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

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему:

«Автоматизація процесу аутентифікації та управління доступом у банківських системах»

(тема роботи)

Виконав: студент 2 курсу, групи АКІТР-23м
спеціальності 174 – Автоматизація, комп'ютерно-інтегровані технології та робототехніка
Ткачук О. О.
(прізвище та ініціали)

Керівник: к.т.н., доцент кафедри АІІТ
Богач І.В.

«12» грудня 2024 р.

Опонент: к.т.н., доцент кафедри САІТ
Горячев Г.В.

«12» грудня 2024 р.

Допущено до захисту
Завідувач кафедри АІІТ
д.т.н., проф. Бісікало О. В
«____» _____ 2024 р.

Вінниця ВНТУ – 2024 рік

Вінницький національний технічний університет

Факультет інтелектуальних інформаційних технологій та автоматизації

Кафедра автоматизації та інтелектуальних інформаційних технологій

Рівень вищої освіти ІІ-ий (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

Завідувач кафедри АІІТ

д.т.н., проф. Олег БІСІКАЛО

"19" 09 2024 року

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Ткачуку Олексію Олександровичу

1. Тема роботи: «Автоматизація процесу аутентифікації та управління доступом у банківських системах»
керівник роботи: к.т.н., доцент каф. Ілона БОГАЧ
затверджено наказом ВНТУ від "17" 09. 2024 року №310
2. Строк подання студентом роботи до 16
3. Вихідні дані до роботи:
Операційна система – macOS 12.1 Monterey або вище; Процесор – Intel Core i5 з тактовою частотою 2.4 ГГц або більше, або Apple M1; Оперативна пам'ять – 8 Гбайт або більше.
4. Зміст текстової частини: Вступ, Аналіз предметної області, Технології та інструменти розробки, Проєктування та реалізація системи, Тестування додатку, Економічний розділ, Висновки, Список використаних джерел.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових закреслень): Діаграма взаємодії контролерів та сервісів, Діаграма роботи Spring Security, Діаграма бази даних (UML), Діаграма верифікації запитів та відповідей, Схема взаємодії клієнта з сервером, Діаграма обробки помилок на сервері.

6. Консультанти розділів магістерської кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв
1-4	Ілона БОГАЧ к.т.н., доцент каф. АІІТ	18.09.2024	18.09.2024
5	Володимир КОЗЛОВСЬКИЙ к.е.н., доцент каф. ЕПтаВМ	20.09.24	30.11.24

7. Дата видачі завдання 18.09.2024

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Дослідження існуючих альтернативних банківських додатків	до 28.09.24	викон.
2	Опис методів авторизації та аутентифікації у додатка	до 15.10.24	викон.
3	Підбір технологій та проектування системи	до 10.11.24	викон.
4	Реалізація та тестування додатку	до 20.11.24	викон.
5	Економічний розділ	до 30.11.24	викон.
6	Оформлення матеріалів до захисту МКР	01.12.24 – 05.12.24	викон.
7	Попередній захист роботи	до 05.12.24 р.	вик.
8	Остаточний захист роботи	17.12.24 р.	викон.

Студент _____ Олексій ТКАЧУК
(підпис) (прізвище та ініціали)

Керівник роботи _____ Ілона БОГАЧ
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.457

Ткачук О. О. Автоматизація процесу аутентифікації та управління доступом у банківських системах. Магістерська кваліфікаційна робота зі спеціальності 174 - Автоматизація, комп'ютерно-інтегровані технології та робототехніка, освітньо-професійна програма – Інтелектуальні комп'ютерні системи. Вінниця: ВНТУ, 2024. 108 с.

На укр. мові. Бібліогр.: 35 назв; рис.: 26; табл.; 9.

У роботі проаналізовано основні підходи до автоматизації серверної частини банківського додатку з акцентом на використанні REST API та Spring Security. Було визначено оптимальні інструменти для створення захищеної системи та спроєктовано її архітектуру. Реалізація серверної частини включала впровадження механізмів авторизації та аутентифікації користувачів, а також розробку REST API для забезпечення взаємодії з клієнтською частиною додатку. Для перевірки коректності роботи системи проведено тестування за допомогою Postman, що підтвердило надійність функціоналу та правильність обміну даними між сервером і клієнтом.

Ключові слова: автоматизація, REST API, Spring Security, серверна частина, авторизація, аутентифікація.

ABSTRACT

Tkachuk O. O. Automation of Authentication and Access Management Processes in Banking Systems. Master's Qualification Thesis in the specialty 174 - Automation, Computer-Integrated Technologies, and Robotics, educational-professional program - Intelligent Computer Systems. Vinnytsia: VNTU, 2024. 108 pages.

In Ukrainian. Bibliography: 35 titles; figures: 26; tables: 9.

The research analysed key approaches to automating the server-side of a banking application, focusing on the use of REST API and Spring Security. Optimal tools were selected to create a secure system, and the application architecture was designed. The implementation of the server-side included the development of user authentication and authorization mechanisms, as well as the creation of a REST API to facilitate interaction with the client-side application. The system's functionality was tested using Postman, which confirmed the reliability of the features and the correctness of data exchange between the server and the client.

Keywords: automation, REST API, Spring Security, server-side, authorization, authentication.

ЗМІСТ

ВСТУП

Актуальність роботи. У сучасному світі інформаційні технології стали невід'ємною частиною багатьох сфер життя, значно впливаючи на їх функціонування. Особливо це стосується фінансової галузі, де розвиток цифрових технологій та електронних платежів привів до зростання популярності банківських додатків та віртуальних фінансових послуг. Створення безпечних, ефективних і надійних банківських додатків є важливою і актуальною частиною фінансового сектора, що забезпечує зручність та доступність фінансових послуг для користувачів.

З ростом популярності банківських програм виникають нові виклики. Захист персональних даних і фінансових операцій стає критичним завданням для розробників. Швидкий технологічний прогрес та поява нових загроз безпеці вимагають створення надійних механізмів захисту та контролю доступу до банківських додатків.

На даний час актуальним є розробка захищеного сервісу, що забезпечить авторизацію, аутентифікацію та безпеку даних користувачів, а також дозволить взаємодіяти з клієнтами через REST API з використанням сучасного програмного забезпечення.

Метою роботи є покращення продуктивності та безпеки банківських додатків для забезпечення безпечної та ефективної взаємодії з клієнтами, що відповідає сучасним вимогам фінансового сектора.

Мета роботи буде досягнута за рахунок застосування RESTful [1] інтерфейсу в поєднанні з механізмами аутентифікації та авторизації. В результаті роботи буде створено серверну частину банківської програми з використанням Spring Security [2] та REST API, що забезпечить безпечну та ефективну взаємодію з клієнтами відповідно до сучасних вимог фінансового сектора.

Для розв’язання поставленої мети потрібно виконати **наступні задачі**:

1. Провести аналіз предметної області, а саме аналіз банківських додатків і їх методи авторизації та аутентифікації, функціональні та технічні вимоги та визначити напрямок удосконалення систем.

2. Розробити архітектуру проєкту та обрати інструменти та програмні засоби для розробки.

3. Розробити серверну частину банківської програми.

4. Протестувати систему та порівняти з аналогами.

Об’єктом дослідження є процес аутентифікації та авторизації користувачів.

Предметом дослідження є алгоритми та методи розробки серверної частини банківських додатків.

Методи дослідження. В роботі застосовуються аналітичні методи, методи оптимізації розробки серверної частини програмного забезпечення, покращення процесу авторизації та аутентифікації, методи розробки банківських додатків.

Наукова новизна роботи полягає в тому, що запропонована архітектура серверної частини банківського додатку, яка за рахунок використання MongoDB для зберігання даних, Spring Security для реалізації захисту та JWT-токенів для авторизації, дозволяє, на відміну від аналогів, підвищити безпеку системи, її масштабованість і продуктивність. Розроблений підхід до реалізації бізнес-логіки через сервіси та інтерфейси сприяє модульності та спрощує інтеграцію з клієнтською частиною, що забезпечує адаптивність системи до змін у функціональності.

Практичне значення роботи полягає у створенні діючого прототипу серверної частини банківського додатку, який забезпечує ефективну та безпечну взаємодію з клієнтською частиною через REST API. Реалізоване програмне забезпечення може бути використане як основа для впровадження у реальних банківських системах, забезпечуючи функціонал аутентифікації та авторизації за допомогою JWT-токенів, управління обліковими записами,

обробки транзакцій і збереження даних у MongoDB. Результати роботи також можуть слугувати навчальною базою для вивчення методів побудови безпечних та масштабованих серверних додатків.

Апробація результатів роботи. Результати роботи було представлено на Всеукраїнській науково-технічній конференції факультету інтелектуальних інформаційних технологій та автоматизації(2024) [3].

1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

Банківські додатки є важливою складовою сучасного фінансового сектору, оскільки вони забезпечують зручну та швидку взаємодію між клієнтами та банками. За допомогою банківських додатків клієнти можуть здійснювати різні фінансові операції та отримувати доступ до важливої банківської інформації.

Основні функції банківських додатків включають можливість перегляду балансу та виписки з рахунків, здійснення переказів коштів, оплату рахунків та кредитів, управління картою, замороження чи скасування платежів та багато іншого. Ці функції надають клієнтам зручний спосіб контролювати та керувати своїми фінансовими операціями в режимі реального часу.

Переваги використання банківських додатків для клієнтів включають зручний доступ до банківських послуг в будь-який час та з будь-якого місця, забезпечення швидких та безпечних фінансових операцій, можливість моніторингу фінансових транзакцій та контролю розходів. Крім того, банківські додатки забезпечують зручність та ефективність управління фінансами шляхом використання інтуїтивно зрозумілого інтерфейсу та персоналізованих налаштувань.

Додатки також відіграють важливу роль для банків, оскільки вони дозволяють знизити навантаження на банківські відділення та покращити якість обслуговування клієнтів. Завдяки банківським додаткам банки можуть автоматизувати багато операцій, забезпечити безпеку та точність обробки фінансових транзакцій, а також надати персоналізовані послуги своїм клієнтам.

1.1 Загальні відомості про банківські додатки

Банківські додатки є важливою частиною сучасної фінансової індустрії. Вони надають клієнтам зручний та швидкий доступ до фінансових послуг, дозволяють здійснювати платежі, перекази коштів, керувати рахунками та отримувати фінансову інформацію через мобільні пристрої. Ці додатки є програмним забезпеченням, розробленим банками для забезпечення доступу клієнтів до їхніх послуг через мобільні пристрої, такі як смартфони чи планшети(рисунок 1.1). Крім того, банківські додатки забезпечують безпеку фінансових операцій, використовуючи різні методи аутентифікації та шифрування даних.

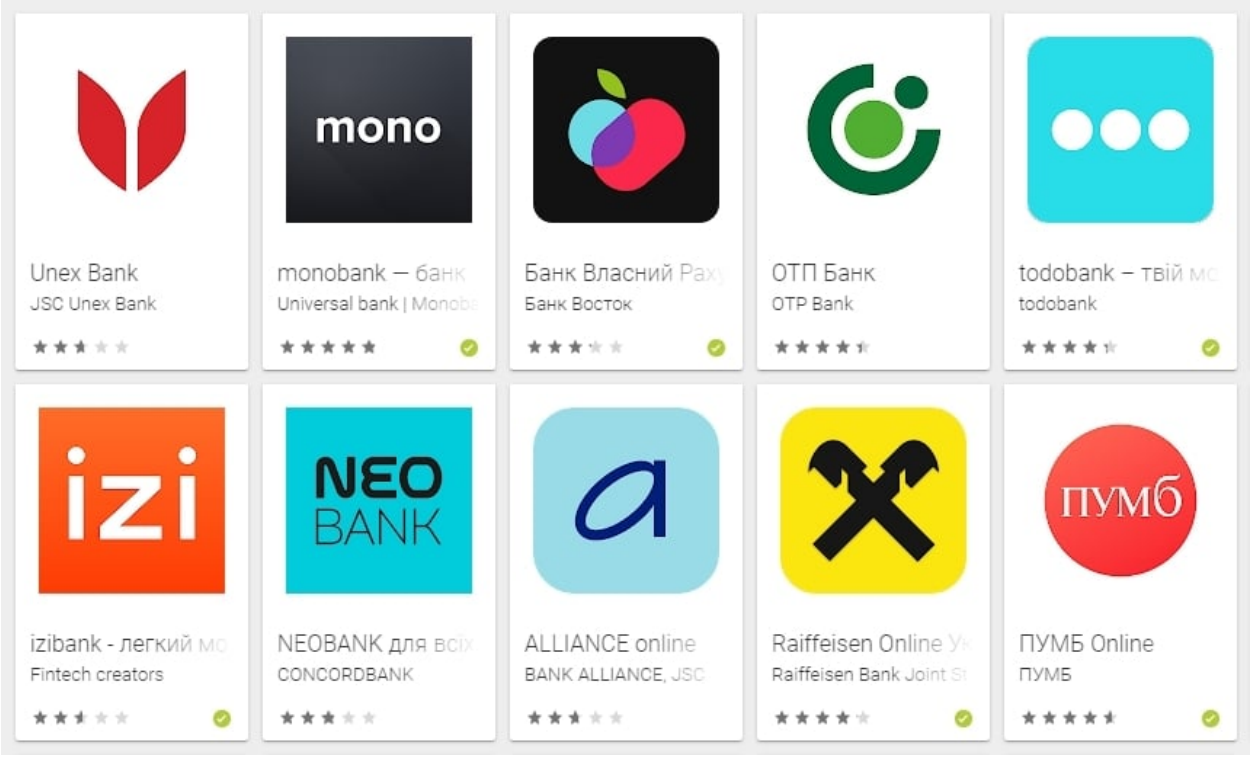

Рисунок 1.1 – Існуючі банківські додатки

Основні переваги банківських додатків включають:

- Зручність: Клієнти можуть здійснювати банківські операції в будь-який час та в будь-якому місці зі своїх мобільних пристроїв.

