /app
COPY package.json package-lock.json ./
RUN npm install
COPY . .
RUN npm run build --prod


FROM nginx:alpine
COPY --from=build /app/dist /usr/share/nginx/html # No subdirectory needed, just
/app/dist
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

Κώδικας 53 το dockerfile

```
{
  "name": "angular.io-example",
  "version": "0.0.0",
  "description": "Example project from an angular.io guide.",
  "license": "MIT",
  "proxy": "http://localhost:8000",
  "scripts": {
    "ng": "ng",
    "start": "ng serve",
    "build": "ng build",
    "watch": "ng build --watch --configuration development",
    "test": "ng test",
    "e2e": "ng e2e"
  },
  "private": true,
  "dependencies": {
    "@angular/animations": "^18.0.5",
    "@angular/cdk": "^18.2.7",
    "@angular/common": "18.0.5",
    "@angular/compiler": "18.0.5",
    "@angular/core": "18.0.5",
    "@angular/forms": "18.0.5",
    "@angular/material": "^18.2.7",
    "@angular/platform-browser": "18.0.5",
    "@angular/platform-browser-dynamic": "18.0.5",
    "@angular/router": "18.0.5",

    "angular-in-memory-web-api": "~0.18.0",
```

```
    "rxjs": "~7.8.1",
    "tslib": "^2.6.3",
    "zone.js": "~0.14.7"
  },
  "devDependencies": {
    "@angular-devkit/build-angular": "18.0.6",
    "@angular/cli": "18.0.6",
    "@angular/compiler-cli": "18.0.5",
    "@types/jasmine": "~5.1.4",
    "@types/node": "^20.14.9",
    "copyfiles": "^2.4.1",
    "jasmine-core": "~5.1.2",
    "jasmine-marbles": "~0.9.2",
    "jasmine-spec-reporter": "~7.0.0",
    "karma": "~6.4.3",
    "karma-chrome-launcher": "~3.2.0",
    "karma-coverage": "~2.2.1",
    "karma-jasmine": "~5.1.0",
    "karma-jasmine-html-reporter": "~2.1.0",
    "protractor": "~7.0.0",
    "ts-node": "~10.9.2",
    "typescript": "^5.4.5"
  }
}
```

Κώδικας 54 το package.json

Κώδικας για Backend

```
@Configuration
public class CorsConfig implements WebMvcConfigurer {

    @Value("${cors.allowed.origins}")
    private String allowedOrigins;

    @Override
    public void addCorsMappings(CorsRegistry registry) {
        registry.addMapping("/**")
                .allowedOrigins(allowedOrigins)
                .allowedOrigins("http://localhost:4200")
                .allowedMethods("GET", "POST", "PUT", "DELETE", "OPTIONS")
                .allowedHeaders("*")
                .allowCredentials(true);
    }
}
```

Κώδικας 55 CorsConfig.java

```
public class HashUtil {

    public static String generateHash(String data) throws
NoSuchAlgorithmException {
        MessageDigest digest = MessageDigest.getInstance("SHA-256");
        byte[] hash = digest.digest(data.getBytes(StandardCharsets.UTF_8));
        return bytesToHex(hash);
    }

    private static String bytesToHex(byte[] bytes) {
```

```
        StringBuilder hexString = new StringBuilder(2 * bytes.length);
        for (byte b : bytes) {
            String hex = Integer.toHexString(0xff & b);
            if (hex.length() == 1) {
                hexString.append('0');
            }
            hexString.append(hex);
        }
        return hexString.toString();
    }
}
```

Κώδικας 56 hashutil.java

```
public class MediaTypeUtil {

    public static MediaType getMediaType(String mimeType) {
        switch (mimeType) {
            case "application/pdf":
                return MediaType.APPLICATION_PDF;
            case "image/jpeg":
                return MediaType.IMAGE_JPEG;
            case "image/png":
                return MediaType.IMAGE_PNG;

            default:
                return MediaType.APPLICATION_OCTET_STREAM;
        }
    }
}
```

Κώδικας 57 MediaTypeutil.java

```
@Configuration
@Profile("default")
public class S3Config {

    private static final Logger logger =
LoggerFactory.getLogger(MethodHandles.lookup().lookupClass());

    @Value("${aws.region}")
    private String awsRegion;

    @Value("${aws.access-key-id}")
    private String accessKeyId;

    @Value("${aws.secret-access-key}")
    private String secretAccessKey;

    @Value("${aws.bucketName}")
    private String bucket_name;

    @PostConstruct
    public void init() {
        logger.info("region :" + awsRegion);
        logger.info("accessKeyId " + accessKeyId);
        logger.info("secretAccessKey " + secretAccessKey);
        logger.info("bucket name " + bucket_name);
    }

    @Bean
    public S3Client s3Client() {
        S3ClientBuilder builder = S3Client.builder()
```

```
                .region(Region.of(awsRegion))
                .credentialsProvider(StaticCredentialsProvider.create(
                        AwsBasicCredentials.create(accessKeyId,
secretAccessKey)));
        S3Client client = builder.build();
        connectionVerify(client);
        return builder.build();
    }

    private void connectionVerify(S3Client s3Client) {

        try {

s3Client.headBucket(HeadBucketRequest.builder().bucket(bucket_name).build()
);
            logger.info("Connected to S3 bucket: {}", bucket_name);
        } catch (Exception e) {
            logger.error("Failed to connect to S3 bucket: {}", bucket_name,
e);
        }
    }

}
```

