КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ ТАРАСА ШЕВЧЕНКА
Факультет комп'ютерних наук та кібернетики
Кафедра інтелектуальних програмних систем

Кваліфікаційна робота
на здобуття освітнього рівня бакалавра
за спеціальністю 121 Інженерія програмного забезпечення
на тему:

РОЗРОБКА ЗАХИЩЕНИХ ІНФОРМАЦІЙНИХ СИСТЕМ НА БАЗІ ТЕХНОЛОГІЇ БЛОКЧЕЙН

Виконав студент 4-го курсу
Дмитро КОРНЕТ _____
(підпис)

Науковий керівник:
доцент, кандидат фіз.-мат. наук
Євгеній ІВАНОВ _____
(підпис)

Засвідчую, що в цій роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент _____
(підпис)

Роботу розглянуто й допущено до захисту
на засіданні кафедри інтелектуальних програмних систем
« 29 » травня 2023 р.,
протокол № __
Завідувач кафедри
Олександр
ПРОВОТАР _____
(підпис)

Київ - 2023

РЕФЕРАТ

Обсяг роботи 45 сторінок, 15 ілюстрацій, 20 використаних джерел.

ІНФОРМАЦІЙНА СИСТЕМА, СЕРВЕРНИЙ ЗАСТОСУНОК, ФІНАНСОВІ ТЕХНОЛОГІЇ, БЛОКЧЕЙН, КІБЕРБЕЗПЕКА, АВТОРИЗАЦІЯ, АУТЕНТИФІКАЦІЯ.

Об’єктом розробки програмного забезпечення є побудова робочого крос-платформного серверного застосунку.

Метою кваліфікаційної роботи є ознайомлення з процесом розробки інформаційних систем, дослідження таких додатків, представлених на ринку, виявлення їхніх переваг та недоліків. Ознайомлення з етапами розробки, вивчення його складових компонентів. Вивчення методології розробки серверних застосунків. Отримані знання використати при створенні власної інформаційної системи з використанням механізмів автентифікації, відмовостійкості та захищеності.

Інструментом створення є інтегроване середовище розробки програмного забезпечення JetBrains GoLand 2023.1. Мова програмування GO. Інтерфейс для обробки HTTP(-S) запитів - безкоштовна, вільно поширювана бібліотека GIN.

Результат роботи: досліджена структура серверного застосунку. Розглянуті етапи обробки систем, основаних на технології блокчейн та систем, які містять механізми авторизації та аутентифікації. Отримані теоретичні знання використані для розробки власного серверного застосунку. Реалізовано відмовостійку блокчейн систему для ведення сімейного бюджету з можливістю отримання статистики.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧЕННЯ

API - Application Programming Interface;

CSS - Cascading Style Sheets;

CSV - comma-separated values;

Dir - directory, директорія;

DRM - Digital Rights Management;

HTML - HyperText Markup Language;

HTTP - HyperText Transfer Protocol;

HTTPS - HyperText Transfer Protocol Secure;

ID - identifier;

IT - Information Technology;

JSON - JavaScript Object Notation;

LRU - least recently used;

NFT - non-fungible token;

PHP - Hypertext Preprocessor;

P2P - peer-to-peer;

REST - Representational state transfer;

RPC - Remote Protocol Call;

SQL - Structured query language;

TCP - Transmission Control Protocol;

TX - transaction;

UDP - User Datagram Protocol;

XSS - Cross Site Scripting;

XML - Extensible Markup Language;

БВ - був у вживанні;

СУБД - система управління базами даних.

ВСТУП

Оцінка сучасного стану об’єкта розробки. Індустрія розробки інформаційних систем є достатньо молодою на ринку. Перші такі системи почали з’являтися близько 30 років тому. За цей час індустрія зазнала значних змін: з’являлися нові методології розробки, нові технології[1].

З’явився новий критерій оцінки якості коду - підтримуваність. Це пов’язано з необхідністю створення регулярних оновлень в умовах конкурентного ринку, який швидко розвивався. В той же час, необхідність в швидкодії такого типу застосунків стала вторинною ознакою. Одними з причин такого стала революція в сфері створення обчислювальних систем та методологія розробки децентралізованого програмного забезпечення.

Однією з важливих характеристик високонавантажених інформаційних систем в умовах вільного доступу до ресурсів стала захищеність даних та відмовостійкість сервісів. Навіть люди, які мало знайомі з хмарними технологіями та інтернетом, чули про масштабні збої в роботі банків та “зливи” персональних даних клієнтів через недосконалість та вразливість існуючих програмних комплексів.

Виходячи з наведених даних, однозначно можна стверджувати, що наразі розробка серверних застосунків потребує впевнених знань з кібербезпеки та правил написання чистого коду, так як некомпетентність в цих сферах може призвести до непередбачуваних наслідків, а також до фінансових та репутаційних втрат.

Актуальність роботи та підстави для її виконання. Наразі сфера розробки серверних інформаційних систем є дуже розвинутою. Інтернет став невід’ємною частиною життя людей всіх вікових категорій та застосовується у більшості сфер життя: від розваг (ігри, фільми, музика і Т.Д.) до фінансів та армії.

Однією з причин такого різкого зростання попиту на подібні системи стала дешевизна підтримки готових рішень. Наприклад, налагоджений процес створення будь-якого фізичного продукту потребує певних чималих фінансових витрат на кожну одиницю виготовленого продукту, в той час як доступ до інформаційних систем потребує лише покриття витрат на підтримку роботоздатності серверів, через що витрати на кожного клієнта є відносно малими навіть в умовах високонавантажених сервісів (стрімінгові, відеохостингові платформи, багатокористувацькі ігри і Т.Д.)

Мета і завдання роботи. Метою кваліфікаційної роботи є ознайомлення з етапами розробки захищених серверних застосунків та аналіз існуючих на ринку рішень, дослідження етапів планування, перевірки на захищеність, відповідності вимогам розширюваних та підтримуваних архітектурних рішень. Отриманий досвід було використано при створенні власної інформаційної системи.

Об’єкт, методи й засоби дослідження або розроблення. Об’єктом розробки програмного засобу є створення захищеного серверного застосунку для проведення фінансових операцій.

Під час розробки було використано алгоритми хешування sha256 та sha1.

Можливі сфери застосування. Кінцевий продукт може використовуватися в якості основи для створення відкритої фінансово-технологічної захищеної системи.

РОЗДІЛ 1. СПЕЦИФІКА СЕРВЕРНИХ ЗАСТОСУНКІВ

1.1 Що таке інформаційна система?

Інформаційна система - серверний застосунок, який виконує певні обчислення та/або надає доступ до певних даних. Створенню індустрії інтернету передувало більш ніж 30 років розробки систем та протоколів передачі даних.

1.2 Передумови розвитку інформаційних систем

Інформаційні системи дозволяли надавати сервіс нового рівня. Клієнти не мали необхідності йти в магазин для того, щоб користуватись послугами. З цього випливають декілька переваг такого підходу: клієнти отримували сервіс з будь-якої точки світу, де є доступ до якісного інтернет з'єднання, а компаніям не доводилось створювати офлайн точки, що дозволяло надавати кращий сервіс, використовуючи меншу кількість ресурсів.

Одним з яскравих прикладів є статистика індустрії відеопрокату. В середині 2000-х років почала набувати популярності індустрія потокових мультимедійних платформ (такі як Netflix, YouTube), через що великі компанії з оренди касет різко втратили популярність та були вимушені закритись[2, 3].

1.3 Розвиток сфери розробки серверних застосунків

Всесвітня мережа у вигляді, близькому до сучасного, почала з'являтись наприкінці 80-х – на початку 90-х років 20-го сторіччя. Перший веб-сайт у звичному нам вигляді було створено у грудні 1990 року. Після того, як у 1991 було скасовано заборону на комерційне використання, за

короткий час, кількість користувачів мережі досягла одного мільйона, а кількість серверів - двадцяти восьми. Вже у 1993 буде створено перший графічний інтернет браузер, а з 1994 почнуть з'являться перші компанії, які все ще являються гігантами індустрії, такі як: Yahoo, Ebay, Google.

У зв'язку з появою новою індустрії, з'явилася необхідність в розвитку технологій. Станом на 2000 рік вже було створено такі технології, які використовуються й досі: HTML, CSS, JavaScript, PHP, а індекс пошуку платформи Google перевищив 1 мільярд позицій.

