Хмельницький національний університет
Факультет інформаційних технологій
Кафедра комп'ютерної інженерії та інформаційних систем

КВАЛІФІКАЦІЙНА РОБОТА

бакалавр
Освітній рівень

Кіберфізична система білінгу та управління ресурсами для віртуальних серверів
Назва теми

КвРКІ 022027.22.02.24 ПЗ
Шифр

Галузь знань *12 «Інформаційні технології»*
Шифр, назва

Спеціальність *123 «Комп'ютерна інженерія»*
Шифр, назва

Освітня програма *«Комп'ютерна інженерія та програмування»*
Назва

Виконав: студент III курсу, група КІ2с-22-2 _____ Олексій МОТУРНЯК
Підпис — Ініціали, прізвище

Керівник _____ Олексій ІВАНОВ
Підпис, дата — Ініціали, прізвище

Нормоконтролер _____ Тетяна КИСІЛЬ
Підпис, дата — Ініціали, прізвище

До захисту допускаю:
зав. кафедри комп'ютерної інженерії та інформаційних систем _____ Ольга ПАВЛОВА
Підпис — Ініціали, прізвище

«5» червня 2025 р.

Хмельницький 2025

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій

Кафедра КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

Освітній рівень БАКАЛАВР

Галузь знань 12 ІНФОРМАЦІЙНІ ТЕХНОЛОГІЇ

Спеціальність 123 КОМП'ЮТЕРНА ІНЖЕНЕРІЯ

Освітня програма «Комп'ютерна інженерія та програмування»

ЗАТВЕРДЖУЮ
Зав. кафедри Ольга ПАВЛОВА

" 10 " 01 2025 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ БАКАЛАВРА

Мотурняк Олексій Андрійович
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Кіберфізична система білінгу та управління ресурсами для віртуальних серверів

Керівник проекту (роботи) Іванов Олексій Валентинович, д.т.н., проф.
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 07.02.2025 р. № 23

2. Строк подання студентом проекту (роботи) на кафедру 01.06.2025 р.

3. Вихідні дані до проекту (роботи) Завдання на кваліфікаційну роботу

4. Зміст пояснювальної записки (перелік питань, які потрібно розробити)

Кіберфізична система білінгу та управління ресурсами для віртуальних серверів та постановка задачі щодо її удосконалення

Проєктування системи обробки інформації у кіберфізичній системі білінгу та управління ресурсами для віртуальних серверів

Програмна реалізація кіберфізичної системи білінгу та управління ресурсами для віртуальних серверів

5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень)

Архітектура ПЗ проєкту

Архітектура ПЗ для кіберфізичної системи

Апаратне забезпечення проєкту

6. Консультанти розділів дипломного проекту (роботи) .

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Тетяна КИСІЛЬ, доцент кафедри КІІС		
Антиплагіат	Андрій НІЧЕПОРУК, доцент кафедри КІІС		

7. Дата видачі завдання «_10_»_____01_____2025 р.

КАЛЕНДАРНИЙ ПЛАН

№з/п	Назва етапів (розділів) дипломного проекту (роботи)	Термін виконання етапів проекту (роботи)	Примітка
1	Вибір напряму дослідження та узгодження тематики кваліфікаційної роботи з керівником	10.01.2025	виконано
2	Ознайомлення з предметною областю; формулювання мети та задач дослідження; визначення об'єкта та предмета дослідження	01.02.2025	виконано
3	Робота над розділом 1 – дослідження предметної області та постановка задачі	01.03.2025	виконано
4	Робота над розділом 2 – вибір елементів для побудови системи білінгу та управління ресурсами для віртуальних серверів	01.04.2025	виконано
5	Робота над розділом 3 – побудови системи білінгу та управління ресурсами для віртуальних серверів	29.04.2025	виконано
6	Оформлення пояснювальної записки згідно вимог	25.05.2025	виконано
7	Попередній захист ВКР	26.05.2025	виконано
8	Захист ВКР на засіданні ЕК	Червень 2025 року	

Студент _____ Олексій МОТУРНЯК
Підпис Ініціали, прізвище

Керівник роботи _____ Олексій ІВАНОВ
Підпис Ініціали, прізвище

№ рядка	формат	Позначення	Найменування	Кіл. листів	№ екз	Примітка
			Текстові документи			
1		КвРКІ 022027.22.02.24 Е8	Пояснювальна записка	60		
			Графічні матеріали			
2		КвРКІ 022027.22.02.24 Е8	Структурна схема	1		
			ПЗ проєкту			
3		КвРКІ 022027.22.02.24 Е8	Data flow diagram	1		
			програмного забезпечення			
4		КвРКІ 022027.22.02.24 Е8	Блок схема Креслення	1		

Зм	Арк	№ докум	Підпис	Дата	КвРКІ 022027.22.02.24 ВП			
Розробив		Мотурняк			Відомість проекту	Літера	Аркуш	Аркушів
Перевір.		Іванов		06.06.25		У	1	1
Н. контр.		Кисіль		06.06.25		ХНУ, КІ2с-22-2		
Затв.		Павлова		06.06.25				

АНОТАЦІЯ

Тема кваліфікаційної роботи: «Кіберфізична система білінгу та управління ресурсами для віртуальних серверів».

Автор роботи: Олексій МОТУРНЯК.

Керівник роботи: Іванов Олексій Валентинович.

Пояснювальна записка: 60 с., 25 рис., 3 дод., 42 джерел.

Графічна частина: 3 креслення.

Метою дипломної роботи є визначення умов та особливостей застосування білінгової системи, а також оцінка механізмів обробки інформації у кіберфізичній системі білінгу та управління ресурсами для віртуальних серверів для забезпечення маштабованості та надійсності.

Об'єктом дослідження є кіберфізична система білінгу та управління ресурсами для віртуальних серверів.

Предметом дослідження є оцінка режимів застосування системи білінгу управління ресурсами для віртуальних серверів.

Під час проведення даного дослідження був використаний метод систематичного огляду літератури для вивчення і аналізу предметної області даного дослідження з текстових джерел інформації.

Підпис студента

30.05.2025

Дата

ЗМІСТ

ВСТУП 4

1 КІБЕРФІЗИЧНА СИСТЕМА БІЛІНГУ ТА УПРАВЛІННЯ РЕСУРСАМИ ДЛЯ ВІРТУАЛЬНИХ СЕРВЕРІВ ТА ПОСТАНОВКА ЗАДАЧІ ЩОДО ЇЇ УДОСКОНАЛЕННЯ 5

1.1 Аналіз предметної області і виявлення наявних проблем і завдан 5

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень 6

1.3 Основні функції білінгових систем 10

1.4 Архітектура програмного забезпечення білінгових систем 11

1.5 Технічні особливості сервісу 12

1.6 Системи оркестрації додактів 13

1.7 Реалізація UI/UX 15

1.8 Постановка задачі 17

1.9 Висновки 18

2 ВИБІР АПАРАТНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ СИСТЕМИ БІЛІНГУ ТА УПРАВЛІННЯ РЕСУРСАМИ ДЛЯ ВІРТУАЛЬНИХ СЕРВЕРІВ 19

2.1 Існуючі системи білінгу та управління ресурсами для віртуальний серверів 19

2.2 Open source проекти 20

2.3 Вибір протоколу передачі даних 27

2.4 Вибір апаратного забезпечення 32

2.5 Вибір програмного забезпечення 36

2.6 Вибір інструментів для створення програмного забезпечення 40

2.7 Висновки 44

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ КІБЕРФІЗИЧНОГО БІЛІНГУ ТА УПРАВЛІННЯ РЕСУРСАМИ ДЛЯ ВІРТУАЛЬНИХ СЕРВЕРІВ 44
3.1 Опис програмного та апаратного забезпечення кіберфізичної системи білінгу та управління ресурсами для віртуальних серверів 44
3.2 Створення БД для роботи системи 51
3.3 Налаштування системи Proxmox, для створення віртуальних машин . 52
3.4 Аналіз обмежень розробленої кіберфізичної системи 58
3.5. Висновки 59
ВИСНОВКИ 60
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ 61
ДОДАТОК А 64
ДОДАТОК Б 65
ДОДАТОК В 66
ДОДАТОК Г 67

ВСТУП

Останнім часом, із розвитком технологій, зростає попит на ефективні та масштабовані системи для потреб бізнесу. Окрім цього, одним із запитів бізнесу для програмного забезпечення, є виконання усіх потреб бізнесу. Одним із рішень, є використання програмного забезпечення із підтримкою модульної системи, що дозволяє розширювати функціонал, внаслідок, модулів.

Для реалізації системи, із присутнім масштабування системи, використовуються різні засоби, для прикладу Kubernetes. Дане рішення дозволяє розміщувати додатки запущені в Docker контейнера, а сама система, самостійно розміщує додаток, залежно від доступності ресурсів.

Бізнес потребує рішення типу, особистого кабінету, який дозволяє надавати клієнтам, доступ до послуг, отримувати оплату за надані послуги. Для реалізації системи, потрібно враховувати функціональність бізнесу. Для вирішення запиту, потрібно використовувати популярні рішення зберігання даних та передачі даних. Для зберігання даних використовують Postgres, та Redis. Використання даних засобів, дозволяє гнучко маніпулювати даними. Для передачі даних використовуються REST API та GRPC, що використовуються для різних цілей.

Отже, основна мета роботи є вивчення та практика в розробці програмного забезпечення, створення високо масштабованого рішення, з можливістю додавання нового функціоналу.

1 КІБЕРФІЗИЧНА СИСТЕМА БІЛІНГУ ТА УПРАВЛІННЯ РЕСУРСАМИ ДЛЯ ВІРТУАЛЬНИХ СЕРВЕРІВ ТА ПОСТАНОВКА ЗАДАЧІ ЩОДО ЇЇ УДОСКОНАЛЕННЯ

1.1 Аналіз предметної області і виявлення наявних проблем і завдань

Кіберфізична система (КС) – технічна концепція, що являє собою вбудування обрахункових ресурсів фізичних та технічних сутностей будь-якого типу.

У кіберфізичних системах матиматична система є розподілені по всій системі, яка є компонентом, та містить тісний зв’язок з її фізичними елементами.

Термін кіберфізичні системи почав використовуватись у рамках четвертої промислової революції.

Для бізнесу, часто виникає питання в особистому кабінеті, та шуканому завданні. Рішення яке шукається, зазвичай потребує, наявність достатнього функціоналу, задовольняння потреб бізнесу. З наявного функціоналу, який потребується, зазвичай мається на увазі, це можливість ведення списку користувачів, введення фінансової звітності, адміністрування користувачів, управління віртуальними послугами, продаж, редагування, видалення. Також, додатково, очікується мати наявність здійснювати різні варіанти періоду, та гнучкості оплати віртуальних послуг. Це мінімальний список функціоналу, який потребується, для бізнесу.

Зазвичай, часто компанії реалізовую своє рішення, яке буде використовуватися в межах компанії. Це варіант розробки програмного засобу дає можливість гнучко та за потреби змінювати кардинально функціонал керуючись потребами компанії. Але цей спосіб ведення та підтримки програмного забезпечення має свої недоліки. Одним із недоліків – це наявність команди розробників, які повинні вести розробку даного засобу. З фінансової сторони це має недолік, який полягає у великих витратах, які має постійний характер. З плюсів - це те, що більшість наявних програмних засобів, можуть мати закриту політику коду,

що не дозволить детально розуміти код, який виконується в роботі, тому, розроблене програмне рішення, дає повний доступ для компанії.

Також, варто внести момент, що з недоліків, є те, що для гнучкого та складного рішення потрібні велика команда розробників, яка складається із DevOps Enginner, Backend Develop, UX/UI Designer та Frontend Developer. Кожен із цих спеціалістів займають важливу ланку розробки програмного забезпечення. DevOps Enginner виконує задачі по налаштування автоматизації, моніторингу, та доставки коду програмного забезпечення на сервер. Backend Develop виконує задачу розробки серверної логіки. Це складний процес, через те, що задач бізнесу можуть бути різноманітними, а рішень безліч. Також варто пам’ятати, що завдання розробника, окрім вирішення бізнес задачі, потребує в ідеальній побудові архітектури додатка, який буде легко підтримувати в майбутньому.

Frontend Develop вирішує потреба побудови користувацького інтерфейсу, який створить UX/UI Designer. Інтерфейс потребу можливості адаптації залежно від екрану девайса, його розмірів. Також, він надає зручний функціонал, для керування з backend частиною додатку. Встановлення стилі, теж є обов’язок цього спеціаліста.

Designer – спеціаліст, як було сказано раніше, створює макет дизайну, та продумує зручний функціонал, який буде зручний для кінцевого користувача.

1.2 Порівняльний аналіз переваг та недоліків існуючих рішень

Із наявних рішень, які є доступні на ринку, є як і безкоштовні, так і платні рішення. З безкоштовних також можна згадати про те, що це може бути рішення Open Source, так і закритого типу. З платних рішень, доступ до коду, закритий.

В наш час, вирішують різні варіанти даної потреби.

Для прикладу, використовують WordPress із додатковими плагінами. З переваг, це готове рішення, але потребує мінімальних налаштувань. З недоліків, це обмеження розширення функціоналу.

Freeside – це рішення, яке використовують для бізнесу, реалізоване на мові Perl. З переваг, це безкоштовне рішення, яке має відкритий код та має безліч функціоналу. Має ліцензію GPL, що може не бути зручним, для комерційного використання. Дане рішення допомагає, використовувати його в таких продуктах, які мають тип ISP, VoIP та хостингового типу. Надає доступ до API користувача, дозволяє реалізовувати автоматизацію, для зменшення рутинних задач. Також, варто зазначити, що дане програмне забезпечення встановлюється на Linux. Із його повних функціональних можливостей, є те, що дане рішення дозволяє наступне:

– можливість створення, редагування та видалення фінансових рахунків;
– підтримка тарифних планів;
– система системи звернень;
– автоматичне налаштування платежів;
– наявність модулів, для інтеграції із платіжними шлюзами.

До переваг даного програмного забезпечення можна віднести такі складові які дозволяють вирішувати проблеми бізнесу. Варто врахувати що наявність системи автоматизації надають велику кількість створення автоматичних сценаріїв адміністрування. Окрім цього враховуючи попит на споживання та розвиток технологій варто зазначити, що наявність відкрити рішення з відкритим кодом надає швидший розвиток технологій, та популярність рішення. Ці переваги покращують досвід користування програмним забезпеченням завдяки збільшення програмного рішення.

Враховуючи переваги, які були отримані із недоліків варто навести наступне, а саме складність в розгортані та підтримці програмного забезпечення, через особливість специфіки програмного забезпечення, та присутність застарілого UI/UX дизайн, який впливає на загальний досвід користувачання.

Kill Bill – B2B рішення, що реалізоване на мові Java. Його призначення, та сфера використання є SaaS послуги, які є популярними в наш час. Окрім цього є можливість використовувати його в E-Commerce та фінансових відділах. Переваги

даного рішення, є використання модулів, підтримка плагінів, які дозволяють розширяти функціонал, та гнучка інтеграція із різними сервісами.

Дане рішення має наступний стек технологів, такі як Java, Mysql або PostgresSQL[11] та дає можливість використовувати REST API.[4]

Через те, що для розміщення платформи використовується Docker, це дозволяє розвернути програмне забезпечення на будь-якому програмному забезпечені.

Також, враховуючи що дане програмне забезпечення містить список програмного забезпечення, яке поєднане та використовується як одне ціле, то як кожен програмне забезпечення є відповідні переваги, які відповідають за гнучке управління підписок за отримані послуги. Дане програмне забезпечення маючи можливість здійснювати керування активними підписками, створювати нові акційні пропозиції, що дозволяють вирішувати базові потреби бізнесу. Окрім цього, присутня складна та глибока система ціно утворення, що дозволяє створювати тарифи будь-якої складності, різного роду акцій та інших подібних пропозицій. Також маючи можливість виставляти рахунки різного роду складності із підтримкою систем оподаткування дозволяє надати кращий досвід користувача кінцевому користувачі у порівняні іншим систем, які не мають даної системи. Враховуючи арсенал переваг, стабільна робота системи під час високого навантаження дозволяє масштабувати систему в горизонтально, завдяки чому стабільність системи зростає. Враховуючи отримані можливості, наявність хорошої та детальної документації впливає на можливість легко розібратися з нуля із системою, та здійснити досконалі налаштування. Але враховуючи велику кількість переваг, через різноманітну кількість налаштувань, зростає потреба в детальному вивчені системи, що впливає на загальну складність підтримки та налаштування системи.

Invoice Ninja – рішення, яке використовується зазвичай для виставлення рахунків та для малого бізнесу. Це рішення включає вихідний код, підтримує

різноманітну кількість валют, налаштування шлюзів оплати. Це програмне забезпечення реалізоване на PHP.

Також, варто зазначити, що дане рішення можна запустити як на Linux, так і на Docker, що дозволяє не бути прив'язаний до платформи, так і на SaaS продуктах.

Функціонал, який надається, в даній системі, дозволяє створювати рахунку, за допомогою генерації. Отримані згенеровані рахунки можливо гнучко надавати кінцевому користувачу, завдяки чому загальний досвід керування системою зростає. Також варто підмітити, що виставлення рахунків для оплати, потребує можливість здійснювати фінансові операції. Ця потреба зумовлена потребами бізнесу, що означає, що можливість налаштування оплати через Stripe та PayPal є обов'язковою можливість, без якої в наш час не можливо обійтися.

Також з переваг можна віднести те, що дане програмне забезпечення має зручний та простий інтерфейс.

Це програмне рішення найкраще підійде тим, чий бізнес тільки розвивається.

BTCPay Server – це нове рішення, яке виникне не так давно. Основна його перевага, це підтримка для роботи із крипто оплатою. Має наступні переваги, такі як підтримка оплати Bitcoin. Рішення має відкрите програмне забезпечення. Реалізоване сучасною мовою C#.

З платних рішень, які використовуються, це Chargify, яке використовується для SaaS продуктів, та B2B рішень. Мають такі переваги, як можливість підписок, перегляд метриків та інтеграцій. Це програмне забезпечення розроблене в США.

Zuora – ще одне рішення, яке має комерційний характер. Використовується для підтримки підписки за послугу. Із переваг, це можливість налаштування складної системи ціни утворення. Окрім цього має можливість здійснювати інтеграцію ERP та CRM інтеграцій. Це поєднання задовільняє базові потреби компанії, що дозволяє завдяки цьому розвивати бізнес гнучко.

Recurly – білінг підписки, який використовується в середині компанії. Має тип оплати - підписка. Має інструменти оптимізації доходу, аналітики.

Stripe Billing – це внутрішнє рішення компанії, яка надає метод оплати Stripe. Дозволяє переглядати отримані кошти за здійснені платежі. Зручне рішення, для тих, кому потрібно налаштувати перші платежі в два кліки. Із переваг даного рішення, яке надається, це доступ через API, до зручно для інтеграції із цією системою. Також варто зазначити, що є підтримка глобальних методів оплати, що зручно в наш час.

Paddle – це рішення для управлення SaaS послугами. Дане рішення продає послуги SaaS платформи. Особливість його в тому, що має повний набір функціоналу білінгової системи, має можливість рахувати податки.

1.3 Основні функції білінгових систем

Основні функції білінгової системи є можливість реєстрації користувачів, ведення звітності, надання можливості здійснення покупки віртуальної послуги, здійснення оплати та взаємодія із замовленими послугами.

Базовий функціонал, який очікується в програмному забезпечення, є можливість реєстрації користувача. Відповідно його авторизація також повинна бути присутня. Здійснення покупки послуг, є теж базовим функціоналом. Через те, що послуга орендується, тип період оплати може бути наступні:

– секундна тарифікація;
– місячна тарифікація;
– річна тарифікація.

Окрім цього, має бути реалізований різний варіант оплати послуги. Поповнення балансу, різними платіжними шлюзами.

Вид послуги, залежить від вибраної інтеграції з послугою. Через те, що дана білінгова система буде взаємодіяти для продажів віртуальних серверів. Через те, що система буде реалізована з інтеграцією з Proxmox, інтеграція буде відбувається з API. Даний варіант інтеграції можна використовувати для реалізації покупки будь-якої із послуг.

1.4 Архітектура програмного забезпечення білінгових систем

Залежно від складності реалізації, та побудови архітектури додатка, можна винести два види архітектури, такі монолітна та мікросервісна, яка в першому випадку, є одним суцільним додатком а в другому – розбита на логічні компоненти.