- Швидкість: Операції в банківських додатках виконуються набагато швидше порівняно з відвідуванням банківських відділень або використанням інших каналів обслуговування.
- Спрощення управління фінансами: Додатки надають користувачам зручні інструменти для контролю та управління своїми фінансами, такі як статистика витрат, планування бюджету тощо.
- Безпека: Банківські додатки застосовують різні заходи безпеки, включаючи аутентифікацію[4], шифрування даних та механізми виявлення шахрайства.

Дослідження існуючих банківських додатків дозволяє виявити тенденції та інновації у галузі, оцінити їх функціональність, дизайн та рівень захищеності. Це допомагає банкам та розробникам створювати більш ефективні та високоякісні банківські додатки, що задовольняють потреби клієнтів та забезпечують надійну роботу з фінансами.

1.2 Аналіз сучасних методів аутентифікації та авторизації в банківських додатках

В сучасному банківському секторі, де безпека та конфіденційність є пріоритетом, важливо оцінювати ефективність та надійність методів аутентифікації та авторизації, які використовуються в банківських додатках. Поглиблений аналіз різноманітних підходів вказує на те, що сучасні методи орієнтовані на забезпечення високого рівня захисту, зручності та відповідності стандартам кібербезпеки.

1.2.1 Дводіапазонна аутентифікація 2 FA

2FA є потужним інструментом для підвищення безпеки в банківських додатках. Ця технологія використовує два різних методи (фактори) для

перевірки ідентичності користувача перед наданням доступу до облікового запису. Основною метою 2FA є ускладнення доступу для несанкціонованих осіб навіть у випадку, коли вони володіють основним паролем або іншими формами ідентифікаційних даних[5-6].

1. Перший фактор (щось, що користувач знає): Це може бути основний пароль, PIN-код або відповідь на секретне питання. Користувач вводить цей перший фактор при спробі входу до свого облікового запису.

2. Другий фактор (щось, що користувач має): Це додатковий елемент, який підтверджує ідентичність користувача. Цей фактор може включати:

- Одноразовий код (OTP): Користувач отримує унікальний одноразовий код на свій мобільний телефон або електронну пошту і вводить його під час входу.
- Біометричні дані: Використання відбитку пальця, скану обличчя або інші біометричні дані для підтвердження ідентичності.
- Фізичний об'єкт: Наприклад, фізичний ключ або картка, яку користувач повинен мати при собі.

Після того, як користувач введе основний пароль (перший фактор), система запитає другий фактор для подальшого підтвердження ідентичності. Цей підхід ускладнює доступ до облікового запису для несанкціонованих осіб, оскільки вони мають отримати не лише основний пароль, а й доступ до другого фактора.

Наприклад, якщо користувач використовує OTP, система генерує унікальний одноразовий код і відправляє його на мобільний телефон або електронну пошту користувача. Користувач вводить цей код під час входу, і тільки після успішного підтвердження обох факторів він має доступ до свого облікового запису.

Таким чином, 2FA гарантує більш високий рівень безпеки для користувачів, оскільки вона потребує не тільки знання основного пароля, а й доступ до додаткового елемента, що важко підробити або зламати.

З цього можна виділити переваги та недоліки цієї технології.

Переваги 2FA:

1. Підвищена безпека: Важко проникнути до облікового запису, навіть якщо хтось зламає основний пароль.

2. Захист від фішингу: Оскільки тому, хто атакує, потрібно мати не лише пароль, а й другий фактор, 2FA допомагає захистити від фішингових атак.

3. Універсальність: Ця технологія широко використовується в багатьох сервісах і застосунках, що робить її зручною для користувачів.

Недоліки 2FA:

1. Складність використання: Для деяких користувачів може виникнути складність у використанні 2FA, особливо якщо вони не знайомі з додатковими методами аутентифікації.

2. Людський фактор: Якщо користувач забув або втратив доступ до другого фактора, можуть виникнути проблеми з доступом до облікового запису.

3. Часові затрати: Додаткові кроки для входу можуть займати більше часу, особливо якщо потрібно отримати одноразовий код або використати біометричні дані.

В цілому, технологія 2FA є важливим кроком у напрямку покращення безпеки у банківських додатках, забезпечуючи додатковий шар захисту для користувачів та допомагаючи уникнути несанкціонованого доступу до фінансової інформації.

1.2.2 Використання Біометричних Даних:

Біометричні дані використовуються для підвищення безпеки банківських додатків завдяки унікальності фізичних характеристик кожної людини. Біометрична аутентифікація включає в себе перевірку ідентичності користувача за допомогою таких характеристик, як відбитки пальців, обличчя,

райдужка ока, голос, або навіть поведінкові ознаки, як-от спосіб введення тексту[7-8].

- Відбитки пальців: Користувач прикладає палець до сканера, який порівнює збережений шаблон відбитка з тим, що зчитується в моменті аутентифікації.

- Розпізнавання обличчя: Камера зчитує обличчя користувача, аналізує різні точки та порівнює їх з раніше збереженим шаблоном.

- Розпізнавання райдужки ока: Камера високої роздільності сканує райдужку ока користувача, порівнюючи з збереженим шаблоном.

- Розпізнавання голосу: Мікрофон записує голос користувача, а спеціальні алгоритми аналізують його унікальні акустичні характеристики.

Після зчитування даних система порівнює їх із збереженими шаблонами і, у разі збігу, дозволяє доступ до банківського додатка.

Плюси:

1. Високий рівень безпеки: Біометричні дані є унікальними для кожної людини, що значно знижує ризик несанкціонованого доступу.

2. Зручність використання: Користувачам не потрібно запам'ятовувати паролі або носити з собою додаткові пристрої. Достатньо лише прикласти палець або подивитися в камеру.

3. Швидкість аутентифікації: Біометрична перевірка займає лічені секунди, що робить процес входу дуже швидким і зручним.

4. Неможливість втрати або забуття: Біометричні дані не можуть бути загублені або забуті, на відміну від паролів або фізичних ключів.

Мінуси:

1. Проблеми з конфіденційністю: Збереження біометричних даних вимагає високого рівня захисту, оскільки їх компрометація може мати серйозні наслідки.

2. Технічні обмеження: Системи біометричної аутентифікації можуть давати хибні результати через різні технічні фактори, такі як погане освітлення, забрудненість сканера або зміни в зовнішності користувача.

3. Висока вартість впровадження: Впровадження біометричних систем може бути дорогим через необхідність спеціалізованого обладнання та програмного забезпечення.

4. Залежність від фізичних факторів: Травми або захворювання, що впливають на біометричні ознаки, можуть завадити користувачеві пройти аутентифікацію.

Таким чином, використання біометричних даних у банківських додатках значно підвищує рівень безпеки та зручності для користувачів, хоча і має певні виклики, які потрібно враховувати при впровадженні цієї технології.

1.2.3 Одноразові Паролі (OTP)

Одноразові паролі (OTP) є одним з найпоширеніших методів двофакторної аутентифікації. OTP – це тимчасовий код, який використовується лише один раз для входу в систему або підтвердження фінансової транзакції. Вони генеруються для кожної сесії або транзакції і зазвичай дійсні лише протягом короткого періоду часу.

Процес роботи з OTP складається з декількох кроків:

1. Запит на аутентифікацію: Користувач вводить свій основний пароль або іншу інформацію для входу в систему.

2. Генерація OTP: Система генерує унікальний одноразовий пароль. Це може бути зроблено за допомогою алгоритмів, таких як TOTP (Time-based One-Time Password)[9-10] або HOTP (HMAC-based One-Time Password)[11].

3. Передача OTP користувачу: Одноразовий пароль надсилається користувачу через вибраний канал зв'язку, наприклад, SMS, електронну пошту або через спеціальний додаток-аутентифікатор.

4. Введення OTP: Користувач вводить отриманий одноразовий пароль у відповідне поле в банківському додатку.

5. Перевірка OTP: Система перевіряє введений користувачем OTP на відповідність згенерованому коду. Якщо все збігається, користувач отримує доступ до системи або підтверджує транзакцію.

Переваги:

1. Підвищена безпека: Використання OTP значно ускладнює несанкціонований доступ, оскільки одноразовий пароль дійсний лише короткий час і для однієї сесії.

2. Зручність: Користувачам не потрібно запам'ятовувати додаткові паролі, оскільки OTP генеруються автоматично і надсилаються на їх пристрій.

3. Мінімізація ризику компрометації: Якщо одноразовий пароль перехоплено або вкрадено, його не можна використовувати повторно, що знижує ризик зловживання.

4. Сумісність: OTP можуть бути використані разом з іншими методами аутентифікації для підвищення загального рівня безпеки.

Недоліки:

1. Залежність від зовнішніх факторів: OTP зазвичай надсилаються через SMS або електронну пошту, що може створювати проблеми у разі затримок доставки або відсутності підключення до Інтернету.

2. Вразливість до атак на канал передачі: SMS та електронна пошта можуть бути перехоплені або скомпрометовані за допомогою соціальної інженерії чи шкідливого програмного забезпечення.

3. Необхідність додаткових пристроїв: Використання додатків-аутентифікаторів потребує наявності смартфону або іншого пристрою, що може бути незручно для деяких користувачів.

4. Короткий час дії: Користувачі можуть зіткнутися з незручностями, якщо OTP швидко стають недійсними, особливо в разі затримок при введенні коду.

Таким чином, OTP є ефективним засобом підвищення безпеки банківських додатків, хоча і має певні обмеження та недоліки, які необхідно враховувати при впровадженні цієї технології.

1.2 Тенденції розвитку банківських додатків

Зважаючи на швидкий технологічний прогрес та зміни в способах спілкування та використання фінансових послуг, банківські додатки також прогресують і розвиваються. Ось деякі з провідних тенденцій розвитку банківських додатків:

- Мобільність та мультиплатформенність: Застосунки стають все більш адаптованими до мобільних пристроїв, таких як смартфони та планшети. Банки активно розробляють додатки для різних платформ, таких як iOS та Android, щоб забезпечити доступність та зручність клієнтів незалежно від їх пристрою.
- Покращений користувацький досвід: Банки ставлять більший акцент на покращення користувацького досвіду у своїх додатках. Це означає простий та інтуїтивно зрозумілий дизайн, персоналізовані функції та швидкість реагування додатків на запити користувачів.
- Розширені функціональні можливості: Банківські додатки надають все більше функцій та послуг, що дозволяють клієнтам здійснювати широкий спектр операцій, таких як перекази коштів, платежі, управління рахунками та кредитними картками, перегляд історії транзакцій та багато іншого.
- Зростання безпеки: Захист даних та безпека фінансових транзакцій є пріоритетними завданнями для банківських додатків. Вони використовують різні технології шифрування, двофакторну аутентифікацію та інші заходи безпеки для забезпечення конфіденційності та захисту даних користувачів.
- Інтеграція з іншими сервісами: Банківські додатки все більше інтегруються з іншими фінансовими та непов'язаними сервісами. Наприклад, додатки можуть надавати можливість оплати рахунків, перегляду стану фінансів на зовнішніх рахунках або використання функцій цифрових гаманців.
- Використання розумних технологій: Банки досліджують можливості використання розумних технологій, таких як штучний інтелект та машинне навчання, для поліпшення банківських додатків. Це може включати

автоматизацію процесів, персоналізацію рекомендацій та аналітичні можливості.

Ці тенденції відображають стрімкий розвиток банківських додатків і підкреслюють важливість інновацій та забезпечення задоволення потреб клієнтів у цифровій епосі.