Починаючи з першої половини 2000-х років, було створено велику кількість знайомих більшості сервісів: Facebook, YouTube, Reddit, Twitter, Spotify, WhatsApp, Instagram, Viber. На той момент вже ставав все більш очевидним той факт, що інтернет індустрія розвивається та буде розвиватись і надалі.

Вже тоді великі компанії стикнулись з необхідністю оптимізації та стандартизації процесів розробки, а також з необхідністю роботи над захистом даних. Створювалися нові алгоритми, нові мережеві протоколи у зв'язку з недосконалістю вже існуючих на той момент.

1.4 Проблеми створення ідеальних веб-сервісів

Не дивлячись на можливості надавати сервіс нового рівня, перед компаніями постали такі проблеми: необхідність монетизації, захисту інформації від інтернет-піратства, захисту персональних даних, захисту від втручання зловмисників, конкурентноздатність на новому ринку, який швидко розвивається.

1.4.1 Проблеми монетизації

Одним з перших питань стало: “а як отримати з цього гроші?”. Одними з перших рішень, які, як показала практика, є стабільними, стали: реклама та платність послуг.

Розглянемо плюси та мінуси кожного з варіантів.

Реклама дозволяла створювати сервіси, які є безкоштовними для користувачів, що допомагало уникати проблеми оплати в інтернеті, але не кожен сервіс дозволяв грамотно інтегрувати рекламу та отримувати достатню кількість фінансів для покриття витрат. Також однією з переваг такого підходу стала недостатня обґрунтованість існування піратських ресурсів, так як реклама була компромісом між якістю послуг та монетизацією, в той час як піратські ресурси існували виключно “на ініціативі” і не надавали такої якості обслуговування, як у великих комерційних компаній.

Платність послуг в якийсь момент була не кращим рішенням, так як тоді не існувало систем миттєвих платежів та систем оплати в інтернеті. Також, була проблема незахищеного контенту, який легко було копіювати та розповсюджувати на піратських ресурсах. З часом, коли з’явились необхідні фінансові технології, до цього ставилися скептично. Навіщо платити за щось, що є у вільному доступі? Це підштовхнуло компанії переглянути цінову політику, покращити якість наданих послуг, співпрацювати над розвитком технологій та унеможливити та/або ускладнити копіювання контенту.

Наразі, більшість людей користуються платними підписками, які коштують відносно небагато та надають необмежений доступ, в той час як раніше, необхідно було окремо купляти кожну пісню або альбом окремо, через що в людей були сумніви щодо обґрунтованості кожної конкретної покупки. Також, великі компанії витрачають чимало ресурсів на

дослідження задоволеності людей якістю сервісу, аби альтернатива у вигляді безкоштовних піратських сервісів не здавалася привабливою.

1.4.2 Захист даних від піратства

Захистити контент від інтернет-піратства на 100% є неможливим, але існують методи, які ускладнювали цей процес. Коли ресурси з розповсюдження контенту стали популярними, гостро постало питання захисту від несанкціонованого копіювання даних. На фізичних носіях це вирішувалося використанням технологій DRM, шифрування та ключів активації, що ускладнювало копіювання.

Одними з найпопулярніших методів захисту контенту в мережі інтернет стали: системи, які не зберігають контент на стороні клієнта повністю, а лише фрагменти, які необхідно дешифрувати з використанням коду, який отримано від серверу, який дозволяє таку дію виключно з використанням методу авторизації/аутентифікації;

Використання власних алгоритмів кодування даних, плеєри для яких створюються лише компанією-розробником сервісу. Такий метод є одним з найпоширеніших серед AAA-компаній, таких як Google, Netflix, які мають достатньо ресурсів для створення таких систем. Одним з методів обходу такого захисту, є використання видозмінених фірмових застосунків, в яких штучно прибрані етапи перевірки контенту та клієнта, з ціллю відтворення піратського контенту.

1.4.3 Захист від втручання зловмисників та захист персональних даних

Першим великим комп’ютерним вірусом вважається Pakistani Brain[4], створений групою пакистанських програмістів. На фоні цієї події, було створено один з перших в світі комп’ютерних антивірусів, який зараз

відомий під назвою McAfee. Цей застосунок був першим програмним продуктом, який розповсюджувався за принципом Умовно-безплатне програмне забезпечення[5], що полягає в тому, що ПЗ розповсюджується безкоштовно, але по закінченню пробного періоду за подальше користування доведеться платити.

Ця подія стала початком ери боротьби з хакерами. В тому числі, з захистом персональних даних. Одними з найбільш небезпечних вразливостей є: SQL ін’єкція, ін’єкція коду, незахищеність ліній зв’язку, недосконаліть прав доступу, невірна інтерпретація спецсимволів, XSS, нехтування налаштуваннями cookies[6, 7, 8].

Розглянемо кожну з цих вразливостей:

SQL ін’єкція. Нехай, ми маємо певний SQL запит в базу даних,який використовує користувацькі дані.

```
SELECT *
FROM users
WHERE id = *user_input*
```

Таким чином, користувацькі дані вигляду “id=123 OR 1=1” призведуть до того, що користувач отримає несанкціонований доступ до змісту бази даних. В тому числі, до записів, які містять персональні дані інших користувачів[9].

Ін’єкція коду. Ця вразливість актуальна не для всіх мов програмування. В часності, розглянемо мову програмування PHP. Сутність проблеми полягає в тому, що користувацькі дані, які не пройшли перевірку та фільтрацію, можуть замінити значення існуючих змінних, через що можливими стають такі вразливості: зміна прав користувача на більш розширені з ціллю отримання несанкціонованого доступу до операцій читання та зміни даних. В тому числі, для проведення більш масштабних атак;

зміна URL адрес, на які посилається сервер з ціллю перехоплення внутрішніх даних;

зміна параметрів з ціллю виведення застосунку з ладу[8].

Незахищеність ліній зв'язку. Зазвичай проявляється у невикористанні захищених протоколів (HTTPS), ігноруванні помилок валідації від центру сертифікації або використання слабких алгоритмів шифрування/хешування. Існує декілька варіантів перехоплення даних: перехоплення даних конкретного клієнта(-ів) або перехоплення всього трафіку серверу для отримання даних всіх клієнтів.

Недосконаліть прав доступу. Програмна вразливість, яка помилково надає клієнту доступ до дій, не передбачених системою. Існує єдиний можливий варіант виключення такої вразливості - принцип "все, що не дозволено - заборонено"[8].

Невірна інтерпретація спецсимволів. Навіть якщо користувацькі дані пройшли перевірку, це ще не означає, що ін'єкція коду є неможливою, так як користувацькі дані можуть використовувати спецсимволи, які виконують ті ж самі дії, але не визначаються механізмами валідації як небезпечні[8].

XSS. Нехай, наш продукт - портал, який підтримує розповсюдження користувацького коду (коментарі, наприклад). Якщо такі дані не пройшли валідацію, то в них можуть бути сховані користувацькі скрипти, які, наприклад, можуть змінювати посилання в коді сторінки на необхідні зловмиснику, що може призвести, в кращому випадку, до компрометації даних користувача[10].

Нехтування налаштуваннями cookies. В умовах розвитку інтернет порталів, одним з популярних алгоритмів авторизації є токен, який зберігається у вигляді cookies та слугує приватним ключем для авторизації. Компрометація такого токену з високою долею ймовірності призведе до

того, що зловмисник отримає доступ до облікового запису клієнта. Однією з вразливостей, яка дозволяє перехопити такі cookies є нехтування налаштуваннями політики cookies. Якщо в налаштуваннях неправильно налаштовано параметр SameSite, такі токени можуть бути передані стороннім сервісам[8,11].

1.5 Висновки розділу

В цьому розділі було розглянуто історію розвитку інформаційних серверних систем та проблематику їх створення. Окремо було розглянуто проблеми та нюанси розробки безпечної архітектури, які були використані під час розробки власного застосунку.

РОЗДІЛ 2. АРХІТЕКТУРА КЛІЄНТ-СЕРВЕРНИХ ЗАСТОСУНКІВ

Одними з найпопулярніших рішень для інтернет застосунків є клієнт-серверна архітектура та peer-to-peer архітектура.

2.1 Види комунікації

В залежності від задачі, доцільним буде використання різних видів комунікації між клієнтами та сервером (якщо такий підтримується архітектурою).