Ці два варіанти мають свої переваги та недоліки. Залежно від бізнес задачі постає вибір, що краще використовувати.

Вибір монолітної архітектури вибирається тоді, коли потрібно побудувати просту систему, без складної архітектури, де вибір між простотою та очікуваним навантаженням стоїть на вибір простої архітектури та де навантаження на програмне забезпечення буде невеликим. В цьому випадку, зі зростанням попиту на послугу, з часом, зростатиме навантаження, що через деякий час буде досягнуло ліміту по продуктивності. В цьому випадку, розв'язання проблеми є перехід на мікросервісну архітектуру, але це складне завдання, якщо вже є готова кодова база.

Також варто звернути увагу на те, що моноліт легше розробляти через те, що уся кодова база знаходиться в одному місці. В цьому випадку, легше розв'язувати базові задачі в розробці.

У випадку із мікросервісній архітектурі, задача розробки та підтримка зростає набагато сильніше, через те, що потрібно підтримувати не одне програмне забезпечення, а велику кількість компонентів, які між собою спілкуються по узагальненому інтерфейсу. В основному, для спілкування мікросервісів використовується різні варіанти спілкування. Для прикладу, це може бути REST API, що є HTTP протоколом. Спілкування відбувається синхронно, тобто один сервіс подає запит до іншого сервісу, і очікує відповіді. В цьому випадку недоліком цього підходу є те, що у випадку, недоступності одного із сервісів, може бути втрата даних, що є великим ризиком, якщо дані, які передаються між сервісом, є важливими. Також, варто визначити, що використовується також GRPC протокол, що є віддалене виконання функцій[7]. Цей підхід кращий за REST API, шляхом

того, що об’єм даних, що передається між сервісами, менший та швидший.[5] Окрім цього для програміста, це буде виглядати як звичайний виклик функцій. Недолік цього підходу залишається таким же, через те, що у разі не доступності одного із сервісів, є недолік втрати даних.

Для розв'язання даної проблеми можна використовувати передачу даних через брокер повідомлень.[10] Це рішення зручне тим, що один сервіс надсилає дані брокеру повідомлень. Брокер повідомлень зберігає ці дані весь час, поки інший сервіс, який очікує ці дані, не забере їх. Також, даний підхід дозволяє реалізовувати систему транзакцій, що дозволяє бути впевненим, що дані успішно надіслані, та оброблені. Через те, що в цьому випадку, використовується брокер повідомлень, обробка даних відбувається в асинхронному режимі.

В більших випадків, використання цих 3-х варіантів передачі даних, залежить від поставленої задачі. Використання REST API, зазвичай використовується для надання доступу до сервісу, браузері, що виступає в ролі клієнта.[5] В цьому випадку, через те, що браузер підтримує REST API, його використовують, як основний варіант спілкування браузера із сервісом.[5]

GRPC протокол, використовується для спілкування між сервісами, за рахунок його простоті, швидкості роботи.[7]

Блокер повідомлень, використовується також для спілкування між сервіс, але у тому випадку, де надійність даних важлива. [4,10]

1.5 Технічні особливості сервісу

Оскільки кожен сервіс унікальний, він може бути реалізований будь-якою мовою програмування. Це з один із переваг мікросервісів. Але в цьому випадку, краще використовувати Go, через те, що ця мова програмування найкраще підходить для побудови сервісів.

Для спілкування між сервісами, як було сказано раніше буде використовувати HTTP протокол, а саме REST API, який буде надавати публічний інтерфейс, для кінцевого користувача.[5]

Для спілкування між сервісами, як було сказано раніше, використовується GRPC протокол, або брокер повідомлень. В цьому випадку, використання залежати, від конкретної задачі, та потреби[7].

Зберігання даних кожного із сервісів, буде використовуватися Postgress, через те, що це популярне рішення, для зберігання даних.[11]

Також, окрім цього, варто зазначити, що під час роботи розробки, можна використовувати кешування запитів, для зменшення навантаження системи. Цей варіант дозволяє використовувати для зменшення навантаження на систему.

1.6 Системи оркестрації додатків

Як було сказано раніше, залежно від архітектури додатку, можна поділити на кілька видів, такі як мікросервісна та монолітна архітектура. Створення додатки на основі цього типу архітектури, під час розробки можна використовувати різний підхід розміщення. Монолітна архітектура може безпроблемно бути запущена на одному Linux сервері. Але через зростання навантаження важко буде розв'язувати проблему збільшення попиту на користування сервісом. Одним із потенційним варіантом розв'язання проблеми є використання збільшення системних характеристик додатка.

У випадку із мікросервісної архітектури є використання звичайного Linux сервера, так і системи оркестрації додатків.

На даний час популярним рішенням є використання Kubernets так і Docker Swarm.[1,2,3] Ці два варіанти чудово підходять для розміщення додатків, але в основному використовується Kubernetes. Натомість через популярність даного інструменту, було розроблено спільнотою K0s, так і K3s, який має свої переваги та недоліки. З переваг K0s можна віднести наступне [8,9]:

- можливість запуску системи за допомогою одного бінарного файлу;
- присутня підтримка роботи в режимі однієї ноди так і в повноцінному кластері;
- менше споживання ресурсів за рахунок відсутності Control Panel та інших додаткових компонентів;
- присутність великого вибір архітектур, що дозволяє запускати на будь чому із доступному, хоч на arm так і на amd архітектурі та інші;
- повна підтримка API k8s, що дозволяє легко замінити k8s на дану подібну систему;
- можливість запуску та встановлення системи k0s, за допомогою файлу конфігурації.

З недоліків, варто віднести те, що через особливість системи, потрібно додатково розбиратися як працює система.

Також, варто звернути увагу на те, що K3s, має із переваг, такі як споживання меншої кількості ресурсів операційної системи, та можливість встановлення за допомогою однієї команди, що спрощує встановлення системи.

З недоліків, варто звернути увагу на те для зберігання даних використовується SQLite, що є не дуже масштабованим рішенням.

Аналогом Kubernetes можна навести це Docker Swarm, що є внутрішнім функціоналом Docker Engine.[2,3] Через те, що це внутрішній функціонал, він потребує лише встановлення Docker на систему, де планується розміщення додатка. Це є перевагою цієї системи, через те, що не потрібно глибоко налаштовувати систему. Для своєї роботи Docker Swarm потребує налаштування ноди в ролі Manager так і використання Worker node. Прикладом побудови архітектури можна побачити нижче.

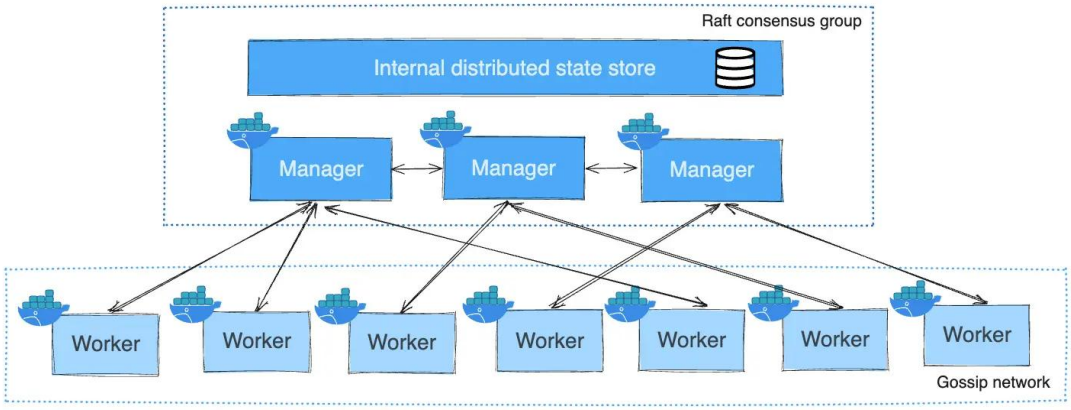


Рисунок 2.1 – Візуалізація cluster Docker Swarm[3]

Але через те, що цей інструмент був розроблений раніше, на зміну його прийшов Kubernetes, що є продовженням функціоналу, із великою кількістю налаштувань. Тому, враховуючи увесь попередній досвід, структура даного кластеру виглядає наступним чином[1]:

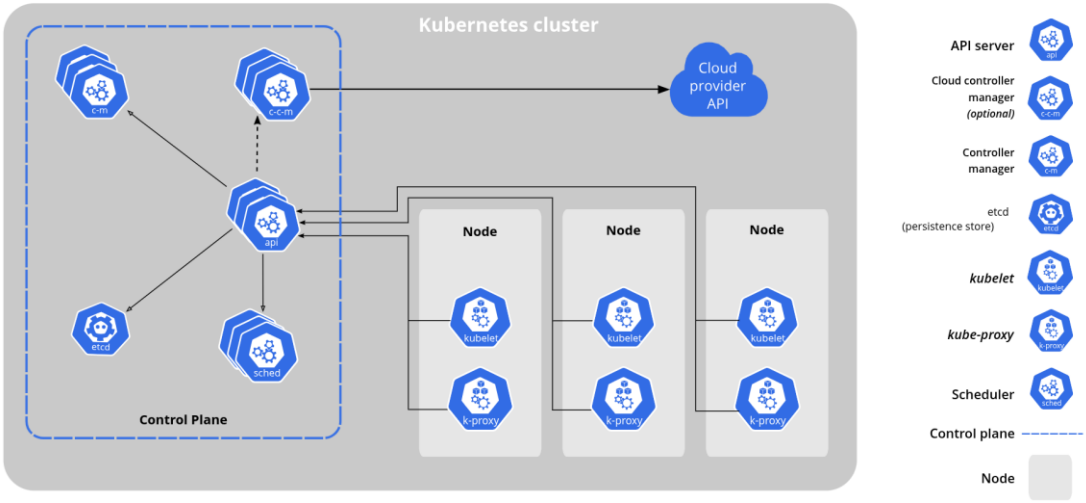


Рисунок 2.2 – Візуалізація cluster Kubernetes[2]

1.7 Реалізація UI/UX

В залежності, від вибору програмного забезпечення, під час реалізації backend частини, виникає питання в потребі взаємодії користувача із backend частиною. [15]

Під час розробки програмного забезпечення в більшості випадків використовують такі під назвою Postman.[13]

Це програмне забезпечення дає можливість здійснювати різного роду запитів до backend частини. У випадку, якщо програмне забезпечення використовує REST API, то Postman дає можливість використовувати його функціонал, для здійснення запитів. Сам Postman окрім REST API, надає можливість використовувати наступні типи: [15]

– HTTP;
– GraphQL;
– AI;
– MCP;
– gRPC;
– WebSocket;
– Socket.IO;
– MQTT;

Ці запити виконують різну задачу. Як було сказано раніше HTTP запит, є звичайним RESTAPI. [4,15]

GraphQL – це метод, який використовується для здійснення запиту, в основному використовується для соціальних мереж, де потрібно мати змогу отримувати лише частину даних.

AI – це зручний інтерфейс спілкування різними моделями штучного інтелекту

MCP – це новий відкритий протокол, який стандартизує спілкування різних моделей між собою[14]

gRPC – протокол відділеного виконання функцій[7]

WebSocket – протокол, який дозволяє в реальному часі утримувати підключення із сервером, та обмінюватися повідомленнями. Рекомендується використовувати для реалізації живого чату, або передачі даних в реальному часі.

MQTT – протокол спілкування, черги повідомлень.

Отже, залежно від поставленої задачі у випадку, спілкування білінгу, буде використовуватися HTTP протокол. REST API.

Для реалізації інтерфейсу використовуватиметься React, для побудови інтерфейси.[23] Використання React, є одним із сучасних рішень, через те, що даний спосіб не потребує реального піднятого backend частини, для роботи системи. Потрібен лише статичний web сервер, який буде надавати браузеру статичний контент.

Для зберігання статики, можна використовувати CDN, який дозволяє кешувати, та маніпулювати статичними файлами. Із популярних рішень є Cloudflare, AWS Cloudfront, яке є рішення CDN від Amazon.[25, 26]

Окрім цього також можна використовувати Nginx, для проксування статичних файлів.[24]

1.8 Постановка задачі

Враховуючи сучасні запити бізнесу, та попит споживання на новітні системи, основне завдання завдання повинно складатися із дослідження принципу роботи білінгових систем, дослідження роботи мікросервісних архітектур, розуміння принципу побудови архітектури додатків на REST API, аналіз спілкування додатків в мікросервісних архітектур, провеведення технічої аналізу сфери роботи, визначення присутніх та наявних проблем в подібних рішень, та спроба пошуку їх вирішення, проведення відповідного аналізу отриманого результату, визначення його переваг та недоліків, потрібно сформувати загальну ситуацію рішення, виконати аналіз, для розуміння про необхідність створення системи, та сформувати мету наступного етапу аналізу та дослідження. Окрім цього, варто здійснити експерту оцінку даного поставленого завдання та технічну оцінку рівня виконання поставленої задачі.

На основі даної інформації створити та розробити систему білінгу для управління ресурсами для віртуальних серверів, після чого зробити висновок виконаної роботи.

1.9 Висновки

У межах розділу 1 проведено аналіз структурних і функціональних особливостей кіберфізичної система білінгу та управління ресурсами для віртуальних серверів, здійснено аналіз програмно забезпечення обробки інформації в система білінгу та управління ресурсами для віртуальних серверів, здійснено постановку задачі оцінки механізмів обробки інформації у програмно-технічному засобі. Крім цього, проведено аналіз наявних варіантів білінгових систем, описано їхні переваги та недолік та наведено їх нюанси. Було визначено вимоги майбутньої системи, основний наявний фунціонал подібних систем, визначені можливі обмеження, які можливі під час розробки подібних систем, визначені межі та можливі ризики. Було визначені основні компоненти системи, які буде створено, що дозволить гнучко створити відповідно масштабовану конструкції програмного середовища.

2 ВИБІР АПАРАТНОГО ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ СТВОРЕННЯ СИСТЕМИ БІЛІНГУ ТА УПРАВЛІННЯ РЕСУРСАМИ ДЛЯ ВІРТУАЛЬНИХ СЕРВЕРІВ

2.1 Існуючі системи білінгу та управління ресурсами для віртуальних серверів

Враховуючи, що є велика кількість подібних систем різного рівня для керування та реалізації білінгових систем, основною задачею є аналіз найбільших та найпопулярніших систем. Для потреби бізнесу, що відповідають сучасним потребам цього бізнесу мають бути гнучкими, зручними для роботи, масштабованими та надійними.

Системи можна поділити на дві категорії:

– програмне забезпечення із закритою кодовою базою, протоколами передачі даних, своїми даними передачі даних;
– програмне забезпечення відкритого типу, що має в назві open source, що є найбільш популярне рішення серед технічного світу.

Даний тип програмного забезпечення може працювати відкрито, що означає, що кожен може дізнатися, як працює система. Завдяки цьому, є можливість вносити зміни в загальну кодову базу програмного забезпечення, що зазвичай зберігається у відкритих репозиторіях GitHub або GitLab. Даний тип програмного забезпечення дає можливість вільно та безкоштовно використовуватися без додаткової плати, що надає велику популярність серед IT спільноти.

В цьому розділі буде розібрано найбільш популярні рішення даного open source рішень, проектів, які можуть використовуватися для бізнес цілей із поставлених задач.

Завдяки відкритості програмного забезпечення є великий прорив та розвиток програмного забезпечення має великий вплив та використання, що завдяки чому

рішення є повністю безкоштовних. Тому, завдяки чому IT спільнота велика завдяки цим перевагам.

2.2 Open source проекти

Kill bill є великим сервісом білінгу, який дозволяє запустити білінгову систему для бізнесу. Цей сервіс має велику достатню базу користувачів, є популярним рішенням для бізнесу. Дане програмне забезпечення зберігається на Github, та має 4.9 тисяч зірочок на репозиторію. Дане програмне забезпечення є монолітним, та створене такою мовою як Java. Дана мова програмування є доволі популярною для створення програним веб додатків. Для зберігання даних, дає можливість зберігати дані Mysql та PostgreSQL[38].

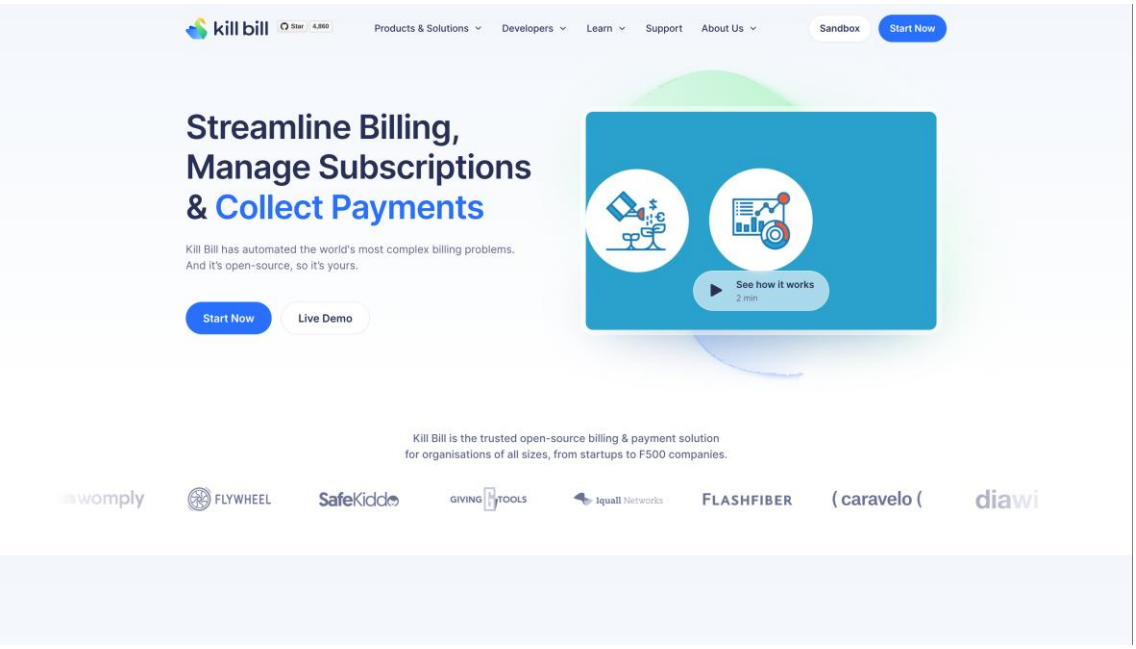


Рисунок 2.1 - Платформа Kill Bill для бізнесу[38]

Дане програмне забезпечення надає великий спектр продуктів такі як Subscription Billing, Payment Platform, Kaui та рішення для бізнесу типу B2C та B2B.

Раніше рішення Kill Bill може задовільнити потреби у виставлені фінансових рахунків для реалізації оплати для будь-якого бізнес рішення. Дає можливість

масштабувати інфраструктуру бізнесу для виставлення рахунків, оплачених платежів для розвитку бізнесу. Надає можливість до аналітики та фінансових звітів у режимі одночасного та реального часу. Окрім цього надає можливість створення свого бізнесу, що дозволяє уникати залежності від інших рішень.

Також варто зазначити, що надані розширення надають можливість розширювати систему для роботи із популярними рішенями, такими як Adyen, Stripe, Braintree та PayPal. Враховуючи дані рішення це доволі популярні рішення. Враховуючи надані та зручні інструменти, є класні рішення документації, що дозволяє вдало налаштувати систему.

Враховуючи наявні рішення для бізнесу, варто розглянути усі рішення. Для початку варто дізнатися про Subscription Managment. Дане рішення дозволяє створювати підписки, оновлювати, призупиняти, скасовувати та оновлювати їх. Також дає можливість управляти розширеннями для різного роду сценаріїв оплати. Можливість автоматичного переходу з одного роду оплати на іншу. Завдяки даного програмного забезпечення, є можливість передбачити наступні особливості, а саме, те що фази можуть мати різну ціну та умови оплати, прайс-листи з правилами зміни плану, щоб надавати спеціальні ціни певним клієнтам та створення правил каталогу для розуміння поведінки системи під час створення, скасування або зміни плану.

Враховуючи дані можливості є додаткове рішення, є наступним. Це можливість використання глобальної системи платежів. Kill Bill дає можливість взаємодіяти з постачальниками платіжних послуг, через використання API.