1.3 Об’єкт автоматизації

Об'єктом автоматизації є IT-відділ банку, який відповідає за підтримку та обслуговування інформаційних систем, зокрема банківських додатків. IT-відділ виконує критично важливі функції, забезпечуючи безперервну роботу внутрішніх систем, захист персональних даних клієнтів, а також ефективну обробку фінансових транзакцій. Задача автоматизації стосується оптимізації та спрощення процесів управління доступом, захисту даних та моніторингу безпеки в рамках використання серверної частини банківського додатку.

Основні процеси, що підлягають автоматизації, включають:

1. Управління користувацьким доступом: автоматизація аутентифікації та авторизації користувачів банківського додатку. Використання REST API та Spring Security дозволить забезпечити централізований контроль доступу до банківських систем і зменшити ризик несанкціонованого доступу.

2. Моніторинг безпеки та виявлення загроз: автоматизація збору та аналізу журналів подій для виявлення потенційних загроз і порушень безпеки. Система дозволить IT-відділу швидше реагувати на інциденти та забезпечувати постійний контроль за станом безпеки системи.

3. Управління базами даних: автоматизація процесів інтеграції та взаємодії з базами даних для зберігання, оновлення та захисту конфіденційної інформації клієнтів. Це шифрування даних і контроль доступу до критичних інформаційних ресурсів.

4. Автоматизація обробки запитів: автоматизація взаємодії з іншими внутрішніми та зовнішніми системами через REST API. Це спрощує роботу

IT-відділу, надаючи зручні інструменти для інтеграції банківського додатку з іншими сервісами, такими як системи обробки платежів, CRM-системи тощо.

Завдяки автоматизації цих процесів IT-відділ зможе скоротити час на виконання рутинних операцій, знизити ймовірність помилок і зосередитися на стратегічних задачах, таких як впровадження нових функцій та послуг для клієнтів банку.

1.4 Висновки до розділу

У ході аналізу предметної області було досліджено ключові аспекти банківських додатків, включаючи їх особливості, переваги та сучасні методи аутентифікації й авторизації. Банківські додатки забезпечують високу зручність для користувачів, швидкість виконання операцій, простоту управління фінансами та високий рівень безпеки.

Розглянуто сучасні методи аутентифікації, які підвищують захищеність фінансових операцій, включаючи дводіапазонну аутентифікацію (2FA), використання біометричних даних та одноразові паролі (OTP). Кожен із цих методів має свої переваги та недоліки, які впливають на зручність використання, ефективність і рівень захищеності системи.

Тенденції розвитку банківських додатків спрямовані на розширення функціональності, підвищення користувацького досвіду та впровадження інноваційних рішень у сфері безпеки. У цьому контексті вибір методів аутентифікації є ключовим для забезпечення захищеності та довіри користувачів.

Нижче наведено порівняльну таблицю основних методів аутентифікації, які використовуються у банківських додатках(таблиця 1.1):

Таблиця 1.1 – Порівняння основних методів аутентифікації

Метод аутентифікації	**Переваги**	**Недоліки**	**Приклади використання**
Дводіапазонна аутентифікація (2FA)	Високий рівень безпекиВисокий рівень безпеки Використання додаткового коду підтвердження знижує ризик несанкціонованого доступу	Потребує додаткового часу для підтвердження Залежність від пристроїв (телефон, додаток)	Використання SMS-кодів або мобільних додатків для підтвердження
Біометричні дані	Зручність для користувача Висока точність і унікальність дани	Можливість помилок через технічні проблеми Вразливість до зловживань у разі компрометації	Відбитки пальців, розпізнавання обличчя
Одноразові паролі (OTP)	Простота реалізації Можливість інтеграції з іншими методами аутентифікації	Обмежений час дії паролю Потреба в додатковому каналі зв'язк	OTP через SMS, електронну пошту або спеціальні генератори

2 ТЕХНОЛОГІ ТА ІНСТРУМЕНТИ РОЗРОБКИ

Розробка серверної частини програмного забезпечення є складним багатогранним процесом, який охоплює вибір підходів, архітектурних рішень, інструментів та технологій. У банківській сфері цей процес набуває ще більшого значення через необхідність забезпечення високого рівня безпеки, стабільності, масштабованості та продуктивності. Кожен елемент, будь то вибір мови програмування, засобів тестування чи архітектурного підходу, впливає на загальну якість і функціональність системи.

Цей розділ присвячений детальному аналізу основних технологій та інструментів, які були використані при створенні серверної частини банківського додатку. З огляду на високу конкуренцію у фінансовій сфері, розробка таких додатків вимагає сучасних рішень, які не лише відповідають поточним вимогам ринку, а й забезпечують гнучкість для майбутнього розвитку.

У цьому контексті архітектура REST API є найбільш оптимальним вибором для забезпечення зв’язку між сервером і клієнтською частиною додатку. Вона забезпечує стандартизацію, простоту інтеграції та високий рівень сумісності з різними платформами. Водночас, у розділі буде наведено порівняння REST API з іншими підходами, такими як GraphQL і SOAP, щоб підкреслити їх сильні та слабкі сторони в контексті банківських додатків.

Окрему увагу приділено вибору мов програмування, фреймворків і засобів розробки. Враховуючи особливості Java як стабільної, широко використовуваної та безпечної мови, її було обрано для реалізації проєкту. Зокрема, використання Spring Boot, популярного фреймворку, дозволило спростити конфігурацію додатку та інтеграцію функцій безпеки, таких як авторизація та аутентифікація.

Середовище розробки також має велике значення для ефективності процесу створення програмного забезпечення. У рамках даного проєкту було обрано IntelliJ IDEA як інтегроване середовище, що забезпечує зручну роботу

з кодом, інтеграцію з іншими інструментами та підтримку технологій, необхідних для створення серверної частини додатку.

Для управління залежностями і спрощення процесів розробки було обрано Maven — інструмент, який автоматизує завантаження бібліотек і полегшує створення проєкту. Підтримка модульності та стандартизації дозволяє легко масштабувати систему у разі зростання вимог.

Тестування є невід'ємною частиною процесу розробки програмного забезпечення, особливо у фінансових системах, де навіть найменша помилка може призвести до серйозних наслідків. У цьому проєкті для тестування REST API використовується Postman — популярний інструмент для моделювання запитів і перевірки роботи серверів. Для автоматизованого тестування бізнес-логіки застосовується JUnit, який дозволяє ефективно перевіряти функціональність системи.

Окремий акцент зроблено на виборі системи управління базами даних (СУБД). У рамках проєкту було вирішено використовувати MongoDB, яка є нереляційною базою даних і забезпечує гнучкість, масштабованість та високу продуктивність. Завдяки підтримці документно-орієнтованої моделі даних MongoDB ідеально підходить для зберігання великих обсягів інформації з різними структурами.

Таким чином, у цьому розділі наведено повний спектр технологій та інструментів, що забезпечують розробку сучасної, безпечної та масштабованої серверної частини банківського додатку. Ці рішення обрані з урахуванням специфіки банківської сфери, потреб клієнтів та перспектив подальшого розвитку

2.1 Підходи до розробки серверних додатків

Одним із ключових завдань у створенні серверного додатку є вибір архітектурного підходу. Сучасні підходи, такі як REST API, GraphQL і SOAP, дозволяють організувати ефективну взаємодію між сервером і клієнтською частиною[12].

2.1.1 Архітектура REST API

REST API (Representational State Transfer Application Programming Interface) — це стиль архітектури веб-сервісів, який дозволяє системам взаємодіяти через протокол HTTP. REST API є популярним вибором для створення серверів, що обслуговують клієнтські додатки (мобільні, веб тощо) завдяки своїй простоті, масштабованості та гнучкості (рисунок 2.1).

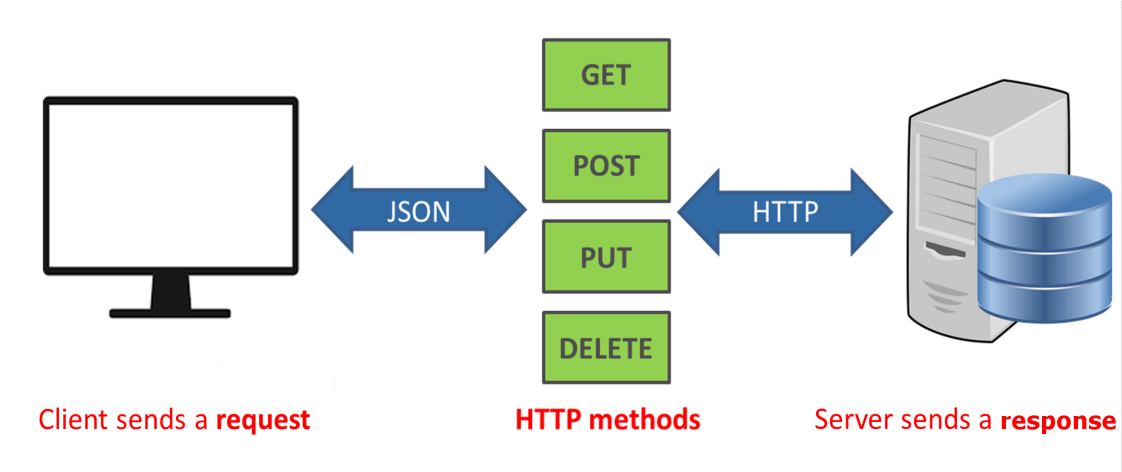

Рисунок 2.1 – Схема роботи Rest Api

Основні принципи REST

1. Клієнт-серверна архітектура: клієнт (наприклад, веб-браузер або мобільний додаток) і сервер працюють незалежно. Клієнт лише відправляє запити та отримує відповіді, а сервер обробляє запити й виконує дії.

2. Статусність (Stateless**)**: кожен запит від клієнта до сервера повинен містити всю необхідну інформацію для його обробки. Сервер не зберігає стану між запитами, що спрощує масштабування.

3. Єдиний інтерфейс (Uniform Interface):

- Ресурси: дані або функціональні можливості представлені як ресурси, що мають унікальні URL-адреси.
- HTTP-методи:
  - GET: отримання ресурсів.
  - POST: створення ресурсів.
  - PUT: оновлення ресурсів.
  - DELETE: видалення ресурсів.
  - Формати відповіді: найчастіше використовується JSON або XML.

4. Кешування: відповіді від сервера можуть бути кешовані, що дозволяє знизити навантаження на сервер і збільшити швидкість обслуговування клієнтів.

5. Шаруваність: клієнт не знає про проміжні шари (наприклад, балансувальники навантаження або проксі-сервери), що дозволяє легко масштабувати систему.

2.1.2 Порівняння REST API з альтернативами (GraphQL, SOAP)

Інші підходи, такі як **GraphQL** і **SOAP**, мають свої сильні сторони.

GraphQL — це запитувальна мова, яка надає клієнтам можливість визначати, які саме дані їм потрібні[13](рисунок 2.2).

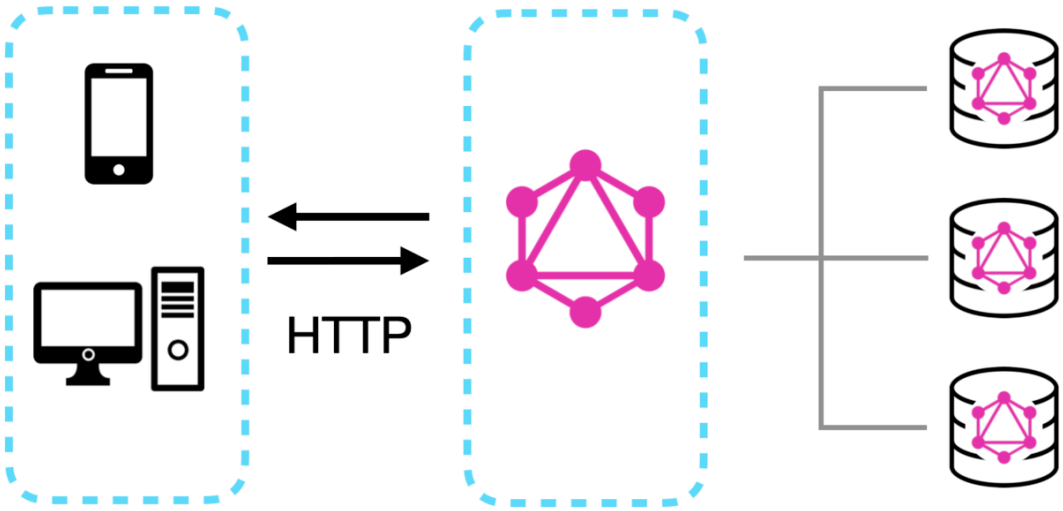

Рисунок 2.2 – Схема роботи GrahpQL

Схожість із REST API:

- Обидва підходи використовують HTTP для комунікації.
- Орієнтовані на клієнт-серверну архітектуру.

Переваги GraphQL над REST API:

1. Гнучкість у запитах: клієнт отримує лише ті дані, які запитує, без надлишкової інформації (over-fetching).

