МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА МАГІСТРА

на тему: **«РОЗРОБКА КЛІЄНТСЬКОГО ВЕБЗАСТОСУНКУ У СФЕРІ БАНКІВСЬКИХ ПОСЛУГ ЗАСОБАМИ NESTJS»**

Виконав: студент 2 курсу, групи 8.1213-іпз-1
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми інженерія програмного забезпечення
(назва освітньої програми)
В.Е. Гугля
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, PhD, Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент завідувач кафедри фундаментальної та прикладної математики, професор, д.т.н. Гребенюк С.М.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти магістр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2024 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Гуглі Владиславу Едуардовичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Розробка клієнтського вебзастосунку у сфері банківських послуг засобами NestJS

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 01 » березня 2024 року № 385-с

2. Строк подання студентом роботи 25.11.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та реалізація програмного забезпечення.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 07.03.2024 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	18.03.2024	
2.	Збір вихідних даних.	22.04.2024	
3.	Обробка методичних та теоретичних	10.06.2024	
	джерел.		
4.	Розробка першого та другого розділу.	30.08.2024	
5.	Розробка третього розділу.	04.11.2024	
6.	Оформлення та нормоконтроль	20.11.2024	
	кваліфікаційної роботи магістра.		
7.	Захист кваліфікаційної роботи.	11.12.2024	

Студент _____ В.Е. Гугля
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота магістра «Розробка клієнтського вебзастосунку у сфері банківських послуг засобами NestJS»: 79 с., 85 рис., 13 табл., 17 джерел.

AWS S3, DOCKER, ELASTICSEARCH, FIREBASE, NESTJS, NODE.JS, POSTGRESQL, SUMSUB, TYPEORM, TYPESCRIPT.

Об’єкт дослідження – fintech вебзастосунок.

Предмет дослідження – розробка fintech вебзастосунку у сфері банківських послуг.

Методи дослідження – аналітичний, проєктувальний, програмний.

Мета дослідження – розробити швидкий та багатофункціональний fintech вебзастосунок для проведення і менеджменту різноманітних типів транзакцій.

У сучасному світі стрімкого розвитку інформаційних технологій банківська сфера є однією з ключових галузей, де цифровізація відіграє вирішальну роль. Впровадження інноваційних вебзастосунків дозволяє фінансовим установам покращувати якість обслуговування клієнтів, автоматизувати бізнес-процеси та забезпечувати високий рівень безпеки транзакцій.

Проєкт присвячений розробці бізнес-логіки вебпорталу з використанням рантайму Node.js та фреймворку для написання серверної частини вебдодатків NestJS. Для досягнення швидкодії застосунку і зручності користування, була зроблена інтеграція з реляційною базою даних PostgreSQL, Elasticsearch, Redis, Firebase, і багатьма іншими ресурсами.

Результатом роботи є зручний, щвидкий, та багатофункціональний вебзастосунок, який має високий рівень безпеки та захисту користувацьких даних.

SUMMARY

Master's qualifying paper "Development of a Client Web Application to the Banking Services by NestJS": 79 p., 85 figures, 13 tables, 17 references.

AWS S3, DOCKER, ELASTICSEARCH, FIREBASE, NESTJS, NODE.JS, POSTGRESQL, SUMSUB, TYPEORM, TYPESCRIPT.

The object of research is a fintech web application.

The subject of the research is the development of a fintech web application in the field of banking services.

Research methods: analytical, design, and programming.

The purpose of the research is to develop a fast and multifunctional fintech web application for conducting and managing various types of transactions.

In today's world of rapidly developing information technology, the banking sector is one of the key industries where digitalization plays a crucial role. The introduction of innovative web applications allows financial institutions to improve the quality of customer service, automate business processes and ensure a high level of transaction security.

The project is devoted to the development of business logic for the web portal using the Node.js framework and the NestJS framework for writing the server side of web applications. To achieve application performance and ease of use, integration with PostgreSQL, Elasticsearch, Redis, Firebase, and many other resources was made.

The result is a convenient, fast, and multifunctional web application that has a high level of security and protection of user data.

ЗМІСТ

Завдання на кваліфікаційну роботу ……..2
Реферат 4
Summary 5
Вступ..... 7
1 Аналіз вимог ПЗ 9
1.1 Опис проєкту 9
1.2 Use-cases..... 11
1.3 User stories..... 12
2 Проєктування ПЗ..... 19
2.1 База даних 19
2.1.1 Короткий опис сутностей..... 22
2.1.2 Загальний опис зв’язків..... 23
2.1.3 ER-діаграма..... 24
2.1.4 Таблиці 26
2.2 Діаграма інфраструктури 35
2.3 Опис технологій реалізації..... 39
3 Реалізація ПЗ..... 49
3.1 Реалізація основного функціоналу..... 49
3.2 Контейнеризація і деплой у хмару 68
Висновки 76
Перелік посилань..... 78

ВСТУП

У сучасному світі стрімкого розвитку інформаційних технологій банківська сфера є однією з ключових галузей, де цифровізація відіграє вирішальну роль. Впровадження інноваційних вебзастосунків дозволяє фінансовим установам покращувати якість обслуговування клієнтів, автоматизувати бізнес-процеси та забезпечувати високий рівень безпеки транзакцій.

Мета дослідження – розробити швидкий та багатофункціональний fintech вебзастосунок для проведення і менеджменту різноманітних типів транзакцій.

Досягнення поставленої мети передбачає реалізацію таких **завдань**:

- провести аналіз різних варіацій реалізації діджиталізованих fintech-рішень;
- проаналізувати вимоги до фінансових вебзастосунків;
- спроєктувати архітектуру системи (бази даних, інфраструктури, классів, і т.д.);
- реалізувати всі задумані функціональні ідеї;
- протестувати розроблений функціонал.

Об’єкт дослідження – fintech вебзастосунок.

Предмет дослідження – розробка fintech вебзастосунку у сфері банківських послуг засобами NestJS.

Для досягнення результату було обрано мову TypeScript задля суворої типизації, яка відсутня в чистому Javascript; фреймворк NestJS, оскільки він дозволяє будувати і масштабувати серверні частини вебдодатків легко через його модульну структуру, якої він змушує притримуватись розробників фреймворк NestJS, а також за багатий вибір вбудованих інтеграцій. Для зберігання інформації використовувалася СКБД PostgreSQL, так як мова програмування SQL дозволяє легко і швидко робити варіативні маніпуляції з

датою, і реляційна природа дозволяє нормалізувати структуру бази даних без втрати можливості робити різні варіативні записи.

Результатом роботи є вебзастосунок, що забезпечує щвидкодію, безпечність, зручність та багатофункціональність задуманого функціоналу з можливістю легкого потенційного масштабування і апгрейд іншими технологіями і ресурсами.

Основними функціональними вимогами до банківського вебдодатку сайту є створення швидкого та безпечного серверного застосунку, ефективної та безперебійної роботи синхронізації банківських аккаунтів, безпеки та надійності операцій, впровадження процесів аутентифікації і авторизації для запобігання несанкціонованого доступу деяких модулів порталу, а також можливості виконувати різних типів фінансових транзакцій.

Кваліфікаційна робота складається зі вступу, трьох розділів, висновків та переліку посилань. Загальний обсяг роботи становить 79 сторінок. У переліку посилань 17 використаних джерел.

Перший розділ присвячений аналізу вимог ПЗ, загальному огляді функціональності проєкту, технологій, задіяних у розробці, а також потенційним use-кейсам та основним user-історіям, яке дасть розуміння, для яких сутностей створювався додаток і як які його функції можуть використовувати користувачі. У другому розділі наведено ER-діаграму бази даних, детальний опис і призначення таблиць, побудованих для додатку, діаграму інфраструктури для розуміння, з чого складається в цілому інфраструктура проєкту, і короткий теоретичний опис технологій, які були задіяні в розробці. В третьому розділі описано реалізацію функціоналу з додатковими рисунками до кожного блоку окремого кейсу використання для кращого розуміння.

1 АНАЛІЗ ВИМОГ ПЗ

1.1 Опис проєкту

У сучасному світі, популярність діджиталізованих fintech додатків (наприклад, так званий «банк в телефоні») дуже висока і вона продовжує набрати обертів.

Для ілюстрації цікаво подивитися на цифри з таких країн, як Данія чи Норвегія. Майже всі люди в цих країнах мають доступ до банківських рахунків через мобільний додаток (91% у Данії та 95% у Норвегії у 2019 році) [16]. Це означає, що споживачі потребують таких послуг, а фінансові установи реагують на попит. З іншого боку, конкуренція у фінтех-бізнесі в країнах з високим відсотком населення, яке вже користується мобільними банківськими додатками, буде жорсткою [16].

Найбільш відчутні вигоди від інвестицій у розробку мобільних банківських додатків можна побачити в таких країнах, як Швеція (84%), Великобританія (78%), Швейцарія (73%), Бельгія (71%), Люксембург (71%), Ірландія (67%), Франція (66%), Австрія (63%), Німеччина (61%), а також Естонія (81%), Латвія (72%), Литва (65%) та Чехія (68%), які мають добре розвинену фінтех-інфраструктуру та вільний простір для створення абсолютно нових фінансових додатків [16].

Наприклад, кількість фізичних осіб, які користуються інтернет-банкінгом у Великій Британії, була високою весь час з 2010 року до сьогодні і становить близько 80%. З 2015 року вона зростає на 5% на рік. Це означає, що мобільні додатки дуже популярні у Великобританії, і все більше споживачів приєднаються до цієї тенденції [16].

Розглянемо **основні переваги** онлайн банкінгу.

Відчуття контролю. Сучасні рішення для мобільного банкінгу дозволяють клієнтам перевіряти баланс свого рахунку або історію транзакцій

у будь-який час і в будь-якому місці. Крім того, вони часто надають персоналізовану фінансову інформацію та інструменти для бюджетування, щоб допомогти користувачам краще управляти своїми фінансами та досягати своїх цілей щодо заощаджень [17].

Сплачуйте рахунки вчасно. Багато людей турбуються про те, щоб вчасно сплачувати щомісячні рахунки. Однак з мобільним банкінгом це більше не є проблемою. Додатки мобільного банкінгу можуть надсилати користувачам сповіщення про наближення термінів оплати і навіть дозволяють налаштувати автоматичні періодичні платежі, що гарантує своєчасну оплату рахунків без пропущених термінів [17].

Фінансова освіта. Використання мобільних банківських додатків може сприяти кращому розумінню фінансових продуктів та послуг. Відповідаючи на запитання про переваги мобільного банкінгу, 38% американців відповіли, що відчувають себе більш обізнаними про послуги, які пропонують їхні банки [17].

Якщо говорити за розроблений fintech проєкт, про який йде мова тут, то це можна назвати проксі-додатком для менеджменту ваших банківських аккаунтів через вебдодаток. Розберемо **основні** його **функції**.

Виконання різних інтернаціональних транзакцій. Користувач має широкий вибір для виконання та менеджменту різних типів транзакцій, такі як: Internal (транзакції виключно між аккаунтами які існують в рамках розробленої системи), SEPA (ініціатива, спрямована на спрощення безготівкових платежів та забезпечення узгодженості транзакцій всередині країн-членів ЄС та між ними), SWIFT (велика мережа обміну повідомленнями, яка використовується фінансовими установами для швидкого, точного та безпечного надсилання та отримання інформації, наприклад, інструкцій про грошові перекази), UKFP (банківська ініціатива Сполученого Королівства, спрямована на скорочення часу проведення платежів між клієнтськими рахунками різних банків до кількох секунд, порівняно з трьома робочими днями, які зазвичай займають перекази за допомогою давно усталеної системи BACS).

Реєстрація та аутентифікація користувачів: користувачі зможуть створювати облікові записи, входити в систему, налаштовувати двохфакторну аутентифікацію для більшого захисту свого облікового запису.

Створення та синхронізація банківських рахунків. Користувачі зможуть створити записи як банківські рахунки у системі і синхронізувати їх з їх реальними прототипами з будь-якого банку. Після створення і синхронізації аккаунту, адміністратор на адмін панелі зможе перевірити аккаунт на анти-шахрайську діяльність, і у випадку чого, відмінити активацію аккаунту у системі.

Створення та синхронізація банківських рахунків: користувачі зможуть створити записи як банківські рахунки у системі і синхронізувати їх з реальними банківськими аккаунтами.

Створення і перегляд квитанцій транзакцій. Користувач має змогу продивитись всю інформацію про транзакцію у зручному PDF – форматі, з можливістю майбутнього перегляду.

Створення і перегляд інвойсів історії балансу аккаунтів: користувачі можуть отримати персоналізовану тенденцію зміни певного їхнього банківського аккаунту протягом якогось проміжку часу.

1.2 Use-cases

Виконуючи кваліфікаційну роботу, замислювалися наступні use-cases.

ФОП-підприємець. Вебдодаток дозволяє ФОП-підприємцям автоматизувати фінансові операції, скорочувати адміністративні витрати та зосередитися на розвитку бізнесу, а не на управлінні фінансами вручну.

Великий та малий бізнеси. Fintech-додатки стають корисними як для малих, так і для великих бізнесів, оскільки спрощують управління фінансами, автоматизують процеси й надають нові можливості для зростання та оптимізації витрат.

Клієнт банку. Клієнт якогось банку шукає діджиталізоване рішення для менеджменту своїх банківських аккаунтів. Користувачі можуть надсилати гроші своїм друзям чи родичам через застосунок, використовуючи лише номер телефону або електронну пошту одержувача. Також є можливість здійснювати міжнародні перекази без великих комісій.

1.3 User stories

На рисунках 1.1 – 1.25 представлено опис бізнес-логіки fintech-додатку порталу у вигляді user-stories.

Як користувач, я можу створити свій особистий аккаунт на порталі (реєстрація).
Design [15]:
https://www.figma.com/design/ME7xItHqxIhX8sg9RoberU/Diplom-(magistration-degree)?node-id=190-3100&node-type=frame&t=qfqgdgktqgTNwftl-0
Given – Користувач заходить на сайт і відкриває форму реєстрації.
And – Користувач вводить валідні данні у всі обов'язкові поля для створення аккаунту (Імя, Прізвище, Email, Пароль, Телефон, Адресу).
When – Користувач клікає на кнопку «Sign Up».
Then – Дані юзера заносяться в базу даних; тепер користувач може авторизуватись, використовуючи новостворені email і пароль.

Рисунок 1.1 – Registration user-story [15]

Як користувач, я можу увійти в свій особистий аккаунт (аутентифікація).
Design [15]:
https://www.figma.com/file/G9CPuFwDVUSPHUq9NGbEQp/Diplom?type=design&node-id=1-581&t=y3b2qmWXiFg45KO5-4
Given – Користувач заходить на сайт і відкриває форму авторизації.
And – Користувач вводить валідні данні у всі обов'язкові поля для авторизації (Email, Пароль).
When – Користувач клікає на кнопку «Sign In».
Then – Юзер проходить процес ідентифікації і авторизації і може користуватися своїм особистим аккаунтом та іншими функціями порталу.

Рисунок 1.2 – Authentication user-story [15]

Як користувач, я можу створити банківський аккаунт, синхронізований з банківським рахунком.

Design:

https://www.figma.com/file/G9CPuFwDVUSPHUq9NGbEQp/Diplom?type=design&node-id=1-1309&t=QTAaLzQ0w2MHdBci-4

Given – Користувач заходить на портал.

And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.

And – Користувач заходить на форму створення банківського аккаунту.

When – Користувач вводить банківські деталі свого рахунку (IBAN, BIC, і т.д.) і клікає на кнопку «ОК».

Then – Портал створює новий запис в базі і виводить правильну інформацію про новостворений аккаунт.

Рисунок 1.3 – “Create bank account” user-story

Як користувач, я можу провести фінансову транзакцію.

Given – Користувач заходить на портал.

And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.

And – Користувач відкриває форму будь-якої з доступних транзакцій.

When – Користувач вводить необхідні для транзакції дані (банківські деталі отримувача, сума трансферу, і т.д) і натискає на кнопку «Зробити трансфер».

Then – Портал успішно створює транзакцію в базі, відправляє на етап перевірки транзакції адміністратору, і підключається до пеймент провайдера для можливого трансферу коштів.

Рисунок 1.4 – “Create transaction” user-story

Як користувач, я можу відслідковувати статус транзакцій і їх деталі.

Design:

https://www.figma.com/file/G9CPuFwDVUSPHUq9NGbEQp/Diplom?type=design&node-id=1-1206&t=QTAaLzQ0w2MHdBci-4

Given – Користувач заходить на портал.

And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.

And – Користувач відкриває будь-який банківський аккаунт, з якого була зроблена транзакція.

When – Користувач відкриває вікно деталей транзакції.

Then – Портал відображає всі деталі транзакції, куди входить банківські деталі отримувача, поточний статус транзакції, можливий коментар, і т.д.

Рисунок 1.5 – “Get transactions” user-story

Як користувач, я можу оновити мої персональні дані в кабінеті.
Given – Користувач заходить на портал.
And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.
And – Користувач переходить на вкладку «Personal Info».
When – Користувач змінює будь який пункт з персональної інформації (Ім'я, Прізвище, Адресу, email, телефон, і т.д.) і клікає на кнопку «ОК».
Then – Портал оновлює адресу у базі даних і відображає оновлену інформацію у персональному кабінеті.

Рисунок 1.6 – "Update user" user-story

Як користувач, я можу видалити банківський аккаунт.
Given – Користувач заходить на портал.
And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.
And – Користувач обирає будь-який банківський аккаунт створений ним.
When – Користувач натискає на кнопку «Delete» і підтверджує свій вибір.
Then – Портал видаляє аккаунт з системи і показує оновлений поточний стан аккаунтів користувача.

Рисунок 1.7 – "Delete bank account " user-story

Як користувач, я можу змінити пароль свого користувацького аккаунту.
Given – Користувач заходить на портал.
And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.
And – Користувач відкриває вікно профілю і обирає опцію «Змінити пароль».
When – Користувач вводить правильний старий пароль і новий пароль (який відповідає правилам оформлення паролів в системі) і натискає на кнопку «ОК».
Then – Портал успішно змінює пароль, і наступний раз, коли користувач намагатимется аутентифікуватися, він має ввести новий пароль.

Рисунок 1.8 – "Change password" user-story

Як користувач, я можу вимкнути двох-факторну аутентифікацію.
Given – Користувач заходить на портал.
And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.
When – Користувач відкриває вікно профілю і деактивує чекбокс «Включити двох-факторну аутентифікацію».
Then – Портал вимикає двох-факторну аутентифікацію в системі і відображає повідомлення, що двох-факторна аутентифікація деактивована.

Рисунок 1.9 – "2FA disable" user-story

Як користувач, я можу ввімкнути двох-факторну аутентифікацію.
Given – Користувач заходить на портал.
And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.
And – Користувач відкриває вікно профілю і активує чекбокс «Включити двох-факторну аутентифікацію».
When – Користувач сканує QR-код і вводить валідний код з будь-якого аутентифікатор додатку.
Then – Портал вмикає двох-факторну аутентифікацію в системі і відображає повідомлення, що двох-факторна аутентифікація активна.

Рисунок 1.10 – “2FA ebable” user-story

Як користувач, я можу аутентифікуватись з увімкненою двохфакторною аутентифікацією з правильно введеним кодом.
Given – Користувач заходить на стартову сторінку сайту.
And – Користувач вводить валідні дані логіну (email або телефон) і паролю.
When – Користувач вводить правильний аутентифікаційний код у полі для введення 2FA коду.
Then – Портал валідує користувача і пускає на головну сторінку порталу.

Рисунок 1.11 – “2FA auth (enabled)” user-story

Як користувач, я не можу аутентифікуватись з увімкненою двохфакторною аутентифікацією з неправильно введеним кодом.
Given – Користувач заходить на стартову сторінку сайту.
And – Користувач вводить валідні дані логіну (email або телефон) і паролю.
When – Користувач вводить неправильний аутентифікаційний код у полі для введення 2FA коду.
Then – Портал не валідує користувача і не пускає на основний портал.

Рисунок 1.12 – “2FA auth (disabled)” user-story

Як користувач, я можу отримати квитанцію до певної транзакції.
Given – Користувач заходить на портал.
And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.
And – Користувач відкриває будь-яку транзакцію з будь-якого аккаунту.
When – Користувач натискає на кнопку «Отримати квитанцію».
Then – Портал генерує квитанцію у вигляді PDF файлі і сповіщає користувача про це. Квитанція містить усі деталі транзакції, включно з банківськими деталями відправника і отримувача, також суми транзакції, комісії, і т.д.

Рисунок 1.13 – “Create receipt” user-story

Як користувач, я можу зберегти шаблон якоїсь транзакції у вигляді бенефіціара.

Given – Користувач заходить на портал.

And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.

When – Користувач робить будь-яку транзакцію.

Then – Портал додає нового отримувача як бенефіціара до списку з його повним ім'ям, типом транзакції, та банківськими деталями.

Рисунок 1.14 – "Create beneficiary" user-story

Як користувач, я можу використати шаблон бенефіціара для майбутніх транзакцій.

Given – Користувач заходить на портал.

And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.

When – Користувач обирає будь-якого бенефіціара.

Then – Портал перенаправляє користувача на сторінку транзакції. Тим і деталі транзакції вже автоматично заповнені і співпадають з тими що були у бенефіціара.

Рисунок 1.15 – "Get beneficiary" user-story

Як користувач, я можу відсортувати бенефіціарів за різними параметрами.

Given – Користувач заходить на портал.

And – Користувач проходить процес аутентифікації і відкриває особистий кабінет.

And – Користувач обирає опцію будь-яку опцію сортування бенефіціарів (наприклад, за датою).

When – Користувач натискає кнопку «Знайти».