2.1.1 Клієнт-серверна архітектура

Переважна більшість рішень використовує саме клієнт-серверну архітектуру. Серед необхідних компонентів такої системи: набір клієнтів, сервер або набір серверів, мережа, яка забезпечує взаємодію клієнтів та серверів[12].

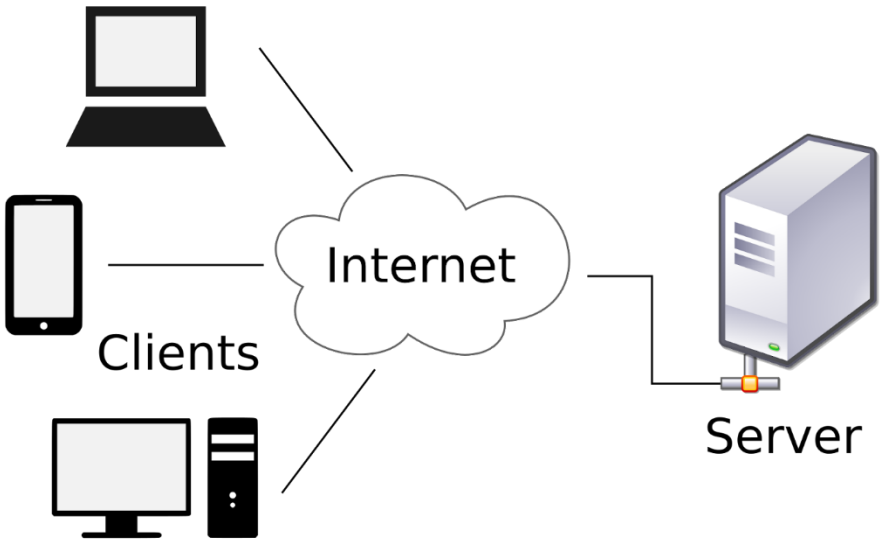

Рисунок 2.1 – схема клієнт-серверної архітектури

Переваги такого рішення:

а) Можливість організації системи з великою кількістю робочих станцій;

б) Централізація обчислювальних потужностей (див. рисунок 2.1), що дозволяє безпечно зберігати конфіденційні дані та полегшує адміністрування системи;

в) Ефективне використання мережевих ресурсів.

Недоліки:

а) Через централізацію обчислювальних потужностей, несправність серверу призводить до несправності програмного комплексу;

б) Адміністрування потребує висококваліфікованих спеціалістів;

в) Платність. Для функціонування таких систем необхідно використовувати сервери, підтримка яких потребує грошових затрат.

2.1.2 Peer-to-peer архітектура

Така архітектура є менш популярною, але нещодавно стала більш розповсюдженою у зв’язку з зростаючою популярністю децентралізованих систем, в основі яких лежать ідеї неконтрольованості третіми сторонами (переважно, органами влади), неможливості блокування і вільного розповсюдження[13].

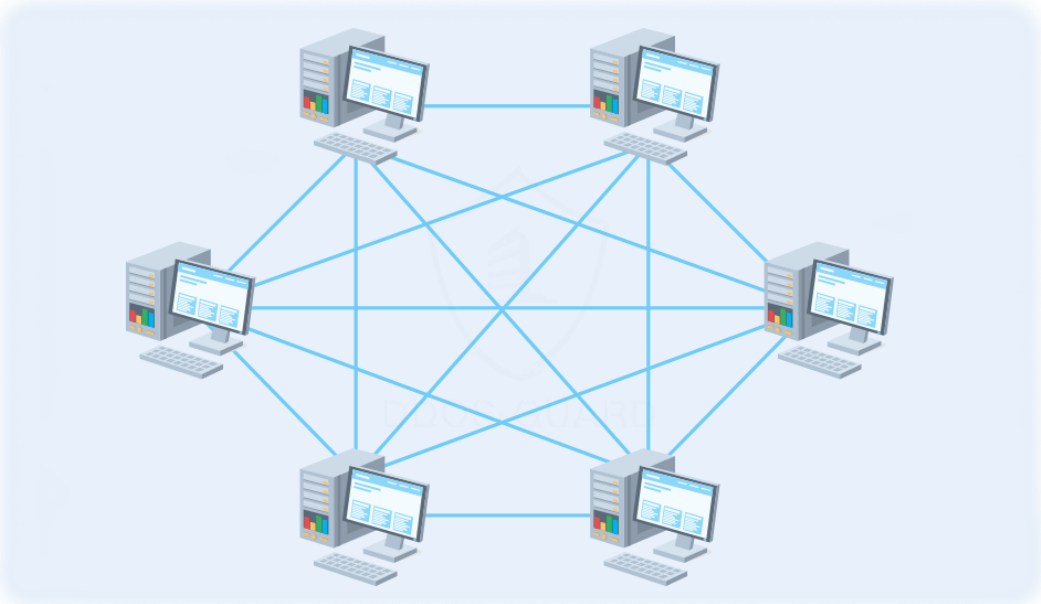

Рисунок 2.2 – схема peer-to-peer архітектури

Переваги такого підходу:

а) Немає серверу (див. рисунок 2.2)- немає плати за сервер. В таких системах вартість підтримки розподіляється між всіма учасниками;

б) Відмовостійкість. Якщо якась зі сторін втратила доступ, завжди можна перерозподілити трафік на інші піри;

в) Автономія. Такі системи неможливо заблокувати, не заблокувавши всіх учасників, що, фактично, неможливо в нормальних умовах;

г) Конфіденційність. P2P системи зазвичай розробляються таким чином, що повідомлення відправляються не напряму адресату, а через інших учасників, що дозволяє анонімізувати відправника.

Недоліки:

а) Так як система динамічна і розрахована на те, що учасники можуть створювати та розривати з’єднання, неможливо гарантувати стабільність системи;

б) Так як такі системи, зазвичай, відкриті, необхідно створювати додаткові алгоритми узгодження та перевірки даних на відповідність, на що витрачається чимало ресурсів.

2.1.3 Гібридна архітектура

P2P архітектура, в якій сервер виконує функцію верифікацію вузлів, містить інформацію про них та займається розподілом ресурсів, а вузли, в свою чергу, виконують функцію забезпечення ресурсами.

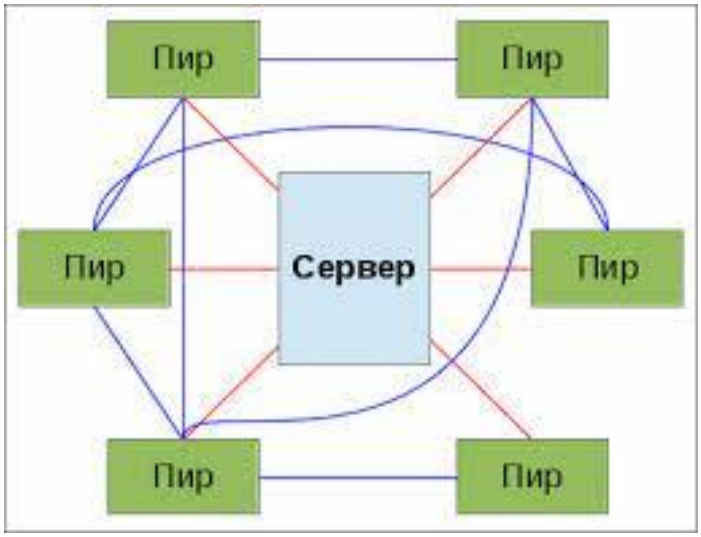

Рисунок 2.3 – схема гібридної архітектури

Переваги такого підходу:

а) Можливість рівномірно розподіляти навантаження на вузли;

б) Можливість валідувати вузли.

Недоліки:

а) Зберігається залежність від сервера (див. рисунок 2.3), як в клієнт-серверній архітектурі;

б) Не забезпечується повна конфіденційність: сервер знає про всі вузли, але не має доступу до інформації, яка передається між ними.

2.2 Високорівневі прикладні програмні інтерфейси

Прикладний програмний інтерфейс - архітектурний стиль розробки клієнт-серверних додатків. До високорівневих, простими словами, відносяться ті API, для створення яких немає необхідності взаємодіяти з низькорівневими (такими як Win32 API чи Linux Kernel API), а достатньо взаємодіяти за допомогою сторонніх програм[14, 15].

