МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Західноукраїнський національний університет
Факультет комп'ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління

РИПІЧ Богдан Віталійович

Веб-застосунок для моніторингу грошових транзакцій на основі машинного навчання / Web application for monitoring money transactions based on machine learning

Спеціальність 122 – Комп'ютерні науки
Освітньо-професійна програма – Комп'ютерні науки

Дипломний проект

Виконав студент групи КН-41
Б.В. Рипіч

Науковий керівник:
к.т.н., доцент Х.В. Ліп'яніна-Гончаренко

Дипломний проект допущено до захисту
«___»_____ 2023 р.

Завідувач кафедри
_____ М.П. Комар

Тернопіль – 2023

Факультет комп’ютерних інформаційних технологій
Кафедра інформаційно-обчислювальних систем і управління
Освітній ступінь «бакалавр»
Спеціальність 122 – Комп’ютерні науки
Освітньо-професійна програма – Комп’ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ М.П. Комар
«____» _____2022р.

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТЦІ

Рипічу Богдану Віталійовичу
(прізвище, ім’я, по батькові)

1. Тема проєкту: Веб-застосунок для моніторингу грошових транзакцій на основі машинного навчання / Web application for monitoring money transactions based on machine learning
керівник проєкту к.т.н., доцент Х.В. Ліп’яніна-Гончаренко

затверджені наказом по університету від 08 грудня 2022 р. № 491.

2. Строк подання студентом закінченого проєкту 13 червня 2023 р.

3. Вихідні дані до проєкту: технічне завдання.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- дослідити предметну область;
- провести аналіз існуючих рішень;
- спроєктувати загальну структуру системи застосунку;
- описати механізми забезпечення безпеки даних та конфіденційності користувачів;
- спроєктувати структуру ієрархічної бази даних;
- описати методи аналізу фінансових транзакцій;
- реалізувати власне API для комунікації компонентів системи;
- розробити UI та UX застосунку з врахуванням різних вимог до цільових пристроїв;
- провести тестування готового застосунку на відповідність заданим вимогам та на стабільність функціонування.

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

- схема структури ієрархічної бази даних застосунку;
- діаграма послідовності обміну даних у застосунку.

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
Н. контроль	к.т.н., доцент Х.В. Ліп'яніна-Гончаренко		

7. Дата видачі завдання 08 грудня 2022 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту	Строк виконання етапів проєкту	Примітка
1	Аналіз предметної області та існуючих рішень	16.01.2023	
2	Розробка структури застосунку та опис математичних алгоритмів	22.02.2023	
3	Програмно-технологічна реалізація застосунку та його тестування	12.05.2023	
4	Повне завершення та оформлення дипломного проєкту	01.06.2023	

Студент_____ Б.В. Рипіч
(підпис)

Керівник проєкту_____ Х.В. Ліп'яніна-Гончаренко
(підпис)

РЕФЕРАТ

Пояснювальна записка до дипломного проєкту: 64 сторінки, 17 рисунків, 2 формули, 6 додатків, 24 джерела.

Метою дипломної роботи є створення надійного та легкого у використанні фінтех-застосунку для контролю особистих коштів, який надає користувачам зручні та ефективні інструменти для ведення бюджету, моніторингу витрат, аналізу фінансових даних та управління особистими фінансами.

Об’єкт дослідження – основи розробки веб-застосунків та окремих його компонентів.

Предмет дослідження – рішення та засоби, які дозволять розробити веб-застосунок для моніторингу та контролю особистих грошових транзакцій.

Досліджено та розроблено програмне забезпечення для моніторингу власних грошових транзакцій.

Запропонована система надає користувачам вичерпні функції для формування та стеження за бюджетом, генерувати статистику витрат та доходів, а також дозволяти переглядати поточний стан рахунків та список здійснених транзакцій за рахунками.

ВЕЛИКІ ДАНІ, СИСТЕМА ОБРОБКИ ІНФОРМАЦІЇ, ВЕБ-ЗАСТОСУНОК, ФІНАНСИ, API.

ABSTRACT

The bachelor’s thesis report: 64 pages, 17 figures, 2 formulas, 6 appendices, 24 sources.

The aim of the thesis project is to create a reliable and easy-to-use fintech application for personal funds control, which provides users with convenient and effective tools for budgeting, monitoring expenses, analysing financial data, and managing personal finances.

The object of research is the basics of developing web applications and its individual components.

The subject of the research is solutions and tools that will allow developing a web application for monitoring and controlling personal money transactions.

Software for monitoring personal money transactions has been researched and developed.

The proposed system provides users with comprehensive functions for creating and monitoring a budget, generating statistics on expenses and income, and allowing them to view the current status of accounts and a list of transactions on accounts.

BIG DATA, INFORMATION PROCESSING SYSTEMS, WEP-APPLICATION, FINANCES, API.

ТЕХНІЧНЕ ЗАВДАННЯ

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

1.1 Програмна система для моніторингу особистих фінансових транзакцій.

1.2 Область застосування – аналіз стану особистих рахунків та фінансових транзакцій.

2. ОСНОВА ДЛЯ РОЗРОБЛЕННЯ

Основою для розроблення є завдання на дипломний проєкт, затверджене кафедрою інформаційно-обчислювальних систем і управління факультету комп'ютерних інформаційних технологій Західноукраїнського національного університету.

3. ПРИЗНАЧЕННЯ РОЗРОБЛЕНОГО КОМПЛЕКСУ

Метою дипломної роботи є створення надійного та легкого у використанні фінтех-застосунку для контролю особистих коштів.

4. ДЖЕРЕЛА РОЗРОБЛЕННЯ

Джерелами даної розробки є матеріали навчальної і реферативної літератури, технічна документація, науково-дослідні статті, журнали, Інтернет.

5. ТЕХНІЧНІ ВИМОГИ

5.1 Основні функціональні вимоги до програмної системи:

- автентифікація користувача у системі;
- автоматизований збір інформації про стан рахунків користувача та здійснені транзакції за рахунками;
- обробка отриманих даних в автоматизованому режимі та за запитом;
- виведення сформованої статистики в графічному варіанті для подальшого аналізу користувачем.

5.2 Вимоги до апаратних засобів:

- комп'ютер з щонайменше 2Гб ОЗУ, багатоядерним ЦПУ, 7ГБ вільного місця

на диску та доступом в Інтернет.

5.3 Вимоги до програмних засобів:

– для розробки програмне забезпечення – Node.js, Firebase CLI, Heroku CLI, веб-орієнтовний IDE;

– для створення графічного інтерфейсу користувача використано – inkscape, Adobe Photoshop, Figma.

6. ПОРЯДОК КОНТРОЛЮ

6.1 Представлення дипломного проєкту на попередній захист.

6.2 Представлення дипломного проєкту на захист.

Завдання прийняв до виконання _____ Б.В. Рипіч

(підпис) (прізвище та ініціали)

Керівник дипломного проєкту_____ Х.В. Ліп'яніна-Гончаренко

(підпис) (прізвище та ініціали)

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API – прикладний програмний інтерфейс.

MVP – мінімально життєздатний продукт.

PWA – Поступовий веб-застосунок.

UI – дизайн інтерфейсу користувача.

ВСТУП

На сьогоднішній день існує безліч фінансових технологій та застосунків, які призначені для спрощення взаємодії клієнтів фінансових установ з самими установами. Однак не менш важливою ознакою для розвитку фінансового сектору країни є фінансова грамотність українців. Згідно досліджень національного банку України, фінансова грамотність українців зросла за останні роки, проте є значна диференціація у рівні фінансової грамотності між різними віковими групами – найвищу фінансову грамотність мають люди віком 25-34 роки та 30-59 років, коли молодь та пенсіонери показують куди гірші результати.

Відповідно створення зручних застосунків для контролю особистих фінансів та грошових транзакцій, які будуть таргетовані на молоду аудиторію, є крайньо актуальною задачею. Подібні застосунки сприяють зростанню фінансової грамотності та покращують ефективність управління власними фінансами.

Метою дипломної роботи є створення надійного та легкого у використанні фінтех-застосунку для контролю особистих коштів, який надає користувачам зручні та ефективні інструменти для ведення бюджету, моніторингу витрат, аналізу фінансових даних та управління особистими фінансами. Застосунок повинен надавати користувачам вичерпні функції для формування та стеження за бюджетом, генерувати статистику витрат та доходів, а також дозволяти переглядати поточний стан рахунків та список здійснених транзакцій за рахунками.

Для реалізації цього застосунку та досягнення мети дипломної роботи необхідно виконати наступні дії:

- дослідити предметну область;
- провести аналіз існуючих рішень;
- спроєктувати загальну структуру системи застосунку;
- описати механізми забезпечення безпеки даних та конфіденційності користувачів;
- спроєктувати структуру ієрархічної бази даних;
- описати методи аналізу фінансових транзакцій;
- реалізувати власне API для комунікації компонентів системи;

- розробити UI та UX застосунку з врахуванням різних вимог до цільових пристроїв;
- провести тестування готового застосунку на відповідність заданим вимогам та на стабільність функціонування.

Об’єктом дослідження є основи розробки веб-застосунків та окремих його компонентів.

Предметом дослідження є рішення та засоби, які дозволять розробити веб-застосунок для моніторингу та контролю особистих грошових транзакцій.

Практичне значення роботи полягає у тому, що створений інструмент має практичне застосування у реальному житті – розроблений фінтех-застосунок може бути використаний реальними користувачами для контролю своїх особистих фінансів, прогнозування витрат та доходів, а також він допомагає робити раціональні фінансові рішення, що в кінцевому рахунку покращує фінансову грамотність користувача та спонукає його глибше аналізувати власні фінанси та шукати додаткові інструменти для подальшого розвитку свої фінансових знань та можливостей.

Дипломна робота складається із вступу, трьох розділів, висновку, списку використаних джерел та додатків з графічними матеріалами та прикладами коду. Повний обсяг роботи становить 64 сторінки друкованого тексту, який включає 17 рисунків та дві формули. Список використаних джерел із 24 найменувань викладено на 3 сторінках. Шість додатків (А-Д) викладено на 8 сторінках, з них 2 додатка викладені на аркушах формату А3.

1 АНАЛІЗ РИНКУ ФІНТЕХ-ЗАСТОСУНКІВ

1.1 Дослідження предметної області

Фінансові технології (більш відомі як фінтех) використовуються для опису нових технологій, спрямованих на покращення та автоматизацію надання та використання фінансових послуг. По суті, фінтех використовується для того, щоб допомогти компаніям, власникам бізнесу та споживачам краще управляти своїми фінансовими операціями, процесами та життям. Він складається зі спеціалізованого програмного забезпечення та алгоритмів, які використовуються на комп'ютерах і смартфонах.

Ринок фінтех-застосунків наповнений безліччю рішень, більшість з яких – це банківські програми, функціонал яких зав’язаний на взаємодії з екосистемою банка та банківськими системами України та світу. Мобільні та веб застосунки для банкінгу зазвичай мають базовий функціонал, який дозволяє клієнтам банку здійснювати операції з рахунками, такі як переказ коштів, оплата рахунків, перегляд історії транзакцій тощо.

Послуги, які надають фінтех-рішення можна розділити на наступні категорії:

1) Позики та кредити
2) Платежі та перекази
3) Інвестиції
4) Особисті фінанси
5) Валюта
6) Блокчейн

В межах цього дипломного проєкту, я зосереджусь на фінтех-застосунках для контролю особистих фінансів. Основна мета мобільних застосунків для особистих фінансів – відстежувати витрати і складати бюджет відповідно до них, що в кінцевому підсумку дозволяє управляти особистими коштами. Підтримання особистих фінансів у належному стані завжди є вирішальним фактором для кожної людини, при цьому часто важко дотримуватися дисципліни та старанності. Мобільні додатки для особистих фінансів полегшують це завдання ефективно та результативно, керуючи доходами, рахунками, витратами, інвестиціями тощо.

Доступність через мобільні телефони забезпечує перевагу на ринку і може призвести до швидкого проникнення.

Згідно аналітики Fact.MR [4], в середньому кожен смартфон має 2.5 встановлених застосунків для контролю особистих фінансів. Це число пов’язане з технічними обмеженнями та закритістю фінансових систем, що не дозволяє створити єдиний універсальний, щоб надаватиме користувачу всі необхідні фінансові функції. Це змушує користувачів встановлювати декілька фінансових застосунків: офіційні застосунки банків, платіжних систем, фінансових менеджерів, застосунки для інвестицій, тощо. При цьому сам ринок таких застосунків показує стабільний ріст – у другому кварталі 2020 року кількість завантажень мобільних додатків для персональних фінансів сягнула 13 мільярдів, а частота використання додатків зросла в геометричній прогресії.

У зв’язку з вище зазначеною інформацією можна зазначити, що ринок застосунків для контролю особистих фінансів є перспективним та залучає великі інвестиції. При цьому самі застосунки ще не досягли плато власного функціонального та технічного розвитку, тому регулярно з’являються нові застосунки, що стають дедалі зручнішими та простішими у використанні.

1.2 Огляд та аналіз існуючих рішень

Аналіз застосунків для контролю особистих фінансів можна розпочати зі з'ясування їх основних функціональних можливостей, інтерфейсу та користувальницьких можливостей. Основні параметри, які можна проаналізувати в таких застосунках, включають, але не обмежуються:

а) Функціональні можливості:

1) Можливість введення даних про доходи та витрати;
2) Автоматичне зчитування даних про транзакції з різних банківських рахунків;
3) Створення різних типів бюджетів (місячного, річного, тижневого тощо);
4) Графіки та інші візуалізації, що допомагають зрозуміти структуру витрат та доходів;

5) Аналіз витрат та доходів за певний період часу.

б) Інтерфейс та досвід користувача:

1) Приємний та легкий інтерфейс;

2) Доступ до застосунку через веб-сайт та мобільний додаток.

Розглянемо детальніше описані можливості застосунків.

Можливість введення даних про доходи та витрати: у зв’язку з технічними обмеженнями банківських систем, далеко не всі банки та платіжні системи мають публічне API для отримання інформації про фінансовий стан користувача та його доходи чи витрати. У зв’язку з цим, застосунки, які роблять ставку на універсальність, дають користувачам можливість самим вносити дані в застосунок у кілька дотиків на екрані. Прикладами застосунків з таким методом отримання даних є MoneyKeeper (приклад UI додавання транзакції представлено на рисунку 1.1), SpendSense та Wallety.

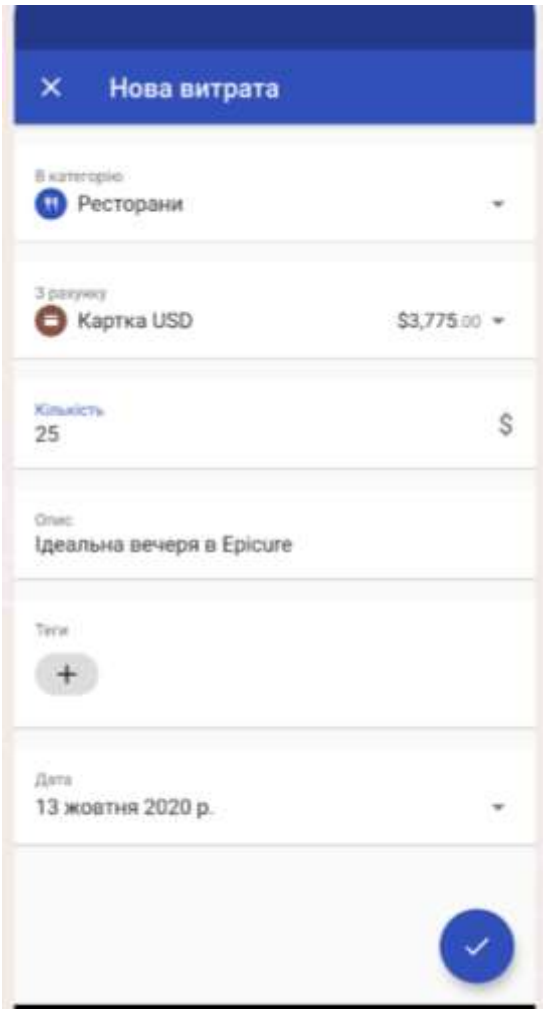


Рисунок 1.1 – Приклад створення транзакції в застосунку MoneyKeeper.