Then – Портал показує список бенефіціарів користувача у правильно відсортованому порядку за обраним параметром.

Рисунок 1.16 – " Get list of beneficiaries" user-story

Як адміністратор, я можу зареєструвати іншого адміністратора у адмін-панелі.

Given – Адміністратор заходить на портал.

And – Адміністратор проходить процес аутентифікації і відкриває особистий кабінет.

And – Адміністратор переходить на вкладку «Адміністратори».

And – Адміністратор натискає на кнопку «Додати нового адміністратора».

When – Адміністратор вводить дані про нового адміністратора і натискає на кнопку «Додати».

Then – Портал створює новий обліковий запис адміністратора у системі.

Рисунок 1.17 – "Register admin" user-story

Як адміністратор, я можу увійти в систему (аутентифікація).

Given – Адміністратор заходить на сайт адмін панелі і відкриває форму авторизації.
And – Адміністратор вводить валідні данні у всі обов'язкові поля для авторизації (Email, Пароль).
When – Користувач клікає на кнопку «Sign In».
Then – Портал валідує користувача і пускає на головну сторінку порталу.

Рисунок 1.18 – "Admin auth" user-story

Як адміністратор, я можу продивитися інформацію про будь якого користувача.

Given – Адміністратор заходить на сайт адмін панелі і відкриває форму авторизації.
And – Адміністратор проходить процес аутентифікації і відкриває адмін панель.
When – Адміністратор відкриває вкладку «Користувачі».
Then – Портал показує повний список зареєстрованих користувачів на порталі. Адміністратор може продивитись якогось індивідуального користувача, його персональну інформацію, банківські аккаунти і транзакції, зроблені ним.

Рисунок 1.19 – "Get users" user-story

Як адміністратор, я можу продивитись користувацькі банківські аккаунти у статусі «На розгляді» і змінити їх статус.

Given – Адміністратор заходить на сайт адмін панелі і відкриває форму авторизації.
And – Адміністратор проходить процес аутентифікації і відкриває адмін панель
When – Адміністратор відкриває вкладку «Аккаунти на розгляді».
Then – Портал показує повний список користувацьких банківських аккаунтів з статусом «На розгляді». Адміністратор може переглянути кожний із них, і змінити його статус на «Затверджено» або «Відхилено» і вказати причину відмови, якщо така є.

Рисунок 1.20 – "Get pending bank accounts" user-story

Як адміністратор, я можу продивитись користувацькі фінансові транзакції у статусі «На розгляді» і змінити їх статус.

Given – Адміністратор заходить на сайт адмін панелі і відкриває форму авторизації.
And – Адміністратор проходить процес аутентифікації і відкриває адмін панель.
When – Адміністратор відкриває вкладку «Транзакції на розгляді».
Then – Портал показує повний список користувацьких фінансових транзакцій з статусом «На розгляді». Адміністратор може переглянути кожний із них, і змінити його статус на «Затверджено» або «Відхилено» і вказати причину відмови, якщо така є.

Рисунок 1.21 – "Get pending transactions" user-story

Як адміністратор, я можу створювати/оновлювати комісійні правила.

Given – Адміністратор заходить на сайт адмін панелі і відкриває форму авторизації.

And – Адміністратор проходить процес аутентифікації і відкриває адмін панель.

And – Адміністратор відкриває вкладку «Комісійні правила».

When – Адміністратор вводить/оновлює дані певого комісійного правила.

Then – Портал створює/оновлює комісійне правило.

Рисунок 1.22 – “Create/update fee rules” user-story

Як адміністратор, я можу створювати новий аккаунт для комісій.

Given – Адміністратор заходить на сайт адмін панелі і відкриває форму авторизації.

And – Адміністратор проходить процес аутентифікації і відкриває адмін панель.

And – Адміністратор відкриває вкладку «Аккаунти для комісій».

And – Адміністратор натискає на кнопку «Створити новий аккаунт».

When – Адміністратор вводить дані аккаунта для комісій.

Then – Портал створює новий аккаунт для комісій, на який буде приходити всі комісії з усіх транзакцій певної валюти (EUR, USD, GBP).

Рисунок 1.23 – “Create/update fee accounts” user-story

Як адміністратор, я можу видаляти аккаунт для комісій.

Given – Адміністратор заходить на сайт адмін панелі і відкриває форму авторизації.

And – Адміністратор проходить процес аутентифікації і відкриває адмін панель.

And – Адміністратор відкриває вкладку «Аккаунти для комісій».

And – Адміністратор обирає будь який аккаунт.

When – Адміністратор натискає на кнопку «Видалити».

Then – Портал видаляє аккаунт.

Рисунок 1.24 – “Delete fee account” user-story

Як адміністратор, я можу переглядати комісійні транзакції.

Given – Адміністратор заходить на сайт адмін панелі і відкриває форму авторизації.

And – Адміністратор проходить процес аутентифікації і відкриває адмін панель.

When – Адміністратор відкриває вкладку «Комісійні транзакції».

Then – Портал показує всі комісійні транзакції, які виникли в результаті фінансових транзакцій.

Рисунок 1.25 – “Get fee transactions” user-story

2 ПРОЄКТУВАННЯ ПЗ

2.1 База даних

В цьому розділі ми обговоримо призначення баз даних в fintech додатках, основні переваги для вибору реляційних СКБД для банківських застосунків, а також, яку роль вони відіграють в поточному проєкті.

Провівши детальний аналіз, було визначено, що більшість fintech додатків використовують за основне сховище даних саме реляційні СКБД, такі як PostgreSQL, MySQL, MariaDB, Timescale, та інші. Чому це так? Ось декілька причин, чому реляційні БД обираються як основну базу даних для фінансових вебзастосунків:

– структуровані дані: ідеально підходять для додатків з чітко визначеною схемою та взаємозв’язками, таких як фінансові транзакції;
– відповідність стандарту ACID: забезпечує цілісність та узгодженість даних, що має вирішальне значення для фінансових додатків;
– розвинена екосистема: надійні інструменти та підтримка складних запитів, звітності та аналітики;
– відповідність нормативним вимогам: часто легше відповідати вимогам нормативних актів завдяки потужним функціям цілісності даних.

Найпоширеніші use-кейси для реляційних БД такі:

– коли транзакції вимагають високої узгодженості та цілісності (наприклад, банківські системи);
– для додатків, які потребують складних запитів та об’єднань;
– якщо у вас є чітко визначена модель даних, яка навряд чи буде часто змінюватися.

За основну СКБД для проєкту було обрано саме PostgreSQL [6]. Ось деякі переваги, які пояснюють причину цього вибору:

– структуровані дані: ідеально підходить для додатків з чітко визначеними даними – PostgreSQL пропонує високу розширюваність, що дозволяє визначати власні типи даних, створювати користувацькі функції та інтегрувати код з різних мов програмування без перекомпіляції вашої бази даних;
– PostgreSQL підтримує багатоверсійний контроль паралельності (MVCC) для керування паралельними транзакціями та зменшення кількості тупикових ситуацій;
– розширені заходи безпеки включають шифрування SSL, режими автентифікації та визначення привілеїв користувачів;
– зовнішні обгортки даних: можливість PostgreSQL, яка дозволяє з’єднуватися з іншими джерелами даних, використовуючи стандартну схему інтерфейсу SQL і взаємозв’язки, наприклад, фінансові транзакції.

Останнім часом для великих fintech-додатків набирають популярність і NoSQL (нереляційні) СКБД, такі, як MongoDB, Redis, Elasticsearch, CouchDB та інші. Ось деякі переваги які схиляють до вибору саме такого типу БД, як основну:

– масштабованість: може обробляти великі обсяги неструктурованих або напівструктурованих даних, що робить їх придатними для додатків з великими даними;
– гнучкість: дизайн без схем дозволяє швидку розробку та ітерації, а також приймати будь-які структуровані і напівструктуровані дані без їх втрати через невідповідність визначеній схемі, як це буває з реляційними БД;
– висока пропускна здатність: часто оптимізовані для високошвидкісних операцій читання і запису, що може бути корисним для аналітики в режимі реального часу.

Найпоширеніші use-кейси для NoSQL БД такі:

– при роботі з великими обсягами різноманітних даних (наприклад, аналітика поведінки користувачів);
– якщо додаток вимагає швидкого масштабування для адаптації до зростання;
– для проєктів, які передбачають високошвидкісне отримання та обробку даних;
– в поточному проєкті використовуються як допоміжні до основної СКБД три NoSQL СКБД – Elasticsearch, Redis та Firebase.

Elasticsearch – це розподілена пошуково-аналітична система з відкритим вихідним кодом і автономна база даних, призначена для обробки великих обсягів даних з можливістю пошуку в режимі, близькому до реального часу. В поточному проєкті використовується як інструмент для повнотекстового пошуку з можливістю толерування людських похибок для шаблонів транзакцій.

Redis (REmote DIctionary Server) – це сховище ключів/значень NoSQL з відкритим вихідним кодом у пам’яті, яке використовується в основному як кеш додатків або база даних швидкого реагування. В проєкті він використовується як месседж брокер для проведення бекграунд робіт через Node.js бібліотеку BullMQ. Також використовується, як шар кешування для зберігання та швидкого доступу інформації про активні браузерні сесії користувачів, які використовують вебдодаток на поточний момент, щоб у разі потреби, сервер знав, по яким ID сокетів пушити повідомлення на клієнтську (фронтенд) частину.

Firebase Realtime Database (Firebase) – це хмарна NoSQL база даних, яка синхронізує та зберігає дані між користувачами в режимі реального часу. Популярна feature гілка цієї БД – Firebase Cloud Messaging (FCM). Firebase Cloud Messaging (FCM) – це крос-платформенне рішення для обміну повідомленнями, яке дозволяє надійно надсилати повідомлення безкоштовно. В поточному проєкті використовується як засіб для швидких та надійних server-push нотифікацій, інтеграцію з якими легко налаштовувати як на

клієнтській частині (frontent), так і на серверній (backend); також, у випадку переривання роботи мережі або збою серверу чи клієнту, повідомлення відправленні через FCM, не втрачаються, як у випадку з вебсокетами, а зберігається у хмарному сховищі для таких повідомлень, і коли додаток повністю відновлює свою роботу і роботу мережі, Firebase автоматично доставляє всі необхідні повідомлення.

У даному розділі буде короткий опис сутностей БД, їх зв’язки між собою, деякі додаткові обґрунтування для не дуже очевидних рішень, які можуть здатися людям, які не були причетні для розробки даного проєкту. В кінці розділу буде наведений рисунок ER-діаграми для кращої візуалізації системи таблиць.

2.1.1 Короткий опис сутностей

База даних системи складається з 13 сутностей – Users, Payment Accounts, Transactions, Fund Ledgers, Beneficiaries, Addresses, Countries, Fee Accounts, Fee Rules, Fee Transactions, Admins, Receipts та Invoices.

Users – таблиця, яка містить інформацію про основних користувачів, які використовують вебдодаток за призначенням.

Payment Accounts – таблиця банківських рахунків користувачів з усіма даними, необхідними для проведення транзакцій.

Transactions – таблиця фінансових транзакцій, зроблених через банківські аккаунти користувачів.

Fund Ledgers – таблиця бухгалтерського обліку (бухгалтерських транзакцій) для певного банківського рахунку. Необхідна для легкого та ефективного відслідковування історії зміни балансу та перегляд транзакцій якогось аккаунту через поля “directionType” (напрямок транзакції) та поля “pitAccountsBefore” та “pitAccountsAfter”, які відображають стан актуального балансу перед виконанням транзакції та після, якщо транзакція буде успішна.

Beneficiaries – таблиця шаблонів транзакцій, містить всю інформацію про банківські деталі отримувача однієї з попередніх виконаних транзакції.

Addresses – таблиця адреси користувача. Адреса була нормалізована в окрему таблицю для більш легких і гнучких запитів у базу.

Countries – таблиця доступних для використання країн. Містять назву країни та її ISO-2 та ISO-3 коди.

Fee Accounts – таблиця банківських аккаунтів для збору комісій. Можна вважати це як накопичувальний рахунок для комісій

Fee Rules – таблиця правил розрахунку комісій для різних комбінацій типу транзакції і валюти, в якій транзакції проводиться.

Fee Transactions – таблиця комісійних транзакцій, які йдуть у відповідний банківський аккаунт для комісій.

Admins – таблиця адміністраторів, які керують додатком через адмін-панель.

Receipts – таблиця для зберігання квитанцій окремих бухгалтерських транзакцій, яка містить поле з посиланням на відповідний файл у хмарному сховищі S3 задля уникнення повторної ресурсоємної операції генерації PDF-файлу.

2.1.2 Загальний опис зв’язків

Довідка щодо позначень:

– One-to-One зв’язок – 1:1;

– One-to-Many зв’язок – 1:M;

– Many-to-Many зв’язок – M:M.

Користувачі можуть мати багато банківських рахунків, з різними валютами для різних платежів, проте один і той же банківський аккаунт може належати тільки одному користувачу (1:M). Також користувачі при реєстрації створюють свою унікальну адресу, яка належить тільки їм (1:1).

Кожний запис адреси користувача може створюватись в рамках тільки однієї країни, і в рамках однієї і тієї ж країни може бути створено багато адрес (M:1). Банківські аккаунти можуть мати багато фінансових та бухгалтерських транзакцій, але одна й та сама фінансова або бухгалтерська транзакція може відноситись тільки до одного аккаунту (1:M). Країна, до якої прив'язаний певний банківських аккаунт, може бути тільки одна, і є багато банківських аккаунтів можуть бутии прив'язані до однієї і тієї ж країни (M:1). Також банківський аккаунт може мати багато інвойсів, кожен з яких відображає стан балансу аккаунту в певний проміжок часу (1:M). Для кожної створеної фінансової транзакції може створитися дві бухгалтерські транзакції: одна для відправника, друга – для отримувача; одна бухгалтерська транзакція може відноситись тільки до однієї фінансової транзакції (1:M).

Для кожної бухгалтерської транзакції може існувати тільки одна квитанція (1:1). Для кожної фінансової транзакції створюється одна комісійна транзакція, якщо цьому передують умови (1:1).

Банківський аккаунт для збору комісій може мати багато комісійних транзакцій, але одна й та ж комісійна транзакція відноситься тільки до одного комісійного аккаунту (1:M). Користувач може мати в управлінні багато шаблонів для майбутніх транзакцій, але один і той же шаблон може відноситись тільки до одного користувача (1:M).

У шаблону транзакції в якості банківських деталей отримувача може бути референс на запис банківського аккаунту; до одного і того ж банківського аккаунту може бути більше ніж один референс від різних шаблонів транзакцій різних користувачів (M:1)

2.1.3 ER-діаграма

У ході проєктування бази даних, була сконструйована така ER-діаграма (див. рис. 2.1) [1].

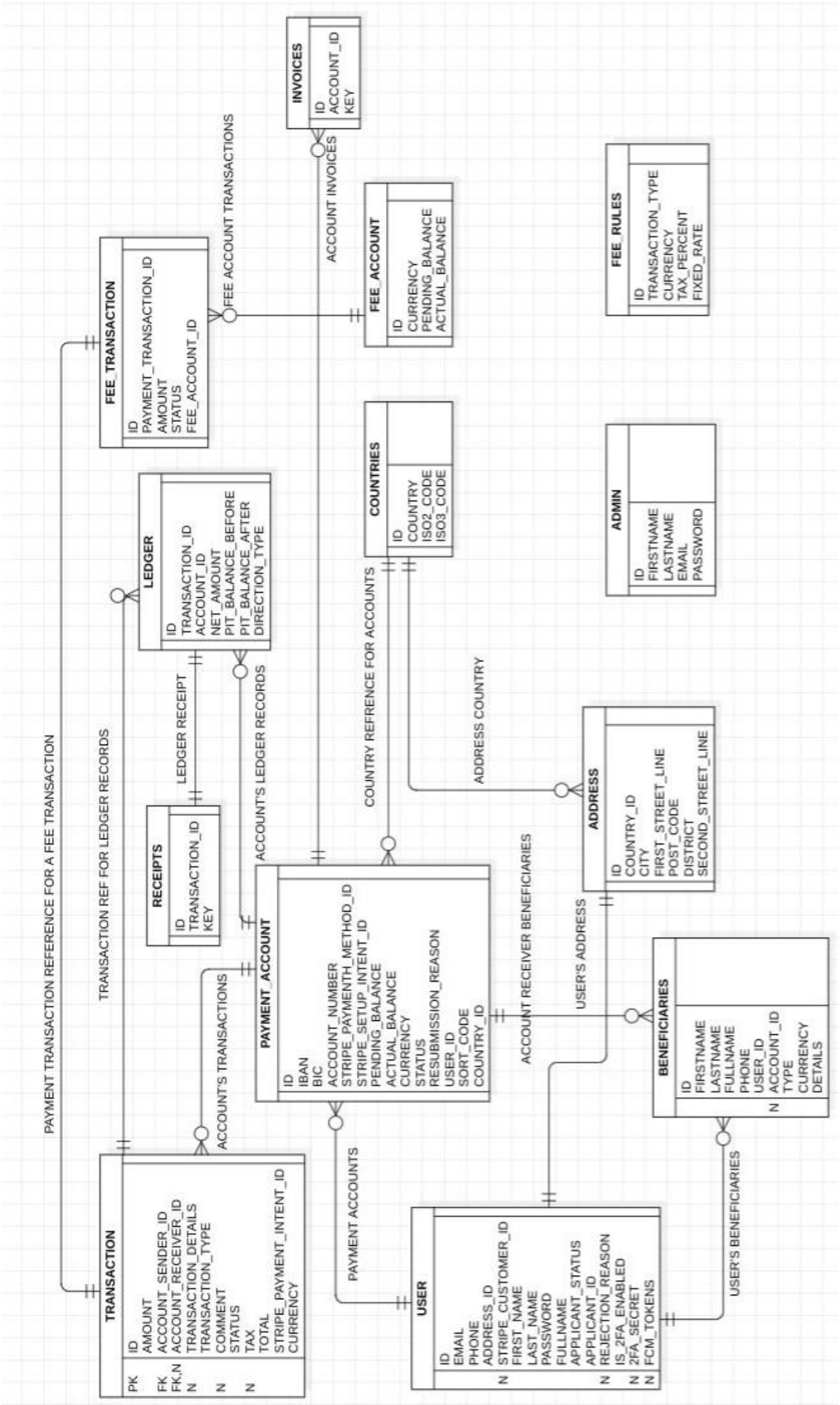


Рисунок 2.1 – ER-діаграма

2.1.4 Таблиці

У таблицях 2.1 – 2.13 і на рисунках 2.2 – 2.14 представлений фізичний опис таблиць бази даних банківського додатку та їх вигляд в перекладі на мову SQL.

