Міністерство освіти і науки України
Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)
Кафедра Інформатики
(повна назва)

КВАЛІФІКАЦІЙНА РОБОТА
Пояснювальна записка

рівень вищої освіти перший (бакалаврський)

РОЗРОБЛЕННЯ ЕЛЕКТРОННОГО ГАМАНЦЯ ДЛЯ УПРАВЛІННЯ КРИПТОВАЛЮТНИМИ АКТИВАМИ
(тема)

Виконав:
студент 4 курсу, групи ІТІНФ-19-1

Балагуш І.М.
(прізвище, ініціали)

Спеціальності 122 Комп'ютерні науки
(код і повна назва спеціальності)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

Керівник доц. Тітова О.В.
(посада, прізвище, ініціали)

Допускається до захисту

Зав. кафедри _____ Кобилін О.А.
(підпис) (прізвище, ініціали)

2023 р.

Харківський національний університет радіоелектроніки

Факультет Інформаційно-аналітичних технологій та менеджменту
(повна назва)

Кафедра Інформатики
(повна назва)

Рівень вищої освіти перший (бакалаврський)

Спеціальність 122 Комп'ютерні науки
(код і повна назва)

Тип програми освітньо-професійна

Освітня програма Інформатика
(повна назва освітньої програми)

ЗАТВЕРДЖУЮ:
Зав. кафедри _____
(підпис)
«____»_____ 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

студентові Балагушу Івану Мар'яновичу
(прізвище, ім'я, по батькові)

1. Тема роботи Розроблення електронного гаманця для управління криптовалютними активами

затверджена наказом університету від 15 травня 2023 року № 474 Ст

2. Термін подання студентом роботи до екзаменаційної комісії 02 червня 2023 р.

3. Вихідні дані до роботи науково-методична та науково-технічна література, матеріали конференцій, дані інтернет-мережі мова програмування Golang, середовище розробки JetBrains GoLand.

4. Перелік питань, що потрібно опрацювати в роботі
1. Опис основних принципів здійснення транзакцій з криптовалютою.
2. Огляд існуючих криптогаманців.
3. Аналіз технологій для створення криптогаманця.
4. Вибір бази даних для проєкту.
5. Розробка електронного гаманця для управління криптовалютними активами.

5. Перелік графічного матеріалу із зазначенням креслеників, схем, плакатів, комп'ютерних ілюстрацій (п.5 включається до завдання за рішенням випускової кафедри) Актуальність проблеми обробки зображень, постановка задачі, тестові зображення.

6. Консультанти розділів роботи (п.6 включається до завдання за наявності консультантів згідно з наказом, зазначеним у п.1)

Найменування розділу	Консультант (посада, прізвище, ім'я, по батькові)	Позначка консультанта про виконання розділу	
		підпис	дата
Консультант з дотримання діючих стандартів та норм	Доцент Творошенко І.С.		

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Терміни виконання етапів роботи	Примітка
1	Отримання завдання на кваліфікаційну роботу	10.04.2023	
2	Аналіз завдання, підбір літератури	11.04.23-17.04.23	
3	Аналіз літератури з досліджуваної проблеми	18.04.23-20.04.23	
4	Аналіз технічних засобів	21.04.23-30.04.23	
5	Проєктування системи	01.05.23-13.05.23	
6	Програмна реалізація	14.05.23-24.05.23	
7	Оформлення пояснювальної записки	25.05.23-31.05.23	
8	Перевірка на плагіат	01.06.23	
9	Рецензування	02.06.23	
10	Підготовка презентації та доповіді	02.06.23-04.06.23	
11	Занесення роботи в електронний архів	06.06.23	
12	Попередній захист кваліфікаційної роботи	11.06.23	

Дата видачі завдання 10 квітня 2023 р.

Студент _____
(підпис)

Керівник роботи _____ доц. Тітова О.В.
(підпис) (посада, прізвище, ініціали)

РЕФЕРАТ/ABSTRACT

Пояснювальна записка до кваліфікаційної роботи містить: 53 с., 21 рис., 32 джерела.

ВЕБЗАСТОСУНОК, СУБД, РЕЛЯЦІЙНА БАЗА ДАНИХ, СУТНІСТЬ, НОРМАЛІЗАЦІЯ ДАНИХ, МОВА ЗАПИТІВ SQL, СУБД POSTGRESQL, ПО DOCKER, СЕРЕДОВИЩЕ РОЗРОБКИ GOLAND IDE, МОВА ПРОГРАМУВАННЯ GOLANG.

Об’єктом роботи є розробка криптогаманця на основі мови програмування Golang та фреймворку ReactJS.

Метою роботи є розробка програмного забезпечення, яке надає безпечне зберігання та управління криптовалютними активами з використанням.

Тема кваліфікаційної роботи – використання технологій Golang та Postgres для розробки застосунку «електронний гаманець для управління криптовалютними активами» Ця робота передбачає реалізацію «Криптогаманця» з використанням технології Golang та створення бази даних у середовищі Postgres. Робота виконана на мові програмування Golang, та з застосуванням інструменту Golang.

WEB APPLICATION, SUBDATABASE, RELATIONAL DATABASE, ENTITY, DATA NORMALISATION, SQL QUERY LANGUAGE, POSTGRESQL SUBDATABASE, DOCKER, GOLAND IDE DEVELOPMENT ENVIRONMENT, GOLANG PROGRAMMING LANGUAGE.

The object of work is the development of a crypto wallet based on the Golang programming language and ReactJS framework.

The aim is to develop software that provides secure storage and management of cryptocurrency assets using.

The theme of the qualification work is the use of Golang and Postgres technologies to develop an application «electronic wallet for managing cryptocurrency assets» This work involves the implementation of the «Crypto Wallet» using Golang technology and the creation of a database in the Postgres environment. The work was performed in the Golang programming language and using the Golang tool.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ETH – Ethereum

BTC – Bitcoin

Polygon (MATIC) – криптовалюта та технологічна платформа, яка була запущена для підключення та розвитку проєктів і блокчейнів, сумісних з Ethereum

API – Application Programming Interface

REST – Representational State Transfer

ООП – об’єктно-орієнтоване програмування

HTTP – HyperText Transfer Protocol

ПЗ – програмне забезпечення

ВСТУП

З кожним роком популярність використання криптовалюти росте, як і попит на безпечні криптовалютні гаманці. Люди використовують криптовалюту з різних причин. Криптовалюта – це цифровий актив, який використовує криптографію для забезпечення безпеки транзакцій та контролю за створенням нових одиниць. Криптовалюти можуть бути використані як засіб обміну та збереження вартості, так само як і звичайні валюти, але вони мають ряд особливостей, які роблять їх унікальними. Вони працюють на базі технології блокчейн, яка дозволяє записувати транзакції у розподіленій базі даних, забезпечуючи безпеку та непідробленість операцій. Криптовалюти не контролюються центральними органами управління, такими як уряди чи банки, і можуть бути використані у будь-якій частині світу без обмежень.

До переваг використання криптовалюти можна віднести анонімність, тобто криптовалюта дозволяє людям надсилати та отримувати платежі без розкриття своєї особистої інформації. Це особливо важливо для тих, хто хоче зберегти конфіденційність своїх фінансових операцій. Крім того, перевагою використання криптовалюти є зручність, бо вона дозволяє швидко та легко здійснювати перекази без необхідності проходити через посередників, таких як банки. Це може бути особливо зручним для міжнародних переказів. Криптовалюта може бути використана для інвестування та заробітку грошей. Багато людей купують криптовалюту, сподіваючись на те, що її вартість у майбутньому зросте, що призведе до прибутку.

Найвідомішою криптовалютою є біткоїн, але на сьогоднішній день існує безліч інших криптовалют, таких як etherium, ripl, tether та багато інших. Криптовалюти можуть бути куплені та продані на спеціальних біржах, а також використані для здійснення платежів в інтернеті. Криптогаманці дозволяють користувачам відстежувати свої баланси та історію транзакцій. Криптогаманці можуть бути онлайн-сервісами, програмами для смартфонів або настільними програмами. Отже, криптогаманець потрібен для зберігання, відправлення та

отримання криптовалюти, а також для забезпечення безпеки та управління криптовалютними засобами.

Загалом, предметом дослідження є створення ефективного та безпечного криптовалютного гаманця на мові програмування Golang, який може бути використаний в різних галузях, що працюють з криптовалютами.

Галузі застосування криптовалютного гаманця є дуже різноманітними. Ось декілька прикладів:

– фінансові послуги: криптовалютний гаманець може використовуватися в банках та інших фінансових установах для зберігання та обробки транзакцій з криптовалютами;

– торгівля: криптовалютний гаманець може бути корисним для трейдерів, які займаються купівлею та продажами криптовалют на різних біржах;

– інтернет-магазини: криптовалютний гаманець може використовуватися для приймання платежів у криптовалюті в інтернет-магазинах;

– туризм: криптовалютний гаманець може бути використаний для оплати подорожей, бронювання готелів та інших послуг, пов’язаних з туризмом;

– благодійність: криптовалютний гаманець може допомогти в зборі коштів для благодійних організацій, що працюють з криптовалютами.

Тож, криптовалютний гаманець може знайти застосування в багатьох галузях, де використовуються криптовалюти. Він може допомогти в зберіганні та обробці транзакцій з криптовалютами, що робить його важливим інструментом для будь-якої компанії чи організації, що працює з цими цифровими активами.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Основні принципи здійснення транзакцій з криптовалютою

За своєю суттю криптовалюта – це, як правило, децентралізовані цифрові гроші, призначені для використання в Інтернеті. Біткойн, який був запущений у 2008 році, був першою криптовалютою, і вона залишається найбільшою, найвпливовішою та найвідомішою. Біткойн та інші криптовалюти, такі як Ethereum, стали цифровою альтернативою грошам, випущеним урядами [1].

Криптовалюта дає змогу передавати цінності в Інтернеті без посередників, таких як банк або платіжний процесор, що дозволяє передавати цінності по всьому світу, майже миттєво, 24/7, за низькі комісії. Найважливіше те, що криптовалюти дозволяють людям отримати повний контроль над своїми активами. Хоча як криптовалюту, так і фіатну валюту можна використовувати як засіб обміну на товари та послуги, спосіб, яким ці транзакції насправді працюють на базовому рівні, насправді зовсім інший.

Коли людина ініціює банківський переказ через традиційну фінансову установу, кошти переміщуються з одного рахунку на інший. З іншого боку, криптовалюта не зберігається в банку, як фіатна валюта, тобто гроші, які ми використовуємо у повсякденному житті, такі як банкноти та монети на вашому поточному рахунку. Натомість криптовалюта «живе» на блокчейні. Блокчейн діє як загальнодоступна книга кожної транзакції, яка коли-небудь була зроблена. Будь-які транзакції, здійснені в криптовалюті, просто переміщують дані між адресами блокчейна. Це означає, що криптофонди ніколи фізично не переходять із рук у руки, як це робить фіат. Навпаки, транзакції постійно реєструються в блокчейні, вказуючи, які кошти належать якій адресі [2].

Блокчейн – збудований за певними правилами безперервний послідовний ланцюжок блоків (зв’язковий список), що містять інформацію [3].

Зв’язок між блоками забезпечується як нумерацією, а й тим, кожен блок містить свою власну хеш-суму і хеш-суму попереднього блоку (рис. 1.1). Зміна будь-якої інформації в блоці змінить його хеш-суму. Щоб відповідати правилам побудови ланцюжка, зміни хеш-суми потрібно буде записати до наступного блоку, що викличе зміни вже його власної хеш-суми. При цьому попередні блоки не торкаються. Якщо змінний блок останній у ланцюжку, то внесення змін може вимагати істотних зусиль. Але якщо після блоку, що змінюється, вже сформовано продовження, то зміна може виявитися вкрай трудомістким процесом. Справа в тому, що зазвичай копії ланцюжків блоків зберігаються на багатьох різних комп’ютерах незалежно один від одного [4].

З моменту появи біткоіна в 2009 році використання блокчейну вибухнуло завдяки створенню різноманітних криптовалют, програм децентралізованого фінансування (DeFi), незамінних токенів (NFT) і смарт-контрактів.

