НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ДОПУСКАЄТЬСЯ ДО ЗАХИСТУ

Завідувач кафедри
комп'ютерних наук
_____ Голуб Б.Л., доц., к.т.н.
підпис ПІБ, вчене звання і ступінь

«___»_____2025 р

БАКАЛАВРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Програмне забезпечення функціонування криптовалюти з використанням технології блокчейн**»**

Спеціальність 121 «Інженерія програмного забезпечення»

Гарант освітньої програми
_____ к.т.н., доцент _____ Вайганг Г.О_____
Науковий ступень та вчене звання підпис ПІБ

Керівник бакалаврської кваліфікаційної роботи :
К.т.н.,с.н.с. _____ Боярінова Ю.Є.
підпис ПІБ
Виконав: _____ Басай О.Ю.
підпис ПІБ

КИЇВ-2025

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ БІОРЕСУРСІВ
І ПРИРОДОКОРИСТУВАННЯ УКРАЇНИ
Факультет інформаційних технологій

ЗАТВЕРДЖУЮ

Завідувач кафедри
комп'ютерних наук
_____ Голуб Б.Л

"__"_____ 2025 р.

ЗАВДАННЯ

на виконання бакалаврської кваліфікаційної роботи студенту

Басаю Олександру Юрійовичу

Спеціальність 121 «Інженерія програмного забезпечення»

1. Тема роботи: Програмне забезпечення функціонування криптовалюти з використанням технології блокчейн.

Затверджена наказом ректора НУБіП України № 2248 "С" від 16.12.2024

2. Термін подання завершеної роботи на кафедру 2025 . 06 . 02
рік, місяць, число

3. Вихідні дані до роботи: опис програмного забезпечення, навчальна та методична література, державні стандарти

4. Перелік питань що розглядаються:
   1. Аналіз проблемної області.
   2. Вибір та обґрунтування засобів для розробки системи.
   3. Проектування інформаційної системи.
   4. Висновки.

Керівник бакалаврської кваліфікаційної роботи _____ Боярінова Ю.Є.
підпис ініціали та прізвище

Завдання прийняв до виконання _____ Басай О.Ю.
підпис ініціали та прізвище

Дата отримання завдання 2025 .___.____
рік, місяць, число

Зміст

Перелік умовних позначень..... 5
ВСТУП..... 6
1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....9
1.1 Опис предметної області.....9
1.2 Аналіз вимог до програмної системи..... 11
1.2.1 Визначення функціональних вимог.....11
1.2.2 Визначення нефункціональних вимог.....13
1.2 Моделювання предметної області.....15
1.3.1 Діаграма прецедентів..... 18
1.3.2 Діаграма послідовності.....20
1.3.3 Діаграма активності..... 23
1.4 Огляд інформаційних джерел та існуючих рішень.....25
1.5 Постановка завдання.....29
2 Проєктування інформаційного та програмного забезпечення.....32
2.1 Логічна модель даних у вигляді ER-діаграми.....32
2.1.1 Загальні відомості про ER-діаграму..... 32
2.1.2 Побудова ER- діаграми.....34
2.2 Діаграма класів та кооперацій..... 36
2.3 Діаграма пакетів..... 40
2.4 Діаграма компонентів..... 42
3 Розробка інформаційного та програмного забезпечення..... 46
3.1 Система управління інформаційною базою.....46
3.2 Розробка інформаційної бази..... 49
3.3 Вибір інструментарію для створення прикладного програмного забезпечення..... 51
3.4 Алгоритмізація та програмування програмних модулів..... 55
3.4.1 Алгоритм авторизації користувача.....55
3.4.2 Алгоритм генерації гаманця користувача.....56
3.4.3 Алгоритм покупки токенів через сайт.....57
3.4.4 Алгоритм перевірки балансу токенів..... 58

3.4.5 Алгоритм підтвердження транзакції..... 59
4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ..... 61
4.1 Тестування системи.....61
4.3 Склад інсталяційного пакету.....64
ВИСНОВКИ.....68
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ..... 70
ДОДАТОК А.....72
ДОДАТОК Б..... 73
ДОДАТОК В.....74
ДОДАТОК Д.....75
ДОДАТОК Е..... 76
ДОДАТОК Ж..... 77
ДОДАТОК И.....78

Перелік умовних позначень

API - це програмний інтерфейс, який дає змогу різним системам взаємодіяти та обмінюватися даними

ER - Entity-Relationship, модель для проєктування баз даних

KYC – це ключова міра у правилах боротьби з відмиванням грошей, що робить її важливим засобом безпеки

MS - Microsoft

ORM - Object-Relational Mapping, технологія, що дозволяє використовувати об'єктно-орієнтовані моделі для роботи з реляційними базами даних

SQL – Structured Query Language

TPS- це показник, за яким визначається продуктивність блокчейн-мережі

UML - Unified Modeling Language, мова для моделювання програмного забезпечення

XML - eXtensible Markup Language, мова розмітки для зберігання та передавання даних

БД – база даних

BEP-20 - Binance Smart Chain (BSC)

ІС – інформаційна система

ПЗ – програмне забезпечення

СУБД – система управління базою даних

ВСТУП

На сьогоднішній день блокчейн-технології є однією з найперспективніших і найдинамічніших сфер розвитку у світі інформаційних технологій. Зокрема, криптовалюти на основі технології блокчейн продовжують стрімко змінювати традиційні фінансові ринки, пропонуючи нові підходи до безпечної, прозорої та децентралізованої передачі цінності.

Стандартизація процесів створення токенів стала важливою складовою розвитку блокчейн-екосистем. Одним із найбільш популярних стандартів є BEP20 — протокол створення токенів у мережі Binance Smart Chain (BSC), який надає розробникам можливість запуску криптовалют із мінімальними витратами ресурсів та максимальними можливостями інтеграції.

З розвитком криптовалютної галузі зростає потреба у спеціалізованому програмному забезпеченні для розгортання, управління та обслуговування власних токенів. Актуальним є завдання створення систем, які забезпечують повноцінне функціонування токенів BEP20 — від емісії токену до взаємодії користувачів через гаманці, контроль транзакцій, аудит безпеки і підтримку додаткової функціональності (заморозка, спалення, випуск додаткової емісії тощо).

Необхідність впровадження сучасних засобів автоматизації в управління криптоактивами обумовлена бажанням розробників та компаній забезпечити високий рівень безпеки, зручність користування, прозорість транзакцій та дотримання вимог регуляторів.

Таким чином, розробка комплексної системи функціонування токена на основі BEP20 спрямована на досягнення кількох ключових цілей:

- забезпечення безпечної емісії та управління токенами в мережі Binance Smart Chain;
- реалізація функцій верифікації користувачів для підвищення рівня довіри в екосистемі;

- підтримка контролю транзакцій і відстеження історії дій користувачів через систему аудиту;
- створення зручного та адаптивного інтерфейсу для інтеграції з Web3-гаманцями, такими як MetaMask.

Мета даної бакалаврської кваліфікаційної роботи полягає у розробці програмного забезпечення для системи функціонування криптовалюти з використанням технології блокчейн BEP20. Створена система повинна забезпечити безпечне створення, управління, передачу токенів та перевірку користувачів через інтуїтивно зрозумілий веб-інтерфейс.

Для досягнення цієї мети необхідно розв’язати такі завдання:

- вивчити принципи роботи технології блокчейн та стандарту BEP20;
- проаналізувати існуючі рішення для управління токенами на Binance Smart Chain;
- сформулювати функціональні та нефункціональні вимоги до системи;
- розробити моделі бізнес-процесів за допомогою UML-діаграм;
- скласти загальну архітектуру програмного продукту;
- провести аналіз інструментів та обрати оптимальні засоби розробки;
- створити прототип користувацького інтерфейсу системи у Figma та реалізувати його;
- розробити смарт-контракт BEP20 для функціонування токену;
- побудувати серверну частину для взаємодії з блокчейн-мережею та базою даних;
- реалізувати механізми верифікації користувачів та управління їх правами;
- виконати тестування системи на коректність роботи, безпеку та продуктивність.

Програмне забезпечення має забезпечити можливість гнучкого управління випущеними токенами, вести облік верифікованих користувачів, переглядати історію транзакцій та оперативно реагувати на будь-які зміни в системі безпеки. Таким чином, створення системи функціонування токену BEP20 є важливим кроком у розвитку децентралізованих фінансових рішень та сприятиме

популяризації блокчейн-технологій серед ширшого кола користувачів та розробників.

1 СИСТЕМНИЙ АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Опис предметної області

Сучасний етап розвитку цифрових технологій характеризується стрімким зростанням популярності криптовалют та блокчейн-систем, які перетворюють традиційні фінансові інститути, забезпечуючи децентралізовані, прозорі та безпечні механізми обміну та зберігання активів.

Криптовалюта — різновид цифрової валюти, емісія та облік якої виконується децентралізованою платіжною системою повністю в автоматичному режимі (без можливості внутрішнього або зовнішнього адміністрування, зокрема і державними органами). Принциповою особливістю криптовалют є збереження інформації у блокчейні, де асиметричне шифрування використовується для перевірки повноважень, а інші криптографічні методи — як доказ виконаної роботи та/або Proof-of-stake.

Блокчейн — розподілена база даних, що зберігає впорядкований ланцюжок записів (так званих блоків), який постійно довшає. Кожен блок містить часову позначку, хеш попереднього блоку та дані транзакцій, подані як хеш-дерево. Інформація про транзакції зазвичай надається відкритою, не зашифрованою. Захистом від підробки та спотворення слугує включення хешу всього блоку у наступний блок.

BEP20 – це технічний стандарт токенів у блокчейні Binance Smart Chain (BSC), який забезпечує сумісність з Ethereum Virtual Machine (EVM). Це означає, що токени BEP20 можуть взаємодіяти зі смарт-контрактами, створеними для Ethereum, але з нижчими комісіями за транзакції та більш високою швидкодією. Основні переваги BEP20:

- низькі транзакційні витрати (gas fees) порівняно з Ethereum;
- висока швидкість обробки транзакцій (до 3000 TPS);

- сумісність з інструментами розробки Ethereum (Metamask, Remix IDE, Hardhat);
- широка підтримка біржами (Binance, PancakeSwap);

Даний стандарт активно використовується для:

- створення децентралізованих фінансів (DeFi) – кредитування, стейкінг, фармінг;
- розробки токенів (утилітарних, стейблкоінів, NFT);
- побудови DApps (децентралізованих додатків).

Сучасний ринок цифрових активів потребує ефективних, масштабованих та безпечних рішень для управління криптовалютами. Існуючі платформи (наприклад, Ethereum) мають проблеми з високими комісіями та низькою пропускною здатністю, тому BSC та BEP20 стають оптимальним вибором для розробки нових фінансових інструментів.

Основні завдання системи:

1. Забезпечення безпеки транзакцій (через смарт-контракти та алгоритми консенсусу).
2. Масштабованість (підтримка великої кількості операцій).
3. Інтеграція з DeFi-протоколами (обмін, ліквідність, стейкінг).
4. Зручність для користувачів (гаманці, мобільні додатки).

Незважаючи на переваги BEP20, існують певні технологічні та регуляторні виклики:

- Централізація BSC (обмежена кількість валідаторів).
- Загрози безпеці (атаки 51%, смарт-контрактні вразливості).
- Відсутність чіткої регуляції у багатьох країнах.

Проте, завдяки сучасним методам шифрування, аудиту смарт-контрактів та децентралізованим механізмам управління (DAO), ці ризики можна мінімізувати.

1.2 Аналіз вимог до програмної системи

1.2.1 Визначення функціональних вимог

При розробці системи функціонування криптовалюти на основі технології блокчейн BEP20 необхідно чітко визначити функціональні вимоги, які будуть керівними принципами у створенні програмного рішення. Функціональні вимоги - це детальний опис поведінки системи, її можливостей та способів взаємодії з користувачами та зовнішніми системами.

Для блокчейн-системи на базі BEP20 функціональні вимоги можна розділити на кілька ключових категорій:

- Управління цифровими активами: створення та емісія токенів стандарту BEP20, механізми трансферу токенів між адресами, баланси користувачів та історія транзакцій, механізми спалення (burn) токенів.
- Смарт-контракти: озробка та розгортання смарт-контрактів, виконання умовних операцій, взаємодія між різними смарт-контрактами, механізми оновлення контрактів.
- Безпека та авторизація: криптографічні алгоритми захисту даних, механізми підпису транзакцій, багаторівнева система авторизації, захист від атак (reentrancy, overflow тощо).
- Взаємодія з блокчейн-мережею:синхронізація з Binance Smart Chain, обробка транзакцій та блоків, взаємодія з нодами мережі, валідація транзакцій.
- Інтерфейси користувача: веб-інтерфейс для управління токенами, мобільний додаток для доступу до системи, API для інтеграції з іншими сервісами, інструменти для розробників.
- Моніторинг та аналітика:відстеження стану мережі, аналіз транзакцій, генерація звітів, система сповіщень.

Для кожної з цих категорій необхідно детально описати: вхідні дані, обробку,вихідні дані, обмеження, обробку помилок

Особливу увагу при проектуванні системи слід приділити:

Масштабованості: система повинна обробляти велику кількість транзакцій

Безпеці: захист від потенційних атак на смарт-контракти

Децентралізації: збереження принципів розподіленої системи

Сумісності: підтримка стандартів BEP20 та ERC20

Прозорості: можливість аудиту всіх операцій

Функціональні вимоги також повинні враховувати різні ролі користувачів системи:

- Звичайні користувачі:створення гаманців, відправка та отримання токенів, перегляд історії транзакцій, взаємодія з DeFi-протоколами;
- Розробники:розгортання смарт-контрактів, інтеграція з API, тестування функціоналу, моніторинг роботи контрактів;
- Адміністратори:моніторинг стану мережі, оновлення системних параметрів, керування комісіями, реагування на інциденти

Для кожної ролі необхідно чітко визначити: доступні функції, обмеження прав, способи автентифікації, інтерфейси взаємодії.

Важливою частиною функціональних вимог є опис роботи основних модулів системи.

Модуль емісії токенів:

- механізми створення нових токенів;;
- обмеження на кількість;
- правила розподілу.

Модуль транзакцій:

- валідація транзакцій;
- підписання операцій;
- включення в блоки.

Модуль смарт-контрактів:

- компіляція кодів;
- виконання контрактів;
- обробка подій.

Модуль безпеки:

- захист від атак;
- контроль доступу;
- шифрування даних.

Модуль API:

- взаємодія з зовнішніми сервісами;
- формати запитів та відповідей;
- обмеження частоти запитів.

Функціональні вимоги повинні бути описані з достатньою деталізацією, щоб: розробники чітко розуміли, що потрібно реалізувати, тестувальники могли перевірити коректність роботи, користувачі розуміли можливості системи, аудитори могли оцінити безпеку рішення.

Кожна функція повинна мати: унікальний ідентифікатор, чітку назву, детальний опис, приклади використання, умови виконання, очікуваний результат, м.ожливі помилки та їх обробку

Для системи на базі BEP20 особливо важливими є вимоги до:

- Сумісності з EVM (Ethereum Virtual Machine);
- Підтримки стандартних функцій BEP20 (transfer, approve тощо);
- Обробки подій (Events);
- Механізмів оновлення контрактів.

Усі функціональні вимоги повинні бути: повними (охоплювати всі необхідні функції), недвозначними (чіткий і зрозумілий опис), перевіреними (можливість тестування), узгодженими (відсутність конфліктів між вимогами), відстежуваними (можливість зв'язку з іншими артефактами проекту).

1.2.2 Визначення нефункціональних вимог

Нефункціональні вимоги представляють собою сукупність критеріїв та характеристик, що визначають якісні аспекти роботи системи функціонування криптовалюти на базі технології блокчейн BEP20. Вони не стосуються безпосередньо функціональних можливостей системи, але є вирішальними для

забезпечення її ефективної, безпечної та стабільної роботи в реальних умовах експлуатації.

Основним призначенням нефункціональних вимог є визначення стандартів якості, які мають забезпечувати: високу продуктивність системи, надійність функціонування, рівень безпеки операцій, зручність взаємодії з системою, можливість масштабування.

Продуктивність системи

Система повинна забезпечувати обробку мінімум 3000 транзакцій на секунду (TPS), що відповідає показникам мережі Binance Smart Chain. Середній час підтвердження транзакції не повинен перевищувати 3 секунди. Час відгуку API має становити не більше 500 мс для 95% запитів.

Надійність та доступність

Рівень доступності системи (uptime) має становити не менше 99,99%. Система повинна підтримувати безперебійну роботу в режимі 24/7 з автоматичним відновленням після збоїв протягом 1 хвилини. Відмовостійкість архітектури має забезпечувати цілісність даних навіть у разі технічних збоїв.

Безпека операцій

Система повинна реалізовувати: криптографічний захист даних (AES-256), механізми захисту від атак (51%, DDoS, reentrancy), двофакторну автентифікацію для критичних операцій, регулярні аудити безпеки смарт-контрактів, відповідність стандартам OWASP Top 10.

Зручність використання

Інтерфейс системи має відповідати наступним вимогам: інтуїтивна навігація та простота використання, адаптивний дизайн для різних пристроїв, підтримка мультимовності (українська, англійська), зрозумілі повідомлення про помилки, деталізована документація для розробників.

Масштабованість

Архітектура системи повинна дозволяти: горизонтальне масштабування для обслуговування зростаючої кількості користувачів, обробку до 1 мільйона

активних користувачів, легке додавання нового функціоналу, оновлення смарт-контрактів без перерв у роботі.

Відповідність стандартам

Система повинна забезпечувати: сумісність зі стандартом BEP20 та EVM, відповідність вимогам GDPR для обробки персональних даних, можливість реалізації KYC/AML процедур, прозорість усіх операцій.

Інтеграційні можливості

Система має надавати: API для інтеграції з біржами та DeFi-протоколами, підтримку популярних веб3-гаманців (MetaMask, Trust Wallet), можливість кросс-чейн взаємодії.

Адміністрування системи

Адміністративний інтерфейс повинен забезпечувати: моніторинг стану мережі в реальному часі, управління системними параметрами, генерацію звітів та аналітику, швидке реагування на інциденти.

Реалізація цих нефункціональних вимог дозволить створити систему функціонування криптовалюти на BEP20, яка поєднує в собі високу продуктивність, надійність і безпеку з зручністю використання. Вони є основою для проектування архітектури системи, вибору технологічного стеку та подальшої реалізації проекту.