Таблиця 2.1 – Опис таблиці «Users»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор користувача
FIRST_NAME	STRING	Ім’я користувача
LAST_NAME	STRING	Прізвище користувача
FULL_NAME	STRING	Повні ініціали користувача
EMAIL	STIRNG	Email користувача
PASSWORD	STRING	Пароль користувача
PHONE	STRING	Телефон користувача
APPLICANT_STATUS	STRING	Статус користувача у SumSub
REJECTION_REASON	STRING	Можлива причина відмови від адміністратора SumSub
STRIPE_CUSTOMER_ID	STRING	ID користувача у платіжній системі Stripe
ADDRESS_ID	UUID	ID запису адреси в базі
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```
CREATE TABLE users (
    id UUID PRIMARY KEY,
    email TEXT UNIQUE NOT NULL,
    phone TEXT UNIQUE NOT NULL,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    full_name TEXT GENERATED ALWAYS AS (first_name || ' ' || last_name) STORED,
    password TEXT NOT NULL,
    applicant_status TEXT,
    rejection_reason TEXT,
    stripe_customer_id TEXT,
    address_id UUID NOT NULL REFERENCES addresses(id) ON DELETE CASCADE,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP NULL,
    CONSTRAINT chk_applicant_status CHECK (applicant_status IN ('pending', 'approved',
        'rejected'))
);
```

Рисунок 2.2 – SQL-код таблиці «Users»

Таблиця 2.2 – Опис таблиці «Address»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор адреси користувача
COUNTRY_ID	UUID	Зовнішній ключ, що посилається на таблицю “Countries”
ADDRESS_LINE_1	STRING	Перший рядок адреси
ADDRESS_LINE_2	STIRNG	Другий рядок адреси
POST_CODE	INTEGER	Поштовий код адреси
CITY	STRING	Місто, в якому проживає користувач
DISTRICT	STRING	Країна, в якій проживає користувач
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```
CREATE TABLE addresses (
    id UUID PRIMARY KEY,
    country_id UUID NOT NULL REFERENCES countries(id),
    district TEXT NOT NULL,
    city TEXT NOT NULL,
    first_street_line TEXT,
    second_street_line TEXT,
    post_code TEXT,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP NULL
);
```

Рисунок 2.3 – SQL-код таблиці «Address»

Таблиця 2.3 – Опис таблиці «Admin»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор кредитної картки
FIRST_NAME	STRING	Ім’я адміністратора
LAST_NAME	STRING	Прізвище адвміністратора
FULL_NAME	INTEGER	Повне ім’я адміністратора
EMAIL	STRING	Email адміністратора
PASSWORD	INTEGER	Пароль адміністратора
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```sql
CREATE TABLE admins (
    id UUID PRIMARY KEY,
    email TEXT UNIQUE NOT NULL,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    full_name TEXT GENERATED ALWAYS AS (first_name || ' ' || last_name) STORED,
    password TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP NULL
);
```

Рисунок 2.4 – SQL-код таблиці «Admins»

Таблиця 2.4 – Опис таблиці «Beneficiaries»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор категорії
FIRST_NAME	STRING	Ім’я отримувача
LAST_NAME	STRING	Прізвище отримувача
FULL_NAME	STRING	Повне ім’я отримувача
PHONE	STRING	Продукти категорії
CURRENCY	STRING	Валюта, в якій відбулась транзакція для отримувача
USER_ID	UUID	Зовнішній ключ, що посилається на таблицю “Users”
ACCOUNT_ID	UUID	Зовнішній ключ, що посилається на таблицю “Payment accounts”
DETAILS	JSONB	Деталі отримувача, якщо ACCOUNT_ID не був наданий
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```sql
CREATE TABLE beneficiaries (
    id UUID PRIMARY KEY,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    full_name TEXT GENERATED ALWAYS AS (first_name || ' ' || last_name) STORED,
    phone TEXT,
    user_id UUID NOT NULL REFERENCES users(id),
    type UUID NOT NULL,
    currency TEXT NOT NULL,
    account_id UUID REFERENCES payment_accounts(id),
    details JSONB,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP NULL
);
```

Рисунок 2.5 – SQL-код таблиці «Beneficiaries»

Таблиця 2.5 – Опис таблиці «Countries»

Поле	Тип даних	Опис
ID	INTEGER	Унікальний ідентифікатор підкатегорії
NAME	STRING	Ім’я підкатегорії
ISO2_CODE	STRING	Іконка підкатегорії
ISO3_CODE	INTEGER	Зовнішній ключ, що посилається на таблицю “Categories”
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```sql
CREATE TABLE countries (
    id UUID PRIMARY KEY,
    name TEXT NOT NULL,
    iso2_code TEXT NOT NULL,
    iso3_code TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP NULL
);
```

Рисунок 2.6 – SQL-код таблиці «Countries»

Таблиця 2.6 – Опис таблиці «Fee Accounts»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор аккаунту
PENDING_BALANCE	DOUBLE	Невизначений баланс аккаунту
ACTUAL_BALANCE	DOUBLE	Актуальний баланс аккаунту
CURRENCY	TEXT	Валюта аккаунту
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```sql
CREATE TABLE fee_accounts (
    id UUID PRIMARY KEY,
    pending_balance DOUBLE PRECISION NOT NULL DEFAULT 0.0,
    actual_balance DOUBLE PRECISION NOT NULL DEFAULT 0.0,
    currency TEXT NOT NULL UNIQUE,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP NULL
);
```

Рисунок 2.7 – SQL-код таблиці «Fee Accounts»

Таблиця 2.7 – Опис таблиці «Fee Rules»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор комісійного правила
TYPE	STRING	Тип транзакції для комісійного правила
CURRENCY	STRING	Валюта транзакції для комісійного правила
FIXED_RATE	DOUBLE	Фіксована ставка комісії
TAX_PERCENT	DOUBLE	Процентна ставка комісії
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```sql
CREATE TABLE fee_rules (
    id UUID PRIMARY KEY,
    type TEXT NOT NULL,
    currency TEXT NOT NULL,
    fixed_rate DOUBLE PRECISION DEFAULT 0.0,
    tax_percent DOUBLE PRECISION DEFAULT 0.0,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP NULL
);

CREATE UNIQUE INDEX IDX_fee_rules_type_currency
ON fee_rules (type, currency);
```

Рисунок 2.8 – SQL-код таблиці «Fee Rules»

Таблиця 2.8 – Опис таблиці «Fee Transactions»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор комійсійної транзакції
TRANSACTION_ID	UUID	Зовнішній ключ, що посилається на таблицю “Transactions”
FEE_ACCOUNT_ID	UUID	Зовнішній ключ, що посилається на таблицю “Fee Accounts”
STATUS	STRING	Статус комісійної транзакції
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```sql
CREATE TABLE fee_transactions (
    id UUID PRIMARY KEY,
    amount DOUBLE PRECISION NOT NULL,
    transaction_id UUID NOT NULL,
    fee_account_id UUID,
    status TEXT NOT NULL DEFAULT 'PENDING',
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP NULL,
    CONSTRAINT fk_fee_account FOREIGN KEY (fee_account_id) REFERENCES fee_accounts(id) ON DELETE SET NULL,
    CONSTRAINT fk_transaction FOREIGN KEY (transaction_id) REFERENCES transactions(id)
);
```

Рисунок 2.9 – SQL-код таблиці «Fee Transactions»

Таблиця 2.9 – Опис таблиці «Fund Ledger»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор бухгалтерського запису
TRANSACTION_ID	UUID	Зовнішній ключ, що посилається на таблицю “Transactions”
ACCOUNT_ID	UUID	Зовнішній ключ, що посилається на таблицю “Accounts”
NET_AMOUNT	DOUBLE	Загальна сума транзакції
PIT_BALANCE_BEFORE	DOUBLE	Баланс аккаунту на момент створення транзакції
PIT_BALANCE_AFTER	DOUBLE	Баланс аккаунту у випадку успішного виконання транзакції
DIRECTION_TYPE	STRING	Тип напрямку транзакції (вхідна чи вихідна)
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```sql
CREATE TABLE fund_ledger (
    id UUID PRIMARY KEY,
    transaction_id UUID NOT NULL,
    account_id UUID NOT NULL,
    net_amount DOUBLE PRECISION NOT NULL,
    pit_balance_before DOUBLE PRECISION NOT NULL,
    pit_balance_after DOUBLE PRECISION NOT NULL,
    direction_type TEXT NOT NULL,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP NULL,
    CONSTRAINT fk_transaction FOREIGN KEY (transaction_id) REFERENCES transactions(id),
    CONSTRAINT fk_account FOREIGN KEY (account_id) REFERENCES payment_accounts(id),
    CONSTRAINT direction_type_check CHECK (direction_type IN ('IN', 'OUT')) -- Replace with actual `DirectionType` values
);
```

Рисунок 2.10 – SQL-код таблиці «Fund Ledger»

Таблиця 2.10 – Опис таблиці «Invoices»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор інвойсу
ACCOUNT_ID	UUID	Зовнішній ключ, що посилається на таблицю “Accounts”
KEY	INTEGER	Ключ файла з S3 сховища
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```
CREATE TABLE invoices (
    id UUID PRIMARY KEY,
    account_id UUID NOT NULL,
    key TEXT UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP NULL,
    CONSTRAINT fk_account FOREIGN KEY (account_id) REFERENCES payment_accounts(id)
);
```

Рисунок 2.11 – SQL-код таблиці «Invoices»

Таблиця 2.11 – Опис таблиці «Payment Accounts»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор банківського аккаунту
ACCOUNT_NUMBER	STRING	Номер банківського аккаунту
IBAN	STRING	Значення IBAN банківського аккаунту
BIC	STRING	BIC код банківського аккаунту
SORT_CODE	STRING	SORT код банківського аккаунту
STRIPE_PAYMENT_METHOD_ID	STRING	Ідентифікатор платіжного методу у платіжній системі Stripe
STRIPE_SETUP_INTENT_ID	STRING	Ідентифікатор сетап інтенту у платіжній системі Stripe
ACTUAL_BALANCE	DOUBLE	Актуальний баланс банківського аккаунта
PENDING_BALANCE	DOUBLE	Волатильний баланс банківського аккаунта
USER_ID	UUID	Зовнішній ключ, що посилається на таблицю “Users”
COUNTRY_ID	UUID	Зовнішній ключ, що посилається на таблицю “Countries”

Продовження табл. 2.11

Поле	Тип даних	Опис
CURRENCY	STRING	Валюта банківського аккаунта
STATUS	STRING	Статус активності аккаунта
RESUBMISSION_REASON	STRING	Можлива прична тимчасової відмови активації аккаунта
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```sql
CREATE TABLE payment_accounts (
    id UUID PRIMARY KEY,
    account_number TEXT UNIQUE NOT NULL,
    iban TEXT UNIQUE NOT NULL,
    bic TEXT NOT NULL,
    sort_code TEXT NOT NULL,
    stripe_payment_method_id TEXT,
    stripe_setup_intent_id TEXT,
    actual_balance DOUBLE PRECISION NOT NULL DEFAULT 0.0,
    pending_balance DOUBLE PRECISION NOT NULL DEFAULT 0.0,
    country_id UUID NOT NULL,
    user_id UUID NOT NULL,
    currency TEXT NOT NULL CHECK (currency IN ('USD', 'EUR', 'GBP')),
    status TEXT NOT NULL DEFAULT 'PENDING' CHECK (status IN ('PENDING', 'APPROVED', 'REJECTED')),
    resubmission_reason TEXT,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP,
    CONSTRAINT fk_user FOREIGN KEY (user_id) REFERENCES users(id),
    CONSTRAINT fk_country FOREIGN KEY (country_id) REFERENCES countries(id)
);
```

Рисунок 2.12 – SQL-код таблиці «Payment Accounts»

Таблиця 2.12 – Опис таблиці «Receipts»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор предмету замовлення
LEDGER_ID	UUID	Зовнішній ключ, що посилається на таблицю “Fund ledgers”
KEY	INTEGER	Зовнішній ключ, що посилається на таблицю “Cart”
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```
CREATE TABLE receipts (
    id UUID PRIMARY KEY,
    ledger_id TEXT UNIQUE NOT NULL,
    key TEXT UNIQUE NOT NULL,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP,
    CONSTRAINT fk_ledger FOREIGN KEY (ledger_id) REFERENCES fund_ledger(id)
);
```

Рисунок 2.13 – SQL-код таблиці «Receipts»

Таблиця 2.13 – Опис таблиці «Transactions»

Поле	Тип даних	Опис
ID	UUID	Унікальний ідентифікатор транзакції
AMOUNT	DOUBLE	Сума транзакції без урахування комісії
TAX	DOUBLE	Сума можливої комісії на транзакцію
TOTAL	DOUBLE	Сума транзакції з урахуванням комісії
STRIPE_PAYMENT_INTENT_ID	STRING	Унікальний ідентифікатор платежу в платіжній системі Stripe
SENDER_ACCOUNT_ID	STRING	Зовнішній ключ, що посилається на таблицю “Payment Accounts” (ID аккаунта – відправника)
RECEIVER_ACCOUNT_ID	STRING	Зовнішній ключ, що посилається на таблицю “Payment Accounts” (ID аккаунта – отримувача)
TYPE	STRING	Тип транзакції
CURRENCY	STRING	Валюта, в якій транзакція була створена
STATUS	STRING	Статус транзакції
COMMENT	STRING	Опціональні коментарі до транзакції від відправника
TRANSACTION_DETAILS	JSONB	Деталі транзакції, якщо ID аккаунта отримувача не був переданий
CREATED_AT	DATETIME	Дата, коли запис був створений
UPDATED_AT	DATETIME	Дата, коли запис був оновлений
DELETED_AT	DATETIME	Дата, коли запис був видалений (soft delete)

```sql
CREATE TABLE transactions (
    id UUID PRIMARY KEY,
    amount DOUBLE PRECISION NOT NULL,
    tax DOUBLE PRECISION,
    total DOUBLE PRECISION NOT NULL,
    stripe_payment_intent_id TEXT,
    sender_account_id UUID NOT NULL,
    receiver_account_id UUID,
    type TEXT NOT NULL,
    currency TEXT NOT NULL,
    status TEXT DEFAULT 'PENDING' NOT NULL,
    comment TEXT,
    transaction_details JSONB,
    created_at TIMESTAMP DEFAULT now() NOT NULL,
    updated_at TIMESTAMP DEFAULT now() NOT NULL,
    deleted_at TIMESTAMP,
    CONSTRAINT fk_sender_account FOREIGN KEY (sender_account_id) REFERENCES payment_accounts(id),
    CONSTRAINT fk_receiver_account FOREIGN KEY (receiver_account_id) REFERENCES payment_accounts(id)
);
```

Рисунок 2.14 – SQL-код таблиці «Transactions»

2.2 Діаграма інфраструктури

В цьому підрозділі буде наведений опис інфраструктури проєкту, його складові, як вони взаємодіють і обґрунтування зроблених рішень.

Проєкт повністю буде задеплоєний у хмарний провайдер, тому з самого почнемо з створення VPC (Virtual Private Cloud). Так як ми хочемо, щоб застосунок завжди відповідав і був активний, навіть у випадку масових збоїв, то 2 зон доступу (availability zone) буде більш, ніж достатньо. Далі буде розділення на 6 підмережж: 4 публічні і 2 приватні (див. далі по тексту пояснення цього вибору). В першу чергу треба створити DNS запис у регістрі з унікальним доменом і піддоменом, щоб користувачі не мали складнощей з зручністю доступу до вебзастосунку через введення адреси в рядку пошуку. Для того, щоб трафік з Інтернету пропускався і оброблювався додатком, буде логічно прив’язати Internet Gateway до VPC.

Клієнтська частина застосунку (frontend) – це те, що зустрічатиме кожного користувача, який хоче користуватися додатком, тому треба очікувати великий обсяг трафіку, і потрібно знайти рішення для того, щоб додаток автоматично пристосовувався до будь-якого об’єму трафіку в будь

який час. Також нам вкрай критично, щоб вебдодаток був доступний і відповідав при будь-яких обставинах, навіть у випадку масового збою в будь-якому з дата центрів в хмарі (Availability Zone Failure). В цьому випадку буде логічно створити одну ASG (Auto Scaling Group) і розтянути її на дві зони доступу разом з віртуальною машиною (VM, або в AWS це ще називається EC2 Instance). Це дасть можливість додатку обслуговувати користувачів навіть у можливому випадку збою «заліза» (hardware) в одній з зон доступу та пристосовуватись до різних об'ємів графіку в залежності від середнього рівня утилізації процессора в ASG, наприклад: якщо рівень утилізації більше 70%, то додати один EC2 інстанс до групи, якщо менше 30% – то знищити будь який EC2 інстанс. На додачу, такий підхід дозволяє значно оптимізувати витрати через природу прайсингу за ці ресурси – on-demand (pay as you go – платиш тільки за те, що використовуєш).

Для того, щоб в групі не було ситуації, коли створюється один «гарячий» інстанс (в проміжку якогось часу утилізується значно більше інших), поки інші простоюють без діла, на допомогу прийде ELB(Elastic Load Balancer), приєднаний до ASG. Він дозволить рівномірно делегувати запити від користувачів між усіма клієнтськими інстансами, забезпечуючи відсутність такого явища як "hot instance".

Швидкість відповіді, стійкість до збоїв, низька затримка – це не менш, а то і більш важливі атрибути для backend-частини застосунку, ніж для frontend групи віртуальних машин, так як основна бізнес-логіка, включно з взаємодією з різними базами даних, відбувається саме на вебсерверах. Тому такі ж самі дії, які були зроблені для обробки високих об'ємів трафіку для фронтенд частини, треба зробити і для backend частини додатку, включно з додаванням ELB між ASG фронтенд машин і ASG бекенд машин. Це все буде в окремих 2 публічних підмережах.

Деякий функціонал backend-частини інтегрується з такими AWS сервісами, як S3 (хмарне сховище файлів), SES (Simple Email Service – сервіс для відправки електронних листів) та SNS (Simple Notification Service – pub/sub сервіс, використовується як хмарий месседж брокер для сповіщення backend-

серверів у випадку завантаження нового файлу в S3; для цього використалася вбудована у сервіс інтеграція з S3 events) які є так званими AWS – managed сервісами: тобто, патчинг, пристосування до різних об’ємів трафіків, основний функціонал повністю контролюється самої організацією AWS, тому вони існують поза будь-якої зони відповідності.

Так як машини, які містять бази даних спілкуються тільки з backend-частиною вебдодатку, і не потребують якогось виходу в публічний Інтернет, буде логічно розмістити всі інстанси сховищ даних в приватних підмережах. Їх буде дві. Це забезпечить безпеку даних в сховищах, так як інстанси в цих підмережах не будуть доступні з публічного інтернету, і доступ до них робиться неможливим для потенційних хакерів.

Один з інстансів RDS (реляційна СКБД PostgreSQL) буде розташований в першій приватній мережі (primary node – Головна нода, яка і буде приймати всі запити в базу з серверів), і ще один – в іншій (standby нода – потрібна у випадку збою primary ноди, щоб замінити її для обробки запитів, поки primary нода неактивна). Це забезпечує стійкість до збоїв для нашої бази даних, оскільки всі дані про користувачів і їх транзакції будуть зберігатися саме в цій базі даних, і вона критична для правильної роботи backend-частини застосунку. Для того, щоб у standby ноді був завжди актуальний стан даних, буде налагоджена асинхронна реплікація даних з primary ноди. Це дозволить отримувати будь-яке оновлення даних чи стану primary ноди в асинхронному порядку, без затримки часу відповіді на запит від серверів.

Аналогічний з організацією інфраструктури RDS алгоритм був використаний для ElastiCache (хмарний варіант бази даних Redis) і OpenSearch (хмарний варіант бази даних ElasticSearch) : основний інстанс бази даних знаходиться в одній зоні доступу, ще один, другорядний – в іншій зоні у випадку якщо треба замінити неактивну основну (primary) ноду. Реплікація, так, як і у випадку з RDS, асинхронна.

Діаграма інфраструктури дозволяє візуалізувати архітектуру системи, визначити основні елементи, їх взаємодію, а також продемонструвати, як різні компоненти забезпечують функціонування вебдодатку (див. рис. 2.15) [2].

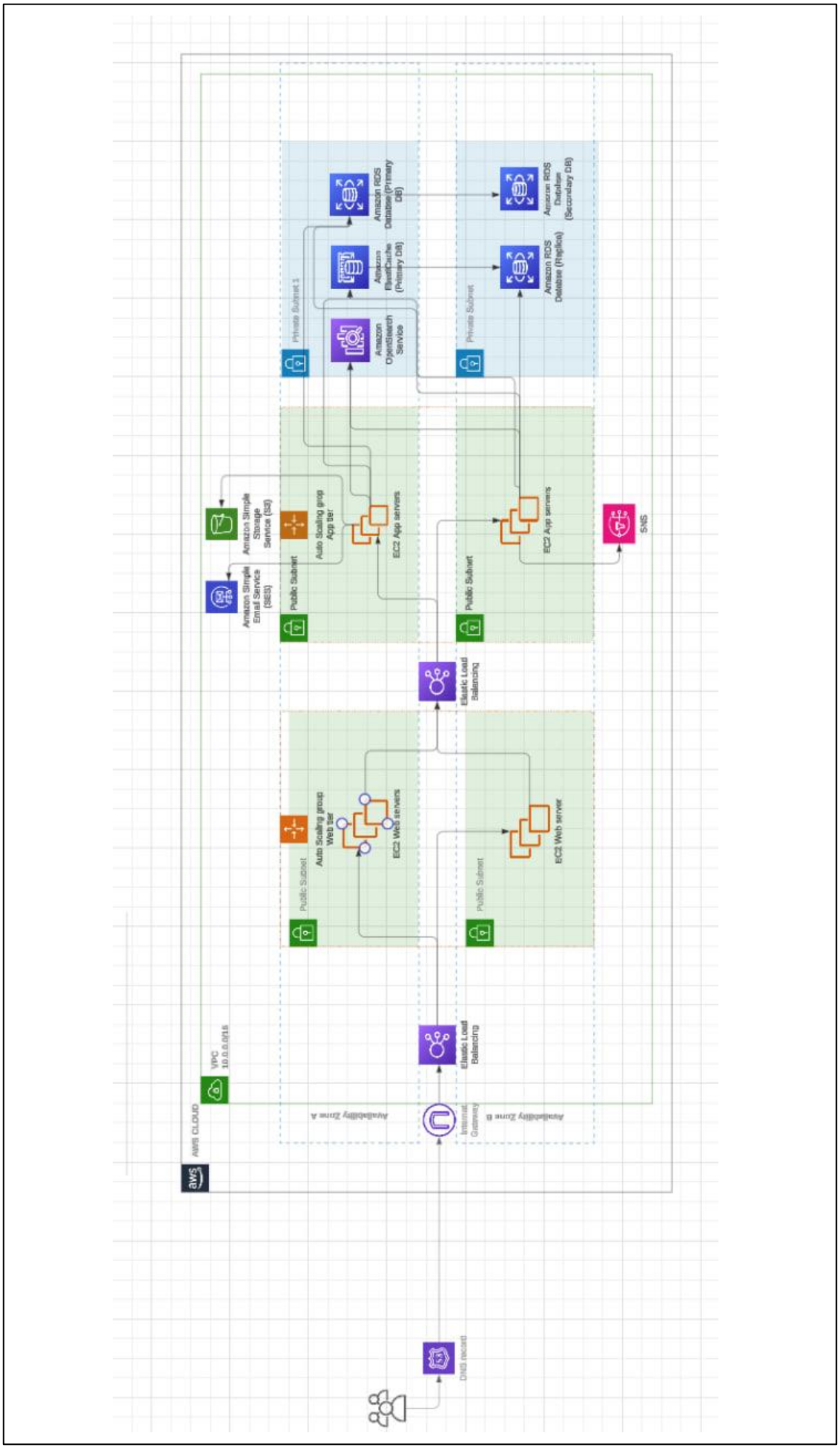


Рисунок 2.15 – Діаграма інфраструктури

Наведена діаграма інфраструктури забезпечує краще розуміння архітектури банківського вебдодатку та підвищує його надійність і ефективність. Вона слугує базою для прийняття обґрунтованих рішень на всіх етапах життєвого циклу системи – від розробки до експлуатації.

2.3 Опис технологій реалізації

Node.js – це кросплатформенне середовище виконання JavaScript з відкритим вихідним кодом. Це популярний інструмент практично для будь-якого типу проєктів. Node.js запускає JavaScript-рушій V8, ядро Google Chrome, за межами браузера. Це дозволяє Node.js бути дуже продуктивним [3].