Завдяки цьому винести наступні переваги, а саме, те що присутня наявна підтримка практично всіх платіжних потоків, що дозволяє масштабувати бізнес, наявна підтримка майже будь-якого способу оплати, що дозволяє приймати оплату будь-яким чином, визначення дій для невдалих та прострочених платежів, що дозволяє зменшити витрати для бізнесу, можливість налаштування додаткового захисту під час оплати, що зменшує ризики бізнесу та можливі витрати під час вирішення питань із коштами, використання та наявність плагінів для підключення

до будь-якого платіжного шлюзу, що дозволяє інтегруватися з будь-якою платіжною системою.

Окрім цього варто зазначити, що є внутрішній адмін інтерфейс, що надає можливість керувати системою. Дана система надає наступний функціонал, що включає створення та керування користувачами, налаштування відповідних дозволів, що дозволяє гнучко керувати системою, редагувати плани підписок та цін в режимі реального часу без потреби в розробці програмного забезпечення, що надає можливість керування системою, без наявності команд розробників, розширена функція пошуку для будь-якого елементу в даній системі, надає можливість аналізувати будь-яку системи, на наявність чи відсутність потрібних даних, наявність створення категорій, сортування даних, що при наявності великої кількості даних дає легше проаналізувати системи на ті чи інші дані, встановлення додаткових плагінів, що дозволяє гнучко розширяти системи, та доповнювати потрібний функціонал та відслідковувати та відстеження історії змін, для детального аналізу системи під час різноманітних змін.

Отже, з наданого функціоналу варто відокремити також і наявність недоліків, що може дозволяє покращувати наявну системи. До цього варто віднести можливість гнучко використовувати модульну архітектуру, наявність та підтримка передплат, метрик, податків та платіжних систем, взаємодіяти із системою через REST API, та наявність активної спільноти та наявність хорошої та повної документації дозволить гнучко керувати системою.

Freeside – це рішення є повністю відкрите, надає можливість використання CRM, відслідковування та реєстрація проблем, моніторинг різноманітних проблем, моніторинг роботи мережі та забезпечення наявних ресурсів[37]. Завдяки цьому, використання цієї системи буде гнучким для бізнесу.

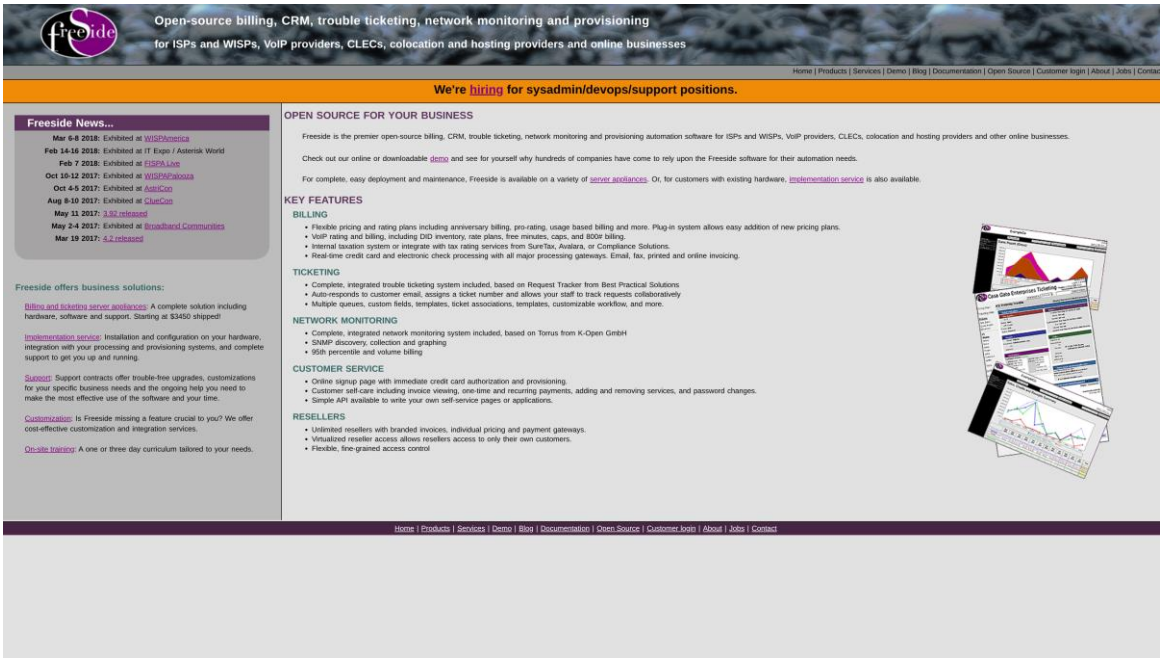


Рисунок 2.2 - Freeside білінгова система [37]

Дане програмне рішення використовується для виставлення рахунків. Із особливостей варто зазначити що у є наступні переваги у роботі із рахунками, що дозволяє гнучке створення цін та тарифних планів, що дає можливість гнучко здійснювати різницю оплати, здійснювати оплату на основі використання тощо, використовувати систему плагінів, що дозволяє додавати нові тарифні плани, створювати системи рейтингів VoIP, включаючи інвентаризацію DID, реалізації тарифних планів, є можливість використання безкоштовних хвилин, використовувати на повну наявність внутрішньої системи оподаткування та інтеграцію з сервісами податкових рейтингів та виставляти рахунки із використанням електронної пошти, системою факсів або онлайн рахунків.

Також варто донести думку про те, що є можливість використання підтримки даної системи. Одна із них плата за яку становить 195$ містить базове технічне обслуговування, оновлення програмного забезпечення, спілкування із компанією за рахунок електронної адреси. Окрім цього є можливість використання знижки на додаткові роботи в розмірі 10%.

У випадку стандартної підтримки вартість якої є 395$, до попереднього плану також входить наступні бонуси такі як, наявність Підтримки в режимі телефонної

розмови в задані години роботи, доступність підтримки до розширених функцій, такі як VoIP, CDR, можливість налаштування системи 2.5 на місяць та наявність знижки для додаткових систем становить 15%

У випадку використання техпідтримки у 800$ надається переваги, такі як – цілодобова підтримка продукту, наявність приватного списку розсилки для великих постачальників, можливість налаштувань системи зростає до 8 годин на місяць та доступність знижки на налаштування додаткових систем становить 20%

Також доступні інші тарифні плани, що надають можливість мати особистого персонального адміністратора, впливати на розвиток системи та прямий доступ до розробників та співробітників системи.

Дана система реалізована на мові Perl, дані зберігаються в PostgreSQL.

Переваги даної системи є наступні, а сапе підтримка інтернет послуг та можливість інтеграції з різними платіжними шлюзами

Система Jbilling надає інструменти виставлення рахунків, зменшувати час на виведення продукту на ринок та покращення досвіду взаємодії із клієнтами. Дана система надає можливість зменшувати навантаження на систему та процес загалом. Дана система дозволяє організаціям та компаніям легше впроваджувати інноваційні системи, швидше вводити на ринок нові системи, що дозволяє оптимізувати виставлення рахунків завдяки автоматизованій системі визначення кількох позицій при цьому легко розгортати у застарілому середовищі[36].

Наявність даної кількості функціоналу, що надається системою jBilling, надає можливість задовільняти протреби бізнесу, створення унікальних задачі, отримання швидкого доходу, задовільняти потреби бізнесу, в короткі часові рамки, надавати ідеальні послуги, як кінцевому користувачу так і наявному бізнесу, який потребує даних послуг.

Відповідно, дана система має свої переваги та недоліки.

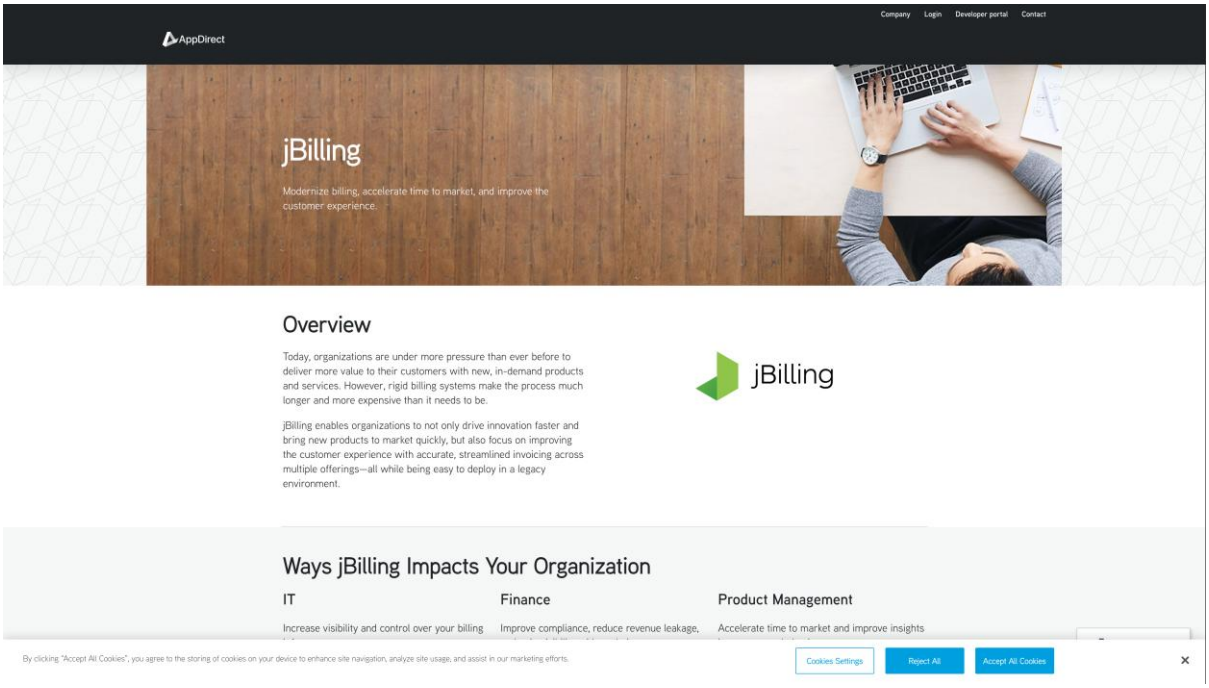


Рисунок 2.3 – Білінгова система JBilling [36]

Завдяки використання даної системи можна прискорити прозорість та контроль над системою платіжної інфраструктури. Дає можливість більш прозоро перевіряти дотримання вимог, зменшувати витік доходів та отримувати прозорість за допомогою прозорої аналітики. Це дозволяє прискорювати вихід на ринок та покращувати аналіз поведінки клієнтів.

Дана система реалізована на монолітній основі, із використанням мови Java. Має можливість використовувати різного роду модулі, що дозволяє розширювати система. Окрім цього є можливість інтегрування до інших систем за рахунок використання протоколу REST API.

Дане програмне забезпечення використовує PostgreSQL для збереження даних, що є одним із популярних рішень на цю мить часу.

Технологія білінгової системи OpenCell надає більший спектр можливостей. Варто зазначити, що використання таких технологій, як Java, дає можливість використання цієї системи на різних платформах. Окрім цього можна сказати, що дана система має такі переваги, як можливість використання тарифів, які нараховують використання лише за використання послуг, дають можливість

використання налаштовувати системи білінгу та створювати складні системи передплат.

Окрім цього варто зазначити, що дана система має зручний UX, що дозволяє розділити користувача в залежності від ролей використання.

Можливість використання різного роду багато рівнової системи, та можливості введення клієнтів в експлуатацію для різного роду співпраці. Також варто зазначити, що попередні налаштовані шаблони, надають можливість налаштовувати системи під майбутні задачі, що спрощує майбутнє використання. Дає можливість легко налаштовувати та керувати даними в системі[35].

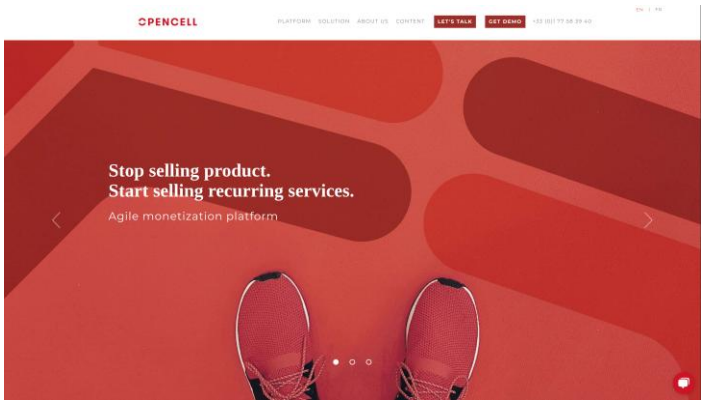


Рисунок 2.4 – Білінгова система OpenCell [35]

Враховуючи постійний та швидкий темп взаємодії, дана система надає можливість запускати складні проекти менше ніж за 3 місяця. В наш час це дуже ціниться, так як це допоможе зменшити потреби та зменшити витрати компаній.

Відкритість вихідного коду, надає можливість оцінити роботу системи, зрозуміти як вона працює в середині, і які розуміння ризиків системи.

Дане рішення має можливість використання у системах SaaS, що полегшує налаштування.

Дане рішення дає можливість запуску та використання для таких галузей:

– Зв’язок.
– Мобільний технології.
– B2B сервіси.

– Система енерії.

– Рентабельність систем.

– Система фінансів.

Дана система налає можливість використання різного роду модулів, такі як використання платіжних систем. Враховуючи потреби бізнесу, використання гнучких систем та налаштованих рішень в бізнесі дуже важливі.

Модуль каталогу використовується для бізнесу, який генерує постійний дохід, який надає можливість впроваджувати складні моделі ціноутворення

Модуль замовлень дає можливість для бізнесу автоматично активовувати системи, досягати значних результатів у витратах.

Модуль звітності надає можливість зручно та легко отримувати звіти по результати, показників орієнтованих на клієнтів

Використання зручних модулів, надає можливість використання будь-якого рівня системи.

2.3 Вибір протоколу передачі даних

Залежно від поставленої задачі є вибір правильно протоколу обміну повідомлень. Залежно від системи, які існують два види, можна винести дві особливості. У випадку монолітної структури, де увесь код знаходить в межах однієї кодової бази, спілкування відбувається спрощено, не потребує в додаткових рішень. Натомість використання систем, які побудовані на основі мікросервісної архітектури, можна сказати, що залежно від задачі, можуть використовуватися різні способи.

Найперше рішення, яке використовувалося, це використання HTTP протоколу, який дозволяє спілкуватися синхронно, та по простому каналу зв’язку. В системах, це може бути як REST API, який дозволяє взаємодіяти мережею, та обмінюватися, отримуватися та маніпулювати даними різним способом. Для прикладу, щоб отримати дані, у REST API, використовується такими методи як

GET. Цей метод, потребує лише вказати URL ресурсу, який очікується отримати. В цьому випадку, не потрібно передавати додаткові дані.

У випадку, якщо потребує додавання нових елементів у систему, або зміну даних, використовуються методи такі як POST, PUT та PATCH. Дані методи використовують різного роду задачі. Для прикладу, POST метод, використовують для створення нових даних. PUT для додавання нових даних. В більшості випадків, для спрощення системи використовується POST метод. Але метод PUT є більш інформативний. Натомість PATCH метод, використовується для зміни в наявному об’єкті, але для зміни даних не потрібно передати усі дані, а лише ті, які потрібно змінити. Це набагато спробує редагування даних, коли потрібно змінити лише одне поле, а не значення усього об’єкту[30].

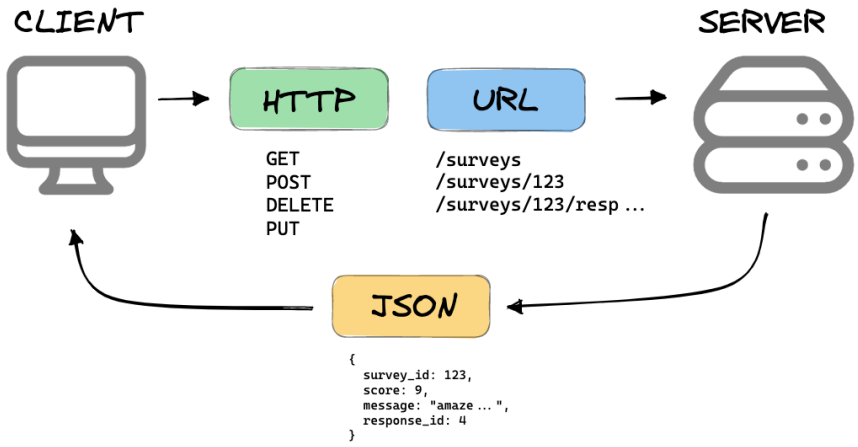


Рисунок 2.5 – Приклад взаємодії в REST API [4]

Метод DELETE із своєї назви, що дає зрозуміти, надає можливість на видалення даних. Він виконує видалення даних на стороні backend, по HTTP протоколу.

Методи HEAD та OPTION використовується не для маніпуляції з даними, а для отримання сервісної інформації. У випадку використання методу HEAD, він дає можливість здійснити запит, та отримати інформацію про сервер, який обслуговує запити. Отримати header, та іншу службову інформацію. У випадку

використання методу OPTION, дозволяє отримати інформацію, які методи доступні для виконання. Цей метод корисний, якщо немає документації, щоб зрозуміти, як спілкуватися з API[30].

Протокол GRPC це сучасна реалізація відкритого методу для високо продуктивного виконання віддалених функцій(RPC) в будь-якому середовищі. Основна його перевага над REST API, є в тому, що клієнт, який виконує запит до сервера заздалегідь знає, які методи доступні для використання, та які параметри очікує отримати сервера. Оскільки кожен аргумент є строго типізованим, це дозволить зменшити помилки під час запитів, ніж у випадків з RESTAPI, які не є чітко типізованим.

Особливість використання даного методу передачі даних є в тому, що він використовується для реалізації спілкування між мікросервісами. Під час створення мікросервісної архітектури надається перевага використання цього протоколу, ніж REST API, шляхом, стабільності, та можливості стиснення даних під час передачі. Це дозволяє більш ефективно спілкуватися між сервісами.

Для реалізації інтерфейсів, і зручності розробки використовується підхід створення proto файлу, в якому описані функції, та вхідні дані, та вихідні параметри функцій. Даний підхід дозволяє мати згенеровані proto файли, які знають як і клієнт, та сервер, і на своїй стороні згенерувати методи, після чого реалізувати відповідні методи своєю бізнес-логікою. Такий підхід зручний, тому що не потрібно реалізувати взаємодію роботи протоколу самостійно.

Натомість варто підмітити, що це не єдині методи, які використовуються для передачі даних. Окрім цих двох, доступний такий як GraphQL[29], який схожий на GRPC завдяки тому, що є proto файл, який також потрібно створити та заповнити основні методи взаємодії. Цей протокол використовується для реалізації браузера та API, наприклад для реалізації соціальних мереж, через те, що цей протокол дає дуже зручно отримувати частину даних. Це маніпулювання запитом, здійснюється на стороні фронту. Окрім цього переваги цього методу є в тому, що backend

розробнику потрібно реалізувати інтерфейс, а вже сам клієнт вирішуватиме, скільки даних він хоче отримати з backend.

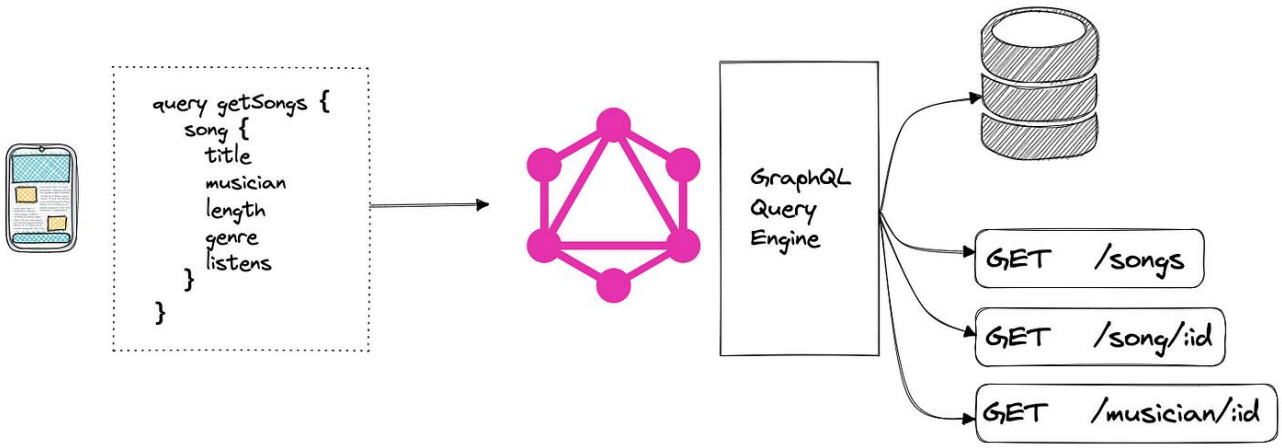


Рисунок 2.6 – Приклад запиту по протоколу GraphQL [4]

Перевага полягає в тому, що завдяки цій реалізації, зменшується кількість запитів до БД, що дозволяє уникнути перевантаження БД, під час витягування даних.