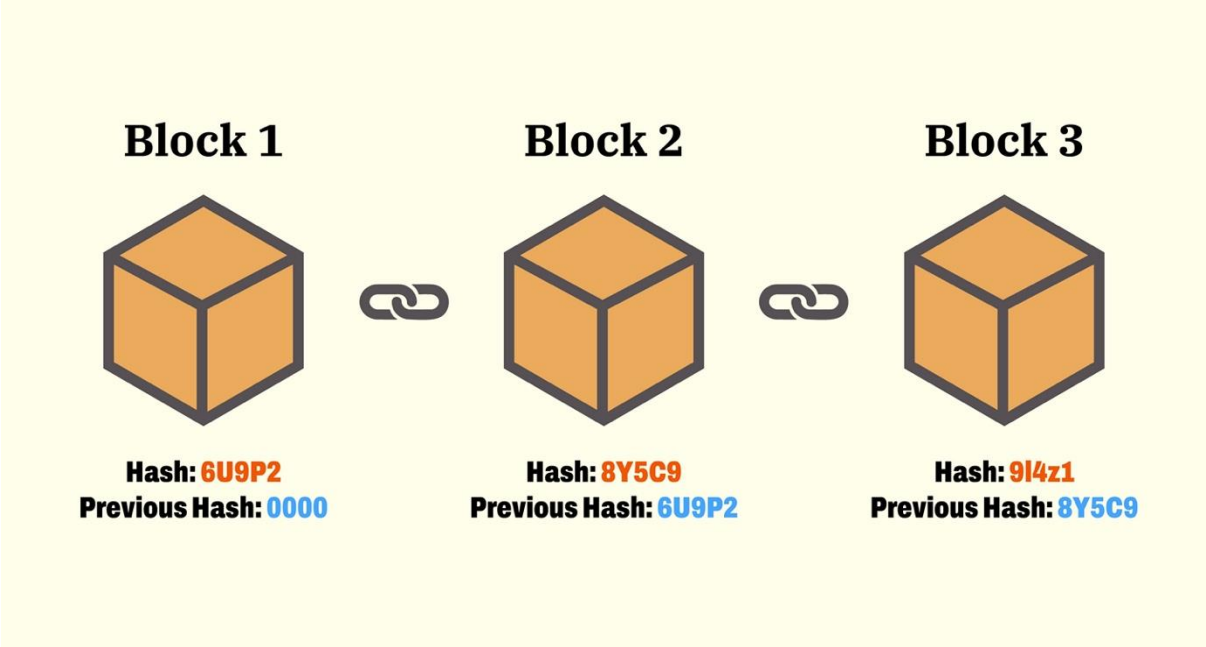


Рисунок 1.1 – Схематична структура блокчейну

Хоча як криптовалюту, так і фіатну валюту можна використовувати як засіб обміну на товари та послуги, спосіб, яким ці транзакції насправді працюють на базовому рівні, насправді зовсім інший.

Перш ніж транзакція буде додана в блокчейн, вона має бути автентифікована та авторизована. Весь процес обробки криптовалютних транзакцій (рис. 1.2) можна розділити на три окремі етапи: створення, трансляція та підтвердження (автентифікація та авторизація) [5].

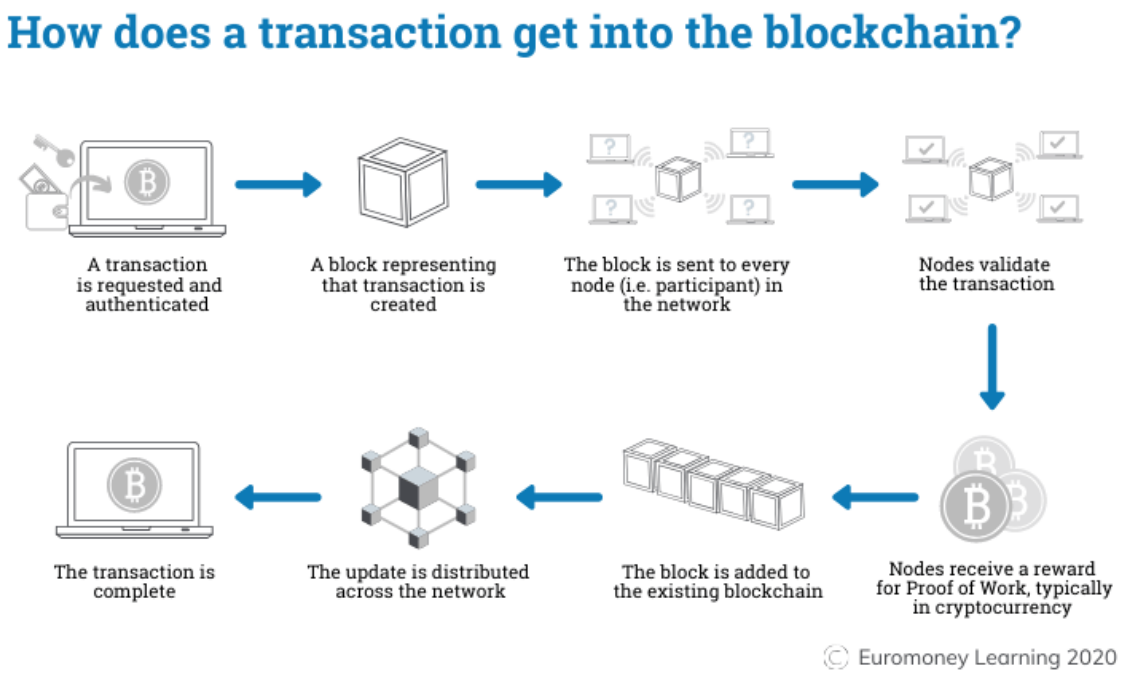


Рисунок 1.2 – Схематичне зображення взаємодії блокчейну та криптотранзакції

1.2 Поняття та різновиди криптовалютних гаманців

Криптовалютний гаманець – це пристрій, фізичний носій, програма або служба, які зберігають публічні та/або закриті ключі для транзакцій у криптовалюті. На додаток до цієї основної функції зберігання ключів, криптовалютний гаманець частіше пропонує функції шифрування та/або підпису інформації. Підписання може, наприклад, призвести до виконання смарт-контракту, транзакції з криптовалютою, ідентифікації або юридичного підписання «документа». Криптогаманець дозволяє зберігати, відправляти та

отримувати криптовалюту. Криптовалюта, така як Bitcoin, Ethereum або Litecoin, не зберігається у фізичному вигляді, як готівка, а знаходиться у цифровому форматі на блокчейні – загальнодоступному реєстрі транзакцій [6].

Криптогаманець потрібен для забезпечення безпеки та управління криптовалютними засобами. У криптогаманці зберігається закритий ключ (private key), який дозволяє користувачеві розпоряджатися своїми криптовалютними засобами. Він використовується для підпису транзакцій та підтвердження їх відправлення.

Критерії вибору криптогаманця:

– безпека криптогаманця. Перша та основна вимога до рішення для зберігання криптовалюти – безпека. Потрібно обирати спосіб, який забезпечує збереження ключів і, що ще важливіше, мінімізує ризик крадіжки коштів. Можна використовувати кастодіальний гаманець, тобто гаманець, в якому користувач передає контроль над своїми криптовалютними активами третій стороні, зазвичай це централізована біржа або якась платформа. Це означає, що користувач зберігає свої криптовалюти на рахунку, володіння яким контролюється та керується провайдером гаманця. Перевага кастодіальних гаманців є їх простотою та зручністю, особливо для новачків, використання яких не вимагає глибокого розуміння криптографії та управління приватними ключами. Однак у них є деякі недоліки, такі як потенційне порушення безпеки, оскільки користувач подається на третю сторону для зберігання та керування своїми активами, а також обмежений контроль над своїми приватними ключами та рахунком. Користувачу обов’язково потрібно проводити комплексну перевірку та переконатися, що компанія, якій довіряють свої кошти, є надійною. Якщо використовувати апаратний гаманець, криптовалюта повністю знаходиться у руках користувача, і тому лише він несе одноосібну відповідальність за її збереження. Завжди потрібно проводити власне дослідження. Надійну біржу зазвичай відрізняють професійна служба безпеки, наявність відомостей про зберігання активів, суворі процедури верифікації (KYC) та доведений досвід захисту користувачів;

– можливості контролю крипто гаманця. Ще один вирішальний фактор, який необхідно враховувати при виборі типу криптогаманця, є можливості контролю своїх коштів. Такі рішення, як апаратні та програмні гаманці, дають користувачу повний контроль над приватними ключами, тоді як, наприклад, біржі зберігають ці ключі за клієнта;

– зручність криптогаманця. Деякі способи зберігання можуть не підійти, наприклад, дей-трейдеру, тобто людині, яка займається короткостроковою покупкою і продажом фінансових інструментів впродовж одного дня, якому потрібен миттєвий доступ до засобів, або користувачу-початківцю, у якого немає досвіду зберігання власного приватного ключа. Якщо користувачу першочергово потрібна зручність, ідеальним варіантом буде кастодіальний гаманець. Користувачі Web2 зазвичай добре знайомі з цим рішенням і не готові до переходу на некастодіальний гаманець. Поки клієнт пам’ятатимете ім’я користувача та пароль, у нього завжди буде доступ до свого облікового запису. А якщо користувач використовуєте двофакторну аутентифікацію (2FA), потрібно стежити за тим, щоб не втратити доступ до свого способу аутентифікації;

– сумісність криптогаманця. Оскільки на ринку є безліч криптовалют, потрібно перевіряти, які токени підтримує гаманець, і переконатися, що користувач зможе вводити і виводити монети, якими він володіє або якими хоче торгувати.

Сьогодні існує три основні типи криптогаманців, такі як апаратний, програмний чи кастодіальний, далі ми більш детально розглянемо кожен з них [6].

Апаратний криптогаманець. З погляду безпеки апаратні гаманці вважаються зразком. Це фізичний пристрій, спеціально розроблений для зберігання приватних ключів – рядків даних, які є доступом до ваших коштів. На відміну від інших способів, про які йшлося вище, апаратні гаманці не

вимагають підключення до інтернету, тому вони є невразливими для онлайн-атак.

Однак апаратні гаманці та інші некастодіальні рішення мають один загальний недолік: якщо користувач втратить доступ до свого ключа, то швидше за все ніколи більше не побачить свої кошти. Тут немає кнопки скидання пароля або служби підтримки, куди можна звернутися, якщо щось не піде.

Приклади**:** криптогаманці від компаній-виробників Ledger, Trezor, CoolWallet, Digital Box, KeepKey та інші.

Програмний криптогаманець. Як випливає з назви, програмні гаманці є цифровими, зазвичай підключені до Інтернету та запускаються на електронному пристрої. До поширених прикладів програмних гаманців належать настільні, мобільні та вебгаманці. Біржі та гаманці браузера належать до категорії вебгаманців. Деякі є кастодиальними (тобто зберігають ключі за вас), інші – некастодіальними. Настільний гаманець – це програма, яка встановлюється на комп’ютер і, на відміну від вебгаманців, дає повний контроль над ключами та засобами.

Мобільні гаманці працюють аналогічно настільним з тією лише різницею, що вони встановлюються на мобільні пристрої та дозволяють керувати засобами через застосунки для смартфонів. Одним з таких поширених застосунків є Trust Wallet – децентралізований мобільний гаманець з відкритим вихідним кодом, який підтримує понад три мільйони криптовалют і близько 60 блокчейн-мереж. Крім інших функцій, Trust Wallet пропонує можливості зберігання NFT, стейкінг у застосунку та вбудований DeFi-браузер [7].

Хоча програмні гаманці забезпечують розумний баланс між зручністю та безпекою, завжди є ризик того, що на пристрій проникне шкідлива програма або вірус.

Кастодіальний / електронний гаманець. У кастодіальному гаманці приватні ключі зберігаються та управляються третьою стороною від вашого

імені. Найпоширенішим прикладом є біржа – сайт, що дозволяє торгувати криптовалютою. Для значної частини користувачів біржі надають найпростіший спосіб управління криптовалютою: не потрібно запамятовувати приватні ключі, інтерфейс користувача зазвичай інтуїтивно зрозумілий і простий у використанні, а служба підтримки готова допомогти у вирішенні різних питань.

Якщо ж користувач використовує некастодіальне рішення, то він сам собі банк, тобто повністю відповідає за збереження свого гаманця та коштів. Якщо клієнт вирішає зберігати криптовалюту на біржі, йому потрібно переконатися, що платформа відповідає місцевим нормативним вимогам і в ідеалі пропонує форму страховки на випадок найгірших сценаріїв.