Додаток Node.js працює в одному процесі, не створюючи новий потік для кожного запиту. Node.js надає набір примітивів асинхронного вводу/виводу у своїй стандартній бібліотеці, які запобігають блокуванню коду JavaScript, і взагалі, бібліотеки в Node.js написані з використанням неблокуючих парадигм, що робить поведінку блокування скоріше винятком, ніж нормою [3].

Коли Node.js виконує операцію вводу/виводу, наприклад, читання з мережі, доступ до бази даних або файлової системи, замість того, щоб блокувати потік і витрачати цикли процесора на очікування, Node.js відновлює операцію, коли відповідь повертається [3].

Це дозволяє Node.js обробляти тисячі паралельних з’єднань з одним сервером, не створюючи тягаря керування паралельністю потоків, що може бути значним джерелом помилок.

Nest (NestJS) – це фреймворк для створення ефективних, масштабованих серверних додатків Node.js [4]. Він використовує прогресивний JavaScript, побудований на TypeScript і повністю підтримує його (але при цьому дозволяє розробникам кодувати на чистому JavaScript) і поєднує в собі елементи ООП (об’єктно-орієнтованого програмування), ФП

(функціонального програмування) і ФРП (функціонально-реактивного програмування).

Під капотом Nest використовує надійні фреймворки HTTP-серверів, такі як Express (за замовчуванням), а за бажанням може бути налаштований на використання Fastify.

Nest забезпечує рівень абстракції над цими поширеними фреймворками Node.js (Express/Fastify), але також відкриває їх API безпосередньо розробнику. Це дає розробникам свободу використовувати безліч сторонніх модулів, доступних для базової платформи.

В останні роки, завдяки Node.js, JavaScript став «лінгва франка» Інтернету як для інтерфейсних, так і для бекенд-додатків. Це призвело до появи таких чудових проєктів, як Angular, React та Vue, які підвищують продуктивність розробників та дозволяють створювати швидкі, тестовані та розширювані інтерфейсні додатки. Однак, незважаючи на те, що для Node (і серверного JavaScript) існує безліч чудових бібліотек, помічників та інструментів, жодна з них не вирішує головну проблему – архітектуру.

Nest надає готову архітектуру додатків, яка дозволяє розробникам і командам створювати додатки, що добре тестуються, масштабуються, слабко пов’язані і легко підтримуються. Архітектура значною мірою натхненна Angular.

TypeORM – це потужний об’єктно-реляційний мапер (ORM) для Node.js, написаний на TypeScript [5]. Він забезпечує зручну роботу з базами даних, дозволяючи взаємодіяти з ними за допомогою об’єктів і типізованих класів замість сирих SQL-запитів. TypeORM підтримує популярні бази даних, такі як PostgreSQL, MySQL, MariaDB, SQLite, Oracle, MongoDB та інші.

Основні особливості TypeORM:

– декоратори: використання декораторів для визначення моделей, зв’язків і конфігурацій таблиць у базі даних;
– міграції: можливість автоматичного створення та застосування міграцій для синхронізації змін у базі даних;

– зв’язки між таблицями: підтримка відносин «один-до-одного», «один-до-багатьох», «багато-до-багатьох»;
– кросплатформеність: підтримка різних баз даних та легке переключення між ними;
– робота з репозиторіями: надання репозиторіїв для CRUD-операцій і складних запитів;
– типізація: забезпечує сильну типізацію завдяки TypeScript, що робить код надійнішим і зручнішим для підтримки.

TypeORM підходить для створення як невеликих проєктів, так і масштабованих корпоративних додатків, забезпечуючи високу продуктивність, чистоту коду та зручність у роботі.

Docker – це відкрита платформа для розробки, доставки та запуску додатків. Docker дозволяє відокремити додатки від інфраструктури для швидкої доставки програмного забезпечення. За допомогою Docker можна керувати інфраструктурою так само, як і своїми додатками. Використовуючи переваги методологій Docker для доставки, тестування та розгортання коду, мож значно скоротити затримку між написанням коду та його запуском у виробництво [7].

Docker надає можливість пакувати та запускати програму у слабко ізольованому середовищі, яке називається контейнер. Ізоляція та безпека дозволяє запускати багато контейнерів одночасно на одному хості. Контейнери легкі і містять все необхідне для запуску програми, тому вам не потрібно покладатися на те, що встановлено на хості. Можна ділитися контейнерами під час роботи і бути впевненими, що кожен, з ким ви ділитеся, отримає такий самий контейнер, який працює так само [7].

Docker надає інструменти та платформу для керування життєвим циклом ваших контейнерів:

– розробляйте свій додаток та його допоміжні компоненти, використовуючи контейнери;
– контейнер стає одиницею для розповсюдження та тестування вашого додатку;

– коли все готово, розгорніть свій додаток у виробничому середовищі, як контейнер або організований сервіс – це працює однаково, незалежно від того, чи є ваше виробниче середовище локальним центром обробки даних, хмарним провайдером або їх гібридом [7].

Redis – найшвидша у світі база даних в пам’яті. Вона надає хмарні та локальні рішення для кешування, векторного пошуку та баз даних NoSQL, які легко вписуються в будь-який технологічний стек, спрощуючи для цифрових клієнтів створення, масштабування та розгортання швидких додатків, на яких працює наш світ [8].

Redis надає структури даних, такі як рядки, хеші, списки, множини, відсортовані множини з діапазонними запитами, растрові зображення, гіперлоги, геопросторові індекси та потоки. Redis має вбудовані засоби реплікації, Lua-скриптування, LRU-виселення, транзакції та різні рівні персистентності на диску, а також забезпечує високу доступність за допомогою Redis Sentinel та автоматичне розбиття на розділи за допомогою Redis Cluster.

Може виконувати атомарні операції над цими типами, такі як додавання до рядка, збільшення значення в хеші, переміщення елемента до списку, обчислення перетину, об’єднання та різниці множин або отримання елемента з найвищим рейтингом у відсортованій множині.

Для досягнення максимальної продуктивності Redis працює з набором даних у пам’яті. Залежно від сценарію використання, Redis може зберігати ваші дані, періодично скидаючи набір даних на диск або додаючи кожну команду до журналу на диску. Ви також можете вимкнути збереження, якщо вам потрібен лише багатофункціональний мережевий кеш у пам’яті.

Redis підтримує асинхронну реплікацію з швидкою неблокуючою синхронізацією та автоматичним відновленням з частковою ресинхронізацією при розриві мережі.

Kubernetes – це портативна, розширювана платформа з відкритим вихідним кодом для управління контейнерними робочими навантаженнями та

сервісами, яка полегшує як декларативну конфігурацію, так і автоматизацію [9]. Вона має велику екосистему, що швидко розвивається.

Розглянемо **основні особливості.**

Виявлення служб і балансування навантаження. Kubernetes може виявити контейнер за допомогою DNS-імені або за власною IP-адресою. Якщо трафік до контейнера високий, Kubernetes здатний збалансувати навантаження і розподілити мережевий трафік так, щоб розгортання було стабільним.

Оркестрування сховища. Kubernetes дозволяє автоматично монтувати систему зберігання на ваш вибір, наприклад, локальні сховища, публічні хмарні провайдери тощо.

Автоматизоване розгортання та відкат. Ви можете описати бажаний стан для розгорнутих контейнерів за допомогою Kubernetes, і він може змінювати фактичний стан на бажаний з контрольованою швидкістю. Наприклад, ви можете автоматизувати Kubernetes для створення нових контейнерів для вашого розгортання, видалення існуючих контейнерів і перенесення всіх їхніх ресурсів до нового контейнера.

Автоматичне пакування контейнерів. Ви надаєте Kubernetes кластер вузлів, які він може використовувати для запуску контейнерних завдань. Ви повідомляєте Kubernetes, скільки процесорної та оперативної пам’яті потрібно кожному контейнеру. Kubernetes може розмістити контейнери на ваших вузлах, щоб найкращим чином використовувати ваші ресурси.

Самовідновлення. Kubernetes перезапускає контейнери, які вийшли з ладу, замінює контейнери, знищує контейнери, які не реагують на перевірку стану, визначену користувачем, і не рекламує їх клієнтам, доки вони не будуть готові до роботи.

Керування секретами та конфігураціями. Kubernetes дозволяє зберігати та керувати конфіденційною інформацією, зокрема паролями, токенами OAuth та ключами SSH. Ви можете розгортати та оновлювати секрети та конфігурацію додатків без перебудови образів контейнерів та без розкриття секретів у конфігурації стеку.

Пакетне виконання. На додаток до сервісів, Kubernetes може керувати вашими пакетними та CI робочими навантаженнями, замінюючи контейнери, які вийшли з ладу, за бажанням.

Горизонтальне масштабування. Масштабуйте свій додаток вгору і вниз за допомогою простої команди, за допомогою інтерфейсу користувача або автоматично на основі використання процесора.

Дворівневий стек IPv4/IPv6. Виділення адрес IPv4 та IPv6 для контейнерів і сервісів.

Розроблено для розширюваності. Додавайте функції до вашого кластера Kubernetes, не змінюючи вихідний код.

Kubernetes є основним інструментом для розробників і DevOps-інженерів, які прагнуть створювати масштабовані, ефективні та стабільні системи.

HashiCorp Terraform – це інструмент «інфраструктура як код», який дозволяє визначати як хмарні, так і локальні ресурси у зручних для читання конфігураційних файлах, які ви можете змінювати, повторно використовувати та ділитися ними [10]. Потім ви можете використовувати послідовний робочий процес для забезпечення та управління всією інфраструктурою протягом її життєвого циклу. Terraform може керувати низькорівневими компонентами, такими як обчислювальні, сховища та мережеві ресурси, а також високорівневими компонентами, такими як записи DNS та функції SaaS.

Основний робочий процес Terraform складається з трьох етапів:

– *написання*: ви визначаєте ресурси, які можуть належати декільком хмарним провайдерам і сервісам, наприклад, ви можете створити конфігурацію для розгортання програми на віртуальних машинах у мережі віртуальної приватної хмари (VPC) з групами безпеки та балансувальником навантаження;
– *план*: Terraform створює план виконання, що описує інфраструктуру, яку він створить, оновить або знищить на основі існуючої інфраструктури та вашої конфігурації;

– *застосування*: після затвердження Terraform виконує запропоновані операції в правильному порядку, враховуючи всі залежності від ресурсів, наприклад, якщо ви оновите властивості VPC і змініте кількість віртуальних машин у цьому VPC, Terraform відтворить VPC перед масштабуванням віртуальних машин.

Stripe – це фінтех-платформа для обробки онлайн-платежів, яка дозволяє бізнесам приймати платежі, керувати підписками та обробляти транзакції у різних валютах по всьому світу [11].

Основні переваги:

– простота інтеграції: API та SDK для швидкого підключення до веб та мобільних додатків;
– безпека: відповідність стандартам PCI DSS та вбудовані інструменти для запобігання шахрайству (Stripe Radar);
– глобальність: підтримка локальних способів оплати та багатьох валют;
– масштабованість: підходить для малого бізнесу та великих компаній.

Stripe є популярним рішенням для зручної, безпечної та масштабованої обробки платежів.

Elasticsearch – це розподілена пошуково-аналітична система з відкритим вихідним кодом, побудована на Apache Lucene і розроблена на Java [12]. Він починався як масштабована версія пошукового фреймворку з відкритим кодом Lucene, потім була додана можливість горизонтального масштабування індексів Lucene.

Elasticsearch дозволяє зберігати, шукати та аналізувати величезні обсяги даних швидко і майже в реальному часі та видавати відповіді за мілісекунди. Він здатний досягати швидких пошукових відповідей, тому що замість того, щоб шукати безпосередньо в тексті, він шукає в індексі. Він використовує структуру, засновану на документах, а не на таблицях і схемах, і постачається з широким набором REST API для зберігання і пошуку даних. Можна думати про Elasticsearch як про сервер, який може обробляти JSON-запити і повертати вам JSON-дані.

По суті, Elasticsearch організовує дані в документи, які є одиницями інформації на основі JSON, що представляють сутності. Документи групуються в індекси, подібно до баз даних, на основі їх характеристик. Для ефективного пошуку Elasticsearch використовує інвертовані індекси – структуру даних, яка зіставляє слова з їхнім розташуванням у документі. Розподілена архітектура Elasticsearch дозволяє здійснювати швидкий пошук і аналіз величезних обсягів даних майже в режимі реального часу.

Sumsub – це рішення для верифікації особистості та боротьби з шахрайством на основі штучного інтелекту, яке захищає і розширює можливості регульованих компаній в Інтернеті, забезпечуючи банківський рівень KYC, відповідність KYT, скринінг AML і зберігання даних клієнтів [13].

Завдяки глобальному покриттю, налаштованим критеріям верифікації, широкій аналітиці та інструментам управління кейсами ви можете спростити процес верифікації для фізичних та юридичних осіб. Знайдіть ідеальний баланс між задоволеністю клієнтів і дотриманням нормативних вимог.

Компанії зазвичай перевіряють різних постачальників, щоб подолати такі труднощі, як загальний брак інформації, обмежене глобальне покриття, ручна оцінка профілю та низький рівень проходження перевірки. Не кажучи вже про ускладнення, які виникають, коли організації та їхні провайдери координують критерії перевірки.

Привітайте швидкі та прозорі перевірки, що стали можливими завдяки зручному інтерфейсу, який дозволяє відстежувати різноманітні заздалегідь створені або налаштовані фактори під час оцінки та вдосконалення операцій.

Ключові особливості:

– мобільний онбордінг;
– верифікація особистих даних;
– перевірка на відмивання коштів;
– перевірка документів;
– оцифрування;
– багаторазова KYC;

– крипто-відповідність;

– налаштовувана аналітика.

В основі SumSub конкретно для нашого проєкту ключовими функціями SumSub є процеси KYC (Know Your Customer) та KYT (Know Your Transaction).

KYC (Know Your Customer) – це процес ідентифікації та верифікації клієнтів. Ідентифікація означає збір персональних даних клієнта, а верифікація – перевірку достовірності цих даних.

Щоб ідентифікувати клієнта, бізнесу зазвичай потрібні такі дані:

– ім’я;

– дата народження;

– адреса.

Щоб перевірити ці дані, бізнес може використовувати підхід верифікації на основі документів. Це передбачає перевірку документів, що посвідчують особу клієнта, та підтвердження адреси (зазвичай рахунок за комунальні послуги), щоб переконатися, що вони є справжніми та дійсними.

KYT (Know Your Transaction) – це важливий процес контролю за дотриманням законодавства та управління ризиками, особливо у фінансовій сфері. Він передбачає перевірку деталей кожної фінансової транзакції, включаючи джерело походження коштів та передбачуваного одержувача. В основному це забезпечує дотримання регуляторних вимог і знижує ризики, пов’язані з фінансовими злочинами, такими як відмивання грошей і фінансування тероризму.

Впроваджуючи протоколи KYT у свої стратегії управління ризиками за допомогою таких інструментів, як рішення SumSub, компанії можуть ефективно виявляти та оцінювати ризики, пов’язані з кожним гаманцем, а отже, діяти відповідно до них:

– *підтримка регуляторних стандартів*: KYT відіграє важливу роль у допомозі бізнесу в дотриманні правил боротьби з відмиванням грошей та фінансуванням тероризму, забезпечуючи таким чином дотримання законодавства;

– *зниження ризиків фінансових злочинів*: досліджуючи деталі транзакцій, KYT допомагає виявляти та зменшувати ризики, пов’язані з фінансовими злочинами, таким чином захищаючи бізнес від потенційної юридичної та репутаційної шкоди;
– *підтримка цілісності фінансової системи*: KYT сприяє загальній цілісності фінансових систем, сприяючи прозорості та підзвітності транзакцій, забезпечуючи довіру між зацікавленими сторонами.

3 РЕАЛІЗАЦІЯ ПЗ

Даний розділ присвячений технічний реалізації проєкту з стислими та зрозумілими поясненнями до кожної з представлених функціональностей [14].

3.1 Реалізація основного функціоналу

Реєстрація: користувач вводить валідні дані в усі необхідні поля: FirstName, LastName, Email, Password, і т.д. і натискає на кнопку “Sign Up”. Якщо користувач з таким email або телефоном вже існує в базі, з серверу прийде 500 помилка, бо відбувається порушення правила унікальності поля «Email» або «Phone». Якщо такого email немає, і всі дані валідні, то в базі створиться новий запис з інформацією про користувача.

Пароль буде зашифрований бібліотекою bcrypt для більшої секьюрності, і в кінцевому JSON об’єкті він не буде повертатися. Також, бекграунд джобою створюється stripe_customer_id у платіжній системі Stripe, який потрібен для проведення транзакцій у реальній платіжній системі (див. рис. 3.1 – 3.2).

```
public async create(dto: CreateUserDTO): Promise<Omit<User, 'password'>> {
  const password : string  = await hash(dto.password,  saltOrRounds: 10);

  const data: DeepPartial<User> = {
    ...dto,
    id: uuidv7(),
    password,
  };

  return await this.repository.create(data);
}
```

Рисунок 3.1 – Код для реєстрації користувача

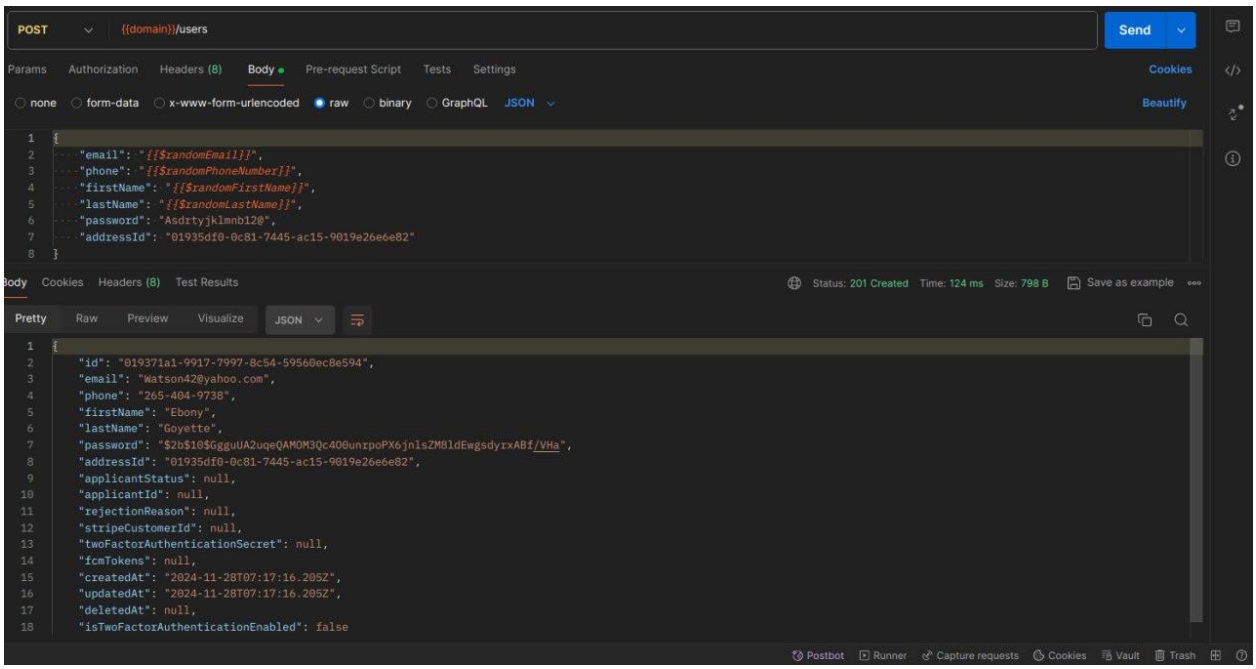


Рисунок 3.2 – Успішна реєстрація користувача

Аутентифікація: Користувач вводить email (або телефон) і пароль у валідному форматі у формі «Sign In». Якщо email (або телефон) коректний, а пароль неправильний, то користувачу повернеться помилка 401 «Invalid credentials».

Якщо введеного email не існує в жодному записі користувачів в базі, сервер поверне 401 помилку з месседжом «User not found».

Якщо email корректний і пароль до нього підходить, сервер підписує два JWT токени – один з них accessToken, який має невеликий expirationTime, і який і буде використовуватись для аутентифікації реквестів, і refreshToken, який викаористовується для постійного оновлення accessToken, коли той заекспайриться.

