Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Розробка автоматизованої системи управління транзакціями та керування фінансами»

Виконав: студент 2 курсу, групи 1АКІТР-23м
спеціальності 174 – Автоматизація,
комп'ютерно-інтегровані технології та
робототехніка
Максим ІВАНИШИН
ім'я ПРІЗВИЩЕ
Керівник: к. т. н., доцент, професор кафедри АІІТ
ступінь, звання, посада
Євген ПАЛАМАРЧУК
ім'я ПРІЗВИЩЕ
«16» грудня 2024 р.

Опонент: д. т. н., професор, професор кафедри КСУ
ступінь, звання, посада
Володимир ДУБОВОЙ
ім'я ПРІЗВИЩЕ
«16» грудня 2024 р.

Допущено до захисту
Зав. Кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
«16» 12 2024 р.

м. Вінниця – 2024 рік

Вінницький національний технічний університет
Факультет інтелектуальних інформаційних технологій та автоматизації
Кафедра автоматизації та інтелектуальних інформаційних технологій
Рівень вищої освіти другий (магістерський)
Галузь знань – 17 – Електроніка, автоматизація та електронні комунікації
Спеціальність – 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Освітньо-професійна програма – Інтелектуальні комп'ютерні системи

ЗАТВЕРДЖУЮ
Зав. кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
"19" вересня 2024 року

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ

студенту Іванишину Максиму Руслановичу.
(прізвище, ім'я, по батькові)

1. Тема роботи. Розробка автоматизованої системи управління транзакціями та керування фінансами
керівник роботи к. т. н., доцент, професор кафедри АІІТ Паламарчук Євген Анатолійович
затверджені наказом ВНТУ від "17" вересня 2024 року №310
2. Термін подання студентом роботи "16" грудня 2024 року
3. Вихідні дані до роботи: фінансові транзакції, отримані через API банківських систем, з обсягом обробки до транзакцій, та інформація про користувачів, яка включає до 20 полів для кожного запису. Аналітичні алгоритми обробки даних з точністю прогнозування понад 75%, враховуючи понад 50 фінансових категорій, курси валют у реальному часі та історичні показники за останні 5 років. Система розрахована на забезпечення швидкої обробки транзакцій із середнім часом виконання до 150 мс і підтримує інтеграцію з понад 10 банківськими платформами.
4. Зміст текстової частини: вступ, аналіз предметної області та існуючих систем управління транзакціями та керування фінансами, архітектура проєкту та вибір технологічного стеку, реалізація клієнтської частини та серверної логіки системи управління фінансами, тестування додатку, економічна частина, висновки
5. Перелік ілюстративного матеріалу (з точним зазначенням обов'язкових креслень). Архітектура системи, структура бази даних, UML діаграми, діаграма послідовності, вигляд екранів розробленого додатку.

6. Консультанти розділів магістерської кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	виконання прийняв
1-4	Паламарчук Є.А – к.т.н., доцент, професор кафедри АІІТ	18.09.24	16.12.24
5	Козловський В. О. – к.е.н., проф кафедри ЕПтаВМ	20.09.24	20.12.24

7. Дата видачі завдання 18.09.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва та зміст етапу	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз існуючих систем для управління транзакціями та керування фінансами	до 05.10.2024	вик.
2	Вибір та огляд архітектури та технологій для реалізації проекту	до 20.10.2024	вик.
3	Реалізація клієнтської та серверної частини проекту	до 10.11.2024	вик.
4	Тестування та налагодження системи	до 24.11.2024	вик.
5	Економічна частина	до 27.11.2024	вик.
6	Оформлення матеріалів до захисту МКР	до 01.12.2024	вик.
7	Попередній захист роботи	до 05.12.2024	вик.
8	Остаточний захист роботи	до 20.12.2024	вик.

Студент _____ Максим ІВАНИШИН
(підпис) (*Ім'я ПРІЗВИЩЕ*)

Керівник роботи _____ Євген ПАЛАМАРЧУК
(підпис) (*Ім'я ПРІЗВИЩЕ*)

АНОТАЦІЯ

УДК 004.457

Іванишин М. Р Розробка автоматизованої системи управління транзакціями та контролю фінансів. Магістерська кваліфікаційна робота зі спеціальності 174 - Автоматизація, комп'ютерно-інтегровані технології та робототехніка, освітньо-професійна програма - Інформаційні комп'ютерні системи. Вінниця: ВНТУ, 2024. 134с. Українською мовою, 5 розділів, 43 джерела, рис.:17, табл.: 10

Ця робота присвячена розробці сучасної системи управління фінансами, яка об'єднує інтеграції з банківськими API та можливості штучного інтелекту для аналізу транзакцій. Основна мета – забезпечити користувачів інструментом для ефективного управління фінансами, що включає персоналізовані рекомендації, аналіз фінансових даних, виявлення шахрайства та визначення фінансових цілей. Розробка відповідає сучасним стандартам безпеки, використовуючи комбінацію OAuth 2.0, JWT та TLS, що гарантує конфіденційність, цілісність і доступність даних. Інноваційність рішення полягає у глибокій інтеграції AI-аналітики з банківськими сервісами, що дозволяє запропонувати користувачам новий рівень функціональності та захисту в управлінні фінансами.

Ключові слова: система, управління, стек технологій, безпека, фінанси, керування, Angular, TypeScript, Node.js, UML-діаграми.

ABSTRACT

Ivanyshyn M. R Development of an automated system for transaction management and financial control. Master's qualification work in the specialty 174 - Automation, computer-integrated technologies and robotics, educational and professional program - Information computer systems. Vinnytsia: VNTU, 2024. 134p. In Ukrainian, 5 chapters, 43 sources, fig.:17, tab.: 10

This work is devoted to the development of a modern financial management system that combines integration with banking APIs and the capabilities of artificial intelligence for transaction analysis. The main goal is to provide users with a tool for effective financial management, which includes personalized recommendations, financial data analysis, fraud detection and definition of financial goals. The development complies with modern security standards, using a combination of OAuth 2.0, JWT and TLS, which guarantees confidentiality, integrity and availability of data. The innovation of the solution lies in the deep integration of AI analytics with banking services, which allows us to offer users a new level of functionality and protection in financial management.

Keywords: system, management, technology stack, security, finance, management, Angular, TypeScript, Node.js, UML diagrams.

ЗМІСТ

ВСТУП

У сучасному інформаційному суспільстві, де кількість та різноманітність фінансових транзакцій стрімко зростає, важливість ефективного та надійного управління фінансами стає критичною для приватних осіб та підприємств. З виникненням цифрової економіки та популярності електронних фінансових платформ, розробка автоматизованих систем управління транзакціями та контролю фінансів стає актуальною та важливою задачею.

Актуальність роботи. На ринку фінансових технологій спостерігається значне зростання електронних платежів та операцій з ними. Сучасні користувачі потребують інтеграції різноманітних фінансових сервісів для зручного доступу до актуальних даних про транзакції, баланси та витрати. Це вимагає забезпечення надійного обміну інформацією з різних джерел, таких як банки, фінансові установи та інші системи. Збільшення кількості транзакцій і обсягів даних потребує новітніх технологій для їх обробки та аналізу, що дозволить користувачам ефективно планувати, управляти витратами та інвестиціями, знижуючи ризики помилок чи шахрайства. Враховуючи ці тенденції, автоматизація фінансових процесів, інтеграція з банківськими та фінансовими системами, а також високий рівень безпеки стали необхідними елементами для сучасного фінансового середовища.

Метою магістерської кваліфікаційної роботи є покращення управління фінансовими процесами та підвищення точності й ефективності аналізу фінансових даних шляхом розробки автоматизованої клієнт-серверної системи. Особливу увагу приділено інтеграції з банківськими системами та застосуванню алгоритмів штучного інтелекту для аналізу транзакцій, прогнозування тенденцій і формування персоналізованих рекомендацій, забезпечуючи при цьому високий рівень безпеки й конфіденційності інформації.

Об’єктом дослідження є процеси управління транзакціями та фінансами, які охоплюють такі фінансові операції, як обробка платежів, контроль надходжень і витрат, ведення обліку транзакцій, управління рахунками та

балансами.

Предметом дослідження є обробка та аналіз фінансових транзакцій у сучасних фінансових технологіях. Вивчаються методи інтеграції з банківськими системами для отримання даних про транзакції та використання штучного інтелекту для їх аналізу, класифікації, надання порад та виявлення аномалій. Акцент робиться на автоматизації процесів управління фінансами, що забезпечує точність, ефективність та безпеку транзакцій.

Методи дослідження включають аналіз предметної області та існуючих фінансових систем, що дозволило визначити вимоги до розробки. Використовувались підходи системного аналізу, математичного моделювання фінансових операцій та програмна реалізація із застосуванням сучасних технологій. Для забезпечення якості проводилось функціональне тестування, а також застосовувались алгоритми штучного інтелекту для аналізу транзакцій та виявлення шахрайства.

Новизна отриманих результатів дослідження полягає у створенні автоматизованої системи управління транзакціями та контролю фінансів, яка відрізняється від існуючих рішень інтеграцією з банківськими системами через API для автоматизованого отримання та обробки даних. Система використовує алгоритми штучного інтелекту для аналізу транзакцій, прогнозування фінансових тенденцій і виявлення аномалій або шахрайських дій. Крім того, розробка забезпечує персоналізовані рекомендації для управління фінансами, ґрунтуючись на історичних даних, фінансових категоріях і актуальних валютних курсах.

Практична цінність цієї роботи полягає у створенні оригінальної автоматизованої системи для ефективного управління фінансовими транзакціями. Завдяки автоматизації процесів, система мінімізує ризик людських помилок і дасть змогу оперативно реагувати на зміни в особистих фінансах. Інтеграція з різними фінансовими установами та системами дасть змогу мати єдину точку доступу до різноманітних фінансових послуг. Система має зрозумілий та простий інтерфейс з оптимізованими UX-функціями.

Апробація та публікації матеріалів дослідження. Результати роботи було представлено на LIII Науково-технічній конференції підрозділів Вінницького національного технічного університету (2024). [1]

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ СИСТЕМ УПРАВЛІННЯ ТРАНЗАКЦІЯМИ ТА КЕРУВАННЯ ФІНАНСАМИ

1.1 Аналіз предметної області

В сучасному світі, де фінансові транзакції стають все більш складними та розгалуженими, створення автоматизованих систем для управління транзакціями та керування фінансами набуває важливого значення. Ефективне фінансове управління є фундаментальною складовою успішної діяльності як фінансових установ, так і корпорацій. Зростання обсягів фінансових транзакцій, впровадження новітніх технологій, таких як блокчейн та штучний інтелект і збільшення вимог до звітності та контролю фінансів вимагають розробки інноваційних систем, що забезпечують точність, безпеку та ефективність [1].

1.1.1 Вимоги користувача

Клієнти, які використовують поточні фінансові програми, очікують, що система допоможе їм зручно розпоряджатися грошима. Це передбачає перегляд ретельного обліку транзакцій, оцінку витрат за категоріями, автоматичне оновлення даних балансу, використання аналітики на основі штучного інтелекту для прогнозування витрат і виявлення аномалій, а також зв’язування з різними платіжними банками або службами. Зручність для користувача має вирішальне значення: інтерфейс має бути зрозумілим і миттєво реагувати на введення користувача.

Безпека даних дуже важлива, тому користувачі очікують двофакторної автентифікації, засобів запобігання проти шахрайства та найсучасніших методів шифрування для обміну та захисту особистих даних. Крім того, надзвичайно важливо дозволити вхід до програми з будь-якого

гаджета, включно зі смартфонами, які вимагають високої ефективності та швидкої обробки даних навіть під час пікового використання.

Потреби користувачів у фінансовій системі, яка включає в себе безпеку, надійність, креативність і просту у використанні, як основу для створення довіри до програми [2].

1.1.2 Роль технологій у фінансах

Технології відіграють велику роль у трансформації фінансового сектору, гарантуючи швидкі, ефективні та доступні фінансові послуги. Інновації, такі як штучний інтелект і блокчейн, значно покращили процеси обробки даних і забезпечення безпеки транзакцій. Штучний інтелект автоматизує обробку великих обсягів даних, що зменшує людські помилки, підвищує точність та відкриває нові інвестиційні можливості, тоді як блокчейн забезпечує прозорість та незмінність записів, створюючи безпечні фінансові транзакції.

Цифрові методи оплати, мобільні додатки та інтернет-банкінг оптимізують фінансові операції, дозволяючи користувачам швидко здійснювати платежі без необхідності фізичного контакту. Вони також забезпечують підвищену безпеку завдяки криптографії та біометричним технологіям, що ще більше підвищує довіру до фінансових послуг. Технології також сприяють фінансовій інклюзивності, надаючи доступ до фінансових послуг людям в віддалених регіонах або тим, хто має обмежений доступ до традиційних банківських установ.

Всі ці технології не тільки роблять фінансові послуги більш персоналізованими, але й забезпечують їх доступність для більш широкого кола користувачів, оптимізуючи процеси, підвищуючи безпеку та надаючи нові можливості для інвестування та збереження коштів [2].

1.1.3 Існуючі рішення та недоліки

Зараз у фінансовій індустрії використовується багато технологічних рішень для покращення фінансового менеджменту, обробки платежів і захисту транзакцій. Приклади включають мобільні банківські додатки, електронні платіжні системи, технологію блокчейн і аналітику великих даних.

Користувач може використовувати мобільні банківські додатки для проведення транзакцій, перегляду залишків на рахунках і за потреби обробляти свої гроші. Сервіси онлайн-платежів, такі як PayPal, Stripe або Apple Pay, забезпечують швидкий і простий спосіб здійснення покупок в Інтернеті, особливо під час онлайн-покупок.

Blockchain пропонує підвищену безпеку та прозорість фінансових транзакцій завдяки своїй децентралізованій конструкції, яка є важливою для крипто валют та смарт-контрактів, що дозволяє автоматизувати процеси, знижуючи витрати, пов’язані з посередницькими послугами.

Тим не менш, хоча ці технології пропонують переваги, вони мають і недоліки. Основна проблема — це безпека, оскільки навіть оновлені системи можуть бути схильні до злому з боку хакерів або шахрайських дій. Цей ризик зростає, коли ви маєте справу з неналежним чином захищеними платіжними каналами або нерегульованим зберіганням особистої інформації. Крім того, доступність залишається проблемою для людей у віддалених регіонах, які не мають доступу до Інтернету та смартфонів.

Виклик також виникає через поєднання різних систем на одній фінансовій платформі. Численні фінансові установи працюють із застарілими системами, що створює проблеми інтеграції з новими технологіями, перешкоджаючи зусиллям з модернізації та знижуючи ефективність процесів.

Наявні на даний момент рішення є багатообіцяючими, але потребують удосконалень, щоб зменшити ризик, підвищити зручність і гарантувати ширший доступ для різних груп користувачів [3].

1.1.4 Напрямки розвитку фінансових систем

Фінансові системи розвиваються на фоні стрімкого технологічного прогресу, що вимагає постійного пристосування до їх швидких змін. Основна увага — оцифрування та автоматизація фінансових процедур, використання таких технологій, як штучний інтелект і машинне навчання, для підвищення ефективності управління транзакціями та аналізу фінансових даних. Ці інновації дозволяють надавати індивідуальні рекомендації користувачам, прогнозувати ризики та оптимізувати фінансові операції [4].

Важливим аспектом є розвиток технології блокчейн. Підвищує прозорість і безпеку фінансових операцій, зменшуючи потребу в таких посередниках, як банки. Це призводить до скорочення витрат і пришвидшення транзакцій, особливо для глобальних переказів. Крім того, технології децентралізації можуть революціонізувати фінансову взаємодію, пропонуючи користувачам більше незалежності та простоти використання [5].

Об’єднання фінансових технологій з мобільними платформами та Інтернетом речей є важливою сферою прогресу. Це полегшує людям проводити грошові операції за допомогою мобільних додатків, цифрових гаманців і подібних інструментів у будь-якому місці та в будь-який час. Ці технології забезпечують автоматизацію за допомогою гаджетів для збору даних, відкриваючи нові можливості для розумного фінансового вибору.

Розвиток включає інклюзивність фінансових послуг. Складні технології створюють можливості для залучення нових користувачів, як-от тих, хто має обмежені економічні знання або мешкають у відокремлених місцях, надаючи їм фінансові послуги через мобільні програми або онлайн-канали. Сучасні фінансові рішення, такі як платформи Open Banking і FinTech, доступні для малих і середніх підприємств, спрощуючи фінансові операції та доступність кредитів.

Крім того, сучасні методи оплати, такі як цифрові валюти та цифрові валюти центральних банків, відкривають нові можливості для швидших і

економічно ефективніших глобальних платежів. Ці шляхи допомагають скоротити витрати, підвищити безпеку та забезпечити гнучкість фінансових операцій, що зрештою відкриває нові перспективи як для окремих осіб, так і для компаній [6].

1.2 Аналіз об’єкта автоматизації

Об’єктом автоматизації в цьому проекті є система управління фінансовими транзакціями, яка підключається до банківських рахунків, обробляє дані про витрати, доходи та аналізує фінансові операції. Дозволяє зберігати та перевіряти дані транзакцій, пропонуючи користувачам зручні інструменти для моніторингу фінансової діяльності.

Система обробляє дані з банківських API, стандартизує їх для узгодженості та зберігає в базі даних для аналізу. За допомогою оброблених даних можна передбачити фінансові тенденції, класифікувати транзакції та виявити аномалії.

Крім того, система може з’єднуватися зі службами штучного інтелекту, щоб перевіряти текстові описи транзакцій, створювати моделі для виявлення шахрайських транзакцій або надавати персональні поради що до грошових витрат. Він пропонує користувачам практичний і ефективний спосіб контролю за своїми фінансами [7].

1.2.1 Функціональність системи

Фінансові системи для управління грошима пропонують користувачам багато функцій, які роблять роботу з фінансами легшою. Головна функція таких систем охоплює можливість слідкування за витратами та доходами, переглядає історію угод, а також створювати детальні звіти про гроші. Завдяки поєднанню з

банківськими системами користувачі можуть отримувати останню інформацію про залишок на рахунках і статус платежів в реальному часі.

Додатки дають змогу автоматично розділяти транзакції, що допомагає краще бачити, як витрачаються гроші. Це дуже корисно для планування бюджету, адже люди можуть ставити межі для різних груп витрат а система попередить про їх перевищення. Ще доступ до інструментів для фінансового прогнозу допомагає оцінити можливі зміни в даній фінансовій ситуації.

Тим не менш, вони мають кілька недоліків. Банківські та інші системи час від часу застосовують складні стандарти безпеки та правила, що створює ускладнення. Подібним чином обробка численних транзакцій може призвести до сповільнення часу відповіді, що вплине на зручність користувача. Проблеми можуть виникнути, якщо життєво важливі служби не працюють належним чином.

Сучасні структури мають на меті забезпечити людей ресурсами для безперебійного управління фінансами за допомогою інтеграції аналітичних можливостей, завдань автоматизації та мінімізації ризики втрати або переривання даних [8].

1.2.2 Заходи безпеки та інтеграція

Забезпечення безпеки є важливим аспектом в системах, що обробляють фінансові дані. Для захисту інформації користувачів використовуються нові методи шифрування при передачі даних між клієнтом і сервером а також коли зберігаються на сервері. Протокол TLS забезпечує безпечний обмін інформацією, а особисті дані, такі як токени доступу та паролі надійно шифруються. Авторизація користувачів відбувається через двофакторну автентифікацію за допомогою електронної пошти Google або Apple ID, так само

й через біометричні способи як Face ID і відбиток пальця, що підвищує рівень захисту і знижує ризик несанкціонованого доступу.

Об’єднання з банківськими системами є важливим елементом для надання доступу користувачам до їх рахунків і угод. Використання сучасних банківських API дозволяє отримувати свіжі дані про фінансові дії, залишки та інші показники. Але, узгодження роботи з різними банками стає складним через відмінності стандартів та протоколів, які використовують фінансові установи. Це потребує створення гнучкої системи, яка може добре пристосовуватися до особливостей кожного API.

Особливу увагу було звернуто на захист від кібератак і обманів. Для цього введено додаткові перевірки запитів, обмеження доступу згідно з ролями користувачів, і використання нових технологій верифікації. Оптимізація роботи баз даних та зменшення затримок у відповіді системи здійснюються через використання механізмів кешування, які прискорюють обробку запитів та забезпечують стабільну роботу навіть при високому навантажені.

Головними проблемами є різні вимоги банківських API, що ускладнює стандартизацію з'єднання. Також є можливі небезпеки безпеки, включаючи фішинг і спроби підроблення токенів доступу. Крім цього система залежить від стабільного інтернет з’єднання, а також наявності банківських серверів, що може впливати на роботу за збоїв.

Дотримання світових норм для приватності, як-от GDPR і CCPA, є дуже важливим для ясності та охорони особистих даних. Впровадження цих кроків допомагає створити система, яка відповідає високим стандартам безпеки; дає зручний та надійний доступ до грошей та інформації, й задовольняє потреби сучасних користувачів у гнучкому управлінні їх фінансами [9].

1.2.3 Гнучкість та персоналізація

На сьогодні фінансові системи мають відповідати низці потреб які включають в себе гнучкість та персоналізацію. Це дозволяє користувачу адаптувати інтерфейс під свої потреби та вподобання. Адаптація зовнішнього вигляду та функцій, являється теж ключовим аспектом до вимог. За звичай це включає в себе вибір мови, теми, керування сповіщеннями й налаштування віджетів та макету. Також функції мають адаптуватися відносно середовища, де використовується додаток, наприклад якщо це веб версія потрібно йде використання функцій.

Гнучкість і адаптація до індивідуальних потреб життєво важливі для впровадження ефективних фінансових систем сьогодні. Вони дозволяють користувачам персоналізувати інтерфейс відповідно до своїх конкретних уподобань, покращуючи їх взаємодію з програмами, пов'язаними з грошима, і зміцнюючи ефективне спілкування з фінансовими інструментами.

Налаштування зовнішнього вигляду та функцій програми відповідно до потреб користувачів є ключовою частиною гнучкості. Це передбачає налаштування вибору діалекту, оформлення вікон, а також контроль сповіщень і вихідних даних грошових угод, що полегшує зручне індивідуальне налаштування. Адаптація в контексті грошових систем дає можливість адаптувати фінансові послуги до фінансових цілей і методів фінансового менеджменту особи.

Аналізуючи попередні покупки, моделі витрат і доходів, а також зовнішні умови (наприклад, геолокцію чи економічні витрати), платформа може запропонувати індивідуальні поради щодо заощаджень, інвестування чи управління фінансами.

Завдяки інтеграції з технологіями штучного інтелекту (AI) і машинного навчання (ML) фінансові платформи можуть постійно налаштовувати свої поради, враховуючи динамічні фінансові обставини користувача. Це спряє можливостям для надання не лише звичайних грошових інструментів, але й для

сприяння окремим особам розрізняти найбільш ефективну тактику при досягненні своєї грошової мети.

Отже, адаптивність і налаштування дозволяють отримувати доступ до індивідуальних фінансових послуг, підвищуючи задоволення від використання та допомагаючи в досягненні поставлених постійних фінансових цілей [10].

1.2.4 Інтеграція з штучним інтелектом

Інтеграція штучного інтелекту в фінансові програми значно покращує аналітичні можливості, надаючи точні результати та персоналізовані рекомендації. Завдяки AI, система може ефективно обробляти великі обсяги фінансових даних, аналізувати структуру витрат і прогнозувати фінансову поведінку клієнтів, що дозволяє краще планувати доходи, витрати та інвестиції.

AI також автоматично виявляє аномалії в транзакціях, такі як незвичайні витрати або операції в нестандартних місцях чи в невідповідні періоди, що значно підвищує рівень безпеки. Він допомагає класифікувати платежі, полегшуючи фінансові операції та дозволяючи створювати детальні звіти про витрати.

Ще однією важливою функцією є прогнозування фінансових результатів, зокрема, моделювання сценаріїв та надання рекомендацій щодо витрат і заощаджень, що дає користувачам можливість приймати обґрунтовані рішення та планувати своє майбутнє.

Однак для успішного впровадження AI необхідно вирішити кілька технічних завдань, таких як захист приватних даних, оскільки обробка значних фінансових відомостей вимагає високого рівня безпеки та ефективності. Крім того, для розвитку таких систем можуть знадобитися потужні обчислювальні ресурси, що вимагає належної інфраструктури.

Загалом програми штучного інтелекту у фінансах сприяють використанню винахідливих методів, які підвищують аналітичну точність, спрощують фіскальне адміністрування та підвищують безпеку користувачів [11].

1.3 Аналіз існуючих систем для автоматизації управління транзакціями та керування фінансами

Аналіз існуючих систем для автоматизації управління транзакціями та керування фінансами, демонструє широкий спектр доступних рішень, які спрямовані на спрощення взаємодії користувачів із фінансовими послугами. Ці системи забезпечують автоматизацію обліку витрат, аналіз транзакцій та допомагають користувачам приймати обґрунтовані фінансові рішення.

Багато сучасних додатків, таких як наприклад Mint, YNAB (You Need A Budget) пропонують функціонал для управління персональними фінансами. Вони інтегруються з банківськими рахунками, автоматично імпортують транзакції та класифікують їх за категоріями. Це дозволяє користувачам бачити свої витрати у зрозумілому форматі, налаштовувати бюджети та отримувати аналітичні дані.

Основні недоліки існуючих систем включають обмежену підтримку локальних банків та різноманітність стандартів API, що ускладнює інтеграцію з фінансовими установами. Ще одним викликом є конфіденційність та безпека даних, оскільки передача та обробка фінансової інформації потребує дотримання суворих стандартів захисту. Деякі програми також обмежені в функціональності через прив'язку до конкретних ринків або валют [12, 13].

Незважаючи на це, розвиток технологій, таких як Open Banking, значно розширює можливості інтеграції з банківськими системами. Завдяки стандартизації протоколів обміну даними стає можливим створення більш універсальних платформ. Водночас все більше компаній впроваджують

алгоритми аналізу транзакцій для виявлення шахрайства, що покращує безпеку користувачів.

Загалом, існуючі системи демонструють значний прогрес у сфері автоматизації управління фінансами, проте залишаються можливості для вдосконалення, такі як забезпечення глобальної інтеграції, підвищення безпеки та розширення функціоналу для користувачів [14].

Тому пропонується розробка нової системи під назвою Wallet, яка спрямована на подолання існуючих недоліків і забезпечення користувачів більш зручним, безпечним і функціональним інструментом для управління фінансами. Система пропонуватиме уніфіковану інтеграцію з банківськими установами незалежно від їхніх стандартів API, забезпечуючи підтримку локальних та міжнародних фінансових інституцій. Крім того, Wallet буде орієнтована на вдосконалений захист персональних даних, відповідність міжнародним стандартам конфіденційності та ефективну автоматизацію обліку витрат і доходів. Завдяки використанню сучасних технологій аналізу транзакцій, Wallet надаватиме користувачам персоналізовані рекомендації та інструменти для оптимального управління своїми фінансами.

Таблиця 1.1 – порівняння додатків

Характеристика	Mint	YNAB	Wallet
Основна Мета	Відстеження витрат та керування бюджетом.	Бюджетування та планування	Управління транзакціями та фінансовий аналіз.
Інтеграція ШІ	Відсутня	Відсутня	Присутня
Аналіз Витрат ШІ	Немає	Немає	Присутні
Персоналізовані Рекомендації	Немає	Немає	Присутні
Планування Бюджету	Немає	Так	Так

Продовження таблиці 1.1

Різноманітність Категорій	Обмежена	Значна	Гнучка
Різноманітність Дебторів	Немає	Немає	Так
Підходить для Фізичних Осіб	Так	Так	Так
Підходить для Компаній	Ні	Ні	Так

1.4 Висновок до розділу

У цьому розділі магістерської кваліфікаційної роботи проведено аналіз предметної області систем автоматизації управління транзакціями та керування фінансами. Розглянуто призначення, основні цілі та завдання майбутньої розробки, а також визначено ключові функціональні можливості, які повинна забезпечувати система для задоволення потреб користувачів.

Проведений аналіз наявних літературних і онлайн-джерел дозволив ідентифікувати основні тенденції розвитку систем автоматизації фінансових процесів та виявити недоліки сучасних рішень, що обмежують їх ефективність. Зокрема, встановлено зростаючий попит на інтеграцію сучасних технологій для оптимізації фінансового планування, підвищення прозорості операцій і зменшення ризиків.

Цей аналіз закладає основу для вибору ефективної архітектури системи та програмних інструментів, які відповідатимуть потребам користувачів і забезпечать високий рівень функціональності, надійності та безпеки.

