ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програмного забезпечення для Web-сервісу керування фінансами та витратами з використанням C# та React»

на здобуття освітнього ступеня бакалавра
зі спеціальності 121 Інженерія програмного забезпечення
освітньо-професійної програми «Інженерія програмного забезпечення»

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Олександр ПАНЧЕНКО
(підпис)

Виконав: здобувач вищої освіти групи ПД-41

Олександр ПАНЧЕНКО

Керівник: Олег ІЛЬЇН
д.т.н., професор

Рецензент: _____

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Інженерії програмного забезпечення

Ступінь вищої освіти Бакалавр

Спеціальність 121 Інженерія програмного забезпечення

Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Інженерії програмного забезпечення
_____ Ірина ЗАМРІЙ
« ____ » _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Панченку Олександру Миколайовичу

1. Тема кваліфікаційної роботи: «Розробка програмного забезпечення для Web-сервісу керування фінансами та витратами з використанням C# та React»
керівник кваліфікаційної роботи д.т.н., професор Олег ІЛЬЇН,
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» лютого 2024 р. № 36.
2. Строк подання кваліфікаційної роботи «28» травня 2024 р.
3. Вихідні дані до кваліфікаційної роботи: науково-технічна література, теоретичні відомості про архітектурний стиль мікросервісів та підходу проектування Domain-Driven-Design, технічна документація з описом роботи брокера повідомлень та API-шлюзу.
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)

1. Огляд та аналіз існуючих веб-застосунків.
2. Визначення вимог.
3. Аналіз інструментальних засобів реалізації.
4. Програмна реалізація та опис функціонування веб-застосунку для керування фінансами та витратами.

5. Тестування веб-застосунку.

5. Перелік графічного матеріалу: *презентація*

1. Аналіз аналогів.
2. Вимоги до застосунку.
3. Програмні засоби реалізації.
4. Контекстна діаграма.
5. Діаграма прецедентів адміністратора.
6. Діаграма прецедентів користувача.
7. Діаграма послідовності.
8. Діаграма класів додавання боргу.
9. Датологічні моделі.
10. Мапа сайту.
11. Екранні форми.
12. Апробація результатів дослідження.
13. Відео роботи застосунку.

6. Дата видачі завдання«28» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір та аналіз науково-технічної літератури	28.02.-06.03.2024	
2	Аналіз та дослідження існуючих аналогів	07.03-16.03.2024	
3	Проектування архітектури системи	17.03-23.03.2024	
4	Розробка веб-застосунку для керування фінансами	24.03-25.04.2024	
5	Тестування застосунку	26.04-28.04.2024	
6	Оформлення роботи: вступ, висновки, реферат	29.04-05.05.2024	
7	Розробка демонстраційних матеріалів	06.05-12.05.2024	
8	Попередній захист роботи	13.05-31.05.2024	

Здобувач вищої освіти _____ Олександр ПАНЧЕНКО
(підпис)

Керівник
кваліфікаційної роботи _____ Олег ІЛЬЇН
(підпис)

РЕФЕРАТ

Текстова частина кваліфікаційної роботи на здобуття освітнього ступеня бакалавра: 68 стор., 4 табл., 76 рис., 23 джерел.

Мета роботи – автоматизація процесу управління фінансами та витратами.

Об'єкт дослідження – процес керування фінансами та витратами.

Предмет дослідження – веб-застосунок для керування фінансами та витратами.

Короткий зміст роботи: у цій роботі використано мікросервісну архітектуру та підхід проектування Domain-Driven Design, була спроектована комунікація між мікросервісами за допомогою HTTP та Kafka. Програмно реалізовані функціональні можливості, а саме: додавання боргів, прибутків, витрат, їх видалення та редагування, отримання звітів за певний період, відображання аналітики. Також було реалізовано систему обліку користувачів, для адміністратора системи. Проведено функціональне, регресійне, інтеграційне та E2E тестування застосунку. В роботі використано брокер повідомлень Apache Kafka та API-шлюз Ocelot для забезпечення продуктивності та масштабованості застосунку, бібліотеку React для побудови користувацького інтерфейсу, мову програмування C# на платформі ASP.NET з використанням архітектурного стилю REST API для побудови серверної частини застосунку.

Сферою використання застосунку є фінансові установи, підприємства або індивідуальні користувачі.

ЗМІСТ

ВСТУП

У сучасному світі, де технології пронизують усі сфери життя, керування фінансами стає не менш важливою та невід'ємною складовою життєдіяльності, як і індивідуальних користувачів, так і для бізнес середовища.

Велика кількість людей відчуває проблеми з керуванням своїми фінансами та намагається використовувати різні сервіси для автоматизації управління ними, задля втілення своєї матеріальної мрії, тож попит на такий продукт суттєво зростає, однак зараз існує безліч програм для керування фінансами, але багато з них мають обмежений функціонал або не враховують потреби користувачів.

На відміну від застосунків конкурентів, мій веб-застосунок відрізняється унікальним функціоналом, виправленими недоліками схожих систем, тож користувач не буде зіштовхуватись з попереднім невдалим досвідом.

Отже, обрана тема дипломної роботи є актуальною.

Об'єктом дослідження є процес управління фінансами.

Предметом роботи є програмне забезпечення для автоматизації керування фінансами.

Метою роботи є автоматизація процесу керування фінансами. Щоб досягти цієї мети, я виділив наступні завдання:

1. Дослідити предметну область, проаналізувати ринок фінансового менеджменту, визначити їх переваги та недоліки.
2. Визначити основні вимоги до застосунку, встановлення функціональних та нефункціональних вимог з урахуванням потреб користувачів та недоліків конкурентів.
3. Підібрати технічні засоби, які будуть використовуватись для розробки програмного забезпечення згідно вимогам.
4. Спроектувати та розробити архітектуру застосунку з урахуванням встановлених вимог.
5. Протестувати застосунок на відповідність вимогам.

Методика дослідження: методи обробки інформації, методи взаємодії між сервісами, методи архітектури програмного забезпечення.

Наукова новизна роботи полягає в удосконаленні графічного інтерфейсу користувача, впровадження нових та ефективних технологій, забезпеченні доступу з будь-якого пристрою.

Практична значущість результатів: даний продукт може бути використаний у будь-яких сферах діяльності таких як: фінансові установи, підприємства або індивідуальні користувачі.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Поняття фінансового менеджменту

Фінансовий менеджмент є ключовим елементом підприємств або життєдіяльності особи та представляє собою підхід до планування, аналізу та контролю фінансових ресурсів. Оскільки правильне та ефективне керування фінансами призводить до успіху та стабільності, це є дуже важливою складовою управління як підприємства, так і особи. Передусім в сучасному світі, де економічна ситуація дуже швидко змінюється, гнучкість та здатність швидко реагувати на події є ключовими факторами успіху.

Під час планування розробляються стратегії використання доступних фінансових ресурсів дивлячись на цілі та потреби. Це включає в себе, встановлення фінансових пріоритетів, та планування витрат. Під час планування важливо враховувати не лише звичайні повсякденні потреби, але й майбутні цілі.

Аналіз включає оцінку фінансової ситуації, що ґрунтується на даних з фінансових звітах та ризиках, для прийняття важливих стратегічних цілей.

Контроль забезпечує моніторинг використання фінансових ресурсів на відповідність до поставлених планів та цілей, виявляє відхилення від плану та відстежує виконання бюджетів. За допомогою контролю, можна вчасно помічати проблеми та вирішувати їх задля уникнення серйозних фінансових втрат.

На рівні компанії, фінансовий менеджмент включає управління фінансовими ресурсами з метою досягнення стратегічних цілей. Основні особливості включають бюджетування, управління кредитами та інвестиціями.

Одним із показників успішності фінансового менеджменту є капіталізація компаній. Якщо капіталізація постійно зростає, то це свідчить про ефективне використання ресурсів компанії.

Фінансовий менеджмент людини - це основна складова нашої життєдіяльності. Фінансова стабільність сильно впливають на наше емоційне та соціальне становище, досягнення та комфорт. Неправильні рішення можуть призвести до стресу та проблем. Тому, щоб не допускати цього, людина повинна освоїти фінансову грамотність. Розуміння її основ дозволяє краще керувати своїми заощадженнями, планувати та досягати цілей. Основні особливості включають управління бюджетом та інвестиціями, планування витрат. Щоб навчитися цьому можна почати з літератури, курсів, тощо, та використовувати отримані знання на власному досвіді.

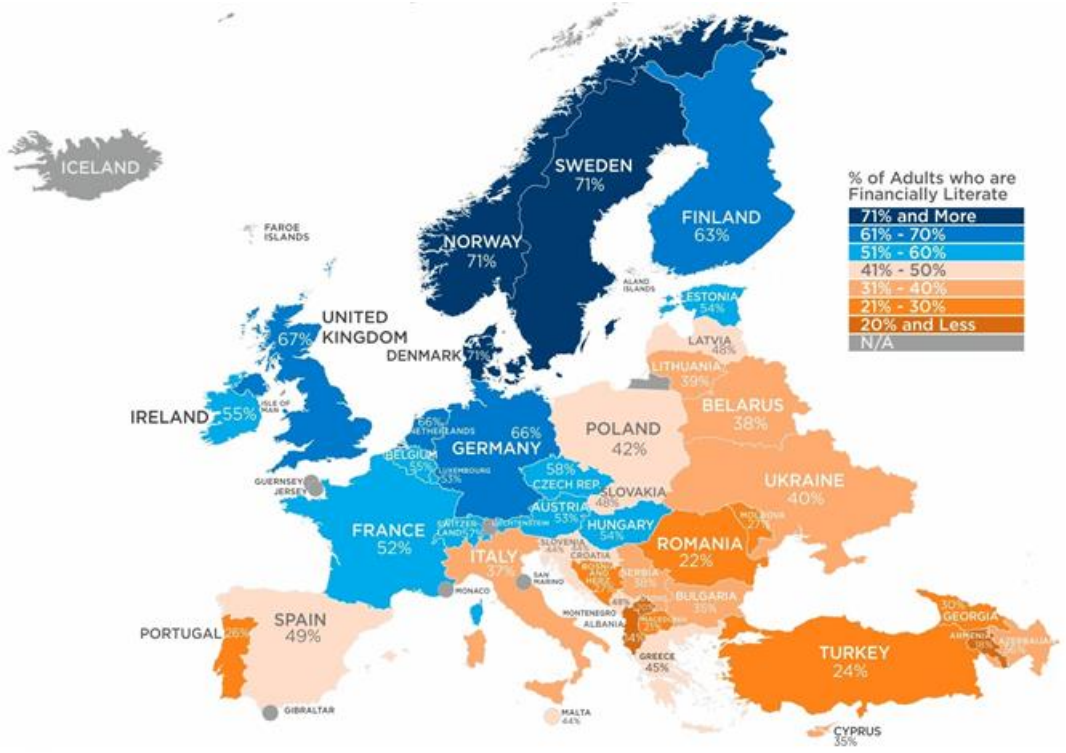


Рис. 1.2 Рівень фінансової грамотності населення країн Європи

Для відображення предметної області була розроблена діаграма предметної галузі. Вона відображає взаємозв’язки між об’єктами в певній предметній області. Ця діаграма допомагає візуалізувати складові частини предметної області.

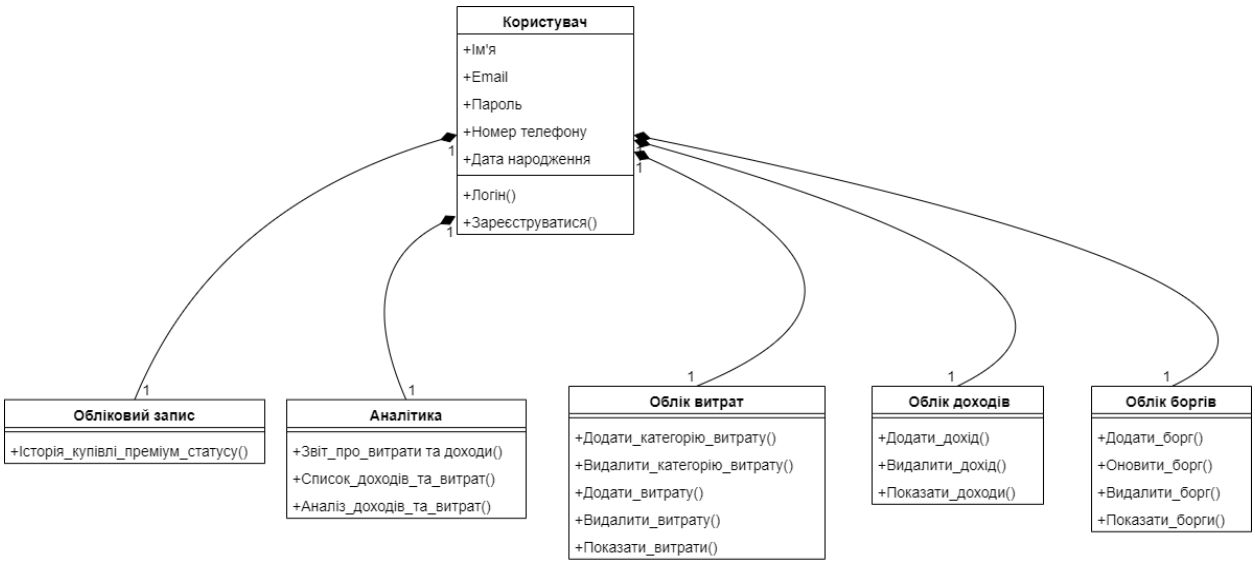


Рис. 1.3 Діаграма предметної галузі

1.2 Огляд аналогів у сфері фінансового менеджменту

1.2.1 Огляд Finmap

Веб-застосунок, який дозволяє легко вести облік грошей своєї компанії. Це український стартап, який залучив мільйони доларів. У Finmap доступні інтеграції з багатьма банками, платіжними інструментами, цифровими гаманцями тощо.

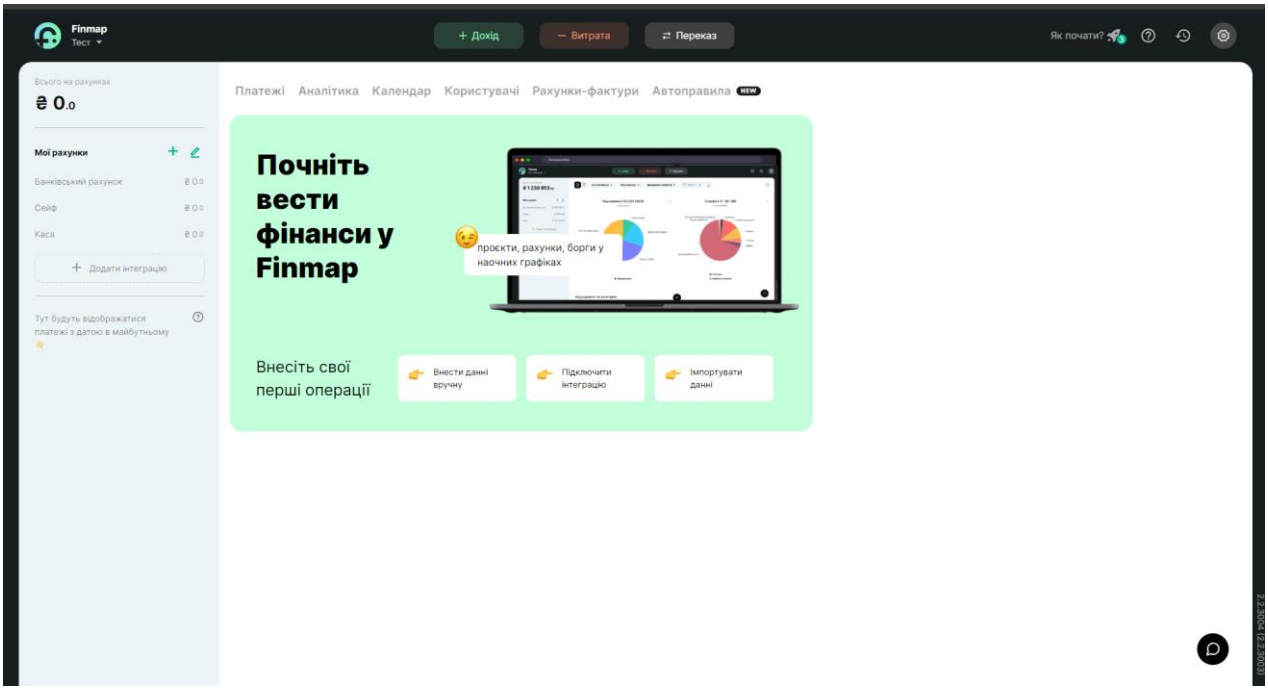


Рис. 1.4 Головний екран

Сервіс має безліч інтеграцій з різними банками, що доволі зручно та ефективно, також можна ввести всі дані в ручну, якщо інтеграція відсутня, що дозволяє користувачам зберігати контроль над своїми фінансами та додає гнучкості у використанні застосунку.

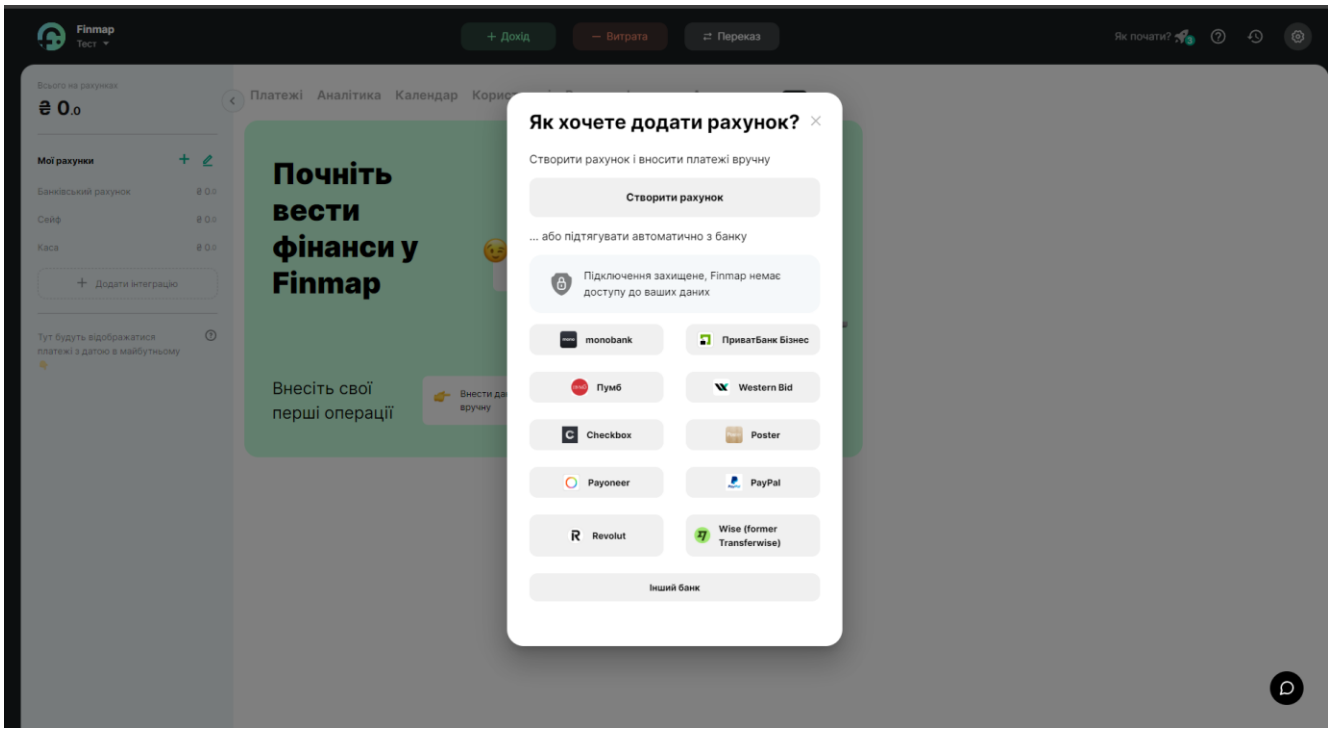


Рис. 1.5 Список всіх доступних інтеграцій з банками

Також сервіс має багато категорій витрат та доходів, які можна обрати по ситуації. Наприклад, витрата на заробітну плату співробітникам, або отримання кредиту, тощо.

З плюсів є те, що користувач сам може додавати категорії, також є можливість обрати опцію «Зробити повторювальним» для того, щоб система автоматично оброблювала витрату чи дохід згідно за налаштуваннями (рис. 1.8). Наприклад, сплата по кредиту, оренду обладнання, тощо.

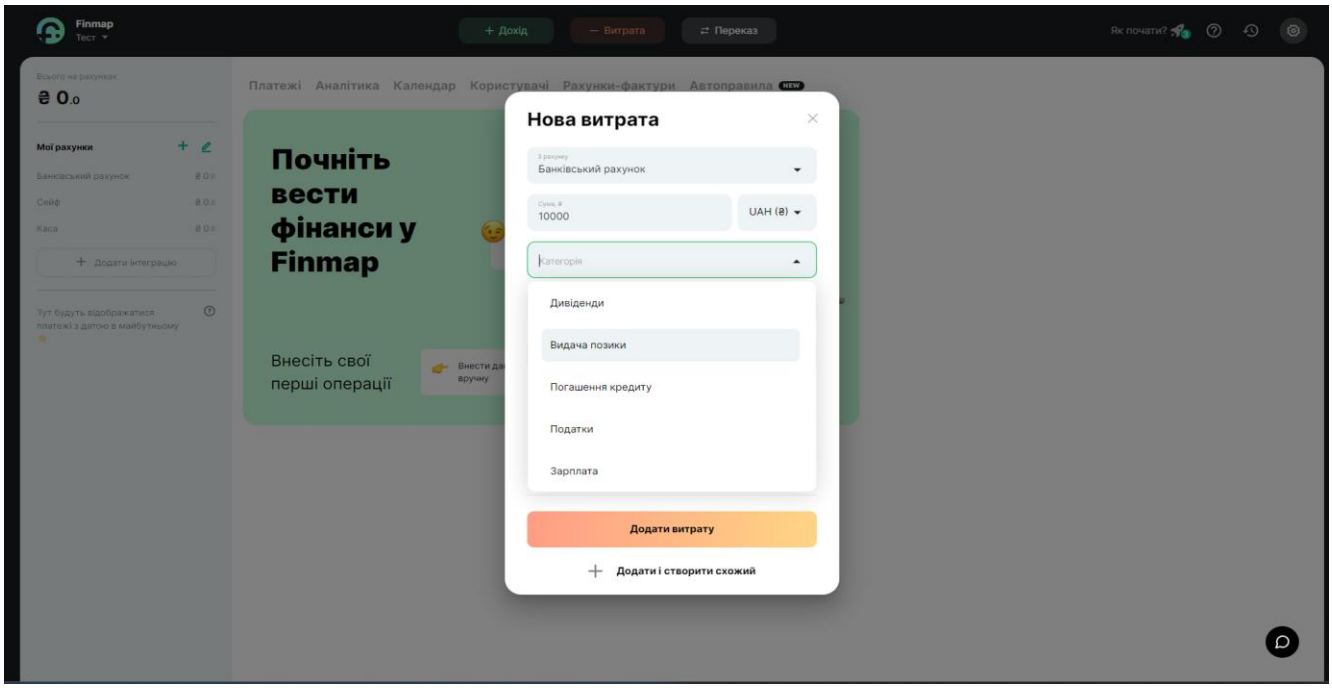


Рис. 1.6 Категорії витрат

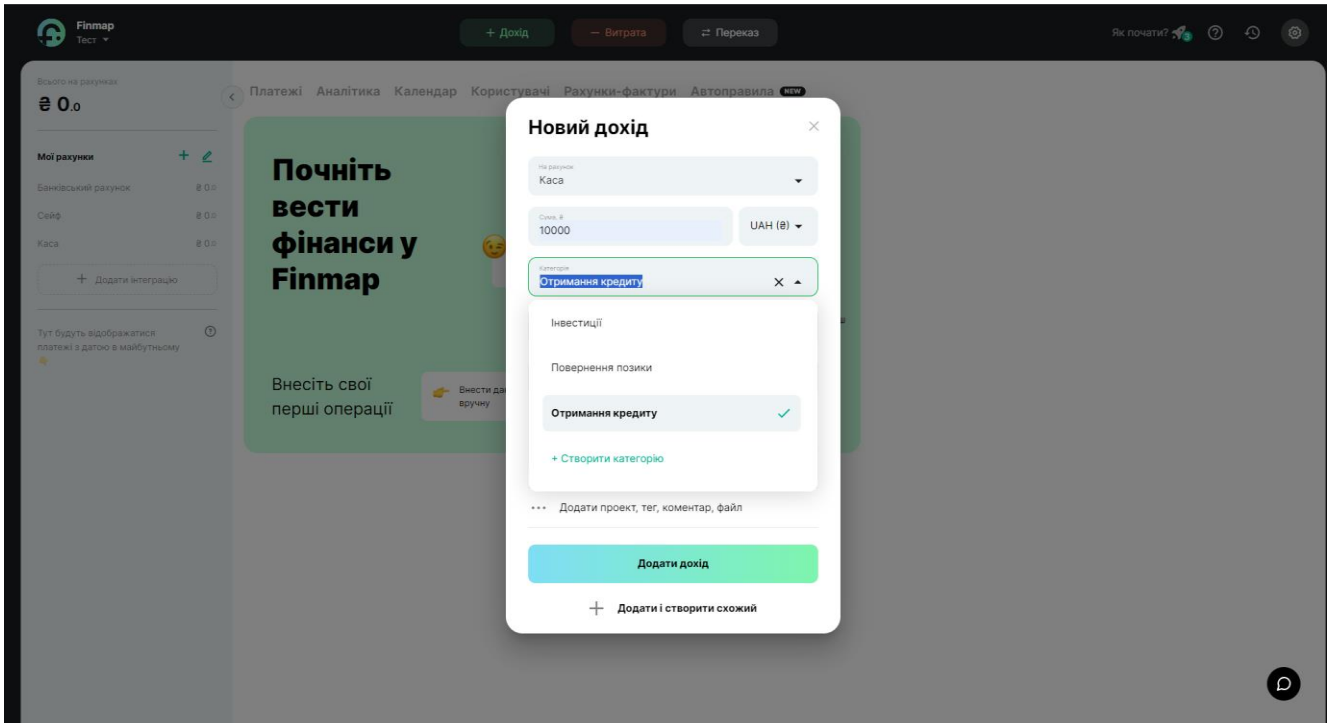


Рис. 1.7 Категорії доходів

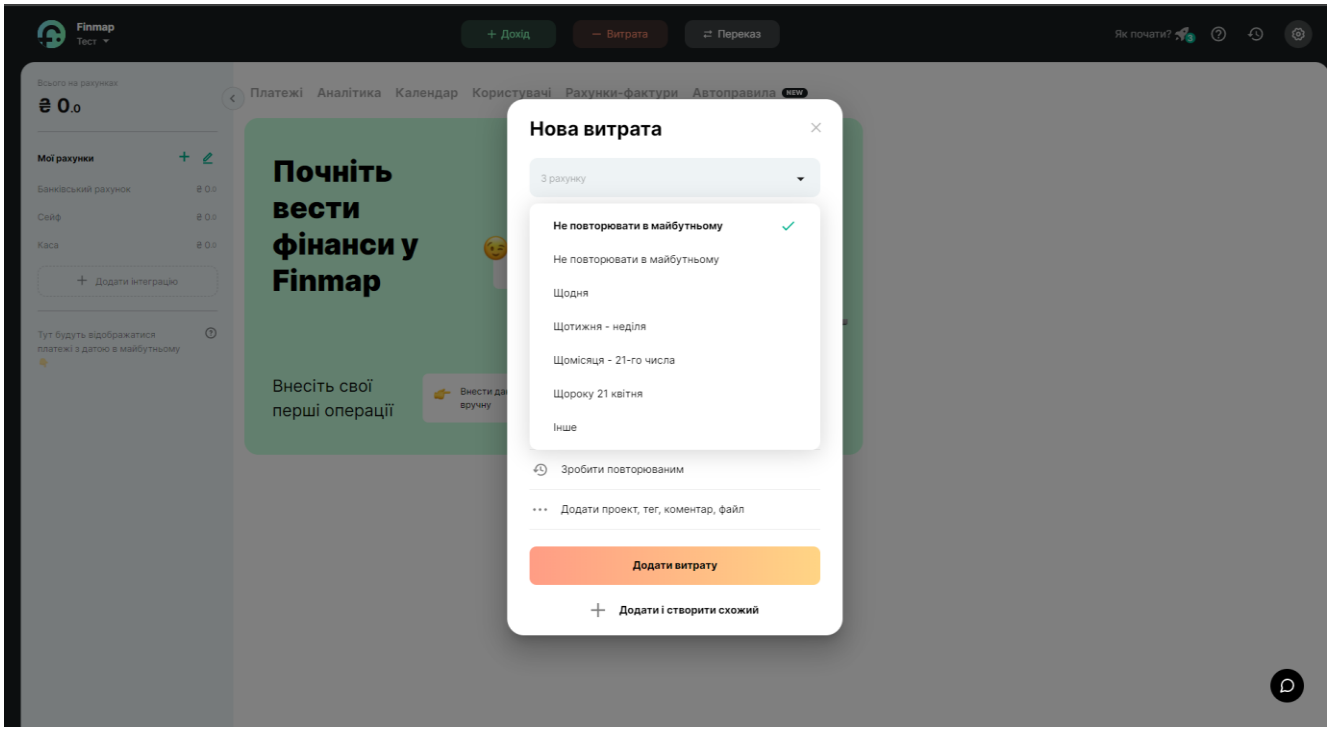


Рис. 1.8 Налаштування повторюваності

Застосунок відображає красиві, детальні графіки та статистику руху грошових коштів підприємства та має доволі функціональну аналітику. Це дозволяє відслідковувати та швидко реагувати на події, що забезпечить фінансову стабільність установи.

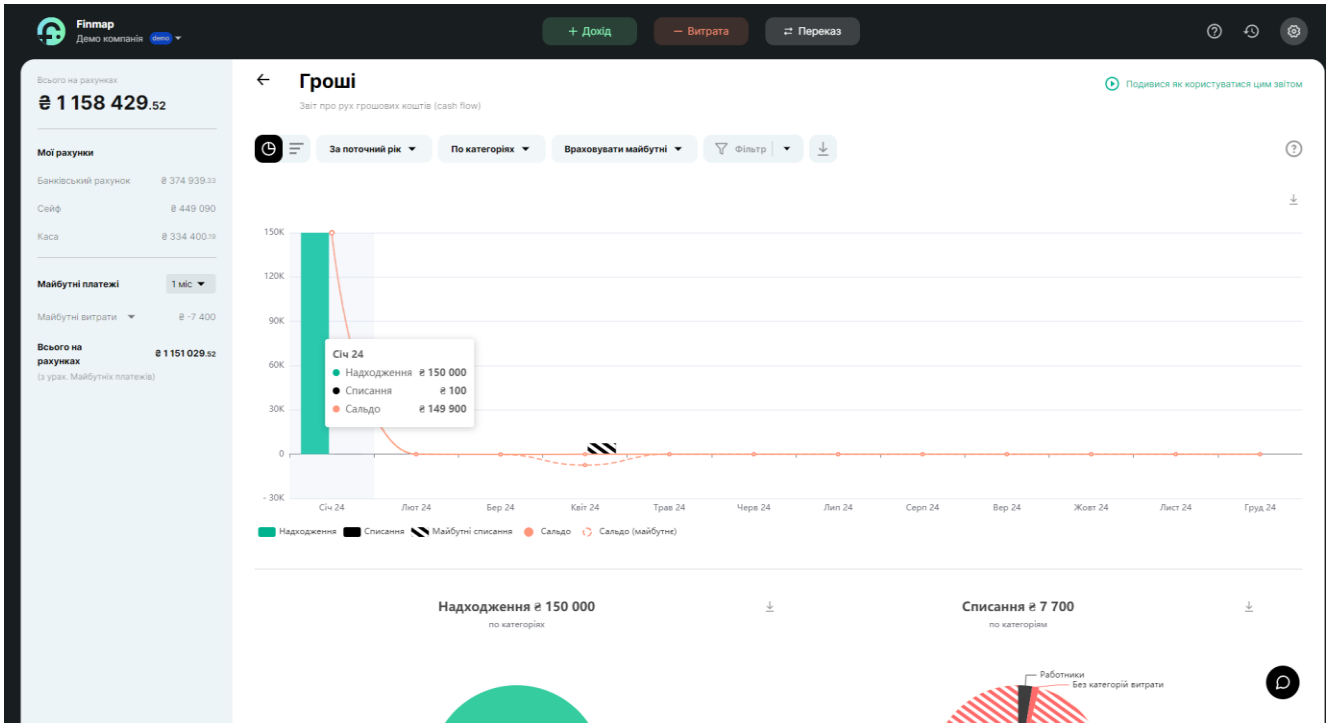


Рис. 1.9 Статистика та графіки руху фінансів

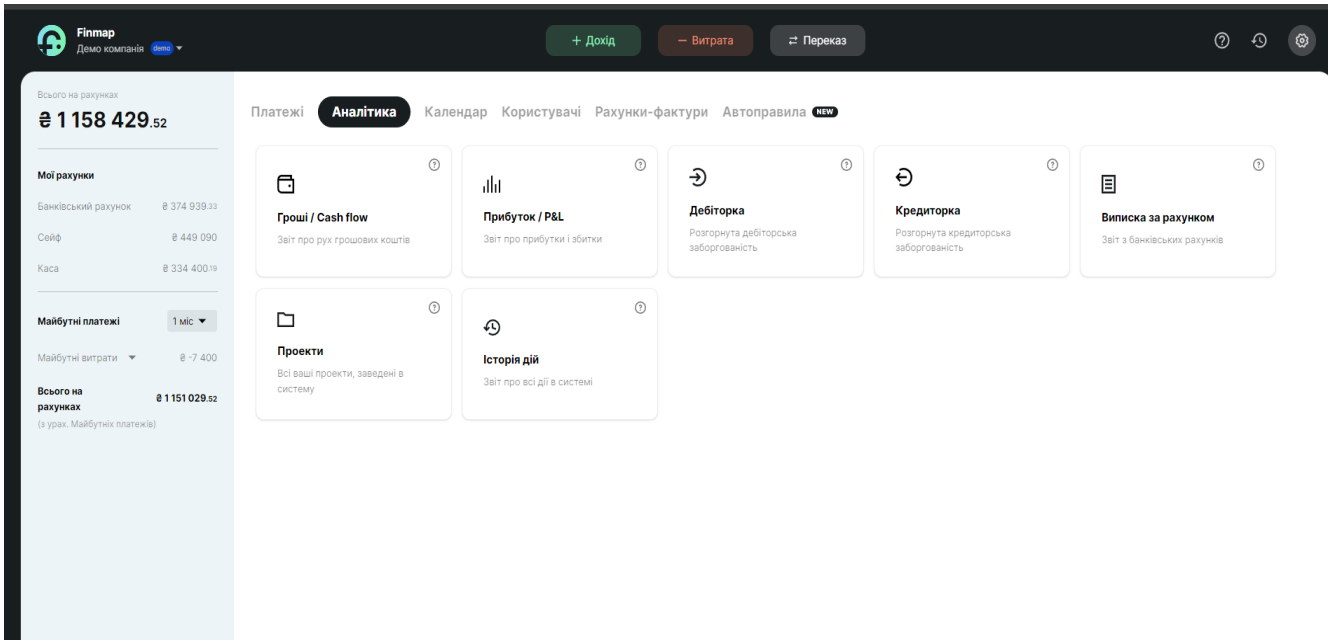


Рис. 1.10 Категорії аналітики

1.2.2 Огляд Spendee

Це сервіс для управління особистими фінансами, аналізу витрат та доходів, допомогає аналізувати фінансові звички. Він сумісний з багатьма платформами, це дає змогу синхронізувати банківські рахунки. Spendee розроблено компанією SPENDEE, яка розташована в Празі, Чехія.

