
# ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΑΤΤΙΚΗΣ

## ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

### ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

**ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

**Ασφαλή Αποκεντρωμένα Ανταλλακτήρια**

**ΓΕΩΡΓΙΟΣ ΠΑΛΛΗΣ**
**Α.Μ. 71347442**

**Εισηγητής: ΙΩΑΝΝΑ ΚΑΝΤΖΑΒΕΛΟΥ**

**ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ**

**ΑΣΦΑΛΗ ΑΠΟΚΕΝΤΡΩΜΕΝΑ ΑΝΤΑΛΛΑΚΤΗΡΙΑ**

**Γεώργιος Πάλλης**
**Α.Μ. 71347442**

**Εισηγητής:**

**ΙΩΑΝΝΑ ΚΑΝΤΖΑΒΕΛΟΥ, ΕΠΙΚΟΥΡΗ ΚΑΘΗΓΗΤΡΙΑ**

Ioanna Kantzavelou
Digitally signed by Ioanna Kantzavelou
Date: 2024.04.11 14:17:36 +03'00'

**Εξεταστική Επιτροπή:**

**ΠΑΝΑΓΙΩΤΗΣ ΚΑΡΚΑΖΗΣ, ΑΝΑΠΛΗΡΩΤΗΣ ΚΑΘΗΓΗΤΗΣ**

Panagiotis Karkazis
Digitally signed by Panagiotis Karkazis
Date: 2024.04.22 10:48:14 +03'00'

**ΖΑΧΑΡΕΝΙΑ ΓΑΡΟΦΑΛΑΚΗ, ΜΕΛΟΣ Ε.ΔΙ.Π.**

Digitally signed by Zacharenia Garofalaki
Date: 2024.04.22 11:55:52 +03'00'

**Ημερομηνία εξέτασης 21/3/2024**

# ΔΗΛΩΣΗ ΣΥΓΓΡΑΦΕΑ ΠΡΟΠΤΥΧΙΑΚΗΣ ΕΡΓΑΣΙΑΣ

Ο/η κάτωθι υπογεγραμμένος Πάλλης Γεώργιος του Δημητρίου, με αριθμό μητρώου 71347442 φοιτητής του Τμήματος Μηχανικών Πληροφορικής και Υπολογιστών της Σχολής Μηχανικών του Πανεπιστημίου Δυτικής Αττικής, δηλώνω ότι:

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

Ο/Η Δηλών

# Περίληψη

Αυτή η εργασία διερευνά την ενσωμάτωση της τεχνολογίας blockchain σε αποκεντρωμένα ανταλλακτήρια (DEX), δίνοντας έμφαση στις δυνατότητές της να ενισχύσει την ασφάλεια και την προστασία των δεδομένων των χρηστών. Ξεκινά με μια λεπτομερή εξέταση των θεμελιωδών στοιχείων του blockchain, ακολουθούμενη από μια ανάλυση των DEX, τονίζοντας τους λειτουργικούς μηχανισμούς, το τρέχον τοπίο και τις εγγενείς προκλήσεις ασφάλειας.

Σημαντική εστίαση της μελέτης είναι η ανάπτυξη μιας εφαρμογής DEX που χρησιμοποιεί έξυπνα συμβόλαια Ethereum και Flutter για το front-end, καταδεικνύοντας την πρακτική εφαρμογή του blockchain στη δημιουργία ασφαλών και αποτελεσματικών πλατφορμών συναλλαγών. Το project περιλαμβάνει ολόκληρη τη διαδικασία ανάπτυξης, από την υλοποίηση έξυπνων συμβολαίων έως την ενσωμάτωση στο front-end.

Οι ανησυχίες για την ασφάλεια εντός των DEX αποτελούν βασικό μέρος της συζήτησης, όπου εξετάζονται κοινά τρωτά σημεία και ιστορικά περιστατικά. Η εργασία αξιολογεί υπάρχουσες λύσεις ασφάλειας, αξιολογώντας την αποτελεσματικότητά τους στο πλαίσιο των DEX.

Συμπερασματικά, η εργασία αντικατοπτρίζει τις επιπτώσεις αυτών των ευρημάτων για το μέλλον των DEX, υπογραμμίζοντας την ανάγκη για συνεχή καινοτομία ασφάλειας και προτείνοντας τομείς για μελλοντική έρευνα. Αυτή η εργασία στοχεύει να συμβάλει στη βάση γνώσεων του blockchain και της αποκεντρωμένης χρηματοδότησης (DeFi), προσφέροντας τόσο θεωρητικές γνώσεις όσο και πρακτικές προοπτικές.

Λέξεις - Κλειδιά: Τεχνολογία Blockchain, Αποκεντρωμένα Ανταλλακτήρια(DEXs), Έξυπνα Συμβόλαια Ethereum, Ασφάλεια στα DEXs, Αυτοματοποιημένοι Δημιουργοί Αγοράς (AMMs), Ανάπτυξη Front-End, Flutter, Liquidity Pools, Διαχείριση Ταυτότητας Χρήστη, DeFi

## Abstract

This thesis explores the integration of blockchain technology in decentralized exchanges (DEXs), emphasizing its potential to enhance security and user data protection. It begins with a detailed examination of blockchain's fundamentals, followed by an analysis of DEXs, highlighting their operational mechanisms, current landscape, and inherent security challenges.

A significant focus of the study is the development of a DEX application using Ethereum smart contracts and Flutter for the front-end, demonstrating the practical application of blockchain in creating secure and efficient trading platforms. The project encapsulates the entire development process, from smart contract implementation to front-end integration.

Security concerns within DEXs form a core part of the discussion, where common vulnerabilities and historical incidents are examined. The thesis evaluates existing security solutions, assessing their effectiveness in the context of DEXs.

In conclusion, the study reflects on the implications of these findings for the future of DEXs, highlighting the necessity for continuous security innovation and suggesting areas for future research. This work aims to contribute to the blockchain and decentralized finance (DeFi) knowledge base, offering both theoretical insights and practical perspectives.

Keywords: Blockchain Technology, Decentralized Exchanges (DEXs), Ethereum Smart Contracts, Security in DEXs,Automated Market Makers (AMMs), Cryptocurrency Trading, DeFi (Decentralized Finance), Smart Contract Vulnerabilities, Front-End Development with Flutter, User Identity Management, Liquidity Pools, Cross-Chain Interoperability, Consensus Mechanisms, Regulatory Compliance in DeFi, Scalability Solutions

# Πίνακας περιεχομένων

# Κατάλογος Εικόνων, Σχημάτων και Πινάκων

# Κεφάλαιο 1: Εισαγωγή

Η έλευση και η επακόλουθη ανάπτυξη της τεχνολογίας blockchain έφερε μια μεταμορφωτική αλλαγή στο τοπίο των ψηφιακών συναλλαγών, προαναγγέλλοντας την έναρξη των αποκεντρωμένων οικονομικών (decentralized finance). Κεντρικό στοιχείο αυτής της αλλαγής είναι τα Αποκεντρωμένα Ανταλλακτήρια (DEX), τα οποία είναι πλατφόρμες που επιτρέπουν τη διαπραγμάτευση κρυπτονομισμάτων ανεξάρτητα από ένα κεντρικό διοικητικό όργανο. Αυτή η εργασία σκιαγραφεί τις θεμελιώδεις αρχές της τεχνολογίας blockchain, υπογραμμίζοντας την καθοριστική συμβολή της στη γένεση και τη λειτουργικότητα των DEX. Η κλιμακούμενη σημασία των DEX στο ευρύτερο πλαίσιο του οικοσυστήματος blockchain εξετάζεται εξονυχιστικά, με ιδιαίτερη έμφαση στη διακριτική ικανότητά τους να διευκολύνουν διαφανείς και άμεσες εμπειρίες συναλλαγών peer-to-peer.

Ωστόσο, η ενοποίηση των DEX συνοδεύεται από αξιοσημείωτες προκλήσεις, ιδιαίτερα στον τομέα της ασφάλειας. Ο στόχος αυτής της εργασίας είναι να διεξαχθεί μια σε βάθος διερεύνηση και ανάλυση αυτών των προκλήσεων ασφαλείας, με στόχο την επινόηση στρατηγικών για την ενίσχυση της ασφάλειας και της ακεραιότητας των δεδομένων των χρηστών στις πλατφόρμες DEX. Αυτή η εργασία εμπλουτίζεται περαιτέρω μέσω της κατασκευής μιας ρεαλιστικής εφαρμογής DEX, η οποία ενσωματώνει τη χρήση έξυπνων συμβολαίων και ολοκληρωμένων στρατηγικών για τη διαχείριση της ταυτότητας των χρηστών, προσφέροντας πολύτιμες γνώσεις σχετικά με την εφαρμογή της τεχνολογίας blockchain στη μείωση των εγγενών κινδύνων ασφαλείας των αποκεντρωμένων ανταλλαγών.

Αυτή η εργασία ορίζει με σαφήνεια τους στόχους της και οριοθετεί το εύρος της, θέτοντας έτσι τις βάσεις για μια εξαντλητική εξέταση της τεχνολογίας blockchain, των DEX και των συναφών επιπτώσεων ασφάλειας.

## 1.1 Εισαγωγή στο Blockchain

Η τεχνολογία Blockchain, είναι στην πραγματικότητα ένα ledger που επιτρέπει την καταγραφή των συναλλαγών σε ένα κατανεμημένο δίκτυο υπολογιστών. Αυτή η τεχνολογία, που δημιουργήθηκε για πρώτη φορά με τη δημιουργία του Bitcoin το 2008 [1] από ένα άτομο ή μια ομάδα γνωστή ως Satoshi Nakamoto, έκτοτε έχει ξεπεράσει την αρχική της εφαρμογή στα κρυπτονομίσματα. Το κύριο χαρακτηριστικό του είναι η δυνατότητα διατήρησης ενός αποκεντρωμένου, διαφανούς και αμετάβλητου αρχείου συναλλαγών, οι οποίες ομαδοποιούνται σε μπλοκ που συνδέονται και ασφαλίζονται χρησιμοποιώντας κρυπτογραφικές αρχές.

Ένα blockchain λειτουργεί ως μια σειρά από συνδεδεμένα μπλοκ, το καθένα από τα οποία περιέχει ένα σύνολο συναλλαγών που επαληθεύονται και ενοποιούνται σε όλο το δίκτυο. Μόλις ένα μπλοκ φτάσει σε μια ορισμένη χωρητικότητα, δημιουργείται ένα νέο μπλοκ και συνδέεται με το προηγούμενο, σχηματίζοντας μια αλυσίδα. Αυτή η δομή διασφαλίζει ότι από τη στιγμή που τα δεδομένα

καταγράφονται, καθίσταται εξαιρετικά δύσκολη η αλλαγή τους, βάζοντας έτσι το σύστημα ένα υψηλό βαθμό ακεραιότητας και ασφάλειας [2] .

Η αποκεντρωμένη φύση του blockchain είναι ένα άλλο χαρακτηριστικό. Σε αντίθεση με τις παραδοσιακές βάσεις δεδομένων που διαχειρίζονται οι κεντρικές αρχές, τα blockchain διατηρούνται συνήθως από ένα δίκτυο κόμβων. Κάθε κόμβος έχει ένα αντίγραφο ολόκληρου του ledger και συμμετέχει στην επικύρωση και την καταγραφή συναλλαγών. Αυτή η αποκέντρωση όχι μόνο εξαλείφει μεμονωμένα σημεία αποτυχίας, αλλά και εκδημοκρατίζει τη διαχείριση δεδομένων, καθιστώντας την ανθεκτική στη λογοκρισία και τον έλεγχο από οποιαδήποτε μεμονωμένη οντότητα.

Κρίσιμο ρόλο στην επίτευξη της ασφάλειας παίζει επίσης η χρήση κρυπτογραφίας, με κάθε χρήστη να διαθέτει ένα δημόσιο και ένα ιδιωτικό κλειδί για την ασφαλή ταυτοποίηση και υπογραφή συναλλαγών. Η χρήση της αλυσίδας μπλοκ επιτρέπει την επιβεβαίωση των συναλλαγών από πολλούς κόμβους πριν ενσωματωθούν σε ένα μπλοκ, αντί για μια κεντρική αρχή. Επιπλέον, η σύνδεση των μπλοκ επιτυγχάνεται με τη χρήση της λειτουργίας Hash, η οποία δυσκολεύει την αλλοίωση των προηγούμενων συναλλαγών, καθιστώντας την διαδικασία αρκετά ανθεκτική. Αυτά τα μέτρα ενισχύουν την ανθεκτικότητα και την ασφάλεια του συστήματος, προστατεύοντας τις συναλλαγές από ανεπιθύμητες παρεμβάσεις και διασφαλίζοντας τη ακεραιότητα των δεδομένων σε όλο το δίκτυο.

Παρά την ασφάλεια που προσφέρει όμως η τεχνολογία αυτή, αντιμετωπίζει και ορισμένες ευπάθειες. Παράδειγμα αποτελεί η επίθεση 51%, όπου κακόβουλοι κατέχουν μεγάλο μέρος του δικτύου, με αποτέλεσμα την δυνατότητα αλλοίωσης της λειτουργίας του, όπως η δημιουργία νέων μπλοκς . Επιπλέον η δυνατότητα διπλής δαπάνης παραμένει πρόβλημα, ενώ η έλλειψη ιδιωτικότητας σε ορισμένα blockchain και προβλήματα κλιμάκωσης επηρεάζουν την αποτελεσματικότητα. Είναι σημαντικό να αντιμετωπίζουμε αυτές τις προκλήσεις για να διασφαλίσουμε τη βελτίωση της απόδοσης και της ασφάλειας στον κόσμο του blockchain.

Η εφαρμογή του της τεχνολογίας αυτής έχει αναπτυχθεί ώστε να περιλαμβάνει διάφορους τομείς, όπως τα οικονομικά, τη διαχείριση της εφοδιαστικής αλυσίδας, την υγειονομική περίθαλψη και τα συστήματα ψηφοφορίας. Οι δυνατότητές του για τη δημιουργία διαφανών, αποτελεσματικών και ασφαλών συστημάτων έχει αναγνωριστεί και διερευνάται σε πολλούς κλάδους.

Τα έξυπνα συμβόλαια, τα συμβόλαια που εκτελούνται μόνοι τους με τους όρους της συμφωνίας γραμμένους απευθείας σε γραμμές κώδικα, είναι μια άλλη καινοτομία που φέρνει το blockchain. Αυτοματοποιούν και επιβάλλουν την εκτέλεση συμβολαίων, δημιουργώντας ένα επίπεδο αποτελεσματικότητας και αξιοπιστίας στις ψηφιακές συναλλαγές. Το Ethereum, που παρουσιάστηκε το 2015, επέκτεινε την εφαρμογή της τεχνολογίας blockchain ενσωματώνοντας μια πλατφόρμα για τη δημιουργία αποκεντρωμένων εφαρμογών (dApps) με χρήση έξυπνων συμβάσεων [3]

Συνοπτικά, η τεχνολογία blockchain προσφέρει μια νέα τεχνολογία  η οποία φέρνει μια νέα προοπτική στο πώς μοιράζονται οι πληροφορίες και πώς καταγράφονται και επαληθεύονται οι ψηφιακές συναλλαγές. Οι δυνατότητές του εκτείνονται πολύ πέρα από την αρχική του εφαρμογή σε κρυπτονομίσματα, υποσχόμενος να φέρει αυξημένη διαφάνεια, ασφάλεια και αποτελεσματικότητα σε ένα ευρύ φάσμα τομέων.

## 1.2 Τα αποκεντρωμένα ανταλλακτήρια

Τα Decentralized Exchanges (DEX) αντιπροσωπεύουν μια καίρια καινοτομία στο οικοσύστημα blockchain, μεταβάλλοντας θεμελιωδώς το τοπίο της εμπορίας ψηφιακών περιουσιακών στοιχείων. Ως πλατφόρμες όπου οι χρήστες μπορούν να ανταλλάσσουν απευθείας κρυπτονομίσματα χωρίς την ανάγκη κεντρικού μεσάζοντα, τα DEX ευθυγραμμίζονται στενά με τις θεμελιώδεις αρχές της τεχνολογίας blockchain: αποκέντρωση, διαφάνεια και ασφάλεια.

Η σημασία των DEX έγκειται κυρίως στην αποκεντρωμένη φύση τους. Τα παραδοσιακά ανταλλακτήρια - centralized exchanges (CEX), λειτουργούν υπό τον έλεγχο ενός μόνο οργανισμού. Αυτή η συγκέντρωση εγκυμονεί πολλούς κινδύνους, συμπεριλαμβανομένης της ευπάθειας σε hacking, της κατάχρησης των κεφαλαίων των χρηστών και της πιθανής ρυθμιστικής παρέμβασης. Τα DEX μετριάζουν αυτούς τους κινδύνους διευκολύνοντας τις συναλλαγές peer-to-peer. Χωρίς κεντρική αρχή, μειώνουν τις πιθανότητες κακής διαχείρισης περιουσιακών στοιχείων και προσφέρουν ανθεκτικότητα έναντι αστοχιών ή επιθέσεων σε όλο το σύστημα [4]

Μια άλλη κρίσιμη πτυχή των DEX είναι η ικανότητά τους να προωθούν τη διαφάνεια. Σε ένα DEX, κάθε συναλλαγή καταγράφεται στο blockchain, καθιστώντας τα δημόσια επαληθεύσιμα και ελεγχόμενα. Αυτή η διαφάνεια χτίζει την εμπιστοσύνη των χρηστών και διασφαλίζει δίκαιες πρακτικές συναλλαγών, σε πλήρη αντίθεση με τους συχνά αδιαφανείς μηχανισμούς των CEX.

Επιπλέον, τα DEX συμβάλλουν στην ευρύτερη προσβασιμότητα και ένταξη στην αγορά κρυπτονομισμάτων. Επιτρέπουν στους χρήστες να διατηρούν τον έλεγχο των ιδιωτικών κλειδιών τους – και κατ’ επέκταση, των περιουσιακών τους στοιχείων – ενισχύοντας την αίσθηση ενδυνάμωσης και ασφάλειας. Αυτό το χαρακτηριστικό είναι ιδιαίτερα ελκυστικό σε όσους δίνουν προτεραιότητα στην αυτονομία και την ιδιωτικότητα στις οικονομικές τους συναλλαγές.

Επιπλέον, τα DEX έχουν γίνει ένα γόνιμο έδαφος για καινοτομία στον χώρο του blockchain. Διαδραματίζουν καθοριστικό ρόλο στην άνοδο και την υιοθέτηση διαφόρων υπηρεσιών DeFi (όπως Αποκεντρωμένη χρηματοδότηση), προσφέροντας λειτουργίες όπως καλλιέργεια απόδοσης, εξόρυξη ρευστότητας και automated market making . Αυτές οι υπηρεσίες, με τη σειρά τους, έχουν προσελκύσει ένα νέο κύμα χρηστών και προγραμματιστών, εμπλουτίζοντας περαιτέρω το οικοσύστημα blockchain.

Παρά τα πλεονεκτήματά τους, τα DEX δεν είναι χωρίς προκλήσεις. Συχνά αντιμετωπίζουν ζητήματα που σχετίζονται με τη ρευστότητα, την εμπειρία χρήστη και την ταχύτητα συναλλαγής. Ωστόσο, οι συνεχείς εξελίξεις και βελτιστοποιήσεις συνεχίζουν να βελτιώνουν την απόδοση και τη χρηστικότητά τους.

Στην ουσία, τα DEX είναι κάτι περισσότερο από πλατφόρμες συναλλαγών. αποτελούν κρίσιμο συστατικό του οικοσυστήματος blockchain, ενσωματώνοντας τις βασικές του αξίες, ενώ παράλληλα οδηγούν την καινοτομία και την υιοθέτηση. Ένα layer 1 chain (πχ ethereum, solana κτλ) ή ένα layer 2 (polygon, arbitrum κτ) είναι απαραίτητο να έχει τουλάχιστον ένα DEX έτσι ώστε να είναι εφικτή η ανταλλαγή tokens πάνω στο blockchain αυτό.

## 1.3 Επισκόπηση των προκλήσεων ασφαλείας στα DEX

Ενώ τα Αποκεντρωμένα Ανταλλακτήρια (DEX) προσφέρουν σημαντικά πλεονεκτήματα έναντι των κεντροποιημένων, ιδίως όσον αφορά την αυτονομία και τη μειωμένη εξάρτηση από μεσάζοντες, δεν είναι απρόσβλητα σε προκλήσεις ασφάλειας. Αυτές οι προκλήσεις πηγάζουν τόσο από την εγγενή πολυπλοκότητα της τεχνολογίας blockchain όσο και από το εξελισσόμενο τοπίο των ψηφιακών απειλών. Η κατανόηση αυτών των προκλήσεων είναι ζωτικής σημασίας για την ανάπτυξη πιο ισχυρών και ασφαλών πλατφορμών DEX.

**Ευπάθειες έξυπνων συμβολαίων**

Στην βάση των περισσότερων DEX βρίσκονται τα έξυπνα συμβόλαια που αυτοματοποιούν τις διαδικασίες ανταλλαγής. Ωστόσο, τα έξυπνα συμβόλαια είναι επιρρεπή σε ευπάθειες λόγω σφαλμάτων στον κώδικα. Δεδομένου ότι τα έξυπνα συμβόλαια είναι αμετάβλητα μετά την ανάπτυξη στο blockchain, οποιαδήποτε ευπάθεια μπορεί να αξιοποιηθεί επανειλημμένα μέχρι να αναπτυχθεί μια νέα έκδοση. . Περιστατικά υψηλού προφίλ που αφορούν εκμεταλλεύσεις έξυπνων συμβολαίων έχουν οδηγήσει σε σημαντικές οικονομικές απώλειες, υπογραμμίζοντας την ανάγκη για ενδελεχή δοκιμή και έλεγχο του κώδικα συμβολαίου [6].

**Κίνδυνοι από το Front-End**

Τα DEX, ενώ είναι αποκεντρωμένα στη βασική λογική συναλλαγών τους, συχνά βασίζονται σε παραδοσιακές τεχνολογίες Web για τις διεπαφές χρήστη τους. Αυτό δημιουργεί ένα σημείο για επιθέσεις όπως τα DNS attakcs ή η πλαστογράφηση ιστοτόπων, όπου οι εισβολείς μπορούν να παραπλανήσουν τους χρήστες να αλληλεπιδράσουν με μια κακόβουλη διεπαφή (phishing) , οδηγώντας ενδεχομένως σε απώλεια κεφαλαίων.

**Κίνδυνοι συγκέντρωσης ρευστότητας**

Τα Liquidity Pools είναι απαραίτητα για τη διευκόλυνση των συναλλαγών σε περιβάλλον DEX. Ωστόσο, μπορεί να είναι επιρρεπείς σε κινδύνους όπως η μόνιμη απώλεια, όπου η αξία των κατατεθειμένων tokens αλλάζει σε σύγκριση με την περίοδο κατάθεσης. Επιπλέον, οι περίπλοκες επιθέσεις μπορούν να σχεδιαστούν για να χειραγωγήσουν τις τιμές της αγοράς και να εκμεταλλευτούν τα συγκεντρωμένα κεφάλαια.

**Θέματα ολοκλήρωσης και διαλειτουργικότητας**

Καθώς τα DEX συχνά αλληλεπιδρούν με διάφορα tokens και άλλες αποκεντρωμένες εφαρμογές (dApps), η ενοποίηση και η διαλειτουργικότητα θέτουν σημαντικές προκλήσεις. Τα ελαττώματα σε αυτές τις αλληλεπιδράσεις μπορούν να οδηγήσουν σε παραβιάσεις ασφάλειας, όπου οι εισβολείς ενδέχεται να εκμεταλλευτούν διαφορετικά πρότυπα και πρακτικές ασφαλείας σε όλες τις πλατφόρμες.

**Ρυθμιστικοί κίνδυνοι και Κίνδυνοι Συμμόρφωσης**

Η αποκεντρωμένη φύση των DEX αποτελεί πρόκληση στη συμμόρφωση με τους κανονισμούς. Χωρίς μια κεντρική αρχή για την επίβλεψη των συναλλαγών, υπάρχει κίνδυνος χρήσης DEX για παράνομες δραστηριότητες. Η διασφάλιση της συμμόρφωσης με τους κανονισμούς καταπολέμησης της νομιμοποίησης εσόδων από παράνομες δραστηριότητες (AML) και η γνώση των κανονισμών των πελατών σας (KYC) είναι μια πρόκληση που ίσως αποκεντρωμένες πλατφόρμες χρειαστεί να αντιμετωπίσουν [5].

**Σφάλμα χρήστη**

Η πολυπλοκότητα της αλληλεπίδρασης με τα DEX και η διαχείριση των δικών του ψηφιακών στοιχείων μπορεί να οδηγήσει σε σφάλματα χρήστη, με αποτέλεσμα την απώλεια κεφαλαίων. Οι επιθέσεις ηλεκτρονικού ψαρέματος (phishing) και οι απάτες είναι επίσης διαδεδομένες, εκμεταλλευόμενες την έλλειψη κατανόησης των χρηστών. Η εκπαίδευση των χρηστών σε ασφαλείς πρακτικές είναι εξίσου σημαντική με τις τεχνολογικές πτυχές της ασφάλειας DEX.

**Επιθέσεις 51%**

Το 51% attack είναι μια από τις προκλήσεις ασφάλειας για τα αποκεντρωμένα ανταλλακτήρια (DEX). Κατά την επίθεση αυτή, μια ομάδα εξορύκτων κατέχει πάνω από το 50% του συνολικού hash rate του δικτύου, δίνοντάς τους τη δυνατότητα να ελέγχουν και να παραποιούν συναλλαγές. Αυτό μπορεί να έχει σοβαρές επιπτώσεις, όπως η διακοπή πληρωμών και η αναστροφή συναλλαγών. Για προστασία, τα DEX πρέπει να λαμβάνουν μέτρα όπως η ενίσχυση του δικτύου, η χρήση προηγμένων αλγορίθμων συναίνεσης και η επαλήθευση των smart contracts [23].

Συμπερασματικά, ενώ τα DEX φέρνουν επανάσταση στον κόσμο της ανταλλαγής ψηφιακών περιουσιακών στοιχείων μετριάζοντας ορισμένους από τους κινδύνους που υπάρχουν στα κεντρικά συστήματα, εισάγουν ένα νέο σύνολο προκλήσεων ασφάλειας. Η αντιμετώπιση αυτών των προκλήσεων απαιτεί μια πολύπλευρη προσέγγιση που περιλαμβάνει ασφαλή ανάπτυξη έξυπνων συμβολαίων, ισχυρά μέτρα ασφαλείας στο front-end, εκπαίδευση των χρηστών και τήρηση ρυθμιστικών προτύπων. Καθώς η τεχνολογία εξελίσσεται, πρέπει να ισχύουν και οι στρατηγικές για την προστασία αυτών των καινοτόμων πλατφορμών.

## 1.4 Σκοπός Εργασίας

Κεντρικός στόχος της παρούσας εργασίας είναι να πραγματοποιήσει μια σχολαστική εξέταση της τεχνολογίας blockchain, με ιδιαίτερη έμφαση στην ανάπτυξή των Αποκεντρωμένων Ανταλλακτηρίων (DEX). Ο πρωταρχικός στόχος αυτής της εργασίας είναι να αξιολογήσει τις δυνατότητες του blockchain ως εργαλείου για την ενίσχυση της ασφάλειας και τη διαφύλαξη των δεδομένων των χρηστών μέσα σε

αποκεντρωμένα οικοσυστήματα συναλλαγών. Αυτή η εργασία βασίζεται στην προϋπόθεση ότι, ενώ τα DEX προσφέρουν σημαντικά πλεονεκτήματα έναντι των συμβατικών, κεντρικών ομολόγων, παρουσιάζουν ταυτόχρονα διακριτές προκλήσεις, κυρίως στους τομείς της ασφάλειας και της προστασίας δεδομένων.

Μια βασική πτυχή αυτής της εργασίας περιλαμβάνει την ανάπτυξη ενός τέτοιου DEX η οποία περιλαμβάνει smart contracts, σύνδεση με ένα τοπικό blockchain καθώς και ένα γραφικό περιβάλλον (front-end) το οποίο αλληλεπιδρά με το blockchain και μπορεί να συνδέεται με γνωστά wallets. Η αναπτυξιακή τροχιά αυτής της πλατφόρμας χρησιμοποιείται ως μελέτη περίπτωσης, παρέχοντας ρεαλιστικά στοιχεία για την εφαρμογή της τεχνολογίας blockchain για την ενίσχυση του πλαισίου ασφαλείας των DEX.

Το αντικείμενο της παρούσας εργασίας περιλαμβάνει πολλούς κρίσιμους τομείς:

**Μελέτη της τεχνολογίας Blockchain**: Μια σε βάθος μελέτη της τεχνολογίας blockchain, των θεμελιωδών αρχών της και του τρόπου με τον οποίο στηρίζει τις αποκεντρωμένες ανταλλαγές. Αυτή η μελέτη παρέχει το απαραίτητο υπόβαθρο για την κατανόηση της επακόλουθης ανάπτυξης εφαρμογών.

**Ανάλυση των DEX**: Μια λεπτομερής εξέταση των DEX, συμπεριλαμβανομένων των μηχανισμών λειτουργίας τους, των πλεονεκτημάτων και των προκλήσεων ασφαλείας που αντιμετωπίζουν.

**Ανάπτυξη μιας εφαρμογής DEX:** Δημιουργία μιας εφαρμογής DEX από την αρχή, χρησιμοποιώντας έξυπνα συμβόλαια για βασικές λειτουργίες. Αυτό περιλαμβάνει το σχεδιασμό, την ανάπτυξη, τη δοκιμή και την ανάπτυξη της εφαρμογής, παρέχοντας μια πρακτική κατανόηση του τρόπου με τον οποίο η τεχνολογία blockchain μπορεί να εφαρμοστεί σε σενάρια πραγματικού κόσμου.

**Αξιολόγηση Εφαρμοσμένων Τεχνολογιών:** Μια κριτική ανάλυση των τεχνολογιών που εφαρμόζονται στην εφαρμογή DEX, εστιάζοντας ιδιαίτερα σε smart contracts και αποκεντρωμένες εφαρμογές (dApps). Αυτό περιλαμβάνει αξιολόγηση της αποτελεσματικότητάς τους στην ενίσχυση της ασφάλειας και της προστασίας των δεδομένων των χρηστών.

# Κεφάλαιο 2: Βιβλιογραφική Ανασκόπηση

Αυτό το κεφάλαιο παρέχει το υπόβαθρο και ανασκόπηση βιβλιογραφίας που θεωρούνται σημαντικά για την κατανόηση των θεμελιωδών στοιχείων που διέπουν τα DEXes. Ξεκινά με μια εις βάθος εξερεύνηση της ιστορίας και της εξέλιξης της τεχνολογίας blockchain ξεκινώντας από το Bitcoin. Στη συνέχεια, προχωρά στην ανάλυση των θεμελιωδών στοιχείων της τεχνολογίας blockchain, περιλαμβάνοντας τα αναπόσπαστα στοιχεία της, όπως μπλοκ, συναλλαγές και μηχανισμούς συναίνεσης. Αυτά τα στοιχεία είναι κρίσιμα για την κατανόηση της πολυπλοκότητας και των εγγενών χαρακτηριστικών ασφαλείας των συστημάτων blockchain.

Μετά, μια εισαγωγική συζήτηση για τα DEX θέτει τις βάσεις για την εκτίμηση του ρόλου και της σημασίας τους στο οικοσύστημα του blockchain Αυτή η κατανόηση εμβαθύνεται περαιτέρω με την αντίθεση των DEX με τα Centralize Exchanges τους, διευκρινίζοντας τις πρωταρχικές διακρίσεις και τα πλεονεκτήματα που προσφέρουν τα DEX, ιδιαίτερα σε ό,τι αφορά την ασφάλεια, την αυτονομία του χρήστη και την αρχή της έλλειψης εμπιστοσύνης. Πραγματοποιείται εξέταση της τρέχουσας κατάστασης των DEX, παρέχοντας μια επισκόπηση της εξελικτικής τους τροχιάς, των τάσεων χρήσης και του ευρύτερου αντίκτυπου που ασκούν στον οικονομικό τομέα και όχι μόνο. Επίσης παρουσιάζεται το Automated Market Maker (AMM) μοντέλο το οποίο ακολουθείται στα DEXes σε αντίθεσή με το Order Book που υπάρχει στα CEXes.

Η ασφάλεια αναδεικνύεται ως πρωταρχικό μέλημα στο πλαίσιο των αποκεντρωμένων ανταλλαγών. Αντίστοιχα, αυτό το κεφάλαιο εξετάζει την πληθώρα των προκλήσεων ασφαλείας που ενδημούν στα DEX, εντοπίζοντας επικρατούντα τρωτά σημεία και εξιστορώντας ιστορικά περιστατικά που έχουν επηρεάσει τη σύγχρονη προοπτική για την ασφάλεια DEX. Αυτή η εξέταση περιλαμβάνει μια ανάλυση διαφόρων τύπων κινδύνου που αντιμετωπίζουν τόσο οι χρήστες όσο και οι πλατφόρμες, συμπεριλαμβανομένων ζητημάτων που προκύπτουν από ευπάθειες έξυπνων συμβολαίων, προκλήσεις εκ των προτέρων και ρευστότητας.

Το κεφάλαιο ολοκληρώνεται με την ανασκόπηση των υφιστάμενων λύσεων και μεθοδολογιών που αναπτύχθηκαν για την ενίσχυση της ασφάλειας των DEX. Αυτό περιλαμβάνει τεχνικές στρατηγικές, όπως η βελτίωση του σχεδιασμού έξυπνων συμβολαίων και των περιεκτικών διαδικασιών ελέγχου, παράλληλα με εννοιολογικά μέτρα όπως ρυθμιστικά πλαίσια και πρωτοβουλίες εκπαίδευσης των χρηστών. Μέσω της εξέτασης αυτών των λύσεων, το κεφάλαιο προσπαθεί να μεταδώσει μια ολιστική κατανόηση των οδών μέσω των οποίων μπορεί να ενισχυθεί η ασφάλεια σε αποκεντρωμένες ανταλλαγές, θέτοντας έτσι τη βάση για τα επόμενα κεφάλαια που εμβαθύνουν στην εφαρμογή αυτών των αρχών στην κατασκευή μιας ασφαλούς πλατφόρμας DEX.

## 2.1 Τεχνολογία Blockchain

Η τεχνολογία του blockchain εμφανίστηκε στα τέλη της δεκαετίας του 2000, μια ταραχώδη εποχή που χαρακτηρίζεται από σημαντική παγκόσμια χρηματοπιστωτική αστάθεια. Ήταν το 2008 που ένα άτομο ή μια συλλογικότητα, που λειτουργούσε με το ψευδώνυμο Satoshi Nakamoto, παρουσίασε το whitepaper με τίτλο «Bitcoin: A Peer-to-Peer Electronic Cash System»[1] . Αυτό το paper ήταν καθοριστικό για την καθιέρωση των θεμελιωδών αρχών του πρώτου blockchain, που χρησιμεύει ως η ραχοκοκαλιά για το Bitcoin που ήταν το πρώτο κρυπτονόμισμα. Η επαναστατική πτυχή του Bitcoin ήταν το αποκεντρωμένο ledger, το οποίο κατέγραφε τις συναλλαγές σε πολλούς υπολογιστές. Αυτό το πλαίσιο εξασφάλισε ότι το ιστορικό αρχείο δεν θα μπορούσε να τροποποιηθεί αναδρομικά χωρίς να τροποποιηθούν όλα τα επόμενα μπλοκ και να επιτευχθεί η συναίνεση του δικτύου.