Κώδικας 58 S3Config.java

```
@Configuration
@Profile("local")
public class LocalStackConfig {

    private static final Logger logger =
LoggerFactory.getLogger(MethodHandles.lookup().lookupClass());

    @Value("${aws.region}")
    private String awsRegion;

    @Value("${aws.access-key-id}")
    private String accessKeyId;

    @Value("${aws.secret-access-key}")
    private String secretAccessKey;

    @Value("${aws.bucketName}")
    private String bucket_name;

    @Value("${aws.endpoint.s3}")
    private String aws_Endpoint;

    @PostConstruct
    public void init() {
        logger.info("region :" + awsRegion);
        logger.info("accessKeyId " + accessKeyId);
        logger.info("secretAccessKey " + secretAccessKey);
        logger.info("bucket name " + bucket_name);


    }

    private void connectionVerify(S3Client s3Client) {
        try {

s3Client.getBucketLocation(GetBucketLocationRequest.builder().bucket(bucket
_name).build());
            logger.info("Connected to S3 bucket: {}", bucket_name);
```

```
        } catch (NoSuchBucketException e) {
            logger.error("Bucket does not exist: {}", bucket_name);
        } catch (S3Exception e) {
            logger.error("S3 error occurred while connecting to bucket:
{}", bucket_name, e);
        } catch (Exception e) {
            logger.error("Failed to connect to S3 bucket: {}", bucket_name,
e);
        }
    }

    @Bean
    @Lazy
    public S3Client s3Client() {
        S3Client s3Client = S3Client.builder()
                .endpointOverride(URI.create(aws_Endpoint))
                .region(Region.of(awsRegion))
                .credentialsProvider(StaticCredentialsProvider.create(
                        AwsBasicCredentials.create(accessKeyId,
secretAccessKey)
                ))
                .forcePathStyle(true)
                .build();
        connectionVerify(s3Client);
        return s3Client;
    }
}
```

Κώδικας 59 localstackconfig.java

```
@RestController
@RequestMapping("/api/document/")
public class DocumentController {

    private static final Logger logger =
LoggerFactory.getLogger(MethodHandles.lookup().lookupClass());
    @Autowired
    private final DocumentService documentService;
    @Value("${aws.region}")
    private String awsRegion;

    @Value("${aws.access-key-id}")
    private String accessKeyId;

    @Value("${aws.secret-access-key}")
    private String secretAccessKey;

    @Value("${aws.bucketName}")
    private String bucket_name;

    @Autowired
    private final S3Client s3Client;

    @Autowired
    public DocumentController(DocumentService documentService,S3Client
s3Client) {

        this.s3Client = s3Client;
        this.documentService = documentService;
    }


    @PostMapping("/upload/{userId}")
    public ResponseEntity<UploadResponse>
```

```
uploadDocument(@RequestParam("file") MultipartFile file,@PathVariable Long
userId) {
        try {
            UploadResponse response = documentService.uploadDocument(file ,
userId);
            return new ResponseEntity<>(response,
HttpStatus.valueOf(response.getStatusCode()));
        } catch (IOException e) {
            e.printStackTrace();
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @GetMapping("/list-files/")
    public ResponseEntity<List<String>> listFiles() {
        try {
            ListObjectsV2Request listObjectsRequest =
ListObjectsV2Request.builder()
                    .bucket(bucket_name)
                    .build();

            ListObjectsV2Response response =
s3Client.listObjectsV2(listObjectsRequest);
            List<String> fileNames = response.contents().stream()
                    .map(S3Object::key)
                    .collect(Collectors.toList());

            return new ResponseEntity<>(fileNames, HttpStatus.OK);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }


    @GetMapping("/list-files/{userId}")
    public ResponseEntity<List<Map<String,
String>>>listFilesForUser(@PathVariable String userId) {
        try {
            String prefix = "users/" + userId + "/";
            ListObjectsV2Request listObjectsRequest =
ListObjectsV2Request.builder()
                    .bucket(bucket_name)
                    .prefix(prefix)
                    .build();

            ListObjectsV2Response response =
s3Client.listObjectsV2(listObjectsRequest);
            List<Map<String, String>> files = response.contents().stream()
                    .map(s3Object -> {
                        Map<String, String> fileData = new HashMap<>();
                        fileData.put("filePath", s3Object.key());
                        fileData.put("fileName",
s3Object.key().substring(s3Object.key().lastIndexOf('/') + 1));
                        return fileData;
                    })
                    .filter(fileData ->
!fileData.get("filePath").endsWith("/"))
                    .collect(Collectors.toList());

            return new ResponseEntity<>(files, HttpStatus.OK);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
```

```
    }

    @GetMapping("/download/{s3Key}/{id}")
    public ResponseEntity<byte[]> downloadDocument(@PathVariable String
s3Key,Long id) {
        try (InputStream inputStream =
documentService.downloadDocument(s3Key,id)) {
            byte[] content = inputStream.readAllBytes();
            HttpHeaders headers = new HttpHeaders();
            headers.add(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=" + s3Key);
            return new ResponseEntity<>(content, headers, HttpStatus.OK);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @DeleteMapping("/deletedocument/{userId}/{filename}")
    public ResponseEntity<String> deleteDocument(
             @PathVariable("userId") String userId,
            @PathVariable("filename") String fileName) {
        logger.info("Start delete for userId: {}, fileName: {}", userId,
fileName);
        try {
            documentService.deleteDocument(userId, fileName);
            return  ResponseEntity.ok("Document deleted successfully.");
        } catch (Exception e) {
            logger.error("Failed to delete document: {}", e.getMessage(),
e);
            return new ResponseEntity<>("Failed to delete document.",
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @GetMapping("/{id}")
    public ResponseEntity<Resource> getDocument(@PathVariable Long id) {
        try {
            Resource file = documentService.getDocument(id);
            return ResponseEntity.ok()
                    .header(HttpHeaders.CONTENT_DISPOSITION, "attachment;
filename=\"" + file.getFilename() + "\"")
                    .body(file);
        } catch (Exception e) {
            logger.error("Failed to retrieve document", e);
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

}
```