Приклади: Cryptopay, Xapo, Blockchain та інші. А також криптогаманці, що створюються при реєстрації облікового запису на криптобіржах (наприклад, Binance, Bitfinex та інші).

1.3 Огляд існуючих криптогаманців

Існує багато різних типів криптовалютних гаманців, кожен з яких має свої сильні та слабкі сторони. Проаналізувавши їх можна відмітити що більшість підтримує лише одну або декілька криптовалют, має user-friendly інтерфейс та більшість має велику комісію на транзакції. Далі ми розглянемо найпопулярніші сьогодні крипто гаманці Coinbase, Metamask та Trezor [8].

1.3.1 Криптогаманець Coinbase

Coinbase відомий зручним інтерфейсом, що полегшує початківцям купівлю або продаж криптовалют. Має страхові поліси для покриття будь яких збитків через злом або інші порушення безпеки. Також coinbase має широкий

вибір криптовалют. Проте ця платформа має високі порівняно з іншими біржами комісії, обмежена підтримка криптовалют. Незважаючи на те, що Coinbase пропонує широкий вибір криптовалют, вона все ще має більш обмежений вибір порівняно з деякими іншими біржами. Відсутність анонімності: Coinbase вимагає від користувачів надання особистих даних та іншої особистої інформації, перш ніж вони зможуть купувати чи продавати криптовалюту. Це означає, що транзакції користувачів можна відстежити до них.

1.3.2 Криптогаманець Metamask

Metamask – безпечний легкий у використанні децентралізований криптовалютний гаманець з широким спектром блокчейнів таких як Ethereum, Polygon та Binance Smart Chain. Проте з мінусів можна відмітити відсутність анонімності, тобто транзакції користувачів можна відстежити до них, адже MetaMask вимагає від користувачів надати особисту ідентифікацію та іншу інформацію. Також користувач залежний від розширень браузера, Metamask є розширенням браузера і другий за популярністю у світі браузер Safari не підтримає його. Це може бути недоліком для користувачів як і те, що для його користування потрібні технічні знання для ефективного використання.

1.3.3 Криптогаманець Trezor

Trezor популярний гаманець, з високим рівнем безпеки та анонімності. Trezor підтримує широкий спектр криптовалют, таких як Bitcoin, Ethereum і Litecoin, а також багато альткойнів. Зручний інтерфейс: інтерфейс Trezor є дружнім та інтуїтивно зрозумілим, що полегшує початківцям навігацію та керування своїми криптовалютами. Програмне забезпечення з відкритим вихідним кодом: програмне забезпечення Trezor є відкритим кодом, що

означає, що воно може перевірятися та перевірятися спільнотою, що підвищує прозорість і безпеку.

1.4 Постановка задачі

Таким чином, створення криптогаманця є актуальним завданням для транзакцій з критовалютами.

Об’єктом роботи є розробка криптогаманця на основі мови програмування Golang та фреймворку ReactJS.

Метою роботи є розробка програмного забезпечення, яке надає безпечне зберігання та управління криптовалютними активами з використанням.

Для досягнення мети необхідно розробити застосунок, що:

– забезпечує безпечну передачу та зберігання конфіденційної інформації за допомогою шифрування;

– дозволяє користувачам взаємодіяти та обмінюватися зашифрованою інформацією з використанням різних алгоритмів шифрування та автентифікації;

– безпечно створює транзакції спектром криптовалют.

2 ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Загальний огляд використаних технологій

У цьому розділі буде надано загальний огляд технологій які були використані під час розробки застосунку для розробки електронного гаманця для управління криптовалютними активами. Застосунок був розроблений з використанням мови програмування Golang версії 1.18 для бекенду, фреймворку ReactJS для фронтенду, системи управління базами даних PostgreSQL та Redis.

2.1.1 Golang

Golang, також званий як Go, є мовою програмування з відкритим кодом, розробленою Google. Розробники програмного забезпечення використовують Go у низці операційних систем і фреймворків для розробки вебзастосунків, хмарних і мережевих служб та інших типів програмного забезпечення [9].

Go є статично типізованою, явною та створеною за мовою програмування C. Через швидкий час запуску мови Go, низькі накладні витрати на виконання та можливість працювати без віртуальної машини (VM), вона стала дуже популярною мовою для написання мікросервісів та інших цілей. Крім того, Go використовується для паралельного програмування – стратегії виконання кількох завдань одночасно, не по порядку або в частковому порядку.

На створення мови Go надихнула продуктивність і відносна простота Python. Він використовує goroutines, або полегшені процеси, і набір пакетів для ефективного керування залежностями. Він був розроблений для вирішення кількох проблем, включаючи повільний час створення,

неконтрольовані залежності, дублювання зусиль, труднощі з написанням автоматичних інструментів.

Деякі особливості мови Golang (Go) включають:

– простий і зрозумілий синтаксис: Go розроблений з упором на читання та простоту коду, що спрощує його розуміння та підтримку;

– висока продуктивність Golang пропонує компіляцію в машинний код, що забезпечує високу продуктивність виконання програм. Він також має збирач сміття для ефективного управління пам’яттю;

– підтримка паралелізму: Go надає вбудовані механізми для роботи з паралельними обчисленнями, такими як горутини (goroutines) та канали (channels), що спрощує розробку конкурентних програм;

– вбудована підтримка мережного програмування: Go має стандартну бібліотеку, яка полегшує створення мережевих програм, включаючи роботу з TCP/UDP, HTTP, WebSocket та іншими протоколами;

– статична типізація: Go є статично типізованою мовою, що забезпечує безпечнішу розробку та високу надійність програм;

– просте складання та розгортання: Go забезпечує просте складання виконуваних файлів без необхідності встановлення додаткових залежностей. Це робить розгортання програм, особливо на серверах, простіше та зручніше;

– активна спільнота та підтримка: Golang має активну спільноту розробників, яка підтримує мову, надає бібліотеки та інструменти, а також забезпечує оновлення та виправлення помилок.

2.1.2 ReactJS

Фреймворк React.js – це фреймворк і бібліотека JavaScript із відкритим кодом, розроблена Facebook. Він використовується для швидкого й ефективного створення інтерактивних інтерфейсів користувача та

вебзастосунків із значно меншою кількістю коду, ніж із ванільним JavaScript [10].

У React можна розробляти свої програми, створюючи повторно використовувані компоненти, які можна розглядати як незалежні блоки Lego. Ці компоненти є окремими частинами кінцевого інтерфейсу, які, будучи зібраними, утворюють весь інтерфейс користувача програми.

Основна роль React у застосунку полягає в створенні і управлінні компонентами користуватського інтерфейсу. Загальна мета React – забезпечити ефективне створення інтерактивних і масштабованих інтерфейсів, що робить його популярним вибором для розробки вебзастосунків [11].

Ось деякі переваги React JS перед іншими фреймворками:

– віртуальний DOM: React використовує віртуальний DOM для ефективного оновлення інтерфейсу користувача. Оновлення відбуваються лише у змінених частинах DOM, що покращує продуктивність програми;

– компонентний підхід: Розробка з використанням компонентів робить код більш читальним, легко підтримуваним і повторно використовуваним. Компоненти дозволяють розбивати інтерфейс на невеликі, незалежні та блоки, що перевикористовуються;

– односторонній поток даних: Дані React передаються по ієрархії компонентів в одному напрямку, що спрощує розуміння і налагодження коду. Це також забезпечує передбачуваність та покращує керування станом програми;

– реактивні оновлення: Зміна даних або стану автоматично оновлює відповідні частини інтерфейсу без необхідності явного втручання розробника. Це робить застосунок чуйним і забезпечує більш плавні користувальницькі взаємодії;

– широка екосистема: React має велику та активну екосистему сторонніх бібліотек, компонентів та інструментів. Це дозволяє розробникам

швидко розширювати функціональність програм, використовуючи готові рішення;

– підходить для масштабування: React підтримує розподіл інтерфейсу на безліч незалежних компонентів, що полегшує масштабування та супровід проєктів будь-якого розміру;

– підходить для розробки мобільних програм: За допомогою React Native, розробники можуть використовувати React для створення мобільних програм, що переносяться для платформ iOS і Android;

– активна спільнота: React має велику та активну спільноту розробників, яка надає навчальні матеріали, посібники, бібліотеки та інструменти. Це робить процес розробки з React доступнішим і забезпечує підтримку з боку досвідчених розробників.

2.1.3 PostgreSQL

PostgreSQL (часто званий просто Postgres) – це потужна, відкрита реляційна база даних з акцентом на розширюваність та відповідність стандартам. Вона надає надійне зберігання та управління структурованими даними, дозволяючи розробникам створювати та масштабувати програми [12].

Ось деякі ключові особливості PostgreSQL:

– реляційна база даних: PostgreSQL надає реляційну модель зберігання даних, дозволяючи організовувати інформацію в таблиці з певними відносинами та зв’язками;

– розширюваність: Postgres має потужну систему розширень, що дозволяє додавати нові функціональні можливості, типи даних та оператори. Це робить його гнучким та здатним адаптуватися до різних вимог застосунків;

– підтримка стандартів SQL: PostgreSQL активно підтримує безліч стандартів SQL та забезпечує відповідність ANSI SQL. Він також пропонує

розширені можливості, такі як віконні функції, загальні таблиці виразів (CTE) та багато іншого;

– масштабованість: Postgres підтримує масштабування як вертикально (підвищення продуктивності одному сервері) і горизонтально (розподіл даних на кілька серверів). Це дозволяє збільшувати пропускну здатність та обробляти великі обсяги даних;

– транзакційність та цілісність даних: PostgreSQL забезпечує ACID-властивості (атомарність, узгодженість, ізольованість та довговічність) для забезпечення надійності та цілісності даних. Він підтримує транзакції, точки збереження (savepoints) та механізми відновлення після збоїв;

– багатомовна підтримка: PostgreSQL підтримує безліч мов програмування та надає інтерфейси для роботи з ними, включаючи Python, Java, C++, PHP та інші.

2.1.4 Redis

Redis (Remote Dictionary Server) – це високопродуктивна система управління даними, яка використовує у пам’яті ключ-значення для зберігання та обробки даних. Вона є базою даних, що працює в оперативній пам’яті, і може використовуватися для різних цілей, включаючи кешування даних, зберігання сеансів користувачів, черги повідомлень та багато іншого [13].

Нижче наведено деякі особливості Redis:

– у пам’яті: Redis зберігає дані в оперативній пам’яті, що забезпечує високу швидкість доступу та обробки даних. Однак він також пропонує можливість зберігати дані на диск для забезпечення збереження інформації;

– ключ-значення: Redis є сховище даних, де кожне значення пов’язане з унікальним ключем. Ключі та значення можуть бути рядками, але Redis також підтримує інші типи даних, включаючи списки, хеші, множини та сортовані множини;

– багатий набір функцій: Redis надає безліч команд для маніпулювання даними, включаючи операції додавання, вилучення, оновлення та видалення. Він також підтримує різні операції над наборами даних, такі як об’єднання, перетин та різницю;

– висока продуктивність: Завдяки своїй архітектурі та збереженню даних у пам’яті, Redis забезпечує високу швидкість виконання операцій читання та запису. Це робить його чудовим вибором для програм, де потрібен швидкий доступ до даних;

– підтримка різних мов та платформ: Redis має клієнтські бібліотеки для різних мов програмування, таких як Python, Java, JavaScript, Golang, Ruby та інші. Це забезпечує просту інтеграцію з програмами на різних платформах.

Redis може використовуватися в різних сценаріях, включаючи кешування даних для покращення продуктивності, зберігання тимчасових даних, керування сеансами користувачів, обробку черг повідомлень та реалізацію лічильників та статистики. Його простота використання та висока продуктивність роблять Redis популярним вибором для багатьох застосунків [14].

2.1.5 Інтегроване середовище розробки (IDE)

Для розробки використовується середовище Goland розроблене компанією JetBrains, що пропонує широкий набір функцій і інструментів, спеціально адаптований для роботи з мовою програмування Golang (рис. 2.1) та одночасно дозволяє робити запити до бази даних (рис. 2.2).