Також варто підмітити, що ці методи є синхронного типу, тобто, поки не обробиться один запит, не можливо надіслати інший метод. На розв'язання даної проблеми взаємодії реалізована система брокеру повідомлень, який дозволяє надсилає дані в брокер, а інші системи будуть слухати дані, які приходять. Це дозволяє асинхронно взаємодіяти із системою, що дозволяє гнучко взаємодіяти із сервісами. Також варто зазначити, що використання даного підходу дозволяє покращити надійність системи, через те, що у разі недоступності одного із сервісів, інші сервіси не можуть надіслати дані, що призводить до втрати даних. Із використання брокеру повідомлень дані будуть надіслані в будь-якому випадку, навіть, якщо сервіс в цей момент часу завантажений роботою.

Ще з переваг системи є те, що дана система дозволяє розподіляти роботу між різними сервісами, через те, що кожен сервіс буде отримувати частину даних. Це зручно коли навантаження на систему зростає і для розв'язання проблеми варто лише масштабувати систему[28].

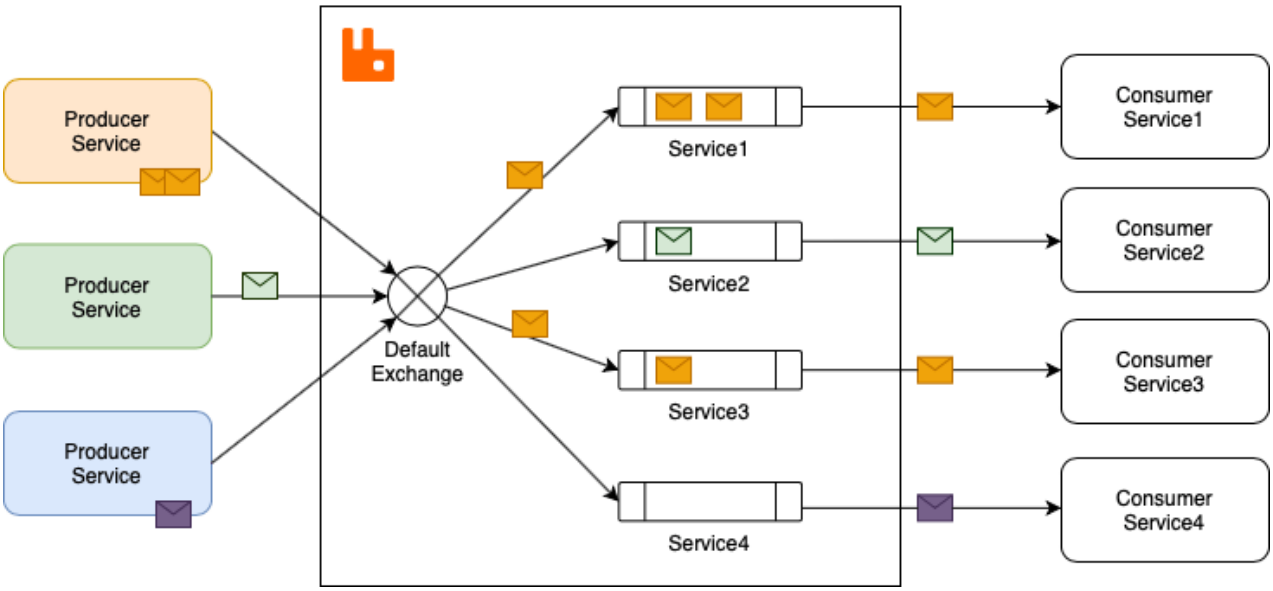


Рисунок 2.7 – Приклад взаємодії брокера повідомлень rabbitmq [10]

Одним із брокером повідомлень є протокол rabbitmq, який є доволі популярним рішенням, на ринку. Дане рішення має вихідний код, який можна переглянути на Github та має понад 12 тисяч зірочок на репозиторію github[21].

Саме програмне забезпечення надає кілька абстракцій, які дають можливість взаємодіяти обмінюватися даними. Для розуміння, в rabbitmq є така абстракція як Exchange, від імені якої здійснюється відправлення повідомлення. Вона визначає куди має повідомлення відправити. Місце куди буде відправлено повідомлення, називається Queue. В це місце відправляється повідомлення, відправлене Exchange-м. Але сам Exchange нічого не знає, про те, кому відправляти повідомлення. Для того, щоб вказати якій черзі відправити повідомлення, використовується Binding. Дана сутність дозволяє зв’язати вибраний Exchange з вибраним queue.

В ідеї використання брокеру повідомлень в мікросервісній архітектурі є те, щоб усі дії, які відбувається в будь-який момент часу надсилалися, не через брокер повідомлень, а вже сервіси, які слухають відповідний channel. Це дозволяє гнучко повідомляти потрібні сервіси, що відбулося і дозволяє зменшити споживання додаткових ресурсів під час роботи мікросервісної архітектури.

Отже для побудови архітектури використовуються такі методи як брокер повідомлень, та GRPC, для синхронно моментального запиту до сервісу та отримання даних з нього.

2.4 Вибір апаратного забезпечення

Вибір між апаратною частиною складний процес, через те, що це потребує правильного рішення, через те, що від вибору залежить технічні обмеження та можливості.

В нас час залежно від поставленої задачі, є можливість вибору від плат Arduino, так і використання Raspberry PI, так і звичайні системи на архітектурі amd64. Використання систем на базі процесора amd64 це не завжди хороше рішення, через те, що ці системи споживають велику кількість ресурсів, тому є потреба у використанні систем іншого типу. Використання типу плат, виду Arduino також мають обмеження, через те, що білінгова система має потребу в роботі ОС, що звичайні плати arduino не надають такої можливості. Тому вибір припадає між amd64, так і arm. Використання пристроїв на архітектурі arm зазвичай має дешевший характер, що дозволяє зберегти додаткові кошти. Для прикладу AWS надає знижки на використання серверів на базі arm архітектурі.

В нашому випадку, вистачить Raspberry PI, який містить на борту linux. Маючи дані можливості, ми можемо запустити будь-яке програмне забезпечення на цій платформі[33].

Дане апаратне забезпечення містить архітектуру процесора arm64, для запуску програм, потребують підтримку у відповідній архітектурі. Дане рішення варто враховувати під час розробки програмного забезпечення. Також розробка програмного забезпечення відбувається на мові Go для запуску програмного забезпечення не має великої проблеми.

Також варто врахувати те, що дане рішення використовувати версію моделі із 8GB ОЗП. Дана кількість оперативної пам’яті дозволить вільно розмістити усі компоненти системи.

За всю історію розвитку Raspberry PI було випущено велику кількість моделей. Найпершим рішенням, яке було створено командою Raspberry PI Foundation, є модель Pi Model B. Дане апаратне забезпечення було випущено в лютому 2012 році. Дана модель містила процесор ARM11, на платі було розміщено 26 контактів GPIO, та розмірами материнської карти, яка приблизно дорівнює звичайній банківській карті[33].

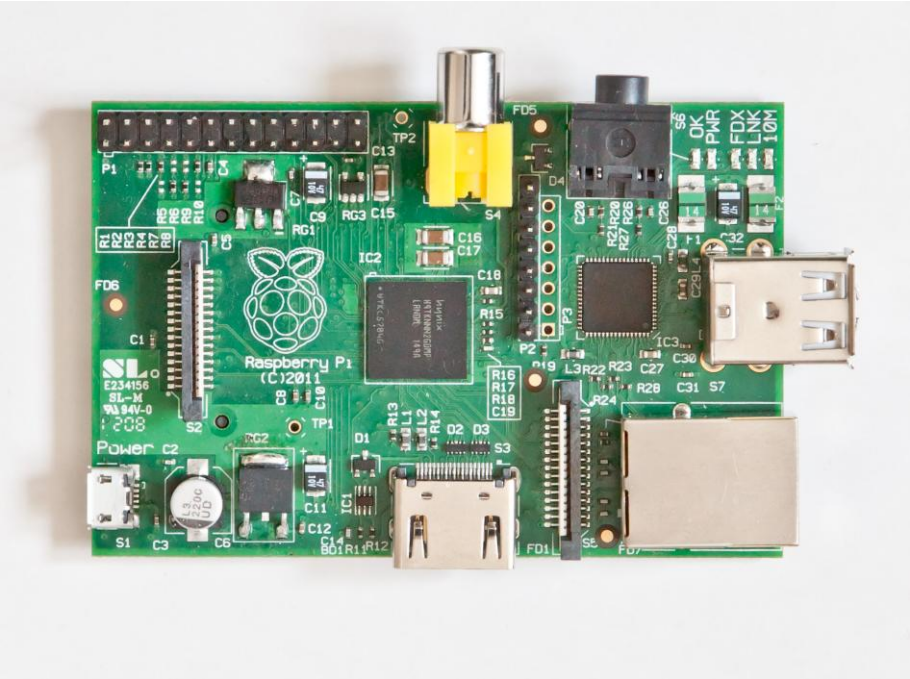


Рисунок 2.8 – Rasbperry PI Model B [33]

Після випуску першої моделі Raspberry PI, була випущена наступна модель, яка мала назву Raspberry PI model A. Ця версія була дешевша за попередню модель, шляхом відсутності Ethernet порту. Окрім цього було зменшено кількість оперативної пам’яті та кількість USB портів. Це допомогло зробити більш доступний варіант для більшості[33].

Після виходу цих двох моделей, на світ людство побачило розвиток моделей B та A. В цьому випадку, це моделі PI B+ та A+. В цих моделях було додано більше портів. У випадку B+ ця кількість дорівнює 40 контактів GPIO, чотири порти USB та слот USB[33].

У випадку A+ мала ті ж самі покращення, що й B+, але була покращеною версією A.

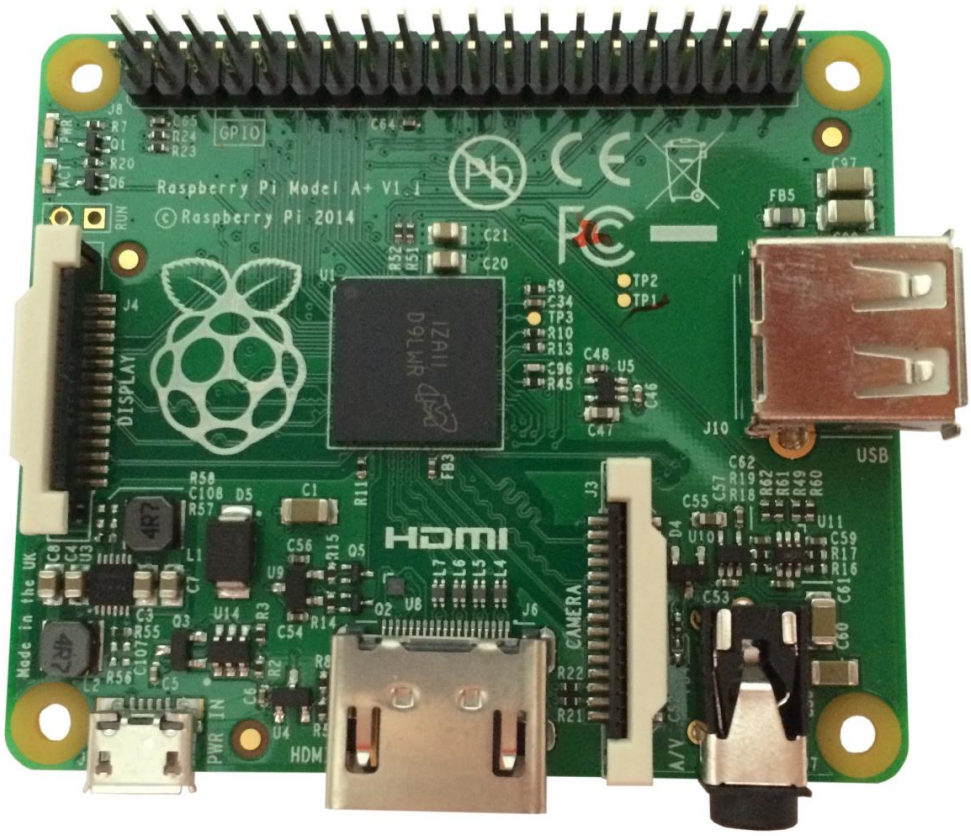


Рисунок 2.9 – Raspberry PI модель A+ [33]

Після випуску цих моделей, в 2015 році була випущена на свій модель PI 2 Model B, що був розміщений 32 бітний процесор ARM Cortex-A7, що мав тактову частоту 900 МГц та 1 ГБ ОЗП. Через деякий час була створена модель, що містить 64 бітний процесор, який був встановлений в PI 3 Model B[33].

На заміну цього процесору в 2016 році, була випущена модель Raspberry PI 3 Model B, яка містила на борту 64 бітний 4 ядерний процесор ARM Cortex-A53 з тактовою частотою 1.2 ГГц. Була присутня підтримка завантаження з USB[33].

Згодом, після цих змін, була випущена 4 версія цієї плати. Ця модель містила назву Paspberry PI 4, випущена в черні 2019 році, та містила 64 бітний процесор ARM Cortex A72, з частотою 1.5 ГГц, можливість використовувати дводіапазонний Wifi, Bluetooth версії 5, використовувати 1 Гбіт/с для доступу до інтернету, два порти USB версії 2.0 та два парти 3.0. Окрім цього є можливість використовувати підключення двох моніторів 4к якістю до плати, через два порти micro HDMI. Були доступні версії з 1, 2, 4 та 8 ГБ оперативної пам’яті, що дозволяло вибрати версію під різні задачі[33].

Після успіху з випуском четвертої версії була випущена остання на цей момент доступна версія, яка має назву Raspberri PI 5, в 2023 році, Дана модель містить ARM Cortex-A76, із тактовою частотою 2.4 ГГц. Містить графічний процесор, контролер вводу та виводу, кнопкою живлення, для зручності включення. Були доступні моделі з 4 та 8 ГБ оперативної пам’яті. Також з’явилися версії з 2 ГБ оперативної пам’яті та 16 ГБ оперативної пам’яті[33].

З додаткових можливостей, варто згадати те, що є можливість підключення карт розширення, для використання M2 накопичувача.

Рисунок 2.10 – Raspberry PI 5 [33]

2.5 Вибір програмного забезпечення

Вибір програмного забезпечення важкий процес. В нашому випадку, потрібно орієнтуватися на тому, що працює на Raspberry PI. Чудовим прикладом підійде мова програмування go яка дозволяє компілювати код програмного забезпечення під різні ОС та архітектури. Це дозволяє створити програмне забезпечення і зібрати в кінцевий варіант та запускати будь-де.

Також варто зазначити що останнім часом популярності набирають системи оркестрації та контейнеризації. Одна із популярних рішень є Docker, який дозволяє контейнеризувати додаток, та запускати будь-де де є Docker. Через те, що Raspberry PI входить у підтримку, дане програмне забезпечення чудово підходить для запуску.

Також варто зазначити, що для запуску системи та надійності, варто використовувати системи оркестрації. Одна із популярних рішень є Kubernetes, який дозволяє створювати кластери із персональних ПК, об'єднуючи їх у кластер. Додаткові можливість дають можливість легко масштабувати горизонтально додаток, що під час великих навантажень. Але одна із моментів Kubernetes є те, що він не потребує наявність двох нод, для створення кластера. Для вирішення цього моменту, є потреба у використанні k0s, який дозволяє запустити програмне забезпечення із використанням лише одного екземпляру Raspberry PI.

На базі даного мікроконтролера є можливість використовувати Linux з дистрибутивом Debian з графічним інтерфейсом, так і без графічного інтерфейсу. Таким, варто зазначити, що окрім Debian[40], є можливість встановлення Ubuntu так і Alpine, що є дистрибутивом, який займає невелику частину дискового простору в порівняні з першими.

Для взаємодії мікросервісної архітектури, є використовуються додаткове програмне забезпечення, таке як база даних, та брокер повідомлень. Найбільш популярне рішення є PostgreSQL, яке найбільше реалізованого функціоналу, із порівнянням з іншими серверами БД. У випадку вибору брокеру повідомлень для

даного проєкту чудовим вибором є RabbitMQ, який є доволі таким популярним рішенням[31].

Також варто додати, що PostgreSQL є проєкт із відкритим програмним кодом. Дане програмне забезпечення має зручну ліцензію використання, що надає можливості вільної модифікації програмного забезпечення, без обмежень з боку ліцензії. Дане програмне забезпечення написане мовою C, що є показником високої продуктивності. Вихідний код даного проєкту можна дізнатися переглянути на github. Даний проєкт містить 17.6 тисяч зірочок, що є показником популярності проєкт[32].

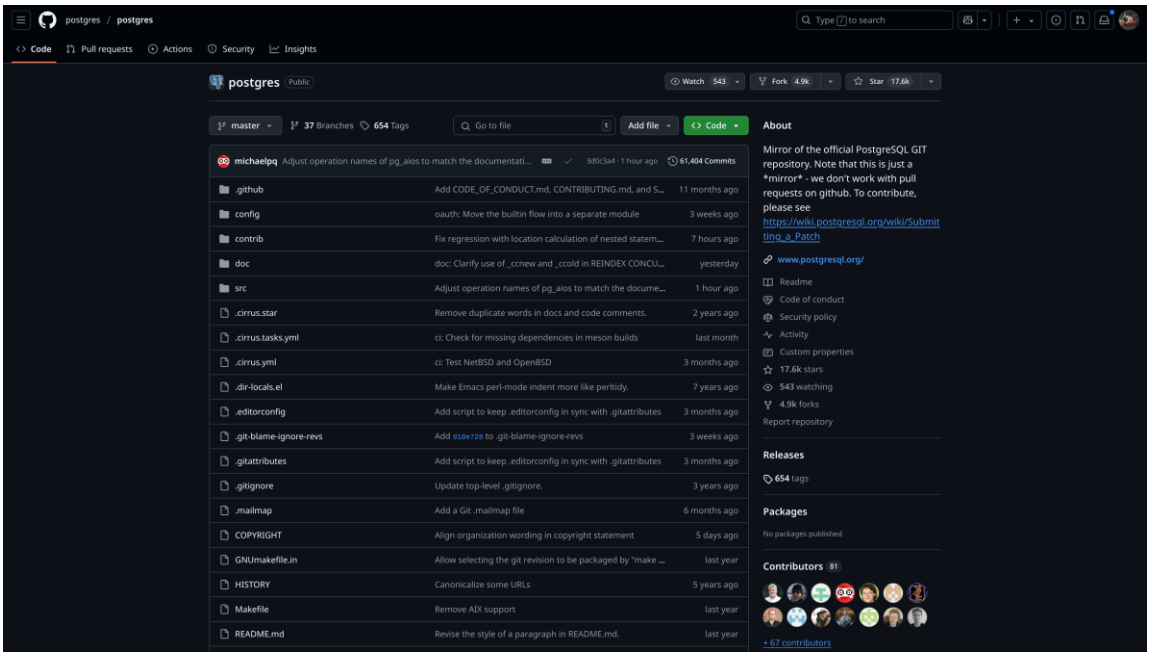


Рисунок 2.11 – Github репозиторій PostgreSQL

Дане програмне забезпечення підтримує найкращі практики відповідності даних. Підтримується атомарність, консистентність, ізолюю та довговічність. Завдяки цим перевагам є впевненість, що у разі збої системи, або апаратної складової не будуть втрачені дані.