Κώδικας 60 Documentcontroller.java

```
@ControllerAdvice
public class GlobalExceptionHandler {

    @ExceptionHandler(Exception.class)
    public ResponseEntity<String> handleException(Exception ex) {
        return new ResponseEntity<>(ex.getMessage(),
HttpStatus.INTERNAL_SERVER_ERROR);
    }
}
```

Κώδικας 61 GlobalExceptionHandler.java

```
@RestController
@RequestMapping("/api/users/")
public class UserController {

    private static final Logger logger =
LoggerFactory.getLogger(MethodHandles.lookup().lookupClass());

    @Autowired
    private final DocumentService documentService;
    @Autowired
    private final UserService userService;

    @Autowired
    public UserController(DocumentService documentService, UserService
userService) {
        this.documentService = documentService;
        this.userService = userService;
    }

    @PostMapping("/createUserFolder")
    public ResponseEntity createUserFolder(@RequestParam("userName") String
userName    ) {
        try {
            userService.createFolderForUser(userName);
            return new ResponseEntity<>("Folder created successfully.",
HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>("Failed to create folder.",
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @PostMapping("/createuser")
    public ResponseEntity<UserResponse> createUser(@RequestBody User user
) {
        try {
            UserResponse response = userService.createUser(user);
            return new ResponseEntity<>(response, HttpStatus.CREATED);
        } catch (RuntimeException e) {
            return new ResponseEntity<>(new UserResponse(null, null, null,
null, e.getMessage()), HttpStatus.CONFLICT);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<>(new UserResponse(null, null, null,
null, "Failed to create user."), HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }


    @DeleteMapping("/delete/{id}")
    public ResponseEntity<String> deleteUser(@PathVariable Long id) {
        try {
            userService.deleteUser(id);
            return new ResponseEntity<>("User deleted successfully.",
HttpStatus.OK);
        } catch (RuntimeException e) {
            return new ResponseEntity<>(e.getMessage(),
HttpStatus.NOT_FOUND);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<>("Failed to delete user.",
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
```

```
    @PutMapping("/edit/{id}")
    public ResponseEntity<UserResponse> editUser(@PathVariable Long id,
@RequestBody User user) {
        try {
            UserResponse response = userService.editUser(id, user);
            return new ResponseEntity<>(response, HttpStatus.OK);
        } catch (RuntimeException e) {
            return new ResponseEntity<>(new UserResponse(null, null, null,
null, e.getMessage()), HttpStatus.NOT_FOUND);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<>(new UserResponse(null, null, null,
null, "Failed to update user."), HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @DeleteMapping("/deleteUserFolder")
    public ResponseEntity<String> deleteUserFolder(
            @RequestParam String organizationId,
            @RequestParam String userId) {
        try {
            userService.deleteUserFolder(organizationId, userId);
            return new ResponseEntity<>("User folder and all contents
deleted successfully.", HttpStatus.OK);
        } catch (Exception e) {
            return new ResponseEntity<>("Failed to delete user folder.",
HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @GetMapping("/{id}")
    public ResponseEntity<UserResponse> getUser(@PathVariable Long id) {
        try {
            UserResponse userResponse = userService.getUserById(id);
            return new ResponseEntity<>(userResponse, HttpStatus.OK);
        } catch (RuntimeException e) {
            return new ResponseEntity<>(new UserResponse(null, null, null,
null, e.getMessage()), HttpStatus.NOT_FOUND);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<>(new UserResponse(null, null, null,
null, "Failed to retrieve user."), HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }

    @GetMapping("/all")
    public ResponseEntity<List<UserResponse>> getAllUsers() {
        try {
            List<UserResponse> users = userService.getAllUsers();
            return new ResponseEntity<>(users, HttpStatus.OK);
        } catch (Exception e) {
            e.printStackTrace();
            return new ResponseEntity<>(HttpStatus.INTERNAL_SERVER_ERROR);
        }
    }
}
```

Κώδικας 62 usercontroller.java

```
@Entity
@SequenceGenerator(name = "doc_seq", sequenceName = "document_sequence",
allocationSize = 1)
```

```java
public class Document {

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "doc_seq")
    private Long id;

    private String documentName;

    private String hash;

    private String previousHash;

    @Column(columnDefinition = "TIMESTAMP")
    private LocalDateTime timestamp;

    @ManyToOne
    @JoinColumn(name = "document_metadata_id")
    private DocumentMetadata documentMetadata;

    @ManyToOne
    @JoinColumn(name = "user_id", nullable = false)
    private User user;

    public DocumentMetadata getDocumentMetadata() {
        return documentMetadata;
    }

    public void setDocumentMetadata(DocumentMetadata documentMetadata) {
        this.documentMetadata = documentMetadata;
    }

    public Document() {
    }
     public Long getId() {
         return id;
     }

     @PrePersist
     protected void onCreate() {
         timestamp = LocalDateTime.now();
     }

     public String getDocumentName() {
        return documentName;
    }

    public void setDocumentName(String documentName) {
        this.documentName = documentName;
    }

    public String getHash() {
        return hash;
    }

    public void setHash(String hash) {
        this.hash = hash;
    }

    public String getPreviousHash() {
        return previousHash;
    }

    public void setPreviousHash(String previousHash) {
        this.previousHash = previousHash;
    }
```

```
    public LocalDateTime getTimestamp() {
        return timestamp;
    }

    public void setTimestamp(LocalDateTime timestamp) {
        this.timestamp = timestamp;
    }

    public void setApproved(boolean b) {
    }

    public void setId(Long id) {
        this.id = id;
    }

    public User getUser() {
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
}
```