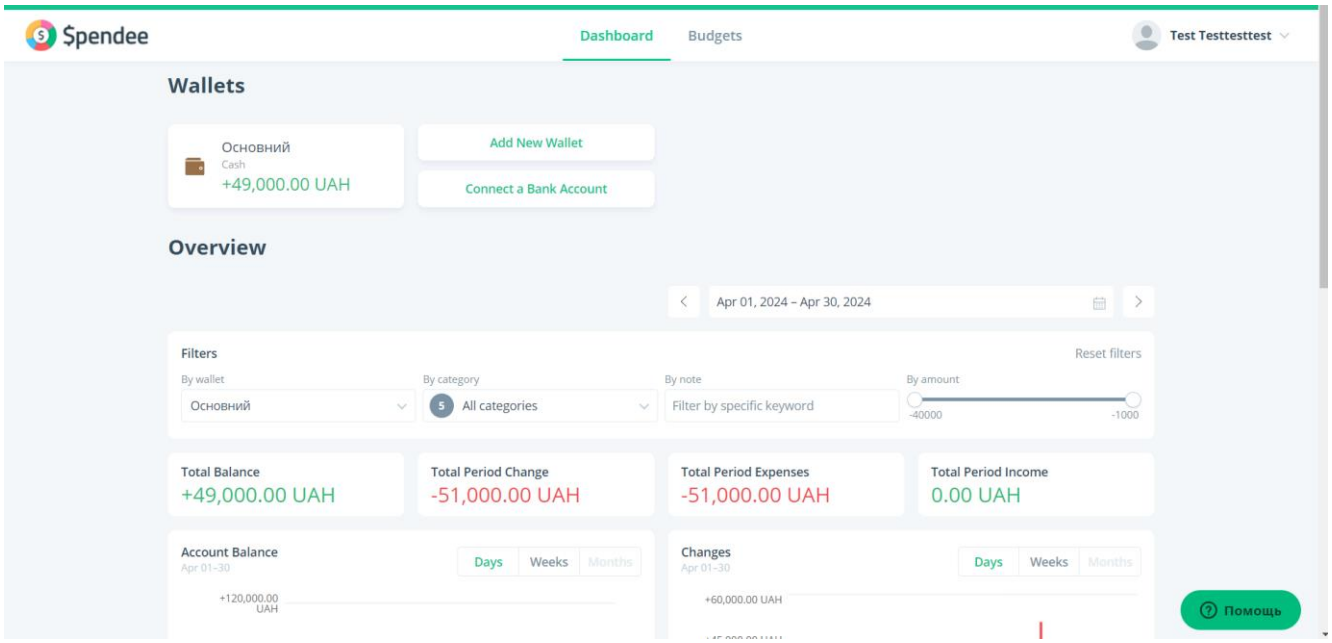


Рис. 1.11 Головна сторінка

Особливістю цього сервісу є те, що тут є поняття гаманця та бюджету.

Для ведення обліку одначасно двох валют, потрібно створити гаманці для кожної з них (рис. 1.12). Таким чином це додає зручності у використанні сервісу, адже дані не будуть перемішуватись, це дає змогу уникати помилок в плануванні та баченні фінансового становища в цілому.

Бюджети допомогають розподілити основну суму на необхідні витрати, кожного дня, місяця, тощо. Система за допомогою аналізу витрат та доходів дає рекомендації по оптимізації ваших звичок та витрат, а також на основі бюджету прораховує кількість грошей, які ви можете витратити, щоб не зайти за його рамки (рис. 1.13).

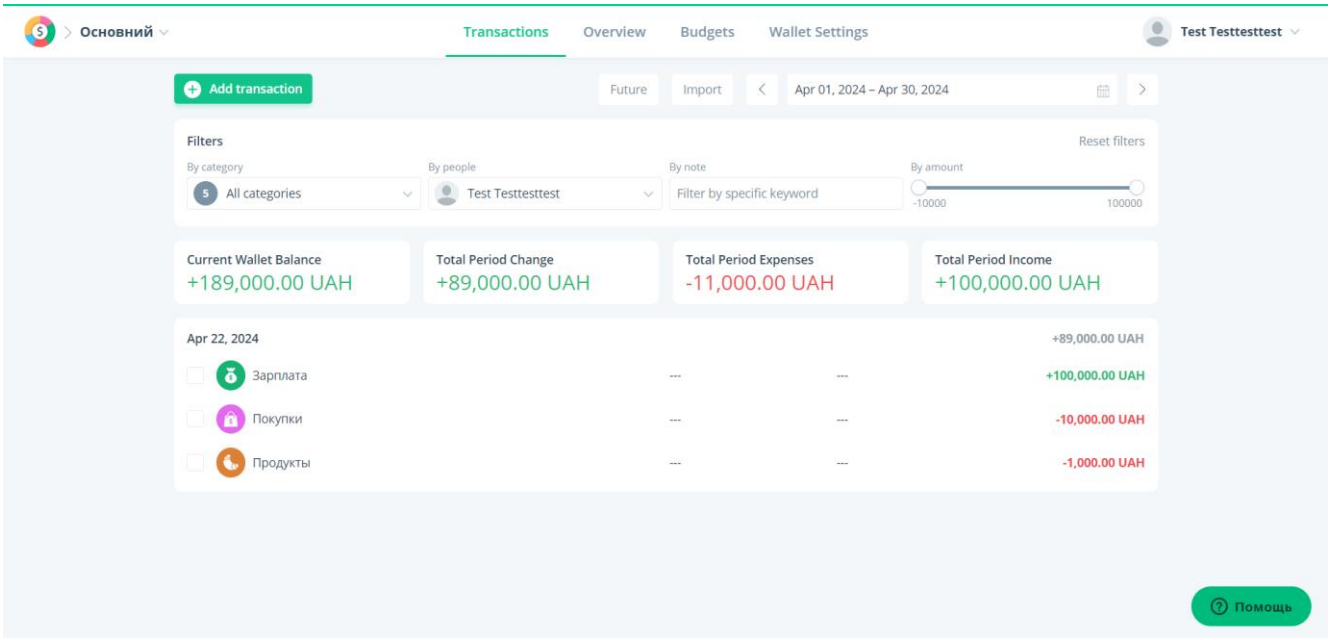


Рис. 1.12 Сторінка гаманця

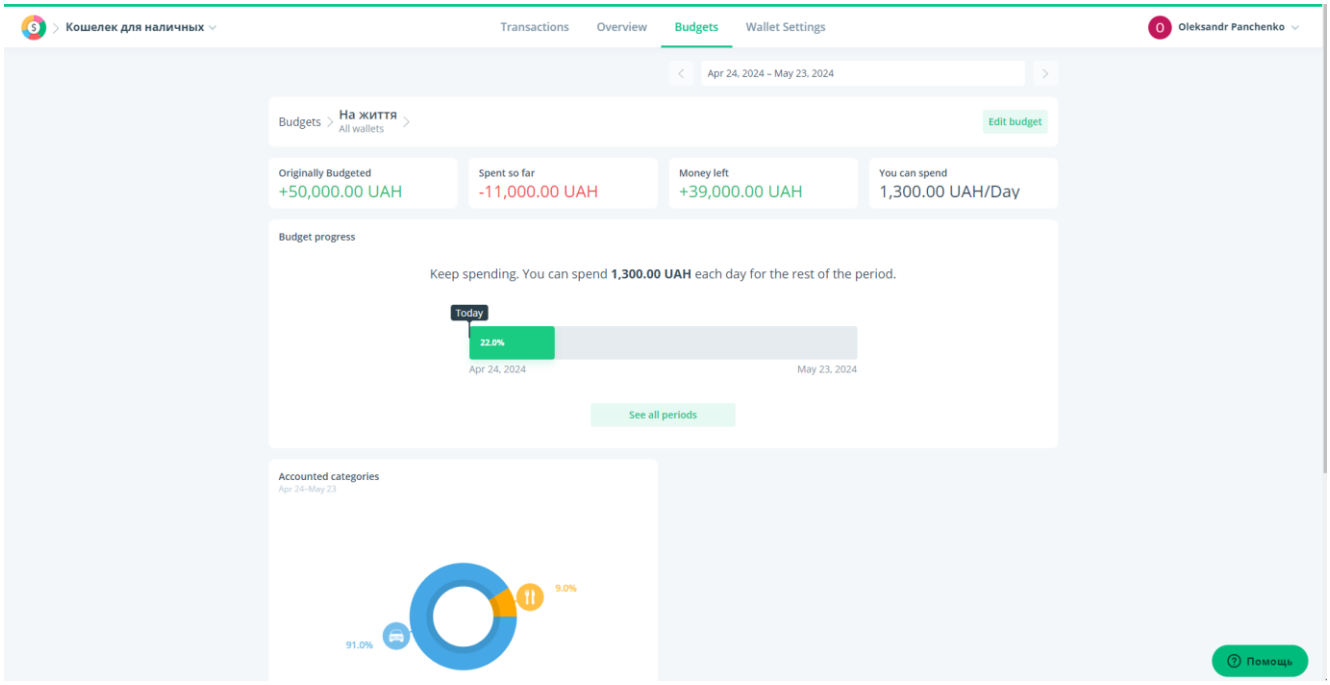


Рис. 1.13 Сторінка бюджету

Система має безліч категорій для витрат та надходжень, також плюсом є те, що можна створювати особисті категорії.

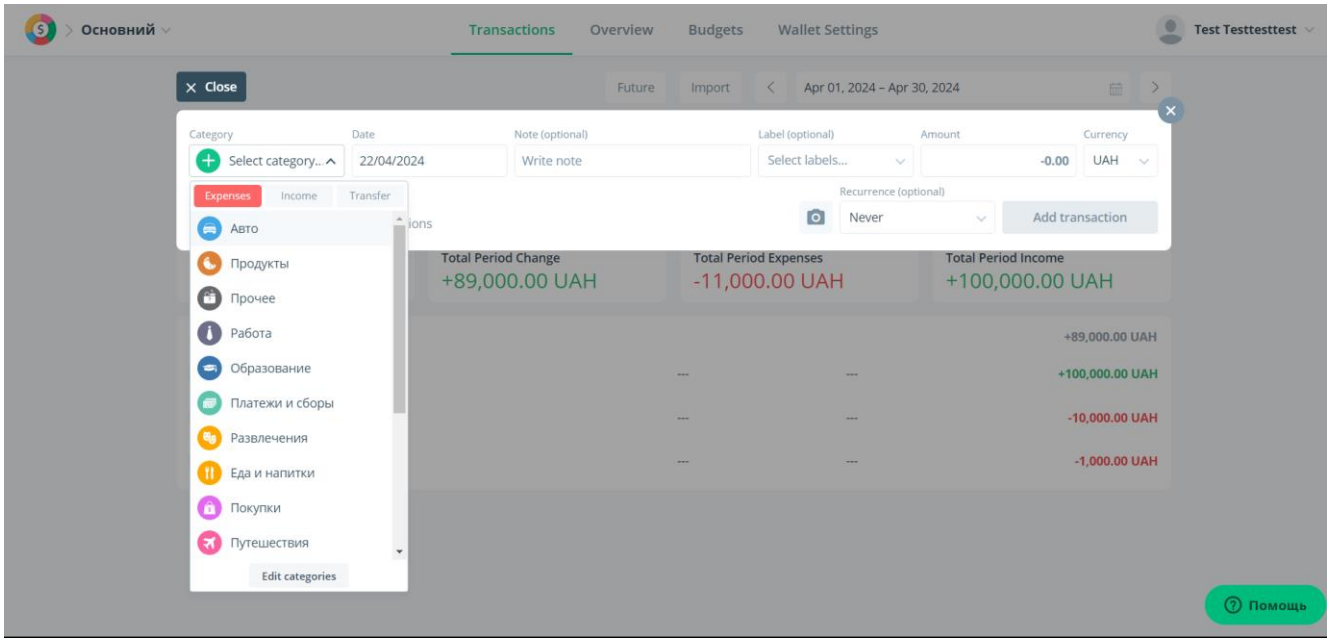


Рис. 1.14 Категорії витрат та надходжень

Цей інструмент має гарну та інформативну статистику, яку можна отримати як за окремий гаманець так і за обраним бюджетом, та на її основі створювати плани на наступний період.

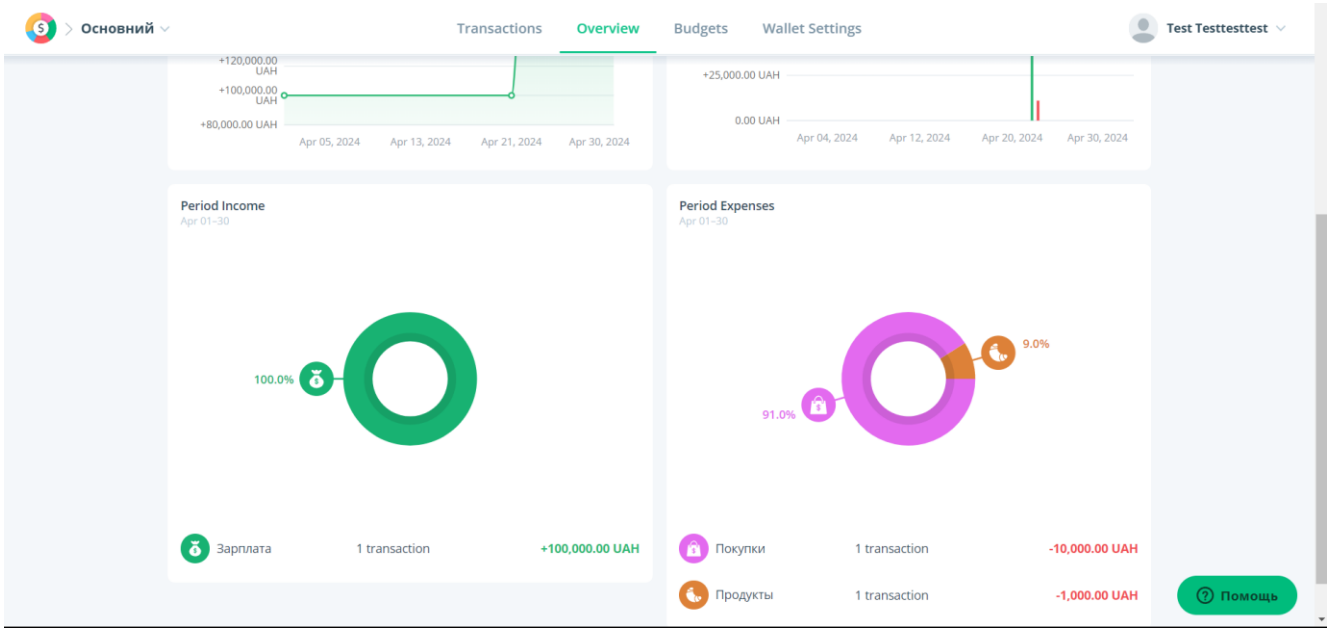


Рис. 1.15 Сторінка статистики за обраним гаманцем

1.2.3 Огляд Goodbudget

Це сервіс для роботи з фінансами та обліку витрат, за його допомогою можна чудово планувати домашній бюджет. Застосунок має змогу заощаджувати гроші завдяки методу складання бюджету в конверти. Також у застосунку можна створити групи з сім'єю або друзями. У безкоштовній версії кількість конвертів та облікових записів буде обмеженою. Розробником є американська компанія Dayspring Partners.

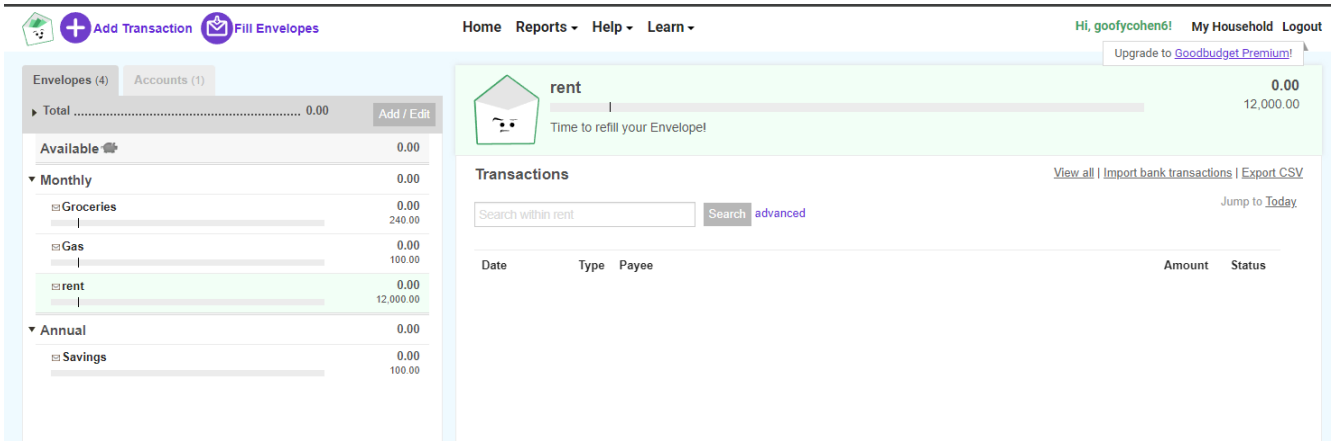


Рис.1.16 Головна сторінка

В застосунку є можливість переглянути транзакції на облікових записах.

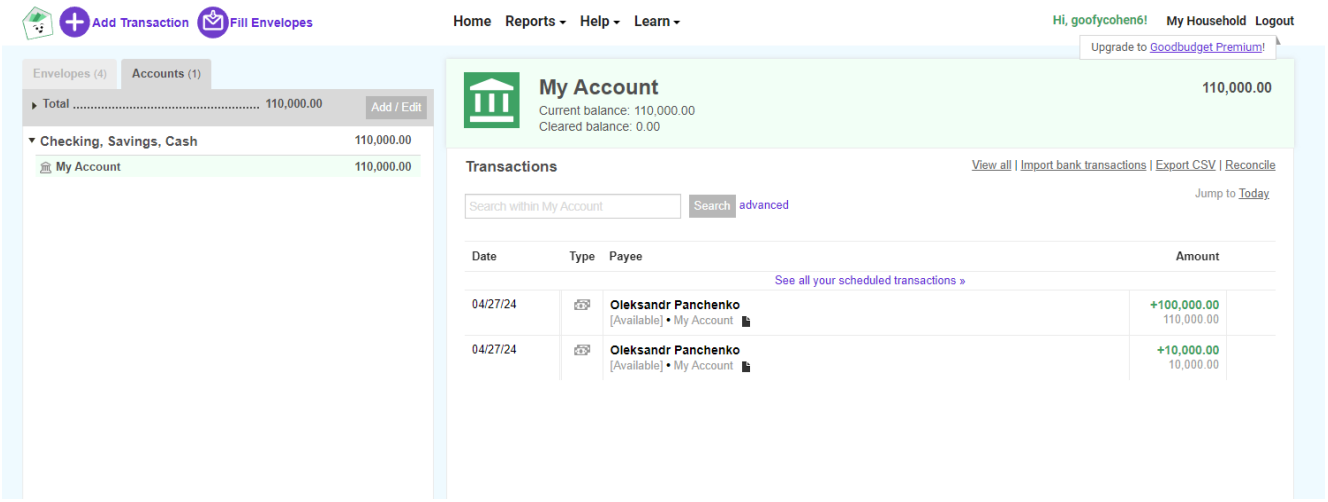


Рис. 1.17 Сторінка перегляду транзакцій

Сервіс має витрати, виплати по кредитам, надходження, трансфер між гаманцями, та борги. Він відрязняється від інших сервісів керування фінансами тим, що тут немає категорій як таких. Витрата грошей відбувається з конвертів користувача.

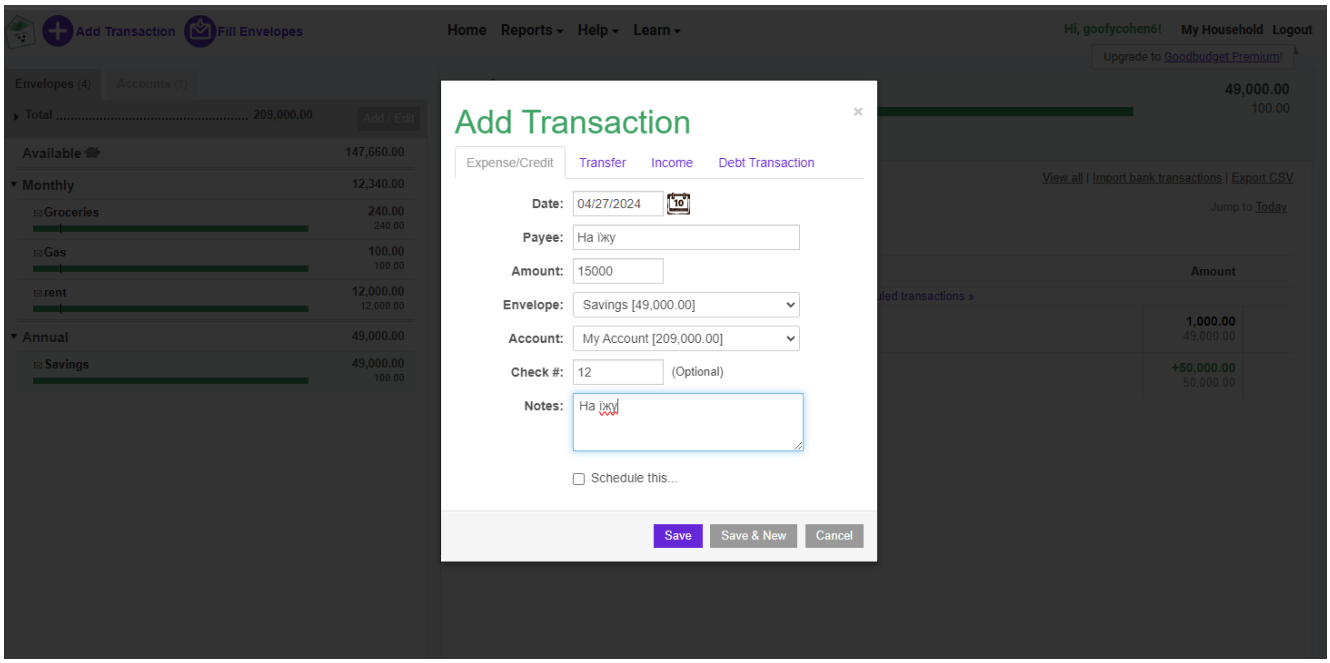


Рис. 1.18 Сторінка додовання витрат

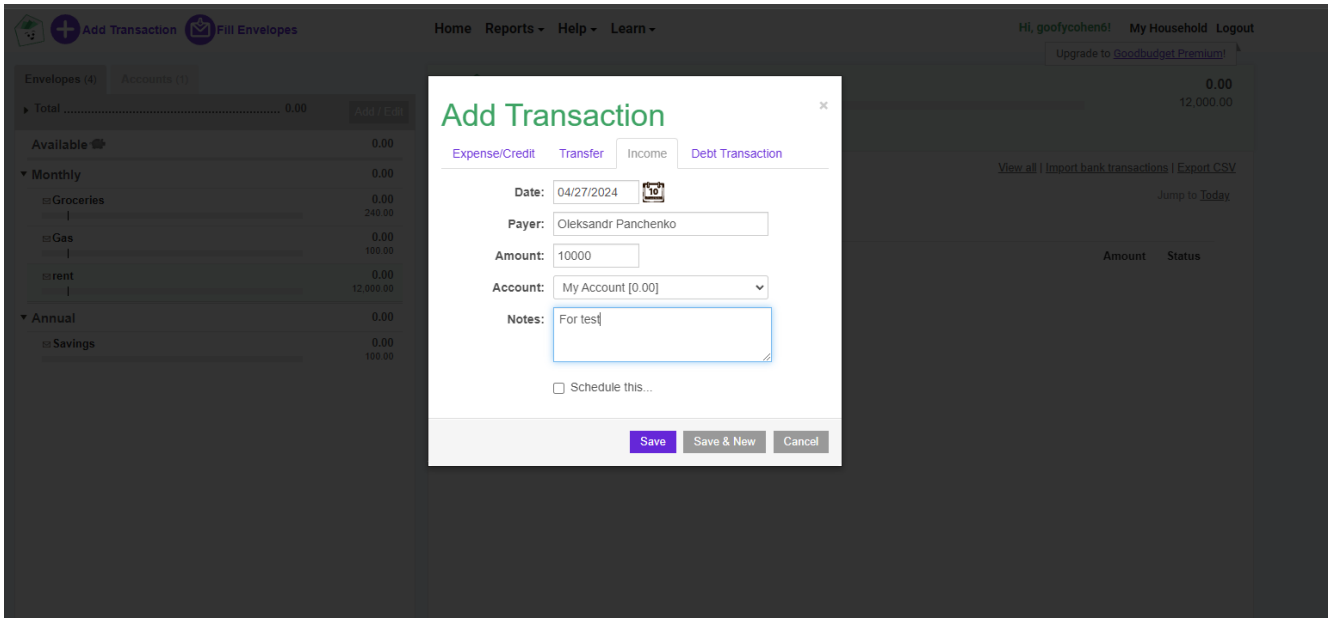


Рис. 1.19 Сторінка додавання надходження

Щоб правильно розподілити вільні гроші по конвертах, застосунок має зручний механізм «Fill Envelopes» (рис. 1.20).

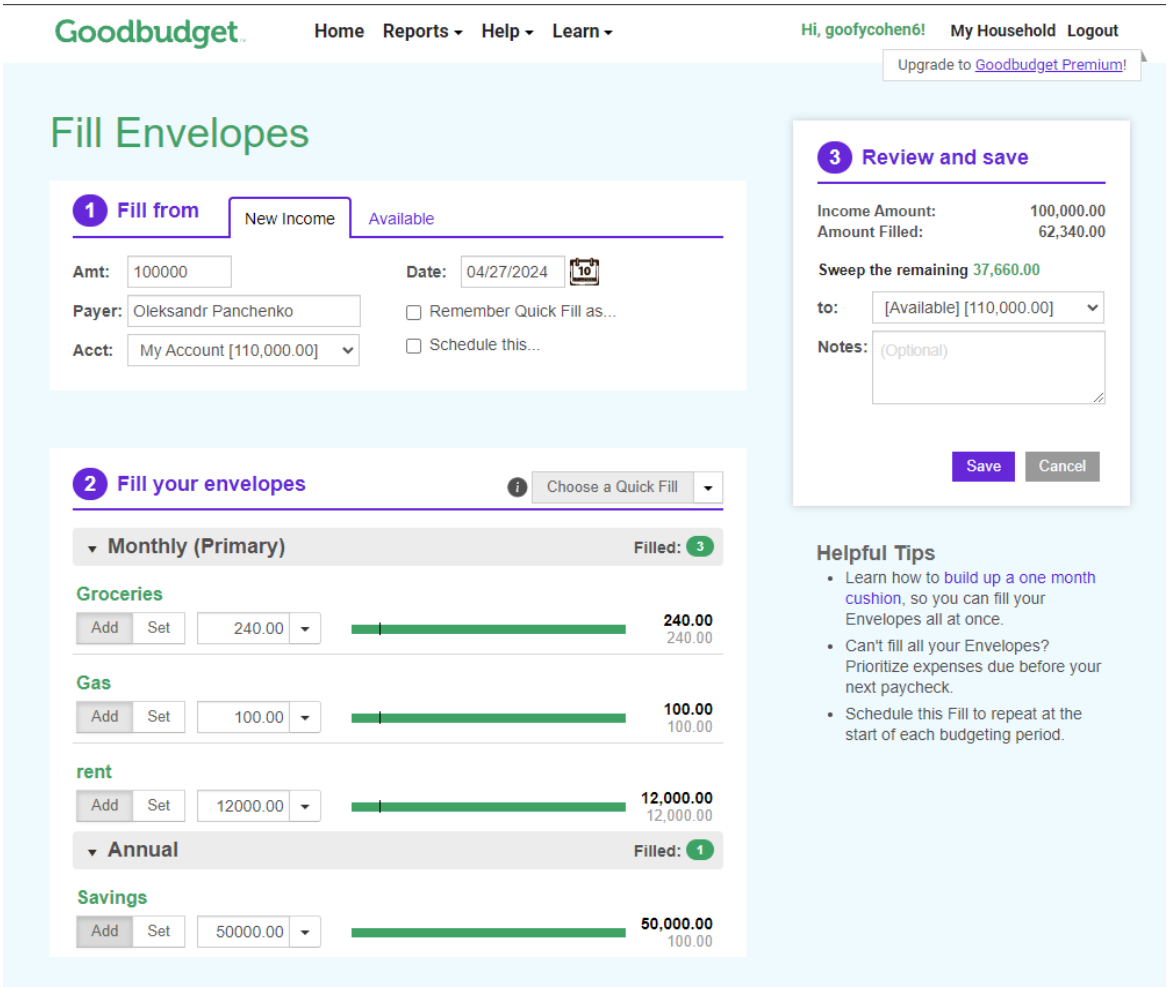


Рис. 1.20 Сторінка для розподілення грошей по конвертам

GoodBudget має безліч категорій по статистиці (рис. 1.21) та відображає красиві графіки та докладну інформацію (рис. 1.22).

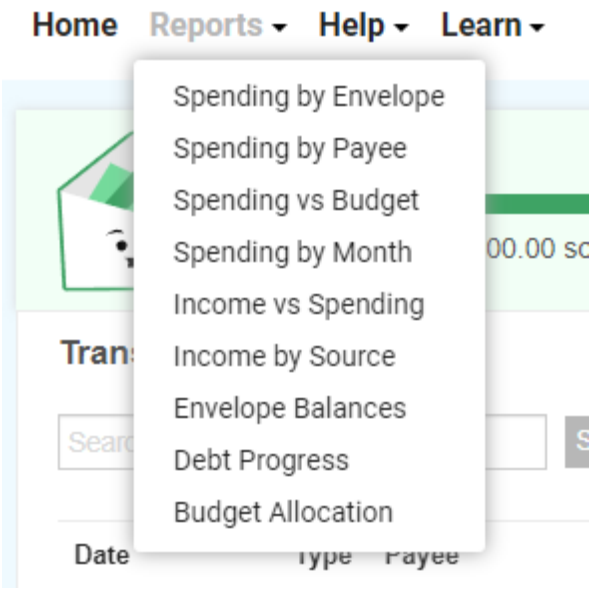


Рис. 1.21 Список категорій для перегляду статистики

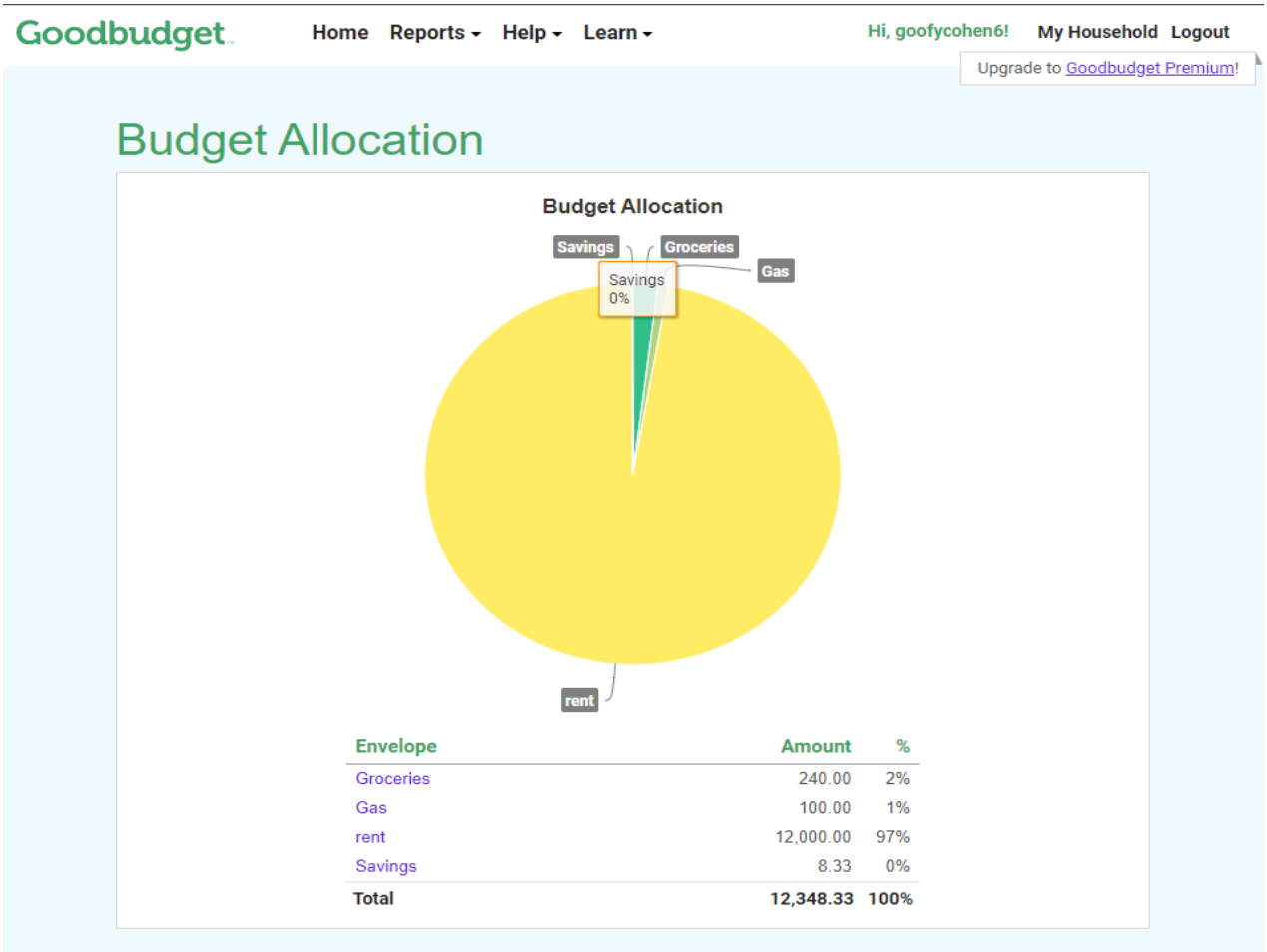


Рис. 1.22 Статистика по бюджету

1.2.4 Переваги та недоліки розглянутих застосунків

Таблиця 1.1

Результати аналізу розглянутих веб-застосунків керування фінансами

Функції	Finmap	Spendee	Goodbudget	FinanceTracking
Наявність статистики	+	+	+	+
Цільова аудиторія	Тільки для бізнесу	Для особистого користування	Для особистого користування	Для бізнесу та особистого користування
Можливість купівлі преміум підписки	+	+	+	+
Отримання звітів	+	+	+	+
Відсутня реклама	-	-	-	+
Синхронізація між пристроями	+	+	+	+
Розділення витрат по категоріям	+	+	-	+
Інтеграції з банками	+	+	+	+
Малі обмеження в базовій підписці	-	-	-	+

Аналіз конкурентів в категорії фінансового менеджменту допомагає проаналізувати та створити вимоги та функціонал до продукту для того, щоб бути конкурентоспроможнім на момент виходу застосунку у зовнішній світ.

Сервіс FinanceTracking має бути розроблений на браузерній платформі. Він повинен мати таку архітектуру програмного забезпечення, яка зможе підтримувати масштабованість, тобто додавання нових функцій без переробки минулого функціоналу та мати наступний набір функціональних та нефункціональних вимог:

Визначено основні функціональні вимоги до застосунку:

Для користувача:

- внесення в систему даних про витрати, надходження та борги;
- представлення статистики по категоріям та їх витратам;
- засоби додавання та видалення категорій витрат;
- представлення даних у вигляді графіків та діаграм;
- отримання фінансового звіту за певний період;

Для адміністратора:

- перегляду списку користувачів;
- блокування та розблокування користувачів;
- скасовування та надавання преміум підписки;
- перегляд історії преміум підписок користувача;
- створювання нового адміністратора;
- пошук користувачів за фільтром або по унікальним ідентифікатором (електронної пошти, унікальний номер користувача, за номером телефону).