Також, залежно від складності системи, є можливість використовувати PostgreSQL в режимі роботи кластера[32]. Це рішення дозволяє розподілити

навантаження між кількома вузлами, що дозволяє зменшити навантаження при продакш середовищі.

Залежно від вибору апаратної складової та бізнес потреб є можливість запускати PostgreSQL на різному програмному забезпечені. Один із варіант, це використання звичайних віртуальних серверів, на якому буде встановлено дане програмне забезпечення. Але в більш продуктивних середовищах, є можливість використовувати системи оркестрації, наприклад Kubernetes. Для даного рішення є готові інструменти, для прикладу HelmChart[34], які дозволяють розміщувати додати в короткі терміни. Найбільш популярним рішенням, у вибору HelmChart є використання продуктів від bitnami[34]. Дана компанія надає великий спектр helmchart із різним програмним забезпеченням. Наданий момент їх кількість перевищує більше ніж 100 різних helmchart із різним програмним забезпеченням. Для програмного забезпечення PostgreSQL[32], також є відповідний helmchart, який надає зручно налаштовувати як в режимі одного екземпляру, так і в режимі кластера.

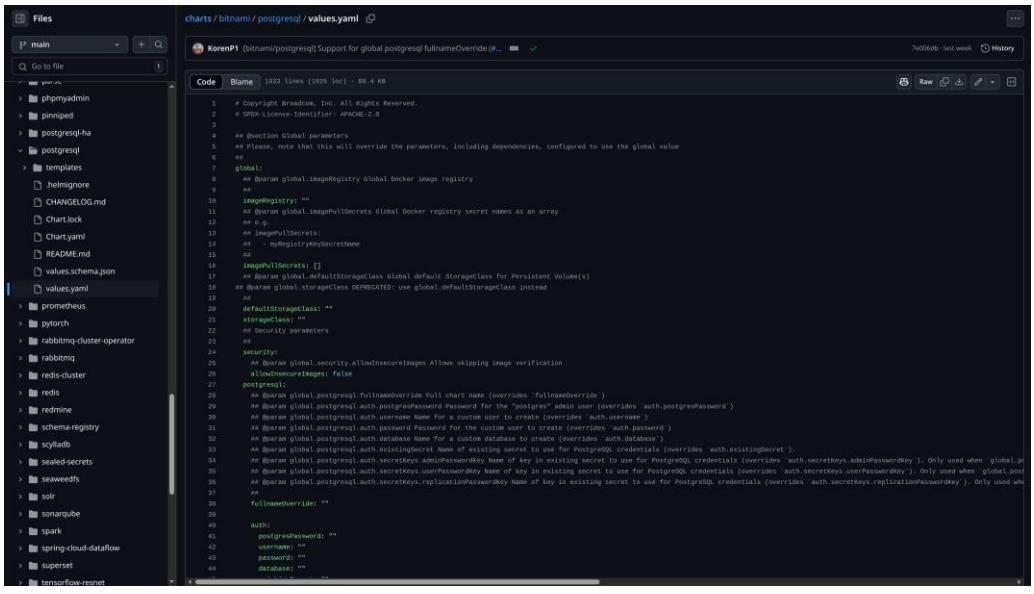


Рисунок 2.12 – HelmChart Values для PostgreSQL

Також, PostgreSQL підтримує виконання складних запитів, до яких відносить JOIN операції, які є об’єднанням таблиць, виконання під запитів.

Також варто зазначити що є підтримка створення власних типів розширення, функцій різноманітних операцій та розширення, які дозволяють використовувати PostgreSQL на повну.

Окрім цього дана система дозволяє використовувати не стандарті типи даних, такі як JSON, XML, масиви та UUID, дані геолокації. Ці можливості дають змогу неповну використовувати функціональність даного сервера баз даних. Використання найновітніших засобів безпеки дозволяють використовувати це рішення в банківській сфері.

Також, варто звернути увагу на RabbitMQ, який використовується в ролі брокера повідомлень. Вихідний код даного проєкт можна побачити в репозиторії github[31].

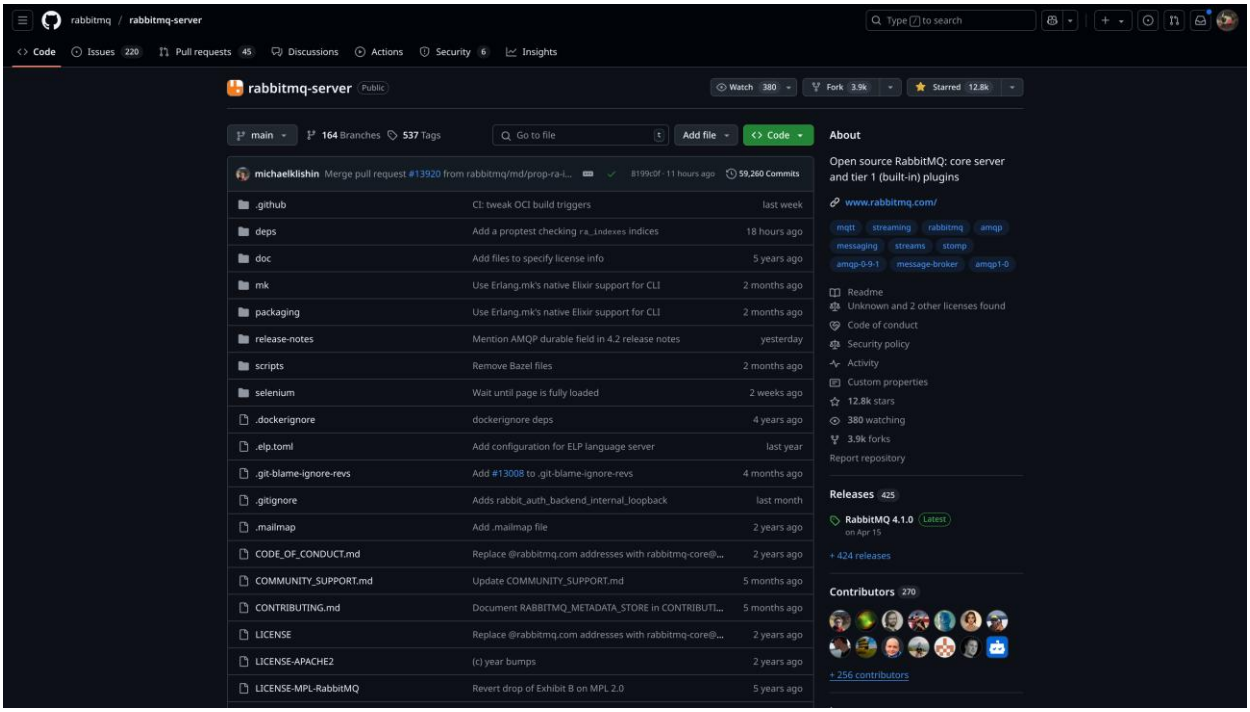


Рисунок 2.13 – Github репозиторій rabbitmq

Дане програмне забезпечення дозволяє бути впевненим, що передача повідомлення від відправника та отримування буде доставлено успішно. Цей механізм досягається за рахунок підтвердження отримання, що дозволяє бути

впевненим, що повідомлення не буде видалене, до моменту того, як отримувач отримає повідомлення.

Окрім цього дане програмне забезпечення дозволяє гнучко будувати складні системи, що реалізовувати складні варіанти доставки повідомлень. Для прикладу доступні наступні типи повідомлень:

– direct;
– topic;
– fanout;
– headers.

Кожен із цих типів виконують різну задачу. Direct тип дозволяє надіслати повідомлення, якщо ключ маршрутизації точно співпадає із ключем на якого підписана черга. У випадку topic - якщо повідомлення яке надсилається, містить ключ, який по шаблону збігатися з ключем черги. Тип маршрутизації fanout дозволяє надіслати усім повідомлення, ті хто підписаний на відповідний exchange, який відправляє це повідомлення. У випадку із headers типу, маршрутизація відбувається за допомогою заголовків, а не за допомогою ключа.

Також, варто згадати про те, що rabbitmq має велику кількість плагінів, які дозволяють інтегрувати дану систему із такими протоколами, як MQTT, STOMP, HTTP, Websocket та інші. Це дозволяє використовувати дану систему в різноманітних цілях. Від задач використання в IT пристроях, так і в складних системах.

2.6 Вибір інструментів для створення програмного забезпечення

Через те, що Raspberry PI є контролером, з підтримкою програмного забезпечення на ядрі Linux, програмне забезпечення яке може використовуватися для створення програмного забезпечення може бути різноманітним. Для більшості продуктивних систем може використовуватися як і C, так і C++. В цьому випадку, для роботи з Raspberry PI, може використовуватися python, так і C. У випадку, для

створення програмного забезпечення у випадку створення використання даного програмного сервісу, найкращим вибором, краще використовувати go. Особливість даної мови є те, що код, має можливість компілюватися в бінарний вигляд. Дана можливість дає те, що використання даного підходу дозволяє отримати бінарний файл, який має велику продуктивність виконання операцій. Також варто навести момент, що go дозволяє комплювати програмне забезпечення для роботи на різній операційні системі, та на різній архітектурі. Для прикладу, go дозволяє компілювати код на наступні архітектури[27]:

- 386;
- amd64;
- amd64p32;
- arm;
- armbe;
- arm64;
- arm64be;
- ppc64;
- ppc64le;
- mips;
- mipsle;
- mips64;
- mips64le;
- mips64p32;
- mips64p32le-;
- ppc;
- riscv;
- riscv64;
- s390;
- s390x;
- sparc;

– sparc64;

– wasm.

Окрім цього, список операційних систем, які підтримується виглядає наступним чином:

– aix;

– android;

– darwin;

– dragonfly;

– freebsd;

– hurd;

– illumos;

– js;

– linux;

– nacl;

– netbsd;

– openbsd;

– plan9;

– solaris;

– windows;

– zos.

Завдяки тому, що є можливість збірки програмного коду, під різну архітекту, це надає перевагу, запуску програмного коду, під різні архітектури.

Але в нас час використання даної можливості варто вувати Linux, через те, що дана система споживає набагато менше ресурсів.

Для універсальності систем, для запуску додатків використовується Docker. Інструмент, який надає можливість запускати додатки в контейнеризованому середовищі, який буде працювати на будь-якій системі, де є Docker Engine. Дана технологія використовується в наш час, для більшості додатків, що дозволяє не хвилюватися із запуском додатків на різних платформах.[2]

Також, в для запуску віртуальних машин, найкраще рішення для запуску використовується Proxmox. Дане рішення дозволяє запускати віртуальні машини на персональних комп'ютерах, що дозволяє створити менші копії віртуальних машин. Особливістю даної системи є те, що Proxmox дозволяє об’єднувати систему в кластера для більшої надійності.[6]

2.7 Висновки

В другому розділі було визначено умови та потреби до систем управління ресурсами, для білінгової системи із рахуванням вимог до архітектури програмного забезпечення. На основі цих вимог, поставлених для виконання завдання, було сформовано перелік потреб в апаратних та програмних вимог, враховуючи сучасні потреби в бізнесі.

Для зберігання даних, які використовуватися для роботи системи, було вибрано використання PostgreSQL. Це рішення має високу надійність під час запису, оновлення та читання даних.

Для спілкування між сервісами, було обрано рішення використання rabbitmq, яке дозволяє асинхронно передавати дані між сервісами, які будуть виконувати поставлену задачу.

Для апаратної складової було вибрано апаратне забезпечення Raspberry PI, яке дозволяє запускати різноманітне програмне забезпечення на Linux. Було вибрану операційну систему для запуску Debian, а система оркестрації kubernetes.

Для вибору програмної складової було вибрано Go, як мова, зручна для компілювання коду під різну архітектуру. Це дозволяє зменшити витрати на переписування код під іншу архітектуру, що спрощує перенесення коду.

Через те, що було вибрано використання окрестрації Kubernetes, для запуску додатків на цій системі було вибрано використання Docker, для контейнеризації додатків.

3 ПРОГРАМНО-АПАРАТНА РЕАЛІЗАЦІЯ КІБЕРФІЗИЧНОГО БІЛІНГУ ТА УПРАВЛІННЯ РЕСУРСАМИ ДЛЯ ВІРТУАЛЬНИХ СЕРВЕРІВ

3.1 Опис програмного та апаратного забезпечення кіберфізичної системи білінгу та управління ресурсами для віртуальних серверів

Кіберфізична система для управлення ресурсами для віртуальних серверів є складним проєкт, який являє собою системи обміну даних, передачі та обробки даних, маніпуляція даними. Основа задача даної системи варто зазначити, що є потрібна в обрахунку, маніпуляція та передачі даних.

Для створення системи було вирішено обрати підхід мікросервісної архітектури, через те, що даний підхід є сучасним рішенням. Також було прийнято рішення використовувати Docker, як основну систему контейнеризації. Для запуску даного програмного забезпечення прийняти було взяти рішення, використання Kubernetes, як систему оркестрації додатків, яка має велику популярність у наш час. Така архітектура додатку, дозволяє гнучко масштабувати, розв'язувати проблеми навантаження. Такий підхід також є зручним в тому, щоб отримати можливість зручно оновлювати додатку, оновлювати окремі компоненти в системі.

Програмна складова даної системи побудована на Go, яка є чудовим вибором в мікросервісній архітектурі. Для запуску системи, яка побудована на архітектурі, було обрано RaspberryPI 5, з характеристиками, такими як CPU, модель яка має назву 64-bit Arm Cortex-A76, вмістом оперативної пам’яті в розмірі 8 GB, та вмістом пам’яті 32 Гб. Дані характеристики достатні для запуску побудованої системи та повноцінної роботи.

Враховуючи, того, що апаратна складова, Raspberry PI, варто зазначити, що для роботи системи потрібно використовувати програмну складову. Як такою, є використання Linux, на базі Debian 11. Окрім цього, компонент, який встановлюється це K0s, який має можливість використовувати, в режимі 1 node. Модульна складова системи містить сервіс, user, який відповідає за маніпуляцію із

користувачами. Сервіс, авторизації, який містить логіку авторизації, та аутентифікації, є окремим сервісом, і виконує лише одну задачу, згадану вище. Додатково, для можливості побудови цієї системи, як білінгової, є також наявність сервісів тарифу, та сервісу підписок. Даний підхід дозволить незалежно оновлювати кожен компонент, не втручаючись в роботу інших компонентів.

Окрім цих сервісів, є модуль взаємодії через API Proxmox, який надає зручний інтерфейс та гнучко інтегрується до наявної системи. Використання модуля, як окремої складової зумовлене потребою в інтеграції до інших систем, для розширення майбутньої системи.

Кожен сервіс містить свою БД, для зберігання постійних даних. Компонентом, для зберігання даних, виступає Postgress версії 17.4.

Для взаємодії між сервісами, використовує компонент, під назвою RabbitMQ, який є основним способом спілкування, та повідомлення в сервісу змін іншим сервісам. Версія програмного забезпечення використовується 4.1

Для побудови, та контейнеризації додатків, використовується Docker версії 28.2.2.

У випадку K0S, як аналог Kubernetes на raspberry PI, використовується 1.33, як одна із останніх рішень.

Програмна складова бекенд частини, реалізована на Go версії 1.24, яка є одна з останніх версій. Для реалізації кожного із сервісів, використовується фреймворки для побудови API із використанням протоколів HTTP та GRPC. Також, як основний компонент використовується клієнтська бібліотека для роботи із RabbitMQ. Окрім цих компонентів, було використані архітектурні підходи, патернів, із використанням модульної структури API. Сервіси використовують патерн Dependency injection, що є вибором побудови складних архітектурних рішень. Використання фреймворків для роботи із БД, та міграції, також є складною системи.

Для синхронної взаємодії та моментальної відповіді на запит сервісу, використовується протокол GRPC. Даний підхід надає можливість моментально отримати відповідь, на задану відповідь, яка потрібна в момент виконання операції

одним із сервісів. Цей підхід є хорошим рішенням, який стосується розділення відповідальності за функціонал кожного із сервісів.

У випадку RabbitMQ, використання якого як чергу повідомлень, зумовлене те, щоб була можливість сповіщати систему про наявні зміни в одного із сервісів. Це означає, що у випадку критичних змін, для прикладу, видалення користувача, інформація, яка зберігається в інших сервісів, які містять дані, пов'язані із цим користувачем, потрібно відповідно відреагувати. Даний підхід дозволяє дотримуватися підходу цілісності даних, коли не має можливості зв'язати явно таблиці зв'язком між таблицями.

Клієнтська складова, яка є основним компонентом системи, є використання NextJS, як інструмент, для побудови сучасних Web додатків. Дана бібліотека дає можливість створити сучасний додатком, дані які дістаються по API, від Backend сервісу.

Розміщення кожного із компонентів використовується за допомогою на k0s, який, за допомогою HelmChart. Даний інструмент надає можливість створювати архіви з k8s manifest, для розверстування додатків.

Кожен сервіс, який згадується вище, знаходиться в кожному namespace, на який встановлюється HelmChart додаток. БД не є створеним компонентом, та використовується як стороннє рішення від компанії bitnami.

Кожен сервіс, який реалізований виконує вузконаправлену задачу. Сервіс api-user-service, виконує задачу зберігання інформації про користувачів, пароль для авторизації користувачів. Також, даний сервіс, реєструє в rabbitmq, Exchange з назвою «user.events», із типом topic. Даний підхід дозволяє реєструвати зміни в БД сервісу api-user-service наприклад, під час видалення користувача з БД. Після видалення надсилається event усім чергам, які підписані на Exchanges, з назвою «user.events» зображеного на рисунку 3.1.

Аналогічно, подібним чином, реалізований сервіс api-tarriff-service, який працює аналогічно, та повідомляє усім підписаним на Exchanges, тим хто очікує зміни від api-tariff-service. Відповідно, створюється відповідний Exchanges, з

назвою «tariff.events» зображеного на рисунку 3.1, та пересилає на ті черги, сервіси яких очікують надіслані дані.

Структура запиту, яка надсилається, в ролі events, виглядає наступним чином: ключ - «user.deleted», та json структура із ключем «user_uuid» та значенням UUID з таблиці користувачів, в даному випадку. Таким чином вдається відслідковувати зміни здійснені одним сервісом, та повідомляти про них, іншим сервісам, яким це дійсно потрібно.

Компонент api-subscription-service відповідає за зв’язок тарифів та користувачів, а саме за активні послуги. Відповідно, при видаленні користувача чи тарифу, є потреба, дізнатися, що дійсно сталося видалення, і в БД здійснити відповідні зміни в даних. Для відслідковування змін, створюється окрема унікальна черга, на яку підписаний компонент api-subscription-service, та очікує вхідні events від api-user-service, та api-tariff-service. Додатково, щоб усі events були надіслані на чергу «subscription.service», прив’язується Exchange з Queue. Для цієї створюється компонент Bind, який зв'язує відправника з чергою. Створюється дві прив’язки відправника user.events та tariff.events, відслідковуючи ключі, які містять ключі «user.*» та «tariff.*». Відповідно, після того, як сервіс із користувачами, в себе видалить користувача, інформація про цю операцію надсилається в чергу subscription.service. Даний підхід дозволяє гнучко та асинхронно реагувати на зміни. Окрім цього, щоб «слухати», усі здійснені event потрібно бути підписаним на чергу, тому, використовується операція Consume, яка зв’язує сервіс із чергою.

Відповідно порядок відправлення повідомлення має декілька кроків, такі як відправка повідомлення за допомогою Exchanges, надсилання повідомлення використовувати ключа повідомлення, чи заголовків, надсилання повідомлення, у відповідну чергу, за допомогою зв’язаного Bind компонента, який зв’язує відправника із чергою. Після надісланого повідомлення очікується видалення повідомлення або відсутність видалення. Завдяки чому, досягається надійність системи, за допомогою підтвердження отримання даних.