Κώδικας 63 Document.java

```
@Entity
public class DocumentMetadata {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    private String filename;
    private String filePath;
    private String s3Key;
    private long fileSize;

    public DocumentMetadata() {
    }

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getFilename() {
        return filename;
    }

    public void setFilename(String filename) {
        this.filename = filename;
    }

    public String getFilePath() {
        return filePath;
    }

    public void setFilePath(String filePath) {
        this.filePath = filePath;
```

```
    }

    public String getS3Key() {
        return s3Key;
    }

    public void setS3Key(String s3Key) {
        this.s3Key = s3Key;
    }

    public long getFileSize() {
        return fileSize;
    }

    public void setFileSize(long fileSize) {
        this.fileSize = fileSize;
    }

    public void setSize(long size) {
    }
}
```

Κώδικας 64 documentmetadata.java

```
public class UploadResponse {
    private int statusCode;
    private String fileUrl;
    private String s3Key;
    private String documentName;
    private String message;
    public UploadResponse(int statusCode, String fileUrl, String s3Key,
String documentName, String message) {
        this.statusCode = statusCode;
        this.fileUrl = fileUrl;
        this.s3Key = s3Key;
        this.documentName = documentName;
        this.message = message;
    }

    public int getStatusCode() {
        return statusCode;
    }

    public void setStatusCode(int statusCode) {
        this.statusCode = statusCode;
    }

    public String getFileUrl() {
        return fileUrl;
    }

    public void setFileUrl(String fileUrl) {
        this.fileUrl = fileUrl;
    }

    public String getS3Key() {
        return s3Key;
    }

    public void setS3Key(String s3Key) {
        this.s3Key = s3Key;
    }
```

```
    public String getDocumentName() {
        return documentName;
    }

    public void setDocumentName(String documentName) {
        this.documentName = documentName;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

Κώδικας 65 uploadresponse.java

```
@Entity
@SequenceGenerator(name = "user_seq", sequenceName = "user_sequence",
allocationSize = 1)
@Table(name = "app_user")
public class User {

    private static final Logger logger =
LoggerFactory.getLogger(MethodHandles.lookup().lookupClass());

    @Id
    @GeneratedValue(strategy = GenerationType.AUTO, generator = "doc_seq")
    private Long id;
    @OneToMany(mappedBy = "user", cascade = CascadeType.ALL, fetch =
FetchType.LAZY)
    private List<Document> documentList;
    private String name;
    private String lastName;
    private String address;
    private String phoneNumber;
    private String year;
    private String email;
    private String password;
    private LocalDate dateOfBirth;
    private String city;
    private String state;
    private String zipCode;
    private String country;
    private boolean active;
    private String role;
    @ElementCollection(fetch = FetchType.EAGER)
    private Collection<String> roles;

    private String profilePictureUrl;
    private LocalDateTime createdDate;
    private LocalDateTime updatedDate;
    private LocalDateTime lastLoginDate;
    @Lob
    private String preferences;

    public User() {
    }

    public User(String username, String password, boolean enabled, boolean
b, boolean b1, boolean b2, ArrayList<Object> objects) {
    }
```

```
    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public List<Document> getDocumentList() {
        return documentList;
    }

    public void setDocumentList(List<Document> documentList) {
        this.documentList = documentList;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public String getPhoneNumber() {
        return phoneNumber;
    }

    public void setPhoneNumber(String phoneNumber) {
        this.phoneNumber = phoneNumber;
    }

    public String getYear() {
        return year;
    }

    public void setYear(String year) {
        this.year = year;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }
```

```
    public String getPassword() {
        return password;
    }

    public void setPassword(String password) {
        this.password = password;
    }

    public LocalDate getDateOfBirth() {
        return dateOfBirth;
    }

    public void setDateOfBirth(LocalDate dateOfBirth) {
        this.dateOfBirth = dateOfBirth;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public String getZipCode() {
        return zipCode;
    }

    public void setZipCode(String zipCode) {
        this.zipCode = zipCode;
    }

    public String getCountry() {
        return country;
    }

    public void setCountry(String country) {
        this.country = country;
    }

    public boolean isActive() {
        return active;
    }

    public void setActive(boolean active) {
        this.active = active;
    }

    public String getRole() {
        return role;
    }

    public void setRole(String role) {
        this.role = role;
    }

    public String getProfilePictureUrl() {
```

```
        return profilePictureUrl;
    }

    public void setProfilePictureUrl(String profilePictureUrl) {
        this.profilePictureUrl = profilePictureUrl;
    }

    public LocalDateTime getCreatedDate() {
        return createdDate;
    }

    public void setCreatedDate(LocalDateTime createdDate) {
        this.createdDate = createdDate;
    }

    public LocalDateTime getUpdatedDate() {
        return updatedDate;
    }

    public void setUpdatedDate(LocalDateTime updatedDate) {
        this.updatedDate = updatedDate;
    }

    public LocalDateTime getLastLoginDate() {
        return lastLoginDate;
    }

    public void setLastLoginDate(LocalDateTime lastLoginDate) {
        this.lastLoginDate = lastLoginDate;
    }

    public String getPreferences() {
        return preferences;
    }

    public void setPreferences(String preferences) {
        this.preferences = preferences;
    }


}
```

Κώδικας 66 user.java

```
public class UserResponse {

    private Long id;
    private String name;
    private String lastName;
    private String email;
    private String message;

    public UserResponse(Long id, String name, String lastName, String
email, String message) {
        this.id = id;
        this.name = name;
        this.lastName = lastName;
        this.email = email;
        this.message = message;
    }

    public Long getId() {
        return id;
    }
```

```
    public void setId(Long id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getLastName() {
        return lastName;
    }

    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

Κώδικας 66 userresponse.java

```
@Repository
public interface DocumentMetadataRepository extends
JpaRepository<DocumentMetadata, Long> {

}
```

Κώδικας 67 documentmetadatarepository.java

```
@Repository
public interface DocumentRepository extends JpaRepository<Document, Long> {
    Document findTopByOrderByIdDesc();
}
```

Κώδικας 68 documentrepository.java

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {
    Optional<User> findByEmail(String email);