2 АРХІТЕКТУРА ПРОЄКТУ ТА ВИБІР ТЕХНОЛОГІЧНОГО СТЕКУ

2.1 Визначення архітектури системи та її складових

Архітектура системи є основою для забезпечення її ефективності, масштабованості та безпеки. Вибір правильної архітектури визначає здатність системи адаптуватися до зростаючих навантажень і змінюваних вимог, зокрема до інтеграції з іншими сервісами і забезпечення стабільної роботи при високому обсязі даних. Система повинна бути здатною працювати з великими обсягами транзакцій і інтегруватися з різними банківськими API та іншими фінансовими платформами, що вимагає її адаптивності до змінних умов. Загальна схема додатку наведена нижче (рис. 2.1)

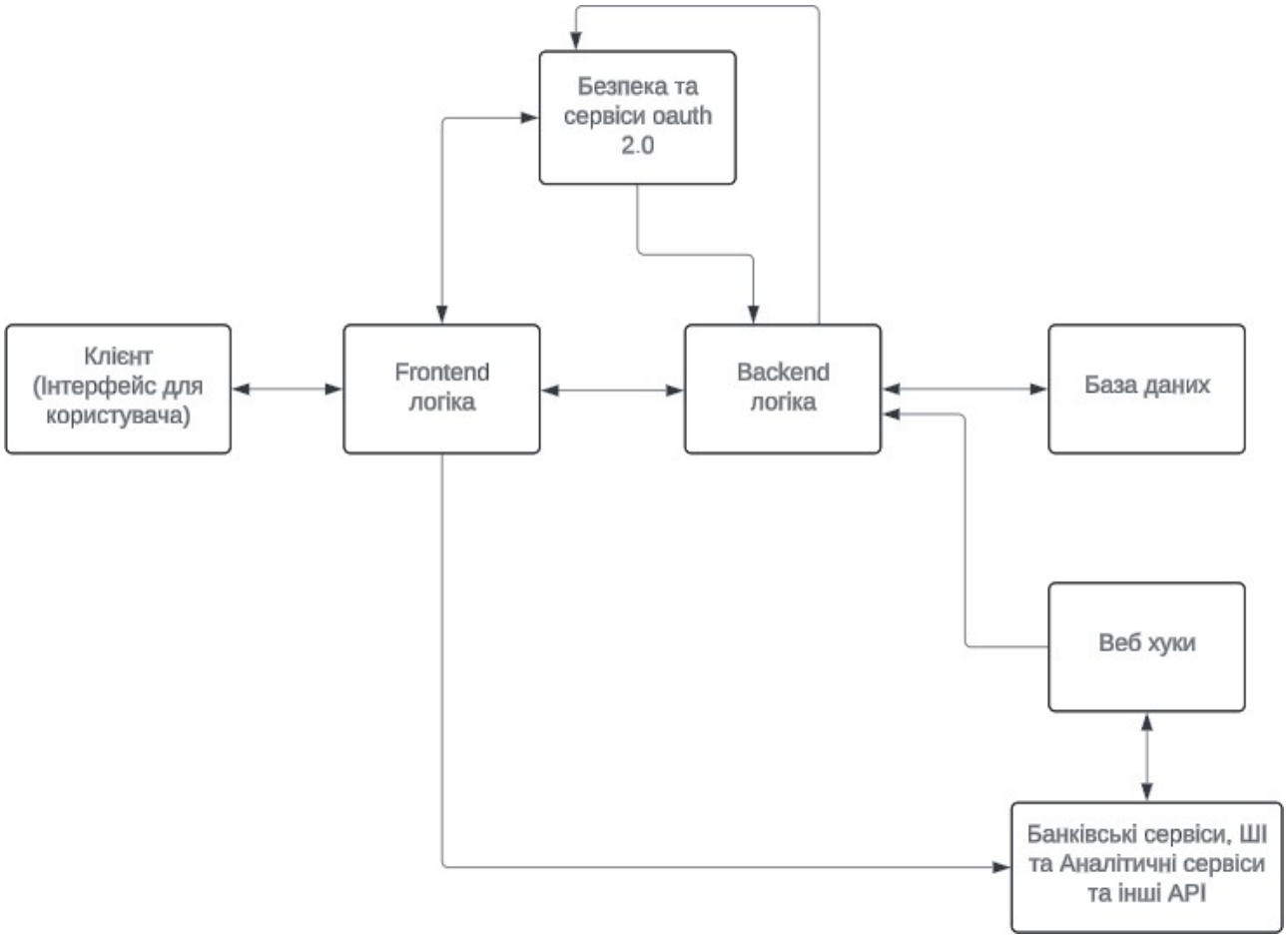

Рисунок 2.1 – Архітектурна схема додатку

Для системи автоматизації управління транзакціями та керування фінансами важливо забезпечити модульну архітектуру (рис. 2.2), що дозволить розподілити навантаження на окремі компоненти. Модульний підхід дозволяє підвищити ефективність і гнучкість, а також спрощує оновлення та модернізацію системи без потреби переписувати всю структуру. Інтеграція з банківськими API та іншими фінансовими сервісами вимагає забезпечення стандартизованого доступу до даних, що дозволить знизити складність роботи з різними фінансовими установами.

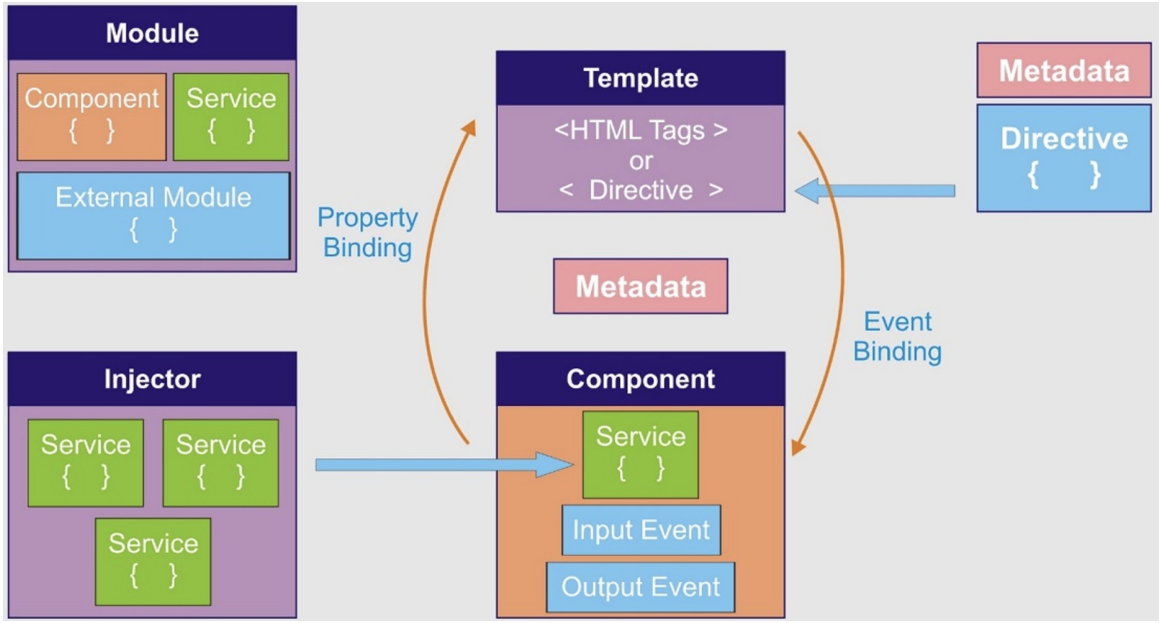

Рисунок 2.2 – Приклад модульної архітектури

Безпека є важливим аспектом для такої системи, оскільки вона обробляє чутливу фінансову інформацію. Застосування шифрування під час передачі та зберігання даних, а також механізмів двофакторної автентифікації і біометричних методів, значно підвищує захист від несанкціонованого доступу і шахрайства. Оскільки система буде працювати з великими обсягами даних і фінансовими операціями в реальному часі, важливо забезпечити швидкість обробки даних і мінімізацію затримок, що вимагає ефективної роботи з базами даних і системами кешування.

З урахуванням цих вимог, обрано архітектуру мікросервісів (рис. 2.3), яка дозволяє створити розподілену, гнучку і масштабовану систему. У цьому підході кожен мікросервіс відповідає за конкретну функціональність, що дає можливість незалежно оновлювати або змінювати окремі частини системи без впливу на інші компоненти. Така архітектура також дозволяє легше впроваджувати нові функції, інтегрувати різноманітні банківські API, а також масштабувати систему для обробки великих обсягів даних. Мікросервіси можуть бути написані різними мовами програмування, що дає можливість вибору найбільш підходящих технологій для кожного компоненту.

Загалом, архітектура мікросервісів відповідає вимогам до безпеки, масштабованості та ефективної інтеграції з зовнішніми сервісами, а також забезпечує можливість адаптації до змін, що можуть виникнути в майбутньому [15].

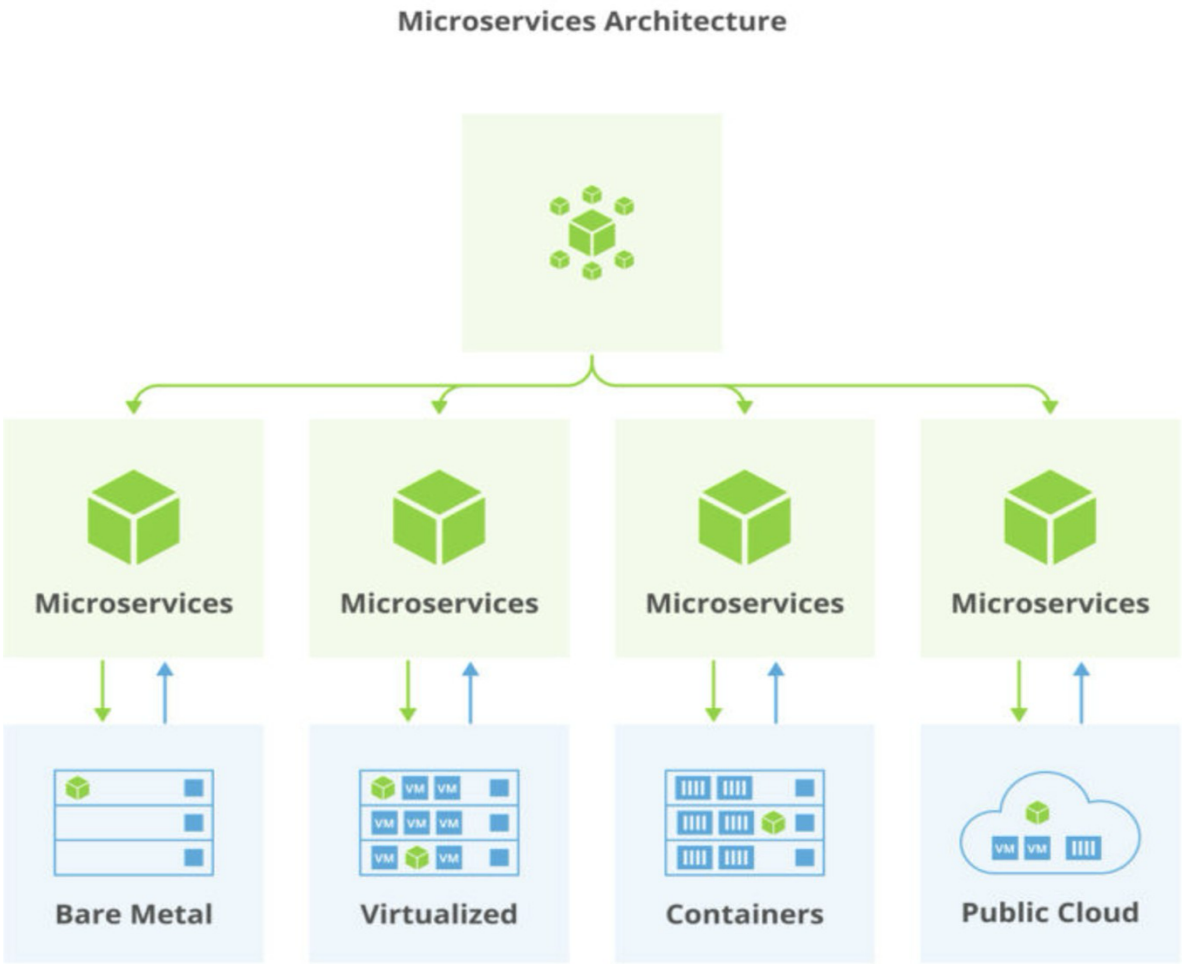

Рисунок 2.3 – приклад мікросервісної архітектури

2.2 Обґрунтування вибору технологічного стеку

2.2.1 Вибір мови програмування для клієнтської частини

Для клієнтської частини системи нам потрібно було забезпечити декілька ключових характеристик від мови програмування. По-перше, висока продуктивність і швидкість виконання операцій, оскільки система повинна обробляти великі обсяги фінансових даних у реальному часі. По-друге, підтримка статичної типізації для зменшення ймовірності помилок на етапі компіляції, що є критичним для точності фінансових операцій. Окрім цього, важлива підтримка сучасних стандартів розробки та сумісність з великою кількістю бібліотек і фреймворків, що дає змогу швидко інтегрувати додаткові функції.

Під час вибору мови програмування для розробки клієнтської частини програмних інструментів для додатку керування та управління фінансами, було розглянуто два основних варінти JavaScript та TypeScript.

JavaScript є широко використовуваною мовою програмування, яка підтримується браузерами та забезпечує динамічні й інтерактивні можливості для розробки клієнтських додатків. Завдяки великій кількості фреймворків і бібліотек, таких як React, Angular і Vue.js, JavaScript дозволяє швидко створювати функціональні клієнтські інтерфейси, що є важливим для проєктів, зокрема в галузі фінансів.

Однією з ключових переваг JavaScript є його простота у вивченні та широке використання, що робить його ідеальним для розробки фінансових додатків, оскільки він легко інтегрується з іншими веб-технологіями, такими як HTML і CSS. Це дає можливість ефективно працювати з DOM-елементами, обробляти події користувача та здійснювати асинхронні запити на сервер, що є необхідним для реалізації функціональностей, таких як обробка транзакцій та динамічне відображення фінансових даних.

JavaScript також має високу переносимість, що дозволяє розробляти універсальні рішення, які працюють на різних пристроях і платформах. Це особливо важливо для фінансових додатків, які мають працювати як на веб-версії, так і на мобільних пристроях, забезпечуючи безперервний доступ до даних і функцій. Завдяки своїй швидкості та доступності, JavaScript дозволяє розробляти проекти ефективно та з мінімальними витратами часу на налагодження та відлагодження, що є критичним для бізнесу, який працює у швидко змінному середовищі [16].

TypeScript є надбудовою над JavaScript, що додає статичну типізацію та розширює можливості розробки програмного забезпечення. Використання TypeScript дозволяє виявляти помилки на етапі компіляції, покращує читабельність коду і робить програмне забезпечення більш надійним.

Ключова перевага TypeScript — це статична типізація, яка дає змогу виявляти помилки під час компіляції, що дозволяє швидше їх виправляти і підвищує надійність коду. Окрім того, TypeScript підтримує орієнтоване об'єктне програмування, що дозволяє писати більш структурований код завдяки підтримці класів, спадкування та інтерфейсів.

TypeScript має великий набір інструментів і розширень, таких як TypeScript Compiler (tsc), редактори коду з автодоповненням та виявленням помилок, а також підтримку популярних фреймворків, зокрема Angular та NestJS. Він також підтримує сучасні функції ECMAScript, як async/await та деструктуризацію, що дає змогу використовувати нові можливості JavaScript навіть без повної підтримки браузерами.

Інтеграція TypeScript з існуючим JavaScript кодом також є простою. Він дозволяє поступово додавати типи до проектів, що вже використовують JavaScript, надаючи можливість поетапного переходу на TypeScript. Це робить його зручним для великих проектів і команд [17].

Порівняльна таблиця JavaScript та TypeScript наведена нижче:

Таблиця 2.1 – порівняння JavaScript та TypeScript

Критерій	**JavaScript**	**TypeScript**
Типізація	Динамічна типізація	Статична типізація
Сумісність	Стандарт мови для веб-розробки	Суперсет JavaScript
Інструменти	Обмежена підтримка	Розширені можливості розробки
Читабельність	Залежить від розробника	Покращена читабельність коду
Бібліотеки та фреймворки	Широкий вибір бібліотек та фреймворків	Сумісні з JavaScript бібліотеки та фреймворки
Складність	Простота вивчення	Додаткова складність через типізацію та строгість
Застосування	Швидкі прототипи та невеликі проекти	Великі проекти, командна розробка, більша надійність

У результаті аналізу, було прийнято рішення використовувати TypeScript для розробки клієнтської частини додатку керування та управління фінансами. TypeScript допомагає зменшити кількість помилок в коді, полегшує рефакторинг та поліпшує підтримку IDE. Він забезпечує сильну типізацію, що дозволяє виявляти помилки на ранніх етапах розробки і зробити код більш надійним та підтримуваним.

Також для розробки клієнтської частини буде використана мова розмітки HTML, яка використовується для створення структури та відображення веб-сторінок. Вона використовується для визначення структури документа та його вмісту, включаючи текстовий контент, зображення, посилання, таблиці, форми та інші елементи [18].

Також для стилізації інтерфейсу буде використана бібліотека Tailwind CSS - це сучасний CSS-фреймворк, який надає набір готових класів та інструментів,

спроектованих для швидкої і простої розробки веб-інтерфейсів. Замість написання власних CSS стилів, Tailwind CSS пропонує вже готові рішення для стилізації елементів, які також можна редагувати для власних потреб [19].

2.2.2 Вибір мови програмування для серверної частини

Розробка серверної частини фінансового додатку вимагає обґрунтованого вибору мови програмування, яка забезпечить високу продуктивність, масштабованість і надійність. Серед популярних варіантів можна розглянути Python, Java, Ruby та Node.js, кожна з яких має свої переваги та недоліки.

Python вирізняється простотою, великою кількістю бібліотек і зручністю у використанні для аналітики та роботи з даними. Вона добре підходить для задач штучного інтелекту й аналізу, однак у контексті обробки великих обсягів запитів у реальному часі її продуктивність може бути обмеженням [20].

Java є відмінним вибором для масштабованих і надійних фінансових систем завдяки її продуктивності, багатопотоковості та перевіреній часом стабільності. Однак, її складність і потреба в більш тривалому часі розробки можуть бути перепоною для проєктів, що вимагають швидкого впровадження [22].

Ruby, особливо у поєднанні з фреймворком Ruby on Rails, є чудовим вибором для швидкої розробки та впровадження додатків. Ця мова пропонує зручність, гнучкість і великий набір готових рішень. Водночас її продуктивність поступається таким технологіям, як Node.js чи Java, що може бути критичним для задач із високим навантаженням [21].

Node.js, як середовище виконання JavaScript, пропонує унікальні переваги завдяки своїй асинхронній обробці подій. Це забезпечує високу швидкість роботи й масштабованість, необхідні для обробки фінансових транзакцій у реальному часі. Node.js дозволяє використовувати одну мову для клієнтської та серверної частин, що знижує складність і спрощує розробку [23].

Враховуючи особливості проекту, який передбачає інтеграцію з банківськими системами, обробку фінансових даних і потребу в швидкому розгортанні функціоналу, найкращим вибором для серверної частини є Node.js. Воно забезпечує необхідний баланс між продуктивністю, простотою розробки та гнучкістю інтеграції з іншими технологіями. Node.js також дозволяє легко масштабувати систему і швидко адаптувати її під нові вимоги, що робить його ідеальним рішенням для створення сучасного фінансового додатку.

2.2.3 Вибір фреймворку

Розробка фінансового додатку вимагає вибору фреймворку або бібліотеки, які забезпечать зручність створення функціонального і стабільного інтерфейсу користувача. Серед популярних технологій розглянемо Angular, React, Vue та Django, оцінюючи їх відповідність потребам проекту [24].

React є бібліотекою для побудови інтерфейсів, яка пропонує високу продуктивність, компонентний підхід і гнучкість у налаштуванні. React дозволяє легко інтегрувати сторонні бібліотеки, однак він не є повноцінним фреймворком, що може вимагати додаткових інструментів для управління станом або маршрутизацією.

Vue забезпечує зручність і простоту розробки завдяки своєму реактивному підходу. Його структура інтуїтивна, а документація доступна навіть для новачків. Однак для великих проектів може знадобитися більше зусиль для масштабування та інтеграції з комплексними рішеннями.

Django, як серверний фреймворк, виділяється своєю безпекою та здатністю швидко розробляти бекенд-частину. Він відмінно підходить для створення фінансових систем, але для фронтенду може потребувати додаткових рішень.

Angular, на відміну від React і Vue, є повноцінним фреймворком, що забезпечує все необхідне для створення складних односторінкових додатків. Angular пропонує потужний інструментарій для управління станом,

маршрутизації та інтеграції із зовнішніми API, що робить його ідеальним для проекту з великим обсягом функціональності [25].

Зважаючи на вимоги фінансового додатку, зокрема потребу в чіткій структурі, масштабованості, ефективній взаємодії з бекендом і можливості створення комплексних UI-рішень, оптимальним вибором є Angular. Його сильні сторони, такі як двостороннє зв'язування даних, модульність і підтримка TypeScript, забезпечують стабільність і гнучкість у розробці.

Нижче наведена таблиця для порівняння фреймворків за ключовими параметрами:

Таблиця 2.2 – таблиця порівняння фреймворків

Фреймворк	**Основні переваги**	**Основні обмеження**	**Типове використання**	**Масштабованість**
Angular	Комплексність, сильна типізація, велика спільнота	Великий розмір, складність вивчення	Великі веб-додатки, корпоративні проекти	Висока
React	Гнучкість, широка спільнота, компонентна модель	Вимагає додаткових бібліотек для повноцінного функціоналу	Легкі та середньої складності проекти	Середня
Vue	Простота, інтуїтивний синтаксис, швидкість розробки	Менша екосистема порівняно з Angular і React	Середньої складності веб-додатки	Середня
Django	Швидкість розробки, вбудовані засоби та адмін-панель	Обмеження в виборі мови (Python)	Веб-додатки на Python	Висока

Враховуючи результати порівняння фреймворків для розробки клієнтської частини, Angular є найбільш збалансованим вибором для цього проекту,

враховуючи його потребу в інтеграції з іншими технологіями, можливість масштабування та забезпечення функціонального і стабільного користувацького інтерфейсу.

2.3 Вибір системи безпеки

Вибір системи безпеки для монетарного управління ґрунтується на гарантії секретності, цілісності та доступності інформації. Система повинна підтримувати шифрування даних як під час їх передачі, так і під час зберігання, забезпечувати багатофакторну автентифікацію для захисту облікових записів користувачів і реалізувати керування доступом на основі ролей. Основні потреби включають виявлення загроз у реальному часі, дотримання глобальних норм, таких як GDPR або PCI DSS, і взаємодію з інфраструктурними системами. План має захищати мережу від хакерів і працювати безперебійно, навіть якщо нею користується багато людей.

2.3.1 Критерії вибору системи безпеки

Критерії вибору системи безпеки мають важливе значення для гарантування захисту грошової інформації, захисту від цифрових загроз і забезпечення зручності для кінцевих користувачів. Головним завданням є секретність даних, що досягається за допомогою методів шифрування для обробки обміну даними між користувачем і сервером, а також захисту під час зберігання. Це дає змогу запобігти витоку інформації та захистити дані від людей, які не повинні їх бачити.

Система безпеки повинна підтримувати багаторівневу аутентифікацію. Переглянуте речення: ця композиція включає записи, включаючи паролі,

одноразові парольні коди, біометричні методи ідентифікації, такі як ідентифікація за відбитками пальців або розпізнавання обличчя. Цей метод ефективно знижує ймовірність викрадення облікового запису користувача. Авторизуйтесь відповідно до принципу найменших привілеїв, надаючи доступ лише до необхідних ресурсів для виконання обов'язків.

Головне, щоб програма могла швидко знаходити небезпечні дії та реагувати на них. Основна мета полягає в тому, щоб потрібні інструменти, які працюють разом і стежать за речами, щоб помітити щось підозріле, коли це станеться. Крім того, система повинна забезпечувати здатність швидко відновлювати дані під час екстрених обставин, таких як кіберагресія або збої в схемі дій.

Масштабованість та інтегрованість також відіграють важливу роль. Система повинна бездоганно взаємодіяти з різними службами та технологіями, наприклад, банківськими API, і враховувати розширення кількості даних і баз користувачів. Переконайтеся, що система безпеки працює ефективно, не повільніше, навіть коли сервер дуже зайнятий [26].

Таким чином, система безпеки має бути продумано підібрана та розроблена таким чином, щоб знайти баланс між надійним захистом, простотою використання та можливістю розширення та масштабування разом із програмою.

2.3.2 Аналіз організації систем безпеки

Захищені конфігурації відіграють важливу роль у створенні програмного забезпечення, особливо під час грошових операцій. Дуже важливо думати про багато рівнів безпеки, щоб захистити наші дані, конфіденційність і переконатися, що вони завжди доступні. Вивчення сучасних методів структурування

оборонних стратегій дає нам змогу точно визначити основні інструменти та механізми виконання обов'язків.

Основний компонент безпеки використовує OAuth 2.0 — процедуру авторизації, яка полегшує доступ до ресурсів без передачі пароля. Це дозволяє інтегрувати сторонні служби, використовуючи видані маркери доступу після успішної автентифікації користувача. Використання токенів допомагає зберегти вашу інформацію в безпеці, оскільки вони існують лише короткий час і можуть бути забрані за потреби.

JWT — це формат токенів, що використовується в протоколах для передачі даних користувачів або даних для входу в компактному та захищеному вигляді. Кожен токен має цифровий підпис, яка підтверджує його справжність і забезпечує захист від несанкціонованого доступу. Цей формат ідеально підходить для взаємодії між багатьма невеликими сервісами, оскільки дозволяє їм обмінюватися короткими та безпечними ключами.

TLS (Secure Transport Layer) забезпечує захист інформації, що передається через Інтернет. Це стандарт безпечного зв'язку, який гарантує конфіденційність даних, що передаються між запитуючим пристроєм і пристроєм, який надає відповідь. TLS запобігає можливості перехоплення даних, забезпечуючи їх недоступність для сторонніх. Цей протокол є важливим для захисту API, веб-додатків та інтеграції сторонніх сервісів.

Сучасні протоколи безпеки додатково спрощують реалізацію вимог багатофакторної автентифікації для підтвердження користувача. Це включає використання паролів, одноразових кодів (OTP) або біометричних даних. Часта перевірка системи допомагає помітити будь-які незвичні дії та швидко впоратися з можливими небезпеками [27].

2.3.3 Обґрунтування вибору системи безпеки

Для забезпечення високого рівня безпеки системи управління транзакціями та фінансами рекомендовано використовувати поєднання сучасних протоколів та технологій. OAuth 2.0 забезпечить безпечну авторизацію, надаючи можливість контрольованого доступу до ресурсів. JWT стане ефективним інструментом для передачі та перевірки токенів, що зручно для масштабованих систем. TLS шифруватиме передачу даних між клієнтами та серверами, забезпечуючи їхню конфіденційність і захист від перехоплення. Така комбінація гарантує відповідність стандартам безпеки фінансових установ, зберігаючи конфіденційність, цілісність та доступність даних, а також мінімізуючи ризики несанкціонованого доступу.

2.4 Вибір бази даних

Вибір системи зберігання даних для цієї системи має бути орієнтований на ефективну обробку великих обсягів транзакцій, забезпечення високої доступності та безпеки інформації. Для досягнення цих цілей необхідно враховувати вимоги до цілісності даних, швидкості обробки запитів та масштабованості. Важливо, щоб система підтримувала транзакційний менеджмент для збереження консистентності даних, а також дозволяла ефективно працювати з великими обсягами фінансових записів. Інтеграція з механізмами безпеки, такими як шифрування даних і контроль доступу, повинна бути обов'язковою складовою для забезпечення конфіденційності та захисту користувацької інформації.

2.4.1 Критерії вибору бази даних

Критерії вибору системи зберігання даних для фінансової системи включають кілька ключових аспектів, які визначають її ефективність, безпеку та зручність у використанні. Першим критерієм є швидкість обробки даних, оскільки система повинна підтримувати високу продуктивність при великих навантаженнях, зокрема при обробці фінансових транзакцій в реальному часі. Важливо також забезпечити масштабованість, щоб система могла без проблем адаптуватися до зростання обсягу даних та кількості користувачів.

Наступним важливим аспектом є захист даних, оскільки фінансова інформація є чутливою і потребує високого рівня безпеки. Вибір системи зберігання даних має включати підтримку шифрування даних, контролю доступу та вбудованих механізмів безпеки, таких як аудит та авторизація.

Крім того, для забезпечення цілісності даних необхідно вибрати систему, яка підтримує транзакційність, що дозволяє гарантовано зберігати узгодженість даних навіть при збою у системі. Важливо також, щоб система підтримувала паралельну обробку запитів, щоб уникнути затримок у роботі з великими обсягами даних.

Іншими критеріями є сумісність з іншими компонентами системи, зокрема API для інтеграції з банківськими системами та сторонніми сервісами, а також можливість автоматичного резервного копіювання для забезпечення відновлення даних у разі втрати або пошкодження [28].

2.4.2 Аналіз систем організації баз даних

Аналіз систем організації баз даних полягає в оцінці різних підходів та технологій, що забезпечують ефективне зберігання, доступ і обробку даних. Зокрема, для фінансових систем критичним є вибір архітектури бази даних, яка

дозволяє надійно зберігати чутливу інформацію та гарантує високу продуктивність при обробці великих обсягів транзакцій.