Автоматичне зчитування даних про транзакції з різних банківських рахунків: менш універсальні застосунки, які працюють на базі API конкретних банківських та платіжних систем, можуть автоматично отримувати дані про здійснені транзакції та автоматично формувати статистику. Це позбавляє користувача необхідності виконувати одну й ту ж рутинну роботу з внесенням даних в застосунок та стеженням за актуальністю інформації. При цьому користувач все ще може самостійно відредагувати деяку інформацію, наприклад додати теги до транзакції для зручного пошуку або змінити категорію, якщо застосунок помилково визначив іншу категорію для транзакції через недоліки у власній базі MCC-кодів. За таким принципом дані отримують такі застосунки як PocketGuard та YNAB (You need a budget).

Створення різних типів бюджетів: персональні фінансові менеджери дозволяють сформувати власний бюджет з прогнозованих та фактичних витрат та доходів і, відповідно, планувати власні витрати та доходи на різні проміжки часу. Частіше всього, бюджет відображається як набір шкал, межі яких користувач формує самостійно, і які поступово заповнюються чи спустошуються при здійсненні транзакцій.

Графіки та візуалізації: застосунок формує візуалізації у вигляді кругових, лінійних, стовпчикових та інших діаграм та презентує користувачу статистику доходів та витрат у зручний для перегляду спосіб та швидкого аналізу. Використання елементів аналізу на основі глибокого машинного навчання для прогнозування витрат та доходів зустрічаються рідко, оскільки витрати та доходи фізичної особи в короткі періоди часу (наприклад поденна статистика протягом місяця) носять часткового хаотичний характер та важко прогнозуються. Наприклад, в поточному місяці може вийти довгоочікуваний фільм, заболіти зуб чи з’явиться раптова причина для свята, що спричинить неочікувані витрати. Саме тому простіші застосунки обходяться більш простими, проте все ще ефективними методами аналізу витрат.

Приклад графіків та візуалізації статистики у застосунку Easy Home Finance представлено на рисунку 1.2.

Рисунок 1.2 – Приклад графіків та візуалізацій витрат у застосунку Easy Home Finance.

Аналіз витрат та доходів за певний період часу: деякі застосунки мають методи глибокого аналізу доходів та витрат. Ці методи аналізу дозволяють отримати інформацію, яку складно обчислити за допомогою графіків. Наприклад, деякі застосунки аналізують середні місячні витрати, враховуючи дисперсію в наборі даних.

Приємний та легкий інтерфейс: немає єдиного робочого підходу для створення дійсно зручного інтерфейсу. Все залежить від функцій застосунку та глибини їх реалізації, проте є певні тенденції, які прослідковуються у багатьох застосунках – це лаконічність (візуальна простота без зайвих елементів оздоблення інтерфейс), знайомість (використання верстки, термінів та іконок, які вже зустрічались в інших популярних застосунках) та швидкість взаємодії (користувач отримує доступ до частини застосунку, яка його цікавить, в найменшу кількість кліків, дотиків чи жестів).

Враховуючи той факт, що на ринку представлено менше застосунків, які мають прямий зв’язок з банківськими та платіжними системами через їх API, а також більшу зручність та автоматизованість цих застосунків, кінцевою ціллю в реалізації проєкту є створення MVP фінтех-застосунку для моніторингу та контролю власних коштів на базі одного з українських банків з відкритим персональним чи корпоративним API.

Проаналізувавши банки, клієнтом яких я являюсь, я вибрав саме Monobank як основу для мого застосунку.

Отже, при аналізі інсуючих рішень було описано їх переваги та недоліки, а також виділено типові функціональні особливості, які користувачі будуть очікувати і в нових рішеннях. Цю інформацію варто враховувати при постановці задачі, яка буде сформульована в наступному параграфі.

1.3 Вибір методів аналізу фінансових транзакцій

Одним з предметів дослідження цієї роботи є методи регресійного аналізу та їх користь для аналізу фінансових транзакцій. Регресійний аналіз дозволяє виявити неочевидні на перший погляд залежності у наборі даних. Розглянемо декілька методів регресійного аналізу та виберемо найоптимальніший варіант для вирішення поставлених задач:

а) лінійна регресія;
б) поліноміальна регресія;
в) логістична регресія.

Лінійна регресія – це статистичний метод, що використовується для встановлення та моделювання лінійного зв'язку між залежною змінною (відгуком) і однією або більше незалежними змінними (прогнозами). Вона покликана досліджувати, як змінюється відгук при зміні прогнозованих значень.

У лінійній регресії припускається, що залежна змінна залежить від незалежних змінних згідно з лінійною функцією. Модель лінійної регресії показує, як змінюється середнє значення відгуку при збільшенні на одиницю незалежної змінної, при умові, що інші незалежні змінні залишаються постійними.

Поліноміальна регресія – це статистичний метод, що використовується для апроксимації регресійних функцій, які не є лінійними. У поліноміальній регресії зв’язок між незалежною змінною x і залежною змінною y описується як поліном n-го ступеня від x. Поліноміальна регресія може бути використана не лише для візуалізації тренду в наборі даних, але й для прогнозування майбутнього значення залежної змінної на основі значень незалежних змінних.

Логістична регресія – це статистичний метод, який використовується для моделювання ймовірності виникнення певної події залежно від значень незалежних змінних. Вона використовується в основному для задач класифікації, де залежна змінна приймає бінарні значення (так/ні, 0/1, позитивний/негативний). Оскільки для нашого набору даних залежна змінна не є бінарною, цей статистичний метод не підходить для аналізу грошових транзакцій.

Що стосується вибору між поліноміальною та лінійною регресією, то я схиляюсь до використання саме лінійної регресії. На відміну від поліноміальної регресії, вона набагато простіша у реалізації, а отримана з її допомогою функція лінії тренду дозволяє миттєво побачити закономірності у зростанні чи спаданні витрат та доходів, а також те, що наразі переважає – витрати чи доходи. Можливість прогнозування даних, використовуючи поліноміальну регресію, також здається сумнівною ідеєю, оскільки для подобової та потижневої статистики характерна велика дисперсія в наборі даних, що робить прогноз неінформативним та некорисним для кінцевого користувача.

Враховуючи ці факти можна зробити висновок, що лінійна регресія є найоптимальнішим вибором, що зумовлено в першу її простотою - як в плані реалізації, так і в плані розуміння її результатів кінцевими користувачами.

1.4 Постановка задачі

Актуальність цієї роботи базується на недостатній фінансовій грамотності молоді, яку кінцевий застосунок має допомогти покращити. Враховуючи цей факт, застосунок повинен мати набір функцій, які спрямовані на інформування

користувачів про стан їх банківських рахунків та аналіз фінансових транзакцій. З цих функцій витікають і вимоги до кінцевого застосунку.

Метою дипломної роботи є створення веб-застосунку для перегляду та аналізу особистих коштів та фінансових транзакцій, а точніше, створення MVP, що при завершенні проєкту повинен мати наступні функції та особливості:

1) Веб-версія та кросплатформа: Monobank позиціонує себе як «банк у телефоні», і тому не має веб-версії та повноцінної версії для планшетів. Extended Report буде реалізований як PWA, що зробить його доступним на будь-якій платформі, що має підтримку сучасних інтернет-браузерів. Застосунок має бути адаптивним до екранів різних розмірів, а також мати підтримку як мишок з клавіатурою, так і тачскрінів;
2) Авторизація: застосунок повинен мати захист від несанкціонованого доступу, тому Extended Report повинен мати панель реєстрації та авторизації, а також можливості для відновлення доступу в разі втрати паролю;
3) Список транзакцій та їх аналіз: можливість переглянути список транзакцій за обраний проміжок часу та деталі конкретної транзакції. Самі транзакції мають підтягуватись автоматично, не змушуючи користувача самостійно вносити дані в застосунок;
4) Бюджет: можливість сформувати свій бюджет на певний період часу, як в загальному, так і по окремим категоріям;
5) Сповіщення: можливість отримувати PUSH-сповіщення на пристрої про здійснені транзакції та перевищення запланованих бюджетом витрат;
6) Розширений пошук: фільтрування транзакцій за опціями, які не доступні у офіційному застосунку, наприклад за наявністю кешбека чи по конкретній валюті;
7) Незалежність від доступу до сервісів monobank: транзакції мають зберігатись в окремій базі даних, а клієнт повинен мати змогу переглянути збережені дані навіть коли сервери monobank є недоступними.

Для зручності опису, в подальшому мій застосунок буде згадуватись під робочою назвою Monobank Extended Report, або ж просто Extended Report.

Для реалізації цього застосунку необхідно вирішити наступні задачі:

1. дослідити предметну область;
2. провести аналіз існуючих рішень;
3. спроєктувати загальну структуру системи застосунку;
4. описати механізми забезпечення безпеки даних та конфіденційності користувачів;
5. спроєктувати структуру ієрархічної бази даних;
6. описати методи аналізу фінансових транзакцій;
7. реалізувати власне API для комунікації компонентів системи;
8. розробити UI та UX застосунку з врахуванням різних вимог до цільових пристроїв;
9. провести тестування готового застосунку на відповідність заданим вимогам та на стабільність функціонування.

Завдання 1-2 розглянуто в параграфах 1.1 - 1.2 відповідно, а виконання завдань 3-9 представлені у параграфах 2.1-3.3.

2 АЛГОРИТМІЧНЕ ТА ІНФОРМАЦІЙНЕ ПРОЄКТУВАННЯ ЗАСТОСУНКУ

2.1 Загальна структура проєктованої системи

Як було описано у параграфі 1.4, проєкт буде реалізовано як PWA з можливістю роботи у якості звичайного веб-застосунку. Веб-застосунки мають кілька переваг порівняно з повноцінними мобільними застосунками:

1) Не потрібно встановлювати: Веб-застосунки не потребують встановлення на пристрій, що дозволяє їм бути легкими і швидкими в розгортанні. Крім того, це також зменшує обсяг пам'яті, необхідної для зберігання додатків на пристрої.
2) Більша доступність: Веб-застосунки можуть бути запущені на будь-якому пристрої з Інтернет-підключенням та браузером, що робить їх більш доступними для користувачів. При цьому, їх можна запустити на різних операційних системах та пристроях, що зменшує витрати на розробку та тестування додатків на різних платформах.
3) Простота оновлення: Оновлення веб-застосунків можуть бути зроблені безпосередньо на сервері, що дозволяє користувачам отримувати оновлення без необхідності встановлення нових версій додатку на пристрої.
4) Зниження вартості розробки: Розробка веб-застосунків може бути більш доступною порівняно з розробкою повноцінних мобільних застосунків, оскільки не потрібно розробляти окремі версії для кожної платформи. Крім того, веб-застосунки можуть бути створені з використанням стандартних веб-технологій, таких як HTML, CSS та JavaScript, що дозволяє використовувати існуючі знання та інфраструктуру для їх розробки.

Звичайно, є деякі недоліки веб-застосунків порівняно з повноцінними мобільними додатками. Наприклад, вони можуть бути менш продуктивними та менш зручними для використання без доступу до Інтернету. Однак, за більшістю випадків, веб-застосунки можуть бути більш оптимальним вибором для багатьох додатків з огляду на їх переваги в доступності, зручності та вартості розробки, а реалізація застосунку як PWA дозволить наблизить користувацький досвід до досвіду використання повноцінних мобільних застосунків.

Термін “Прогресивний веб-застосунок” (PWA) з’явився в 2015 році. Так дизайнери Google Chrome називали новий тип застосунків, які використовують нові технології сучасних інтернет-браузерів, наприклад Service Workers та файли маніфесту, і дозволяють користувачам встановлювати веб-застосунки, так як застосунки першого класу в своїй рідній операційній системі. Активна популяризація PWA розпочалась відносно недавно - починаючи з 2016 року, проте популярність стрімко росте з кожним роком. Обсяг глобального ринку прогресивних веб-застосунків у 2019 році становив 1,13 мільярда доларів США, а до 2027 року очікується, що він досягне 10,44 мільярда доларів США при середньорічному темпі зростання 31,9% протягом прогнозованого періоду. Використання прогресивних веб-застосунків зростає, і вони досить популярні серед малих і середніх підприємств (згідно прогнозу від Emergen Research). Серед відомих прикладів PWA є Uber, Twitter, Spotify і Pinterest, та багато інших популярних сервісів.

Pinterest, соціальна мережа для обміну зображеннями, повідомила [5] про покращення кількох показників ефективності після впровадження нових веб-технологій та переходу на PWA, як-от:

- на 60% зросла основна взаємодія користувачів із сервісом
- на 40% зросла кількість рекламних переходів
- на 44% зросли доходи від реклами, створеної користувачами

Схожі результати має і соціальна мережа Twitter [6]:

- на 65% зросла кількість сторінок, яку відвідує середньостатистичний користувач за один сеанс
- на 75% зросла кількість твітів
- на 20% знизився bounce rate

Прогресивні веб-застосунки мають кілька переваг порівняно зі звичайними веб-застосунками. Наприклад, PWA можуть працювати в автономному режимі, тобто без доступу до Інтернету, за рахунок кешування необхідних для роботи застосунку ресурсів та HTTP-реквестів у пам’яті пристрою, що є дуже важливим для обходу обмежень API Monobank, які були описані вище. Також PWA можуть бути встановлені на домашній екран користувача і мати доступ до особливих

функцій пристрою, таких як камера або мікрофон, без необхідності запитувати дозвіл кожного разу. PWA, як і звичайні веб-застосунки, можуть працювати на різних платформах і пристроях. Найбільшим недоліком PWA є те, що станом на початок 2023 року велика кількість браузерів (Mozilla Firefox, Safari, Opera) досі не підтримують цю технологію, проте в таких випадках застосунок буде сприйматись операційною системою та браузером як звичайний веб-застосунок.

В якості технологічного стеку для розробки власного прогресивного веб-застосунку було обрано наступний набір технологій:

1) React для Frontend;
2) Node.js та Express для Backend;
3) Google Firestore в якості бази даних;
4) Google Firebase Hosting для хостингу ресурсів застосунку;
5) Heroku для хостингу Backend-частини застосунку.

Цей стек має кілька переваг:

React для Frontend: React є дуже популярною бібліотекою для розробки інтерфейсів користувача. Вона дозволяє створювати компоненти, які можна повторно використовувати і розширювати. Вона також має велику спільноту, що робить її легкою для вивчення та підтримки. Крім того, React працює зі статичними та динамічними сторінками, що дозволяє легко і швидко створювати масштабовані застосунки.