2.2.1 REST API

Representational State Transfer - стиль, який рекомендую для кожної атомарної дії використовувати окремий path, мати власний endpoint з відповідним методом(-и). В якості протоколу зазвичай використовується HTTP. Формати передачі даних:

а) JSON;

б) XML;

в) Бінарні дані;

г) CSV.

Однак, найпопулярнішим є JSON. Також допускається використання protobuf в цілях оптимізації використання ресурсів[15].

2.2.2 RPC

Протилежність до REST API. Містить всього один endpoint та викликає різну логіки в залежності від змісту отриманого повідомлення. Також зазвичай використовує HTTP в якості протоколу транспортного рівня[15].

2.2.3 GraphQL

Розробка компанії Facebook (нині Meta). Містить певний перелік функціоналу та йде “по графу”. Використовує JSON для передачі даних, достатньо всього одного ендпоінта. Сильна типізація, за рахунок чого не може бути неоднозначності та помилок, наприклад, в застосунках/сервісах, які написані мовами програмування зі слабкою системою типів. Відповідь завжди повертається в форматі, який відповідає певній конкретній схемі даних. Найбільш доцільним буде використання під час розробки великих систем[15].

2.3 Протоколи обміну даних

Одними з найбільш відомих є HTTP(-S), TCP, UDP, WebSocket. Кожен з них має свої плюси та мінуси та широко використовується в залежності від задачі.

2.3.1 HTTP(-S)

Найбільш розповсюджений протокол в мережі інтернет. Використовується для створення клієнт-серверних застосунків.

Плюси:

а) З версії 2.0 передає дані у бінарному вигляді;

б) Підтримує механізми цілісності та впорядкованості даних;

в) Високий рівень стандартизації;

г) Підтримує безпечну передачу даних у версії HTTPS.

Недоліки:

а) Потребує використання додаткової пам’яті для відправки заголовку запиту;

б) Потребує додаткового часу на перевірку цілісності та впорядкованості даних.

2.3.2 TCP

Використовується для передачі великої кількості даних з дотриманням впорядкованості та цілісності. Створює з’єднання один раз та використовує його для подальшої передачі даних.

Плюси:

а) Підтримує механізми цілісності та впорядкованості даних;

б) Швидший за HTTP.

Недоліки:

а) Потребує використання додаткової пам’яті для відправки заголовку запиту;

б) Потребує додаткового часу на перевірку цілісності та впорядкованості даних.

2.3.3 UDP

Використовується для швидкої передачі даних, але не підтримує механізми цілісності та впорядкованості даних.

2.3.4 WebSocket

Особливий протокол, дозволяє обмінюватись даними в режимі real-time. Всі протоколи, які були розглянуті до цього працюють за схожою схемою: створення з’єднання (за необхідністю), клієнт відправляє запит, сервер обробляє запит та надсилає відповідь, з’єднання розривається (якщо було створено).

Протокол WebSocket працює за іншим принципом: створюється з’єднання, клієнт та сервер можуть обмінюватись повідомленнями один з одним, з’єднання розривається. Підтримується всіма сучасними браузерами.

2.4 Алгоритми захисту аутентифікаційних даних

Автентифікація — процедура ідентифікації користувача за інформацією в системі пред'явленого ним ідентифікатора[20].

Автентифікація є частиною процедури надання доступу для роботи в інформаційній системі, наступною після ідентифікації і передує авторизації.

Авторизація – процедура керування рівнями доступу користувача до захищеного ресурсу.

Алгоритми захисту аутентифікаційних даних в базі даних містять декілька можливих ступенів захисту

2.4.1 Незахищені дані автентифікації

Паролі зберігаються у відкритому вигляді. Такий підхід є найбільш небезпечним, так як компрометація бази даних призведе до компрометації аутентифікаційних даних для всіх користувачів, чиї дані було скомпрометовано. Рівень небезпеки - червоний.

2.4.2 Захешовані дані автентифікації

Такий підхід відрізняється від минулого хешуванням даних перед зберіганням в базу даних. Так як хешування - необоротний процес, для компрометації даних доведеться підібрати алгоритм хешування, який використовується в даній системі.

Основним недоліком є необмеженість спроб перевірки пароля локально на стороні зловмисника. Цей процес займає певний час, тому такий алгоритм вважається більш досконалим, ніж минулим, але все ще недостатньо захищеним. Рівень небезпеки - помаранчевий.

2.4.3 Застосування “солі” до даних автентифікації

Сутність полягає в тому, щоб хешувати пароль не в чистому вигляді, а з використанням форматування з певним приватним ключем.

Для компрометації, такий підхід вимагає, окрім доступу до бази даних, ще й доступ до приватного ключа та алгоритму форматування. Однак, якщо були скомпрометовані і ці дані, складність компрометації майже така-ж сама, що й в минулому випадку. Рівень небезпеки - жовтий.

2.4.4 Використання алгоритму хешування з великою часовою складність

Найбільш досконалий механізм, який існує на даний момент. В якості алгоритму хешування обирається такий, часова складність якого не дозволяє перебирати всі варіації пароля, так як це займає забагато часу навіть для компрометації одного набору аутентифікаційних даних.

З мінусів слід відмітити великі часові витрати на розрахування хешу, що може вирішуватись завдяки асинхронності операцій (клієнт отримає відповідь від сервера про те, що його пароль відповідає вимогам безпеки, токен для авторизації, а хеш від його даних ще якийсь час буде рахуватись і збережеться в базу даних). Рівень небезпеки - зелений.

2.5 Висновки розділу

В цьому розділі було розглянуто різновиди архітектури серверних застосунків, методи створення та різновиди прикладних програмних інтерфейсів і транспортні протоколи.

Ця інформація була використана під час розробки власного застосунку.

РОЗДІЛ 3. БЛОКЧЕЙН ЗАСТОСУНКИ

В основі застосунків з використанням блокчейн лежать блоки - події, які змінюють стан системи. Загалом база даних блокчейн застосунку має вигляд однозв'язного списку з посиланням на предка, де кожен вузол (блок) містить інформацію про зміни, внесені в стан системи (транзакції) та хеш, який використовується для валідації цілісності та достовірності існуючого списку.

З плюсів можна відмітити неможливість непомітно змінити минулі стани системи, так як це впливає на хеш блоку та, за ланцюжком, змінить хеші всіх подальших блоків.

З мінусів слід зазначити велику кількість додаткових операцій, направлених на підтримання безпеки.

3.1 Структура блокчейн блоку

До [16] обов'язкових полів відносяться:

а) Транзакції;

б) Номер блоку;

в) Хеш блоку-предка;

г) Час створення блоку;

д) Мінімальна можлива комісія за всі транзакції блоку;

е) Посилання на стан систему з урахуванням цього блоку.

3.2 Структура блокчейн транзакції

Обов'язкові поля:

а) Адреса отримувача - унікальний 20-байтовий ключ (в ethereum);

б) Підпис відправника. Генерується з використанням приватного ключа і підтверджує валідність транзакції;

в) Сума - значення суми, яка буде відправлена отримувачу;

г) Ліміт комісії - максимальне значення комісії, яке може бути використано транзакцією;

д) Максимальне значення комісії валідатора;

е) Максимальне значення, що буде виплачено за транзакцію;

ж) Номер транзакції, розраховується вже після відправлення.

Необов'язкові поля:

а) Дата - містить будь-яку додаткову інформацію.

3.3 Механізми створення блоків

В основі блокчейн застосунків лежать розподілені системи (вони ж peer-to-peer, див. 2.1.2). З метою запобігання зловживанням відкритості системи, існують механізми аутентифікації користувачів та розподілу нагороди (комісії).

Одним з найпопулярніших механізмом розподілу нагороди серед валідаторів/майнерів є proof-of-work. Для того, щоб створити блок необхідно методом перебору визначити число, яке буде по певному правилу співвідноситись з випадково згенерованим числом. Після того, як число було обчислено, валідатор надсилає блок всім учасникам мережі, вони перевіряють, що знайдене число дійсно співвідноситься з випадковим. Після цього майнер, який створив блок, отримає більшу частину комісії з транзакцій, які містить блок та константну винагороду. Частину від залишку комісії буде розподілено між іншими валідаторами, а решта буде знищена.