До його основних можливостей входять [15]:

– інтелектуальна підтримка: Goland має потужний механізм автодоповнення, підказок, рефакторингу та перевірки синтаксису, що спрощує розробку та підвищує продуктивність;

– інструменти для налагодження: IDE надає зручні засоби для налагодження Go-коду, включаючи точки зупинки, перегляд змінних та стека викликів;

– управління залежностями: Goland інтегрується з інструментом управління залежностями Go, таким як Go Modules, забезпечуючи зручну роботу із зовнішніми пакетами та управління залежностями проєкту;

– інтеграція із системами контролю версій: IDE дозволяє інтегрувати проєкти Go з різними системами контролю версій, такими як Git, Subversion та Mercurial, спрощуючи спільну роботу над кодом;

– тестування: Goland пропонує інструменти для написання та запуску модульних тестів Go, а також інтеграцію з фреймворками тестування;

– інтеграція з інструментами складання: IDE інтегрується з популярними інструментами складання проєктів Go, такими як go build, go run та go test, що дозволяє легко збирати, запускати та тестувати код;

– рефакторинг та кодогенерація: Goland пропонує широкий набір інструментів для рефакторингу коду, автоматичної генерації коду та підвищення його якості.

Рисунок 2.1 – Приклад роботи з Golang в IDE Goland

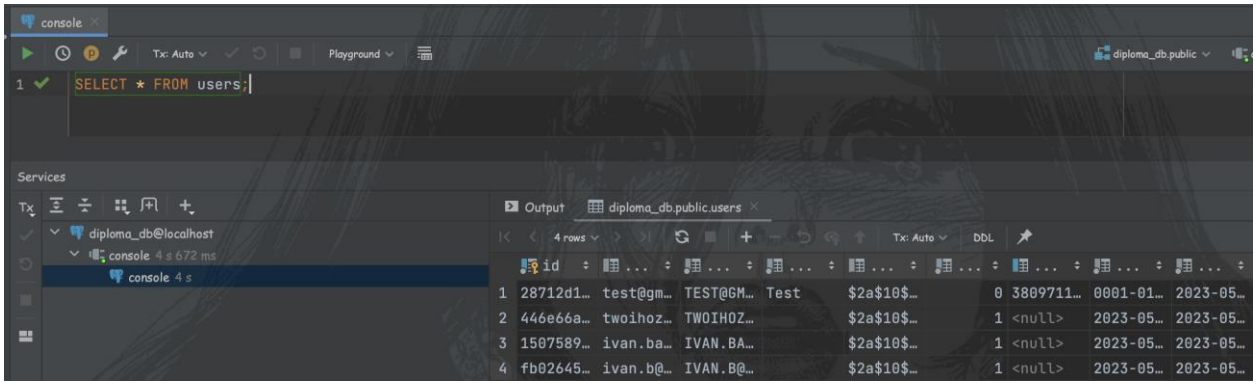


Рисунок 2.2 – Приклад роботи з PostgreSQL в IDE Goland

Ці технології та інструменти використовуються разом для розробки застосунку електронного гаманця для управління криптовалютними активами. За допомогою них можна створити швидку, безпечну та функціональну програму [16].

2.2 Вибір бази даних для проєкту

Вибір бази даних для криптогаманця, написаного мовою Golang, є важливим з кількох причин, таких як збереження та безпека даних: криптогаманець зберігає конфіденційну інформацію про користувацькі облікові записи та транзакції. PostgreSQL пропонує потужні функції безпеки, включаючи підтримку шифрування даних, автентифікацію та авторизацію на рівні користувача та ролей, а також механізми аудиту та контролю доступу. Це допомагає забезпечити надійний захист користувачів [12].

Реляційна модель даних: криптогаманець може знадобитися зберігати пов’язані дані, такі як користувачі, акаунти, транзакції та історію операцій. Реляційна модель PostgreSQL забезпечує ефективне зберігання та керування такими зв’язаними даними за допомогою таблиць, зв’язків та індексів. Це полегшує структурування та виконання складних запитів до даних.

Масштабованість і продуктивність: Криптогаманець може мати безліч користувачів і обробляти велику кількість транзакцій. PostgreSQL пропонує

можливості масштабування, включаючи підтримку горизонтального та вертикального масштабування, а також індексування та оптимізацію запитів. Це дозволяє забезпечити високу продуктивність та обробку великих обсягів даних [17].

Підтримка транзакцій та цілісності даних: криптогаманець вимагає надійної підтримки транзакцій для забезпечення цілісності даних при виконанні операцій, таких як переказ коштів або оновлення балансу облікового запису.

Широкі можливості розширення: PostgreSQL є однією з найбільш розширюваних баз даних, підтримуючи функції користувача, процедури, що зберігаються, тригери і можливість створення розширень. Це дозволяє розширити функціональність криптогаманця за допомогою певних типів даних або функцій, специфічних для вашої програми.

Всі ці фактори роблять PostgreSQL привабливим вибором для розробки криптогаманця на мові Golang. Однак, при ухваленні рішення про вибір бази даних, також важливо враховувати особливості вашої програми, вимоги до продуктивності, масштабованості та інші фактори, щоб вибрати найбільш відповідне рішення.

Проте, Redis може бути корисним доповненням до бази даних PostgreSQL для криптогаманця з кількох причин:

– кешування даних: Redis є швидкою базою даних, що працює в оперативній пам’яті. Він може використовуватися для кешування даних криптогаманця, що часто запитуються, таких як баланси акаунтів, історія транзакцій та інші показники. Кешування даних у Redis може значно покращити продуктивність та знизити навантаження на базу даних PostgreSQL;

– керування сеансами користувачів: Криптогаманці часто вимагають автентифікації та керування сеансами користувачів. Redis пропонує функціональність зберігання тимчасових даних, що робить його

відповідним інструментом для керування сеансами користувачів, включаючи зберігання інформації про вхід у систему, авторизацію та інші сеансові дані;

– черги повідомлень: Redis підтримує структуру даних «черга», яка може бути корисною для обробки асинхронних завдань у криптогаманці. Наприклад, при виконанні транзакцій або оновленні балансів акаунтів можна поміщати завдання в чергу Redis для подальшої обробки у фоновому режимі;

– публікація-підписка: Redis підтримує патерн «публікація-підписка», який дозволяє різним компонентам криптогаманця обмінюватися повідомленнями в режимі реального часу. Це може бути корисним для оповіщення користувачів про важливі події, такі як отримання нової транзакції або зміна стану облікового запису.

В цілому, Redis надає додаткові можливості для покращення продуктивності, управління сеансами та обробки завдань у криптогаманці.

2.3 Контейнеризація застосунку

Docker – це платформа для контейнеризації застосунків. Він надає середовище виконання, в якому програми можуть бути упаковані в контейнери з усіма необхідними залежностями, включаючи операційну систему, бібліотеки та інші компоненти. Контейнери Docker (рис. 2.3) пропонують легковажне та ізольоване оточення, яке дозволяє запускати та розгортати програми на різних платформах з мінімальними проблемами сумісності [18].

Контейнери Docker можуть бути запущені практично на будь–якій платформі, яка підтримує Docker, локальний комп’ютер, сервер або хмарна інфраструктура. Це забезпечує високу портабельність застосунків та знижує проблеми сумісності. Кожен контейнер Docker має своє ізольоване оточення, яке гарантує, що програми не будуть взаємодіяти один з одним або з хост–

системою. Це дозволяє уникнути конфліктів залежностей та забезпечує надійність та безпеку виконання застосунків.

Docker спрощує процес розробки та розгортання програм. Можна створити Docker-образи, що містять всі необхідні компоненти програми, та відтворювати їх легко на різних середовищах, включаючи розробку, тестування та виробництво. Це забезпечує одноманітне та повторюване оточення для роботи над застосунком.

Docker має можливості масштабування застосунків. Можна створювати кілька екземплярів контейнерів та керувати ними за допомогою оркестраторів, таких як Docker Swarm або Kubernetes. Це дозволяє керувати та масштабувати програму залежно від навантаження. Також Docker оптимізує використання ресурсів шляхом поділу та ізоляції застосунків у контейнерах. Він дозволяє запускати кілька контейнерів на одному хості та ефективно використовувати обчислювальні ресурси, пам’ять та мережеві ресурси.

Загалом, Docker надає простий та ефективний спосіб упаковки, розгортання та управління застосунками, що робить його незамінним інструментом у сфері розробки та розгортання програмного забезпечення.

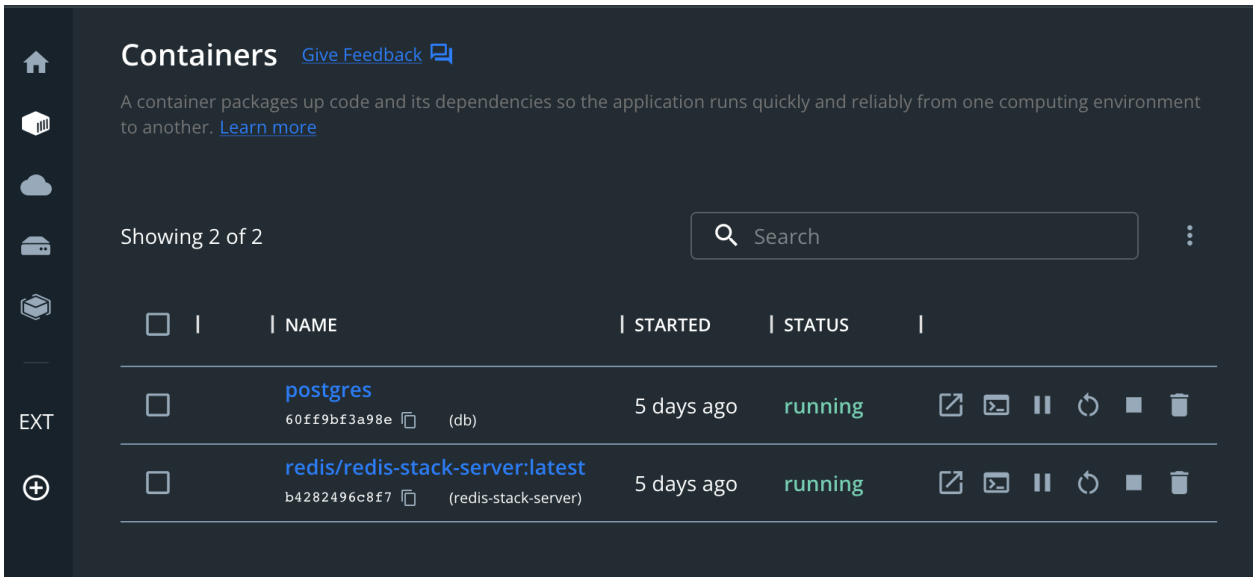


Рисунок 2.3 – Вигляд головної сторінки програми Docker

Лістинг 2.1 Команда для запуска контейнеру redis:

docker run –d —name redis–stack–server –p 6379:6379 redis/redis–stack–server:latest;

2.4 Взаємодія між фронтендом та бекендом

Фронтенд (англ. «frontend») – це частина веброзробки, яка відповідає за створення інтерфейсу користувача і взаємодію з користувачем. Фронтенд розробники працюють з мовами програмування, такими як HTML, CSS та JavaScript, для створення інтерактивних та чуйних вебсторінок. Вони займаються верстkою, стилізацією та додаванням функціональності до сайтів та вебзастосунків, щоб забезпечити зручність використання та гарний користувальницький досвід [19].

Бекенд (англ. «backend») – це частина веброзробки, яка відповідає за серверну частину програми. На відміну від фронтенда, який працює з інтерфейсом користувача, бекенд обробляє дані та бізнес-логіку на стороні сервера. Бекенд розробники працюють з мовами програмування, такими як Python, Java, Golang або Ruby, та використовують бази даних, сервери та інші інструменти для створення та управління серверною інфраструктурою. Вони відповідають за обробку запитів від фронтенду, взаємодію з базами даних, обробку даних, автентифікацію, забезпечення безпеки та інші завдання, пов’язані із серверною частиною програми.

HTTP-сервер – це частина програмного забезпечення або сервіс, який розуміє URL-адреси (вебадреси) і HTTP, тобто протокол прикладного рівня передачі даних, спочатку – у вигляді гіпертекстових документів у форматі HTML, в даний час використовується для передачі довільних даних. Загалом, коли браузеру потрібен файл, розміщений на вебсервері, браузер запитує його через протокол HTTP. Коли запит досягає потрібного вебсервера («залізо»),

сервер HTTP (ПЗ) приймає запит, знаходить запитуваний документ (якщо ні, то повідомляє про помилку 404) і відправляє назад, також через HTTP [13].