Визначено основні нефункціональні вимоги до застосунку:

- система повинна підтримувати масштабованість, для швидкого додавання нових функцій;
- система повинна бути захищена від несанкціонованого доступу;
- Система повинна набрати показник не менше 60 відсотків продуктивності в Lighthouse;

2 АНАЛІЗ ІНСТРУМЕНТАЛЬНИХ ЗАСОБІВ РЕАЛІЗАЦІЇ

2.1 Вибір технологій та засобів розробки програмного забезпечення

Веб-сервіси управління фінансами, потребує високої надійності, швидкодії та масштабованості, саме тому слід ретельно підбирати кожну технологію, порівнювати аналоги і обирати на основі їх плюсів, мінусів та майбутніх планів.

До засобів розробки належить: інтегроване середовище розробки, мова програмування, система керування базами даних, клієнт для роботи з базою даних, розподілене сховище подій, мережевий шлюз, система контролю версій.

В проекті необхідно вирішити наступний набір завдань:

- спланувати архітектуру проекту з урахуванням можливості подальшого розширення функціоналу та інтеграції нових технологій.
- розробити комунікацію між мікросервісами за допомогою HTTP та Kafka;
- застосувати підхід до моделювання програмного забезпечення DDD (Domain-Driven Design);
- визначити зони відповідальності кожного мікросервісу, спроектувати до них бази даних;
- впровадити систему контролю версій для управління змінами в коді та спрощення співпраці між розробниками;
- впровадити політики безпеки та управління доступом, щоб обмежити доступ до чутливих даних та функціоналу тільки уповноваженим користувачам;
- створити інтеграцію з платіжною системою «PayPal»;

2.1.1 Розподілене сховище подій Apache Kafka

Щоб забезпечити швидку реакцію на події та масштабованість у проекті використано Apache Kafka. Це розподілене сховище подій і платформа для їх багатопотокового оброблення. Мета цього програмного забезпечення є створення уніфікованої високопродуктивної платформи з низькою затримкою для обробки потоків даних у реальному часі. Вона забезпечує надійний, масштабований потік між системами.

Kafka має таку архітектуру (рис. 2.9):

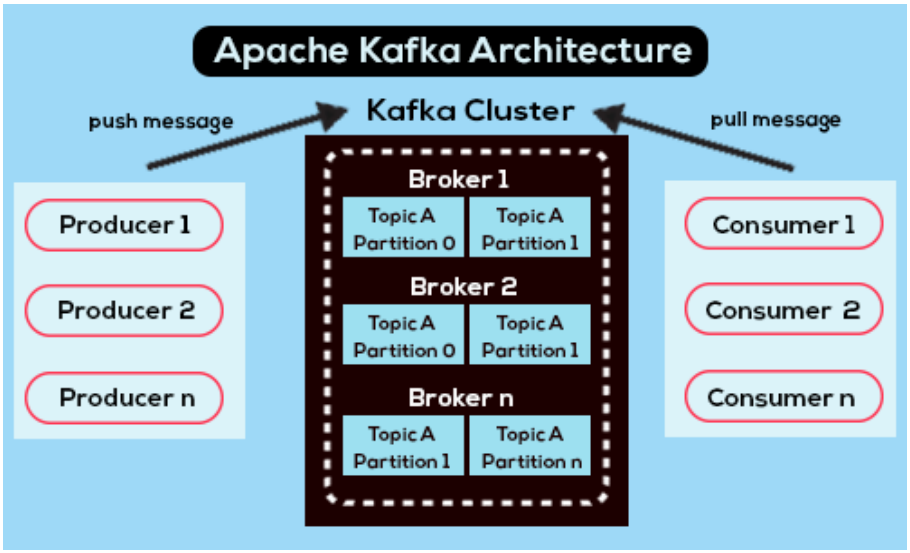


Рис. 2.9 Архітектура Apache Kafka

2.1.2 Клієнт Offset Explorer

Offset Explorer дозволяє швидко переглядати об’єкти та повідомлення, які містяться в кластері, також містить функціонал, як для розробників, так і для адміністраторів. Offset Explorer має такий вигляд (рис. 2.10)

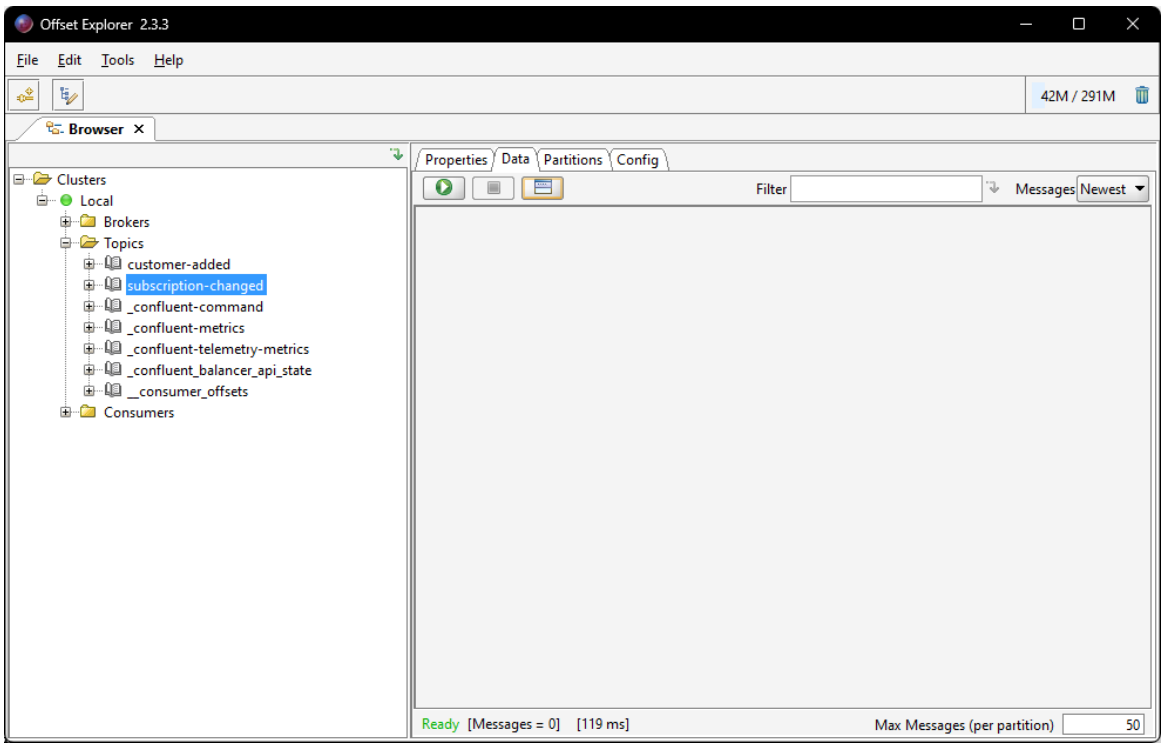


Рис. 2.10 Інтерфейс Offset Explorer

2.1.3 Ocelot Gateway

В проекті, який побудований на мікросервісній архітектурі, для розподілу навантаження, захисту та майбутніх інтеграцій з іншими системами було використано Ocelot. Це відкритий API шлюз та зворотний проксі-сервер. Він призначений для безперебійної роботи з застосунками. Таким чином він дозволяє користувацькому інтерфейсу відправляти запити тільки на його адресу, а сам він перенаправляє запити на відповідні мікросервіси, таким чином він створює єдину точку входу, що збільшує захист від втрати конфіденційної інформації та можливість інтегрування із зовнішніми сервісами.

Ocelot забезпечує простий у використанні спосіб маршрутизації та управління HTTP-запитами, обробляти навантаження, автентифікацію, авторизацію, кешування та інші функції, як правило, необхідні в мікросервісах і розподілених архітектурах.

Приклад використання на рисунку 2.11.

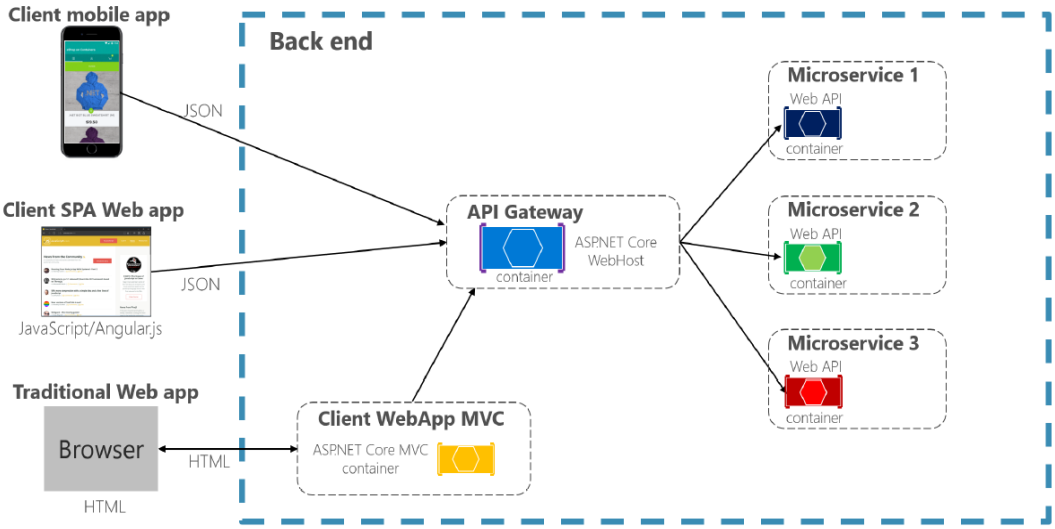


Рис. 2.11 Приклад використання API шлюзу.

2.1.4 База даних PostgreSQL

Обираючи базу даних для проекту, враховано безліч аспектів, такі як: надійність, продуктивність масштабованість, тощо. Тому було обрано PostgreSQL.

PostgreSQL об’єктно-реляційна система керування базами даних. Вона має підтримку стандартів SQL, підтримує багато різних типів даних, та дозволяє гнучко проектувати базу, виконувати складні операції та запити, має активну спільноту. PostgreSQL успішно застосовується у різних галузях. У веб-розробці вона слугує основою для сайтів і додатків, широко використовується в аналізі даних.

Рис. 2.12 Логотип PostgreSQL

2.1.5 Клієнт для роботи з базою даних DBeaver

Для адміністрування баз даних в роботі було використано DBeaver. Він має багато корисних функцій, які забезпечують краще сприйняття та зручність використання, такі як: функції завершення коду, підсвічування синтаксису, забезпечує архітектуру плагінів, яка дозволяє змінювати більшу частину роботи програми, це дозволяє підтримувати специфічні для бази даних функції. DBeaver має такий вигляд (рис. 2.13)

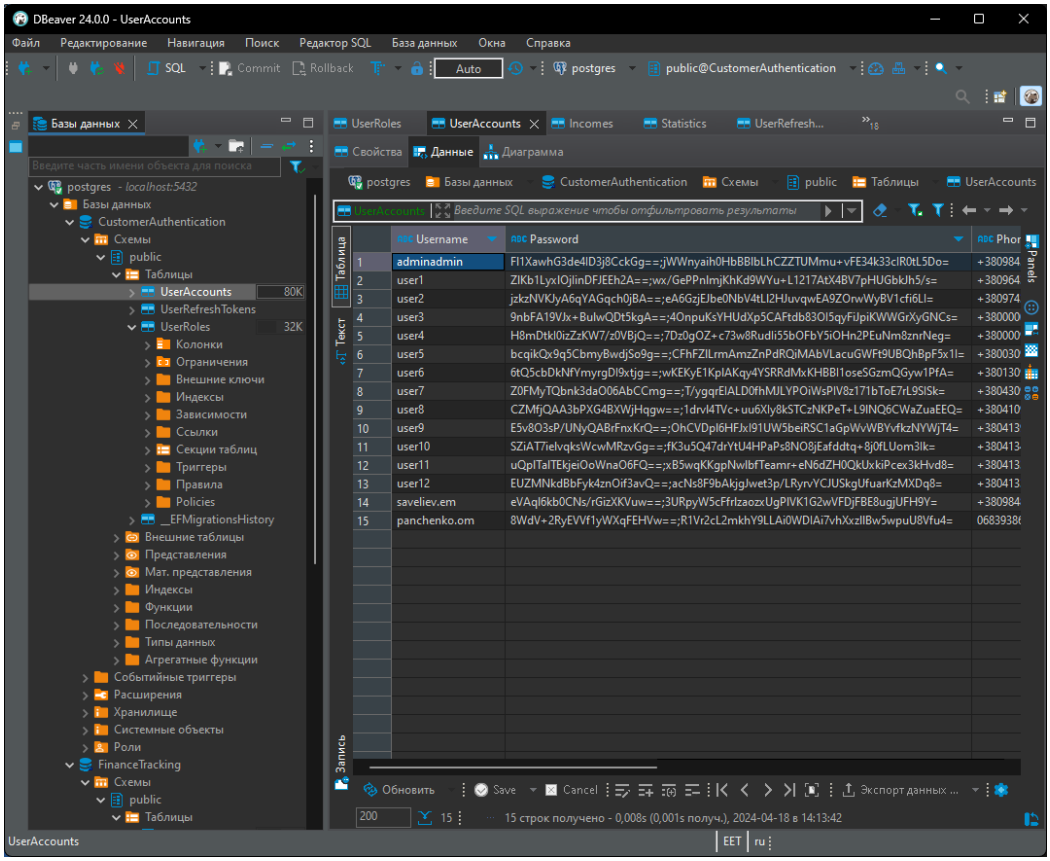


Рис. 2.13 Інтерфейс DBeaver

2.1.6 Мова програмування C#

Для реалізації серверної частини, обрано мову програмування C#, тому що, це універсальна, об'єктно-орієнтована мова програмування. Вона є основною мовою програмування для платформи .Net та широко використовується для розробки різноманітних програмних засобів. Має безліч переваг, наприклад, строгу типізацію, автоматичне приведення типів, сміттєзбірник, який допомагає уникнути помилок в пам’яті, асинхронне програмування тощо. Вона постійно

розвивається, додаються нові функції та покращення, має активну спільноту та підтримку.

Екосистема .Net надає великий обсяг бібліотек, за допомогою яких можна реалізовувати складні задачі швидше та ефективніше. Будь-яке програмне забезпечення написане на C# можна запустити та розгорнути на різних операційних системах або хмарній платформі.

Рис. 2.14 Логотип мови програмування C#

2.1.7 Платформа ASP .NET

Для реалізації серверної частини в роботі, обрано ASP .Net Framework, тому що, вона має безліч переваг, які роблять її привабливим вибором для розробки веб-сервісів. Цей фреймворк дуже швидкий та продуктивний, що робить його незмінним інструментом, для розробки високонавантажених мікросервісів. В її основі знаходиться Common Language Runtime (CLR) – це вбудована віртуальна машина, яка виконує програми, відповідає за керування пам’яттю, обробку виключень, тощо.

Платформа складається з безліч інструментів, мов програмування та бібліотек для розробки будь-яких типів програм.

Однією з ідей цього фреймворку є сумісність служб, які написані різними мовами. Таким чином, розробники можуть вибрати різні мови для розробки необхідної програми. Популярними є Visual Basic і C#.

Рис. 2.15 Логотип ASP.Net Framework

2.1.8 Entity Framework Core

Для ефективної роботи з базою даних в роботі використано популярну та потужну ORM - Entity Framework Core (EF Core). Це фреймворк, що допомагає зіставляти таблиці у реляційній базі даних з об’єктами у мовах програмування. Він дозволяє уникнути використання SQL скриптів у переважній кількості випадків, тому що його API дозволяє швидко виконувати різноманітні операції з даними. Також він має асинхронність та надає потужний механізм міграцій, завдяки цьому спрощується розгортання та оновлення застосунку.

```
namespace CustomerAuthentication.PostgreSql.Dal.Migrations
{
    /// <inheritdoc />
    AquariuS
    public partial class Initial : Migration
    {
        /// <inheritdoc />
        AquariuS
        protected override void Up(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.CreateTable(
                name: "UserAccounts",
                columns: table:ColumnsBuilder => new
                {
                    PlayerId = table.Column<int>(type: "integer", nullable: false) // OperationBuilder<AddColumnOperation>
                        .Annotation(name:"Npgsql:ValueGenerationStrategy", NpgsqlValueGenerationStrategy.IdentityByDefaultColumn),
                    Username = table.Column<string>(type: "text", nullable: false),
                    Password = table.Column<string>(type: "text", nullable: false),
                    Role = table.Column<string>(type: "text", nullable: false),
                    Statuses = table.Column<string[]>(type: "text[]", nullable: false)
                },
                constraints: table =>
                {
                    table.PrimaryKey(name:"PK_UserAccounts", columns:x:{PlayerId,Username,...} => x.PlayerId);
                });

            migrationBuilder.CreateIndex(
                name: "IX_UserAccounts_Username_Password",
                table: "UserAccounts",
                columns: new[] { "Username", "Password" },
                unique: true);
        }

        /// <inheritdoc />
        AquariuS
        protected override void Down(MigrationBuilder migrationBuilder)
        {
            migrationBuilder.DropTable(
                name: "UserAccounts");
        }
    }
}
```

Рис. 2.16 Приклад міграції в мікросервісі

2.1.9 Мова програмування JavaScript

Для реалізації клієнтської частини обрано мову програмування JavaScript. Це високорівнева мова програмування, яка має динамічну типізацію та застосовується для написання скриптів. Вона відповідає за інтерактивність й взаємодію з користувачем. JavaScript використовує методологію об’єктно-орієнтованого підходу, що дає змогу представити програму у вигляді об’єктів. По суті вона є основою для розробки користувацького інтерфейсу, тому обрав її у свій проект.

Рис. 2.17 Логотип мови програмування JavaScript

2.1.10 Бібліотека React

Для швидкого та гнучкого створення інтерфейсу була обрана бібліотека React. Це бібліотека для створення інтерфейсів. Вона створена компанією Meta та швидко набула популярність.

Основою метою React є створення динамічних, інтерактивних інтерфейсів. Проект написаний на цій бібліотеці є легко розширюваним, тому що, вона має компонентну структуру. Ця бібліотека має активну спільноту розробників і, як на мене, вона досить проста у використанні, тому в роботі використано цю бібліотеку.

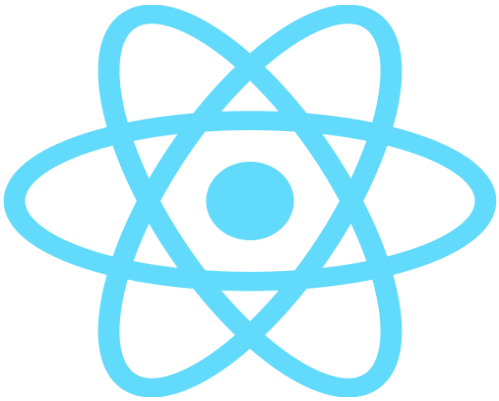


Рис. 2.18 Логотип бібліотеки React

2.1.11 Середовище розробки Visual Studio Code

Visual Studio Code – це простий та легкий редактор коду, один з найпопулярніших середовищ розробки, яку використовують здебільшого для розробки хмарних технологій та веб-застосунків. Має безліч розширень, які покращують роботу та комфорт.

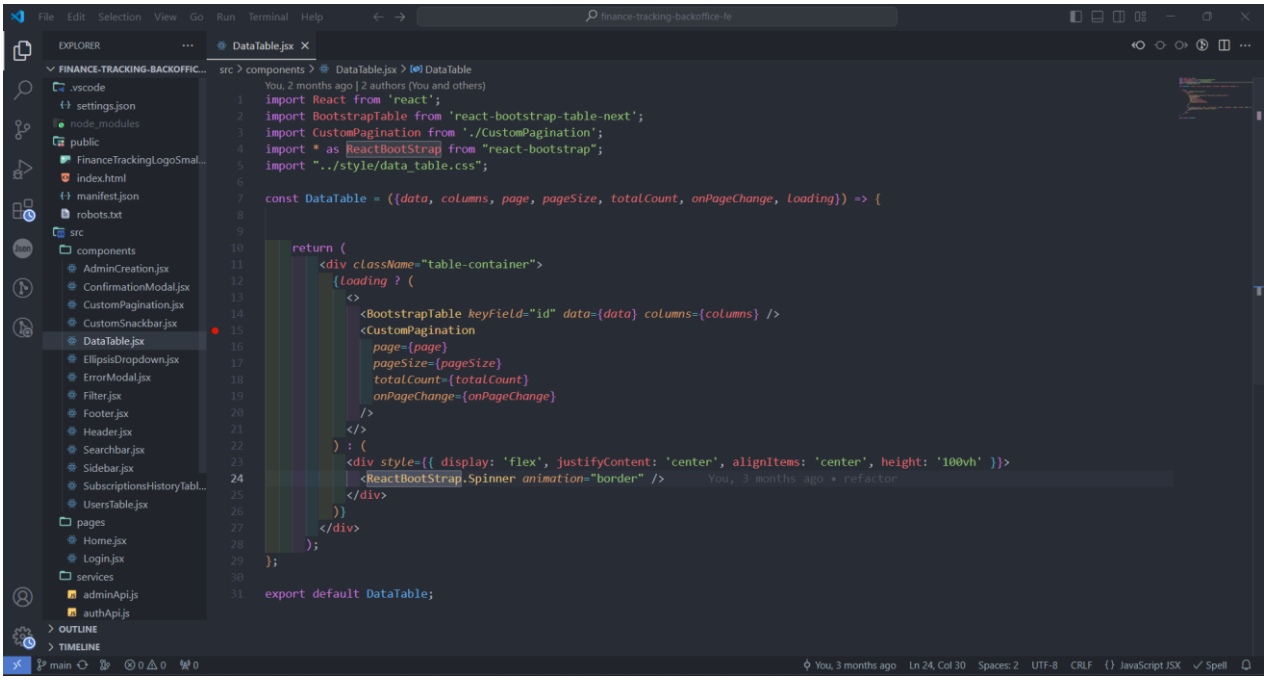


Рис. 2.19 середовище розробки Visual Studio Code

2.1.12 Середовище розробки Rider

JetBrains Rider потужне кросплатформове інтегроване середовище, створене компанією JetBrains для розробки програмного забезпечення під платформу .Net . Rider підтримає такі мови, як: C#, VB.NET, F# та доступний на різних операційних системах. Він надає понад 2000 інспекцій, сотні контекстних дій та рефакторингів коду, вбудований відладчик. Незважаючи на це, Rider – дуже швидке та зручне середовище розробки.

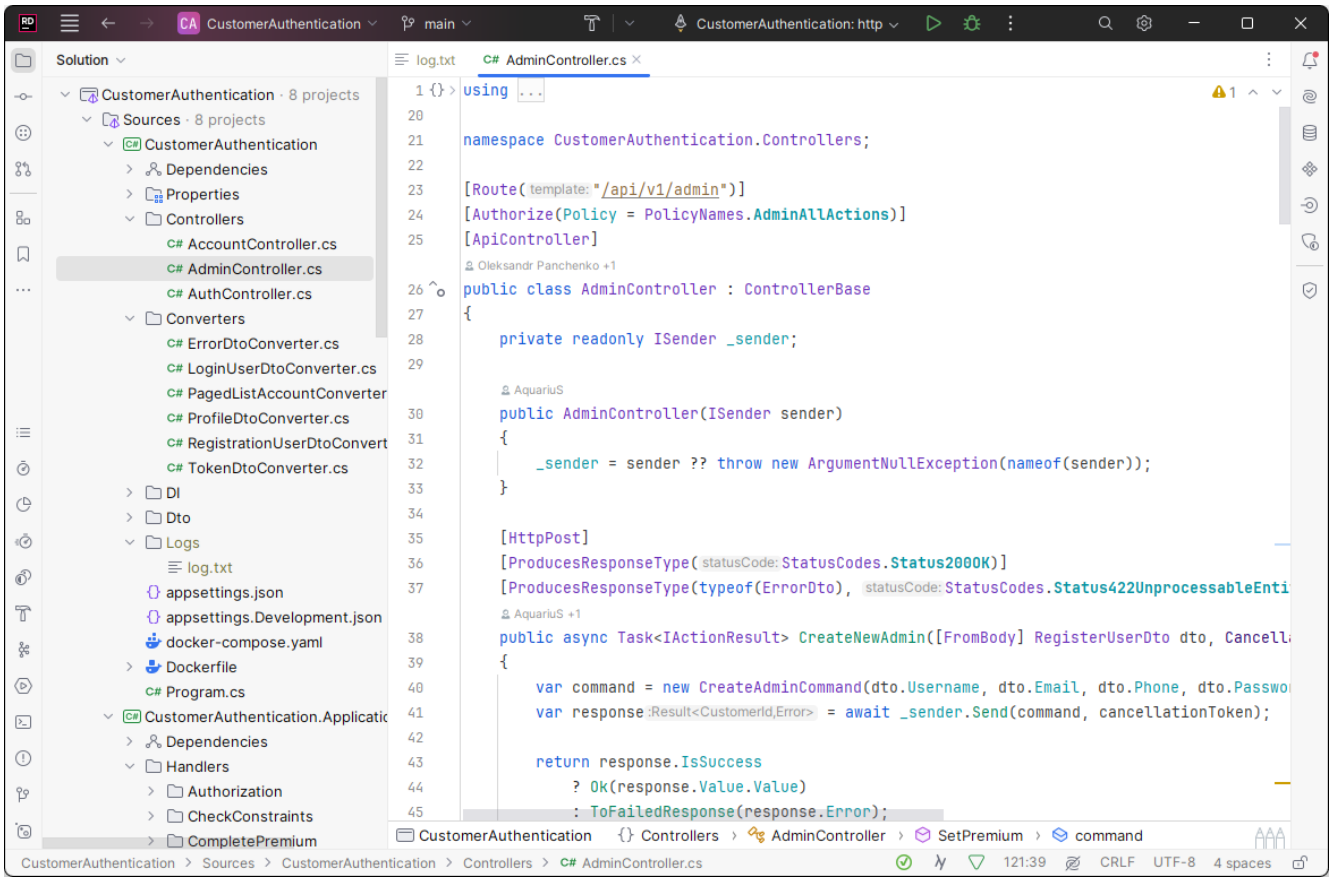


Рис. 2.20 Середовище розробки Rider

2.1.13 Docker

Docker використовується для управління ізольованими контейнерами. У межах цієї ізольованої частини можна встановити потрібну операційну систему і код буде працювати так, ніби він на комп’ютері один.

Для розробки цього проекту в docker контейнер я помістив PostgreSQL та сервіси, які необхідні для роботи з Kafka, це: Zookeeper та Broker. Нижче ви можете побачити, як це виглядає (рис. 2.21).

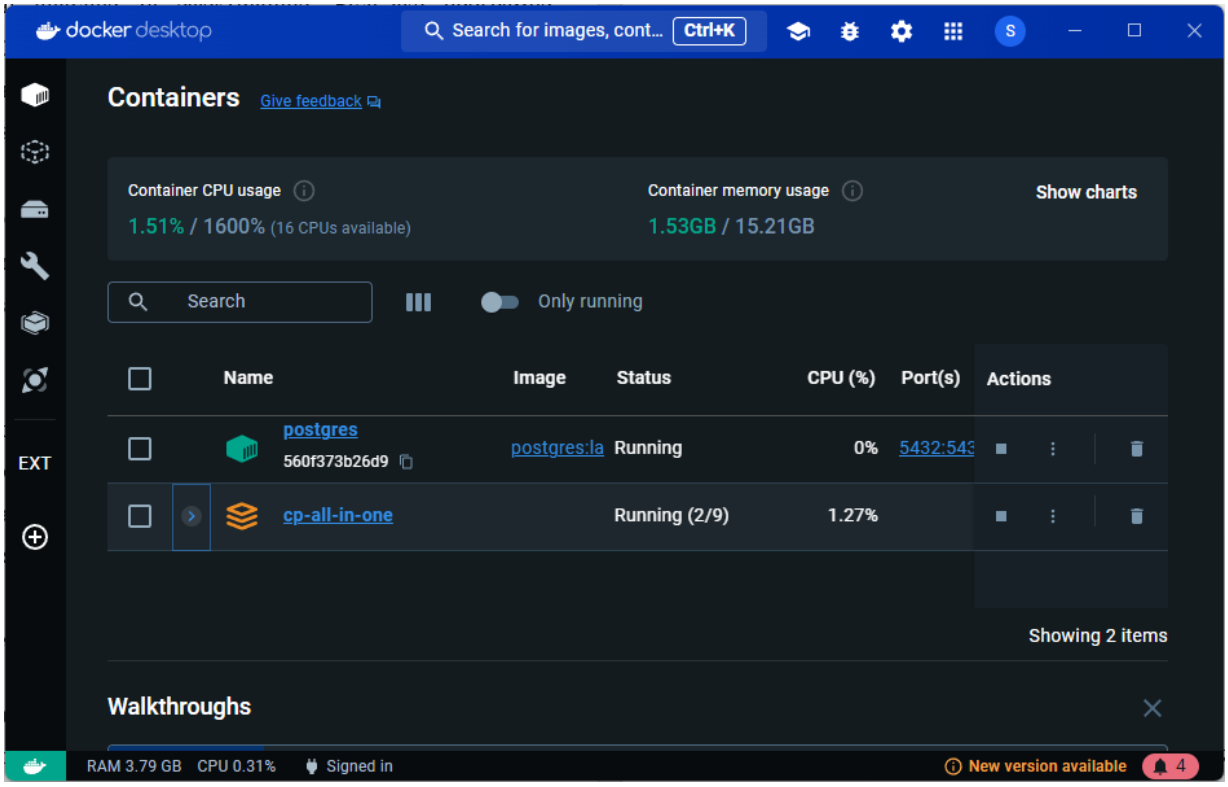


Рис. 2.21 Інтерфейс Docker та контейнери в ньому

2.1.14 Система контролю версій Git

Для зручності розробки програмного забезпечення в роботі використано систему контролю версій Git. Це система, яка зберігає зміни у файлах та дозволяє розроблювати різний функціонал у окремих створених гілках, таким чином, задачу не потрібно обов’язково завершувати, щоб почати наступну, потрібно лише зафіксувати зміни і переключитись на іншу гілку.

Рис. 2.22 Логотип системи контролю версій Git

2.1.15 Автентифікація користувачів за допомогою JWT

В проекті для автентифікації користувача використано JSON Web token (JWT). Це токен доступу на основі JSON, в якому можна зберігати додаткову інформацію про користувача, а саме: унікальний ідентифікатор, електронну пошту, телефон, роль в системі, тощо. Використання JWT дає змогу налаштувати права доступу користувачів та відокремити функціонал, який буде доступний для окремої ролі або груп ролів. У застосунку є 4 ролі, а саме:

– Admin: адміністратор не має прав користування основним застосунком, в його юрисдикції тільки окремий застосунок для обліку користувачів.
– User: має майже всі функції обліку фінансів, є обмеження на отримання звітності за певний період часу.
– UserPremium: має всі ті ж функції, що і у User, додатково має доступ до отримання звітності.
– BlockedUser: всі функції заблоковано для цієї ролі.

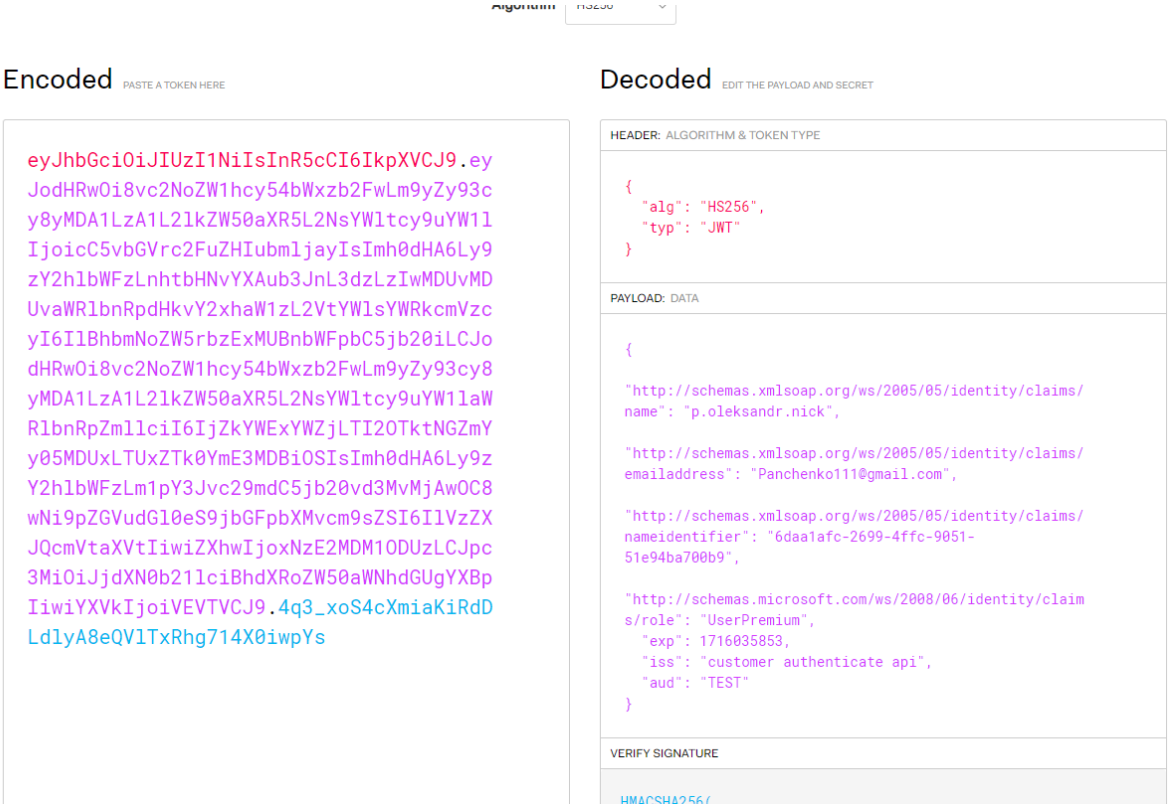


Рис. 2.23 Приклад JWT для преміум користувача

3 ПРОЕКТУВАННЯ ТА РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Планування розробки програмного забезпечення

Під час розробки програмного забезпечення потрібно відслідковувати список та статус задач. Для ефективної роботи, команді обов’язково слід мати дошку проекту, де завдання розбиваються по пріоритету, складністю, таким чином всі працівники, не блокують один одного та ефективно вирішують свої завдання. Особливо це допомагає вирішувати складні та великі задачі, які розбиваються на більш малі і декілька розробників можуть працювати над однією функціональністю. Одним із багатьох таких інструментів є Trello.