Дотримання вказаних нефункціональних вимог забезпечить стабільну роботу системи в умовах реального навантаження, захистить активи користувачів і створить основу для подальшого масштабування та розвитку платформи. Ці характеристики є критично важливими для забезпечення конкурентноспроможності розроблюваної системи на ринку блокчейн-рішень.

1.2 Моделювання предметної області

Моделювання предметної області є критично важливим етапом під час проєктування будь-якої складної інформаційної системи, зокрема системи

функціонування криптовалюти на базі технології блокчейн BEP20. Від якості створених моделей залежить не тільки точність розуміння системи, але й її подальша реалізація, супровід та масштабування.

Особливість предметної області криптовалют полягає в тому, що вона об’єднує поняття фінансових операцій, мережевої безпеки, криптографії, децентралізації та смарт-контрактів. У такій багатогранній області правильно побудована модель дозволяє зменшити ризики прорахунків на етапі реалізації та забезпечити ефективну взаємодію всіх компонентів системи.

На ринку інформаційних технологій існує велика кількість CASE-засобів для підтримки процесів моделювання (наприклад, Visual Paradigm, Enterprise Architect, Draw.io, Lucidchart). Проте особливої популярності набула уніфікована мова моделювання UML (Unified Modeling Language), яка є міжнародним стандартом для побудови візуальних моделей складних програмних систем.

Процес проєктування програмного забезпечення часто є найдорожчою складовою життєвого циклу розробки (витрати на етап моделювання можуть складати до 80% загальної вартості проєкту). Саме тому створення коректної, несуперечливої та збалансованої моделі предметної області має величезне значення.

Поняття інформаційної моделі системи

Інформаційна модель — це сукупність структурованих даних, додатків і взаємодій, які утворюють єдину систему, функціонуючу за певними правилами та протоколами. У випадку розробки системи функціонування токена BEP20 інформаційна модель включає такі компоненти:

- смарт-контракт BEP20;
- користувацький інтерфейс для взаємодії з токенами (DApp);
- модулі авторизації та верифікації користувачів;
- підсистеми управління транзакціями й аудитом;
- базу даних для зберігання інформації про користувачів, їх статуси і права доступу.

Чим більше компонентів має система, тим важливіше на етапі моделювання уникнути протиріч та надмірної складності, зберігаючи при цьому усі критично важливі аспекти функціонування.

Використання UML для моделювання системи

Вибір UML як основної мови моделювання для розробки системи функціонування криптовалюти обумовлений такими перевагами:

- чітка формалізація складних процесів;
- можливість створення як концептуальних, так і фізичних моделей;
- підтримка багатьох типів діаграм для опису різних аспектів системи (структура, поведінка, розгортання);
- широке розповсюдження серед інструментів розробки.

UML використовується як системними аналітиками, так і розробниками, тестувальниками, що дозволяє забезпечити єдину мову спілкування між усіма учасниками проєкту.

При створенні моделей для системи BEP20 використовуються такі основні типи UML-діаграм:

- Діаграма класів
- Діаграма діяльності
- Діаграма розгортання
- Діаграма послідовностей
- Діаграма компонентів

Принципи побудови моделей складної системи

При моделюванні інформаційної системи для функціонування токена BEP20 застосовуються такі основні принципи:

- Принцип абстрагування: в моделі враховуються тільки ті характеристики системи, які мають безпосереднє значення для функціонування токенів і взаємодії користувачів, інші аспекти опускаються задля зменшення складності.
- Принцип багатомодельності: жодна окрема модель не може повністю відобразити всю систему. Тому створюються окремі моделі для різних

аспектів: структура даних, обмін повідомленнями, процеси транзакцій та верифікації.

- Принцип інтеграції: різні моделі повинні бути узгоджені між собою: наприклад, об’єкти діаграми класів мають відповідати сутностям у діаграмах діяльності й розгортання.

Моделювання предметної області є фундаментальним етапом при розробці системи функціонування криптовалюти на базі BEP20. Правильне використання UML-діаграм дозволяє забезпечити узгоджене бачення структури, поведінки та розгортання компонентів системи, підвищити якість розробки та уникнути суттєвих помилок на етапах реалізації.

1.3.1 Діаграма прецедентів

Діаграма прецедентів є важливим інструментом моделювання інформаційної системи, особливо на етапі аналізу вимог до функціональності. У контексті розробки криптовалютної системи на базі блокчейн-технології BEP20, така діаграма дозволяє візуально представити взаємодію користувачів (акторів) із системою, описати її основні функції, а також визначити межі відповідальності кожного учасника.

У нашій системі основними акторами виступають:

- Користувач — звичайний учасник криптосистеми, який створює гаманець, надсилає й отримує токени, перевіряє баланс, переглядає історію транзакцій тощо.
- Адміністратор системи — особа, яка відповідає за управління загальними параметрами системи, такими як встановлення комісій, блокування підозрілих адрес, оновлення контракту в разі потреби.
- Смарт-контракт BEP20 — автономний програмний компонент, який виконує всі транзакції, перевірки та облік токенів згідно з протоколом BEP20. Він виконує роль внутрішнього актора в системі.

- Платформа обміну (біржа) — зовнішня система, з якою взаємодіє наша криптовалюта, наприклад для лістингу токену, надання ліквідності чи обміну на інші цифрові активи.

Основні прецеденти взаємодії включають:

- створення гаманця
- поповнення балансу
- надсилання токенів
- перевірка балансу
- перегляд історії транзакцій
- оплата в токенах за товари/послуги
- отримання нагород
- блокування адреси

Зв’язки між акторами та прецедентами демонструють, хто ініціює ту чи іншу дію. Наприклад, тільки Адміністратор має доступ до функцій зміни конфігурації або блокування адрес, тоді як Користувач має змогу лише надсилати токени, перевіряти баланс і користуватись функціоналом системи на рівні взаємодії з токеном.

Варто зазначити наявність зв’язків типу «include» — наприклад, «Надсилання токенів» обов’язково включає «Перевірку балансу» та «Перевірку підпису транзакції», які гарантують правильність виконання дії. Зв’язки типу «extend» застосовуються, коли деякі функції не є обов’язковими. Наприклад, при «Надсиланні токенів» може бути розширення «Відображення повідомлення про недостатній баланс», яке спрацьовує лише у винятковому випадку.

Генералізація дозволяє згрупувати прецеденти в узагальнені дії, наприклад, усі операції користувача можуть входити до загального прецеденту «Керування гаманцем».

Діаграма прецедентів для системи функціонування криптовалюти на BEP20 забезпечує:

- Формалізацію функціональних вимог до системи.
- Візуалізацію ролей користувачів та системних компонентів.

- Легке розуміння обсягу роботи для розробників та аналітиків.
- Раннє виявлення сценаріїв взаємодії, які потребують додаткової уваги або перевірки.

У межах цієї роботи створена діаграма прецедентів, що ілюструє взаємодію між користувачами, адміністратором та смарт-контрактом BEP20. Вона наведена в додатку А. Опис акторів та їхніх функцій систематизовано в таблиці 1.1.

1.3.2 Діаграма послідовності

У процесі розробки системи функціонування криптовалюти з використанням технології блокчейн BEP20 важливо точно і зрозуміло відобразити взаємодію ключових компонентів. Одним із дієвих способів такого відображення є діаграма послідовності (sequence diagram).

Діаграма послідовності — різновид діаграми в UML. Діаграма послідовності відображає взаємодії об'єктів впорядкованих за часом. Зокрема, такі діаграми відображають задіяні об'єкти та послідовність надісланих повідомлень.

Опис акторів

Таблиця 1.1

Актор	Короткий опис
Користувач	Основний учасник системи, який створює гаманець, надсилає та отримує токени BEP20, перевіряє баланс і переглядає історію транзакцій. Може брати участь у стейкінгу, оплачувати товари/послуги токенами та отримувати нагороди.
Адміністратор	Відповідає за адміністрування смарт-контракту, включаючи блокування підозрілих адрес, зміну технічних параметрів контракту, а також формування та перегляд аналітичної звітності. Має повноваження керування мережею токена.
Біржа	Зовнішній учасник, що забезпечує інтеграцію криптовалюти BEP20 до торгових майданчиків (листинг). Платформа, на якій користувачі можуть обмінювати токени на інші активи.

У випадку нашої системи криптовалюти BEP20, діаграма послідовності використовується для моделювання таких основних сценаріїв:

- Підключення користувача через криптогаманець (наприклад, MetaMask);
- Перевірка балансу токенів на гаманці;
- Ініціювання та підписання транзакції;
- Відправлення токенів на іншу адресу через смарт-контракт;
- Отримання підтвердження успішної операції.

Основні елементи діаграми послідовності для проєкту BEP20:

- Актори: користувач системи (власник гаманця), адміністратор системи (має спеціальні права).
- Об'єкти: веб-інтерфейс (DApp), криптогаманець (MetaMask або інший), блокчейн BEP20, смарт-контракт токена.
- Лінія життя (lifeline): вертикальна пунктирна лінія, що позначає існування об'єкта протягом певного часу.

- Фокус управління (activation bar): прямокутник на лінії життя, що показує період активної обробки запиту.
- Повідомлення (messages): стрілки між об'єктами, що вказують на запити або відповіді в процесі взаємодії.

Уся діаграма логічно організована уздовж часової осі, яка умовно спрямована зверху вниз, від початку взаємодії до завершення процесу.

Послідовність взаємодій у нашій системі:

1. Користувач через DApp-інтерфейс підключає свій гаманець.
2. Система через Web3 API запитує адресу користувача та отримує її.
3. Користувач ініціює перевірку балансу, і DApp надсилає запит до смарт-контракту BEP20.
4. Смарт-контракт повертає баланс токенів для обраної адреси.
5. Користувач вирішує здійснити переказ токенів — через інтерфейс формується транзакція.
6. Гаманець запитує підтвердження (підписання транзакції).
7. Користувач підписує транзакцію особистим ключем.
8. Підписана транзакція надсилається в блокчейн.
9. Смарт-контракт обробляє запит і повертає результат.
10. Система відображає успішність або невдачу транзакції користувачу.

Переваги використання діаграми послідовності у нашому проєкті:

- Вона дозволяє чітко відобразити всі етапи взаємодії користувача із системою.
- Допомагає виявити можливі помилки чи "вузькі місця" ще на етапі проектування.
- Формує підґрунтя для подальшого створення класів, методів та сервісів на бекенді і фронтенді.

Особливості для проєкту BEP20:

- Часто застосовуються асинхронні запити через Web3 бібліотеки.
- Необхідно враховувати валідацію транзакцій і можливість відмов (наприклад, якщо користувач відмовляється підписувати).

- Важливо показувати на діаграмі й обробку помилок (fail-потоки), які можуть виникати при взаємодії з мережею Binance Smart Chain.

Таким чином, діаграма послідовності є обов’язковим етапом проектування системи функціонування криптовалюти на базі BEP20, оскільки вона дозволяє ретельно змоделювати всі варіанти взаємодії між компонентами системи і гарантує успішне виконання критичних бізнес-процесів.

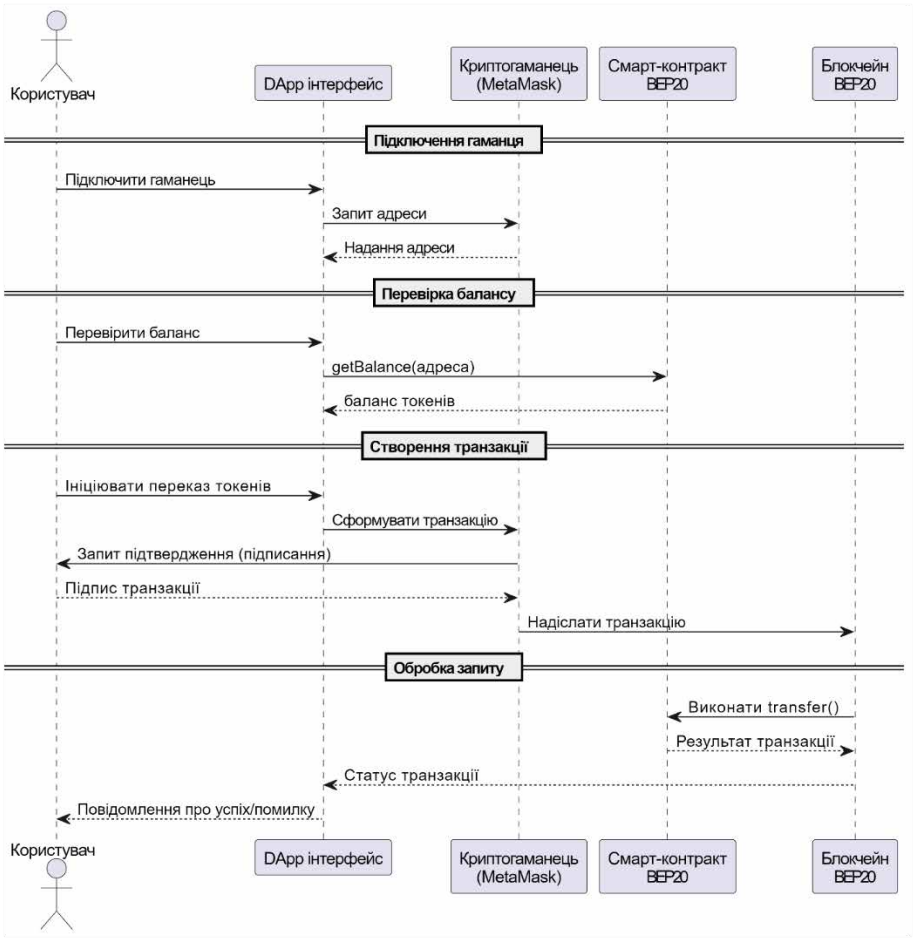


Рис. 1.1 Діаграма послідовності

1.3.3 Діаграма активності

Діаграма діяльності (діаграма активності) є одним із базових інструментів моделювання бізнес-процесів і алгоритмів системи. Вона зовні нагадує блок-схему алгоритму, проте на відміну від класичних схем, в ній зображуються не тільки послідовні дії, а й паралельні процеси, обробка даних, події, логічні розгалуження та об'єднання потоків виконання.

Діаграми діяльності широко використовуються під час опису поведінки складних інформаційних систем, особливо таких, де одночасно відбуваються численні паралельні або умовні процеси. Саме тому для проектування системи функціонування криптовалюти на основі технології блокчейн BEP20 використання діаграми діяльності є обов'язковим етапом.

У контексті розробки криптовалютної системи такі діаграми дозволяють чітко моделювати процеси:

- взаємодії користувачів із Web3 DApp інтерфейсом;
- ініціації транзакцій та їх обробки через смарт-контракт BEP20;
- перевірки балансу токенів;
- виконання адміністративних дій над токеном (спалення, емісія, заморозка).

У нашій системі взаємодії виглядатимуть наступним чином:

1. Користувач підключає гаманець до системи через інтерфейс DApp.
2. Система перевіряє стан підключення.
3. Якщо гаманець успішно підключений – користувачу стають доступні основні функції:
    - Перевірити баланс токенів.
    - Ініціювати переказ токенів на іншу адресу.
    - Авторизувати переказ через підпис транзакції.
4. У випадку адміністратора – можливість:
    - Спалити частину токенів.
    - Провести емісію додаткових токенів.
    - Заморозити або розморозити окремі гаманці.
5. Всі результати взаємодії відображаються у вигляді повідомлень про успіх або помилки.

Значення діаграм діяльності для даної системи

Перед розробкою інформаційної системи надзвичайно важливо чітко візуалізувати всі алгоритми обробки даних, які будуть автоматизовані в системі. Це дозволяє: уникнути неврахованих варіантів розвитку подій; краще спланувати взаємодію між фронтендом (інтерфейсом користувача) та бекендом

(обробкою транзакцій через блокчейн); виявити слабкі місця в процесі розробки ще на ранньому етапі.

Діаграма діяльності служить своєрідною картою руху даних у системі BEP20: від моменту підключення користувача до моменту успішного проведення транзакції чи адміністративної дії над токенами. Діаграма діяльності представлена в додатку Б.

Узагальнений опис основних елементів діаграми

- Старт активності: підключення гаманця користувача до DApp.
- Основні дії: відправка запиту до смарт-контракту BEP20, підписання транзакції особистим ключем користувача, збереження результатів транзакцій в блокчейні.
- Умовні переходи: успішне чи неуспішне підключення гаманця, підтвердження чи відмова від підпису транзакції.
- Фініш активності: отримання результату операції користувачем.

1.4 Огляд інформаційних джерел та існуючих рішень

На сучасному етапі розвитку цифрових технологій значну увагу приділено створенню децентралізованих фінансових систем, заснованих на блокчейн-технологіях. Однією з найбільш поширених і водночас перспективних платформ для створення криптовалют і токенізованих активів є Binance Smart Chain (BSC) із підтримкою стандарту BEP20. Рішення, що реалізуються на базі BEP20, отримали широке розповсюдження завдяки простоті розробки, високій швидкості обробки транзакцій та низьким комісіям у мережі.

Для забезпечення якісного проєктування системи функціонування криптовалюти з використанням технології BEP20 було здійснено аналіз інформаційних джерел та вже існуючих рішень у сфері децентралізованих фінансів (DeFi), смарт-контрактів, а також користувацьких інтерфейсів для взаємодії з токенами. Розглядалися як технічні документації (whitepapers), так і

практичні кейси реалізації криптовалютних систем, доступних у відкритих репозиторіях, таких як GitHub, а також вебсайти, форуми (Stack Overflow, Binance Community), і наукові публікації, присвячені архітектурі смарт-контрактів та механізмам транзакцій у BEP20.

У межах огляду особливу увагу було приділено аналізу трьох найбільш розповсюджених платформ, що забезпечують створення, обіг та інтеграцію токенів BEP20:

- **Trust Wallet Token (TWT)** – приклад утилітарного токена, інтегрованого з мобільним криптогаманцем Trust Wallet. Забезпечує гнучкі механізми для винагородження користувачів і знижок у межах сервісу. Підтримує взаємодію з різними децентралізованими додатками (dApps) через WalletConnect.
- **PancakeSwap (CAKE)** – децентралізована біржа на базі BEP20, що реалізує торгові пари токенів, стейкінг і фармінг. Має відкритий смарт-контракт, що служить прикладом складної реалізації логіки обміну та інтерфейсу для взаємодії з токенами.
- **SafeMoon** – токен, відомий за рахунок вбудованого механізму обмеженого обігу (burn + reflection), який служить як приклад побудови дефляційної токеноміки в BEP20-мережі. Його смарт-контракт вивчається як зразок впровадження спеціалізованих фінансових механізмів.