- У REST API сервер визначає структуру відповіді, а в GraphQL — клієнт.

2. Менше запитів до сервера: замість виконання кількох запитів для отримання різних ресурсів у REST, GraphQL дозволяє об'єднати все в один запит.

3. Розширення API без розриву сумісності: додавання нових полів або типів не впливає на існуючий функціонал.

Недоліки GraphQL порівняно з REST API:

1. Складність впровадження: потребує створення спеціального графа даних та інструментів для обробки запитів.

2. Високе навантаження на сервер: кожен запит виконується динамічно, що може уповільнити продуктивність при складних запитах.

3. Відсутність кешування на рівні HTTP: на відміну від REST API, GraphQL складніше інтегрувати з механізмами кешування (наприклад, CDN).

GraphQL краще підходить для складних і багатофункціональних додатків із великою кількістю взаємодій між клієнтами та сервером.

SOAP (Simple Object Access Protocol) — це протокол для передачі повідомлень у веб-службах, який ґрунтується на XML і забезпечує високий рівень стандартизації[14].

Схожість із REST API:

- Обидва використовуються для побудови веб-служб.
- Використовують мережеві протоколи, як-от HTTP чи SMTP, для передачі даних.

Переваги SOAP над REST API:

1. Безпека: SOAP підтримує розширені механізми безпеки, як-от WS-Security, що робить його ідеальним для фінансових та банківських систем.

2. Транзакції: підтримка складних транзакцій із високою точністю та надійністю, що важливо для систем із високими вимогами до цілісності даних.

3. Стандартизація: чітко визначені правила та специфікації, що забезпечує сумісність між платформами.

Недоліки SOAP порівняно з REST API:

1. Складність: використання XML замість JSON робить SOAP важчим для впровадження та розуміння.

2. Масштабованість: через високу накладну складно створювати масштабовані додатки.

3. Продуктивність: через додаткові метадані запити SOAP займають більше часу та споживають більше ресурсів, ніж REST API.

SOAP є підходящим для додатків із суворими вимогами до безпеки та транзакційності, але менш зручним для розробки сучасних веб-додатків, де простота та гнучкість важливіші.

Нижче у таблиці 2.1 наведено порівняльну таблицю архітектурних стилів

Таблиця 2.1 – Порівняння архітектурних стилів

Критерій	**REST API**	**GraphQL**	**SOAP**
Архітектурний стиль/Протокол	Архітектурний стиль, заснований на HTTP.	Архітектурний стиль, що використовує один endpoint.	Протокол, заснований на XML.
Гнучкість запитів	Фіксовані запити до конкретних ресурсів.	Дозволяє клієнту вибирати, які дані отримувати.	Запити та відповіді строго структуровані.
Продуктивність	Залежить від кількості запитів і обсягу даних у відповіді.	Ефективний, оскільки повертає тільки потрібні дані.	Може бути повільним через великий обсяг XML.
Простота використання	Простий у реалізації та розумінні.	Складніший у впровадженні через динамічні запити.	Вимагає значних зусиль для налаштування.
Підтримка типів даних	Підтримує JSON, XML та інші формати.	Використовує JSON.	Використовує XML.
Безпека	Підтримує HTTPS для шифрування даних.	Підтримує HTTPS.	Вбудовані механізми безпеки (WS-Security).

2.2 Обрані технології

Для розробки серверної частини банківського додатку були обрані сучасні технології, що забезпечують надійну й ефективну роботу.

2.2.1 Мова програмування та фреймворки

Java — це об'єктно-орієнтована мова програмування високого рівня, яка була розроблена компанією Sun Microsystems у 1995 році і згодом придбана компанією Oracle. Вона широко використовується для створення різноманітних програмних рішень, починаючи від мобільних додатків до складних серверних систем[15].

Java має кілька важливих характеристик, які роблять її популярною для розробки різноманітних типів додатків:

1. Висока продуктивність: Java має ефективну компіляцію та працює швидко на більшості платформ, що дозволяє забезпечити високу продуктивність при обробці запитів і даних.
2. Широка підтримка: Java — це одна з найбільш популярних мов програмування в світі, що дозволяє мати велику кількість ресурсів і спільнот, які можуть допомогти при вирішенні технічних проблем.
3. Інтеграція з сучасними технологіями: Java добре інтегрується з сучасними бібліотеками та інструментами, що дозволяє інтегрувати систему з різними типами баз даних, платіжними шлюзами та іншими сервісами.
4. Масштабованість та розширюваність: Оскільки банківський додаток може обслуговувати велику кількість користувачів одночасно, Java дозволяє створювати рішення, які можуть масштабуватись без значних витрат часу і ресурсів.

Фреймворки є важливими компонентами при розробці програмного забезпечення, оскільки вони дозволяють пришвидшити розробку, стандартизувати процеси та покращити безпеку й продуктивність додатків. Для серверної частини банківського додатку, зокрема при використанні мови програмування Java, були обрані фреймворки, які відповідають вимогам до масштабованості, безпеки, зручності тестування та інтеграції з іншими системами. Нижче детально розглянуті основні фреймворки, які використовуються для цього проекту: Spring Boot, Spring Security, Hibernate та Swagger.

Spring Boot — це фреймворк для розробки Java-додатків, який значно спрощує налаштування та створення програм. Він є частиною більш широкої екосистеми Spring, яка є популярною в Java-середовищі для створення корпоративних додатків[16].

Переваги:

- Простота налаштування та конфігурації.
- Інтеграція з іншими компонентами Spring (Spring Security, Spring Data тощо).
- Вбудовані сервери для легкого запуску.
- Підтримка мікросервісної архітектури.

Недоліки:

- Великий розмір фінального артефакту.
- Може вимагати додаткових налаштувань для складних систем.

Spring Security — це потужний фреймворк для забезпечення безпеки веб-додатків. Він надає функціональні можливості для авторизації, аутентифікації, контролю доступу та захисту від атак, таких як CSRF (Cross-Site Request Forgery) та XSS (Cross-Site Scripting)[17-18].

Переваги:

- Гнучкість у налаштуванні безпеки.
- Підтримка різних механізмів аутентифікації.

- Захист від поширених атак (CSRF, XSS).
- Легка інтеграція з іншими бібліотеками Spring.

Недоліки:

- Складність початкового налаштування.
- Потребує ретельної конфігурації для забезпечення максимального рівня безпеки.

Hibernate — це об'єктно-реляційний маппер (ORM), який дозволяє працювати з реляційними базами даних у об'єктно-орієнтованому стилі. Hibernate автоматично перетворює об'єкти Java в таблиці бази даних і навпаки[19].

Переваги:

- Спрощує взаємодію з базою даних через об'єктно-орієнтований підхід.
- Підтримка кешування для покращення продуктивності.
- Зручне мапування Java-об'єктів на таблиці БД.

Недоліки:

- Може бути не таким гнучким при складних запитах.
- Вимагає уважного налаштування для ефективного використання.

Swagger (тепер відомий як OpenAPI) — це інструмент для автоматичної генерації документації API та тестування веб-сервісів. Swagger дозволяє розробникам швидко створювати, тестувати і документувати RESTful API[20].

Переваги:

- Автоматична генерація документації для API.
- Тестування API без додаткових інструментів.
- Покращує взаємодію між розробниками та іншими учасниками проєкту.

Недоліки:

- Потребує оновлення документації разом із змінами в коді.
- Інтерфейс може бути обмеженим для складних тестів API.

2.2.2 Інтегровані середовища розробки (IDE)

Інтегровані середовища розробки (IDE) — це програмні засоби, які надають комплекс інструментів для написання, налагодження та тестування програмного коду. Вони значно спрощують розробку, допомагаючи зменшити час на реалізацію проєкту завдяки автоматизації багатьох процесів. Для розробки серверної частини банківського додатку з використанням Java, Spring Boot та інших технологій було обрано популярні IDE, що оптимізують процес розробки та забезпечують ефективність команди розробників[21].

Популярні IDE для розробки на Java

IntelliJ IDEA

Переваги:

1. Інтелектуальний автодоповнення коду: Інтелектуальна система підказок та автозавершення значно знижує кількість помилок та підвищує продуктивність програміста.

2. Потужні інструменти для тестування: Вбудована підтримка для тестування (JUnit, TestNG, тощо) дозволяє зручно запускати тести і працювати з результатами.

3. Інтеграція з Git: Вбудована підтримка для роботи з системами керування версіями (Git, SVN тощо)[22].

4. Підтримка Spring Framework: IntelliJ IDEA має вбудовану підтримку для розробки з використанням Spring Boot та Spring Security, що є основною технологією проєкту.

Недоліки:

1. Високе навантаження на систему: IDE може бути важким для старіших або малопотужних машин.

2. Платність: Повна версія IDE є платною, хоча є безкоштовна версія з обмеженими можливостями.

Eclipse

Переваги:

1. Безкоштовність та відкритий код: Eclipse є безкоштовним і відкритим інструментом, що дозволяє зберегти бюджети проєктів.

2. Гнучкість: Завдяки великій кількості плагінів, Eclipse можна налаштувати для роботи з різними технологіями, зокрема Spring, Hibernate, JavaFX.

3. Масштабованість: Підходить для великих проєктів завдяки широкій підтримці для командної розробки та інтеграції з CI/CD інструментами.

Недоліки:

1. Менше функцій у порівнянні з IntelliJ IDEA: Хоча Eclipse є потужним інструментом, деякі функції IntelliJ IDEA, як автозавершення та рефакторинг, є більш зручними.

2. Складність налаштування: Встановлення додаткових плагінів і налаштування середовища може зайняти більше часу, ніж в IntelliJ IDEA.

NetBeans

Переваги**:**

1. Безкоштовність: Як і Eclipse, NetBeans є відкритим програмним забезпеченням і не вимагає додаткових витрат на ліцензію.

2. Простота використання: Зручний інтерфейс для новачків, що полегшує процес вивчення та налаштування IDE.

3. Підтримка для Java, PHP, HTML, JavaScript: Це дозволяє використовувати NetBeans для різноманітних видів проєктів, а не лише для Java.

Недоліки:

1. Менша популярність у порівнянні з іншими IDE: NetBeans не має такої великої підтримки та популярності серед розробників, як IntelliJ IDEA чи Eclipse.

2. Обмежена кількість плагінів: Порівняно з Eclipse, NetBeans має менший набір плагінів та додаткових інструментів для налаштування середовища.

Вибір IDE для проєкту

У проєкті розробки серверної частини банківського додатку було вибрано IntelliJ IDEA як основне середовище для розробки. Це обґрунтовано

високою продуктивністю IDE, її зручністю для розробки з використанням Spring Boot, а також широкою підтримкою для тестування та налагодження коду. Оскільки проєкт вимагає високого рівня інтеграції з різними технологіями (Spring Security, REST API), IntelliJ IDEA забезпечує потужні інструменти для автоматичного доповнення коду, управління залежностями та інтеграції з системами контролю версій.[23]

Переваги вибору IntelliJ IDEA:

- Зручність використання: Швидкий доступ до інструментів та функцій розробки.
- Інтеграція з фреймворками: Підтримка для Spring Boot і Spring Security забезпечує ефективну розробку без потреби в додаткових налаштуваннях.
- Підтримка мікросервісної архітектури: Зручність у роботі з мікросервісами та їх тестуванні.

Недоліки:

- Вимоги до ресурсів: IDE може бути важким для машин з обмеженими ресурсами, що може вплинути на швидкість роботи при великих проєктах.
- Платна ліцензія: Для використання всієї функціональності IDE необхідно придбати ліцензію.

Отже, вибір IntelliJ IDEA був обумовлений її функціональністю, швидкістю налаштування та підтримкою технологій, що використовуються в проєкті.

2.2.3 Інструменти управління залежностями та проєктами

Інструменти управління залежностями та проєктами є важливим аспектом при розробці програмного забезпечення, оскільки вони дозволяють зручно керувати бібліотеками та пакетами, що використовуються в проєкті. Вони автоматизують процес завантаження, оновлення та налаштування

залежностей, що дозволяє розробникам зосередитись на створенні функціоналу замість того, щоб вручну керувати бібліотеками. В даному проєкті для управління залежностями та проєктами використовуються такі інструменти:

Maven — це популярний інструмент для автоматизації зборки проєктів, який забезпечує управління залежностями, виконання тестів, пакування проєкту та інші функції. Maven використовує конфігураційний файл pom.xml, який описує залежності, плагіни та інші параметри проєкту[24].

Переваги:

- Управління залежностями: Maven дозволяє автоматично завантажувати необхідні бібліотеки з центрального репозиторію, що спрощує процес інтеграції нових компонентів.
- Стандартизована структура проєкту: Maven використовує єдиний шаблон для організації проєктів, що дозволяє зберігати порядок у великих проєктах.
- Інтеграція з CI/CD: Maven легко інтегрується з різними системами безперервної інтеграції та доставки (наприклад, Jenkins, GitLab CI).