# Bitcoin: A Peer-to-Peer Electronic Cash System

Satoshi Nakamoto
satoshin@gmx.com
www.bitcoin.org

**Abstract.** A purely peer-to-peer version of electronic cash would allow online payments to be sent directly from one party to another without going through a financial institution. Digital signatures provide part of the solution, but the main benefits are lost if a trusted third party is still required to prevent double-spending. We propose a solution to the double-spending problem using a peer-to-peer network. The network timestamps transactions by hashing them into an ongoing chain of hash-based proof-of-work, forming a record that cannot be changed without redoing the proof-of-work. The longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. As long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they'll generate the longest chain and outpace attackers. The network itself requires minimal structure. Messages are broadcast on a best effort basis, and nodes can leave and rejoin the network at will, accepting the longest proof-of-work chain as proof of what happened while they were gone.

Εικόνα 2.1: Το whitepaper του bitcoin [1].

Στην αρχική φάση μετά την εισαγωγή του Bitcoin, η προσοχή επικεντρώθηκε κυρίως γύρω από την πτυχή του κρυπτονομίσματος. Ωστόσο, δεν άργησε να αρχίσει να προσελκύει σημαντικό ενδιαφέρον

το εγγενές δυναμικό της υποκείμενης τεχνολογίας blockchain. Η κύρια καινοτομία του blockchain έγκειται στην ικανότητά του να διευκολύνει τη συναίνεση σε ένα αποκεντρωμένο δίκτυο, διατηρώντας έτσι την ακεραιότητα και τη διαφάνεια των δεδομένων χωρίς την εξάρτηση από έναν αξιόπιστο μεσάζοντα. Αυτό πραγματοποιήθηκε μέσω μιας μίξης κρυπτογραφικών τεχνικών και ενός μηχανισμού συναίνεσης που είναι γνωστός ως Proof of Work (PoW) [7].

Με την αυξανόμενη αναγνώριση των δυνατοτήτων της τεχνολογίας blockchain, οι εφαρμογές της άρχισαν να διαφοροποιούνται, εκτείνοντας πέρα από τη σφαίρα των κρυπτονομισμάτων. Μια κομβική στιγμή σε αυτήν την εξελικτική τροχιά ήταν η εισαγωγή του Ethereum το 2015. Το Ethereum έφερε στο προσκήνιο τη νέα έννοια των έξυπνων συμβολαίων, τα οποία είναι αυτοεκτελούμενα συμβόλαια με τους όρους της συμφωνίας ενσωματωμένους στον ίδιο τον κώδικα. Αυτή η πρόοδος επέκτεινε τη χρησιμότητα του blockchain από τις στοιχειώδεις συναλλαγές σε πιο περίπλοκες αποκεντρωμένες εφαρμογές (dApps), επιτρέποντας μια πληθώρα εφαρμογών όπως η αποκεντρωμένη χρηματοδότηση (DeFi), η διαχείριση της εφοδιαστικής αλυσίδας και η επαλήθευση ψηφιακής ταυτότητας [8].

Οι μεταγενέστερες εξελίξεις στο τοπίο του blockchain χαρακτηρίστηκαν από ταχεία καινοτομία. Αξιοσημείωτα μεταξύ αυτών είναι η εμφάνιση του Proof of Stake (PoS), ενός μηχανισμού συναίνεσης που προσφέρει μεγαλύτερη ενεργειακή απόδοση σε σύγκριση με το PoW, και οι εξελίξεις στις λύσεις επεκτασιμότητας και διαλειτουργικότητας [9]. Επί του παρόντος, το blockchain αναγνωρίζεται όχι μόνο για το ρόλο του στα κρυπτονομίσματα, αλλά ως μια τεχνολογία μετασχηματισμού με την ικανότητα να φέρει επανάσταση σε διάφορους κλάδους. Υπόσχεται πρωτοφανή επίπεδα διαφάνειας, ασφάλειας και αποτελεσματικότητας.

Αυτή η τροχιά, που ξεκινά με την έναρξη του Bitcoin και καταλήγει στις πολλές εφαρμογές του blockchain σήμερα, αποτελεί την επιτομή μιας αλλαγής στη διαχείριση δεδομένων και στις μεθοδολογίες συναλλαγών.

Τα βασικά χαρακτηριστικά του blockchain είναι η αποκέντρωση, η ανωνυμία, η ανθεκτικότητα και η δυνατότητα ελέγχου.

Στον πυρήνα της τεχνολογίας blockchain βρίσκονται τρεις θεμελιώδεις έννοιες: **μπλοκ**, **συναλλαγές** και **μηχανισμοί συναίνεσης**. Η κατανόηση αυτών των στοιχείων είναι ζωτικής σημασίας για την κατανόηση του τρόπου με τον οποίο το blockchain λειτουργεί ως ένα ασφαλές και διαφανές ψηφιακό καθολικό.

**Μπλοκ**

Το blockchain είναι μια αλυσίδα μπλοκ, το καθένα χρησιμεύει ως container για δεδομένα. Αυτά τα μπλοκ συνδέονται διαδοχικά, δημιουργώντας ένα μη αναστρέψιμο χρονοδιάγραμμα δεδομένων όταν υλοποιούνται με αποκεντρωμένο τρόπο. Κάθε μπλοκ περιέχει μια λίστα συναλλαγών, μια αναφορά στο προηγούμενο μπλοκ (γνωστό ως κατακερματισμός) και το δικό του μοναδικό κατακερματισμό (hash). Ο κατακερματισμός λειτουργεί ως ψηφιακό δακτυλικό αποτύπωμα, που δημιουργείται από μια μαθηματική συνάρτηση που μετατρέπει τις ψηφιακές πληροφορίες σε μια σειρά από αριθμούς και γράμματα. Εάν

αυτές οι πληροφορίες υποβληθούν σε επεξεργασία με οποιονδήποτε τρόπο, αλλάζει και ο κωδικός κατακερματισμού.

| Block Number | Previous Hash | Timestamp | Data (Transactions) | Hash |

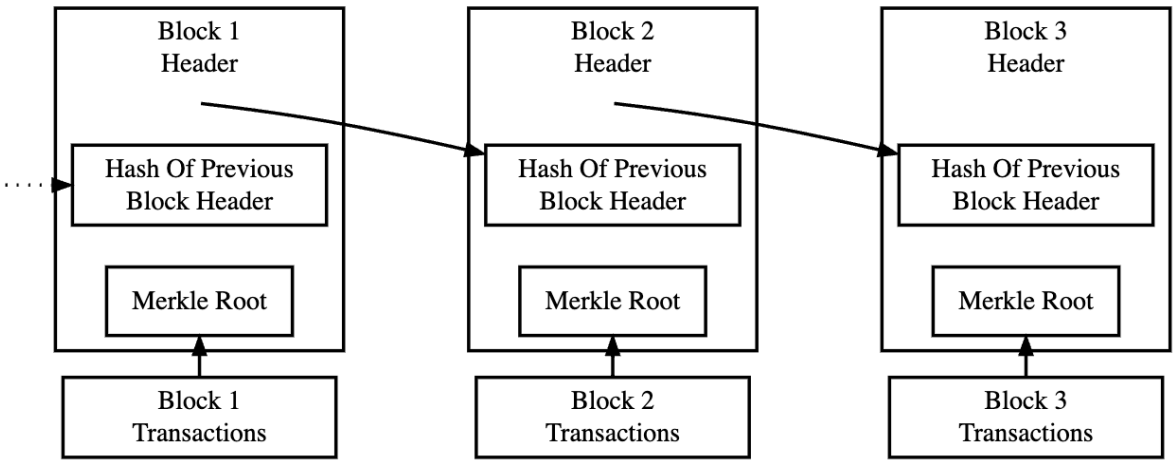
Η δομή ενός block φαίνεται παρακάτω:

**Block Header**: Περιέχει μεταδεδομένα σχετικά με το μπλοκ, όπως:

- Αριθμός μπλοκ: Η θέση του μπλοκ στην αλυσίδα μπλοκ.
- Timestamp: Η ημερομηνία και η ώρα που δημιουργήθηκε το μπλοκ.
- Nonce: Μια τιμή που χρησιμοποιείται κατά τη διαδικασία εξόρυξης σε αλυσίδες μπλοκ Proof of Work (PoW) για την εύρεση ενός κατακερματισμού κάτω από έναν συγκεκριμένο στόχο.
- Merkle Root: Κατακερματισμός όλων των συναλλαγών στο μπλοκ, διασφαλίζοντας την ακεραιότητα των συναλλαγών. [10]

**Δεδομένα συναλλαγής:** Η πραγματική λίστα των συναλλαγών που περιλαμβάνονται στο μπλοκ. Κάθε συναλλαγή περιέχει συνήθως στοιχεία αποστολέα και παραλήπτη, την τιμή που μεταφέρθηκε και έναν κατακερματισμό συναλλαγής.

**Previous Block Hash:** Ένας κρυπτογραφικός κατακερματισμός (hash) της κεφαλίδας του προηγούμενου μπλοκ, ο οποίος συνδέει τα μπλοκ μεταξύ τους σε μια αλυσίδα και διασφαλίζει την αμετάβλητη της αλυσίδας μπλοκ.




Εικόνα 2.2: Αφαιρετική αναπαράσταση του blockchain του bitcoin.

Μια συνάρτηση κατακερματισμού στην τεχνολογία blockchain είναι ένας μαθηματικός αλγόριθμος που μετατρέπει μια είσοδο (ή «μήνυμα») σε μια συμβολοσειρά byte σταθερού μεγέθους. Η έξοδος, γνωστή ως κατακερματισμός, εμφανίζεται τυχαία και αλλάζει σημαντικά με ακόμη και μια μικρή αλλαγή στην είσοδο.

**Ιδιότητες μιας συνάρτησης κατακερματισμού:**

- Ντετερμινιστική: Η ίδια είσοδος παράγει πάντα την ίδια έξοδο.
- Γρήγορος Υπολογισμός: Η τιμή κατακερματισμού υπολογίζεται γρήγορα από την είσοδο.
- Μη αντιστρέψιμη: Δεδομένης μιας τιμής κατακερματισμού, θα πρέπει να είναι υπολογιστικά ανέφικτο να βρεθεί η αρχική είσοδος.
- Μικρές αλλαγές στην είσοδο αλλάζουν την έξοδο σημαντικά: Ακόμη και μια μικρή αλλαγή στα δεδομένα εισόδου θα πρέπει να παράγει μια εντελώς διαφορετική έξοδο.
- Αντίσταση σε σύγκρουση: Θα πρέπει να είναι δύσκολο να βρείτε δύο διαφορετικές εισόδους που παράγουν τον ίδιο κατακερματισμό εξόδου.

**Μαθηματική απεικόνιση:**
Εξετάζουμε μια απλοϊκή συνάρτηση κατακερματισμού για επεξηγηματικούς σκοπούς:

$$hash(x) = (x * 2654435761 \bmod 2^{32}).$$

Αυτό δεν χρησιμοποιείται στην πράξη, αλλά δείχνει την ιδέα του πώς μια μικρή αλλαγή στο x αλλάζει σημαντικά το αποτέλεσμα κατακερματισμού.

Για x = 1, ο κατακερματισμός μπορεί να είναι 12345678. Η αλλαγή του x σε 2 μπορεί να αποφέρει έναν εντελώς διαφορετικό κατακερματισμό όπως το 87654321, που δείχνει την ευαισθησία των συναρτήσεων κατακερματισμού στις αλλαγές εισόδου.

Στο blockchain, η συνάρτηση κατακερματισμού διαδραματίζει κρίσιμο ρόλο στη διατήρηση της ακεραιότητας και της ασφάλειας του blockchain. Για παράδειγμα, σε συστήματα PoW όπως το Bitcoin, οι εξορύκτες ανταγωνίζονται για να βρουν ένα nonce που έχει ως αποτέλεσμα έναν κατακερματισμό κεφαλίδας μπλοκ που είναι μικρότερος από τον στόχο δυσκολίας του δικτύου. Αυτή η διαδικασία, που ονομάζεται εξόρυξη, είναι υπολογιστικά εντατική και προστατεύει το δίκτυο από δόλιες τροποποιήσεις του blockchain. [11]

**Συναλλαγές**

Οι συναλλαγές είναι οι ενέργειες που πραγματοποιούνται σε ένα δίκτυο blockchain. Στο πλαίσιο των κρυπτονομισμάτων, μια συναλλαγή περιλαμβάνει τη μεταφορά ψηφιακού νομίσματος μεταξύ δύο μερών. Αυτές οι συναλλαγές συγκεντρώνονται σε μπλοκ. Πριν προστεθούν σε ένα μπλοκ, επαληθεύονται από τους συμμετέχοντες στο δίκτυο, γνωστούς ως κόμβους, διασφαλίζοντας την εγκυρότητά τους [12].

Μια τυπική συναλλαγή blockchain περιέχει πολλά βασικά στοιχεία:

- Διευθύνσεις αποστολέα και παραλήπτη: Οι διευθύνσεις του δημόσιου πορτοφολιού του αποστολέα και του παραλήπτη της συναλλαγής.
- Αξία: Η ποσότητα του κρυπτονομίσματος ή των δεδομένων που αποστέλλονται.
- Δεδομένα εισόδου: Πρόσθετα δεδομένα που μπορεί να συνοδεύουν τη συναλλαγή, όπως οδηγίες έξυπνου συμβολαίου.
- Προμήθεια συναλλαγής: Προμήθεια που καταβάλλεται σε επικυρωτές δικτύου ή εξορύκτες για την επεξεργασία της συναλλαγής.
- Υπογραφή: Ψηφιακή υπογραφή που δημιουργείται από το ιδιωτικό κλειδί του αποστολέα, το οποίο παρέχει απόδειξη ιδιοκτησίας και εξουσιοδότησης της συναλλαγής.
- Nonce: Ένας διαδοχικός αριθμός μοναδικός για κάθε συναλλαγή από έναν συγκεκριμένο αποστολέα, που διασφαλίζει ότι οι συναλλαγές διεκπεραιώνονται με τη σειρά και αποτρέπει τη διπλή δαπάνη.

**Μετάδοση και Επικύρωση Συναλλαγών**

**Έναρξη**: Μια συναλλαγή ξεκινά όταν ένας αποστολέας δημιουργεί και υπογράφει μια συναλλαγή χρησιμοποιώντας το ιδιωτικό του κλειδί. Η υπογραφή εγγυάται τη γνησιότητα και την ακεραιότητα της συναλλαγής.

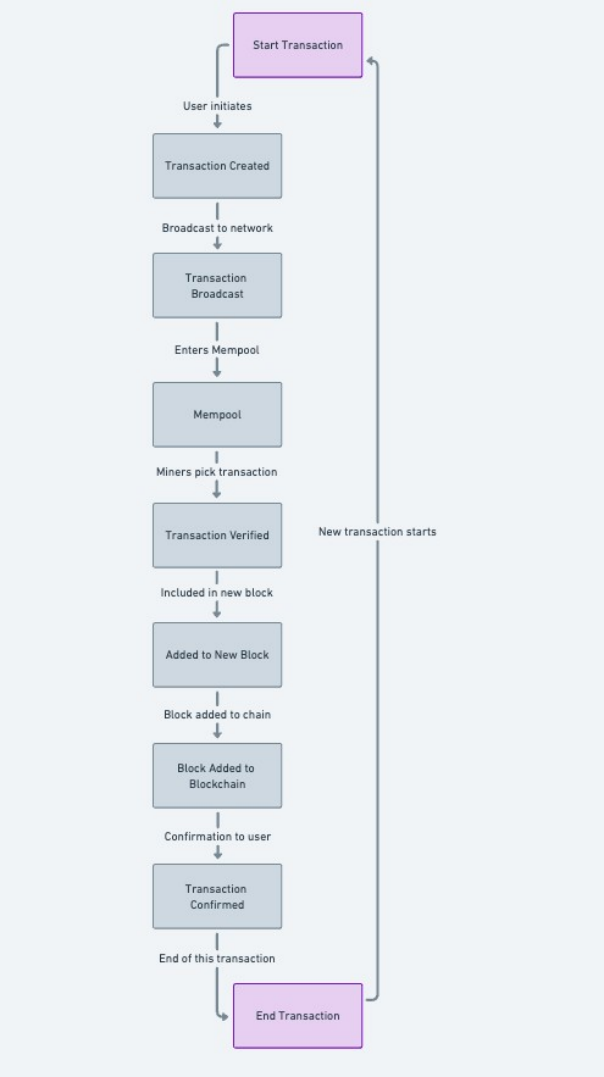
**Μετάδοση**: Μόλις υπογραφεί, η συναλλαγή μεταδίδεται στο δίκτυο και γίνεται μέρος μιας δεξαμενής μη επιβεβαιωμένων συναλλαγών, που συχνά αναφέρεται ως mempool (δεξαμενή μνήμης).

**Επικύρωση**: Πριν προστεθούν σε ένα μπλοκ, οι συναλλαγές στο mempool επικυρώνονται από συμμετέχοντες στο δίκτυο (κόμβους ή εξορύκτες). Η επικύρωση περιλαμβάνει τον έλεγχο της υπογραφής της συναλλαγής, τη διασφάλιση ότι ο αποστολέας έχει επαρκές υπόλοιπο για την ολοκλήρωση της συναλλαγής και η επιβεβαίωση ότι η συναλλαγή δεν έχει ήδη συμπεριληφθεί στο blockchain.

**Συμπερίληψη μπλοκ:** Μετά την επικύρωση, οι συναλλαγές επιλέγονται από το mempool και περιλαμβάνονται σε ένα νέο μπλοκ. Τα κριτήρια επιλογής μπορεί να εξαρτώνται από διάφορους παράγοντες, όπως τα τέλη συναλλαγής (οι υψηλότερες χρεώσεις μπορούν να δώσουν προτεραιότητα στη συμπερίληψη) και τη συμφόρηση δικτύου.

**Εξόρυξη/Διαδικασία συναίνεσης**: Σε δίκτυα blockchain που χρησιμοποιούν Απόδειξη Εργασίας (PoW), το νέο μπλοκ που περιέχει τη συναλλαγή εξορύσσεται. Οι miners ανταγωνίζονται για να λύσουν ένα κρυπτογραφικό παζλ και ο πρώτος που θα το λύσει προσθέτει το μπλοκ στο blockchain. Στο Proof of Stake (PoS) και σε άλλους μηχανισμούς συναίνεσης, η διαδικασία διαφέρει αλλά εξυπηρετεί τον ίδιο σκοπό επικύρωσης και προσθήκης μπλοκ στην αλυσίδα.

**Επιβεβαίωση**: Μόλις προστεθεί επιτυχώς ένα μπλοκ στο blockchain, η συναλλαγή θεωρείται επιβεβαιωμένη. Πρόσθετες επιβεβαιώσεις προκύπτουν καθώς προστίθενται επόμενα μπλοκ, διασφαλίζοντας περαιτέρω τη μονιμότητα της συναλλαγής.




Εικόνα 2.3: Ο κύκλος μιας συναλλαγής στο δίκτυο του Bitcoin

**Μηχανισμοί Συναίνεσης**

Ο μηχανισμός συναίνεσης είναι η μέθοδος με την οποία τα δίκτυα blockchain επιτυγχάνουν αξιοπιστία και ασφαλή συμφωνία σχετικά με την κατάσταση του καθολικού. Διασφαλίζει ότι κάθε νέο μπλοκ που προστίθεται στο blockchain είναι η μοναδική και μοναδική εκδοχή της αλήθειας που συμφωνείται από όλους τους κόμβους του δικτύου. Δύο κύριοι τύποι μηχανισμών συναίνεσης είναι το Proof of Work (PoW) και το Proof of Stake (PoS).

**Απόδειξη Εργασίας (PoW):** Αυτός ο μηχανισμός περιλαμβάνει την επίλυση πολύπλοκων μαθηματικών παζλ, που απαιτεί υπολογιστική ισχύ. Η διαδικασία, γνωστή ως εξόρυξη, επικυρώνει τις συναλλαγές και προσθέτει νέα μπλοκ στην αλυσίδα. Ο πρώτος εξορύκτης που θα λύσει το παζλ έχει το δικαίωμα να προσθέσει το μπλοκ και ανταμείβεται με το εγγενές κρυπτονόμισμα του δικτύου. Το Bitcoin είναι το πιο γνωστό παράδειγμα χρήσης PoW [13] .

Η διαδικασία του mining στο POW είναι η παρακάτω:

- **Συλλογή συναλλαγών**: Οι miners συγκεντρώνουν συναλλαγές από το mempool για να σχηματίσουν ένα νέο μπλοκ.

- **Δημιουργία κεφαλίδας μπλοκ:** Περιλαμβάνει τον κατακερματισμό του προηγούμενου μπλοκ, μια χρονική σήμανση, την έκδοση του πρωτοκόλλου, τη δυσκολία στόχου και ένα nonce.

- **Υπολογισμός κατακερματισμού:** Οι miners υπολογίζουν τον κατακερματισμό της κεφαλίδας του μπλοκ. Στο Bitcoin, αυτό γίνεται χρησιμοποιώντας την κρυπτογραφική συνάρτηση κατακερματισμού SHA-256. Ο στόχος είναι να βρεθεί ένας κατακερματισμός που να είναι χαμηλότερος από έναν αριθμό-στόχο που ορίζεται από το επίπεδο δυσκολίας του δικτύου.

  Λειτουργία κατακερματισμού: Hash = SHA-256 (Κεφαλίδα μπλοκ)
  Συνθήκη για έγκυρο μπλοκ: Hash < Δυσκολία στόχου

- **Nonce Iteration:** Ο nonce, ένας αυθαίρετος αριθμός που μπορεί να αλλάξει, χρησιμοποιείται από τους miners για επανάληψη σε πιθανές λύσεις. Ο miner αλλάζει την τιμή nonce και υπολογίζει εκ νέου τον κατακερματισμό μέχρι να βρει έναν έγκυρο κατακερματισμό που πληροί τα κριτήρια δυσκολίας του δικτύου.

- **Προσθήκη αποκλεισμού και ανταμοιβή**: Μόλις βρεθεί ένας έγκυρος κατακερματισμός, ο miner μεταδίδει το νέο μπλοκ στο δίκτυο. Στη συνέχεια, άλλοι συμμετέχοντες επαληθεύουν το μπλοκ. Εάν είναι έγκυρο, προστίθεται στο blockchain και ο miner λαμβάνει μια ανταμοιβή, συνήθως στο εγγενές κρυπτονόμισμα του blockchain.

**Proof of Stake (PoS)**: Το PoS είναι μια πιο ενεργειακά αποδοτική εναλλακτική του PoW. Στο PoS, οι επικυρωτές επιλέγονται για τη δημιουργία νέων μπλοκ με βάση τον αριθμό των νομισμάτων που κατέχουν και είναι πρόθυμοι να «ποντάρουν» ως εγγύηση. Όσο περισσότερα νομίσματα ποντάρίζονται, τόσο μεγαλύτερες είναι οι πιθανότητες να επιλεγεί ως επικυρωτής. Το Ethereum πρόσφατα άλλαξε από POW σε POS [15].

Η Διαδικασία του Staking: [14]

- **Επιλογή Επικυρωτών**: Οι validators επιλέγονται για να προτείνουν νέα μπλοκ βάσει του μεριδίου τους στο δίκτυο. Όσο μεγαλύτερο το stake τους, τόσο μεγαλύτερες είναι οι πιθανότητές τους να επιλεγούν. Αυτό συχνά εφαρμόζεται ως διαδικασία τυχαίας επιλογής, επηρεασμένη από το μέγεθος του πονταρίσματος.

- **Δημιουργία μπλοκ**: Το επιλεγμένο εργαλείο επικύρωσης κατασκευάζει ένα μπλοκ συναλλαγών και το προτείνει στο δίκτυο.

- **Επικύρωση από Peers:** Άλλοι validators επαληθεύουν το προτεινόμενο μπλοκ. Εάν κριθεί έγκυρο, προστίθεται στο blockchain.

- **Ανταμοιβές και ποινές**: Για τις προσπάθειές τους, οι validators λαμβάνουν αμοιβές συναλλαγών ως ανταμοιβές. Το PoS περιλαμβάνει επίσης μηχανισμούς για την τιμωρία των ανέντιμων επικυρωτών, όπως η περικοπή ενός μέρους του μεριδίου τους για δόλιες δραστηριότητες.

- **Υπολογισμός ανταμοιβής**: Συνήθως είναι συνάρτηση των προμηθειών συναλλαγών και, σε ορισμένες περιπτώσεις, πρόσθετων κινήτρων δικτύου.

- **Μηχανισμός τιμωρίας**: Εάν ένας επικυρωτής προτείνει ένα δόλιο μπλοκ, ένα μέρος του μεριδίου του μπορεί να αφαιρεθεί.

**Σύγκριση PoW και PoS**

**Κατανάλωση ενέργειας**: Το PoW απαιτεί σημαντική υπολογιστική ισχύ και ενέργεια, ενώ το PoS είναι πιο ενεργειακά αποδοτικό καθώς δεν περιλαμβάνει περίπλοκη επίλυση των πάζλ.

**Ασφάλεια**: Ενώ το PoW έχει αποδεδειγμένο ιστορικό όσον αφορά την ασφάλεια, το PoS συχνά θεωρείται ότι έχει δυνητικά υψηλότερους κινδύνους συγκεντροποίησης (centralization), αλλά προσφέρει καινοτόμους μηχανισμούς για την αντιμετώπιση της ανέντιμης συμπεριφοράς.

**Προσβασιμότητα**: Το PoS επιτρέπει ευρύτερη συμμετοχή, καθώς δεν απαιτεί σημαντικές επενδύσεις υλικού όπως το PoW.

## 2.2 Λειτουργία Αποκεντρωμένων Ανταλλακτηρίων

Τα αποκεντρωμένα ανταλλακτήρια, όπως ήδη έχουμε συζητήσει στην Εισαγωγή, είναι η σημαντικότερη εφαρμογή σε μια πλατφόρμα η οποία υποστηρίζει smart contract είτε είναι layer1 (πχ ethereum, solana) είτε είναι layer 2 (πχ arbitrum, optimism). Λειτουργούν με βάση την αρχή ότι οι

συναλλαγές πρέπει να πραγματοποιούνται απευθείας μεταξύ των χρηστών (peer-to-peer) μέσω μιας αυτοματοποιημένης διαδικασίας. Αυτό διευκολύνεται από τα smart contracts ta opo;ia τρέχουν σε δίκτυα blockchain όπως το Ethereum και είναι υπεύθυνα για την αντιστοίχιση εντολών συναλλαγών, την εκτέλεση συναλλαγών και τον ασφαλή χειρισμό της μεταφοράς περιουσιακών στοιχείων.

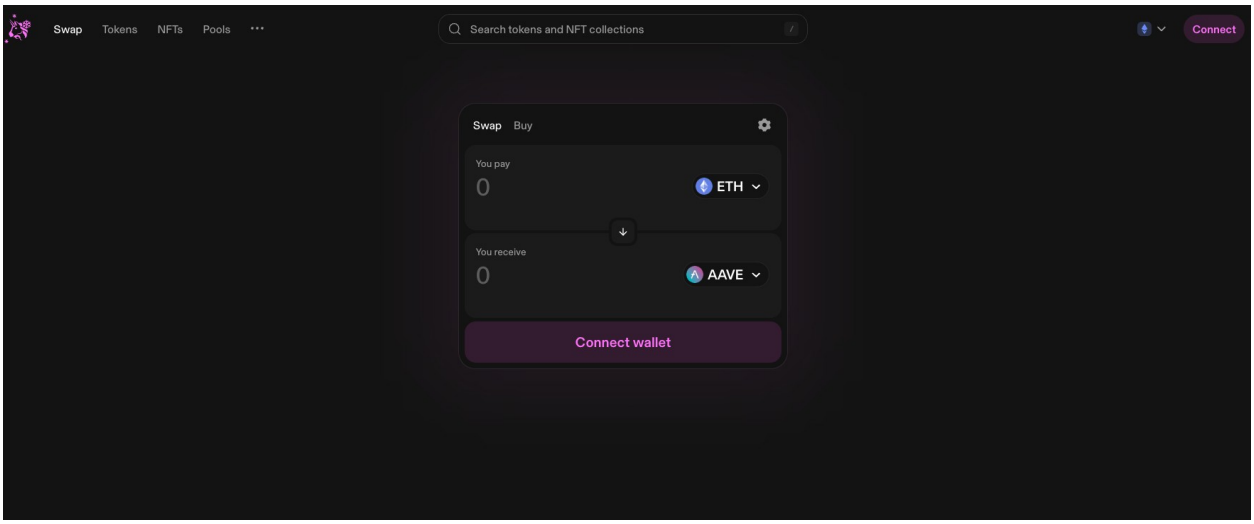
**Βασικά χαρακτηριστικά των DEX**

**Peer-to-Peer Trading:** Οι traders συναλλάσσονται απευθείας μεταξύ τους χωρίς να χρειάζεται μια κεντρική αρχή για τη διευκόλυνση των συναλλαγών.

**Custody κεφαλαίων:** Οι χρήστες διατηρούν τον έλεγχο των κεφαλαίων τους. Σε αντίθεση με τα κεντρικά ανταλλακτήρια, όπου το ανταλλακτήριο ελέγχει τα κεφάλαια, στα DEX, τα χρήματα διατηρούνται στα πορτοφόλια των χρηστών. Σαν αντίστοιχεια με τον offline κόσμο, μπορούμε να πούμε ότι οι χρήστες συναλλάσσονται με μετρητά τα οποία κρατούν στα χέρια τους και όχι με χρήματα τα οποία θεωρητικά έχουν σε μία τράπεζα.

**Απόρρητο και ανωνυμία:** Τα περισσότερα DEX δεν απαιτούν από τους χρήστες να υποβάλλονται σε διαδικασίες Know Your Customer (KYC), προσφέροντας υψηλότερο βαθμό απορρήτου.

**Μειωμένος κίνδυνος hacks και απάτης:** Δεδομένου ότι δεν υπάρχει κεντρικό σημείο αποτυχίας (όπως ένας κεντρικός διακομιστής) και οι χρήστες διατηρούν τα ιδιωτικά τους κλειδιά, ο κίνδυνος μεγάλης κλίμακας hacks μειώνεται σημαντικά.




Εικόνα 2.4: Screenshot από το uniswap, το πιο γνωστό DEX στο Ethereum

Υπάρχουν δύο τύποι ανταλλακτηρίων. Αυτά που ακολουθούν το Order Book Model και αυτά που ακολουθύν το Automated Market Maker (AMM) Model. Τα DEX ακολουθούν συνήθως το AMM και τα CEX το Order Book χωρίς αυτό να είναι απόλυτο. Ας δούμε τις διαφορές τους.

**Μοντέλο Automated Market Maker (AMM).**

Τα AMM DEX λειτουγούν με τη χρήση Liquidity Pool, όπου εισάγεται ρευστότητα για τα pairs που θέλουμε να γίνουν trade από πριν.

- **Απλούστερη λογική:** Τα AMM χρησιμοποιούν μαθηματικούς τύπους (όπως ο διάσημος τύπος x * y = k του Uniswap) για να καθορίσουν τις τιμές και να εκτελέσουν συναλλαγές. Αυτοί οι τύποι είναι σχετικά απλοί στην εφαρμογή σε έξυπνα συμβόλαια.

- **Διαχείριση κατάστασης:** Τα AMM δεν απαιτούν τη διατήρηση μιας περίπλοκης κατάστασης βιβλίου παραγγελιών. Απλά πρέπει να παρακολουθούμε την κατάσταση των liquidity pools.

- **Λιγότερες συναλλαγές**: Σε ένα AMM, οι συναλλαγές δεν απαιτούν αντισυμβαλλόμενο, γεγονός που απλοποιεί τον χειρισμό των συναλλαγών. Ουσιαστικά αλληλεπιδράμε απευθείας με το έξυπνο συμβόλαιο που διαχειρίζεται το pool.

- **Δημοφιλή frameworks και παραδείγματα**: Υπάρχουν πολλά παραδείγματα και frameworks διαθέσιμα για τη δημιουργία AMM, δεδομένης της δημοτικότητας πλατφορμών όπως το Uniswap. Αυτή η διαθεσιμότητα πόρων διευκολύνει την ανάπτυξη.

Ο τύπος x * y = k είναι μια θεμελιώδης ιδέα πίσω από πολλά μοντέλα Automated Market Maker (AMM) στην αποκεντρωμένη χρηματοδότηση (DeFi), ιδιαίτερα δημοφιλή από το Uniswap. Αυτός ο τύπος βοηθά στον προσδιορισμό της τιμής των περιουσιακών στοιχείων σε μια ομάδα ρευστότητας και διασφαλίζει ότι η ομάδα παραμένει ισορροπημένη μετά τις συναλλαγές. Ας το αναλύσουμε:

x και y: Αυτά αντιπροσωπεύουν τις ποσότητες δύο διαφορετικών tokens σε ένα liquidity pool. Για παράδειγμα, σε μια ομάδα για ένα ζεύγος tokens ETH/DAI, x θα μπορούσε να είναι το ποσό του ETH, και το y θα μπορούσε να είναι το ποσό του DAI στο pool.

k: Αυτή είναι μια σταθερή τιμή, που σημαίνει ότι το γινόμενο των x και y πρέπει να είναι πάντα το ίδιο, ακόμη και όταν αλλάζουν τα x και y.

**Πως δουλεύει**

**Αρχική Πρόβλεψη Ρευστότητας**: Όταν η ρευστότητα προστίθεται για πρώτη φορά στο pool, τα x και y ορίζονται και το k υπολογίζεται ως το γινόμενο τους (k = x * y). Αυτό καθορίζει την αρχική τιμή των tokens σε σχέση μεταξύ τους με βάση τα αρχικά τους ποσά.

**Διαπραγμάτευση και προσαρμογή τιμής**: Όταν κάποιος θέλει να ανταλλάξει ένα token με ένα άλλο στην pool, θα προσθέσει σε μία από τις ποσότητες κουπόνι και θα αφαιρέσει από την άλλη. Ο τύπος x * y = k διασφαλίζει ότι η συνολική αξία της ομάδας παραμένει σταθερή, αλλά οι σχετικές τιμές των κουπονιών αλλάζουν.

Για παράδειγμα, αν κάποιος αγοράσει ETH από το pair ETH/DAI (οπότε αφαιρείται το ETH και προσθέτει DAI), το ποσό του ETH (x) μειώνεται και το ποσό του DAI (y) αυξάνεται. Δεδομένου ότι το k πρέπει να παραμείνει σταθερό, η τιμή του ETH σε όρους DAI αυξάνεται.

Χωρίς βιβλίο παραγγελιών: Σε αντίθεση με τις παραδοσιακές ανταλλαγές, δεν υπάρχει βιβλίο παραγγελιών ή αντιστοίχιση εντολών αγοράς/πώλησης. Οι τιμές καθορίζονται καθαρά από την αναλογία των κουπονιών στο pool και οι συναλλαγές εκτελούνται έναντι του ίδιου του pool.

**Επιπτώσεις**

**Slippage**: Οι μεγαλύτερες συναλλαγές έχουν πιο σημαντικό αντίκτυπο στα σχετικά ποσά των x και y, οδηγώντας σε πιο ουσιαστικές αλλαγές στην τιμή (slippage). Αυτό το αποτέλεσμα σημαίνει ότι το slippage είναι εγγενής στα AMM με βάση το μοντέλο x * y = k.