}
```

Κώδικας 69 userrepository.java

```
@Service
public class DocumentService {

    private static final Logger logger =
LoggerFactory.getLogger(MethodHandles.lookup().lookupClass());
    @Autowired
    private S3Client s3Client;
    @Autowired
    private DocumentRepository documentRepository;
    @Autowired
    private DocumentMetadataRepository metadataRepository;
    @Autowired
    private UserRepository userRepository;
    private  String bucketName;
    @Autowired
    public DocumentService(S3Client s3Client, @Value("${aws.bucketName}")
String bucketName) {
        this.s3Client = s3Client;
        this.bucketName = bucketName;
    }

    public MediaType getMediaType(String mimeType) {
        switch (mimeType) {
            case "application/pdf":
                return MediaType.APPLICATION_PDF;
            case "image/jpeg":
                return MediaType.IMAGE_JPEG;
            case "image/png":
                return MediaType.IMAGE_PNG;

            default:
                return MediaType.APPLICATION_OCTET_STREAM;
        }
    }

    public UploadResponse uploadDocument(MultipartFile file ,Long userId)
throws IOException {
        logger.info("starting the upload");
        User user = userRepository.findById(userId)
                .orElseThrow(() -> new RuntimeException("User not found"));

        String bucketFolder =null;
        Document lastDocument =
documentRepository.findTopByOrderByIdDesc();
        logger.info("Starting the process for upload to s3 send sns
etc..");
        String filename = file.getOriginalFilename();
        String fileExtension = "";
        if (filename != null && filename.contains(".")) {
            fileExtension = filename.substring(filename.lastIndexOf(".") +
1);
        }
        if (userId !=null){
            bucketFolder = String.format("users/%d/", userId);
        }
        logger.info("File extension: " + fileExtension);
        String mimeType = fileExtension ;

        String s3Key = bucketFolder + filename;
        String uniqueFileName = UUID.randomUUID().toString();

        logger.info(s3Key + "to path");
        MediaType mediaType = MediaTypeUtil.getMediaType(mimeType);
```

```
        try (InputStream fileInputStream = file.getInputStream()) {
            PutObjectRequest putObjectRequest = PutObjectRequest.builder()
                    .bucket(bucketName)
                    .key(s3Key)
                    .contentType(mediaType.toString())
                    .build();

            s3Client.putObject(putObjectRequest,
RequestBody.fromInputStream(fileInputStream, file.getSize()));

            String fileUrl =
String.format("https://%s.s3.amazonaws.com/%s", bucketName, s3Key);
            logger.info("success uploading file adding metadata to database
");
            DocumentMetadata metadata = new DocumentMetadata();
            metadata.setFilename(file.getOriginalFilename());
            metadata.setS3Key(s3Key);
            metadata.setFilename(filename);
            metadata.setSize(file.getSize());

            metadataRepository.save(metadata);
            String previousHash = lastDocument != null ?
lastDocument.getHash() : "0";
            String currentHash = HashUtil.generateHash(new String("") +
previousHash);

            Document doc = new Document();
            doc.setDocumentMetadata(metadata);
            doc.setDocumentName(filename);
            doc.setApproved(true);
            doc.setUser(user);
            doc.setHash(currentHash);
            doc.setPreviousHash(previousHash);
            doc.setTimestamp(LocalDateTime.now());


            documentRepository.save(doc);
            logger.info(" metadata added to database returning response
code");

            return new UploadResponse(HttpStatus.OK.value(), fileUrl,
s3Key,"","");
        } catch (S3Exception | NoSuchAlgorithmException e) {
            e.printStackTrace();
            throw new RuntimeException("Error uploading file to S3", e);
        }
    }

    public InputStream downloadDocument(String s3Key,Long id ) {
        logger.info("donwload document");
        try {
            GetObjectRequest getObjectRequest = GetObjectRequest.builder()
                    .bucket(bucketName)
                    .key(s3Key)
                    .build();

            return s3Client.getObject(getObjectRequest);
        } catch (S3Exception e) {

            logger.info("Error getting object from S3: " +
e.awsErrorDetails().errorMessage());
            e.printStackTrace();
            throw new RuntimeException("Error downloading file from S3",
```

```
e);
        }
    }

    public UploadResponse editDocument(Long id, MultipartFile file) throws
IOException {
        Document doc = documentRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Document not
found"));

        String oldS3Key = doc.getDocumentMetadata().getS3Key();
        s3Client.deleteObject(DeleteObjectRequest.builder()
                .bucket(bucketName)
                .key(oldS3Key)
                .build());

        return uploadDocument(file, doc.getUser().getId());
    }

    public Resource getDocument(Long id) {
        logger.info("get document");
        Document doc = documentRepository.findById(id)
                .orElseThrow(() -> new RuntimeException("Document not
found"));

        String s3Key = doc.getDocumentMetadata().getS3Key();

        GetObjectRequest getObjectRequest = GetObjectRequest.builder()
                .bucket(bucketName)
                .key(s3Key)
                .build();


        ResponseInputStream<GetObjectResponse> responseStream =
s3Client.getObject(getObjectRequest);


        return new InputStreamResource(responseStream);
    }


    public void deleteDocument(String userId, String fileName) {
        logger.info("Service delete started for userId: {}, fileName: {}",
userId, fileName);

        String fileKey = String.format("users/%s/%s", userId, fileName);
        logger.info("Constructed fileKey for deletion: {}", fileKey);

        try {

            DeleteObjectRequest deleteObjectRequest =
DeleteObjectRequest.builder()
                    .bucket(bucketName)
                    .key(fileKey)
                    .build();


            s3Client.deleteObject(deleteObjectRequest);
            logger.info("Delete operation invoked for: {}", fileKey);


            try {
                s3Client.headObject(HeadObjectRequest.builder()
                        .bucket(bucketName)
                        .key(fileKey)
```