Node.js та Express для Backend: Node.js є популярним серверним середовищем JavaScript. Воно дозволяє розробникам використовувати JavaScript як мову програмування для створення Backend частини застосунку. Це дозволяє під час розробки використовувати ті ж знання та інструменти, які використовуються для розробки Frontend частини. Express є фреймворком для Node.js, який дозволяє швидко створювати API та Web-додатки. Він має простий та зрозумілий API та підтримує широкий спектр плагінів та модулів.

Google Firestore в якості бази даних: Firestore є документовою базою даних від Google. Вона дозволяє легко зберігати, оновлювати та запитувати дані. Firestore має дуже швидкий API та дозволяє легко масштабувати застосунки. Firestore дозволить використовувати комплексі пошукові запити, а ієрархічна структура

даних, що базується на колекціях документів, дозволяє швидко створити API для комунікації між Frontend та Backend частинами застосунку. Вона також має добру підтримку для Node.js та React, що дозволяє легко інтегрувати її в обраний стек технологій.

Google Firebase Hosting для хостингу ресурсів застосунку: використання цього сервісу для хостингу ресурсів проєкту забезпечує надійне шифрування по протоколу HTTPS, а також високу швидкість з'єднання.

Heroku для хостингу Backend-частини застосунку: Heroku – це хмарний хостинг, який забезпечує зручний та простий спосіб розгортання та хостингу веб-застосунків. Він дозволяє легко масштабувати додатки в залежності від потреби. Це дозволяє збільшувати потужність сервера при збільшенні навантаження на додаток. Також Heroku забезпечує безпечний та захищений спосіб розгортання застосунків в інтернеті. Це включає в себе захист від атак, захист даних та безпеку мережі.

В першу чергу варто розібратись з розгортанням застосунку. Весь застосунок можна розділити на 3 компоненти, які повинні бути розгорнуті. UML-діаграма розгортання представлена на рисунку 2.1:

1) Клієнтська частина, яка виконується на пристрої клієнта, використовуючи браузер як середовище виконання коду, і хоститься на серверах Google Firebase Hosting;
2) Серверна частина, яка виконується в контейнеризованому сервері Heroku, використовуючи Node.js як середовище виконання коду;
3) База даних, яка зберігається та опрацьовується на серверах Google Firestore.

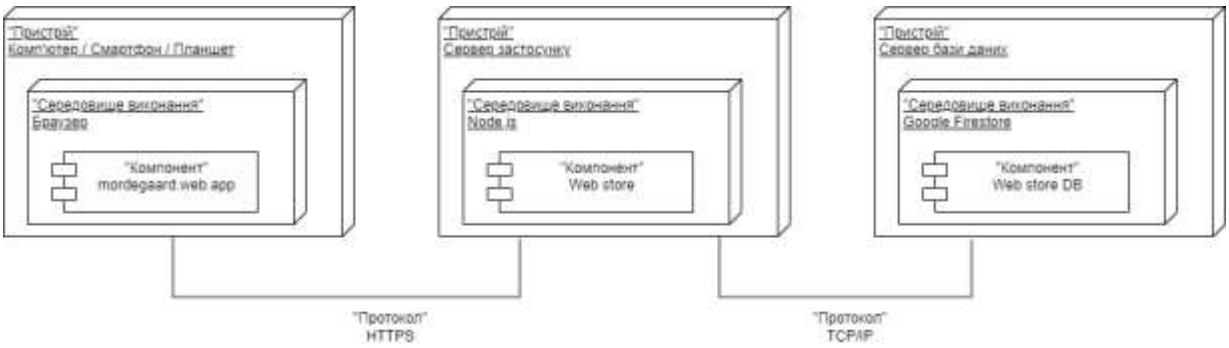


Рисунок 2.1 – UML діаграма розгортання застосунку

Процес розгортання компонентів проєкту в Google Firebase Hosting та Heroku відбуваються схожим чином – ці сервіси мають власні CLI-застосунки, за допомогою яких файли проєкту заливаються на сервера сервісів. Відповідно, ці CLI-застосунки можна легко інтегрувати у власний цикл розробки, використовуючи bat чи bash файли з необхідними командами.

Розгортання бази даних відбувається безпосередньо в консолі проєкту Google Firestore, а підключення до неї відбувається в Node.js, використовуючи набір криптографічно-стійких ключів.

Взаємодія між серверною та клієнтською частиною відбуватиметься за допомогою REST API.

REST (Representational State Transfer) API є архітектурним стилем для розробки програмного забезпечення, який використовується для створення веб-служб та API. REST API використовує принципи та протоколи Інтернету, такі як HTTP (Hypertext Transfer Protocol), для забезпечення обміну даними між клієнтами та серверами. Основними характеристиками REST API є використання URI (Uniform Resource Identifier), за допомогою якого кожен ресурс REST API представляється та ідентифікується його місцезнаходження. Для взаємодії з ресурсами в REST API використовуються HTTP-методи. Найпоширенішими методами в REST API є GET (для отримання ресурсів), POST (для створення ресурсів), PUT (для оновлення чи редагування ресурсів) та DELETE (для видалення ресурсів). Передаються дані в REST API зазвичай у форматі JSON через свою простоту та легкість використання. Так як JSON є ієрархічною репрезентацією даних, це спрощує процес формування результатів запиту від ієрархічної No-SQL бази даних Google Firestore.

Дослідивши документацію до REST API Monobank [8], варто відмітити певні обмеження API, які впливають на функціональність та простоту розробки застосунку:

1) Доступ до корпоративного API потребує верифікації застосунку, а персональне API потребує ручного створення ключа доступу та внесення його у застосунок. Також персональне API дозволено використовувати лише якщо

дані клієнтів не будуть надходити на ваші сервери або ви робите сервіс для своєї родини;

2) Отримувати інформацію про клієнта, стан його рахунків та транзакції можна не частіше, ніж один раз у шістдесят секунд, тому варто кешувати інформацію, щоб запобігти зайвим запитам;
3) Максимальний час, за який можливо отримати виписку по транзакціям – 31 доба + 1 година (2682000 секунд). Відповідно, слід передбачити механізми протидії небажаним запитам, а також кешувати результати запитів, які не змінюються протягом часу або змінюються нечасто (як наприклад інформація про відкриті рахунки);
4) Максимальна кількість транзакцій, яку повертає виписка – 500, тому варто надати перевагу отримання транзакцій напряму з власної бази даних, структура якої буде описана в параграфі 2.3;
5) Транзакції мають MCC-код, проте не мають категорії, тому необхідно зібрати власну базу MCC-кодів з їх категоріями, які будуть відповідати категоріям Monobank;
6) Дані про транзакцію, які повертає API, не мають тегів, які користувач додає до транзакцій у застосунку, що ускладнює пошук транзакцій за межами офіційного мобільного застосунку Monobank;
7) REST API Monobank не дозволяє отримати інформацію про стан накопичень (депозити, облігації, кешбек, тощо).

Взаємодія з Monobank REST API відбувається за допомогою HTTP-запитів. Ці запити можна поділити на ті, які ініціює застосунок, та ті, які ініціює сам сервер Monobank.

Запити, які ініціює застосунок, повинні мати спеціальний ключ доступу. У випадку персонального API, цей ключ генерується на сторінці https://api.monobank.ua/ після проходження авторизації в офіційному мобільному застосунку. Оскільки Extended Report використовуватиме саме персональне API, користувачу необхідно самостійно згенерувати ключ або скопіювати вже існуючий ключ на сторінці та вставити його в поле у застосунку.

Цей ключ використовується для Bearer-автентифікації запитів та має бути надійно захищений від несанкціонованого перехоплення та витоків. Про методи забезпечення безпеки при збереженні та передачі ключа буде детальніше розписано у параграфі 2.2.

Запити, які ініціює сервер Monobank, відправлятимуться за допомогою вебхука, який попередньо необхідно встановити, використовуючи API.

Вебхук (Webhook) – це механізм, який дозволяє веб-застосункам передавати автоматичні повідомлення або дії іншим затосункам після виникнення певної події. Іншими словами, вебхук - це HTTP-запит, який автоматично відправляється до вказаної URL-адреси, коли відбувається певна подія в системі. У випадку Monobank API, єдиною подією, яка відправляє дані через вебхук, є здійснення транзакції за рахунками клієнта. Після здійснення транзакції сервер Monobank спробує надіслати за вказаною у вебхуці адресою деталі транзакції. Після отримання цих деталей застосунок повинен зберегти їх у базі даних та відправити користувачу сповіщення про здійснену транзакцію.

Важливою функцією застосунку є PUSH-сповіщення – це спосіб надсилання повідомлень або сповіщень користувачам на їх пристрої без потреби активної взаємодії із застосунком або веб-сторінкою. Вони надсилаються на пристрої через механізми оперативної системи або браузера і можуть містити текстові повідомлення, зображення, звуки або посилання. Особливістю PUSH-сповіщень є їх активне надходження навіть тоді, коли відповідний додаток або веб-сторінка не активні або не відкриті на пристрої користувача. Це дозволяє надсилати користувачам важливі повідомлення, оновлення або сповіщення в режимі реального часу.

PUSH-сповіщення працюють за наступним алгоритмом:

1) Користувач всередині застосунку дає згоду на отримання PUSH-сповіщень. Користувач має вибір дозволити або заборонити програмі або веб-сайту надсилати йому сповіщення;
2) Після того, як користувач надасть дозвіл, пристрій реєструється в службі push-повідомлень, наданій операційною системою (наприклад, Apple Push Notification Service для iOS або Firebase Cloud Messaging для Android) або веб-

службі PUSH-сповіщень. Ця реєстрація передбачає створення унікального маркера пристрою або ідентифікатора реєстрації, який ідентифікує пристрій користувача;

3) Сервер застосунку (також відомий як сервер-постачальник) інтегрується зі службою PUSH-сповіщень, використовуючи її API або SDK. Він отримує дозвіл користувача з реєстрацією та маркером та зберігає її у себе для подальшої ідентифікації пристрою користувача;
4) Сервер застосунку використовує збережений маркер, щоб надіслати запит з необхідними даними (текст, зображення, тощо) до служби PUSH-сповіщень;
5) Служба PUSH-сповіщень отримує запит від сервера застосунку та ідентифікує цільовий пристрій за допомогою маркера. Потім служба доставляє сповіщення на відповідний пристрій через інфраструктуру PUSH-сповіщень відповідної платформи;
6) Коли пристрій отримує PUSH-сповіщення, він запускає відображення сповіщення на системному рівні, яке може містити повідомлення, піктограму, звук або інші візуальні чи звукові елементи. Сповіщення відображається користувачеві, навіть якщо програма або веб-сайт не запущено, що дозволяє йому переглянути відповідну інформацію або взаємодіяти зі сповіщенням;
7) користувач може взаємодіяти з PUSH-повідомленням, торкнувшись його чи окремої кнопки під сповіщенням. Ця дія може запустити програму чи веб-сайт і спрямувати користувача на певну сторінку або виконати попередньо визначену дію в програмі чи на веб-сайті.

Варто розуміти, що дозвіл асоціюється не з конкретним користувачем, а з пристроєм конкретного користувача. Тобто якщо користувач використовує застосунок на планшеті та на смартфоні, то і дозволи на показ сповіщень користувач може налаштувати по-різному на планшеті та смартфоні, і ці дозволи будуть мати різну реєстрацію.

Схематично маршрут сповіщення від сервера застосунку Extended Report до кінцевого користувача представлено на рисунку 2.2.

Рисунок 2.2 – Маршрут PUSH-повідомлення від сервера застосунку до користувача

Як висновок можна сказати, що обраний стек технологій дозволяє розробникам швидко створювати масштабовані та ефективні веб-застосунки без особливих фінансових затрат, а просто схема розгортання застосунку дозволяє швидко та просто доставляти нові оновлення для застосунку в майбутньому.

2.2 Опис алгоритмів забезпечення безпеки даних та конфіденційності

Компоненти застосунку обмінюються даними, використовуючи захищені протоколи, як-от HTTPS при обміні даними між клієнтською та серверною компонентами застосунку.

Проте шифрування під час обміну даними не є достатнім, аби забезпечити необхідний рівень безпеки, тому варто попіклуватись над шифруванням даних, які зберігатимуться в базі даних, та над шифруванням чутливих даних, які передаватимуться по спроєктованому HTTP REST API між серверною та клієнтською компонентами.

Для шифрування даних використовуватиметься алгоритм симетричного шифрування AES-CBC. Це симетричний шифр блочного типу, який використовується для шифрування даних у багатьох сферах, де потрібно забезпечити конфіденційність та цілісність даних, наприклад для шифрування конфіденційної інформації в базах даних, таких як паролі, персональні дані користувачів та інші дані, а також для шифрування збережених файлів, таких як документи, зображення та інші файли.

AES (Advanced Encryption Standard) - це стандарт шифрування, який використовується для захисту даних від несанкціонованого доступу. Цей стандарт використовується у багатьох системах та протоколах, включаючи TLS, SSH та IPsec.

CBC (Cipher Block Chaining) - це режим роботи шифрування, який використовується в AES. У режимі CBC кожен блок відкритого тексту шифрується окремо, після чого він комбінується з попереднім блоком за допомогою операції XOR, перед тим як бути зашифрованим. Цей процес забезпечує більшу безпеку, оскільки шифрування кожного блоку залежить від шифрування попереднього блоку.

Проте, при використанні режиму CBC потрібно враховувати деякі недоліки. Наприклад, він може бути вразливим до атак на вектор ініціалізації. Microsoft у своєму документі [9] повідомляє, що розшифровувати дані, зашифровані за допомогою режиму симетричного шифрування Cipher-Block-Chaining (CBC), коли було застосовано перевірену вставку без попереднього забезпечення цілісності зашифрованого тексту, більше не є безпечним, за винятком дуже специфічних обставин. Цей тип атаки на зашифровані методом AES-CBC дані називається «Атакою оракула підстановки» (Padding oracle attack). На щастя, на сьогоднішній день існують способи запобігти цьому типу атак, тому AES-CBC є достатньо надійним методом шифрувати чутливі персональні дані в базі даних.

За допомогою AES-CBC будуть шифруватись паролі користувачів та їх ключі доступу до API Monobank. Шифруватись ці дані будуть як при збереженні в базі даних, так і при передачі, при чому для цього будуть використовуватись різні секрети – секрет бази даних та секрет для передачі. Секрет для передачі зберігається як на стороні сервера, так і на стороні клієнта. При першому отриманні ключа доступу до API Monobank, цей ключ має бути зашифрований та переданий до сервера застосунку, використовуючи безпечні транспортні протоколи. Основу безпеку при передачі ключа забезпечує використання HTTPS-протоколу, який використовує криптографічні протоколи SLL чи TLS для шифрування основних даних в запиті – URL-адреси, заголовків запиту, кукі та безпосередньо вмісту

запита. Використання симетричного шифрування для самого ключа забезпечує додаткову безпеку від перехоплення запитів на рівні клієнта.

Зберігатись секрети мають в змінних середовища системи. Це необхідно, щоб секрети були відокремлені від коду програми, щоб запобігти випадковому витоку цих секретів на хмарні системи.

Процес AES-CBC шифрування та подальшого збереження зашифрованих даних відбувається за наступним алгоритмом:

- Сервер отримує зашифровані транспортним ключем (він же ключ для передачі) дані від пристрою клієнта;
- Ці дані розшифровуються, використовуючи транспортний ключ;
- У випадку успішної розшифровки, розшифровані дані шифруються ключем бази даних;
- Зашифровані новим ключем дані зберігаються в БД;
- У випадку помилки при розшифровці, сервер оброблює цю помилку за принципом, що детальніше описаний в параграфі 3.1.