**Impermanent Loss**: Όταν η τιμή των tokens αλλάζει σημαντικά, οι πάροχοι ρευστότητας μπορεί να αντιμετωπίσουν «Impermanent Loss», η οποία είναι η διαφορά στην αξία μεταξύ των tokens που κρατάει κάποιος σε σχέση με αυτά που είναι μέσα στο pool.

**Αυτο-εξισορρόπηση**: Το μοντέλο είναι αυτο-ισορροπούμενο. Καθώς πραγματοποιούνται συναλλαγές, οι τιμές προσαρμόζονται με τρόπο που ενθαρρύνει τους traders να επαναφέρουν το pool σε μια πιο ισορροπημένη κατάσταση.

**Ας δούμε ένα παράδειγμα για να δείξουμε πώς αλλάζει η τιμή όταν κάποιος κάνει μια ανταλλαγή σε έναν Automated Market Maker (AMM) όπως το Uniswap, χρησιμοποιώντας τον τύπο x * y = k.**

**Αρχική κατάσταση**

Ας υποθέσουμε ότι έχουμε ένα pool για δύο tokens: ETH και DAI. Ας ξεκινήσουμε με τα ακόλουθα ποσά στο pool:

x = 100 ETH
y = 20.000 DAI
Η σταθερά k υπολογίζεται ως x * y, άρα:

k = 100 ETH * 20.000 DAI = 2.000.000 ETH*DAI

**Παράδειγμα ανταλλαγής**

Ας υποθέσουμε ότι ένας trader θέλει να αγοράσει 10 ETH από αυτό το pool. Ο trader θα προσθέσει DAI στο pool και θα αφαιρέσει το ETH από αυτό. Πρέπει να υπολογίσουμε πόση DAI απαιτείται.

Η τιμή k (2.000.000 ETH*DAI) πρέπει να παραμείνει σταθερή. Μετά τη διαπραγμάτευση, το ETH στο pool (x) θα είναι 90 ETH (αφού αφαιρεθούν 10 ETH). Πρέπει να βρούμε το νέο y (ποσό DAI) που διατηρεί τη σταθερά k.

Χρησιμοποιώντας τον τύπο k = x * y, τον αναδιατάσσουμε για να λύσουμε για το y:

y = k / x
y = 2.000.000 ETH*DAI / 90 ETH ≈ 22.222,22 DAI

**Υπολογισμός του DAI που απαιτείται για την ανταλλαγή**

Αρχικά, υπήρχαν 20.000 DAI στο pool. Μετά την ανταλλαγή, πρέπει να υπάρχουν περίπου 22.222,22 DAI. Έτσι, ο trader πρέπει να προσθέσει:

22.222,22 DAI - 20.000 DAI = 2.222,22 DAI

**Αποτέλεσμα της ανταλλαγής**

Ο trader προσθέτει 2.222,22 DAI στο pool και αφαιρεί 10 ETH. Η νέα κατάσταση του pool είναι:

x = 90 ETH
y ≈ 22.222,22 DAI

**Επίπτωση στην τιμή**

Παρατηρήστε πώς έχει αλλάξει η τιμή του ETH σε όρους DAI λόγω της ανταλλαγής:

Αρχική τιμή: 1 ETH = 200 DAI (20.000 DAI / 100 ETH)
Νέα τιμή: 1 ETH ≈ 246,91 DAI (22.222,22 DAI / 90 ETH)

Αυτή η αλλαγή στην τιμή είναι ένα παράδειγμα "slippage". Οι μεγαλύτερες συναλλαγές (σε σχέση με το μέγεθος) έχουν ως αποτέλεσμα πιο σημαντικό slippage.

Το μοντέλο AMM προσαρμόζει αυτόματα την τιμή με βάση τη δυναμική προσφοράς και ζήτησης εντός του pool. Το βασικό στοιχείο είναι ότι το σχετικό μέγεθος της συναλλαγής προς το μέγεθος του pool επηρεάζει σημαντικά την κίνηση της τιμής – όσο μεγαλύτερη είναι η συναλλαγή σε αναλογία με τη συγκέντρωση, τόσο πιο σημαντική είναι η μετατόπιση της τιμής.

**Μοντέλο παραδοσιακού βιβλίου παραγγελιών**

- **Σύνθετη λογική αντιστοίχισης:** Η εφαρμογή ενός βιβλίου παραγγελιών απαιτεί εξελιγμένη λογική για τη διαχείριση της αντιστοίχισης παραγγελιών, την καταχώρηση παραγγελιών, την

ακύρωση παραγγελιών και τις ενημερώσεις. Αυτό είναι πιο περίπλοκο από τους μαθηματικούς τύπους που χρησιμοποιούνται στα AMM.

- **Κατάσταση και Ένταση δεδομένων:** Τα βιβλία παραγγελιών πρέπει να διαχειρίζονται πολύ περισσότερη κατάσταση – κάθε παραγγελία πρέπει να παρακολουθείται και το βιβλίο πρέπει να ενημερώνεται με κάθε νέα παραγγελία, συναλλαγή ή ακύρωση.
- **Χειρισμός race conditions**: Σε ένα πλαίσιο blockchain, ο χειρισμός των race conditions (π.χ. δύο χρήστες που προσπαθούν να συμπληρώσουν την ίδια παραγγελία ταυτόχρονα) γίνεται δύσκολος.
- **Ανησυχίες επεκτασιμότητας:** Τα βιβλία παραγγελιών on-chain μπορεί να αντιμετωπίσουν προβλήματα επεκτασιμότητας λόγω του μεγάλου όγκου συναλλαγών και δεδομένων, που μπορεί να είναι δαπανηρά σε δίκτυα όπως το Ethereum.

Ας εξετάσουμε ένα απλοποιημένο παράδειγμα για να κατανοήσουμε πώς λειτουργούν οι παραδοσιακές ανταλλαγές βιβλίων παραγγελιών. Φανταστείτε μια ανταλλαγή όπου οι άνθρωποι μπορούν να ανταλλάσσουν μήλα με πορτοκάλια.

Trader : Υπάρχουν άνθρωποι που θέλουν να αγοράσουν πορτοκάλια χρησιμοποιώντας μήλα και το αντίστροφο.

Exchange: Παρέχει μια πλατφόρμα όπου αυτοί οι έμποροι μπορούν να αναφέρουν τις προσφορές τους.

Το Βιβλίο Παραγγελιών

Το βασικό συστατικό αυτής της ανταλλαγής είναι το βιβλίο παραγγελιών. Απαριθμεί όλες τις παραγγελίες αγοράς και πώλησης για τα δύο φρούτα.

**Παραγγελίες αγορά**ς: Ας υποθέσουμε ότι η Αλίκη θέλει να αγοράσει 10 πορτοκάλια και είναι πρόθυμη να ανταλλάξει 5 μήλα για αυτά. Η προσφορά της είναι μια «εντολή αγοράς» για πορτοκάλια.

**Παραγγελίες πώλησης:** Ο Μπομπ έχει πορτοκάλια και είναι πρόθυμος να πουλήσει 10 πορτοκάλια για 6 μήλα. Η προσφορά του είναι μια «παραγγελία πώλησης» για πορτοκάλια.
Το βιβλίο παραγγελιών σε αυτήν την ανταλλαγή μπορεί να μοιάζει κάπως έτσι:

| Sell Orders | Buy Orders |
|---|---|
| 10 oranges for 6 apples (Bob) | 10 oranges for 5 apples (Alice) |
| 20 oranges for 12 apples | 15 oranges for 7 apples |

Η δουλειά του ανταλλακτηρίου είναι να αντιστοιχίζει τις εντολές αγοράς με τις εντολές πώλησης. Όταν μια εντολή αγοράς ανταποκρίνεται σε μια εντολή πώλησης με αντίστοιχους ή αποδεκτούς όρους, πραγματοποιείται μια συναλλαγή.

Εάν ο Τσάρλι έρθει στο ανταλλακτήριο και είναι πρόθυμος να πουλήσει 10 πορτοκάλια για 5 μήλα (που ταιριάζουν με την παραγγελία αγοράς της Αλίκης), η ανταλλαγή θα ταιριάζει με τις παραγγελίες της Αλίκης και του Τσάρλι.

Η ανταλλαγή στη συνέχεια διευκολύνει τη μεταφορά 5 μήλων από την Αλίκη στον Τσάρλι και σε αντάλλαγμα, ο Τσάρλι δίνει 10 πορτοκάλια στην Αλίκη.

Ενημέρωση του βιβλίου παραγγελιών: Μετά από αυτή τη συναλλαγή, το βιβλίο παραγγελιών ενημερώνεται. Η εντολή αγοράς της Alice καταργείται καθώς έχει εκπληρωθεί.

Οι τιμές σε ένα βιβλίο παραγγελιών καθορίζονται από το τι είναι διατεθειμένοι να πληρώσουν ή να δεχτούν οι αγοραστές και οι πωλητές. Εάν περισσότεροι άνθρωποι θέλουν να αγοράσουν πορτοκάλια παρά να τα πουλήσουν, η «τιμή» (σε μήλα) μπορεί να αυξηθεί καθώς οι αγοραστές ανταγωνίζονται για να εξασφαλίσουν ένα εμπόριο.

Σε ένα πραγματικό σενάριο, οι δυνάμεις της αγοράς της προσφοράς και της ζήτησης διαμορφώνουν συνεχώς τις τιμές και το βιβλίο παραγγελιών ενημερώνεται δυναμικά καθώς έρχονται νέες παραγγελίες και εκπληρώνονται ή αποσύρονται οι παλιές παραγγελίες.

Ένα παράδειγμα ενός order book φαίνεται στη παρακάτω εικόνα:

Εικόνα 2.5: Παράδειγμα από ένα πραγματικό order book.

## 2.3 Τα DEX σήμερα

Το τοπίο των Αποκεντρωμένων Χρηματιστηρίων (DEX) εξελίσσεται ταχέως, αντανακλώντας τη δυναμική και καινοτόμο φύση των τομέων blockchain και κρυπτονομισμάτων. Από τα αρχικά τους στάδια, όπου τα DEX ήταν περισσότερο μια εννοιολογική καινοτομία, έχουν εξελιχθεί σε εξελιγμένες πλατφόρμες που προσφέρουν μια σειρά από υπηρεσίες και χαρακτηριστικά που ανταγωνίζονται, και σε ορισμένες πτυχές, ξεπερνούν τις παραδοσιακές χρηματοοικονομικές ανταλλαγές.

Ένας από τους σημαντικούς μοχλούς αυτής της εξέλιξης ήταν η πρόοδος στην ίδια την τεχνολογία blockchain, ιδιαίτερα η ανάπτυξη και υιοθέτηση έξυπνων συμβολαίων. Πλατφόρμες όπως το Ethereum έχουν προσφέρει το τέλειο οικοσύστημα για την ανάπτυξη των DEX, με τις δυνατότητες έξυπνων συμβολαίων τους που επιτρέπουν την αυτοματοποιημένη, ασφαλή και διαφανή επεξεργασία συναλλαγών. Αυτό οδήγησε στη δημιουργία πιο φιλικών προς τον χρήστη διεπαφών, μειώνοντας τα εμπόδια εισόδου για τους μέσους χρήστες που προηγουμένως φοβόντουσαν από την πολυπλοκότητα των συναλλαγών blockchain.

Τα σημερινά DEX χαρακτηρίζονται από το ευρύ φάσμα των υπηρεσιών τους. Προσφέρουν περισσότερα από απλά swaps. Πολλά είναι ενσωματωμένα με άλλες εφαρμογές DeFi, παρέχοντας υπηρεσίες όπως **staking, δανεισμός και δανεισμός**. Αυτή η ενοποίηση οδήγησε σε ένα πιο διασυνδεδεμένο και πολυλειτουργικό οικοσύστημα DeFi, όπου οι χρήστες μπορούν να μετακινούν απρόσκοπτα περιουσιακά στοιχεία σε διαφορετικές πλατφόρμες, μεγιστοποιώντας τις αποδόσεις τους και διαχειρίζονται τους κινδύνους πιο αποτελεσματικά.

Επιπλέον, με τις συνεχείς εξελίξεις στις λύσεις επεκτασιμότητας blockchain, όπως τα layer 2 protocols και sidechains, τα DEX είναι καλύτερα εξοπλισμένα για να χειρίζονται υψηλότερους όγκους συναλλαγών με χαμηλότερες χρεώσεις, καθιστώντας τα πιο ανταγωνιστικά έναντι των κεντρικών ανταλλαγών.

Η ασφάλεια παραμένει πρωταρχική εστίαση, με συνεχείς προσπάθειες για την ενίσχυση της ασφάλειας αυτών των πλατφορμών. Ενώ τα DEX μειώνουν εγγενώς ορισμένους κινδύνους, όπως εκείνους που σχετίζονται με κεντρικά σημεία αποτυχίας και τη φύλαξη περιουσιακών στοιχείων, δεν είναι απρόσβλητοι από προκλήσεις. Οι ευπάθειες των έξυπνων συμβολαίων και οι εκ των προτέρων τρέχουσες είναι ανησυχίες για τις οποίες τόσο οι προγραμματιστές όσο και οι χρήστες πρέπει να προσέχουν. Κατά συνέπεια, δίνεται συνεχής έμφαση στους ελέγχους ασφαλείας, στα προγράμματα επιβράβευσης σφαλμάτων και στην ανάπτυξη πιο ασφαλών πρακτικών κωδικοποίησης εντός του χώρου DEX.

Ακολουθεί μια λίστα με μερικά από τα πιο αξιοσημείωτα DEX για κάθε σημαντικό blockchain:

**Ethereum**

Uniswap: Ένα από τα πρώτα και πιο σημαντικά DEX στο οικοσύστημα Ethereum, το Uniswap είναι γνωστό για το μοντέλο Automated Market Maker (AMM), την απλότητα και την υψηλή ρευστότητά του.

**Binance Smart Chain (BSC)**

PancakeSwap: Ένα κορυφαίο DEX στο BSC, το PancakeSwap προσφέρει παρόμοια λειτουργικότητα με το Uniswap, αλλά λειτουργεί εντός της Binance Smart Chain, επωφελούμενος από χαμηλότερες χρεώσεις συναλλαγών και μεγαλύτερες ταχύτητες.

**Solana**

Raydium: Το Raydium είναι ένας αυτοματοποιημένος κατασκευαστής αγορών (AMM) που βασίζεται στο blockchain Solana. Έχει κερδίσει εξέχουσα θέση λόγω της ενσωμάτωσής του με το Serum DEX, το κεντρικό βιβλίο παραγγελιών ορίων της Solana. Αυτός ο μοναδικός συνδυασμός επιτρέπει στο Raydium να προσφέρει γρήγορες και αποτελεσματικές εμπειρίες συναλλαγών, αξιοποιώντας την υψηλή απόδοση και το χαμηλό κόστος συναλλαγών της Solana.

**Polkadot**

PolkaSwap: Ως μέρος του δικτύου SORA, το PolkaSwap λειτουργεί εντός του οικοσυστήματος Polkadot. Έχει σχεδιαστεί για να διευκολύνει τις ομάδες και τις ανταλλαγές διακριτικών διασταυρούμενης αλυσίδας, αξιοποιώντας τις δυνατότητες διαλειτουργικότητας του Polkadot.

**Avalanche**

Trader Joe: Ο Trader Joe συνδυάζει υπηρεσίες DEX με αποκεντρωμένο δανεισμό και δανεισμό, καθιστώντας το μια πολύπλευρη πλατφόρμα εντός του δικτύου Avalanche.

**Cardano**

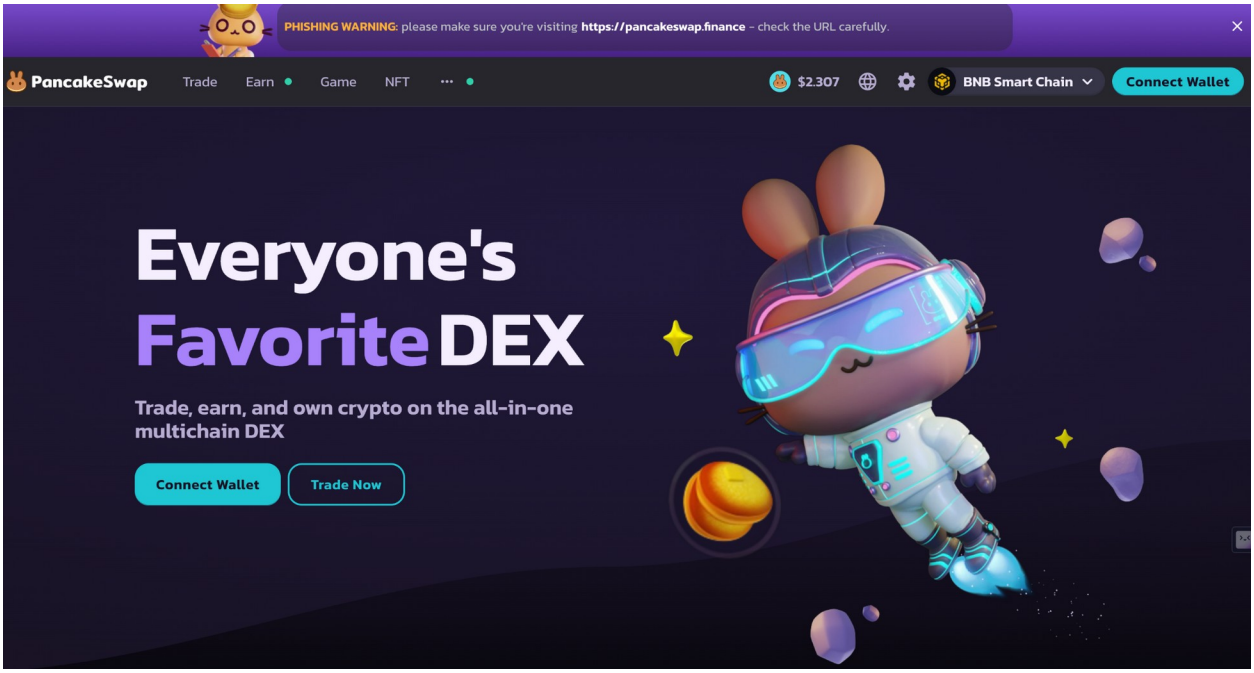
SundaeSwap: Ως εγγενές, επεκτάσιμο DEX στο Cardano, το SundaeSwap έχει σχεδιαστεί για να αξιοποιεί τις χαμηλές χρεώσεις συναλλαγών και την υψηλή χωρητικότητα συναλλαγών του Cardano.

**Algorand**

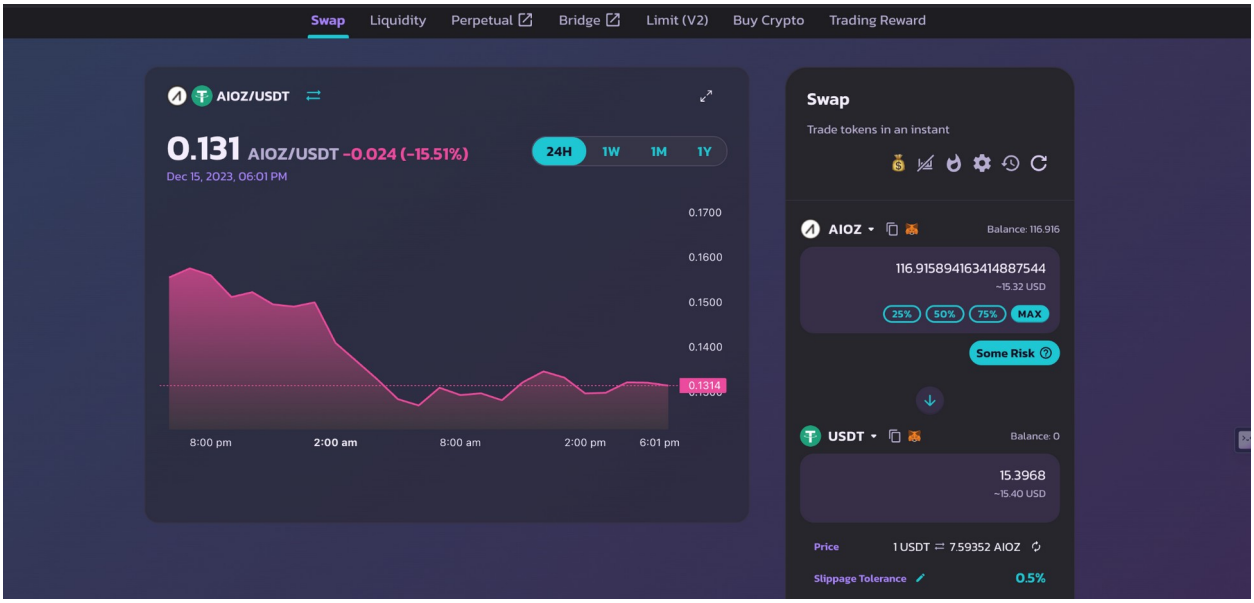
Tinyman: Το Tinyman είναι ένα DEX που βασίζεται σε AMM στο Algorand, που προσφέρει γρήγορες και χαμηλού κόστους συναλλαγές, παροχή ρευστότητας και άλλες υπηρεσίες DeFi εντός του οικοσυστήματος Algorand.

**Tezos**

QuipuSwap: Ένα DEX ανοιχτού κώδικα στο Tezos, το QuipuSwap προσφέρει ένα μοντέλο AMM, που επιτρέπει στους χρήστες να παρέχουν ρευστότητα και να συμμετέχουν σε αποκεντρωμένες ανταλλαγές διακριτικών.




Εικόνα 2.6: Το landing page του Pancakeswap




Εικόνα 2.7: Το Swap στο Pancakeswap

## 2.4 Προβλήματα Ασφαλείας

Η αποκεντρωμένη φύση των DEX, ενώ παρέχει πολυάριθμα οφέλη, όπως η αυτονομία του χρήστη και ο μειωμένος κίνδυνος κεντρικού ελέγχου, εισάγει επίσης μοναδικές προκλήσεις ασφάλειας. Η κατανόηση αυτών των ανησυχιών είναι ζωτικής σημασίας για τη συνεχή ανάπτυξη και βελτίωση των πλατφορμών DEX.

**Κοινά τρωτά σημεία σε DEX**

Τα DEX, βασικά εξαρτημένα από τα έξυπνα συμβόλαια και την τεχνολογία blockchain, αντιμετωπίζουν αρκετές ευπάθειες ασφαλείας όπως αναφέραμε και νωρίτερα.:

**Σφάλματα έξυπνων συμβολαίων**: Δεδομένου ότι τα DEX εξαρτώνται σε μεγάλο βαθμό από έξυπνα συμβόλαια, οποιοδήποτε ελάττωμα ή σφάλμα στον κώδικα συμβολαίου μπορεί να οδηγήσει σε σημαντικές ευπάθειες. Αυτά μπορεί να κυμαίνονται από μικρές δυσλειτουργίες έως μεγάλα κενά που μπορεί να επιτρέψουν στους επιτιθέμενους να αποστραγγίσουν χρήματα από το χρηματιστήριο.

**Front-Running**: Στο blockchain, οι συναλλαγές είναι δημόσια ορατές στο mempool πριν επιβεβαιωθούν. Αυτή η διαφάνεια μπορεί να οδηγήσει σε Front-Running, όπου ένας εισβολέας βλέπει μια συναλλαγή σε εκκρεμότητα και πληρώνει υψηλότερη χρέωση συναλλαγής για να επιβεβαιωθεί πρώτα η δική του συναλλαγή (η οποία επωφελείται από την εκκρεμή συναλλαγή).

**Impermanent Loss:** Σε DEX που βασίζονται σε AMM, οι πάροχοι ρευστότητας μπορεί να υποφέρουν από Impermanent Loss, το οποία συμβαίνει όταν η τιμή των κατατεθέντων περιουσιακών στοιχείων αλλάζει σε σύγκριση με την περίοδο κατάθεσης. Αυτό δεν είναι κίνδυνος ασφάλειας με την παραδοσιακή έννοια, αλλά μπορεί να οδηγήσει σε οικονομική απώλεια.

**Rug Pulls**: Μια κατάσταση όπου οι προγραμματιστές ενός έργου DEX ή κρυπτογράφησης αποσύρουν ξαφνικά όλα τα κεφάλαιά τους από τη δεξαμενή ρευστότητας, οδηγώντας σε κατάρρευση της τιμής και απώλεια για άλλους παρόχους ρευστότητας και εμπόρους.

**Sybil Attack**s: Οι εισβολείς μπορούν να δημιουργήσουν πολλαπλές πλαστές ταυτότητες για να επηρεάσουν το δίκτυο, ιδιαίτερα σε μικρότερα DEX ή σε αυτά με λιγότερο ισχυρές δομές διακυβέρνησης.

Η ιστορία των DEX έχει δει πολλά αξιοσημείωτα περιστατικά ασφαλείας:

**Η επίθεση DAO**: Ένα από τα πιο διαβόητα περιστατικά στις πρώτες μέρες των έξυπνων συμβολαίων ήταν η επίθεση στο The DAO στο Ethereum, όπου μια ευπάθεια στον κώδικα έξυπνου συμβολαίου του οδήγησε στην κλοπή περίπου του ενός τρίτου των κεφαλαίων του DAO [16].

**Flash Loan Attacks**: Αρκετοί DEX έχουν υποφέρει από επιθέσεις flash loan, όπου οι εισβολείς δανείζονται μεγάλο όγκο περιουσιακών στοιχείων από ένα DEX, χειραγωγούν τις τιμές της αγοράς σε άλλο DEX και στη συνέχεια επωφελούνται από αυτόν τον χειρισμό.

**Πρωτόκολλα Exploits**: Υπήρξαν περιπτώσεις όπου οι εισβολείς εκμεταλλεύτηκαν αδυναμίες στα πρωτόκολλα ορισμένων DEX, οδηγώντας σε απώλεια κεφαλαίων. Αυτές συχνά περιλαμβάνουν πολύπλοκες αλληλεπιδράσεις πολλαπλών έξυπνων συμβάσεων.

**Αντιμετώπιση ανησυχιών για την ασφάλεια**

Η αντιμετώπιση αυτών των προβλημάτων ασφάλειας περιλαμβάνει μια πολύπλευρη προσέγγιση:

**Auditing Smart Contracts** : Ο τακτικός και ενδελεχής έλεγχος του κώδικα έξυπνων συμβολαίων από ανεξάρτητες εταιρείες ασφάλειας είναι ζωτικής σημασίας.

**Βελτιωμένος σχεδιασμός πρωτοκόλλου**: Ανάπτυξη πιο ισχυρών και ασφαλών πρωτοκόλλων, εξέταση πιθανών φορέων επίθεσης και σχεδιασμός διασφαλίσεων εναντίον τους.

**Εκπαίδευση χρηστών**: Εκπαίδευση των χρηστών σχετικά με τους κινδύνους, ειδικά όσον αφορά τις αλληλεπιδράσεις έξυπνων συμβολαίων και τις αποχρώσεις των πλατφορμών AMM.

**Ενεργή επίβλεψη κοινότητας**: Ενθάρρυνση μιας κοινότητας σε εγρήγορση που μπορεί να υψώσει γρήγορα σημαίες για ύποπτες δραστηριότητες.

Καθώς το οικοσύστημα DEX συνεχίζει να εξελίσσεται, το ίδιο συμβαίνει και με τις στρατηγικές για την ασφάλεια αυτών των πλατφορμών. Η συνεχιζόμενη μάχη κατά των τρωτών σημείων και των εκμεταλλεύσεων στα DEX δεν είναι μόνο τεχνική, αλλά περιλαμβάνει επίσης μια προσέγγιση με γνώμονα την κοινότητα για τη διαφύλαξη των περιουσιακών στοιχείων. Η κατανόηση αυτών των ανησυχιών για την ασφάλεια είναι πρωταρχικής σημασίας τόσο για τους χρήστες όσο και για τους προγραμματιστές για την ασφαλή και αποτελεσματική πλοήγηση στο τοπίο του DEX.

# Κεφάλαιο 3: Προγραμματισμός σε Ethereum Virtual Machine (EVM)

Με βάση τις θεμελιώδεις γνώσεις που δημιουργήθηκαν στα προηγούμενα κεφάλαια, η εστίασή μας τώρα μετατοπίζεται σε μια πιο λεπτομερή εξερεύνηση του τρόπου με τον οποίο η τεχνολογία blockchain ενσωματώνεται και λειτουργεί στα DEX. Στο επίκεντρο αυτής της συζήτησης βρίσκονται τα έξυπνα συμβόλαια.

Στο κεφάλαιο αυτό θα δούμε πως λειτουργούν τα smart contracts και ποια είναι η διαδικασία δημιουργίας τους. Θα επικεντρωθούμε στο Ethereum και στη γλώσσα προγραμματισμού Solidity.

## 3.1 Έξυπνα Συμβόλαια

Τα έξυπνα συμβόλαια είναι κομμάτια κώδικα τα οποία αποθηκεύονται στο blockchain.

**Πώς λειτουργούν τα έξυπνα συμβόλαια**

**Ανάπτυξη**: Ένας χρήστης (συνήθως ο δημιουργός συμβολαίου) στέλνει μια συναλλαγή στο blockchain, που περιέχει τον μεταγλωττισμένο κώδικα του έξυπνου συμβολαίου. Αυτή η συναλλαγή, αφού εξορυχθεί, δημιουργεί ένα νέο συμβόλαιο στο blockchain, το οποίο μπορεί να αναγνωριστεί από μια διεύθυνση.

**Εκτέλεση**: Όταν οι χρήστες αλληλεπιδρούν με ένα contract (π.χ. πραγματοποιούν συναλλαγές σε ένα DEX), στέλνουν συναλλαγές στη διεύθυνση του συμβολαίου. Αυτές οι συναλλαγές καθορίζουν ποια λειτουργία του συμβολαίου θα καλεστεί και με ποιες παραμέτρους.

**Αλλαγή κατάστασης**: Το contract εκτελείται όπως έχει προγραμματιστεί, αλλάζοντας την κατάστασή της εάν είναι απαραίτητο. Για παράδειγμα, μια συναλλαγή σε ένα DEX μπορεί να αλλάξει τα υπόλοιπα των διακριτικών που κατέχονται από το συμβόλαιο.

**Transaction Inclusion**: Το αποτέλεσμα της εκτέλεσης καταγράφεται σε ένα μπλοκ, το οποίο στη συνέχεια διαδίδεται σε όλο το blockchain, διασφαλίζοντας ότι όλοι οι κόμβοι ενημερώνουν το αντίγραφό τους για την κατάσταση του contract.

**Ανάπτυξη Έξυπνων Συμβολαίων**

Η ανάπτυξη ενός έξυπνου συμβολαίου περιλαμβάνει συνήθως διάφορα στάδια:

**Σύνταξη του συμβολαίου:** Η πιο δημοφιλής γλώσσα για τη σύνταξη έξυπνων συμβολαίων στο Ethereum είναι η Solidity. Είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού για τη σύνταξη έξυπνων συμβολαίων που εκτελούνται στην εικονική μηχανή Ethereum (EVM).

**Σύνταξη Solidity**: Παρόμοια με την JavaScript και τη C++, το Solidity πληκτρολογείται στατικά και υποστηρίζει κληρονομικότητα, βιβλιοθήκες και σύνθετους τύπους που ορίζονται από τον χρήστη.

**Testing** : Πριν από την ανάπτυξη, οι συμβάσεις πρέπει να ελέγχονται αυστηρά. Αυτό μπορεί να γίνει χρησιμοποιώντας frameworks όπως το Truffle ή το Hardhat, τα οποία προσομοιώνουν το blockchain Ethereum.

**Deployment**: Μετά τη δοκιμή, το contract γίνεται deploy στο blockchain. Αυτό γίνεται με την αποστολή μιας συναλλαγής με τον bytecode του contract.

**Αλληλεπίδραση**: Μόλις γίνει το deploy, οι χρήστες και άλλα contract μπορούν να αλληλεπιδράσουν με το contract στέλνοντας συναλλαγές στη διεύθυνσή της.

Το contract μπορεί να αποθηκεύσει το native token του chain (στο ethereum το ETH) καθώς και οποιοδήποτε άλλο token, μιας και έχει διεύθυνση και λειτουργεί σαν κανονικό wallet. Στο contract των token υπάρχει ένα mapping που δείχνει τα πόσα tokens έχει το κάθε address. Το contract μπορεί να είναι ένα από αυτά τα addresses. Μετά εμείς σαν προγραμματιστές μέσα στο δικό μας contract, εάν πολλοί χρήστες αλληλεπιδρούν με αυτό και έχουν tokens που τους ανήκουν, πρέπει να έχουμε δικές μας δομές που να δείχνουν πόσα ανήκουν σε ποιον.

## 3.2 Διαδικασία Υλοποίησης Smart Contracts

**Πώς λειτουργούν τα έξυπνα συμβόλαια**

**Ανάπτυξη**: Ένας χρήστης (συνήθως ο δημιουργός συμβολαίου) στέλνει μια συναλλαγή στο blockchain, που περιέχει τον μεταγλωττισμένο κώδικα του έξυπνου συμβολαίου. Αυτή η συναλλαγή, αφού εξορυχθεί, δημιουργεί ένα νέο συμβόλαιο στο blockchain, το οποίο μπορεί να αναγνωριστεί από μια διεύθυνση.

**Εκτέλεση**: Όταν οι χρήστες αλληλεπιδρούν με το contract (π.χ. πραγματοποιούν συναλλαγές σε ένα DEX), στέλνουν συναλλαγές στη διεύθυνση του συμβολαίου. Αυτές οι συναλλαγές καθορίζουν ποια λειτουργία του συμβολαίου θα καλέσετε και με ποιες παραμέτρους.

**Αλλαγή κατάστασης:** Το contract εκτελείται όπως έχει προγραμματιστεί, αλλάζοντας την κατάστασή της εάν είναι απαραίτητο. Για παράδειγμα, μια συναλλαγή σε ένα DEX μπορεί να αλλάξει τα υπόλοιπα των tokens που κατέχονται από το συμβόλαιο.

**Συμπερίληψη συναλλαγής**: Το αποτέλεσμα της εκτέλεσης καταγράφεται σε ένα μπλοκ, το οποίο στη συνέχεια διαδίδεται σε όλο το blockchain, διασφαλίζοντας ότι όλοι οι κόμβοι ενημερώνουν το αντίγραφό τους για την κατάσταση του contract.

**Ανάπτυξη Έξυπνων Συμβάσεων**

Εδώ θα δούμε ένα μικρό παράδειγμα της διαδικασία υλοποίησης ενός contract στο ethereum ή σε οποιοδήποτε άλλο EVM based chain (Polygon, Avalanche, Moonriver κτλ).

**Σύνταξη του συμβολαίου**: Η πιο δημοφιλής γλώσσα για τη σύνταξη έξυπνων συμβολαίων στο Ethereum είναι η Solidity. Είναι μια αντικειμενοστραφής γλώσσα προγραμματισμού για τη σύνταξη έξυπνων συμβολαίων που εκτελούνται στην εικονική μηχανή Ethereum (EVM).

```
pragma solidity ^0.8.0;

contract SimpleDEX {
    // Contract code goes here
}
```

**Σύνταξη Solidity:** Παρόμοια με την JavaScript και τη C++, το Solidity πληκτρολογείται στατικά και υποστηρίζει κληρονομικότητα, βιβλιοθήκες και σύνθετους τύπους που ορίζονται από τον χρήστη.