Недоліки:

- Складність для початківців: Вивчення та налаштування Maven може бути складним для новачків, особливо при налаштуванні репозиторіїв та залежностей.
- Повільність у порівнянні з іншими інструментами: Maven може бути повільним при великих проєктах через необхідність постійно перевіряти та завантажувати залежності.

Gradle є більш сучасним інструментом для керування збірками та залежностями, що є альтернативою Maven. Він пропонує високий рівень гнучкості і може працювати з різними мовами програмування, включаючи Java, Kotlin, Groovy.

Переваги:

- Швидкість: Gradle використовує інкрементальні зборки, що дозволяє зменшити час на виконання завдань, порівняно з Maven.
- Гнучкість: Завдяки скриптам на Groovy або Kotlin, Gradle дозволяє гнучко налаштовувати процес зборки під специфічні вимоги проєкту.
- Легко інтегрується з іншими технологіями: Gradle чудово працює з різними фреймворками та інструментами, такими як Spring Boot, Android, Maven.

Недоліки:

- Складність налаштування: Для початківців Gradle може бути складнішим у використанні порівняно з Maven, особливо в контексті налаштування специфічних плагінів.

У даному проєкті вибір було зроблено на користь Maven, оскільки він забезпечує стабільність і широко використовується в екосистемі Java, що дозволяє ефективно керувати залежностями та інтегруватися з іншими частинами проєкту.

2.3 Логування

Логування є важливим аспектом у розробці програмного забезпечення, особливо у системах, що працюють з конфіденційними даними, такими як банківські додатки. Воно дозволяє фіксувати події, що відбуваються у додатку, для діагностики, моніторингу та забезпечення безпеки. У даному проєкті для реалізації логування було використано Slf4j у поєднанні з Logback як реалізацією.

Slf4j (Simple Logging Facade for Java) — це універсальний інтерфейс для роботи з різними бібліотеками логування у Java, який відокремлює API логування від його реалізації. Завдяки цьому розробники можуть змінювати бібліотеки логування без змін у коді додатку, що робить систему логування більш гнучкою та масштабованою. Slf4j дозволяє використовувати один і той

самий код для взаємодії з різними бібліотеками логування, такими як Logback, Log4j чи java.util.logging, мінімізуючи залежності додатку[25].

Крім того, Slf4j підтримує маркери (Markers) для додавання контекстної інформації до логів і надає компактний синтаксис, що робить його простим у використанні та підвищує читабельність коду. Ці особливості забезпечують зручну інтеграцію з логуючими бібліотеками, надаючи розробникам можливість створювати зрозумілу й ефективну систему запису подій у додатку.

Logback — це сучасна бібліотека логування, яка є рекомендованою реалізацією для Slf4j. Розроблена авторами Log4j, вона вирізняється високою продуктивністю, гнучкими можливостями конфігурації та простотою інтеграції. Logback підтримує XML-файли для налаштування логування, що дозволяє легко задавати параметри запису логів та динамічно змінювати конфігурацію без перезапуску програми. Серед важливих функцій також варто відзначити автоматичну ротацію логів, що забезпечує створення нових файлів залежно від дати чи розміру, а також підтримку контекстної інформації (MDC), яка дозволяє додавати до логів додаткові дані, наприклад, ідентифікатор користувача або транзакції.

Logback підтримує кілька рівнів логування, які дозволяють фільтрувати події залежно від їх важливості. Рівень TRACE використовується для максимально докладної діагностики, наприклад, фіксації кожного виклику методу. Рівень DEBUG застосовується для налагодження бізнес-логіки, надаючи детальні повідомлення про дії користувача або систему. INFO фіксує ключові події програми, що є корисним для аналітики або моніторингу. Рівень WARN попереджає про можливі ризики або некритичні проблеми, а ERROR використовується для запису критичних помилок, які впливають на роботу системи та потребують негайного втручання. За необхідності логування можна повністю вимкнути, встановивши рівень OFF. Завдяки цим можливостям Logback є ефективним та зручним інструментом для організації системи логування в сучасних Java-додатках.

Переваги Slf4j та Logback**:**

1. Продуктивність**:** Logback є оптимізованим для роботи в системах із високим навантаженням.

2. Гнучкість**:** завдяки Slf4j можлива швидка зміна реалізації логування без змін у коді.

3. Масштабованість**:** підтримка різних апендерів і рівнів логування дозволяє адаптувати систему до зростання обсягів даних.

4. Простота конфігурації**:** XML-файли забезпечують зрозумілий і зручний спосіб налаштування.

Таким чином, використання Slf4j та Logback у цьому проекті гарантує зручне та ефективне логування, забезпечуючи стабільність роботи системи та можливість швидкого реагування на проблеми.

2.4 Огляд систем управління базами даних (СУБД)

Для зберігання та обробки даних банківського додатку, необхідно вибрати відповідну систему управління базами даних (СУБД). Вибір бази даних залежить від багатьох факторів, таких як тип даних, вимоги до масштабованості, продуктивності та безпеки. В даному проєкті використовується MongoDB — документно-орієнтована СУБД, що підходить для розробки високопродуктивних додатків, що потребують гнучкості в зберіганні та обробці даних.

MongoDB — це одна з найпопулярніших NoSQL баз даних, що зберігає дані у форматі BSON (Binary JSON). Вона підтримує горизонтальне масштабування та дозволяє зберігати складні структури даних без потреби у схемах, як це є в реляційних базах даних[26].

Переваги:

- Гнучкість у зберіганні даних: MongoDB не вимагає попереднього визначення схеми бази даних, що дозволяє зберігати дані в будь-якому форматі.
- Масштабованість: MongoDB легко масштабується горизонтально, що робить її підходящою для великих проєктів, що потребують обробки великого обсягу даних.
- Продуктивність: Висока продуктивність завдяки зберіганню даних у форматі JSON, що спрощує читання та запис даних.
- Підтримка агрегацій: MongoDB має потужний механізм агрегацій, що дозволяє виконувати складні запити та обробляти дані без необхідності у складних JOIN операціях.

Недоліки:

- Не підтримує транзакції на рівні документа: На відміну від реляційних баз даних, MongoDB не підтримує транзакції на рівні таблиць, що може бути проблемою для деяких типів операцій.
- Обмежена підтримка для складних зв’язків: MongoDB не є оптимальним вибором для додатків, де необхідно активно використовувати складні зв'язки між даними, такі як багаточасткові транзакції.

MongoDB була вибрана для цього проєкту через її здатність ефективно працювати з великими обсягами даних та гнучкість у зберіганні структурованої та неструктурованої інформації, що важливо для банківських додатків.

2.5 Висновки до розділу

У цьому розділі було проведено аналіз підходів до розробки серверних додатків, розглянуто архітектуру REST API та її переваги порівняно з альтернативними підходами, такими як GraphQL та SOAP. REST API

продемонстрував свою ефективність завдяки простоті, масштабованості та широкій підтримці.

Обрані технології для реалізації серверної частини включають сучасні фреймворки, такі як Spring Framework для Java, що забезпечує високу продуктивність, надійність та простоту інтеграції. Інструменти управління залежностями, зокрема Maven, сприяли оптимізації розробницького процесу.

Також було детально розглянуто логування за допомогою Slf4j та Logback. Ці інструменти забезпечують можливість гнучкого налаштування та багаторівневого відстеження стану системи, що є критично важливим для серверних додатків. У частині огляду систем управління базами даних обґрунтовано вибір MongoDB, яка забезпечує гнучкість, масштабованість і високу продуктивність для роботи з даними.

3 ПРОЕКТУВАННЯ ТА РЕАЛІЗАЦІЯ СИСТЕМИ

Проєктування є одним із найбільш відповідальних етапів розробки програмного забезпечення, що визначає не лише технічні аспекти роботи системи, а й її довговічність, ефективність та адаптивність до майбутніх змін. У даному розділі детально розглянуто ключові аспекти, пов’язані з архітектурою, модульною структурою, організацією бази даних та інтеграцією між серверною і клієнтською частинами.
Особливу увагу було приділено тому, щоб усі компоненти системи були взаємопов’язані, відповідали сучасним стандартам і забезпечували високу продуктивність, безпеку та надійність.

3.1 Вибір архітектурних рішень

Вибір архітектурних рішень є ключовим етапом у проєктуванні програмного забезпечення, оскільки він визначає структуру, логіку роботи та масштабованість системи. У даному проєкті акцент зроблено на забезпеченні надійності, безпеки, простоти підтримки та можливості інтеграції з іншими системами. Для досягнення цих цілей були обрані сучасні шаблони проєктування, такі як MVC (Model-View-Controller), Dependency Inversion, Repository, а також Security Filter.

Архітектурні рішення допомагають структурувати код таким чином, щоб його було легко зрозуміти, протестувати та модифікувати. Завдяки цьому система стає більш модульною, а її компоненти можуть бути розроблені, протестовані та розгорнуті незалежно один від одного. Це особливо важливо для банківських систем, які повинні працювати з великою кількістю даних, забезпечувати високу швидкість обробки транзакцій та відповідати суворим стандартам безпеки.

Шаблон "Модель-Вид-Контролер" (MVC)

Model-View-Controller (MVC) — це один із найбільш популярних шаблонів проєктування, який забезпечує чітке розділення відповідальностей у додатку. Його основна мета — відокремити бізнес-логіку від представлення даних, що дає змогу розробникам зосереджуватись на окремих аспектах системи. Він дозволяє розділити логіку додатку на три компоненти: модель, яка представляє дані та бізнес-логіку, вид, який відповідає за відображення даних та користувацький інтерфейс, та контролер, який обробляє запити користувача та керує взаємодією між моделлю та видом. Грубо кажучи він розділяє компоненти програми на три основних шари: модель (Model), представлення (View) та контролер (Controller). Кожен з цих шарів виконує свою функцію і має свої відповідності[27].

Модель представляє собою представлення даних або бізнес-логіки програми(Рисунок 3.1). Вона відповідає за збереження, обробку та маніпуляцію даними, а також за виконання операцій із цими даними.

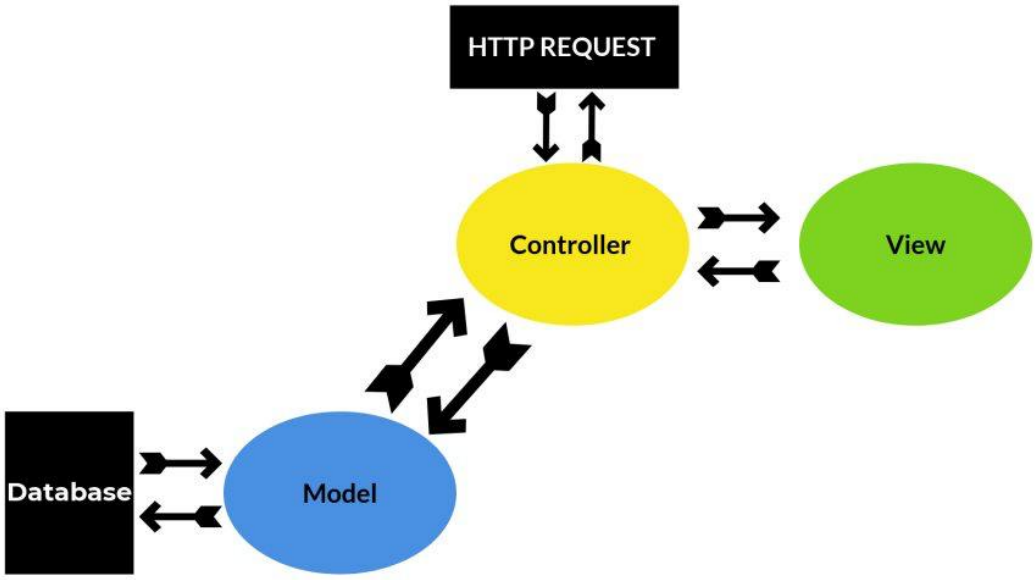

Рисунок 3.1 – Схема роботи шаблону MVC

Представлення відповідає за візуалізацію даних та взаємодію з користувачем. Воно відображає дані з моделі і надає можливість користувачеві взаємодіяти з цими даними.

Контролер служить посередником між моделлю та представленням. Він обробляє вхідні дані від користувача, взаємодіє з моделлю для отримання та оновлення даних і передає їх до відповідного представлення.

Компоненти MVC**:**

1. Модель (Model):

Містить бізнес-логіку програми та відповідає за управління даними. У цьому проєкті модель реалізує роботу з базою даних MongoDB, забезпечуючи доступ до даних користувачів та транзакцій.

2. Вид (View):

Представляє дані користувачам у зручному вигляді. У контексті серверної частини додатку вид може бути представлений у формі JSON-даних, які передаються через REST API.