Одним з основних підходів є реляційні бази даних, які використовують структуру таблиць з чітко визначеними зв'язками між ними. Такі системи добре підходять для фінансових додатків, оскільки забезпечують цілісність даних за допомогою транзакцій, а також підтримують складні запити, агрегування та аналітику. Вони також мають добре розвинену систему управління доступом, що дозволяє забезпечити високу безпеку даних. Однак реляційні бази можуть стати менш ефективними при високих навантаженнях, коли потрібно обробляти дуже великий обсяг даних.

Іншим варіантом є NoSQL бази даних, які використовуються в ситуаціях, де необхідна висока масштабованість, гнучкість у зберіганні структурованих і неструктурованих даних, а також швидкість доступу. Такі системи підтримують горизонтальне масштабування, що робить їх привабливими для великих фінансових платформ, що обробляють величезні обсяги транзакцій. Однак відсутність жорсткої структури може ускладнити забезпечення цілісності даних та їх консистентності в реальному часі.

Гібридні рішення також стають все більш популярними, коли використовуються переваги як реляційних, так і NoSQL систем. Такий підхід дозволяє комбінувати гнучкість у зберіганні даних та потужність для виконання транзакцій з високою цілісністю. Крім того, гібридні системи можуть підтримувати різні типи даних, що зберігаються в окремих сховищах, які можуть взаємодіяти між собою.

Особливу увагу також варто приділити системам розподілених баз даних, які дозволяють зберігати дані на декількох серверах. Вони забезпечують високу доступність і відмово стійкість, що важливо для фінансових систем, які потребують гарантованої доступності інформації 24/7. При цьому розподілені бази даних часто мають складніші вимоги до налаштування та управління, що потребує більше ресурсів на етапі підтримки [29].

Усі ці підходи мають свої переваги та недоліки, і вибір системи залежить від конкретних вимог до швидкості, надійності, масштабованості та безпеки фінансової платформи.

Для автоматизованої системи управління транзакціями та керування фінансами можна розглянути кілька популярних варіантів баз даних, кожен з яких має свої переваги та недоліки.

1) MySQL:

Переваги: Висока продуктивність, надійність, активна спільнота підтримки, підтримка ACID-транзакцій.

Недоліки: Обмежена масштабованість у порівнянні з деякими NoSQL базами, складність налаштування складних реляційних схем.

2) PostgreSQL:

Переваги: Підтримка складних запитів та реляційних зв'язків, розширюваність, висока продуктивність.

Недоліки: Відносна складність у налаштуванні та адмініструванні для новачків.

3) MongoDB:

Переваги: Гнучкість у роботі з неструктурованими даними, горизонтальне масштабування, висока продуктивність при роботі з великими обсягами даних.

Недоліки: Відсутність підтримки ACID-транзакцій до версії 4.0, обмежена підтримка складних запитів.

4) Redis:

Переваги: Висока швидкість доступу до даних, підтримка складних структур даних, підходить для кешування та обробки даних у реальному часі.

Недоліки: Обмежена підтримка складних запитів та реляцій, обмеження на обсяг даних у пам'яті.

2.4.3 Обґрунтування вибору бази даних

З огляду на вимоги до системи та переваги різних баз даних, вибір бази даних повинен бути обґрунтований балансом між продуктивністю, масштабованістю та функціональністю. Для автоматизованої системи управління транзакціями та керування фінансами рекомендується використовувати комбінацію реляційної та NoSQL бази даних.

1. Основна база даних: PostgreSQL

Обґрунтування: PostgreSQL забезпечує високу продуктивність та надійність, підтримку складних реляційних зв'язків та транзакцій, що важливо для обробки фінансових даних. Вона також має розширені можливості для масштабування та оптимізації [30].

2. Допоміжна база даних: Redis

Обґрунтування: Redis використовується для кешування даних і обробки запитів у реальному часі, що забезпечує високу швидкість доступу до часто використовуваних даних. Це дозволяє знизити навантаження на основну базу даних і підвищити загальну продуктивність системи [31].

Комбінація PostgreSQL та Redis дозволяє забезпечити надійність і продуктивність системи, а також гнучкість у роботі з різними типами даних. Такий підхід забезпечує оптимальне виконання вимог до систем.

2.5 Інтеграція зі сторонніми сервісами

Штучний інтелект є потужним інструментом для покращення функціональності системи. Інтеграція з AI-сервісами, такими як OpenAI API або інші рішення на основі машинного навчання. Для забезпечення максимальної функціональності інтеграції з AI необхідно визначити детальну структуру транзакцій та описати як об’єкт транзакції взаємодіє із застосунком. Це дозволяє

користувачам отримувати реальний досвід взаємодії з системою, а штучному інтелекту – аналізувати дані та генерувати корисну інформацію [32].

2.5.1 Структура транзакції

Об’єкт **"**Транзакція**"** є ключовим елементом у системі управління фінансами. Він відображає фінансову операцію, яка може бути виконана користувачем або автоматичною системою. У цьому підрозділі описуються структура об’єкта транзакції, його взаємодія з іншими компонентами системи, а також процеси зберігання та обробки даних.

Таблиця 2.3 – таблиця основних полів транзакції

Атрибут	**Тип даних**	**Опис**	**Приклад**
ID транзакції	string або UUID	Унікальний ідентифікатор транзакції. Використовується для пошуку або зміни запису.	"txn_123456789"
Дата і час	Date	Мітка часу, коли була здійснена транзакція.	"2024-11-27T12:45:00Z"
Сума	number	Грошовий обсяг операції. Відображає кількість коштів, витрачених або отриманих.	125.50
Валюта	string	Тип валюти, в якій була здійснена операція.	"USD", "EUR", "UAH"
Одержувач / відправник	string	Назва, ідентифікатор чи опис сторони транзакції.	"Amazon", "Joe Biden"
Тип транзакції	string (enum)	Категорія операції: покупка, переказ, зняття готівки тощо.	"purchase", "transfer", "withdrawal"
Локація	string або object	Географічне місце проведення операції (адреса або координати).	"New York, NY", { lat: 40.7128, lng: -74.0060 }

Продовження таблиці 2.3

Опис	string	Додаткова текстова інформація про операцію.	"Оплата за підписку на Spotify"
Категорія	string (enum)	Тематична категорія транзакції.	"Розваги", "Продукти", "Подорожі"
Статус транзакції	string (enum)	Стан операції: успішна, очікує підтвердження, відхилена.	"completed", "pending", "declined"
Метод оплати	string	Спосіб, яким була здійснена транзакція: банківська картка, електронний гаманець.	"Credit Card", "PayPal", "Cash"
Теги транзакції	array<string>	Додаткові ключові слова для класифікації або пошуку транзакції.	["Подорож", "Сімейні витрати"]
Комісія	number	Додаткові витрати, пов'язані із транзакцією (за потреби).	2.50
Баланс	number	Баланс на коштів на карті	10000.25
ID квитанції	string	Унікальний ідентифікатор квитанції про оплату	"txn_123456789"
Кешбек	number	Повернута сума коштів організацією	2.50

Пояснення атрибутів полів транзакції:

1. ID транзакції**:** дозволяє унікально ідентифікувати транзакцію в базі даних.
2. Дата і час: використовується для сортування операцій у хронологічному порядку.
3. Сума**:** обсяг операції дозволяє аналізувати загальні витрати або прибуток.
4. Валюта**:** потрібна для забезпечення мультивалютної підтримки у системі.

5. Одержувач/відправник: важливий для відображення інформації про інші сторони операції.
6. Тип транзакції: визначає її контекст і допомагає в групуванні витрат.
7. Локація: використовується для географічного аналізу витрат і безпеки.
8. Опис: допомагає користувачу деталізувати операцію, роблячи її більш зрозумілою.
9. Категорія: забезпечує сортування витрат за темами (продукти, розваги тощо).
10. Статус: інформує користувача про стан операції, дозволяючи відстежувати її виконання.
11. Метод оплати: дає змогу аналізувати, як часто використовуються певні способи платежів.
12. Теги транзакції: додають гнучкість у пошуку й персоналізації.
13. Комісія: використовується для прозорості у фінансових звітах.
14. Баланс: дає змогу користувачу відстежувати зміну залишку після кожної транзакції. Також може бути корисним для аналізу руху коштів.
15. ID квитанції: пов'язує транзакцію з електронною квитанцією, що дозволяє користувачу підтвердити факт оплати або переглянути додаткові деталі.
16. Кешбек: зручний для відображення економії, яка була отримана користувачем у результаті акцій або програм лояльності.

Взаємодія із застосунком:

- Фільтрація: Користувач може фільтрувати транзакції за датою, статусом, сумою чи категорією.
- Візуалізація: Дані відображаються у вигляді таблиць, списків або графіків для швидкого аналізу.

- Рекомендації AI: Використання атрибутів дозволяє системі аналізувати витрати та створювати персоналізовані поради.
- Безпека: Система перевіряє незвичні атрибути (наприклад, локацію або суму) для попередження шахрайства [33].

2.5.2 Аналіз транзакцій та виявлення шахрайства

Інтеграція з AI-сервісами дозволяє реалізувати систему моніторингу транзакцій в реальному часі. Моделі машинного навчання можуть автоматично виявляти підозрілі транзакції, порівнюючи їх з історичними даними користувача та виявляючи аномалії, які можуть вказувати на шахрайські операції. Наприклад, якщо користувач здійснює транзакцію, що значно перевищує його звичайний витратний шаблон, система може миттєво відреагувати, надіславши сповіщення про потенційне шахрайство.

Моделі, навчання яких базується на методах класифікації та аналізу великих обсягів даних, можуть:

- Оцінювати ризик кожної транзакції.
- Використовувати методи виявлення аномалій для прогнозування потенційних загроз.
- Порівнювати поведінку користувача з типовими шаблонами транзакцій на основі факторів, таких як сума, частота, місце та тип операцій [34].

2.5.3 Персоналізовані фінансові рекомендації

Штучний інтелект також може бути використаний для надання персоналізованих фінансових рекомендацій користувачам. Використовуючи алгоритми машинного навчання, система здатна вивчити фінансові звички

користувача, аналізуючи його витрати, доходи та цілі. На основі цих даних AI може пропонувати:

- Оптимізацію витрат: Рекомендації щодо зменшення витрат на певні категорії товарів або послуг.
- Планування бюджету: Створення персоналізованих бюджетів з урахуванням доходів і витрат.
- Інвестиційні поради: Оцінка потенційних інвестиційних можливостей на основі фінансових звичок користувача [35].

2.5.4 Технічна реалізація ШІ у фінансові сервіси

Для інтеграції з AI-сервісами можуть використовуватися такі API, як OpenAI для природної обробки мови (NLP), щоб забезпечити аналіз транзакцій на основі текстових описів або інші платформи для машинного навчання, наприклад, TensorFlow або Scikit-learn, для створення кастомізованих моделей.

Процес інтеграції AI-сервісів виглядатиме наступним чином:

1. Збір даних: Збір історії транзакцій користувача та додаткових даних (наприклад, профіль витрат).
2. Обробка даних: Передача даних через API до AI-сервісу для аналізу.
3. Моделювання: Використання алгоритмів для виявлення аномалій або прогнозування фінансових патернів.
4. Рекомендації: Генерація рекомендацій або сповіщень, які можуть бути відправлені користувачу через систему.

Інтеграція з AI-сервісами значно покращує точність та ефективність системи "Wallet", даючи користувачам можливість отримувати більш точні фінансові рекомендації та забезпечує виявлення шахрайських операцій на ранніх етапах [32].

2.6 Середовище розробки

Для реалізації проекту було обрано WebStorm як основне середовище для роботи з кодом. Це потужний інструмент, який забезпечує ефективну підтримку JavaScript та TypeScript, а також зручну інтеграцію з популярними фреймворками, такими як React, Angular та Vue. Вбудовані функції для рефакторингу, тестування та налагодження коду дозволяють розробникам працювати швидко та зручно, а інтеграція з системами контролю версій забезпечує стабільний процес командної роботи та управління кодом [36].

Для організації розробки та забезпечення сумісності між середовищами розгортання було використано Docker. Цей інструмент дозволяє контейнеризувати кожен компонент системи, що значно полегшує тестування, розгортання та підтримку додатка. Завдяки Docker можна забезпечити стабільність середовища на всіх етапах життєвого циклу програми, незалежно від конфігурацій машин розробників або серверів у продакшн середовищі [37].

Для автоматизації процесів розробки та доставки продукту було обрано систему CI/CD, що включає інструменти на кшталт Jenkins, GitLab CI або GitHub Actions. Ці інструменти дозволяють автоматично виконувати тестування, перевіряти код на помилки та здійснювати деплой на різні середовища. Це не лише значно прискорює розробку, але й зменшує ймовірність помилок на етапах розгортання, забезпечуючи безперервне вдосконалення та надійне оновлення продукту [38].

2.7 Висновок до розділу

У даному розділі ми розглянули проектування клієнтської частини для системи обробки та керування фінансами, а також обгрунтували вибір технологічного стеку. Для розробки клієнтської частини ми вибрали Angular та TypeScript як основні технології. Angular надає масштабованість, стабільність та

багатофункціональність, що особливо важливо для веб-застосунків зі складним функціоналом, таким як управління фінансами. TypeScript дозволяє підвищити безпеку та продуктивність розробки завдяки сильній типізації та підтримці сучасних стандартів JavaScript.

На сервері ми вибрали Node.js як платформу для серверної частини. Node.js забезпечує швидкість виконання, високу масштабованість і гнучкість у виборі інструментів для розробки. Його асинхронний характер і можливість використання JavaScript для розробки є значущими перевагами у контексті сучасних веб-застосунків.

Вибір бази даних впливає на продуктивність, масштабованість та безпеку системи. Для забезпечення ефективного управління фінансами ми обрали PostgreSQL як реляційну базу даних. PostgreSQL відома своєю надійністю, підтримкою транзакцій і ACID властивостей, що є критичними для систем з фінансовими операціями.

Для зберігання та управління тимчасовою інформацією ми використовуємо Redis. Він забезпечує швидкий доступ до даних завдяки своєму ін-меморі сховищу та підтримці різних структур даних.

Безпека є ключовим аспектом будь-якої фінансової системи. Ми забезпечуємо безпеку за допомогою різних заходів, включаючи аутентифікацію за стандартом OAuth2, використання токенів JWT для авторизації, шифрування даних та використання захищених протоколів передачі даних (наприклад, HTTPS). Це забезпечує конфіденційність, цілісність та доступність даних для користувачів і системних адміністраторів.

Обрані технології та підходи дозволяють створити сучасну та безпечну систему для управління фінансами, що відповідає найвищим стандартам розробки програмного забезпечення і відповідає потребам сучасних користувачів.

3 РЕАЛІЗАЦІЯ КЛІЄНТСЬКОЇ ЧАСТИНИ ТА СЕРВЕРНОЇ ЛОГІКИ СИСТЕМИ УПРАВЛІННЯ ФІНАНСАМИ

3.1. Реалізація клієнтської частини

3.1.1 Архітектура клієнтської частини

Реалізація клієнтської частини додатку базувалася на сучасних підходах до розробки веб та мобільних додатків, орієнтованих на інтерактивність, високу продуктивність і зручність користувача. Основою клієнтської частини стала модульна архітектура, що забезпечує розділення функціональності на логічно незалежні частини. Такий підхід дозволяє покращити масштабованість, спрощує підтримку та полегшує інтеграцію нових функцій.

Кожен функціональний блок додатка реалізовано у вигляді окремого модуля, який об’єднує компоненти, сервіси та інші необхідні ресурси. Ця модульна структура дозволяє розробникам ізолювати логіку роботи, уникати конфліктів при оновленні або додаванні нового функціоналу, а також спрощує тестування та відлагодження.

можливостей пристроїв через спеціальні плагіни.

Основні модулі архітектури:

- Core Module: Цей модуль містить всі критично важливі сервіси та конфігурації, які використовуються в усьому додатку. Він включає сервіси для авторизації користувачів, налаштування API-запитів, а також глобальні утиліти для роботи з токенами доступу, налаштувань мережі та управління помилками. Це також включає інтерфейси та типи даних, що використовуються на рівні всього додатку.
- Shared Module: Модуль Shared містить компоненти, які можуть бути використані в різних частинах додатку, такі як кнопки, таблиці, форми, а також модальні вікна, анімації та директиви. Цей модуль дозволяє уникнути дублювання коду, оскільки всі ці компоненти можна повторно використовувати

у різних функціональних модулях додатку. Наприклад, тут можуть бути також сервіси, які відповідають за управління анімаціями чи роботу з даними, які використовуються по всьому додатку.функціональності додатка, наприклад:

- Модуль транзакцій для управління фінансовими операціями.
- Модуль статистики для аналізу доходів і витрат.
- Модуль профілю для налаштування персональних даних користувача.

- Store**:** Для централізованого управління станом додатка використовується модуль Store, де зберігаються дані про користувачів, акаунти, транзакції та інші важливі дані. Це дозволяє забезпечити синхронізацію даних між різними частинами додатка, а також ефективно управляти станом додатку через спеціалізовані сервіси. Наприклад, модули, які відповідають за accounts, auth, user та transaction, взаємодіють з Store для отримання та збереження даних у реальному часі.
- Utils Module: Модуль утиліт містить допоміжні функції, які використовуються в різних частинах додатка. Це можуть бути функції для обробки дат, чисел, форматування даних, а також інші загальні утиліти, що полегшують розробку і забезпечують зручність в роботі з різними даними та форматами.
- Styles Module: В цьому модулі зберігаються глобальні стилі, які визначають зовнішній вигляд додатка. Це може бути як загальний дизайн, так і специфічні стилі для певних компонентів, що адаптуються під різні пристрої (наприклад, стилі для мобільної та десктопної версії додатка).

Основні компоненти модуля :

Компоненти (Components):

• Вони відповідають за відображення інформації та взаємодію з користувачем. Компоненти можуть бути як окремими екранами (наприклад, екран для транзакцій, профілю, налаштувань), так і частинами інтерфейсу

(наприклад, кнопки, списки, форми). У вашому додатку компоненти можуть використовуватися для:

- Відображення даних (наприклад, історія транзакцій, статистика).
- Інтерактивних елементів (кнопки, поля введення).
- Модальних вікон для взаємодії з користувачем (наприклад, підтвердження транзакції).

Сервіси (Services):

• Сервіси реалізують бізнес-логіку та здійснюють взаємодію між клієнтським додатком і сервером. Вони використовуються для обробки запитів до API, обробки помилок, авторизації користувачів, а також для бізнес-операцій (наприклад, обробка транзакцій, оновлення профілю). У вашому додатку сервіс може:

- Залишати і отримувати дані з серверної частини через HTTP-запити.
- Виконувати функції, які не пов'язані з UI, наприклад, обробка платежів, валідація користувацьких даних.

Директиви (Directives):

• Директиви використовуються для розширення можливостей HTML. Вони можуть додавати спеціальну поведінку елементам DOM. Наприклад, можна створити директиву для автоматичного фокусування на елементі після завантаження сторінки або для перевірки правильності введення даних у формах.

Пайпи (Pipes):

• Пайпи використовуються для обробки даних перед їх відображенням. Вони можуть форматувати дату, валюту, числові значення або виконувати будь-яку іншу трансформацію даних перед тим, як вони з'являться на екрані користувача.

- Інтерфейси та Типи (Interfaces and Types):

• Інтерфейси та типи забезпечують типову безпеку для даних, які передаються між компонентами та сервісами. Вони визначають структуру даних, наприклад, для транзакцій, користувацьких профілів, налаштувань.

Модулі (Modules):

• Кожен модуль в вашому додатку є контейнером для логіки, компонентів, сервісів та інтерфейсів. Це дозволяє підтримувати масштабованість і чітке розділення між різними функціональними частинами програми. Модулі також відповідають за імпорт та експорт залежностей. Наприклад:

- Transaction Module відповідає за всі компоненти та сервіси, пов'язані з фінансовими транзакціями.
- User Module забезпечує всі функції, пов'язані з профілем користувача.

Маршрутизація (Routing):

• Система маршрутизації відповідає за навігацію між різними сторінками та компонентами додатка. У вашому випадку вона реалізована через Angular Router, що дозволяє динамічно змінювати вигляд додатка на основі вибору користувача або взаємодії з додатком.

Контроль стану (State Management):

• Використання сховища стану, наприклад, NgRx або RxJS, дозволяє централізовано керувати даними додатка, такими як статус авторизації, історія транзакцій, налаштування користувача тощо. Це дозволяє забезпечити консистентність даних і полегшити їх оновлення в різних частинах додатка.

Мобільні плагіни Capacitor:

• Оскільки додаток підтримує мобільні платформи, використання Capacitor дозволяє інтегрувати нативні функції, такі як доступ до камери, геолокації, збереження даних в локальному сховищі. Ці плагіни додаються через модулі mobile в вашій архітектурі [39].

Кожен з цих компонентів працює в тісній взаємодії з іншими, що забезпечує функціональність та продуктивність додатка. Модульна архітектура дає змогу не лише чітко організувати код, але й дозволяє легко масштабувати додаток, додавати нові функціональності без шкоди для існуючих частин системи.

3.1.2 Інтерфейс користувача та UX/UI дизайн

Інтерфейс користувача (UI) та досвід користувача (UX) є критично важливими аспектами в розробці додатка, оскільки вони безпосередньо впливають на взаємодію користувача з системою, її ефективність та загальне враження від використання. У додатку акцент зроблено на простоті, інтуїтивно зрозумілому інтерфейсі та швидкому доступі до основних функцій. Це дозволяє користувачам з різним рівнем досвіду у використанні фінансових додатків ефективно та комфортно здійснювати операції. Особлива увага приділяється узгодженості досвіду між веб-версією та мобільними платформами (iOS і Android), використовуючи принципи адаптивного дизайну. Загальний Flow інтерфейсу наведений нижче (рис. 3.1).

UI/UX дизайн принципи та підхід:

1. Інтуїтивно зрозумілий інтерфейс:

• Дизайн додатка орієнтований на простоту використання. Кожен екран має чітко визначену функцію, а елементи керування логічно розташовані для полегшення доступу. Для цього були створені окремі екрани для таких функцій, як управління транзакціями, перегляд статистики, налаштування профілю, доступ до історії платежів, налаштувань безпеки, доступ до фінансового радника та інші.

2. Мобільна та веб-адаптивність:

• Оскільки додаток підтримує як мобільні (iOS та Android), так і веб-платформи, інтерфейс був спроектований так, щоб коректно відображатися на різних розмірах екранів. Використовуються адаптивні стилі та компоненти, що дозволяють забезпечити оптимальне відображення як на мобільних пристроях, так і на десктопних.

3. Чітка навігація:

• Інтерфейс включає просту та зручну навігаційну панель для переходу між різними розділами додатка. Використовуються знайомі патерни навігації, такі як

вкладки та бокове меню для мобільної версії, що дозволяє швидко знайти необхідні функції. Модульна архітектура також дозволяє зручно переходити між модулями, наприклад, між фінансовими операціями та статистикою.

4. Фокус на функціональність:

• UI спроектовано так, щоб фокус був на основних функціях додатка, таких як створення транзакцій, перегляд балансу та витрат. Всі функції забезпечують швидкий доступ, де кнопки та інші елементи керування мають очевидну мету, а їхнє розташування спрощує взаємодію.

5. Персоналізація досвіду користувача:

• Додаток надає можливість налаштовувати профіль користувача, змінювати мову інтерфейсу, встановлювати параметри безпеки (наприклад, двофакторна автентифікація). Це дозволяє кожному користувачеві адаптувати додаток під свої потреби та вподобання.

Дизайн та візуальні елементи

1. Колірна палітра та шрифти:

• Вибір кольорів базується на поєднанні спокійних та контрастних відтінків для того, щоб забезпечити чітке сприйняття інформації. Для основних кнопок використано яскраві кольори, що привертають увагу до важливих дій, таких як підтвердження транзакцій або налаштування профілю. Водночас для фону та менш важливих елементів застосовуються нейтральні відтінки, що не відволікають користувача.

2. Типографіка:

• Вибрані шрифти прості для читання на різних пристроях, з чітким поділом заголовків та основного тексту. Важливі елементи виділяються жирним шрифтом або великими літерами для підвищення помітності.

3. Анімації та ефекти:

• Легкі анімації використовуються для покращення взаємодії з користувачем. Наприклад, плавні переходи між екранами та анімації кнопок при

натисканні допомагають створити більш динамічний та приємний досвід. Використання анімацій також допомагає привертати увагу до ключових елементів інтерфейсу.

UX: забезпечення комфортного досвіду користувача

1. Швидкість роботи:

• Завдяки ретельній оптимізації коду та інтеграції з серверною частиною додатка, користувачі можуть переглядати історію та отримувати необхідну інформацію в реальному часі, що забезпечує високу продуктивність додатка.

2. Зворотний зв'язок і повідомлення:

• Користувачі отримують миттєві зворотні реакції на свої дії, такі як підтвердження успішних транзакцій, попередження про помилки або відсутність доступних коштів. Це знижує ймовірність помилок і підвищує довіру користувача до системи.

3. Доступність:

• Інтерфейс був спроектований з урахуванням принципів доступності, що дозволяє людям з обмеженими можливостями також використовувати додаток. Для цього використовуються налаштування контрасту, великий шрифт, можливість навігації з клавіатури або допоміжними пристроями [40].

Після реалізації первинного дизайну було проведено аналіз дизайну для виявлення можливих проблем з інтерфейсом та взаємодією. Результати дозволили внести покращення, такі як коригування розміру кнопок, зміна деяких кольорових рішень для кращого контрасту, а також поліпшення зручності навігації між екранами.

У результаті застосування сучасних підходів до дизайну та постійного вдосконалення інтерфейсу, користувачі можуть безперешкодно та зручно користуватися функціями додатка, що позитивно впливає на загальне задоволення від його використання.

Рисунок 3.1 – загальна Flow користувача для доступу до функціоналу застосунку

3.1.3 Інтеграція з сервером

Інтеграція з сервером є однією з ключових частин роботи клієнтської частини додатку, оскільки вона забезпечує зв'язок між додатком та серверною частиною для виконання основних операцій, таких як авторизація користувачів, обробка транзакцій, отримання статистики, а також взаємодія з іншими зовнішніми сервісами.

У додатку інтеграція з сервером здійснюється за допомогою HTTP-запитів через Angular HttpClient, що дозволяє клієнтській частині обмінюватися даними

з сервером, наприклад, для отримання списку транзакцій, обробки введених користувачем даних або для автентифікації.

Реалізація в додатку:

1) Запити до серверу: Клієнтська частина для формування HTTP-запитів до API серверу використовує Angular HttpClient . Запити можуть бути різних типів: GET для отримання даних, POST для відправлення даних, PUT для оновлення, DELETE для видалення.

2) Аутентифікація та авторизація: Одним з основних завдань інтеграції є автентифікація користувача. Після того як користувач авторизується, отриманий токен (наприклад, JWT) зберігається на клієнтській стороні і додається до заголовків кожного запиту, що забезпечує доступ до захищених ресурсів на сервері.

3) Обробка помилок: Важливою частиною інтеграції є обробка помилок, що може статися під час запитів до сервера. Це можуть бути помилки мережі, помилки серверу (наприклад, 500 або 404), або помилки авторизації (якщо токен недійсний). У додатку розроблена система обробки таких помилок, яка дозволяє належним чином реагувати на проблеми, інформувати користувача і дозволяти додатку продовжити працювати без збою.

4) Обробка відповіді від сервера: Після виконання запиту сервер повертає відповідь у вигляді JSON, який потім обробляється в клієнтській частині додатку. Це можуть бути дані, які необхідно відобразити користувачу, або повідомлення про успішну операцію (наприклад, підтвердження транзакції).

Як обробляються помилки:

- Мережеві помилки: Якщо запит не може бути відправлений через мережеву проблему, Angular видасть помилку, яку можна обробити в методі handleError. Наприклад, це може бути помилка типу "Failed to fetch" через відсутність інтернет-з’єднання.
- Помилки сервера: Якщо сервер відповідає з кодом помилки (наприклад, 500 Internal Server Error або 404 Not Found), то ми можемо отримати деталі про

помилку через об'єкт error. Користувачеві буде показано відповідне повідомлення про помилку, яке надасть йому розуміння того, що трапилось.

- Авторизація: Якщо сервер повертає помилку, що стосується авторизації (наприклад, токен не дійсний або відсутній), на клієнтській стороні можна реалізувати механізм повторної авторизації або перенаправлення користувача на екран входу [41].

Важливість обробки помилок:

Коректна обробка помилок гарантує, що додаток буде стабільно працювати в умовах непередбачених ситуацій, таких як проблеми з мережею, сервером чи помилки на стороні клієнта. Вона дозволяє покращити взаємодію з користувачем, адже він отримає чітке пояснення про проблему та можливі шляхи вирішення, а не просто зупинку роботи додатку чи загальне повідомлення про помилку.

Приклад інтеграції:

У нашому додатку для отримання транзакцій використовується GET-запит, який відправляється на сервер із заголовками, що включають токен авторизації. Сервер перевіряє токен і повертає відповідь із необхідними даними. Якщо виникає помилка (наприклад, токен недійсний або сервер не доступний), система обробляє це та інформує користувача про помилку. Схему запитів до сервера да отримання даних наведена нижче (рис. 3.2)

Інтеграція з сервером в нашому додатку реалізована таким чином, щоб забезпечити максимальну гнучкість, надійність та зручність користувача. Усі запити обробляються асинхронно, що дозволяє додатку залишатися продуктивним, динамічно оновлювати дані, забезпечувати оперативну їх обробку та гарантувати стабільну роботу, що є зручним для користувачів навіть при великих обсягах інформації або складних операціях.