Взаємодія між ReactJS та Golang з використанням архітектурного стилю REST API включає наступні кроки:

На стороні бекенд частини, написаної на Go створюється HTTP-сервер за допомогою пакету «net/http» або інших подібних пакетів.

Щоб використати пакет в Golang необхідно його імпортувати наступним чином.

Лістинг 2.2 Імпорт пакету в Golang:

```
import (
   "net/http"
)
```

Після підключення пакету можна встановити порт з’єднання та функцію-обробник http.ListenAndServe(":9088”, nil), яка буде виконена після запуску програми та з’єднає сервер з портом «http://localhost:9088»

Далі визначаються маршрути та обробники, які відповідатимуть на запити від ReactJS. Реалізуються обробники HTTP-запитів на серверній стороні Golang. Залежно від вимог застосунка, це може бути читання або запис даних до бази даних, виконання обчислень або взаємодія із зовнішніми сервісами.

У ReactJS потрібно використовувати бібліотеку, таку як 'fetch' або 'axios', щоб упакувати дані у формат JSON. Це можна зробити шляхом серіалізації JavaScript об’єкта в JSON, тобто процес перетворення об’єктів у рядкову форму подання, за допомогою JSON.stringify(). Для відправки HTTP-запитів на серверну сторону Golang. Далі викликаються відповідні методи виконання GET, POST, PUT, DELETE та інші типів запитів.

У запитах ReactJS потрібно вказати URL-адресу сервера Golang і шлях до відповідного маршруту API. У разі потреби також можна передавати параметри або тіло запиту, щоб надіслати дані на сервер.

Наприклад, коли користувач виконує функцію логіну, фронтенд отримує дані від користувача, такі як email та пароль, та надсилає ці дані у форматі JSON на адресу http://localhost:9088/auth/login з методом POST.

Лістинг 2.3 Приклад взаємодії ReactJS з Golang:

```
const path = `${this.ROOT_PATH}/auth/login`;
      const response = await this.http.post(
        path,
        JSON.stringify(authorizationParameters)
      );
```

Далі на сервері Golang потрібно обробити отримані запити, тобто вийняти параметри та дані, якщо вони є, та виконати відповідні операції. Можливо, вам знадобиться використовувати пакети для аналізу JSON або обробки даних. На серверній стороні Golang можна використовувати пакет encoding/json, щоб декодувати отримані дані в JSON-об’єкти. Цей пакет надає можливість перетворювати JSON-дані у структури даних Golang і назад, використовуючи функції json.Marshal() та json.Unmarshal(). Потім можна відправляти цей JSON-рядок у відповіді HTTP.

У ReactJS програмі, при отриманні відповіді від сервера Golang, використовується функція JSON.parse() для перетворення отриманого рядка JSON назад в JavaScript-об’єкт.

Далі надсилаються відповіді від сервера Golang назад до програми ReactJS. Потрібно обов’язково переконатися, що сервер повертає правильні коди стану HTTP (наприклад, 200 для успішних запитів або 404 для неіснуючих ресурсів).

У програмі ReactJS оброблюються отримані дані або помилки, надіслані від сервера Golang та оновлюється стан компонентів та/або відображає отримані дані на інтерфейсі користувача.

2.5 Порівняння ефективності мов програмування Java та Golang

Для обґрунтування вибору мови програмування Golang, я вирішив взяти для приклада два популярні криптогаманці, такі як Trust Wallet, що написаний на Java, та Exodus, який в свою чергу написаний на Golang [20]. На жаль, не маючи точної інформації про внутрішні деталі розробки конкретних криптогаманців, необхідних даних та їх порівняння, неможливо надати повноцінне детальне порівняння ефективності Java та Go для цих проєктів:

– продуктивність: Go спочатку розроблявся з упором на високу продуктивність та ефективне використання ресурсів, що може бути важливим для криптогаманця з великим обсягом даних та вимогливих обчислень. Java також є продуктивною мовою і має потужну віртуальну машину Java (JVM), яка оптимізує виконання коду;

– керування пам’яттю: Java використовує автоматичне керування пам’яттю за допомогою збирача сміття, що спрощує розробку та позбавляє від ручної роботи з керування пам’яттю. Go також має збирач сміття, але він працює трохи інакше і забезпечує більш передбачувану продуктивність;

– паралелізм та конкурентність: Go надає потужні інструменти для роботи з паралелізмом та конкурентністю, включаючи горутині та канали. Це може бути корисно для асинхронних операцій та обробки одночасних запитів у криптогаманці. Java також має підтримку багатопоточності за допомогою потоків та синхронізації;

– екосистема та інструменти: Java має багату екосистему з безліччю бібліотек, фреймворків та інструментів для розробки різних програм, включаючи криптогаманці. Цей пункт має дві сторони: з одного боку мова Golang не має такої великої кількості бібліотек, як Java, проте мова є мінімалістичною і не має лишніх конструкцій, саме через цей пункт швидкість розробки значно зростає при відносно однаковій ефективності цих мов.

3 АРХІТЕКТУРА ІНФОРМАЦІЙНОЇ СИСТЕМИ

3.1 Програмна реалізація застосунку

Усю розробку застосунку можна розділити на такі розділи, як представлення (Presentation Layer), бізнес логіка (Business Logic Layer) та доступ до даних (Data Access Layer). Ці розділи є ключовими у застосунку, створеного з використанням тришарової архітектури (рис. 3.1).

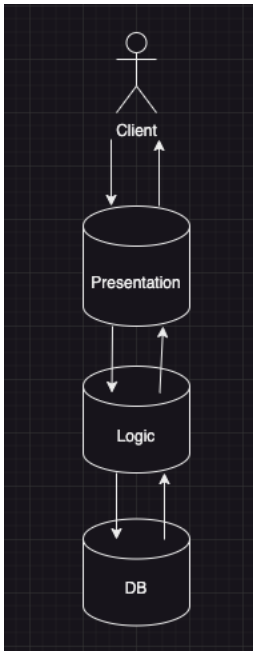


Рисунок 3.1 – Схема тришарової архітектури

Тришарова архітектура сприяє поділу відповідальності та модульності у застосунку. Кожен шар має своє специфічне завдання і може бути незалежно розроблено, змінено або замінено без впливу на інші шари. Це спрощує супровід, тестування та масштабування програми [21].

Спершу було розроблено шар доступу до даних, а саме проєктування бази даних шляхом визначення сутностей та атрибутів, тобто кожна сутність є окремим типом даних, а кожен атрибут визначає конкретну характеристику

сутності. Визначення зв’язків між сутностями та визначення первинних та зовнішніх ключів. Зв’язки визначають відносини між таблицями, а ключі забезпечують унікальність та цілісність даних. Процес нормалізації, який допомагає усунути надмірність даних та запобігти аномалії при оновленні, вставці або видаленні даних. Нормалізація дозволяє організувати дані в окремі таблиці таким чином, щоб вони були більш ефективними та масштабованими. Створення конкретної схеми бази даних, що включає визначення таблиць, атрибутів, зв’язків, ключів, обмежень цілісності та інших необхідних елементів. Схема бази даних визначає структуру бази даних і визначає, як дані зберігатимуться та організовуватимуться. Визначення методів оптимізації запитів та створення індексів для покращення продуктивності бази даних. Це може містити вибір відповідних типів індексів, створення оптимальних запитів та використання інших технік оптимізації [22].

Структура бази даних застосунку має наступні сутності:

– Users – зареєстровані користувачі застосунку;

– Transactions – створені користувачами транзакції для створення їх історії;

– Users Secret – код від двохфакторної автентифікації;

– Users Tokens – сесії користувачів з даними про пристрій, з якого вони зайшли та токеном авторизації;

– Temporary Tokens – тимчасовий токен, що надається після проходження двохфакторної автентифікації для того, щоб користувач міг робити захищені дії, такі як зміну своїх облікових даних та транзакції криптовалюти.

Після визначення зв’язків сутностей та їх створення, схема бази даних приймає вигляд, що зображено на рисунку 3.2.

Отримання історії транзакцій на рівні доступу до даних має вигляд, що наведено на рисунку 3.3.

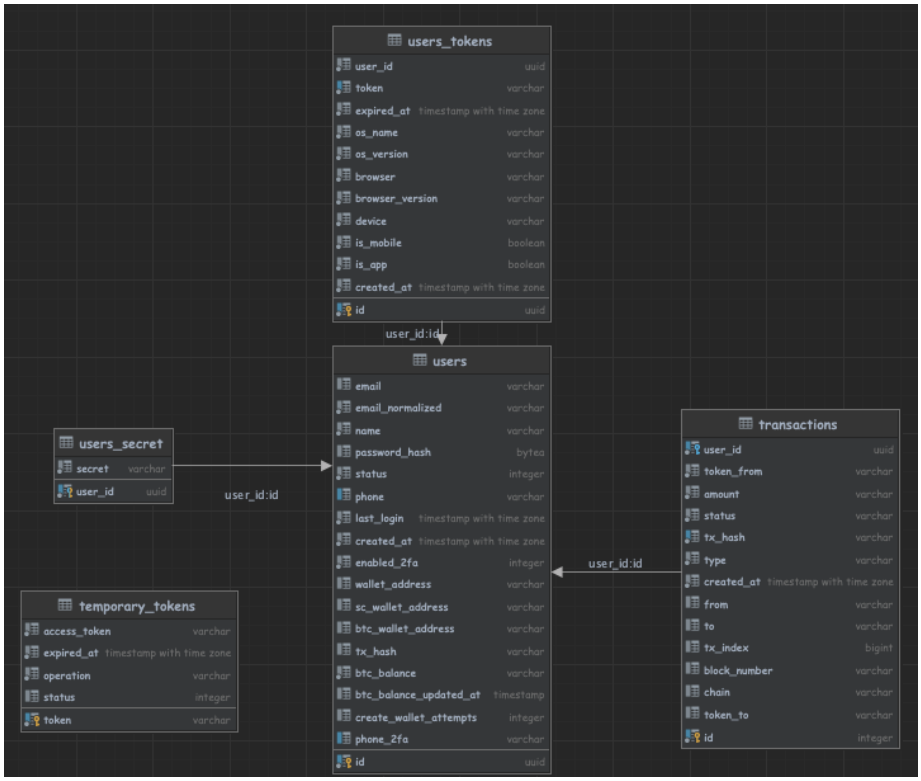


Рисунок 3.2 – Вигляд схеми бази даних у застосунку

```go
func (transactionsDB *transactionsDB) GetByFilter(
   ctx context.Context,
   filter *wallets.TransactionFilter,
   pagination *util.PaginationReq,
) ([]*wallets.Transaction, error) {

   query, params, err := goqu.
      Select(
         "id",
         "chain",
         "tx_index",
         "block_number",
         "user_id",
         "token_from",
         "token_to",
         "from",
         "to",
         "amount",
         "status",
         "tx_hash",
         "type",
         "created_at").
      From("transactions").
      Where(filter.ToExpressions()...).
      Limit(uint(pagination.Size)).
      Offset(uint(pagination.GetDBOffset())).
      Order(goqu.C("created_at").Desc()).
      ToSQL()
   if err != nil {
      return nil, ErrTransactions.Wrap(err)
   }
```

Рисунок 3.3 – Функція GetByFilter()

В даному коді не використовується звичайний SQL код. На заміну застосовується SqlBuilder goqu. SqlBuilder – це бібліотека, яка намагається полегшити створення SQL-запитів у програмах. Використання однієї мови програмування (Golang) для створення коду для іншої мови (тобто SQL) завжди є проблемою. Завжди виникають проблеми з екрануванням символів у рядкових літералах, встановленням пробілів у потрібному місці та збігом дужок. І часто, навіть після того, як код налагоджено та повністю протестовано, він залишається дуже крихким. Найменша зміна виведе ситуацію з рівноваги та потребує ще одного тестування та налаштування. Тому для спрощення коду використовується sql builder [23].

Наступним кроком було створення сервісів та побудова логіки застосунку. Проаналізувавши структуру бази даних, безперечно необхідно використовувати методологію програмування ООП.

ООП (об’єктно-орієнтоване програмування) – це методологія програмування, яка дозволяє структурувати код у вигляді об’єктів, що поєднують дані та функціональність, пов’язану з цими даними. Ось кілька основних переваг та цілей, які досягаються за допомогою ООП [24].

Модульність та повторне використання коду: ООП дозволяє розбити програму на незалежні об’єкти, які можуть бути використані повторно в різних частинах програми або інших проєктах. Це сприяє підвищенню продуктивності розробки та полегшує підтримку коду.