Блок-схема цього алгоритму представлена на рисунку 2.3.

Процес AES-CBC розшифровування даних з БД та подальшої відправки цих даних користувачу відбувається за наступним алгоритмом:

- Сервер отримує зашифровані дані з бази даних;
- Сервер розшифровує отримані дані, використовуючи ключ бази даних;
- У випадку успішної розшифровки, сервер форматує розшифровані дані та повторно їх шифрує, використовуючи вже ключ для передачі даних;
- Сервер відправляє дані на пристрій користувача;
- У випадку помилки при розшифровці, сервер оброблює цю помилку за принципом, що детальніше описаний в параграфі 3.1.

Блок-схема цього алгоритму представлена на рисунку 2.4.

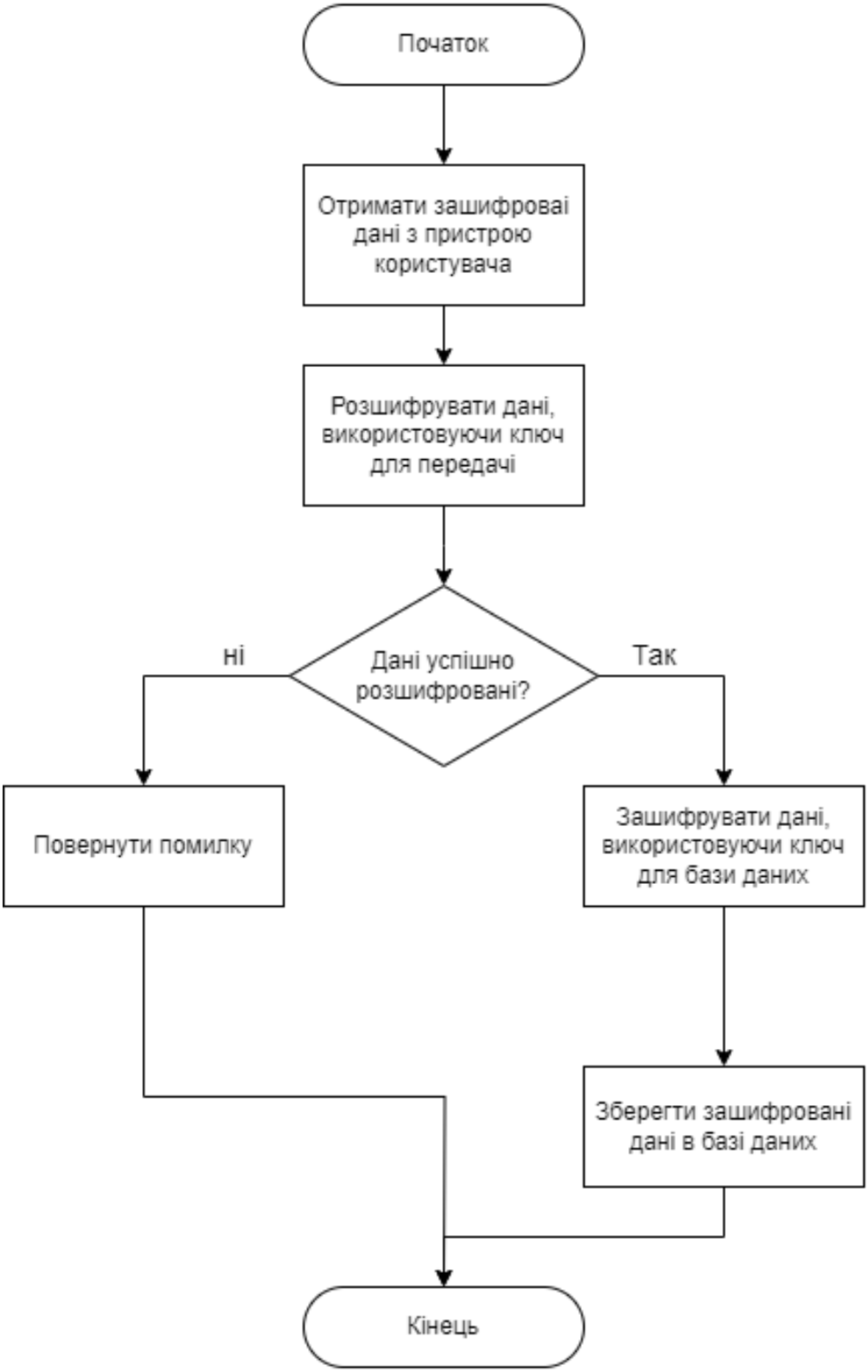


Рисунок 2.3 – Блок-схема алгоритму збереження чутливих даних у базі даних

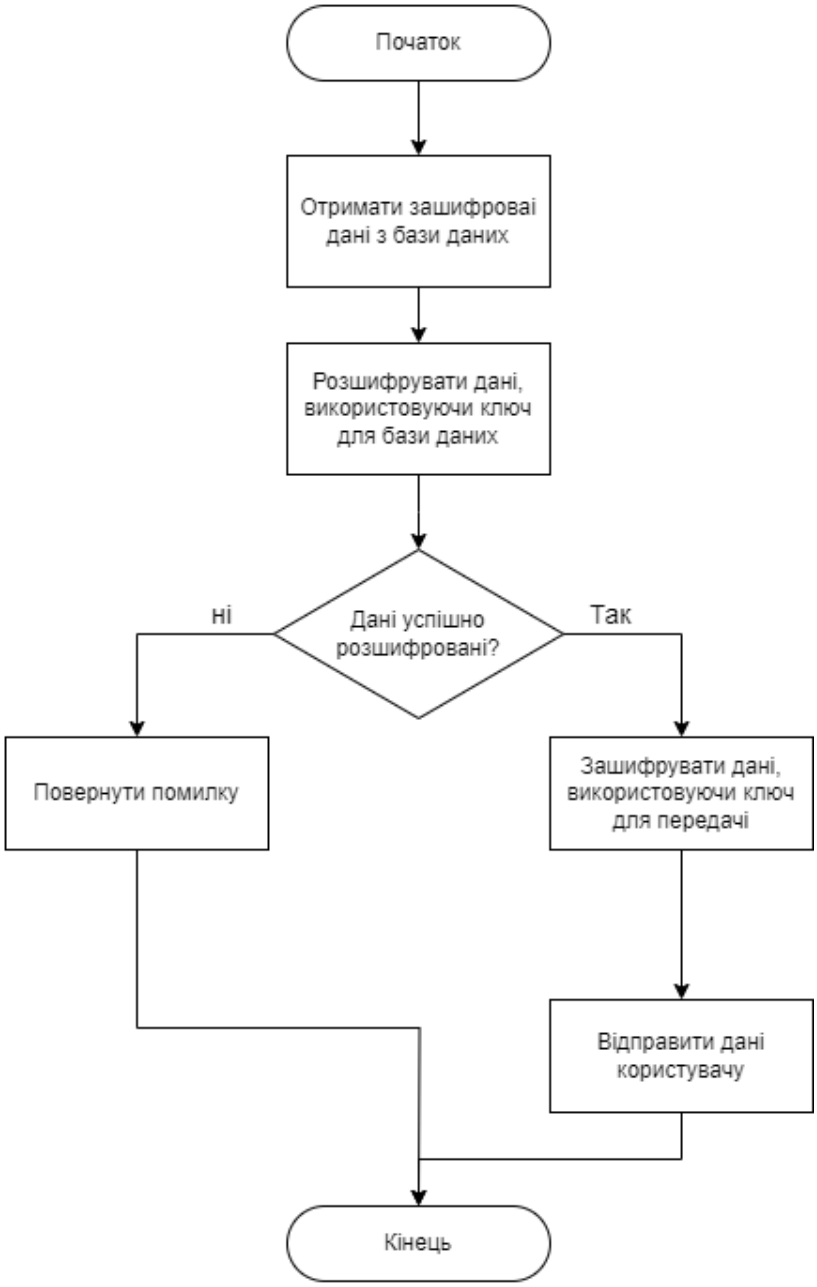


Рисунок 2.4 – Блок-схема алгоритму відправки чутливих даних клієнту зі сторони сервера

Отже основою безпеки застосунку є використання безпечних сервісів з власними механізмами безпеки, як-от наявність HTTPS-сертифікатів у Google Firebase Hosting та SSH-шифрування при розгортанні коду сервера на Heroku. Що стосується чутливих персональних даних, то використання симетричного шифрування дозволяє швидко та просто шифрувати та дешифрувати дані, що дає додатковий шар захисту.

2.3 Структура ієрархічної бази даних

Як було описано вище, проєкт використовує ієрархічну No-SQL базу даних Google Firestore. В ієрархічних базах даних інформація структурно представляється не набором взаємопов'язаних таблиць, а у вигляді дерева з батьківськими та дочірніми вузлами. У цих базах даних інформація організована в ієрархічній моделі, де кожен вузол має одного батька (окрім кореневого), і може мати багато дочірніх вузлів.

В Google Firebase, модель даних реалізована у вигляді документів, які об'єднуються в колекції. Документ може мати необмежену кількість колекцій, а колекції можуть мати необмежену кількість документів, проте колекція не може містити в собі інші колекції, рівно як і документи не можуть містити в собі інші документи. Це зумовлено особливостями індексації даних, щоб забезпечити найкращу швидкість читання та запису даних у базі. Також варто розуміти, що в Google Firebase не має строгої типізації та структури вузлів – будь-який вузол може змінити як кількість своїх дочірніх вузлів, так і типи даних у цих вузлах. Через це у майбутньому описана у цій роботі структура бази даних може втратити свою актуальність. Ці особливості варто враховувати при проєктуванні структури бази даних.

На вершині ієрархії БД буде 3 основні колекції: Transactions, Users та Logs:

- В колекції Transactions зберігаються документи всіх транзакцій, які сервер отримує від вебхука Monobank та які були модифіковані на стороні сервера.
- В колекції Users зберігаються документи з даними всіх користувачів застосунку включно з їх налаштуваннями, ключами доступу та додатковими даними, які асоціюються з конкретним користувачем (відкриті рахунки, наприклад).
- В колекції Logs зберігаються документи з технічними логами та помилками для подальшого аналізу. Ці дані здебільшого неструктуровані, щоб мати змогу зберігати логи різного формату та структури.

Приклад представлення цих даних в інтерфейсі Google Firestore зображено на рисунку 2.5. На ньому також можна помітити автоматично згенеровані ключі для документів та типові поля, які має збережені в БД транзакції.

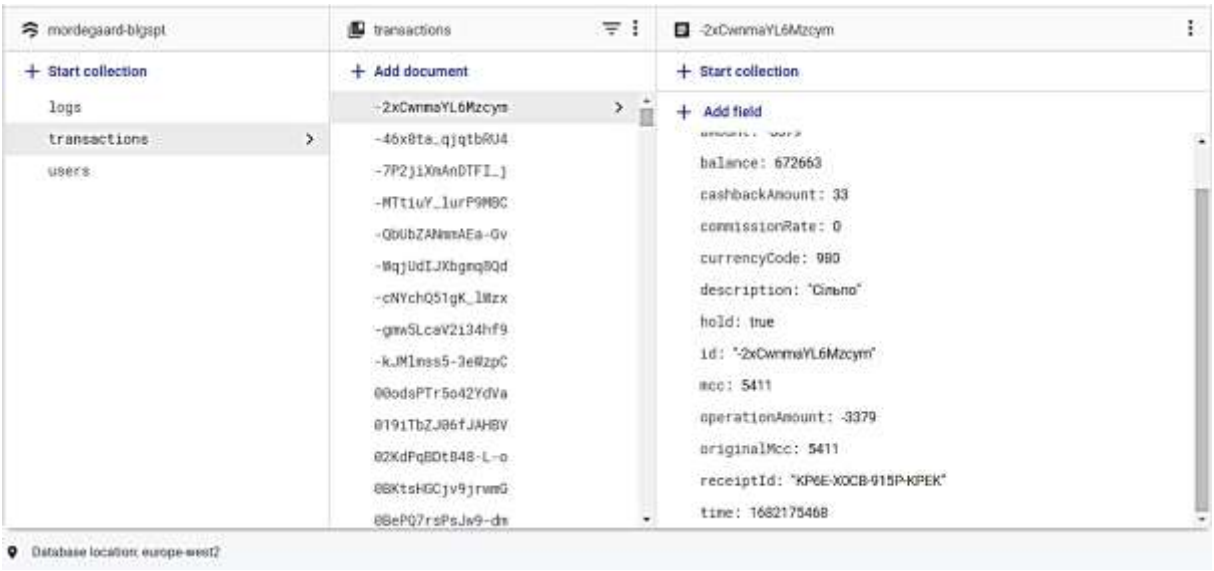


Рисунок 2.5 – Огляд бази даних в інтерфейсі Google Firebase

Зберігати транзакції в глибині ієрархії відповідного користувача, до якого ця транзакції відноситься, є не найкращою ідеєю – вебхук Monobank повертає лише ідентифікаційний номер карточки, якої ця транзакція стосується. Отже якщо зберігати транзакції всередині вузла користувача, то доведеться щоразу під час транзакції шукати серед даних усіх користувачів ідентифікаційні номери карток, які належать цим користувачам, щоб визначити, якому користувачу належить здійснена транзакція. Це дуже складний як в плані формулювання, так і в плані виконання запит.

Схему ієрархії колекцій та документів в базі даних на момент завершення MVP застосунку можна детальніше побачити у додатку Д.

Незважаючи на ієрархічну структуру, Google Firebase дозволяє робити комплексні запити – шукати документи за значенням конкретного поля, сортувати результати, об’єднувати документи, масово оновлювати документи, тощо. Єдина вимога – заранння провести індексацію необхідних полів пере створенням комплексних запитів. При цьому, завдяки розділенню основної інформації на 3 окремі колекції, можливо використовувати особливості реляційних баз даних для створення пошукових запитів. Наприклад, одним запитом можна отримати ключі банківських рахунків клієнта, а іншим – знайти транзакції, що здійснені за цими рахунками, тим самим створивши зв’язок між цими гілками.

Окрім цього, Google Firestore має систему правил – додаткової валідації запитів на стороні бази даних. В правилах можна (і навіть бажано) прописати додаткові правила, які обмежуватимуть несанкціонований доступ до інформації. Наприклад, в правилах можна заборонити додавання нових транзакцій, якщо користувача більше не існує (додаток А).

Отже описана структура бази даних є не лише масштабованою, але й простою для створення запитів до бази даних, що суттєво спрощує розробку застосунку, а використання правил дозволяє захистити базу даних від несанкціонованого доступу та помилок зі сторони розробника.

2.4 Опис методів аналізу фінансових транзакцій

Для аналізу витрат та доходів використовуватимемо кругову діаграму для аналізу витрат/доходів по категоріям та лінійну діаграму для наочного порівняння витрат з доходами.