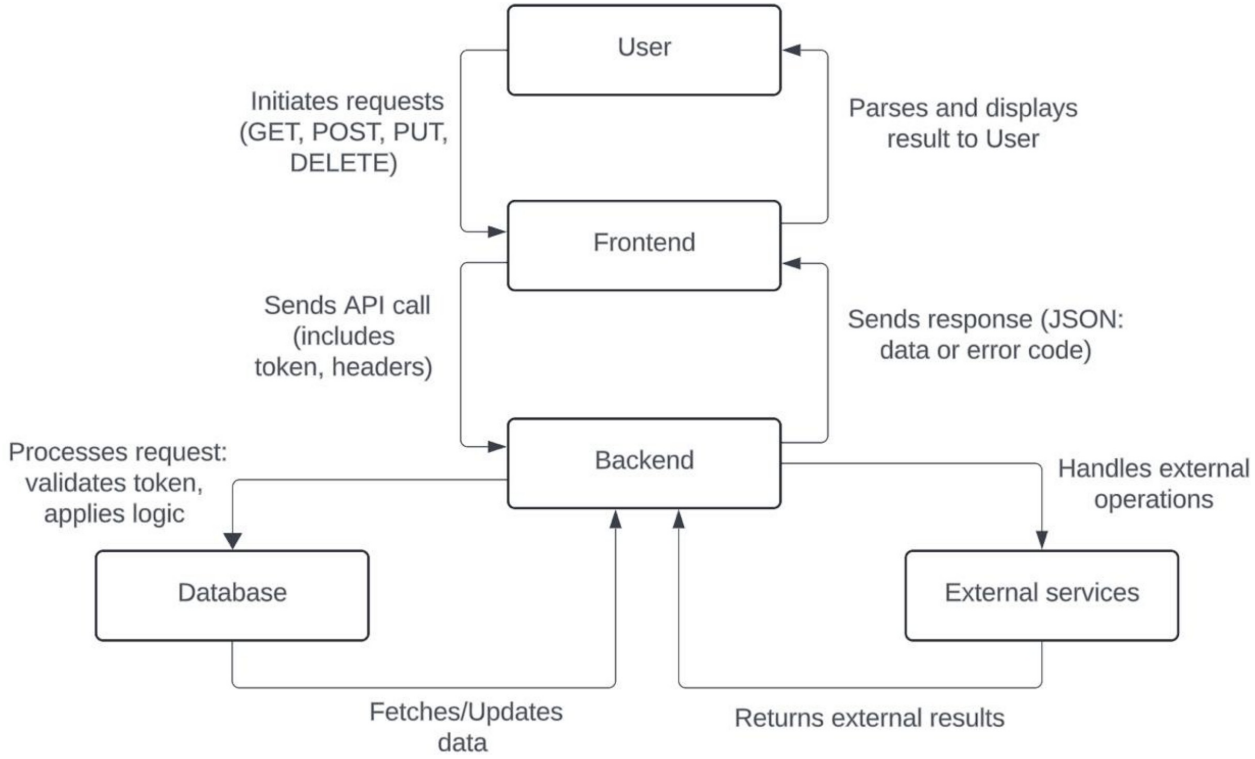

Рисунок 3.2 – графічна діаграма інтеграції з сервером

3.1.4 Реалізація основних функцій

Основні функції клієнтської частини системи охоплюють управління фінансами, аналіз витрат та доходів, планування бюджету та ціле а також налаштування профілю користувача. Кожна з цих функцій реалізована з урахуванням високої зручності та інтуїтивності, що дозволяє користувачам легко взаємодіяти з системою та отримувати корисну інформацію про їх фінансову діяльність.

1. Управління транзакціями: Користувачі можуть переглядати, додавати, редагувати та видаляти транзакції, що зберігаються в системі. Для цього реалізовано спеціальне інтерфейсне рішення, яке дозволяє зручно працювати з усіма даними транзакцій, включаючи сортування та фільтрацію за різними параметрами. Важливим аспектом є також надання можливості переглядати детальну інформацію щодо кожної транзакції, що значно покращує зручність користування.

2. Статистика доходів і витрат: Користувачам надається можливість аналізувати свої фінансові звички за допомогою графіків та діаграм, що надають візуальне представлення доходів і витрат за обраний період. В системі є можливість побудови звітів по категоріях витрат, що допомагає користувачам краще розуміти їх фінансові потоки та робити обґрунтовані фінансові рішення.

3. Налаштування профілю: Для персоналізації користувацького досвіду передбачені функції зміни особистих даних, налаштування мови інтерфейсу, управління сповіщеннями та зміни паролю. Це дає змогу користувачам адаптувати систему під свої індивідуальні потреби та вимоги безпеки.

4. Планування цілей та подій:

Користувачам надається можливість встановлювати фінансові цілі, такі як накопичення певної суми грошей, досягнення бажаного рівня економії або витрат на конкретні категорії. Ці цілі можуть бути пов'язані з конкретними подіями, такими як подорожі, покупка великої речі або інші важливі фінансові плани. Користувач може налаштувати нагадування та сповіщення, які допоможуть йому залишатись на шляху до досягнення своїх цілей. Інтерфейс для планування цілей простий і надає візуальні елементи для відслідковування прогресу в реальному часі. Це дозволяє користувачам краще організувати свої фінанси та активно управляти своїми заощадженнями та витратами.

Всі ці функціональні можливості реалізовані на основі клієнтських компонентів Angular, що забезпечують інтерактивність і зручність використання. Взаємодія з сервером відбувається через HTTP-запити, обробка яких здійснюється спеціальними сервісами. Кожна з функцій інтегрована з відповідними backend-сервісами для коректної обробки та збереження даних. Під час розробки цих функцій особлива увага приділялась оптимізації продуктивності, інтуїтивності інтерфейсу та безпеці даних.

3.2. Реалізація серверної частини

3.2.1 Архітектура серверної частини

Серверна частина додатку реалізована на основі мікросервісної архітектури, що забезпечує високу масштабованість, надійність і гнучкість системи. Мікросервісна архітектура передбачає розподіл функціональних частин додатку на невеликі, незалежні сервіси, кожен з яких відповідає за виконання конкретних завдань. Це дозволяє ефективно розподіляти навантаження, спрощувати підтримку і розширення системи, а також забезпечувати незалежний розвиток окремих модулів.

Основні компоненти архітектури:

1) Мікросервіси

Кожен мікросервіс відповідає за окрему функціональність. У додатку виділені наступні сервіси:

- Сервіс авторизації та аутентифікації: Забезпечує управління користувачами, верифікацію їхніх облікових даних, а також видачу та перевірку токенів доступу.
- Сервіс обробки транзакцій: Реалізує логіку створення, перегляду та модифікації фінансових операцій.
- Сервіс інтеграції з банками: Забезпечує взаємодію з API банківських систем для отримання даних про рахунки, баланси та історію транзакцій.
- Сервіс аналітики: Використовує алгоритми аналізу даних для формування статистики, прогнозів та рекомендацій.
- Сервіс обробки файлів: Відповідає за завантаження, збереження та передачу документів, таких як виписки чи звіти [39].

2) API-шлюз

- API-шлюз виступає єдиною точкою доступу для клієнтської частини додатку. Він перенаправляє запити до відповідних мікросервісів, а також виконує роль фільтрації запитів, кешування даних і обробки авторизації [43].

3) База даних

Для кожного мікросервісу може використовуватися окрема база даних, що дозволяє уникнути конфліктів і забезпечує кращу продуктивність. Наприклад, сервіс транзакцій може використовувати реляційну базу даних для зберігання структурованих даних, тоді як сервіс аналітики може працювати з базами NoSQL для обробки великих обсягів даних.

4) Система обміну повідомленнями

Використання системи обміну повідомленнями, наприклад, RabbitMQ або Kafka, дозволяє сервісам взаємодіяти асинхронно. Це забезпечує високу продуктивність і мінімізує залежності між мікросервісами.

Реалізація в нашому додатку

У нашому додатку мікросервісна архітектура дозволяє розподіляти навантаження між різними компонентами та масштабувати їх залежно від потреб. Наприклад, сервіс транзакцій може працювати з великою кількістю запитів від клієнтів, тоді як сервіс аналітики може виконувати складні обчислення у фоновому режимі.

Для кожного мікросервісу передбачені незалежні розгортання, що дозволяє оновлювати або модифікувати окремі компоненти без впливу на роботу всієї системи. Захищеність даних реалізована через систему токенів та обмеження доступу на рівні API-шлюзу.

Приклад обробки запиту

Приклад роботи мікросервісної архітектури: коли користувач запитує історію транзакцій, API-шлюз приймає запит і перенаправляє його до сервісу транзакцій. Сервіс перевіряє права доступу користувача через сервіс авторизації, отримує необхідні дані з бази і повертає їх до клієнта. У разі виникнення помилки, наприклад, через недоступність бази даних, користувач отримує відповідне повідомлення через систему обробки помилок.

Цей підхід забезпечує надійну, гнучку і масштабовану систему, готову до зростання вимог та збільшення навантаження.

3.2.2 Інтеграція з базою даних

Інтеграція з базою даних є ключовим компонентом архітектури серверної частини системи, оскільки всі дані, які стосуються транзакцій, користувачів, налаштувань профілю та інших важливих аспектів роботи системи, зберігаються в базі даних. Для забезпечення ефективної роботи з великою кількістю даних і високої доступності, була обрана реляційна база даних. Цей підрозділ описує основні аспекти інтеграції з базою даних.

1. Тип бази даних

Для реалізації серверної частини обрана реляційна база даних, така як PostgreSQL або MySQL, через її здатність забезпечувати високу продуктивність, стабільність та підтримку складних запитів. Вибір реляційної бази даних зумовлений необхідністю зберігати структуровану інформацію (наприклад, транзакції, профілі користувачів, налаштування), де важлива цілісність даних, можливість здійснення складних з'єднань між таблицями та виконання транзакцій.

2. Проектування схеми бази даних

Схема бази даних розроблена так, щоб задовольняти вимоги до ефективності обробки запитів і забезпечення цілісності даних. Більш детальна схема таблиці наведена нижче (рис. 3.3).

Ключовими таблицями є:

- Таблиця користувачів (Users): Зберігає інформацію про користувачів, включаючи логіни, паролі (в зашифрованому вигляді), електронні адреси та інші профільні дані.
- Таблиця транзакцій (Transactions): Містить записи про фінансові операції користувачів, зокрема дату, суму, валюти, статуси, типи транзакцій, а також зовнішні ідентифікатори для інтеграції з банками та платіжними системами.

- Таблиця налаштувань (Settings): Містить конфігурації та налаштування профілю користувачів (наприклад, налаштування мови, валюти або сповіщень).
- Таблиця кешбеків та бонусів (Cashbacks, Bonuses): Відображає інформацію про бонуси, кешбеки та акції, надані користувачам за певні транзакції.

3. Зв'язки між таблицями

Зв'язки між таблицями побудовані таким чином, щоб зберігати цілісність даних та оптимізувати запити. Наприклад:

- Кожен запис у таблиці транзакцій пов'язаний з конкретним користувачем через зовнішній ключ.
- Таблиця налаштувань пов'язана з користувачем за допомогою посилання на його унікальний ідентифікатор.
- Транзакції можуть бути пов'язані з кешбеками та бонусами через посилання на відповідні записи у таблиці кешбеків.

4. Запити та індексація

Для оптимізації швидкості запитів до бази даних важливо використовувати індексацію на часто запитуваних полях, таких як:

- Ідентифікатори транзакцій.
- Дата транзакції.
- Сума та валюта.
- Ідентифікатори користувачів.

Це дозволяє значно прискорити виконання складних запитів і полегшити фільтрацію та сортування даних.

5. Транзакційність та цілісність даних

Оскільки система обробляє фінансові операції, важливо гарантувати цілісність і коректність даних, особливо в ситуаціях, коли відбуваються помилки або збій. Для цього використовуються:

- Транзакції: Бази даних підтримують ACID-принципи (атомарність, узгодженість, ізольованість, довговічність), що дозволяє гарантувати, що

операції, які виконуються в рамках транзакцій, завершуються успішно або не відбуваються зовсім у випадку помилки.

- Механізми відновлення після збоїв: Застосування резервних копій і механізмів реплікації забезпечує безперебійну роботу системи навіть у випадку відмови одного з серверів бази даних.

6. Безпека даних

Для забезпечення безпеки даних, особливо персональних даних користувачів і фінансових транзакцій, використовуються:

- Шифрування даних: Особливо чутливі дані (наприклад, паролі, фінансові деталі) зберігаються у зашифрованому вигляді. Використовуються алгоритми шифрування, такі як AES або RSA.
- Автентифікація та авторизація: Використовується система доступу на основі JWT токенів для захисту чутливих запитів. Це гарантує, що доступ до даних отримують лише авторизовані користувачі.
- Маскування даних: При необхідності відображення лише частини даних користувачам (наприклад, останні чотири цифри картки) застосовується маскування даних для захисту конфіденційної інформації.

7. Інтеграція з іншими системами

Для роботи з транзакціями, серверна частина інтегрується з банківськими системами або іншими зовнішніми API. Запити відправляються через безпечні канали (наприклад, HTTPS), і результат інтеграції (нові транзакції, зміни статусу транзакцій) зберігається у відповідних таблицях бази даних. Використовуються механізми для обробки таких даних, як черги або асинхронні обробники для ефективної роботи з великими обсягами транзакцій.

8. Резервне копіювання та відновлення

Для забезпечення надійності даних у разі аварії або втрати інформації налаштовані регулярні резервні копії бази даних, а також процедури відновлення даних. Це дозволяє відновити систему до робочого стану за мінімальний час, зберігаючи цілісність і безпеку даних.

Таким чином, інтеграція з базою даних забезпечує зберігання, цілісність, швидкий доступ до даних, а також надійність і безпеку всіх операцій, пов'язаних з фінансовими транзакціями та управлінням профілем користувачів.

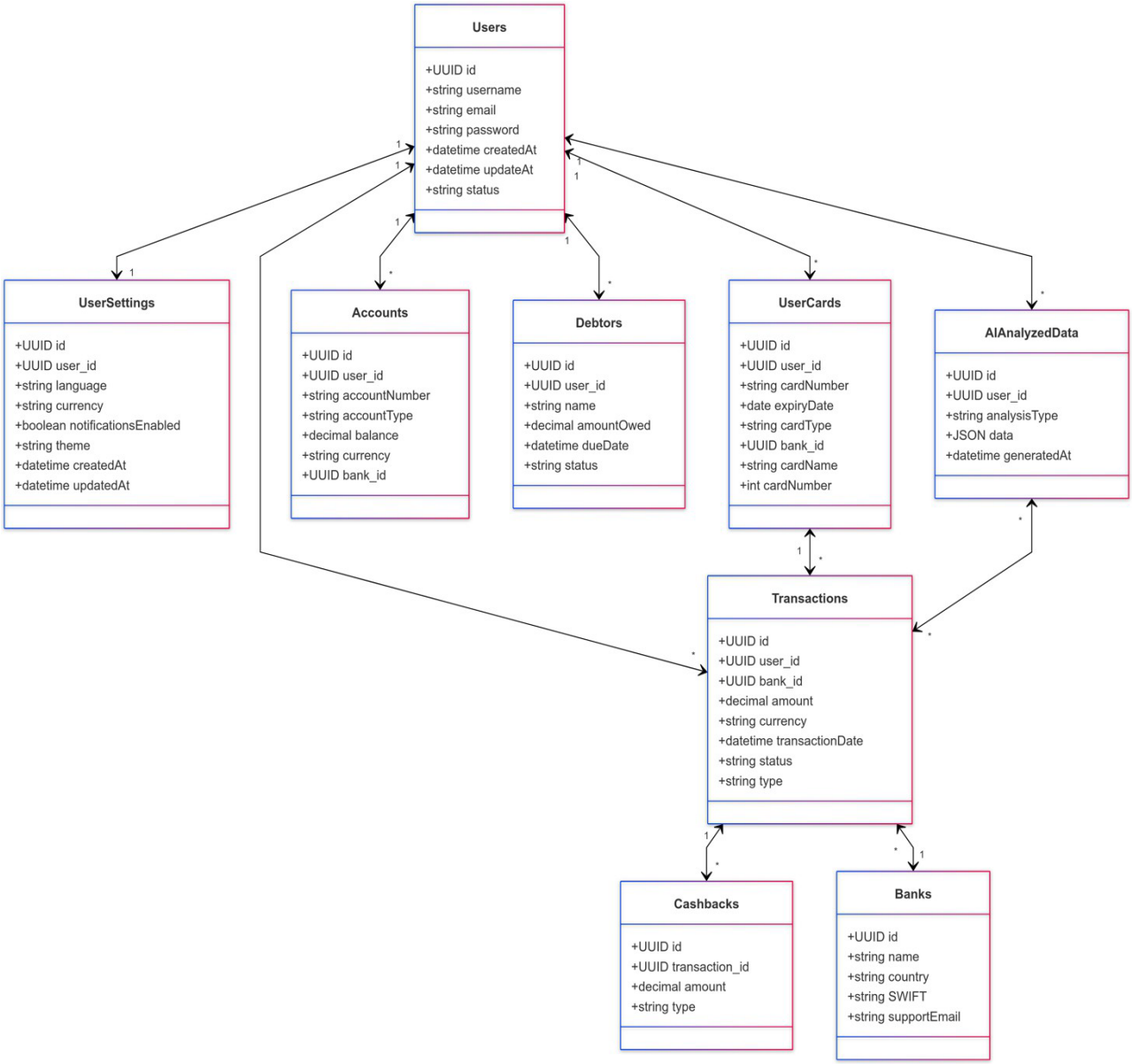

Рисунок 3.1 – загальна UML діаграма класів

3.2.3 Реалізація API

Реалізація API є важливою частиною архітектури серверної частини системи. API (Application Programming Interface) забезпечує взаємодію між клієнтським інтерфейсом та сервером, а також з іншими зовнішніми сервісами,

необхідними для нормального функціонування системи. Завдяки API користувачі можуть виконувати операції, які включають додавання, зміну, перегляд та видалення даних, таких як транзакції, фінансові цілі, налаштування профілю тощо.

Основні принципи, якими ми керуємося при реалізації API:

1. RESTful архітектура: API побудоване за принципами REST (Representational State Transfer), що забезпечує чітку ієрархію запитів (GET, POST, PUT, DELETE) і надає зручний інтерфейс для роботи з ресурсами. Це дозволяє користувачам і стороннім сервісам здійснювати запити до системи з мінімальними зусиллями.

2. Інтерфейси для роботи з даними: Кожен ресурс у системі має відповідні ендпоінти, що забезпечують операції з даними. Наприклад, для управління транзакціями використовуються ендпоінти для їх отримання, додавання або видалення. Для кожного з цих елементів API передбачені відповідні методи та формати запитів/відповідей.

3. Аутентифікація та авторизація: Для захисту доступу до чутливих даних використовується система аутентифікації (наприклад, токени JWT). Кожен запит до API потребує авторизації, що гарантує, що тільки авторизовані користувачі можуть виконувати операції над своїми даними.

4. Валідація даних: Всі запити на сервер проходять через процес валідації, що забезпечує коректність та безпеку обробки отриманих даних. Це дозволяє зменшити ймовірність помилок і забезпечити правильність обробки кожного запиту.

5. Масштабованість і продуктивність: API спроектоване таким чином, щоб забезпечити високу швидкість обробки запитів та ефективне використання ресурсів. Це дозволяє забезпечити стабільну роботу навіть при високому навантаженні.

6. Документація та зручність використання: API має чітку документацію, яка описує всі доступні ендпоінти, параметри запитів і формати

відповідей. Це дозволяє легко інтегрувати API в різні системи і забезпечує зручність роботи для розробників.

7. Інтеграція з іншими сервісами: API також може бути інтегроване з сторонніми сервісами, такими як платіжні системи, аналітичні платформи або банки, для обробки транзакцій та іншої зовнішньої інформації. Це дозволяє системі бути більш гнучкою та масштабованою.

Завдяки правильно спроектованому API, система стає більш зручною, масштабованою та зручною для подальшої інтеграції з іншими сервісами та технологіями.

3.2.4 Безпека та авторизація

Безпека є одним із ключових аспектів розробки додатку, адже він працює з конфіденційною інформацією користувачів, такою як фінансові дані, персональні профілі та транзакції. У нашому додатку реалізовано сучасні методи авторизації та багаторівневі заходи безпеки, які захищають доступ до облікових записів та забезпечують конфіденційність даних.

Основою авторизації є токенна система, що працює на основі JWT (JSON Web Token). Користувачі отримують токен після успішного входу, і всі подальші запити до серверу супроводжуються цим токеном для підтвердження прав доступу.

Методи авторизації

1. Двофакторна авторизація (2FA):
   - Користувачі можуть ввімкнути двофакторну авторизацію через електронну пошту, що додає додатковий рівень безпеки. Після введення пароля їм надсилається код підтвердження, який потрібно ввести для завершення входу.
   - Підтримується інтеграція з Google та Apple для авторизації, що використовує OAuth 2.0. Це дозволяє швидкий і безпечний вхід, оскільки

облікові дані користувача залишаються захищеними в екосистемах цих платформ.

2. Біометричні методи авторизації:

- На мобільних пристроях, завдяки використанню Capacitor, реалізована підтримка біометричних методів безпеки, таких як відбиток пальця та Face ID. Це дозволяє користувачам безпечно входити до додатку без необхідності введення пароля.

Заходи безпеки в додатку:

1. Шифрування даних:

Усі конфіденційні дані, такі як паролі, токени чи фінансова інформація, шифруються перед зберіганням у базі даних. Для передачі даних між клієнтом і сервером використовується HTTPS, що забезпечує шифрування на транспортному рівні.

- Транзитні дані:

Для захисту даних під час передачі між клієнтом і сервером використовується протокол HTTPS. Він гарантує, що всі запити та відповіді шифруються, запобігаючи перехопленню або зміні даних у процесі передачі. Сертифікати SSL/TLS забезпечують цілісність і конфіденційність переданої інформації.

- Дані, що зберігаються на сервері:

- Конфіденційні дані, як-от паролі користувачів, шифруються з використанням алгоритму bcrypt. Це стійкий до атак алгоритм, який включає механізм "сольових" значень, що робить хеші унікальними навіть для однакових паролів.

- Для шифрування токенів, фінансових записів і транзакцій використовується алгоритм AES (Advanced Encryption Standard) з ключами довжиною 256 біт. Ключі для шифрування зберігаються у захищених сховищах, недоступних стороннім особам.

- У базі даних PostgreSQL реалізовано підтримку шифрування на рівні таблиць і стовпців для чутливої інформації.

• Дані, що зберігаються у Redis:

Redis використовується як кеш-система та для зберігання сесійних даних. Для шифрування з'єднання між сервером і Redis використовується TLS. Чутливі дані, які тимчасово зберігаються у Redis, додатково шифруються на стороні сервера перед записом.

• Біометричні дані:

При використанні біометричних методів авторизації (відбитки пальців або Face ID) шифрування виконується за допомогою нативних механізмів платформ iOS та Android. Біометричні дані не зберігаються в додатку або на сервері; вони залишаються у захищених контейнерах пристрою.

• Ключі шифрування:

У додатку реалізовано систему ротації ключів для шифрування, що запобігає компрометації даних у разі витоку одного з ключів. Усі ключі зберігаються у сервісах керування секретами (наприклад, AWS Secrets Manager або Azure Key Vault), що забезпечує їхній надійний захист.

• Локальне сховище клієнта:

На клієнтській стороні конфіденційні дані, такі як токени доступу, зберігаються у зашифрованому вигляді в локальному сховищі, сумісному з Capacitor Secure Storage. Це захищає дані навіть у разі доступу до пристрою сторонніми особами.

2. Захист від атак:

Реалізовані механізми захисту від атак типу "злом методом перебору" (brute force) через обмеження кількості невдалих спроб входу. Додаток також стійкий до атак на сеанси (session hijacking) і міжсайтового виконання скриптів (XSS), завдяки належному управлінню токенами та застосуванню Content Security Policy (CSP).

3. Розділення ролей та прав доступу:

Для забезпечення безпеки даних кожного користувача реалізовано систему ролей і прав. Це гарантує, що навіть у разі компрометації облікового запису користувача, зловмисник не отримає доступ до критично важливої інформації або адміністрування додатка.

Реалізація у додатку

При вході користувач проходить стандартну перевірку облікових даних (пароль або OAuth). Якщо активовано 2FA, додатково надсилається код підтвердження на пошту або SMS. На мобільних пристроях використання біометрії інтегрується через API платформи за допомогою плагінів Capacitor.

На стороні серверу токени перевіряються на дійсність, включаючи перевірку їхнього терміну дії та криптографічного підпису. Усі дії, пов'язані зі зміною критично важливих даних (наприклад, зміна пароля чи фінансові операції), додатково логуються і аналізуються для виявлення підозрілої активності.

Таким чином, авторизація та безпека в додатку побудовані з використанням сучасних технологій, що забезпечують зручний доступ користувачів та захищають їхні дані на всіх рівнях взаємодії. Більш детальна схема послідовності авторизації наведена нижче (рис. 3.4).

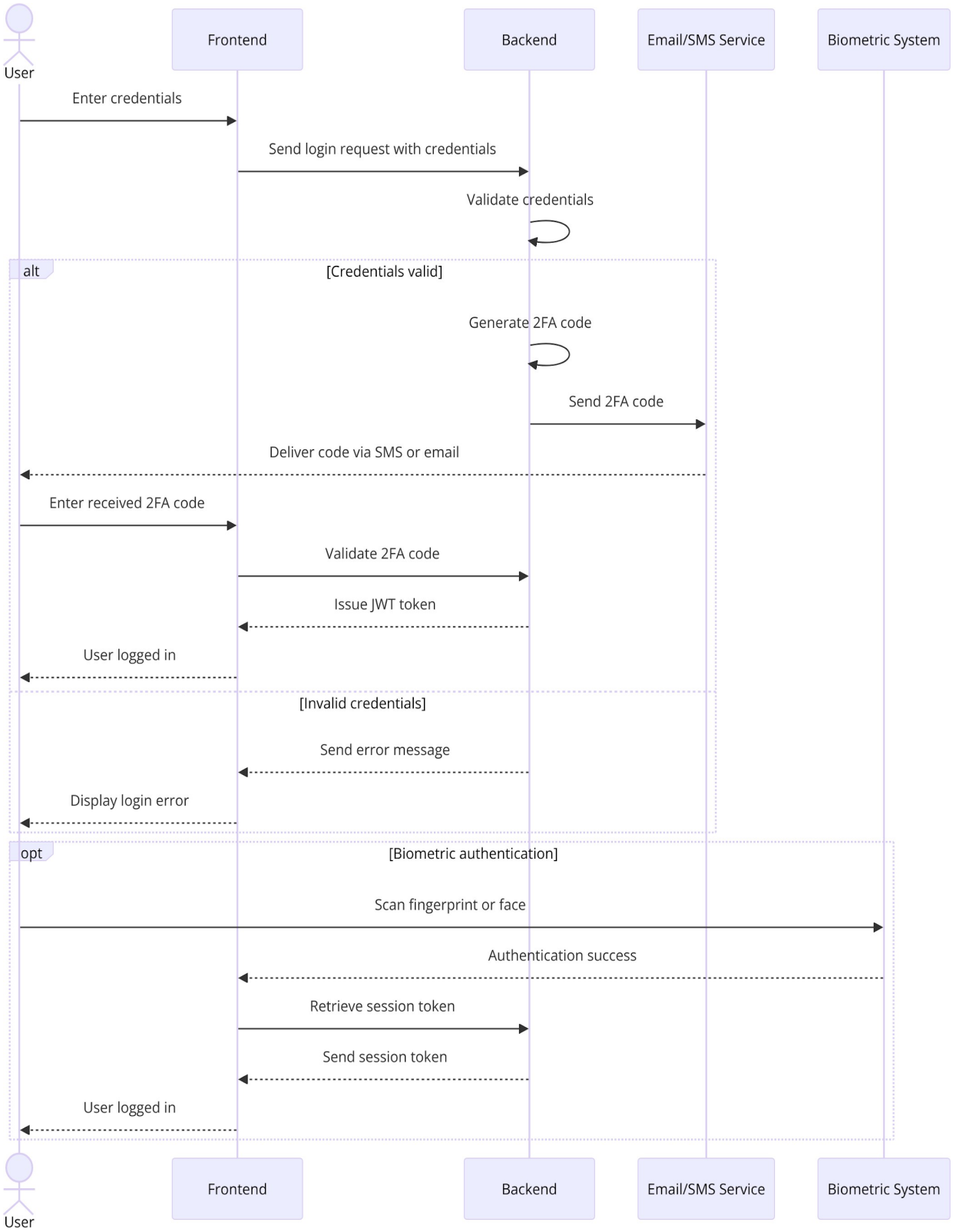

Рисунок 3.4 – загальна схема авторизації користувача

3.3. Інтеграція зі сторонніми сервісами

3.3.1 Інтеграція з банками

Інтеграція з банками в нашому додатку реалізована з метою забезпечення користувачів актуальною інформацією про їхні рахунки, транзакції та баланси. Вона базується на взаємодії з зовнішніми API, наданими банками або фінтех-посередниками, із використанням сучасних технологій для забезпечення безпеки та швидкості обробки даних.