**Δοκιμή**: Πριν από την ανάπτυξη, τα contracts πρέπει να ελέγχονται αυστηρά. Αυτό μπορεί να γίνει χρησιμοποιώντας frameworks όπως το Truffle ή το Hardhat, τα οποία προσομοιώνουν το blockchain Ethereum.

**Deployment**: Μετά τη δοκιμή, το contract γίνεται deploy στο blockchain. Αυτό γίνεται με την αποστολή μιας συναλλαγής με τον bytecode του συμβολαίου.

**Αλληλεπίδραση**: Μόλις αναπτυχθούν, οι χρήστες και άλλες συμβάσεις μπορούν να αλληλεπιδράσουν με το contract στέλνοντας συναλλαγές στη διεύθυνσή της.

## 3.3 Smart Contracts στο Ethereum

Το Ethereum έχει αναδειχθεί ως η κορυφαία πλατφόρμα για την ανάπτυξη και την ανάπτυξη έξυπνων συμβολαίων, σε μεγάλο βαθμό λόγω του ισχυρού και ευέλικτου οικοσυστήματος του. Προσφέρει έναν μοναδικό συνδυασμό χαρακτηριστικών που το καθιστούν ιδανικό περιβάλλον για τη δημιουργία αποκεντρωμένων εφαρμογών (dApps), συμπεριλαμβανομένων των αποκεντρωμένων ανταλλαγών (DEX)

**Εικονική μηχανή Ethereum (EVM)**

Στο επίκεντρο των δυνατοτήτων του Ethereum βρίσκεται το EVM, μια εικονική μηχανή ενσωματωμένη σε κάθε πλήρη κόμβο Ethereum. Επιτρέπει σε οποιονδήποτε να εκτελέσει αυθαίρετο Κώδικα Byte EVM. Κάθε κόμβος Ethereum τρέχει στο EVM για να διατηρήσει τη συναίνεση σε όλο το blockchain. Τα έξυπνα συμβόλαια γράφονται σε γλώσσες υψηλού επιπέδου, όπως το Solidity, και στη συνέχεια μεταγλωτίζονται σε bytecode, τον οποίο το EVM μπορεί να διαβάσει και να εκτελέσει.

**Gas Fees**

Οι συναλλαγές στο Ethereum, συμπεριλαμβανομένων των αναπτύξεων έξυπνων συμβολαίων και των αλληλεπιδράσεων, απαιτούν υπολογιστικούς πόρους. Το gas είναι η μονάδα που μετρά το ποσό της υπολογιστικής προσπάθειας που απαιτείται για την εκτέλεση λειτουργιών. Κάθε λειτουργία σε ένα συμβόλαιο, από απλές συναλλαγές έως πολύπλοκες αλληλεπιδράσεις, κοστίζει μια συγκεκριμένη ποσότητα gas. Οι χρήστες πληρώνουν gas fees σε Ether (ETH), το native κρυπτονόμισμα του Ethereum, δίνοντας κίνητρα στους miners να επεξεργάζονται και να επικυρώνουν συναλλαγές.

Για παράδειγμα, μια τυπική μεταφορά ETH απαιτεί 21.000 gas units.

Οι χρήστες καθορίζουν μια τιμή του gas για τη συναλλαγή τους, που συμβολίζεται σε Gwei (1 Gwei = 10^-9 ETH). Η τιμή του gas είναι το ποσό του ETH που ο χρήστης είναι διατεθειμένος να πληρώσει ανά μονάδα αερίου. Οι minners δίνουν προτεραιότητα στις συναλλαγές με υψηλότερες τιμές gas επειδή πρόκειται να κερδίσουν περισσότερα από αυτές τις συναλλαγές.

Η συνολική χρέωση gas μιας συναλλαγής υπολογίζεται πολλαπλασιάζοντας τις απαιτούμενες μονάδες gas με την τιμή του gas: Total Gas Fee = Gas Units * Gas Price

Για παράδειγμα, εάν μια συναλλαγή απαιτεί 21.000 gas units και η τιμή του gas είναι 50 Gwei, η συνολική χρέωση φυσικού αερίου θα είναι 1.050.000 Gwei ή 0,00105 ETH.

Το update EIP-1559 του Ethereum [17] εισήγαγε έναν νέο μηχανισμό για τα fees. Περιλαμβάνει βασική χρέωση ανά gas που καίγεται και προαιρετικό tip στους miners. Το βασικό fee προσαρμόζεται αλγοριθμικά, καθιστώντας τις τιμές του gas πιο προβλέψιμες.Το update αυτό έκανε επίσης το ETH αποπληθωριστικό σε περιπτώσεις ότι υπάρχει υψηλή χρήση του δικτύου μιας και τότε τα ETH που καίγονται από τα fees είναι περισσότερα από αυτά που παράγονται.


Εικόνα 3.1: Το supply του ETH από το merge μέχρι τις 17 Δεκεμβρίου 2023 [18].

Επιστρέφοντας στην Solidity ας δούμε πως λειτουργεί η γλώσσα και μερικά παραδείγματα.

Ένα αρχείο Solidity έχει συνήθως επέκταση .sol και ξεκινά με μια έκδοση pragma για να δηλώσει ότι η έκδοση μεταγλωττιστή Solidity είναι συμβατή. Εδώ είναι μια βασική δομή:

```
// Καθορίζει την έκδοση της solidity
pragma solidity ^0.8.0;

// Ορίζει το contract, παρόμοιο με τις κλάσεις στις γνωστές γλώσσες προγραμματισμού
contract SimpleStorage {
    // State variable για την αποθήκευση αριθμών
    uint storedData;

    //Συνάρτηση για  αποθήκευση αριθμού στο contract
    function set(uint x) public {
        storedData = x;
    }

    // Get συνάρτηση αριθμού από το contract
    function get() public view returns (uint) {
        return storedData;
    }
```

```
}
```

**Βασικές Έννοιες**

**Μεταβλητές κατάστασης και συναρτήσεις:** Οι μεταβλητές κατάστασης αποθηκεύονται μόνιμα στο contract. Οι συναρτήσεις μπορούν να δηλωθούν ως public, private, internal ή external.

**Τύποι δεδομένων**: Οι συνηθισμένοι τύποι δεδομένων στο Solidity περιλαμβάνουν bool, uint, int, address, string και πιο σύνθετους τύπους όπως πίνακες και structs.

**Δομές ελέγχου:** Το Solidity υποστηρίζει δομές ελέγχου όπως οι βρόχοι if, else, while, for και do-while.

**Visibility Specifiers:** Οι συναρτήσεις και οι μεταβλητές κατάστασης έχουν προσδιοριστές ορατότητας:

public: Προσβάσιμο εξωτερικά και εσωτερικά (δημιουργεί μια συνάρτηση getter για μεταβλητές κατάστασης).
private: Πρόσβαση μόνο εντός του contract.
internal: Προσβάσιμο εντός του contract και σε παράγωγα contracts .
external: Προσβάσιμο μόνο εξωτερικά (μόνο για λειτουργίες)

**Modifiers**: Οι Modifiers συναρτήσεων μπορούν να χρησιμοποιηθούν για την αλλαγή της συμπεριφοράς των συναρτήσεων. Για παράδειγμα, ένα require statement μπορεί να χρησιμοποιηθεί για την επικύρωση εισόδων ή συνθηκών πριν από την εκτέλεση της λογικής συνάρτησης.

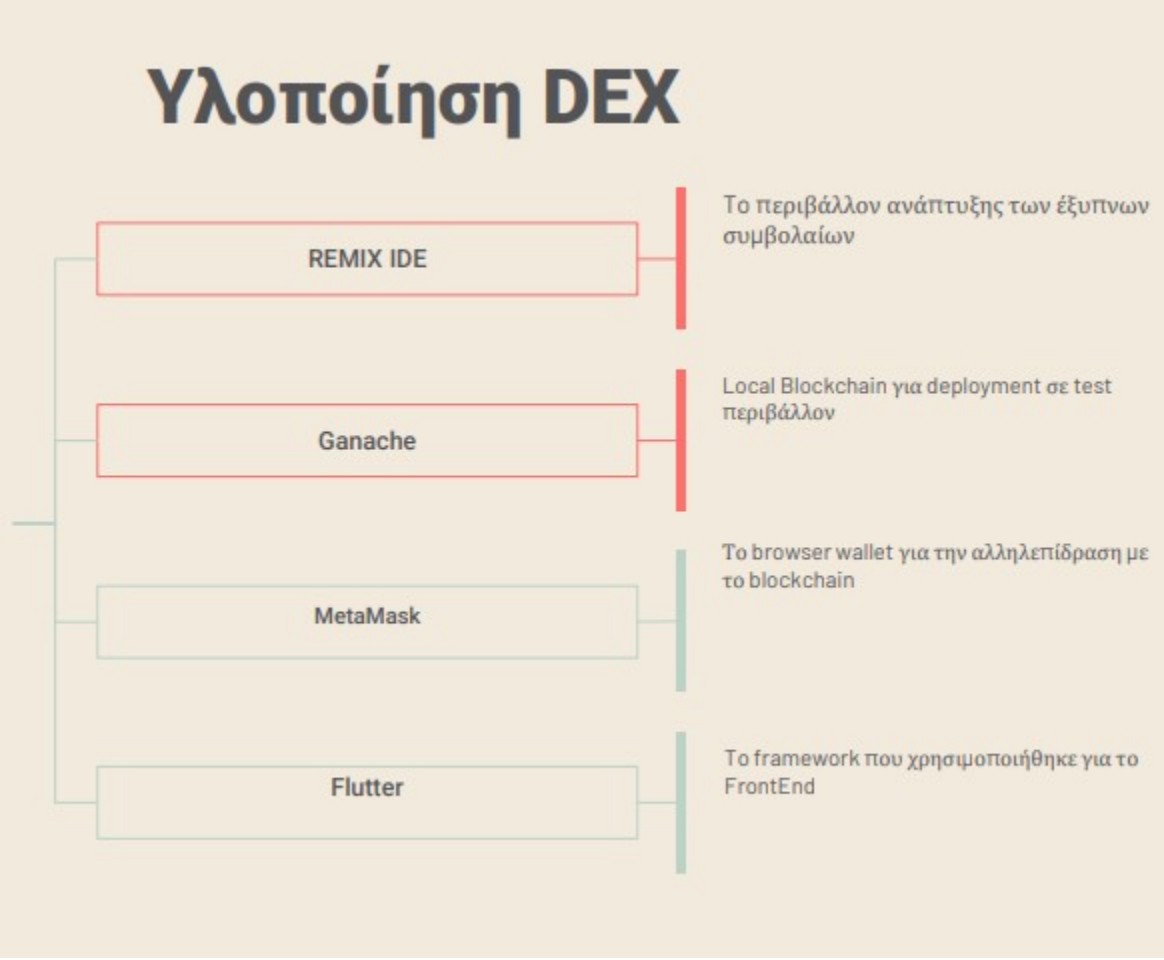
**Events**: Τα events στη Solidity επιτρέπουν τη σύνδεση στο blockchain Ethereum. Αυτά είναι χρήσιμα για την ειδοποίηση των clients σχετικά με ορισμένες αλλαγές.

# Κεφάλαιο 4: Σχεδιασμός και Υλοποίηση του DEX CryptoTrades.

Στο κεφάλαιο αυτό θα παρουσίασουμε το DEX το οποίο υλοποιήθηκε στο πλαίσιο αυτής της εργασίας. Το DEX αναπτύχθηκε χρησιμοποιώντας Solidity σε EVM chain και οι δοκιμές έγιναν σε ένα local blockchain χρησιμοποιώντας στο Ganache. Επίσης χρησιμοποιήθηκε το Remix IDE για την συγγραφή των contract και το deployment.

Στο κεφάλαιο αυτό θα αναλύσουμε με τη σειρά τη διαδικασία ανάπτυξης όλων των συστατικών μερών του smart contact  και θα παρουσιάσουμε και την λειτουργία του μέσα από παραδείγματα και code snippets.

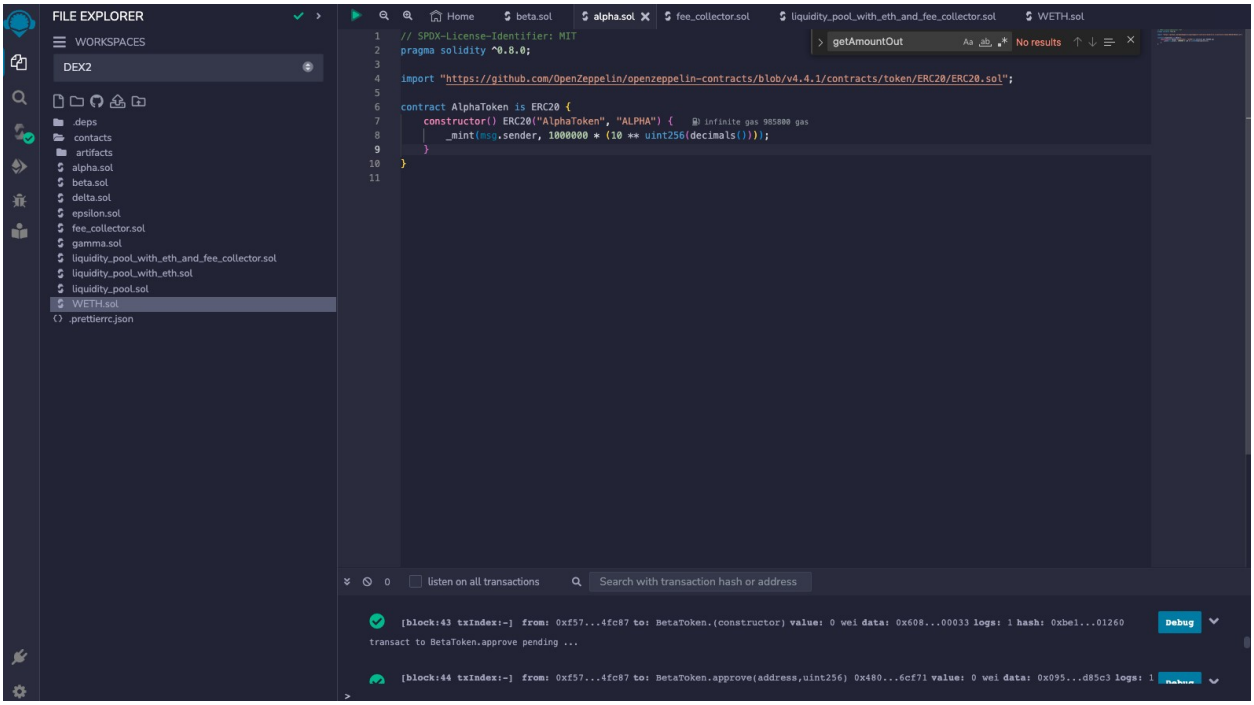
**Διάγραμμα Εργαλείων Υλοποίησης**




## 4.1 Remix IDE

Το Remix IDE [19] είναι ένα ολοκληρωμένο εργαλείο το οποίο επιτρέπει την εκτέλεση, το τεστάρισμα και το deployment EVM compatible smart contract.

Ένα από τα ξεχωριστά χαρακτηριστικά του Remix IDE είναι η φιλική προς το χρήστη διεπαφή του, η οποία είναι τόσο διαισθητική για τους νεοφερμένους όσο και αποτελεσματική για έμπειρους προγραμματιστές. Διευκολύνει τη διαδικασία ανάπτυξης από την αρχή μέχρι το τέλος. Αυτό περιλαμβάνει τη σύνταξη κώδικα με χρήσιμη επισήμανση σύνταξης και ανίχνευση σφαλμάτων, τη σύνταξη έξυπνων συμβολαίων, την ανάπτυξή τους είτε σε ένα εικονικό blockchain που παρέχεται από το Remix είτε σε πραγματικά δοκιμαστικά δίκτυα και την αλληλεπίδραση με αναπτυγμένες συμβάσεις για δοκιμαστικούς σκοπούς.



Εικόνα 4.1: Το γραφικό περιβάλλον του Remix

Εμείς τρέχουμε το Remix σαν app στον υπολογιστή, αλλά υπάρχει και η web έκδοση η οποία τρέχει από τον browser και έχει ακριβώς τις ίδιες δυνατότητες.

Αρχικά θα χρησιμοποιήσουμε το Remix VM για να δοκιμάσουμε τα contracts και στη συνέχεια θα τρέξουμε ένα local blockchain.

## 4.2 Υλοποίηση Tokens

Μιας και στο περιβάλλον που βρισκόμαστε (είτε στο Remix VM είτε στο Ganache local blockchain), δεν υπάρχουν tokens τα οποία είναι deployed, το πρώτο βήμα είναι να υλοποιήσουμε και να

κάνουμε deploy κάποια tokens για να μπορούμε να κάνουμε δοκιμές. Εάν το deploy γινόταν στο mainnet υπάρχουν ήδη πάρα πολλά tokens, συνεπώς δεν χρειάζεται να κάνουμε deploy εμείς τα δικά μας.

Το ERC20 είναι ένα ευρέως διαδεδομένο πρότυπο για tokens στο Ethereum. Καθορίζει μια κοινή λίστα κανόνων και λειτουργιών που πρέπει να τηρεί ένα token στο Ethereum. Αυτή η τυποποίηση επιτρέπει τη διαλειτουργικότητα μεταξύ διαφορετικών εφαρμογών και πλατφορμών στο οικοσύστημα Ethereum. Οι βασικές λειτουργίες που ορίζονται στο πρότυπο ERC20 περιλαμβάνουν:

**totalSupply**: Επιστρέφει τη συνολική προσφορά του token (πόσα tokens υπάρχουν).
**BalanceOf**: Παρέχει τον αριθμό των tokens που κατέχει μια δεδομένη διεύθυνση.
**transfer**: Επιτρέπει τη μεταφορά ενός καθορισμένου αριθμού tokens σε μια καθορισμένη διεύθυνση.
**transferFrom**: Επιτρέπει σε ένα συμβόλαιο να μεταφέρει tokens για λογαριασμό μιας διεύθυνσης, με προηγούμενη έγκριση.
**approve**: Επιτρέπει στον spender να αποσύρει ένα συγκεκριμένο ποσό tokens από έναν καθορισμένο λογαριασμό.
**allowance**: Επιστρέφει το ποσό των tokens που ένας κάτοχος επέτρεψε σε έναν spender (που μπορεί να είναι ένα smart contract, όπως ένα DEX) να αποσύρει από τον λογαριασμό του.

Τα ERC20 tokens είναι ουσιαστικά έξυπνα συμβόλαια στο Ethereum που ακολουθούν αυτούς τους κανόνες, καθιστώντας τα διαλειτουργικά με άλλα συμβόλαια, wallets και αποκεντρωμένες εφαρμογές συμβατές με το ERC20.

Με βάση τα παραπάνω, δημιουργήσαμε στο Remix κάποια demo tokens τα οποίο κληρονομούν όλα τα χαρακτηριστικά του προτύπου ERC20.

Ένα τέτοιο token είναι το παρακάτω:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/token/ERC20/ERC20.sol";

contract AlphaToken is ERC20 {
    constructor() ERC20("AlphaToken", "ALPHA") {
        _mint(msg.sender, 1000000 * (10 ** uint256(decimals())));
    }
}
```

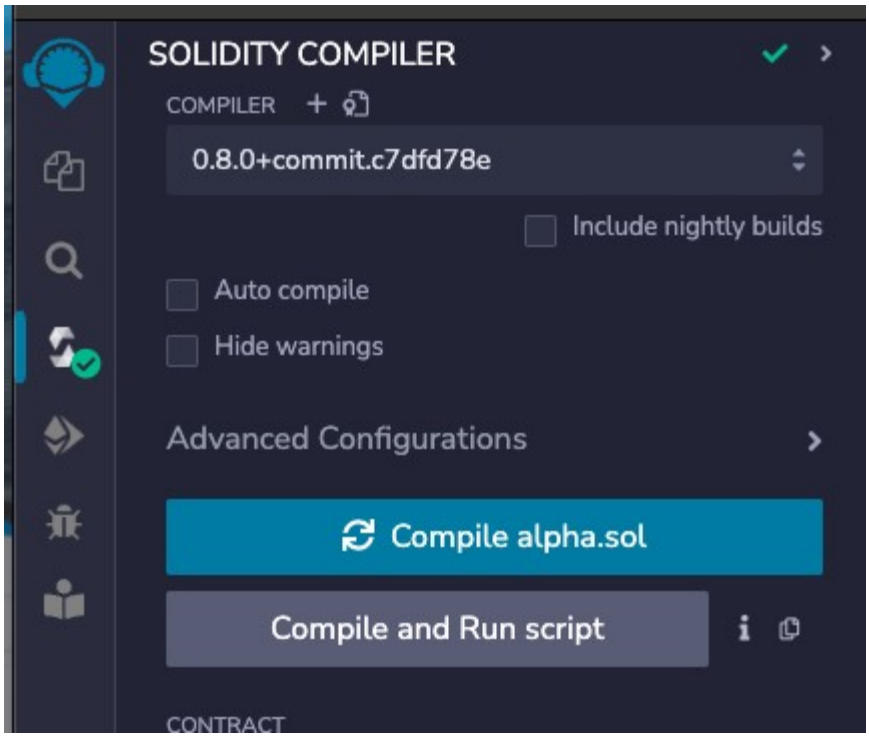
Αρχικά κάνουμε import κατευθείαν από το github (κάτι που είναι εφικτό στη solidity) το ERC20 standard από την υλοποίηση του OpenZeppelin μια βιβλιοθήκη ασφαλών και ελεγμένων από την κοινότητα έξυπνων συμβολαίων.

Στη συνέχεια έχουμε τα παρακάτω:

**contract AlphaToken is ERC20:** Αυτή η γραμμή δηλώνει ένα νέο έξυπνο συμβόλαιο που ονομάζεται AlphaToken, το οποίο κληρονομείται από την εφαρμογή του προτύπου ERC20 από το OpenZeppelin.

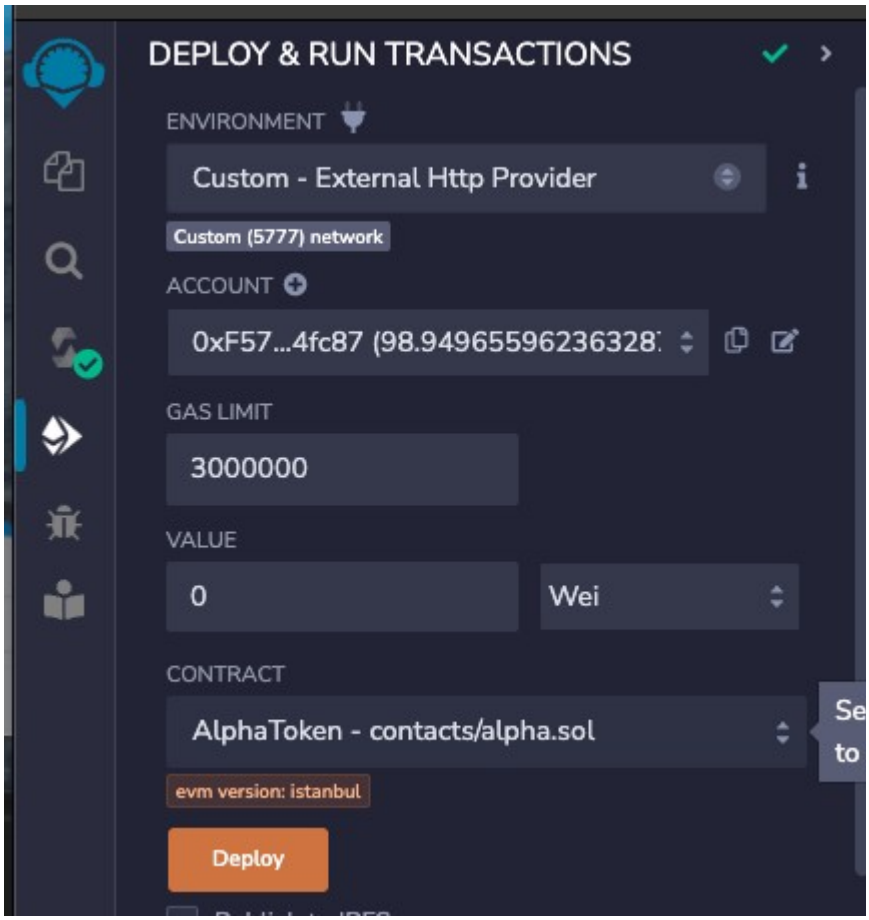
**constructor() ERC20("AlphaToken", "ALPHA"):** Ο constuctor είναι μια ειδική συνάρτηση που εκτελείται μία φορά, μόνο όταν γίνεται deploy το contract. Καλεί τον κατασκευαστή ERC20 από την υλοποίηση του OpenZeppelin, αρχικοποιώντας το διακριτικό με ένα όνομα (AlphaToken) και ένα σύμβολο (ALPHA).

**_mint(msg.sender, 1000000 * (10 ** uint256(decimals()))):** Αυτή η γραμμή είναι υπεύθυνη για τη δημιουργία της αρχικής προσφοράς των tokens. Δημιουργεί 1.000.000 AlphaTokens (πολλαπλασιάζοντας επί 10 ** δεκαδικά () για να ληφθούν υπόψη τα δεκαδικά ψηφία) και τα εκχωρεί στον λογαριασμό ο οποίος δημιούργησε το contract (msg.sender). Η συνάρτηση decimals(), που ορίζεται στο ERC20, επιστρέφει τον αριθμό των δεκαδικών ψηφίων που χρησιμοποιεί το διακριτικό. Αυτό είναι συνήθως 18, που σημαίνει ότι ένα AlphaToken διαιρείται σε 18 δεκαδικά ψηφία.



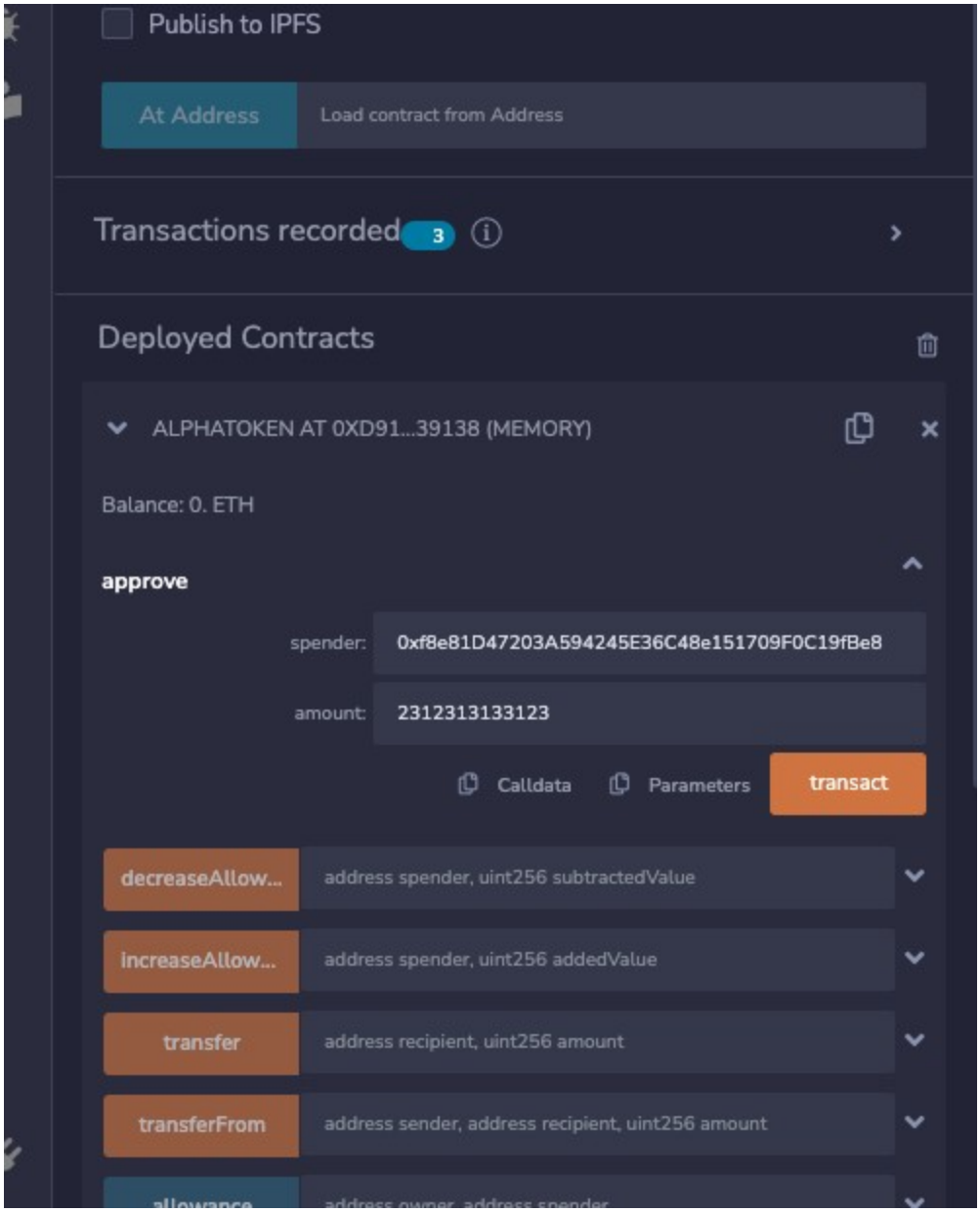

Εικόνα 4.2: Η διαδικασία του compile του alpha.sol

Αφού κάνουμε compile και deploy το token στο Remix (είτε στο Remix VM είτε στο Ganache), μπορούμε να αλληλεπιδράσουμε με αυτό από το μενού “Deploy & Run Transactions”



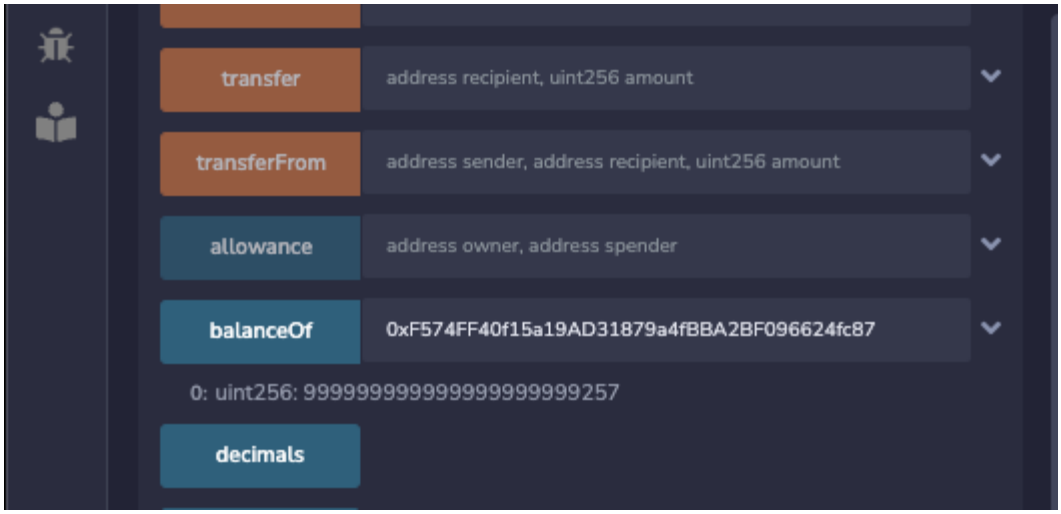

Εικόνα 4.3: Η διαδικασία του Deploy του AlphaToken. Στο environment επιλέγουμε που θέλουμε να γίνει το deploy (πχ στο remix VM, ή στο Ganache όπως στο παράδειγμα, ή ακόμα και σε ένα testnet ή mainnet. Στο account επιλέγουμε το account από το οποίο θέλουμε να γίνει το transaction. Κάθε virtual account ξεκινάει με 100 ETH.

Από το “Deploy & Run Transactions” μπορούμε να κάνουμε copy το address του token (στο παράδειγμα είναι το: 0xd9145CCE52D386f254917e481eB44e9943F39138) και να καλέσουμε όλες τις συναρτήσεις που υπάρχουν στο ERC20 standard από το γραφικό περιβάλλον που παρέχεται.




Εικόνα 4.4: Αλληλεπίδραση με το Smart Contract.

Αν καλέσουμε τη balanceOf για παράδειγμα με όρισμα το address του wallet μας θα δούμε πόσα tokens έχουμε στο wallet μας.




Εικόνα 4.5: Παράδειγμα balanceOf στο AlphaToken

## 4.3 Πρώτη Υλοποίηση Liquidity Pool

Μιας και το DEX ακολουθεί το μοντέλο AMM ουσιαστικά πρέπει να υλοποιήσουμε ένα Liquidity Pool όπου οι χρήστες θα μπορούν να προσθέσουν Liquidity αρχικά έτσι ώστε αργότερα να μπορούν να κάνουν τα swaps.

Η πρώτη έκδοση που υλοποιήσαμε βρίσκεται στο αρχείο liquidity_pool.sol και είναι η παρακάτω:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/token/ERC20/ERC20.sol";

contract LiquidityPool {
    // Define the structure of the pool
    struct TokenPool {
        uint256 token1Amount;
        uint256 token2Amount;
    }

    // Mapping of token pairs to their respective liquidity pool
    mapping(address => mapping(address => TokenPool)) public tokenPools;

    // Function to add liquiditys
    function addLiquidity(address token1, address token2, uint256 amountToken1, uint256 amountToken2) public {
        IERC20(token1).transferFrom(msg.sender, address(this), amountToken1);
        IERC20(token2).transferFrom(msg.sender, address(this), amountToken2);

        tokenPools[token1][token2].token1Amount += amountToken1;
        tokenPools[token1][token2].token2Amount += amountToken2;
    }

    // Function to remove liquidity
    function removeLiquidity(address token1, address token2, uint256 amountToken1, uint256 amountToken2) public {
        require(tokenPools[token1][token2].token1Amount >= amountToken1, "Not enough liquidity");
        require(tokenPools[token1][token2].token2Amount >= amountToken2, "Not enough liquidity");

        IERC20(token1).transfer(msg.sender, amountToken1);
        IERC20(token2).transfer(msg.sender, amountToken2);

        tokenPools[token1][token2].token1Amount -= amountToken1;
        tokenPools[token1][token2].token2Amount -= amountToken2;
    }

    // Function to swap tokens
    function swapTokens(address token1, address token2, uint256 amountToken1) public {
        TokenPool storage pool = tokenPools[token1][token2];
        require(pool.token1Amount > 0 && pool.token2Amount > 0, "Pool not initialized");
```

```
        uint256 token1Balance = IERC20(token1).balanceOf(address(this));
        uint256 token2Balance = IERC20(token2).balanceOf(address(this));
        require(token1Balance >= amountToken1, "Not enough liquidity");

        uint256 amountToken2 = getAmountOut(amountToken1, token1Balance, token2Balance);

        IERC20(token1).transferFrom(msg.sender, address(this), amountToken1);
        IERC20(token2).transfer(msg.sender, amountToken2);

        pool.token1Amount = token1Balance + amountToken1;
        pool.token2Amount = token2Balance - amountToken2;
    }

    // Pricing function based on the x * y = k model
    function getAmountOut(uint256 amountIn, uint256 reserveIn, uint256 reserveOut) public pure
returns (uint256) {
        require(amountIn > 0, "Insufficient input amount");
        require(reserveIn > 0 && reserveOut > 0, "Insufficient liquidity");

        uint256 amountInWithFee = amountIn * 997;
        uint256 numerator = amountInWithFee * reserveOut;
        uint256 denominator = (reserveIn * 1000) + amountInWithFee;
        return numerator / denominator;
    }
}
```

Αυτό το contract επιτρέπει στους χρήστες να προσθέτουν ρευστότητα σε pools, να αφαιρούν ρευστότητα και να πραγματοποιούν swaps.