Розглянемо, які категорії транзакцій використовує Monobank. Ці категорії можна переглянути як в офіційному мобільному застосунку, так і на дашборді на їх сайті:

1) Продукти та супермаркети
2) Кафе та ресторани
3) Інше
4) Розваги та спорт
5) Поповнення мобільного
6) Краса та здоров’я
7) Подорожі
8) Таксі
9) Комуналка та інтернет
10) Авто
11) Одяг та взуття
12) Ремонт
13) Побутова техніка
14) Кіно

15) Тварини
16) Бюджет та податки
17) Книги
18) Duty Free
19) Інше

Банк розпізнає ці категорії, використовуючи MCC-коди транзакцій. MCC-коди (Merchant Category Codes) - це чотиризначні числа, які використовуються в платіжних системах для класифікації та ідентифікації видів діяльності торговельних точок. Кожен MCC-код відповідає конкретній галузі бізнесу або типу послуги, яку надає торгова точка. Ці коди створюються та управляються Міжнародним товариством з платіжних карт. Варто розуміти, що ці коди не мають категоризації – лише сам код та назва галузі, якої стосується цей код (наприклад, 8211: Школи та університети). Відповідно кожен банк самостійно формує список категорій та створює карту залежностей між MCC-кодами та цими категоріями.

Для повної відповідності категорій у застосунку Extended Report з категоріями транзакцій в офіційному застосунку Monobank, необхідно створити власну картку залежностей кодів та категорій. Застосунок Extended Report використовує власну картку залежностей з описами 1088 різних MCC-кодів, які розбиті на 19 категорій, увідповідності з категоріями в Monobank. Частину цієї карти у форматі JSON можна побачити у додатку Б.

Тепер коли є можливість визначити категорію транзакції, маючи її MCC-код, можливо визначити відсоток витрат чи доходів по категоріям та використати цю інформацію для створення кругової діаграми. Для цього необхідно:

1) Обчислити суму всіх витрат/доходів серед транзакцій. Ця сума рахується за 100%;
2) Розділити транзакції за категоріями;
3) Обчислити суму всіх витрат/доходів в кожній категорії;
4) Поділити суму витрат/доходів у певній категорії на суму всіх витрат/доходів серед транзакцій, щоб отримати відсоток витрат/доходів у вибраній категорії серед усіх;
5) Відсортувати отримані відсотки в порядку зростання/спадання.

Для створення кругової діаграми також потрібно:

1) Помножити отримані відсотки на 2 радіани (~6.283), щоб отримати розмір секторів на круговій діаграмі
2) Відрисувати сектори, де кут початку – сума всіх розмірів попередній транзакцій на діаграмі, а кут кінця – розмір поточного сектора.

Лінійну діаграму можна генерувати, використовуючи різні періоди, наприклад, бо дням або по місяцям. Оскільки операції з датами можуть бути досить складними, я рекомендую використовувати готові рішення для роботи з датами. Оскільки в своєму стеку я використовую JavaScript як основну мову програмування, я рекомендую використовувати бібліотеку date-fns, яка має широкий набір функцій для роботи з датами. Сам спосіб обчислення координат точок на графіку не сильно міняється від обраного періоду, тому нижче я опишу спосіб обчислення точок для поденного періоду статистики:

1) Відсортувати транзакції по даті створення;
2) Створити N точок, де N – різниця в кількості днів між першою та останньою транзакцією. Початкове значення ординати для цих точок – 0;
3) Для кожної транзакції додати її грошове значення до ординати точки, що знаходиться під індексом, що дорівнює різниці в кількості днів між даною транзакцією та першою транзакцією в списку;
4) Інтерполювати значення проміжних точок, використовуючи лінійну інтерполяцію або кубічний сплайн.

Кубічний сплайн є математичною кривою, яка складається з кубічних функцій на кожному відрізку між точками. Він використовується для апроксимації або інтерполяції даних, які задані на певному відрізку або в просторі. Завдяки кубічному сплайну, лінія на графіку стає згладженою і користувачеві стає простіше визначати проміжні значення на графіку. Приклад графіка, інтерпольованого лінійно та за допомогою кубічного сплайну можна побачити на рисунку 2.7. Суцільною лінією зображено лінійну інтерполяцію, а пунктирною – інтерполяцію за допомогою кубічного сплайну.

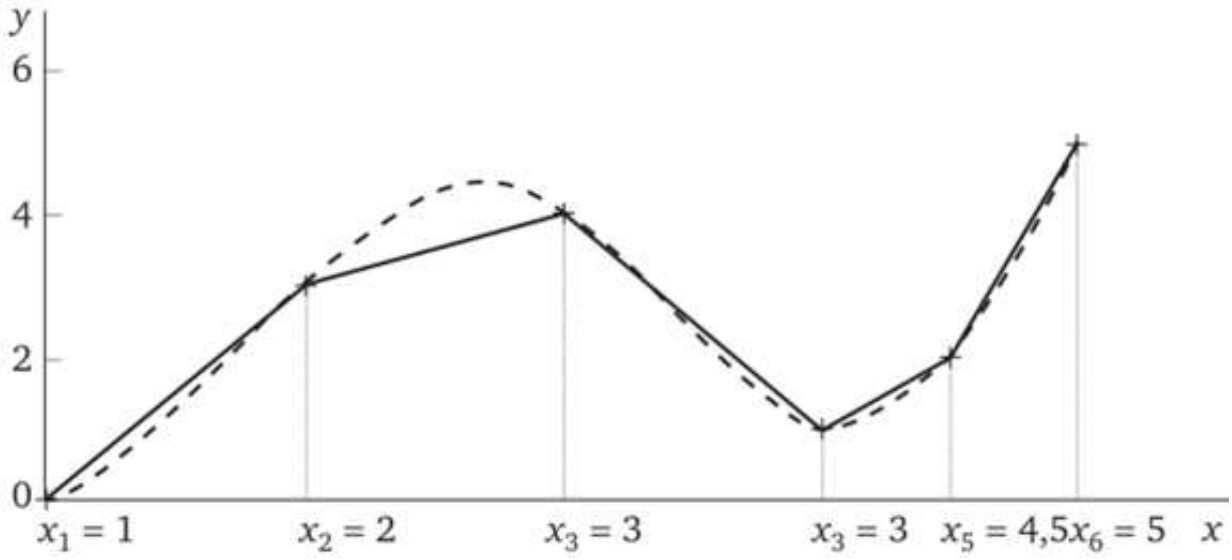


Рисунок 2.7 – Приклад використання кубічного сплайну для інтерполяції значень на лінійному графіку

Якщо лінійний графік містить високу густину точок, то не бажано використовувати кубічний сплайн, оскільки його згладжування буде непомітним, і ресурси комп'ютера будуть витрачені дарма. Проте для низької густини точок, кубічний сплайн є надзвичайно корисним, оскільки він забезпечує гладкість та допомагає зробити графік більш приємним для очей і покращує сприйняття даних. Також кубічний сплайн здатний добре відтворювати форму оригінальної кривої. Він намагається максимально відповідати оригінальним даним, що сприяє точній інтерполяції. Це особливо корисно, коли потрібно зберегти основні особливості даних і показати їхню наочність.

Маючи лінійний графік з форматованими даними, є можливість використати методи регресійного аналізу, а саме лінійну регресію, яка була обрана як найоптимальніший метод регресійного аналізу в параграфі 1.3. Використовуючи лінійну регресію можна побудувати лінію тренду на лінійному графіку, що дозволить користувачам відразу побачити тренди у своїх доходах та витратах за увесь звітній період.

Пошук лінії тренду можна спростити до рівняння прямої $Y = a + bx$, де b – коефіцієнт нахилу прямої, a – початкове зміщення прямої. Коли в даних прослідковується позитивний тренд і витрати чи доходи ростуть, то коефіцієнт b буде додатнім числом. У випадку, якщо прослідковується негативний тренд, то

коефіцієнт b буде від'ємним числом. Якщо тренд залишається немінним протягом усього набору даних, то b дорівнює нулю, і формула спрощується до виду Y = a, де a – середнє значення даних.

Різні види лінійної регресії відрізняються саме методами пошуку тренду у наборі даних. Так як наш графік базується на наборі точок, краще всього знайти формулу лінії тренду, використовуючи метод найменших квадратів (МНК). Цей метод дозволяє знайти лінійну функцію, яка найкраще підходить до набору точок, мінімізуючи суму квадратів відхилень між фактичними значеннями і прогнозованими значеннями. Оскільки ми маємо лінійну регресію з єдиною скалярною змінною, то процес знаходження чисел a та b можна описати наступним чином:

Нехай дано набір початкових даних (x_i, y_i) розміру M, де абсциса визначає час здійснення транзакцій, а ордината – суму всіх транзакцій у відповідний час.

В такому випадку числа a та b можна знайти наступним чином:

Формула для коефіцієнта нахилу b обчислюється, діливши суму добутків різниць між кожним значенням незалежної змінної та її середнім значенням на суму квадратів різниць між кожним значенням незалежної змінної та її середнім значенням.

Формула для зсуву a обчислюється шляхом віднімання добутку коефіцієнта нахилу та середнього значення незалежної змінної від середнього значення залежної змінної.

$$b = \frac{M\Sigma_i x_i y_i - \Sigma_i x_i \Sigma_i y_i}{M\Sigma_i x_i^2 - (\Sigma_i x_i)^2}, \qquad (2.1)$$

$$a = \frac{1}{M}\Sigma_i y_i - \frac{b}{M}\Sigma_i x_i \qquad (2.2)$$

Підводячи підсумки можна сказати, що правильна база MCC-кодів є надзвичайно важливою для подальшого аналізу витрат та доходів, оскільки без неї не буде технічної можливості правильно рахувати статистику по окремим категоріям. Що стосується методів регресійного аналізу, то їх використання є важливим для розширення базових методів формування статистики, що робить застосунок кориснішим для користувача.

3 ПРОГРАМНО-ТЕХНОЛОГІЧНЕ ЗАБЕЗПЕЧЕННЯ ЗАСТОСУНКУ

3.1 Програмна реалізація власного API

Для клієнт-серверної комунікації у застосунку було створено власне REST API, використовуючи фреймворк Express.js та допоміжні бібліотеки та функції.

Створення API включає в себе наступні компоненти:

1) Розробка власної системи URL-шляхів;
2) Розробка проміжних маршрутів для автентифікації користувача та обробки помилок;
3) Розробка допоміжних класів для конструювання JSON-відповідей.

Як було описано в пункті 2.1, URI використовується для опису місцезнаходжень кінцевих точок API. URI може мати вигляд URL (Uniform Resource Locator) або URN (Uniform Resource Name). Для реалізації власного API, я використовую URL-адреси, які складаються зі схеми, доменного імені та шляху до ресурсу (це і є місцезнаходження кінцевих точок).

Оскільки мій сервер хоститься в контейнерах Heroku, типова URL-адреса ресурсу виглядає наступним чином: https://mrdgrd.herokuapp.com/path/to/endpoint, де https – це схема, mrdgrd.herokuapp.com – це доменне ім’я, а path/to/endpoint – це шлях до кінцевої точки API.

Для створення шляхів, варто користуватися принципом SEF URL (Clean URL), тобто URL-адреси мають бути зручні для сприйняття людиною та використовувати знайомі слова.

Правильні URL-адреси:

- GET https://mrdgrd.herokuapp.com/monobank/notifications (отримання підписок на сповіщення)
- PUT https://mrdgrd.herokuapp.com/monobank/transactions/:id (редагування транзакції з ідентифікатором :id)
- POST https://mrdgrd.herokuapp.com/monobank/auth (авторизація користувача)

Неправильні URL-адреси:

- GET https://mrdgrd.herokuapp.com/system/new_api/transactions-list (недопустиме чередування різних прописів)
- POST https://mrdgrd.herokuapp.com/application/settings/post (не варто створювати окремі шляхи для кожного HTTP-методу)

Код, який обробляє звернення до створених шляхів, розділяється на окремі контролери, кожен з яких обробляє свій набір шляхів. Наприклад, контролер TransactionsController.js обробляє пошук транзакцій, їх оновлення та показ деталей конкретної транзакції. Код з реалізацією шляхів API наведений в додатку В.

Окрім самих шляхів, існують також проміжні маршрути з кодом, який обов'язково виконається, перед попаданням до відповідного контролера. В цих проміжних маршрутах описується логіка, пов'язана з автентифікацією користувача та обробкою помилок. Так всі шляхи мають кінцевий проміжний маршрут ErrorHandler, який при виникненні помилки формує зрозуміле клієнту повідомлення про помилку, а також записує деталі помилки в базу даних в колекцію Logs. Завдяки цьому, клієнтська частина застосунку здатна отримувати деталі помилки та по-різному опрацьовувати їх. Наприклад, якщо при спробі авторизуватись у застосунку не буде знайдено користувача з відповідною електронною поштою та паролем, сервер поверне помилку з кодом 401 та повідомленням "Bad credentials", що можна побачити на рисунку 3.1.

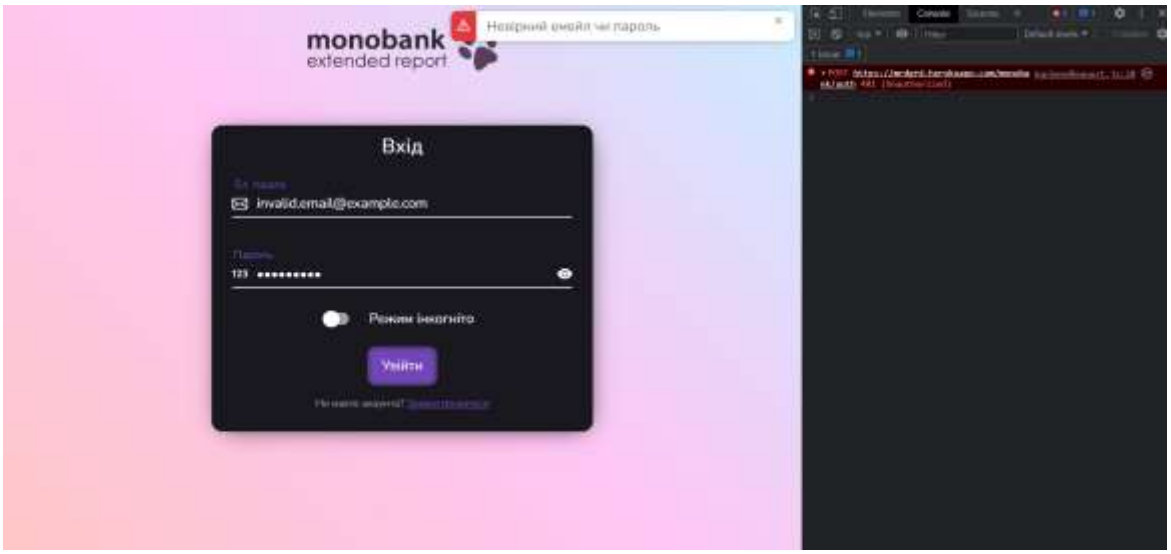


Рисунок 3.1 – Показ деталей помилки при авторизації

Коди помилок визначаються у відповідності до стандартів HTTP response status codes, що покращує розпізнавання таких помилок браузером та робить API простішим для сторонніх розробників.

Проміжний маршрут MonobankAuthorization.validate перевіряє, чи дійсно користувач авторизований у застосунку, та до яких ресурсів він має доступ. Завдяки цьому, якщо спробувати відкрити в браузері якийсь з URL-адрес застосунку, який має цей проміжний маршрут, користувач побачить повідомлення про помилку, як на рисунку 3.2.