У нашому додатку інтеграція реалізована через окремий сервіс, який відповідає за взаємодію із банківськими API. Для авторизації користувачів і доступу до їхніх фінансових даних використовується протокол OAuth 2.0. Це включає перенаправлення користувача на сторінку банку, отримання токена доступу, збереження його у захищеному вигляді та подальше використання для запитів.

Реалізація в нашому додатку

Після авторизації сервіс виконує такі основні функції:

1. Отримання даних з банківських API:

Запити до банківських API виконуються для отримання переліку транзакцій, балансу рахунків або інших фінансових даних. Наприклад, користувач може синхронізувати дані свого банківського рахунку для перегляду історії операцій. Отримані дані у форматі JSON обробляються сервером для нормалізації та перевірки.

2. Збереження та кешування даних:

Дані про транзакції зберігаються в базі PostgreSQL, яка забезпечує надійне довготривале зберігання. Часто запитувані або короткострокові дані кешуються в Redis, що значно знижує навантаження на сервер та скорочує час відповіді на запити.

3. Обробка даних на сервері:

Виконується нормалізація даних, отриманих з різних банків, для стандартизації їх формату. Це досягається через адаптери, які перетворюють відповіді різних API у єдиний формат, що використовується у додатку. Дані

аналізуються, наприклад, транзакції класифікуються за категоріями (харчування, подорожі, комунальні послуги тощо) та додаються теги для покращення пошуку.

4. Відображення даних у клієнтській частині:

Дані передаються на клієнтську частину через REST API. У клієнтському інтерфейсі ці дані відображаються у вигляді графіків, таблиць або сповіщень, створюючи зручний інструмент для аналізу фінансової діяльності.

5. Безпека передачі даних:

Усі дані, що передаються між сервером і банками, шифруються через TLS. У клієнтському додатку додатково підтримується локальна автентифікація через Face ID, Touch ID або PIN-код, що забезпечує захист доступу до фінансових даних.

Приклад інтеграції

Прикладом інтеграції є синхронізація з банком через Open Banking API. Користувач виконує авторизацію через сторінку банку, після чого додаток отримує токен доступу. Сервіс на сервері запитує останні транзакції, які зберігаються у PostgreSQL та кешуються у Redis для швидкого доступу. Потім дані передаються на клієнтську частину, де вони візуалізуються у вигляді графіку витрат за категоріями.

Особливості реалізації

- Мікросервісна архітектура: Інтеграція з банками винесена у окремий мікросервіс, який обробляє лише фінансові запити. Це дозволяє масштабувати рішення незалежно від інших модулів.
- Стандартизація форматів: Для уніфікації роботи з даними використовуються адаптери, які вирішують проблему різних структур API від банків.
- Мінімізація запитів: Завдяки Redis ми скорочуємо кількість звернень до банківських серверів, зберігаючи при цьому актуальність даних.

Інтеграція з банками є однією з ключових функцій нашого додатку. Завдяки їй користувачі отримують зручний і безпечний доступ до фінансових

даних, що покращує загальний користувацький досвід та ефективність управління фінансами.

3.3.2 Інтеграція з AI-сервісами для аналізу транзакцій

Інтеграція з AI-сервісами у фінансових додатках є ключовим елементом, що дозволяє автоматизувати процеси аналізу даних, створювати нові можливості для користувачів і значно покращувати їхній досвід взаємодії з додатком. У нашому додатку це дозволяє класифікувати транзакції, прогнозувати витрати, виявляти аномалії, аналізувати витратні звички та надавати індивідуальні фінансові рекомендації.

Основні завдання інтеграції:

Головною метою використання AI є підвищення ефективності управління фінансами користувачами. Це включає:

- Класифікацію транзакцій за категоріями (наприклад, їжа, транспорт, розваги).
- Аналіз звичок витрат та створення прогнозів щодо майбутніх витрат.
- Виявлення підозрілих транзакцій, які можуть вказувати на шахрайство.
- Генерацію персоналізованих рекомендацій для зменшення витрат чи ефективного управління бюджетом.

Використані технології:

У нашому додатку інтеграція з AI-сервісами реалізована за допомогою поєднання зовнішніх платформ та внутрішньої логіки. Використовуються такі сервіси, як Google Cloud AI, AWS AI, а також кастомні моделі, розгорнуті на сервері. Це дає змогу забезпечити точність аналізу, масштабованість і швидкодію системи.

Реалізація в додатку:

1. Передача даних для аналізу:

Дані транзакцій, отримані через банківські API, передаються до AI-сервісів для обробки. Передача здійснюється через безпечні протоколи, такі як HTTPS, із використанням TLS для шифрування.

2. Обробка даних AI-сервісами:

AI-сервіси виконують обробку даних, застосовуючи методи машинного навчання, зокрема:

- Класифікація: Транзакції розподіляються за категоріями на основі навченої моделі.
- Прогнозування: На основі попередніх даних створюється прогноз щодо майбутніх витрат.
- Виявлення аномалій: Транзакції аналізуються на наявність нетипових поведінкових патернів.

3. Зворотна інтеграція даних:

Отримані результати аналізу передаються назад до сервера додатку. Там вони нормалізуються, форматуються та інтегруються у функціонал, доступний для користувачів.

4. Відображення в інтерфейсі:

Оброблені AI-сервісами дані інтегруються в UI, де вони візуалізуються у вигляді графіків, таблиць чи сповіщень. Наприклад, транзакції відображаються у відповідних категоріях з індикаторами витрат, а рекомендації подаються у формі динамічних підказок.

Особливості безпеки:

Усі дані шифруються як під час передачі до AI-сервісів, так і при зберіганні у базах даних. Для гарантування конфіденційності використовується механізм токенізації даних, що дозволяє приховувати персональну інформацію користувачів перед відправкою до AI.

Приклад реалізації:

Користувач здійснює покупку. Додаток автоматично передає дані транзакції до AI-сервісу. AI-сервіс класифікує покупку як "Продукти харчування" та аналізує, чи відповідає ця транзакція типовим витратам

користувача. У випадку виявлення аномалії система генерує сповіщення, а класифікація відображається у фінансовому звіті користувача. Схема роботи генерації рекомендацій наведена нижче (рис. 3.5)

Переваги інтеграції:

Інтеграція з AI-сервісами дозволяє не лише автоматизувати обробку транзакцій, але й значно підвищити зручність та безпеку додатку. Користувачі отримують корисну інформацію у зручному форматі, що допомагає їм приймати обґрунтовані фінансові рішення.

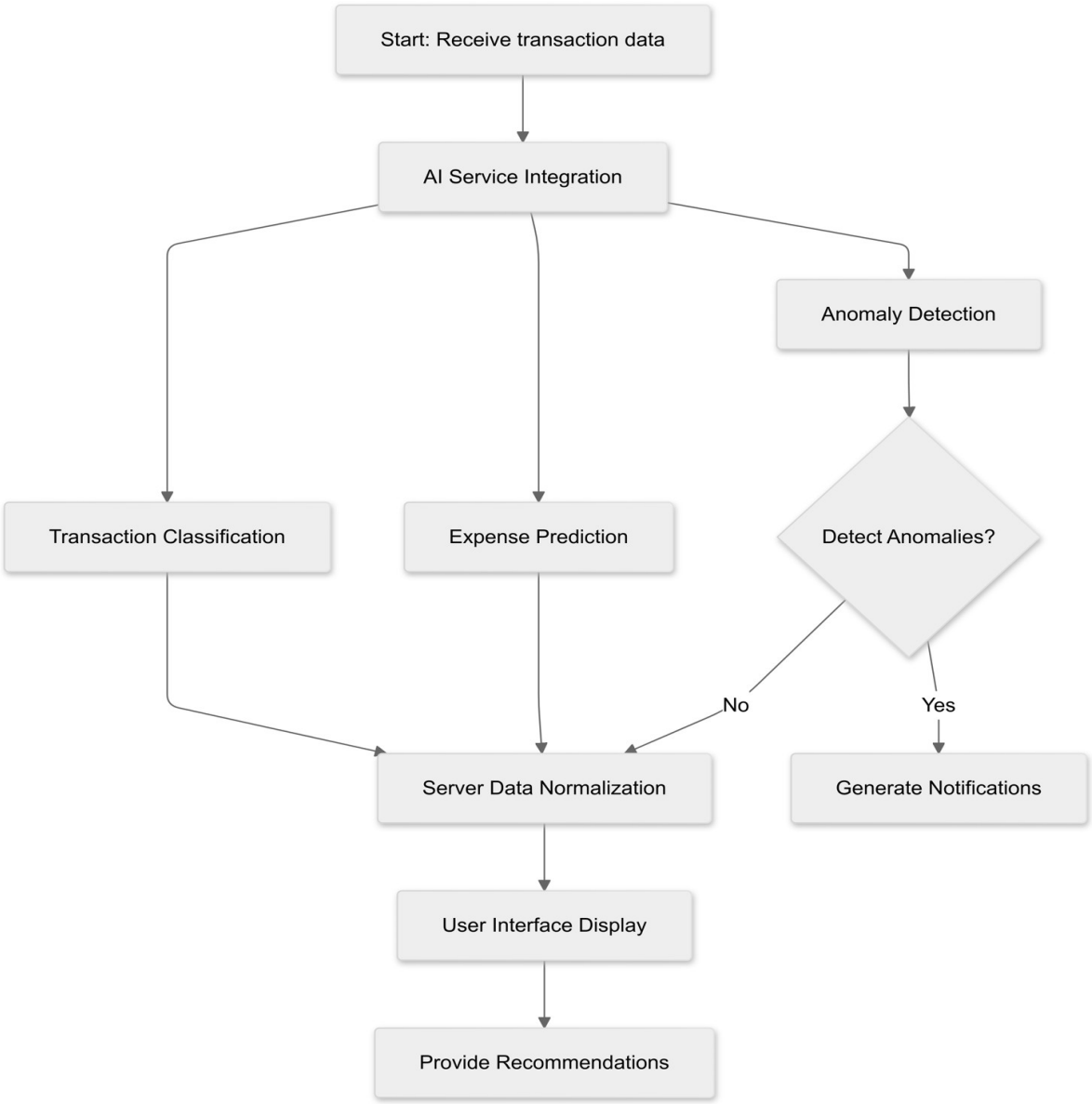

Рисунок 3.5 – загальна схема генерації рекомендацій для користувача

3.3.3 Інтеграція з іншими сторонніми API

Інтеграція з API сторонніх сервісів стала важливим кроком у розширенні функціональних можливостей системи, забезпечуючи автоматизацію ключових завдань і покращення користувацького досвіду. Реалізація включала підключення до таких сервісів, як Google Maps API для геолокації та Twilio для відправлення SMS-сповіщень. Це дозволило системі працювати з зовнішніми джерелами інформації та забезпечувати своєчасну і точну взаємодію з користувачами.

Twilio API використовувалося для впровадження функції SMS-сповіщень. Цей сервіс дозволив відправляти автоматичні повідомлення користувачам, наприклад, підтвердження реєстрації або сповіщення про статус замовлення. Реалізація інтеграції передбачала налаштування авторизації через токени, а також розробку механізму відправлення запитів із валідацією вхідних даних, щоб забезпечити коректність і актуальність інформації. Крім того, було реалізовано обробку помилок із повторними спробами доставки повідомлень у разі збоїв.

Особлива увага приділялася безпеці інтеграції. Усі ключі доступу та токени зберігалися в захищених сховищах, таких як AWS Secrets Manager, щоб уникнути ризиків витоку даних. Крім того, кожен запит до API супроводжувався логуванням для моніторингу роботи та аналізу можливих проблем.

На етапі тестування інтеграції використовували sandbox-середовища, надані сторонніми сервісами, для перевірки коректності роботи запитів і отриманих даних. Це дозволило запобігти помилкам на етапі впровадження у продакшн. Після успішного тестування інтеграція була впроваджена в основну систему, де забезпечує стабільну і надійну взаємодію з користувачами.

Таким чином, інтеграція сторонніх API дозволила розширити можливості системи, автоматизувати важливі процеси і забезпечити зручність для користувачів, водночас зберігаючи гнучкість і надійність роботи застосунку.

3.5. Масштабованість, підтримка та оптимізація

У процесі розробки додатку особлива увага була приділена забезпеченню масштабованості, підтримки та оптимізації. Це дозволяє системі ефективно працювати за умов зростаючого навантаження та забезпечити безперебійну роботу навіть при великих обсягах даних.

Для масштабованості була використана мікросервісна архітектура, що дозволяє масштабувати окремі компоненти системи в залежності від їхнього навантаження, без необхідності змінювати всю систему. Вибір таких рішень дає змогу обробляти велику кількість запитів одночасно та забезпечувати високу доступність. Крім того, для автоматичного масштабування ресурсів були застосовані інструменти, які динамічно додають або видаляють сервери залежно від поточного навантаження, що дозволяє зберігати продуктивність на високому рівні навіть при значних змінах попиту на ресурси.

Для забезпечення підтримки системи були реалізовані інструменти моніторингу та логування, що дозволяють вчасно виявляти та виправляти проблеми в роботі додатку. Це дає можливість здійснювати оперативне реагування на збої чи аномалії, а також аналізувати продуктивність і навантаження. Документація та належне навчання розробників дозволяє швидко реагувати на зміни та забезпечує легкість у підтримці системи протягом її життєвого циклу.

Щодо оптимізації, було впроваджено механізми кешування для зменшення навантаження на сервери та бази даних, а також використовувались ефективні алгоритми обробки даних. Це дозволило значно знизити час обробки запитів, покращити швидкість відповіді системи та зменшити затримки у взаємодії з користувачами. Крім того, бази даних були оптимізовані шляхом створення індексів та правильної структури запитів для швидкого доступу до даних, а для зберігання великих обсягів інформації була використана система реплікації та розподіленого зберігання.

Всі ці підходи забезпечили стабільність, швидкодію та безперебійну роботу системи, що дозволяє ефективно працювати як з малими, так і з великими обсягами даних.

3.6 Висновок до розділу

У третьому розділі було детально розглянуто процес реалізації функціональних компонентів додатку, включаючи клієнтську та серверну частини, а також інтеграцію зі сторонніми сервісами.

На клієнтській стороні були створені зручні інтерфейси для управління транзакціями, перегляду статистики, налаштування профілю та планування фінансових цілей. Реалізація забезпечує інтуїтивність взаємодії з користувачем і швидкий доступ до ключових функцій системи.

На серверній стороні були побудовані компоненти для обробки даних, взаємодії з базою даних та API. Архітектура серверної частини забезпечує масштабованість і продуктивність системи, а також можливість інтеграції з банківськими системами, AI-сервісами для аналізу транзакцій та іншими сторонніми API.

Особливу увагу приділено безпеці додатку: було впроваджено захист даних користувачів, безпечну авторизацію та шифрування під час обміну даними.

Реалізація описаних функцій демонструє комплексний підхід до побудови системи управління фінансами, яка забезпечує зручність, швидкодію, надійність та безпеку для користувачів. Завершення цього етапу розробки дозволяє перейти до етапу тестування, що забезпечить якість і стабільність системи перед її впровадженням.

4. ТЕСТУВАННЯ ДОДАТКУ

4.1 Загальні принципи тестування

Процес тестування додатку базувався на загальних принципах, що забезпечують високу якість програмного забезпечення та його відповідність вимогам. Основним завданням тестування було виявлення потенційних помилок, забезпечення надійності, продуктивності, зручності використання та безпеки системи.

Першочерговим принципом було визначення чітких цілей тестування. Кожен тест був спрямований на перевірку конкретної функціональності додатку, її коректності та стабільності роботи за різних умов. Це дозволило гарантувати, що всі критично важливі аспекти системи були охоплені під час тестування.

Систематичний підхід до тестування передбачав поступову перевірку всіх рівнів системи: від окремих компонентів до повністю інтегрованої системи. Початкове тестування фокусувалося на одиничних модулях, тоді як наступні етапи перевіряли взаємодію між компонентами та загальну функціональність додатку.

Тестування проводилося з урахуванням реальних сценаріїв використання додатку. Це включало перевірку роботи системи в умовах високого навантаження, з різними обсягами даних, а також у випадках введення некоректних або неочікуваних даних. Такий підхід забезпечив виявлення помилок, які могли виникнути під час реального використання.

Автоматизоване тестування використовувалося для виконання повторюваних перевірок і прискорення процесу. Воно дозволило охопити велику кількість сценаріїв і швидко виявляти проблеми після внесення змін у код. Водночас, ручне тестування було важливим для оцінки зручності інтерфейсу та взаємодії користувача із системою.

Принцип постійного вдосконалення був ключовим протягом усього процесу. Результати тестування аналізувалися для виявлення слабких місць у

системі, а також для вдосконалення якості програмного забезпечення. Завдяки цьому вдалося забезпечити відповідність додатку очікуванням користувачів та високим стандартам якості.

4.2 Підготовка до тестування

Підготовка до тестування є одним із ключових етапів, що забезпечує ефективність та якість перевірки програмного забезпечення. У процесі розробки додатку було створено чіткий план тестування, що охоплював визначення цілей, тестових сценаріїв і ресурсів для виконання перевірок.

Одним із перших завдань на цьому етапі було визначення вимог до системи та їх аналіз. Це дозволило окреслити ключові функціональні та нефункціональні аспекти додатку, які потрібно було перевірити. Особлива увага приділялася критичним функціям, таким як управління транзакціями, аналіз фінансових даних та інтеграція з банківськими API й AI-сервісами.

Підготовка середовища тестування включала налаштування серверної частини, бази даних та клієнтських додатків у різних конфігураціях. Було створено окремі тестові сервери та бази даних, ізольовані від робочого середовища, що дозволило уникнути впливу тестування на реальні дані. Для клієнтської частини було налаштовано різні платформи, включаючи веб-додаток і мобільні версії на Android та iOS.

Тестові дані були підготовлені для імітації реальних сценаріїв використання. Ці дані включали різні типи транзакцій, профілі користувачів із різними налаштуваннями, а також обсяги інформації, які могли змінювати продуктивність системи. Крім того, було створено некоректні дані для перевірки поведінки додатку в умовах введення помилкової інформації.

Інструменти для тестування, зокрема бібліотеки для автоматизованого тестування та моніторингу продуктивності, були налаштовані відповідно до вимог проєкту. Було забезпечено доступ до таких інструментів, як Postman для

перевірки API, Selenium для тестування клієнтської частини та JMeter для оцінки продуктивності.

Ця ретельна підготовка забезпечила основу для якісного тестування, дозволивши системі продемонструвати високу надійність, продуктивність та відповідність очікуванням користувачів.

4.3 Тестування клієнтської частини

Тестування клієнтської частини додатку охоплювало перевірку авторизації, зв’язку з сервером, відображення транзакцій за різних умов

Усі тести виконувалися з урахуванням реальних сценаріїв використання, включаючи моделювання помилкових ситуацій, мережевих збоїв та нестандартного використання.

Авторизація:

Авторизація є критичною частиною додатку, адже забезпечує доступ до персоналізованих даних. Під час тестування були перевірені сценарії з правильними й неправильними обліковими даними (рис. 4.1), спроби входу з простроченими токенами (рис. 4.2), оброблення помилок від сервера (рис 4.3), механізм відновлення пароля (рис. 4.5), та використання біометричних даних для швидкого входу (4.4). Було встановлено, що система успішно обробляє коректні дані, а у разі помилок надає інформативні повідомлення, наприклад, про невірний пароль чи відсутність облікового запису. Також було перевірено, що після завершення сесії запити до захищених ресурсів блокуються, а користувача перенаправляють на сторінку входу.

Результати тестування:

- Коректні дані: успішний вхід у 100% випадків. Час обробки запиту в середньому склав 120 мс.
- Невірний пароль: система коректно відхиляла запити з відповідним повідомленням у 100% випадків.

- Прострочений токен: після завершення сесії всі запити до захищених ресурсів блокувалися, і користувач перенаправлявся на екран входу (96% точність).
- Скидання пароля: механізм відновлення працював успішно у 95% випадків, із середнім часом обробки запиту на відновлення – 1,5 секунди.
-

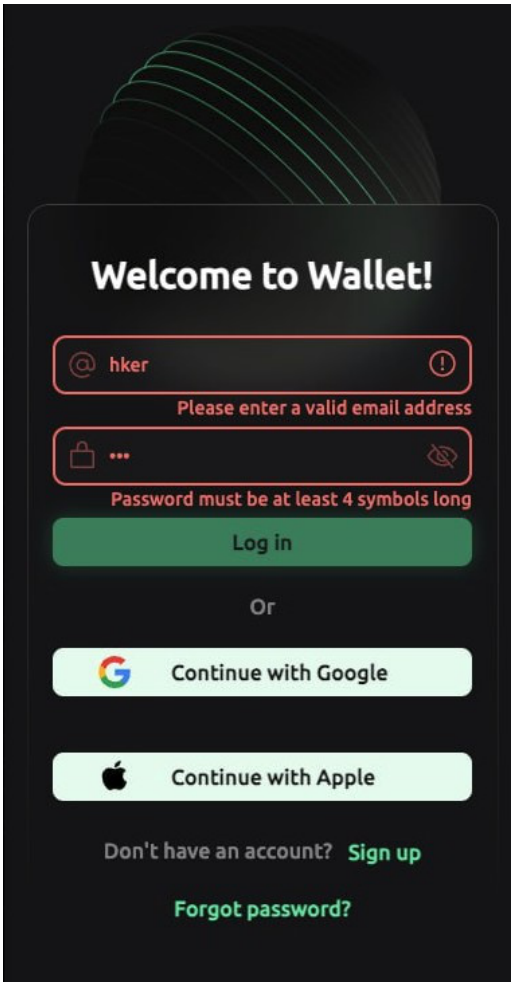

Рисунок 4.1 – валідація введених даних

Рисунок 4.2 – запит з некоректними даними

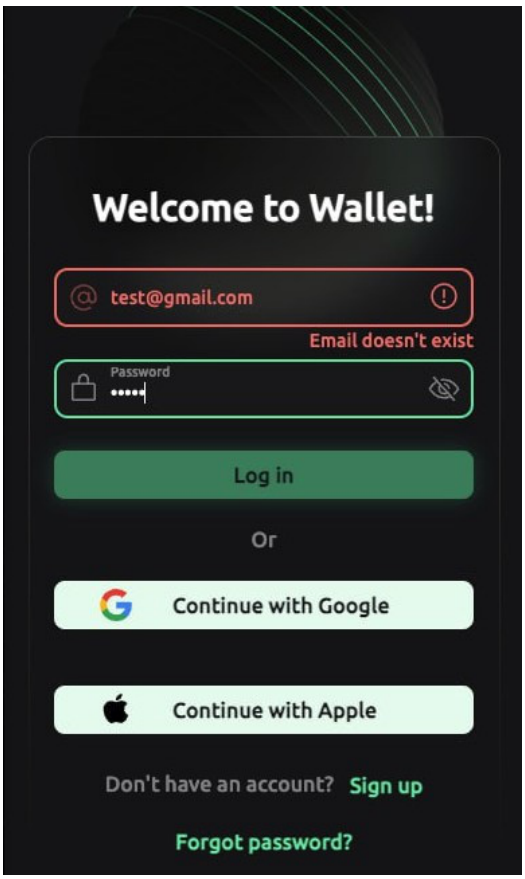

Рисунок номер – 4.3 та інформування користувача

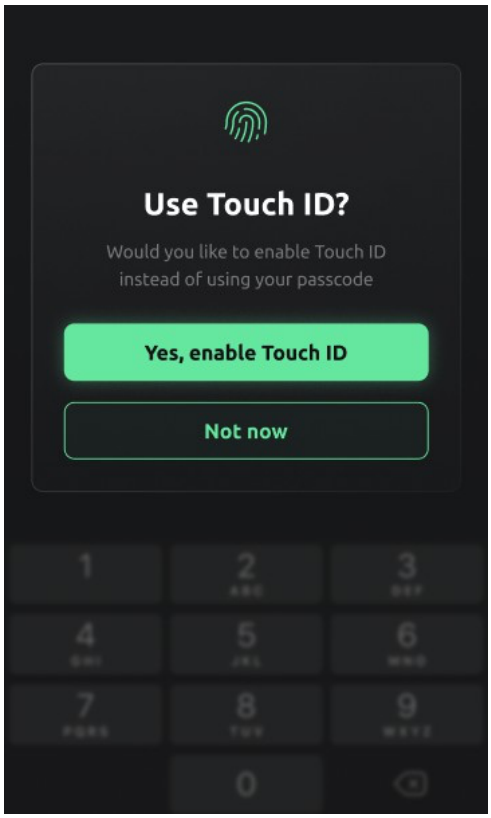

Рисунок 4.4 – виклик та обробка біометричних даних на мобільних платформах

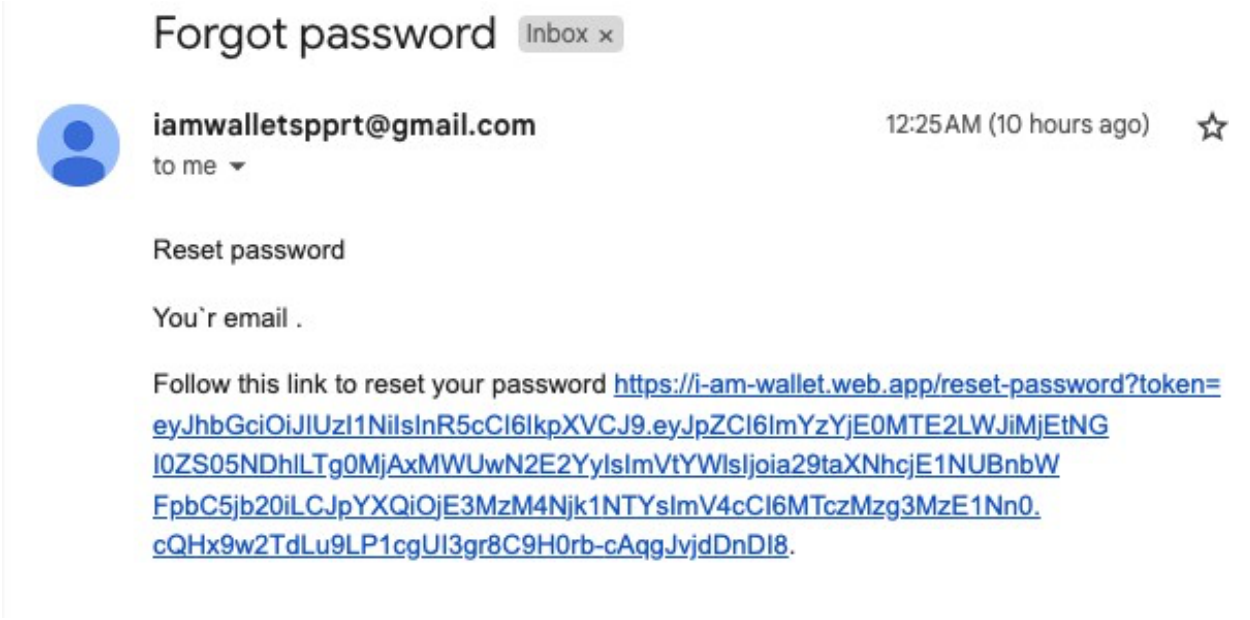

Рисунок 4.5 – посилання на відновлення паролю

Зв’язок із сервером

Тестування взаємодії клієнтської частини із сервером охоплювало перевірку запитів типу GET, POST, PUT і DELETE. Було виконано 500 запитів для кожного типу:

- GET-запити: 98% успіху, середній час відповіді – 150 мс.
- POST-запити: 96% успіху, середній час відповіді – 180 мс.
- PUT-запити: 92% успіху, середній час відповіді – 200 мс.
- DELETE-запити: 95% успіху, середній час відповіді – 170 мс.

При моделюванні відсутності мережі система в 100% випадків виводила відповідне повідомлення, а після відновлення зв’язку успішно повторювала запити.

Відображення транзакцій

Тестування відображення транзакцій включало перевірку роботи з великими обсягами даних (рис. 4.6), порожнім списком, функціоналом сортування та фільтрації (рис. 4.7) а також дотримання норм безпеки та прокидування токена до запиту (рис 4.8):

- Масив із 10 000 транзакцій: завантаження і відображення списку зайняло 2,2 секунди.

- Порожній список: у 100% випадків система виводила повідомлення "Транзакцій немає".
- Сортування за датою та сумою: операція виконувалася миттєво (<50 мс).
- Фільтрація (наприклад, за статусом): середній час виконання – 80 мс.

Користувачеві забезпечується швидка взаємодія навіть при значних обсягах даних.