```
                        .build());
                logger.error("File still exists after delete: {}",
fileKey);
            } catch (NoSuchKeyException e) {
                logger.info("File successfully deleted: {}", fileKey);
            }
        } catch (S3Exception e) {
            logger.error("S3Exception during delete: {}",
e.awsErrorDetails().errorMessage(), e);
            throw new RuntimeException("Error deleting file from S3", e);
        } catch (Exception e) {
            logger.error("General exception during delete: {}",
e.getMessage(), e);
            throw new RuntimeException("General error deleting file", e);
        }
    }




}
```

Κώδικας 70 documentservice.java

```
@Service
public class UserService {


    private static final Logger logger =
LoggerFactory.getLogger(MethodHandles.lookup().lookupClass());
    @Autowired
    private S3Client s3Client;
    @Autowired
    private DocumentRepository documentRepository;
    @Autowired
    private DocumentMetadataRepository metadataRepository;
    @Autowired
    private UserRepository userRepository;
    @Value("${aws.bucketName}")
    private  String bucketName;


    public void createFolderForUser( String userId) {
        String folderKey = bucketName + "/" + userId + "/";
        try {
            PutObjectRequest putObjectRequest = PutObjectRequest.builder()
                    .bucket(bucketName)
                    .key(folderKey)
                    .build();


            s3Client.putObject(putObjectRequest, RequestBody.fromBytes(new
byte[0]));
        } catch (S3Exception e) {
            e.printStackTrace();
            throw new RuntimeException("Error creating folder in S3", e);
        }
    }


    public void deleteUserFolder(String organizationId, String userId) {
        String folderKey = organizationId + "/" + userId + "/";
```

```
        try {

            ListObjectsV2Request listObjectsV2Request =
ListObjectsV2Request.builder()
                    .bucket(bucketName)
                    .prefix(folderKey)
                    .build();

            ListObjectsV2Response listObjectsV2Response;
            do {
                listObjectsV2Response =
s3Client.listObjectsV2(listObjectsV2Request);

                for (S3Object s3Object : listObjectsV2Response.contents())
{
                    s3Client.deleteObject(DeleteObjectRequest.builder()
                            .bucket(bucketName)
                            .key(s3Object.key())
                            .build());
                }
                listObjectsV2Request = listObjectsV2Request.toBuilder()

.continuationToken(listObjectsV2Response.nextContinuationToken())
                        .build();
            } while (listObjectsV2Response.isTruncated());

            s3Client.deleteObject(DeleteObjectRequest.builder()
                    .bucket(bucketName)
                    .key(folderKey)
                    .build());

        } catch (S3Exception e) {
            e.printStackTrace();
            throw new RuntimeException("Error deleting user folder from
S3", e);
        }
    }
    public UserResponse createUser(User user) {

        if (userRepository.findByEmail(user.getEmail()).isPresent()) {
            throw new RuntimeException("User with this email already
exists.");
        }
        user.setActive(true);
        user.setCreatedDate(LocalDateTime.now());
        User savedUser = userRepository.save(user);
        createFolderInS3(user.getId());
        return new UserResponse(
                savedUser.getId(),
                savedUser.getName(),
                savedUser.getLastName(),
                savedUser.getEmail(),
                "User created successfully."
        );
    }

    public void deleteUser(Long id) {
        User user = userRepository.findById(id).orElseThrow(() -> new
RuntimeException("User not found."));
        userRepository.delete(user);
    }

    public UserResponse editUser(Long id, User user) {
        User existingUser = userRepository.findById(id).orElseThrow(() ->
new RuntimeException("User not found."));
```

```
        existingUser.setName(user.getName());
        existingUser.setLastName(user.getLastName());
        existingUser.setEmail(user.getEmail());
        existingUser.setAddress(user.getAddress());

        User updatedUser = userRepository.save(existingUser);

        return new UserResponse(
                updatedUser.getId(),
                updatedUser.getName(),
                updatedUser.getLastName(),
                updatedUser.getEmail(),
                "User updated successfully."
        );
    }
    private void createFolderInS3(Long userId) {
        String folderName = "users/" + userId + "/";
        PutObjectRequest putObjectRequest = PutObjectRequest.builder()
                .bucket(bucketName)
                .key(folderName)
                .contentType("application/x-directory")
                .build();

        s3Client.putObject(putObjectRequest, RequestBody.empty());
    }

    public UserResponse getUserById(Long id) {
        Optional<User> userOptional = userRepository.findById(id);
        if (userOptional.isPresent()) {
            User user = userOptional.get();
            return new UserResponse(user.getId(), user.getName(),
user.getEmail(), user.getLastName(), null);
        } else {
            throw new RuntimeException("User not found");
        }
    }

    public List<UserResponse> getAllUsers() {
        List<User> users = userRepository.findAll();
        return users.stream()
                .map(user -> new UserResponse(user.getId(), user.getName(),
user.getLastName(),user.getEmail(),  null))
                .collect(Collectors.toList());
    }
}
```

Κώδικας 71 userservice.java

```
package com.giorgoch.documentmanagementsystem;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableScheduling;

@SpringBootApplication
@EnableScheduling
public class DocumentmanagementsystemApplication {

    public static void main(String[] args) {
       SpringApplication.run(DocumentmanagementsystemApplication.class,
args);
    }