Це онлайн-платформа для управління проектами та завданнями. Під кожен проект створюються окремі дошки з певним набором завдань. Інтерфейс складається з простих систем канбан-дощок для організації роботи із завданнями. З основних плюсів цього інструменту це:

- можна впровадити у роботу будь-якому етапі;
- безліч розширень для інтегрування;
- застосунок працює на всіх версіях браузерів;

Для розробки свого проекту, я додав такі колонки:

- «To Do» - для завдань, які потрібно зробити;
- «In Progress» - для завдань, які вже в роботі;
- «Ready For Testing» - для завдань, які вже зроблені та чекають на тестування;
- «Done» - для завдань, які вже виконані та протестовані;

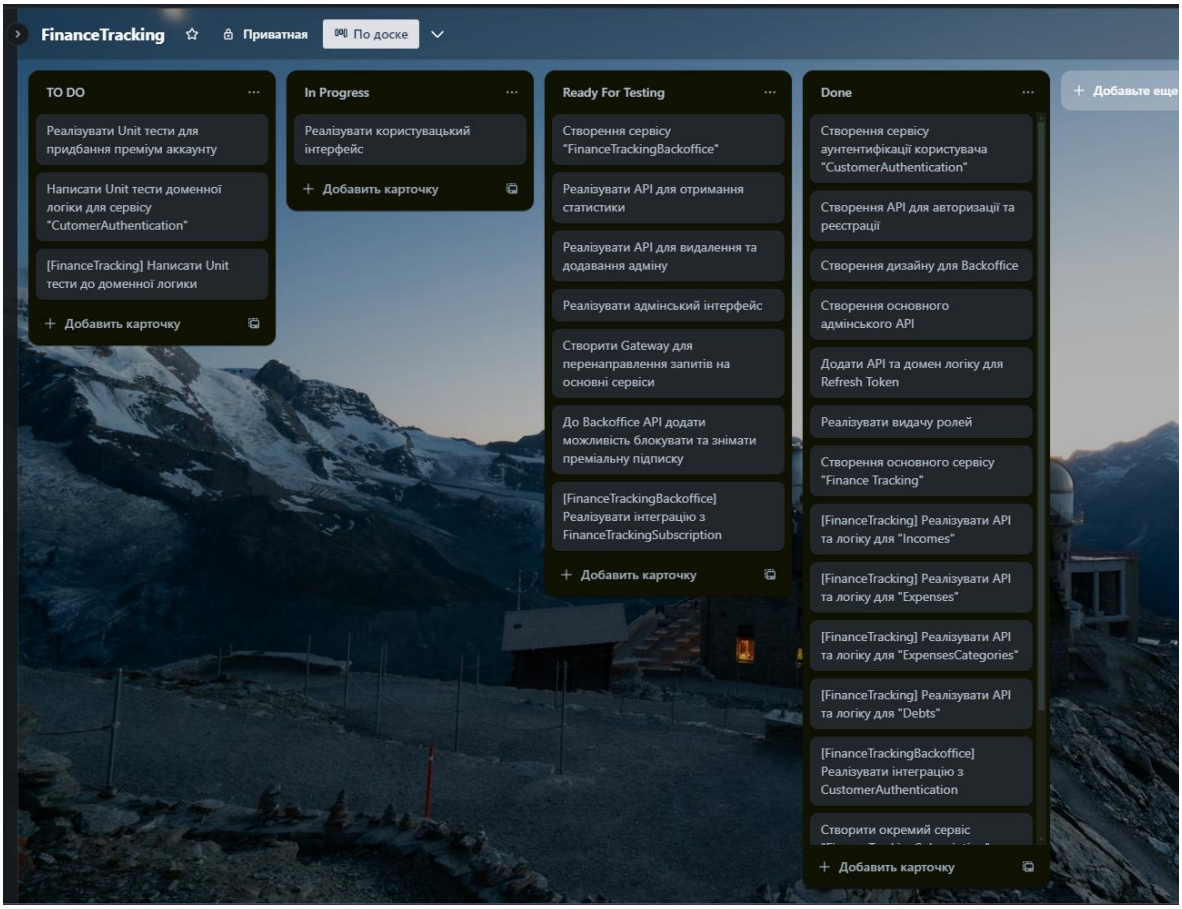


Рис. 3.1 Дошка проекту

3.2 Проектування архітектури

Декілька років тому був популярний архітектурний стиль моноліту. Використання цього стилю означало, що вся серверна частина була реалізована в рамках одного проекту, це додавало зручності розробки на початкових етапах. Однак проект з часом розростається, здатність до масштабованості втрачається, підтримувати таке стає набагато важче та дорожче. Тому для високонавантажених веб-застосунків де швидкодія, безпека та масштабування мають важливе значення, набув популярність архітектурний стиль мікросервісів.

Використовуючи цей підхід, створюють таке програмне забезпечення, що складається з невеликих, незалежних, не тісно зв'язаних сервісів. Завдяки цьому, новий функціонал можливо легко інтегрувати в систему не переробляючи минулу логіку. Головна перевага цього архітектурного стилю від моноліту є те,

що при припиненні роботи одного з мікросервісів, застосунок продовжує працювати та оброблю запити клієнтів, але без функціоналу за який відповідав відключений мікросервіс.

Нижче наведено діаграму компонентів, яка представляє архітектуру проекту загалом.

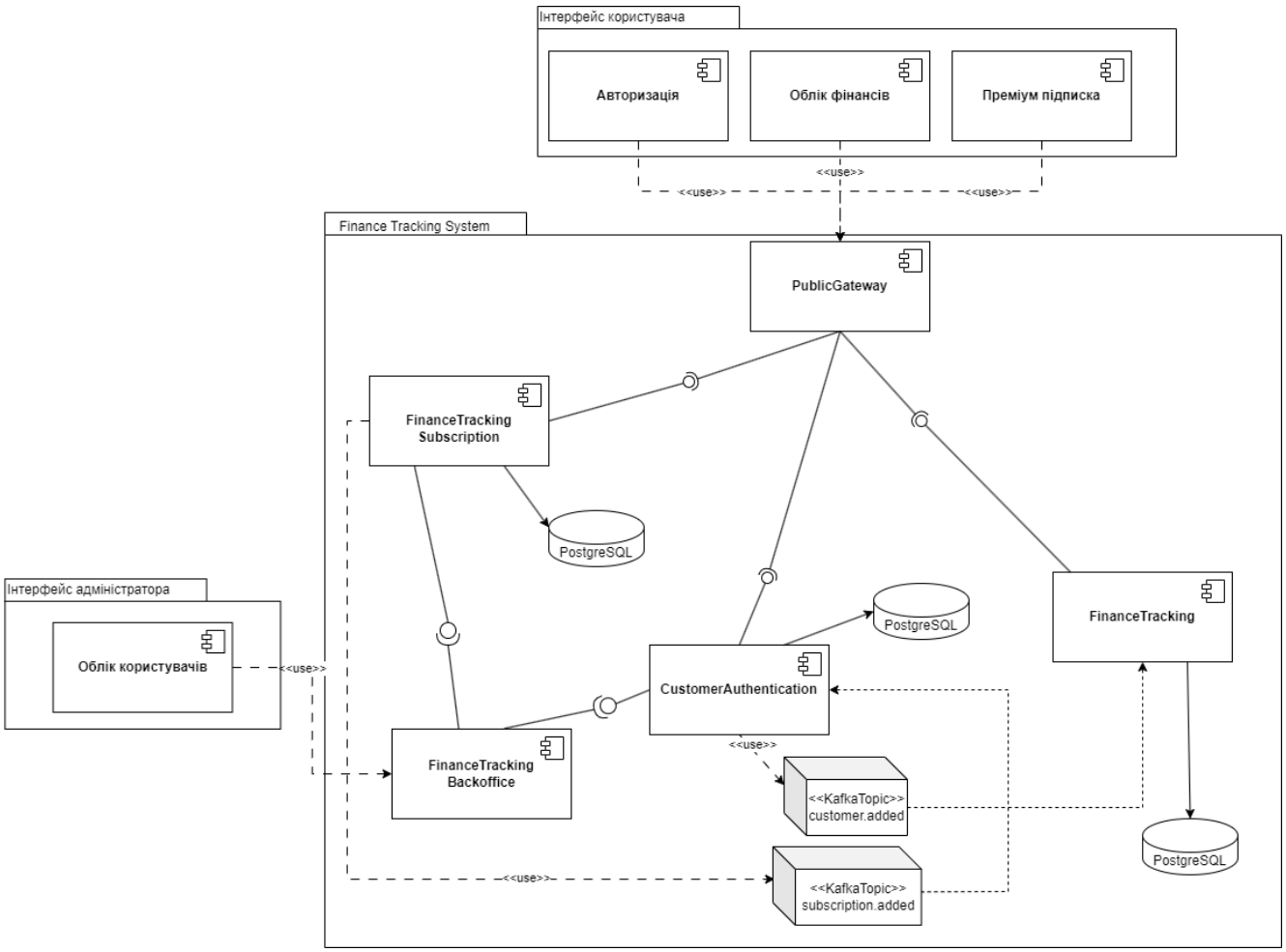


Рис. 3.2 Діаграма компонентів архітектури серверної частини

Для архітектури в середині мікросервісів використано Domain-Driven Design (DDD). Це підхід до проектування програмного забезпечення, який вивчає предметну область бізнесу в цілому. Основний принцип полягає в розділенні програм на домени або контексти.

Домен – це предметна область, яка описує проблеми та цілі бізнесу, наприклад це, діяльність підприємства, яку потрібно автоматизувати. Домени поділяються на субдомени – це області, які відповідають окремій проблематиці.

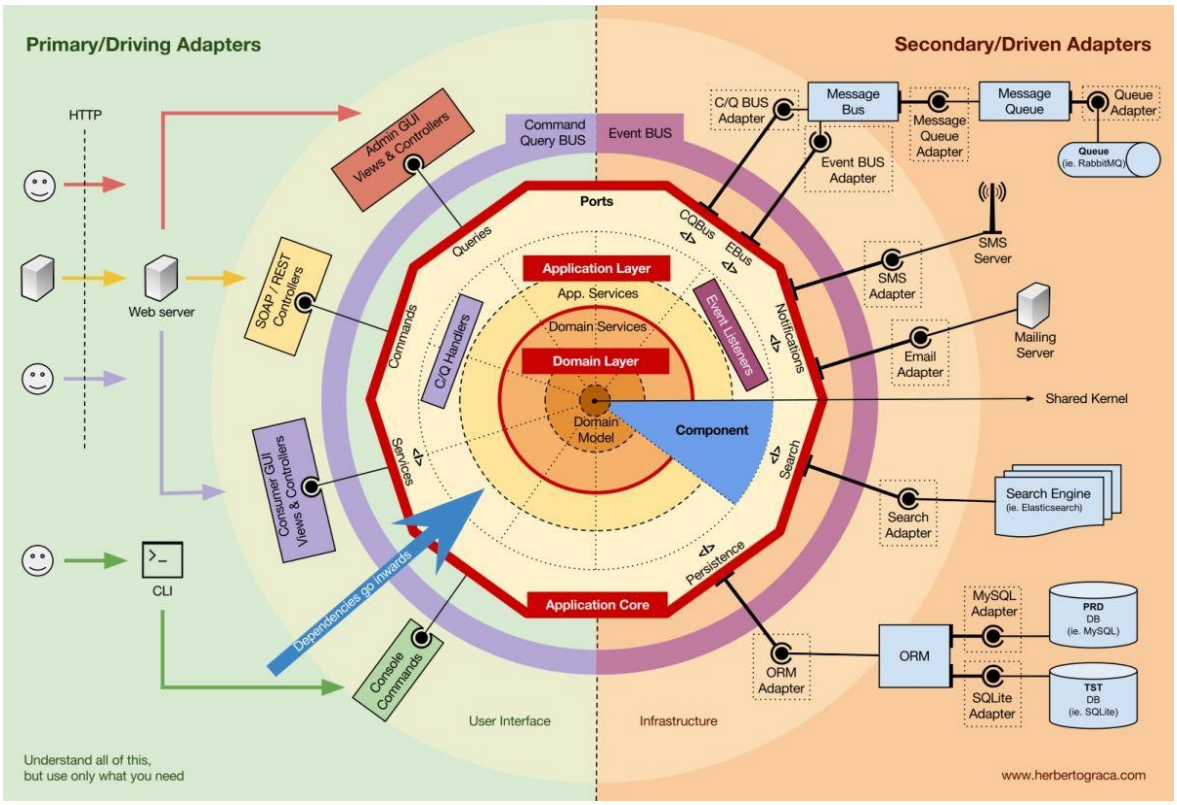


Рис. 3.3 Вигляд структури Domain-Driven Design

Дипломна робота складається з п’яти мікросервісів:

– CustomerAuthentication: цей сервіс відповідає за ідентифікацію користувача та їх облік. Він містить детальну інформацію про всіх користувачів.
– FinanceTrackingBackoffice: виступає в ролі API шлюзу, який посилає запити на інші сервіси. Не зберігає ніяких даних. Використовується виключно адміністратором.
– FinanceTrackingSubscription: відповідальний за преміум статуси. Містить детальну інформацію про підписки та транзакції.
– FinanceTracking: відповідальний за облік коштів. Зберігає в собі інформацію про витрати, заощадження, борги користувачів.
– PublicGateway: API-шлюз, для перенаправлення запитів до відповідних мікросервісів.

Структура мікросервісів поділені на такі шари:

- Controller: в цьому шарі розташовується контролери та налаштовується інфраструктура проекту.
- ApplicationService: в цьому шарі розташовується оброблювачі бізнес логіки.
- Data Access Layer (DAL): в цьому шарі відбувається конфігурація та взаємодія з базою даних.
- Kafka: в цьому шарі, відбувається налаштування Kafka, оброблення повідомлень з топіку та запис в топік.
- Domain: в цьому шарі розташовуються доменні сутності, які необхідні для бізнес логіки.
- Infrastructure: в цьому шарі розташовується додаткові інфраструктурні класи, наприклад хешування паролю чи авторизації.

Нижче наведено діаграму пакетів сервісу «FinanceTracking», який відповідає за функціональність обліку фінансів, яка відображає залежності між пакетами.

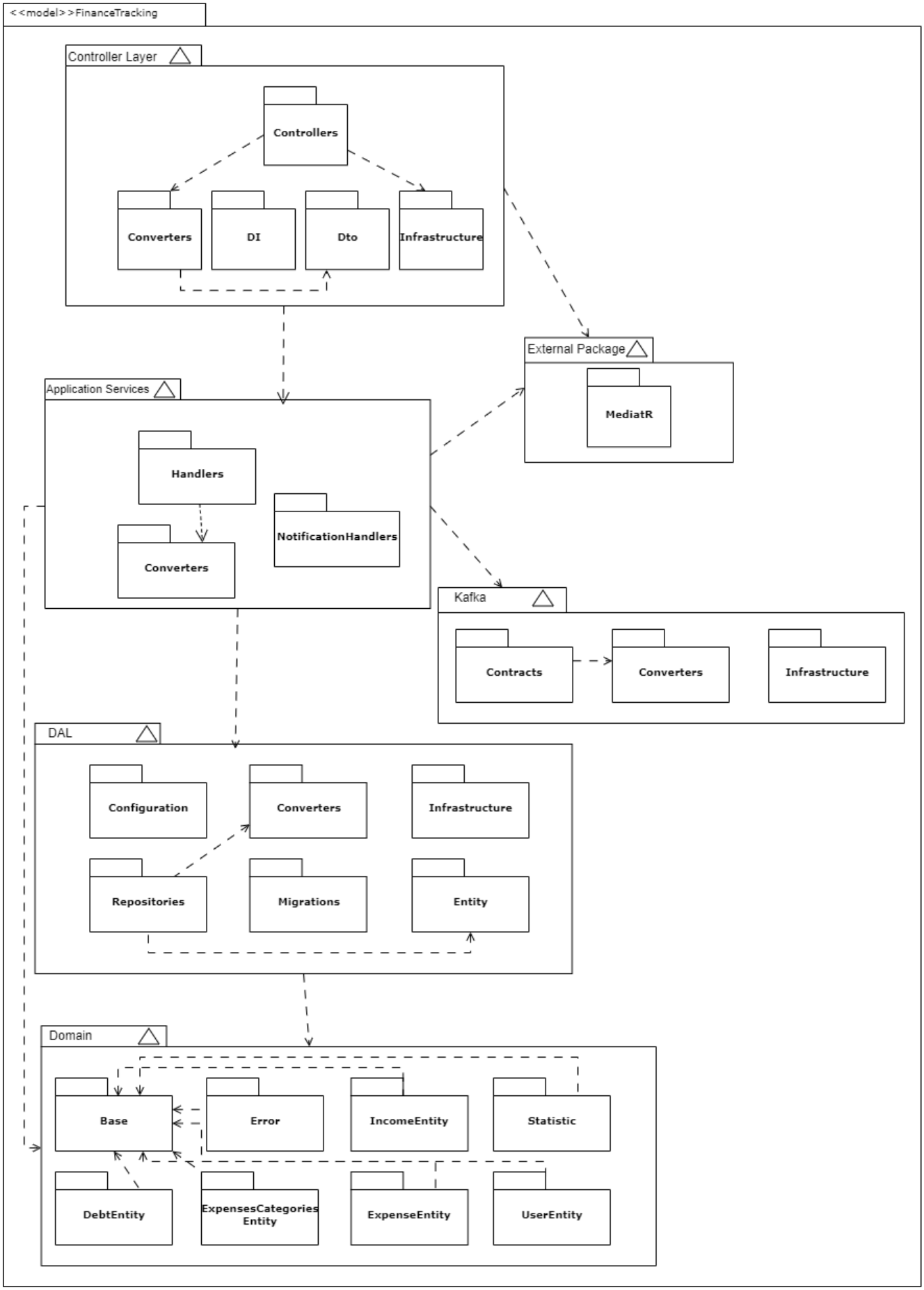


Рис. 3.4 Діаграма пакетів структури мікросервісу “FinanceTracking”

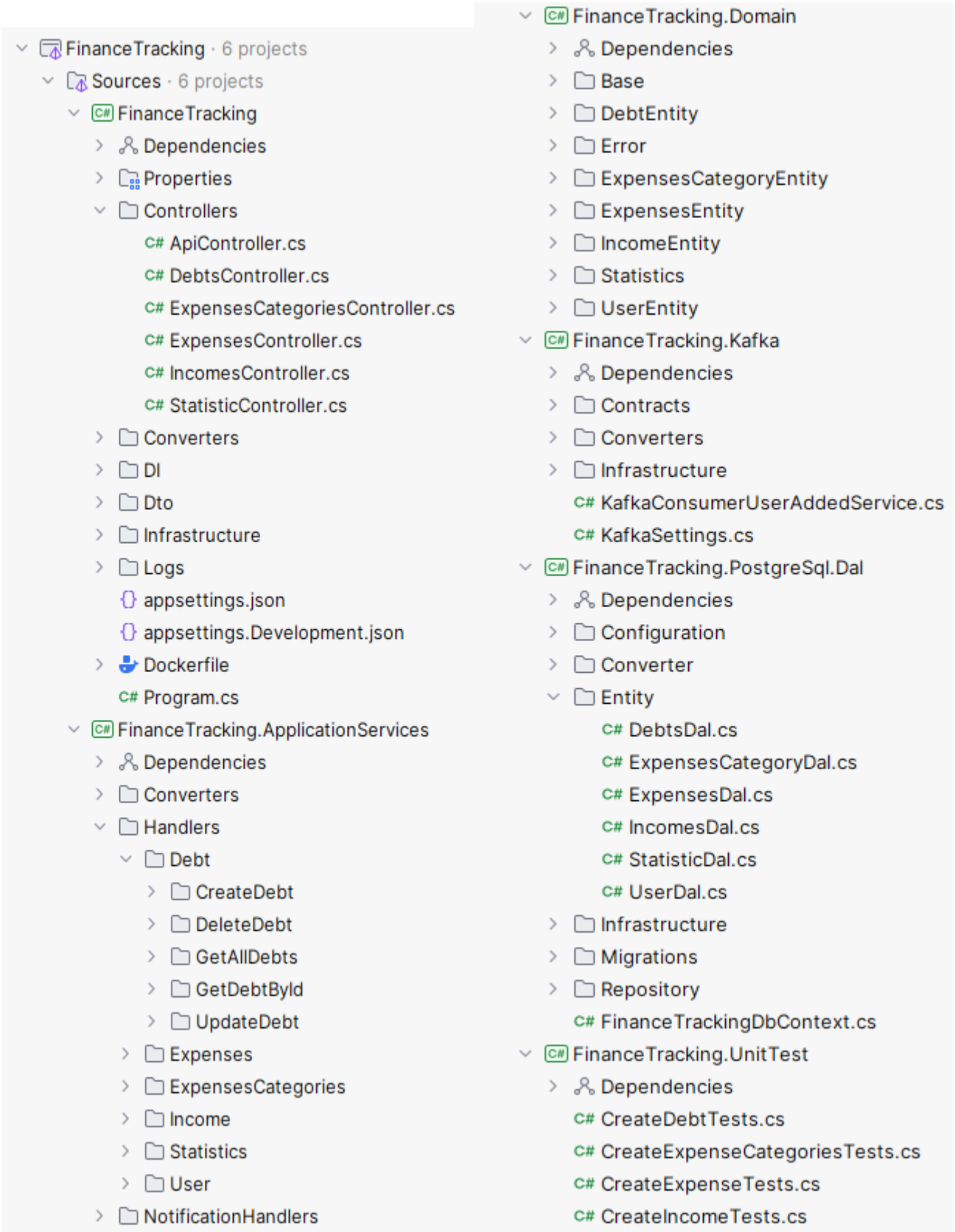


Рис. 3.5 Приклад структури мікросервісу

Також в проекті є клієнт для адміністратора та користувача. Мікросервіси між собою комунікують за допомогою HTTP запитів, а важлива інформація, котру в жодному разі не можна втратити, передається брокером повідомлень Kafka.

API мікросервісів побудовано за допомогою архітектурного стилю REST. Це набір правил для реалізації серверного застосунку для того, щоб компоненти системи могли легко обмінюватись даними, також це допомагає в реалізації масштабованості застосунку.

Рис. 3.6 API мікросервіса, розроблений за допомогою REST

За допомогою використання Ocelot Gateway в дипломній роботі створюється надійний захист конфіденційних даних та спрощується інтеграція з сторонніми сервісами. Його використання дозволяє створити єдину точку входу до застосунку тим самим забезпечує централізоване керування, моніторинг. На рисунку 3.7 показано, як налаштований Ocelot в дипломній роботі.

```
{
  "UpstreamPathTemplate": "/api/v1/debts",
  "UpstreamHttpMethod": [ "GET" ],
  "DownstreamPathTemplate": "/api/v1/debts?page={page}&pagSize={pageSize}&fromDate={fromDate}&toDate={toDate}",
  "DownstreamScheme": "http",
  "DownstreamHostAndPorts": [
    {
      "Host": "localhost",
      "Port": 5208
    }
  ]
},
{
  "UpstreamPathTemplate": "/api/v1/debts",
  "UpstreamHttpMethod": [ "POST" ],
  "DownstreamPathTemplate": "/api/v1/debts",
  "DownstreamScheme": "http",
  "DownstreamHostAndPorts": [
    {
      "Host": "localhost",
      "Port": 5208
    }
  ]
},
{
  "UpstreamPathTemplate": "/api/v1/debts/{debtId}",
  "UpstreamHttpMethod": [ "GET", "PUT", "DELETE" ],
  "DownstreamPathTemplate": "/api/v1/debts/{debtId}",
  "DownstreamScheme": "http",
  "DownstreamHostAndPorts": [
    {
      "Host": "localhost",
      "Port": 5208
    }
  ]
},
```

Рис. 3.7 Налаштування API-шлюзу Ocelot

Для моніторингу стану системи, в кожному мікросервісі було додано механізм логування. Всі виключні ситуації, дії користувачів потрапляють до файлу, який можна переглянути та швидко і ефективно вирішувати проблеми (рис. 3.8).

```
2024-05-06 16:54:57.200 +03:00 [DBG] Opening connection to database 'CustomerAuthentication' on server ''.
2024-05-06 16:54:58.513 +03:00 [DBG] Opened connection to database 'CustomerAuthentication' on server 'tcp://localhost:5432'.
2024-05-06 16:54:58.522 +03:00 [DBG] Closing connection to database 'CustomerAuthentication' on server 'tcp://localhost:5432'.
2024-05-06 16:54:58.536 +03:00 [DBG] Closed connection to database 'CustomerAuthentication' on server '' (6ms).
2024-05-06 16:54:58.547 +03:00 [DBG] Creating DbCommand for 'ExecuteScalar'.
2024-05-06 16:54:58.556 +03:00 [DBG] Created DbCommand for 'ExecuteScalar' (5ms).
2024-05-06 16:54:58.559 +03:00 [DBG] Initialized DbCommand for 'ExecuteScalar' (12ms).
2024-05-06 16:54:58.560 +03:00 [DBG] Opening connection to database 'CustomerAuthentication' on server ''.
2024-05-06 16:54:58.580 +03:00 [DBG] Opened connection to database 'CustomerAuthentication' on server 'tcp://localhost:5432'.
2024-05-06 16:54:58.591 +03:00 [DBG] Executing DbCommand [Parameters=[], CommandType='"Text"', CommandTimeout='30']
SELECT EXISTS (
    SELECT 1 FROM pg_catalog.pg_class c
    JOIN pg_catalog.pg_namespace n ON n.oid=c.relnamespace
    WHERE n.nspname='public' AND
          c.relname='__EFMigrationsHistory'
)
2024-05-06 16:54:58.640 +03:00 [INF] Executed DbCommand (52ms) [Parameters=[], CommandType='"Text"', CommandTimeout='30']
SELECT EXISTS (
    SELECT 1 FROM pg_catalog.pg_class c
    JOIN pg_catalog.pg_namespace n ON n.oid=c.relnamespace
    WHERE n.nspname='public' AND
          c.relname='__EFMigrationsHistory'
)
2024-05-06 16:54:58.646 +03:00 [DBG] Closing connection to database 'CustomerAuthentication' on server 'tcp://localhost:5432'.
2024-05-06 16:54:58.649 +03:00 [DBG] Closed connection to database 'CustomerAuthentication' on server '' (3ms).
2024-05-06 16:54:58.653 +03:00 [DBG] Opening connection to database 'CustomerAuthentication' on server ''.
2024-05-06 16:54:58.660 +03:00 [DBG] Opened connection to database 'CustomerAuthentication' on server 'tcp://localhost:5432'.
2024-05-06 16:54:58.662 +03:00 [DBG] Closing connection to database 'CustomerAuthentication' on server 'tcp://localhost:5432'.
2024-05-06 16:54:58.663 +03:00 [DBG] Closed connection to database 'CustomerAuthentication' on server '' (1ms).
2024-05-06 16:54:58.665 +03:00 [DBG] Creating DbCommand for 'ExecuteScalar'.
2024-05-06 16:54:58.666 +03:00 [DBG] Created DbCommand for 'ExecuteScalar' (1ms).
2024-05-06 16:54:58.667 +03:00 [DBG] Initialized DbCommand for 'ExecuteScalar' (2ms).
2024-05-06 16:54:58.668 +03:00 [DBG] Opening connection to database 'CustomerAuthentication' on server ''.
2024-05-06 16:54:58.670 +03:00 [DBG] Opened connection to database 'CustomerAuthentication' on server 'tcp://localhost:5432'.
2024-05-06 16:54:58.671 +03:00 [DBG] Executing DbCommand [Parameters=[], CommandType='"Text"', CommandTimeout='30']
SELECT EXISTS (
    SELECT 1 FROM pg_catalog.pg_class c
    JOIN pg_catalog.pg_namespace n ON n.oid=c.relnamespace
    WHERE n.nspname='public' AND
          c.relname='__EFMigrationsHistory'
)
```

Рис. 3.8 Приклад логів у файлі

3.3 Проектування інтерфейсу

Кожен користувач повинен бачити красивий дизайн та зручний інтерфейс, це є важливим пунктом до заохочення нових клієнтів. Тому, щоб бути в конкурентному середовищі треба спроектувати якісний UX/UI дизайн, для цього в роботі використано Figma (рис. 3.9).

Це хмарний сервіс для дизайнерів інтерфейсів та веб-розробників. Він дозволяє працювати над проектами на будь-якому браузері та підтримує спільну роботу у реальному часі.

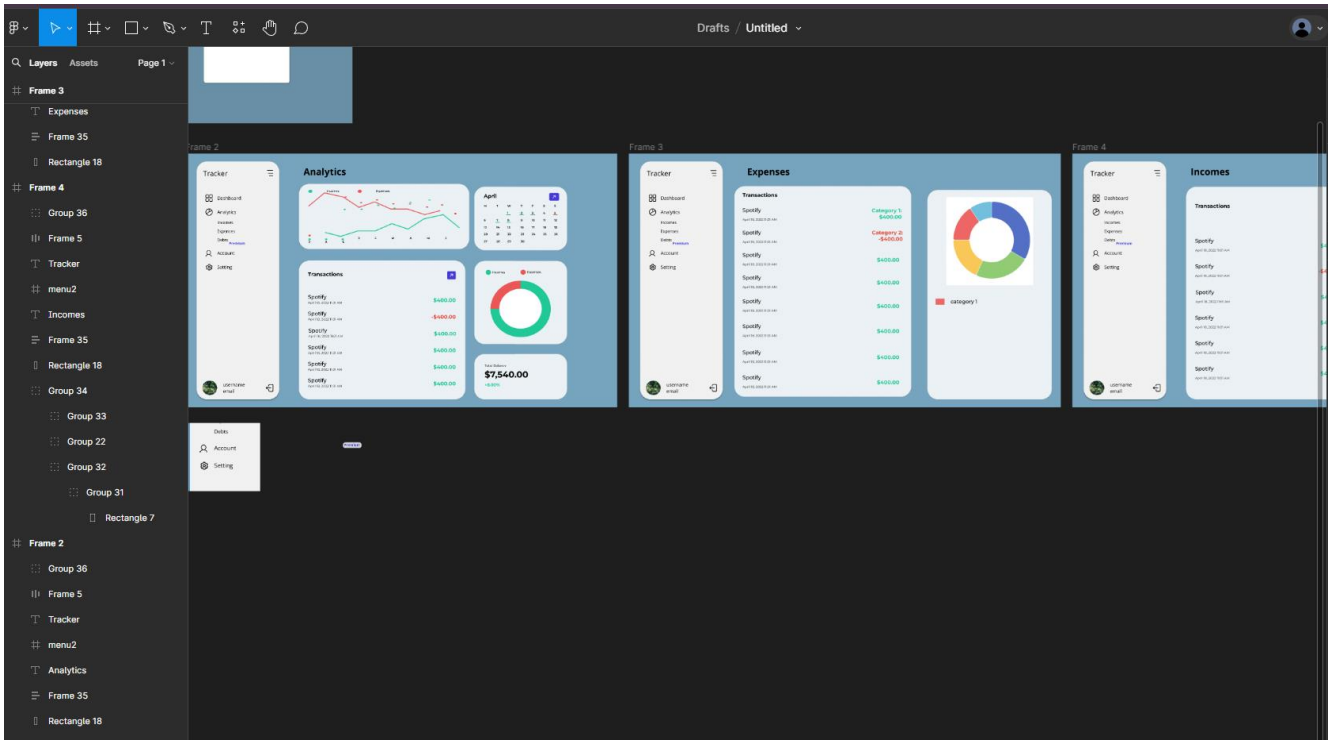


Рис. 3.9 Дизайн проекту Figma

3.3.1 Проектування інтерфейсу адміністратора

На мою думку для адміністратора не потрібен гарний дизайн, замість цього інтерфейс має бути зручним та функціональним. Через це дизайн можна спростити, тим самим зменшити час на розробку. Спираючись на встановленні функціональні вимоги для адміністратора, я виділив декілька компонентів, які мають бути в інтерфейсі:

- Сторінка входу містить тільки форму авторизації.
- У верхній частині екрану повинен розміщуватись логотип «FinanceTracking Backoffice».
- У головній частині сторінки повинен розміщуватись пошук, фільтрація та таблиця користувачів. При наведенні на будь-який запис в таблиці, повинно з’являтись випадаюче меню, з варіантами взаємодії із записом.
- Головна сторінка повинна містити бокове меню, яке забезпечить швидку навігацію. Воно складається з меню створення нового адміністратора, перехід на таблицю всіх користувачів, таблиця преміум підписок, та вихід з облікового запису;

Всі елементи мають логічне положення, що дозволяє швидше розібратися у інтерфейсі застосунку.

Кольори мають великий вплив на емоційне становище, тому кольорова палітра була підібрана у нейтральних кольорах, для того, щоб створити атмосферу концентрації та уважності до деталей. Чорний, білий, та темно-синій кольори, допомагають зменшити відволікання та зосередитись на завданнях.

Так як застосунок повинен підтримуватись на різних пристроях, то обов’язково увага приділялась адаптивності інтерфейсу до різних діагоналей дисплеїв.

Нижче наведено декілька скріншотів сторінки адміністратора.

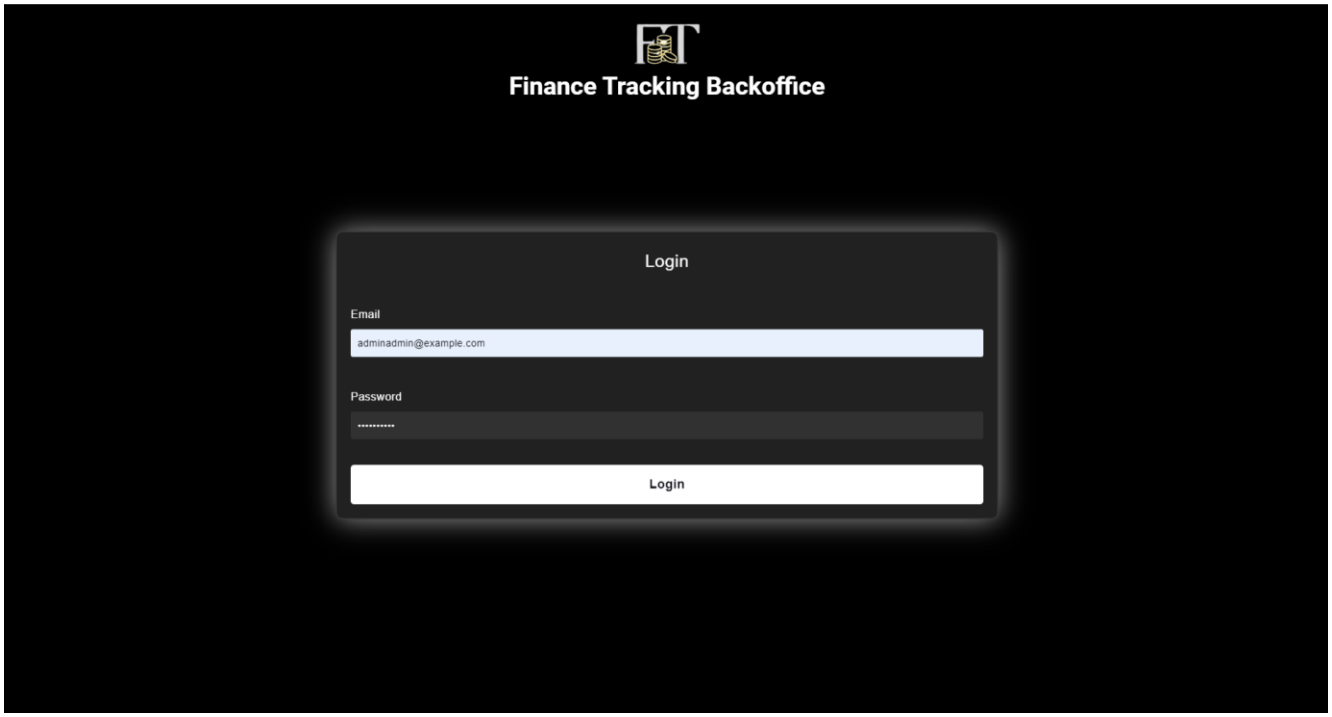


Рис. 3.10 Сторінка авторизації адміністратора

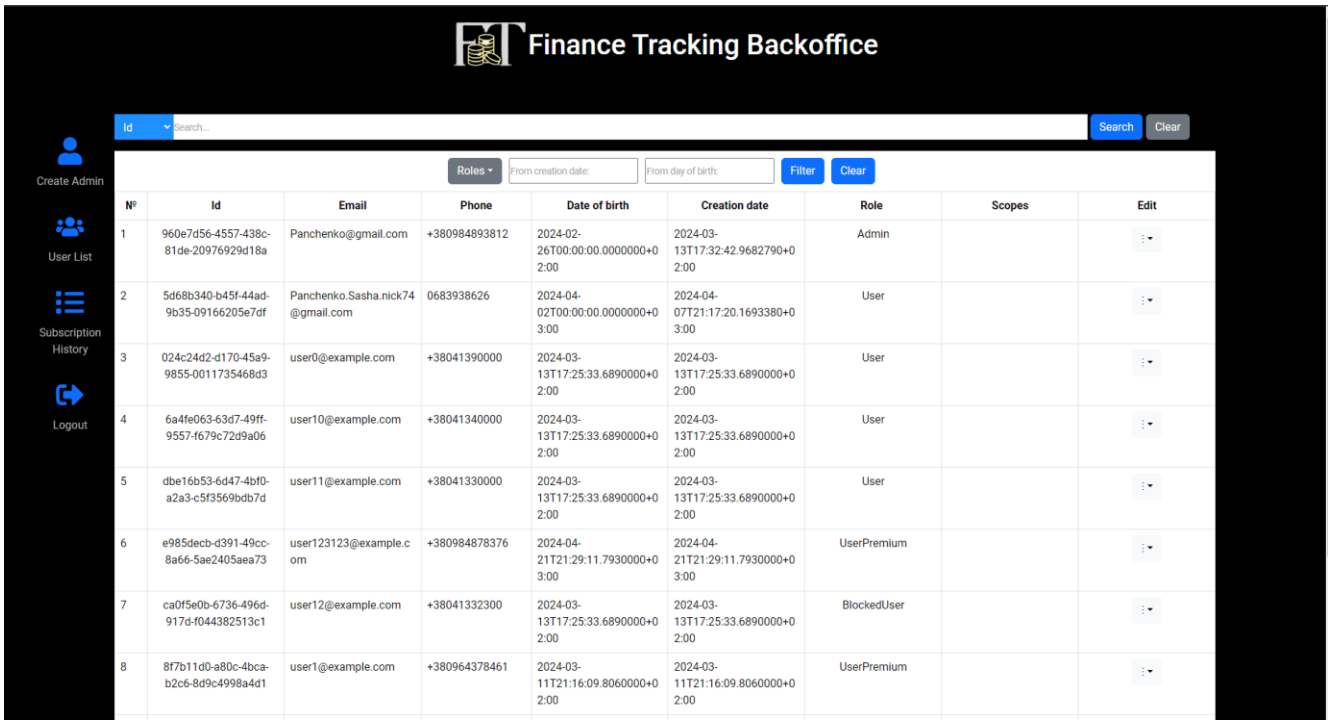


Рис. 3.1 Головна сторінка

На головній сторінці під пошуком знаходиться фільтрація користувачів по ролі, даті створення облікового запису, даті народження (рис. 3.11).