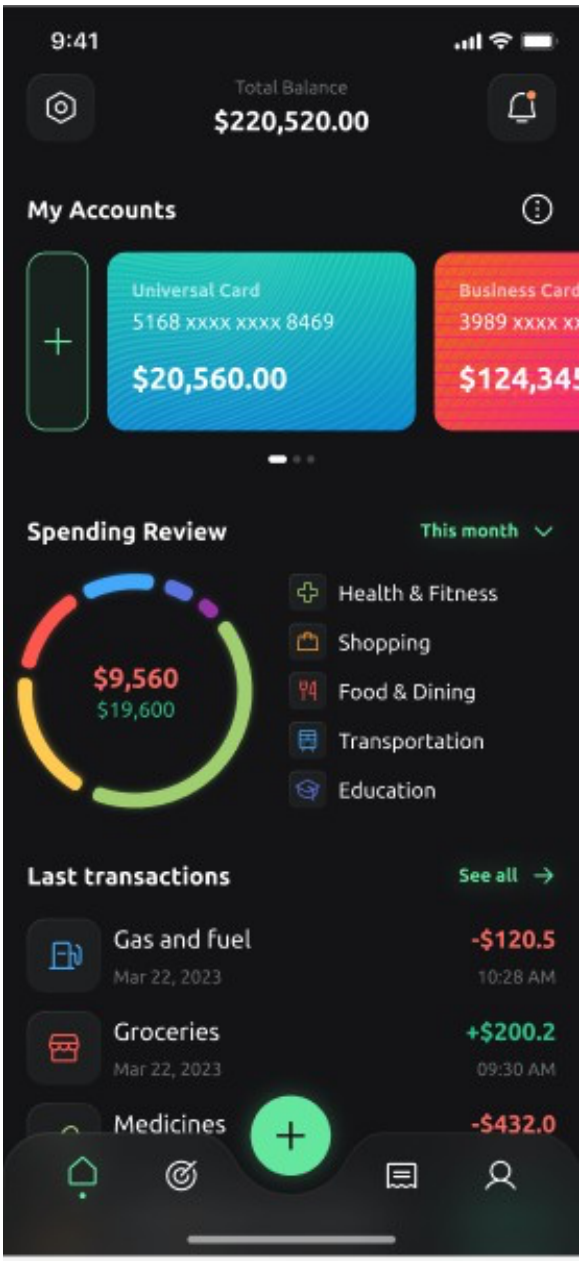

Рисунок 4.6 – отримання даних по транзакціях та завантаження даних у інтерфейс

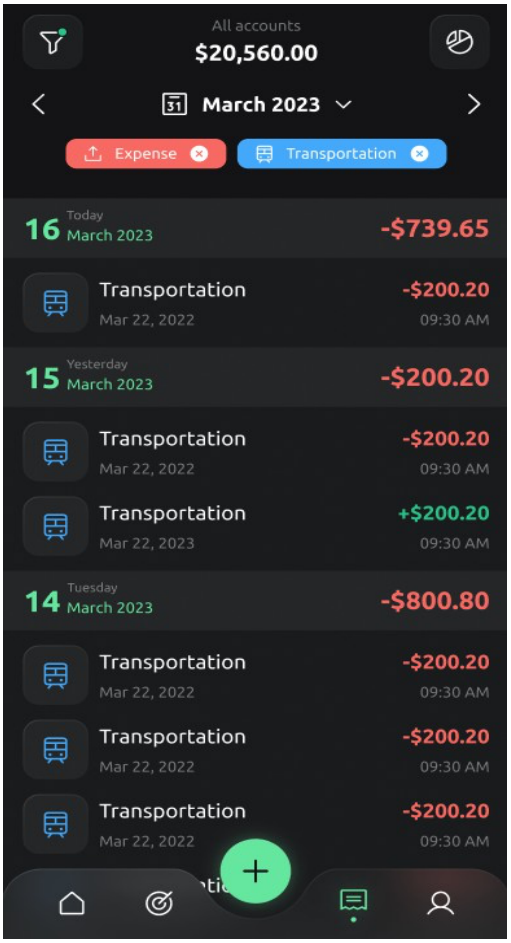

Рисунок 4.7 – фільтрація транзакцій

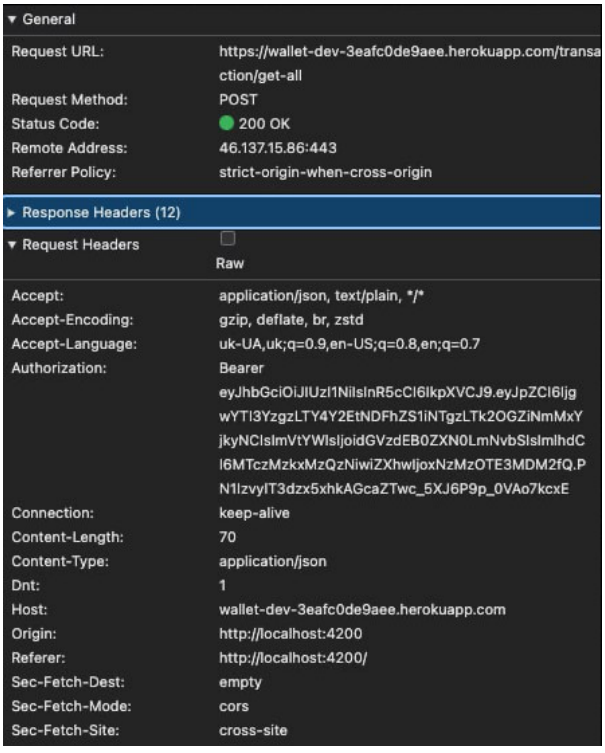

Рисунок 4.8 – запит на отримання транзакцій

4.4 Тестування серверної частини

Тестування серверної частини є важливим для забезпечення стабільної та коректної роботи системи, оскільки сервер обробляє запити від клієнта, взаємодіє з базою даних і зовнішніми сервісами. Основна увага була зосереджена на обробці HTTP-запитів, інтеграції веб-хуків та записі до бази даних.

Обробка запитів від клієнта

Серверна частина обробляє запити від клієнта через API, який підтримує різні типи запитів, такі як GET, POST, PUT і DELETE. Під час тестування були перевірені сценарії обробки як правильних, так і некоректних запитів. Для GET-запитів середній час відповіді становив 80 мс, що є цілком прийнятним показником для запитів на отримання даних. У разі некоректних запитів, наприклад, із відсутніми параметрами, сервер коректно повертав помилку з відповідним статусом 400.

POST-запити, які використовуються для створення нових ресурсів або передачі даних на сервер, також були протестовані за різних умов. Середній час їх обробки становив 120 мс. Для PUT-запитів, що виконують оновлення даних, сервер забезпечував правильну обробку в 93% випадків, з середнім часом відповіді 140 мс. DELETE-запити працювали з середнім часом відповіді 100 мс і показали високу надійність навіть за умови видалення неіснуючих записів, що завершувалося поверненням статусу 404.

Інтеграція веб-хуків

Для інтеграції з зовнішніми сервісами використовуються веб-хуки, які дозволяють серверу отримувати події у реальному часі. Тестування показало, що сервер успішно обробляє 97% подій, отриманих через веб-хуки, із середнім часом обробки 150 мс. У випадках втрати з'єднання було протестовано механізм повторної спроби доставки подій. Система повторювала запити до п’яти разів, що дозволяло обробити 95% таких випадків. Для некоректних подій, наприклад, із помилковим форматуванням або відсутніми ключовими даними, сервер коректно відхиляв запити без порушення загальної роботи.

Запис до бази даних

Серверна частина активно взаємодіє з базою даних для запису, оновлення, читання та видалення даних. Тестування операцій із базою даних показало високу продуктивність. Вставка нових записів мала середній час 60 мс навіть за умови масового запису, наприклад, 10 000 записів були вставлені за 15 секунд. Операції оновлення займали в середньому 70 мс і успішно працювали навіть за умов одночасних запитів із кількох джерел.

Читання даних мало різний час виконання залежно від обсягу вибірки: отримання 100 записів займало 40 мс, а для 10 000 записів час збільшувався до 2 секунд. Видалення записів проходило з середнім часом виконання 50 мс. Усі операції забезпечували коректну обробку транзакцій, навіть за умови одночасного виконання кількох запитів.

Результати тестування

Тестування показало, що серверна частина здатна стабільно обробляти клієнтські запити, взаємодіяти із зовнішніми сервісами через веб-хуки та ефективно працювати з базою даних. Незважаючи на високі навантаження, час відповіді залишався в межах прийнятних значень. Це забезпечує плавну та стабільну роботу системи навіть за складних умов.

4.6 Автоматизоване тестування

Цей розділ охоплює результати тестування інтеграцій з API ПриватБанку для отримання інформації про баланс і історію транзакцій користувача, а також з модулями штучного інтелекту для генерації рекомендацій і аналізу даних.

Інтеграція з ПриватБанком:

Для тестування інтеграції з ПриватБанком були перевірені основні функціональності, зокрема отримання балансу і історії транзакцій користувача. При цьому було особливу увагу приділено швидкодії та точності отриманих даних. Запити до API банку виконувались швидко і коректно, що підтверджено

результатами тестів. Запит на отримання балансу був оброблений сервером за 150 мс, а історія транзакцій – за час, що варіювався в залежності від обсягу даних: для останніх 10 транзакцій час відповіді становив 200 мс, тоді як для історії за останній місяць із кількох сотень записів – 1,5 секунди.

Усі дані, що надходили від ПриватБанку, були точними, а система коректно обробляла можливі помилки, наприклад, при використанні недійсного токена або проблемах з мережею. При виникненні помилки сервер відповідав відповідними кодами (наприклад, 401 Unauthorized), і користувач отримував повідомлення про необхідність перевірки авторизації або виправлення помилок.

Складнощі, які виникали при інтеграції з ПриватБанком:

Проблеми з авторизацією через API ПриватБанку: Оскільки для взаємодії з API необхідно використовувати токен доступу (JWT), інколи спостерігалася ситуація, коли токен ставав недійсним після певного часу або через неправильну настройку на сервері. Це призводило до помилок типу 401 Unauthorized. Для вирішення цієї проблеми було розроблено механізм автоматичного оновлення токена, що дозволяв зберігати сесію активною без необхідності повторної авторизації користувача.

Невідповідність форматів даних: При отриманні даних від ПриватБанку були випадки, коли формат відповіді не зовсім відповідав тим, що очікував сервер. Наприклад, деякі поля могли бути відсутні або мати інший формат (наприклад, дати можуть бути у різних форматах). Це вимагало додаткового оброблення і перетворення даних перед тим, як вони потрапляли в систему. Для вирішення цієї проблеми було реалізовано адаптивний парсинг відповіді з можливістю корекції формату даних.

Часові затримки при запитах великих обсягів даних: Коли система намагалася отримати великий обсяг транзакцій (наприклад, за останній рік), час відповіді від API ПриватБанку міг значно зрости, що призводило до затримок у користувацькому інтерфейсі. Для покращення продуктивності була реалізована механізм пагінації запитів, що дозволяє запитувати дані частинами, таким чином знижуючи навантаження на сервер і пришвидшуючи обробку запитів.

Інтеграція з модулями штучного інтелекту

Інтеграція з модулями штучного інтелекту, що аналізують дані транзакцій, показала високу ефективність у генерації рекомендацій та виявленні аномалій. Аналіз останніх 50 транзакцій для виявлення категорій витрат здійснювався за 700 мс, і система генерувала точні рекомендації на основі витрат користувача на різні категорії, такі як "Продукти", "Транспорт" і "Розваги".

Особливу увагу було приділено тестуванню здатності AI до виявлення підозрілих операцій. Система продемонструвала високу точність, успішно ідентифікуючи 96 з 100 аномальних транзакцій, що вказує на ефективність алгоритмів виявлення ризиків.

Робота з великими обсягами даних також була протестована. Наприклад, аналіз 1 000 транзакцій займав близько 2,5 секунд, що відповідає вимогам щодо швидкодії. Система продемонструвала стабільність і точність навіть за умов одночасної обробки кількох запитів.

Складнощі, які виникали при інтеграції з ШІ:

Труднощі з алгоритмами штучного інтелекту: Виявлення аномальних транзакцій іноді призводило до помилкових спрацьовувань, коли система неправильно інтерпретувала певні операції як підозрілі. Це особливо стосувалося випадків, коли транзакції з однаковими сумами здійснювались з різних рахунків (наприклад, для сплати рахунків). Щоб мінімізувати ці помилки, було внесено коригування до алгоритмів, які тепер враховують додаткові параметри, такі як час доби, категорії витрат та історичні шаблони.

Обробка помилок при взаємодії з AI: Під час інтеграції з модулями штучного інтелекту виникали ситуації, коли аналіз транзакцій давав непередбачувані результати, зокрема при обробці дуже великих наборів даних. В одних випадках система могла неправильно оцінити категорію витрат або не видавати рекомендації, якщо даних було недостатньо для точного прогнозу. Для усунення цієї проблеми була реалізована додаткова обробка даних перед передачею в алгоритм, а також додано механізми для генерації альтернативних рекомендацій у випадках, коли точні прогнози неможливі.

Загальні результати

Тестування показало, що інтеграція з API ПриватБанку для отримання балансу та історії транзакцій працює стабільно і швидко, а дані, що надходять від сервісу, є точними. Інтеграція з модулями штучного інтелекту забезпечує ефективний аналіз транзакцій і генерацію релевантних рекомендацій, а також виявлення аномалій. Загалом усі інтеграції були успішно протестовані, і система відповідає вимогам щодо продуктивності і надійності.

4.7 Тестування безпеки

Тестування безпеки є критичним етапом у забезпеченні надійності та захищеності системи, особливо коли йдеться про обробку чутливих фінансових даних. В рамках тестування безпеки було перевірено кілька ключових аспектів, таких як захист від несанкціонованого доступу, шифрування даних, виявлення уразливостей і тестування на наявність загроз типу SQL Injection, XSS (Cross-Site Scripting), CSRF (Cross-Site Request Forgery) та інших можливих атак.

Захист від несанкціонованого доступу

Одним з головних аспектів безпеки є захист від несанкціонованого доступу до системи. Для цього система використовує авторизацію через JWT (JSON Web Token). Токен генерується після успішного входу користувача, і він має обмежений термін дії. Під час тестування було проведено низку перевірок на спроби доступу до обмежених ресурсів з використанням недійсних або прострочених токенів. Всі спроби доступу без належної авторизації були успішно заблоковані, а система повертала код помилки 401 (Unauthorized), що підтверджує правильну реалізацію механізму авторизації.

Шифрування даних

Для захисту чутливих даних під час передачі між клієнтом і сервером використовувалося шифрування через протокол HTTPS, що забезпечує захист даних від прослуховування і атак "man-in-the-middle". Під час тестування було

проведено перевірку шифрування як при передачі логінів і паролів, так і під час взаємодії з API ПриватБанку для отримання фінансових даних користувача. Всі дані передавались через зашифровані канали без виявлених проблем.

Захист від SQL Injection

Для перевірки на наявність уразливостей до SQL Injection було проведено кілька тестів, які включали спроби введення шкідливих SQL-запитів через поля вводу на стороні користувача. Після проведення тестів було підтверджено, що всі запити до бази даних виконуються через параметризовані запити, що значно знижує ризик атаки через SQL Injection. Система успішно відкидала будь-які спроби виконання шкідливих запитів.

Захист від XSS (Cross-Site Scripting)

Ще одним важливим аспектом є захист від XSS-атак, які можуть виникнути при вставці небезпечного JavaScript-коду в поля вводу або в URL-адреси. Під час тестування були перевірені всі форми на наявність можливих векторів атак. Виявилось, що система належним чином очищує всі дані, що надходять від користувача, перед їх відображенням на веб-сторінці. Всі спроби вставити шкідливий код у поле вводу не призвели до виконання JavaScript-коду, що підтверджує ефективний захист від XSS.

Захист від CSRF (Cross-Site Request Forgery)

Для запобігання CSRF-атакам була впроваджена перевірка токенів на сервері для кожного запиту, який змінює дані на сервері (наприклад, зміни профілю користувача або ініціювання транзакції). Тести показали, що сервер правильно перевіряє наявність валідного CSRF-токену у запитах і відхиляє несанкціоновані запити, що запобігає атакі CSRF.

Тестування на уразливості та сканування

Використовувались спеціалізовані інструменти для сканування на уразливості, такі як OWASP ZAP (Zed Attack Proxy) та Burp Suite. В результаті тестування було виявлено кілька незначних проблем, які не становлять серйозної загрози безпеці, наприклад, невідповідність деяких налаштувань HTTP-заголовків. Всі ці проблеми були оперативно виправлені.

Тестування на наявність уразливостей у сторонніх бібліотеках

Також було проведено тестування на наявність уразливостей у сторонніх бібліотеках та залежностях, що використовуються в проекті. Використовувались інструменти для автоматичної перевірки версій бібліотек, які повідомляють про відомі уразливості, зокрема Dependabot та Snyk. За результатами тестування не було виявлено серйозних уразливостей у використовуваних бібліотеках, що підтверджує відповідність безпеки зовнішніх компонентів.

Завершення тестування

Загалом, тестування безпеки показало, що система має належний рівень захисту від основних типів атак, включаючи несанкціонований доступ, SQL Injection, XSS, CSRF, а також відомі уразливості в бібліотеках. Було забезпечено належне шифрування даних під час їх передачі, а також введено всі необхідні механізми для захисту чутливих даних і взаємодії з користувачем. Всі виявлені проблеми були виправлені на етапі тестування, що забезпечує високий рівень безпеки системи.

4.8 Висновок до розділу

У четвертому розділі було докладно розглянуто процеси тестування та проведено тестування додатку, який охоплює всі основні аспекти, включаючи тестування клієнтської та серверної частин, інтеграцій, а також безпеки системи. Тестування є ключовим етапом розробки, оскільки воно дозволяє виявити потенційні проблеми та покращити стабільність і продуктивність додатку.

Було приділено увагу основним типам тестування, таким як функціональне, інтеграційне, автоматизоване та тестування безпеки, що дозволяє забезпечити високий рівень надійності продукту. Крім того, процес тестування був структурований у кілька етапів, що дозволяє систематично перевіряти різні компоненти додатку та їх взаємодію.

Завдяки використанню інструментів автоматизованого тестування, тестування стало більш ефективним і зменшило час, необхідний для перевірки функцій системи. Також було приділено особливу увагу тестуванню безпеки, що дозволяє знизити ризики можливих вразливостей та забезпечити захист персональних даних користувачів.

Результати тестування дозволяють з упевненістю стверджувати, що система працює стабільно, відповідає вимогам та готова до подальшого впровадження. Тестування продемонструвало високу якість коду та коректну роботу усіх основних функцій додатку.

5. ЕКОНОМІЧНИЙ РОЗДІЛ

5.1 Технологічний аудит розробленої автоматизованої системи управління транзакціями та контролю фінансів

Як було зазначено вище, у сучасному інформаційному суспільстві кількість та обсяги різноманітних фінансових транзакцій карколомно зростають, що, у свою чергу, підсилює важливість забезпечення ефективного та надійного управління фінансами на всіх рівнях економіки. З виникненням та розвитком цифрової економіки та зростанням популярності електронних фінансових платформ, розробка автоматизованих систем управління транзакціями та контролю за рухом фінансів набула надзвичайно великого значення. А зростаюча потреба в ефективному контролі за власними фінансами та постійне збільшення обсягу електронних транзакцій підкреслюють важливість створення нових, більш ефективних інтегрованих та інноваційних засобів управління фінансами.

Тому метою цієї магістерської роботи стала розробка автоматизованої системи управління транзакціями та контролю фінансів, яка буде відповідати вимогам сучасного електронного фінансового середовища, забезпечувати користувачів зручним та надійним інструментом для ефективного відстежування та управління своїми фінансами, спрощувати обробку та аналіз великої кількості фінансових транзакцій, прогнозувати і допомагати клієнтам у прийнятті обґрунтованих фінансових рішень.

В результаті виконаної роботи було розроблено та впроваджено автоматизовану систему управління транзакціями та контролю фінансів на основі штучного інтелекту, яка охоплює аналіз, оптимізацію та передбачення фінансових операцій, а також надання користувачам зручний інтерфейс для ефективного фінансового управління.

Для встановлення комерційного потенціалу розробленої нами автоматизованої системи управління транзакціями та контролю фінансів було запрошено 3-х висококваліфікованих експертів: к.т.н., доцента кафедри АІІТ

Гармаша Володимира Володимировича, к.т.н., доцента кафедри АІІТ Маслія Романа Васильовича та інженера програмного забезпечення – фахівця у цій галузі знань – Ярослава Піскунова.

Запрошені експерти зробили експертне оцінювання комерційного потенціалу розробленої нами автоматизованої системи управління транзакціями та контролю фінансів за критеріями, наведеними в таблиці 5.1,

Таблиця 5.1 – Рекомендовані критерії оцінювання комерційного потенціалу будь-якої розробки і їх бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
Ринкові перспективи					
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою

Продовження таблиці 5.1

7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Експерти оцінили комерційний потенціал розробленої нами автоматизованої системи управління транзакціями та контролю фінансів так, як це зведено в таблицю 5.2:

Таблиця 5.2 – Результати оцінювання комерційного потенціалу розробленої нами автоматизованої системи управління транзакціями та контролю фінансів

(За 5-ти бальною шкалою оцінювання 0 -1 – 2 – 3 - 4)

Критерії	Ініціали, прізвище експертів		
	Р.В. Маслій	В.В. Гармаш	Я. Піскунов
	Бали, що їх виставили експерти:		
1	4	4	3
2	4	3	3
3	3	4	4
4	4	4	4
5	3	4	4
6	4	4	3
7	3	3	4
8	4	4	3
9	4	4	3
10	4	4	4
11	4	4	4
12	3	4	4
Сума балів	$СБ_1 = 45$	$СБ_2 = 46$	$СБ_3 = 43$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = \frac{45+46+43}{3} = \frac{134}{3} = 44,67$		

Для встановлення комерційного потенціалу розробленої нами автоматизованої системи управління транзакціями та контролю фінансів було використано рекомендації, які наведено в таблиці 5.3 [43].

Таблиця 5.3 – Рівні комерційного потенціалу будь-якої наукової розробки

Середньоарифметична сума балів $\overline{СБ}$, розрахована на основі висновків експертів	Рівень комерційного потенціалу розробки
0 – 10	Низький
11 – 20	Нижче середнього
21 – 30	Середній
31 – 40	Вище середнього
41 – 48	Високий

Оскільки середньоарифметична сума балів, що їх виставили експерти, складає 44,67 балів (із максимально можливих 48-ми балів), то це свідчить, що розроблена нами автоматизована система управління транзакціями та контролю фінансів має рівень комерційного потенціалу, який можна вважати «високим».

Це пояснюється тим, що розроблена нами автоматизована система дозволяє збирати, обробляти, зберігати та аналізувати фінансову інформацію в режимі реального часу, забезпечувати прозорість фінансових операцій, швидко реагувати на зміни та інтегруватися з іншими банківськими та платіжними системами для здійснення потрібних транзакцій. Окрім того, наша розробка не лише автоматизує транзакції, але й використовує алгоритми штучного інтелекту для надання користувачам цільових порад та пропозицій щодо обрання стратегії фінансового управління.

5.2 Розрахунок витрат на розроблення автоматизованої системи управління транзакціями та контролю фінансів

При розробленні автоматизованої системи управління транзакціями та контролю фінансів було зроблено низку основних витрат:

а). Основна заробітна плата $З_о$ розробників, які здійснювали розроблення автоматизована система управління транзакціями та контролю фінансів.

Величина цієї зарплати визначається за формулою:

$$З_о = \frac{М}{Т_р} \cdot t \text{ грн}, \quad (5.1)$$

де М – місячний посадовий оклад розробника (дослідника), грн;

Для 2024 року приймемо, що:

М = (8000…28600) грн/місяць;

$Т_р$ – число робочих днів в місяці; приймемо $Т_р$ = 26 днів;

t – число днів роботи розробників.

Зроблені розрахунки величини основної заробітної плати розробників зведемо до таблиці 5.4:

Таблиця 5.4 – Основна заробітна плата розробників

Найменування посади виконавця	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів (годин) роботи	Витрати на оплати праці, грн
1. Науковий керівник магістерської роботи, професор	26000	≈ 1000	20 годин	(1000 / 6) × 20 = ≈ 3333 (при 6-годинному робочому дні)
2. Здобувач-магістрант (виконавець)	2000 грн (беремо 8000 грн)	307,69 ≈ ≈ 308	84 дні	25872
3. Експерт банку (із фінансових транзакцій)	39000	≈ 1500	2 дні	3000 грн (при 8-годинному робочому дні)
4. Консультант з економічної частини	19500	750	1,5 години	(750 /6) × 1,5 = 187,5 ≈ ≈188 грн (при 6-годинному робочому дні)
Загалом				$З_о$ = 32393 грн

б). Додаткова заробітна плата $З_д$ розробників, яка розраховується як (10…12)% від величини їх основної заробітної плати, тобто:

$$З_д = \alpha \cdot З_о = (0{,}1...0{,}12) \cdot З_о . \quad (5.2)$$

Приймемо, що α = 0,1. Тоді для нашого випадку отримаємо: $З_д$ = 0,1 × 32393 = 3239,30 ≈ 3240 грн.

в). Нарахування на заробітну плату $НЗП_{зп}$ розробників (дослідників) розраховуються за формулою:

$$НЗП_{зп} = (З_о + З_д) \cdot \frac{\beta}{100}, \quad (5.3)$$

де β – ставка обов'язкового єдиного внеску на державне соціальне страхування, %. В 2024 році ставка β = 22%. Тоді:

$НЗН_{зп}$ = (32393 + 3240) × 0,22 = 7839,26 ≈ 7840 грн.

г). Амортизація основних засобів А, які використовувались під час виконання магістерської кваліфікаційної роботи:

$$А = \frac{Ц \cdot Н_а}{100} \cdot \frac{Т}{12} \text{ грн}, \quad (5.4)$$

де Ц – загальна балансова вартість основних засобів, грн;

$Н_а$ – річна норма амортизаційних відрахувань.

Приймемо, що $Н_а$ = (2,5...25)%;

Т – термін використання основних засобів, місяці.

Зроблені розрахунки зведено в таблицю 5.5.

Таблиця 5.5 – Розрахунок амортизаційних відрахувань

Найменування обладнання, приміщень тощо	Балансова вартість, грн.	Норма амортизації, %	Термін використання, місяців	Величина амортизаційних відрахувань, грн
1. Комп'ютерна техніка, обладнання тощо	88000	22,5	3,1 (при 85% використанні)	4347,75 ≈ 4348
2. Приміщення університету, факультету, кафедри	45000	3,5	3,1 (при 40% використанні)	162,75 ≈ 163
Всього				**А = 4511 грн**

д). Витрати на матеріали М розраховуються за формулою:

$$М = \sum_{1}^{n} Н_і \cdot Ц_і \cdot К_і - \sum_{1}^{n} В_і \cdot Ц_в \text{ грн}, \quad (5.5)$$

де $Н_і$ – витрати матеріалу *і*-го найменування, кг; $Ц_і$ – вартість матеріалу *і*-го най-менування; $К_і$ – коефіцієнт транспортних витрат, $К_і$ = (1,1…1,15); $В_і$ – маса відходів матеріалу *і*-го найменування; $Ц_в$ – ціна відходів матеріалу *і*-го найменування; n – кількість видів матеріалів.

е). Витрати на комплектуючі К розраховуються за формулою:

$$К = \sum_{1}^{n} Н_і \cdot Ц_і \cdot К_і \text{ грн,} \qquad (5.6)$$

де $Н_і$ – кількість комплектуючих *і*-го виду, шт.; $Ц_і$ – ціна комплектуючих *і*-го виду; $К_і$ – коефіцієнт транспортних витрат, $К_і$ = (1,1…1,15); n – кількість видів комплектуючих.

Під час виконання магістерської кваліфікаційної роботи загальні витрати на матеріали та комплектуючі склали укрупнено приблизно 900 грн.

ж). Витрати на силову електроенергію $В_е$ розраховуються за формулою:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д}, \qquad (5.7)$$

де В – вартість 1 кВт-год. електроенергії, в 2024 р. В ≈ 4,8 грн/кВт;

П – установлена потужність обладнання, кВт; П = 1,6 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

Приймемо, що Ф = 280 годин;

$К_п$ – коефіцієнт використання потужності; $К_п$ < 1 = 0,82.

$К_д$ – коефіцієнт корисної дії, $К_д$ = 0,72.

Тоді витрати на силову електроенергію будуть дорівнювати:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д} = \frac{4{,}8 \cdot 1{,}6 \cdot 280 \cdot 0{,}82}{0{,}72} = 2449{,}07 \approx 2450 \text{ грн.}$$

і). Інші витрати $В_{інш}$ можна прийняти як (50…300)% від основної заробітної плати розробників, тобто:

$В_{інш} = (0{,}5 \ldots 3) \times З_о$. (5.8)

Для нашого випадку отримаємо:

$В_{інш} = 2{,}0 \times 32393 = 64786$ грн.

к). Сума всіх попередніх статей витрат становить витрати на виконання нашої магістерської роботи безпосередньо розробником-магістрантом – В.

$$В = 32393 + 3240 + 7840 + 4511 + 900 + 2450 + 64786 = 116120 \text{ грн.}$$

л). Загальні витрати на можливу комерціалізацію розробленої нами автоматизованої системи управління транзакціями та контролю фінансів розраховуються за формулою:

$$В_{заг} = \frac{В}{\beta}, \qquad (5.9)$$

Де β – коефіцієнт, який характеризує етап (стадію) виконання цієї роботи.