Порівняльна таблиця BEP20-рішень

Таблиця 1.2

Назва проекту	**Основна мета**	**Переваги**	**Недоліки**
Trust Wallet Token	Винагородження користувачів	Інтеграція в мобільний гаманець, підтримка багатьох блокчейнів	Обмежена функціональність поза Trust Wallet
PancakeSwap	Децентралізований обмін токенів	Розвинута DeFi-інфраструктура, велика база користувачів	Високий поріг входу для новачків
SafeMoon	Побудова дефляційного токена	Вбудована токеноміка burn/reflection, підтримка спільноти	Низька прозорість щодо аудиту контрактів

Проаналізовані рішення продемонстрували широкий спектр можливостей BEP20-стандарту – від простих транзакцій і гаманців до розширеної логіки фінансових операцій. Водночас спостерігається ряд типових проблем, що потребують врахування під час розробки власної системи:

- **Безпека смарт-контрактів** – велика кількість проектів піддається атакам через вразливості у коді контракту (наприклад, reentrancy-атаки).
- **Прозорість і верифікація коду** – більшість успішних проектів відкрито публікують код на Etherscan/BscScan, проходять аудит (CertiK, Hacken).
- **Юзабіліті користувача** – проекти з простими інтерфейсами (наприклад, MetaMask, Trust Wallet) отримують більшу аудиторію через легкість взаємодії навіть з мінімальними технічними знаннями.

- **Швидкість і вартість транзакцій** – використання BSC замість Ethereum дозволяє зменшити витрати, що є суттєвою перевагою для кінцевих користувачів.

Також було проаналізовано технічні стандарти та бібліотеки, які активно використовуються у створенні BEP20-систем:

- **OpenZeppelin Contracts** – набір перевірених шаблонів смарт-контрактів, що дозволяє швидко реалізувати токен BEP20, а також розширити його функціональність (minting, burning, pausing).
- **Web3.js / Ethers.js** – JavaScript-бібліотеки для інтеграції фронтенду з блокчейном через смарт-контракти.
- **MetaMask / WalletConnect** – технології, що забезпечують підключення користувача до системи без необхідності централізованої авторизації.

На основі вивчених джерел та рішень сформульовано такі ключові вимоги до власної системи:

- Створення токена BEP20 із базовими функціями (переведення, перевірка балансу, підпис транзакцій).
- Інтеграція з MetaMask або WalletConnect для безпечної ідентифікації користувачів.
- Вебінтерфейс для ініціації транзакцій, перегляду балансу та перевірки статусу операцій.
- Відкритість смарт-контракту та перевірка на BscScan.
- Урахування асинхронної взаємодії і можливості відмов (fail-сценарії).

Таким чином, аналіз інформаційних джерел та існуючих рішень дозволив сформувати чітке уявлення про сучасні підходи до побудови BEP20-систем, а також виявити як ефективні практики, так і критичні недоліки, яких слід уникати в процесі реалізації проєкту. Це створює ґрунтовну основу для переходу до етапу проектування, розробки та впровадження власної криптовалютної системи.

1.5 Постановка завдання

У сучасному світі технології блокчейн дедалі більше впливають на традиційні сфери фінансів, бізнесу й цифрової економіки. Одним із найбільш популярних напрямків є розробка і функціонування криптовалют, заснованих на стандартизованих протоколах. Зокрема, стандарт BEP20 для мережі Binance Smart Chain (BSC) став одним із найпоширеніших способів створення власних токенів.

Основним завданням даної роботи є розробка системи функціонування криптовалюти, побудованої за стандартом BEP20. Ця система має забезпечити повний життєвий цикл криптотокена: від його створення до подальшого обігу в мережі, верифікації користувачів, обробки транзакцій і підтримки інфраструктури для взаємодії з токеном.

Основна мета проєкту:

- Створити інфраструктуру для роботи токенів BEP20.
- Реалізувати механізм верифікації користувачів із використанням криптографії для підвищення безпеки взаємодії.
- Розробити зручний інтерфейс (DApp) для роботи з токенами через Web3-гаманці.
- Організувати облік операцій за допомогою журналу транзакцій та аудиту.
- Підтримати масштабування та гнучке адміністрування системи токенів.

Процеси функціонування криптовалютної системи пов’язані з рядом завдань і викликів:

- необхідністю забезпечення високого рівня безпеки збереження і передачі токенів;
- важливістю побудови точного механізму верифікації користувачів (KYC/AML принципи);
- складністю управління дозволами на обіг токенів (white list, black list);
- необхідністю інтеграції із зовнішніми системами моніторингу блокчейн-подій (наприклад, через API BscScan).

Розроблювана система має надати власнику криптовалюти інструментарій для:

- моніторингу гаманців користувачів;
- контролю обігу токенів;
- безпечного випуску або спалювання токенів;
- обмеження або відкриття доступу до транзакцій за допомогою спеціальних налаштувань.

Серед функцій, які виконуватиме розроблена інформаційна система, слід виділити:

- створення та розгортання смарт-контракту BEP20 для випуску токена;
- передача токенів між гаманцями в мережі Binance Smart Chain;
- верифікація користувачів для забезпечення контролю за обігом токенів;
- контроль над правами користувачів через механізм whitelist/blacklist;
- спалювання або додаткову емісію токенів за потреби;
- фіксацію всіх дій у спеціальному журналі аудиту, який доступний лише адміністраторам системи.

У підсумку, розроблювана система повинна бути інтуїтивно зрозумілою, безпечнішою, гнучкою для налаштування та відповідати наступним базовим вимогам:

- зберігання інформації про користувачів системи (адреси гаманців, статуси верифікації);
- ведення обліку транзакцій між користувачами і контрактами;
- можливість керування статусами токенів та адміністративними параметрами;
- надійна система автентифікації і авторизації користувачів із використанням криптографічних ключів;
- обмеження доступу до адміністративних функцій за допомогою багаторівневої авторизації;
- функції пошуку і сортування даних у системі (за адресами, датами, статусами тощо);
- можливість експорту даних про транзакції та користувачів у форматі CSV або JSON для зовнішнього аналізу.

Система повинна мати такі ключові властивості:

- Безпека — захист від несанкціонованого доступу та підробки транзакцій.
- Прозорість — всі зміни і транзакції фіксуються в блокчейні та доступні для перегляду.
- Масштабованість — можливість підтримки великої кількості токенів і користувачів.
- Доступність — підтримка роботи через браузер з використанням сучасних Web3-рішень.

2 Проєктування інформаційного та програмного забезпечення

2.1 Логічна модель даних у вигляді ER-діаграми

2.1.1 Загальні відомості про ER-діаграму

У процесі розробки інформаційної системи, що забезпечує функціонування криптовалюти на основі технології блокчейн BEP20, особливе місце посідає проектування структури даних. Створення чіткої та логічно узгодженої моделі зберігання даних є критичним етапом для забезпечення надійної, безпечної та масштабованої взаємодії користувачів із криптосистемою. Для досягнення цієї мети широко використовується ER-діаграма (діаграма "сутність-зв'язок" або Entity-Relationship diagram), яка виступає візуальним інструментом моделювання логічної структури даних.

ER-діаграма дозволяє формалізувати предметну область криптовалюти шляхом визначення основних сутностей, їхніх властивостей (атрибутів), а також зв’язків між ними. У системах, які базуються на блокчейні BEP20, це дозволяє логічно описати взаємодії таких ключових компонентів, як користувачі (власники токенів), смарт-контракти, транзакції, гаманці, події токена, а також адміністративні ролі.

Сутності (Entities) у такій системі можуть включати:

- Користувач (User) – учасник системи, який має у своєму розпорядженні криптогаманець. Атрибути: ID, адреса гаманця, рівень доступу, дата реєстрації, статус активності.
- Транзакція (Transaction) – запис у блокчейні, що відображає факт переказу токенів між адресами. Атрибути: хеш транзакції, час створення, сума, відправник, одержувач, статус.

- Гаманець (Wallet) – структура, що відображає баланс токенів на конкретній адресі. Атрибути: адреса, загальний баланс, заморожений баланс (якщо реалізовано блокування коштів), тип (користувацький/контрактний).
- Смарт-контракт (SmartContract) – програмний код, який автоматично виконує логіку криптовалюти. Атрибути: контрактний адрес, версія, дата розгортання, автор.
- Подія токена (TokenEvent) – лог, що відображає дію смарт-контракту (напр., Transfer, Approval). Атрибути: тип події, об’єкт події, адреси-учасники, пов’язана транзакція.

Зв’язки (Relationships) між цими сутностями визначаються логікою блокчейн-взаємодій:

- Користувач володіє одним або кількома гаманцями.
- Гаманець ініціює або отримує транзакції.
- Кожна транзакція записується у вигляді події токена.
- Смарт-контракт генерує події у відповідь на дії користувачів.
- Користувач взаємодіє з одним або кількома смарт-контрактами.

Кардинальність у таких зв’язках відіграє важливу роль. Наприклад, один користувач може мати декілька гаманців (зв’язок 1:N), одна транзакція може створити одну або більше подій токена (зв’язок 1:N), а один смарт-контракт може бути пов’язаний із багатьма транзакціями (1:N).

ER-діаграма, як частина логічної моделі, не залежить від вибраної бази даних чи інфраструктури, що особливо важливо при роботі з децентралізованими системами, де частина інформації зберігається «на ланцюгу» (ончейн), а інша – у централізованих чи квазіцентралізованих базах даних (офчейн), які забезпечують додаткову функціональність, наприклад, аналітику, кабінети користувача чи адміністративну панель.

На підставі ER-моделі можна обрати відповідну СУБД: для ончейн-інформації безпосередньо використовується блокчейн, а для офчейн-частини – реляційні бази даних, як-от PostgreSQL або MySQL. У випадку, коли йдеться про

обробку журналів подій або логів взаємодії з токенами, може бути доцільно використовувати NoSQL-рішення, такі як MongoDB або ElasticSearch.

ER-діаграми в контексті криптовалютного проєкту не лише забезпечують ефективне зберігання та доступ до інформації, але й допомагають уникнути критичних помилок у логіці реалізації смарт-контрактів. Наприклад, некоректно спроектовані зв’язки між сутностями можуть призвести до дублювання записів транзакцій або помилок при валідації взаємодій користувачів з токеном.

Крім того, ER-діаграма є незамінним інструментом у комунікації між розробниками смарт-контрактів, backend-розробниками, аналітиками та бізнес-замовниками. Вона дозволяє всім учасникам бачити цілісну картину структури криптовалютної системи, узгоджувати технічні деталі, і своєчасно виявляти можливі проблеми ще на етапі проєктування, до реалізації у вигляді коду.

2.1.2 Побудова ER- діаграми

Під час створення інформаційної системи для функціонування криптовалюти з використанням технології блокчейн BEP20 дуже важливим етапом є проектування бази даних. Щоб база даних була правильно структурованою, логічною та ефективною, для побудови її концептуальної моделі було обрано спеціалізоване програмне забезпечення — ERWin Data Modeler.

CA ERWin Data Modeler — це потужний інструмент для проектування, впровадження та обслуговування сховищ та баз даних, який широко використовується у великих комерційних та інженерних проєктах. Його можливості дозволяють наочно відобразити навіть найскладніші структури даних, що особливо актуально для систем на базі блокчейну, де працюють великі обсяги транзакцій, користувацьких даних та адміністративної інформації.

Серед головних можливостей ERWin Data Modeler:

- Моделювання даних високого рівня (логічна модель);
- Перетворення логічних моделей у фізичні (реальні бази даних);
- Автоматична генерація SQL-скриптів для створення баз даних;

- Можливість зворотного інжинірингу існуючих баз даних;
- Контроль цілісності даних та дотримання стандартів проектування;
- Порівняння моделей і синхронізація зі справжніми базами даних.

Також ERWin підтримує імпорт і експорт моделей у різні формати, що важливо для інтеграції з іншими системами або перенесення даних.

Особливості побудови ER-діаграми для системи BEP20

Проєктована система функціонування криптовалюти повинна містити точні взаємозв'язки між основними сутностями: Користувачами, Транзакціями, Адміністраторами, Токенами та Даними верифікації.

Використовуючи ERWin, для нашої системи були побудовані дві основні моделі:

- Логічна модель даних, яка відображає понятійні зв'язки у вигляді сутностей і зв'язків між ними;
- Фізична модель даних, яка вже конкретизує типи даних полів, їх обмеження та правила індексації для реалізації у СУБД.

При побудові логічної моделі використовувалися:

- Сутності: Wallet (Гаманець користувача), Token (Токен), Transaction (Транзакція), Verification (Верифікація користувача), Admin (Адміністратор).
- Атрибути сутностей: адреса гаманця, баланс, дата транзакції, сума транзакції, статус верифікації, статус адміністратора і т.д.
- Зв'язки між сутностями: "ініціює", "керує", "виконує", "підтверджує".

Також було заплановано введення обмежень цілісності даних:

- Забезпечення унікальності адреси гаманця;
- Забезпечення обов'язкової наявності запису про верифікацію перед виконанням великих транзакцій.

Можливості ERWin, що були використані в проєкті

- Створення типових моделей для прискорення розробки (наприклад, шаблон для таблиці токенів BEP20).
- Контроль атрибутів сутностей та їх обмежень (типи даних, первинні ключі, зовнішні ключі).

- Автоматичне створення документації на основі побудованої ER-діаграми.
- Функція порівняння моделей дозволила на етапі розробки співставити логічну і фізичну моделі та усунути можливі неузгодженості.

Таким чином, побудова ER-діаграми за допомогою ERWin Data Modeler стала важливим етапом, який дозволив:

- Створити точну структуру даних для майбутньої бази;
- Забезпечити цілісність і правильність взаємозв'язків у системі;
- Спростити подальшу реалізацію бази даних на фізичному рівні.

ER-діаграма представлена у додатку В.

2.2 Діаграма класів та кооперацій

У процесі проєктування системи функціонування криптовалюти на основі технології блокчейн BEP20 діаграми класів та кооперацій відіграють ключову роль у візуалізації структури, поведінки та взаємодії об’єктів системи. Вони дозволяють описати як статичну структуру програмного забезпечення (через діаграму класів), так і динамічну взаємодію об’єктів під час виконання операцій (через діаграму кооперацій). Таке представлення необхідне для ефективної реалізації логіки взаємодії користувачів із криптовалютною інфраструктурою, яка включає створення, передавання токенів, перегляд балансу, валідацію транзакцій тощо.

Діаграма класів

Діаграма класів у системі BEP20-криптовалюти відображає ключові сутності, які беруть участь у процесі функціонування криптовалютного токена та обміну ним. Основними класами є:

- Користувач (User) — абстракція кінцевого користувача, який взаємодіє із системою через криптогаманець. Має атрибути як-от адреса, публічний_ключ, баланс, а також методи: відправитиТокени(), отриматиТокени(), перевіритиБаланс().

- Токен (Token) — представляє цифрову одиницю обміну. Містить такі атрибути як назва, символ, загальнаКількість, адресаКонтракту. Методи: емісія(), спалити(), передати().
- Смарт-контракт (SmartContract) — головний елемент взаємодії з блокчейном. Він керує поведінкою токена за стандартом BEP20. Має методи approve(), transfer(), balanceOf(), allowance() та атрибути власник, карта_дозволів.
- Транзакція (Transaction) — одиниця взаємодії, яка фіксує переказ токенів. Містить відправник, одержувач, кількість, підпис, хеш, статус. Методи: підписати(), верифікувати().
- Блок (Block) — структура, що містить одну або більше транзакцій. Має атрибути номер_блоку, час, хеш_попереднього, хеш_блоку, список_транзакцій. Метод додатиТранзакцію().
- Мережа (BlockchainNetwork) — загальна інфраструктура, що координує розподілення блоків. Містить методи перевіритиДійсність(), додатиБлок().

Взаємозв’язки між цими класами мають різну природу:

- Асоціації: Користувач пов’язаний з Токеном через транзакції. Один користувач може ініціювати або отримати багато транзакцій.
- Композиція: Блок складається з транзакцій, які не можуть існувати без блоку.
- Агрегація: Смарт-контракт оперує токенами, але вони можуть існувати і незалежно від одного конкретного контракту (особливо в тестовому середовищі).
- Спадкування: клас "Адміністратор" може бути похідним від класу "Користувач", розширюючи доступ до адміністративних функцій, наприклад: паузаКонтракту(), зміна Власника().

Ця ієрархія дозволяє структурувати систему гнучко та масштабовано. Завдяки діаграмі класів можна легко моделювати, як зберігаються баланси, яким чином передаються токени і як обробляються транзакції, що є критично важливим для безпечної та надійної криптовалютної системи.

Діаграма кооперацій

Діаграма кооперацій у системі криптовалюти на базі BEP20 моделює сценарії взаємодії об’єктів під час виконання операцій із токенами. Вона фіксує часову послідовність і взаємозв’язки між об'єктами у процесі конкретного сценарію — наприклад, переказу токенів від одного користувача іншому.

Ключові сценарії кооперацій:

Переказ токенів між користувачами

Користувач ініціює транзакцію через свій гаманець → викликає метод transfer() у смарт-контракті → смарт-контракт перевіряє дозволи → списує токени з балансу відправника → додає токени на баланс отримувача → транзакція записується у новий блок → блок додається до блокчейну.

У кооперації бере участь: User, SmartContract, Transaction, Block, BlockchainNetwork.

Перевірка балансу

Користувач надсилає запит до смарт-контракту → викликається метод balanceOf(address) → смарт-контракт повертає поточний баланс.

Випуск токенів (емісія)

Адміністратор (власник контракту) викликає метод mint() → смарт-контракт створює нову кількість токенів і додає їх на баланс адміністратора → оновлена загальна кількість токенів зберігається в контракті.

Підпис транзакції

Користувач підписує транзакцію своїм приватним ключем → транзакція проходить валідацію → якщо підпис дійсний, вона передається до мережі.