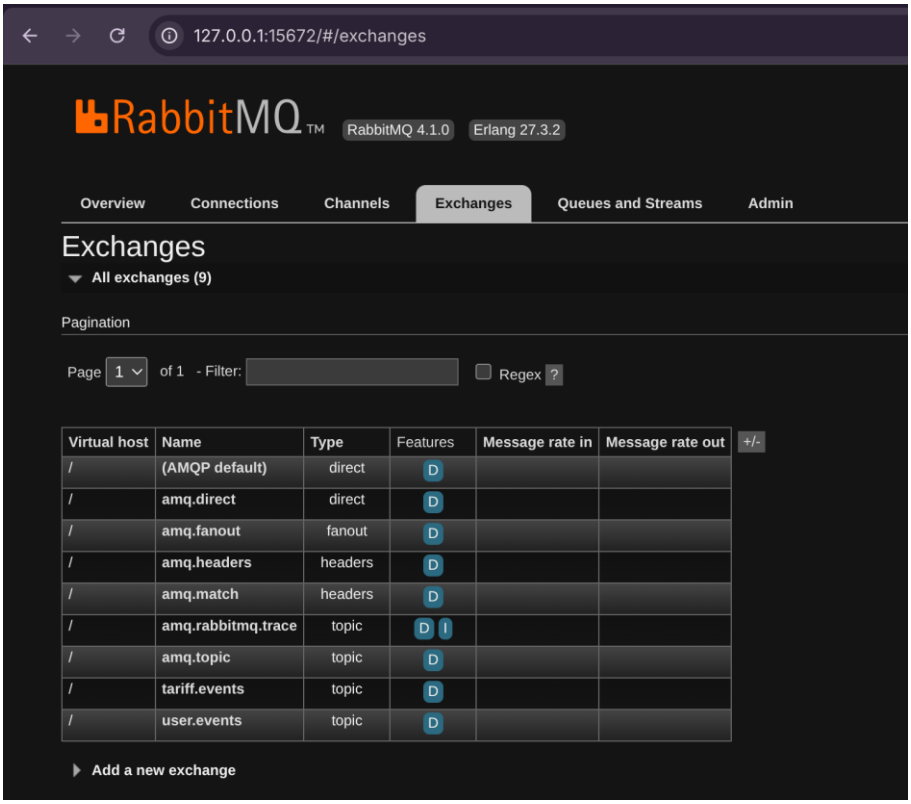


Рисунок 3.1 – Створені Exchanges для обміну events

Для зручності розділення логіки виконання коду, за типом events, реалізована окрема бібліотека, яка ловить відповідний event, визначає та визначає, яка функція має виконати цю операцію. Ця бібліотека має назву kinkajou, та складається з кількох компонентів, такі як основна структура App, яка містить інформацію про зареєстровані handlers, та посилання на функцію, яка виконується у випадку, якщо прийшов event, який не зареєстрований в сервісі. Відповідно, для реалізації розпаралелювання events, використовується глобальна функція, яка слухає усі повідомлення, які надсилаються в чергу, та по ключу, визначає, яка функція буде виконувати ту чи іншу операцію. Цей функціонал надає можливість розділити виконання коду, поділивши кожен компонент на один логічний вузл, який виконує одну задачу, яка запускається по event.

Дана перевага дозволить бути впевненим, що усі дані надісланні через rabbitmq, що збільшує надійність системи.

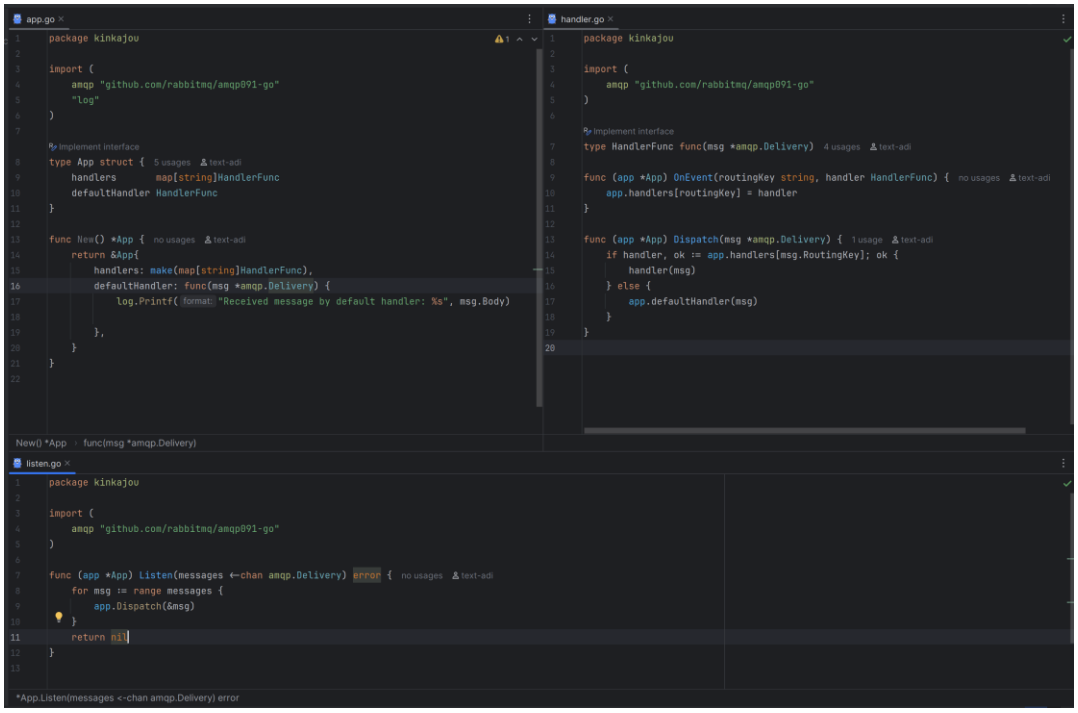


Рисунок 3.2 – Програмна реалізація бібліотеки kinkajou

Підхід, який реалізовується за допомогою events зручний у випадку, коли отримати повідомлення про стан системи. Але у випадку, коли потрібно отримати інформацію моментально, використання events не підійде. Для цього використовується GRPC виклики.

У випадку, використання компоненту api-subscription-service реалізована API, яка дозволяє виконувати GRUD операції над даними. Під час операції Create, перевіряється наявність користувача та тарифу у відповідних сервісах. Для цього виконується прямий GRPC виклик функції, на наявність операції користувача та тарифу відповідно.

Для реалізації GRPC викликів, реалізовані файли, які описують структуру GRPC методів. У випадку tariff та user сервісів, методи аналогічні, які на вхід функції ExistsByID подають ID елемента. Після виконання функції отримується результат, який повідомляє, чи є наявний елемент в БД, чи ні.

Наступний компонент, який використовується це модуль api-proxmox-module, який дозволяє взаємодіяти із віртуальними машина, за допомогою запитів GRPC та HTTP. Даний модуль надає можливість створення, видалення послуги. Також

перезавантаження, запуск та зупинки віртуальної машини. Перші дві операції створення та видалення послуги, є базовими в кожному модулі. Даний підхід взято, для можливості розширювати взаємодію системи, не змінюючи логіку базових сервісів. Спілкування api-proxmox-module із сервісом api-subscription-service реалізовується за допомогою GRPC протоколу. Для зручності реалізована бібліотека, яка grpc-dispatcher-library, яка схожим чином в порівнянні kinkajou, але містить додаткові інструменти, потрібні саме для реалізації зовнішніх модулів.

Дана бібліотека складається із наступним частин:

– app.go;
– ctx.go;
– handler.go;
– listen.go;
– router.go.

Кожен із компонентів, складається виконує важливу роботу. Для ознайомлення файл app.go, містить базову логіку, обробки бібліотеки. В цьому файлі присутній метод Dispatch, який виконує логіку запуску функцій по ключу.

Окрім цього, реалізовані базові методи, які є системними. Для прикладу, це методи ListAvailableAction, який надає інформацію про доступні операції із модулем. Функція GetActionSchema надає інформацію, по конкретний Action. Надає опис операції, його статус, чи він публічний, чи ні, та вхідні параметри. Відповідно вхідні параметри містять наступні дані:

– назва;
– опис;
– тип змінної;
– чи обов’язковий параметр.

Відповідні, дані можливості дають можливість розшити функціональність GRPC, надаючи гнучкість системи.

Відповідно, після того, як виконується функція, яка прив’язані до відповідної дії, під час виконання параметрів, доступні наступні поля, а саме, контекст

виконання функції. В цей контекст надсилається базовий контекст go, та вхідні поля. Вхідні поля містять інформацію по ID запиту, та відповідна дія, яка пов’язана із цією дією. Окрім цих двох параметрів, на вхід подається вхідні параметри

Вхідні параметри можуть містити наступні типи:

– тип None;
– список;
– рядок;
– логічний;
– число.

Одна із можливостей, окрім розподілення обробки GRPC запитів, використовуватися можливість визначати чи метод публічний чи приватний. Ця особливість потрібна, щоб api-subscription-service мав можливість використовувати стандартні методи, так і надав доступ до методів, та виступав в ролі proxy сервісу

3.2 Створення БД для роботи системи

Для побудови системи використовується окремі сервера БД. Для вирішення проблеми було вибрано Postgress, в режимі 1 екземпляра. Для роботи системи, використовується БД, розвернутому на кластері K0s. Для цього встановлено postgres в режимі helmchart. Для роботи системи у випадку сервісу api-user-subsription використовується наступні таблиці:

– users;
– user_metadata;
– credentials.

Таблиці users зберігає загальну інформацію про користувача. Електрона пошта, номер телефону, який пов’язаний із користувачем, чи підтверджена пошта та електронна пошта, коли створена та оновлення інформація про користувача.

Таблиці user_metadata зберігається інформація про час останнього входу в аккаунт та IP адреса, з якого було здійснено авторизацію.

Таблиці credentials, містить хешований пароль, та час створення користувача.

Відповідно, структура БД для сервісу api-tariff-service містить таблицю tariff, який зберігає інформацію про тариф, його ID, назва та опис.

У випадку сервісу api-subscription-service використовується інші таблиці, під назвою subscriptions. Дана таблиця, зберігає user_id та tariff_id.

Для створення структури БД, використовується підхід міграції. Цей підхід дозволяє вносити зміни до БД, при цьому, мати можливість здійснювати повернення до попередньої версії структури. Даний підхід корисний у випадку, здійснення системи rollback до системи.

3.3 Налаштування системи Proxmox, для створення віртуальних машин

Для розширення функціоналу реалізований модуль api-proxmox-module, який за допомогою API, виконує операції до Proxmox сервера. На стороні proxmox створені базові шаблони, які виступають в ролі тарифів, та за допомогою викликів, клонують віртуальну машину. Після запуску за допомогою DHCP видається IP адреса, та встановлюється користувач та пароль до сервера.

Для реалізації, з точки зору безпеки, використовується окрема підмережа в Proxmox, яка типу Linux Bridge. Дана підмережа містить адресу 10.0.0.0/18, що дозволяє розмістити велику кількість віртуальних машин.

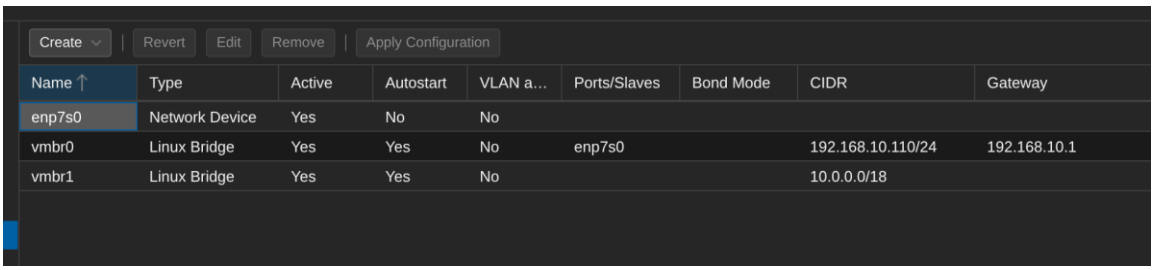


Рисунок 3.3 – Налаштування Linux Bridges для віртуальних мереж

Для створення шаблону використовується образ Ubuntu 24, із форматом диску img. Даний образ займає лише 600 Мб, та має підтримку Cloud Init. Завдяки цьому,

під час клонування віртуальної машини, є можливість налаштувати ОС. Через те, що усі нові віртуальні машини розміщенні на новій підмережі, потрібно налаштувати DHCP сервер, який використовується для автоматичного встановлення IP адреси на серверах. Окрім цього варто звернути увагу на таку особливість як файл який зберігає ID віртуальної адреси. Даний файл розташований за адресою /etc/machine_id, тому для коректної роботи DHCP та отримання унікальної IP адреси, варто очистити вміст файл. Відсутній вміст даного файлу, під час наступного завантаження ОС, автоматично створить machine_id для віртуальної машини. Для налаштування служби, потрібно вказати діапазон IP адрес, які можна використовувати для встановлення IP адрес. Так як використовується 18 підмережа, потрібно було встановити діапазон IP адрес. 10.0.0.1 та 10.0.63.254 в цьому випадку є коректним діапазоном мереж.

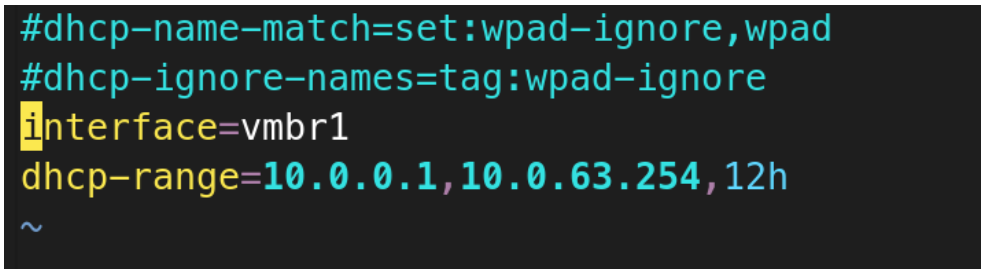


Рисунок 3.4 – Налаштування DHCP сервера

Також варто звернути увагу на те, що для доступу до мережі, варто налаштувати NAT, щоб віртуальні машини мали доступ до мережі інтернет. Для цього виконуємо команди для iptables.

Окрім цих налаштувань є потреба у встановлені qemu агента, який буде визначити IP адресу, яка встановлена на віртуальному сервері (рисунок 3.4). Дана можливість дозволить отримати інформацію по API, та встановити у білінгову систему.

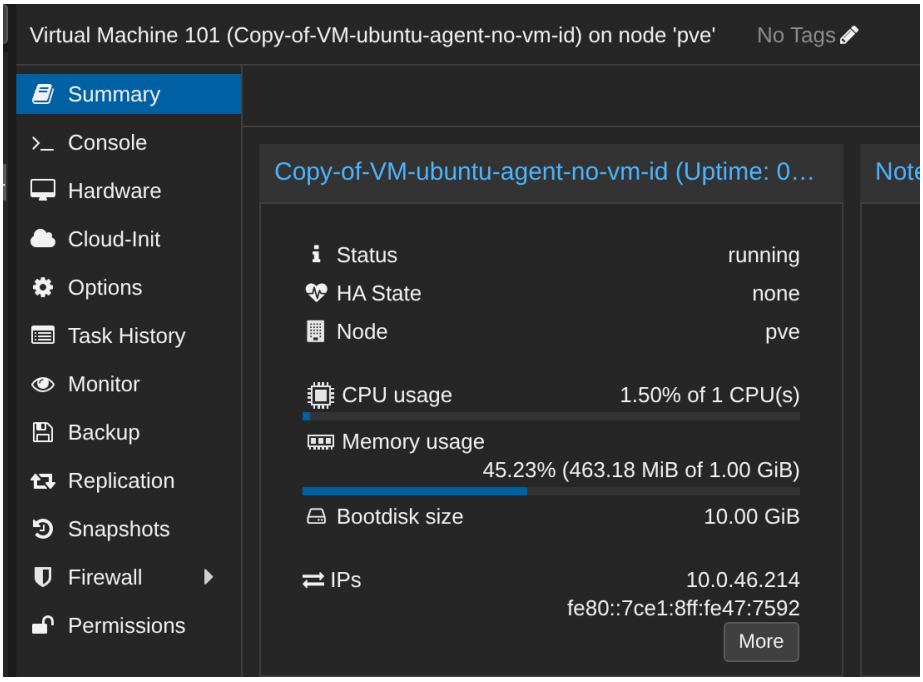
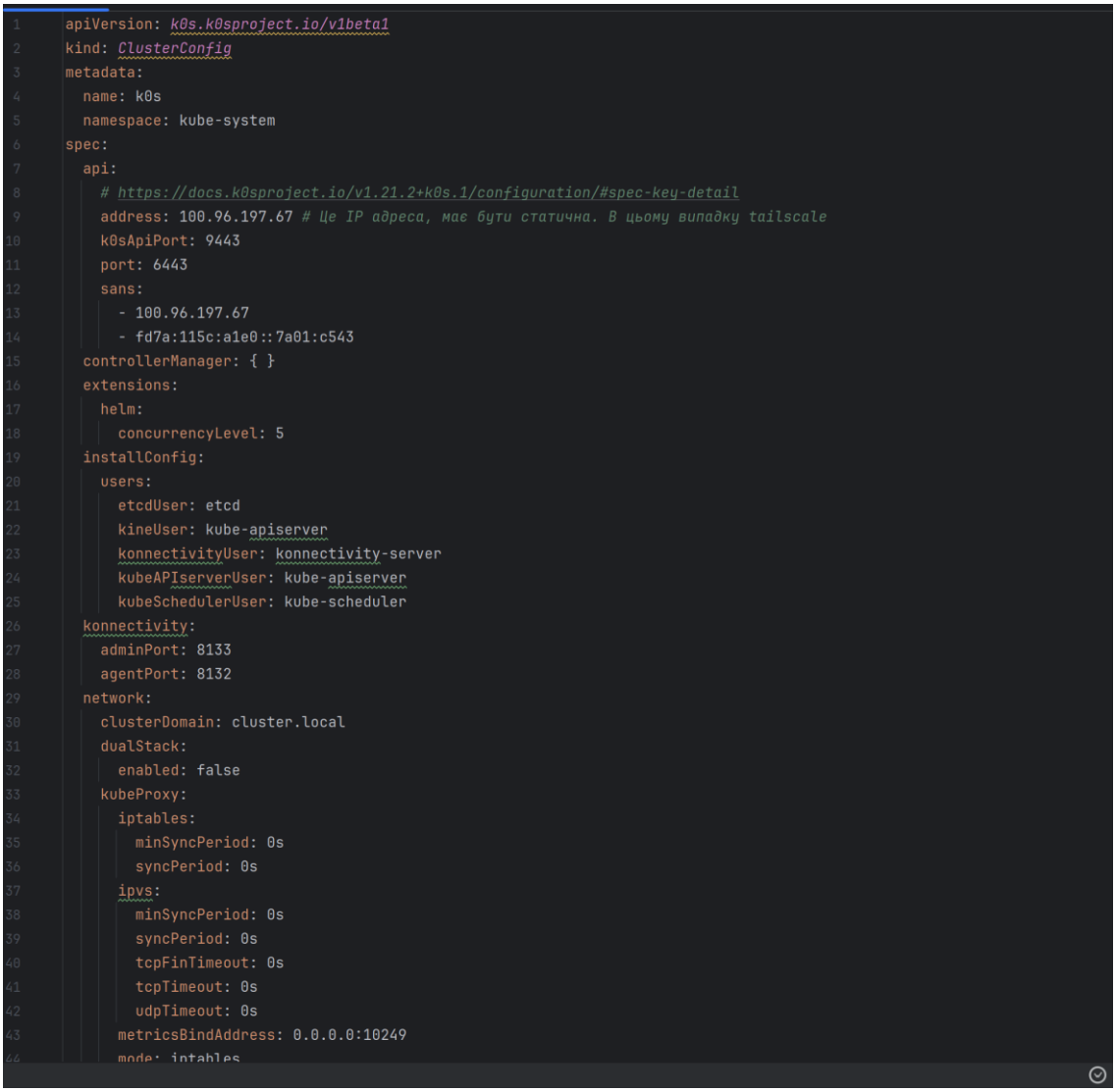


Рисунок 3.5 – Отримана IP адреса для віртуальної машини

Для розміщення програмної частини на апаратну частину Raspberry PI, є потреба у використанні k0s. Для налаштування системи, потрібно встановити IP адресу, яка буде використовуватися для системи. Для цього в параметрі spec.api.sans, прописаний список адрес, які можуть використовуватися для роботи системи. В параметрі spec.api.address пропишемо адресу, яка буде використовувати для доступу до API Kubernetes.

Відповідно дані налаштування дозволяють гнучко налаштовувати систему, ці зміни дозволяють опимізовувати розвертування, та здійснювати відповіді запуски. Це дозволить гнучко розвернути систему, що дозволяє налаштовувати масово ноди за допомогою повноцінно.