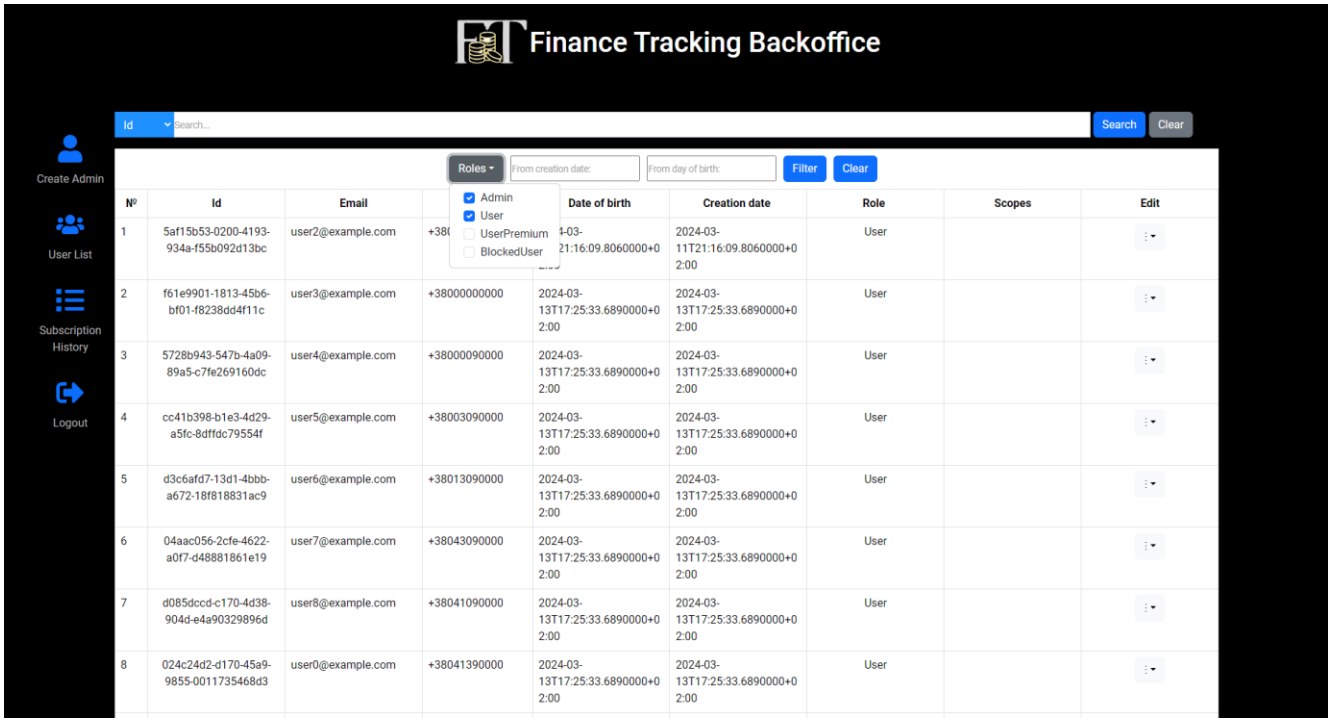


Рис. 3.12 Відфільтровані користувачі

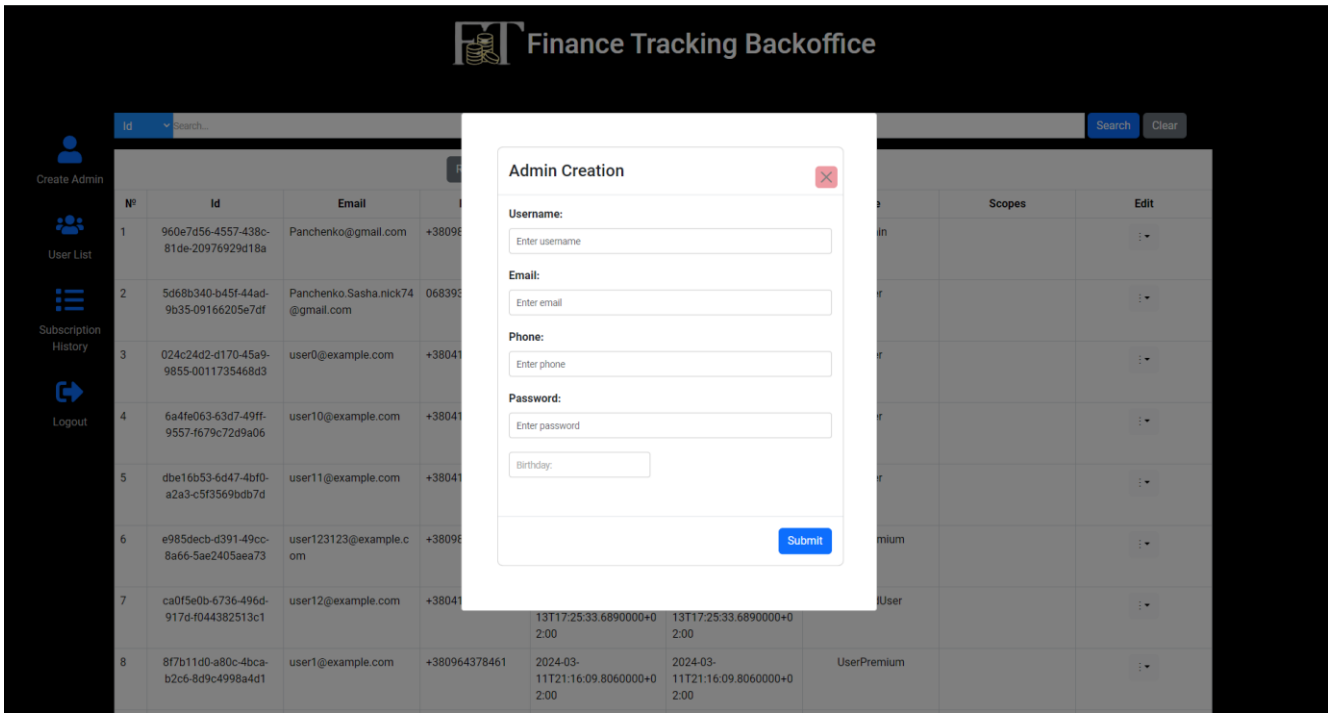
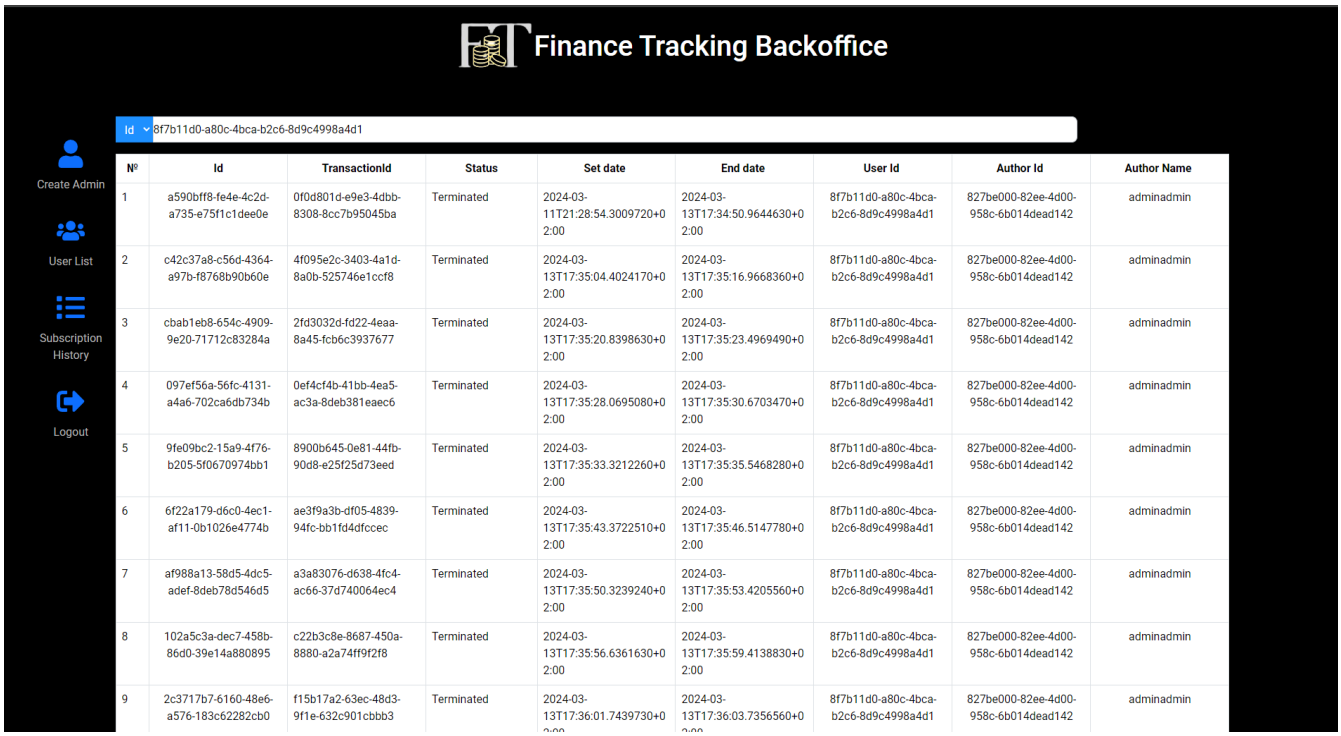


Рис. 3.13 Компонент додавання адміністратора

Для кращого обліку користувачів в застосунку адміністратора також додана можливість перегляду їх історій преміум підписок.



№	Id	TransactionId	Status	Set date	End date	User Id	Author Id	Author Name
1	a590bff8-fe4e-4c2d-a735-e75f1c1dee0e	0f0d801d-e9e3-4dbb-8308-8cc7b95045ba	Terminated	2024-03-11T21:28:54.3009720+02:00	2024-03-13T17:34:50.9644630+02:00	8f7b11d0-a80c-4bca-b2c6-8d9c4998a4d1	827be000-82ee-4d00-958c-6b014dead142	adminadmin
2	c42c37a8-c56d-4364-a97b-f8768b90b60e	4f095e2c-3403-4a1d-8a0b-525746e1ccf8	Terminated	2024-03-13T17:35:04.4024170+02:00	2024-03-13T17:35:16.9668360+02:00	8f7b11d0-a80c-4bca-b2c6-8d9c4998a4d1	827be000-82ee-4d00-958c-6b014dead142	adminadmin
3	cbab1eb8-654c-4909-9e20-71712c83284a	2fd3032d-fd22-4eaa-8a45-fcb6c3937677	Terminated	2024-03-13T17:35:20.8398630+02:00	2024-03-13T17:35:23.4969490+02:00	8f7b11d0-a80c-4bca-b2c6-8d9c4998a4d1	827be000-82ee-4d00-958c-6b014dead142	adminadmin
4	097ef56a-56fc-4131-a4a6-702ca6db734b	0ef4cf4b-41bb-4ea5-ac3a-8deb381eaec6	Terminated	2024-03-13T17:35:28.0695080+02:00	2024-03-13T17:35:30.6703470+02:00	8f7b11d0-a80c-4bca-b2c6-8d9c4998a4d1	827be000-82ee-4d00-958c-6b014dead142	adminadmin
5	9fe09bc2-15a9-4f76-b205-5f0670974bb1	8900b645-0e81-44fb-90d8-e25f25d73eed	Terminated	2024-03-13T17:35:33.3212260+02:00	2024-03-13T17:35:35.5468280+02:00	8f7b11d0-a80c-4bca-b2c6-8d9c4998a4d1	827be000-82ee-4d00-958c-6b014dead142	adminadmin
6	6f22a179-d6c0-4ec1-af11-0b1026e4774b	ae3f9a3b-df05-4839-94fc-bb1fd4dfccec	Terminated	2024-03-13T17:35:43.3722510+02:00	2024-03-13T17:35:46.5147780+02:00	8f7b11d0-a80c-4bca-b2c6-8d9c4998a4d1	827be000-82ee-4d00-958c-6b014dead142	adminadmin
7	af988a13-58d5-4dc5-adef-8deb78d546d5	a3a83076-d638-4fc4-ac66-37d740064ec4	Terminated	2024-03-13T17:35:50.3239240+02:00	2024-03-13T17:35:53.4205560+02:00	8f7b11d0-a80c-4bca-b2c6-8d9c4998a4d1	827be000-82ee-4d00-958c-6b014dead142	adminadmin
8	102a5c3a-dec7-458b-86d0-39e14a880895	c22b3c8e-8687-450a-8880-a2a74ff9f2f8	Terminated	2024-03-13T17:35:56.6361630+02:00	2024-03-13T17:35:59.4138830+02:00	8f7b11d0-a80c-4bca-b2c6-8d9c4998a4d1	827be000-82ee-4d00-958c-6b014dead142	adminadmin
9	2c3717b7-6160-48e6-a576-183c62282cb0	f15b17a2-63ec-48d3-9f1e-632c901cbbb3	Terminated	2024-03-13T17:36:01.7439730+02:00	2024-03-13T17:36:03.7356560+02:00	8f7b11d0-a80c-4bca-b2c6-8d9c4998a4d1	827be000-82ee-4d00-958c-6b014dead142	adminadmin

Рис. 3.14 Сторінка перегляду історії преміум підписок користувача

3.3.2 Проектування інтерфейсу користувача

До UX/UI дизайну інтерфейсу користувача, я приділив найвищу увагу, оскільки це сприятиме до покращення користувацького досвіду та заохочення нових клієнтів.

Спираючись на встановленні функціональні вимоги для користувача, я виділив декілька компонентів, які мають бути в інтерфейсі:

- Для навігації по сайту, повинно використовуватись бокове меню. Воно повинно мати такі пункти: домашня сторінка, сторінка витрат, сторінки доходів, сторінка заборгованостей, сторінка аналітики, сторінка облікового запису.
- На головній сторінці по центру має бути присутнє привітання клієнта та кнопка переходу на сторінку аналітики.
- На сторінці аналітики в правому нижньому куті повинно розміщуватись кругова діаграма співвідношення витрат до надходжень та загальний баланс, а по центру знаходитись список транзакцій, у верхній частині графік витрат та кнопка завантаження звіту.
- На сторінці витрат по центру сторінки має розміщуватись список витрат, а з правої сторони має розміщуватись кругова діаграма з відсотками витрат по категоріям, нижче має знаходитись список категорій та загальна сума витрат. У верхній частині присутні кнопки для створення та видалення категорій, а також кнопка додавання витрати.
- На сторінці боргів, по центру має розміщуватись список боргів та кнопки біля кожного запису для редагування та видалення.
- На сторінці надходжень по центру має розміщуватись список транзакцій та кнопка видалення запису. У верхній частині сторінки має розміщуватись кнопка для додавання доходу.
- Сторінка облікового запису має містити список історії купівлі преміум статусів.

Всі елементи мають логічне положення, що дозволяє швидше розібратися у інтерфейсі застосунку.

Для створення приємного враження при роботі з застосунком, я використовую комбіновану кольорову палітру, а саме холодну та нейтральну. Холодна кольорова палітра сприяє відчуттю спокою, а холодна сприяє до концентрації.

Нижче наведено скріншоти користувацького інтерфейсу.

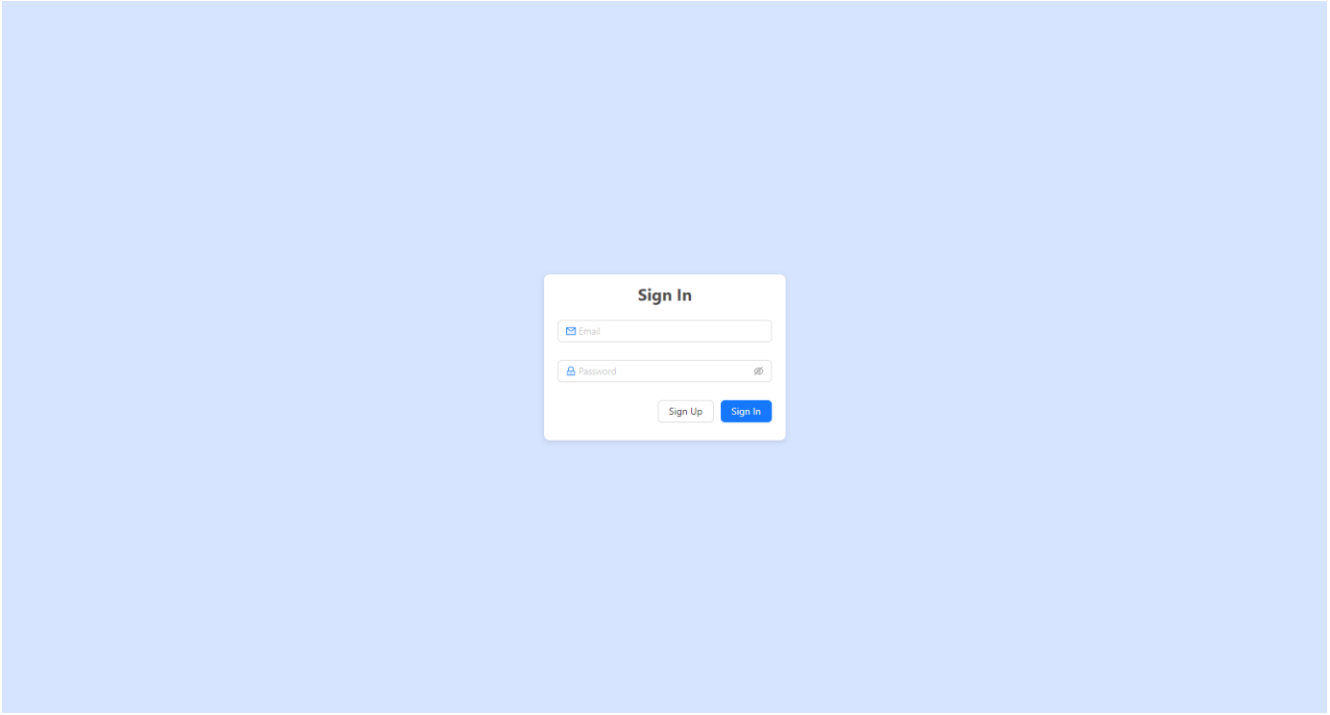


Рис. 3.15 Сторінка авторизації

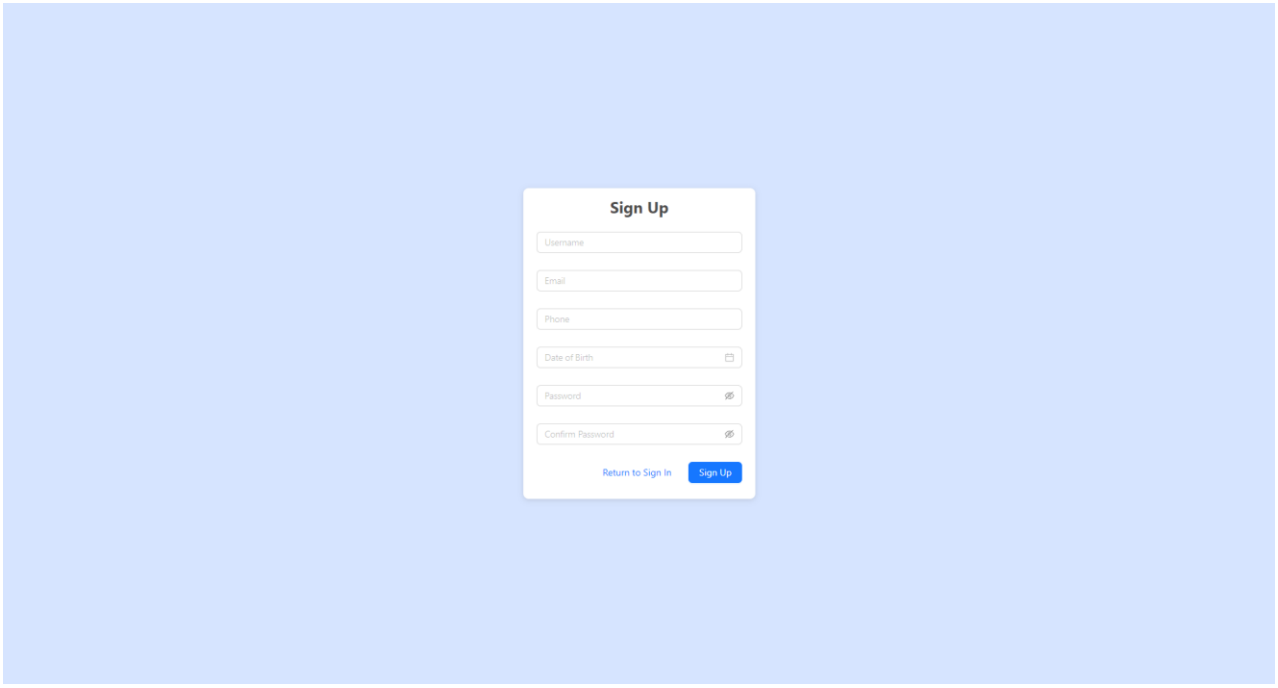


Рис.3.16 Сторінка реєстрації

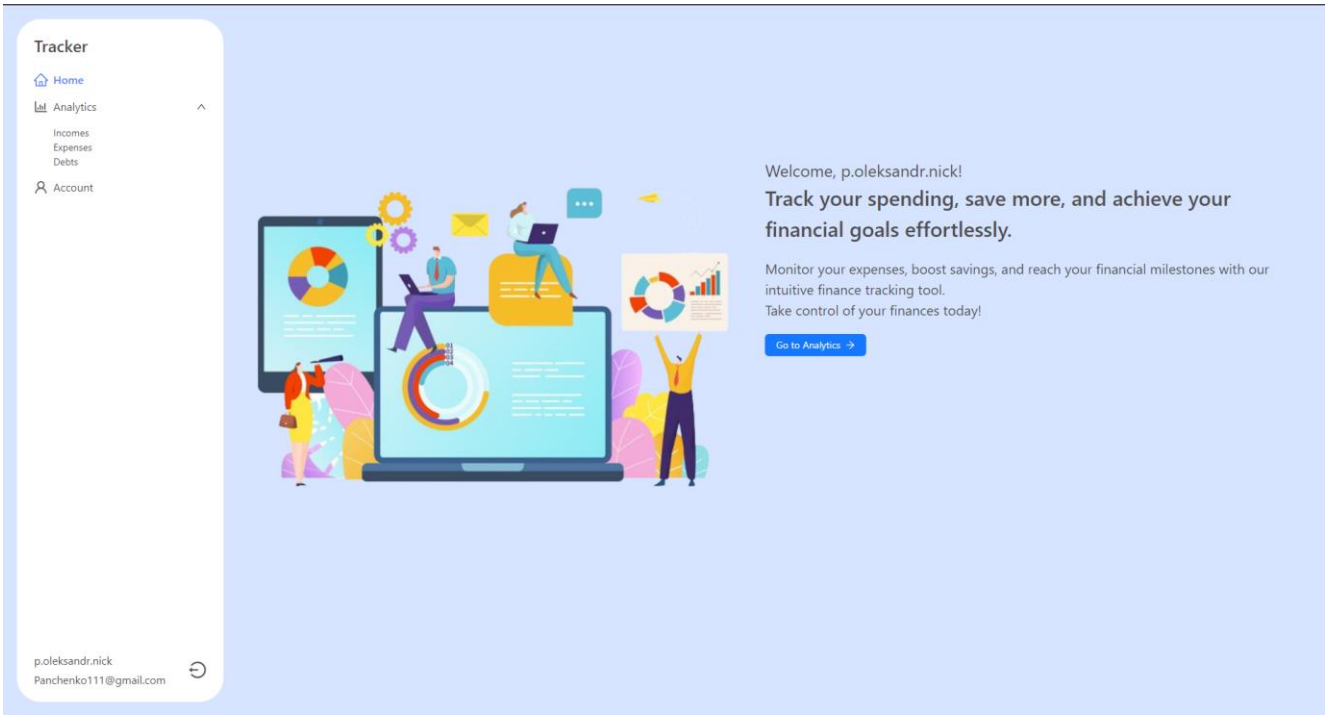


Рис. 3.17 Головна сторінка застосунку

На сторінці аналітики користувач може бачити статистику та отримати звіт по витратам та прибуткам, але отримання звіту доступно тільки для преміальних користувачів.

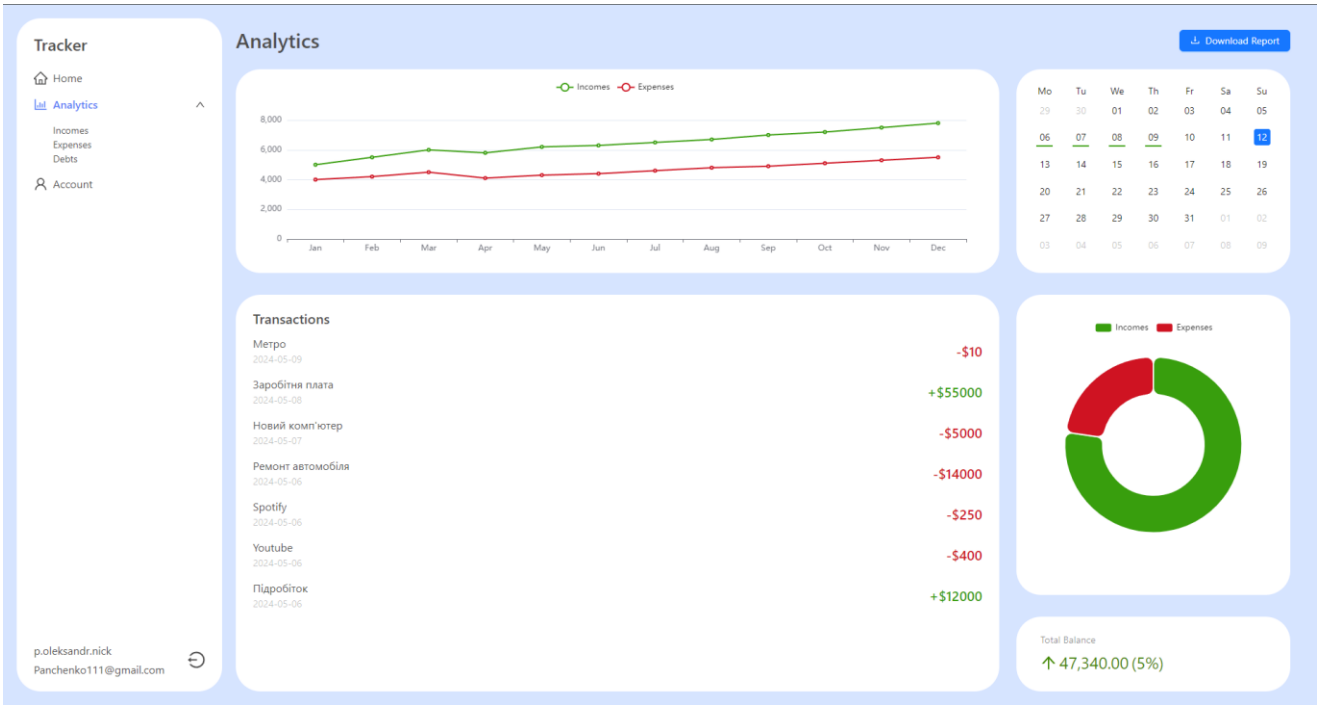


Рис. 3.18 Сторінка аналітики

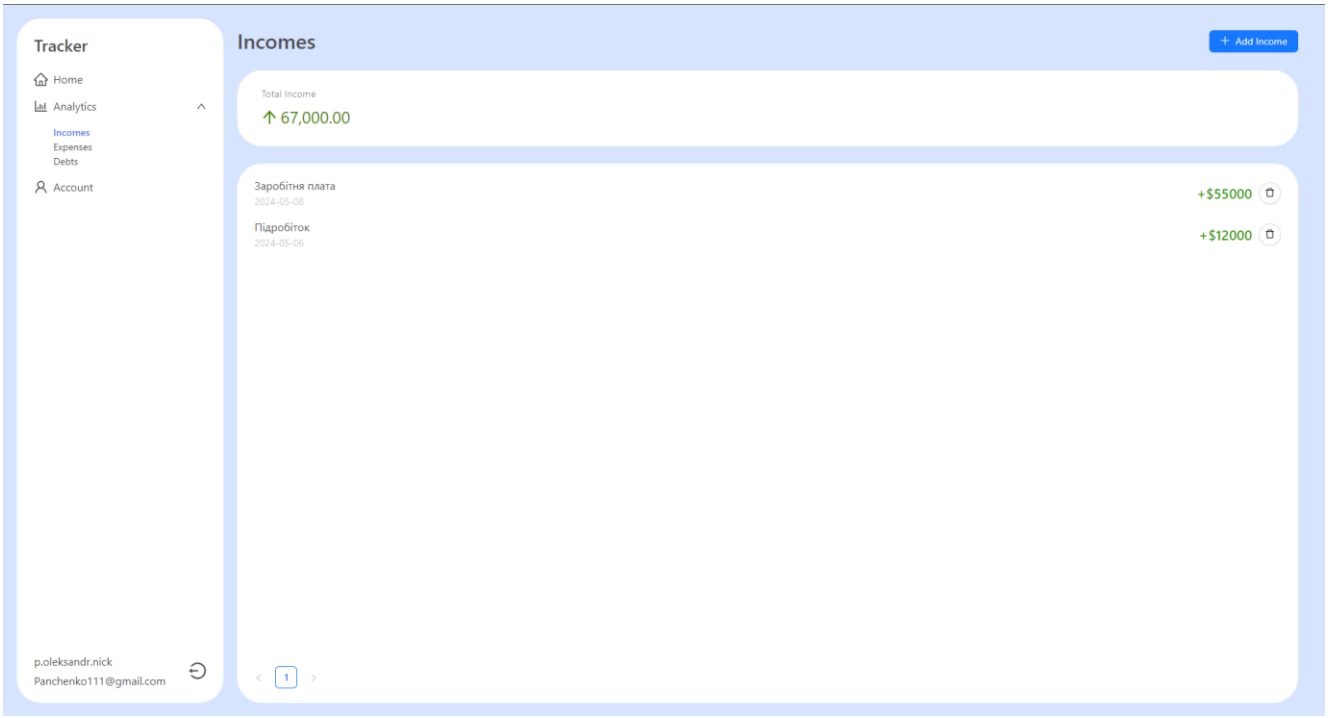


Рис. 3.19 Сторінка доходів

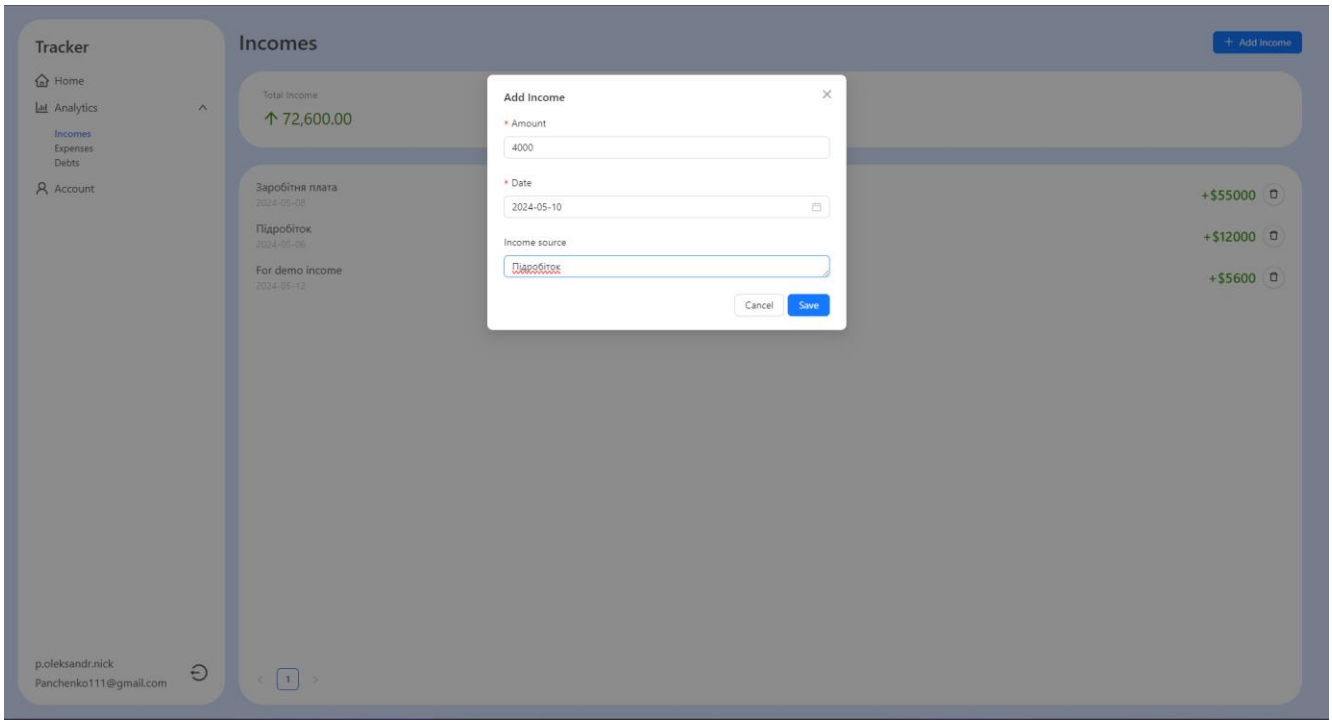


Рис. 3.20 Додавання прибутку

На сторінці витрат користувач має можливість додавати, видаляти категорію та витрати.

Важливо підкреслити, якщо в категорії, яку ви хочете видаляти, є витрати, то вони будуть автоматично видалені разом з категорією із системи.