Кожен об'єкт у діаграмі кооперацій виконує конкретну роль. Наприклад, смарт-контракт — центр логіки, який отримує всі запити; користувач — ініціатор дій; блокчейн — база, що забезпечує незмінність та перевірку.

Значення діаграм

Детально спроектовані діаграми класів та кооперацій дозволяють:

- Візуалізувати архітектуру BEP20-контракту.
- Забезпечити узгодженість між логікою смарт-контракту та зовнішніми користувачами.
- Спроектувати гнучкий код, який можна адаптувати до майбутніх змін (наприклад, додавання NFT-підтримки).
- Знизити складність впровадження та тестування системи.
- Формалізувати ролі користувачів (інвесторів, валідаторів, адміністратора).

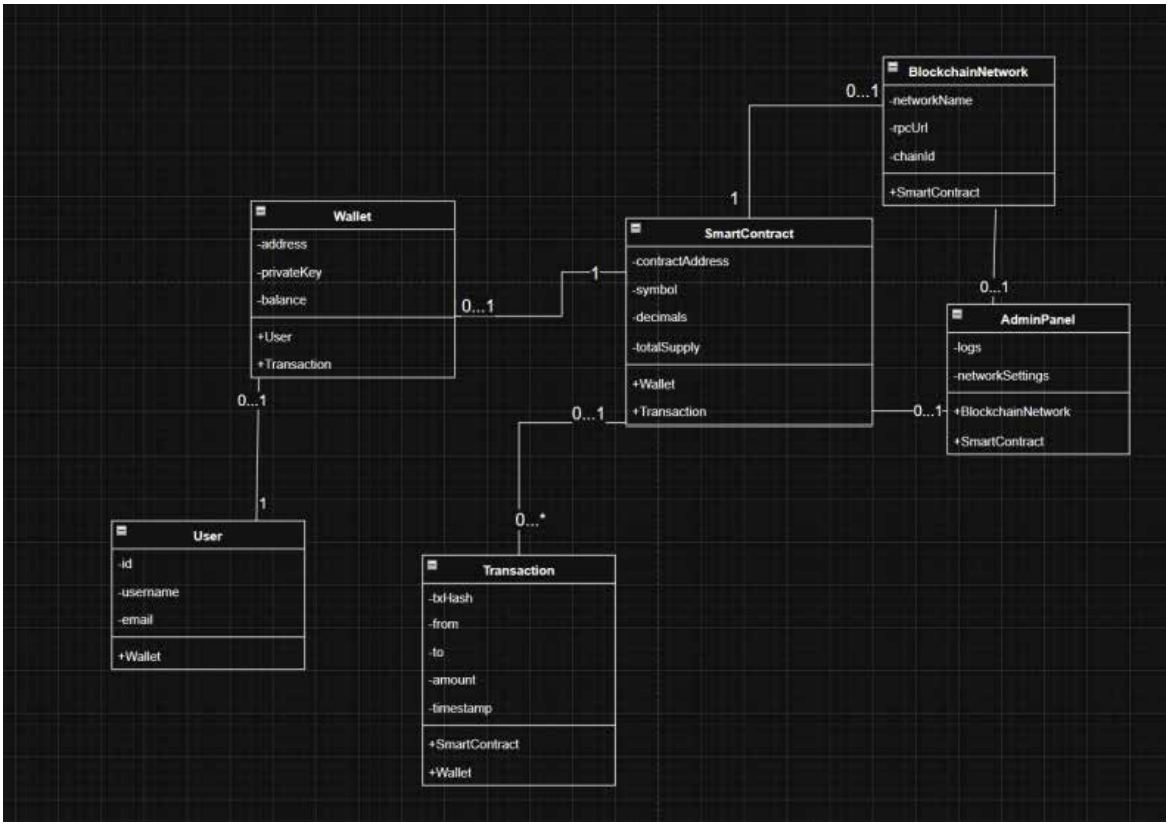


Рис. 2.1 Діаграма класів

2.3 Діаграма пакетів

Діаграма пакетів є важливим інструментом для представлення структури високорівневої архітектури системи. Вона дозволяє чітко розподілити систему на окремі компоненти або пакети, які містять класи, інтерфейси та інші елементи моделі. Це допомагає візуалізувати взаємозв'язки між модулями, зрозуміти, як компоненти взаємодіють один з одним і забезпечити легкість в управлінні та розширенні системи. У випадку розробки системи функціонування криптовалюти з використанням технології блокчейн BEP20, діаграма пакетів дозволяє побудувати чітку структуру з урахуванням різних аспектів системи, таких як обробка транзакцій, взаємодія з блокчейном, управління токенами та безпека.

Важливість діаграми пакетів

Діаграма пакетів для системи криптовалюти дозволяє логічно структурувати функціональність, зокрема для таких критичних частин, як управління гаманцями, проведення транзакцій, взаємодія з блокчейном, обробка користувачів і взаємодія з іншими сервісами. Це дозволяє:

Поділ на логічні модулі. Система буде розділена на кілька основних пакетів, що дозволить команді зосередитись на кожній частині окремо.

Оптимізація взаємодії компонентів. Визначення залежностей між модулями допомагає зрозуміти, які частини системи взаємодіють між собою, що дозволяє ефективніше проектувати й тестувати.

Забезпечення масштабованості та безпеки. Модульність дозволяє ефективно масштабувати систему та забезпечити високий рівень безпеки через ізоляцію окремих компонентів.

Основні пакети системи криптовалюти BEP20

Презентаційний рівень (Presentation Layer):

- Цей рівень включає компоненти, що відповідають за взаємодію з користувачем, а також забезпечують відображення даних на інтерфейсі. У випадку криптовалюти, це може бути веб-інтерфейс

або мобільний додаток, через який користувачі можуть перевіряти свої гаманці, здійснювати транзакції і переглядати баланс.

- Пакет може включати:
  - Клас для управління інтерфейсами користувача.
  - Компоненти для відображення статусу транзакцій.
  - Взаємодія з бізнес-логікою для ініціації транзакцій.

Бізнес рівень (Business Layer):

- Основна логіка системи зосереджена саме в цьому пакеті. Тут містяться основні компоненти для роботи з транзакціями, керування токенами, реалізація смарт-контрактів та управління криптовалютними операціями.
- Пакет може включати:
  - Клас для управління транзакціями.
  - Клас для генерації та обробки смарт-контрактів BEP20.
  - Логіка для перевірки стану балансу, обміну токенів.
  - Система для обробки помилок та валідації даних.

Рівень даних (Data Layer):

- Цей пакет відповідає за доступ до бази даних та її зберігання. Тут зберігаються всі транзакційні дані, інформація про користувачів і їх гаманці, а також історія виконаних операцій.
- Пакет може включати:
  - Компоненти для роботи з базою даних блокчейну.
  - Зберігання історії транзакцій та їх статусів.
  - Зберігання інформації про гаманці та токени користувачів.

Глобальні компоненти (Global Components):

- Цей пакет включає компоненти, що працюють на рівні всієї системи і забезпечують її цілісність, надійність та безпеку. Включає механізми для валідації даних, обробки помилок, захисту системи від несанкціонованого доступу та захисту від атак.
- Пакет може включати:

- Механізми для аутентифікації та авторизації користувачів.
- Компоненти для шифрування та безпеки даних.
- Обробка помилок і ведення журналу системи.

Інтеграція з зовнішніми сервісами (External Services Integration):

- Цей пакет відповідає за взаємодію системи з іншими платформами, такими як інші блокчейн-мережі, обмінники криптовалют або сторонні сервіси для перевірки транзакцій.
- Пакет може включати:
  - Модулі для підключення до інших блокчейн-систем.
  - Сервери для обробки запитів до сторонніх обмінників.

Діаграма пакетів представлена в додатку Д.

Пояснення зв'язків:

Користувацький інтерфейс взаємодіє з Транзакцією для ініціації операцій.

Транзакція записується в Блокчейн, що підтверджує операцію і зберігає її в Базі даних.

Смарт-контракт виконується через Блокчейн, для реалізації автоматичних операцій.

Гаманець взаємодіє з Базою даних для оновлення балансу користувача.

Безпека забезпечує захист даних і аутентифікацію користувачів.

2.4 Діаграма компонентів

Діаграма компонентів є важливим інструментом для моделювання архітектури програмного забезпечення, оскільки вона показує фізичну структуру системи через її компоненти та взаємодії між ними. У системі криптовалюти на основі технології блокчейн BEP20 діаграма компонентів дозволяє чітко визначити ключові елементи, такі як інтерфейси для взаємодії з користувачами, компоненти для обробки транзакцій, механізми безпеки та зв'язки між ними. Вона є основним інструментом для планування та організації функціональних

частин системи, що є критичними для забезпечення безпеки, надійності та масштабованості.

Головною метою діаграми компонентів є демонстрація основних елементів системи криптовалюти, їх взаємодії і точок інтеграції з іншими системами. Вона дозволяє візуалізувати, як працюють окремі частини системи, які сервіси та бібліотеки використовуються, і як здійснюється зв'язок між різними компонентами.

Презентаційний рівень (UI)

- Цей рівень відповідає за взаємодію з користувачем. Це інтерфейс, через який користувачі можуть переглядати свої баланси, здійснювати транзакції, отримувати підтвердження операцій тощо. Інтерфейси можуть бути представлені у вигляді веб-сторінок або мобільних додатків.
- Основні компоненти на цьому рівні:
  - Web-інтерфейс: інтерфейс для здійснення транзакцій, перевірки балансу та перегляду історії.
  - Мобільний додаток: для здійснення криптовалютних операцій на мобільних пристроях.
  - Кошик для транзакцій: дозволяє користувачам ініціювати та підтверджувати платежі.

Бізнес-логіка (Business Logic)

- На цьому рівні відбувається основна обробка бізнес-операцій, таких як створення, підтвердження та відправка транзакцій, взаємодія з блокчейн-мережею та генерація смарт-контрактів.
- Основні компоненти на цьому рівні:
  - Контролери транзакцій: управляють логікою виконання транзакцій, перевіряють їх правильність, відправляють їх на блокчейн.

- Генерація та виконання смарт-контрактів: відповідають за автоматичне виконання умов операцій згідно з визначеними смарт-контрактами на блокчейні.
- Перевірка балансу: перевірка наявності достатнього балансу перед здійсненням транзакції.

Рівень доступу до даних (Data Layer)

- Цей рівень забезпечує доступ до бази даних і зберігання важливої інформації, такої як гаманці користувачів, історія транзакцій і статуси смарт-контрактів.
- Основні компоненти:
  - Гаманець: відповідає за зберігання та обробку гаманців користувачів, їх баланси та історія операцій.
  - База даних блокчейну: зберігає транзакції, статуси смарт-контрактів, відомості про активи та інші необхідні дані.

Глобальні компоненти:

- Цей рівень включає компоненти, які забезпечують безпеку, аутентифікацію користувачів і захист даних в межах усієї системи.
- Основні компоненти:
  - Механізм безпеки: відповідає за шифрування даних, автентифікацію користувачів і забезпечення безпеки всіх транзакцій.
  - Аутентифікація та авторизація: системи для підтвердження особистості користувачів через їх облікові записи та криптографічні ключі.

Інтеграція з зовнішніми сервісами:

- Компоненти, що інтегрують систему з іншими платформами, такими як сторонні обмінники криптовалют, інші блокчейн-мережі або API для перевірки транзакцій та інформації про блоки.
- Основні компоненти:

- Обмінники криптовалют: інтеграція з обмінниками для переведення токенів або монет.
- Інтерфейс до іншої блокчейн-мережі: дозволяє здійснювати перевірки та транзакції в інших мережах, підтримуючи взаємодію з іншими типами криптовалют.

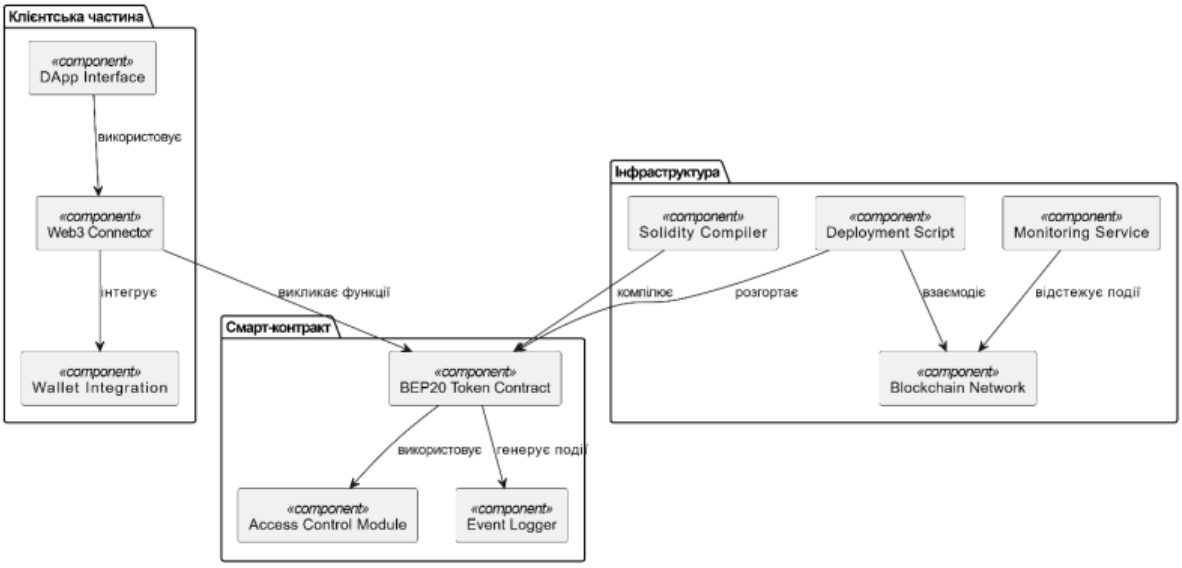


Рис. 2.2 Діаграма компонентів

3 Розробка інформаційного та програмного забезпечення

3.1 Система управління інформаційною базою

Система управління інформаційною базою в контексті криптовалюти на базі технології блокчейн BEP20 є основою для зберігання, обробки та аналізу даних, що надходять від користувачів, транзакцій та інших компонентів системи. У цьому випадку криптовалютна система має специфічні вимоги до зберігання даних і управління ними, оскільки працює в рамках дистрибутивної та нерегульованої структури блокчейн-мережі, що вимагає застосування нових підходів до обробки та збереження інформації.

Для криптовалютної системи на основі технології BEP20 важливою складовою є ефективне зберігання даних про транзакції, баланси користувачів, а також інформацію про смарт-контракти та їхні взаємодії. На відміну від традиційних централізованих систем, де дані зберігаються на єдиному сервері, блокчейн забезпечує дистрибутивну модель, де дані реплікуються між численними вузлами мережі. Це дозволяє забезпечити високу відмовостійкість і безпеку даних, але також вимагає застосування специфічних технологій для обробки транзакцій в реальному часі.

Крім того, необхідно використовувати спеціалізовані бази даних для зберігання метаданих, які не зберігаються безпосередньо в блокчейні, але є важливими для функціонування криптовалюти. Це можуть бути бази даних для зберігання інформації про користувачів, їх транзакції, баланси та історію операцій.

Вибір між централізованою та розподіленою системою зберігання даних для криптовалюти залежить від конкретних вимог до масштабованості, безпеки та продуктивності. У випадку з BEP20, блокчейн сам по собі є розподіленою системою, де кожен блок зберігає частину інформації про транзакції. Це дозволяє досягти високого рівня безпеки та прозорості, оскільки всі зміни в системі

фіксуються у вигляді незмінних записів у блоках, які перевіряються та підтверджуються мережею.

Проте для деяких типів даних, таких як інформація про користувачів або адміністрування системи, доцільно застосовувати централізовані бази даних, які будуть зберігати необхідну метаінформацію про користувачів, їх акаунти, налаштування та історію взаємодії з системою. Такі бази даних можуть бути використані для зберігання даних, які не є критичними для самого процесу транзакцій в блокчейні, але необхідні для роботи з криптовалютою на рівні додатку.

Для зберігання метаданих, таких як профілі користувачів, статистика, історія транзакцій, налаштування мережі та інші операції, використовуються різні типи систем управління базами даних (СУБД). Вибір конкретної СУБД залежить від типу даних і вимог до продуктивності системи.

1. Реляційні СУБД: Вони є традиційним вибором для фінансових систем і застосовуються в криптовалютних додатках для зберігання структурованої інформації, такої як дані про користувачів, їх рахунки та баланси. Реляційні бази, наприклад, PostgreSQL або MySQL, можуть забезпечити високу узгодженість даних, ефективну підтримку транзакцій (ACID) та можливість виконання складних запитів для обробки великих обсягів інформації.
2. Документо-орієнтовані СУБД: Такі СУБД, як MongoDB, дозволяють зберігати неструктуровані дані, наприклад, додаткову інформацію про транзакції або контекст транзакцій в блокчейні. Вони забезпечують високу масштабованість і гнучкість у зберіганні даних, які можуть бути динамічними або змінюватися з часом.
3. Гібридні СУБД: Для криптовалютних систем, де є вимога до швидкої обробки транзакцій і збереження неструктурованих даних, інколи використовуються гібридні СУБД, які поєднують риси реляційних і документо-орієнтованих баз. Наприклад, Cassandra є розподіленою базою

даних, яка забезпечує високу доступність і масштабованість, що є важливим для криптовалютної системи з великою кількістю транзакцій.

Для криптовалютних систем важливим є забезпечення цілісності даних та можливість виконання транзакцій в рамках СУБД, що зберігають метадані.

Нормалізація даних допомагає усунути дублювання, зменшити аномалії та підвищити продуктивність запитів, особливо коли мова йде про складні запити до даних, які стосуються профілю користувача, його балансу та історії транзакцій.

Для нашої криптовалютної системи, що використовує технологію BEP20, найкращим варіантом є використання реляційних СУБД, таких як PostgreSQL або MySQL, оскільки ці системи дозволяють ефективно працювати з транзакційними даними та забезпечують цілісність даних при високих навантаженнях.

PostgreSQL є відкритим продуктом з підтримкою складних запитів та високою надійністю. Вона також підтримує функціональність JSON, що дозволяє працювати з неструктурованими даними, які можуть зберігатися для додаткової інформації, що надходить з блокчейну.