Такий механізм дозволяє уникати стрімкому збільшенню валюти в обігу, аби зменшити знецінювання, викликане існуванням константної винагороди, яка береться “з повітря”.

3.3.1 Механізм консенсусу

Так як блокчейн - відкрита децентралізована платформа, задля унеможливлення діяльності зловмисників, існує механізм консенсусу.

Алгоритм proof-of-work вимагає знаходження випадкового числа, на що витрачаються ресурси для розрахунку. Для того, щоб блок вважався достовірним, необхідно, аби 50% + 1 користувачів погодились, що число, отримане валідатором, є правильним.

Існуючий механізм, який вимагає пошуку числа, існує задля унеможливлення вразливості “атака 51%”, так як для володіння навіть декількома відсотками розрахункових потужностей – надзвичайно складна задача. Більш того, при виявленні “нечесної гри”, учасники мережі можуть розірвати з’єднання з таким вузлом.

3.3.2 Алгоритм вирівнювання дерева

Так як на один блок може посилатись одразу декілька блоків, це створить дерево, що протирічить системі та створює неоднозначності.

Задля уникнення таких протирічь, достовірним буде вважатись лише той блок, після якого створено найбільший ланцюжок. Таким чином, в стан системи попадуть лише блоки, які були найбільш схвалювані, або ті, які базувались на їх основі.

3.3.3 Протидія атаці Сивілли

Атака Сивілли полягає в тому, що в розподіленій системі

певний сегмент мережі буде штучно ізольований зловмисниками, що дозволить отримувати транзакції для обробки та дозволить уникнути конкуренції при розрахунку блоку.

Число, яке необхідно знайти, залежить від ланцюжка блоків, які передують поточному. В системі також зберігається інформація про майбутні транзакції та комісію, яку можна за них отримати. Таким чином, найбільш пріоритетними транзакціями для попадання в блок будуть ті, які містять найбільшу нагороду валідатору.

Згідно з алгоритмом вирівнювання ланцюжка, обирається та гілка, яка містить найдовший список блоків. Таким чином, валідатор-зловмисник має 2 варіанти: створювати блок на основі вже створеного і не звертати увагу на нові створені або починати розрахунки спочатку після створення кожного нового блоку.

Таким чином, в першому варіанті, блок не попаде в основний список блоків та за нього буде отримано мінімальну нагороду, а транзакція буде передана в мережу і попаде вже в інший блок, за який нагороду отримає чесний валідатор.

В другому випадку, розрахункові потужності зловмисника мають бути достатніми для обчислення числа першим, аби мати можливість помістити транзакцію в блок і отримати за неї нагороду. Так як необхідні розрахункові потужності для розрахунку блоку в числі перших, мають бути величезними, а учасники мережі залишають за собою можливість розривати з’єднання з нечесними майнерами, “грати по правилам” виявляється більш стабільною та прибутковою стратегією.

Таким чином, проведення атаки Сивілли є неприбутковим в будь-якому випадку, що призводить до необгрунтованості її використання.

3.4 Огляд існуючих систем

Окрім вище наведених прикладів з криптовалютами, технологія блокчейн також дозволяє створювати багато інших систем, серед яких особливу увагу хочу приділити таким:

а) Система NFT, яка створюється для обміну та/або продажу будь-яких незамінних токенів. В якость таких токенів можуть виступати будь-які товари. Навіть два однакових товари можуть містити різні характеристики.

На прикладі БВ автомобілів можна виділити такі відмінності навіть серед машин однакової моделі та комплектації: стан, пробіг.

В той же час, токени для нових авто можуть бути взаємозамінними, так як окремо взяті нові автомобілі нової комплектація мають однакові характеристики;

б) Система сертифікатів коронавірусу для аеропортів. Така система дозволяє децентралізовано (що важливо) обмінюватись інформацією між аеропортами щодо пасажирів. Основним плюсом можна виділити відмовостійкість. Навіть якщо якийсь з аеропортів тимчасово від’єднався від мережі, це не призведе до проблем в інших аеропортах, а коли з’єднання буде відновлено, вся інформація буде передана іншим учасникам мережі (аеропортам).

Така система буде містити не всі характеристики, притаманні блокчейн, так як створена задля спільної мети та не містить конкуренції серед вузлів.

3.5 Висновок розділу

В цьому розділі було розглянуто особливості систем, які базуються на технології блокчейн та алгоритмів, які забезпечують стабільність та захищеність системи.

Ця технологія дозволяє створювати відмовостійкі системи, які будуть мати такі характеристики:

а) Відмовостійкість;

б) Захищеність;

в) Відкритість.

Було розглянуто приклади існуючих систем. Отриманий досвід було використано під час розробки власного застосунку.

РОЗДІЛ 4. ОПИС ТЕХНОЛОГІЙ ТА МОВ ПРОГРАМУВАННЯ ДЛЯ РІШЕННЯ ЗАДАЧІ

4.1 Опис технологій

На сьогоднішній день сфера IT розвивається у багатьох галузях і для роботи кожної з них використовується свій набір технологій та мов програмування. Кожна з технологій має свої правила застосування та використання для вирішення задач у певній галузі застосування.

Використані технології:

а) Мова програмування Go/Golang;

б) NoSql СУБД MongoDB;

в) Веб-фреймворк GIN;

г) Бібліотека golang-lru.

4.2 Мова програмування GO

Мова GO була створена наприкінці 2000-х років компанією Google, так як існуючі на ринку рішення не відповідали таким вимогам:

а) Простий інтерфейс взаємодії з потоками/корутинами;

б) Швидкодія, достатня для використання в якості інструменті для створення високонавантажених сервісів (мова створювалась для написання мікро-сервісів, в першу чергу, для YouTube);

в) Читаємість та підтримуваність коду;

г) Існування якісних інструментів для тестування, бенчмаркінгу та аналізу коду;

д) Сильна система типів.

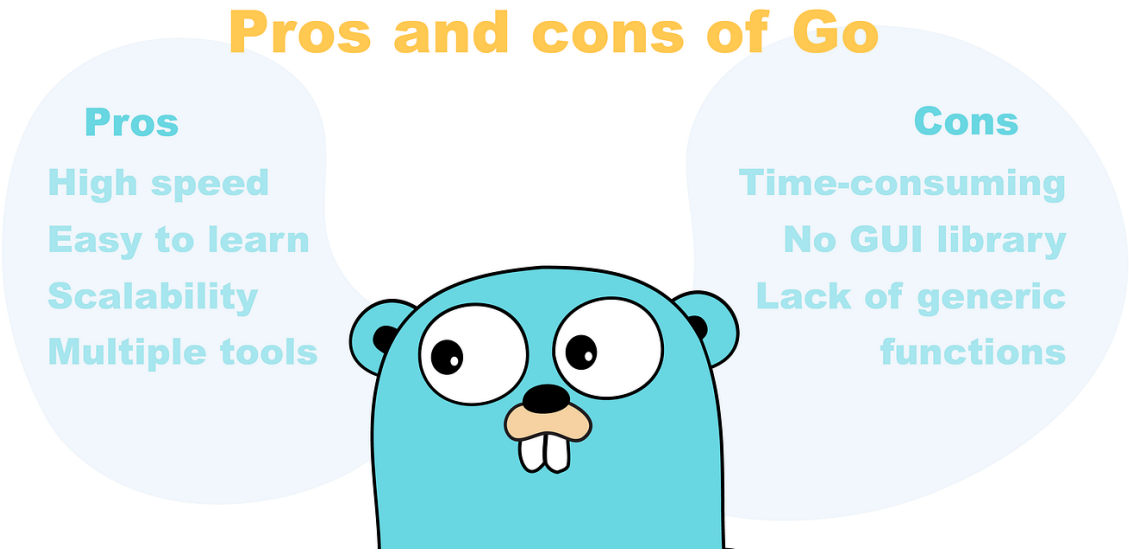

Рисунок 4.1 – переваги та недоліки мови програмування GO [18]

Також хочу відмітити, що криптовалютна платформа ethereum була написана на трьох мовах програмування: Python, C++ та GO[16].

Реалізація на перших двох не стала популярною, на відміну від GO. Python через недостатню швидкодію, C++ – складність підтримки та розробки.

Обгрунтування вибору. Мова програмування GO поєднує в собі характеристики (див. рисунок 4.1), які дозволяють створити веб-сервіс, який може обробляти велику кількість запитів та дозволяє писати зрозумілий, підтримуваний та масштабований код[18].