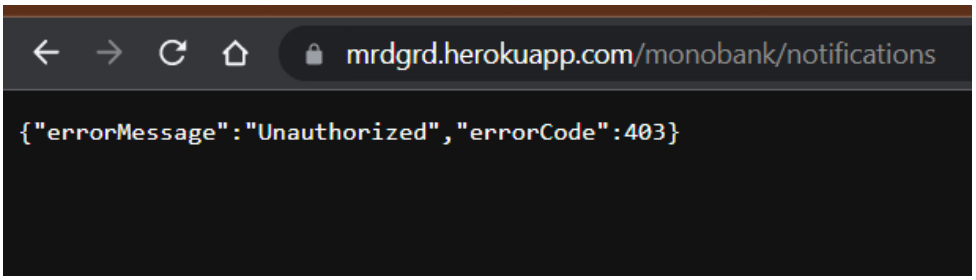


Рисунок 3.2 – Повідомлення, що користувач неавторизований

JavaScript-інтеграція бази даних Google Firebase повертає результати у вигляді стандартних JavaScript-об’єктів. Такі ж об’єкти приймає Express.js для відправки клієнту у форматі JSON. Це спокушає напряму передавати об’єкт від бази даних до Express.js, але це дуже небезпечний підхід – у відповідь на запит клієнта можуть просочитись небезпечні дані, які не варто взагалі передавати по транспортним протоколам, наприклад ключі для внутрішнього використання.

Щоб запобігти таким випадкам, було створено базовий клас JsonResource та його дочірні класи. Їх призначення – налаштовувати, які дані з об’єкта будуть передаватись клієнту. Реалізацію базового класу та приклад його використання можна побачити у додатку Г.

В результаті ми маємо основу для комунікації клієнта та сервера. Розроблене API буде легко масштабувати та додавати нові кінцеві точки, що є дуже важливим для веб-застосунків.

3.2 Розробка UI та UX веб-застосунку

Уся верстка застосунку створена за допомогою UI бібліотеки React для створення динамічної та «реактивної» верстки. Завдяки використанню React для верстки та Webpack для збірки проєкту, уся клієнтська частина застосунку реалізована за принципом all-in-one – усі компоненти інтерфейсу, їх логіка, стилі та іконки зібрані в єдиному файлі, який можна легко закешувати. В розробці зовнішнього вигляду застосунку та стилі графічних елементів використовується UI-бібліотека Bootstrap. Це одна з найпопулярніших UI бібліотек для розробки веб-інтерфейсів. Вона надає набір готових стилів, компонентів та JavaScript-інструментів, які допомагають швидко та зручно будувати сучасні дизайни та легко реалізовувати адаптивність за рахунок системи кінцевих точок (breakpoints), за допомогою яких задаються стилі елементів інтерфейсу для різних розмірів екрана.

В основі UX та UI застосунку лежать парадигми, описані Google в їх методичці Material Design, а саме:

1) Використання системи сітки для розміщення об'єктів на екрані, з урахуванням пропорцій, розміщення відступів та інтервалів між елементами. Це допомагає створити збалансований та привабливий дизайн.
2) Використання яскравих, насичених кольорів та їх поєднань для виділення важливих елементів інтерфейсу, створення ієрархії та передавання настрою. Material Design надає рекомендації щодо використання кольорів, тіней та переходів між ними.
3) Використання чіткої та зрозумілої типографіки для полегшення читання та передачі інформації. Material Design надає рекомендації щодо вибору шрифтів, розмірів та відступів між текстовими елементами.
4) Використання анімацій та переходів для плавності та відчуття просторовості в інтерфейсі. Це допомагає візуально показати зв'язок між об'єктами та станами, підкреслити важливі події та додати динаміки до взаємодії.
5) Забезпечення простоти та зручності використання інтерфейсу для користувачів. Це включає такі аспекти, як чітка навігація, доступність

елементів управління, інтуїтивно зрозумілі жести та дії, консистентність стилів та взаємодії.

6) Рекомендація використовувати адаптивний дизайн, що адаптується до різних розмірів екранів та пристроїв, забезпечуючи зручне використання на різних платформах та пристроях.
7) Використання контекстних меню та дій для забезпечення швидкого доступу до відповідних опцій та функцій, які пов'язані з певним об'єктом чи контекстом.

При плануванні та реалізації інтерфейсу застосунку я намагався дотримуватись усіх цих парадигм.

У веб-розробці система сітки реалізується за допомогою використання системи Grid та Flexbox, що дозволяє зручно створювати адаптивну верстку для сторінок, на якій розміри різних елементів адаптуються до розмірів екрану.

Завдяки використанню колірних акцентів, кожна категорія транзакцій має свій набір кольорів, що використовується у застосунку як при відображенні іконки транзакції, але й під час показу кругової діаграми, бюджету та деталей транзакції. На рисунку 3.3 можна побачити, як UI використовує синій колір, що асоціюється з категорією «Переказ коштів» для кращого акцентування елементів користувацького інтерфейсу.

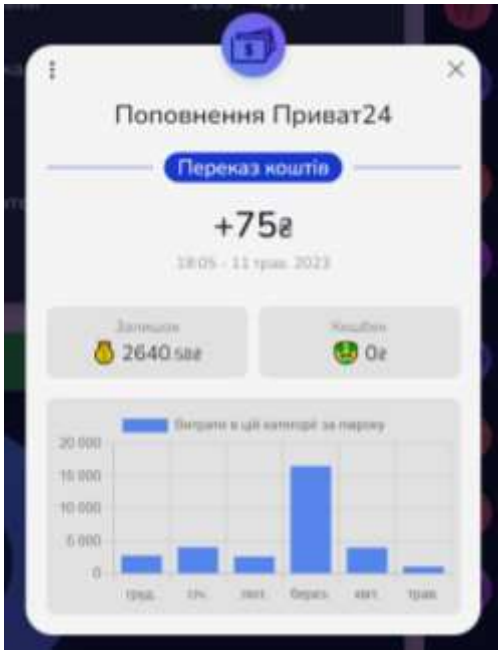


Рисунок 3.3 – Використання власних кольорів категорії «Переказ коштів» у модальному вікні з деталями транзакції

Користувачів застосунку можна поділити за платформою, яку вони використовують для взаємодії із застосунком. Різні пристрої надають різні можливості для взаємодії користувача з програмами, тому застосунок повинен підлаштовуватись під ці можливості. Основними платформами, на яких буде працювати застосунок, – це смартфони (тачскрін, малий розмір екрану, портретна орієнтація), планшети (тачскрін, великий розмір екрану, довільна орієнтація) та персональні комп'ютери (миша та клавіатура, великий розмір екрану, альбомна орієнтація).

UI має адаптуватись до вимог цільових платформ. В окремих випадка вся верстка застосунку може відрізнятись, залежно від платформи. Так, наприклад, мобільні пристрої мають значно менший екран, ніж ПК, то інтерфейс мобільних застосунків повинен бути більш компактним та мінімалістичним. У мобільних застосунках меню зазвичай знаходиться у верхньому, нижньому або боковому рядку, як це представлено на рисунку 3.4, щоб звільнити місце на екрані. У ПК-застосунках, меню може бути розташоване у верхній частині екрана, або мати власну панель інструментів.

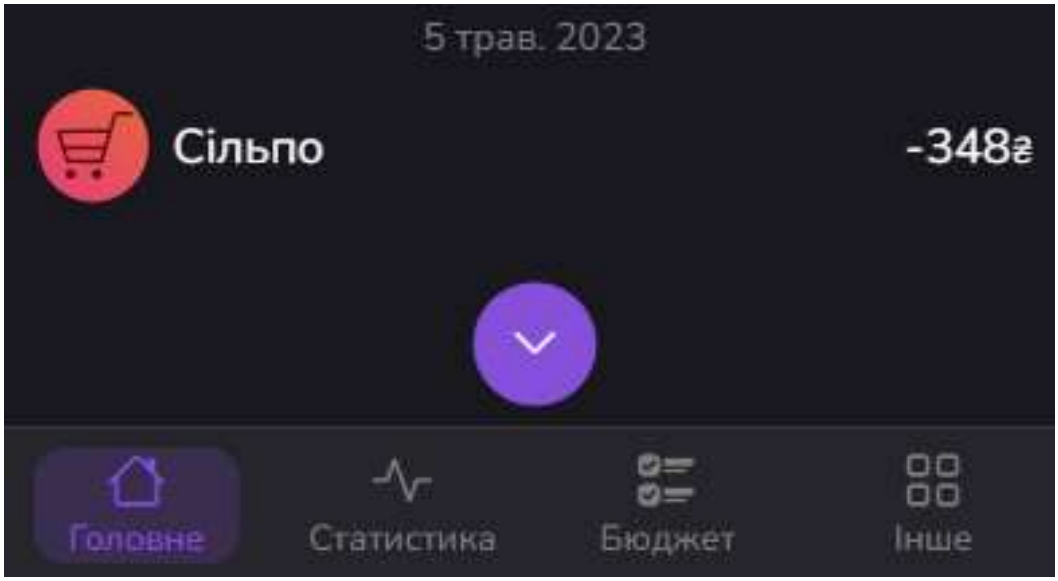


Рисунок 3.4 – Приклад навігаційного меню, розташованого внизу екрана

Загалом зовнішній інтерфейсу намагається відповідати зовнішньому вигляду сторінки дашборду Monobank для комп'ютерної та планшетної версії та дизайну мобільного застосунку Monobank для мобільної версії. Це проявляється у використанні схожої колірної палітри, градієнтів та окремих деталей інтерфейсу.

Порівняння інтерфейсу комп’ютерної версії застосунку з сайтом Monobank показано на рисунку 3.5, а на рисунку 3.6 показано порівняння інтерфейсів мобільної версії застосунку Extended Report з офіційним мобільним застосунком Monobank.

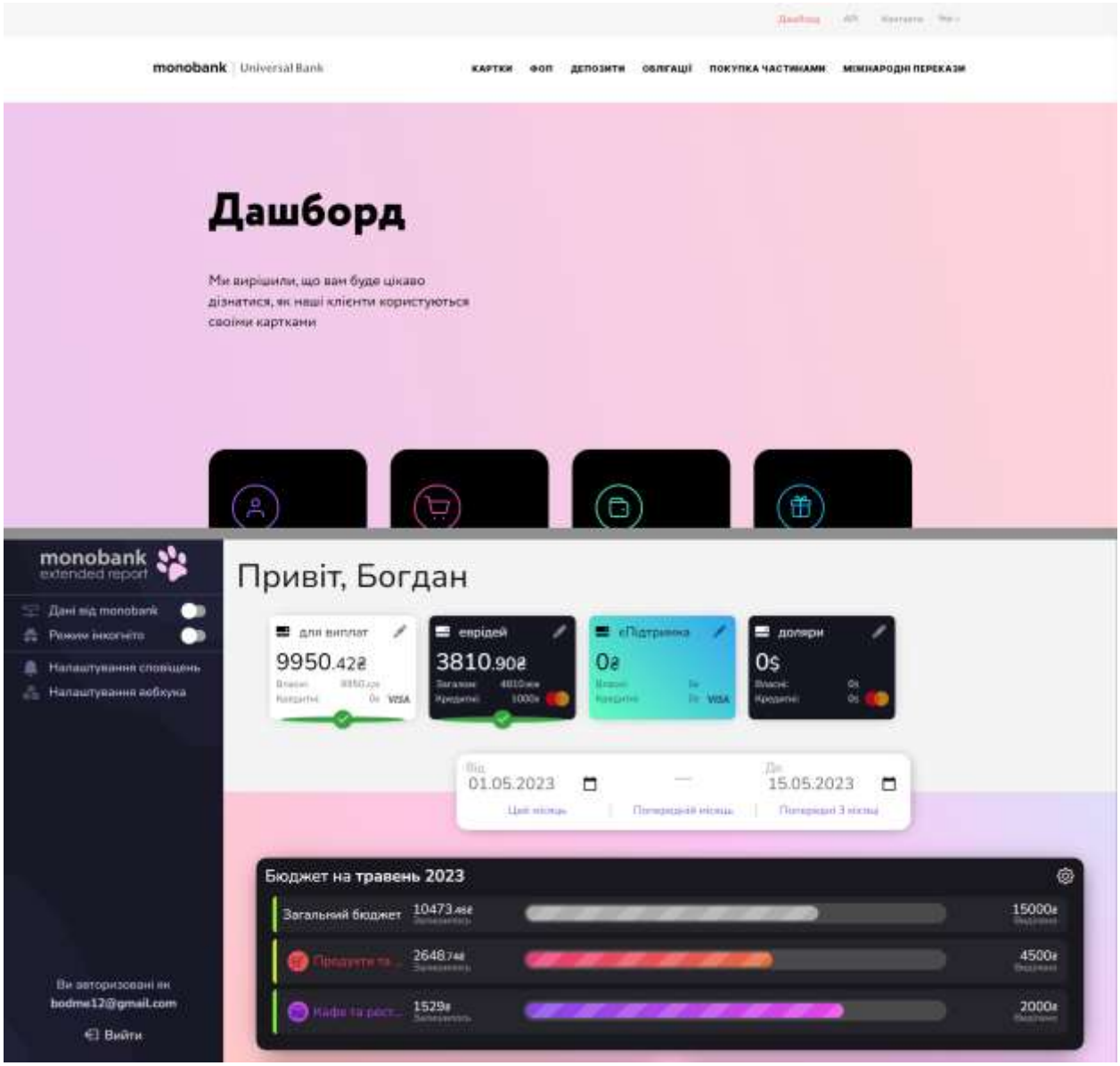


Рисунок 3.5 – Порівняння сторінки дашборду (зверху) та комп’ютерної версії застосунку (знизу)

Так, наприклад, реалізація списку транзакцій майже повністю відповідає тому, як вона реалізована в офіційному мобільному застосунку, включаючи кольорову диференціацію витрат та доходів, сортування від найновіших до найстаріших транзакцій, групування транзакцій по дням та показ іконки категорії транзакції.