MySQL, у свою чергу, також є відкритим і безкоштовним рішенням, яке забезпечує високу продуктивність і підтримку транзакцій, що є важливим для обробки великої кількості криптовалютних транзакцій.

Для нашої системи обрано PostgreSQL**,** оскільки ця СУБД забезпечує високий рівень безпеки, підтримує складні запити та є гнучким рішенням для обробки як структурованих, так і неструктурованих даних, що особливо важливо для інтеграції з блокчейн-мережею(див. табл.3.1).

Порівняльний аналіз СУБД для криптовалютних систем

Таблиця 3.1

СУБД	**Переваги**	**Недоліки**
PostgreSQL	Відкрите ПЗ, висока надійність, підтримка JSON	Складність налаштування, менш популярне, ніж MySQL
MySQL	Висока продуктивність, поширена підтримка	Менша надійність порівняно з PostgreSQL
MongoDB	Гнучкість, підтримка неструктурованих даних	Відсутність підтримки транзакцій ACID

3.2 Розробка інформаційної бази

Процес розробки інформаційної бази для системи криптовалюти, яка використовує технологію блокчейн BEP20, має свої особливості, що пов'язані з вимогами безпеки, масштабованості та високої ефективності обробки транзакцій. У цьому контексті важливо розглядати не тільки традиційні бази даних для зберігання інформації, а й інтеграцію з блокчейн-технологіями для забезпечення надійності, прозорості та безпеки даних.

В основі архітектури бази даних для криптовалюти на технології BEP20 лежить використання дистрибуційної моделі з елементами централізованої та розподіленої структури. Блокчейн буде використовуватися для запису всіх транзакцій, тоді як традиційні бази даних можуть бути застосовані для зберігання додаткової інформації, яка не потребує постійного оновлення або яка не підлягає безпосередньо обробці на блокчейні (наприклад, інформація про користувачів, їх акаунти, мета-дані тощо).

Для забезпечення цілісності даних використовуються механізми зовнішніх ключів та транзакцій, які дозволяють синхронізувати дані між основною базою

даних і блокчейном. Враховуючи, що транзакції на блокчейні є незмінними після їх підтвердження, всі операції, які стосуються криптовалютних переказів, повинні бути відображені в блокчейні, а додаткові дані — в традиційних базах даних, таких як реляційні СУБД.

База даних для криптовалюти повинна містити кілька ключових компонентів:

1. Користувачі: Зберігає дані про користувачів системи, включаючи їх ідентифікаційні номери (адреси гаманців), особисту інформацію (якщо це необхідно), статуси акаунтів та історію транзакцій.
2. Транзакції: Зберігає записи про кожну криптовалютну операцію, що включає інформацію про відправника, отримувача, суму, тип транзакції, дату та статус.
3. Баланс: Структура, що забезпечує зберігання балансу користувача в різних валютах, включаючи криптовалюту, що базується на стандартах BEP20.
4. Сесії: Дані про поточні сесії користувачів, що дозволяють відслідковувати активність користувачів на платформі, їхні авторизації та транзакції.
5. Контракти: Інформація про смарт-контракти, включаючи їх адреси, статуси виконання та умови.

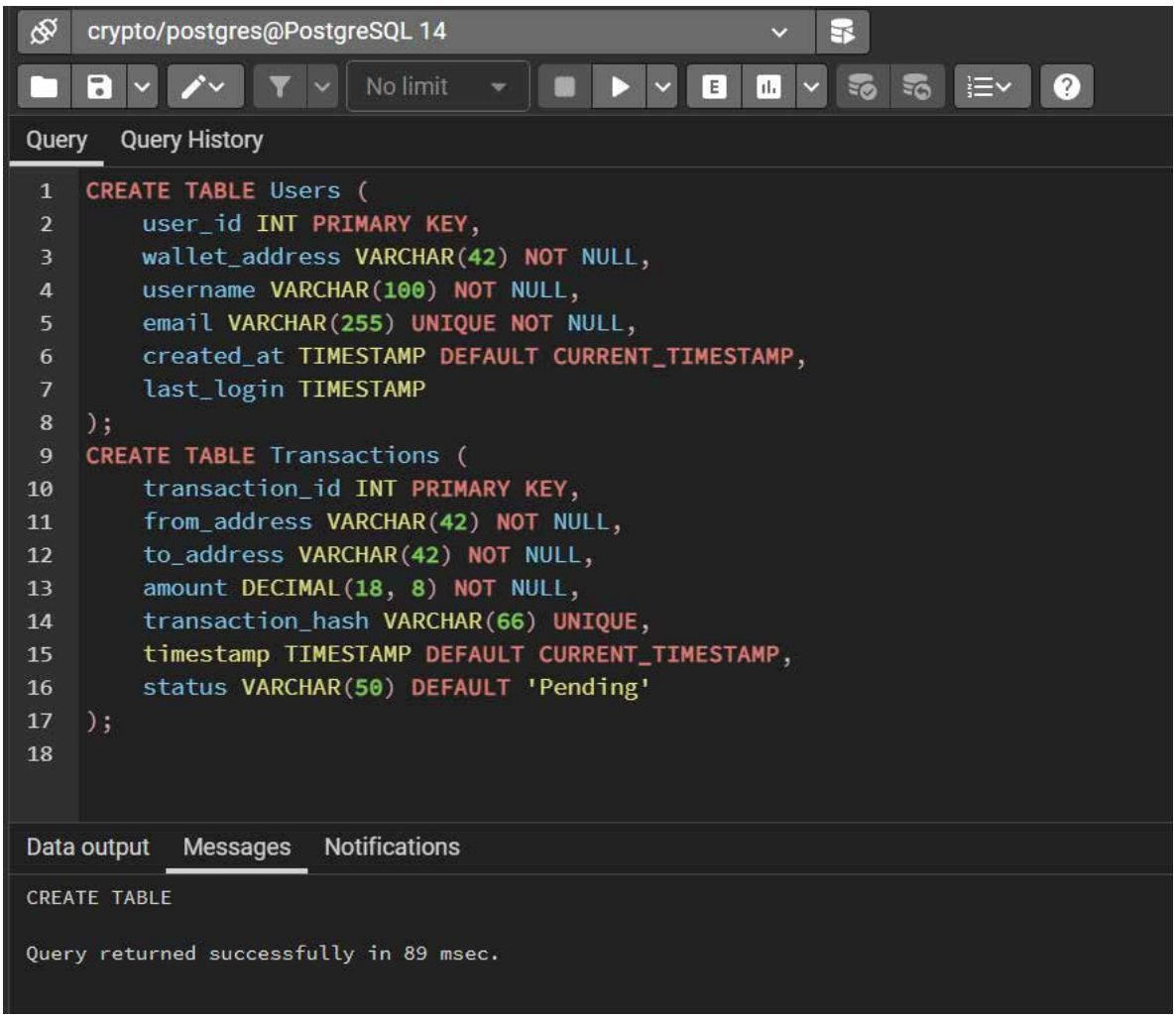


Рис. 3.1 Запит на створення таблиці в БД

3.3 Вибір інструментарію для створення прикладного програмного забезпечення

Для створення програмного забезпечення, яке реалізує функціонування криптовалюти за стандартом BEP20 на платформі Binance Smart Chain (BSC), необхідно було підійти до вибору інструментарію максимально відповідально. Вибір технологій безпосередньо впливає на надійність, безпеку, масштабованість та швидкість розробки системи.

Основними критеріями стали: підтримка написання та деплою смарт-контрактів, інтеграція з блокчейн-мережами та підтримка Web3-технологій, зручність

розробки DApp-фронтенду та бекенду, масштабованість проєкту та можливість його подальшого розвитку, безпечне зберігання і обробка даних користувачів.

Перед остаточним вибором розглядалися альтернативи серед мов програмування і середовищ розробки.

Порівняння мов програмування

3.3.1 JavaScript/TypeScript

Широко використовується для веб-розробки та створення взаємодії із блокчейном через бібліотеки Web3.js або Ethers.js.

Переваги: пряма інтеграція з браузером, велика кількість готових рішень для роботи з Web3, висока швидкість розробки прототипів.

Недоліки: потреба в суворих підходах до типізації (виправляється використанням TypeScript), можливі ризики безпеки при неправильній організації коду.

3.3.2 Solidity

Спеціалізована мова для створення смарт-контрактів на блокчейнах з Ethereum Virtual Machine, таких як Binance Smart Chain.

Переваги: спеціалізація на децентралізованих додатках, велика кількість прикладів стандарту BEP20, можливість аудиту безпеки коду.

Недоліки: вузька сфокусованість — потрібні окремі середовища для розробки, тестування та деплою контрактів.

Порівняння середовищ розробки

3.3.3 Visual Studio Code

Найпопулярніше кросплатформене середовище розробки для JavaScript, TypeScript і навіть Solidity (через відповідні плагіни).

Переваги:підтримка роботи з Web3, Solidity, Hardhat, інтеграція із системами контролю версій (GitHub), розширюваність і підтримка величезної кількості плагінів.

Недоліки: потреба налаштовувати середовище вручну для оптимальної роботи з блокчейн-проєктами.

3.3.5 Remix IDE

Браузерне середовище, яке ідеально підходить для створення, тестування і деплою смарт-контрактів на Solidity.

Переваги:не потребує встановлення — доступний у браузері, інтеграція з тестовими мережами через MetaMask, підтримка статичного аналізу коду для перевірки безпеки контрактів.

Недоліки: обмежена зручність для великих комплексних систем, не підходить для роботи над фронтенд/бекенд-розробкою.

Враховуючи специфіку розробки смарт-контрактів та побудову повноцінної системи функціонування криптовалюти BEP20, остаточний стек технологій (табл.3.2).

Вибір технологій для системи BEP20

Таблиця 3.2

Компонент системи	Інструмент	Причина вибору
Смарт-контракти	Solidity + Remix IDE	Спеціалізоване середовище для створення і тестування контрактів
Фронтенд (DApp)	Next.js + TailwindCSS + Ethers.js	Швидкий розвиток і легкість інтеграції з Web3
Бекенд	Node.js (Express.js)	Масштабованість, простота роботи з API та базами даних

Компонент системи	Інструмент	Причина вибору
База даних	PostgreSQL	Стабільність і надійність при роботі з великими обсягами даних
IDE	Visual Studio Code	Гнучкість, підтримка багатьох мов, інтеграція з Web3

Мова програмування Solidity була обрана як основна через її пряму орієнтацію на розробку смарт-контрактів під мережу Binance Smart Chain та відповідність стандарту BEP20. Solidity дозволяє гнучко та безпечно реалізувати логіку токенів, систем винагород, верифікацій та інших механізмів криптовалютної системи.

Середовище Remix IDE обрано через простоту налаштування, миттєву компіляцію і зручний деплой у тестові та реальні мережі через Web3-провайдери. Це особливо важливо для проєкту, де потрібно багаторазово тестувати контракти, виправляти вразливості й забезпечувати відповідність високим стандартам безпеки.

Доповнення стеку Next.js для фронтенду та Node.js для бекенду дає змогу будувати повноцінну екосистему: від користувацького інтерфейсу до інтеграції з блокчейном та зовнішніми сервісами, такими як API для бірж або сервісів верифікації.

База даних PostgreSQL була обрана завдяки своїй безкоштовності, потужності, підтримці ACID-транзакцій і масштабованості, що важливо при обробці облікових записів користувачів, їхніх токенів і історії транзакцій.

Таким чином, обраний інструментарій забезпечує стабільну, масштабовану і безпечну основу для розробки системи функціонування криптовалюти з використанням технології блокчейн BEP20.

3.4 Алгоритмізація та програмування програмних модулів

Алгоритмізація та програмування програмних модулів є критичними етапами у створенні функціональної системи криптовалюти, що базується на стандарті BEP20. Ця система включає не лише розробку смарт-контракту для самої криптовалюти, але й створення веб-інтерфейсу для взаємодії з користувачем, обробку транзакцій, реєстрацію, автентифікацію, а також механізми покупки токенів через ваш вебсайт.

В процесі алгоритмізації розробляються блок-схеми, які відображають логіку роботи основних модулів — від генерації гаманця користувача та інтеграції з Web3, до обробки платежів через MetaMask, оновлення балансу в реальному часі та підтвердження транзакцій у блокчейні Binance Smart Chain (BSC).

Програмна реалізація модулів виконується переважно мовами Solidity (для смарт-контрактів), JavaScript/TypeScript (для інтеграції Web3 на стороні клієнта) та Node.js або Python (для серверної частини, яка взаємодіє з базою даних та виконує логіку управління користувачами й сесіями). Основна мета — забезпечити надійну, прозору та безпечну взаємодію між користувачем і криптовалютою, яку можна придбати на сайті.

3.4.1 Алгоритм авторизації користувача

Авторизація в системі покупки криптовалюти через сайт — ключова функція, що гарантує доступ лише автентифікованим користувачам до персоналізованих функцій, таких як перегляд гаманця, виконання транзакцій, перевірка балансу та історії операцій.

Код для виконання усіх алгоритмів представлений у додатку И.

Алгоритм виглядає наступним чином:

1. Користувач переходить на сторінку входу та вводить логін (або email) і пароль.

2. Система перевіряє, чи заповнені всі поля.
3. Відбувається запит до бази даних для отримання інформації про користувача.
4. Якщо користувача не знайдено — виводиться повідомлення про помилку.
5. Якщо користувача знайдено — перевіряється хеш пароля (за допомогою, наприклад, bcrypt).
6. У разі успіху створюється сесія або генерується JWT токен для подальшої взаємодії.
7. Користувач перенаправляється до особистого кабінету, де може переглядати гаманець та купувати токени.

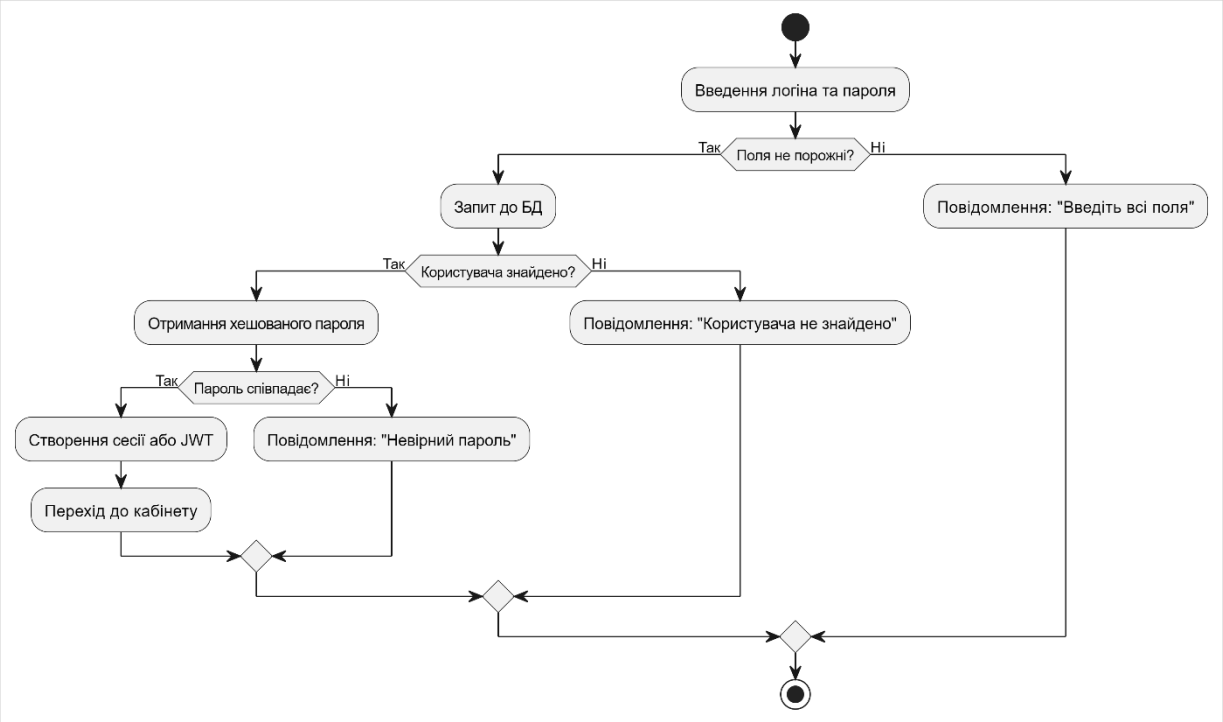


Рис. 3.2 Блок-схема алгоритму авторизації

3.4.2 Алгоритм генерації гаманця користувача

У системі кожен користувач має унікальну адресу гаманця BEP20, яка або генерується автоматично на сервері, або користувач може підключити існуючий гаманець (наприклад, MetaMask).

Етапи генерації нового гаманця:

1. При реєстрації або після авторизації користувач обирає опцію "Створити гаманець".
2. Система за допомогою бібліотеки web3.js або ethers.js генерує нову пару приватного/публічного ключів.
3. Приватний ключ зашифровується і зберігається у захищеному вигляді (наприклад, у вигляді зашифрованого JSON).
4. Користувачу показується лише публічна адреса.
5. Адреса додається до профілю користувача у базі даних.

Рис. 3.3 Блок-схема алгоритму генерації гаманця

3.4.3 Алгоритм покупки токенів через сайт

Основна функція сайту — можливість придбати BEP20-токени безпосередньо через інтерфейс. Це реалізується за допомогою інтеграції з MetaMask або іншими Web3-гаманцями.

Алгоритм дій користувача:

1. Користувач переходить на сторінку покупки.
2. Підключає MetaMask.

3. Вибирає кількість токенів, яку хоче купити.
4. Система обчислює необхідну кількість BNB або іншої підтримуваної валюти.
5. Користувач підтверджує транзакцію у MetaMask.
6. Смарт-контракт приймає платіж і надсилає токени на адресу користувача.
7. Транзакція записується у блокчейн, а статус покупки оновлюється на сайті.

На сервері може працювати вебхуками або інтервальною перевіркою стану транзакції через BSCScan API чи інші засоби.

Рис. 3.4 Блок-схема алгоритму покупки токенів

3.4.4 Алгоритм перевірки балансу токенів

Цей модуль дозволяє користувачу переглянути актуальний баланс його BEP20-токенів у реальному часі:

1. Користувач заходить у профіль.
2. Відправляється запит через Web3 до смарт-контракту токена (використовується метод balanceOf()).