Інкапсуляція та абстракція: ООП дозволяє об’єднувати дані та методи, які з ними працюють, усередині об’єктів. Це дозволяє приховати внутрішню реалізацію об’єкта та надати лише необхідний інтерфейс для взаємодії з ним. Інкапсуляція полегшує розуміння та використання об’єктів іншими розробниками, абстракція дозволяє зосередитись на важливих деталях, приховуючи непотрібні деталі реалізації.

Спадкування та поліморфізм: ООП підтримує успадкування, дозволяючи створювати нові класи на основі існуючих. Це дозволяє спадковим класам отримати властивості та методи батьківського класу та

додати свою специфічну функціональність. Поліморфізм дозволяє використовувати об’єкти різних класів з тим самим інтерфейсом, що сприяє гнучкості та розширюваності коду.

Спрощення розробки та підтримки: ООП дозволяє розбити складні завдання на дрібніші, логічно пов’язані об’єкти. Це полегшує розуміння коду, його поділ на модулі та розподіл роботи між розробниками. Код, організований за принципами ОВП, легше підтримувати, модифікувати та тестувати.

Моделювання реального світу: ООП дозволяє створювати моделі об’єктів та процесів реального світу у програмному коді. Це сприяє розумінню та уявленню реальних сутностей та їх взаємодії у програмному оточенні.

У застосунку ООП дозволяє створювати моделі об’єктів, які відповідають реальним криптогаманцям та їх функціональності. Структури (в мові програмування Golang не існує класів) можуть представляти сутності, такі як адреси гаманців, транзакції, баланси та інші атрибути та операції, пов’язані з криптогаманцем. Це спрощує розробку та розуміння логіки роботи з криптогаманцями у програмному коді. Також ООП застосоване для забезпечення безпеки криптогаманців. Класи та методи розроблені з урахуванням принципів безпеки, таких як шифрування даних, автентифікація користувачів, контроль доступу тощо. Це допомагає захистити приватні ключі та іншу конфіденційну інформацію, пов’язану з криптогаманцями [25].

Для прикладу реалізації методу сервісу я взяв отримання балансу (рис. 3.4).

Даний код зв’язується з архітектурним рівнем бази даних, отримує вартість за одну одиницю конкретного токена (BTC / MATIC) в доларах США, які туди потрапляють за допомогою циклічного звертання до мережі раз на годину для оновлення курсу валюти незалежно від валюти.

```
func (service *Service) GetWalletBalance(ctx context.Context, scWalletAddress
common.Address, date time.Time) (*Wallet, error) {
   latestTokensPrice, err :=
service.cryptoTokens.GetLatestTMPByCreatedAt(ctx, time.Now())
   if err != nil {
      return nil, ErrWallets.Wrap(err)
   }

   latestTokensPriceMap :=
crypto_tokens.TokenPriceArrayToMap(latestTokensPrice)

   tokensPriceByDate, err := service.cryptoTokens.GetHistoryByCreatedAt(ctx,
date)
   if err != nil {
      return nil, ErrWallets.Wrap(err)
   }

   tokensPriceByDateMap :=
crypto_tokens.TokenPriceArrayToMap(tokensPriceByDate)

   wallet := new(Wallet)
   balance, decimals, err := service.ethereum.GetBalance(ctx, token,
scWalletAddress)
   if err != nil {
      return nil, ErrWallets.Wrap(err)
   }

   latestTokenPrice, ok := latestTokensPriceMap[token]
   if !ok {
      latestTokenPrice = &crypto_tokens.TokenPrice{}
   }

   balanceDecimal := util.ToDecimal(balance, int(decimals.Int64()))
   usdValue, _ :=
balanceDecimal.Mul(decimal.NewFromFloat(latestTokenPrice.USDPrice)).Round(2).
Float64()
```

Рисунок 3.4 – Частина функції GetWalletBalance()

Далі зв’язується зі сторонньою бібліотекою «go-ethereum», викликає метод GetBalance(), який приймає токен та адресу гаманця та повертає його баланс. Потім баланси по черзі конвертуються в долари США, підсумовуються та округлюються для зручності (рис. 3.5).

В останню чергу було створено користувацький інтерфейс та рівень презентації, який дозволяє користувачам взаємодіяти з застосункам, а саме зареєструватись, створити гаманець, переглянути балансу, відправити, отримати криптовалюту та переглянути історію транзакцій [26]. Presentation layer (шар представлення) в тришаровій архітектурі є компонентом, що відповідає за відображення даних і взаємодію з користувачем. Цей шар

забезпечує інтерфейс користувача та подання даних у зручній і зрозумілій формі.

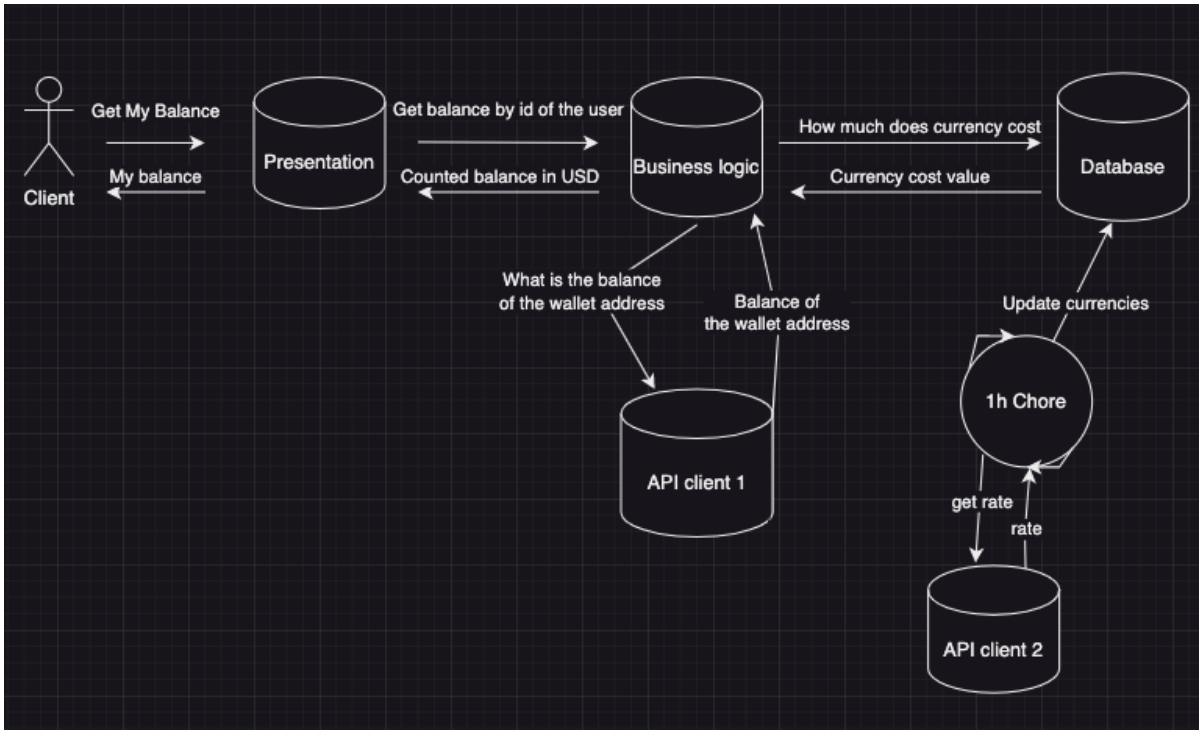


Рисунок 3.5 – Схема отримання балансу за схемою тришарової архітектури

Основні завдання Presentation layer включають [27]:

– візуалізацію даних: Presentation layer відповідає за відображення даних та інформації користувача. Він опрацьовує запити користувача, відображає дані у вигляді тексту, графіки, таблиць та інших елементів інтерфейсу;

– керування введенням користувача: Presentation layer обробляє введення користувача, такий як натискання кнопок, введення тексту, вибір елементів інтерфейсу та інше. Він відповідає за обробку подій та передачу відповідних дій на бізнес-логіку програми;

– взаємодія з бізнес-логікою: Presentation layer служить сполучною ланкою між інтерфейсом користувача і бізнес-логікою програми. Він передає запити та дані між Presentation layer та Business layer (шар бізнес-логіки), а також отримує результати назад для відображення користувача;

– обробка помилок та повідомлень: Presentation layer відповідає за обробку помилок, повідомлень та повідомлень, які потрібно показати користувачу. Він може виводити повідомлення про помилки, попередження, підтвердження дій та інші типи повідомлень.

Оскільки на рівні представлення є необхідність в наданні даних з сервера користувачеві, на етапі розробки потрібно застосувати REST API, тобто архітектурний стиль проєктування API із використанням протоколу HTTP. Головна перевага REST – велика гнучкість [28].

REST API взаємодіє з допомогою HTTP запитів, виконуючи стандартні функції: створення, оновлення, читання, видалення записів у ресурсі. Існує чотири методи, що описують, що потрібно робити з ресурсом (рис. 3.6):

– POST – створення ресурсу;
– GET – отримання ресурсу;
– PUT – оновлення ресурсу;
– DELETE – видалення ресурсу.

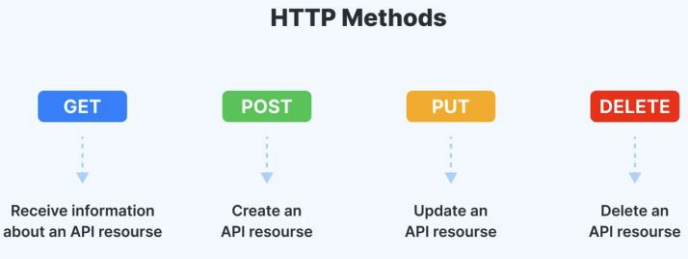


Рисунок 3.6 – Основні методи REST API

3.2 Інструкція користувача

Після ініціалізації застосунку користувач має створити або авторизуватись в обліковий запис. Для цього потрібно натиснути кнопку Login / Register та заповнити необхідні поля. Кожне поле проходить валідацію для запобігання внесення некоректних даних до застосунку.

Реалізція функції валідації пароля, який має мати хоча б один символ верхнього регістру, одну цифру та більше ніж 8 символів.

Лістинг 3.1 Функція IsPasswordValid():

```
// IsPasswordValid check the password for all conditions.
func IsPasswordValid(s string) bool {
   var number, upper bool
   letters := 0
   for _, c := range s {
      switch {
      case unicode.IsNumber(c):
         number = true
      case unicode.IsUpper(c):
         upper = true
      case unicode.IsLetter(c) || c == ' ':
         letters++
      }
   }

   return len(s) >= 8 && letters >= 1 && number && upper
}
```

Результат відображено на рисунку 3.7.

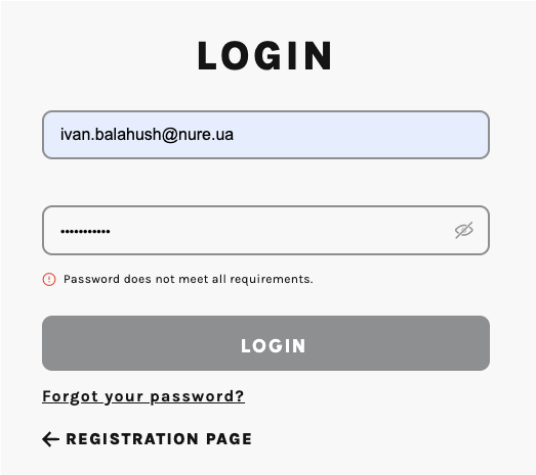


Рисунок 3.7 – Логін

Після реєстрації користувач має змогу створити гаманець, де задля безпеки він має записати секретну фразу відновлення з 12 слів і зберегти її в безпечному місці, доступному лише користувачу. Якщо він втрачений, програма не зможе допомогти відновити його та отримати доступ до гаманця. (рис. 3.8).

Рисунок 3.8 – Створення секретної фрази відновлення

Далі користувач має записати в окремі поля усі збережені слова, для підтвердження секретної фрази відновлення (рис. 3.9). При неправильному введенні слів, гаманець не буде створений.

Рисунок 3.9 – Підтвердження секретної фрази відновлення

Після того, як користувач пройшов попередні кроки, в нього будуть створені два гаманці: ETH/Polygon Wallet Address, та BTC Wallet Address (рис. 3.10).