3. Контролер (Controller):

Обробляє запити користувача, координує взаємодію між моделлю та видом. У цьому проєкті контролери відповідають за обробку HTTP-запитів до REST API.

Переваги MVC:

- Чітке розділення відповідальностей між компонентами.
- Зручність підтримки та розширення.
- Підтримка повторного використання компонентів (наприклад, одного виду можна використовувати з кількома моделями).
- Полегшення тестування: кожен компонент може тестуватися окремо.

Недоліки MVC:

- Для невеликих проєктів структура може бути надмірно складною.
- Взаємодія між компонентами може вимагати додаткових зусиль у налаштуванні.

У цьому проєкті MVC використовується для організації серверної частини додатку: модель управляє взаємодією з MongoDB, контролер обробляє запити до REST API, а вид відповідає за форматування відповідей клієнтам.

Також у роботі було використано шаблон Dependency Inversion. Він є основою принципу SOLID, який передбачає, що модулі вищого рівня не повинні залежати від модулів нижчого рівня — обидва повинні залежати від абстракцій(Рисунок 3.2). Цей підхід забезпечує слабке зв'язування між компонентами, що є важливим для гнучкості та підтримуваності системи[28].

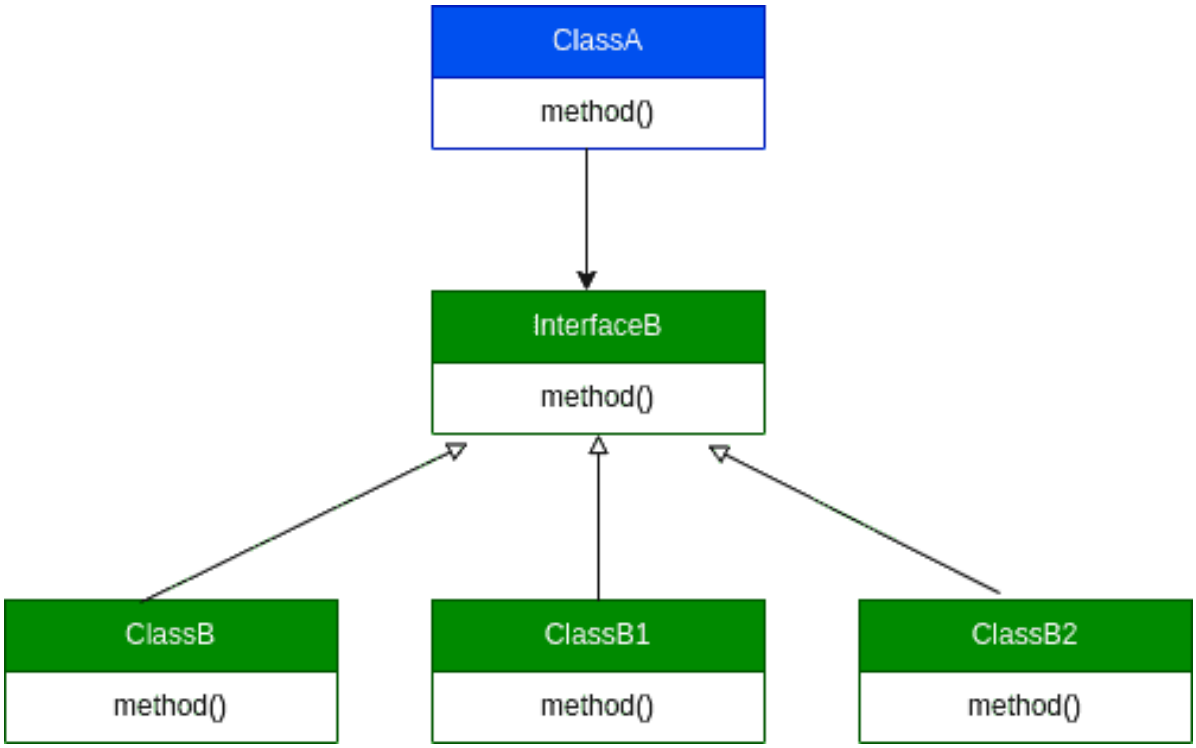

Рисунок 3.2 – Шаблон Dependency Inversion

Реалізація в проєкті**:**

У цьому проєкті шаблон Dependency Inversion реалізується через використання інтерфейсів для взаємодії між бізнес-логікою та репозиторіями. Наприклад, бізнес-логіка взаємодіє не з конкретною реалізацією репозиторію для MongoDB, а з інтерфейсом, що дозволяє легко замінити MongoDB на іншу базу даних у майбутньому.

Переваги**:**

- Полегшує тестування за допомогою замінників (mock).
- Підвищує гнучкість системи.
- Полегшує інтеграцію нових функціональностей.

Недоліки**:**

- Вимагає більше зусиль на етапі проєктування.
- Додає складність через необхідність роботи з інтерфейсами.

Шаблон Repository сприяє створенню зручного механізму для роботи з базою даних, відокремлюючи логіку доступу до даних від інших частин програми. Це дозволяє підтримувати чисту структуру коду, підвищувати його модульність і спрощує взаємодію з базою даних[29].

Repository відповідає за керування об'єктами даних: їх збереження, отримання та видалення з бази даних або інших джерел. Він забезпечує уніфікований інтерфейс для роботи з даними, приховуючи деталі їх обробки чи взаємодії з базою даних або зовнішніми системами.

Цей шаблон дозволяє легко додавати нові джерела даних і підтримувати існуючі без необхідності змінювати код високорівневих компонентів системи. Крім того, він полегшує процес тестування, адже для репозиторіїв легко створюються замінники (mocks), що особливо важливо для юніт-тестування.

Ключовий принцип роботи шаблону Repository полягає в тому, що компоненти системи, такі як сервіси чи контролери, взаємодіють із репозиторіями через стандартизований інтерфейс. Водночас репозиторій відповідає за безпосередню роботу з базою даних чи іншим джерелом інформації (рисунок 3.3). Це дозволяє чітко розподілити обов'язки між компонентами системи, забезпечуючи організованість і структурованість коду.

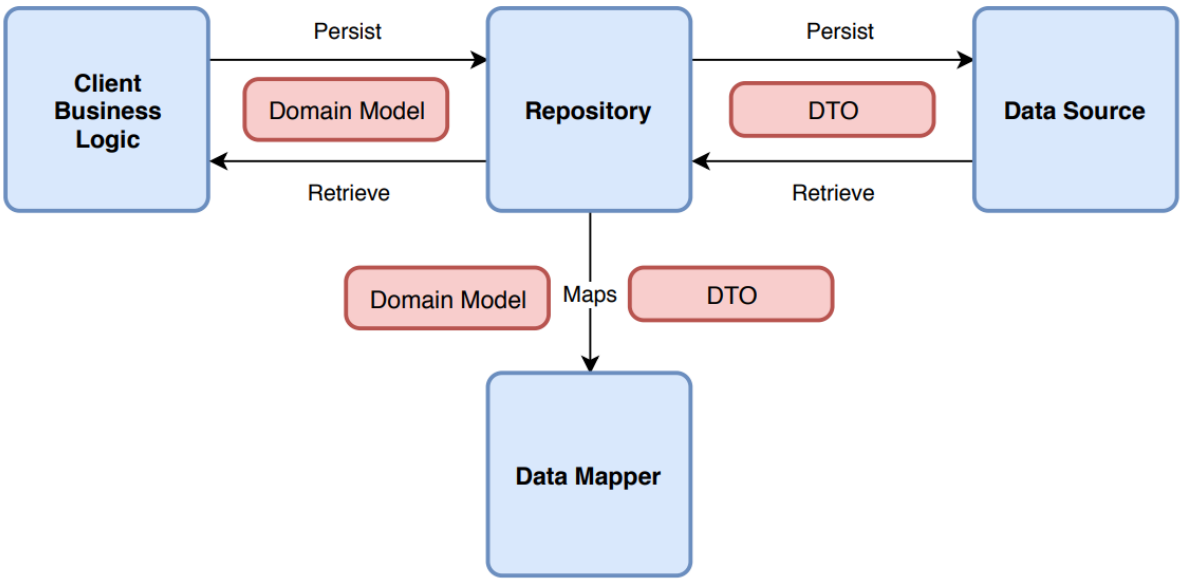

Рисунок 3.3 – Схема шаблону Repository

Реалізація в проєкті:

- Репозиторії забезпечують взаємодію між MongoDB та іншими модулями системи.
- Вони містять методи для збереження, оновлення та отримання даних, які можуть бути повторно використані в різних частинах системи.

Переваги:

- Зменшує дублювання коду.
- Забезпечує централізований доступ до даних.
- Полегшує заміну бази даних без змін у бізнес-логіці.

Недоліки:

- Додає рівень абстракції, що може ускладнити початкову розробку.

Використання цього шаблону значно покращує керованість і підтримку доступу до даних у програмних проєктах. Воно забезпечує зрозумілість, високу гнучкість і легкість у перенесенні коду, що сприяє створенню якісних та ефективних програмних рішень.

Для забезпечення надійного рівня безпеки системи та захисту конфіденційної інформації був обраний шаблон Security Filter. Його використання дає змогу ефективно контролювати доступ до ресурсів системи,

а також впроваджувати аутентифікацію та авторизацію користувачів, що є критично важливим для розробки банківських застосунків.

Security Filter відповідає за перехоплення і обробку всіх запитів, що надходять у систему, до моменту їх обробки основною логікою програми(рисунок 3.4). Він виконує широкий спектр функцій, включаючи перевірку доступу, ідентифікацію користувачів, авторизацію, шифрування даних та інші заходи безпеки[30].

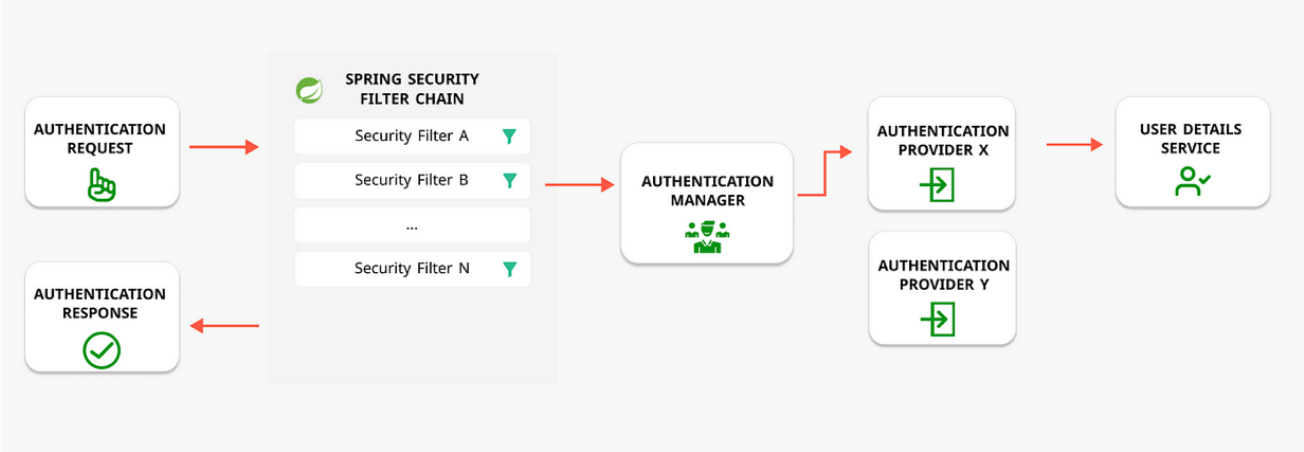

Рисунок 3.4 – Схема роботи Security Filter

Цей шаблон дозволяє централізовано управляти всіма механізмами безпеки, застосовуючи їх до всіх запитів у системі. Його структура забезпечує можливість визначення правил безпеки на рівні фільтра, що значно полегшує адміністрування та вдосконалення системи з точки зору безпеки.

Основний принцип функціонування Security Filter полягає в тому, що всі запити обов'язково проходять через фільтр безпеки перед тим, як потрапляють до логіки програми. Цей фільтр відповідає за перевірку прав доступу, підтвердження автентичності користувача, реалізацію політик безпеки та інші дії, спрямовані на забезпечення надійного функціонування системи.

Використання Security Filter дозволяє відокремити безпекові процеси від основної бізнес-логіки застосунку, що забезпечує модульність та підвищує гнучкість архітектури. Це, у свою чергу, полегшує налаштування і розширення

механізмів захисту, залежно від змінних вимог або нових викликів у сфері безпеки.

Застосування шаблону Security Filter гарантує високий рівень захищеності програмного забезпечення, зменшує ризики вразливостей і зловмисного доступу, а також сприяє дотриманню нормативних стандартів та вимог безпеки.