}
```

Κώδικας 72 documentmanagementsystemapplication.java

```
spring.application.name=documentmanagement
server.port=${SERVER_PORT:8080}
spring.servlet.multipart.enabled=true
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=10MB


spring.datasource.url=jdbc:postgresql://localhost:5432/block_db
spring.datasource.username=username
spring.datasource.password=password
spring.datasource.driver-class-name=org.postgresql.Driver

spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDia
lect
spring.datasource.hikari.auto-commit=false

logging.level.org.springframework.jdbc=DEBUG
#logging.level.org.hibernate=DEBUG
#logging.level.com.zaxxer.hikari=DEBUG

aws.access-key-id=
aws.secret-access-key=
aws.region=us-east-1
aws.bucketName=dmsbucketthesis2
spring.profiles.active=${APP_PROFILE:default}


cors.allowed.origins=*
```

Κώδικας 73 Αρχείο application.settings

```


spring.application.name=documentmanagement
server.port=${SERVER_PORT:8080}
spring.servlet.multipart.enabled=true
spring.servlet.multipart.max-file-size=10MB
spring.servlet.multipart.max-request-size=10MB

spring.datasource.url=jdbc:postgresql://localhost:5432/block_db
spring.datasource.username=username
spring.datasource.password=password
spring.datasource.driver-class-name=org.postgresql.Driver


spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.PostgreSQLDia
lect
spring.datasource.hikari.auto-commit=false

logging.level.org.springframework.jdbc=DEBUG


aws.endpoint.s3=http://localhost:4566
aws.endpoint.sqs=http://localhost:4566
aws.region=us-east-1
aws.bucketName=dmsbucketthesis2
aws.secret-access-key=test
aws.access-key-id=test
```

```
cors.allowed.origins=*
```

Κώδικας 74 το Αρχείο application-local.properties

```
FROM openjdk:17-jdk-slim

WORKDIR /app

COPY target/documentmanagementsystem-0.0.1-snapshot.jar
documentmanagementsystem-0.0.1-snapshot.jar


ENTRYPOINT ["java", "-jar", "documentmanagementsystem-0.0.1-snapshot.jar"]

EXPOSE 8080
```

Κώδικας 76 το dockerfile

```
version: '3.8'
services:
  localstack:
    image: localstack/localstack
    ports:
      - "4566:4566"
      - "4571:4571"
    environment:
      -
SERVICES=s3,sqs,sns,apigateway,lambda,dynamodb,ec2,firehose,iam,kinesis,kms
,secretsmanager,stepfunctions,route53,redshift,cloudwatch    1
    volumes:
      - "./localstack:/var/lib/localstack"
```

Κώδικας 77 dockercompose.yml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
       <groupId>org.springframework.boot</groupId>
       <artifactId>spring-boot-starter-parent</artifactId>
       <version>3.3.2</version>
       <relativePath/>
    </parent>
    <groupId>com.giorgoch</groupId>
    <artifactId>documentmanagementsystem</artifactId>
    <version>0.0.1-snapshot</version>
    <name>documentmanagementsystem</name>
    <description>Demo project for Spring Boot</description>
    <url/>
    <licenses>
       <license/>
    </licenses>
    <developers>
       <developer>
          <id>georgios.chatziefstratiou    mscacs22026 </id>
          <name>Georgios Chatziefstratiou</name>
          <email>chatziefstratiougeorgios@gmail.com</email>
```

```
            <timezone>+2</timezone>
            <properties>
               <project> Document Management system Thesis for advance
computing system Master Uniwa</project>
            </properties>
         </developer>
      </developers>
      <scm>
         <connection/>
         <developerConnection/>
         <tag/>
         <url/>
      </scm>
      <properties>
         <java.version>17</java.version>
      </properties>
      <dependencies>
         <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-actuator</artifactId>
         </dependency>
         <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>s3</artifactId>
         </dependency>
         <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-data-jpa</artifactId>
         </dependency>
         <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
         </dependency>
         <dependency>
            <groupId>software.amazon.awssdk</groupId>
            <artifactId>s3</artifactId>
            <version>2.27.7</version>
         </dependency>
         <dependency>
            <groupId>org.postgresql</groupId>
            <artifactId>postgresql</artifactId>
            <scope>runtime</scope>
         </dependency>
         <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-test</artifactId>
            <scope>test</scope>
         </dependency>
      </dependencies>
      <build>
         <plugins>
            <plugin>
               <groupId>org.springframework.boot</groupId>
               <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
         </plugins>
      </build>

</project>
```

Κώδικας 78 Αρχείο pom.xml

Κώδικας Terraform

```
resource "aws_security_group" "instance_sg" {# Security Group for EC2
  name        = "instance_sg"
  description = "Allow inbound traffic for HTTP and SSH"
  vpc_id      = var.vpc_id
  ingress {
    from_port   = 22
    to_port     = 22
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }

  ingress {
    from_port   = 8080
    to_port     = 8080
    protocol    = "tcp"
    cidr_blocks = ["0.0.0.0/0"]
  }
}

resource "aws_instance" "app_instance" {
  ami                    = "ami-0c55b159cbfafe1f0"
  instance_type          = "t2.micro"
  iam_instance_profile   = aws_iam_instance_profile.ec2_instance_profile.name
  vpc_security_group_ids = [aws_security_group.instance_sg.id]

  tags = {
    Name = "SpringBootAppInstance"
  }
}
```

Κώδικας 79 Ec2.tf

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/aws"
      version = "~> 4.0"
    }
  }

  required_version = ">= 1.0"
}

provider "aws" {
  region = var.aws_region
}
```

Κώδικας 80 Main.tf

```
output "s3_bucket_name" {
  value = aws_s3_bucket.documents_bucket.bucket
}

output "db_endpoint" {
  value = aws_db_instance.postgresql.endpoint
}

output "instance_ip" {
  value = aws_instance.app_instance.public_ip
}
```

Κώδικας 81 Outputs.tf