3. Отриманий баланс конвертується у зручний формат (наприклад, відображається з 2 знаками після коми).
4. Баланс оновлюється на сторінці автоматично або вручну.

Скріншоти з сайту та функціоналом покупки монет представлені в додатках Е та Ж.

Рис. 3.5 Блок-схема алгоритму перевірки балансу

3.4.5 Алгоритм підтвердження транзакції

Коли користувач здійснює купівлю токенів, транзакція повинна бути підтверджена мережею:

1. Після ініціації транзакції клієнт отримує хеш транзакції.
2. Система на бекенді (через API BSCScan або Web3) регулярно перевіряє статус цієї транзакції.
3. Як тільки транзакція отримує необхідну кількість підтверджень, користувачу надсилається повідомлення про успішну покупку.
4. Баланс оновлюється.

Рис. 3.6 Блок-схема алгоритму підтвердження транзакції

4 РЕКОМЕНДАЦІЇ ЩОДО ВПРОВАДЖЕННЯ ТА ЕКСПЛУАТАЦІЇ СИСТЕМИ

4.1 Тестування системи

Тестування — це один із найважливіших етапів впровадження інформаційної системи, особливо якщо мова йде про систему функціонування криптовалюти на базі технології блокчейн BEP20. Забезпечення надійності, безпеки та відповідності всім технічним вимогам стає критично важливим фактором успішного запуску проєкту.

Відповідно до міжнародного стандарту IEEE Std 829-1983, тестування визначається як процес, який має на меті виявлення відмінностей між реальними та очікуваними властивостями програмного продукту, а також оцінку його якості. При цьому особливу увагу приділяється саме виявленню дефектів, невідповідностей та слабких місць у системі.

Головні завдання тестування включають:

- Виявлення і документування всіх знайдених дефектів.
- Перевірку відповідності розробленого продукту технічному завданню.
- Оцінку продуктивності та безпеки рішення.
- Підготовку рекомендацій для вдосконалення продукту.

Особливо важливим є тестування в системах, що працюють з фінансовими транзакціями, такими як криптовалютні системи, оскільки тут навіть мінімальні помилки можуть призвести до значних матеріальних втрат.

Процес тестування системи розробки криптовалюти BEP20 був побудований за класичними методиками забезпечення якості програмного забезпечення, з урахуванням специфіки блокчейн-технологій і особливих вимог до роботи з криптоактивами.

Етапи тестування включали:

Проведення тестування смарт-контрактів

Основна частина перевірок припадала на смарт-контракти, написані мовою Solidity. Була проведена повна верифікація:

- Виконання основних функцій стандарту BEP20 (transfer, approve, transferFrom, mint, burn).
- Перевірка коректного генерування подій (Transfer, Approval) після кожної транзакції.
- Тестування поведінки контракту при граничних значеннях.
- Аналіз вразливостей типу "переповнення чисел", "атаки повторного виклику" та інших.

Кожен тест виконувався як вручну (через MetaMask), так і автоматизовано.

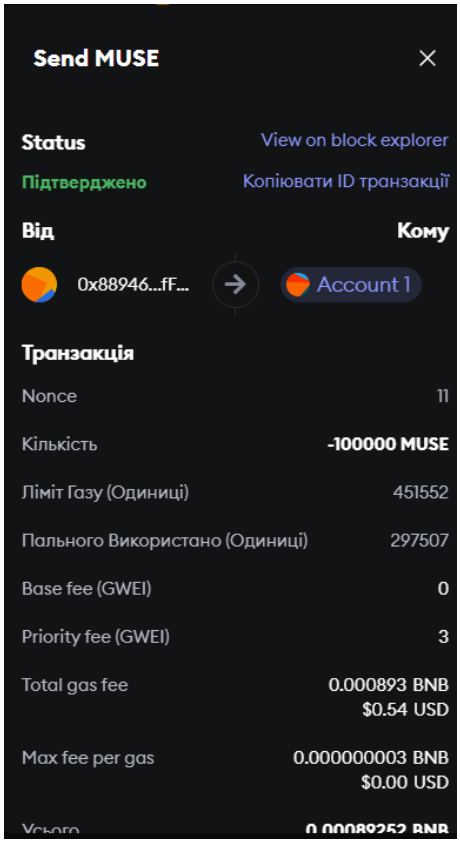


Рис. 4.1 Скріншот тестового переказу

У тестуванні застосовувалися наступні методи:

Тестування «білої скриньки»

При тестуванні смарт-контрактів використовувався доступ до їхнього вихідного коду: відстежувалися всі логічні гілки в коді, перевірялися всі можливі

комбінації вхідних параметрів, проводився аналіз покриття тестами всіх функцій контракту.

Тестування «чорної скриньки»

Користувацька частина системи тестувалася виключно через стандартні інтерфейси:

- Перевірялися переходи між сторінками.
- Тестувалися функціональність кнопок, форм введення та повідомлень про помилки.
- Симулювалися запити на виконання транзакцій через гаманці MetaMask.

Стратегії тестування безпеки

Враховуючи, що наша система працює з фінансовими транзакціями, особливу увагу було приділено тестуванню безпеки:

- Перевірка коректності обмежень ролей (тільки адміністратор може створювати нові токени).
- Аналіз механізмів верифікації транзакцій.
- Тестування можливості злому API бекенду через SQL-ін'єкції або XSS-атаки.

4.3 Склад інсталяційного пакету

Розробка системи функціонування криптовалюти з використанням технології блокчейн BEP20 вимагає ретельного підбору апаратних та програмних засобів, оскільки вони безпосередньо впливають на продуктивність, безпеку та стабільність мережі. BEP20 — це технічний стандарт токенів у блокчейні Binance Smart Chain (BSC), який забезпечує сумісність з Ethereum Virtual Machine (EVM) і дозволяє створювати смарт-контракти, токени та децентралізовані додатки (DApps).

Сучасні блокчейн-системи мають модульну архітектуру, де кожен компонент виконує певну функцію:

- Ноди (вузли) – комп’ютери, що підтримують роботу блокчейну, перевіряють транзакції та синхронізують дані.
- Сервери баз даних – зберігають інформацію про транзакції, смарт-контракти та стан мережі.
- Клієнтські пристрої – забезпечують взаємодію користувачів з системою (гаманці, DApps).

Для стабільної роботи блокчейн BEP20 необхідно розгорнути серверні вузли, які будуть обробляти транзакції та виконувати смарт-контракти. Оскільки BSC є EVM-сумісним, його можна розгортати на стандартному обладнанні, але з урахуванням високих вимог до продуктивності.

Апаратні вимоги для серверів

Для роботи нод BSC та серверів баз даних необхідні мінімальні та рекомендовані параметри.

Апаратні вимоги для серверів

Таблиця 4.1

Ресурс	Мінімальний	Рекомендований
Процесор	4-ядерний, 2.4 ГГц	8-ядерний, 3.5 ГГц+
Оперативна пам’ять	8 ГБ DDR4	16 ГБ DDR4 або більше
Жорсткий диск	SSD 256 ГБ	NVMe SSD 1 ТБ+
Мережа	100 Мбіт/с	1 Гбіт/с
ОС	Linux (Ubuntu 20.04 LTS)	Linux (Ubuntu/Debian)

- Процесор: BSC використовує консенсус Proof of Staked Authority (PoSA), який потребує потужних CPU для обробки транзакцій.
- Оперативна пам’ять: Великий обсяг RAM необхідний для обробки смарт-контрактів та синхронізації блоків.
- Диск: NVMe SSD забезпечує швидкий доступ до даних, що критично для блокчейну.
- Мережа: Висока пропускна здатність зменшує затримки при передачі даних між вузлами.

Клієнтські апаратні вимоги

Таблиця 4.1

Ресурс	Мінімальний	Рекомендований
Процесор	2-ядерний, 1.8 ГГц	4-ядерний, 2.5 ГГц+
Оперативна пам’ять	4 ГБ	8 ГБ
Жорсткий диск	128 ГБ HDD	256 ГБ SSD
ОС	Windows 10 / macOS 10+	Windows 11 / Linux

Для розгортання блокчейн BEP20 необхідне таке ПЗ:

Серверна частина

- Операційна система: Linux (Ubuntu/Debian) – найстабільніша для нод.
- Блокчейн-клієнт: Geth або Nethermind для підключення до BSC.
- База даних: LevelDB (вбудована в BSC) або PostgreSQL для додаткових сервісів.
- Сервер смарт-контрактів: Node.js + Web3.js / Ethers.js.

Клієнтська частина

- Гаманці: MetaMask, Trust Wallet (підтримка BEP20).
- Інструменти розробки: Remix IDE, Hardhat, Truffle.
- Мережеві утиліти: MyEtherWallet (для BSC).

Розробка системи на основі BEP20 вимагає ретельної оптимізації апаратних та програмних ресурсів. Основний акцент слід робити на потужні серверні ноди з SSD та високошвидкісним інтернетом, а також на безпеку мережі.
Використання сучасних фреймворків (Hardhat, Web3.js) дозволить забезпечити стабільну роботу смарт-контрактів та токенів стандарту BEP20.

Система функціонування криптовалюти з використанням технології BEP20 складається з декількох основних компонентів, які формують інсталяційний пакет. Цей пакет необхідний для розгортання смарт-контракту та забезпечення роботи веб-інтерфейсу для взаємодії з криптовалютою.
До складу інсталяційного пакету входить:

- Смарт-контракт BEP20, написаний мовою Solidity. Він компілюється та деплоїться в мережу Binance Smart Chain за допомогою середовища Hardhat або Remix IDE. Після публікації контракту користувачі можуть взаємодіяти з токеном у своїх гаманцях, таких як MetaMask.
- Фронтенд-додаток, створений з використанням сучасних веб-технологій (наприклад, React.js). Він дозволяє користувачам переглядати інформацію про токен, підключати гаманець та виконувати транзакції з токеном. Усі необхідні дані про контракт підключаються через ABI-файл та Web3-бібліотеку.

- Конфігураційні файли (наприклад, .env, hardhat.config.js) для підключення до мережі BSC, збереження приватних ключів, RPC-адрес та параметрів мережі.
- Готова збірка сайту, яка може бути розміщена на будь-якому хостингу, наприклад GitHub Pages, Vercel або Firebase. Після публікації сайт доступний для користувачів через доменне ім’я.

Таким чином, інсталяційний пакет забезпечує повний цикл функціонування криптовалюти: від запуску смарт-контракту до надання веб-доступу для користувачів. Установка не потребує класичної інсталяції — достатньо завантажити сайт на хостинг та підключити контракт до мережі.

ВИСНОВКИ

У процесі виконання дипломної роботи розроблено програмне забезпечення для функціонування криптовалюти на основі технології блокчейн BEP20. Основною метою даної роботи було створення повноцінної децентралізованої криптовалюти, яка дозволяє здійснювати емісію, зберігання та передавання токенів, а також взаємодію користувача з криптовалютою через зручний веб-інтерфейс. Поставлену мету вдалося досягти, оскільки створена система забезпечує базову функціональність сучасної токен-економіки.

Під час реалізації проєкту проведено детальний аналіз функціональних можливостей стандарту BEP20, а також дослідження наявних прикладів криптовалют у мережі Binance Smart Chain. Це дозволило виявити основні особливості та сформулювати чітке технічне завдання для розробки смарт-контракту. Створена система включає смарт-контракт, реалізований мовою Solidity, який забезпечує облік токенів, перевірку балансу, передачу активів між адресами, а також можливість взаємодії з гаманцем користувача.

На основі отриманих результатів спроєктовано і реалізовано вебсайт, який дозволяє користувачам купувати криптовалюту, переглядати баланс токенів, а також здійснювати транзакції. Для зв’язку з блокчейном було використано бібліотеки Web3.js та Ethers.js, що забезпечують високу швидкодію і стабільність у роботі з BSC. Система має простий і зрозумілий інтерфейс, орієнтований як на досвідчених користувачів, так і на новачків у сфері криптовалют.

Однією з ключових переваг розробленої системи є її децентралізованість і відсутність потреби у централізованому сервері для основних операцій з токенами. Це дозволяє гарантувати високу надійність, безпеку і прозорість усіх транзакцій, що виконуються в системі.

На етапі тестування було перевірено працездатність смарт-контракту та веб-інтерфейсу в мережі BEP-20. Результати показали стабільну роботу системи, правильне оброблення транзакцій, а також коректне оновлення даних у гаманці користувача.

Система продемонструвала відповідність поставленим функціональним і технічним вимогам, зокрема щодо безпеки, швидкодії та зручності користування. Вона забезпечує ефективну реалізацію процесів купівлі, зберігання і передачі токенів, а також надає простий механізм для впровадження токена у бізнес-моделі або спільноті.

У перспективі можливе подальше розширення функціоналу системи. Зокрема, можна реалізувати мобільну версію додатку, інтегрувати мультивалютну підтримку, додати панель адміністратора для аналітики обігу токена, або розробити механізми стейкінгу та NFT-функціоналу. Крім того, є сенс розширити інтеграцію з DeFi-платформами, що дозволить підвищити ліквідність токена та залучити нових користувачів.

Отже, створена система функціонування криптовалюти BEP20 повністю готова до впровадження та подальшого використання в реальному середовищі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Binance Smart Chain Documentation [Електронний ресурс]. – Режим доступу: https://docs.bnbchain.org

2. Ethereum Smart Contract Best Practices [Електронний ресурс]. – Режим доступу: https://consensys.github.io/smart-contract-best-practices/

3. BEP20 Token Standard [Електронний ресурс]. – Режим доступу: https://github.com/binance-chain/BEPs/blob/master/BEP20.md

4. Solidity Documentation [Електронний ресурс]. – Режим доступу: https://docs.soliditylang.org

5. OpenZeppelin Contracts Library [Електронний ресурс]. – Режим доступу: https://docs.openzeppelin.com/contracts

6. Метєлєв Д.І. Основи технології блокчейн. – Київ: Видавництво Ліра-К, 2020. – 236 с.

7. Nakamoto S. Bitcoin: A Peer-to-Peer Electronic Cash System [Електронний ресурс]. – Режим доступу: https://bitcoin.org/bitcoin.pdf

8. Журавльов А.І., Гончарук Т.П. Технології Web 3.0: Блокчейн та децентралізовані застосунки. – Харків: ХНУРЕ, 2021. – 185 с.

9. Mastering Blockchain / Imran Bashir. – 3rd ed. – Packt Publishing, 2020. – 786 p.

10. Ethereum White Paper [Електронний ресурс]. – Режим доступу: https://ethereum.org/en/whitepaper/

11. Козак В.І. Основи розробки децентралізованих застосунків. – Львів: ЛНУ, 2022. – 214 с.

12. Remix IDE – Solidity IDE for Smart Contracts [Електронний ресурс]. – Режим доступу: https://remix.ethereum.org

13. Truffle Suite Documentation [Електронний ресурс]. – Режим доступу: https://trufflesuite.com/docs

14. Ganache – Personal Blockchain for Ethereum Development [Електронний ресурс]. – Режим доступу: https://trufflesuite.com/ganache

15. Hardhat – Ethereum development environment [Електронний ресурс]. – Режим доступу: https://hardhat.org

16. Салатова І.М., Клименко А.В. Використання технології блокчейн у фінансових технологіях. – Економіка і управління, 2021. – №3. – С. 92–98.

17. ERC-20 Token Standard [Електронний ресурс]. – Режим доступу: https://eips.ethereum.org/EIPS/eip-20

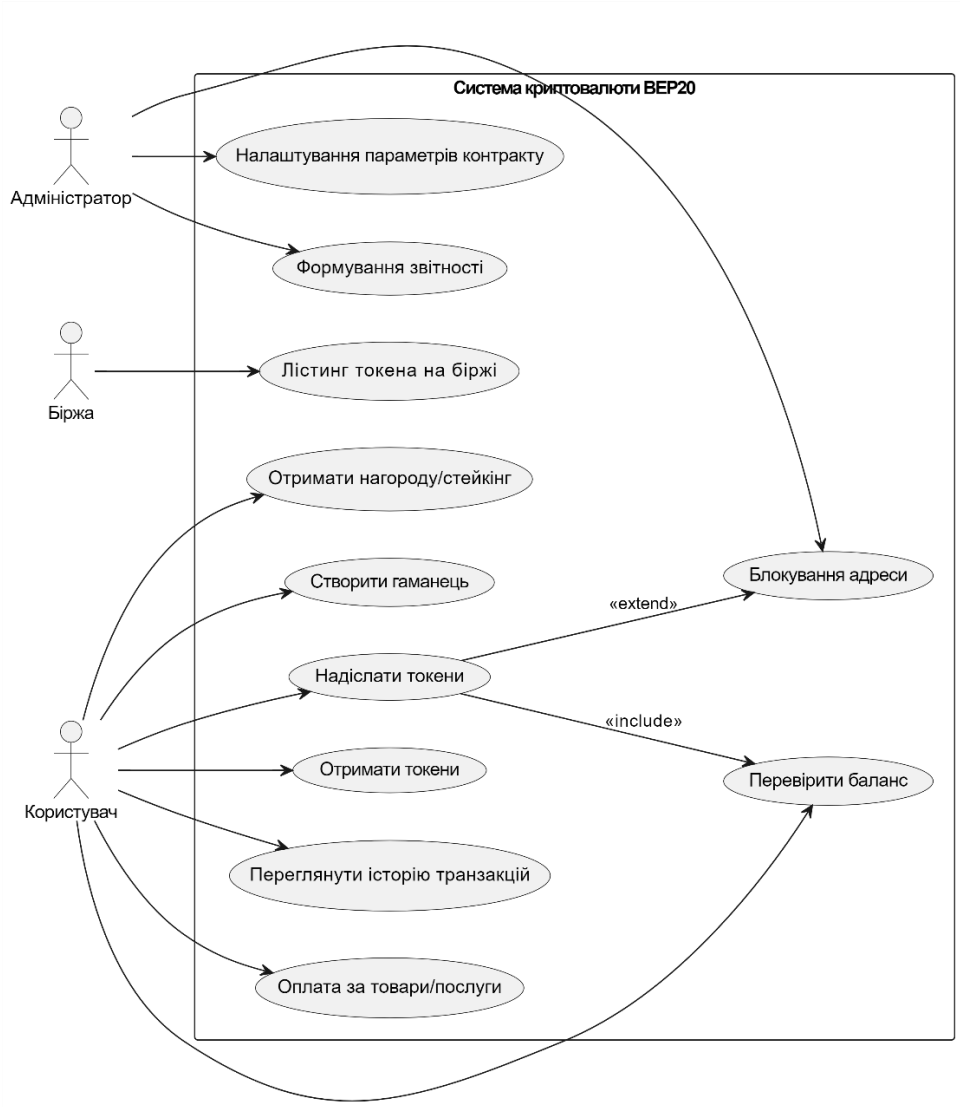
18. Blockchain Technology Overview / U.S. National Institute of Standards and Technology (NIST). – [Електронний ресурс]. – Режим доступу: https://doi.org/10.6028/NIST.IR.8202