Також, дані налаштуваннядозволяють, додатково встановлювати обов’язкові компоненти за допомогою параметра helm, який є не обов’язковим. Даний параметр зручний у випадку, зменшення часу налаштування системи, та збільшення автоматизації загальної системи.



```yaml
apiVersion: k0s.k0sproject.io/v1beta1
kind: ClusterConfig
metadata:
  name: k0s
  namespace: kube-system
spec:
  api:
    # https://docs.k0sproject.io/v1.21.2+k0s.1/configuration/#spec-key-detail
    address: 100.96.197.67 # Це IP адреса, має бути статична. В цьому випадку tailscale
    k0sApiPort: 9443
    port: 6443
    sans:
      - 100.96.197.67
      - fd7a:115c:a1e0::7a01:c543
  controllerManager: { }
  extensions:
    helm:
      concurrencyLevel: 5
  installConfig:
    users:
      etcdUser: etcd
      kineUser: kube-apiserver
      konnectivityUser: konnectivity-server
      kubeAPIserverUser: kube-apiserver
      kubeSchedulerUser: kube-scheduler
  konnectivity:
    adminPort: 8133
    agentPort: 8132
  network:
    clusterDomain: cluster.local
    dualStack:
      enabled: false
    kubeProxy:
      iptables:
        minSyncPeriod: 0s
        syncPeriod: 0s
      ipvs:
        minSyncPeriod: 0s
        syncPeriod: 0s
        tcpFinTimeout: 0s
        tcpTimeout: 0s
        udpTimeout: 0s
      metricsBindAddress: 0.0.0.0:10249
      mode: iptables
```

Рисунок 3.6 – Конфігурація K0s

Після здійснених змін, встановлюємо k0s, в режимі 1 node. Через деякий час, cluster перейде в режим роботи, та можна його повноцінно використовувати.

```
raspberry@raspberrypi:~ $ sudo k0s status
Version: v1.32.4+k0s.0
Process ID: 789
Role: controller
Workloads: true
SingleNode: true
Kube-api probing successful: true
Kube-api probing last error:
raspberry@raspberrypi:~ $
```

Рисунок 3.7 – Перевірка роботи k0s

Для роботи варто встановити rabbimq за допомогою helm. Для кращої безпеки варто використовувати HelmChart від Bitnami, який надає пріоритет безпеці. Також, для роботи варто налаштувати Storage Classes. Для даного завдання буде достатньо використання локального volume. Для цього встановимо проєкт під назвою rancher.io/local-path за допомогою kubectl.

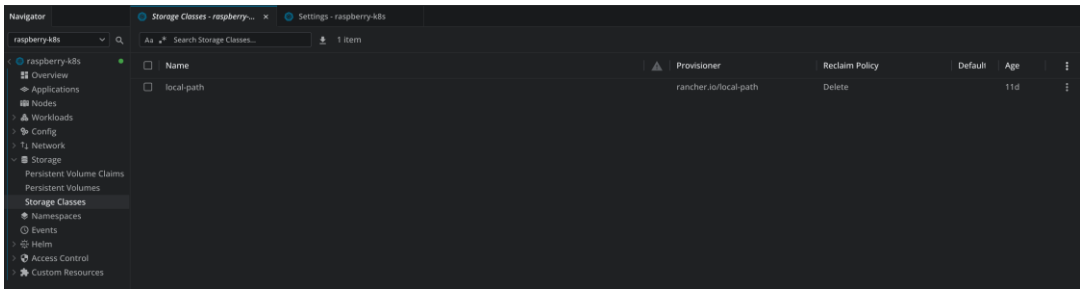


Рисунок 3.8 – Налаштований Storage Classes

Також, для можливості запуску системи рекомендується встановити LoadBalancer. Для цього встановимо ingress за допомогою helm, як helmchart. Враховуючи обмеження роботи кластера в такому режимі, відсутня можливість використовувати зовнішню IP адресу, тому, для усунення даного обмеження було встановлено helmchart під назвою cloudflare/cloudflare-tunnel. Даний проєкт, дозволяє створити тунель надавши глобальний доступ до кластера за межами NAT. Це дозволяє обійти обмеження відсутності зовнішньої IP адреси.

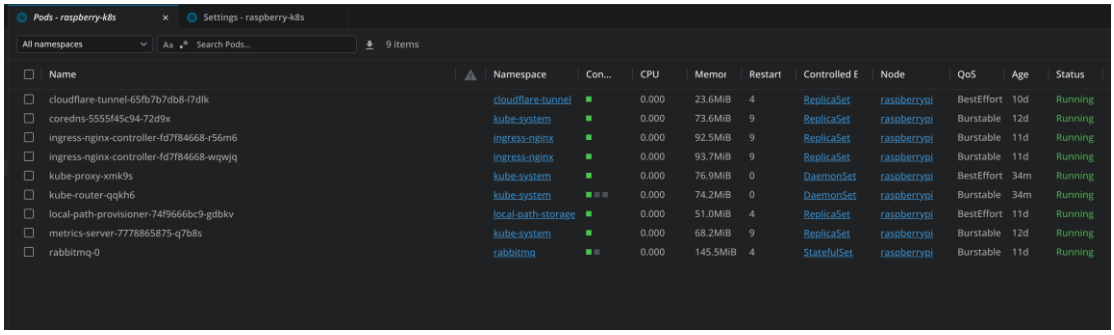


Рисунок 3.9 – Налаштовані компоненти кластеру

Для автоматичного процесу компілювання та запуску коду на cluster k8s використовується підхід із використання CI/CD. Через те, що компоненти системи реалізовані на мові Go, процес CI, складається із наступним складових:

– тестування коду на наявність помилок;

– виконання перевірок в файлах конфігурації Docker;

– створення Docker image за допомогою Dockerfile;

– завантаження DockerImage до репозиторію Docker.

В свою чергу процес CD складається в розвертувані додатку за допомогою helm, як HelmChart додаток.

Відповідно, github workflows складається із jobs, який виконує базову задачу. Тобто, процес складається із етапу створення змінних. Після цього виконується базове тестування Dockerfile, тестування Go на наявність помилок в коді, та тестування unit тестів.

Після виконання базових етапів, виконується процес збірки Docker Images.

```yaml
jobs:
  build:
    steps:
      - uses: docker/setup-qemu-action@v3
      - uses: docker/setup-buildx-action@v3

      - name: Login to GitHub Container Registry
        id: login-repo
        uses: docker/login-action@v3
        with:
          registry: ghcr.io
          username: ${{ github.actor }}
          password: ${{ secrets.GITHUB_TOKEN }}

      - name: Create meta tags
        id: meta
        uses: docker/metadata-action@v5
        with:
          images: ghcr.io/${{ github.repository }}
          tags: |
            type=sha,priority=1000,prefix=sha-,suffix=-app
            type=raw,value=latest,priority=0,suffix=-app,enable={{is_default_branch}}
        env:
          DOCKER_METADATA_SHORT_SHA_LENGTH: 12

      - name: Build and Push Docker Image
        uses: docker/build-push-action@v6
        env:
          DOCKER_BUILD_SUMMARY: true
        with:
          context: .
          platforms: linux/amd64,linux/arm64
          file: Dockerfile
          push: true
          provenance: false
          target: prod
          tags: |
            ${{ steps.meta.outputs.tags }}
          cache-from: type=gha
          cache-to: type=gha,mode=max
```

Рисунок 3.10 - Конфігурація github workflows збірки docker image

3.4 Аналіз обмежень розробленої кіберфізичної системи

Враховуючи, мету досягнення, поставленої задачі, надійність та доступність системи, швидкість роботи, варто визначити недоліки даної системи та наявні обмеження. Одним із основних є використання системи продажу віртуальних послуг. Враховуючи складову системи, є наступні обмеження - для роботи системи є потрібна встановлення Docker на системі, де використовується. Це обмеження зумовлене тим, що є усі додатки запущені в Docker Image. Враховуючи архітектуру, яка використовується для роботи системи, є обмеження в arm64 та amd64, це обмеження зумовлене обмеженням компіляції та перевірки системи.

Ще одним із фактором надійності системи, є можливість горизонтального масштабування. Завдяки цієї функції, яка є функціонал Kubernetes, це дозволить масштабуватися в залежності від навантаження.

Використання rabbitmq дозволяє збільшити надійність втрати даних. Ця можливість надає можливість бути впевненим, що дані з одного сервісу не будуть успішно синхронізовані в інших сервісах.

Обмеження системи, яка присутня, є в системи модульності, так як складно розширяти функціонал, створюючи нову логіку. Обмеження наявні в тому, що не має можливості створити новий функціонал, через обмеження мікросервісної архітектури.

Також із недоліків варто вказати, що використання великої кількості компонентів призводить до збільшення витрат, що відповідає збільшенням фінансових витрат.

Для поглибленого аналізу системи, було перевірено, як буде вести система при запиту на створення системи. У межах аналізу, система успішно створює віртуальну машину, та дозволяє отримати доступ до системи.

Також, варто зазначити, через обмеження ресурсів на proxmox, у випадку якщо вони будуть закінчені, система не буде мати змогу активізувати нові віртуальні машини.

Окрім цього особливостей поведінки системи важко проаналізувати роботоспособність системи у випадку великого навантаження системи. Це зумовлене відсутності потужних обчислень.

У випадку економічних витрат, можна підрахувати загальні витрати на обчислення системи.

3.5 Висновки

У третьому розділі дипломної роботи створено кіберфізичну системи, яка складається з апаратної та програмної складової, для адміністрування ресурсів віртуальних машин на Proxmox. Основна ідея була зосереджена на використання сучасних компонентів, використання сучасних інструментів, методів використання сучасних патернів та підходів. Було розроблено систему, яка дозволяє взаємодіяти із віртуальними машина, створювати користувачів, створювати тарифи та активувати послуги. Окрім цього створена можливість видалення та оновлення інформація. Розроблена система, із підтримкою модулів, яка дозволяє розширяти функціонал наявної системи не впливаючи на зміни в коді в наявних сервісів. Було проаналізовано фінансові витрати системи.

Також було розроблено бібліотеки, які дозволяють відловлювати event, від rabbitmq, та виконувати відповідну функцію, завдяки цьому вдалося розділити логіку на частини. Це прискорило використання.

Було налаштовано Proxmox, для створення віртуальних машин, налаштування роботи DHCP для видачі IP адрес, налаштування віртуальних машин за допомогою Cloud Init, налаштування автоматизації CI/CD та налаштування helmchart.

ВИСНОВОК

У роботі було розроблено кіберфізичну систему білінгу та управління ресурсами для віртуальних серверів, побудовану на Raspberry PI. Було використано сучасний підхід використання на мікросервісах, взаємодію за допомогою GRPC, .використання брокеру повідомлень та налаштування віртуальних машин Proxmox.

У процесі розробки було здійснено аналіз сучасних технологій, для створення сучасної системи мікросервісів. В результату аналізів було обрано протокол GRPC для спілкування між мікросервісами та взаємодією по API.

Після вибору програмної складової, було вибрано Raspberri PI, як основний компонент запуску цієї системи. Як основному операційну систему, було образу Debian 11, та систему оркестрації K0S. Вибрана архітектура була обрана гнучкою та масштабованою, яка дозволяє розширяти систему за допомогою системи модулів.

Під час роботи було побудовано взаємодію системи, перевірено функціонал роботоспособності системи, та інтеграцію із Proxmox кластером.

В результаті роботи було розроблено проєкт, для створення взаємодії віртуальних системи, управління ресурсами з можливістю розширення функціоналу.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Kubernetes. URL: https://kubernetes.io/docs/home/ (дата звернення: 12.05.25)
2. Docker. URL: https://docs.docker.com/engine/ (дата звернення: 12.05.25)
3. Режим роботи Docker Swarm. URL: https://docs.docker.com/engine/swarm/ (дата звернення: 13.05.25)
4. REST API як спосіб спілкування компонент веб-додатків. URL: https://foxminded.ua/shcho-take-rest-api/(дата звернення: 13.05.25)
5.
6. Proxmox. URL: https://www.proxmox.com/en/ (дата звернення: 13.05.25)
7. GRPC. URL: https://grpc.io/ (дата звернення: 13.05.25)
8. k0s - The Zero Friction Kubernetes. URL: https://docs.k0sproject.io/stable/ (дата звернення: 13.05.25)
9. What is K3s? Режим доступу: https://docs.k3s.io/ (дата звернення: 13.05.25)
10. RabbitMQ. One broker to queue them all. URL: https://www.rabbitmq.com/ (дата звернення: 13.05.25)
11. PostgreSQL: The World's Most Advanced Open Source Relational Database. URL: https://www.postgresql.org/ (дата звернення: 13.05.25)
12. How to build microservices. URL: https://www.atlassian.com/microservices/microservices-architecture/building-microservices (дата звернення: 13.05.25)
13. Microservice Architecture pattern. URL: https://microservices.io/patterns/microservices.html (дата звернення: 13.05.25)
14. Get started with the Model Context Protocol (MCP). URL: https://modelcontextprotocol.io/introduction (дата звернення: 13.05.25)
15. Send API requests and get response data in Postman. URL: https://learning.postman.com/docs/sending-requests/requests/ (дата звернення: 13.05.25)

16. Production-Grade Container Orchestration. URL: https://kubernetes.io/ (дата звернення: 13.05.25)

17. Financial infrastructure to grow your revenue. URL: https://stripe.com/ (дата звернення: 14.05.25)

18. An online API documentation with examples so you can start building web apps with Fiber right away! URL: https://docs.gofiber.io/ (дата звернення: 14.05.25)

19. JSON Web Tokens. URL: https://jwt.io/ (дата звернення: 14.05.25)

20. Basics tutorial | Go | gRPC. URL: https://grpc.io/docs/languages/go/basics/ (дата звернення: 15.05.25)

21. Argon2. URL: https://argo-cd.readthedocs.io/en/stable/ (дата звернення: 20.05.25)

22. Lovisa Johansson. The RabbitMQ Management Interface. URL: https://www.cloudamqp.com/blog/part3-rabbitmq-for-beginners_the-management-interface.html (дата звернення: 01.06.2025)

23. The library for web and native user interfaces. URL: https://react.dev/ (дата звернення: 22.05.25)

24. Nginx. URL: https://nginx.org/ (дата звернення: 23.05.25)

25. Static Assets. URL: https://developers.cloudflare.com/workers/static-assets/ (дата звернення: 24.05.25)

26. Amazon CloudFront. URL: https://aws.amazon.com/cloudfront/ (дата звернення: 24.05.25)

27. Go (Golang) GOOS and GOARCH. URL: https://gist.github.com/danielalvarenga/58cead013d9303ed84d2fdf0ac157fc9 (дата звернення: 24.05.25)

28. What is RabbitMQ? URL: https://seventhstate.io/what-is-rabbitmq/ (дата звернення: 24.05.25)

29. What is GraphQL? What is Difference? URL: https://www.khmer168.com/dbms/what-is-graphql/ (дата звернення: 24.05.25)

30. What Is a REST API? Examples, Uses, and Challenges. URL: https://blog.postman.com/rest-api-examples/ (дата звернення: 24.05.25)

31. Github репозиторій RabbitMQ. URL: https://github.com/rabbitmq/rabbitmq-server (дата звернення: 24.05.25)

32. Github репозиторій PostgreSQL. URL: https://github.com/postgres/postgres (дата звернення: 24.05.25)

33. Getting started with your Raspberry Pi. URL: https://www.raspberrypi.com/documentation/ computers/getting-started.html (дата звернення: 24.05.25)

34. Helm Chart. URL: https://helm.sh/docs/ (дата звернення: 24.05.25)

35. OpenCell. URL: https://opencellsoft.com/platform/modules/ (дата звернення: 25.05.25)

36. jBilling Overview. URL: https://www.appdirect.com/jbilling (дата звернення: 25.05.25)

37. FreeSide. URL: http://freeside.biz/freeside/ (дата звернення: 25.05.25)

38. Kill Bill. URL: https://killbill.io/ (дата звернення: 25.05.25)

39. Github репозиторій Kill Bill. URL: https://github.com/killbill/killbill (25.05.25)

40. Debian. URL: https://www.debian.org/index.en.html (дата звернення: 26.05.25)

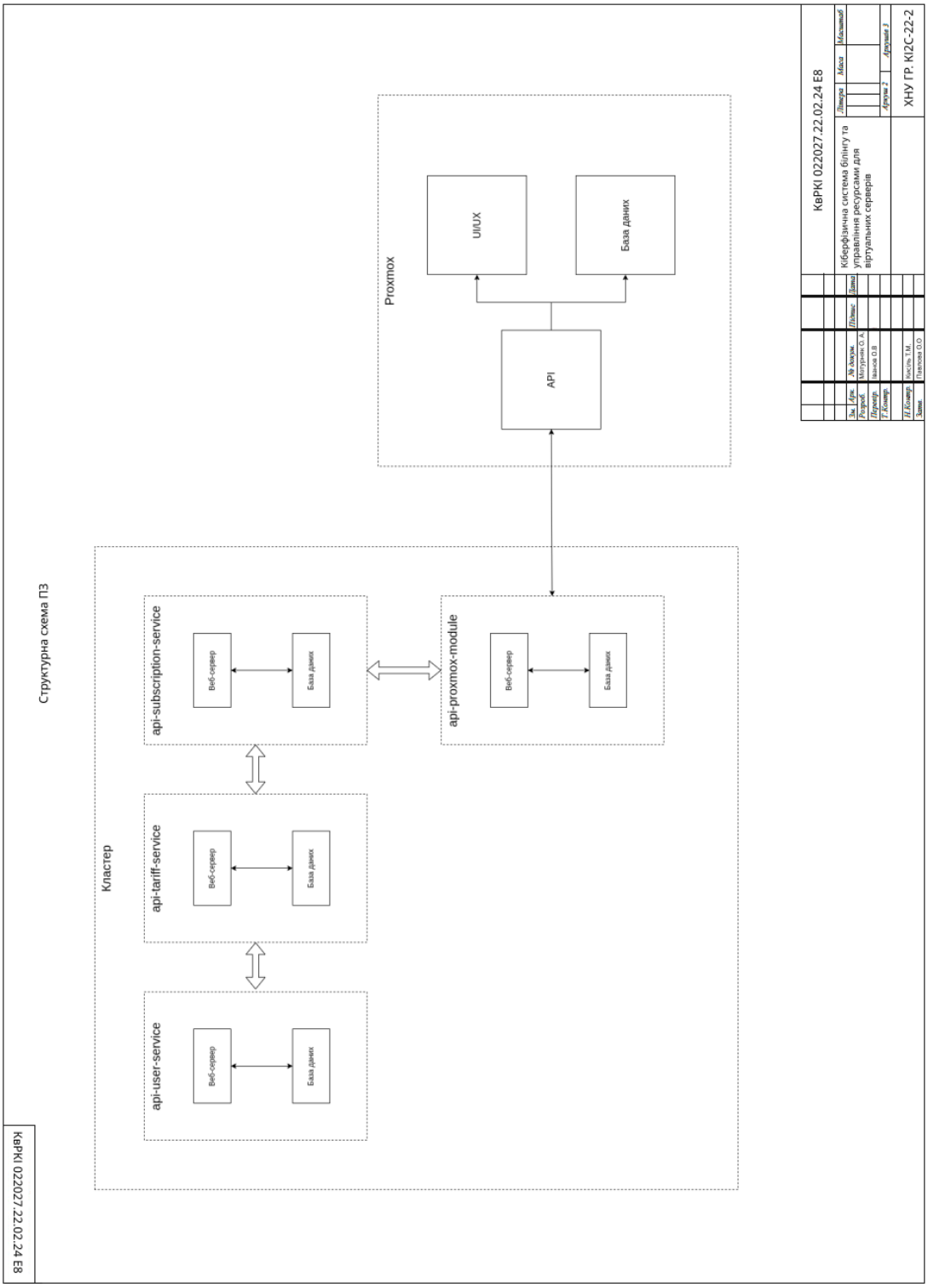
41. Nigel Poulton. The Kubernetes Book. 2025. No Starch Press. p. 248

42. Fx is a dependency injection system for Go. URL: https://github.com/uber-go/fx (дата звернення: 21.05.25)

Додаток А

(обов’язковий)