4.3 NoSQL СУБД MongoDB

MongoDB - документоорієнтована система управління базами даних, яка не потребує опису таблиці. Математично доведено ефективність її використання на великих об’ємах даних (до кількох петабайт). Містить алгоритми оптимізації пошуку. Всі записи зберігаються у відсортованому за часом виді. Має офіційний драйвер для мови програмування GO.

Обгрунтування вибору. Задачі, з якою MongoDB добре справляється: зберігання інформації про події, задачі електронної комерції, що корелюється з темою кваліфікаційної роботи.

4.4 Веб-фреймворк GIN

Швидкий веб-фреймворк, який підтримує такий функціонал[19]:

а) middleware;

б) групування маршрутів;

в) шорт-кати для JSON;

г)управління помилками, що спрощує налаштування відмовостійкості.

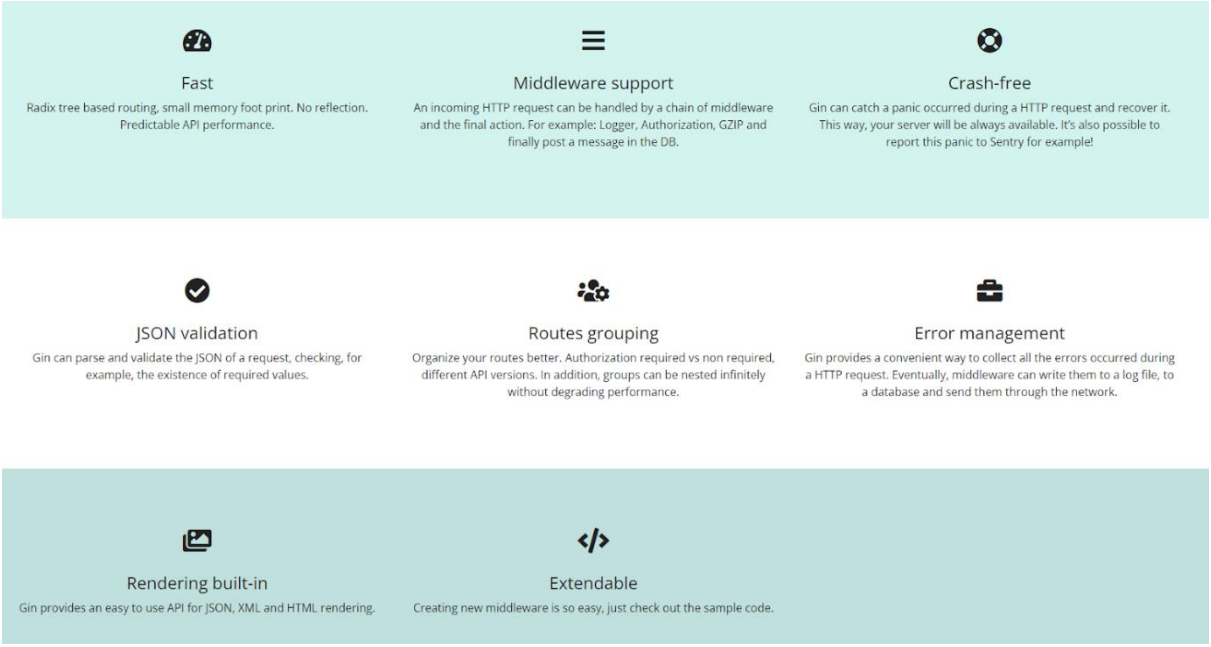

Рисунок 4.2 – переваги веб-фреймворку Gin [25]

Обгрунтування вибору. Найбільш доцільна бібліотека з існуючих. Поєднує швидкодію та необхідний функціонал (див. рисунок 4.2).

4.5 Бібліотека golang-lru

Містить в собі реалізацію контейнеру LRU cache.

Обгрунтування вибору. Використовується для зберігання станів системи. Автоматично видаляє з операційної пам’яті застарілі записи при переповненні заданого розміру і заміняє новим записом.

РОЗДІЛ 5. РЕАЛІЗАЦІЯ ДОДАТКУ КРИПТОВАЛЮТНИХ ПЛАТЕЖІВ

5.1 Функціональні вимоги до створюваного застосунку

Зважаючи на отриману інформацію, сформуємо функціональні вимоги до розробляємого додатка:

а) Містить алгоритми автентифікації;

б) Використовує алгоритми захисту даних для записів бази ;

в) Містить механізми відмовостійкості;

г) Захищена від найбільш популярних методів атак (див 1.4.3);

д) Використовує систему блокчейн;

е) Дозволяє створювати такі типи транзакцій:

1. Відкладені:

   за часом;

   за номером блоку.

2. Відкладені умовні:

   баланс користувача більше/менше, ніж;

   користувач відпривив транзакцію.

ж) Дозволяє переглядати статистику з підтримкою таких фільтрів:

1. Тип(-и) транзакції;
2. Отримувач;
3. Відправник;
4. Сума транзакції (проміжок);
5. Час транзакції (проміжок).

з) Дозволяє гнучко налаштовувати систему за допомогую параметрів конфігурації.

5.2 Опис архітектури

Для розробки додатку для ведення сімейного бюджету було обрано використовувати такий набір технологій:

Go/Golang, MongoDB, Gin framework, golang-lru

В якості мети розробки було обрану клієнт-серверну архітектуру. Як було зазначено раніше, даних підхід є основою для написання власного веб-додатку.

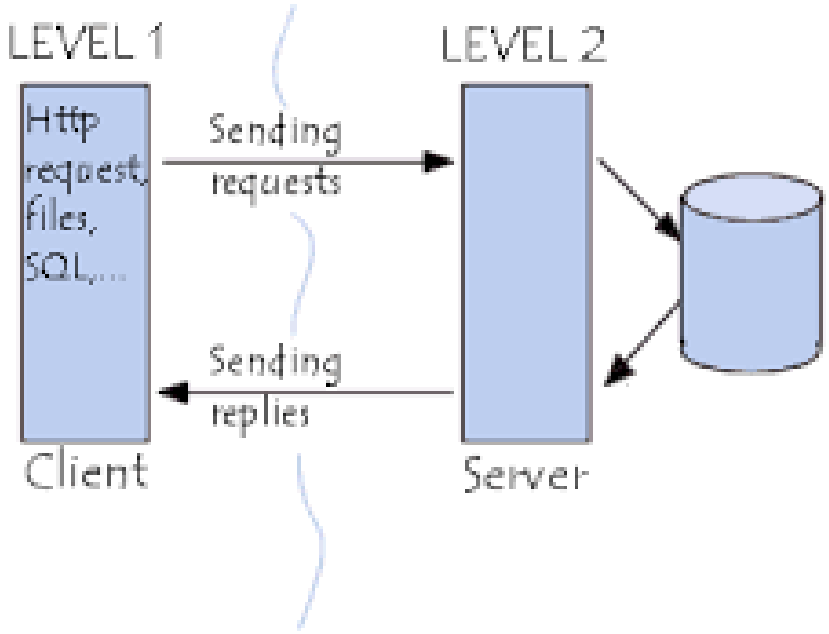

Рисунок 5.2 – діаграма клієнт-серверного застосунку з СУБД

Як можна бачити на рисунку 5.2, стандартна архітектура клієнт-сервер складається з трьох частин:

1) Користувацький інтерфейс або рівень клієнта: це частина програмного
забезпечення, яка взаємодіє з користувачами. Цей рівень містить екрани входу, меню, екрани даних та звіти, які передають та приймають

інформацію до та від користувачів, тобто відповідає за відображення інформації. В моєму додатку такий рівень не передбачено.

2) Сервер додатків: Це сервер, на якому встановлені програмні модулі програми. Він підключається до бази даних і взаємодіє з користувачами. Являється логікою програми, що є рівнем обробки, діє як міст для з'єднання логіки програми з даними на серверах баз даних тощо[17].

3) Сервер бази даних: Цей сервер містить таблиці, індекси та дані, керовані додатком. Тут виконуються операції пошуку, вставки, видалення, оновлення та інші операції з даними.

5.3 Опис API

API містить такі ендпоінти:

а) register - POST ендпоінт створення нового користувача. Серед вимог: унікальне ім’я користувача, пароль довжиною мінімум 8 символів, який містить букви різного регістру та цифри. Використання спецсимволів не є обов’язковим;

б) sendTx - POST ендпоінт відпривки транзакції.