19. Antonopoulos A.M., Wood G. Mastering Ethereum. – O'Reilly Media, 2018. – 420 p.

20. Binance Chain Docs: Token Management [Електронний ресурс]. – Режим доступу: https://docs.bnbchain.org/docs/learn/token-management

ДОДАТОК А

Діаграма прецедентів

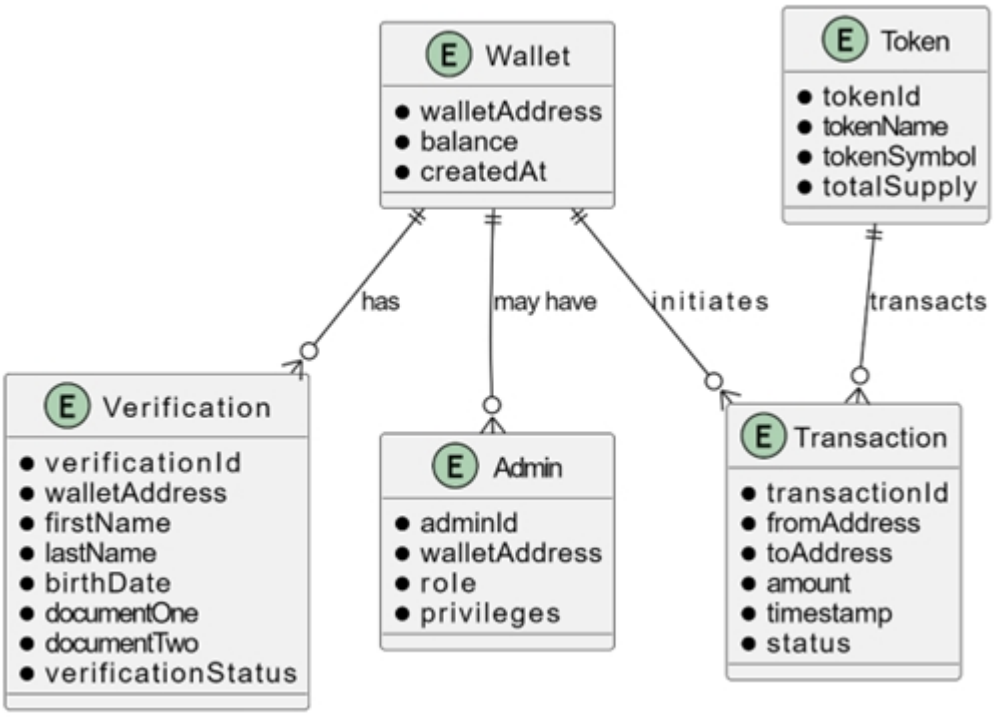


ДОДАТОК Б

Діаграма активності

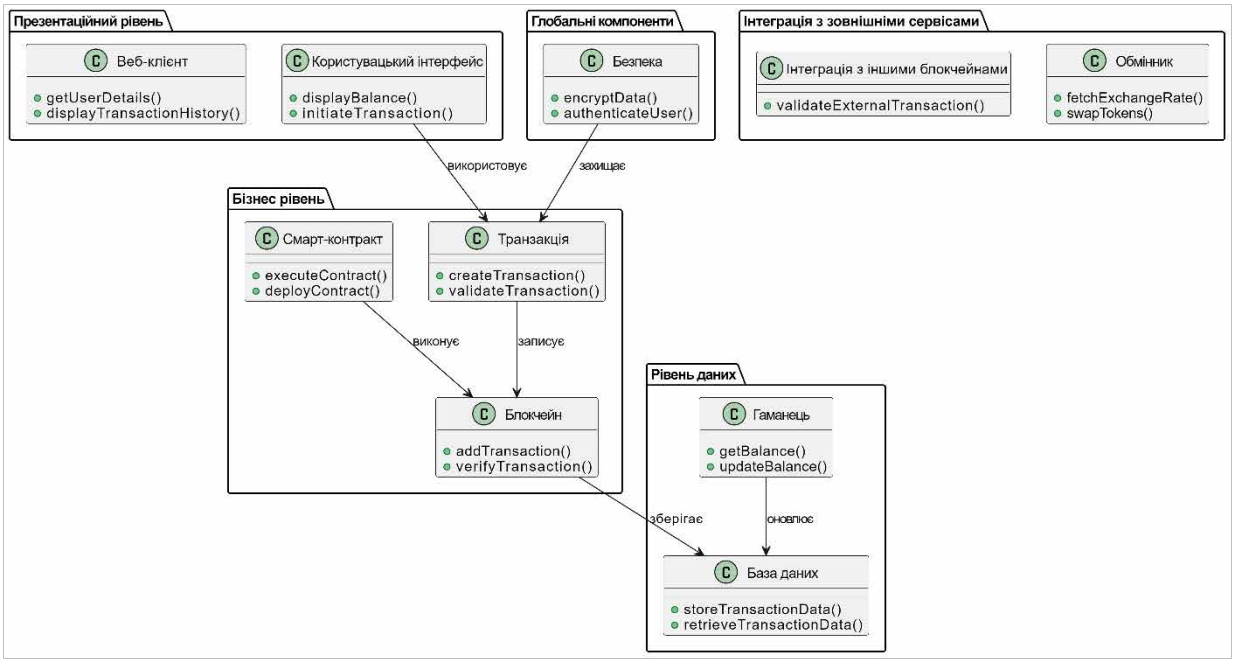
ДОДАТОК В

ER-діаграма



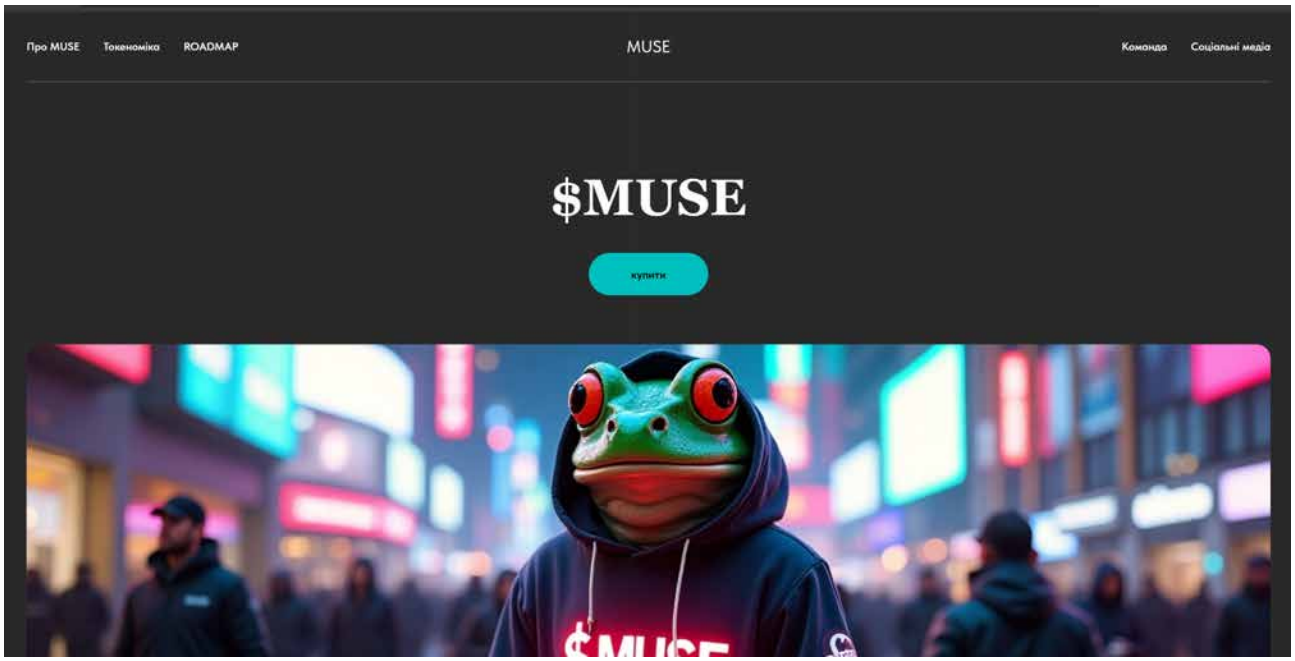
ДОДАТОК Д

Діаграма пакетів



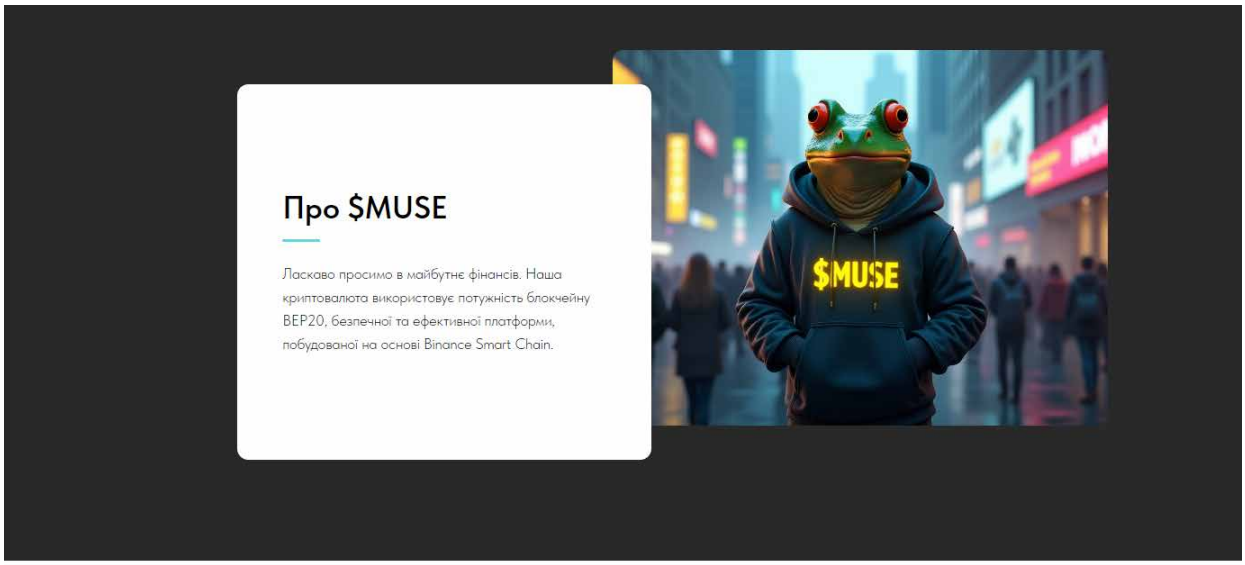
ДОДАТОК Е

Покупка монет через сайт



ДОДАТОК Ж

Інформація про монету на головній сторінці сайту



Токеноміка

«Огляд розподілу токенів для команди, спільноти, пулу ліквідності та використання, що забезпечує прозорість та ілюструє фінансову структуру проекту».

Загальна пропозиція

1 000 000 000 $MUSE

Розподіл токенів

- Спільнота: 40%
- Пул ліквідності: 50%
- Команда: 5%
- Розвиток: 5%

Використання $MUSE

- Управління: голосування за пропозиції та майбутній розвиток.
- Ставки: отримуйте нагороди за зберігання токенів.
- Екосистема: доступ до ексклюзивних послуг і знижок.