3.2 Проєктування структури бази даних

Проєктування структури бази даних є критично важливим етапом, який забезпечує ефективне управління даними та їхню інтеграцію в систему. Для моделювання бази даних у цьому проєкті було використано ER-модель (Entity-Relationship Model), яка допомагає візуалізувати взаємозв'язки між сутностями, їхні атрибути та зв'язки.

ER-модель (Entity-Relationship model) є концептуальною моделлю бази даних, яка використовується для опису взаємозв'язків між різними сутностями (ентитетами) в системі. Вона використовує графічні символи та правила для візуалізації та опису структури даних.

Ця модель була обрана через її простоту та універсальність, яка дозволяє ефективно структурувати складні дані. Вона забезпечує:

1. Чіткість структури даних – дозволяє візуально уявити, як дані будуть організовані.

2. Можливість масштабування – зручне додавання нових сутностей і зв'язків без значних змін у моделі.

3. Зрозумілість для всіх учасників проєкту – як для технічних фахівців, так і для замовників.

Для цього проєкту розробка бази даних почалася зі створення концептуальної ER-моделі, яка описує основні сутності системи, їхні атрибути та типи зв’язків між ними.

1. ER-модель базується на 3 основних поняттях, а саме: Сутності (Entities):

Сутність — це об'єкт або концепція, яку можна чітко визначити та про яку потрібно зберігати дані в системі. Сутності мають унікальні ідентифікатори (первинні ключі) і атрибути, що описують характеристики цих об'єктів. Наприклад, у базі даних для банківської системи сутністю може бути "Користувач", "Транзакція", "Картка", кожна з яких матиме свої власні атрибути (наприклад, ім'я, дата народження, сума транзакції).

2. Взаємозв'язки (Relationships):

Взаємозв'язки описують, як сутності пов'язані між собою. Це відображення взаємодії між двома або більше сутностями, яке вказує на те, як одна сутність взаємодіє з іншою. Наприклад, у банківській системі може бути взаємозв'язок "Має", який вказує, що один користувач може мати кілька карток, або "Здійснює", що позначає, що користувач може здійснювати транзакції.

3. Атрибути (Attributes):

Атрибути — це характеристики сутностей або взаємозв'язків, що описують деталі або властивості об'єкта. Атрибути можуть бути простими, складними, множинними тощо. Для сутності "Користувач" атрибутами можуть бути ім'я, дата народження, номер телефону, а для транзакції — сума, дата і час, тип транзакції. Атрибути є важливими для конкретизації даних і їхнього зберігання в базі даних.

Ці три основні поняття допомагають побудувати ER-модель, яка дозволяє візуалізувати структуру бази даних і забезпечити логічну організацію інформації, що дозволяє ефективно зберігати і обробляти дані в системі.

Самі взаємозв’язки бувають трьох типів:

1. Один до одного (One-to-One):

Використовується для представлення унікальних відповідностей між сутностями. Наприклад, зв’язок між Користувачем та його детальним профілем.

2. Один до багатьох (One-to-Many):

Використовується для зв’язків, коли одна сутність пов’язана з багатьма іншими. Наприклад, один Користувач може мати кілька Акаунтів.

3. Багато до багатьох (Many-to-Many):

Використовується для відображення складних зв’язків між сутностями. Для зберігання таких зв’язків застосовується додаткова таблиця, наприклад, для зв’язку між Транзакціями та Категоріями.

Основна ідея цієї моделі(рисунок 3.5) полягає в тому, щоб чітко уявити структуру даних і забезпечити правильне моделювання зв’язків між сутностями.

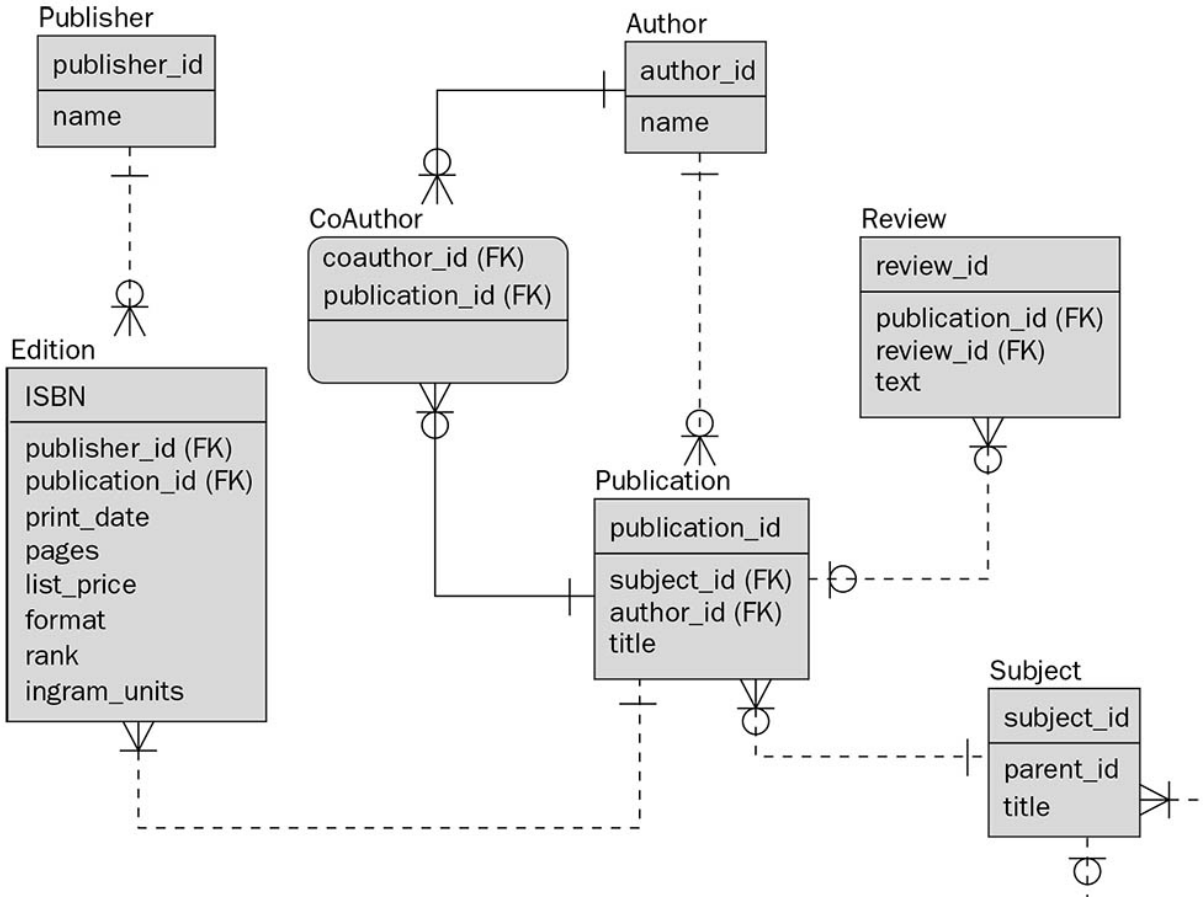

Рисунок 3.5 – Приклад ER моделі

Розробка ER-моделі є базовим кроком, який дозволяє створити логічну та фізичну схему бази даних, що забезпечує ефективне управління даними в системі.

3.3 Реалізація модулів

3.3.1 Модуль безпеки та авторизації

Для реалізації механізмів безпеки було обрано використання фреймворку Spring Security, зокрема його інструментів і методів. Основним елементом цієї реалізації є механізм аутентифікації та авторизації, що базується на використанні JWT токенів.

JWT (JSON Web Token) — це стандарт (RFC 7519) для представлення заявок безпеки в форматі JSON. Токен JWT застосовується для безпечної передачі інформації між сторонами у вигляді токенів. JWT складається з трьох основних частин: заголовка (header), заяви (payload) та підпису (signature).

1. Заголовок (Header): Описує тип токена (JWT) та алгоритм підпису, що використовувався для його створення. Заголовок кодується в форматі JSON і далі шифрується в Base64 URL-safe.

2. Заява (Payload): Містить корисну інформацію, таку як ідентифікатор користувача, ролі, додаткові дані та інші атрибути. Заява також кодується у JSON та перекодується в Base64 URL-safe.

3. Підпис (Signature): Підпис генерується на основі заголовка, заяви та секретного ключа, що дозволяє перевірити автентичність токена і гарантує, що він не був змінений після створення.

JWT токени мають низку переваг:

- Легкість передачі: Токени можуть передаватися через HTTP-заголовки, параметри URL або в тілі запиту, оскільки вони представлені в текстовому форматі.
- Самодостатність: JWT містить всю необхідну інформацію, що дозволяє перевіряти токен без необхідності звертатися до сервера або бази даних, що знижує навантаження на сервер та прискорює обробку запитів.
- Розширюваність: Заява в JWT може містити додаткові поля, що дозволяє легко розширювати функціональність токенів.

Використання JWT у поєднанні з Spring Security дозволяє ефективно забезпечити безпеку серверної частини банківського додатку та контролювати доступ до захищених ресурсів.

Аутентифікація — це процес підтвердження особи користувача або системи. Під час аутентифікації користувач надає свої облікові дані, наприклад, ім'я користувача та пароль, або використовує інші методи підтвердження (відбитки пальців, токен аутентифікації тощо). Система перевіряє ці дані та підтверджує ідентичність користувача, дозволяючи або відмовляючи в доступі.

Авторизація — це процес контролю доступу до конкретних ресурсів або функцій після того, як аутентифікація була успішною. Вона визначає, які дії або ресурси доступні для користувача на основі його прав або ролей. Після аутентифікації система перевіряє, чи має користувач відповідні права для доступу до певних ресурсів, і приймає рішення про надання або обмеження доступу.

Процес налаштування Spring Security був виконаний наступним чином: Спочатку створено конфігураційний клас, що розширює WebSecurityConfigurerAdapter і позначено анотацією @EnableWebSecurity. У цьому класі налаштовано правила безпеки, зокрема доступ до URL-адрес, а також методи для обробки аутентифікації та авторизації. Використовувався метод configure(HttpSecurity http), щоб налаштувати правила безпеки для різних запитів.

Далі була реалізована підтримка JWT. Для цього створено клас для генерації та перевірки токенів. У класі визначено методи для створення токенів, отримання інформації з них, перевірки їх валідності та розкодування.

Наступним етапом була інтеграція JWT з аутентифікацією через створення фільтра, який перевіряє наявність токена в заголовку запиту, перевіряє його валідність і встановлює дані для аутентифікації в контексті Spring Security. Щоб фільтр працював, його потрібно додати до ланцюга фільтрів у класі конфігурації для Spring Security (Рисунок 3.6)[31].