Πιο αναλυτικά:

```
struct TokenPool {
    uint256 token1Amount;
    uint256 token2Amount;
}
```

Αυτή η δομή ορίζει ένα liquidity pool για ένα ζεύγος tokens, παρακολουθώντας το ποσό κάθε token στο pool.

Πρακτικά αυτό που γίνεται είναι ότι κάποιος στέλνει tokens στο liquidity pool contract. Αυτό σημαίνει ότι στο mapping του contract του ERC20 token θα γράφει πλέον ότι η address του liquidity pool

contract έχει x tokens. Εμείς παράλληλα στο liquidity pool contract πρέπει να κρατάμε πόσα tokens υπάρχουν για κάθε ζευγάρι που έχει εισαγθεί.

**Mapping: tokenPools**

```
mapping(address => mapping(address => TokenPool)) public tokenPools;
```

Αυτό το mapping αποθηκεύει στα structs TokenPool για κάθε ζεύγος διακριτικών, επιτρέποντας στο συμβόλαιο να χειρίζεται πολλαπλά liquidity pools.

**Function: addLiquidity**

```
function addLiquidity(address token1, address token2, uint256 amountToken1, uint256 amountToken2) public {...}
```

Επιτρέπει στους χρήστες να προσθέτουν ρευστότητα σε ένα συγκεκριμένο poll. Μεταφέρει τα καθορισμένα ποσά του token1 και του token2 από τη διεύθυνση του χρήστη στο contract (
IERC20(token1).transferFrom(msg.sender, address(this), amountToken1);
IERC20(token2).transferFrom(msg.sender, address(this), amountToken2);
).
Στη συνέχεια, τα υπόλοιπα των pools του contract ενημερώνονται ανάλογα (
tokenPools[token1][token2].token1Amount += amountToken1;
tokenPools[token1][token2].token2Amount += amountToken2;
).

**Function: removeLiquidity**

```
function removeLiquidity(address token1, address token2, uint256 amountToken1, uint256 amountToken2) public {...}
```

Επιτρέπει στους χρήστες να αποσύρουν ρευστότητα από ένα pool. Ελέγχει αν υπάρχει αρκετή ρευστότητα και στη συνέχεια μεταφέρει τα καθορισμένα ποσά του token1 και του token2 από το contract πίσω στον χρήστη. Τα υπόλοιπα του pool ενημερώνονται στη συνέχεια.

**Function: swapTokens**

```
function swapTokens(address token1, address token2, uint256 amountToken1) public {...}
```

Αυτή η λειτουργία επιτρέπει στους χρήστες να ανταλλάξουν το token1 με το token2. Υπολογίζει το ποσό του token2 που θα ληφθεί χρησιμοποιώντας τη συνάρτηση getAmountOut (με βάση το μοντέλο AMM) και εκτελεί την ανταλλαγή, ενημερώνοντας τα υπόλοιπα της ομάδας.

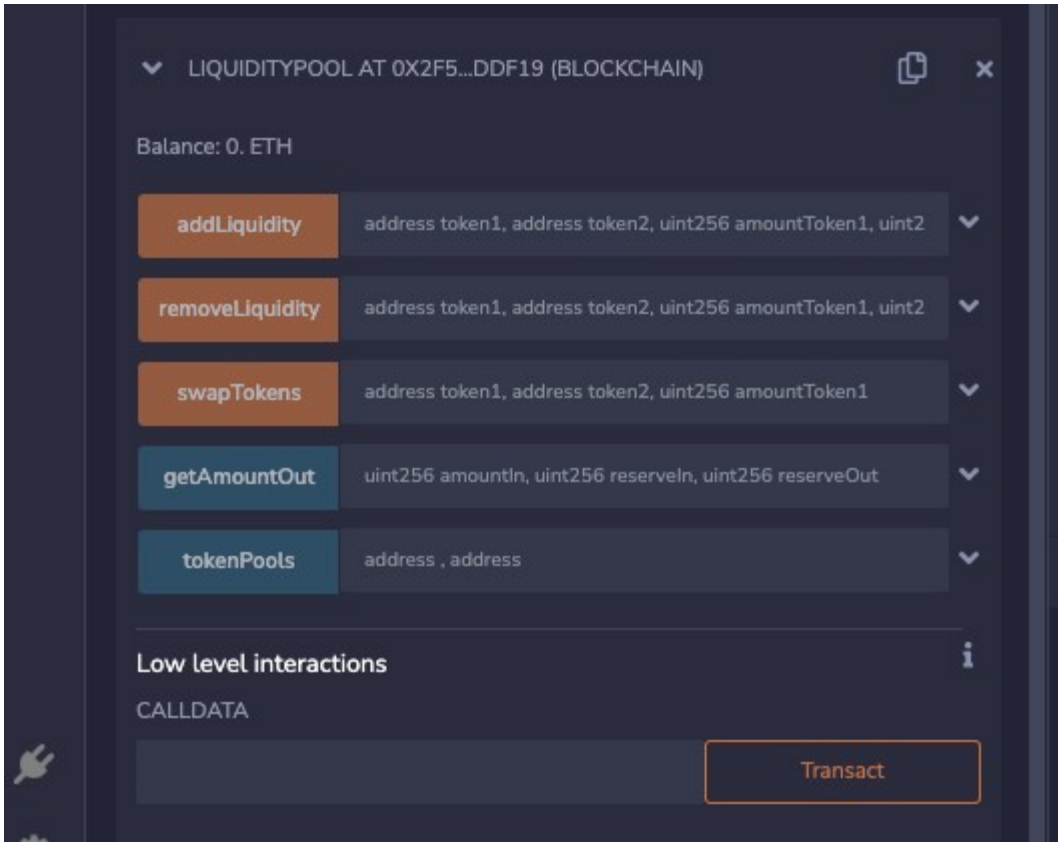
**Pricing Function: getAmountOut**

Εφαρμόζει τον τύπο τιμολόγησης AMM (x * y = k). Για ένα δεδομένο ποσό εισροών (amountIn) και τα τρέχοντα αποθεματικά στο pool (reserveIn και reserveOut), υπολογίζει το ποσό παραγωγής μετά την εφαρμογή ενός fee (0,3% σε αυτό το συμβόλαιο, όπως υποδεικνύεται από amountInWithFee = amountIn * 997, με 997/1000 είναι ο fee factor).

Με τον ίδιο τρόπο που κάνουμε deploy τα tokens, κάνουμε deploy και το Pool.

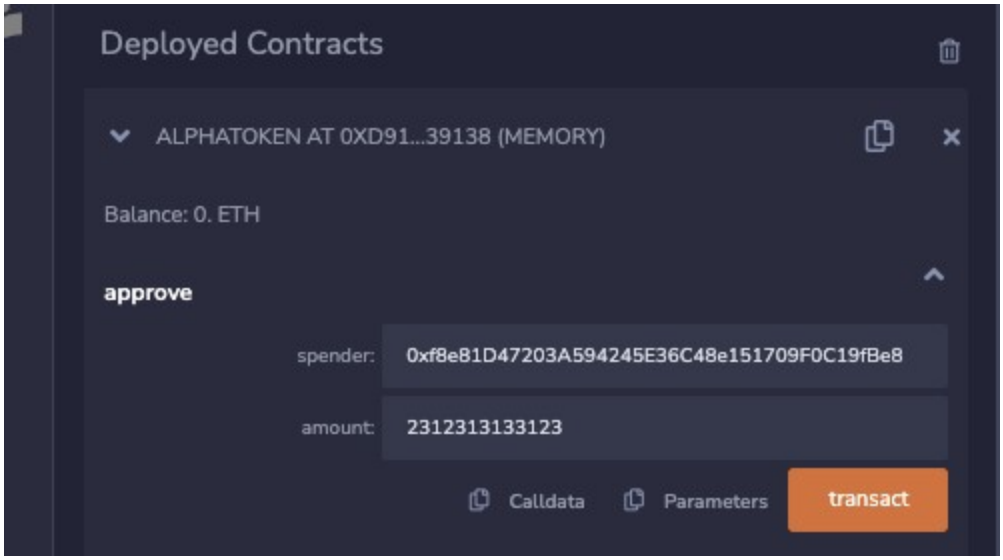

Εικόνα 4.6: Deployment του LiquidityPool Contract

Στο παράδειγμα το pool έχει τη διεύθυνση: 0xf8e81D47203A594245E36C48e151709F0C19fBe8.



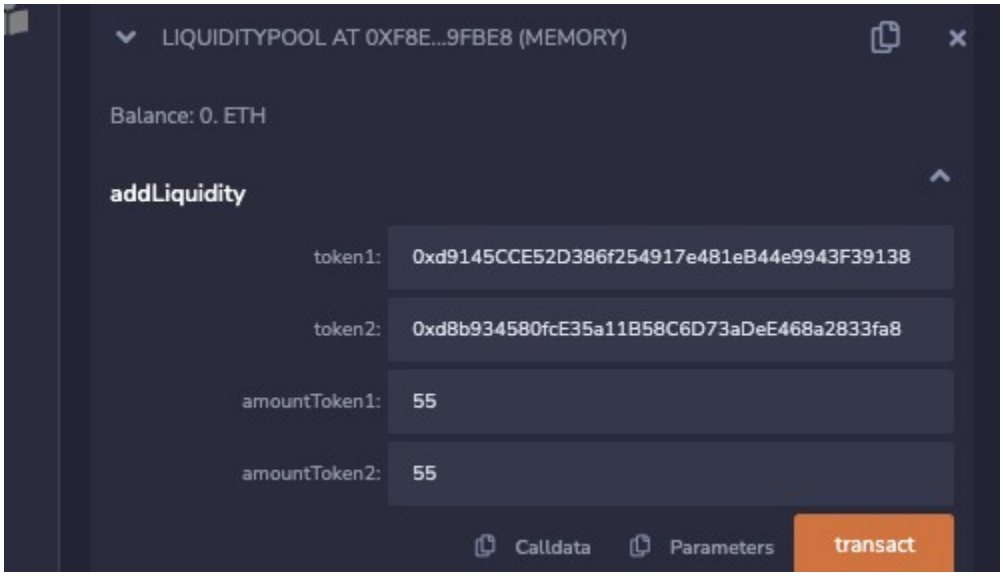

Εικόνα 4.7: Οι συναρτήσεις στο LiquidityPool Contract.

Για να μπορέσουμε να βάλουμε tokens από το AlphaToken ή από το BetaToken που έχουμε κάνει deploy, πρέπει να επιστρέψουμε στο smart contract του Pool να χρησιμοποιήσει τα tokens. Για να γίνει αυτό καλούμε την approve του AlphaToken και του BetaToken με όρισμα στο address του Pool Contract και το μέγιστο ποσό το οποίο μπορεί να ξοδευτεί στο pool, όπως φαίνεται στη παρακάτω εικόνα:



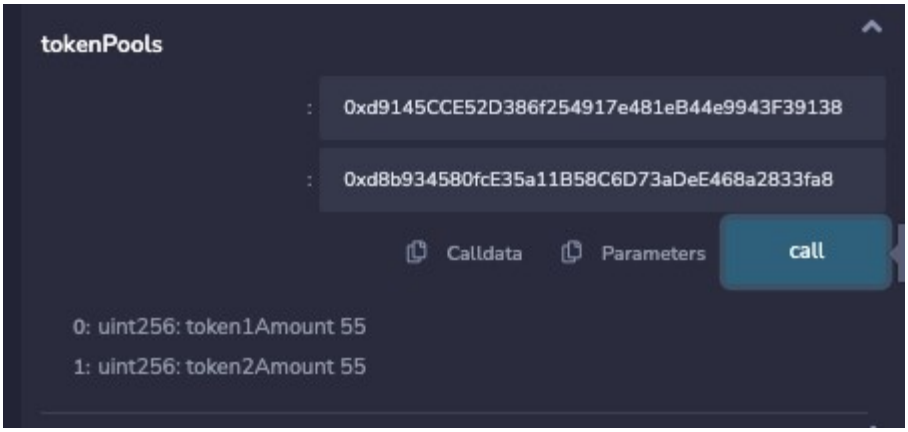

Εικόνα 4.8: Κλήση της συνάρτησης approve στο Alpha Token

Αφού το κάνουμε αυτό και για το token2, μπορούμε να προσθέσουμε liquidity για το ζευγάρι AlphaToken - BetatToken καλώντας την AddLiquidity όπως φαίνεται παρακάτω:



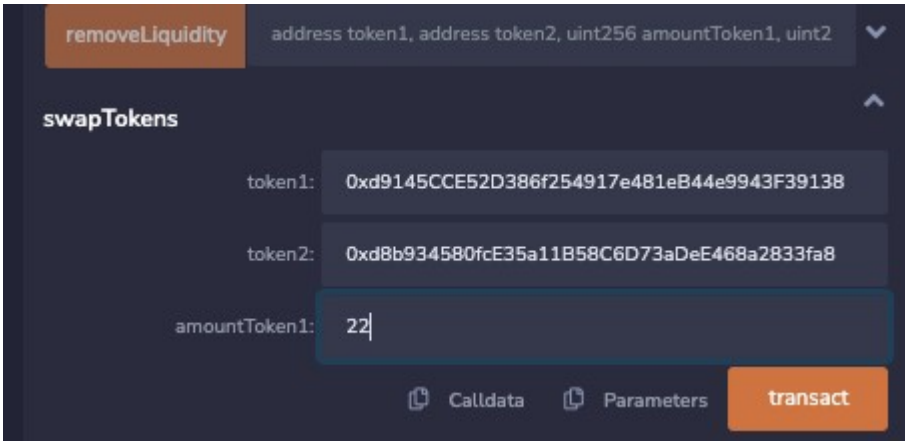

Εικόνα 4.9: Κλήση της addLiquidity για το pair AlphaToken - BetaToken

Μετά από αυτό το transaction το tokenPools για τα δύο αυτά tokens θα δείχνει το πλήθος των tokens που είναι στο pool όπως φαίνεται παρακάτω:



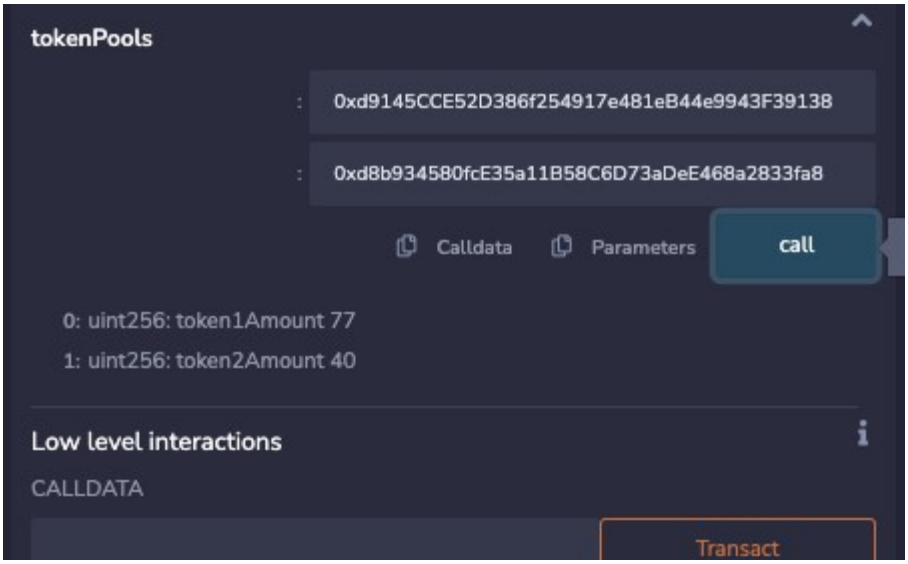

Εικόνα 4.10: Το pool για τα δύο tokens μετά το addLiquidity

Όσον αφορά το swap, ένα παράδειγμα φαίνεται παρακάτω:




Εικονα 4.11: Παράδειγμα swap των tokens

Μετά το swap, το pool μοιάζει ως εξής.




Εικόνα 4.12: Το pool μετά το swap.

## 4.4 Υποστήριξη του ETH

Ένα πρόβλημα που υπάρχει με το αρχικό αυτό contract είναι ότι δεν υποστηρίζει ETH, το native token του ethereum, μιας και το ETH δεν είναι ένα απλό ERC20 token, αλλά είναι native στο δίκτυο.

Για να το κάνουμε αυτό θα χρησιμοποιήσουμε το WETH έναν wrapper στο ETH. Με τον τρόπο αυτό όταν κάποιος κάνει deposit ETH εμείς θα το μετατρέπουμε σε ένα ERC20 token (το WETH) το οποίο κρατάει πάντα 1 προς 1 ETH με WETH. Με τον τρόπο αυτό θα χρειαστεί να κάνουμε μόνο λίγες αλλαγές στο προηγούμενο contract..

Το WETH contract που υλοποιήσαμε φαίνεται παρακάτω:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

/// used in local environments where we don't have the real ETH

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/token/ERC20/ERC20.sol";


contract WETH is ERC20 {
    constructor() ERC20("Wrapped Ether", "WETH") {}
```

```
    // Deposit ETH and mint WETH to sender account
    function deposit() external payable {
        _mint(msg.sender, msg.value);
    }

    // Burn WETH and withdraw ETH to sender account
    function withdraw(uint amount) external {
        require(balanceOf(msg.sender) >= amount, "Insufficient balance");
        _burn(msg.sender, amount);
        payable(msg.sender).transfer(amount);
    }
}
```

Με βάση αυτό το contract μπορούμε να φτιάξουμε το  ανανεωμένο liquidity pool contract ως εξής:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/token/ERC20/
ERC20.sol";

interface IWETH is IERC20 {
    function deposit() external payable;
    function withdraw(uint amount) external;
}

contract LiquidityPool {
    address public wethAddress;

    // Define the structure of the pool
    struct TokenPool {
        uint256 token1Amount;
        uint256 token2Amount;
    }

    // Mapping of token pairs to their respective liquidity pool
    mapping(address => mapping(address => TokenPool)) public tokenPools;

    constructor(address _wethAddress) {
        wethAddress = _wethAddress;
    }
```

```
    // Handle ETH deposits by converting to WETH
    function depositETH() external payable {
        IWETH(wethAddress).deposit{value: msg.value}();
    }

    // Handle ETH withdrawals by converting from WETH
    function withdrawETH(uint256 amount) external {
        IWETH(wethAddress).withdraw(amount);
    }

    // Function to add liquidity for ERC20 tokens or ETH (wrapped as WETH)
    function addLiquidity(address token1, address token2, uint256 amountToken1, uint256
amountToken2) public payable {
        if (token1 == wethAddress) {
            require(msg.value == amountToken1, "Incorrect ETH sent");
            IWETH(wethAddress).deposit{value: msg.value}();
        } else {
            IERC20(token1).transferFrom(msg.sender, address(this), amountToken1);
        }

        if (token2 == wethAddress) {
            require(msg.value == amountToken2, "Incorrect ETH sent");
            IWETH(wethAddress).deposit{value: msg.value}();
        } else {
            IERC20(token2).transferFrom(msg.sender, address(this), amountToken2);
        }

        tokenPools[token1][token2].token1Amount += amountToken1;
        tokenPools[token1][token2].token2Amount += amountToken2;
    }

    // Function to remove liquidity
    function removeLiquidity(address token1, address token2, uint256 amountToken1, uint256
amountToken2) public {
        require(tokenPools[token1][token2].token1Amount >= amountToken1, "Not enough liquidity");
        require(tokenPools[token1][token2].token2Amount >= amountToken2, "Not enough liquidity");

        if (token1 == wethAddress) {
            IWETH(wethAddress).withdraw(amountToken1);
            payable(msg.sender).transfer(amountToken1);
        } else {
            IERC20(token1).transfer(msg.sender, amountToken1);
        }
```

```
        if (token2 == wethAddress) {
            IWETH(wethAddress).withdraw(amountToken2);
            payable(msg.sender).transfer(amountToken2);
        } else {
            IERC20(token2).transfer(msg.sender, amountToken2);
        }

        tokenPools[token1][token2].token1Amount -= amountToken1;
        tokenPools[token1][token2].token2Amount -= amountToken2;
    }


    // Function to swap tokens
    function swap(address tokenIn, address tokenOut, uint256 amountIn) external {
        require(tokenIn != tokenOut, "Cannot swap the same token");
        require(tokenPools[tokenIn][tokenOut].token1Amount > 0 && tokenPools[tokenIn]
[tokenOut].token2Amount > 0, "Pool not initialized");

        // Calculate the amount of tokenOut that the user will receive
        uint256 amountOut = getAmountOut(amountIn, tokenPools[tokenIn][tokenOut].token1Amount,
tokenPools[tokenIn][tokenOut].token2Amount);

        // Transfer the tokenIn from the user to the pool
        IERC20(tokenIn).transferFrom(msg.sender, address(this), amountIn);

        // Handle ETH case
        if (tokenOut == wethAddress) {
            IWETH(wethAddress).withdraw(amountOut);
            payable(msg.sender).transfer(amountOut);
        } else {
            // Transfer the tokenOut from the pool to the user
            IERC20(tokenOut).transfer(msg.sender, amountOut);
        }

        // Update pool balances
        tokenPools[tokenIn][tokenOut].token1Amount += amountIn;
        tokenPools[tokenIn][tokenOut].token2Amount -= amountOut;
    }

    // Pricing function based on the x * y = k model
    function getAmountOut(uint256 amountIn, uint256 reserveIn, uint256 reserveOut) public pure
returns (uint256) {
```

```
        require(amountIn > 0, "Insufficient input amount");
        require(reserveIn > 0 && reserveOut > 0, "Insufficient liquidity");

        uint256 amountInWithFee = amountIn * 997; // 0.3% fee
        uint256 numerator = amountInWithFee * reserveOut;
        uint256 denominator = (reserveIn * 1000) + amountInWithFee;
        return numerator / denominator;
    }

}
```

**Contractor και διεύθυνση WETH:** Ο Contractor του νέου LiquidityPool παίρνει τώρα μια διεύθυνση για τη το contract του WETH. Αυτή η διεύθυνση χρησιμοποιείται σε όλη τη διάρκεια του contract για να προσδιοριστεί εάν πρέπει να γίνουν ενέργειες με το WETH ή με άλλο ERC20 token.

**Χειρισμός καταθέσεων και αναλήψεων ETH:**

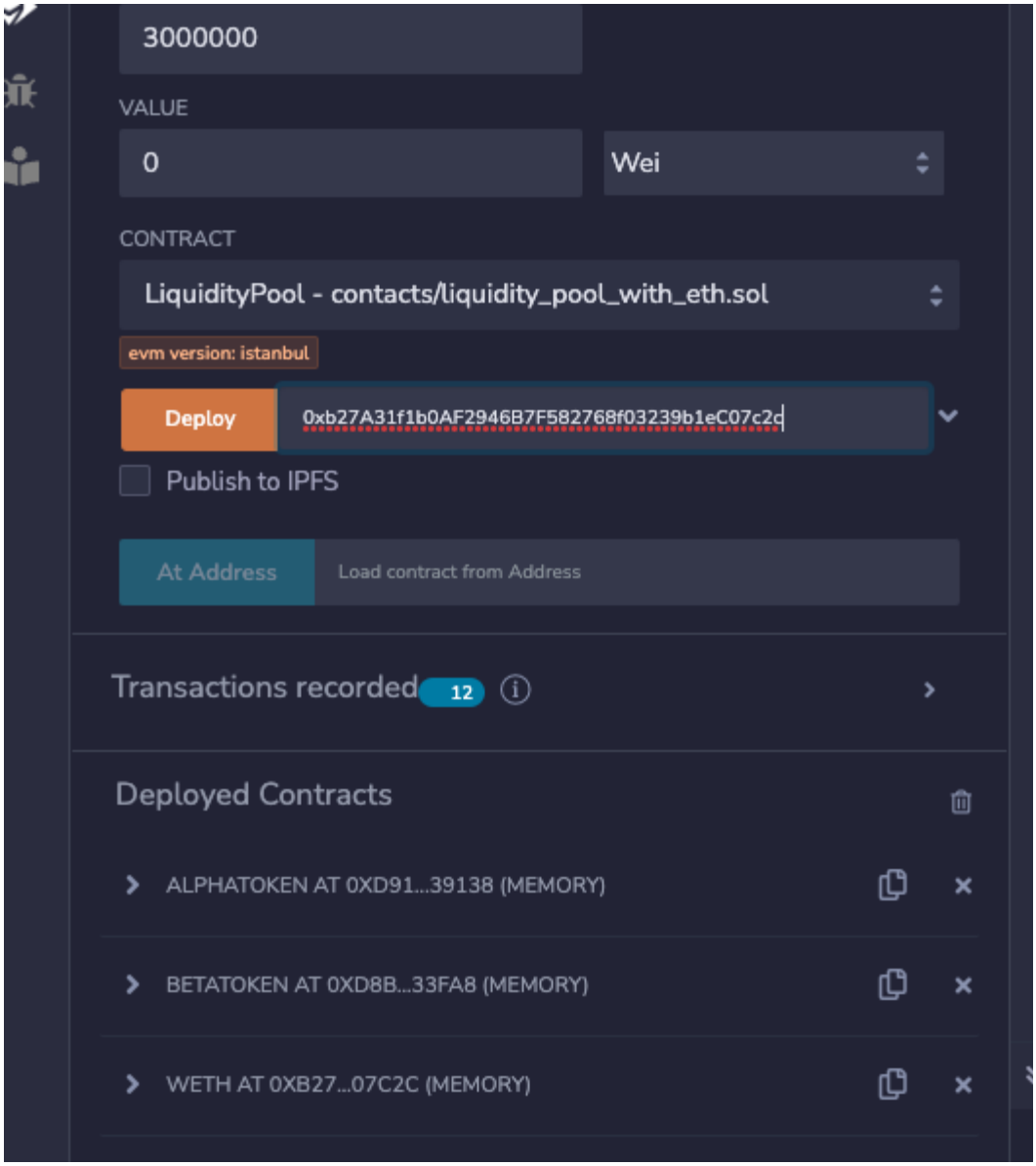
Η συνάρτηση depositETH επιτρέπει στους χρήστες να στείλουν ETH στο συμβόλαιο, το οποίο στη συνέχεια γίνεται wrap στο WETH. Η συνάρτηση αυτή τώρα, όπως και οι άλλες έχουν το χαρακτηριστικό "**payable**" που σημαίνει ότι ο χρήστης μπορεί να στείλει χρήματα (δηλαδή ETH) μαζί με το call που καλεί στη συνάρτηση.

Η συνάρτηση removeETH επιτρέπει στους χρήστες να κάνουν ανάληψη ETH κάνοντας unwrap το WETH.

**Προσθήκη και αφαίρεση ρευστότητας για ETH**: Οι συναρτήσεις addLiquidity και removeLiquidity έχουν τροποποιηθεί για να χειρίζονται περιπτώσεις όπου ένα από τα tokens είναι το ETH (WETH). Αυτό περιλαμβάνει τον έλεγχο εάν η διεύθυνση του token ταιριάζει με τη διεύθυνση WETH και τον χειρισμό καταθέσεων ή αναλήψεων αναλόγως.

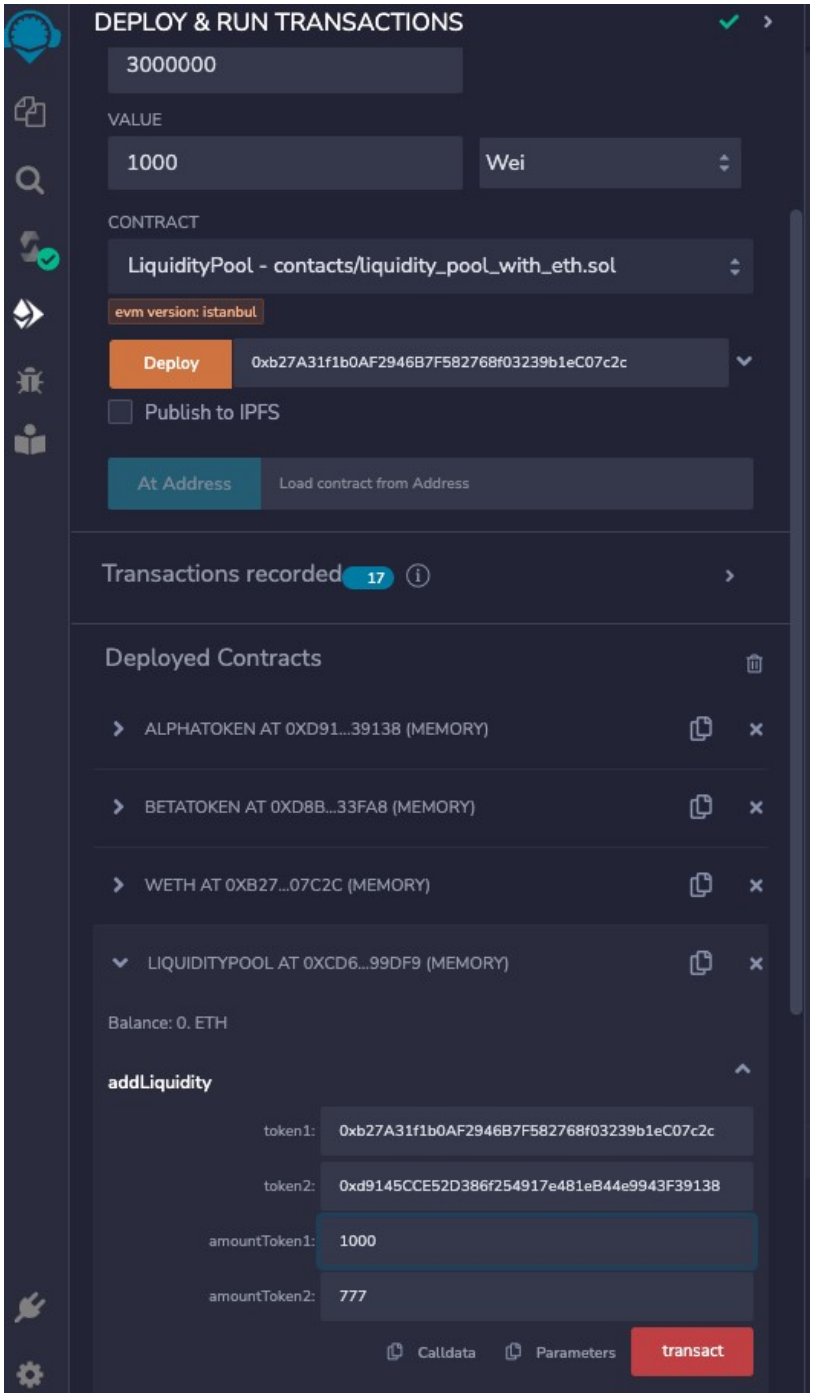
**Λειτουργικότητα swap Συμπεριλαμβανομένου του ETH**: Η συνάρτηση ανταλλαγής επιτρέπει πλέον την εναλλαγή μεταξύ δύο οποιωνδήποτε διακριτικών, συμπεριλαμβανομένου του ETH (με τη μορφή WETH). Διαχειρίζεται τη μετατροπή μεταξύ WETH και ETH όπως απαιτείται.

Για να κάνουμε deploy το νέο LiquidityPool contract πρέπει πρώτα να δώσουμε στον contractor το address του WETH (το οποίο πρέπει να έχουμε κάνει deploy πριν).



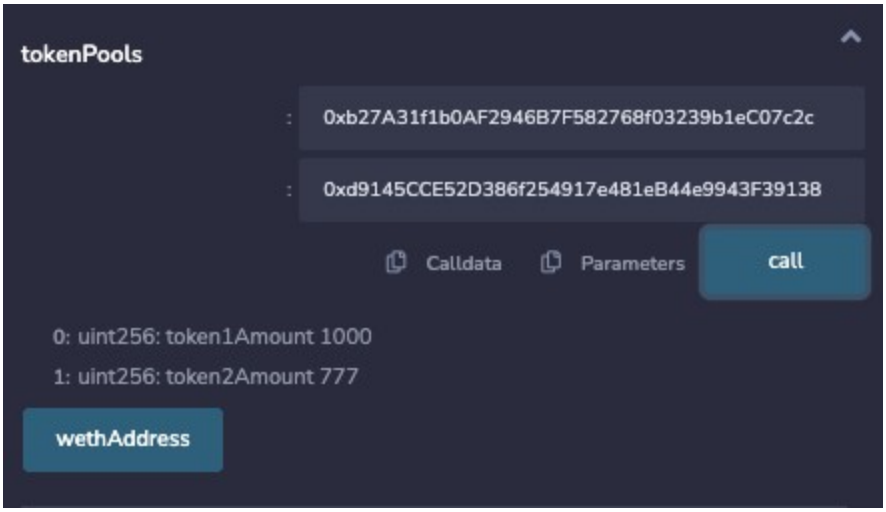

Εικόνα 4.13: Deploy του νέο LiquidityPool μαζί με το WETH address.

Αφού καλέσουμε την approve για το νέο LiquidityPool contract και τα τρία tokens (alpha, beta, weth), μπορούμε να στείλουμε κατευθείαν ETH στο contract και να κάνουμε swap με αυτό. Για να γίνει αυτό πρέπει στη VALUE να βάλουμε τα wei που θέλουμε να στείλουμε μέσω της payble συνάρτησης. Σαν token1 βάζουμε το WETH και σαν amount 1 πρέπει να βάλουμε το ίδιο με αυτό που στέλνουμε μέσω του VALUE (σε wei).




Εικόνα 4.14: Δημιουργία ETH-AlphaToken pair με addLiquidity

Όπως φαίνεται στην παρακάτω εικόνα το pool έχει ανανεωθεί:




Εικόνα 4.15: Το ανανεωμένο pool

Με τον ίδιο τρόπο λειτουργεί και το swap.

## 4.5 Υλοποίηση Fee Collector

Ένα άλλο θέμα το οποίο πρέπει να διευθετήσουμε είναι ο χειρισμός των fees. Στα προηγούμενα δύο contrats έχουμε τις παρακάτω γραμμές:

```
uint256 amountInWithFee = amountIn * 997; // 0.3% fee
uint256 numerator = amountInWithFee * reserveOut;
uint256 denominator = (reserveIn * 1000) + amountInWithFee;
```

Στις γραμμές αυτές ορίζουμε αυθαίρετα ένα ποσοστό του tokenIn το οποίο θα είναι το fee του DEX.

Συγκεκριμένα υπάρχει ένα μοντέλο βασικής προμήθειας που εφαρμόζεται στη συνάρτηση getAmountOut, όπου μια μικρή χρέωση (0,3% αντιπροσωπεύεται από τον παράγοντα 997/1000) συνυπολογίζεται στον υπολογισμό ανταλλαγής. Ωστόσο, είναι σημαντικό να σημειωθεί πώς γίνεται ο χειρισμός αυτής της χρέωσης και πού καταλήγει.

Στην τρέχουσα εφαρμογή, η προμήθεια παραμένει ουσιαστικά εντός του contract και λειτουργεί ως εξής:

**Είσπραξη προμήθειας**: Όταν εκτελείται μια ανταλλαγή (swap), η προμήθεια (0,3%) δεν μεταφέρεται ρητά ως ξεχωριστή συναλλαγή. Αντίθετα, παραμένει στην πισίνα, αλλοιώνοντας ελαφρώς τη συμβολική ισορροπία στο pool.

Ας υποθέσουμε ότι ένας χρήστης ανταλλάσσει το token1 με το token2. Χωρίς κανένα fee, εάν έβαζαν X ποσό του token1, θεωρητικά θα έβγαζαν το Y ποσό του token2 με βάση το μοντέλο x * y = k.