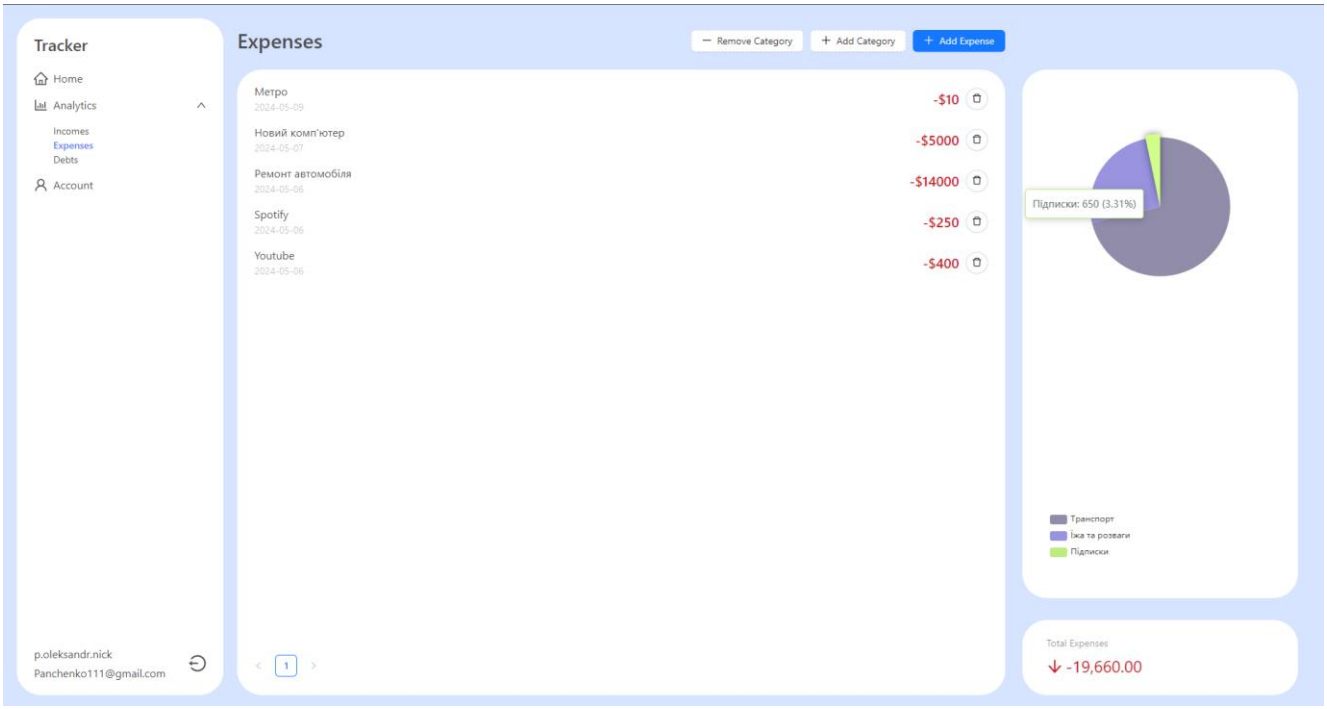


Рис. 3.21 Сторінка витрат

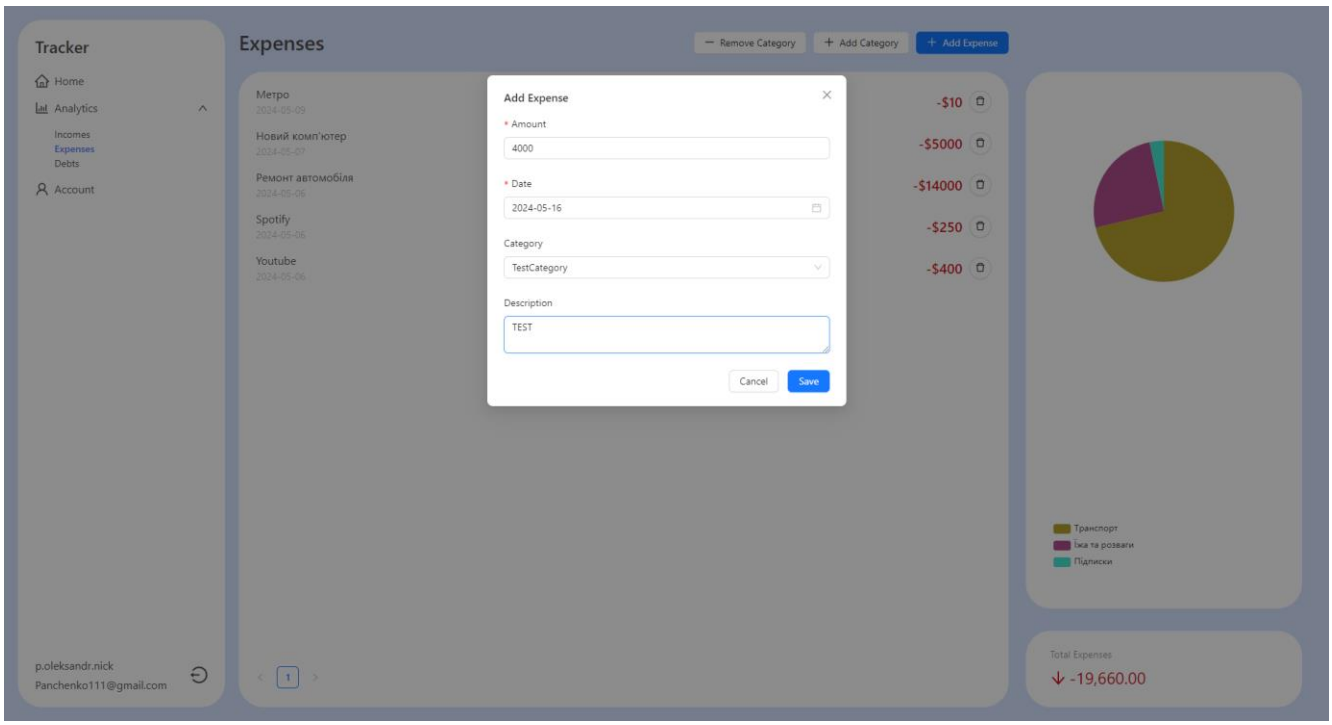


Рис. 3.22 Додавання витрати

Рис. 3.23 Сторінка боргів

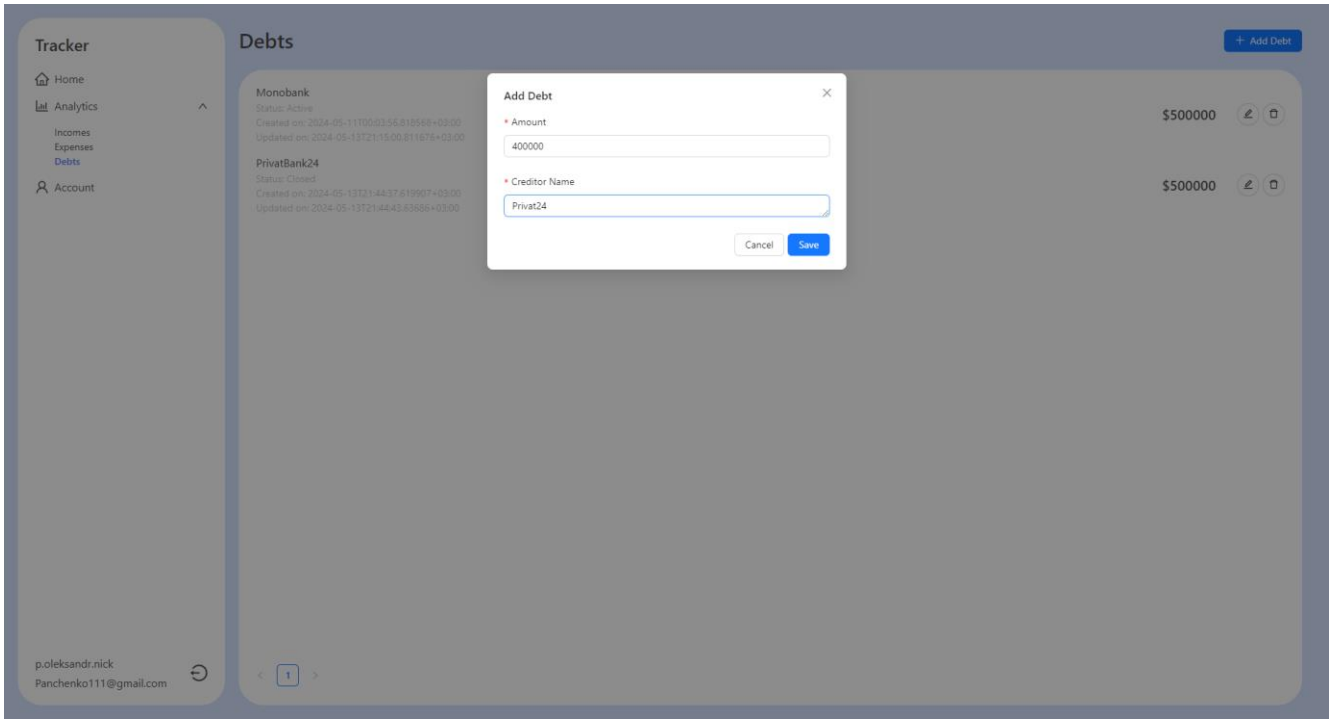


Рис. 3.24 Додавання боргу

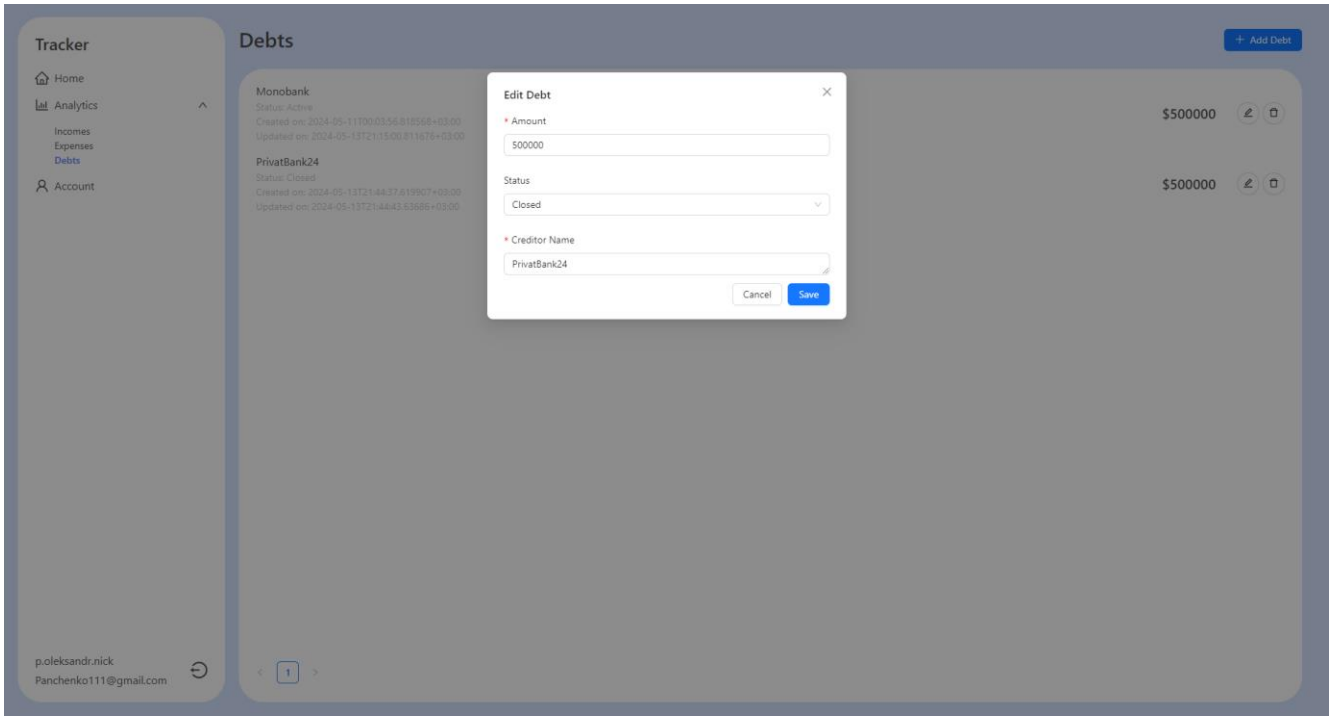


Рис. 3.25 Редагування боргу

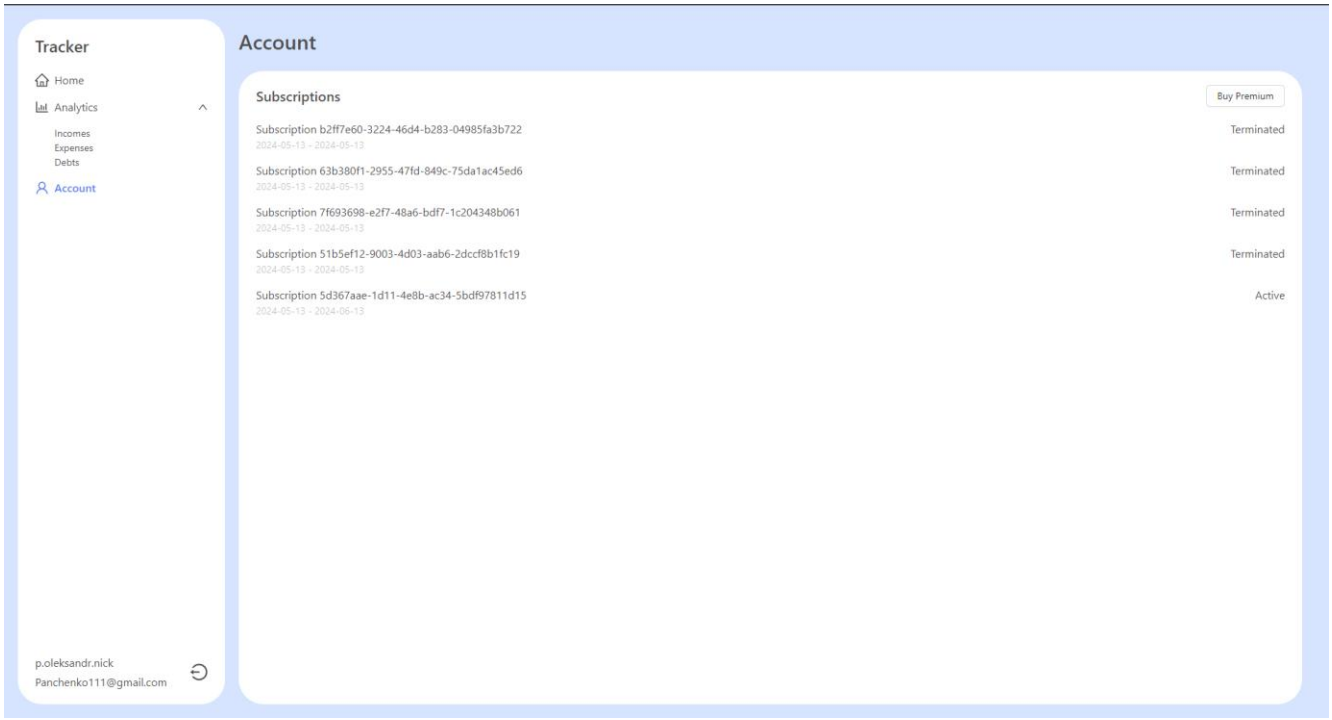


Рис. 3.26 Сторінка облікового запису

На цій сторінці користувач має можливість придбати преміум статус. Застосунок має інтеграцію з платіжною системою PayPal. Нижче на рисунку показано процес оплати.

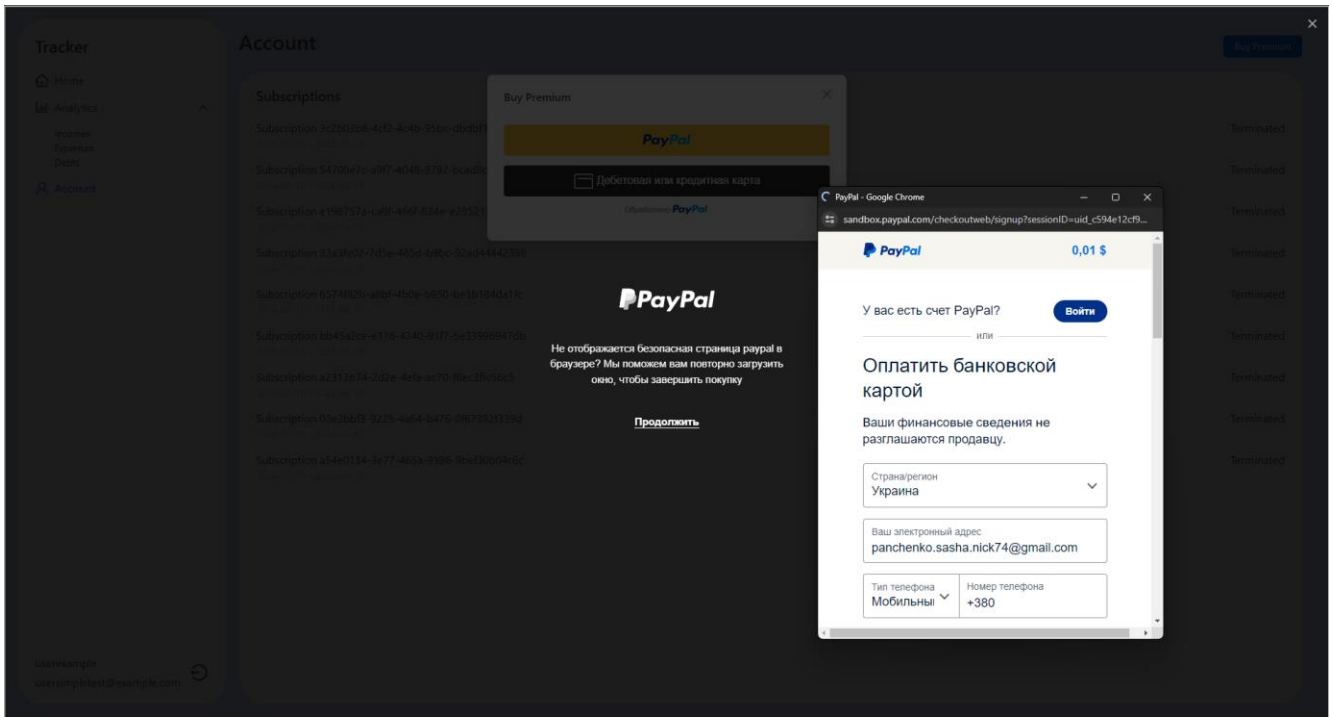


Рис. 3.27 Купівля преміум статусу

3.4 Моделювання функціональності

Діаграма прецедентів відображає взаємодію між актором та системою. Іншими словами це граф, який складається з множини акторів, прецедентів та відношень між акторами. Суть цієї діаграми в тому, що система має вигляд множин сутностей чи акторів, які взаємодіють з системою через варіанти використання, тобто вона вказує, як актор може взаємодіяти з різними функціями застосунку.

Актори поділяються на первинних та вторинних.

Первинні актори, є ініціаторами використання і є незалежними. Вторинні актори використовуються системою, але вони не є ініціаторами, тобто вони не взаємодіють із системою самостійно.

Я розробив дві діаграми. Перша діаграма використання для користувача, друга для адміністратора.

3.4.1 Діаграма прецедентів користувача

Функції, які не ідентифікований користувач може зробити:

- увійти;
- зареєструватися;

Функції користувача на головній сторінці:

- перейти до аналітики;
- перейти до надходжень;
- перейти до боргів;
- перейти до витрат;
- перейти до інформації по обліковому запису;
- вийти з облікового запису;

Функції користувача на сторінці надходжень:

- перегляд надходжень;
- додавання надходжень;
- видалення надходжень;
- перегляд загальної суми всіх надходжень;

Функції користувача на сторінці витрат:

- додавання витрат;
- перегляд витрат;
- перегляд кругової діаграми витрат по категоріям;
- видалення витрат;
- перегляд загальної суми витрат за весь час;

Функції користувача на сторінці боргів:

- додавання боргу;
- видалення боргу;
- редагування боргу;

Функції користувача на сторінці аналітики:

- перегляд детальної статистики;
- перегляд транзакцій;
- отримання звіту за обраним періодом;

Функції користувача на сторінці профілю:

– перегляд всіх транзакцій, які були здійснені для купівлі преміум статусу;
– купівля преміум статусу;

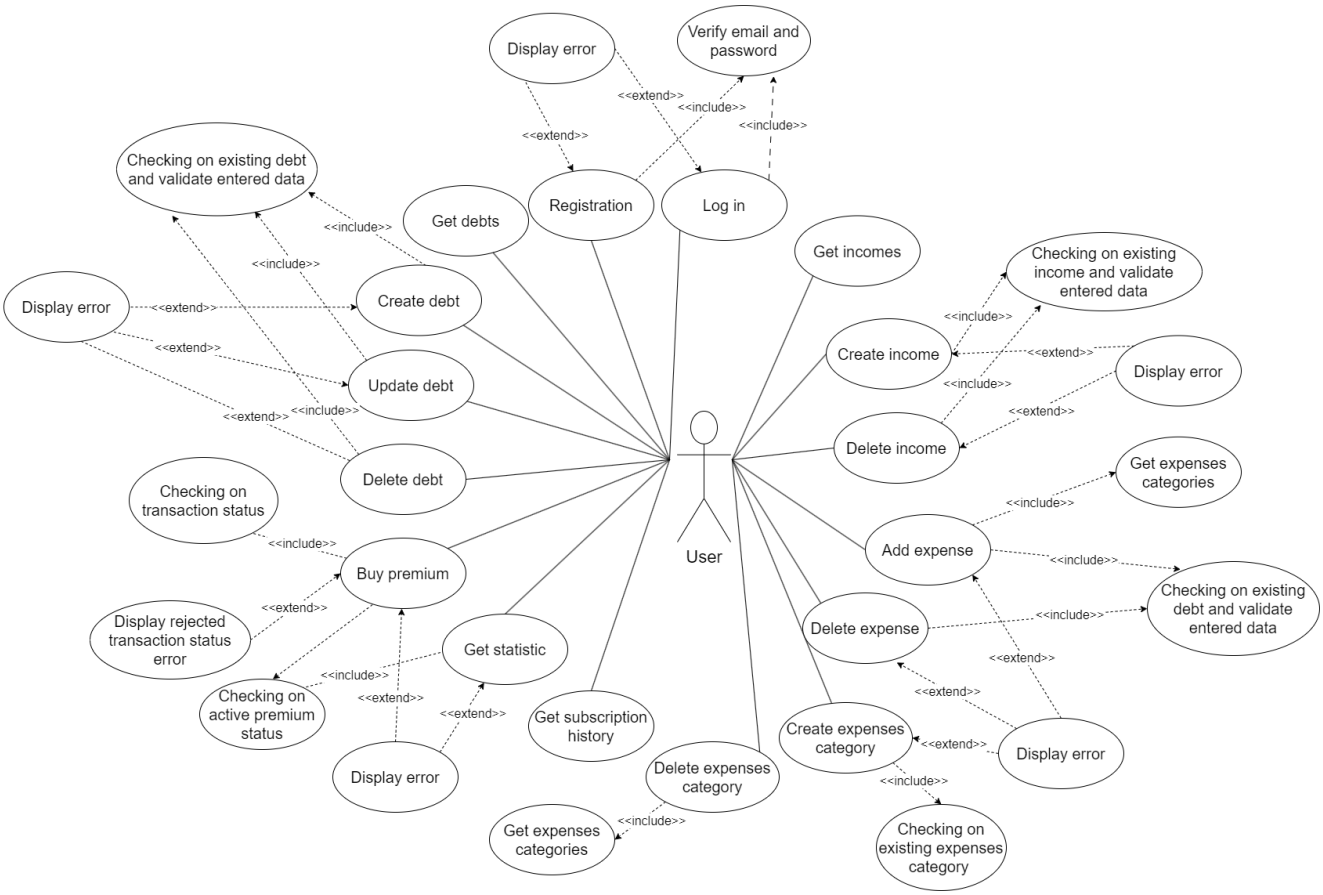


Рис. 3.28 Діаграма прецедентів користувача

3.4.2 Діаграма прецедентів адміністратора

Функції, які не ідентифікований користувач може зробити:

– увійти;

Функції адміністратора на головній сторінці:

– перегляд користувачів;
– пошук користувачів (за ідентифікатором, поштою, номером телефону);
– блокування та розблокування користувача;
– видача преміум статусу для користувача;

– скасування преміум статусу для користувача;
– створення нового адміністратора;
– фільтрування користувачів;
– перегляд історії преміум статусів користувача;
– вихід;

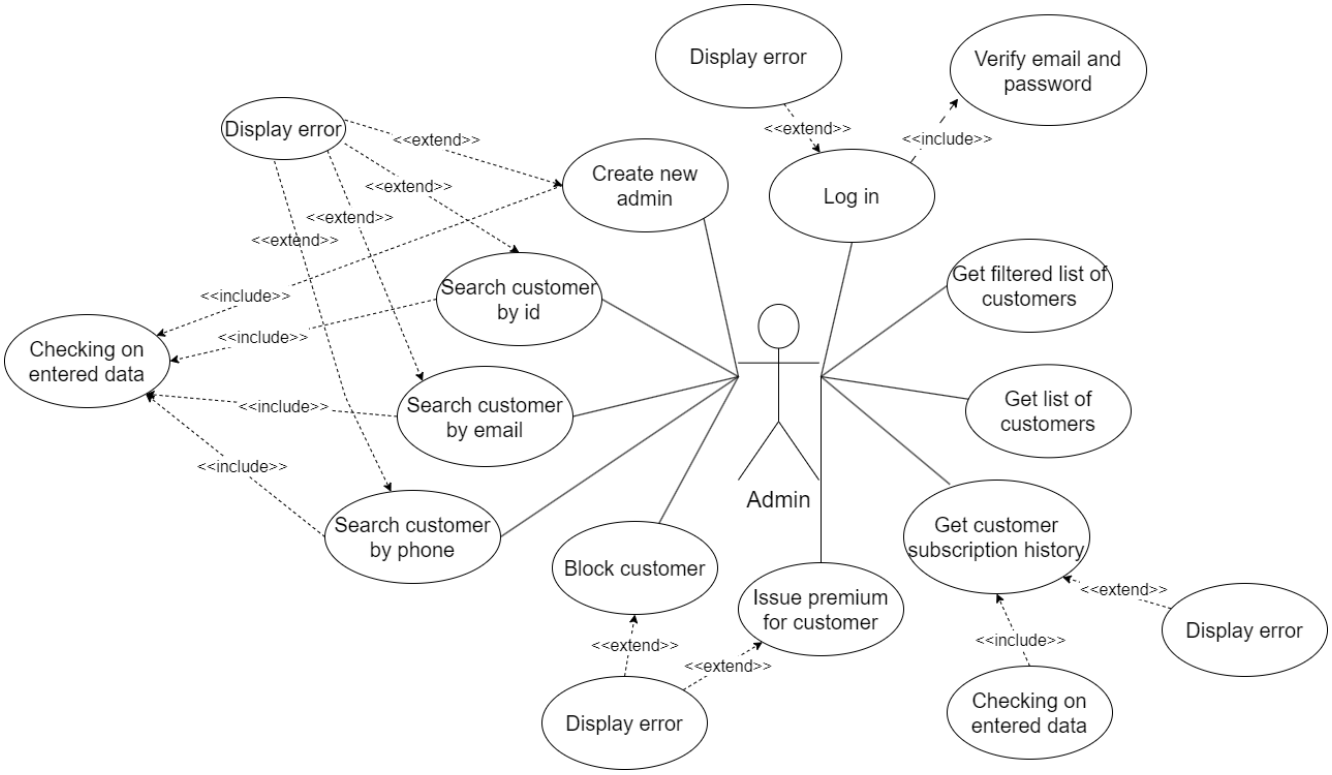


Рис. 3.29 Діаграма використання адміністратора

3.5 Діаграма класів

Діаграма класі демонструє загальну структуру класів програмного забезпечення, а саме класи, атрибути, методи, інтерфейси та взаємовідношення між класами. Діаграма дозволяє розібрати систему на окремі компоненти, щоб наглядно побачити, як вони взаємодіють між собою. Вона широко застосовуються для моделювання, проектування та аналізу програмного забезпечення.

Для прикладу, я підготував діаграму класів в сервісі “FinanceTracking”, яка описує додавання боргу в систему (рис. 3.18).

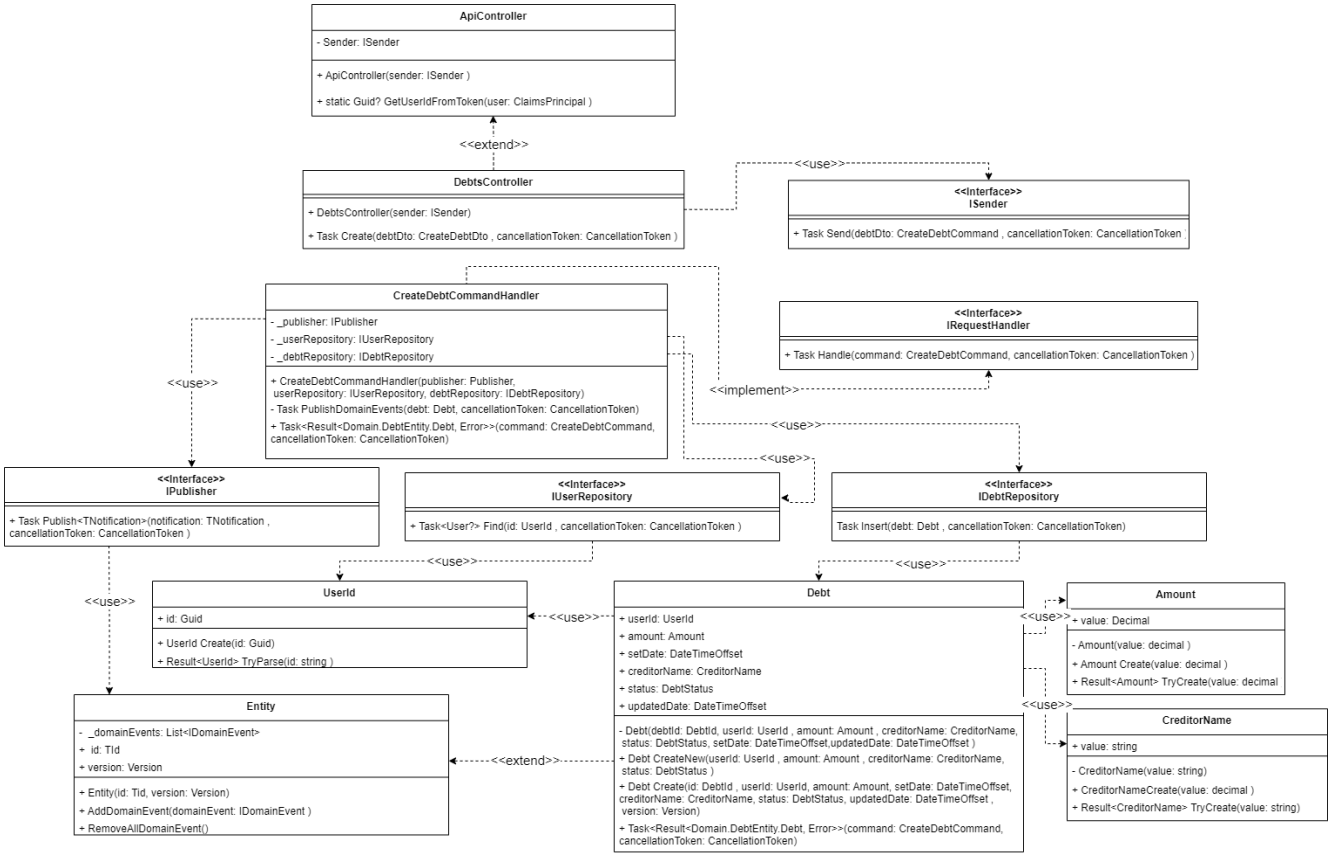


Рис. 3.30 Діаграма класів сервісу “FinanceTracking” додавання боргу в систему.

3.6 Моделювання взаємодії об’єктів

Діаграма послідовностей відображає послідовність взаємодії об’єктів під час конкретного сценарію. На діаграмі зображують об’єкт, які беруть участь у сценарії та повідомлення, які об’єкти передають один одному. Я підготував діаграму, яка відображає послідовність виконання отримання списку доходу, додавання нового доходу та додавання витрат (рис. 3.10).

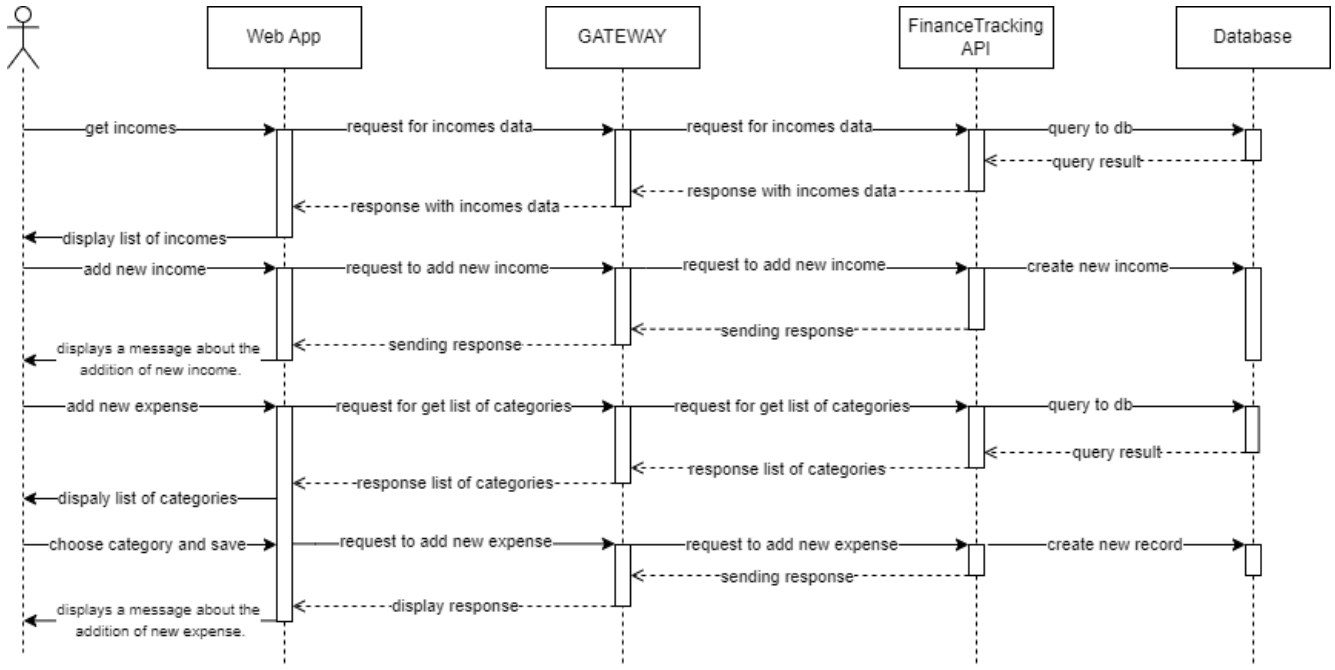


Рис. 3.31 Діаграма послідовності

3.7 Структура баз даних

3.7.1 Даталогічне проектування

Датологічне проектування – це проектування логічної моделі бази даних, яка представляє схему бази даних і зв’язки між об’єктами. Вона визначає як дані будуть зберігатись та керуватись. Мета датологічного проектування полягає у створенні чіткої та ефективної структури даних, яка забезпечить оптимальну продуктивність та масштабованість.

На рисунку нижче показано даталогічну модель бази даних “FinanceTracking” .