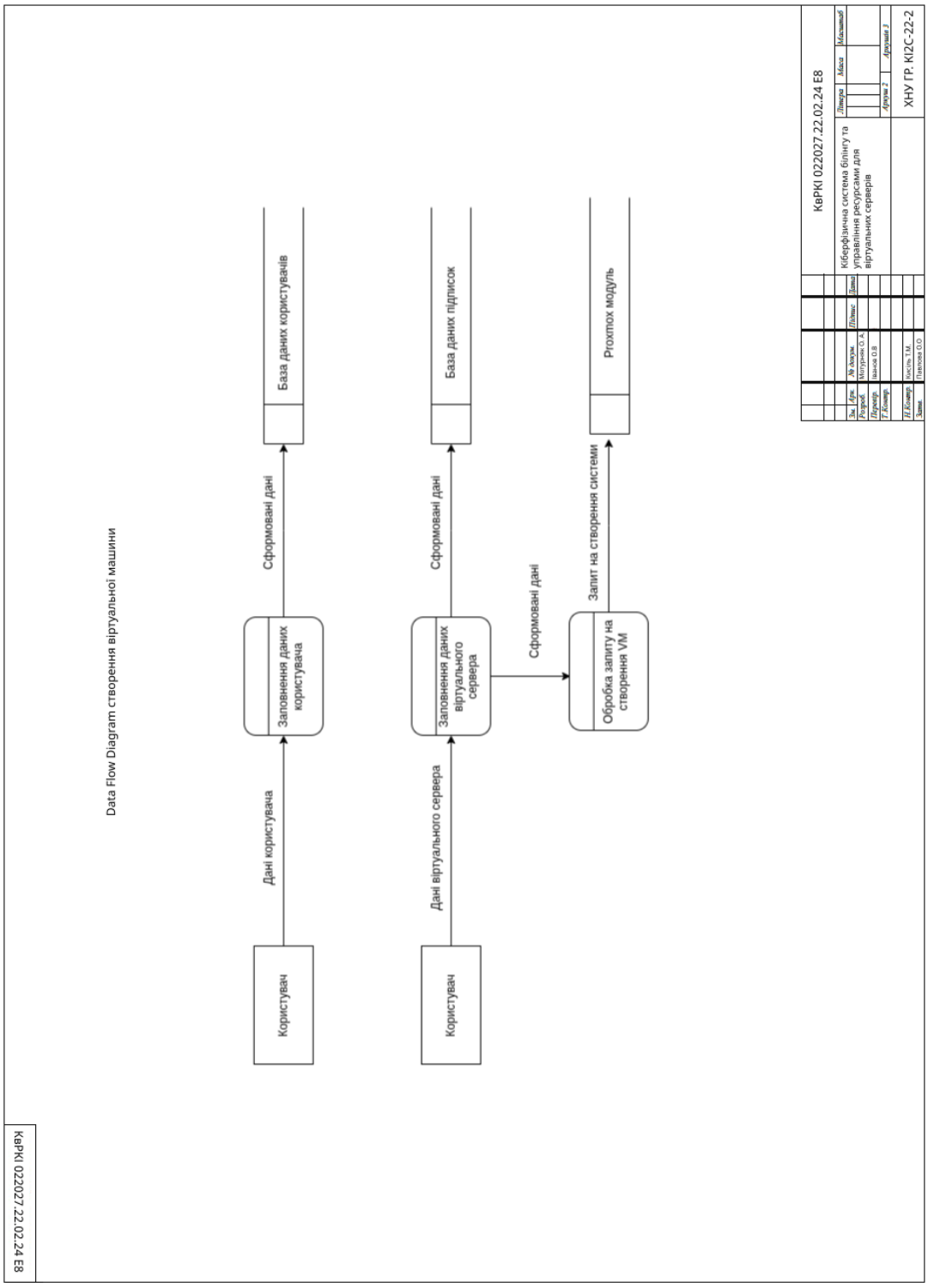
КОПІЯ КРЕСЛЕННЯ «СТРУКТУРНА СХЕМА ПЗ ПРОЄКТУ»



Додаток Б

(обов’язковий)

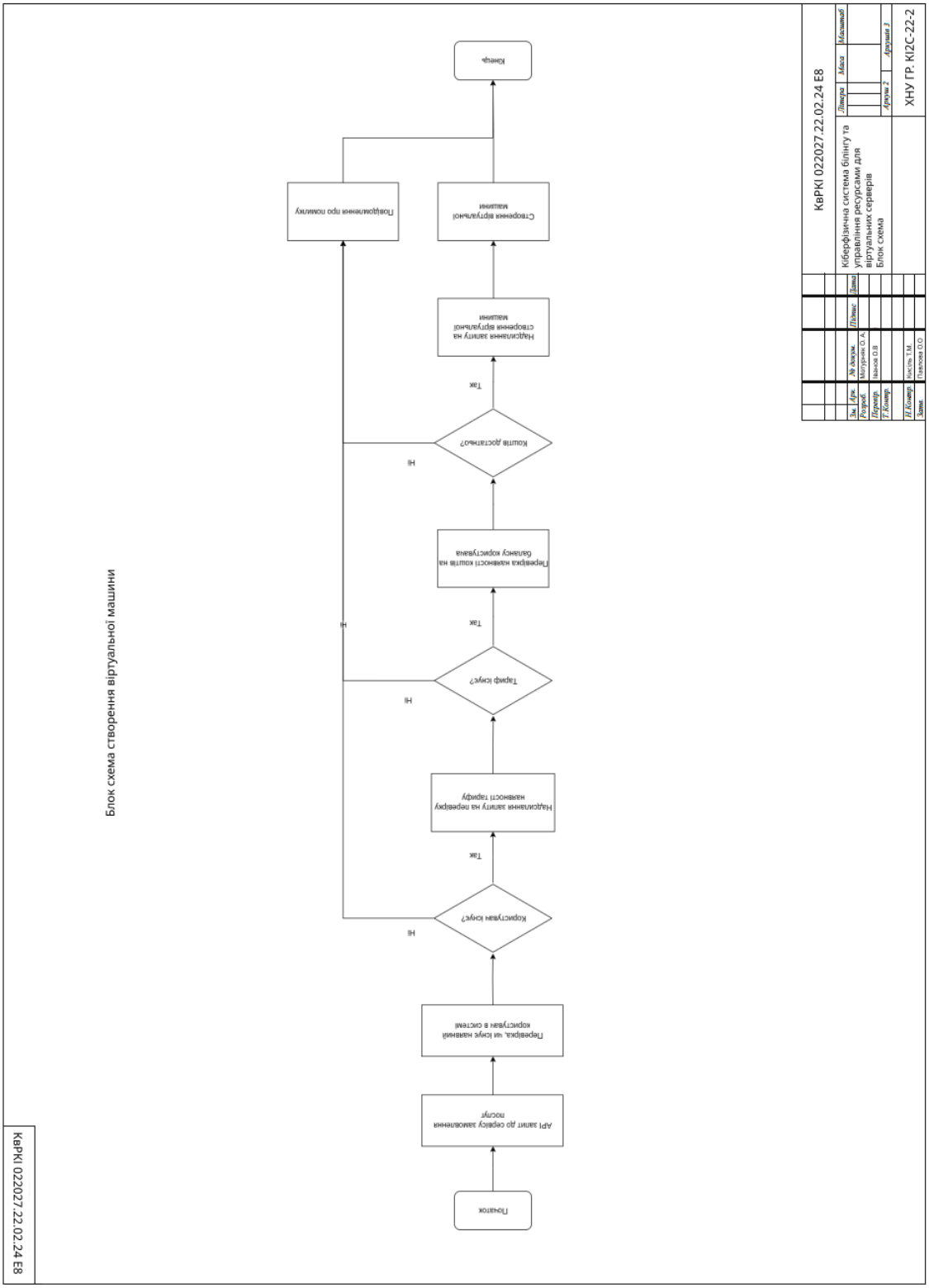
КОПІЯ КРЕСЛЕННЯ «DATA FLOW DIAGRAM ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ»



Додаток В

(обов’язковий)

КОПІЯ КРЕСЛЕННЯ «БЛОК СХЕМА КРЕСЛЕННЯ»

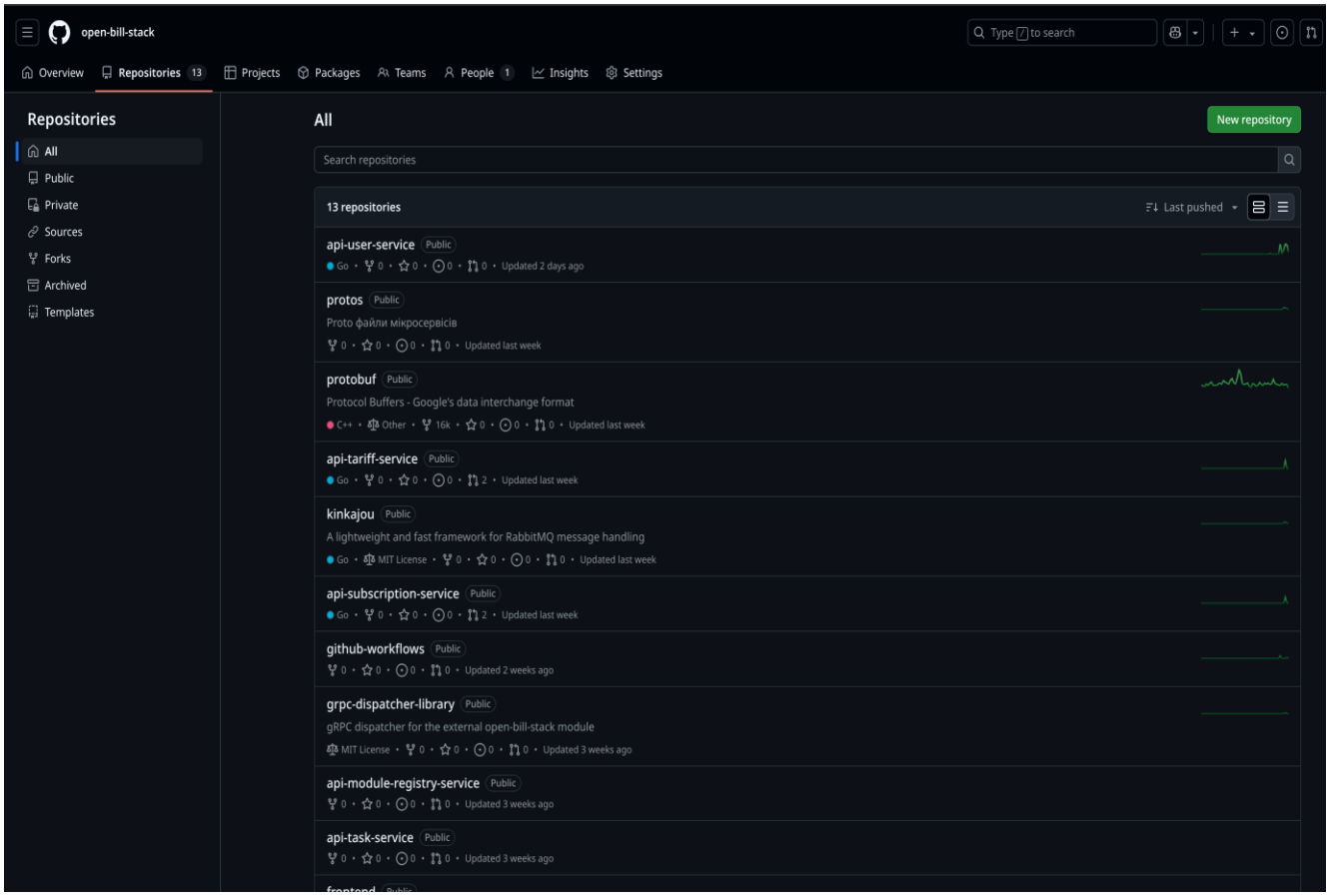


Додаток Г

(обов’язковий)

ВИХІДНИЙ КОД

Вихідний програмний код, що використаний у розробці, є відкритим для загального доступу та доступним у відкритих репозиторіях GitHub: https://github.com/orgs/open-bill-stack/repositories



Репозиторії містять вихідний кожного із компонентів, які взаємодіють між собою за допомогою новітніх програмних методів

Протокол аналізу звіту подібності експертом

Заявляю, що я ознайомився (-лась) з Повним звітом подібності, який був згенерований Системою виявлення і запобігання плагіату щодо роботи:

Автор: Олексій МОТУРНЯК

Співавтор:

Назва: Мотурняк_Кіберфізична система білінгу та управління ресурсами для віртуальних серверів

Експерт:

Підрозділ: Кафедра комп'ютерної інженерії та інформаційних систем

Коефіцієнт подібності 1:1.3%

Коефіцієнт подібності 2:0.3%

Мікропробіли: 6

Заміна букв: 5

Інтервали: 0

Білі знаки: 0

Дата створення звіту: 2025-06-06 03:25:14.0

Після аналізу Звіту подібності констатую наступне:

☒ **Запозичення, виявлені в роботі є законними і не є плагіатом. Рівень подібності не перевищує допустимої межі. Таким чином робота незалежна і приймається.**

☐ **Запозичення не є плагіатом, але перевищено граничне значення рівня подібностей. Таким чином робота повертається на доопрацювання.**

☐ **Виявлено запозичення і плагіат або навмисні текстові спотворення (маніпуляції), як передбачувані спроби укриття плагіату, які роблять роботу невідповідною вимогам законодавства (Ст. 32. ЗУ Про вищу освіту, пункт 3.1, Ст. 42. ЗУ Про освіту) та вимог НАЗЯВО (Критерій 5), а також кодексу етики і процедурам. Таким чином робота не приймається.**

Обгрунтування:

2025-06-06 Доцент Андрій Нічепорук

Дата експерт

Thu Jun 05 18:00:35 EEST 2025, Медзатий Дмитро Миколайович, Хмельницький національний університет, ХНУ

Anti-Plagiarism (UA) v-15.281 Educational

The maximum coincidence with one document 9.0%

Dictionaries check: en_US, ru_RU, ua_UA. **Errors in the documents: 9%**

	Document		Sum coincidence on the DB	
ID: 243752 Title: БКР Кіберфізична система білінгу та управління ресурсами для віртуальних серверів Added in a DB: 2025-06-05 Authors: Олексій МОТУРНЯК Heads: Олексій ІВАНОВ Consultants: Opponents:	Symbols	Lexemes	Symbols	Lexemes
	81961	738	7950 (10%)	97 (13%)

Plagiarism sources

ID	Description	Plagiarism presence in the document	
		Symbols	Lexemes
240845	Title: Звіт з ПДП Розроблення кіберфізичної система білінгу та управління ресурсами для віртуальних серверів Added in a DB: 2025-05-05 Authors: Мотурняк О.А. Heads: Павлова О.О. Consultants: Opponents:	7050 (9.0%)	85 (12.0%)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

Дипломник: Мотурняк Олексій Андрійович

Тема: Кіберфізична система білінгу та управління ресурсами для віртуальних серверів

Спеціальність**:** 123 «Комп'ютерна інженерія»

Обсяг кваліфікаційної роботи:

Кількість листів креслень ___3____ Кількість сторінок записки ___60_____

1. Короткий зміст роботи та прийнятих рішень: Метою кваліфікаційної роботи є створення системи білінгу та управління ресурсами для віртуальних серверів

2. Висновок про відповідність роботи дипломному завданню: Робота повністю відповідає поставленому завданню.

3. Характеристика виконання кожного розділу, ступінь використання останніх досягнень науки і техніки і передових методів роботи: У першому розділі кваліфікаційної роботи проведено комплексне дослідження предметної області, пов'язаної з інспекцією аналогів систем білінгу. Зокрема, виконано аналіз сучасних кіберфізичних систем збору даних, розглянуто специфіку Kubernetes як засобу запуску системи, визначено критерії вибору програмних засобів, здійснено огляд сучасних систем та наведено порівняльну характеристику. Також досліджено методи розробки кіберфізичних систем. Це дозволило закласти теоретичну основу для формування архітектури системи та методів автоматизованого аналізу.

У другому розділі роботи розроблено архітектуру кіберфізичної системи білінгу та управління ресурсами для віртуальних серверів. Розглянуто метод розробки програмного продукту аналізу апаратної складової. Визначено апаратні компоненти системи, описано типи апаратних складових і методи аналізу критичності виявлених дефектів із використанням елементів нечіткої логіки. Також розглянуто використання програмного забезпечення Proxmox для запуску віртуальних серверів, описано

алгоритми створення та керування систем. У розділі активно використано сучасні технічні рішення, включаючи побудова мікросервісної архітектури.

У третьому розділі здійснено реалізацію прототипу програмно-апаратної системи та проведено її тестування. Описано архітектуру розробленого рішення, структуру програмного забезпечення та склад апаратного комплексу. Проведено експериментальні дослідження, у ході яких порівняно ефективність системи, та здійснено відповідні налаштування.. Висновки підкріплено кількісними оцінками та прикладами результатів роботи прототипу, що демонструє високий ступінь практичної реалізації теоретичних рішень.

4. Позитивні сторони роботи: Робота відзначається високим рівнем практичної реалізації та використанням сучасних технологій. Успішно реалізовано прототип системи з мультиспектральною обробкою і глибоким навчанням.

5. Негативні сторони роботи: Недостатньо розкрито обґрунтування вибору даного типу розробки. Аспекти кібербезпеки системи залишилися поза увагою.

6. Оцінка графічного оформлення та пояснювальної записки роботи: Пояснювальна записка оформлена коректно, згідно діючих стандартів оформлення документації.

7. Відгук про роботу в цілому: Робота виконана на належному науково-технічному рівні.

8. Інші зауваження: немає

9. Оцінка дипломної роботи: добре

Рецензент (прізвище, ім'я, по батькові, посада, місце роботи) Праворська Наталія Іванівна, доцент кафедри ТПЗ, к. пед. наук

"04" червня 2025 р. (підпис)

Завідувачу кафедри КІІС
д-р. філософії, доц. Ользі ПАВЛОВІЙ

Олексій МОТУРНЯКА[ПІБ здобувача вищої освіти]
ФІТ, 3 курсу, групи КІ2с-22-2

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності у Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту та застосування заходів дисциплінарної та академічної відповідальності, ознайомлений (а). Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений(а) та надаю свою згоду на обробку та збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Strike-Plagiarism та Anti-Plagiarism) та використання роботи для виявлення плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

05 06 2025 року

РІШЕННЯ ЕКСПЕРТНОЇ КОМІСІЇ

КАФЕДРИ КОМП'ЮТЕРНОЇ ІНЖЕНЕРІЇ ТА ІНФОРМАЦІЙНИХ СИСТЕМ

ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатом звіту подібності щодо роботи, генерованого системою виявлення текстових збігів/ідентичності/схожості:

Назва: Кіберфізична система білінгу та управління ресурсами для віртуальних серверів

Автор: Олексій Мотурняк

Спеціальність: 123- Комп'ютерна інженерія

Освітня програма: освітньо-професійна

Науковий керівник: Іванов Олексій Валентинович к.т.н., доцент

Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	відповідає
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої та електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) Запозичення розміщені в розділах аналізу існуючих аналогів та прототипів, які не описують безпосередньо авторське дослідження і не стосуються результатів роботи.;
2) усі запозичення фрагментарні, або мають належним чином оформленні посилання;
3) Окремі збіги представлені загальновживаними фразами, наприклад: «на рисунку зображено», «загальна структура системи», «висновки до розділу» тощо.
4) Якість запозичень відповідає технічним особливостям дослідження: виявлено збіги в кодах, формулах і термінах, які є вихідними даними до великої кількості задач і не можуть вважатися авторськими порушеннями.
5) Система зафіксувала технічні модифікації тексту, зокрема: заміну окремих символів ,скорочення індексів у формулах, зміну розміщення символів. Це є наслідком форматування або експорту документа, а не цілеспрямованого уникнення перевірки.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів/ ідентичності/схожості StrikePlagiarism, складає 1.34% і адресується до 14 першоджерела; та системою Anti-Plagiarism складає 0.21%, що, з урахуванням наведених обгрунтувань, відповідає характеру наукового дослідження і свідчить на користь кваліфікаційної роботи.

Керівник роботи _____ О ІВАНОВ

Гарант ОП _____ Андрій Нічепорук

Завідувач кафедри КІІС _____ Ольга ПАВЛОВА