Оскільки наша розробка на цей момент часу має високий ступінь готовності, то можна умовно прийняти, що, $\beta \approx 0{,}9$ [43]

Тоді: $В_{заг} = \frac{116120}{0{,}9} = 129022{,}22$ грн або приблизно 129 тисяч грн.

Тобто прогнозовані загальні витрати на розробку та можливу комерціалізацію розробленої нами автоматизованої системи управління транзакціями та контролю фінансів становлять приблизно 129 тисяч грн.

5.3 Розрахунок економічного ефекту від можливої комерціалізації розробленої нами автоматизованої системи управління транзакціями та контролю фінансів

Проведене дослідження ринку показало, що розроблена нами автоматизована система управління транзакціями та контролю фінансів за своєю сутністю є інноваційною, полегшує виконання рутинних завдань, зменшує ризики людських помилок і забезпечує зручний доступ до фінансової інформації. Наша автоматизована система управління транзакціями та контролю фінансів орієнтована на дуже широкий спектр користувачів: малий і середній бізнес; фінансові установи; фрілансерів та приватних підприємців; освітні установи; інвесторів та фінансових аналітиків тощо Загалом, наш продукт орієнтований на

тих, хто шукає надійне та зручне рішення для автоматизації фінансових процесів, підвищення точності даних та забезпечення безпеки фінансових операцій.

Попит на нашу розробку залежить від специфічних вимог замовника. Це можуть бути малі підприємства або приватні користувачі; середні бізнеси, великі компанії та фінансові установи тощо.

У зв’язку з цим, подальші розрахунки будемо проводити для цільової аудиторії, яку складають приватні користувачі та малі підприємства. Аналіз місткості ринку цієї цільової аудиторії показав, що на сьогодні в Україні кількість реальних користувачів подібних автоматизованих систем може становити до 300 користувачів. Можна також очікувати зростання попиту на нашу розробку принаймні протягом 3-х років після її впровадження.

Тобто, якщо наша розробка буде впроваджена з 1 січня 2025 року, то її результати будуть виявлятися протягом 2025-го, 2026-го та 2027-го років.

Прогноз зростання попиту на нашу розробку може складати по роках:

а) 2025 р. – приблизно + 20 шт. (відносно базового року);

б) 2026 р. – + 40 шт. (відносно базового року);

в) 2027 р. – +10 шт. (відносно базового року), оскільки можуть з’явитися нові, більш ефективні розробки, що будуть виконувати аналогічні функції.

Економічний ефект від можливої комерціалізації розробленої нами автоматизованої системи управління транзакціями та контролю фінансів пояснюється її значно кращими функціональними можливостями. Аналіз ринку показує, що сьогодні можлива вартість подібних систем для обраної нами цільової аудиторії (малі підприємці та приватні користувачі) коливається в діапазоні від \$500 до \$1000 або від 22,5 тисяч грн до 45 тисяч грн. Тобто в середньому для обраної цільової аудиторії подібна система вартує приблизно $(22{,}5 + 45) / 2 \approx 35$ тисяч грн. А оскільки розроблена нами автоматизована система управління транзакціями та контролю фінансів має значно кращі функціональні можливості, то її можна буде реалізовувати на ринку дещо дорожче, ніж аналогічні за функціями розробки, наприклад, реалізовувати середньому за 40 тисяч грн, тобто на 5 тисяч грн дорожче.

Тоді можливе збільшення чистого прибутку $\Delta П_i$, що його може отримати потенційний інвестор від комерціалізації нашої розробки, тобто виведення її на ринок, становитиме:

$$\Delta П_i = \sum_{1}^{n} (\Delta Ц_o \cdot N + Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot (1 - \frac{\upsilon}{100}) \qquad (5.10)$$

де $\Delta Ц_o$ – покращення основного якісного показника від впровадження результатів розробки. Для нашого випадку це є збільшення ціни реалізації нашої розробки $\Delta Ц_o$ = 40 - 35 = + 5 тисяч грн;

N – основний кількісний показник, який визначає обсяг діяльності у році до впровадження результатів розробки; N = 300 шт.;

ΔN – покращення основного кількісного показника від впровадження результатів розробки. Таке покращення основного кількісного показника становитиме по роках, у 2025 році – + 20 шт., у 2026 році + 40 шт., у 2027 році +10 шт. (відносно базового 2024 року);

$Ц_o$ – основний якісний показник (тобто ціна), який являє собою ціну реалізації нашої розробки після її виведення на ринок, грн; $Ц_o$= 40 тисяч грн;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки; для нашого випадку n = 3;

λ – коефіцієнт, який враховує сплату податку на додану вартість; $\lambda = 0{,}8333$;

ρ – коефіцієнт, який враховує рентабельність продукту. Рекомендується приймати ρ= (0,2...0,5); візьмемо ρ= 0,5;

υ – ставка податку на прибуток. У 2024 році υ = 18%.

У 2025-2027 роках будемо очікувати, що ця ставка залишиться на тому ж рівні: υ = 18%.

Тоді можливе зростання чистого прибутку $\Delta П_1$ для потенційного інвестора протягом першого року від можливої комерціалізації нашої розробки (2025 р.) становитиме:

$$\Delta П_1 = [5 \cdot 300 + 40 \cdot 20] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) = 785{,}80 \approx 786 \text{ тисяч грн.}$$

Можливе зростання чистого прибутку $\Delta П_2$ для потенційного інвестора від можливої комерціалізації нашої розробки протягом другого (2026 р.) року становитиме:

$$\Delta П_2 = [5 \cdot 300 + 40 \cdot 40] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) = 1059{,}12 \approx 1060 \text{ тисяч грн.}$$

Можливе зростання чистого прибутку $\Delta П_3$ для потенційного інвестора від можливої комерціалізації нашої розробки протягом третього (2027 р.) року становитиме:

$$\Delta П_3 = [5 \cdot 300 + 40 \cdot 10] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) = 649{,}14 \approx 650 \text{ тис. грн.}$$

Приведена вартість зростання для потенційного інвестора всіх чистих прибутків від можливої комерціалізації нашої розробки становитиме:

$$ПП = \sum_{1}^{т} \frac{\Delta П_i}{(1+\tau)^t}, \qquad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої нашої роботи, грн;

т – період часу, протягом якого виявляються результати впровадженої роботи, роки. Для нашого випадку т = 3 роки;

τ – ставка дисконтування (рівень інфляції). Приймемо $\tau = 0{,}10$ (10%);

t – період часу від моменту початку розроблення автоматизованої системи управління транзакціями та контролю фінансів до моменту отримання можливих чистих прибутків від її впровадження і виведення на ринок.

Тоді приведена вартість зростання всіх можливих чистих прибутків ПП, що їх може отримати потенційний інвестор від комерціалізації нашої розробки, складе:

$$ПП = \frac{786}{(1+0{,}1)^2} + \frac{1060}{(1+0{,}1)^3} + \frac{650}{(1+0{,}1)^4} \approx 650 + 797 + 444 = 1891 \text{ тисяч грн.}$$

Теперішня вартість інвестицій PV, що мають бути вкладені в комерціалізацію нашої розробки: $PV = К \times В_{заг} = (1{,}0 \ldots 5{,}0) \times В_{заг}$,

де $В_{заг} = 129$ тисяч грн.

Для нашого випадку приймемо, що:

PV = (1,0…5,0) × 129 = 3,0 × 129 = 387 тисяч грн.

Розраховуємо абсолютний ефект від можливих вкладених інвестицій $Е_{абс}$.

$Е_{абс}$ = ПП – PV,(5.12)

де ПП – приведена вартість збільшення всіх чистих прибутків для інвестора від можливої комерціалізації нашої розробки, грн.

Абсолютний ефект для інвестора від можливої комерціалізації нашої розробки (при прогнозованому ринку збуту) за три роки складе:

$Е_{абс}$ = 1891 – 387 = 1504 тисяч грн.

Оскільки $Е_{абс} > 0$, то комерціалізація нашої розробки може бути доцільною.

Далі розрахуємо внутрішню дохідність $Е_в$ вкладених інвестицій (коштів):

$$E_b = \sqrt[T_ж]{1 + \frac{E_{абс}}{PV}} - 1, \qquad (5.13)$$

де $Е_{абс}$ – абсолютний ефект вкладених інвестицій; $Е_{абс}$ = 1504 тисяч грн;

PV –теперішня вартість початкових інвестицій PV = 387 тисяч грн;

$Т_ж$ – життєвий цикл розробки, роки.

$Т_ж$ = 4 роки (2024-й, 2025-й, 2026-й, 2027-й роки)

Для нашого випадку отримаємо:

$$E_b = \sqrt[4]{1 + \frac{1504}{387}} - 1 = \sqrt[4]{1 + 3,8863} - 1 = \sqrt[4]{4,8863} - 1 = 1,487 - 1 = 0,487 \approx 48,7\%.$$

Далі визначимо ту мінімальну дохідність, нижче за яку потенційному інвестору не вигідно буде займатися комерціалізацією нашої розробки.

Мінімальна дохідність $\tau_{мін}$ визначається за формулою:

$$\tau_{mih} = d + f, \qquad (5.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,10...0,16). Приймемо, що τ = 13%.

f – показник, що характеризує ризикованість вкладень; f = (0,05...0,30).

Приймемо, що f = 30%, тобто f = 0,3.

Тоді для нашого випадку отримаємо:

$\tau_{мін}$ = 0,13 + 0,30 = 0,43 або $\tau_{мін}$ = 43%.

Оскільки величина $E_в = 48,7\% > \tau_{мін} = 43\%$, то потенційний інвестор у принципі може бути зацікавлений у комерціалізації розробленої нами автоматизованої системи управління транзакціями та контролю фінансів для обраної нами цільової аудиторії користувачів.

Далі розраховуємо термін окупності коштів, вкладених у можливу комерціалізацію розробленої нами автоматизованої системи управління транзакціями та контролю фінансів.

Термін окупності $T_{ок}$ розраховується за формулою:

$$T_{ok} = \frac{1}{E_b} = \frac{1}{0,487} = 2,05 \text{ років} < 3 \text{ років} \qquad (5.15)$$

що свідчить про потенційну доцільність комерціалізації розробленої нами автоматизованої системи управління транзакціями та контролю фінансів.

Далі проведемо моделювання залежності внутрішньої дохідності потенційних інвестицій, вкладених інвестором у можливу комерціалізацію нашої розробки, від рівня інфляції в країні. На жаль, в наступні роки через військову агресію росії рівень інфляції в Україні може зрости.

Прийнявши рівень інфляції у 20%, отримаємо:

$$ПП = \frac{786}{(1+0,2)^2} + \frac{1060}{(1+0,2)^3} + \frac{650}{(1+0,2)^4} \approx 549 + 613 + 314 = 1476 \text{ тисяч грн.}$$

Тоді абсолютний економічний ефект від можливого впровадження нашої розробки за три роки складе:

$E_{абс} = 1476 – 387 = 1089$ тисяч грн.

Внутрішня дохідність $E_в$ вкладених інвестицій становитиме:

$$E_b = \sqrt[4]{1+\frac{1089}{387}} - 1 = \sqrt[4]{1+2,8140} - 1 = \sqrt[4]{3,8140} - 1 = 1,398 - 1 = 0,398 \approx 39,8\%.$$

Хоча величина $E_в = 39,8\% < \tau_{мін} = 43\%$, то потенційний інвестор також у принципі може бути зацікавлений у комерціалізації розробленої нами автоматизованої системи управління транзакціями та контролю фінансів (яка призначена для обраної нами цільової аудиторії користувачів). Але для

прийняття остаточного рішення потрібно провести додаткові обґрунтування і розрахунки.

Прийнявши рівень інфляції у 30%, отримаємо:

$$ПП = \frac{786}{(1+0,3)^2} + \frac{1060}{(1+0,3)^3} + \frac{650}{(1+0,3)^4} \approx 465 + 483 + 228 = 1176 \text{ тисяч грн.}$$

Тоді абсолютний економічний ефект від можливого впровадження нашої розробки за три роки складе:

$E_{абс} = 1176 – 387 = 789$ тисяч грн.

Внутрішня дохідність $E_в$ вкладених інвестицій становитиме:

$$E_b = \sqrt[4]{1+\frac{789}{387}} - 1 = \sqrt[4]{1+2,0386} - 1 = \sqrt[4]{3,0386} - 1 = 1,32 - 1 = 0,32 \approx 32,0\%.$$

Оскільки величина $E_в = 32,0\% < \tau_{мін} = 43\%$, то потенційний інвестор у принципі може бути не зацікавлений у комерціалізації розробленої нами автоматизованої системи управління транзакціями та контролю фінансів (яка призначена для обраної нами цільової аудиторії користувачів). Але для прийняття остаточного рішення потрібно провести додаткові обґрунтування і розрахунки (наприклад, знизити рівень прийнятого ризику, підняти ціну реалізації нашої розробки, збільшити попит на нашу розробку тощо).

Зроблені розрахунки у вигляді діаграм наведено на рис. 5.1.

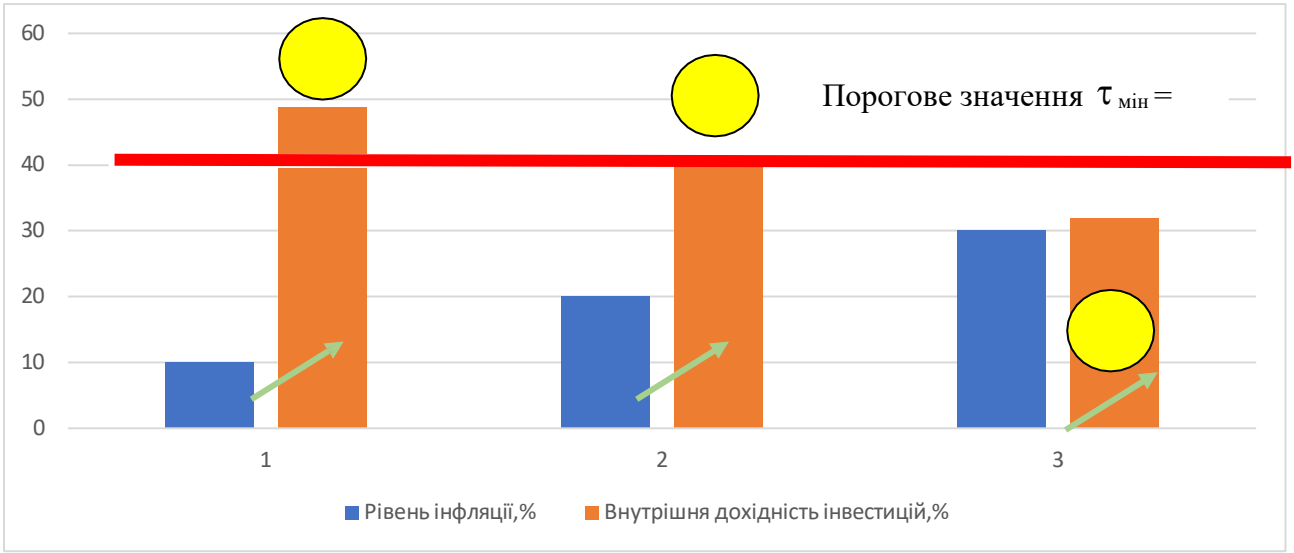

Рисунок 5.1 – Моделювання залежності величини внутрішньої дохідності

Потенційних інвестицій, які можуть бути вкладені у комерціалізацію нашої розробки, від рівня інфляції в країни

Результати виконаної економічної частини магістерської кваліфікаційної роботи зведено у таблицю:

Таблиця 5.6 економічні результати

Показники	Задані у ТЗ	Досягнуті у магістерській кваліфікаційній роботі	Висновок
1. Витрати на розробку	Не більше 130 тисяч грн	129 тисяч грн	Досягнуто
2. Абсолютний ефект від впровадження розробки, тисяч грн	Не менше 1500 тисяч грн (за три роки)	1504 тисячі грн (при 10% інфляції)	Виконано
3. Внутрішня дохідність інвестицій (коштів), %	не менше 43,0%	48,7%	Виконано
4. Термін окупності інвестицій (коштів), роки	до 3-ти років	2,05 років	Виконано

Таким чином, основні техніко-економічні показники розробленої нами автоматизованої системи управління транзакціями та контролю фінансів, визначені у технічному завданні, повністю виконані.

ВИСНОВКИ

У першому розділі магістерської кваліфікаційної роботи проведено аналіз предметної області систем автоматизації управління транзакціями та керування фінансами. Розглянуто призначення, основні цілі та завдання майбутньої розробки, а також визначено ключові функціональні можливості, які повинна забезпечувати система для задоволення потреб користувачів.

Проведений аналіз наявних літературних і онлайн-джерел дозволив ідентифікувати основні тенденції розвитку систем автоматизації фінансових процесів та виявити недоліки сучасних рішень, що обмежують їх ефективність. Зокрема, встановлено зростаючий попит на інтеграцію сучасних технологій для оптимізації фінансового планування, підвищення прозорості операцій і зменшення ризиків.

Цей аналіз закладає основу для вибору ефективної архітектури системи та програмних інструментів, які відповідатимуть потребам користувачів і забезпечать високий рівень функціональності, надійності та безпеки.

У другому розділі ми розглянули проектування клієнтської частини для системи обробки та керування фінансами, а також обґрунтували вибір технологічного стеку. Для розробки клієнтської частини ми вибрали Angular та TypeScript як основні технології. Angular надає масштабованість, стабільність та багатофункціональність, що особливо важливо для веб-застосунків зі складним функціоналом, таким як управління фінансами. TypeScript дозволяє підвищити безпеку та продуктивність розробки завдяки сильній типізації та підтримці сучасних стандартів JavaScript.

На сервері ми вибрали Node.js як платформу для серверної частини. Node.js забезпечує швидкість виконання, високу масштабованість і гнучкість у виборі інструментів для розробки. Його асинхронний характер і можливість використання JavaScript для розробки є значущими перевагами у контексті сучасних веб-застосунків.

Вибір бази даних впливає на продуктивність, масштабованість та безпеку системи. Для забезпечення ефективного управління фінансами ми обрали PostgreSQL як реляційну базу даних. PostgreSQL відома своєю надійністю, підтримкою транзакцій і ACID властивостей, що є критичними для систем з фінансовими операціями.

Для зберігання та управління тимчасовою інформацією ми використовуємо Redis. Він забезпечує швидкий доступ до даних завдяки своєму сховищу та підтримці різних структур даних.

Безпека є ключовим аспектом будь-якої фінансової системи. Ми забезпечуємо безпеку за допомогою різних заходів, включаючи аутентифікацію за стандартом OAuth2, використання токенів JWT для авторизації, шифрування даних та використання захищених протоколів передачі даних (наприклад, HTTPS). Це забезпечує конфіденційність, цілісність та доступність даних для користувачів і системних адміністраторів.

Обрані технології та підходи дозволяють створити сучасну та безпечну систему для управління фінансами, що відповідає найвищим стандартам розробки програмного забезпечення і відповідає потребам сучасних користувачів.

У третьому розділі було детально розглянуто процес реалізації функціональних компонентів додатку, включаючи клієнтську та серверну частини, а також інтеграцію зі сторонніми сервісами.

На клієнтській стороні були створені зручні інтерфейси для управління транзакціями, перегляду статистики, налаштування профілю та планування фінансових цілей. Реалізація забезпечує інтуїтивність взаємодії з користувачем і швидкий доступ до ключових функцій системи.

На серверній стороні були побудовані компоненти для обробки даних, взаємодії з базою даних та API. Архітектура серверної частини забезпечує масштабованість і продуктивність системи, а також можливість інтеграції з

банківськими системами, AI-сервісами для аналізу транзакцій та іншими сторонніми API.

Особливу увагу приділено безпеці додатку: було впроваджено захист даних користувачів, безпечну авторизацію та шифрування під час обміну даними.

Реалізація описаних функцій демонструє комплексний підхід до побудови системи управління фінансами, яка забезпечує зручність, швидкодію, надійність та безпеку для користувачів. Завершення цього етапу розробки дозволяє перейти до етапу тестування, що забезпечить якість і стабільність системи перед її впровадженням.

У четвертому розділі було докладно розглянуто процеси тестування та проведено тестування додатку, який охоплює всі основні аспекти, включаючи тестування клієнтської та серверної частин, інтеграцій, а також безпеки системи. Тестування серверної частини та інтеграцій з зовнішніми сервісами, зокрема ПриватБанком і модулями штучного інтелекту, продемонструвало високу продуктивність та стабільність системи. Сервер здатний ефективно обробляти різні типи HTTP-запитів, забезпечуючи швидкий час відповіді навіть при високих навантаженнях, що підтверджується середнім часом обробки GET-запитів на рівні 80 мс, а POST та PUT запитів на рівні 120 мс і 140 мс відповідно. Інтеграція з ПриватБанком працює без значних затримок, де час обробки запиту на баланс становить 150 мс, а для історії транзакцій — до 1,5 секунд для великих обсягів даних. Тестування взаємодії з ШІ показало ефективність у генерації рекомендацій та виявленні аномалій, з точністю 96% у виявленні підозрілих операцій. Проблеми, що виникали, були вирішені шляхом реалізації механізмів автоматичного оновлення токенів та адаптивного парсингу даних, а також поліпшення алгоритмів аналізу транзакцій для зменшення кількості помилкових спрацьовувань. Загалом, система продемонструвала високу надійність і швидкодію, відповідаючи вимогам щодо інтеграцій та обробки великих обсягів даних.

Результати тестування дозволяють з упевненістю стверджувати, що система працює стабільно, відповідає вимогам та готова до подальшого впровадження. Тестування продемонструвало високу якість коду та коректну роботу усіх основних функцій додатку.

Отримані економічні результати підтверджують високу ефективність розробленої автоматизованої системи управління транзакціями та контролю фінансів. Витрати на розробку склали 129 тисяч грн, що відповідає ліміту, визначеному в технічному завданні, а абсолютний ефект від впровадження перевищив прогнозовані 1500 тисяч грн, досягнувши 1504 тисячі грн, навіть з урахуванням інфляції. Внутрішня дохідність інвестицій склала 48,7%, що значно перевищує мінімально необхідний рівень у 43%. Крім того, термін окупності інвестицій становить 2,05 роки, що є набагато коротшим за заплановані 3 роки. Таким чином, всі техніко-економічні показники були повністю виконані, що свідчить про високу ефективність проекту.

В результаті проведених робіт, система демонструє високу ефективність, стабільність і безпеку, відповідаючи вимогам користувачів та готова до подальшого впровадження та експлуатації.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Іванишин М.Р., Іванишин (vntu.edu.ua) Матеріали доповідей LIII Всеукраїнська науково-технічна конференція факультету інтелектуальних інформаційних технологій та автоматизації (2024). URL: Іванишин (vntu.edu.ua)
2. Валерій Опарін, Віктор Федосов, Петро Юхименко, Андрій Крисоватий. Modern Ukrainian Financial Science. Theoretical paradigm & practical concept of public finance. Ebook 2019. 447ст.
3. Боб Тапскотт. TRIVERGENCE: Accelerating Innovation with AI, Blockchain, and the Internet of Things. Ebook 2023. 256ст.
4. Перспективи розвитку фінансової системи України URL: http://journals.khnu.km.ua/vestnik/wp-content/uploads/2021/01/16-14.pdf (дата звернення: 2.10.2024).
5. Перспективи розвитку блокчейн технологій у сфері глобального фінансовго ринку URL: https:/economyandsociety.in.ua/index.php/journal/article/view/3609/3540 (дата звернення: 13.10.2024).
6. Розвиток «вбудованих банківських фінансових технологій» (DIGITAL FinTech) URL: https://un-sci.com/ru/2024/01/12/rozvitok-vbudovanih-bankivskih-finansovih-tehnologij-digital-fintech/ (дата звернення: 2.11.2024).
7. І Штучний інтелект у фінансах URL: https://www.facerua.com/iak-shtuchnii-intieliekt-zminiuie-finansi-top-5-pieriedovikh-proghram-ta-yikh-pierievaghi-ta-niedoliki/ (дата звернення: 12.09.2024).
8. Які бувають системи для фінансових операцій та обліку URL: https://buhgalter911.com/uk/news/news-1076422.html (дата звернення: 12.11.2024).

9. Електронні платіжні системи: захист вашої інформації URL: https://crosspay.net/elektronni-platizhni-sistemi-zahist-vashoi-informacii/ (дата звернення: 21.10.2024).
10. Dina Darwish, Sanjeev Kumar. Utilizing AI and Machine Learning in Financial Analysis. Cambridge University Press 2021. 224ст.
11. Андерс Ханссон, Мартін Андерсен Нексе. Optimization for Learning and Control. Willey 2022. 354ст.
12. Ynab URL: https://www.ynab.com (дата звернення: 10.10.2024).
13. Mint. URL: https://mint.intuit.com/ (дата звернення: 14.10.2024).
14. Топ додатків для управління транзакціями. URL: https://ua.lutums.net/articles/budgeting/12-best-mint-com-alternatives-to-manage-your-finances.html (дата звернення: 13.12.2024).
15. Інформаційна архітектура: Основний посібник URL: https://dizz.in.ua/uk/informaczijna-arhitektura-osnovnij-posibnik/ (дата звернення: 13.11.2024).
16. Елізабет Робсон, Ерік Фрімен. Електронна книга Head First. Програмування на JavaScript. Фабула 2022
17. TypeScript: VANDERKAM, Dan. Effective TypeScript. " O'Reilly Media, Inc.", O'Reilly Media, Incorporated 2024. 224ст.
18. HTML Підручник URL: https://w3schoolsua.github.io/html/index.html#gsc.tab=0 (дата звернення: 9.10.2024).
19. Tailwind CSS URL: https://tailwindcss.com/docs/installation (дата звернення: 8.10.2024).
20. Крістіан Хілл. Learning Scientific Programming with Python. 2020.
21. Майкл Хартл. Ruby on Rails Tutorial: Learn Web Development with Rails. Addison-Wesley 2023. 317ст.
22. Елізабет Робсон, Ерік Фріман. Head First JavaScript Programming: A Learner's Guide to Modern JavaScript. Britain: Logman 2024. 322ст.

23. Марк Вандшнайдер. Learning Node.js: A Hands-On Guide to Building Web Applications in JavaScript. Britain: Logman 2017. 412ст.
24. Фреймворки у веб-розробці URL: https://highload.today/uk/frejmvorki-u-veb-rozrobtsi-shho-tse-yaki-isnuyut-i-dlya-chogo-potribni/ (дата звернення: 3.11.2024).
25. Angular URL: https://angular.io/ (дата звернення: 7.11.2024).
26. Безпечні електронні транзакції URL: https://www.vpnunlimited.com/ua/help/cybersecurity/secure-electronic-transactions?srsltid=AfmBOooZXSvmzo58xTuUde53-lCiPnBq8MNfHx64y086WHJDwkev0zbl (дата звернення: 18.11.2024).
27. Oauth-2.0 та JWT: SAKIMURA, N.; BRADLEY, J.; JONES, M. RFC 9101: The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR). USA: Michigan University 2021. 117ст.
28. Інформаційні системи і технології в економіці URL: https://studfile.net/preview/5118185/ (дата звернення: 16.11.2024).
29. Бази даних: SEGHIER, Nadia Ben; KAZAR, Okba. Performance benchmarking and comparison of NoSQL databases: Redis vs mongodb vs Cassandra using YCSB tool. In: 2021 International Conference on Recent Advances in Mathematics and Informatics (ICRAMI). IEEE, Algeria: Computer Science Department, University of Biskra 2021. 188ст.
30. PostgreSQL URL: https://www.postgresql.org/ (дата звернення: 15.11.2024).
31. Redis URL: https://habr.com/ru/companies/wunderfund/articles/685894/ (дата звернення: 10.10.2024).
32. German Creamer, Gary Kazantsev, Tomaso Aste. Machine Learning and AI in Finance German: Routledge 2023. 224ст.
33. БОСКІН, Костянтин Олександрович. Інформаційна система з обробки транзакцій розподіленого реєстру. Київ 2020. 193ст.
34. Захист додатку URL: https://pnn.com.ua/ua/blog/detail/mobile-security-in-2021-best-practices-for-ios-android-ordinary-users-and-mobile-developers (дата звернення: 24.11.2024).

35. Аспекти реалізації інформаційних потреб користувачів фінансової інформації URL: http://www.visnyk-econom.uzhnu.uz.ua/archive/8_2_2016ua/31.pdf (дата звернення: 22.10.2024).
36. WebStorm URL: https://www.jetbrains.com/webstorm/ (дата звернення: 13.12.2024).
37. Docker (software) URL: https://en.wikipedia.org/wiki/Docker_(software) (дата звернення: 1.12.2024).
38. Неперервна інтеграція CI/CD: бізнес-переваги URL: https://soft-den.com/continuous-integration-and-deployment (дата звернення: 12.10.2024).
39. Архітектура системи URL: https://elearning.sumdu.edu.ua/free_content/lectured:de1c9452f2a161439391120eef364dd8ce4d8e5e/20160217112601/170352/index.html (дата звернення: 13.10.2024).
40. What is UX / UI Design? URL: https://flatironschool.com/blog/what-is-ux-ui-design/ (дата звернення: 15.11.2024).
41. Сторінки помилок HTTP URL: https://freehost.com.ua/ukr/faq/wiki/chto-takoe-stranitsi-oshibok-http/ (дата звернення: 13.12.2024).
42. API-шлюз URL: https://integralsolutions.pl/uk/zarzadzanie-api/ (дата звернення: 11.11.2024).
43. Методичні вказівки до виконання економічної частини магістерських кваліфікаційних робіт. / Укладачі В.О. Козловський, О.Й. Лесько, В.В.Кавецький. Вінниця : ВНТУ, 2021. 42 с.