Коли refreshToken заекспайриться, то користувач мусить знову залогінитися і вводити свої креди. В ці токени записується повна інформація користувача з бази, окрім пароля. Потім accessToken сетиться у browserStorage, і відтепер, усі реквести, зроблені з клієнту, будуть автоматично мати цей токен у headers, як показник, що юзер залогінений. На клієнті юзер одразу переходить на сторінку свого персонального аккаунту (див. рис. 3.3 – 3.8).

```
private async generateTokens(user: User): Promise<AuthResult> {  Show usages
  const accessTokenPayload: TokenData<Partial<User>> = {
    expiresIn: this.jwtConfig.expiresIn,
    secret: this.jwtConfig.secret,
  };

  const refreshTokenPayload: TokenData<Partial<User>> = {
    payload: user,
    expiresIn: this.jwtConfig.refreshExpiresIn,
    secret: this.jwtConfig.refreshSecret,
  };

  const [accessToken, refreshToken] = await Promise.all([
    this.jwtService.generateToken(accessTokenPayload),
    this.jwtService.generateToken(refreshTokenPayload),
  ]);

  return { accessToken, refreshToken };
}

Edit | Explain | Test | Document | Fix
public async refreshAccessToken(  Show usages  Vladislav Huhlia
  user: User,
): Promise<Pick<AuthResult, 'accessToken'>> {
  const generateTokenData: TokenData<Partial<User>> = {
    payload: user,
    secret: this.jwtConfig.adminSecret,
    expiresIn: this.jwtConfig.expiresIn,
  };

  const accessToken = await this.jwtService.generateToken(generateTokenData);

  return { accessToken };
}
```

Рисунок 3.3 – Код для генерації JWT токенів і рефрешу accessToken

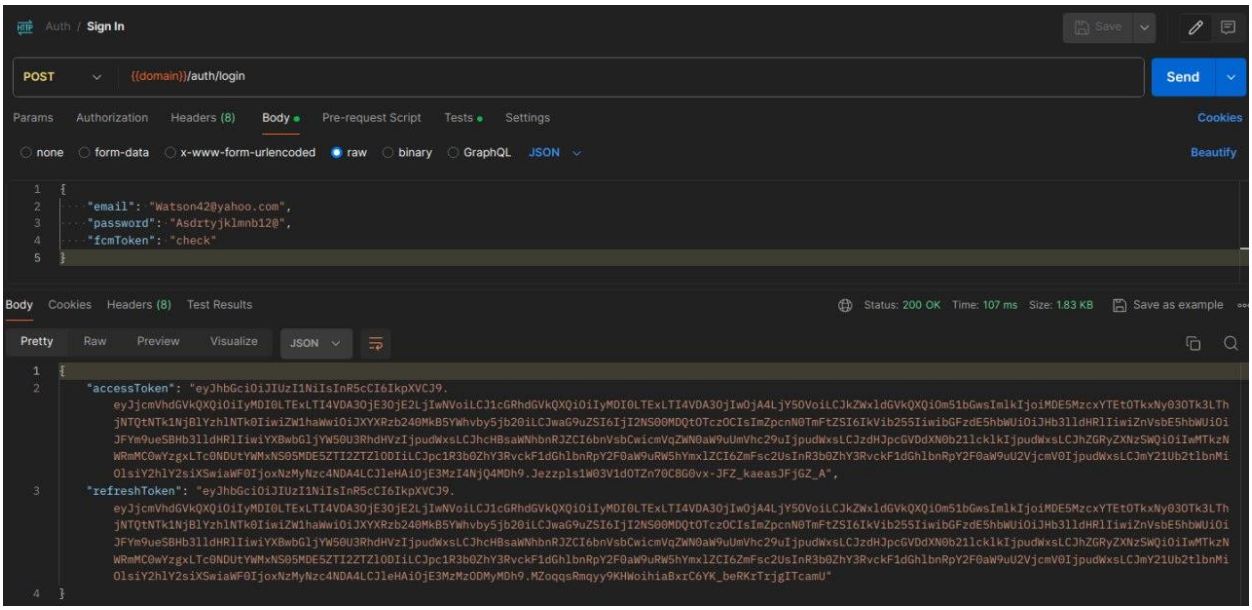


Рисунок 3.4 – Успішна аутентифікація і видача токенів

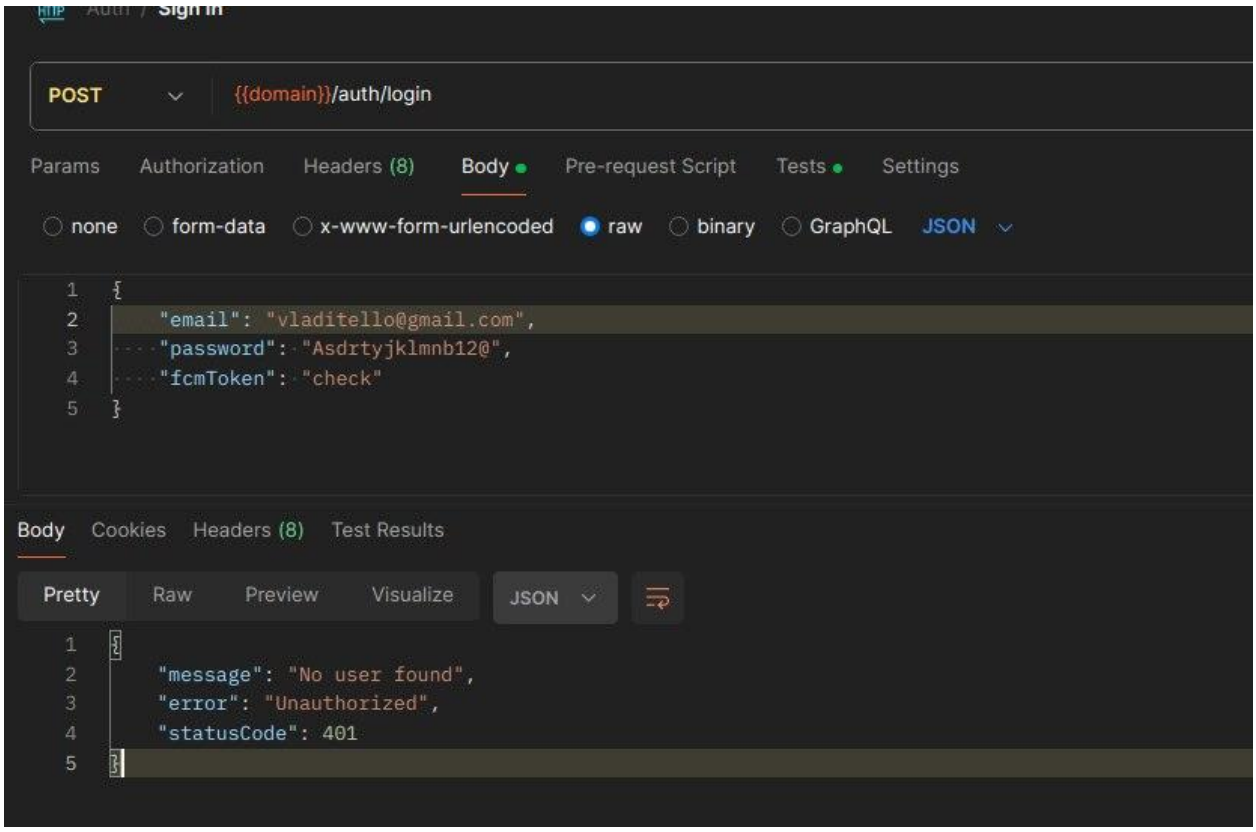


Рисунок 3.5 – Помилка при відсутності юзера в базі з заданим email

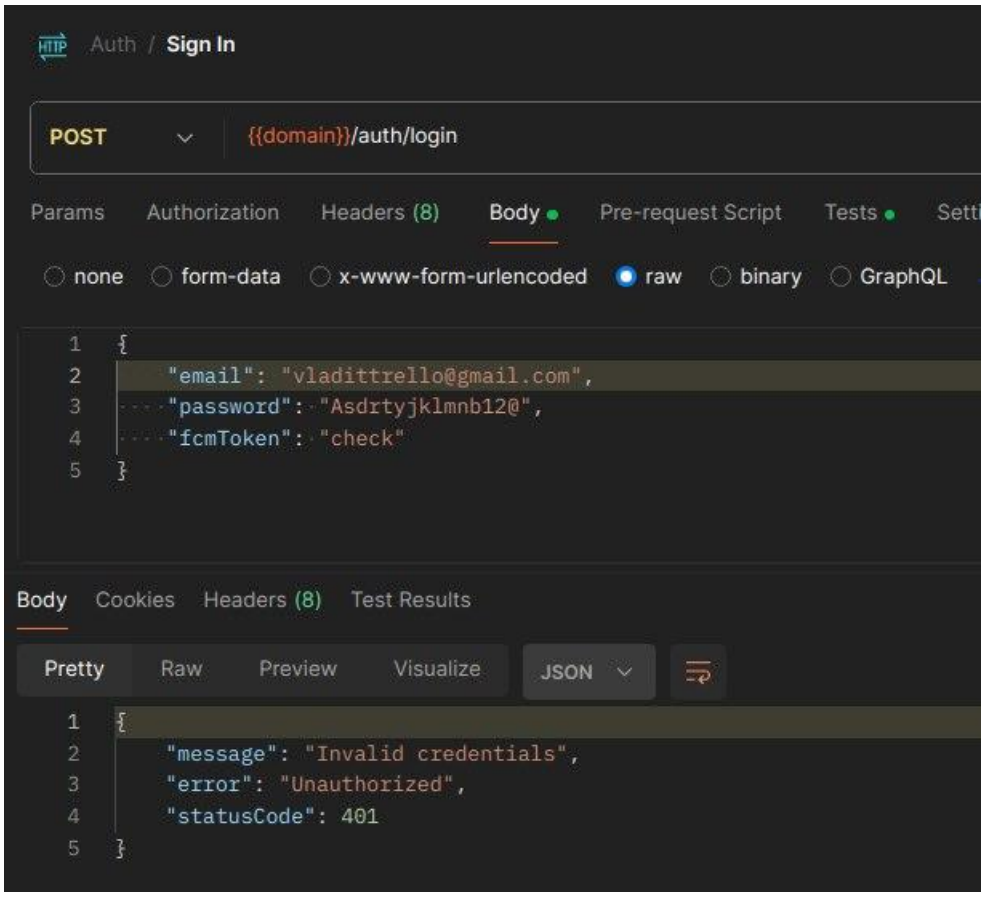


Рисунок 3.6 – Помилка «Invalid credentials» при введенні неправильного паролю

```
@Injectable()  Show usages  Vladislav Huhlia
export class JwtAccessStrategy extends PassportStrategy(
  Strategy,
    name: 'jwt-access',
) {
  constructor(private readonly jwtConfig: JwtConfigService) {  no usages  Vladislav Huhlia
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: jwtConfig.secret,
    });
  }

  public async validate(payload: TokenPayload): Promise<TokenPayload> {  no usages  Vladislav Huhlia
    return payload;
  }
}
```

Рисунок 3.7 – Код функції-захисника від неаутентифікованого доступу

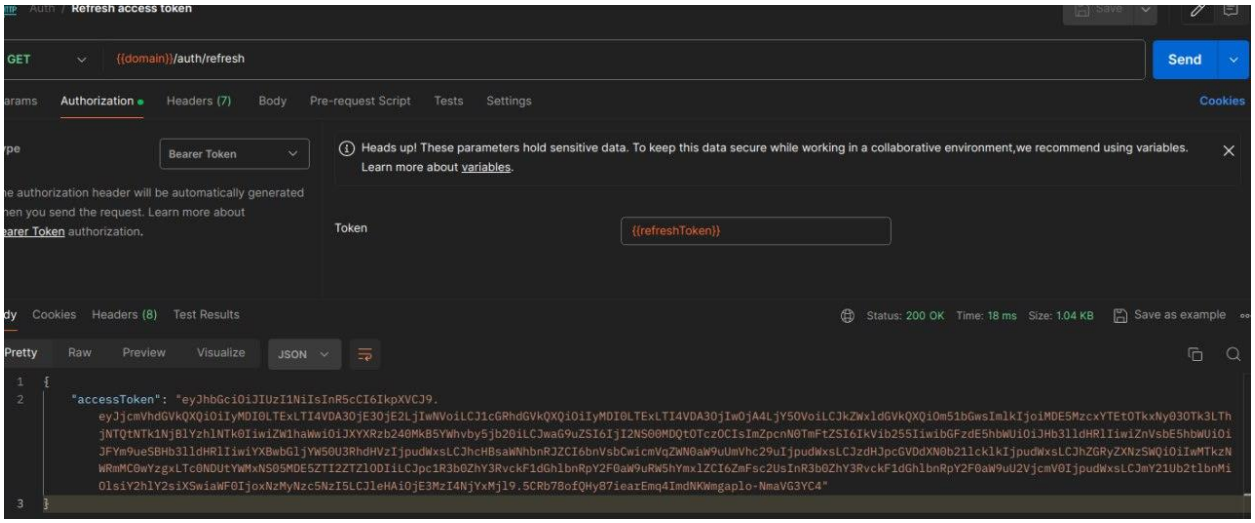


Рисунок 3.8 – Успішний рефреш accessToken

Зміна паролю: користувач може змінити пароль, ввівши правильний поточний пароль і новий пароль, який він хоче встановити (див. рис. 3.9 – 3.11). Якщо поточний пароль неправильний, то сервер поверне 400 код помилки з текстом «The Current Password is incorrect». Якщо поточний пароль правильний, і новий пароль відповідає стандартам валідації, новий пароль хешується, і старе значення замінюється новим. Якщо користувач захоче залогінитись після зміни паролю, він має ввести новий пароль.

```
public async changePassword( Show usages  Vladislav Huhlia
  dto: UpdatePasswordDTO,
  params: User,
): Promise<string> {
  const filter: FindOptionsWhere<User> = {
    id: params.id,
  };

  const getOnePayload: GetOne<any> = {
    filter,
    select: ['password'],
  };

  const user: User = await this.repository.getOne(getOnePayload);

  if (!(await compare(dto.currentPassword, user.password))) {
    throw new BadRequestException( objectOrError: 'The Current Password is incorrect');
  }

  const password : string  = await hash(dto.newPassword,  saltOrRounds: 10);

  const update: DeepPartial<User> = {
    password,
  };

  await this.repository.update(filter, update);

  return 'Password updated successfully';
}
```

Рисунок 3.9 – Код для зміни паролю

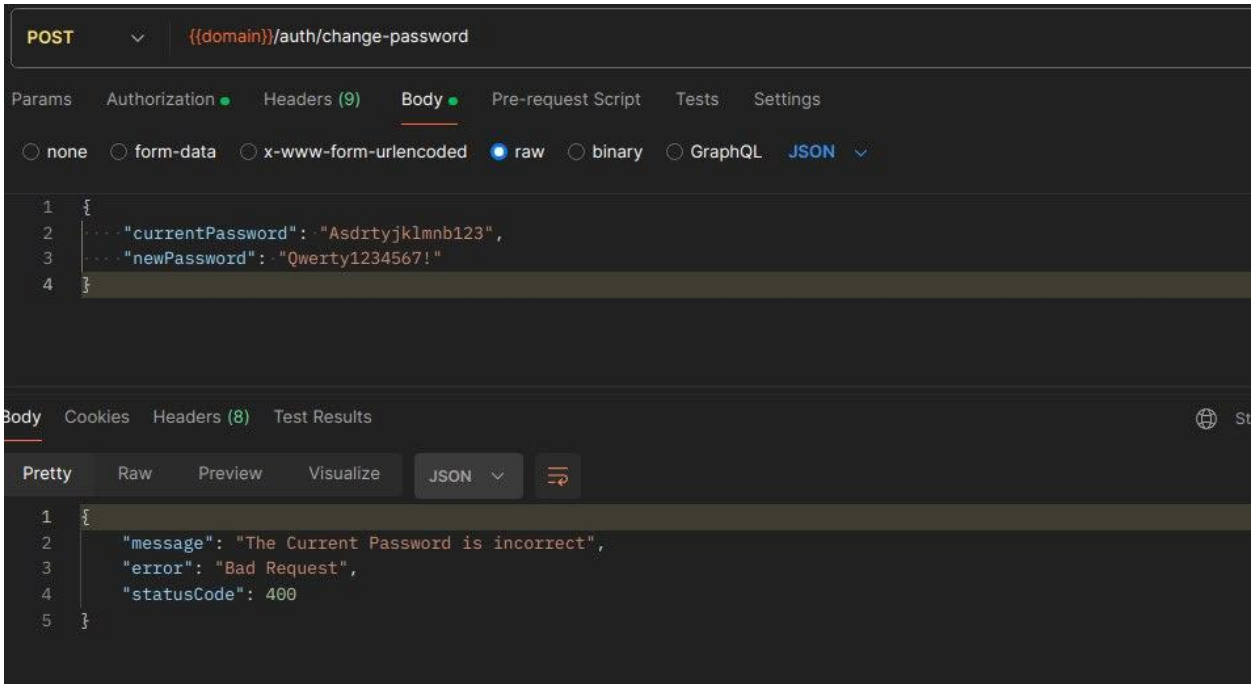


Рисунок 3.10 – Помилка при неправильному поточному паролі

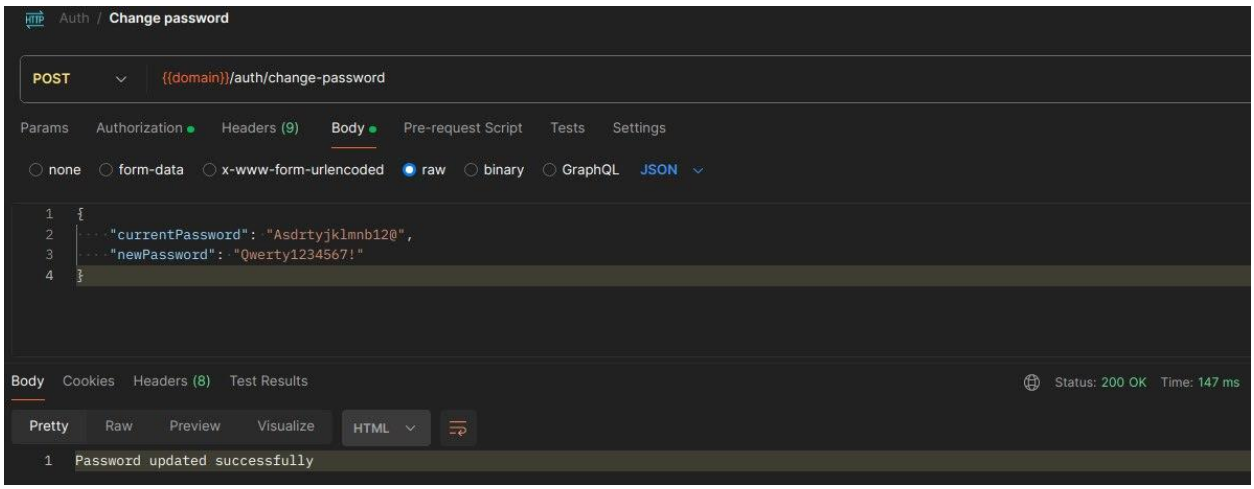


Рисунок 3.11 – Успішна зміна паролю

Відновлення паролю: користувач може відновити пароль у випадку його втрати (див. рис. 3.12 – 3.15). Спочатку, він має ввести email. Якщо користувач з введеним email існує в системі, на нього відправиться електронний лист з посиланням на сторінку для відновлення паролю. В системі ж створюється новий JWT токен (не плутати з токеном для аутентифікації), який служить токеном для валідації коду, щоб підтвердити, що пароль для користувача змінює саме цей користувач, а не хтось інший. Коли юзер переходить по посиланню в листі на сторінку, тай його чекає два поля: newPassword та confirmPassword. Якщо значення двох полей співпадають, то пароль хешується і значення у відповідному записі в базі даних успішно замінюється.

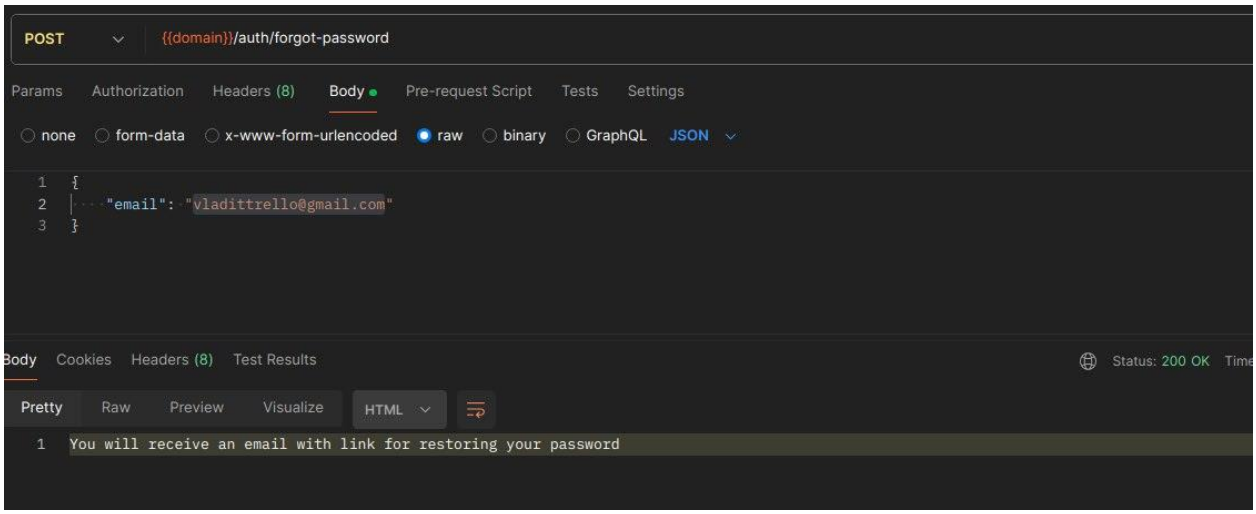


Рисунок 3.12 – Успішне відправлення електронного листа

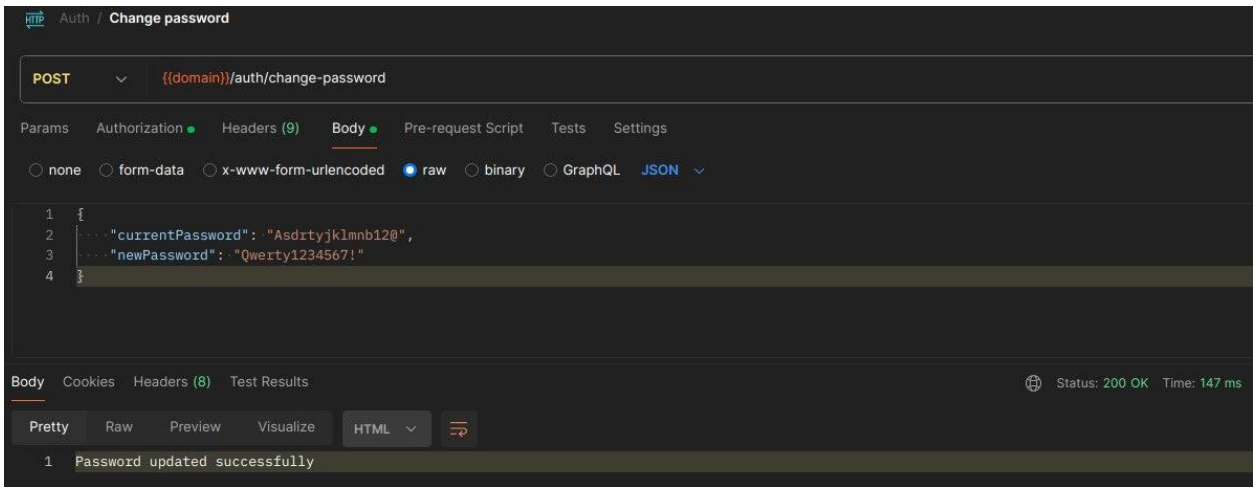


Рисунок 3.13 – Успішне відновлення паролю

```
public async resetPassword( Show usages  Vladislav Huhlia
  dto: ResetPasswordDTO,
  params: User,
): Promise<string> {
  const { email :string  } = params;

  if (dto.newPassword !== dto.repeatedPassword) {
    throw new BadRequestException(
      objectOrError: 'New and repeated passwords are not the same',
    );
  }

  const filter: FindOptionsWhere<User> = {
    email,
  };

  const password :string  = await hash(dto.newPassword,  saltOrRounds: 10);

  const update: DeepPartial<User> = {
    password,
  };

  await this.repository.update(filter, update);

  return 'Password updated successfully';
}
```

Рисунок 3.14 – Код для відновлення паролю



Рисунок 3.15 – Зразок листа з посиланням для відновлення паролю

Створення банківського аккаунту: користувач вводить валідні дані у всі необхідні поля: IBAN, BIC, Sort Code, номер аккаунту, і т.д. (див. рис. 3.16 – 3.18). Після відправки реквесту на сервер, тіло реквеста проходить процес валідації на правильні типи, і потім створюється запис в базі даних, прив'язаний до користувача зовнішнім ключем «userId». Бекграунд джобою також створюється унікальний айді платіжного метода в платіжній системі Stripe, який потрібен потім для проведення платіжок і цей ID кладеться у відповідне поле у новоствореному записі в базі.