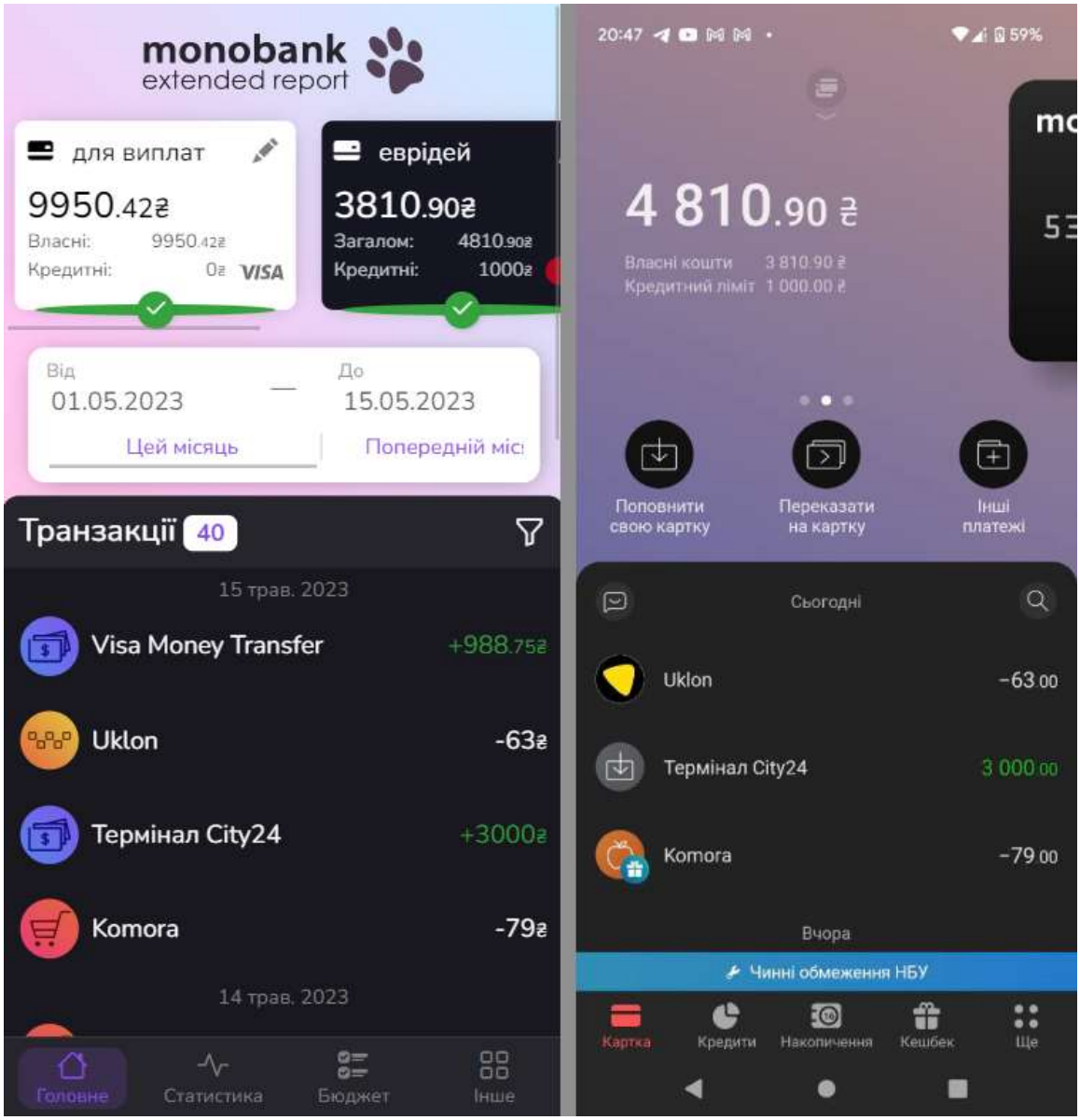


Рисунок 3.6 – Порівняння мобільної версії застосунку Extended Report (зліва) та офіційного мобільного застосунку Monobank (справа)

Для зручного створення діаграм та графіків використовується Javascript-бібліотека react-chart-js-2. Вона використовує технологію HTML5 Canvas для відрисування адаптивних 2D-примітивів з анімаціями. Ця бібліотека не лише підтримує різні види взаємодії, як-от кліки мишкою та дотики пальцем по тачскріну, але й надає зручне API для детального налаштування зовнішнього вигляду діаграм. Приклад того, як виглядає кругова діаграма, створена за допомогою бібліотеки react-chart-js-2 представлено на рисунку 3.7.

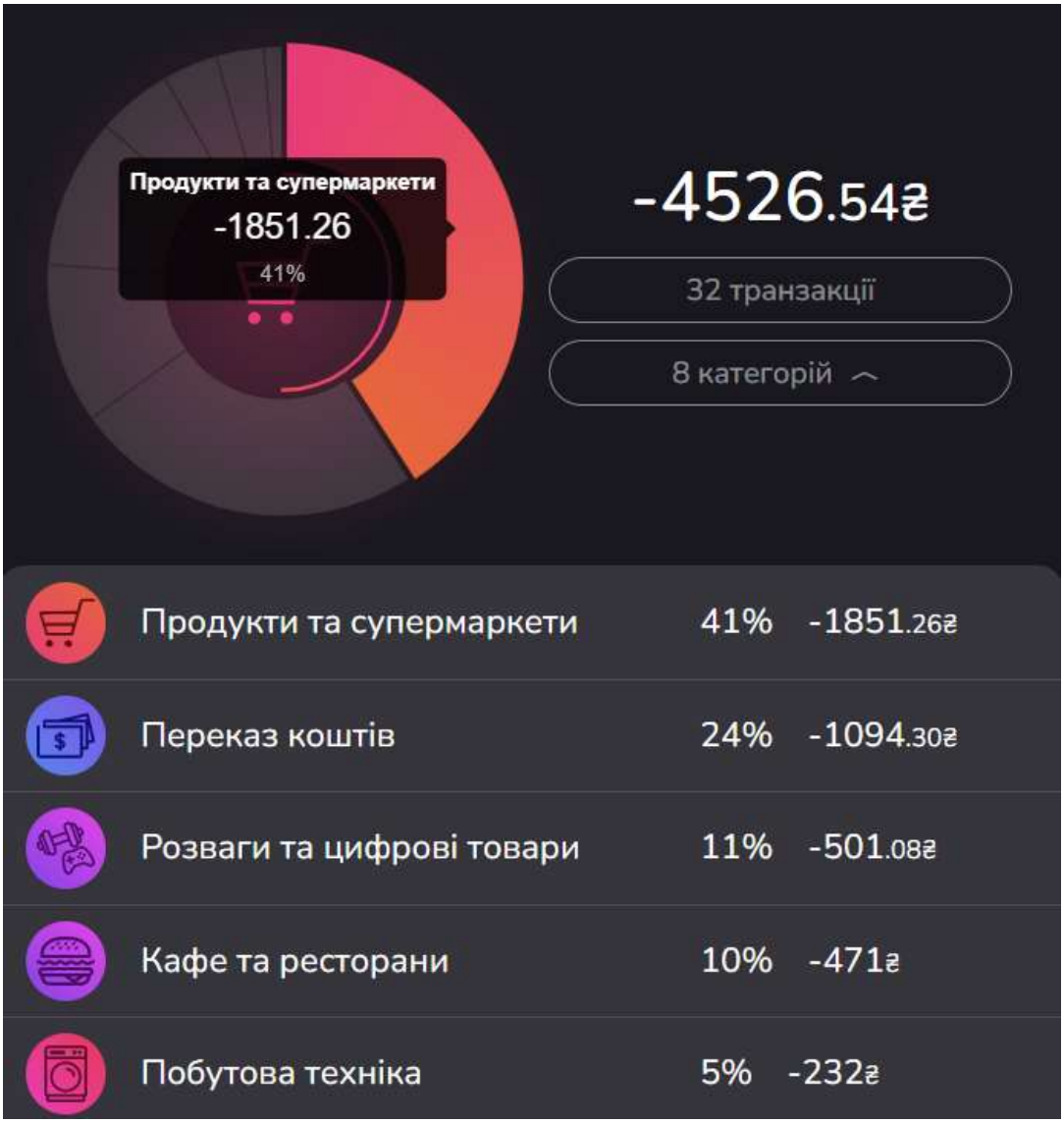


Рисунок 3.7 – Кругова діаграма та відсортований список витрат по категоріям

Для відрисовки лінії тренду використовується плагін chartjs-plugin-trendline, який дозволяє обчислити та відрисувати лінію тренду для відформатованого набору даних. На рисунку 3.8 показано, як у застосунку виглядає лінійний графік, включно з лінією тренду та інтерполяцією проміжних значень за допомогою кубічного сплайна. При цьому графік формується у двох режимах – поденно та помісячно. Помісячний режим доступний лише за умови, що звітній період включає більше одного місяця. Якщо у звітньому періоді немає транзакцій, зроблених у різні дні, то лінійний графік не показуватиметься взагалі.

Рисунок 3.8 – Лінійна діаграма порівнянь витрат та доходів помісячно з лініями тренду

Отже створений графічний користувацький інтерфейс є не лише адаптивним під різні цільові пристрої, але й естетично приємним та, що важливіше, зручним для користувача.

3.3 Тестування застосунку

Тестування MVP базується на кількох принципах, які допомагають перевірити припущення та отримати важливі відгуки від користувачів.

В першу чергу варто дослідити, чи має MVP реалізований функціонал, який потрібен користувачу. Як було описано в параграфі 3.2, користувач очікує бачити наступні функції в MVP: веб-версія та кросплатформа, авторизація, список транзакцій, налаштування бюджету, сповіщення, розширений пошук та незалежність від доступу до сервісів Monobank.

Веб-версія та кросплатформа: завдяки реалізації застосунку як PWA, користувач може переглянути стан власний рахунків, список транзакцій та статистику навіть без доступу до офіційного мобільного застосунку Monobank, за виключенням процедури реєстрації.

Авторизація: застосунок має форму авторизації та верифікації користувача, а також різноманітні механізми шифрування, які захищають дані користувача.

Список транзакцій: як було описано в параграфі 3.2, список транзакцій працює майже ідентично списку транзакцій з офіційного мобільного застосунку Monobank. Єдина відмінність полягає у тому, що в мобільному застосунку Monobank використовується динамічна підгрузка пізніших транзакцій, тоді як в Extended Report список транзакцій підгружається увесь і зразу за обраний період часу.

Налаштування бюджету: користувач має можливість налаштувати свій бюджет по категоріям та загальний. Застосунок в реальному часі відслідковує витрати у звітній період та оновлює стан бюджету, відображаючи його у вигляді набору шкал.

Сповіщення: застосунок відправляє PUSH-сповіщення на кожен підписаний пристрій користувача щоразу як здійснюється транзакція. На жаль, реалізувати сповіщення про вичерпання бюджету реалізувати не вдалось через брак часу.

Розширений пошук: Extended Report дозволяє відфільтрувати транзакції за обраний період часу по умовам, які недоступні в мобільному застосунку Monobank. Наприклад, користувач може знайти всі транзакції по конкретній валюті або за наявністю кешбеку.

Незалежність від доступу до сервісів Monobank: застосунок має механізми кешування інформації та створення резервних копій стану рахунків, завдяки чому користувач має змогу користуватись застосунком навіть якщо сервери Monobank знаходяться у даунтаймі.

Сам застосунок тестується протягом довгого періоду часу. Незважаючи на те, що він створювався для персонального використання, наразі ним активно користуються три людини, а в базі даних зберігається більше тисячі транзакцій.

Що стосується продуктивності застосунку, то тут ще є що покращувати. Наприклад, застосунку після відправки запита на отримання транзакцій за поточний місяць (28 транзакцій) необхідно 420мс щоб отримати відповідь. Переважну кількість цього часу застосунок чекає відповіді від сервера, тоді як сама передача даних займає незначні долі секунди. Подальші оптимізації на стороні

сервера дозволять скоротити цей час і, відповідно, пришвидшити швидкість загрузки даних у застосунку.

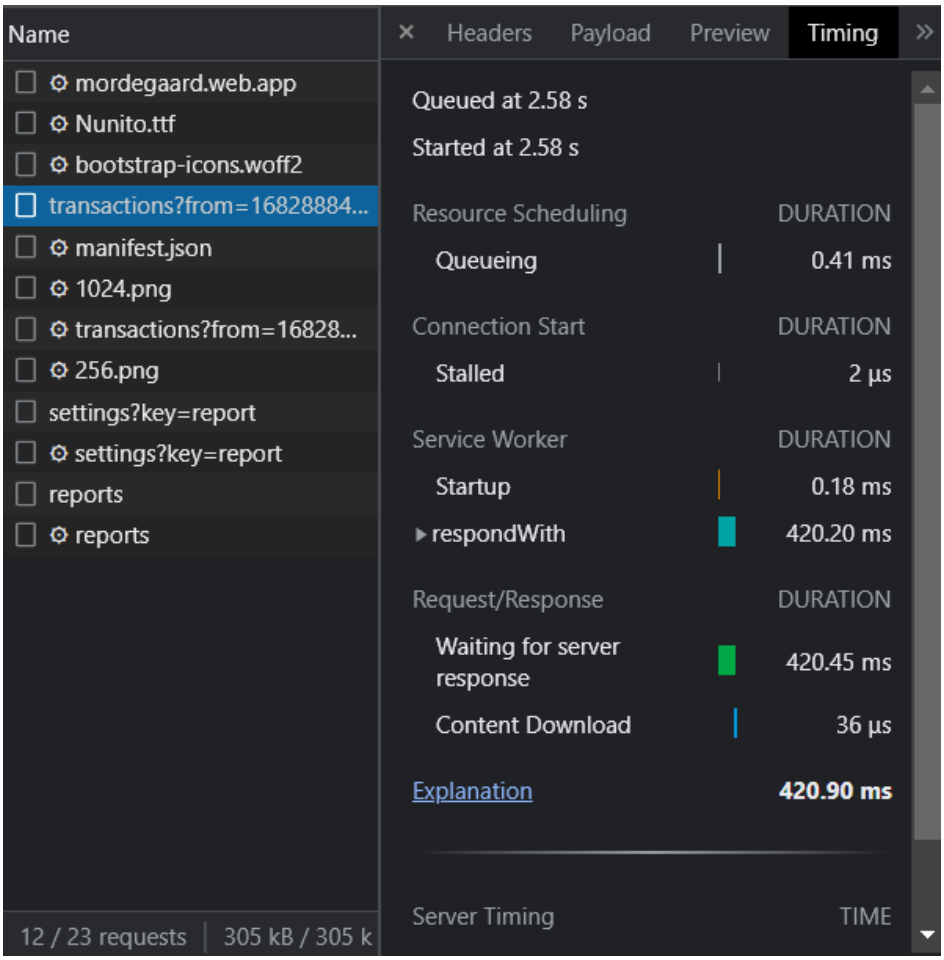


Рисунок 3.9 – Тестування швидкості HTTP-запитів

Розмір застосунку займає приблизно 10Мб, не враховуючи додаткових файлів типу шрифтів та зображень. Завдяки кешуванню, це не сильно впливає на швидкість відкриття застосунку, проте саме кешування може бути проблемою, оскільки без надійного механізму очищення застарілого кешу, розгортання нових функцій може бути проблемним. В цілому, це можна покращити за рахунок розбиття застосунку на окремі частини, які будуть асинхронно завантажуватись в разі потреби. Також обробка кешування в Service Worker дозволить валідувати застарілий кеш та очищувати його.

В результаті можна сказати, що в цілому застосунок виконує всі поставлені задачі, а його технічні обмеження та недоліки є некритичними для користувацького досвіду користувача.

ВИСНОВОК

В процесі виконання дипломної роботи було проаналізовано предметну область ринку фінтех-застосунків та розроблено власний PWA застосунок для моніторингу та контролю грошових транзакцій.

У першому розділі дипломної роботи було детально проаналізовано ринок фінансових технологій у загальному та більш детально досліджено фінтех-застосунки для контролю особистих коштів, у тому числі розглянуто їх основний функціонал та недоліки. Як результат було сформульовано задачу, яка має бути вирішена у процесі реалізації проєкту.

У другому розділі було досліджено методи забезпечення інформаційної безпеки при роботі з чутливими персональними даними користувачів, а також розглянуто методи аналізу транзакцій. В результаті було створено інформаційне підґрунтя для подальшої реалізації проєкту.

У третьому розділі було обрано технологічний стек застосунку, сформовано вимоги до методів комунікації між клієнтом та сервером, продумано UI та UX застосунку, а також протестовано готову реалізацію та виділено поточні недоліки. Сам застосунок можна переглянути та протестувати самостійно за наступною адресою: https://mordegaard.web.app/

Веб-застосунок було спроєктовано, використовуючи сучасні підходи до розробки веб-застосунків, що стосується як вибору технологічного стеку, так і методів забезпечення безпеки даних користувача. При розробці застосунку було використано популярні та надійні бібліотеки та фреймовки, які спростили та пришвидшили процес розробки застосунку, а також забезпечили якісніший функціонал застосунка.