ДОДАТОК И

Код виконання усіх алгоритмів

```
pragma solidity ^0.7.6;

library SafeMath {
    function add(uint256 a, uint256 b) internal pure returns (uint256) {
        uint256 c = a + b;
        require(c >= a, "SafeMath: addition overflow");

        return c;
    }
    function sub(uint256 a, uint256 b) internal pure returns (uint256) {
        return sub(a, b, "SafeMath: subtraction overflow");
    }
    function sub(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b <= a, errorMessage);
        uint256 c = a - b;

        return c;
    }
    function mul(uint256 a, uint256 b) internal pure returns (uint256) {
        if (a == 0) {
            return 0;
        }

        uint256 c = a * b;
        require(c / a == b, "SafeMath: multiplication overflow");

        return c;
    }
    function div(uint256 a, uint256 b) internal pure returns (uint256) {
        return div(a, b, "SafeMath: division by zero");
    }
    function div(uint256 a, uint256 b, string memory errorMessage) internal pure
returns (uint256) {
        require(b > 0, errorMessage);
```

```
        uint256 c = a / b;
        return c;
    }
}

/**
 * BEP20 standard interface.
 */
interface IBEP20 {
    function totalSupply() external view returns (uint256);
    function decimals() external view returns (uint8);
    function symbol() external view returns (string memory);
    function name() external view returns (string memory);
    function getOwner() external view returns (address);
    function balanceOf(address account) external view returns (uint256);
    function transfer(address recipient, uint256 amount) external returns (bool);
    function allowance(address _owner, address spender) external view returns
(uint256);
    function approve(address spender, uint256 amount) external returns (bool);
    function transferFrom(address sender, address recipient, uint256 amount)
external returns (bool);
    event Transfer(address indexed from, address indexed to, uint256 value);
    event Approval(address indexed owner, address indexed spender, uint256 value);
}

abstract contract Auth {
    address internal owner;
    mapping (address => bool) internal authorizations;

    constructor(address _owner) {
        owner = _owner;
        authorizations[_owner] = true;
    }

    modifier onlyOwner() {
        require(isOwner(msg.sender), "!OWNER"); _;
    }

    modifier authorized() {
```

```
        require(isAuthorized(msg.sender), "!AUTHORIZED"); _;
    }

    function authorize(address adr) public onlyOwner {
        authorizations[adr] = true;
    }

    function unauthorize(address adr) public onlyOwner {
        authorizations[adr] = false;
    }

    function isOwner(address account) public view returns (bool) {
        return account == owner;
    }

    function isAuthorized(address adr) public view returns (bool) {
        return authorizations[adr];
    }

    function transferOwnership(address payable adr) public onlyOwner {
        owner = adr;
        authorizations[adr] = true;
        emit OwnershipTransferred(adr);
    }

    event OwnershipTransferred(address owner);
}

interface IDEXFactory {
    function createPair(address tokenA, address tokenB) external returns (address
pair);
}

interface IDEXRouter {
    function factory() external pure returns (address);
    function WETH() external pure returns (address);

    function addLiquidity(
        address tokenA,
```

```
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin,
    address to,
    uint deadline
) external returns (uint amountA, uint amountB, uint liquidity);

function addLiquidityETH(
    address token,
    uint amountTokenDesired,
    uint amountTokenMin,
    uint amountETHMin,
    address to,
    uint deadline
) external payable returns (uint amountToken, uint amountETH, uint liquidity);

function swapExactTokensForTokensSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external;

function swapExactETHForTokensSupportingFeeOnTransferTokens(
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
) external payable;

function swapExactTokensForETHSupportingFeeOnTransferTokens(
    uint amountIn,
    uint amountOutMin,
    address[] calldata path,
    address to,
    uint deadline
```

```
    ) external;
}

interface IDividendDistributor {
    function setDistributionCriteria(uint256 _minPeriod, uint256 _minDistribution)
external;
    function setShare(address shareholder, uint256 amount) external;
    function deposit() external payable;
    function process(uint256 gas) external;
}

contract DividendDistributor is IDividendDistributor {
    using SafeMath for uint256;

    address _token;

    struct Share {
        uint256 amount;
        uint256 totalExcluded;
        uint256 totalRealised;
    }

    IBEP20 RWRD = IBEP20(0xe9e7CEA3DedcA5984780Bafc599bD69ADd087D56);
    address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c;
    IDEXRouter router;

    address[] shareholders;
    mapping (address => uint256) shareholderIndexes;
    mapping (address => uint256) shareholderClaims;

    mapping (address => Share) public shares;

    uint256 public totalShares;
    uint256 public totalDividends;
    uint256 public totalDistributed;
    uint256 public dividendsPerShare;
    uint256 public dividendsPerShareAccuracyFactor = 10 ** 36;

    uint256 public minPeriod = 45 * 60;
```

```
    uint256 public minDistribution = 1 * (10 ** 13);

    uint256 currentIndex;

    bool initialized;
    modifier initialization() {
        require(!initialized);
        _;
        initialized = true;
    }

    modifier onlyToken() {
        require(msg.sender == _token); _;
    }

    constructor (address _router) {
        router = _router != address(0)
            ? IDEXRouter(_router)
            : IDEXRouter(0x10ED43C718714eb63d5aA57B78B54704E256024E);
        _token = msg.sender;
    }

    function setDistributionCriteria(uint256 _minPeriod, uint256 _minDistribution)
external override onlyToken {
        minPeriod = _minPeriod;
        minDistribution = _minDistribution;
    }

    function setShare(address shareholder, uint256 amount) external override
onlyToken {
        if(shares[shareholder].amount > 0){
            distributeDividend(shareholder);
        }

        if(amount > 0 && shares[shareholder].amount == 0){
            addShareholder(shareholder);
        }else if(amount == 0 && shares[shareholder].amount > 0){
            removeShareholder(shareholder);
        }
```

```
        totalShares = totalShares.sub(shares[shareholder].amount).add(amount);
        shares[shareholder].amount = amount;
        shares[shareholder].totalExcluded =
getCumulativeDividends(shares[shareholder].amount);
    }

    function deposit() external payable override onlyToken {
        uint256 balanceBefore = RWRD.balanceOf(address(this));

        address[] memory path = new address[](2);
        path[0] = WBNB;
        path[1] = address(RWRD);

        router.swapExactETHForTokensSupportingFeeOnTransferTokens{value:
msg.value}(
            0,
            path,
            address(this),
            block.timestamp
        );

        uint256 amount = RWRD.balanceOf(address(this)).sub(balanceBefore);

        totalDividends = totalDividends.add(amount);
        dividendsPerShare =
dividendsPerShare.add(dividendsPerShareAccuracyFactor.mul(amount).div(totalShares))
;
    }

    function process(uint256 gas) external override onlyToken {
        uint256 shareholderCount = shareholders.length;

        if(shareholderCount == 0) { return; }

        uint256 gasUsed = 0;
        uint256 gasLeft = gasleft();

        uint256 iterations = 0;
```

```
        while(gasUsed < gas && iterations < shareholderCount) {
            if(currentIndex >= shareholderCount){
                currentIndex = 0;
            }

            if(shouldDistribute(shareholders[currentIndex])){
                distributeDividend(shareholders[currentIndex]);
            }

            gasUsed = gasUsed.add(gasLeft.sub(gasleft()));
            gasLeft = gasleft();
            currentIndex++;
            iterations++;
        }
    }

    function shouldDistribute(address shareholder) internal view returns (bool) {
        return shareholderClaims[shareholder] + minPeriod < block.timestamp
                && getUnpaidEarnings(shareholder) > minDistribution;
    }

    function distributeDividend(address shareholder) internal {
        if(shares[shareholder].amount == 0){ return; }

        uint256 amount = getUnpaidEarnings(shareholder);
        if(amount > 0){
            totalDistributed = totalDistributed.add(amount);
            RWRD.transfer(shareholder, amount);
            shareholderClaims[shareholder] = block.timestamp;
            shares[shareholder].totalRealised =
shares[shareholder].totalRealised.add(amount);
            shares[shareholder].totalExcluded =
getCumulativeDividends(shares[shareholder].amount);
        }
    }

    function claimDividend() external {
        distributeDividend(msg.sender);
```

```
    }

    function getUnpaidEarnings(address shareholder) public view returns (uint256) {
        if(shares[shareholder].amount == 0){ return 0; }

        uint256 shareholderTotalDividends =
getCumulativeDividends(shares[shareholder].amount);
        uint256 shareholderTotalExcluded = shares[shareholder].totalExcluded;

        if(shareholderTotalDividends <= shareholderTotalExcluded){ return 0; }

        return shareholderTotalDividends.sub(shareholderTotalExcluded);
    }

    function getCumulativeDividends(uint256 share) internal view returns (uint256)
{
        return share.mul(dividendsPerShare).div(dividendsPerShareAccuracyFactor);
    }

    function addShareholder(address shareholder) internal {
        shareholderIndexes[shareholder] = shareholders.length;
        shareholders.push(shareholder);
    }

    function removeShareholder(address shareholder) internal {
        shareholders[shareholderIndexes[shareholder]] =
shareholders[shareholders.length-1];
        shareholderIndexes[shareholders[shareholders.length-1]] =
shareholderIndexes[shareholder];
        shareholders.pop();
    }
}

contract Muse is IBEP20, Auth {
    using SafeMath for uint256;

    address WBNB = 0xbb4CdB9CBd36B01bD1cBaEBF2De08d9173bc095c;
    address DEAD = 0x000000000000000000000000000000000000dEaD;
    address ZERO = 0x0000000000000000000000000000000000000000;
```

```
    address DEV = 0x371870617Df3D6C8b389650dE35b37D6A1b6FA22;

    string constant _name = "Muse";
    string constant _symbol = "MUSE";
    uint8 constant _decimals = 9;

    uint256 _totalSupply = 1000 * 10**6 * 10**_decimals;

    uint256 public _maxTxAmount = _totalSupply;
    uint256 public _maxWalletToken = _totalSupply;

    mapping (address => uint256) _balances;
    mapping (address => mapping (address => uint256)) _allowances;

    bool public blacklistMode = true;
    mapping (address => bool) public isBlacklisted;

    mapping (address => bool) isFeeExempt;
    mapping (address => bool) isTxLimitExempt;
    mapping (address => bool) isTimelockExempt;
    mapping (address => bool) isDividendExempt;

    uint256 public liquidityFee    = 2;
    uint256 public reflectionFee   = 3;
    uint256 public marketingFee    = 6;
    uint256 public devFee          = 1;
    uint256 public totalFee        = marketingFee + reflectionFee + liquidityFee +
devFee;
    uint256 public feeDenominator  = 100;

    uint256 public sellMultiplier  = 100;

    address public autoLiquidityReceiver;
    address public marketingFeeReceiver;
    address public devFeeReceiver;

    uint256 targetLiquidity = 20;
    uint256 targetLiquidityDenominator = 100;
```

```
IDEXRouter public router;
address public pair;

bool public tradingOpen = false;

DividendDistributor public distributor;
uint256 distributorGas = 500000;

bool public buyCooldownEnabled = true;
uint8 public cooldownTimerInterval = 10;
mapping (address => uint) private cooldownTimer;

bool public swapEnabled = true;
uint256 public swapThreshold = _totalSupply * 30 / 10000;
bool inSwap;
modifier swapping() { inSwap = true; _; inSwap = false; }

constructor () Auth(msg.sender) {
    router = IDEXRouter(0x10ED43C718714eb63d5aA57B78B54704E256024E);
    pair = IDEXFactory(router.factory()).createPair(WBNB, address(this));
    _allowances[address(this)][address(router)] = uint256(-1);

    distributor = new DividendDistributor(address(router));

    isFeeExempt[msg.sender] = true;
    isFeeExempt[address(DEV)] = true;
    isTxLimitExempt[msg.sender] = true;
    isTxLimitExempt[address(DEV)] = true;

    isTimelockExempt[msg.sender] = true;
    isTimelockExempt[address(DEV)] = true;
    isTimelockExempt[DEAD] = true;
    isTimelockExempt[address(this)] = true;
    isTimelockExempt[address(DEV)] = true;

    isDividendExempt[pair] = true;
    isDividendExempt[address(this)] = true;
    isDividendExempt[address(DEV)] = false;
    isDividendExempt[DEAD] = true;
```

```
        autoLiquidityReceiver = msg.sender;
        marketingFeeReceiver = msg.sender;
        devFeeReceiver = address(DEV);

        _balances[msg.sender] = _totalSupply;
        emit Transfer(address(0), msg.sender, _totalSupply);
    }

    receive() external payable { }

    function totalSupply() external view override returns (uint256) { return
_totalSupply; }
    function decimals() external pure override returns (uint8) { return _decimals;
}
    function symbol() external pure override returns (string memory) { return
_symbol; }
    function name() external pure override returns (string memory) { return _name;
}
    function getOwner() external view override returns (address) { return owner; }
    function balanceOf(address account) public view override returns (uint256) {
return _balances[account]; }
    function allowance(address holder, address spender) external view override
returns (uint256) { return _allowances[holder][spender]; }

    function approve(address spender, uint256 amount) public override returns
(bool) {
        _allowances[msg.sender][spender] = amount;
        emit Approval(msg.sender, spender, amount);
        return true;
    }

    function approveMax(address spender) external returns (bool) {
        return approve(spender, uint256(-1));
    }

    function transfer(address recipient, uint256 amount) external override returns
(bool) {
        return _transferFrom(msg.sender, recipient, amount);
```

```
    }

    function transferFrom(address sender, address recipient, uint256 amount)
external override returns (bool) {
        if(_allowances[sender][msg.sender] != uint256(-1)){
            _allowances[sender][msg.sender] =
_allowances[sender][msg.sender].sub(amount, "Insufficient Allowance");
        }

        return _transferFrom(sender, recipient, amount);
    }

    function setMaxWalletPercent_base1000(uint256 maxWallPercent_base1000) external
onlyOwner() {
        _maxWalletToken = (_totalSupply * maxWallPercent_base1000 ) / 1000;
    }
    function setMaxTxPercent_base1000(uint256 maxTXPercentage_base1000) external
onlyOwner() {
        _maxTxAmount = (_totalSupply * maxTXPercentage_base1000 ) / 1000;
    }

    function setTxLimit(uint256 amount) external authorized {
        _maxTxAmount = amount;
    }

    function _transferFrom(address sender, address recipient, uint256 amount)
internal returns (bool) {
        if(inSwap){ return _basicTransfer(sender, recipient, amount); }

        if(!authorizations[sender] && !authorizations[recipient]){
            require(tradingOpen,"Trading not open yet");
        }

        // Blacklist
        if(blacklistMode){
            require(!isBlacklisted[sender] &&
!isBlacklisted[recipient],"Blacklisted");
        }
```

```
        if (!authorizations[sender] && recipient != address(this)  && recipient != address(DEAD) && recipient != pair && recipient != marketingFeeReceiver && recipient != devFeeReceiver  && recipient != autoLiquidityReceiver){
            uint256 heldTokens = balanceOf(recipient);
            require((heldTokens + amount) <= _maxWalletToken,"Total Holding is currently limited, you can not buy that much.");}

        if (sender == pair &&
            buyCooldownEnabled &&
            !isTimelockExempt[recipient]) {
            require(cooldownTimer[recipient] < block.timestamp,"Please wait for 1min between two buys");
            cooldownTimer[recipient] = block.timestamp + cooldownTimerInterval;
        }

        // Checks max transaction limit
        checkTxLimit(sender, amount);

        if(shouldSwapBack()){ swapBack(); }

        //Exchange tokens
        _balances[sender] = _balances[sender].sub(amount, "Insufficient Balance");

        uint256 amountReceived = shouldTakeFee(sender) ? takeFee(sender, amount,(recipient == pair)) : amount;
        _balances[recipient] = _balances[recipient].add(amountReceived);

        // Dividend tracker
        if(!isDividendExempt[sender]) {
            try distributor.setShare(sender, _balances[sender]) {} catch {}
        }

        if(!isDividendExempt[recipient]) {
            try distributor.setShare(recipient, _balances[recipient]) {} catch {}
        }

        try distributor.process(distributorGas) {} catch {}

        emit Transfer(sender, recipient, amountReceived);
```

```
        return true;
    }

    function _basicTransfer(address sender, address recipient, uint256 amount)
internal returns (bool) {
        _balances[sender] = _balances[sender].sub(amount, "Insufficient Balance");
        _balances[recipient] = _balances[recipient].add(amount);
        emit Transfer(sender, recipient, amount);
        return true;
    }

    function checkTxLimit(address sender, uint256 amount) internal view {
        require(amount <= _maxTxAmount || isTxLimitExempt[sender], "TX Limit
Exceeded");
    }

    function shouldTakeFee(address sender) internal view returns (bool) {
        return !isFeeExempt[sender];
    }

    function takeFee(address sender, uint256 amount, bool isSell) internal returns
(uint256) {

        uint256 multiplier = isSell ? sellMultiplier : 100;
        uint256 feeAmount = amount.mul(totalFee).mul(multiplier).div(feeDenominator
* 100);


        _balances[address(this)] = _balances[address(this)].add(feeAmount);
        emit Transfer(sender, address(this), feeAmount);

        return amount.sub(feeAmount);
    }

    function shouldSwapBack() internal view returns (bool) {
        return msg.sender != pair
        && !inSwap
        && swapEnabled
        && _balances[address(this)] >= swapThreshold;
```

```
    }

    function clearStuckBalance(uint256 amountPercentage) external authorized {
        uint256 amountBNB = address(this).balance;
        payable(marketingFeeReceiver).transfer(amountBNB * amountPercentage / 100);
    }

    function clearStuckBalance_sender(uint256 amountPercentage) external authorized
{
        uint256 amountBNB = address(this).balance;
        payable(msg.sender).transfer(amountBNB * amountPercentage / 100);
    }

    function set_sell_multiplier(uint256 Multiplier) external onlyOwner{
        sellMultiplier = Multiplier;
    }

    // switch Trading
    function tradingStatus(bool _status) public onlyOwner {
        tradingOpen = _status;
    }

    // enable cooldown between trades
    function cooldownEnabled(bool _status, uint8 _interval) public onlyOwner {
        buyCooldownEnabled = _status;
        cooldownTimerInterval = _interval;
    }

    function swapBack() internal swapping {
        uint256 dynamicLiquidityFee = isOverLiquified(targetLiquidity,
targetLiquidityDenominator) ? 0 : liquidityFee;
        uint256 amountToLiquify =
swapThreshold.mul(dynamicLiquidityFee).div(totalFee).div(2);
        uint256 amountToSwap = swapThreshold.sub(amountToLiquify);

        address[] memory path = new address[](2);
        path[0] = address(this);
        path[1] = WBNB;
```

```
        uint256 balanceBefore = address(this).balance;

        router.swapExactTokensForETHSupportingFeeOnTransferTokens(
            amountToSwap,
            0,
            path,
            address(this),
            block.timestamp
        );

        uint256 amountBNB = address(this).balance.sub(balanceBefore);

        uint256 totalBNBFee = totalFee.sub(dynamicLiquidityFee.div(2));

        uint256 amountBNBLiquidity =
amountBNB.mul(dynamicLiquidityFee).div(totalBNBFee).div(2);
        uint256 amountBNBReflection =
amountBNB.mul(reflectionFee).div(totalBNBFee);
        uint256 amountBNBMarketing = amountBNB.mul(marketingFee).div(totalBNBFee);
        uint256 amountBNBDev = amountBNB.mul(devFee).div(totalBNBFee);

        try distributor.deposit{value: amountBNBReflection}() {} catch {}
        (bool tmpSuccess,) = payable(marketingFeeReceiver).call{value:
amountBNBMarketing, gas: 30000}("");
        (tmpSuccess,) = payable(devFeeReceiver).call{value: amountBNBDev, gas:
30000}("");

        // Supress warning msg
        tmpSuccess = false;

        if(amountToLiquify > 0){
            router.addLiquidityETH{value: amountBNBLiquidity}(
                address(this),
                amountToLiquify,
                0,
                0,
                autoLiquidityReceiver,
                block.timestamp
            );
```

```
            emit AutoLiquify(amountBNBLiquidity, amountToLiquify);
        }
    }

    function setIsDividendExempt(address holder, bool exempt) external authorized {
        require(holder != address(this) && holder != pair);
        isDividendExempt[holder] = exempt;
        if(exempt){
            distributor.setShare(holder, 0);
        }else{
            distributor.setShare(holder, _balances[holder]);
        }
    }

    function enable_blacklist(bool _status) public onlyOwner {
        blacklistMode = _status;
    }

    function manage_blacklist(address[] calldata addresses, bool status) public
onlyOwner {
        for (uint256 i; i < addresses.length; ++i) {
            isBlacklisted[addresses[i]] = status;
        }
    }

    function setIsFeeExempt(address holder, bool exempt) external authorized {
        isFeeExempt[holder] = exempt;
    }

    function setIsTxLimitExempt(address holder, bool exempt) external authorized {
        isTxLimitExempt[holder] = exempt;
    }

    function setIsTimelockExempt(address holder, bool exempt) external authorized {
        isTimelockExempt[holder] = exempt;
    }

    function setFees(uint256 _liquidityFee, uint256 _reflectionFee, uint256
_marketingFee, uint256 _feeDenominator) external authorized {
```

```
        liquidityFee = _liquidityFee;
        reflectionFee = _reflectionFee;
        marketingFee = _marketingFee;
        devFee = 1;
        totalFee =
_liquidityFee.add(_reflectionFee).add(_marketingFee).add(devFee);
        feeDenominator = _feeDenominator;
        require(totalFee < feeDenominator/3, "Fees cannot be more than 33%");
    }

    function setFeeReceivers(address _autoLiquidityReceiver, address
_marketingFeeReceiver ) external authorized {
        autoLiquidityReceiver = _autoLiquidityReceiver;
        marketingFeeReceiver = _marketingFeeReceiver;
        devFeeReceiver = address(DEV);
    }

    function setSwapBackSettings(bool _enabled, uint256 _amount) external
authorized {
        swapEnabled = _enabled;
        swapThreshold = _amount;
    }

    function setTargetLiquidity(uint256 _target, uint256 _denominator) external
authorized {
        targetLiquidity = _target;
        targetLiquidityDenominator = _denominator;
    }

    function setDistributionCriteria(uint256 _minPeriod, uint256 _minDistribution)
external authorized {
        distributor.setDistributionCriteria(_minPeriod, _minDistribution);
    }

    function setDistributorSettings(uint256 gas) external authorized {
        require(gas < 750000);
        distributorGas = gas;
    }
```

```
    function getCirculatingSupply() public view returns (uint256) {
        return _totalSupply.sub(balanceOf(DEAD)).sub(balanceOf(ZERO));
    }

    function getLiquidityBacking(uint256 accuracy) public view returns (uint256) {
        return accuracy.mul(balanceOf(pair).mul(2)).div(getCirculatingSupply());
    }

    function isOverLiquified(uint256 target, uint256 accuracy) public view returns
(bool) {
        return getLiquidityBacking(accuracy) > target;
    }


/* Airdrop */
function multiTransfer(address from, address[] calldata addresses, uint256[]
calldata tokens) external onlyOwner {

    require(addresses.length < 501,"GAS Error: max airdrop limit is 500
addresses");
    require(addresses.length == tokens.length,"Mismatch between Address and token
count");

    uint256 SCCC = 0;

    for(uint i=0; i < addresses.length; i++){
        SCCC = SCCC + tokens[i];
    }

    require(balanceOf(from) >= SCCC, "Not enough tokens in wallet");

    for(uint i=0; i < addresses.length; i++){
        _basicTransfer(from,addresses[i],tokens[i]);
        if(!isDividendExempt[addresses[i]]) {
            try distributor.setShare(addresses[i], _balances[addresses[i]]) {}
catch {}
        }
    }
```

```
    // Dividend tracker
    if(!isDividendExempt[from]) {
        try distributor.setShare(from, _balances[from]) {} catch {}
    }
}

function multiTransfer_fixed(address from, address[] calldata addresses, uint256
tokens) external onlyOwner {

    require(addresses.length < 801,"GAS Error: max airdrop limit is 800
addresses");

    uint256 SCCC = tokens * addresses.length;

    require(balanceOf(from) >= SCCC, "Not enough tokens in wallet");

    for(uint i=0; i < addresses.length; i++){
        _basicTransfer(from,addresses[i],tokens);
        if(!isDividendExempt[addresses[i]]) {
            try distributor.setShare(addresses[i], _balances[addresses[i]]) {}
catch {}
        }
    }

    // Dividend tracker
    if(!isDividendExempt[from]) {
        try distributor.setShare(from, _balances[from]) {} catch {}
    }
}

event AutoLiquify(uint256 amountBNB, uint256 amountBOG);

}
```