Με χρέωση 0,3%, ο χρήστης εξακολουθεί να εισάγει X ποσό του token1, αλλά αφαιρεί ελαφρώς μικρότερο από το Y ποσό του token2. Η διαφορά, που είναι η αμοιβή, μένει στο pool.

Σε πολλαπλές συναλλαγές, αυτές οι μικρές διαφορές αθροίζονται, αυξάνοντας ελαφρώς τη συνολική αξία που είναι κλειδωμένη στο pool.

Προφανώς αυτό δεν είναι ένα καλό fee management. Υπάρχουν διάφοροι τρόποι να διαχειριστούμε τα fees σαν creators του DEX. Κάποιοι από αυτούς φαίνονται παρακάτω:

**Συσσώρευση feed σε ξεχωριστό συμβόλαιο ή λογαριασμό**

Αυτό περιλαμβάνει την τροποποίηση της λογικής ανταλλαγής για να διαχωριστούν ρητά οι χρεώσεις και να αποσταλούν σε ξεχωριστό πορτοφόλι ή contract. Θα χρειαστεί να εφαρμόσουμε πρόσθετη παρακολούθηση και διαχείριση αυτών των κεφαλαίων, συμπεριλαμβανομένων μηχανισμών για τη διανομή ή τη χρήση αυτών των fees.

Απαιτεί προσεκτικό σχεδιασμό για να διασφαλιστεί η ασφάλεια, ειδικά εάν τα συσσωρευμένα τέλη γίνουν σημαντικά. Επίσης, θα πρέπει να αποφασίσουν για τους κανόνες διακυβέρνησης ή λειτουργίας για τη διαχείριση αυτών των κεφαλαίων.

Πολυπλοκότητα: Υψηλή

Αυτή είναι η πιο περίπλοκη επιλογή, καθώς απαιτεί όχι μόνο διαχωρισμό των fees αλλά και δημιουργία μηχανισμού διανομής. Αυτό θα μπορούσε να περιλαμβάνει πληρωμές που μοιάζουν με μερίσματα σε κατόχους διακριτικών, μηχανισμούς επαναγοράς και καύσης ή χρηματοδότηση συγκεκριμένων αναπτυξιακών δραστηριοτήτων.

Απαιτεί μια πιο προηγμένη κατανόηση της ανάπτυξης έξυπνων συμβολαίων και μπορεί να περιλαμβάνει τη δημιουργία πρόσθετων συμβολαίων για τη διαχείριση της διανομής. Εισάγει επίσης περισσότερες μεταβλητές στο οικοσύστημα DEX, όπως αποφάσεις διακυβέρνησης σχετικά με τον τρόπο χρήσης ή κατανομής των τελών.

**Στο δικό μας DEX επιλέγουμε να φτιάξουμε ένα ακόμα Smart Contract και να αποθηκεύουμε εκεί τα fees.**

Για να διαχειριστούμε τα fees συγκεντρώνοντάς τα σε ξεχωριστό contract και επιτρέποντας μόνο στον κάτοχο (εμάς) να κάνουμε withdraw, θα χρειαστεί να δημιουργήσετε ένα συμβόλαιο είσπραξης fees. Αυτή η contact θα συγκρατεί τις συσσωρευμένες χρεώσεις και θα παρέχει στον ιδιοκτήτη τη δυνατότητα να τις αποσύρει.

Το contract αυτό είναι το παρακάτω:

```
// SPDX-License-Identifier: MIT
```

```
pragma solidity ^0.8.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/access/
Ownable.sol";
import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/token/ERC20/
ERC20.sol";


contract FeeCollector is Ownable {
    // Function to withdraw collected ETH fees
    function withdrawETH(address payable recipient, uint256 amount) external onlyOwner {
        require(address(this).balance >= amount, "Insufficient ETH balance");
        recipient.transfer(amount);
    }

    // Function to withdraw collected ERC20 token fees
    function withdrawToken(address token, address recipient, uint256 amount) external onlyOwner {
        require(IERC20(token).balanceOf(address(this)) >= amount, "Insufficient token balance");
        IERC20(token).transfer(recipient, amount);
    }

    // Function to view the ETH balance stored in the contract
    function getETHBalance() external view returns (uint256) {
        return address(this).balance;
    }

    // Function to view the ERC20 token balance stored in the contract
    function getTokenBalance(address token) external view returns (uint256) {
        return IERC20(token).balanceOf(address(this));
    }

    // Allow the contract to receive ETH
    receive() external payable {}
}
```

**Ownable Contract:** Το συμβόλαιο χρησιμοποιεί το contract **Ownable** του OpenZeppelin, το οποίο παρέχει βασικές λειτουργίες ελέγχου εξουσιοδότησης, απλοποιώντας την υλοποίηση των αδειών χρήστη. Ο δημιουργός του contract (εμείς) θα οριστεί ως ιδιοκτήτης.

**Λειτουργία ανάληψης**: Αυτή η συνάρτηση επιτρέπει **μόνο στον κάτοχο** να κάνει withdraw κεφάλαια σε μια καθορισμένη διεύθυνση.

Στο νέο liquidity pool contract, προσθέτουμε τώρα τη δυνατότητα να ορίζουμε το address του feeColector κατά τη δημιουργία (στον contractor). Επίσης τώρα στο Swap τα fee πηγαίνουν στο contract αυτό:

```
        // Calculate fee
        uint256 fee = amountIn * 3 / 1000; // 0.3% fee
        emit FeeCalculated(fee);

        // Send fee to Fee Collector


        IERC20(tokenIn).transfer(feeCollectorAddress, fee);
```

Το τελικό smart contract του Pool είναι το παρακάτω:

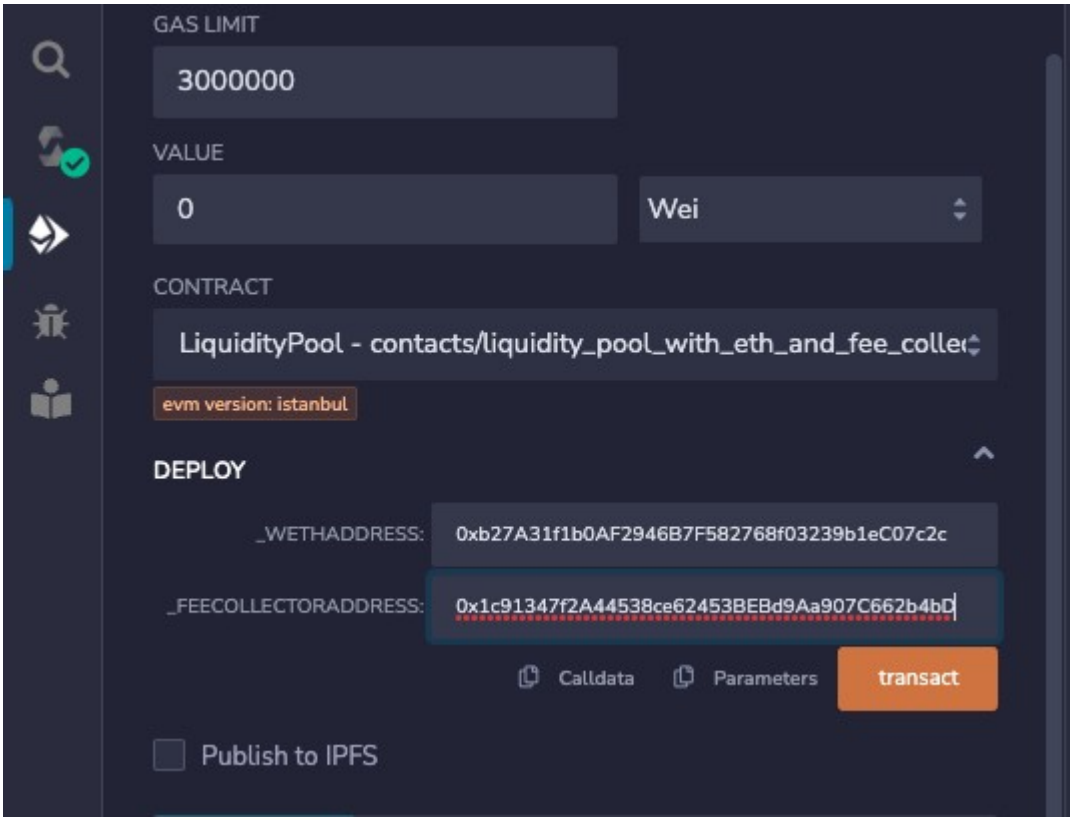
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.1/contracts/token/ERC20/ERC20.sol";

interface IWETH is IERC20 {
    function deposit() external payable;
    function withdraw(uint amount) external;
}

contract LiquidityPool {
    // Events
    event LiquidityAdded(address indexed provider, address token1, address token2, uint256 amountToken1, uint256 amountToken2);
    event LiquidityRemoved(address indexed provider, address token1, address token2, uint256 amountToken1, uint256 amountToken2);
    event SwapExecuted(address indexed user, address tokenIn, address tokenOut, uint256 amountIn, uint256 amountOut);
    event FeeCalculated(uint256 fee);

    address public wethAddress;
    address public feeCollectorAddress;

    // Define the structure of the pool
    struct TokenPool {
```

```
        uint256 token1Amount;
        uint256 token2Amount;
    }

    // Mapping of token pairs to their respective liquidity pool
    mapping(address => mapping(address => TokenPool)) public tokenPools;



    constructor(address _wethAddress, address _feeCollectorAddress) {
        wethAddress = _wethAddress;
        feeCollectorAddress = _feeCollectorAddress;
    }


    // Handle ETH deposits by converting to WETH
    function depositETH() external payable {
        IWETH(wethAddress).deposit{value: msg.value}();
    }

    // Handle ETH withdrawals by converting from WETH
    function withdrawETH(uint256 amount) external {
        IWETH(wethAddress).withdraw(amount);
    }

    // Function to add liquidity for ERC20 tokens or ETH (wrapped as WETH)
    function addLiquidity(address token1, address token2, uint256 amountToken1, uint256
amountToken2) public payable {
        if (token1 == wethAddress) {
            require(msg.value == amountToken1, "Incorrect ETH sent");
            IWETH(wethAddress).deposit{value: msg.value}();
        } else {
            IERC20(token1).transferFrom(msg.sender, address(this), amountToken1);
        }

        if (token2 == wethAddress) {
            require(msg.value == amountToken2, "Incorrect ETH sent");
            IWETH(wethAddress).deposit{value: msg.value}();
        } else {
            IERC20(token2).transferFrom(msg.sender, address(this), amountToken2);
        }

        tokenPools[token1][token2].token1Amount += amountToken1;
```

```
        tokenPools[token1][token2].token2Amount += amountToken2;

        emit LiquidityAdded(msg.sender, token1, token2, amountToken1, amountToken2);

    }

    // Function to remove liquidity
    function removeLiquidity(address token1, address token2, uint256 amountToken1, uint256
amountToken2) public {
        require(tokenPools[token1][token2].token1Amount >= amountToken1, "Not enough liquidity");
        require(tokenPools[token1][token2].token2Amount >= amountToken2, "Not enough liquidity");

        if (token1 == wethAddress) {
            IWETH(wethAddress).withdraw(amountToken1);
            payable(msg.sender).transfer(amountToken1);
        } else {
            IERC20(token1).transfer(msg.sender, amountToken1);
        }

        if (token2 == wethAddress) {
            IWETH(wethAddress).withdraw(amountToken2);
            payable(msg.sender).transfer(amountToken2);
        } else {
            IERC20(token2).transfer(msg.sender, amountToken2);
        }

        tokenPools[token1][token2].token1Amount -= amountToken1;
        tokenPools[token1][token2].token2Amount -= amountToken2;

        emit LiquidityRemoved(msg.sender, token1, token2, amountToken1, amountToken2);

    }


    // Function to swap tokens
    function swap(address tokenIn, address tokenOut, uint256 amountIn) external {
        require(tokenIn != tokenOut, "Cannot swap the same token");
        require(tokenPools[tokenIn][tokenOut].token1Amount > 0 && tokenPools[tokenIn]
[tokenOut].token2Amount > 0, "Pool not initialized");

        // Calculate the amount of tokenOut that the user will receive
        uint256 amountOut = getAmountOut(amountIn, tokenPools[tokenIn][tokenOut].token1Amount,
tokenPools[tokenIn][tokenOut].token2Amount);
```

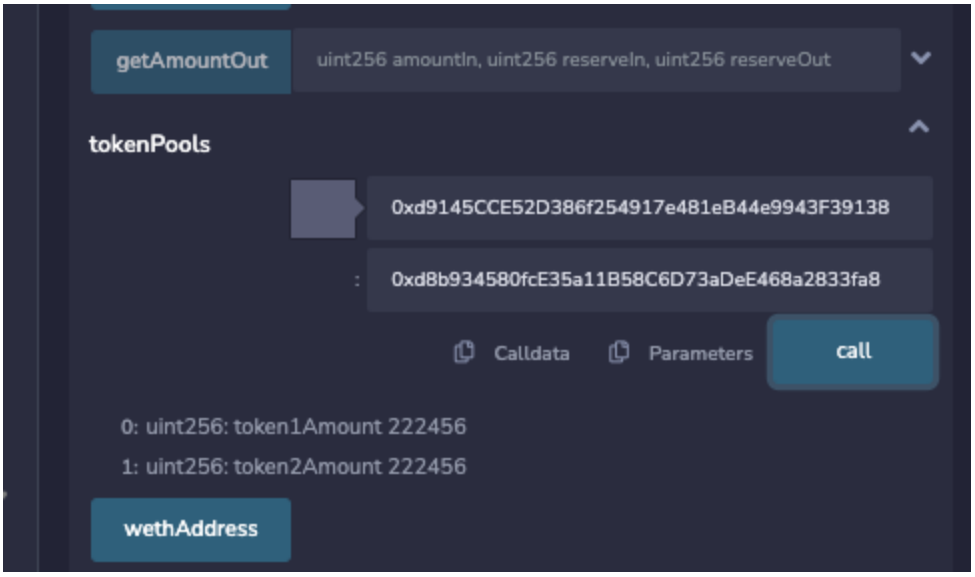
```
        // Transfer the tokenIn from the user to the pool
        IERC20(tokenIn).transferFrom(msg.sender, address(this), amountIn);

        // Calculate fee
        uint256 fee = amountIn * 3 / 1000; // 0.3% fee
        emit FeeCalculated(fee);

        // Send fee to Fee Collector
        //test αυτό πρέπει να δουλευει κανονικ΄ά!

        IERC20(tokenIn).transfer(feeCollectorAddress, fee);

        // Handle ETH case
        if (tokenOut == wethAddress) {
            IWETH(wethAddress).withdraw(amountOut);
            payable(msg.sender).transfer(amountOut);
        } else {
            // Transfer the tokenOut from the pool to the user
            IERC20(tokenOut).transfer(msg.sender, amountOut);
        }

        // Update pool balances
        tokenPools[tokenIn][tokenOut].token1Amount += amountIn;
        tokenPools[tokenIn][tokenOut].token2Amount -= amountOut;

        emit SwapExecuted(msg.sender, tokenIn, tokenOut, amountIn, amountOut);

    }

    // Pricing function based on the x * y = k model
    function getAmountOut(uint256 amountIn, uint256 reserveIn, uint256 reserveOut) public pure
returns (uint256) {
        require(amountIn > 0, "Insufficient input amount");
        require(reserveIn > 0 && reserveOut > 0, "Insufficient liquidity");

        uint256 amountInWithFee = amountIn * 997; // 0.3% fee
        uint256 numerator = amountInWithFee * reserveOut;
        uint256 denominator = (reserveIn * 1000) + amountInWithFee;
        return numerator / denominator;
    }