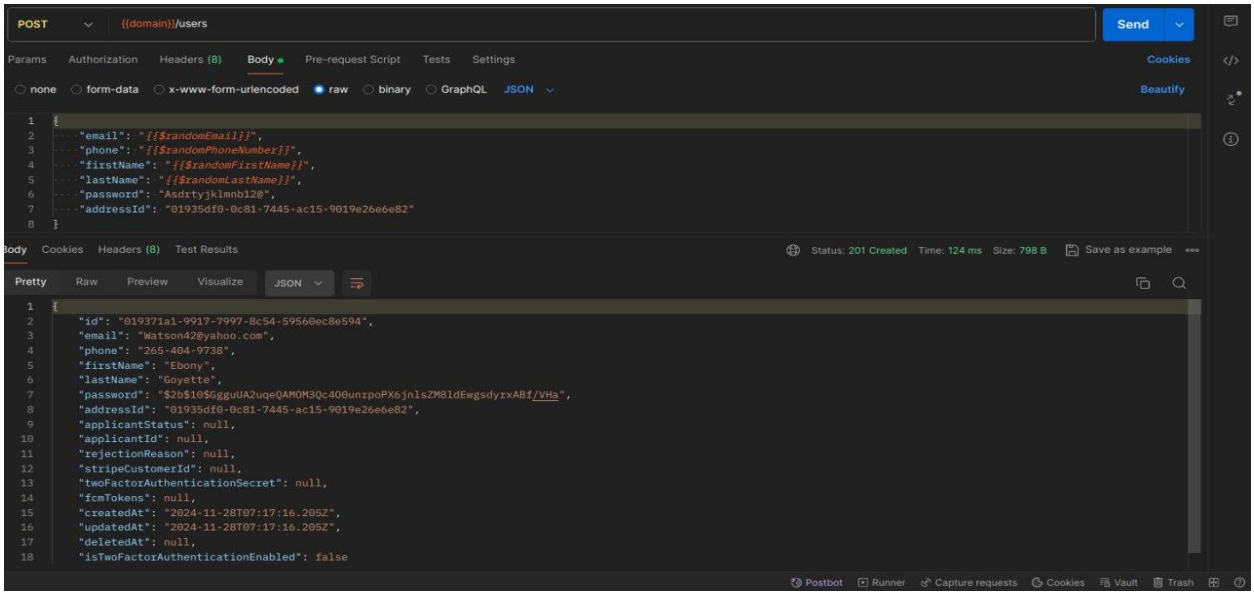


Рисунок 3.16 – Успішне створення банківського аккаунту

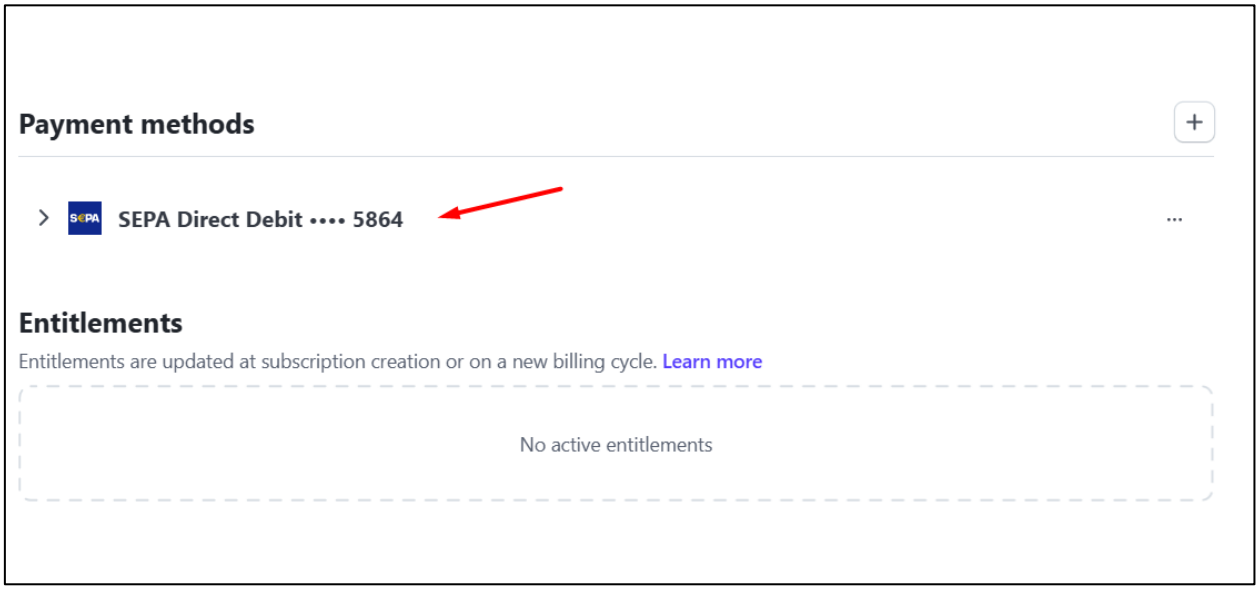
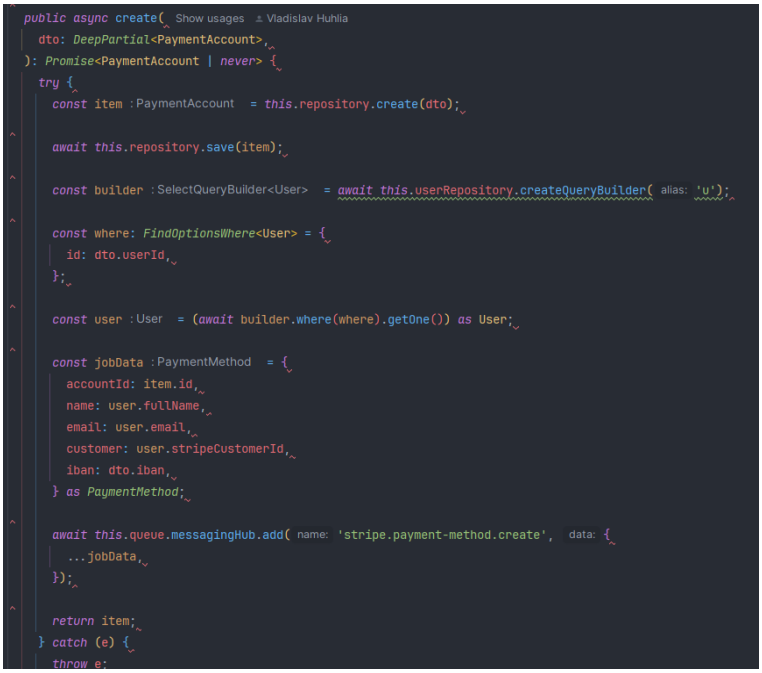


Рисунок 3.17 – Створений платіжний метод в Stripe



```
public async create( Show usages  Vladislav Huhlia
  dto: DeepPartial<PaymentAccount>,
): Promise<PaymentAccount | never> {
  try {
    const item : PaymentAccount  = this.repository.create(dto);

    await this.repository.save(item);

    const builder : SelectQueryBuilder<User>  = await this.userRepository.createQueryBuilder( alias: 'u');

    const where: FindOptionsWhere<User> = {
      id: dto.userId,
    };

    const user : User  = (await builder.where(where).getOne()) as User;

    const jobData : PaymentMethod  = {
      accountId: item.id,
      name: user.fullName,
      email: user.email,
      customer: user.stripeCustomerId,
      iban: dto.iban,
    } as PaymentMethod;

    await this.queue.messagingHub.add( name: 'stripe.payment-method.create',  data: {
      ...jobData,
    });

    return item;
  } catch (e) {
    throw e;
```

Рисунок 3.18 – Код для створення запису банківського аккаунту в базі даних

Активація банківського аккаунту: адміністратор може продивитись список усіх аккаунтів у статусі «На розгляді» і активувати їх (див. рис. 3.19 – 3.21). Він може переглянути усі банківські деталі, які користувач заповнив, і якщо адміністратор вважає, що все добре, то може змінити статус аккаунта на «ACTIVE». Або тимчасово відхилити активацію, змінити статус на «Перевідкрито» і вказати причини відмови у полі «Resubmission reason».

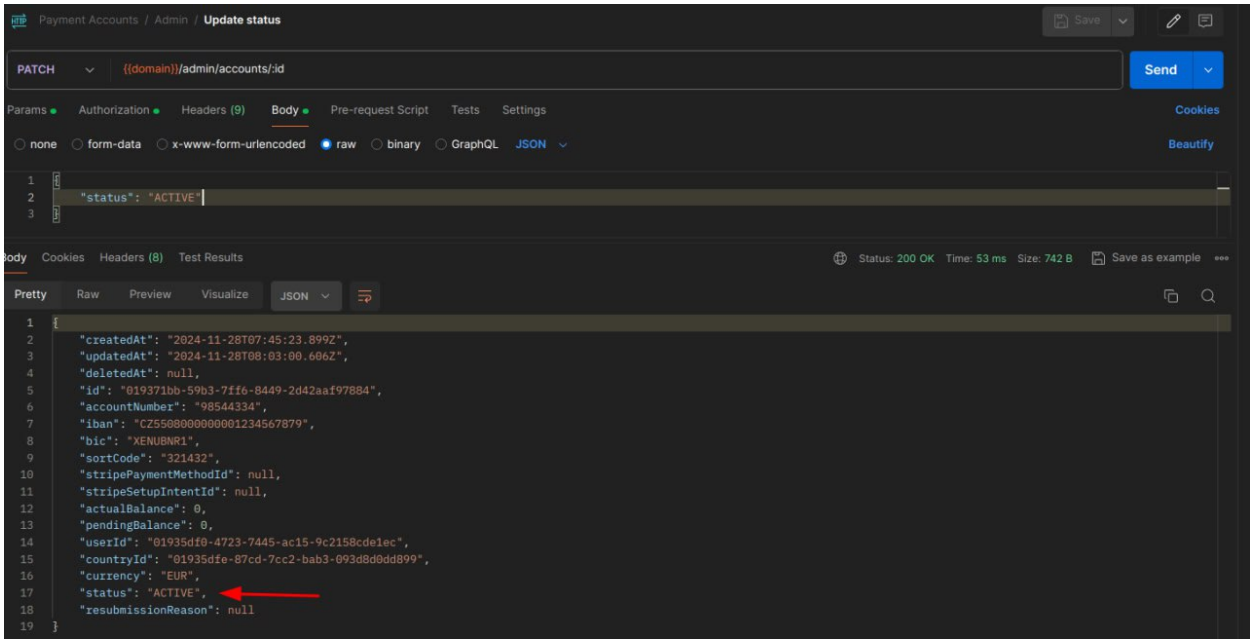


Рисунок 3.19 – Активація аккаунту

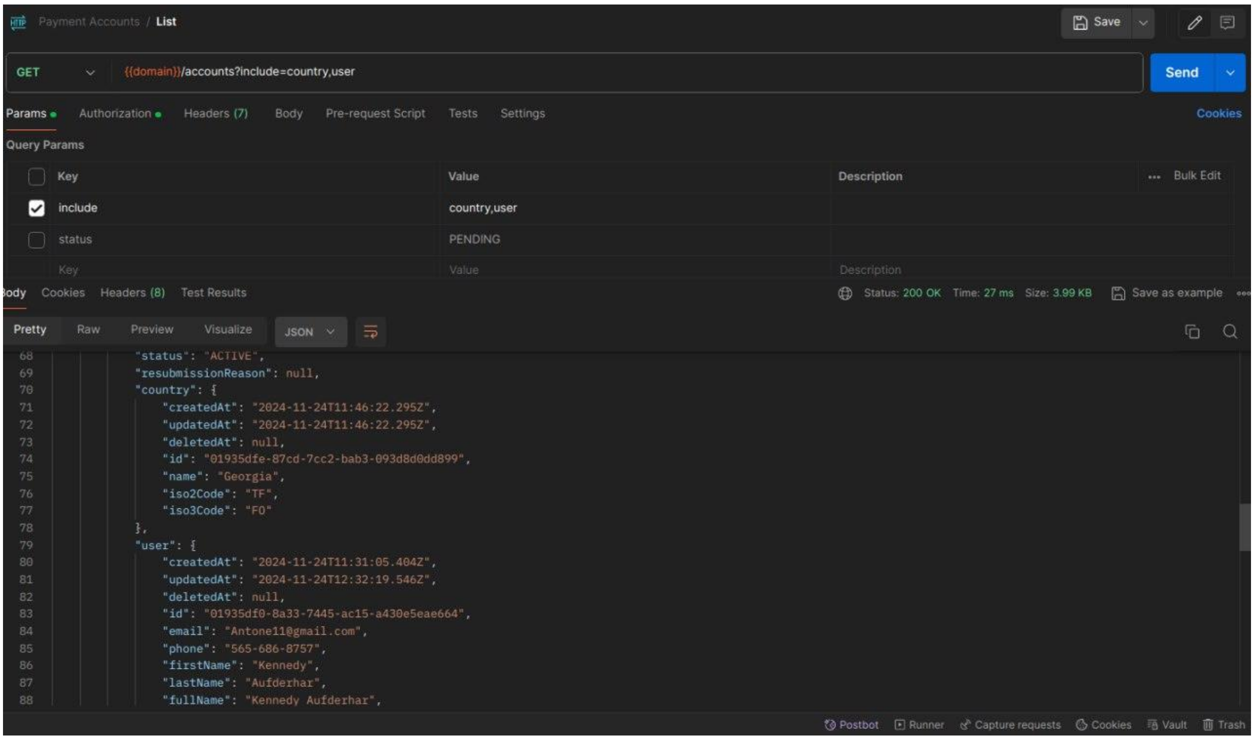


Рисунок 3.20 – Список аккаунтів з можливістю фільтрації по статусам (наприклад, «PENDING»)

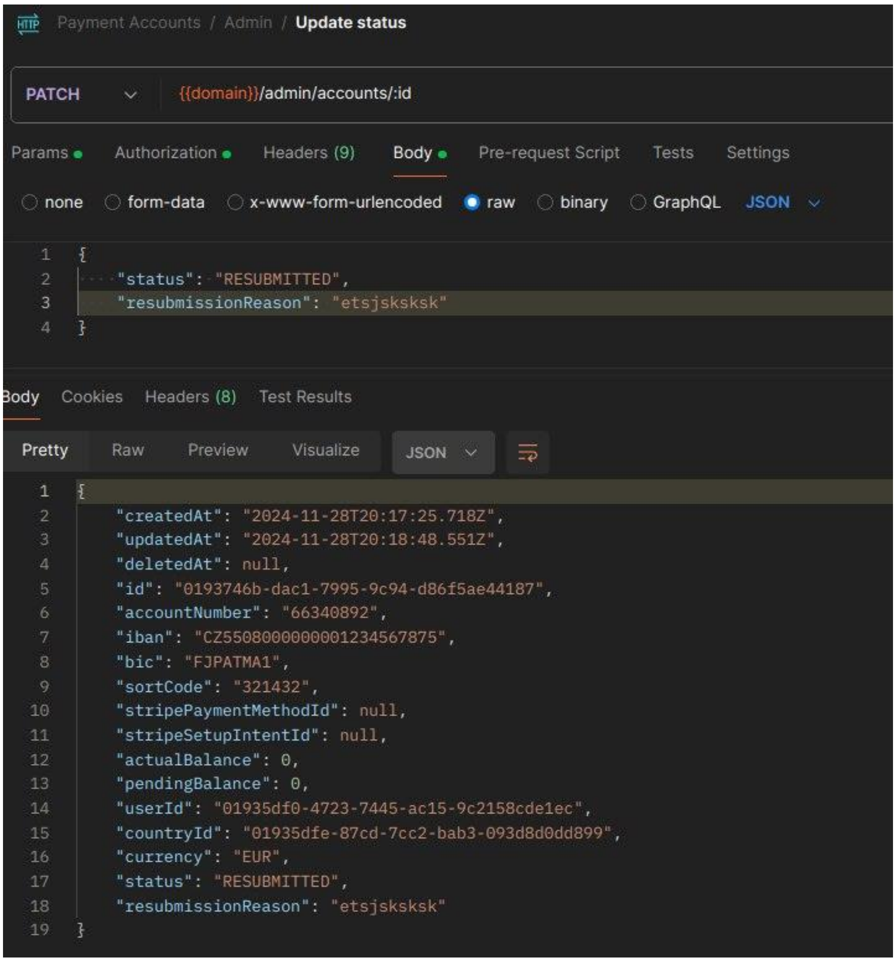


Рисунок 3.21 – Перевідкриття аккаунту з вказанням причини

Створення транзакції: користувач може створити будь-яку транзакції з доступними в системі типами (див. рис. 3.22 – 3.27).

```
const { raw: result } = await this.transactionRepository
  .createQueryBuilder()
  .setQueryRunner(runner)
  .useTransaction(true)
  .insert()   Huhlia, 11/18/24, 4:07 PM • - Added transaction mar
  .values(transactionDto)
  .returning(pgReturning(this.transactionRepository))
  .execute();
```