Основні види транзакцій: Transfer, Spending, Obtaining.

Вторинні характеристики транзакцій: звичайні, відкладені;

в) getKey - GET ендпоінт. Дозволяє отримати публічний ключ користувача. Використовується для отримання публічного ключа користувача, який необхідний для збору статистики та відправки транзакцій;

д) getTxsWithFilters - GET ендпоінт. Дозволяє отримати список транзакцій за фільтрами: тип(-и) транзакції, отримувач, відправник, сума транзакції (проміжок), час транзакції (проміжок). Кожний з цих фільтрів є необов’язковим. Підтримується запит без фільтрів. В такому випадку, клієнту повернеться відповідь, яка міститиме всі транзакції.

5.4 Опис бази даних

База даних містить три таблиці:

а) users;

б) state;

в) transactions.

5.4.1 Таблиця users

Містить в собі інформація про облікові записи користувачів

а) nickname - поле, яке містить інформацію про ім’я користувача;

б) created_at - час створення аккаунту;

в) hashed_password - хеш пароля користувача, використовується для автентифікації.

5.4.2 Таблиця state

Містить в собі всі записи зі створеними блоками

а) number - номер блоку;

б) transactions - масив транзакцій, які містить блок;

в) parentHash - хеш батьківського блоку;

г) timeStamp - час створення блоку;

д) hash - хеш блоку.

5.4.3 Таблиця transactions

Містить в собі всі майбутні транзакції. Існує з ціллю запобігання втрати транзакцій, які ще не потрапили в блок

а) ID - унікальний номер транзакції;

б) timestamp - час отримання запиту про відправку транзакції;

в) from - відправник транзакції;

г) to - отримувач транзакції;

д) value - сума транзакції;

е) description - необов'язкове поле з будь-якою додатковою текстовою інформацією;

ж) txType - тип транзакції;

з) condition - необов'язкове поле з інформацією про умову відкладеної транзакції.

5.5 Демонстрація можливостей застосунку

В результаті розроблений додаток відповідає всім функціональним вимогам. Далі ми детальніше розглянемо розроблену систему продемонструючи її інтерфейс та інструменти які вона надає.

Після того, як програму було запущено, отримуємо лог з інформацією про параметри конфігурації

```
Config:{
    "DataDirectory":".data/node",
    "HttpAddress":"localhost",
    "HttpPort":"9545",
    "LogsDir":"",
    "DataBaseAddress":"localhost",
    "DataBasePort":"27017",
    "DataBaseName":"BC",
    "UsersCollectionName":"users",
    "StateCollectionName":"state",
    "TransactionsCollectionName":"transactions",
```

```
        "ApiOnly":true
    }
```

Також отримуємо повідомлення про те, що сервер успішно запущено: “*server is running*”

Відправимо запити на створення облікових записів

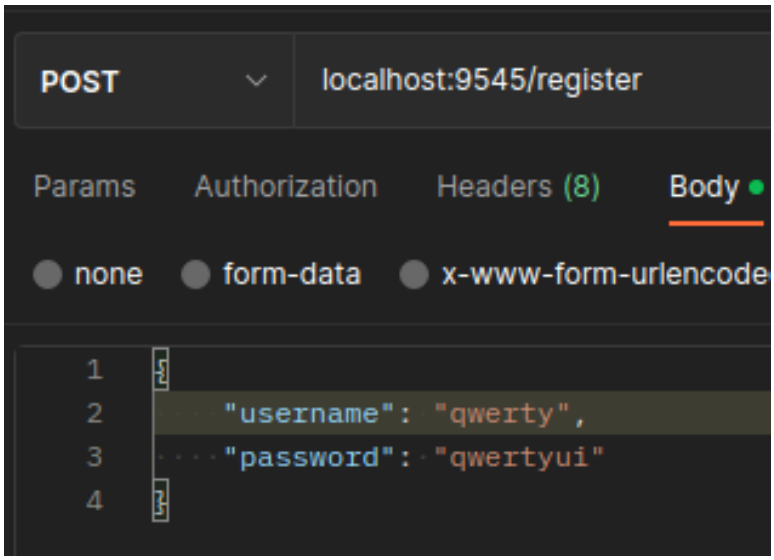

Рисунок 5.1 – запит на створення облікового запису

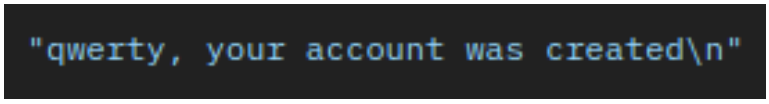

Рисунок 5.2 – приклад відповіді серверу на запит реєстрації

Скориставшись інтерфейсом бази даних, можемо впевнитись, що користувачі дійсно були створені

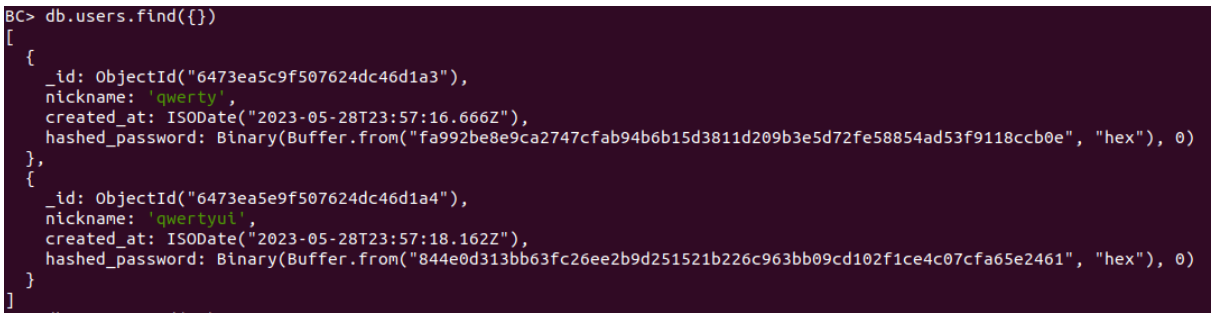

Рисунок 5.3 – записи в базі даних, які свідчать, що акаунти були створені

Відправимо запит на отримання публічного ключа

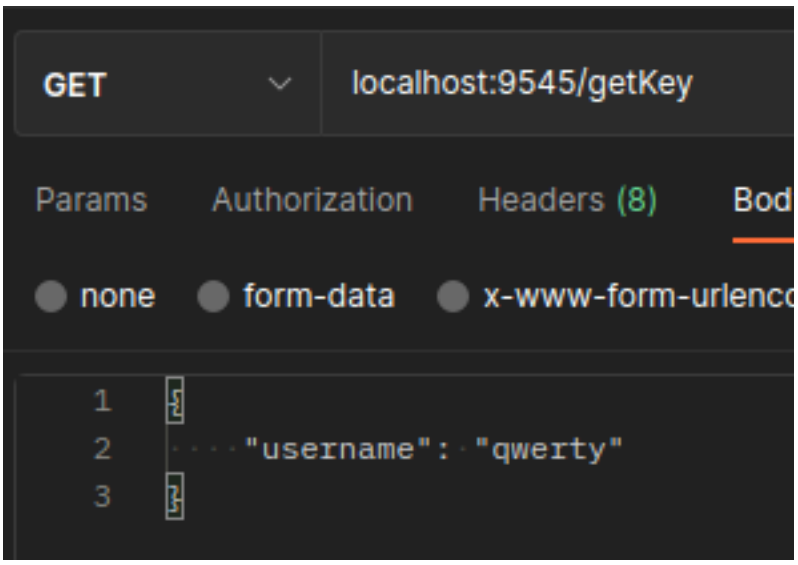

Рисунок 5.4 – запит на отримання публічного ключа

У відповідь отримуємо унікальний 20-бітовий ключ.

Відправимо декілька транзакцій. Перевіримо базу даних. Під номером 0 було створено так званий стейт блок, який не містить транзакцій та виконує функцію батьківського блоку для першого. Також бачимо два блоки, кожен з яких містить транзакції, які ми відправили.

Відправимо запити на перевірку балансу гаманця.