```
# PostgreSQL RDS Instance
resource "aws_db_instance" "postgresql" {
  allocated_storage    = 20
  engine               = "postgres"
  instance_class       = "db.t2.micro"
  username             = var.db_username
  password             = var.db_password
  parameter_group_name = "default.postgres12"
  publicly_accessible  = true
  skip_final_snapshot  = true

  tags = {
    Name = "MyPostgresDB"
  }
}
```

Κώδικας 82 Rds.tf

```
# S3 Bucket
resource "aws_s3_bucket" "documents_bucket" {
  bucket_prefix = var.s3_bucket_prefix
  acl           = "private"
}

resource "aws_iam_role" "ec2_role" {
  name = "ec2_s3_role"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Action = "sts:AssumeRole"
        Effect = "Allow"
        Principal = {
          Service = "ec2.amazonaws.com"
        }
      }
    ]
  })
}

resource "aws_iam_policy" "s3_access_policy" {
```

```
  name        = "s3_access_policy"
  description = "Policy to allow EC2 instance to access S3 bucket"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Effect = "Allow"
        Action = [
          "s3:PutObject",
          "s3:GetObject",
          "s3:ListBucket"
        ]
        Resource = [
          "${aws_s3_bucket.documents_bucket.arn}/*",
          "${aws_s3_bucket.documents_bucket.arn}"
        ]
      }
    ]
  })
}

#
```

Κώδικας 84 S3_iam.tf

```
variable "db_username" {
  description = "The database username"
  type        = string
}

variable "db_password" {
  description = "The database password"
  type        = string
  sensitive   = true
}

variable "s3_bucket_prefix" {
  description = "Prefix for the S3 bucket"
  type        = string
}

variable "vpc_id" {
  description = "The VPC ID where the security group will be created"
  type        = string
}

variable "aws_region" {
  description = "The AWS region to deploy resources in"
  type        = string
  default     = "us-east-1"
}
```

Κώδικας 85 Variables.tf

ΒΙΒΛΙΟΓΡΑΦΙΑ

- Armbrust, M., Fox, A., Griffith, R., Joseph, A. D., Katz, R., Konwinski, A., ... & Zaharia, M. (2010). *A view of cloud computing.* Communications of the ACM, 53(4), 50-58.
- Mell, P., & Grance, T. (2011). *The NIST definition of cloud computing.* National Institute of Standards and Technology. https://nvlpubs.nist.gov/nistpubs/Legacy/SP/nistspecialpublication800-145.pdf
- Amazon Web Services. (n.d.). *Overview of Amazon Web Services.* Retrieved from https://docs.aws.amazon.com/whitepapers/latest/aws-overview/introduction.html
- Krutz, R. L., & Vines, R. D. (2010). *Cloud Security: A Comprehensive Guide to Secure Cloud Computing.* Wiley Publishing.
- Lehr, W., & McKnight, L. W. (2003). *Wireless Internet access: 3G vs. WiFi?* Telecommunications Policy, 27(5-6), 351-370.
- https://developer.hashicorp.com/terraform/tutorials/aws-get-started/infrastructure-as-code
- Douglas, K. (2019). *PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database* (4th ed.). O'Reilly Media.
- Momjian, B. (2015). *PostgreSQL: Introduction and Concepts*. Addison-Wesley.
- Chaudhuri, S., Dayal, U., & Ganti, V. (2001). "Database Technology: Challenges and Opportunities." *IEEE Computer* 34(12): 72-75.
- Riggs, D., & Kline, K. (2008). *SQL in a Nutshell: A Desktop Quick Reference* (3rd ed.). O'Reilly Media.
- https://www.postgresql.org/
- https://www.jetbrains.com/de-de/idea/
- https://code.visualstudio.com/
- https://aws.amazon.com/about-aws/?nc2=h_header
- https://nodejs.org/en/learn/getting-started/introduction-to-nodejs
- https://docs.npmjs.com/
- https://www.docker.com/get-started/
- https://docs.localstack.cloud/overview/
- https://en.wikipedia.org/wiki/Main_Page
- https://github.com/
- https://about.gitlab.com/
- Jung, H.-S., & Seo, Y.-K. (2011). "An Analysis of Open Source Database Management System
- PostgreSQL." *Journal of Software Engineering and Applications*, 4(3), 161-168.
- Amazon Web Services Documentation:
  Amazon S3 Documentation:
  https://docs.aws.amazon.com/AmazonS3/latest/userguide/Welcome.html
- *Amazon EC2 Documentation*:
- https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html
- *Amazon RDS Documentation*:

- https://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/Welcome.html
- *AWS Lambda Documentation* https://docs.aws.amazon.com/lambda/latest/dg/welcome.html
- *Security Groups for Your VPC*:
- https://docs.aws.amazon.com/vpc/latest/userguide/VPC_SecurityGroups.html
- AWS Certified Solutions Architect Official Study Guide: Associate Exam by Ben Piper and David Clinton
- Amazon Web Services in Action by Michael Wittig and Andreas Wittig
- Μεταπτυχιακή Διπλωματική Εργασία Σχεδιασμός και Ανάπτυξη Διαδραστικού, Διαδικτυακού Σχεδιαστικού Εργαλείου Χατζηευστρατίου Γεώργιος Σχολή Θετικών Επιστημών και Τεχνολογίας
- Τμήμα Επιστήμης και Τεχνολογίας Υπολογιστών **https://amitos.library.uop.gr/xmlui/bitstream/handle/123456789/2025/Georgios%20%20Chatziefstratiou.pdf?sequence=1**
- Ανάπτυξη συστήματος βάσει την μεθοδολογία iconix Αλέξανδρος Ν.Χατζηγεωργιου ΕΑΠ πλη 24 2008 Σχεδιασμός Λογισμικού
- Σύστημα έλεγχου ανελκυστήρων Βασίλης Χ.Γερογιαννης ΕΑΠ πλη 24 2004 Σχεδιασμός Λογισμικού 8 Μελέτη Περίπτωσης Ηλεκτρονικό κατάστημα Πάνος φιτσιλης ΕΑΠ πλη 24 2004 Σχεδιασμός Λογισμικού