Рисунок 3.22 – Код створення запису транзакції в базі даних

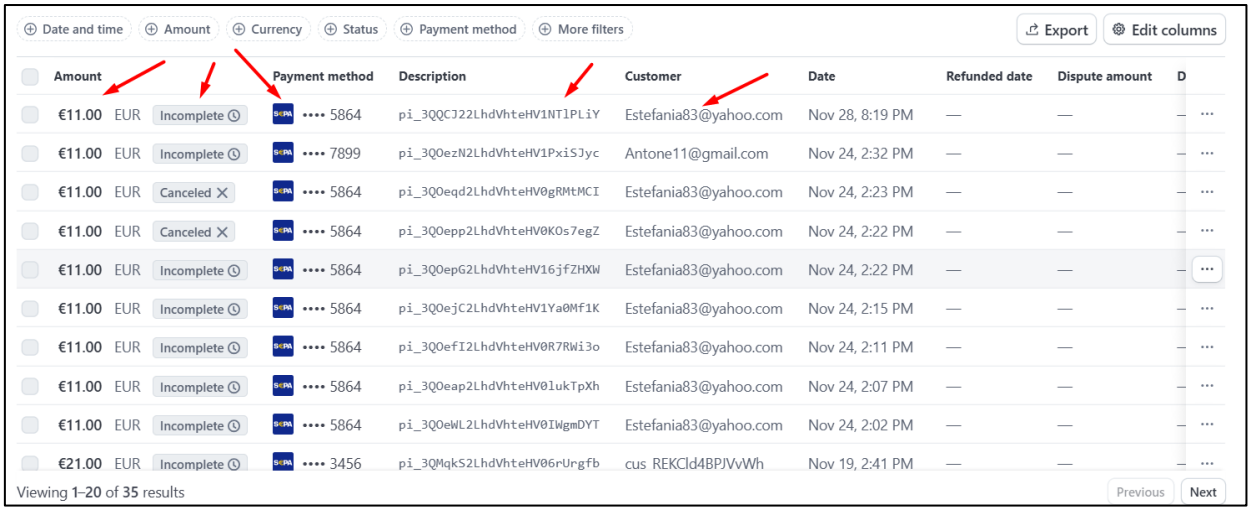


Рисунок 3.23 – Створення платіжного наміру в Stripe

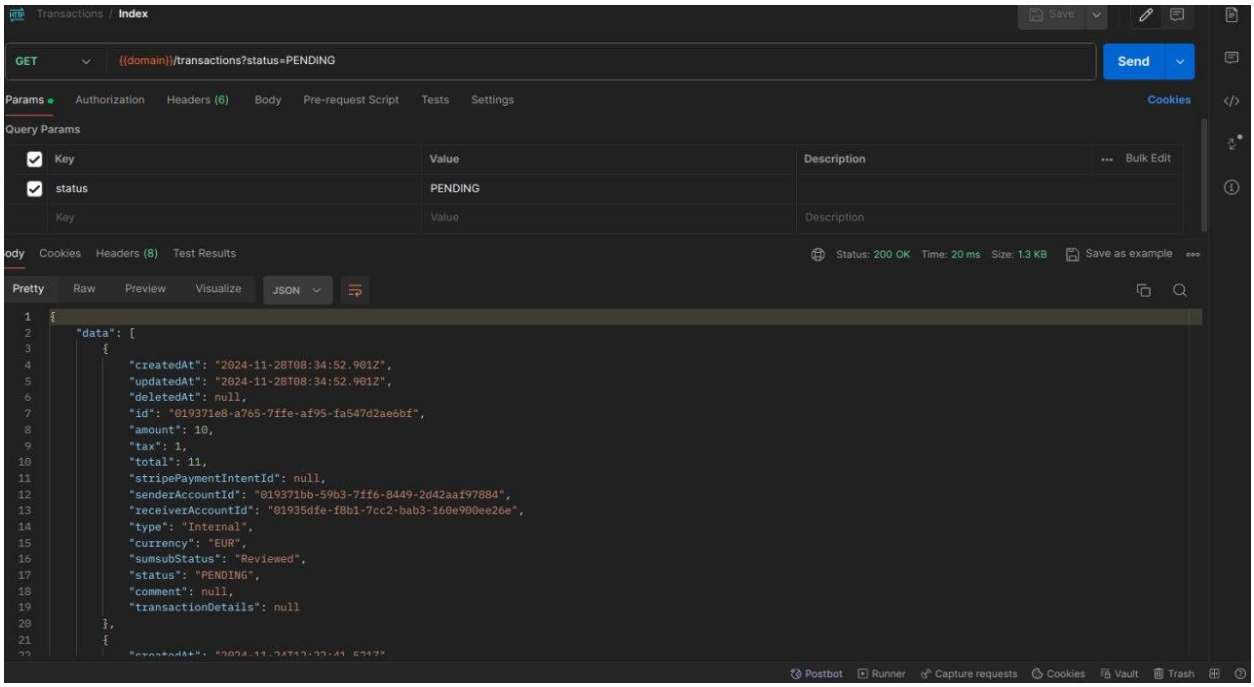


Рисунок 3.24 – Новостворена транзакція з значенням «total» = 11

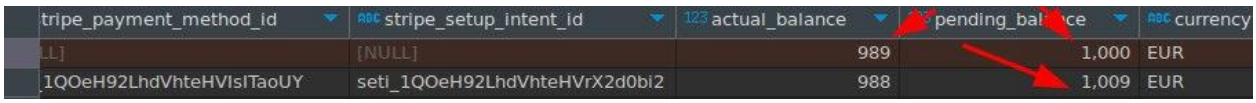


Рисунок 3.25 – Зміна балансів банківських аккаунтів відправника і отримувача в базі даних



Рисунок 3.26 – Створені комісійні транзакції в базі даних від фінансових трансферів

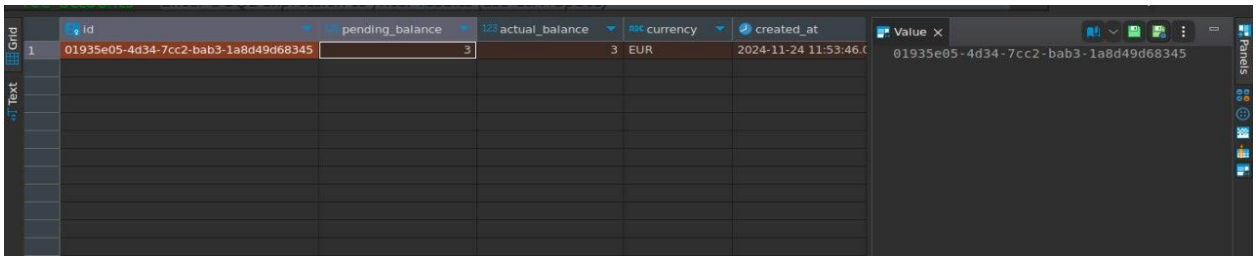


Рисунок 3.27 – Змінений невизначений баланс аккаунта для комісій після створення комісійної транзакції

Якщо аккаунт отримувача є активованим (статус «APPROVED») з певними введеними даними і однаковою з аккаунтом відправника, валютою є в системі, фонди відправляться напряму в його аккаунт, і баланс банківського аккаунта отримувача може відслідковувати користувач, який створював аккаунт або будь-який з адміністраторів.

Якщо аккаунт отримувача не був знайдений, то користувач зможе відправити фонди напряму на банківський рахунок, але тільки якщо тип транзакції будь-який доступний від «Internal». В платіжній системі Stripe буде створений платіжний намір з дефолтним статусом «Pending» і його ID буде згодом записаний у запис в базі у відповідне поле.

Разом з транзакцією буде створені два записи в таблиці «Ledger» с бухгалтерською інформацією для аккаунтів відправника і отримувача, якщо банківський аккаунт отримувача є в системі, і тільки одна (для відправника), якщо отримувач – це банківський рахунок, якого немає в системі.

Також в базі буде створена комісійна транзакція, якщо поле «tax» в тілі запиту було присутнє або там було не нульове значення, і якщо аккаунт для комісій, створюваний адміністраторами, з валютою в якій проводиться транзакція, був створений.

Якщо комісійна транзакція була створена, то невизначений баланс аккаунта для комісії буде збільшений на значення поля «tax». Статус транзакції за замовчуванням буде «PENDING» («На розгляді») і вона має бути або прийнята, або відхилена адміністратором, щоб пройти повний цикл.

Невизначений і актуальний баланси аккаунтів відправника (і отримувача, якщо такий є в системі) будуть змінені: актуальний баланс зменшиться на значення поля «total» (вираховується на стороні клієнта (frontend)), яке є сумою поля «amount» (сума без урахування комісії) та поля «tax» (значення комісії; може бути процентним значенням, або фіксованим, або і те, і те одразу); якщо отримувач – це банківський аккаунт в системі, то його невизначений баланс буде збільшений на значення поля «amount».

Під кінець операції, буде створений бекграунд джобою запис в таблиці «Beneficiaries», який буде прив’язаний до відправника зовнішнім ключем «userId». Це по суті шаблон отримувача для потенційних майбутніх транзакцій, якщо відправник захоче повторити транзакцію для того ж самого отримувача, але з іншою сумою трансферу, наприклад. Він буде містити одне поле з двох: або accountId – ID аккаунта отримувача в системі – або transactionDetails – об’єкт, який містить всю необхідну інформацію банківського рахунку, який не є в системі. Таким чином, користувач звільняє себе від повторного заповнення всіх необхідних деталей отримувача для нової транзакції, бо ця інформація вже зберіглася в базі даних.

Всі вищеописані операції виконуються в рамках однієї суцільної СКБД транзакції з рівнем ізоляції «read committed», що дозволяє «відмінити» всі операції у випадки збою на сервері або в базі, і не дозволяє іншим паралельним СКБД транзакціям отримати доступ до записів, які якимось чином зчитуються або модифікуються в рамках СКБД транзакції, яка на поточний момент

працює з цими записами, допоки поточна транзакція не закінчить і не вивільнить «замок» (від англ. – «lock»), і тим самим забезпечує цілісність і валідність даних у базі.

Створення комісійного правила: адміністратор може створювати, змінювати і видаляти комісійні правила (див. рис. 3.28 – 3.29).

Вводиться унікальна комбінація з полів «type» (тип фінансової транзакції) та «currency» (валюта в якій проводиться фінансова транзакція), а також за бажанням, числові значення «fixedRate» (фіксована ставка, завжди додається як статичне число до суми транзакції) і «taxPercent» (процентна ставка, вираховується по формулі знаходження процентів від числа від суми фінансової транзакції).

Якщо поля «fixedRate» або «taxPercent» відсутні в тілі запиту при створенні правила, ставиться значення «0.00» за замовчуванням. Якщо таке правило вже існує, повертається 400 помилка «Fee rule already exists».

Фронтенд при розрахунку остаточної суми транзакції, робить запит на сервер і шукає комісійне правило за типом і валютою транзакції, яку на поточний момент намагається виконати користувач. Якщо правило не знайдено, то значення «tax» буде 0.00. Якщо знайдено, то вираховується тотальна сума трансферу з урахуванням комісії.

```
public async create(dto: CreateFeeRuleDTO): Promise<FeeRules | never> {  Show usages  Vladislav Huhlia
  try {
    const item :FeeRules  = this.repository.create(dto);

    await this.repository.save(item);

    return item;
  } catch (e: any) {
    if (e?.code === '23505') {
      throw new BadRequestException( objectOrError: 'Fee rule already exists');
    } else {
      throw e;
    }
  }
}
```

Рисунок 3.28 – Код для створення комісійного правила

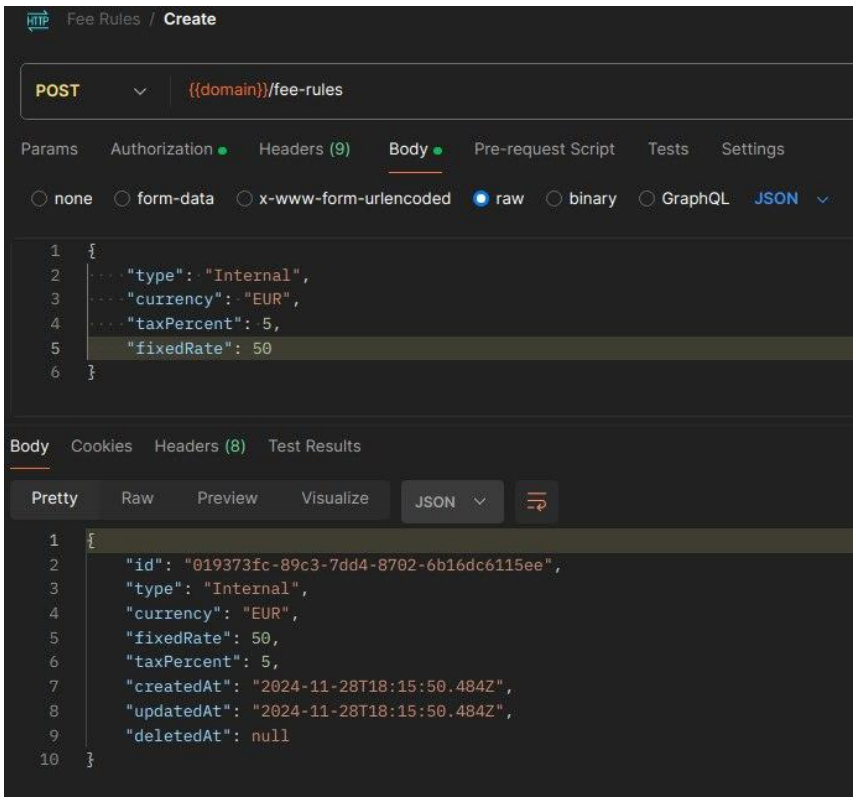


Рисунок 3.29 – Успішне створення комісійного правила

Створення аккаунта для комісій: адміністратор може створювати, і видаляти аккаунти для комісій (див. рис. 3.30 – 3.31).

```
public async create(dto: CreateFeeAccountDTO): Promise<FeeAccount | never> {
  try {
    const item :FeeAccount  = this.repository.create(dto);

    await this.repository.save(item);

    return item;
  } catch (e: any) {
    if (e?.code === '23505') {
      throw new BadRequestException(
        objectOrError: 'Fee account with such currency already exists',
      );
    } else {
      throw e;
    }
  }
}
```

Рисунок 3.30 – Код для створення аккаунта для комісії

Для створення в тілі запиту потрібно лише ввести значення валюти, значення яких доступні в системі. Треба зазначити, що поле «currency» в

таблиці «Fee Accounts» – унікальне, тобто, не може бути більше ніж одного комісійного аккаунта задля уникнення плутанини при визначенні, в який з аккаунтів має піти комісійна транзакція. Якщо аккаунт з такою валютою вже існує, повернеться помилка «Fee account with such currency already exists».

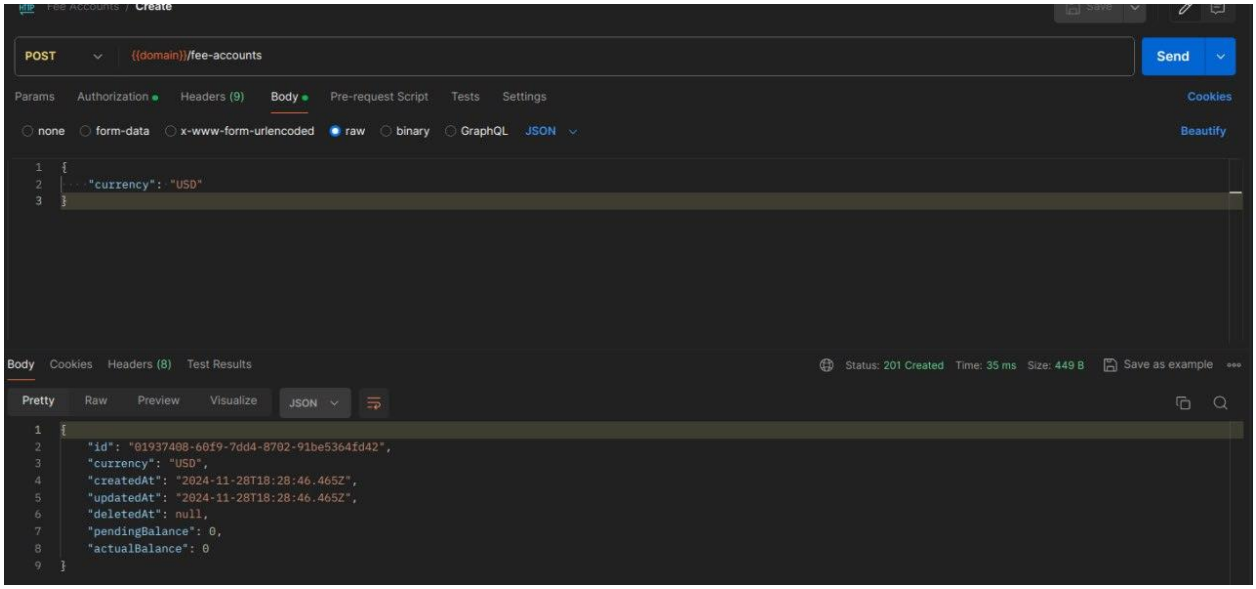


Рисунок 3.31 – Успішне створення аккаунта для комісій

Зміна статусу транзакцій: адміністратор може схвалювати чи відхиляти фінансові транзакції у статусі «На розгляді» (див. рис. 3.32 – 3.33). Він може продивитись всі транзакції з статусом «На розгляді» на адмін-панелі. Доступні значення в тілі запиту тільки «APPROVED» і «REJECTED».

Якщо адміністратор ***схвалює*** транзакцію, то платіжний намір в платіжній системі «Stripe» підтверджується; невизначений баланс банківського аккаунта відправника зменшується на значення поля «total» в транзакції, якій змінюється статус; якщо отримувачем є банківський аккаунт, визначений в системі, то його актуальний баланс збільшується на значення поля «amount» транзакції. Якщо поле «tax» у транзакції не нульове, то комісійна транзакція змінює свій статус на «APPROVED» і актуальний баланс комісійного банківського аккаунта збільшується на значення «tax» транзакції.

Якщо адміністратор ***відхиляє*** транзакцію, то платіжний намір в платіжній системі «Stripe» скасовується; невизначений баланс банківського

аккаунта відправника збільшується на значення поля «total» в транзакції (тобто, фонди повертаються), якій змінюється статус; якщо отримувачем є банківський аккаунт, визначений в системі, то його актуальний баланс не змінюється, а невизначений баланс зменшується на значення поля «amount» транзакції. Якщо поле «tax» у транзакції не нульове, то комісійна транзакція змінює свій статус на «REJECTED» і актуальний баланс комісійного банківського аккаунта зменшується на значення «tax» транзакції.

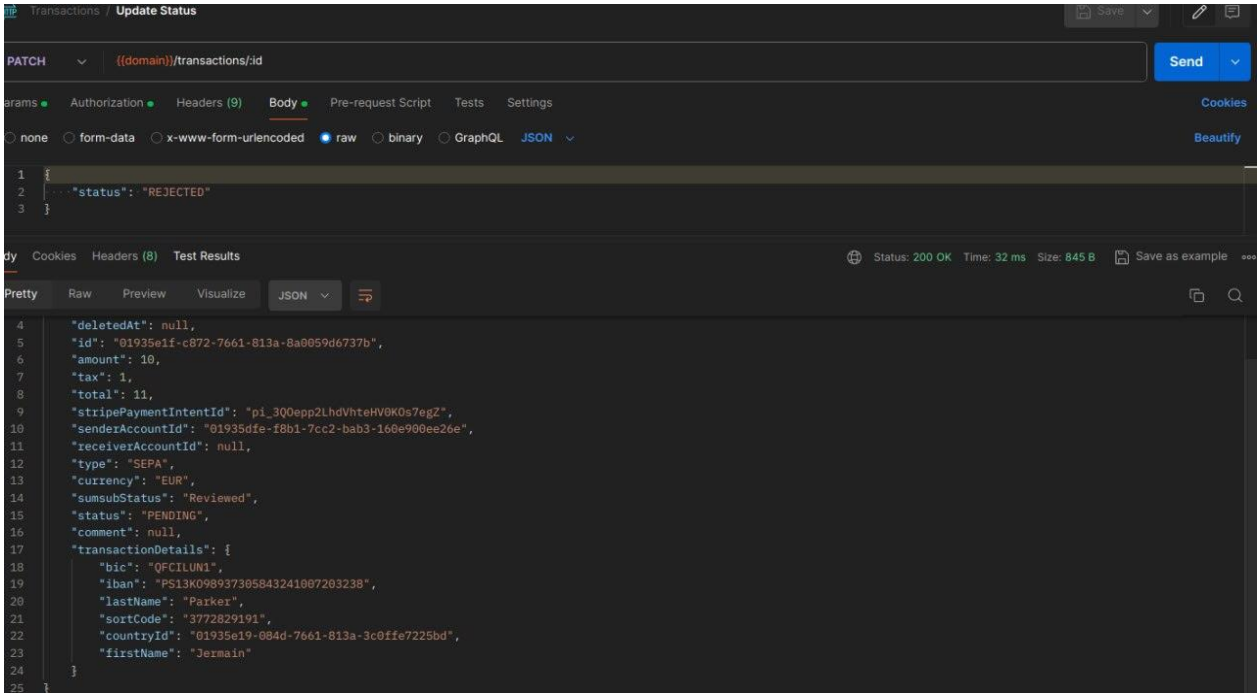


Рисунок 3.32 – Відхилення транзакції

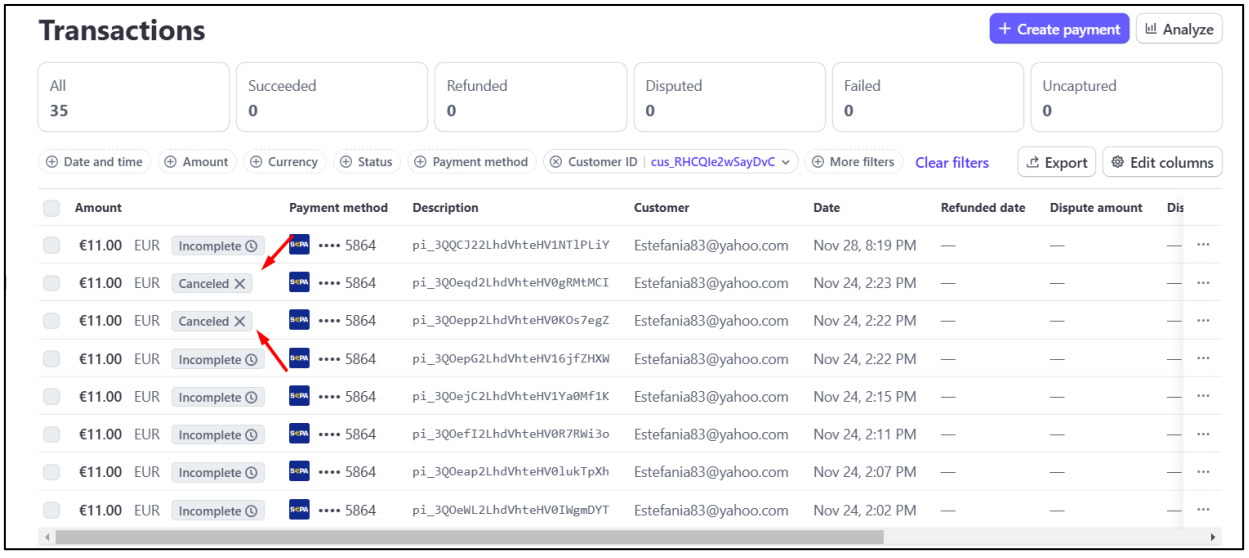


Рисунок 3.33 – Скасований платіжний намір в Stripe

В найкращому варіанті, якщо в обох банківських аккаунтах немає транзакцій з статусом «На розгляді», то баланси повинні співпасти: у аккаунта-відправника – невизначений баланс дорівнює актуальному балансу; теж саме і у аккаунта-отримувача.

Перегляд бенефіціарів (шаблони для майбутніх транзакцій): як вже зазначено в частині поточного підрозділу «Створення транзакції», запис бенефіціара буде автоматично створений після проведення будь-якої фінансової транзакції, а після створення запису в основній БД, додатковий документ буде створений в СКБД Elasticsearch.