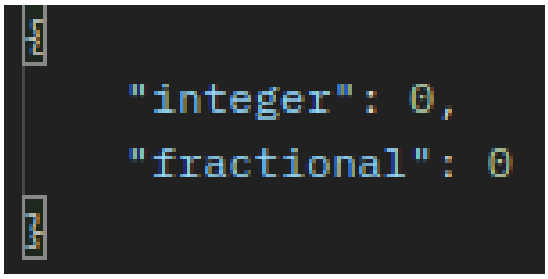

Рисунок 5.5 – баланс в блоці під номером 0

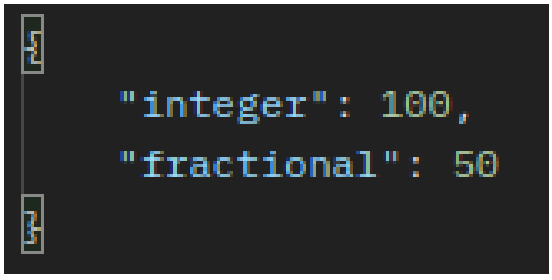

Рисунок 5.6 – баланс в блоці під номером 1

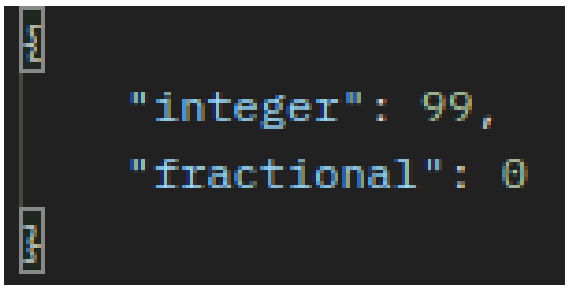

Рисунок 5.7 – баланс в блоці під номером 2

Дійсно, все правильно. В блоці під номером 0 наш баланс нульовий.

На момент першого блоку баланс - 100.5 одиниць, після зарахування транзакції-поповнення.

На момент другого блоку баланс - 99 одиниць, так як була зарахована транзакція-переказ на інший гаманець.

Також можемо бачити логи з базовою інформацією про запити, які були оброблені

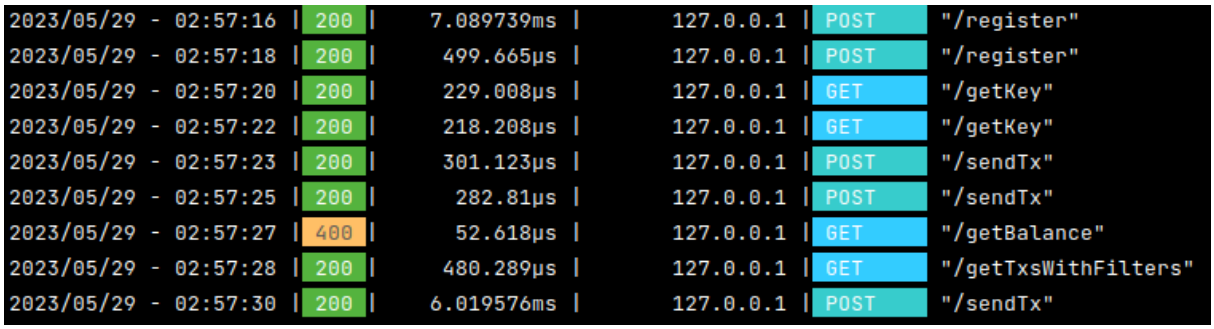

Рисунок 5.8 – приклад логів з серверу

ВИСНОВКИ

Мною було вивчено такі аспекти розробки програмного забезпечення:

а) Вразливості та методи їх усунення, унеможливлення або ускладнення;

б) Відмовостійкість та методи її забезпечення;

в) Транспортні протоколи, їх відмінності та задачі, для виконання яких вони створені;

г) Архітектура, організація та відмінності багатокористувацьких застосунків.

При розробці серверного застосунку було використано передові технології. Було отримано захищений від більшості атак веб-застосунок, який дозволяє переглядати службову інформацію про стан системи та гнучко конфігурувати її.

Можливі ідеї для розвитку проекту: підтримка ключів сесій; підключення системи до міжнародних банківських систем та/або інших криптовалютних мереж; створення додаткових механізмів безпеки, таких як захист від перевантаження та великої кількості запитів. В такому випадку, система може стати конкурентноздатною в сфері централізованих криптовалют.

Також, в ході розробки, були виявлено, що використання у якості СУБД mongoDB було не кращим рішенням. Запити з фільтрами реалізовані не дуже зручно. Ймовірно, кращим рішенням було б використання реляційних СУБД з підтримкою SQL.

Також було виявлено, що створення складних захищених веб-сервісів потребує від розробника певного багажу знань з таких дисциплін: кібербезпека, бази даних, комп’ютерні мережі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Opinion There May Soon Be Three Internets. America's Won't Necessarily Be the Best. (Published 2018). Архів оригіналу. [Електронний ресурс]: https://www.nytimes.com/2018/10/15/opinion/internet-google-china-balkanization.html

2. Gaston Gazette. «Video stores still making a go at attracting business (answer poll).» [Електронний ресурс]: https://web.archive.org/web/20101109021952/http://www.gastongazette.com/news/business-51015-stores-folks.html

3. Dawson, Jennifer. «The incredible shrinking video stores!». [Електронний ресурс]: https://web.archive.org/web/20071009193056/http://houston.bizjournals.com/houston/stories/2006/04/24/story2.html

4. John Leyden. PC virus celebrates 20th birthday (англ.). The Register. [Електронний ресурс]: https://web.archive.org/web/20181225040640/https://www.theregister.co.uk/2006/01/19/pc_virus_at_20/

5. Умовно-безкоштовне програмне забезпечення. [Електронний ресурс]: http://lua.pp.ua/1/228492.html

6. Термінологія в галузі захисту інформації в комп’ютерних системах від несанкціонованого доступу. Київ: Департамент спеціальних телекомунікаційних систем та захисту інформації Служби безпеки України.

7. The Web Application Security Consortium / Web Application Security Statistics. [Електронний ресурс]: http://projects.webappsec.org/w/page/13246989/Web-Application-Security-Statistics#APPENDIX2ADDITIONALVULNERABILITYCLASSIFICATION

8. "OWASP Top 10 2013 A1: Injection Flaws". OWASP. [Електронний ресурс]: https://owasp.org/www-project-top-ten/’

9. SQL injection. [Електронний ресурс]: https://www.w3schools.com/sql/sql_injection.asp

10. Martin Anderson (23 Feb 2016). Cross-site scripting enabled on 1000 major sites – including financial sites. [Електронний ресурс]: https://thestack.com/security/2016/02/23/cross-site-scripting-enabled-on-1000-major-sites-including-financial-sites/

11. PH22157: ADD SUPPORT FOR THE SAMESITE COOKIE ATTRIBUTE. IBM. [Електронний ресурс]:

https://www.ibm.com/docs/ru/control-desk/7.6.1.x?topic=checklist-vulnerability-cookie-without-samesite-attribute

12. Іван Змерзлий. Клієнт-серверна архітектура та ролі серверів. Feb 1, 2017. [Електронний ресурс]: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229

13. Knowledge Base DDoS-Guard. What Is Peer-to-Peer (P2P) Network? [Електронний ресурс]: https://ddos-guard.net/en/terminology/technology/peer-to-peer-p2p

14. Clarke, Steven. Measuring API Usability. Dr. Dobb's. [Електронний ресурс]: https://www.drdobbs.com/windows/measuring-api-usability/184405654

15. Oleksii Ostapov. Що таке API та які вони бувають? 9 Червня 2022. [Електронний ресурс]: https://qamania.org/blog/що-таке-api-та-які-вони-бувають/

16. ETHEREUM DEVELOPMENT DOCUMENTATION. [Електронний ресурс]: https://ethereum.org/en/developers/docs/

17. Fielding Roy. Architectural Styles and the Design of Network-based Software Architectures. — Каліфорнійський університет в Ірвайні, 2000. [Електронний ресурс]: https://www.webcitation.org/67gOwyTek

18. GO programing language official documentation. [Електронний ресурс]: https://go.dev/doc/

19. Gin web-framework official documentation. [Електронний ресурс]: https://gin-gonic.com/docs/

20. Todorov D. Mechanics of User Identification and Authentication: Fundamentals of Identity Management. — CRC Press, 2007.