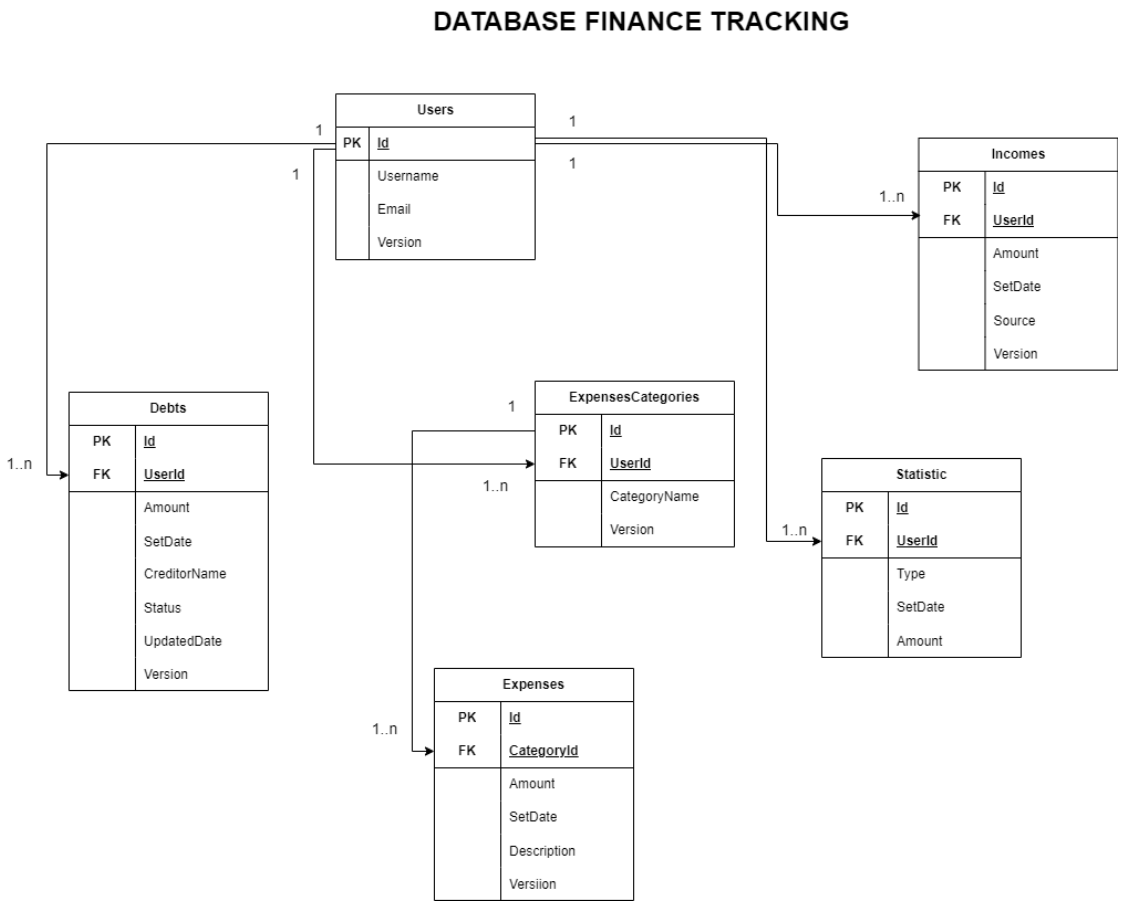


Рис. 3.32 Даталогічна модель бази даних «FinanceTracking»

Пояснення до кожної таблиці:

Таблиця Users:

– Id (тип guid) – унікальне значення, первинний ключ.
– Username (тип text) – містить ім’я клієнта, обмежене 65535 байтами.
– Email (тип text) – містить електронну пошту клієнта, унікальне значення, обмежене 65535 байтами.
– Version (тип int8) – містить версію запису.

Таблиця ExpensesCategories:

– Id (тип guid) – унікальне значення, первинний ключ.
– UserId (тип guid) – зовнішній ключ до таблиці «Users», тип зв’язку один до багатьох.
– CategoryName (тип text) – містить назву категорії, обмежене 65535 байтами.
– Version (тип Int8) – містить версію запису.

Таблиця Expenses:

– Id (тип guid) –унікальне значення, первинний ключ.
– CategoryId (тип guid) – зовнішній ключ до таблиці «ExpenseCategories», тип зв’язку однин до багатьох, увімкнене каскадне видалення.
– Amount (тип numeric) – містить кількість грошей, обмежене до 131072 цифр перед комою та до 16383 цифр після коми.
– SetDate(тип timestamptz) – містить дату створення запису.
– Description (тип text) – містить опис, обмежене 65535 байтами.
– Version (тип Int8) – містить версію запису.

Таблиця Incomes:

– Id (тип guid) – унікальне значення, первинний ключ.
– UserId (тип guid) – зовнішній ключ до таблиці «Users», тип зв’язку один до багатьох.
– Amount (тип numeric) – містить кількість грошей, обмежене до 131072 цифр перед комою та до 16383 цифр після коми.
– SetDate(тип timestamptz) – містить дату створення запису.
– Source (тип text) – містить опис, обмежене 65535 байтами.
– Version (тип Int8) – містить версію запису.

Таблиця Debts:

– Id (тип guid) – унікальне значення, первинний ключ.
– UserId (тип guid) – зовнішній ключ до таблиці «Users», тип зв’язку один до багатьох.
– Amount (тип numeric) – містить кількість грошей, обмежене до 131072 цифр перед комою та до 16383 цифр після коми
– SetDate(тип timestamptz) – містить дату створення запису.
– CreditorName (тип text) – містить опис, обмежене 65535 байтами.
– Status (тип text) – містить опис, обмежене 65535 байтами.
– UpdatedDate(тип timestamptz) – містить дату оновлення запису.
– Version (тип Int8) – містить версію запису.

Таблиця Statistic:

– UserId (тип guid) – зовнішній ключ до таблиці «Users», тип зв’язку один до багатьох.
– Id (тип guid) –унікальне значення, первинний ключ.
– Amount (тип numeric) – містить кількість грошей, обмежене до 131072 цифр перед комою та до 16383 цифр після коми
– SetDate(тип timestamptz) – містить дату створення запису.
– Type (тип text) – містить тип , обмежене 65535 байтами.

На рисунку нижче показано даталогічну модель бази даних “CustomerAuthentication”.

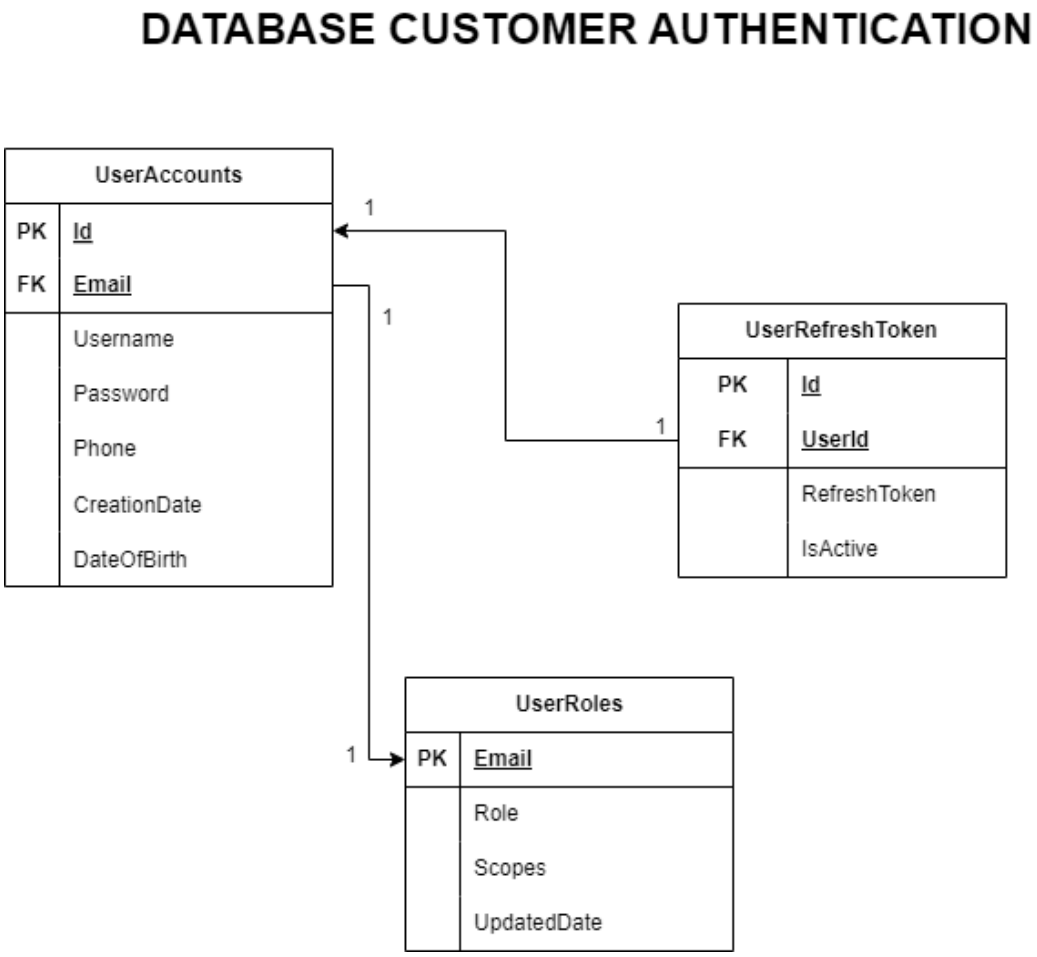


Рис. 3.33 Даталогічна модель бази даних «CustomerAuthentication»

Пояснення до кожної таблиці:

Таблиця UserAccounts:

- Id (тип guid) – унікальне значення, первинний ключ.
- Email (тип text) – містить електронну пошту клієнта, унікальне значення, обмежене 65535 байтами, зовнішній ключ до таблиці “UserRoles” тип зв’язку один до одного.
- Username (тип text) – містить ім’я клієнта, обмежене 65535 байтами.
- Password (тип text) – містить захешований пароль клієнта, обмежене 65535 байтами.
- Phone (тип text) – містить телефон клієнта, унікальне значення, обмежене 65535 байтами.
- CreationDate(тип timestamptz) – містить дату створення запису.
- DateOfBirth(тип timestamptz) – містить дату народження клієнту.

Таблиця UserRefreshTokens:

- Id (тип guid) –унікальне значення, первинний ключ.
- UserId (тип guid) – зовнішній ключ до таблиці «Users», тип зв’язку один до одного.
- RefreshToken (тип text) – містить токен, обмежене 65535 байтами.
- IsActive (тип boolean) – містить перемикач активності токену.

Таблиця UserRoles:

- Email (тип guid) –унікальне значення, первинний ключ.
- Role (тип text) – містить роль клієнту, обмежене 65535 байтами.
- Scopes (тип text) – містить права доступу користувача, обмежене 65535 байтами.
- UpdatedDate (тип timestamptz) – містить дату оновлення запису.

На рисунку нижче показано даталогічну модель бази даних “Finance Tracking Subscription”.

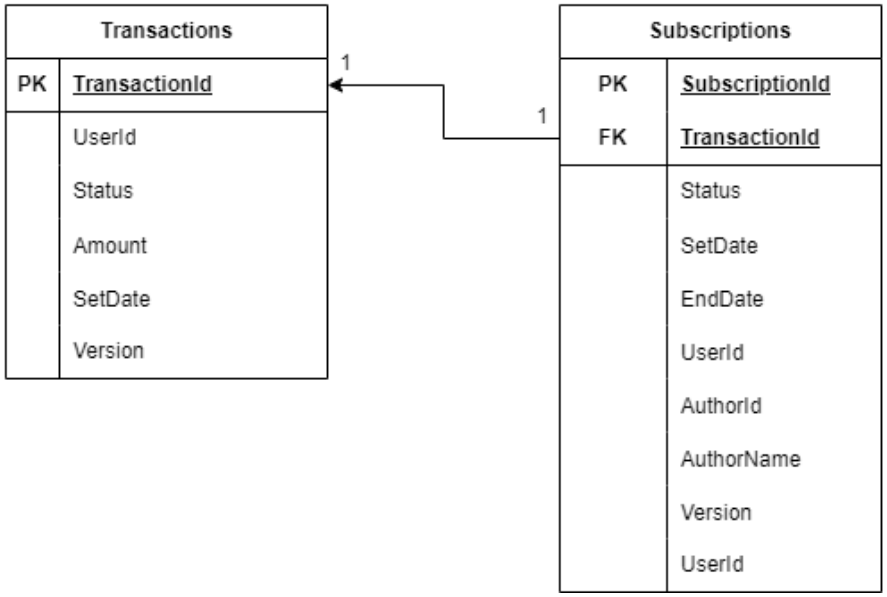


Рис. 3.34 Даталогічна модель бази даних «FinanceTrackingSubscription»

Таблиця Subscriptions:

– SubscriptionId (тип guid) – унікальне значення, первинний ключ.
– TransactionId (тип text) - зовнішній ключ до таблиці “Transactions”, тип зв’язку один до одного.
– Status (тип text) – містить статус активації, обмежене 65535 байтами.
– AuthorName (тип text) – містить ім’я адміністратора, обмежене 65535 байтами.
– UserId (тип guid) – містить ідентифікатор користувача.
– AuthorId (тип guid) – містить ідентифікатор адміністратора, який змінив цей запис.
– Version (тип int8) – містить версію запису.
– SetDate (тип timestamptz) – містить дату створення підписки.
– EndDate (тип timestamptz) – містить дату закінчення підписки.

Таблиця Transactions:

– TransactionId (тип text) - унікальний ключ.
– Status (тип text) – містить статус активації, обмежене 65535 байтами.

- UserId (тип guid) – містить ідентифікатор користувача.
- Amount (тип numeric) – містить кількість грошей, обмежене до 131072 цифр перед комою та до 16383 цифр після коми.
- Version (тип int8) – містить версію запису.
- SetDate (тип timestamptz) – містить дату створення підписки.

3.7.2 Нормалізація баз даних

При проектуванні баз даних до мікросервісів, було дотримано всіх пунктів нормалізації баз даних, а саме:

1. Кожна таблиця має первинний ключ та мінімальний набір колонок, які визначають запис.
2. Правильно визначені ключові атрибути для уникнення повторювань.
3. Дотримано атомарність значень.
4. Дані, що могли повторюватись в різних рядках були винесені в окремі таблиці.
5. Поле, що залежало від первинного ключа та будь-якого іншого, виносились в окрему таблицю.

3.8 Тестування застосунку

Тестування програмного забезпечення необхідний процес перевірки надійності і вимог до системи. Цей процес спрямований на виявлення помилок, дефектів та інших проблем, які можуть виникнути в продакшені, це забезпечує якість та стабільність програмного продукту.

Під час розробки сервісу було проведено такі етапи тестування:

- функціональне тестування;
- регресійне тестування системи або окремого мікросервіса після внесення будь-яких змін;

– інтеграційне тестування, для перевірки інтеграцій між мікросервісами;
– написання Unit тестів, для важливої бізнес логіки;
– E2E(End-To-End) тестування;
– тестування продуктивності застосунку за допомогою Lighthouse.

3.8.1 Функціональне тестування застосунку

Функціональне тестування застосунку – це процес тестування, який порівнює фактичну поведінку програми на відповідність функціональних вимог. Під час підготовки до тестування прописуються тести кейси системи. Нижче наведено приклад тест кейсів, які були використані для функціонального тестування застосунку.

Таблиця 3.1

Тест кейси

№	Назва	Передумови	Кроки	Очікуваний результат
TC-01	Реєстрація користувача.	Користувач не має існуючого облікового запису.	1.Відкрити сторінку реєстрації. 2.Ввести валідні дані у необхідні поля. 3.Натиснути кнопку "Sign Up".	Користувач успішно зареєстрований і перенаправлений на головну сторінку веб-застосунку.
TC-02	Вхід користувача у систему.	Користувач має існуючий обліковий запис.	1.Відкрити сторінку входу. 2.Ввести валідні email та пароль. 3.Натиснути кнопку "Sign in".	Користувач успішно входить у систему і перенаправлений на головну сторінку.

Продовження таблиці 3.1

Тест кейси

№	Назва	Передумови	Кроки	Очікуваний результат
ТС-03	Відображення помилки при реєстрації.	Користувач має існуючий обліковий запис.	1.Відкрити сторінку реєстрації 2.Ввести валідні дані у необхідні поля. 3.Натиснути кнопку "Sign Up".	Користувачу відобразиться помилка про те, що користувач з такою поштою вже існує.
ТС-04	Відображення помилки при авторизації.	Користувач не має існуючого облікового запису.	1.Відкрити сторінку входу. 2.Ввести електронну пошту та пароль. 3.Натиснути кнопку "Sign in".	Користувачу відобразиться помилка про те, що такого облікового запису не знайдено.
ТС-05	Відображення помилки при введені невалдних даних.	Користувач не має існуючого облікового запису.	1.Відкрити сторінку реєстрації. 2.Ввести некоректну електронну пошту . 3.Натиснути кнопки “Sign Up”.	Користувачу відобразиться помилка про те, що електронна пошта має не правильний формат.

Продовження таблиці 3.1

Тест кейси

№	Назва	Передумови	Кроки	Очікуваний результат
TC-06	Успішне додавання прибутку у систему	Користувач має існуючий обліковий запис та авторизований в систему	1.Відкрити сторінку доходів. 2.Натиснути на кнопку “Add Income”. 3.У відкритому вікні ввести валідні дані. 4.Натиснути кнопку “Save”	Прибуток успішно доданий до системи та відображається в списку транзакцій.
TC-07	Успішне додавання боргу у систему	Користувач має існуючий обліковий запис та авторизований в систему.	1.Відкрити сторінку боргів 2.Натиснути кнопку “Add Debt” 3.У відкритому вікні ввести валідні дані. 4. Натиснути кнопку “Save”	Борг успішно доданий до системи та відображається в списку боргів.
TC-08	Успішне редагування боргу	Користувач має існуючий обліковий запис та авторизований в систему.	1.Відкрити сторінку боргів 2.Натиснути кнопку редагування біля потрібного запису 3.У відкритому вікні оновити дані. 4. Натиснути кнопку “Save”	Борг успішно оновлений та відображається в списку боргів.

3.8.2 Регресійне тестування

В проекті під час додання нового функціоналу або виправлення дефектів проводилося регресійне тестування. А саме вибіркове тестування на основі аналізу змін у програмі для того, щоб переконатись чи не зламали внесені зміни інший функціонал або не породило нові дефекти у виправленому функціоналі.

3.8.3 Інтеграційне тестування

Інтеграційне тестування – це вид тестування, під час якого перевіряють інтеграції різних модулів системи на відповідність поставленим вимогам.

В проекті інтеграційне тестування проводилося для перевірки інтеграції мікросервіса, який відповідає за облік користувачів з сервісом обліку преміальних облікових записів. Під час купівлі преміум статусу сервіс обліку преміальних облікових записів повинен зберегти дані та опублікувати їх в Kafka. Сервіс, який відповідає за облік користувачів, читає топік та оновляє статус облікового запису користувача.

3.8.4 Unit тестування

Unit тестування – це автоматичне тестування, яке перевіряє окремі частки коду котрі виконують важливу бізнес логіку. В роботі було написано юніт тести для логіки авторизації та реєстрації користувача, створення боргів, доходів, витрат. Результати тестування показано на рис. 3.33 та рис 3.34:

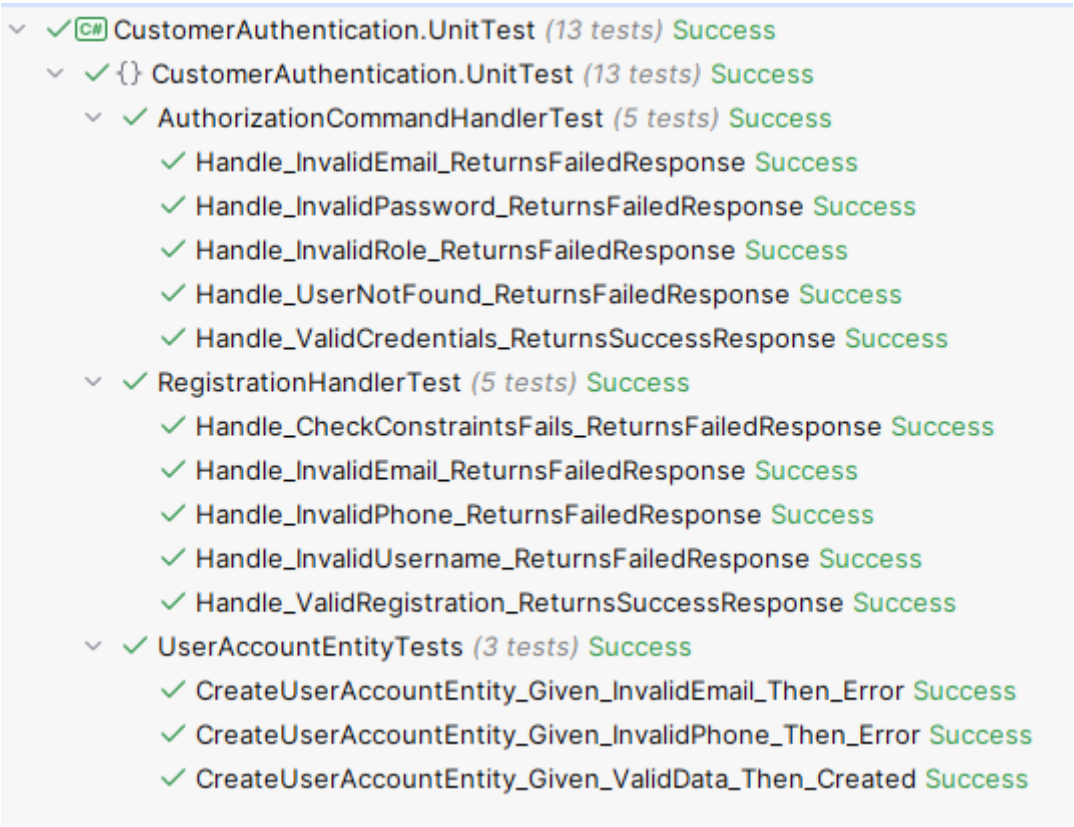


Рис. 3.33 Результати тестування реєстрації та авторизації користувача

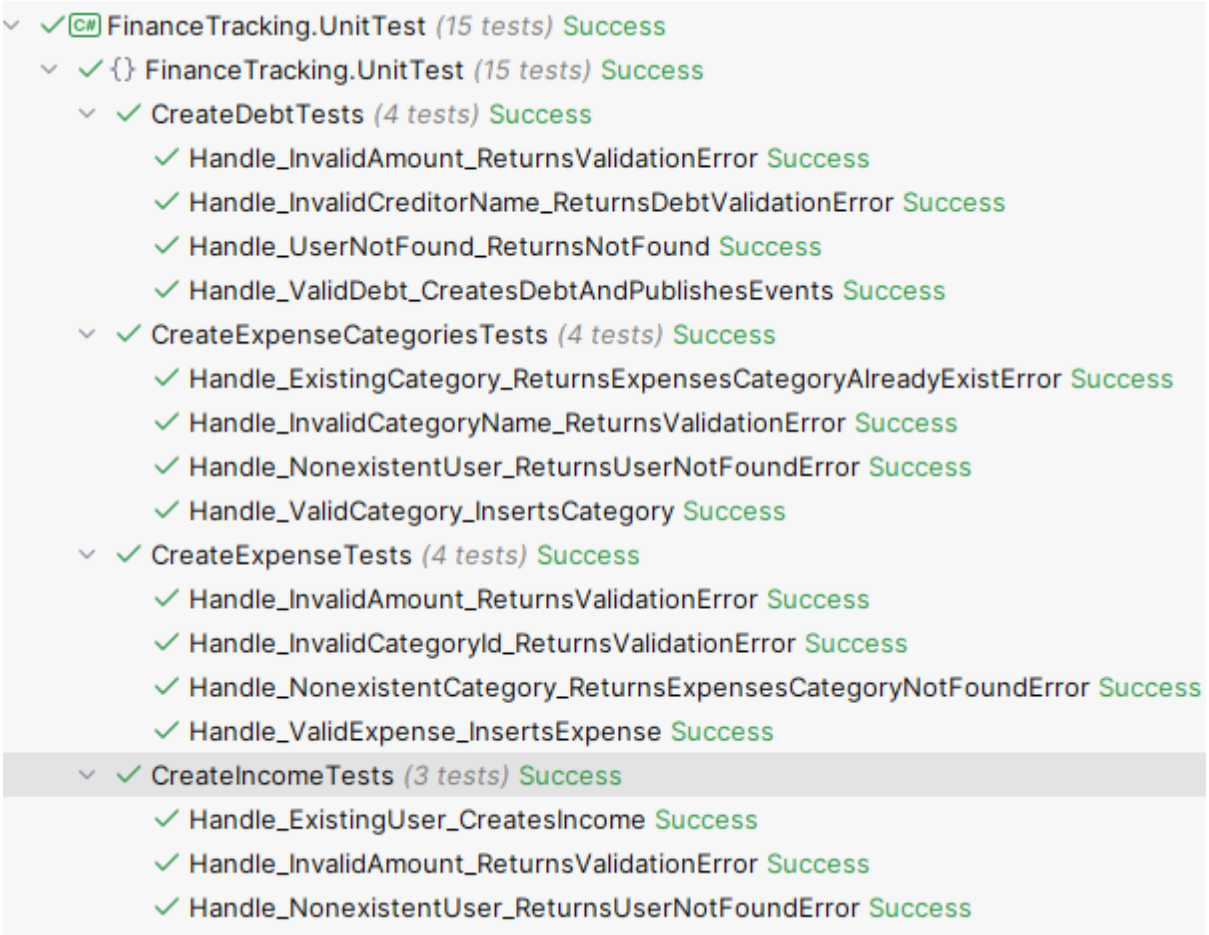


Рис. 3.34 Тестування основної функціональсоті системи керування фінансами.

3.8.5 Тестування продуктивності веб-застосунку в Lighthouse

Lighthouse – це інструмент для оцінки веб-застосунків. Він проводить аналіз продуктивності, доступності та надає оцінку дотриманням порадам Google по пошуковій оптимізації. Нижче показано результат тестування в Lighthouse.

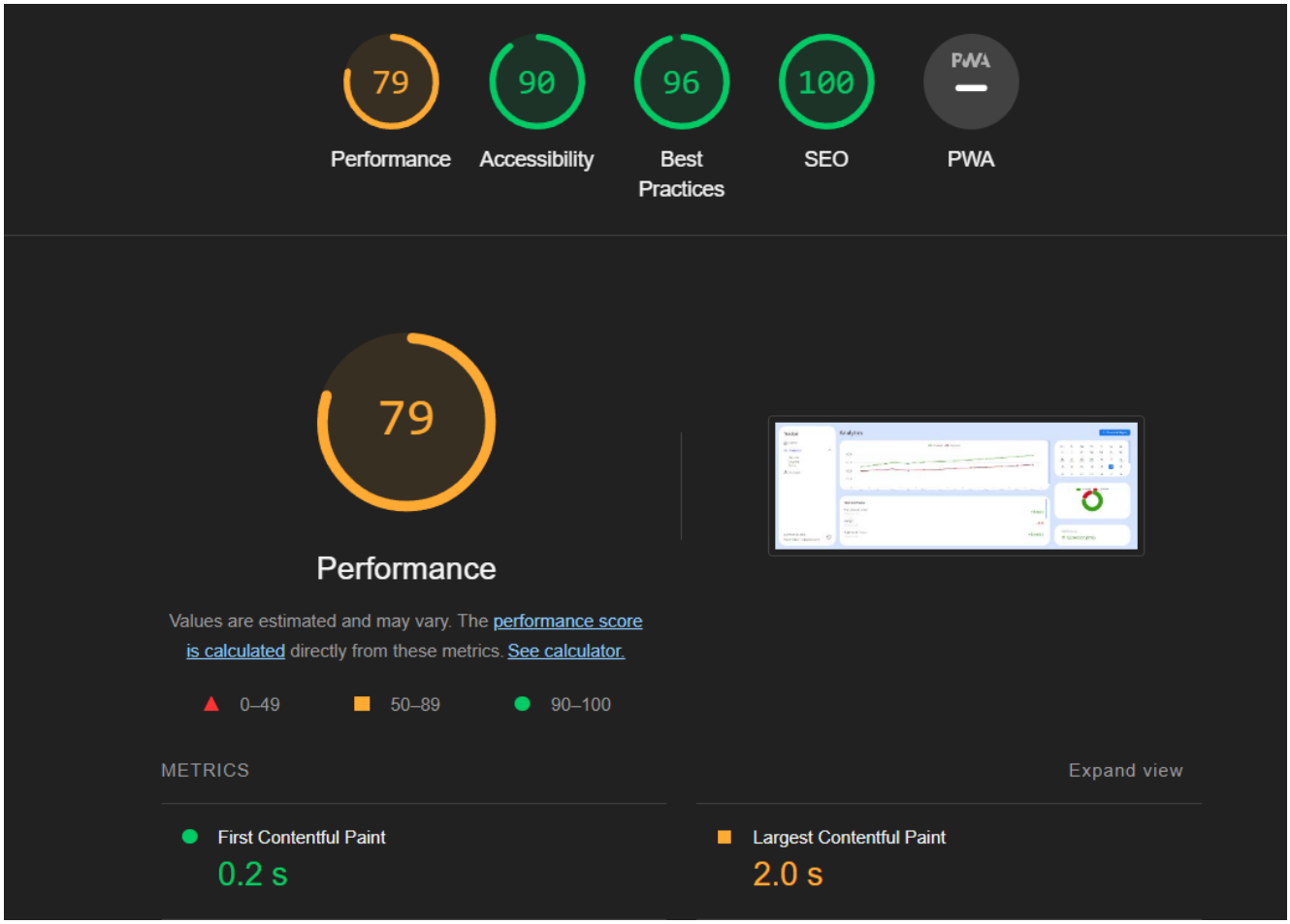


Рис. 3.35 Тестування продуктивності застосунку у Lighthouse

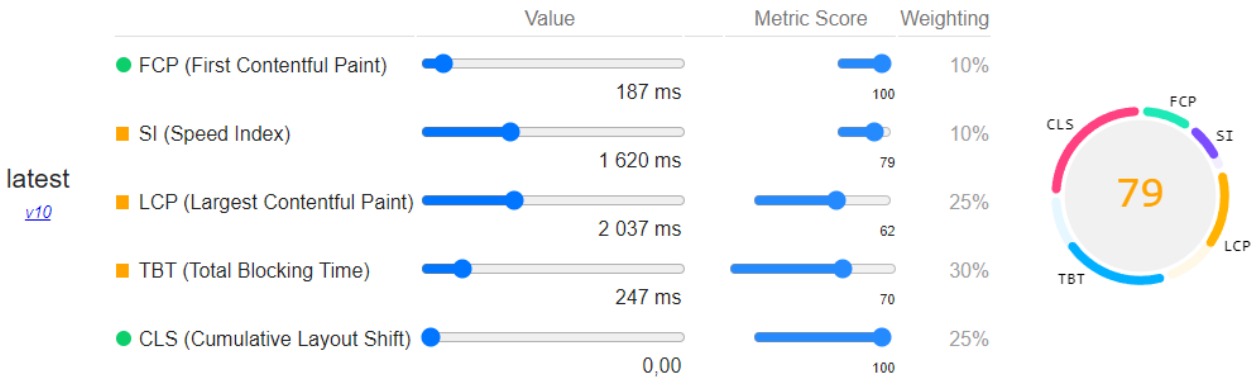


Рис. 3.36 Детальна інформація про тестування продуктивності

ВИСНОВКИ

У результаті розробленої системи для керування фінансами було виконано всі поставлені завдання, а саме:

1. Проведено аналіз обраної предметної області в сфері менеджменту фінансів.
2. Проаналізовано існуючі сервіси. Визначено ключові функції, їх переваги та недоліки.
3. Визначено основні функціональні та нефункціональні вимоги системи.
4. Спроектовано та розроблено веб-застосунок з використанням архітектурного стилю мікросервісів та підходом Domain-Driven Design, який підтримує масштабованість за допомогою використання брокеру повідомлень Apache Kafka та API-шлюзу Ocelot. Перед розробкою було виконано наступні етапи:
   - змодельовано модель прецедентів. Діаграма прецедентів;
   - спроектовані зміст та взаємовідношення між класами. Діаграма класів;
   - спроектовано взаємодію об'єктів. Діаграма послідовності;
   - спроектовано базу даних. Схема бази даних;

   Реалізований функціонал, який відповідає визначеним вимогам.
5. Проведено регресійне, інтеграційне, E2E тестування застосунку з метою виявлення та усунення проблем системи. Під час тестування було перевірено виконання всіх вимог.

Робота пройшла апробацію на наступних наукових конференціях:

1. Панченко О.М., Ільїн О.Ю. Використання Apache Kafka для швидкого та надійного методу спілкування між мікросервісами у веб-застосунку для керування фінансами. Всеукраїнська науково-технічна конференція «Сучасні інтелектуальні інформаційні технології в науці та світі». 14.05.2024, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ. С. 69–72.

2. Панченко О.М А.Д., Ільїн О.Ю. Впровадження Ocelot Gateway у веб-застосунок керування фінансами. Всеукраїнська науково-технічна конференція «Новітні апаратні та програмні засоби інтелектуальних технологій». 20.05.2024, Київ, Державний університет інформаційно-комунікаційних технологій. Збірник тез. К.: ДУІКТ. С. 247–249.

ПЕРЕЛІК ПОСИЛАНЬ

1. Microsoft .Net documentation. (2024). URL: https://learn.microsoft.com/en-us/dotnet/
2. Ocelot Gateway documentation. (2024). URL: https://ocelot.readthedocs.io/en/latest/introduction/gettingstarted.html
3. Apache Kafka documentation. (2024) URL: https://kafka.apache.org/documentation/
4. PostgreSQL documentation. (2024). URL: https://www.postgresql.org/docs/
5. Docker documentation. (2024). URL: https://docs.docker.com/manuals/
6. Kolade Chris. (2021) REST API best practice. URL: https://www.freecodecamp.org/news/rest-api-best-practices-rest-endpoint-design-examples/
7. React documentation. (2024). URI: https://legacy.reactjs.org/docs/hello-world.html
8. Chaojie Xiaoю (2022). Domain-Driven Design best practice – Domain Event. URL: https://medium.com/@chaojie.xiao/domain-driven-design-practice-domain-events-15b38f3c58fc
9. JavaScript documentation. (2024) URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript
10. McClure, R., & Shah, Q. (2020). Mastering PostgreSQL 12: Advanced Techniques to Build and Administer Scalable and Reliable PostgreSQL Database Solutions. Packt Publishing.
11. Petkovic, L. (2021). PostgreSQL: Up and Running: A Practical Guide to the Advanced Open Source Database. O'Reilly Media.
12. Philip Japikse. (2022). Pro C# 10 with .NET 6: Foundational Principles and Practices in Programming.
13. Gutierrez, G., & Gomez-Uribe, C. (2021). Kafka: The Definitive Guide: Real-Time Data and Stream Processing at Scale. O'Reilly Media.

14. Sharma, N., & Baghel, P. (2019). Apache Kafka Quick Start Guide: Leverage Apache Kafka 2.0 to Simplify Real-Time Data Processing for Distributed Applications. Packt Publishing.
15. Rauschmayer, A. (2020). JavaScript for Impatient Programmers. Dr. Axel Rauschmayer.
16. Marcotte, E. (2020). Responsive Web Design (2nd ed.). A Book Apart.
17. Nair, P. (2019). Hands-On Microservices with Kubernetes: Build, deploy, and manage scalable microservices on Kubernetes. Packt Publishing.
18. Banks, A., & Porcello, E. (2020). Learning React: Modern Patterns for Developing React Apps. O'Reilly Media.
19. Newman, S. (2021). Building Microservices: Designing Fine-Grained Systems. O'Reilly Media.
20. Richardson, C. (2018). Microservices Patterns: With examples in Java. Manning Publications.
21. Robert Martin. (2017). Clean Architecture: A Craftsman's Guide to Software Structure and Design (Robert C. Martin Series).
22. Vernon, V. (2019). Implementing Domain-Driven Design. Addison-Wesley.
23. Chris Minnick. (2022). Beginning ReactJS Foundations Building User Interfaces with ReactJS: An Approachable Guide.