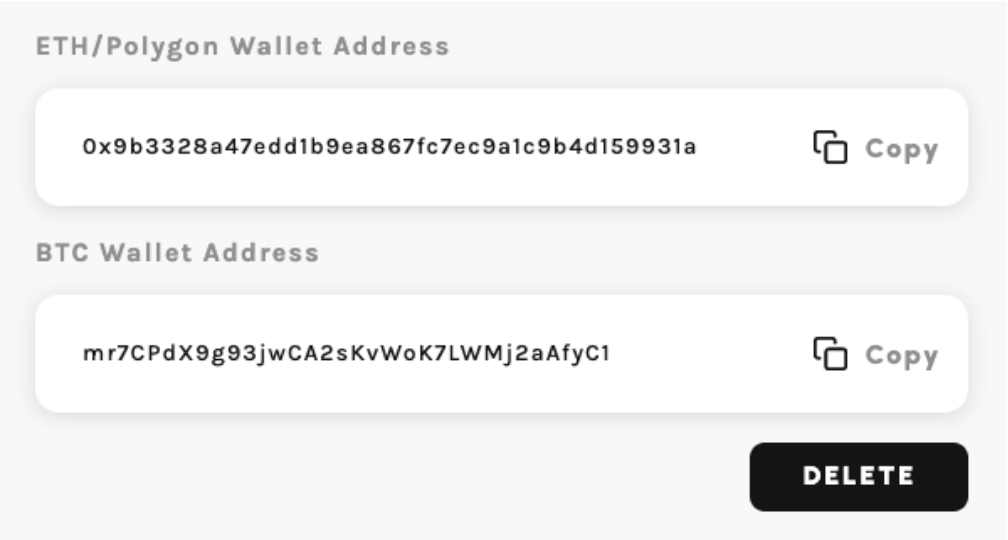


Рисунок 3.10 – Гаманці користувача

ETH/Polygon Wallet Address – це адреса гаманця, який використовується для збереження та передачі криптовалюти Ethereum (ETH) або Polygon (MATIC) на відповідних блокчейнах. Це унікальний рядок символів, що складається з цифр та букв латинського алфавіту, що починається з префіксу «0x». Гаманець з такою адресою можна використовувати для надсилання або отримання ETH або MATIC між різними учасниками мережі. Polygon – це блокчейн, який забезпечує масштабовані, безпечні та миттєві транзакції валют Ethereum, таких як ETH, USDC і DAI DAI.

BTC Wallet Address – це чіткий ідентифікатор, який дозволяє надсилати та отримувати біткойни. Це віртуальна адреса, яка вказує на призначення або джерело транзакції біткойнів, повідомляючи людям, куди надсилати біткойни та звідки вони отримали платіж у біткойнах. Це можна порівняти з системою електронної пошти, через яку користувач надсилає та отримуєте електронні листи. У цьому випадку електронні листи – це біткойни користувача, адреса

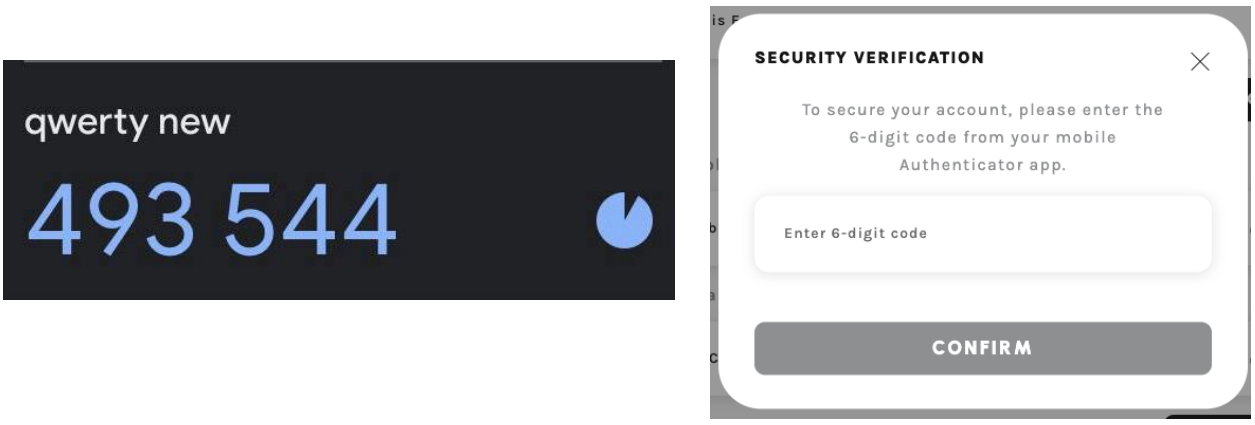
електронної пошти – це адреса в біткойнах, а електронна скринька – це гаманець у біткойнах.

Адреси біткойнів зазвичай пов’язані з біткойн–гаманцем, що допомагає керувати своїми біткойнами [29].

Структурно адреси Bitcoin зазвичай складаються з 26–35 символів і зазвичай є буквено-цифровими. У них є закриті ключі, які вам потрібні для здійснення транзакцій між адресами. Ці адреси мають стандартний формат: хеш відкритого ключа оплати (P2PKH).

Хоча гаманці успішно створені, ними не можна користуватись до тих пір, доки користувач не увімкне двохфакторну автентифікацію.

Двофакторна автентифікація (2FA) – це система безпеки, яка потребує двох різних форм ідентифікації, щоб отримати доступ до чогось [30]. Щоб виконати будь яку захищену дію в застосунку, наприклад зміну пароля, потрібно ввести код з застосунка (рис. 3.11).



а) б)

Рисунок 3.11 – Отримання та введення PIN-коду:
а) PIN-код з застоснку Authenticator; б) форма «Security verification» для введення PIN-коду

Далі програма попередить користувача про те, що він втратить доступ до своїх коштів та гаманців через застосунок (рис. 3.12).

Рисунок 3.12 – Видалення гаманців

Двофакторну автентифікацію можна використовувати для посилення безпеки онлайн-облікового запису, смартфона або навіть дверей. 2FA робить це, вимагаючи від користувача два типи інформації – пароль або персональний ідентифікаційний номер (PIN-код), код, надісланий на смартфон користувача, на електронну пошту, або відбиток пальця – перш ніж отримати доступ до захищеної інформації [31].

У застосунку використовується Authenticator, який генерує PIN-коди кожні 15 секунд. Для реалізації двухфакторної автентифікації я використовував бібліотеку «github.com/dgryski/dgoogauth» мови програмування Golang.

Зображений метод активується кожного разу, коли користувач вводить шестизначний PIN-код. Метод перевіряє секретний код користувача, викликає метод бібліотеки dgoogauth, створює певну конфігурацію за замовчуванням та викликає метод конфігурації otcp.Authenticate(code string) (bool, error), який повертає значення типу boolean, що означає чи коректний PIN-код ввів користувач.

Лістинг 3.2 Функція підтвердження 2FA PIN-коду із застосунку Authenticator:

```
// Match2FACode gets the secret key of a user and compares it with code
from application.
func (service *Service) Match2FACode(ctx context.Context, id uuid.UUID,
code string) (bool, error) {
   user, err := service.Get(ctx, id)
   if err != nil {
      return false, ErrUsers.Wrap(err)
   }

   secret, err := service.users.GetSecret(ctx, user.ID)
   if err != nil {
      return false, ErrUsers.Wrap(err)
   }

   otpc := &dgoogauth.OTPConfig{
      Secret:      secret,
      WindowSize:  3,
      HotpCounter: 0,
   }

   isAuthorized, err := otpc.Authenticate(code)

   return isAuthorized, ErrUsers.Wrap(err)
}
```

Після проходження усіх засобів безпеки для створення гаманця, наступним кроком користувач може поповнити баланс гаманця. Робиться це у вкладці transactions, де можна скопіювати адресу будь якого з гаманців та поповнити його через різні платформи та послуги. Наприклад через криптовалютні біржі, такі як Binance, Coinbase, Kraken або Bitstamp; через платіжні системи, які підтримують поповнення гаманців; біткоїн-банкомати або peer-to-peer, тобто людина, яка володіє криптовалютою, може надіслати її безпосередньо на потрібний гаманець [32].

Незалежно від обраного способу поповнення, важливо стежити за безпекою та переконатися, що користувач працює з надійними платформами та особами.

Після поповнення рахунку гаманця, у вкладці dashboard ми можемо перевірити наш баланс (рис. 3.13), актуальну ціну в доларах США за одне ціле значення та значення користувача конкретної монети, окремий баланс монети,

та різні дії, такі як deposit – де можна скопіювати свою адресу для використання її на біржі, як було вказано вище.

Також можна перейти на вкладку withdraw у якій можна обрати валюту та зробити переказ на інший гаманець (рис. 3.14).

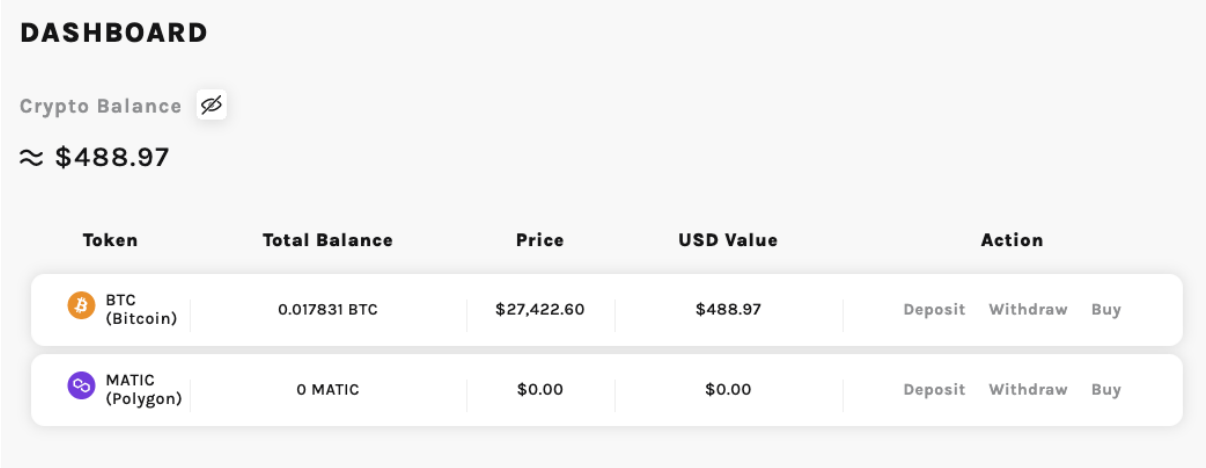


Рисунок 3.13 – Перевірка балансу різних монет

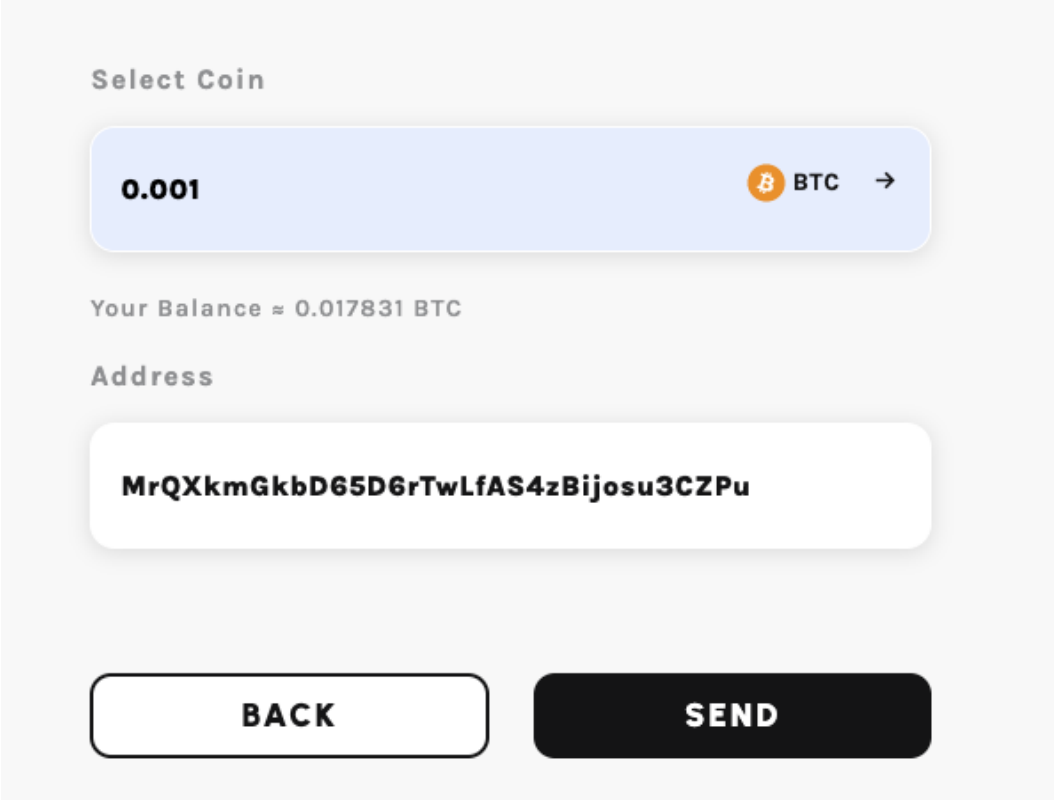


Рисунок 3.14 – Створення транзакції

У вкладці «My Account» користувач має можливість зміни своїх даних, таких як email, телефон, пароль, тип двохфакторної автентифікації, видалення своїх адрес та оскільки мета цього застосунку було забезпечення безпеки, є можливість керування активними сесіями (рис. 3.15).