Підводячи підсумки можна сказати, що розроблений MVP застосунку здебільшого повністю задовільняє вимогам користувачів, а незначні технічні недоліки можуть бути виправлені у майбутньому за рахунок гнучкої моделі розповсюдження застосунку. Окрім цього мною було здобуто навички комплексної розробки програмного забезпечення та його розгортання.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Moodle WUNU: “Методичні рекомендації до виконання кваліфікаційної роботи” [Електронний ресурс] – Режим доступу: https://moodle.wunu.edu.ua/pluginfile.php/524478/mod_resource/content/1/Work%281%29.pdf
2. Economy-Pedia: “Fintech – що це таке, визначення та поняття” [Електронний ресурс] – Режим доступу: https://uk.economy-pedia.com/11040127-fintech
3. Національний банк України: “Фінансова грамотність українців поліпшилася” [Електронний ресурс] – Режим доступу: https://bank.gov.ua/ua/news/all/za-ostanni-tri-roki-riven-finansovoyi-gramotnist-ukrayintsiv--polipshivsya--rezultati-doslidjennya
4. Fact.MR: “Personal Finance Mobile App Market Outlook” [Електронний ресурс] – Режим доступу: https://www.factmr.com/report/personal-finance-mobile-app-market
5. Investopedia: “Financial Technology (Fintech): Its Uses and Impact on Our Lives” [Електронний ресурс] – Режим доступу: https://www.investopedia.com/terms/f/fintech.asp
6. Emergen Research: “Progressive Web Application Market By Offering (Services, Tools & Libraries), By Application (Dynamic Web Application, Static Web Application), By End-user (Small- & Medium-sized Enterprises, Large Enterprises), By Industry Vertical (Retail & E-commerce, Media & Entertainment), and By Region Forecast to 2027” [Електронний ресурс] – Режим доступу: https://www.emergenresearch.com/industry-report/progressive-web-application-market
7. Medium: “A Pinterest Progressive Web App Performance Case Study” [Електронний ресурс] – Режим доступу: https://medium.com/dev-channel/a-pinterest-progressive-web-app-performance-case-study-3bd6ed2e6154
8. Web.dev: “Twitter Lite PWA Significantly Increases Engagement and Reduces Data Usage” [Електронний ресурс] – Режим доступу: https://web.dev/twitter/
9. Material Design Guidelines [Електронний ресурс] – Режим доступу: https://m2.material.io/

10. Monobank open API [Електронний ресурс] – Режим доступу: https://api.monobank.ua/docs/
11. RestCase: “4 Most Used REST API Authentication Methods” [Електронний ресурс] – Режим доступу: https://blog.restcase.com/4-most-used-rest-api-authentication-methods/
12. Techtarget: “What is cipher block chaining?” [Електронний ресурс] – Режим доступу: https://www.techtarget.com/searchsecurity/definition/cipher-block-chaining
13. Timing vulnerabilities with CBC-mode symmetric decryption using padding [Електронний ресурс] – Режим доступу: https://learn.microsoft.com/en-us/dotnet/standard/security/vulnerabilities-cbc-mode
14. Wikipedia: “Block cipher mode of operation” [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Block_cipher_mode_of_operation
15. Wikipedia: “HTTPS” [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/HTTPS
16. Apple Developer: “Setting up a remote notification server” [Електронний ресурс] – Режим доступу: https://developer.apple.com/documentation/usernotifications/setting_up_a_remote_notification_server
17. Cloud Firestore Documentation [Електронний ресурс] – Режим доступу: https://firebase.google.com/docs/firestore
18. Wikipedia: “Кубічний сплайн” [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Кубічний_сплайн
19. Pidru4niki: “Регресійний аналіз” [Електронний ресурс] – Режим доступу: https://pidru4niki.com/17280924/ekonomika/regresiyniy_analiz
20. Wikipedia: “Linear regression” [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Linear_regression
21. Wikipedia: “Ordinary least squares” [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Ordinary_least_squares
22. Analytics Vidhya: “Understanding Polynomial Regression Model” [Електронний ресурс] – Режим доступу:

https://www.analyticsvidhya.com/blog/2021/10/understanding-polynomial-regression-model/

23. Wikipedia: “Logistic regression” [Електронний ресурс] – Режим доступу: https://en.wikipedia.org/wiki/Logistic_regression
24. TIBCO: “Fitting a Trend Line for Linearly Dependent Data Values” [Електронний ресурс] – Режим доступу: https://docs.tibco.com/pub/sfire-dsc/6.5.0/doc/html/TIB_sfire-dsc_user-guide/GUID-141E4F08-1478-451C-AAFF-BFF4279E2BBE.html

Додаток А

```
rules_version = '2';

service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth != null
    }

    match /users/{user} {
      allow create: if request.auth != null && if
exists(/databases/$(database)/documents/users/$(request.auth.uid)/data/cards/$(resource
.data.cardId))
    }
  }
}
```

Додаток Б


```
[
{
"mcc":"1761",
"group":{
"type":"CS",
"description":"Ремонт",
"shortDescription":"Будівники. Облицювання"
}
},
{
"mcc":"1771",
"group":{
"type":"CS",
"description":"Ремонт",
"shortDescription":"Будівництво. Бетон"
}
},
{
"mcc":"1799",
"group":{
"type":"CS",
"description":"Ремонт",
"shortDescription":"Спеціалізовані підрядники"
}
},
{
"mcc":"2741",
"group":{
"type":"WSM",
"description":"Книги",
"shortDescription":"Друкарська справа"
}
},
{
"mcc":"2744",
"group":{
"type":"WSM",
"description":"Книги",
```

"shortDescription":"Друкарська справа"
}
},
{
"mcc":"2791",
"group":{
"type":"WSM",
"description":"Книги",
"shortDescription":"Набір текстів та друк"
}
},
{
"mcc":"3000",
"group":{
"type":"CR",
"description":"Подорожі",
"shortDescription":"Авіалінії"
}
},
{
"mcc":"3001",
"group":{
"type":"CR",
"description":"Подорожі",
"shortDescription":"Авіалінії"
}
},
{
"mcc":"3002",
"group":{
"type":"CR",
"description":"Подорожі",
"shortDescription":"Авіалінії"
}
}
]

Додаток В

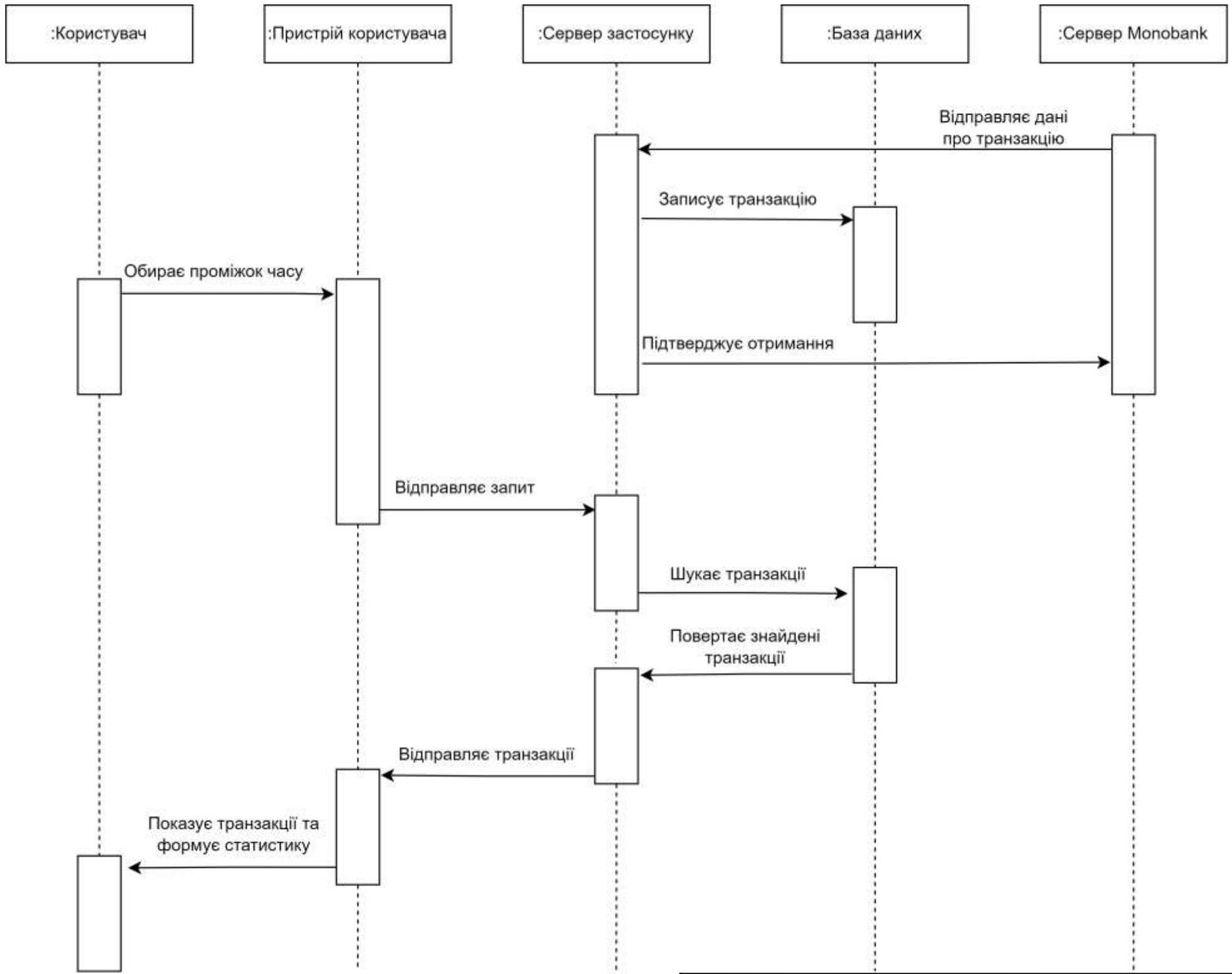
```
export default function InitRoutes (app) {
  app.get(`${InitRoutes.SLUG}/webhook/0yqt2lgVN5kwI6CT`,
WebhookController.init)
  app.post(`${InitRoutes.SLUG}/webhook/0yqt2lgVN5kwI6CT`,
asyncHandler(TransactionsController.create))

  app.post(`${InitRoutes.SLUG}/auth`, asyncHandler(MonobankAuthorization.auth))
  app.get(`${InitRoutes.SLUG}/auth/me`, asyncHandler(MonobankAuthorization.read))

  app.use(`${InitRoutes.SLUG}/transactions`,
asyncHandler(MonobankAuthorization.validate))
  app.get(`${InitRoutes.SLUG}/transactions`,
asyncHandler(TransactionsController.index))
  app.get(`${InitRoutes.SLUG}/transactions/show_many`,
asyncHandler(TransactionsController.showMany))
  app.post(`${InitRoutes.SLUG}/transactions/update_many`,
asyncHandler(TransactionsController.updateMany))
  app.get(`${InitRoutes.SLUG}/transactions/:id`,
asyncHandler(TransactionsController.read))
  app.put(`${InitRoutes.SLUG}/transactions/:id`,
asyncHandler(TransactionsController.update))

  app.use(`${InitRoutes.SLUG}/data`, asyncHandler(MonobankAuthorization.validate))
  app.get(`${InitRoutes.SLUG}/data/:path`, asyncHandler(DataController.index))
  app.post(`${InitRoutes.SLUG}/data/:path/update_many`,
asyncHandler(DataController.updateMany))
  app.put(`${InitRoutes.SLUG}/data/:path/:id`, asyncHandler(DataController.update))

  app.use(`${InitRoutes.SLUG}/settings`,
asyncHandler(MonobankAuthorization.validate))
  app.get(`${InitRoutes.SLUG}/settings`,
asyncHandler(MonobankSettingsController.index))
  app.post(`${InitRoutes.SLUG}/settings`,
asyncHandler(MonobankSettingsController.create))
  app.put(`${InitRoutes.SLUG}/settings`,
asyncHandler(MonobankSettingsController.update))
```

```
  app.use(`${InitRoutes.SLUG}/notifications`,
asyncHandler(MonobankAuthorization.validate))
  app.get(`${InitRoutes.SLUG}/notifications/`,
asyncHandler(NotificationsController.index))
  app.post(`${InitRoutes.SLUG}/notifications/notify`,
asyncHandler(NotificationsController.notify))
  app.post(`${InitRoutes.SLUG}/notifications/:path`,
asyncHandler(NotificationsController.subscribe))
  app.delete(`${InitRoutes.SLUG}/notifications/:path`,
asyncHandler(NotificationsController.unsubscribe))

  app.get(`${InitRoutes.SLUG}/reports/`, asyncHandler(ReportsController.index))
}

InitRoutes.SLUG = '/monobank'
```

Додаток Г

```
// Оголошення класу JsonResource
export default class JsonResource {
  constructor (object) {
    this.object = object
  }

  except (keys) {
    if (!Array.isArray(keys)) keys = [keys]

    keys.forEach(key => delete this.object[key])

    return this.object
  }

  only (keys) {
    if (!Array.isArray(keys)) keys = [keys]

    const result = {}

    keys.forEach(key => result[key] = this.object[key])

    return result
  }

  get () {
    return this.object
  }
}

//Приклад його використання
static async read (req, res) {
  const user = await MonobankAuthorization.me(req)
  user.data = {
    cards: await user.data(MonobankAuthorization.SCOPE, 'cards').retrieve()
  }

  res.send(new JsonResource(user).except([ 'password', 'auth_token', 'scopes' ]))
}
```

Додаток Ґ



					ДП.КН.8091495.001.А1	
					Послідовність обміну даних у застосунку	Літера: — Маса: — Масштаб: —
Змн.	Арк.	№ докум.	Підпис	Дата		Аркуш 1 / Аркушів 1
Розроб.		Рипіч Б.В.				
Перевір.		Ліп'яніна-Гончаренко Х.В.				ЗУНУ.ФКІТ.КН-41
Консультант						
Т. Контр.						
Н. Контр.		Комар М.П.				
Затверд.		Саченко А.О.				

Додаток Д

Log	
1	message: string
2	stack: string
3	time: integer

Transaction	
1	account: string
2	amount: integer
3	balance: integer
4	cashbackAmount: integer
5	commissionRate: integer
6	currencyCode: integer
7	description: string
8	hold: bool
9	id: string
10	mcc: integer
11	operationAmount: integer
12	ignore: bool
13	originalMcc: integer
14	receiptId: string
15	time: integer

User	
1	auth_token: string
2	email: string
3	password: string
4	scopes: array
5	data: collection
6	notifications: collection
7	settings: collection

Settings Document	
1	any: object

Notification Document	
1	id: string
2	subscriptions: object

Monobank Data Document	
1	cards: collection
2	tokens: collection

Token Document	
1	name: string
2	value: string
3	time: integer

Card Document	
1	balance: integer
2	cashbackType: string
3	costType: string
4	creditLimit: integer
5	currencyCode: integer
6	iban: string
7	id: string
8	maskedPan: array
9	name: string
10	sendId: string
11	type: string

Змн.	Арк.	№ докум.	Підпис	Дата	ДП.КН.8091495.002.А1			
Розроб.		Рипіч Б.В.			Структура ієрархічної бази даних застосунку	Літера	Маса	Масштаб
Перевір.		Ліп'яніна-Гончаренко Х.В.						
Консультант						Аркуш 1	Аркушів 1	
Т. Контр.								
Н. Контр.		Комар М.П.				ЗУНУ.ФКІТ.КН-41		
Затверд.		Саченко А.О.						