Користувач може продивитись весь список своїх шаблонів, відфільтрувати (наприклад, по типу транзакції) і відсортувати (наприклад, по даті створення) їх за різними полями.

Якщо він захоче знайти якийсь шаблон по частковому значенню полю «fullName», з можливістю допущення похибок, то буде задіяний повнотекстовий пошук, в якому нам допоможе Elastic. Elastic знайде всі можливі співпадіння (так звані «хіти»), і поверне значення ID документів – хітів, які дорівнюють значенням ID записів бенефіціарів з подібними до запитуваного часткового тексту поля «fullName». Потім, в PostgreSQL можна виконати використати запит типу «<field_name> IN (…<ids>)», щоб знайти співпадіння з знайденими документами в Elasticsearch (див. рис. 3.34).

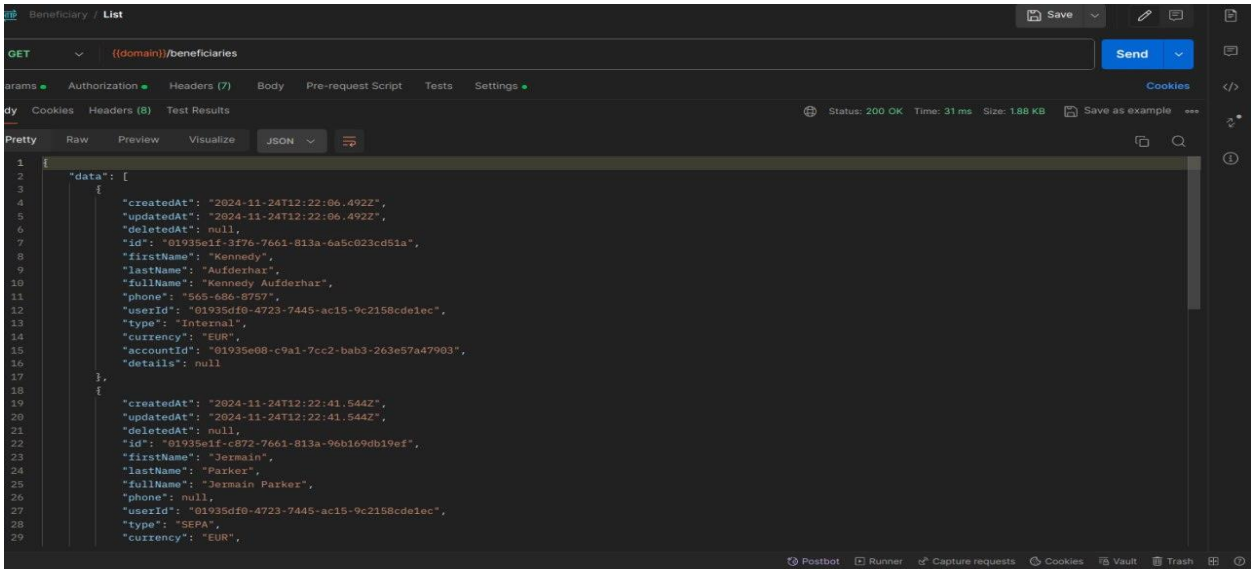


Рисунок 3.34 – Список усіх бенефіціарів користувача

3.2 Контейнеризація і деплой у хмару

Docker конфігурація: проєкт можна контейнеризувати і запустити через Докер. Спочатку, Докер побудує Docker Image (див. рис. 3.40), який описаний в докер-файлі.

В докерфайлі спочатку створюється і робиться активною в файловій системі контейнера нова директорія, де і будуть зберігатися файли проєкта. Потім в цю дерикторію копіюються package.json, package-lock.json, yarn.lock (ці всі файли необхідні для коректного встановлення залежностей).

Після цього пакетним менеджером apk встановлюється headless, легковажний браузер Chromium, який потрібен для генерації квитанцій і інвойсів.

Після цього запускається команда «yarn install --only=production --frozen-lockfile»,яка встановлює залежності і бібліотеки, які об’явлені в package.json і yarn.lock файлах.

Після цього до поточної директорії копіюються усі інші файли, включно з папкою «node_modules», яка з’явилася після попереднього кроку.

Фінальним кроком імеджу в докерфайлі є запуск команди «yarn build», яка транс-компілює ТайпСкрипт код у зрозумілий для браузера і Node.js, JavaScript код.

Сам контейнер серверу стартує з команди «yarn run start:dev», який запускає збудований командою «yarn build:server» код (тобто, сервер) у watch моді, який перезапускає сервер при кожній зміні коду в реальному часі.

Перед запуском контейнера сервера, спочатку піднімається контейнер з PostgreSQL (основна база даних), Redis (база даних, яка слугує, як месседж брокер в нашому проєкті, і як шар кешування), Elasticsearch (база даних для повнотекстового пошуку), Kibana (GUI портал для Elasticsearch для візуалізації даних). Після того, як контейнер з сервером і іншими контейнерами активні, нарешті піднімається контейнер з клієнтом, який будує вебпортал, який і буде першим використовувати потенційний користувач (див. рис. 3.35 – 3.36).

```dockerfile
FROM node:21-alpine AS development

WORKDIR /app

COPY package*.json yarn*.lock ./

RUN apk update && \
    apk add --no-cache \
    chromium

RUN --mount=type=cache,target=/root/.yarn YARN_CACHE_FOLDER=/root/.yarn yarn install --only=production --frozen-lockfile

COPY . .

RUN yarn build

# Stage 2: Run the application
FROM node:21-alpine As production

ARG NODE_ENV=production
ENV NODE_ENV=${NODE_ENV}

WORKDIR /app

COPY package.json yarn.lock ./

COPY --from=development /app/dist ./dist   Huhlia, 11/25/24, 1:17 PM • - Added Invoices and Receipts generation support

CMD [ "yarn", "start:prod" ]
```

Рисунок 3.35 – Dockerfile для контейнера сервера

```yaml
  server:
    container_name: server
    restart: always
    build:
      context: ./server
      dockerfile: Dockerfile
      target: development
    ports:
      - "${PORT}:${PORT}"
    env_file: '.env'
    depends_on:
      - redis
      - kibana
      - elastic
      - db
    volumes:
      - ./server:/app
      - /app/node_modules
    networks:
      - shared_network
    command: yarn run start:dev

  db:
    container_name: db
    image: postgres:17-alpine
    restart: always
    ports:
      - '5432:5432'
    env_file: '.env'
    environment:
      POSTGRES_USER: ${DB_USERNAME}
      POSTGRES_PASSWORD: ${DB_PASSWORD}
      POSTGRES_DB: ${DB_DATABASE}
    volumes:
      - pgdata:/var/lib/postgresql/data
    networks:
      - shared_network
```

Рисунок 3.36 – Фрагмент Docker Compose файлу

Kubernetes конфігурація: Kubernetes конфігурація майже повністю повторює конфігурацію Docker.

Було створено три деплоймент конфігурації, 4 сервіс конфігурації, 1 крон джоба, 1 storage class і 1 persistent volume claim.

Одна деплоймент конфігурація створена для контейнерів з кодом фронтенду. Друга – з кодом бекенду. В обох конфігураціях значення replica стоїть на 2 для балансування навантаження і більшої надійності. Третя – з описом контейнеру Elasticsearch.

Для того, щоб функціонал фронтенду мав доступ до інтернету і балансував запити між двома репліками, був створений сервіс з типом Load Balancer (external), який після деплою в хмарний Kubernetes cluster, через публічний IP викриває frontend-поди в публічний інтернет.

Для внутрішньої комунікації і балансування навантаження між фронтендом і бекендом був створений Load Balancer (internal), який недоступний з зовнішнього інтернету, але доступний для комунікації з усіма елементами всередині Kubernetes кластеру.

Для того, щоб backend приймав івенти на вебхук від Stripe і Sumsub з публічної мережі, був створений ще один Load Balancer (External).

Для комунікації між backend-подами і Elasticsearch-подом був створений сервіс з типом ClusterIP, який дає випадкове значення IP для внутрішньої комунікації.

Для того, щоб дані Elasticsearch були постійними і не зникали, була створена конфігурація для Azure Disk Storage Class який виділяє мережевий диск ємністю на 5 Гб, а також, persistent volume claim, який резервує 2 Гб з цих 5 саме на цьому диску и прив’язує до Elasticsearch поду.

Також була створена конфігурація під крон джобу (контейнер, який автоматично запускається з вказаним інтервалом), яка один раз на добу в 00:00 робить бекап бази даних, архівує його і завантажує в S3 сховище (див. рис. 3.37 – 3.41).

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: frontend-deployment
  labels:
    app: webapp
spec:
  replicas: 2
  selector:
    matchLabels:
      tier: frontend
  template:
    metadata:
      name: client
      labels:
        tier: frontend
    spec:
      containers:
        - name: client
          image: googlevlad/easypay-frontend:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 3001
          command: [ "npm", "run", "start" ]
          args: ["--env-file=.env"]
          resources:
            requests:
              cpu: "500m"
```

Рисунок 3.37 – Конфігурація для Frontend деплою

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: backend-deployment
  labels:
    app: webapp
spec:
  replicas: 2
  selector:
    matchLabels:
      tier: backend
  template:
    metadata:
      name: server
      labels:
        tier: backend
    spec:
      containers:
        - name: server
          image: googlevlad/easypay-backend:latest
          imagePullPolicy: Always
          ports:
            - containerPort: 3000
          command: [ "yarn", "run", "start:dev" ]
          resources:
            requests:
              cpu: "500m"
```

Рисунок 3.38 – Конфігурація для Backend деплою

```yaml
apiVersion: batch/v1
kind: CronJob
metadata:
  name: db-backup
spec:
  schedule: "0 0 * * *"
  successfulJobsHistoryLimit: 1
  failedJobsHistoryLimit: 1
  jobTemplate:
    spec:
      template:
        spec:
          containers:
            - name: db-backup
              image: googlevlad/easypay-backup:latest
              imagePullPolicy: IfNotPresent
              command: ["/bin/bash", "-c", "sh ./backup.sh"]
          restartPolicy: OnFailure
```

Рисунок 3.39 – Конфігурація для крон джоби для бекапу

```yaml
apiVersion: v1
kind: Service
metadata:
  name: frontend-svc
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-resource-group: "rg-easypay-hardy-goblin"
    service.beta.kubernetes.io/azure-load-balancer-ipv4: "40.79.27.189"
spec:
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 3001
      targetPort: 3001
  selector:
    tier: frontend
```

Рисунок 3.40 – Конфігурація для Load Balancer сервісу для фронтенд деплою

```yaml
apiVersion: v1
kind: Service
metadata:
  name: backend-external-lb
  annotations:
    service.beta.kubernetes.io/azure-load-balancer-resource-group: "rg-easypay-hardy-goblin"
    service.beta.kubernetes.io/azure-load-balancer-ipv4: "40.79.26.208"
spec:
  type: LoadBalancer
  ports:
    - protocol: TCP
      port: 3000
      targetPort: 3000
  selector:
    tier: backend
```

Рисунок 3.41 – Конфігурація для Load Balancer сервісу для бекенд деплою

Terraform конфігурація. В якості cloud провайдера для Terraform конфігурації був обраний Azure Cloud від Microsoft, через доволі демократичну цінову політику на ресурси у хмарі, які він пропонує, а також різномаїття з різними сервісами і в тому числі, інтеграція з Kubernetes.

Мова піде за основні елементи, що описуються в конфігурації. По перше створюється окрема ресурс група під всі сервіси проєкту. Далі створюється окрема віртуальна мережа, в ній 3 підмережі: одна – під Kubernetes cluster, друга – під PostgreSQL БД і третя – під Redis cluster.

Далі створюється безпосередньо AKS (Azure Kubernetes Service) кластер, куди і буде деплоїтись описана раніше Kubernetes інфраструктура. Після цього створюється PostgreSQL Flexible Server в окремій підмережі.

Для того, щоб backend поди з одної підмережі могли комунікувати з PostgreSQL базою даних в іншій підмережі, було створено приватне сполучення з нетворк конфігурацією з правилом, яке пропускає трафік який надходить від CIDR підмережі AKS кластеру до CIDR підмережі, де розташований PostgreSQL сервер. Аналогічна ситуація і з Redis кластером.

Для того, щоб External Load Balancer сервіси Kubernetes отримували статичні IP адреси, а не випадкові, було створено дві публічні IP адреси, поверх яких додатково було створено ще два DNS записи для більш зручної маршрутизації (див. рис. 3.42 – 3.45).

```
resource "azurerm_postgresql_flexible_server" "db_server" {  7 usages
  name                = "${random_string.db_random.result}-postgresql"
  location            = var.location
  resource_group_name = azurerm_resource_group.rg.name

  sku_name   = var.pg_sku_name
  version    = var.pg_version
  storage_mb = var.pg_storage

  administrator_login    = var.pg_login
  administrator_password = var.pg_password
}

resource "azurerm_postgresql_flexible_server_configuration" "require_secure_transport" {  no usages
  name      = "require_secure_transport"
  value     = "OFF"
  server_id = azurerm_postgresql_flexible_server.db_server.id

  depends_on = [
    azurerm_postgresql_flexible_server.db_server
  ]
}
```

Рисунок 3.42 – Конфігурація для створення PostgreSQL серверу

```
resource "azurerm_kubernetes_cluster" "k8s" {  no usages
  name                = var.aks_name
  location            = azurerm_resource_group.rg.location
  dns_prefix          = var.aks_dns_prefix
  private_cluster_enabled = var.private_cluster
  resource_group_name = azurerm_resource_group.rg.name
  sku_tier            = var.aks_sku_tier

  linux_profile {
    admin_username = var.vm_user_name

    ssh_key {
      key_data = file(var.public_ssh_key_path)
    }
  }

  default_node_pool {
    name            = "agentpool"
    node_count      = var.aks_agent_count
    vm_size         = var.aks_agent_vm_size
    os_disk_size_gb = var.aks_agent_os_disk_size
    vnet_subnet_id  = data.azurerm_subnet.kubesubnet.id
  }

  network_profile {
    network_plugin    = "kubenet"
    dns_service_ip    = var.aks_dns_service_ip
    service_cidr      = var.aks_service_cidr
```

Рисунок 3.43 – Конфігурація для AKS кластеру

```
resource "azurerm_virtual_network" "vnet" {  6 usages
  name                = var.vnet_name
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  address_space       = var.vnet_address_space

  subnet {
    name             = var.aks_subnet_name
    address_prefixes = var.aks_subnet_address_prefix
  }
}

resource "azurerm_role_assignment" "vnet_network_contributor" {  no usages
  scope                = azurerm_virtual_network.vnet.id
  role_definition_name = "Network Contributor"
  principal_id         = var.service_principal

  depends_on = [
    azurerm_virtual_network.vnet
  ]
}

data "azurerm_subnet" "kubesubnet" {  4 usages
  name                 = var.aks_subnet_name
  virtual_network_name = azurerm_virtual_network.vnet.name
  resource_group_name  = azurerm_resource_group.rg.name
  depends_on           = [azurerm_virtual_network.vnet]
}
```

Рисунок 3.44 – Конфігурація для віртуальної мережі і підмереж

```
resource "azurerm_redis_cache" "redis_cache" {  1 usage
  name                = "${random_string.redis_random.result}-redis"
  location            = azurerm_resource_group.rg.location
  resource_group_name = azurerm_resource_group.rg.name
  capacity            = var.redis_capacity
  family              = var.redis_sku_family
  sku_name            = var.redis_sku_name
}

resource "azurerm_private_endpoint" "redis_endpoint" {  1 usage
  name                = "${random_string.redis_random.result}-endpoint"
  location            = var.location
  resource_group_name = azurerm_resource_group.rg.name
  subnet_id           = azurerm_subnet.redis_subnet.id

  depends_on = [
    azurerm_subnet.redis_subnet
  ]
```

Рисунок 3.45 – Конфігурація для створення PostgreSQL серверу

ВИСНОВКИ

У кваліфікаційній роботі була розглянута тема призначення fintech додатків, актуальність їхньої розробки у сучасному світі. Були визначені і описані потенційні тест-кейси використання розроблюваного вебзастосунку. Проведений аудит технологій, задіяних у розробці даного вебдодатку і описані їх слабкі і сильні сторони. Також був обґрунтований вибір інфраструктурних рішень, задіяних при плануванні деплою у клауд. Разом з цим, були представлені і описані зв'язки між таблицями в базі даних та структура кожної з них окремо. В завершаючому розділі основного контенту, був наведений опис роботи бізнес-логіки основних функцій вебзастосунку разом з супроводжуючою візуалізацією.

Основний стек технологій – рантайм для мови програмування JavaScript NodeJS, фреймворк для написання серверної частини NestJS, бази даних PostgreSQL, клауд платформи AWS та Azure, NoSQL бази даних Redis та Elasticsearch. Всім їм знайшлось місце в цій роботі для детального опису і обґрунтування вибору для розробки.

Метою роботи було створення функціонального та ефективного fintech вебпорталу, який би задовольняв потреби як індивідуальних користувачів, так і бізнесу у бажанні мати зручний додаток для виконання фінансових транзакцій у веб форматі.

Під час розробки fintech-додатку були реалізовані основні функціональні можливості, такі як реєстрація та авторизація користувачів, створення і синхронізація банківських аккаунтів, створення транзакцій, генерація квитанцій та інвойсів, схвалення або відхилення банківських аккаунтів/транзакцій. Було також забезпечено можливість пошуку бенефіціарів, транзакцій, банківських аккаунтів і фільтрації за різними параметрами та відображення деталей цих сутностей.

Додаток був повністю протестований на наявність критичних помилок і запевнив у коректній роботі додатку і відповідності заявленим критеріям якості.

Результати роботи свідчать про те, що розроблений банківський вебдодаток є функціональним та ефективним рішенням для онлайн-торгівлі. Використання NodeJS та NestJS, та інших інструментів, описаних в цій роботі, дозволяє створити масштабований та легко розширюваний додаток, який може задовольнити потреби дуже широкого кола користувачів: як індивідуальних клієнтів (в якості додатку, що дозволяє легко і швидко проводити різни типи міжнародних транзакцій), так і стейкхолдерів застосунку (отримувати дохід з комісій транзакцій, зробленими індивідуальними користувачами).

Однак, варто відзначити, що розроблений додаток має потенціал для подальшого розширення та вдосконалення. Наприклад, можна додати дата пайплайн з потоками для подальшої аналітики в data warehouse; додати можливість проведення трансферів за допомогою криптовалют і т.д.. Крім того, можна вдосконалити процес оптимізації швидкості виконання транзакції та покращити безпеку системи на стороні бізнес-логіки.

У цілому, розробка fintech-додатку з використанням NodeJS та NestJS є актуальною та перспективною задачею. Технології, використані у цій роботі, дозволяють створювати потужні та ефективні додатки, які можуть привернути широке коло користувачів і забезпечити успішну систему онлайн-банкінгу.

ПЕРЕЛІК ПОСИЛАНЬ

1. ER diagram link. *Google Drive*. URL: https://drive.google.com/file/d/11nMgNHF3MDuLjtKzrQm12e9T99ePMcSA/view?usp=sharing (дата звернення: 08.09.2024).

2. Infrastructure diagram link. *Google Drive*. URL: https://drive.google.com/file/d/11nMgNHF3MDuLjtKzrQm12e9T99ePMcSA/view?usp=sharing (дата звернення: 09.09.2024).

3. Node.js docs. *Node.js Introduction*. URL: https://nodejs.org/en/learn/getting-started/introduction-to-nodejs (дата звернення: 10.09.2024).

4. NestJS Introduction. *NestJS Documentation*. URL: https://docs.nestjs.com/ (дата звернення: 11.09.2024).

5. TypeORM Homepage. *TypeORM Documentation*. URL: https://typeorm.io/ (дата звернення: 12.09.2024).

6. PostgreSQL Homepage. *PostgreSQL Documentation*. URL: https://www.postgresql.org/docs/17/index.html (дата звернення: 13.09.2024).

7. Docker «Get Started» page. *Docker Documentation*. URL: https://docs.docker.com/get-started/ (дата звернення: 14.09.2024).

8. Redis Homepage. Redis Documentation. URL: https://redis.io/docs/latest/ (дата звернення: 15.09.2024).

9. Kubernetes Overview. *Kubernetes Documentation*. URL: https://kubernetes.io/docs/concepts/overview/ (дата звернення: 16.09.2024).

10. Terraform Intro. *Terraform Documentation*. URL: https://developer.hashicorp.com/terraform/intro (дата звернення: 17.09.2024).

11. Stripe Homepage. *Stripe Documentation*. URL: https://docs.stripe.com/ (дата звернення: 18.09.2024).

12. Elasticsearch Homepage. *Elasticsearch Documentation*. URL: https://typeorm.io/ (дата звернення: 19.09.2024).

13. SumSub Homepage. SumSub Documentation. URL: https://docs.sumsub.com/ (дата звернення: 20.09.2024).

14. Git Repository with project. *GitHub*. URL: https://github.com/VladGoogle/easypay (дата звернення: 21.09.2024).

15. Гугля В. Е. Розробка E-commerce порталу з використанням React та NodeJS : кваліфікаційна робота бакалавра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник О. М. Мильцев. Запоріжжя : ЗНУ, 2023. 67 с. URL: https://dspace.znu.edu.ua/jspui/handle/12345/12738 (дата звернення: 21.09.2014).

16. Prokhasko M. How Do Mobile and Online Banking Services Benefit Your Bank? 2020. *Kindgeek*. URL: https://tinyurl.com/438mac54 (дата звернення: 21.09.2024).

17. Pala P., Puczyk A. 5 Reasons Why Mobile Banking is Popular. 2022. *Neontri*. URL: https://neontri.com/blog/mobile-banking-popularity/ (дата звернення: 21.09.2024).