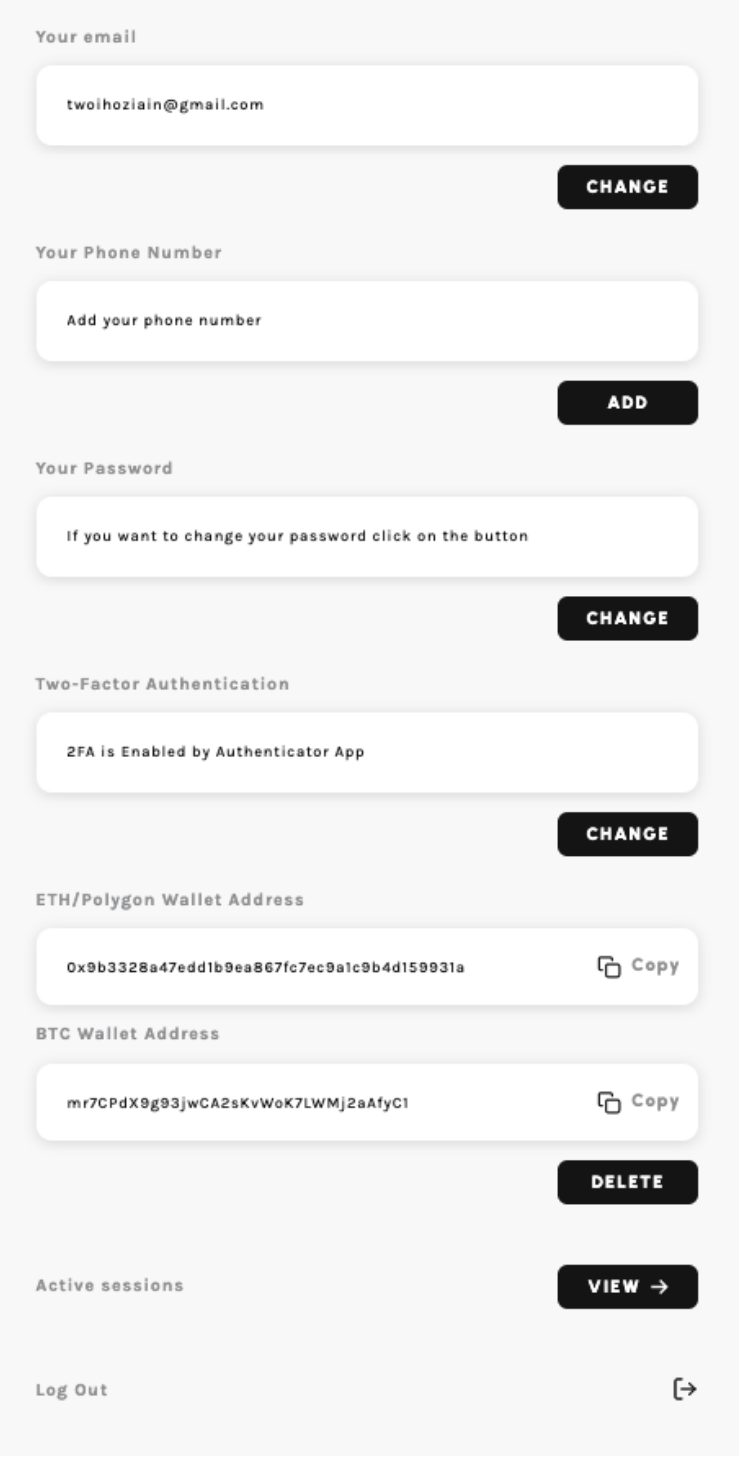


Рисунок 3.15 – Вкладка «My account»

Якщо користувач натисне кнопку «VIEW», він побачить список усіх активних сесій свого облікового запису та може ними керувати (рис. 3.16). Це створено задля безпеки, адже він може зайти в обліковий запис через чужий пристрій та забути про це, або зловмисники якимось чином отримають доступ до його акаунта, а користувач зайшовши до списку активних сесій зможе видалити лишні та змінити пароль.

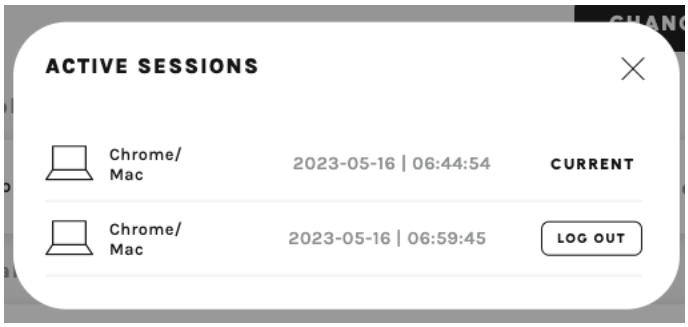


Рисунок 3.16 – Список активних сесій

ВИСНОВКИ

У рамках кваліфікаційної роботи було розроблено застосунок та проведене дослідження зі створення ефективного та безпечного криптовалютного гаманця на мові програмування Golang, який може бути використаний в різних галузях, що працюють з криптовалютами.

Дослідження призводить до таких висновків:

– безпека є пріоритетом, тобто створення криптовалютного гаманця потребує особливої обережності та уваги безпеки. Захист приватних ключів та даних користувача є невід'ємною частиною розробки гаманця. Дослідження показує, що важливо використовувати прийняті стандарти безпеки, добре вивчати вразливість та використовувати перевірені методи шифрування та зберігання даних;

– ефективність для зручності користувачів: Користувачі чекають швидких та чуйних криптовалютних гаманців. Оптимізація продуктивності, зменшення затримок та покращення швидкості транзакцій є важливими аспектами дослідження. Застосування ефективних алгоритмів, оптимізація роботи мережі та поліпшення інтерфейсу користувача сприяють зручності використання гаманця;

– інтеграція з екосистемою криптовалюти: Криптовалютні гаманці часто вимагають взаємодії з блокчейн-мережею та іншими сервісами. Важливо перевіряти можливість інтеграції з криптовалютними протоколами, врахувати вимоги до обміну даними та можливості інтеграції із зовнішніми сервісами.

Розробники повинні приділяти особливу увагу безпеці, оптимізації продуктивності та тестуванню, щоб забезпечити надійність та зручність використання для користувачів.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. What is cryptocurrency. URL: https://academy.binance.com/uk/articles/what–is–a–cryptocurrency (дата звернення 20.04.2023).

2. Sitnikov, D., Ryabov, O., Titova, O., & Kovalenko, A. (2018). *Assessment of extended aggregated association rules.* The 9th IEEE International Conference on Dependable Systems, Services and Technologies. IEEE Xplore, Kyiv, Ukraine.

3. Drescher D. (2017) Blockchain Basics: A Non–Technical Introduction in 25 Steps.

4. How transactions get into the blockchain. URL: https://www.euromoney.com/learning/blockchain–explained/how–transactions–get–into–the–blockchain (дата звернення 20.04.2023).

5. Andreas M. Antonopulos (2017) Mastering Bitcoin: Programming the Open Blockchain. Second Edition.

6. Where to store cryptocurrency. URL: https://inventure.com.ua/uk/analytics/articles/de–zberigati–kriptovalyutu:–yakij–krashe–vibrati–kriptogamanec (дата звернення 21.04.2023).

7. Grishin, I. A., Timirgaleeva, R., Titov, S. V., & Titova, Y. V. (2017). *Technology of wireless transmission of energy to remote objects based on multi-frequency system of transmitters.* Antenna Theory and Techniques (ICATT), 2017 XI International Conference. IEEE Xplore.

8. Antonopoulos A. M. (2018) Mastering Bitcoin.

9. Varghese S. (2018) Web Development with Go: Building Scalable Web Apps and RESTful Services.

10. What is React JS. URL:https://blog.hubspot.com/website/react–js (дата звернення 22.04.2023).

11. Sitnikov D., Titova O., Minukhin S., Kovalenko A., Titov S. (2018). *Informativity of Association Rules from the Viewpoint of Information Theory*. Conference: 2018 IEEE International Scientific–Practical Conference Problems of Infocommunications. Science and Technology, Kharkiv, Ukraine.

12. Douglas K. (2020) PostgreSQL: A Comprehensive Guide to Building, Programming, and Administering PostgreSQL Databases.

13. What is a web server. URL: https://developer.mozilla.org/ru/docs/Learn/Common_questions/Web_mechanics/What_is_a_web_server (дата звернення 25.04.2023).

14. Titova O., Vysotskyi D. (2021) The development of a subsystem for palliative patients information support. *Trends in science and practice of today. Abstracts of V International Scientific and Practical Conference*. Ankara, Turkey. Pp. 406–408.

15. Hunt A. and Thomas D. (2015). The Pragmatic Programmer: Your Journey to Mastery.

16. Тітова, О. В., Тітов, С. В. (2015). Оцінка юзабіліті освітніх сайтів: методи і технології. *Вісник ХДАК*, 47, 127–134.

17. Тітов С.В., Тітова О.В. (2018). Аналіз якості інформаційно-технологічного забезпечення обробки документів у хмарному сервісі MICROSOFT ONEDRIVE. *Вісник ХДАК*, 52, 142-148.

18. Cook J. (2022) Docker for Data Science: Building Scalable and Extensible Data Infrastructure around the Jupyter Notebook Server.

19. Тітов С.В., Тітова О.В., Чорна О.С. (2022). Опис нескоротних наборів ознак в приблизних множинах з використанням систем числення. *Збірник наукових праць Харківського національного університету Повітряних Сил*. № 1(71), 106-110.

20. Cox–Buday K. (2019) Concurrency in Go: Tools and Techniques for Developers.

21. Kennedy W. (2021) Effective Go: Programming.

22. Тітов С.В., Тітова О.В., Чорна О.С. (2023). Метод знаходження апроксимацій приблизних множин з використанням систем числення. *Системи обробки інформації.* 2(173), 15-19.

23. Kennedy W., Ketelsen B., St. Martin E. (2015) Go in Action.

24. Two–factor authentication. URL: https://www.investopedia.com/terms/t/twofactor–authentication–2fa.asp (дата звернення 25.04.2023).

25. Angel Garcia M., Kurt R. (2017) REST API Development with Node.js: Manage and Understand the Full Capabilities of Successful REST Development.

26. Bitcoin wallet address. URL: https://www.makeuseof.com/bitcoin–address–types–explained/#:~:text=A%20Bitcoin%20wallet%20address%20is,have%20received%20a%20bitcoin%20payment (дата звернення 28.04.2023).

27. Bugl D. (2017). Learning Redux.

28. Erikson M. (2017) Practical Redux: Building Scalable Applications with Redux and React.

29. Ethereum wallet address. URL: https://bitkan.com/learn/what–is–eth–contract–address–on–matic–is–eth–on–polygon–the–same–as–matic–9215 (дата звернення 28.04.2023).

30. Garreau M. , Faurot W. (2018) Redux in Action.

31. Idsrasiri, K. & Kuruppu, D. (2020). gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes 1st, pp. 10–22.

32. Martin R. C. (2019) Clean Code: A Handbook of Agile Software Craftsmanship.