ДОДАТКИ

Додаток А (обов'язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

ЗАТВЕРДЖУЮ
Завідувач кафедри АІІТ
д.т.н., проф. Олег БІСІКАЛО
_____«___» ___2024 р.

ТЕХНІЧНЕ ЗАВДАННЯ
на магістерську кваліфікаційну роботу
«Автоматизація процесу аутентифікації та управління доступом у банківських системах»
08-31.МКР.01.02.000 ТЗ

Керівник роботи
к. т. н., доцент, професор кафедри АІІТ
Євген ПАЛАМАРЧУК
«_»_____2024 р.

Виконавець:
ст. гр. 1АКІТР-23М
Максим ІВАНИШИН
«__»_____2024 р.

Вінниця ВНТУ 2024

1. Назва та галузь застосування

1.1. Назва – Розробка автоматизованої системи управління транзакціями та контролю фінансів.

1.2. Галузь застосування – фінанси.

2. Підстава для проведення розробки.

Тема магістерської кваліфікаційної роботи затверджена наказом по ВНТУ від Протокол №310 від «___» вересня 2024року.

3. Мета та призначення розробки.

Метою розробки є створення багатофункціональної системи управління фінансами, яка поєднує інтеграцію з банківськими сервісами та можливості штучного інтелекту. Система забезпечує користувачам безпечний доступ до фінансової аналітики, індивідуальних рекомендацій і механізмів виявлення шахрайства.

4. Джерела розробки.

Магістерська кваліфікаційна робота виконується вперше. В ході проведення розробки повинні використовуватись такі документи:

1. Методичні вказівки до виконання магістерських кваліфікаційних робіт для студентів спеціальностей 126 – «Інформаційні системи та технології», 151 – «Автоматизація та комп'ютерно-інтегровані технології», 174 – «Автоматизація, комп'ютерно-інтегровані технології та роботехніка» / Уклад. О. В. Бісікало, Ю.Ю. Іванов, Р.В. Маслій. – Вінниця : ВНТУ, 2019. – 26 с.
2. German Creamer, Gary Kazantsev, Tomaso Aste. Machine Learning and AI in Finance
3. Oauth-2.0 та JWT: SAKIMURA, N.; BRADLEY, J.; JONES, M. RFC 9101: The OAuth 2.0 Authorization Framework: JWT-Secured Authorization Request (JAR). 2021.

5. Вимоги до розробки.

5.1. Перелік головних функцій:

- автоматизація та оптимізація процесу навчання;
- керування базою даних;
- модуль підтримки дистанційного навчання;

5.2. Основні технічні вимоги до розробки.

5.2.1. Вимоги до програмної платформи:

- PostgreSQL,
- Redis,
- Node.js
- Angular,
- AWS,
- Windows 11/10,
- Android 11+/IOS

5.2.2. Умови експлуатації системи:

- безперебійне цілодобове функціонування системи;
- регулярне оновлення та підтримка актуальності даних
- надійний захист даних

6. Економічні показники

- витрати на розробку – не більше 70 тис. грн;
- абсолютний ефект від впровадження розробки – не менше 900 тис. грн;
- внутрішня дохідність інвестицій – не менше 48%;
- термін окупності – не більше 3 років.

7. Стадії розробки

1. Розділ 1 «Аналіз предметної області та існуючих систем управління та керування транзакціями» має бути виконаний до 05.10.2024.

2. Розділ 2 «Архітектура проєкту та вибір технологічного стеку» має бути виконаний до 20.10.2024.

3. Розділ 3 «Реалізація клієнтської частини та серверної логіки системи управління фінансами» має бути виконаний до 10.11.2024.

4. Розділ 4 «Тестування додатку» має бути виконаний до 24.11.2024.

5. Економічний розділ має бути виконаний до 27.11.2024.

8. Порядок контролю та приймання

1. Рубіжний контроль провести до 15.11.2024.

2. Попередній захист магістерської кваліфікаційної роботи провести до 05.12.2024.

3. Захист магістерської кваліфікаційної роботи
провести в період з 17.12.2024 до 20.12.2024.

Додаток Б (обов’язковий)

ГРАФІЧНА ЧАСТИНА

Розробка автоматизованої системи управління транзакціями та керування фінансами

Продовження додатку Б

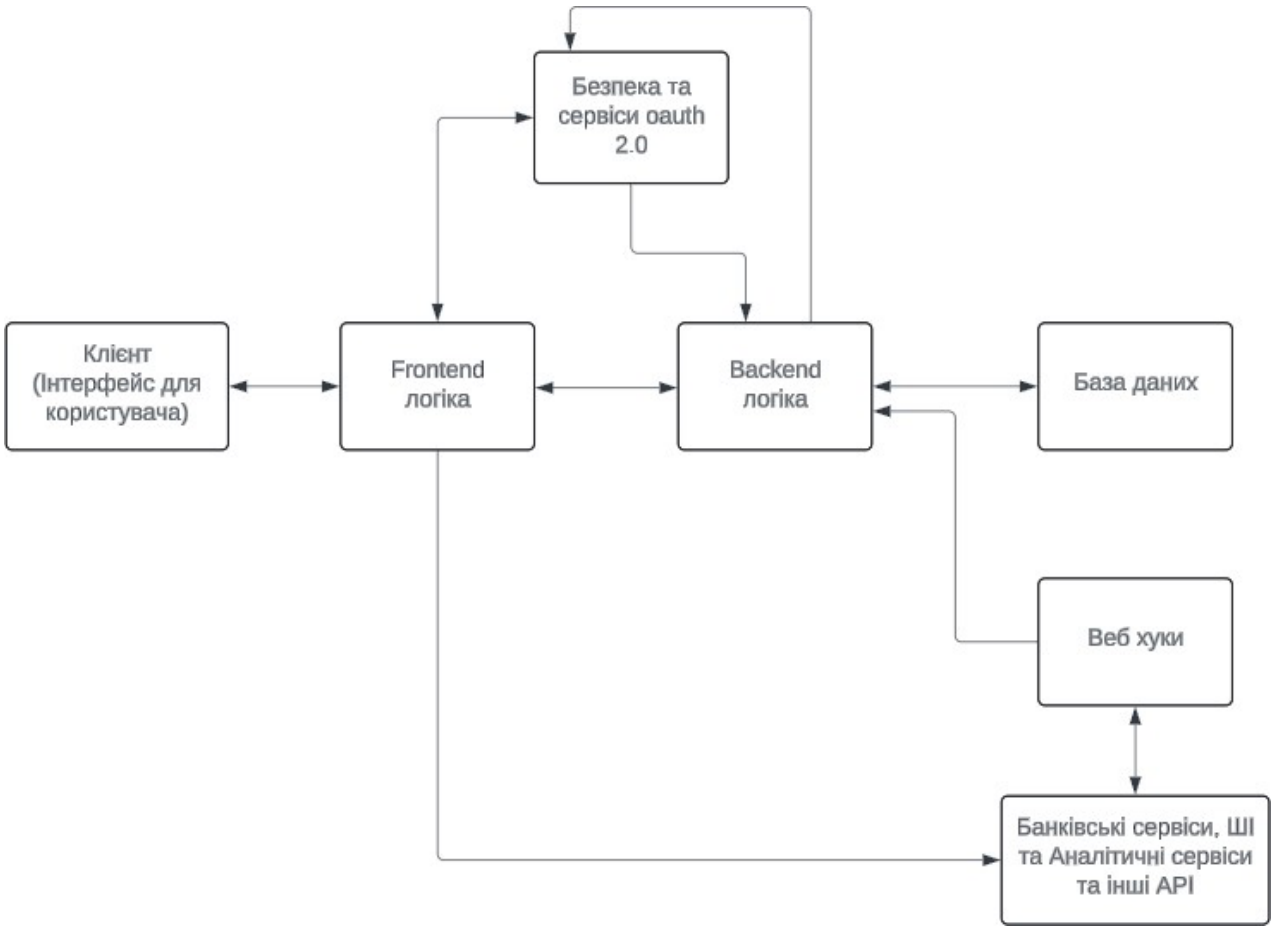

Рисунок Б.1 – Архітектурна схема додатку

Продовження додатку Б

Рисунок Б.2 – загальна Flow користувача для доступу до функціоналу застосунку

Продовження додатку Б

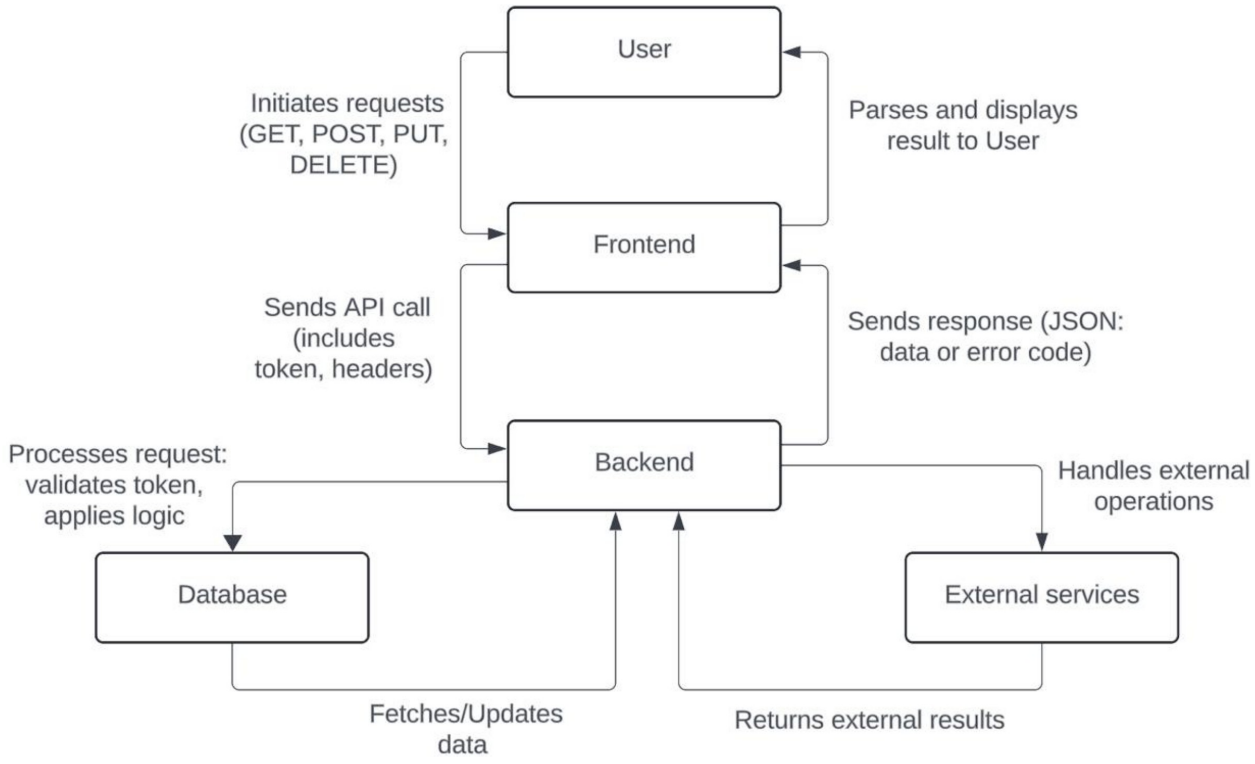

Рисунок Б.3 – графічна діаграма інтеграції з сервером

Продовження додатку Б

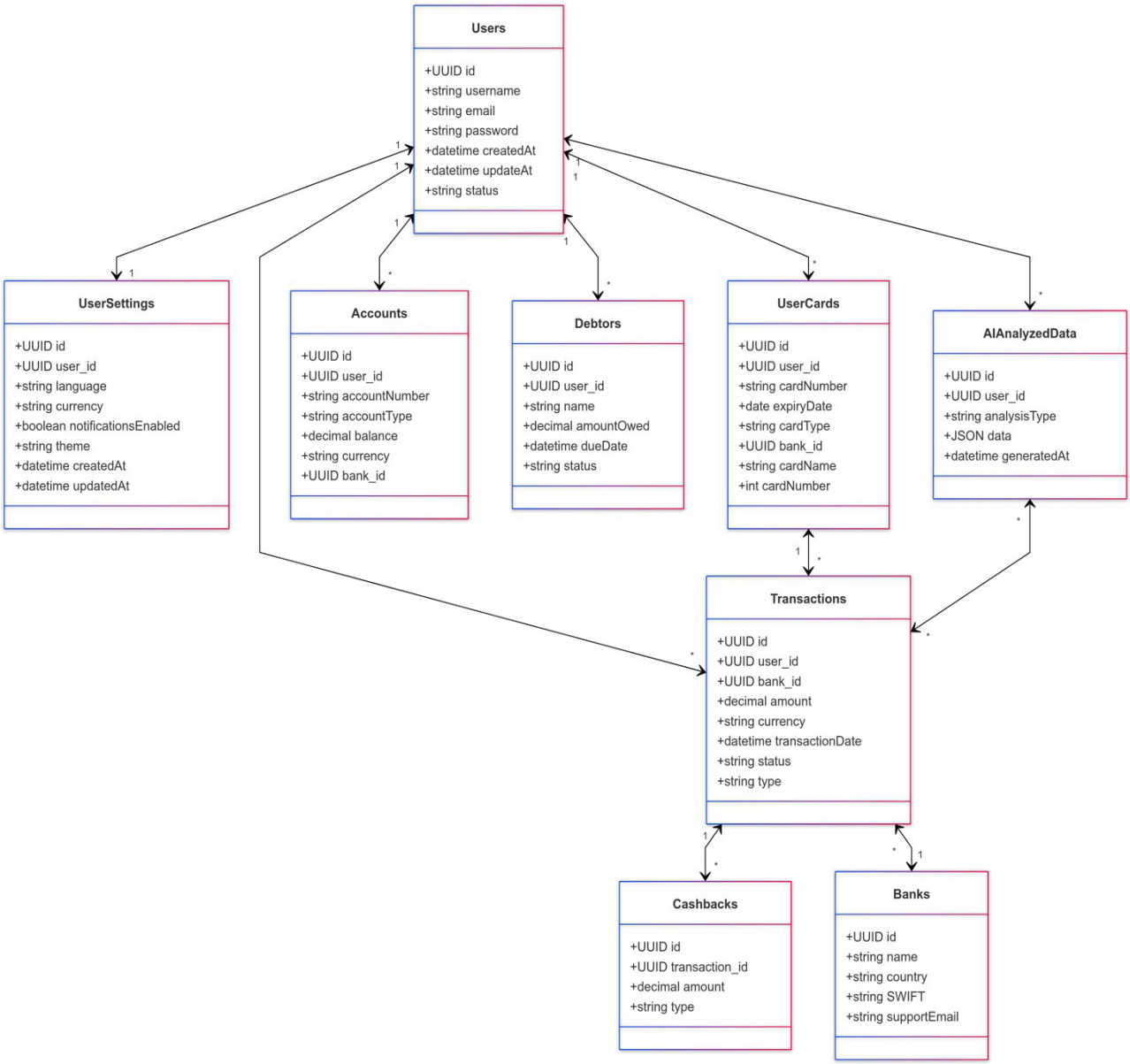

Рисунок Б.4 – загальна UML діаграма класів

Продовження додатку Б

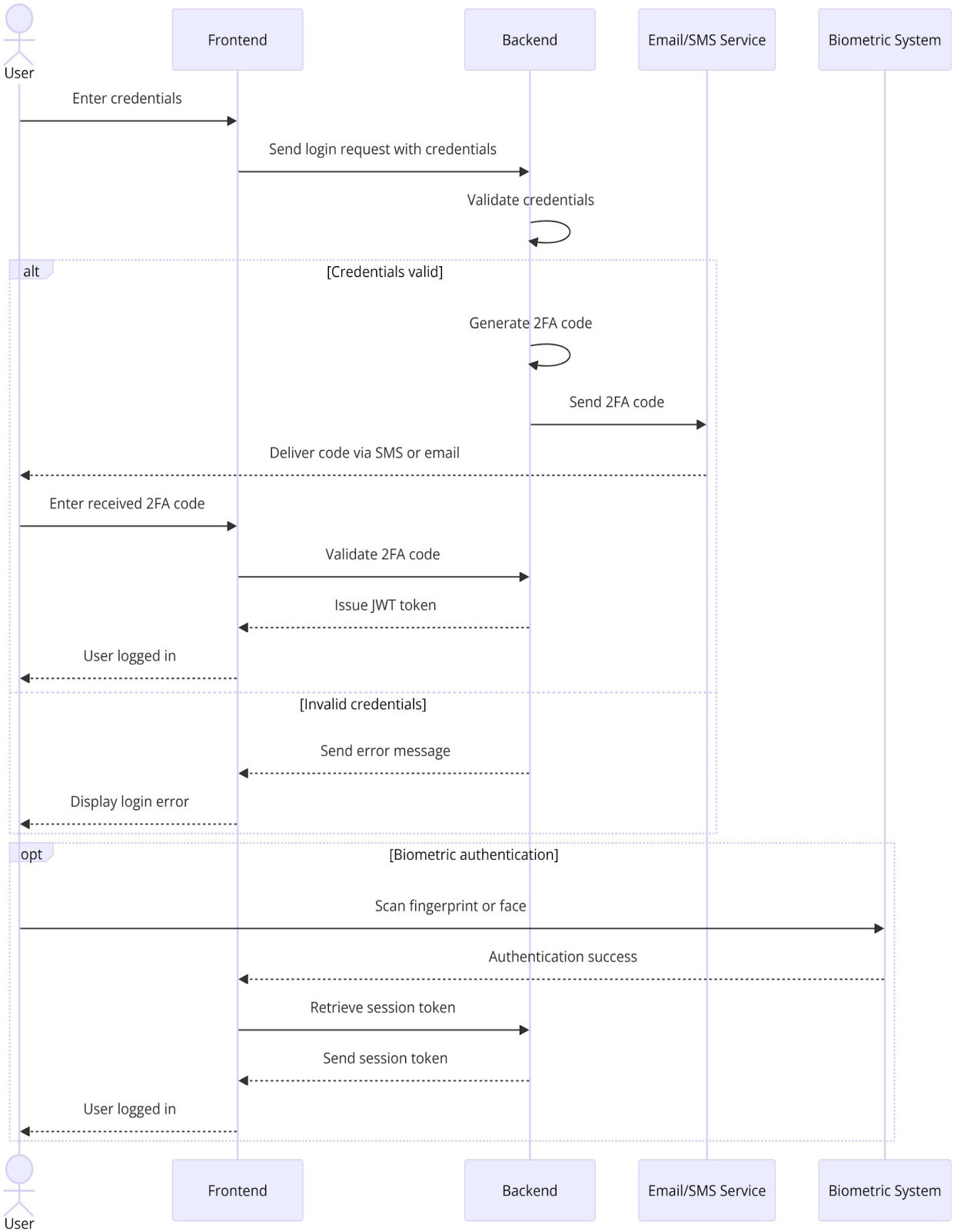

Рисунок Б.5 – загальна схема авторизації користувача

Продовження додатку Б

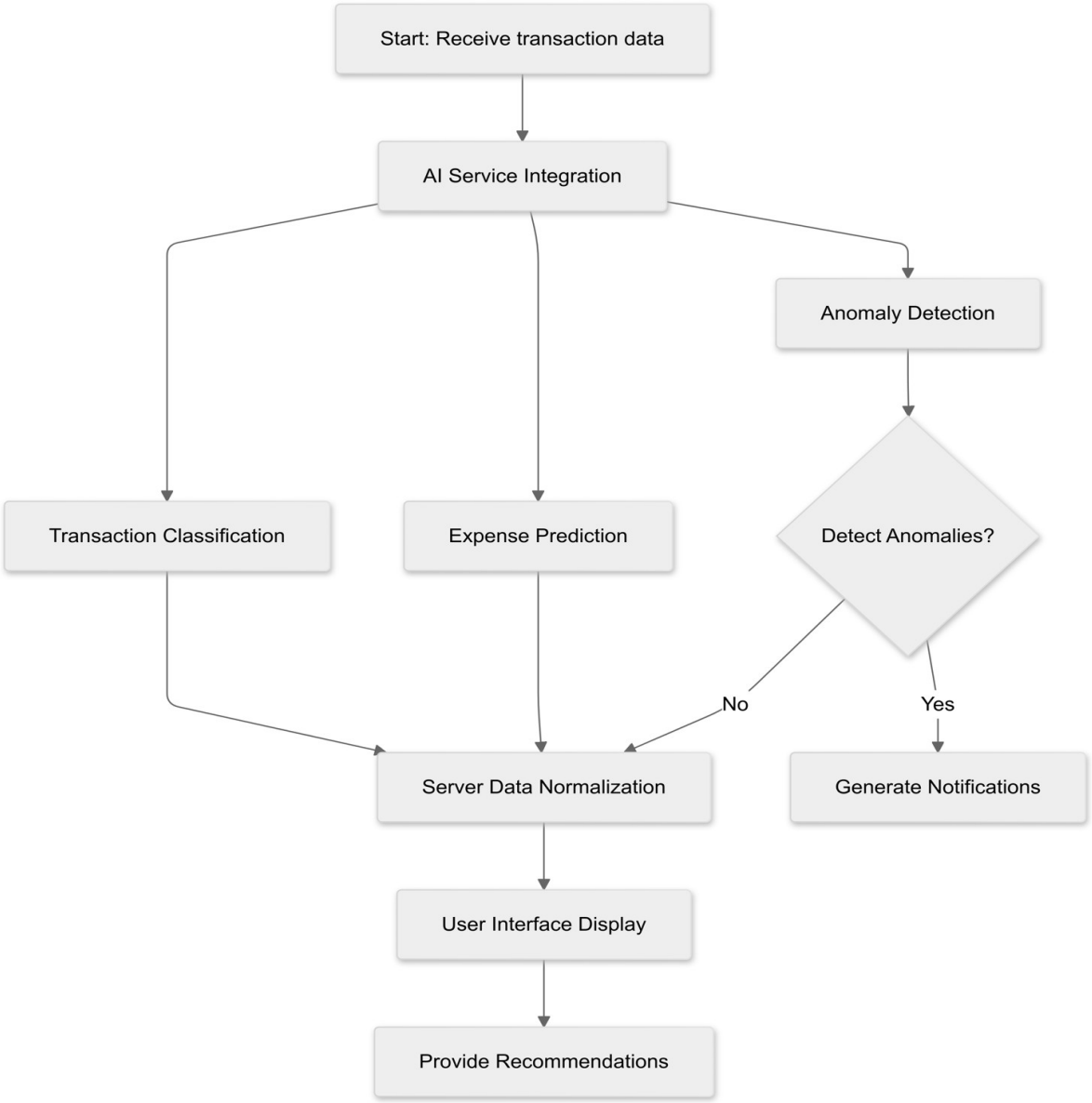

Рисунок Б.6 – загальна схема генерації рекомендацій для користувача

Додаток В (обов'язковий)

ЛІСТИНГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

```
import { Injectable } from '@angular/core';
import { BiometricAuth } from '@aparajita/capacitor-biometric-auth';
import { BiometricAuth as AuthBiometric } from '@store/auth/auth.action';
import { finalize } from 'rxjs';
import { routes } from '@core/constants/routes.const';
import { UntilDestroy, untilDestroyed } from '@ngneat/until-destroy';
import { Router } from '@angular/router';
import { Store } from '@ngxs/store';
import { BiometricAuthOptions } from '@core/constants/biometricAuthOptions';

@UntilDestroy()
@Injectable({
  providedIn: 'root',
})
export class BiometricService {
  constructor(private router: Router, private store: Store) {}

  public async checkCredentialByNativeBiometric() {
    const { isAvailable } = await BiometricAuth.checkBiometry();
    if (isAvailable) {
      try {
        await BiometricAuth.authenticate(BiometricAuthOptions);
        this.store
          .dispatch(new AuthBiometric())
          .pipe(
            finalize(async () => {
              await this.router.navigate([routes.home]);
            }),
            untilDestroyed(this)
          )
          .subscribe();
      } catch (error) {
        console.warn('[NativeBiometric issue]: ', error);
      }
    }
    return;
  }
}

import { AuthGuard } from '@nestjs/passport';
import { AuthStrategyEnum } from '../enums';
import { ExecutionContext, Injectable, UnauthorizedException } from '@nestjs/common';
import { getGoogleAudience } from 'src/config';
import { OAuth2Client } from 'google-auth-library';

@Injectable()
export class GoogleGuard extends AuthGuard(AuthStrategyEnum.GOOGLE) {
        private googleAudience: string;
```

```
        private client: OAuth2Client;

        constructor() {
                super();
                this.googleAudience = getGoogleAudience();
                this.client = new OAuth2Client();
        }

        async canActivate(context: ExecutionContext): Promise<boolean> {
                const request = context.switchToHttp().getRequest();
                const accessToken = request.headers.authorization?.split(' ')[1];

                if (!accessToken) {
                        throw new UnauthorizedException('Invalid auth headers');
                }

                try {
                        const ticket = await this.client.verifyIdToken({
                                idToken: accessToken,
                                audience: this.googleAudience,
                        });
                        const payload = ticket.getPayload();

                        const audCorrespondsGoogleAudience = payload.aud ===
this.googleAudience;

                        if (!audCorrespondsGoogleAudience) {
                                throw new Error('Invalid token');
                        }

                        request.user = payload;
                        return true;
                } catch (err) {
                        console.error(err);
                        return false;
                }
        }
}

import { InjectModel } from '@nestjs/sequelize';
import { Account, Transaction, User, Category, Subcategory } from 'src/models';
import { Injectable } from '@nestjs/common';
import {
        FindAllTransactionRequest,
        rangeStartAndEndDateUtils,
        TransactionRequest,
} from 'src/common';
import { Includeable } from 'sequelize/types/model';
import { FindOptions, Op, WhereOptions } from 'sequelize';

@Injectable()
export class TransactionRepository {
```

```
constructor(
	@InjectModel(Transaction)
	private readonly transactionModel: typeof Transaction
) {}

async create(dto: TransactionRequest): Promise<Transaction> {
	return this.transactionModel.create({ ...dto });
}

async getById(id: string, userId: string): Promise<Transaction> {
	return this.transactionModel.findByPk(id, {
		include: [
			{
				model: Account,
				required: true,
				include: [
					{
						model: User,
						where: { id: userId },
						required: true,
					},
				],
			},
		],
	});
}

async getAll(id: string, dto: FindAllTransactionRequest): Promise<Transaction[]> {
	const { categoryIds, offset, limit, search, startDate, endDate } = dto;
	const { field, order } = dto.sorting;
	const where: WhereOptions<Transaction> = {};

	if (startDate || endDate) {
		where.createdAt = await rangeStartAndEndDateUtils(startDate, endDate);
	}
	if (search) {
		where.amount = { [Op.eq]: search };
	}
	if (categoryIds) {
		where.categoryId = { [Op.or]: [categoryIds] };
	}

	const options: FindOptions = {
		where,
		include: [
			{
				model: Account,
				required: true,
				include: [
					{
						model: User,
						where: { id: id },
```

```
                                        required: true,
                                    },
                                ],
                            },
                            {
                                model: Category,
                                required: true,
                                include: [
                                    {
                                        model: Subcategory,
                                    },
                                ],
                            },
                        ] as Includeable[],
                        offset,
                        limit,
                        order: [[field, order]],
                    };

                    return await this.transactionModel.findAll(options);
                }

                async getAllByDate(dateFrom: Date, dateTo: Date): Promise<Transaction[]> {
                    const query = {
                        where: {
                            createdAt: {
                                [Op.gte]: dateFrom,
                                [Op.lt]: dateTo,
                            },
                        },
                    };
                    return await this.transactionModel.findAll(query);
                }

                async delete(id: string): Promise<void> {
                    await this.transactionModel.destroy({ where: { id } });
                }

                async deleteAllAccountTransactions(id: string): Promise<void> {
                    await this.transactionModel.destroy({ where: { accountId: id } });
                }
            }
```

Додаток Г (обов'язковий)

ПРОТОКОЛ ПЕРЕВІРКИ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: «Розробка автоматизованої системи управління транзакціями та контролю фінансів»
Тип роботи: магістерська кваліфікаційна робота
Підрозділ: кафедра АІІТ

Показники звіту подібності Turnitin

Оригінальність 99,0 % Схожість 1,0 %

Аналіз звіту подібності (відмітити потрібне)

○ Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
○ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і самостійності її автора. Роботу направити на розгляд експертної комісії кафедри.
○ Виявлені у роботі запозичення є недобросовісними і мають ознаки плагіату та/або в ній містяться навмисні спотворення тексту, що вказують на спроби приховування недобросовісних запозичень.

Особа, відповідальна за перевірку _____ Роман МАСЛІЙ
(підпис)

Ознайомлені з повним звітом подібності, який був згенерований системою Unicheck щодо роботи.

Автор роботи _____ Максим ІВАНИШИН
(підпис)

Керівник роботи _____ Євген ПАЛАМАРЧУК