}
```

Για να κάνουμε deploy το νέο αυτό contract, πρέπει να δώσουμε στον contractor και το address του WETH και το FeeCollector όπως φαίνεται στη παρακάτω εικόνα:



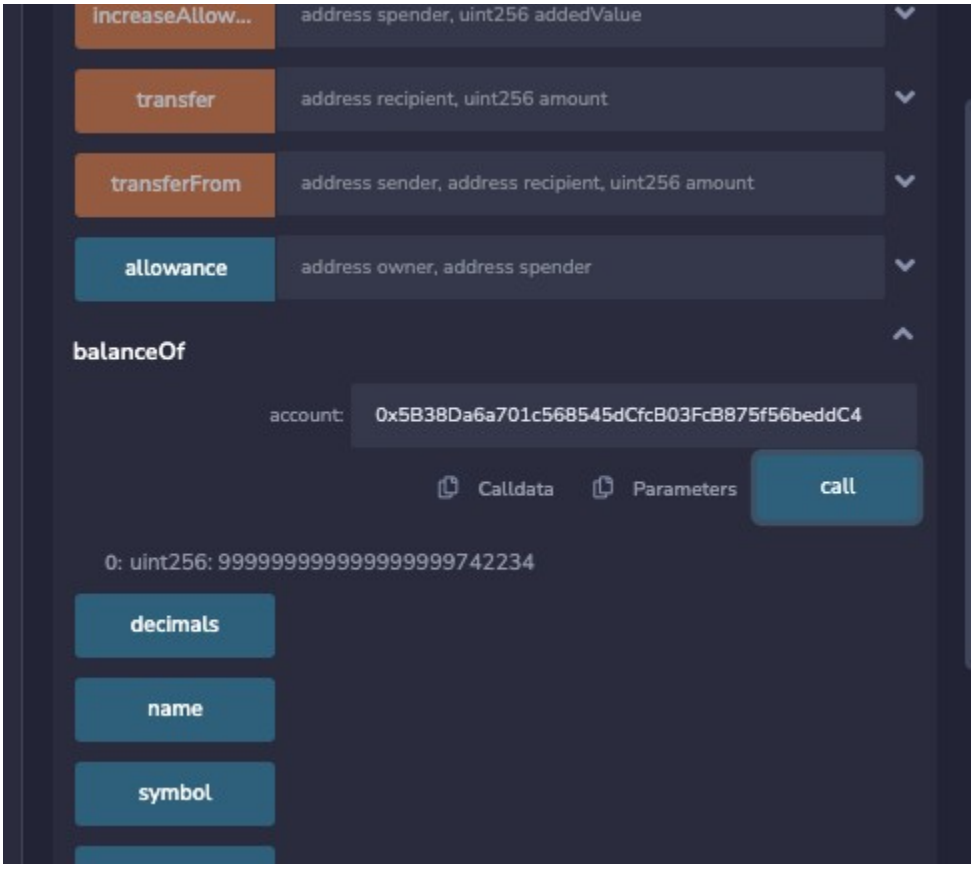

Εικόνα 4.16: Deploy του τελικού Contract

Στη συνέχεια αφού κάνουμε τα κατάλληλα approves και φτιάξουμε ένα Pool όπως φαίνεται στην εικόνα 4.17, κάνουμε ένα swap.

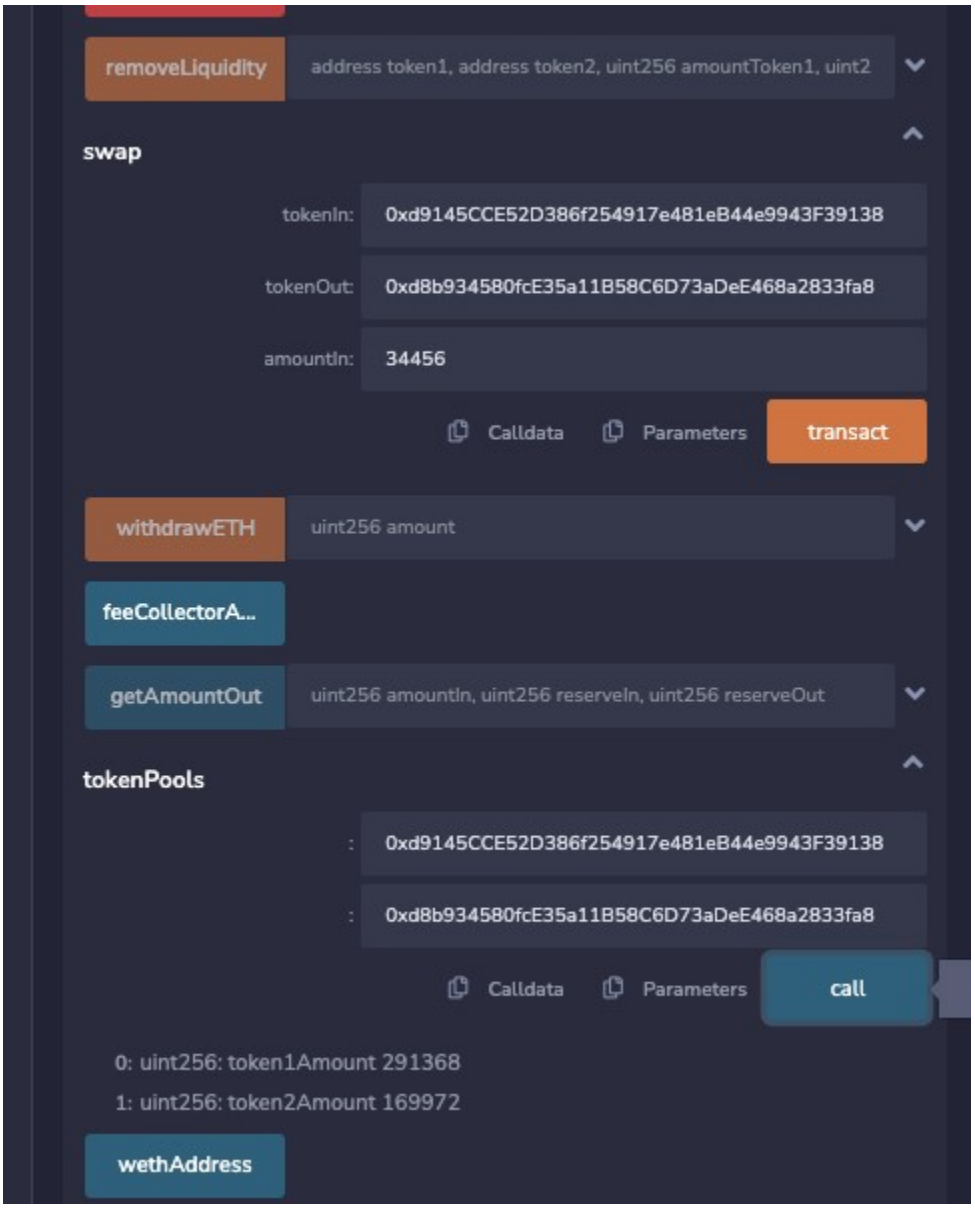



Εικόνα 4.17: Το pool του alphaToken - betaToken

Αφού το swap ολοκληρωθεί ελέγχουμε τον feeCollector και βλέπουμε ότι τα fees έχουν όντως μεταφερθεί εκεί.

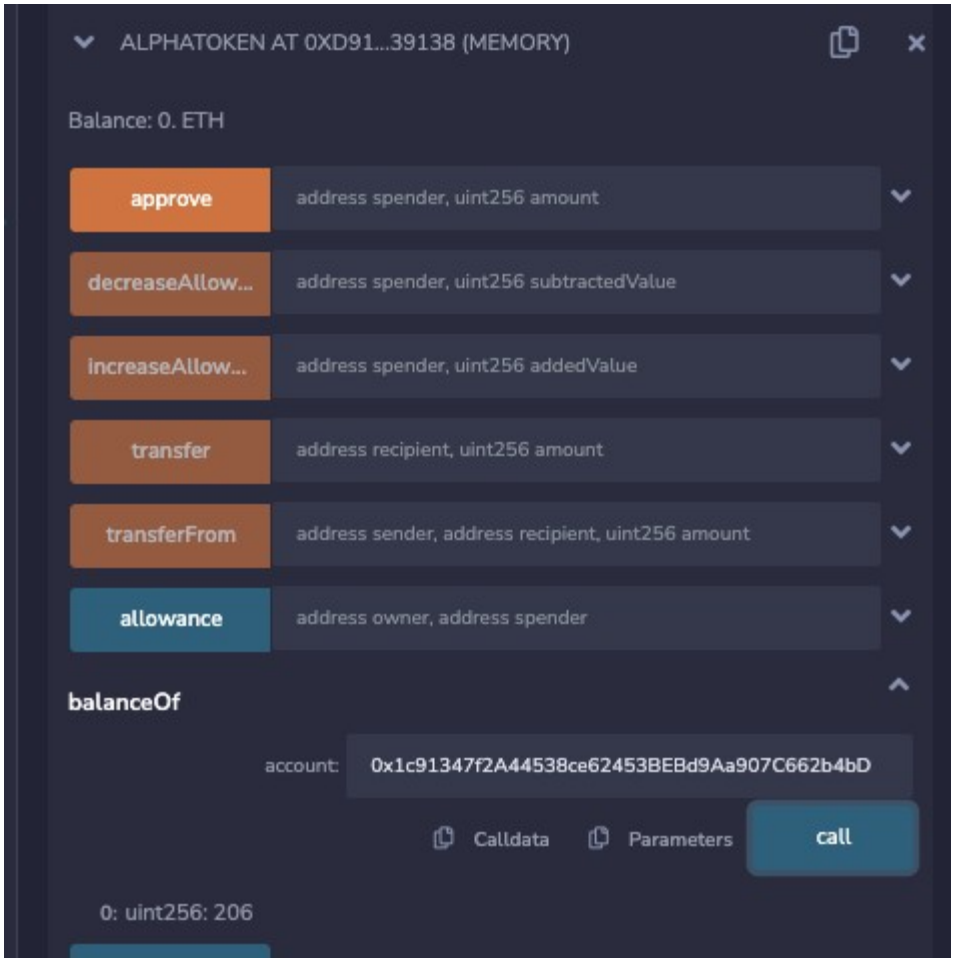



Εικόνα 4.18: Τα AlphaToken που μας ανήκουν (από το alphaToken contract)




Εικόνα 4.19: Κλήση ενός swap και εμφάνιση του tokenPool


Εικόνα 4.20: Κλήση του getTokenBalane μετά από κάποια swaps. Τα tokens έχουν μεταφερθεί εκεί.



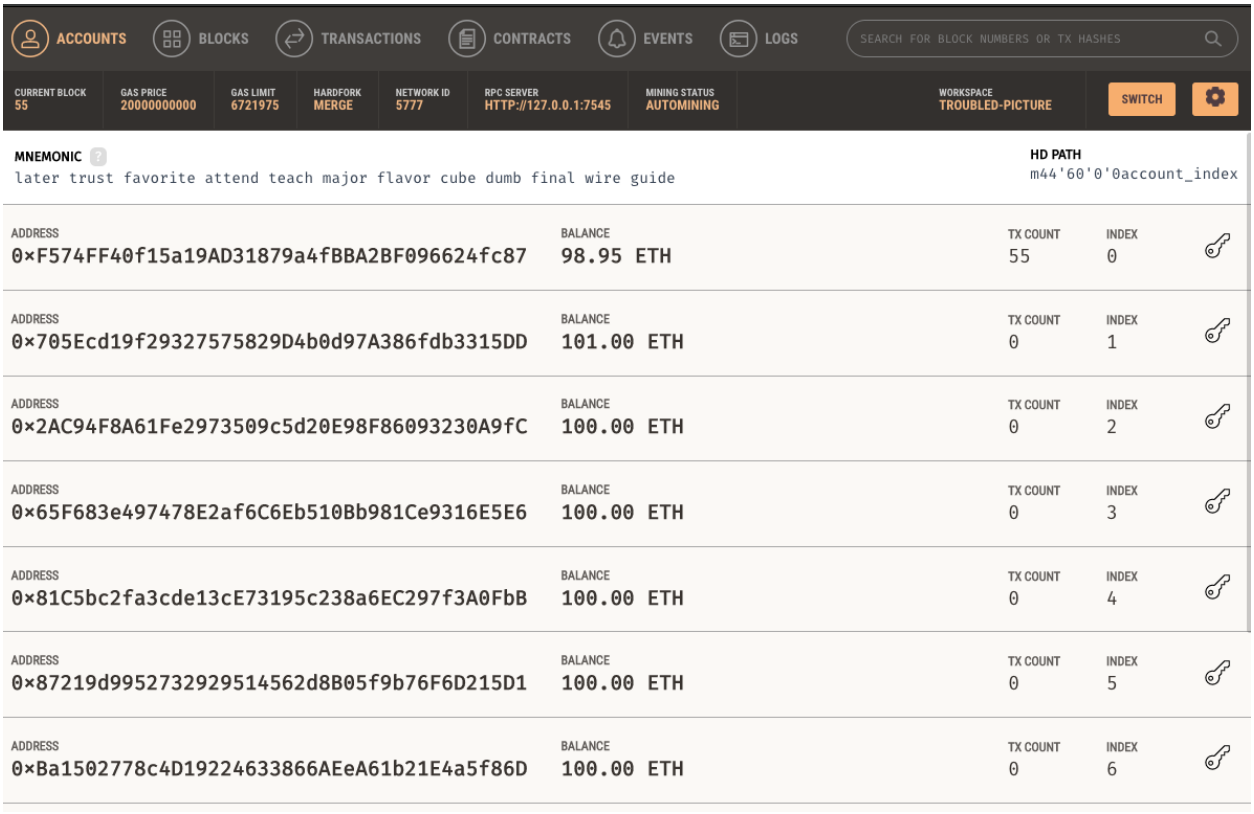

Εικόνα 4.21: Επαλήθευση, καλώντας την balanceOf από το AlphaToken δίνοντας σαν address το address του FeeCollector.

## 4.6 Deployment τελικού Liquidity Pool Contract

Μέχρι τώρα χρησιμοποιούσαμε το Remix VM, αλλά για να μπορέσουμε να κάνουμε ένα κανονικό deploy στο οποίο θα μπορέσουμε να φτιάξουμε και ένα frontend, θα κάνουμε deploy στο ganache [20].

Το Ganache είναι είναι local blockchain για να γίνονται tests.

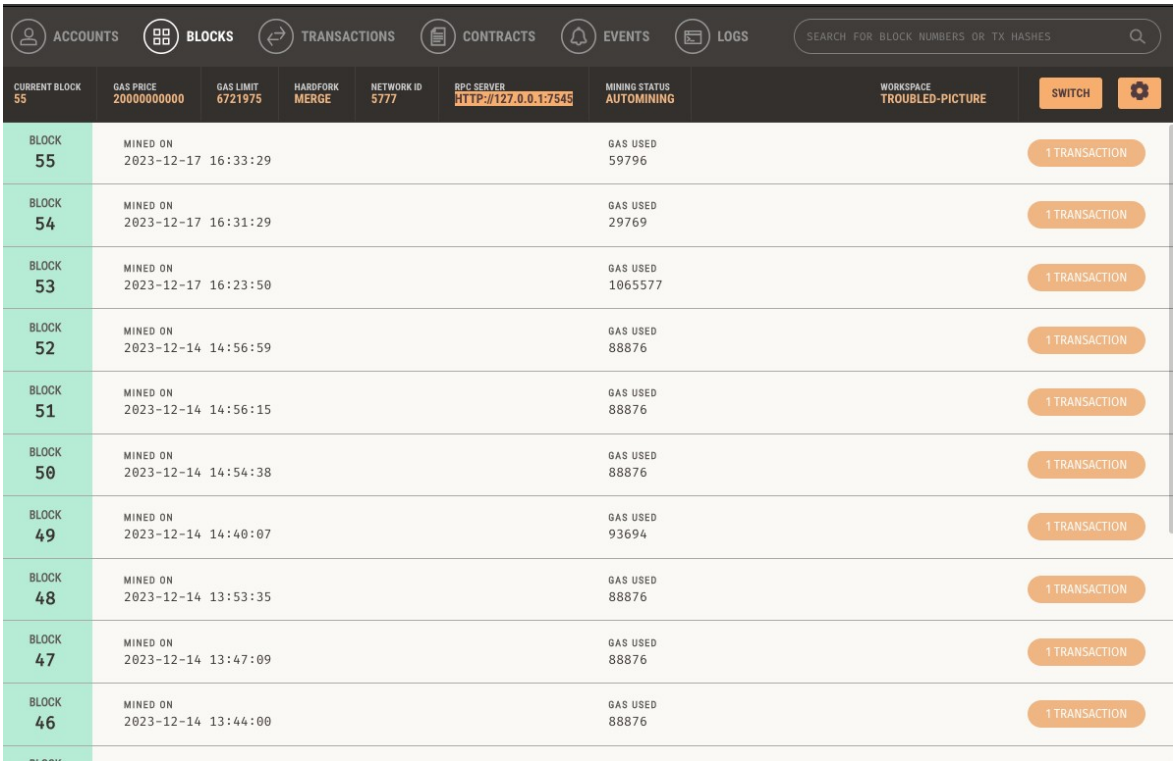
Είναι ένα απλό app και το γραφικό του φαίνεται στην εικόνα 4.22.



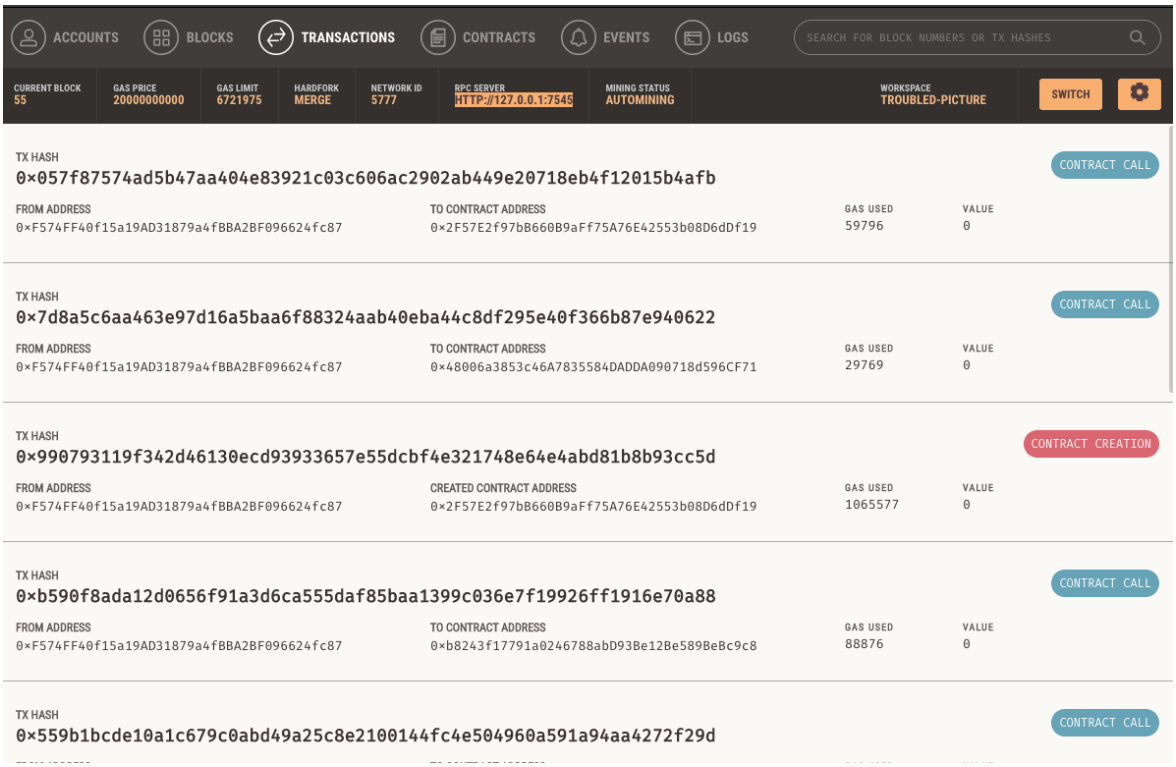

Εικόνα 4.22: Το Ganache

Το local blockchain, όπως φαίνεται, τρέχει locally στο HTTP://127.0.0.1:7545 και το mining των blocks γίνεται αυτόματα.

Στο tab “BLOCKS” φαίνονται όλα τα blocks και στο tab “TRANSACTIONS” όλα τα transactions.

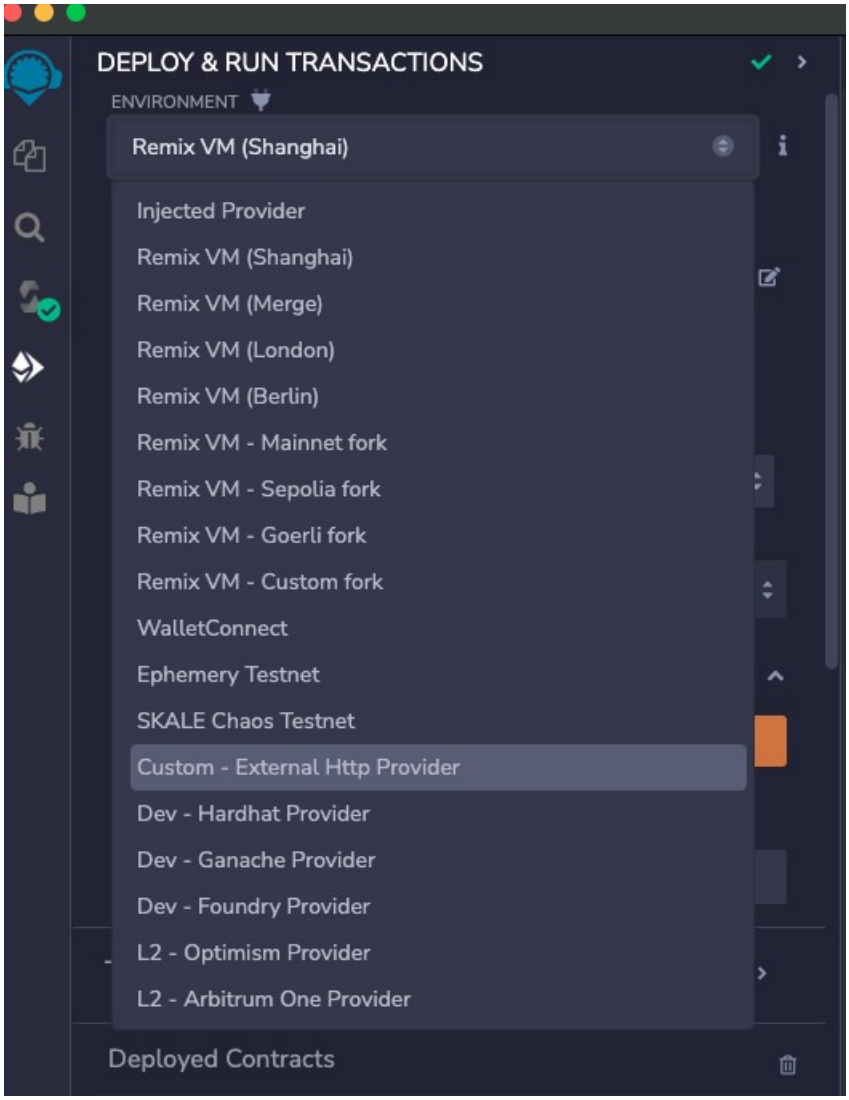



Εικόνα 4.23: Το tab “BLOCKS” του Ganache




Εικόνα 4.24: Το tab “TRANSACTIONS” του Ganache

Για να συνδέσουμε το Remix με το blockchain αυτό πηγαίνουμε στο “Deploy & Run Transactions” και στο Environment επιλέγουμε Custom - External Http Provider, όπως φαίνεται στην εικόνα 4.25.




Εικόνα 4.25: Επιλογή custom - external http provider

Από εκεί προσθέτουμε τη διεύθυνση που τρέχει το Ganache (HTTP://127.0.0.1:7545). Με τον ίδιο τρόπο θα μπορούσαμε να συνδέσουμε ένα RPC ενός κανονικού node που τρέχει σε κάποιο mainnet.

Στην συνέχεια κάνουμε deploy τα contracts κανονικά και αλληλεπιδρούμε με αυτά όπως κάναμε και στο Remix VM.

# Κεφάλαιο 5: Υλοποίηση του Front End

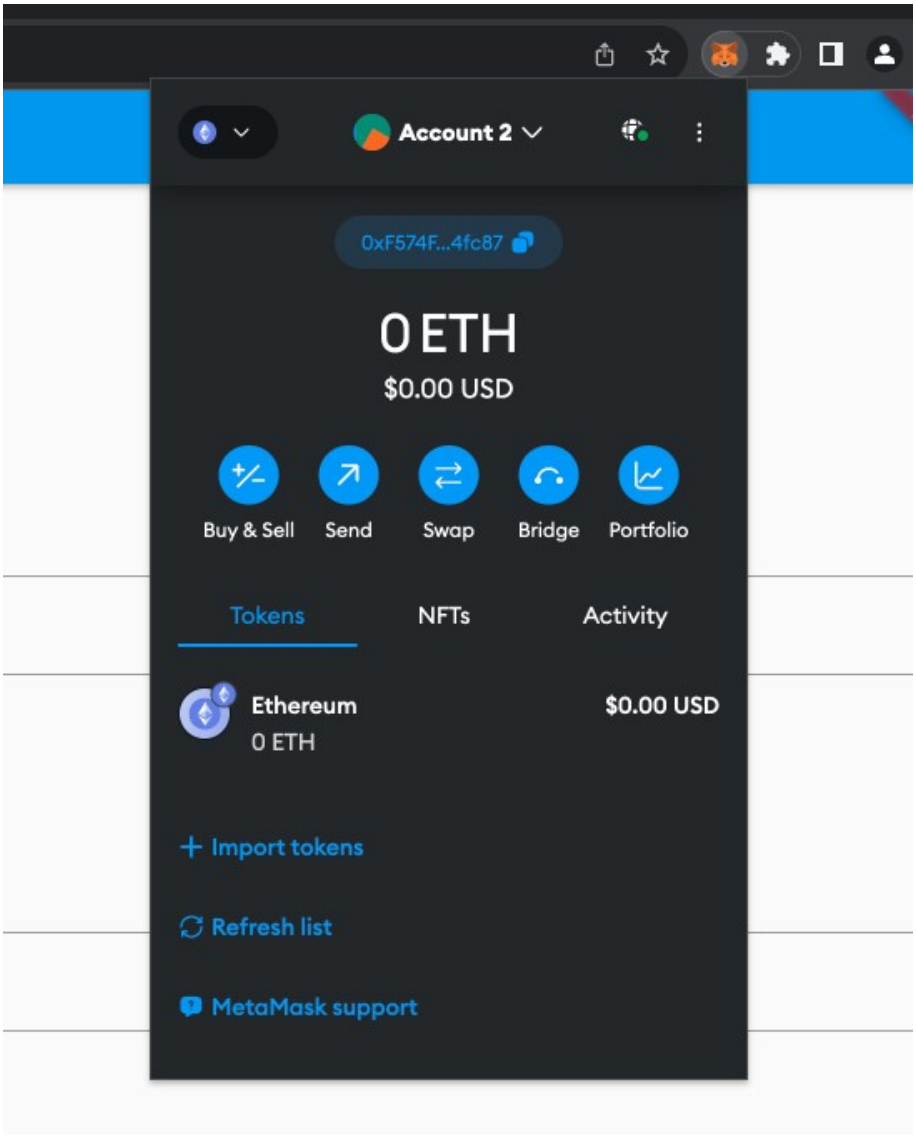
Για το frontend χρησιμοποιήθηκε το Flutter και η γλώσσα προγραμματισμού Dart. Η Web εφαρμογή που υλοποιήθηκε μπορεί να μιλάει με το blockchain και να εκτελεί τις συναρτήσεις του smart contract. Δημιουργήθηκε ένα απλό γραφικό περιβάλλον το οποίο μπορεί να υλοποιει σύνδεση με wallet (MetaMask [21]) και επιτρέπει την προσθήκη ρευστότητας και την εκτέλεση swaps.

## 5.1 Το MetaMask Wallet

Το MetaMask είναι ένα δημοφιλές και ευρέως χρησιμοποιούμενο πορτοφόλι κρυπτονομισμάτων και πύλη σε εφαρμογές blockchain, κυρίως γνωστό για την ενσωμάτωσή του με το blockchain Ethereum. Διαθέσιμο ως browser extention και εφαρμογή για κινητά, το MetaMask επιτρέπει στους χρήστες να αλληλεπιδρούν με το δίκτυο Ethereum, παρέχοντας μια ασφαλή και φιλική προς το χρήστη διεπαφή για τη διαχείριση των κουπονιών Ethereum και ERC20, τη σύνδεση με αποκεντρωμένες εφαρμογές (dApps) και την εκτέλεση λειτουργιών έξυπνων συμβολαίων.

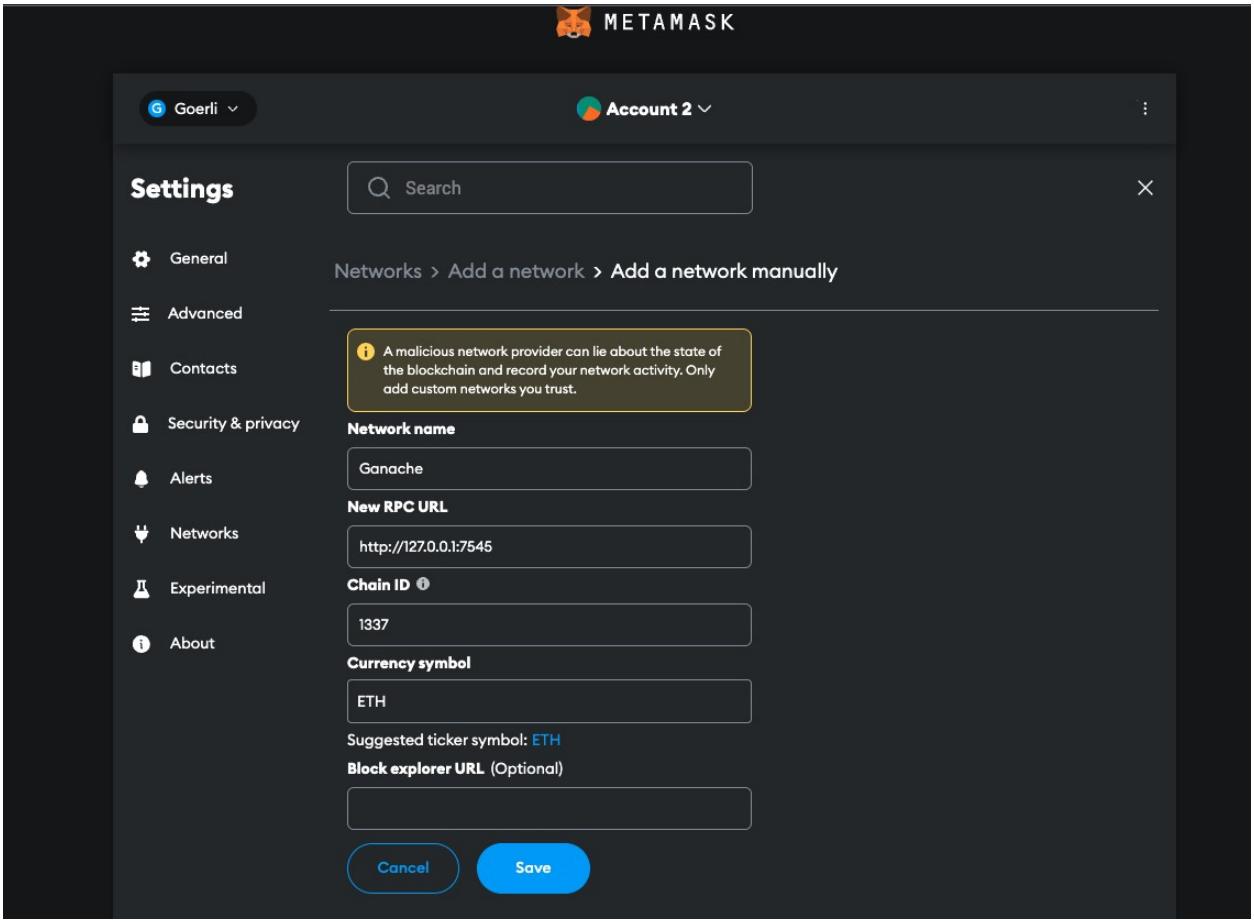
Αυτό που ξεχωρίζει το MetaMask είναι η προσβασιμότητα και η ευκολία χρήσης του, γεγονός που το καθιστά ιδανική επιλογή τόσο για νεοφερμένους όσο και για έμπειρους χρήστες στον χώρο των κρυπτονομισμάτων. Λειτουργεί ως γέφυρα μεταξύ των τυπικών προγραμμάτων περιήγησης ιστού και του blockchain Ethereum, επιτρέποντας στους χρήστες να πραγματοποιούν συναλλαγές Ethereum και να αλληλεπιδρούν με dApps απευθείας από το πρόγραμμα περιήγησής τους χωρίς να εκτελούν έναν πλήρη κόμβο Ethereum. Το MetaMask αποθηκεύει τις διευθύνσεις πορτοφολιού Ethereum και τα ιδιωτικά κλειδιά των χρηστών με ασφάλεια, τα οποία χρησιμοποιούνται για την υπογραφή συναλλαγών και την αλληλεπίδραση με έξυπνα συμβόλαια. Η ενσωμάτωσή του με το Ethereum dApps ήταν καθοριστικής σημασίας για την ευρεία υιοθέτηση διαφόρων εφαρμογών που βασίζονται σε blockchain, συμπεριλαμβανομένων των πλατφορμών Defi, αγορών NFT και άλλων.

Αφού κάνουμε install το MetaMask στον chrome, βλέπουμε το παρακάτω:



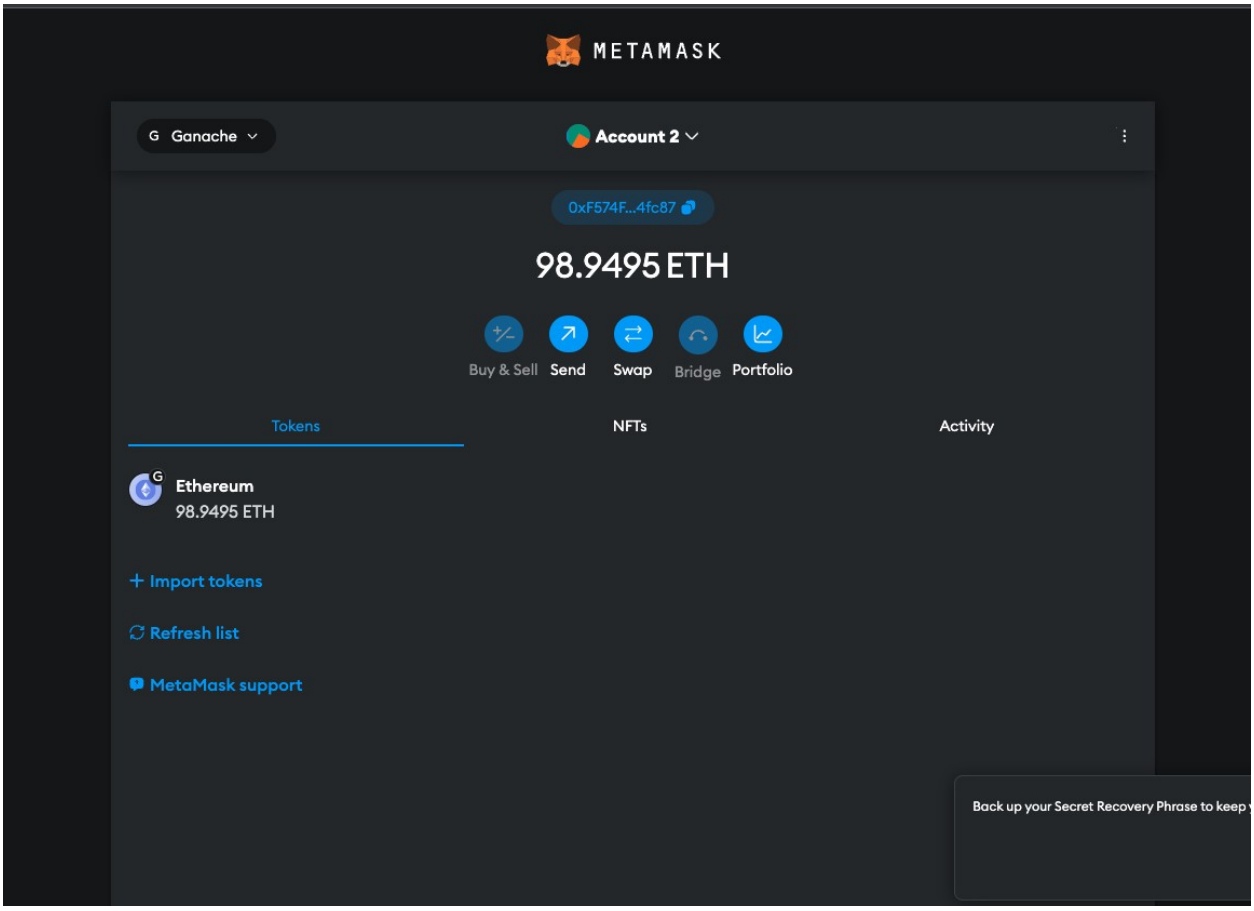

Εικόνα 5.1: Η αρχική οθόνη του MetaMask αφού έχουμε δημιουργήσει το wallet και έχουμε γράψει το seed σε κάποιο χαρτί για να μπορούμε να το ανακτήσουμε.

Το wallet τώρα βλέπει στο Ethereum mainnet. Για να το κάνουμε να βλέπει στο δικό μας blockchain στο Ganache πηγαίνουμε στο “Add Network” και “Add network manually”. Εκεί προσθέτουμε το Ganache όπως φαίνεται στην εικόνα 5.2.



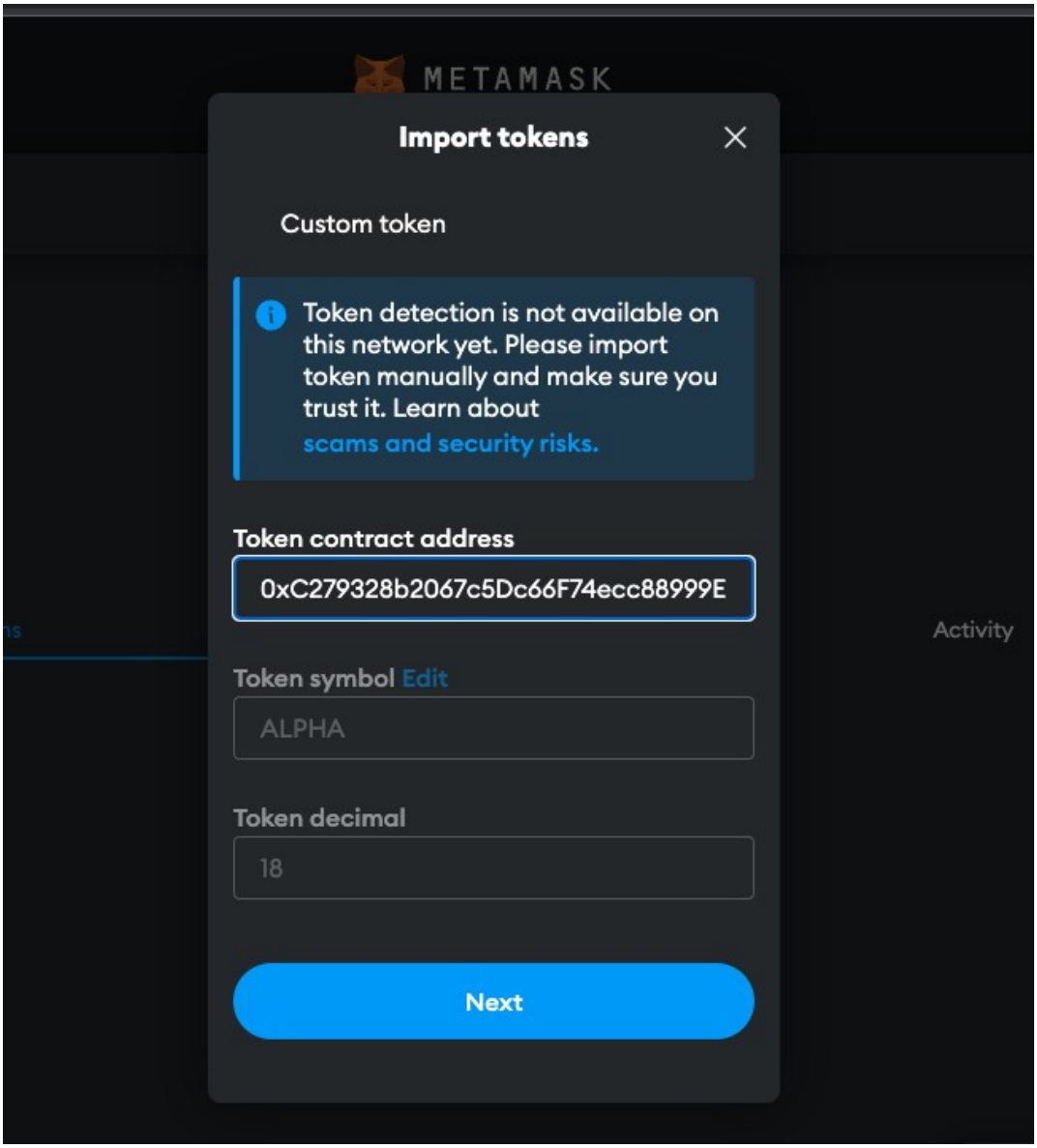

Εικόνα 5.2: Προσθήκη του Ganache στο MetaMask

Αφού το προσθέσουμε πλέον θα λειτουργούμε σαν το πρώτο account του Ganache όπως φαίνεται παρακάτω:




Εικόνα 5.3: Το MetaMask συνδεδεμένο στο Ganache

Αφού κάνουμε deploy τα contracts μας από το Remix μπορούμε να κάνουμε και import tokens όπως φαίνεται στην εικόνα 5.4.




Εικόνα 5.4: Προσθήκη του Alpha Token στο Ganache

## 5.2 Τα βασικά του Flutter Web

Το Flutter, ένα κιτ ανάπτυξης λογισμικού UI ανοιχτού κώδικα που δημιουργήθηκε από την Google, έχει αναδειχθεί ως ένα ισχυρό εργαλείο για τη δημιουργία εγγενών μεταγλωττισμένων εφαρμογών για κινητά, ιστό και επιτραπέζιους υπολογιστές από μια ενιαία βάση κώδικα. [22] Η γλώσσα, Dart, είναι βελτιστοποιημένη για τη δημιουργία γρήγορων, επεκτάσιμων και αντιδραστικών εφαρμογών. Η απόφαση να χρησιμοποιηθεί το Flutter, και συγκεκριμένα το Flutter Web, για την ανάπτυξη του frontend ενός αποκεντρωμένου ανταλλακτηρίου (DEX) βασίζεται σε πολλά από τα συναρπαστικά χαρακτηριστικά και πλεονεκτήματά του:

**Ανάπτυξη πολλαπλών πλατφορμών**

Η ικανότητα του Flutter να μεταγλωττίζει σε εγγενή κώδικα για πολλές πλατφόρμες είναι ένα σημαντικό πλεονέκτημα. Αυτό σημαίνει ότι η ίδια βάση κώδικα μπορεί να χρησιμοποιηθεί για τη δημιουργία μιας ενοποιημένης εμπειρίας χρήστη σε προγράμματα περιήγησης ιστού, κινητές συσκευές, ακόμη και εφαρμογές επιτραπέζιου υπολογιστή, προσφέροντας ευρεία εμβέλεια και προσβασιμότητα στους χρήστες διαφορετικών συσκευών.

**Πλούσιο και προσαρμόσιμο περιβάλλον χρήστη**

Το Flutter είναι γνωστό για το πλούσιο σύνολο προσχεδιασμένων γραφικών στοιχείων και την ικανότητα δημιουργίας πολύπλοκων προσαρμοσμένων στοιχείων διεπαφής χρήστη. Αυτό το καθιστά ιδανική επιλογή για το σχεδιασμό της διεπαφής του DEX, όπου μια διαισθητική, ανταποκρινόμενη και οπτικά ελκυστική διεπαφή είναι ζωτικής σημασίας για την αφοσίωση και την ικανοποίηση των χρηστών.

**Εκτέλεση**

Το Flutter μεταγλωττίζεται σε εγγενή κώδικα ARM και διαθέτει μηχανή απόδοσης υψηλής απόδοσης. Αυτό έχει ως αποτέλεσμα μια ομαλή και γρήγορη εμπειρία χρήστη, η οποία είναι σημαντική για το DEX όπου οι ενημερώσεις σε πραγματικό χρόνο και η ανταπόκριση είναι ζωτικής σημασίας για τις συναλλαγές.

**Γλώσσα Dart:**

Το Dart, η γλώσσα προγραμματισμού που χρησιμοποιείται με το Flutter, προσφέρει λειτουργίες όπως το hot reload για έναν γρήγορο και παραγωγικό κύκλο ανάπτυξης. Είναι επίσης type safe, το οποίο βοηθά στην αποφυγή πολλών κοινών σφαλμάτων κατά τη στιγμή της μεταγλώττισης, ενισχύοντας τη συνολική αξιοπιστία και ασφάλεια της εφαρμογής – μια κρίσιμη πτυχή για μια οικονομική πλατφόρμα όπως το DEX. Επίσης είναι πιο εύκολη και εύχρηστη σε σχέση με Javascript και React

**Ενοποίηση με Smart Contracts και Web3**

Το Flutter Web μπορεί να ενσωματωθεί με τα έξυπνα συμβόλαια Ethereum και το ευρύτερο οικοσύστημα Web3, επιτρέποντας την απρόσκοπτη αλληλεπίδραση με τις λειτουργίες blockchain. Οι βιβλιοθήκες και οι προσθήκες για το Flutter διευκολύνουν αυτές τις ενσωματώσεις, καθιστώντας ευκολότερη τη σύνδεση του frontend του DEX με το δίκτυο Ethereum και τις έξυπνες συμβάσεις για συνδέσεις πορτοφολιού, συναλλαγές και αναζήτηση δεδομένων blockchain.

Μερικά χαρακτηριστικά του Flutter:

**Widgets**: Στον πυρήνα της αρχιτεκτονικής του Flutter βρίσκεται η έννοια των widget. Τα πάντα σε ένα Flutter UI είναι ένα widget—από ένα απλό πλαίσιο κειμένου έως ένα πολύπλοκο κινούμενο σχέδιο. Τα γραφικά στοιχεία μπορούν να συνδυαστούν και να ενσωματωθούν για τη δημιουργία πολύπλοκων διεπαφής χρήστη.

**Rendering Engine**: Το Flutter χρησιμοποιεί τη μηχανή γραφικών Skia, η οποία επανασχεδιάζει τη διεπαφή χρήστη σε κάθε καρέ, συνήθως με ταχύτητα 60 καρέ ανά δευτερόλεπτο. Αυτό δίνει στις εφαρμογές Flutter τα ομαλά και καθαρά γραφικά τους. Όταν αλλάζει η κατάσταση ενός γραφικού στοιχείου, το Flutter επανασχεδιάζει μόνο το υποδέντρο του γραφικού στοιχείου που έχει πραγματικά αλλάξει, βελτιώνοντας την απόδοση.

**Dart Platform:** Το Dart είναι η γλώσσα προγραμματισμού που χρησιμοποιείται για την ανάπτυξη Flutter. Είναι μια σύγχρονη γλώσσα με χαρακτηριστικά όπως η συλλογή just-in-time (JIT) και ahead-of-time (AOT). Το JIT ενισχύει την εμπειρία ανάπτυξης με λειτουργίες όπως το hot reload (επιτρέποντας στους προγραμματιστές να βλέπουν αλλαγές σε πραγματικό χρόνο χωρίς επανεκκίνηση της εφαρμογής), ενώ το AOT χρησιμοποιείται για τη μεταγλώττιση εκδόσεων εκδόσεων, με αποτέλεσμα εξαιρετικά βελτιστοποιημένο και αποτελεσματικό εγγενή κώδικα.

**Flutter Engine**: Γραμμένο κυρίως σε C++, το Flutter Engine είναι υπεύθυνο για την απόδοση χαμηλού επιπέδου χρησιμοποιώντας τη βιβλιοθήκη γραφικών Skia της Google, τη διάταξη κειμένου και την παροχή πρόσθετων για πρόσβαση σε υπηρεσίες συγκεκριμένης πλατφόρμας, όπως κάμερα, Bluetooth και άλλα.

Ένα τυπικό αρχείο Dart σε ένα project Flutter ακολουθεί μια καθαρή και δομημένη μορφή:

```
import 'package:flutter/material.dart';

void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      home: Scaffold(
        appBar: AppBar(
          title: Text('Welcome to Flutter'),
        ),
        body: Center(
```

```
        child: Text('Hello World'),
      ),
    ),
  );
 }
}
```

Imports: Τα αρχεία dart ξεκινούν με δηλώσεις import. Το πακέτο material.dart χρησιμοποιείται συνήθως για εξαρτήματα Material Design.

main Function: Το σημείο εισόδου μιας εφαρμογής Flutter είναι η κύρια λειτουργία, η οποία καλεί το runApp, περνώντας στο γραφικό στοιχείο root.

Stateless και Stateful Widgets: Τα γραφικά στοιχεία είναι είτε Stateless (αμετάβλητη κατάσταση) είτε Stateful (μεταβλητή κατάσταση). Στο παραπάνω παράδειγμα, το MyApp είναι ένα γραφικό στοιχείο χωρίς κατάσταση, το οποίο παρακάμπτει τη μέθοδο δημιουργίας για την περιγραφή της διεπαφής χρήστη.

Η διεπαφή χρήστη ως κώδικας: Η διεπαφή χρήστη περιγράφεται με τη μορφή κώδικα. Τα γραφικά στοιχεία είναι ένθετα το ένα μέσα στο άλλο για να σχηματίσουν την ιεραρχία της διεπαφής χρήστη.

MaterialApp Widget: Αυτό είναι ένα γραφικό στοιχείο ευκολίας που περικλείει έναν αριθμό γραφικών στοιχείων που απαιτούνται συνήθως για εφαρμογές σχεδιασμού υλικού.

Για το integration με το blockchain χρησιμοποιούνται βιβλιοθήκες όπως ή flutter_web3 και η web3dart.

## 5.3 Σύνδεση μέσω MetaMask

Το πρώτο πράγμα που θέλουμε να μπορεί να κάνει ο χρήστης είναι να συνδέσει το Wallet του. Για τον σκοπό αυτό υλοποιήσαμε την παρακάτω κλάση:

```
import 'package:flutter_web3/flutter_web3.dart';

class WalletService {
  static Future<void> connect() async {
    // This method will prompt the MetaMask extension to connect.
    await ethereum!.requestAccount();
  }
```

```
  static Future<String> getAccount() async {
    // Get a list of accounts from MetaMask.
    var accounts = await ethereum!.requestAccount();
    return accounts.first;
  }

  static Future<String> getBalance(String account) async {
    // Get the balance of the given account in wei.
    var balanceWei = await provider!.getBalance(account);
    // Convert wei to ether and format it to a string with appropriate decimal places.
    var balanceEth = balanceWei / BigInt.from(10).pow(18);
    return balanceEth
        .toStringAsFixed(18); // Display up to 18 decimal places of ether.
  }
}
```

Εδώ ορίζονται οι συναρτήσεις για το connect και για το getAccount και getBalance.

Η αρχική οθόνη είναι απλά ένα button όπως φαίνεται στον παρακάτω κώδικα της build της main:

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    appBar: AppBar(
      title: const Text('DEX'),
    ),
    body: Center(
      child: Column(
        mainAxisAlignment: MainAxisAlignment.center,
        children: <Widget>[
          ElevatedButton(
            onPressed: _connectWallet,
            child: const Text('Connect to MetaMask'),
          ),
          const SizedBox(
            height: 20,
          ),
          if (_account.isNotEmpty) ...[
            Text(
              'Account: $_account',
              style: const TextStyle(color: Colors.black),
```

```
            ),
            Text('Balance: $_balance ETH',
              style: const TextStyle(color: Colors.black)),
          ],
        ],
      ),
    ),
  );
}
```

Όταν πατηθεί το κουμπί καλείται η connectWallet.

Μέσω του πακέτου provider, έχουμε επιπλέον δημιουργήσει έναν provider, τον UserDataProvider.

```
import 'package:flutter/material.dart';

class UserDataProvider extends ChangeNotifier {
  String? accountAddress;
  String? accountBalance;

  void setAddress(String address) {
    accountAddress = address;
  }

  void setBalance(String balance) {
    accountBalance = balance;
  }
}
```

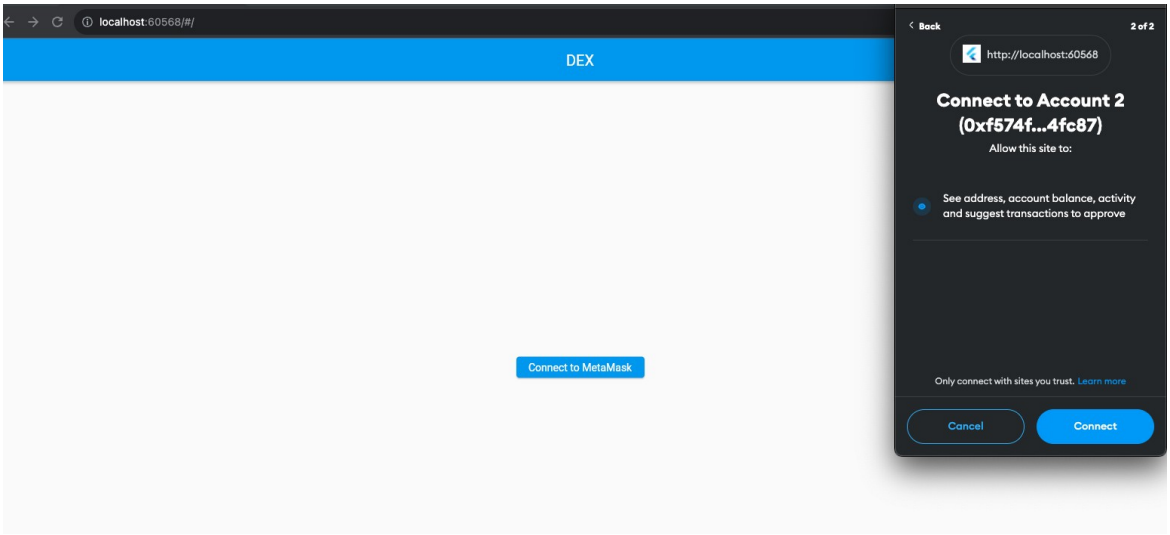
Με τον τρόπο αυτό μπορούμε να έχουμε global μεταβλητές οι οποίες είναι διαθέσιμες σε όλο το app (δηλαδή και σε διαφορετικές οθόνες).