ДОДАТОК А. ДЕМОНСТРАЦІЙНІ МАТЕРІАЛИ
(Презентація)

ДЕРЖАВНИЙ УНІВЕРСИТЕТ ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ ВЕБ-СЕРВІСУ КЕРУВАННЯ ФІНАНСАМИ ТА ВИТРАТАМИ З ВИКОРИСТАННЯМ C# ТА REACT

Виконав студент 4 курсу
Групи ПД-41
Панченко Олександр Миколайович
Керівник роботи
д.т.н.,проф., професор кафедри ІПЗ *Ільїн Олег Юрійович*

Київ – 2024

МЕТА, ОБ’ЄКТ ТА ПРЕДМЕТ ДОСЛІДЖЕННЯ

- **Мета роботи:** автоматизація процесу управління фінансами та витратами.
- **Об’єкт дослідження:** процес керування фінансами та витратами.
- **Предмет дослідження:** веб-застосунок для керування фінансами та витратами.

2

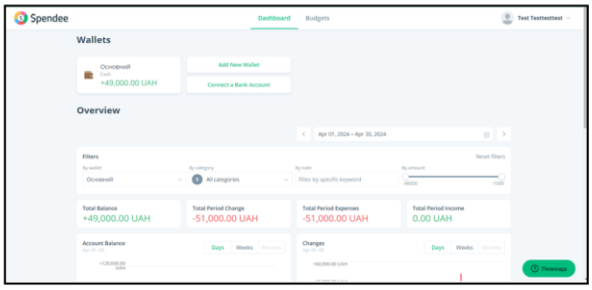
ЗАДАЧІ ДИПЛОМНОЇ РОБОТИ

1. Дослідити предметну область, проаналізувати ринок веб-застосунків фінансового менеджменту. Визначити їх переваги та недоліки.
2. Визначити основні вимоги до застосунку, встановити функціональні та технічні вимоги з урахуванням потреб користувачів та недоліків конкурентів.
3. Підібрати технічні засоби, які будуть використовуватись для розробки програмного забезпечення згідно вимогам.
4. Спроекутвати та розробити архітектуру застосунку з урахуванням встановлених вимог.
5. Протестувати застосунок на відповідність вимогам.

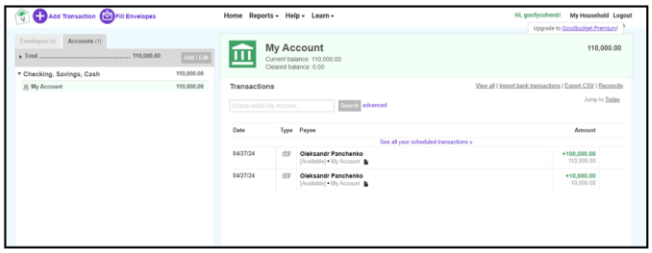
3

АНАЛІЗ АНАЛОГІВ

Finmap



Spendee



Goodbudget

4

АНАЛІЗ АНАЛОГІВ

	Finmap	Spendee	Goodbudget	FinanceTracking
Наявність статистики	+	+	+	+
Цільова аудиторія	Тільки для бізнесу	Для особистого користування	Для особистого користування	Для бізнесу та особистого користування
Можливість купівлі преміум підписки	+	+	+	+
Отримання звітів	+	+	+	+
Відсутня реклама	-	-	-	+
Синхронізація між пристроями	+	+	+	+
Розділення витрат по категоріям	+	+	-	+
Інтеграція з банками	+	+	+	+

5

ВИМОГИ ДО ЗАСТОСУНКУ

Функціональні вимоги:

Для користувачів:
1. Витрати повинні розділятись по категоріям.
2. Користувачі повинні мати можливість додавати в систему витрати, надходження та борги.
3. Всі введені дані мають бути перевірені.
4. Користувачі повинні мати можливість переглядати статистику по витратам та надходженням.
5. Користувачі повинні мати можливість отримання звітів за певний період часу.
6. Користувачі повинні мати можливість купівлі преміум статусу.
7. Користувачі повинні мати можливість переглядати історію купівель преміум статусу.

Для адміністратора:
1. Адміністратор повинен мати можливість блокувати та розблоковувати користувачів.
2. Адміністратор повинен мати можливість власноруч видавати та забирати преміум підписку.
3. Адміністратор повинен мати можливість переглядати список всіх користувачів.
4. Адміністратор повинен мати можливість переглядати історію преміум підписок користувача.
5. Адміністратор повинен мати можливість здійснювати пошук за ідентифікаторами або за фільтром.

Нефункціональні вимоги:
1. Система повинна підтримувати масштабованість, для швидкого додавання нових функцій.
2. Система повинна бути захищена від несанкціонованого доступу.
3. Система повинна набрати показник не менше 60 відсотків продуктивності в LightHouse.

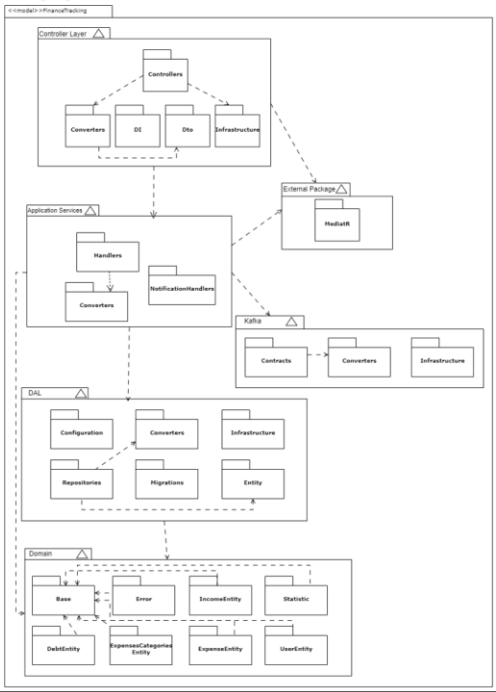
6

ПРОГРАМНІ ЗАСОБИ РЕАЛІЗАЦІЇ

PostgreSQL
Microsoft® .NET
JavaScript
docker
kafka
OCELOT
RD
Entity Framework Core
DBeaver

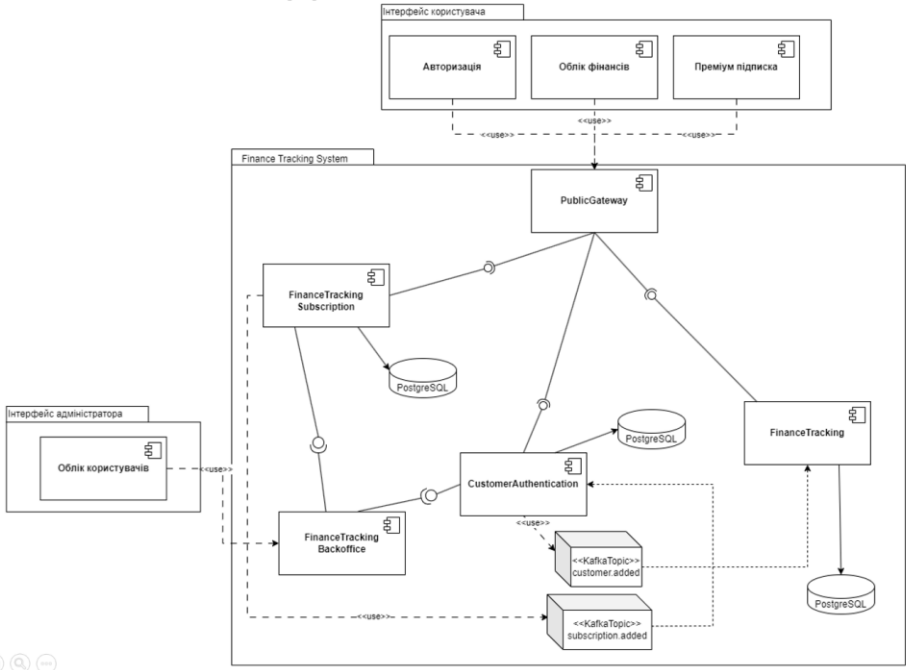
7

ДІАГРАМА ПАКЕТІВ



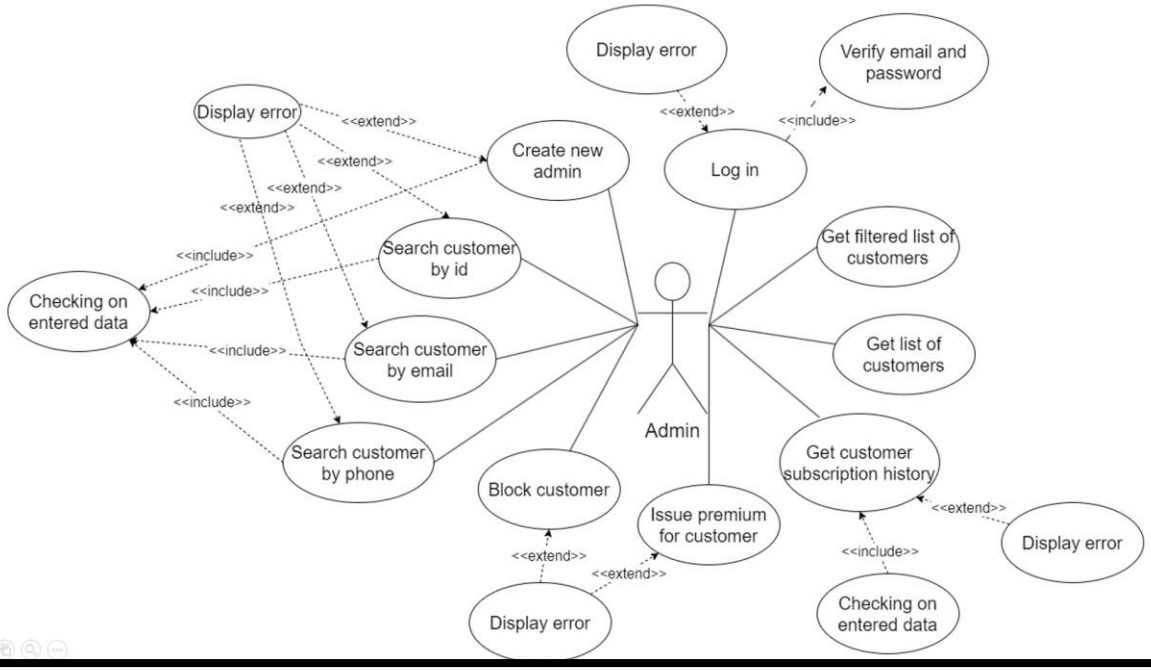
8

ДІАГРАМА КОМПОНЕНТІВ



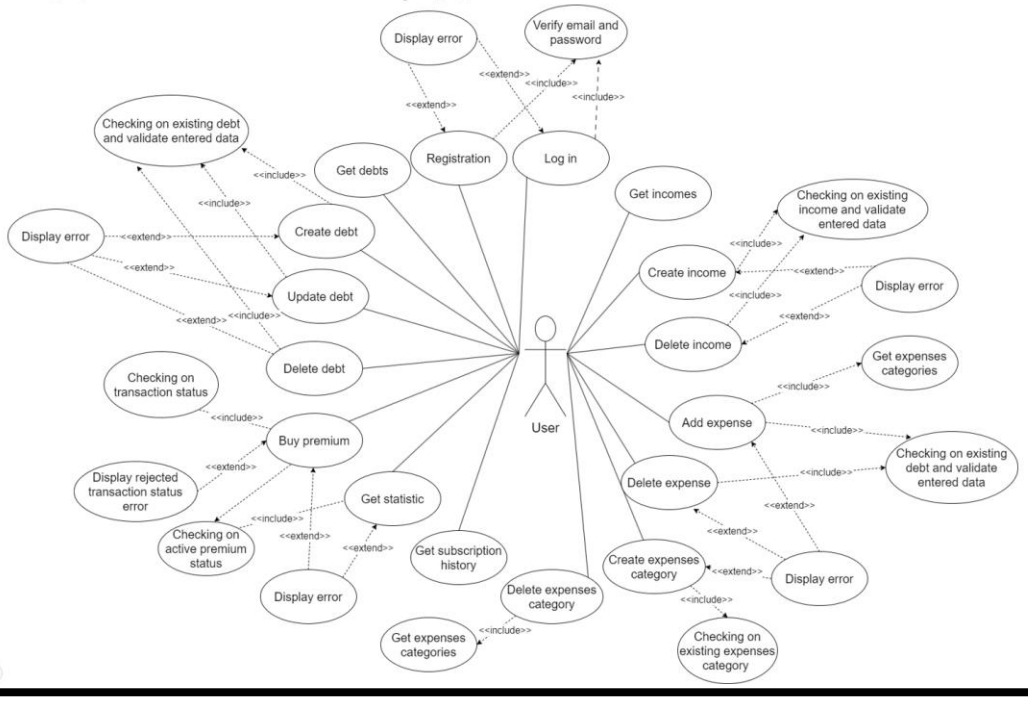
9

ДІАГРАМА ПРЕЦЕДЕНТІВ АДМІНІСТРАТОРА



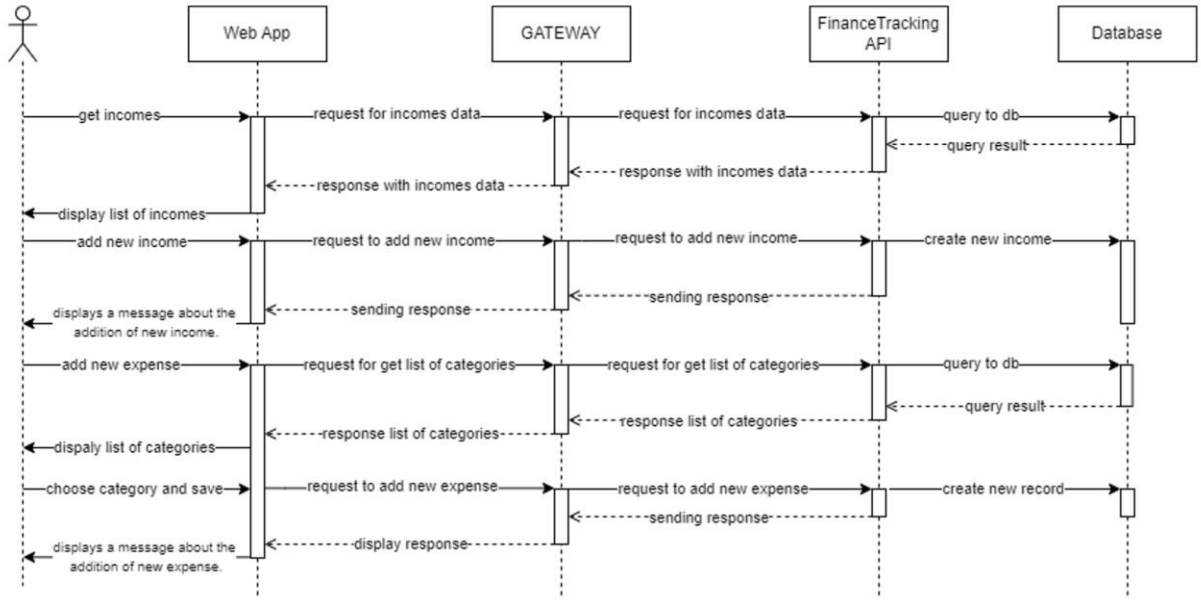
10

ДІАГРАМА ПРЕЦЕДЕНТІВ КОРИСТУВАЧА



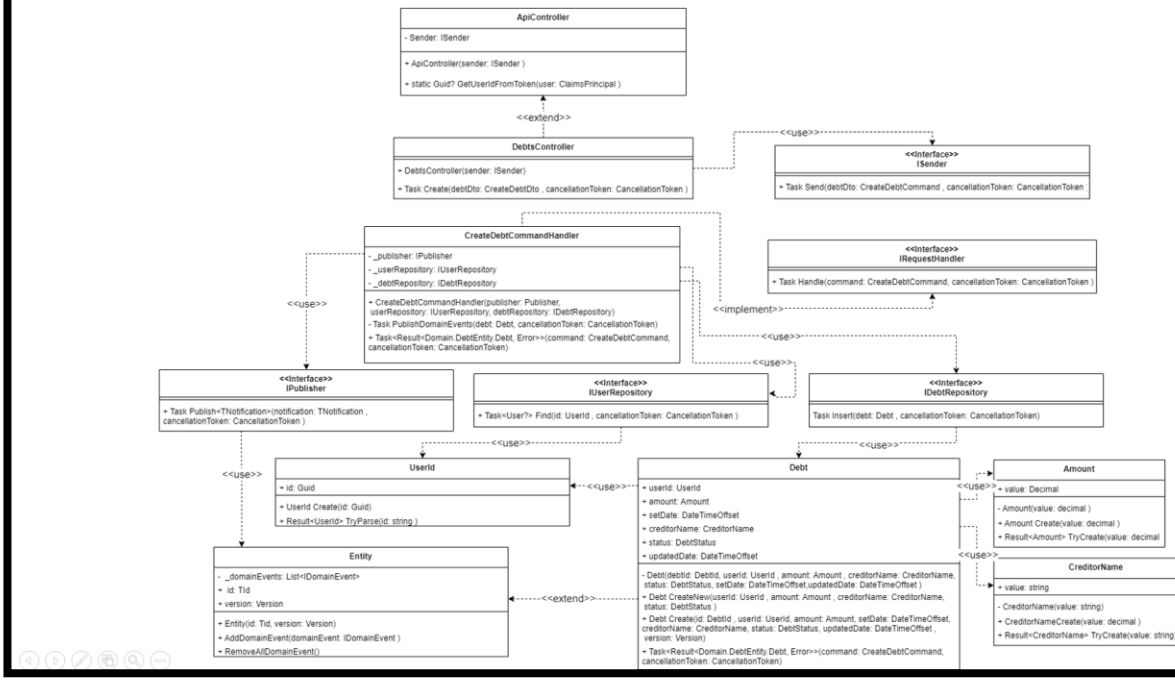
11

ДІАГРАМА ПОСЛІДОВНОСТІ

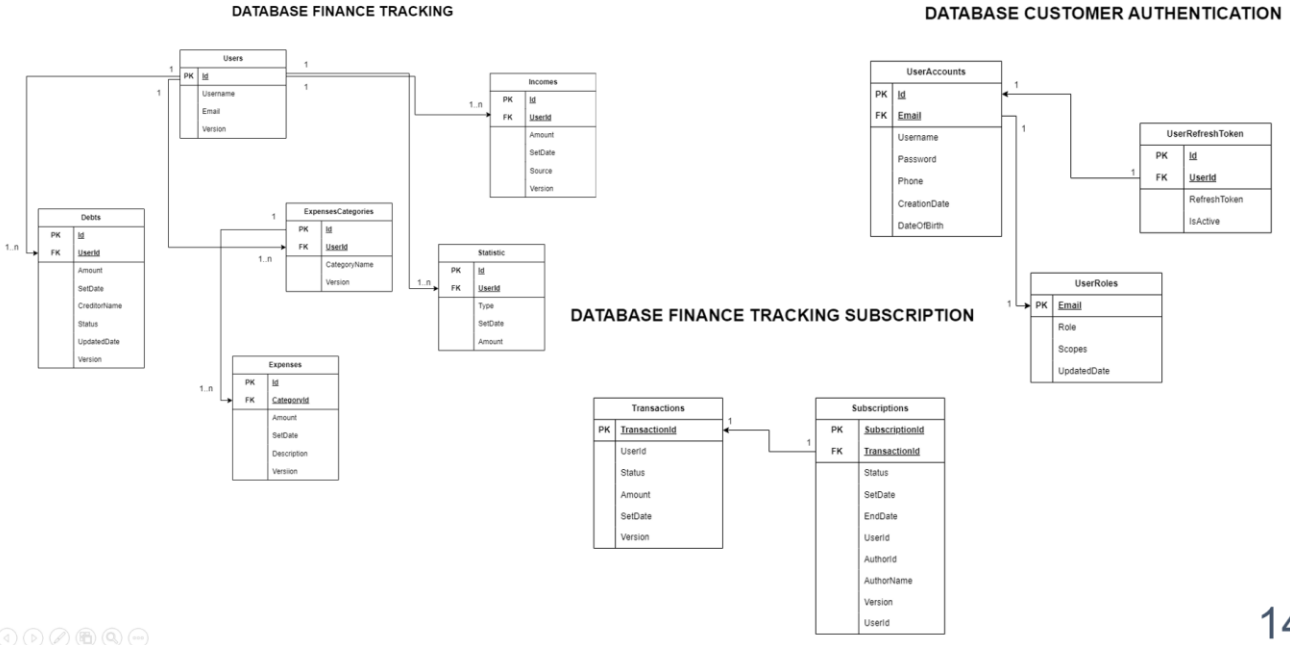


12

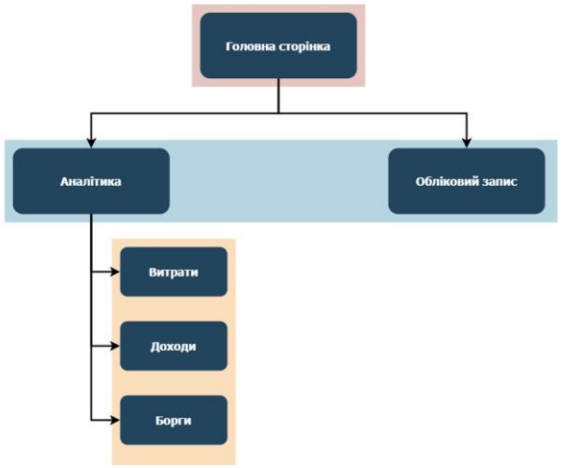
ДІАГРАМА КЛАСІВ ДОДАВАННЯ БОРГУ



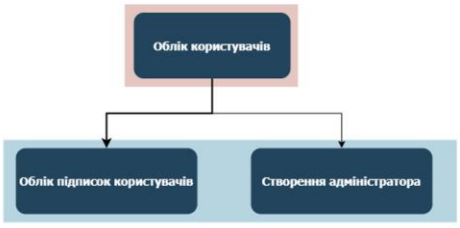
ДАТАЛОГІЧНІ МОДЕЛІ



МАПА САЙТУ



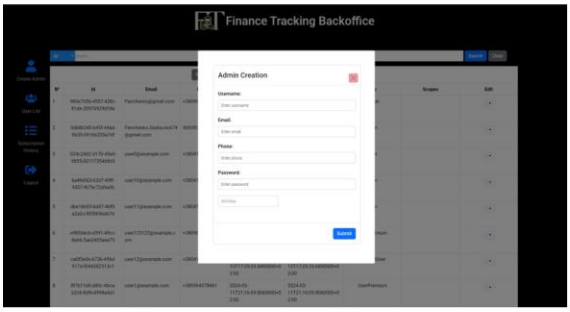
Мапа сайту користувача



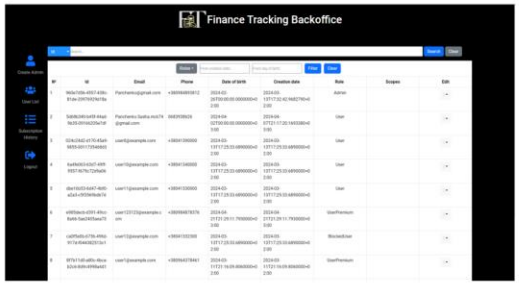
Мапа сайту адміністратора

15

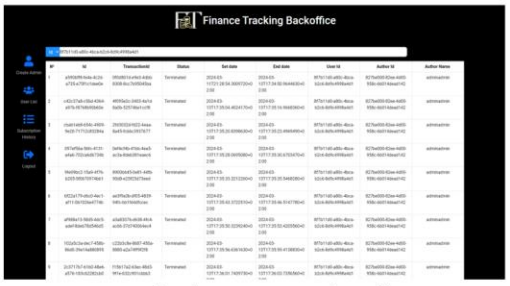
ЕКРАННІ ФОРМИ



Форма створення адміністратора



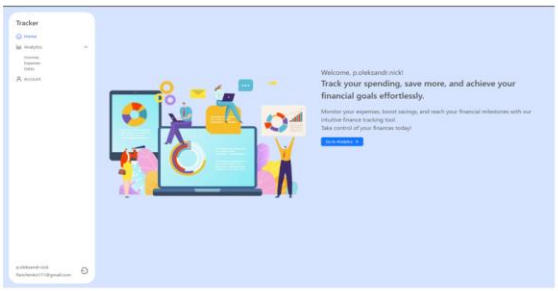
Сторінка перегляду користувачів



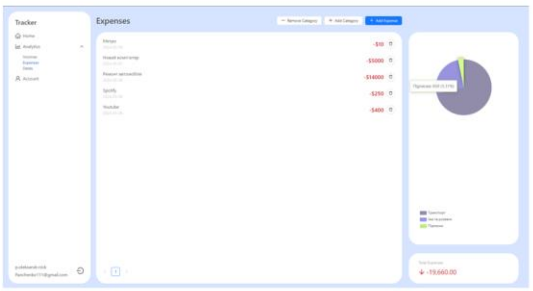
Сторінка перегляду історії преміум підписок

16

ЕКРАННІ ФОРМИ



Головна сторінка



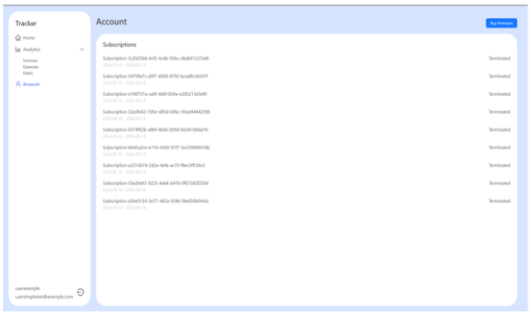
Сторінка витрат

Сторінка аналітики

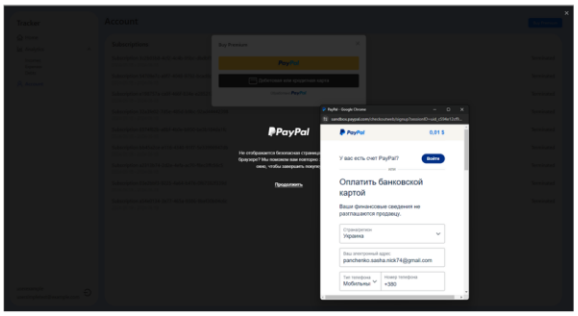
Сторінка заборгованостей

17

ЕКРАННІ ФОРМИ



Сторінка облікового запису



Купівля преміум статусу

18

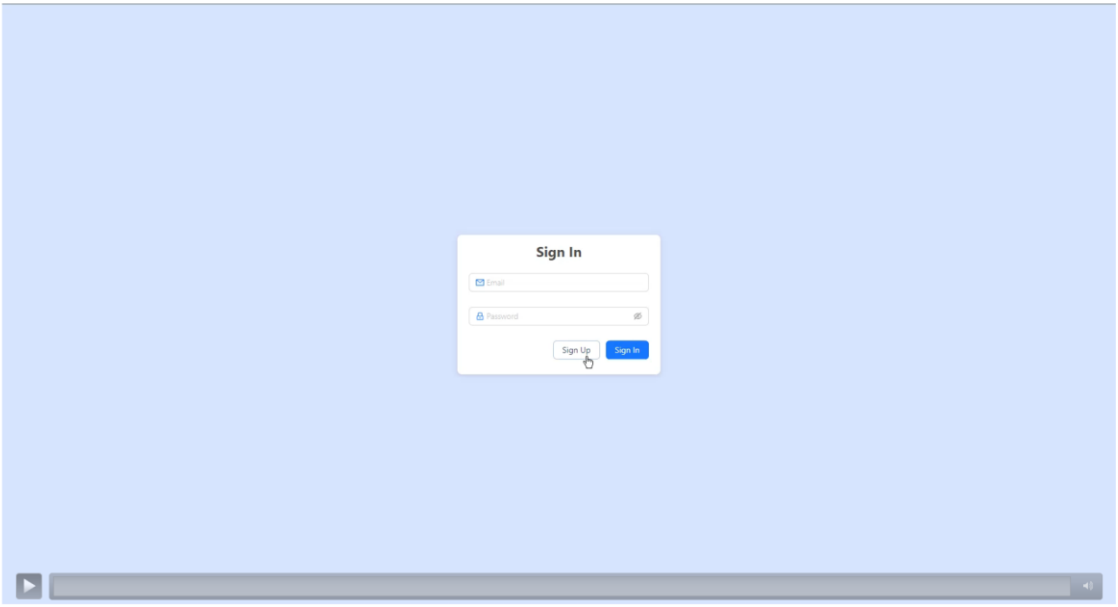
АПРОБАЦІЯ РЕЗУЛЬТАТІВ ДОСЛІДЖЕННЯ

Панченко О.М., Ільїн О.Ю. «Використання Apache Kafka для швидкого та надійного методу спілкування між мікросервісами у веб-застосунку для кервування фінансами». Всеукраїнська науково-технічна конференція «Сучасні інтелектуальні інформаційні технології в науці та світі». 14.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ. Подано до друку.

Панченко О.М., Ільїн О.Ю. «Впровадження Ocelot Gateway у веб-застосунок керування фінансами». Всеукраїнська науково-технічна конференція «Новітні апаратні та програмні засоби інтелектуальних технологій». 20.05.2024, ДУІКТ, м. Київ. К.: ДУІКТ. Подано до друку.

19

ВІДЕО РОБОТИ ЗАСТОСУНКУ



20

ВИСНОВКИ

1. Проведено аналіз обраної предметної області в сфері менеджменту фінансів.
2. Проаналізовані існуючі засоби. Визначено ключові функції, переваги та недоліки веб-застосунків фінансового менедженту.
3. Визначено основні функціональні та нефункціональні вимоги системи.
4. Змодельовано функціональність користувача, адміністратора.
5. Змодельовано взаємодію об’єктів.
6. Спроектовано та розроблено веб-застосунок з використанням мови програмування C# та JavaScript з використанням бібліотеки React. Реалізовано функціонал, який відповідає всім встановленим вимогам.
7. Проведено функціональне, регресійне, інтеграційне, E2E тестування застосунку з метою виявлення та усунення проблем системи. Під час тестування було перевірено виконання всіх вимог.

21

ДЯКУЮ ЗА УВАГУ!

ДОДАТОК В. ЛІСТИНГ ПРОГРАМНИХ МОДУЛІВ ДОДАВАННЯ ВИТРАТ В СИСТЕМУ

```
//Бізнес логіка додавання витрати серверній частині
using CSharpFunctionalExtensions;
using FinanceTracking.Domain.Base;
using FinanceTracking.Domain.Error;
using FinanceTracking.Domain.ExpensesCategoryEntity.Error;
using FinanceTracking.Domain.ExpensesCategoryEntity.ValueObjects;
using FinanceTracking.Domain.ExpensesEntity;
using FinanceTracking.Domain.ExpensesEntity.ValueObjects;
using FinanceTracking.Domain.UserEntity.ValueObjects;
using FinanceTracking.PostgreSql.Dal.Repository;
using MediatR;

namespace
FinanceTracking.ApplicationServices.Handlers.Expenses.CreateExpense;

public class CreateExpenseCommandHandler :
IRequestHandler<CreateExpenseCommand, Result<Expense, Error>>
{
    private readonly IPublisher _publisher;
    private readonly IExpensesRepository _expensesRepository;
    private readonly IExpensesCategoryQueryRepository
_expensesCategoryQueryRepository;

    public CreateExpenseCommandHandler(IPublisher publisher,
IExpensesRepository expensesRepository,
        IExpensesCategoryQueryRepository expensesCategoryQueryRepository)
    {
```

```
        _publisher = publisher ?? throw new ArgumentNullException(nameof(publisher));
        _expensesRepository = expensesRepository ?? throw new ArgumentNullException(nameof(expensesRepository));
        _expensesCategoryQueryRepository = expensesCategoryQueryRepository ?? throw new ArgumentNullException(nameof(expensesCategoryQueryRepository));
    }

    public async Task<Result<Expense, Error>> Handle(CreateExpenseCommand request, CancellationToken cancellationToken)
    {
        var categoryIdResult = CategoryId.TryParse(request.CategoryId);
        if (categoryIdResult.IsFailure)
        {
            return ExpensesCategoryValidationError.InvalidCategoryId(categoryIdResult.Error);
        }

        var userId = UserId.Create(request.UserId);

        var maybeCategory = await _expensesCategoryQueryRepository.Get(categoryIdResult.Value, userId , cancellationToken);
        if (maybeCategory is null)
        {
            return ExpensesCategoryCommonError.NotFound;
        }

        var amountResult = Amount.TryCreate(request.Amount);
        if (amountResult.IsFailure)
```

```
        {
            return ValidationError.InvalidAmount(amountResult.Error);
        }

        var description = Description.Create(request.Description);

        var expenses = Expense.CreateNew(userId, maybeCategory.Id,
amountResult.Value, request.SetDate, description);

        await _expensesRepository.Insert(expenses, cancellationToken);
        await PublishDomainEvents(expenses, cancellationToken);

        return expenses;
    }

    private async Task PublishDomainEvents(Expense expense,
CancellationToken cancellationToken)
    {
        foreach (var domainEvent in expense.DomainEvents)
        {
            await _publisher.Publish(domainEvent, cancellationToken);
        }

        expense.RemoveAllDomainEvent();
    }
}
```

```
//Логіка додавання витрати на клієнтській частині
import { message, Modal, Form, InputNumber, DatePicker, Input, Button, Select } from 'antd';
import { ExpenseCategory } from 'models';
import { useEffect, useState } from 'react';
import { addExpense, getExpensesCategoriesData } from 'services/expensesService';
import mockService from 'services/mockService';
import PropTypes from 'prop-types';
import './add-expense-modal.scss';

interface AddExpenseModalProps {
  visible: boolean;
  onClose: () => void;
}

const AddExpenseModal: React.FC<AddExpenseModalProps> = ({ visible, onClose }) => {
  const [loading, setLoading] = useState(false);
  const [categories, setCategories] = useState<ExpenseCategory[]>([]);

  useEffect(() => {
    fetchCategories();
  }, [visible]);

  const fetchCategories = async () => {
    try {
      const response = await getExpensesCategoriesData(1, 100);
      setCategories(response.items);
```

```
    } catch (error) {
      setCategories(mockService.generateMockExpenseCategories(10));
      message.error('Failed to fetch categories. Please try again later.');
    }
  };

  const handleAddExpense = async (values: any) => {
    setLoading(true);
    try {
      const selectedDate = new Date(values.setDate);
      const utcDate = new Date(
        Date.UTC(selectedDate.getFullYear(), selectedDate.getMonth(), selectedDate.getDate())
      );
      await addExpense(values.categoryId, values.amount, utcDate.toISOString(), values.description);
      message.success('Expense was added successfully');
      onClose();
    } catch (error) {
      message.error('Failed to add expense. Please try again later.');
    } finally {
      setLoading(false);
    }
  };

  return (
    <Modal
      title="Add Expense"
      open={visible}
```

```
      onCancel={onClose}
      footer={null}
      className="expense-modal">
      <Form layout="vertical" onFinish={handleAddExpense}>
        <Form.Item
          name="amount"
          label="Amount"
          rules={[{ required: true, message: 'Please enter amount' }]}>
          <InputNumber style={{ width: '100%' }} />
        </Form.Item>
        <Form.Item
          name="setDate"
          label="Date"
          rules={[{ required: true, message: 'Please select date' }]}>
          <DatePicker style={{ width: '100%' }} />
        </Form.Item>
        <Form.Item name="categoryId" label="Category">
          <Select placeholder="Select a category">
            {categories.map((category) => (
              <Select.Option                                key={category.categoryId}
value={category.categoryId}>
                {category.categoryName}
              </Select.Option>
            ))}
          </Select>
        </Form.Item>
        <Form.Item name="description" label="Description">
          <Input.TextArea rows={2} />
        </Form.Item>
```

```
      <Form.Item>
        <Button htmlType="button" onClick={onClose} style={{ marginRight: 8
}}>
          Cancel
        </Button>
        <Button type="primary" htmlType="submit" loading={loading}>
          Save
        </Button>
      </Form.Item>
    </Form>
  </Modal>
 );
};

AddExpenseModal.propTypes = {
 visible: PropTypes.bool.isRequired,
 onClose: PropTypes.func.isRequired
};

export default AddExpenseModal;
```