Στη _connectWallet, κάνουμε τη σύνδεση μέσω τη getAccount της WalletService και στη συνέχεια κάνουμε set το address και το balance στον Provider που φτιάξαμε.

```
void _connectWallet() async {
  var account = await WalletService.getAccount();
  print(account);
  var balance = await WalletService.getBalance(account);
  print(balance);
```

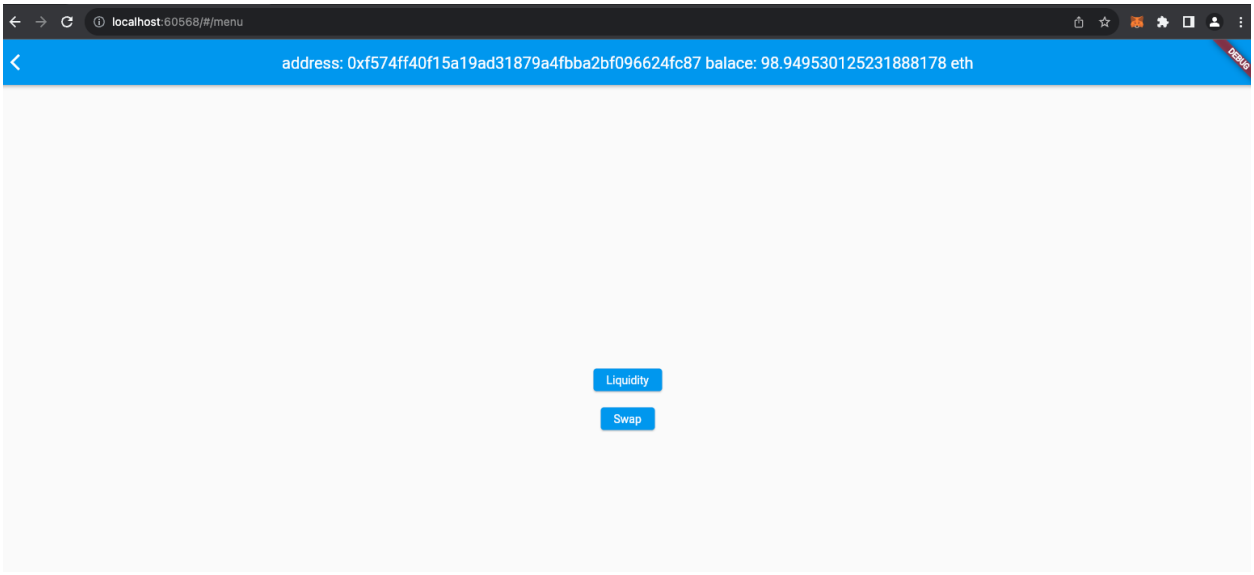
```
  setState(() {
    _account = account;
    _balance = balance;
  });
  // ignore: use_build_context_synchronously
  Provider.of<UserDataProvider>(context, listen: false).setAddress(_account);
  // ignore: use_build_context_synchronously
  Provider.of<UserDataProvider>(context, listen: false).setBalance(_balance);

  // ignore: use_build_context_synchronously
  Navigator.of(context).pushNamed("/menu");
}
```

Αφού πατήσουμε Connect Wallet στο γραφικό θα εμφανιστεί το MetaMask και θα πρέπει να κάνουμε appove.




Εικόνα 5.5: Σύνδεση MetaMask στο app

Στην δεύτερη οθόνη (menu_page.dart) έχουμε ένα μενού επιλογή για τη ρευστότητα ή για το swap. Στο app bar μέσω του provider βλέπουμε το address του wallet μας και το balance.




Εικόνα 5.6: Μενού επιλογής action στο DEX

## 5.4 Blockchain Service

Μια πολύ σημαντική κλάση που υλοποιήθηκε είναι η blockchain service η οποία ουσιαστικά υλοποιεί όλη την επικοινωνία με το blockchain υλοποιώντας τις συναρτήσεις για το swap, το liquidity κτλ.

Η κλάση φαίνεται παρακάτω στο παράρτημα 1.

Αρχικά φορτώνουμε το abi το οποίο έχουμε αποθηκεύσει στο αρχείο “assets/contract_abi.json”. Το ABI (Application Binary Interface) είναι ένα interface το οποίο παίζει το ρόλο της υπογραφής του contract. Κατά την ανάπτυξη μιας αποκεντρωμένης εφαρμογής (DApp), το ABI επιτρέπει στον κώδικα διεπαφής να αλληλεπιδρά με το έξυπνο συμβόλαιο που αναπτύσσεται στο blockchain Ethereum. Εργαλεία όπως το Web3.js ή το ethers.js χρησιμοποιούν το ABI για την κωδικοποίηση κλήσεων συναρτήσεων στον κώδικα χαμηλού επιπέδου που μπορεί να επεξεργαστεί το δίκτυο Ethereum. Αποκωδικοποιούν επίσης τις εξόδους από αυτές τις κλήσεις συναρτήσεων σε αναγνώσιμες μορφές.

Το ABI είναι σε μορφή json και το παίρνουμε από το Remix.

Με αυτό το τρόπο μπορούμε να κάνουμε load ένα contract.

```
final contract = DeployedContract(
      ContractAbi.fromJson(_abiCode, 'LiquidityPool'),
      EthereumAddress.fromHex(_contractAddress));
```

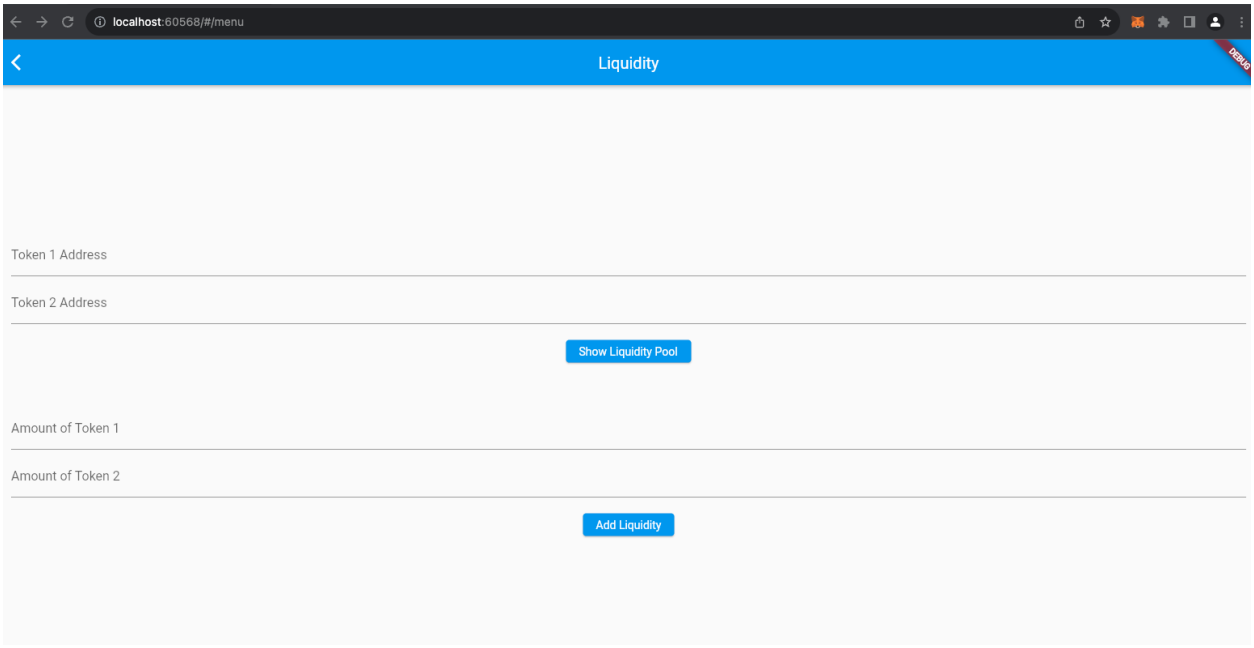
Και να καλέσουμε συναρτήσεις σε αυτό με ορίσματα:

```
await contract
      .send('swap', [tokenIn, tokenOut, amountIn]).then((txHash) async {
    print("===============");
    print("Transaction hash: $txHash");
  }).catchError((error) {
    // Handle error
    print("Error: $error");
  });
```

Στο constants.dart βάζουμε επίσης hardcoded το PoolContractAddress.

## 5.5 Προσθήκη Ρευστότητας

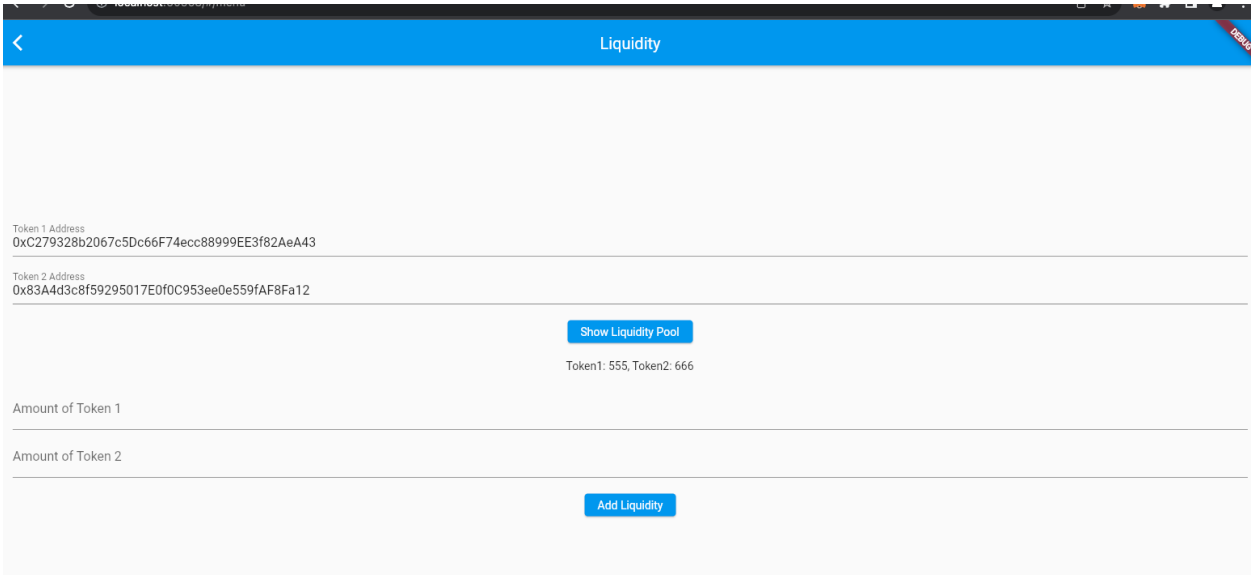
Για τη προσθήκη ρευστότητας έχουμε το παρακάτω page (liquidity_page.dart).




Εικόνα 5.7: Το liquidity page

Η showPooInfo καλεί τη getPoolInfo από το BlockchainService

```
void _showPoolInfo() async {
  String token1Address = _token1Controller.text;
  String token2Address = _token2Controller.text;

  if (token1Address.isNotEmpty && token2Address.isNotEmpty) {
    var info =
        await _blockchainService!.getPoolInfo(token1Address, token2Address);
    setState(() {
      _poolInfo = 'Token1: ${info[0]}, Token2: ${info[1]}';
    });
  }
}
```

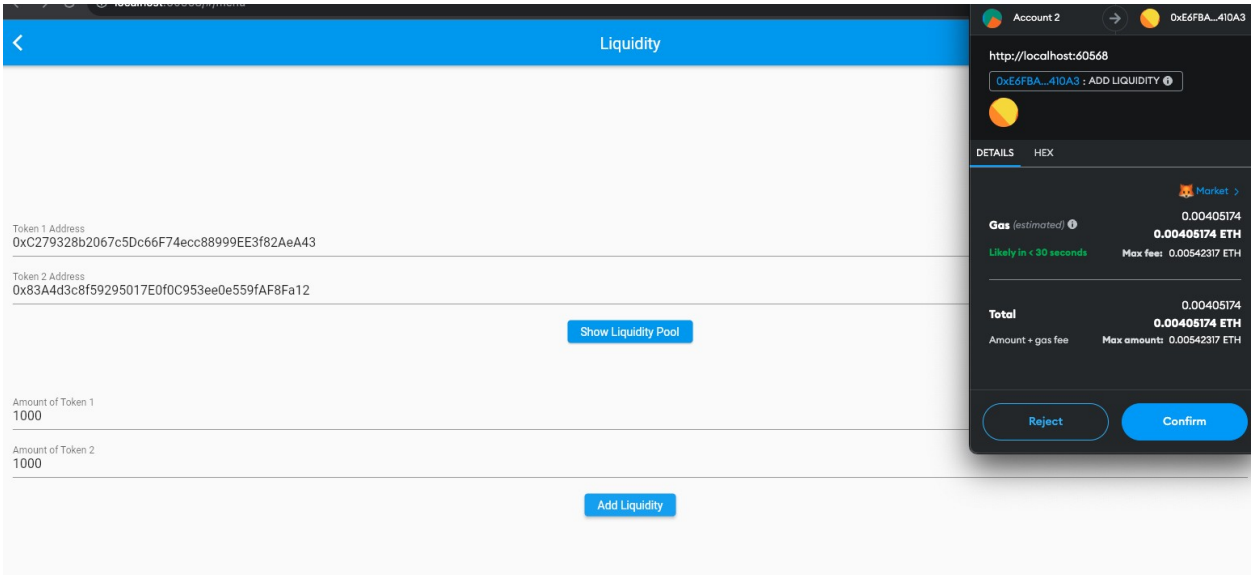



Εικόνα 5.8: Προβολή amount των token στο pool

Η addLiquidity καλεί τον ομώνυμη συνάρτηση της BlockchainService έτσι ώστε να προσθέσουμε ρευστότητα.

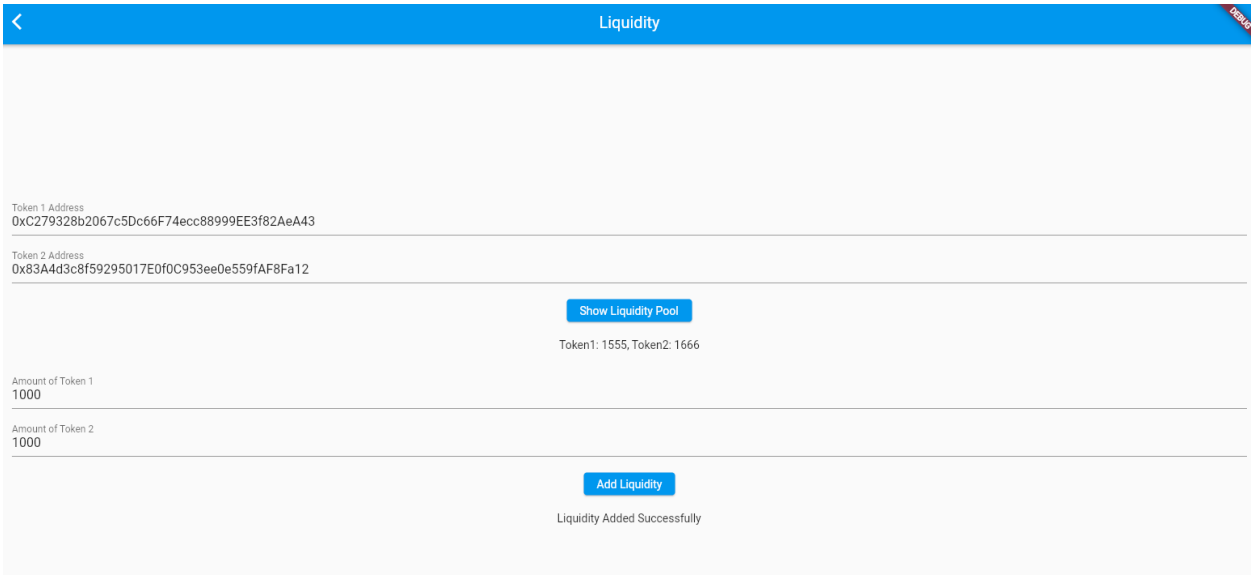
```
void _addLiquidity() async {
  String token1Address = _token1Controller.text;
  String token2Address = _token2Controller.text;
  String amountToken1 = _amountToken1Controller.text;
  String amountToken2 = _amountToken2Controller.text;

  if (token1Address.isNotEmpty &&
      token2Address.isNotEmpty &&
      amountToken1.isNotEmpty &&
      amountToken2.isNotEmpty) {
    try {
      await _blockchainService!.addLiquidity(token1Address, token2Address,
          BigInt.parse(amountToken1), BigInt.parse(amountToken2));
      setState(() {
        _transactionStatus = 'Liquidity Added Successfully';
      });
    } catch (e) {
      setState(() {
        _transactionStatus = 'Failed to Add Liquidity: $e';
      });
    }
  }
}
```

Όταν πατήσουμε AddLiquidity το MetaMask θα εμφανιστεί και θα μας ζητήσει επιβεβαίωση. Στο transaction αυτό θα πληρώσουμε και fees .




Εικόνα 5.9: Επιβεβαίωση προσθήκης ρευστότητας

Μετά τη προσθήκη ρευστότητας θα δούμε ότι το pool έχει αλλάξει:



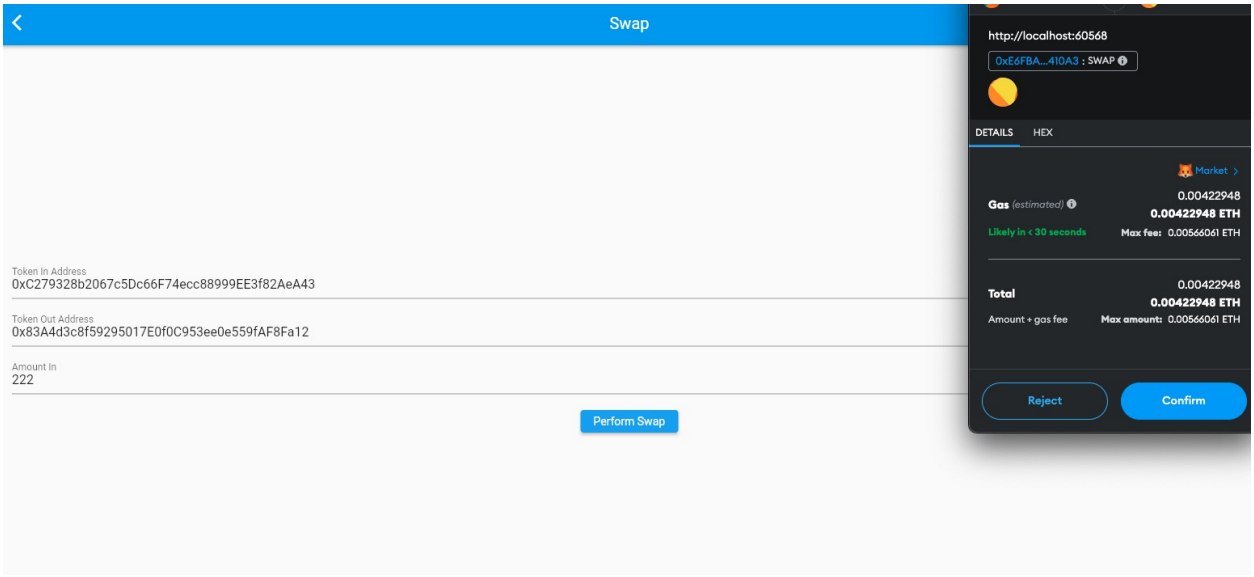

Εικόνα 5.10: Η ρευστότητα προστέθηκε

## 5.6 Εκτέλεση και Δοκιμή Swap

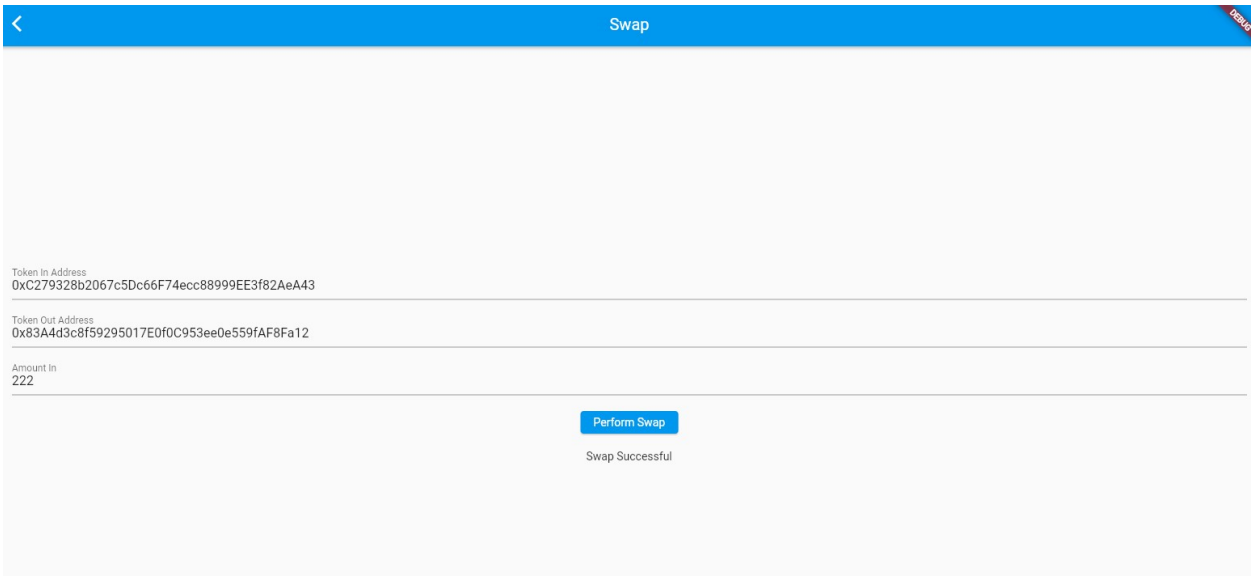
Στο swap (swap_page.dart) έχουμε αντίστοιχα ένα απλό περιβάλλον. Η συνάρτηση που εκτελεί το swap είναι η παρακάτω:

```
void _performSwap() async {
  String tokenIn = _tokenInController.text;
  String tokenOut = _tokenOutController.text;
  String amountInStr = _amountInController.text;
  BigInt amountIn = BigInt.tryParse(amountInStr) ?? BigInt.zero;

  if (tokenIn.isNotEmpty && tokenOut.isNotEmpty && amountIn != BigInt.zero) {
    try {
      // Call the swap function from BlockchainService
      await _blockchainService!.swapTokens(tokenIn, tokenOut, amountIn);
      setState(() {
        _swapStatus = 'Swap Successful';
      });
    } catch (e) {
      setState(() {
        _swapStatus = 'Failed to Swap: $e';
      });
    }
  }
}
```

Επιβεβαιώνουμε το transaction με το MetaMask.



Εικόνα 5.11: Επιβεβαίωση transaction μέσω MetaMask




Εικόνα 5.12: Το Swap ολοκληρώθηκε

# Κεφάλαιο 6: Συμπεράσματα και Μελλοντικές Εργασίες

Αυτή η εργασία ασχολήθηκε πολύπλευρο τοπίο της τεχνολογίας blockchain και των αποκεντρωμένων ανταλλαγών (DEX), φωτίζοντας τις περίπλοκες λειτουργίες, τις προκλήσεις και τις καινοτομίες σε αυτόν τον τομέα. Ξεκινήσαμε με μια εξερεύνηση της ιστορίας και της εξέλιξης του blockchain, εμβαθύνοντας στη μηχανική του τρόπου λειτουργίας των blockchain και στον κεντρικό ρόλο που διαδραματίζουν στην υποστήριξη των DEX. Μια κριτική ανάλυση της τρέχουσας κατάστασης των DEX αποκάλυψε τη σημασία τους στο σημερινό χρηματοπιστωτικό οικοσύστημα και τις προκλήσεις ασφαλείας που αντιμετωπίζουν.

Η πρακτική πτυχή αυτής της εργασίας υλοποιήθηκε μέσω της ανάπτυξης ενός έξυπνου συμβολαίου DEX, που καταδεικνύει την εφαρμογή των δυνατοτήτων έξυπνων συμβολαίων του Ethereum. Η βήμα προς βήμα διαδικασία υλοποίησης, ξεκινώντας από τη δημιουργία διακριτικών έως την περίπλοκη λειτουργία των δεξαμενών ρευστότητας και των μηχανισμών είσπραξης fees, παρείχε πρακτικές πληροφορίες για τις λειτουργίες DEX. Αυτό συμπληρώθηκε από την ανάπτυξη μιας διεπαφής front-end χρησιμοποιώντας Flutter, η οποία έδειξε την ενσωμάτωση της τεχνολογίας blockchain σε εφαρμογές που αντιμετωπίζουν οι χρήστες.

Καθ' όλη τη διάρκεια, η εστίαση παρέμεινε σταθερή στις προκλήσεις ασφαλείας που ενυπάρχουν στα DEX. Αναλύοντας διάφορα θέματα ασφάλειας και αξιολογώντας τις υπάρχουσες λύσεις, η διατριβή υπογραμμίζει την κρίσιμη ανάγκη για συνεχή καινοτομία σε αυτόν τον τομέα.

Από αυτή την εργασία προκύπτουν αρκετοί δρόμοι για μελλοντική εργασία και έρευνα:

**Προηγμένα πρωτόκολλα ασφαλείας**: Η συνεχής έρευνα για πιο ισχυρά και εξελιγμένα πρωτόκολλα ασφαλείας για DEX είναι επιτακτική. Αυτό περιλαμβάνει την εξερεύνηση νέων κρυπτογραφικών μεθόδων και προηγμένων τεχνικών ελέγχου έξυπνων συμβολαίων.

**Λειτουργικότητα Cross-Chai**n: Καθώς το οικοσύστημα blockchain γίνεται πιο διασυνδεδεμένο, η ανάπτυξη DEX με απρόσκοπτες δυνατότητες cross-chain θα είναι ένα σημαντικό βήμα προς τα εμπρός. Αυτό περιλαμβάνει έρευνα για πρωτόκολλα διαλειτουργικότητας και διαχείριση περιουσιακών στοιχείων πολλαπλών αλυσίδων.

**Λύσεις επεκτασιμότητας**: Με την αυξανόμενη δημοτικότητα των DEX, η επεκτασιμότητα παραμένει πρόκληση. Η μελλοντική εργασία μπορεί να εξερευνήσει λύσεις κλιμάκωσης layer 2 και τεχνικές διαμοιρασμού blockchain για τη βελτίωση της απόδοσης των συναλλαγών και τη μείωση του κόστους.

**Σχεδιασμός εμπειρίας χρήστη (UX):** Απαιτείται περαιτέρω εργασία για τη βελτίωση του UX του DEX, καθιστώντας το πιο προσιτό και εύχρηστο για μια ευρύτερη βάση χρηστών. Αυτό περιλαμβάνει τεχνικές βελτιώσεις και μελέτες σχεδιασμού με επίκεντρο τον χρήστη.

Συγκριτικά το DEX που δημιουργήθηκε με κάποιο πραγματικό DEX όπως το Uniswap υπερτερεί σε θέματα ασφάλειας όπως :

**Αντιμετώπιση Front-Running**: Η Uniswap χρησιμοποιεί τον μηχανισμό Constant Product Market Maker, ο οποίος μειώνει τον κίνδυνο των επιθέσεων front-running.

**Εποπτεία Flash Loans**: Η Uniswap δεν είναι απόλυτα ανθεκτική στις επιθέσεις flash loan, αλλά οι χρήστες μπορούν να χρησιμοποιήσουν λογισμικά προστασίας για να μειώσουν τον κίνδυνο.

**Επαλήθευση Ταυτότητας**: Η Uniswap δεν εφαρμόζει αυστηρούς έλεγχους ταυτότητας, καθώς είναι αποκεντρωμένο, αλλά μπορεί να ενισχύσει τους έλεγχους μέσω της οικονομικής διαχείρισης.

**Ασφαλής Σχεδιασμός Συμβολαίων**: Η Uniswap επικεντρώνεται στον ασφαλή σχεδιασμό των έξυπνων συμβολαίων της για να αποτρέψει πιθανά προβλήματα εκτέλεσης.

**Καλή Διαχείριση Καταστάσεων Συναλλαγών**: Οι αλγόριθμοι της Uniswap εξασφαλίζουν τη σωστή διαχείριση των καταστάσεων συναλλαγών για να αποφευχθούν προβλήματα.

Παρόλα αυτά, κανένα DEX δεν είναι απόλυτα ασφαλές, και η ασφάλεια παραμένει σημαντική πρόκληση στον τομέα των αποκεντρωμένων ανταλλαγών.

Συνοπτικά, αυτή η εργασία συνέβαλε στην κατανόηση του blockchain και των αποκεντρωμένων ανταλλαγών, τονίζοντας τις δυνατότητές τους, τις προκλήσεις και το πεδίο για μελλοντικές καινοτομίες. Η διασταύρωση τεχνολογίας, finance και ασφάλειας στη σφαίρα των DEX προσφέρει ένα γόνιμο έδαφος για μελλοντική έρευνα και ανάπτυξη, υπόσχεται να αλλάξει το οικονομικό τοπίο τα επόμενα χρόνια.

## Αναφορές

[1] Nakamoto, S. (2009) 'Bitcoin: A Peer-to-Peer Electronic Cash System'.

[2] Zheng, Z., Xie, S., Dai, H.N., Chen, X. and Wang, H. (2018) 'Blockchain challenges and opportunities: A survey', International Journal of Web and Grid Services, 14(4), pp. 352-375.

[3] Zheng, Z., Xie, S., Dai, H.N., Chen, W., Chen, X., Weng, J. and Imran, M. (2020) 'An overview on smart contracts: Challenges, advances and platforms', Future Generation Computer Systems, 105, pp. 475-491.

[4] Lehar, A. and Parlour, C.A. (2021) 'Decentralized exchanges', SSRN 3905316.

[5] Johnson, K.N. (2020) 'Decentralized finance: Regulating cryptocurrency exchanges', William & Mary Law Review, 62, p. 1911.

[6] Aspris, A., Foley, S., Svec, J. and Wang, L. (2021) 'Decentralized exchanges: The “wild west” of cryptocurrency trading', International Review of Financial Analysis, 77, p. 101845.

[7] Liu, D. and Camp, L.J. (2006) 'Proof of Work can Work', in Proceedings of WEIS, June.

[8] Tikhomirov, S. (2018) 'Ethereum: state of knowledge and research perspectives', in Foundations and Practice of Security: 10th International Symposium, FPS 2017, Nancy, France, October 23-25, 2017, Revised Selected Papers 10, Springer International Publishing, pp. 206-221.

[9] Vashchuk, O. and Shuwar, R. (2018) 'Pros and cons of consensus algorithm proof of stake. Difference in the network safety in proof of work and proof of stake', Electronics and Information Technologies, 9(9), pp. 106-112.

[10] Zheng, Z., Xie, S., Dai, H., Chen, X. and Wang, H. (2017) 'An overview of blockchain technology: Architecture, consensus, and future trends', in 2017 IEEE International Congress on Big Data (BigData Congress), June, IEEE, pp. 557-564.

[11] Sobti, R. and Geetha, G. (2012) 'Cryptographic hash functions: A review', International Journal of Computer Science Issues (IJCSI), 9(2), p. 461.

[12] Gupta, S. and Sadoghi, M. (2021) 'Blockchain transaction processing', arXiv preprint arXiv:2107.11592.

[13] Gervais, A., Karame, G.O., Wüst, K., Glykantzis, V., Ritzdorf, H. and Capkun, S. (2016) 'On the security and performance of proof of work blockchains', in Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, October, pp. 3-16.

[14] Vashchuk, O. and Shuwar, R. (2018) 'Pros and cons of consensus algorithm proof of stake. Difference in the network safety in proof of work and proof of stake', Electronics and Information Technologies, 9(9), pp. 106-112.

[15] Ledger Academy (n.d.) 'Ethereum Proof-of-Stake (PoS) Explained'. Available at: https://www.ledger.com/academy/ethereum-proof-of-stake-pos-explained

[16] Gemini (n.d.) 'The DAO Hack & MakerDAO'. Available at: The DAO: What Was the DAO Hack? | Gemini

[17] ConsenSys (n.d.) 'What is EIP-1559? How Will It Change Ethereum?'. Available at: What is EIP-1559? How Will It Change Ethereum? | Consensys

[18] Ultrasound Money (n.d.) Available at: https://ultrasound.money/

[19] Remix Project (n.d.) Available at: https://remix-project.org/

[20] TruffleSuite (n.d.) 'Ganache'. Available at: https://trufflesuite.com/ganache/

[21] MetaMask (n.d.) Available at: https://metamask.io/

[22] Flutter (n.d.) Available at: https://flutter.dev/

[23] investopedia Available at: https://www.investopedia.com/terms/1/51-attack.asp

## Παράρτημα Α : Blockchain Service

```
import 'dart:io';

import 'package:flutter_web3/flutter_web3.dart';
import 'package:http/http.dart';
import 'package:web3dart/web3dart.dart';
import 'package:flutter/services.dart' show rootBundle;
import 'package:path/path.dart' show join, dirname;
import 'dart:convert';

import 'constants.dart';

class BlockchainService {
  final String _rpcUrl = 'http://127.0.0.1:7545'; // Your Ganache RPC URL
  final String _contractAddress = PoolContractAdress;
  String _abiCode = '';

  late Web3Client _client;
  late DeployedContract _contract;

  BlockchainService() {
    _client = Web3Client(_rpcUrl, Client());
    listenToEvents();
    //_loadAbi();
  }

  Future<void> loadAbi() async {
    final String res = await rootBundle.loadString('assets/contract_abi.json');
    print(res);
    _abiCode = res;
    print(_abiCode);
    _contract = await _loadContract();
  }

  Future<DeployedContract> _loadContract() async {
    final contract = DeployedContract(
        ContractAbi.fromJson(_abiCode, 'LiquidityPool'),
        EthereumAddress.fromHex(_contractAddress));
    return contract;
  }
```

```
  Future<List<dynamic>> getPoolInfo(String token1, String token2) async {
    print("_THE CONTRACT_");
    print(_contract.address);
    print(_contract.functions);
    for (int i = 0; i < _contract.functions.length; i++) {
      print(_contract.functions[i].name);
    }
    print(_contract.abi);
    final poolInfoFunction = _contract.function('tokenPools');

    final response = await _client.call(
        contract: _contract,
        function: poolInfoFunction,
        params: [
          EthereumAddress.fromHex(token1),
          EthereumAddress.fromHex(token2)
        ]);
    return response;
  }

  Future<void> addLiquidity(String token1, String token2, BigInt amountToken1,
      BigInt amountToken2) async {
    Contract contract =
        Contract(_contractAddress, _abiCode, provider!.getSigner());
    await contract.send('addLiquidity',
        [token1, token2, amountToken1, amountToken2]).then((txHash) async {
      print("===============");
      print("Transaction hash: $txHash");
    }).catchError((error) {
      // Handle error
      print("Error: $error");
    });
  }

  Future<void> swapTokens(
      String tokenIn, String tokenOut, BigInt amountIn) async {
    Contract contract =
        Contract(_contractAddress, _abiCode, provider!.getSigner());
    await contract
        .send('swap', [tokenIn, tokenOut, amountIn]).then((txHash) async {
      print("===============");
      print("Transaction hash: $txHash");
    }).catchError((error) {
      // Handle error
```

```
      print("Error: $error");
    });
  }

  void listenToEvents() {
    // Listen for LiquidityAdded event
    ethereum!.on('LiquidityAdded',
        (provider, token1, token2, amountToken1, amountToken2) {
      // Handle the event data here
      print('Liquidity added by $provider for tokens $token1 and $token2');
    });
  }
}
```