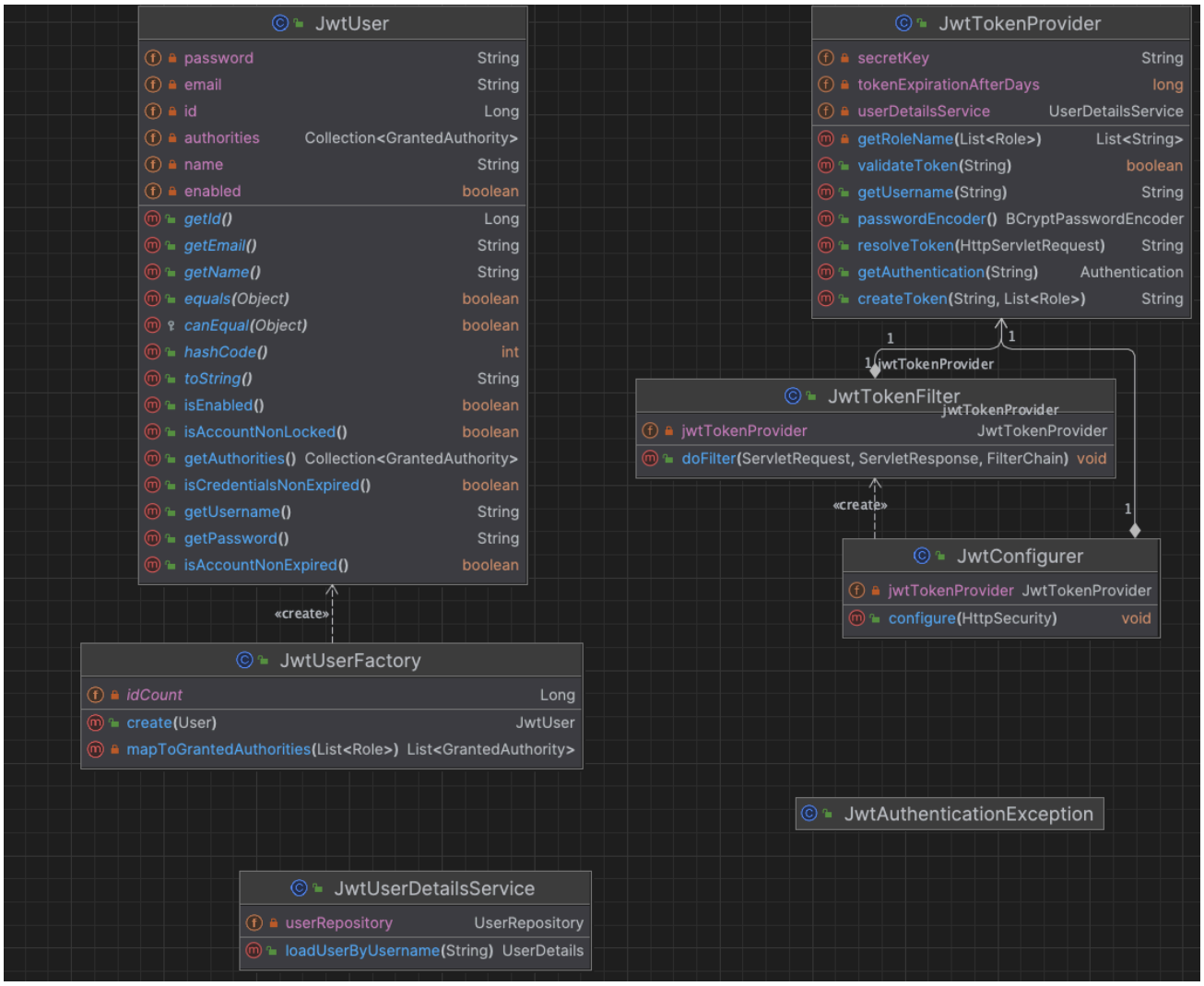

Рисунок 3.6 - Діаграма класів модулю безпеки

3.3.2 Базовий модуль бізнес логіки

Бізнес-логіка реалізована шляхом створення інтерфейсів сервісів та їх подальшою реалізацією (Рисунок 3.7). Інтерфейс є типом даних, що описує набір методів, які повинні бути імплементовані в класах, що реалізують цей інтерфейс. В інтерфейсі зазначаються тільки сигнатури методів (без їх реалізації), а також можуть бути визначені константи (фінальні та статичні змінні).

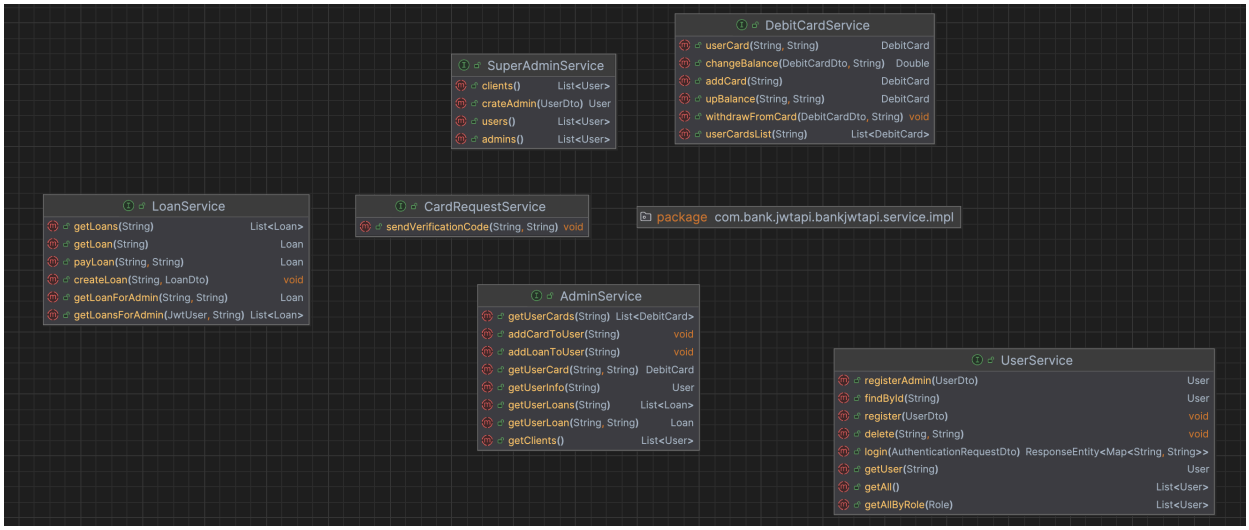

Рисунок 3.7 – Діаграма інтерфейсів

Інтерфейси використовуються для створення контракту між класами, що взаємодіють між собою. Клас, який імплементує інтерфейс, зобов'язаний реалізувати всі методи, що вказані в інтерфейсі. Це дозволяє створювати код, який залежить від абстракцій, а не від конкретних реалізацій, що забезпечує більшу гнучкість і підтримує принципи об'єктно-орієнтованого програмування, такі як поліморфізм і розподіл відповідальності.

В рамках цієї роботи сервіси (інтерфейси) задають основну логіку роботи програми — визначають її функціональність та поведінку. Реалізації цих інтерфейсів, такі як класи типу UserServiceImpl, визначають конкретну поведінку для виконання необхідних операцій.

Точки входу в програму становлять контролери, які звертаються до сервісів через їх інтерфейси. Останні через відповідні класи реалізують необхідну логіку для обробки запитів.

3.3 Інтеграція з клієнтською частиною

Для інтеграції серверної частини з клієнтською частиною програми використовуються різноманітні механізми, які забезпечують передачу даних і виконання операцій між ними. У нашому випадку інтеграція здійснюється

через використання REST API та JWT аутентифікації для забезпечення безпеки, а також через бізнес-логіку, яка реалізована за допомогою інтерфейсів сервісів та їх конкретних реалізацій. Це дозволяє з'єднати клієнтську та серверну частини програми та забезпечити взаємодію між ними.

Інтеграція через безпеку з використанням JWT

Для забезпечення безпеки між клієнтською та серверною частинами застосовуються механізми аутентифікації та авторизації через JWT токени, що реалізовані за допомогою фреймворку Spring Security. Клієнтська частина взаємодіє з сервером через HTTP запити, включаючи JWT токен у заголовку кожного запиту для ідентифікації користувача та перевірки його прав доступу(рисунок 3.8).

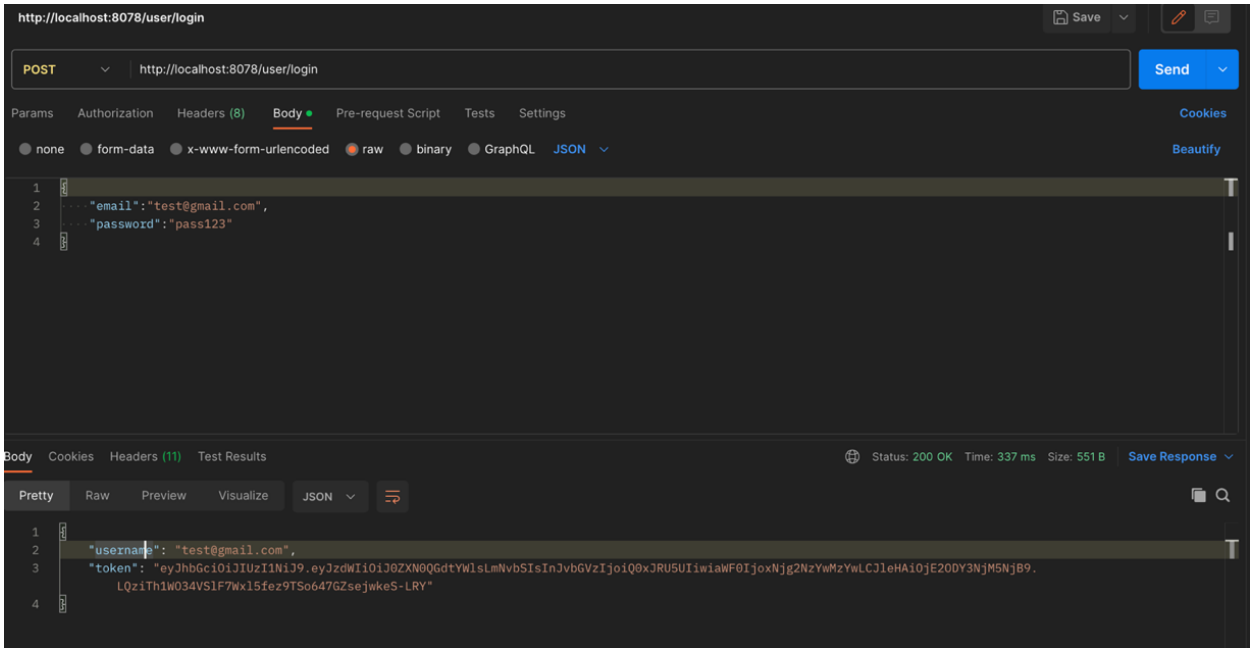

Рисунок 3.8 – приклад роботи з JWT токенами

1. Аутентифікація та Авторизація:

- Клієнт (зазвичай веб-браузер або мобільний додаток) надсилає запит на сервер для аутентифікації (наприклад, логін з ім'ям користувача та паролем).
- Після успішної аутентифікації сервер генерує JWT токен, який клієнт зберігає на боці користувача і додає до заголовка запитів для подальшої авторизації.

- Кожен запит до захищених ресурсів (які потребують авторизації) включає токен, що дозволяє серверу перевірити, чи має користувач відповідні права доступу.

2. Робота з токенами:

- Токен містить в собі необхідну інформацію для перевірки користувача (наприклад, його ідентифікатор та ролі).
- Завдяки тому, що токен самостійно містить усі необхідні дані, сервер може перевіряти автентичність токена без необхідності звертатися до бази даних або інших серверів, що покращує швидкість обробки запитів.

3. Фільтри безпеки:

- На сервері використовується фільтр, який перевіряє наявність токена в заголовку запиту та здійснює його валідацію.
- Якщо токен є валідним, фільтр додає користувацькі дані в контекст безпеки, дозволяючи подальшу обробку запиту.

Інтеграція через бізнес-логіку за допомогою інтерфейсів

Бізнес-логіка клієнтської частини програми взаємодіє з сервером через REST API, яке на сервері надається контролерами. Контролери викликають відповідні сервіси, які реалізують бізнес-логіку.

1. Використання інтерфейсів:

- Інтерфейси сервісів визначають загальний набір методів для обробки запитів клієнта. Наприклад, через інтерфейс UserService можуть бути реалізовані методи для реєстрації користувача, входу в систему, отримання профілю тощо.
- Клас, який реалізує цей інтерфейс (наприклад, UserServiceImpl), забезпечує конкретну бізнес-логіку для кожного з методів.
- Завдяки такому підходу, сервісний рівень програми є гнучким і модульним, що спрощує підтримку та розширення програми.

2. Контролери як точка входу:

- Контролери є точкою входу для клієнтських запитів. Вони отримують HTTP запити від клієнтської частини (наприклад, запит на

створення нового користувача або отримання даних профілю) і передають їх до відповідних сервісів для обробки.

- Клієнтська частина надсилає запити через HTTP (зазвичай через AJAX або інші HTTP-бібліотеки), що дозволяє отримувати потрібні дані або виконувати операції на сервері.

3. Взаємодія між клієнтом та сервером:

- Клієнтська частина відправляє запити за допомогою REST API до серверної частини, яка на основі отриманих запитів викликає відповідні сервіси для обробки даних.
- Для отримання даних або виконання операцій користувач через клієнтський додаток може здійснити запит (наприклад, через метод GET або POST), що буде оброблено сервером і відповіді на запити будуть повернуті клієнту(рисунок 3.9).

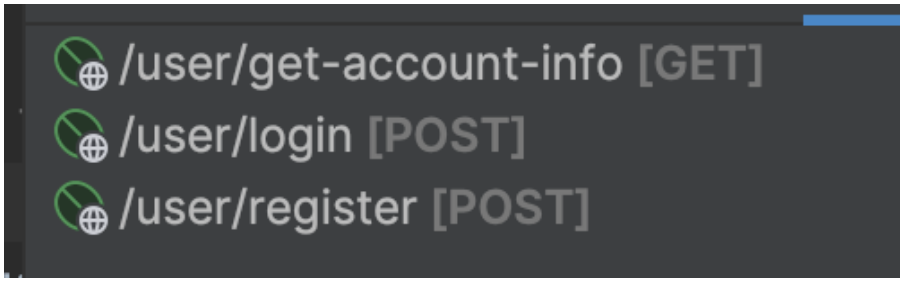

Рисунок 3.9 – Приклад запитів

Інтеграція клієнтської та серверної частини програми забезпечується через використання REST API, JWT токенів для аутентифікації та авторизації користувачів, а також через чітко визначену бізнес-логіку, реалізовану за допомогою інтерфейсів сервісів. Клієнтська частина надсилає запити на сервер, а сервер обробляє ці запити, виконуючи необхідні операції відповідно до бізнес-логіки, і повертає відповіді назад клієнту.

3.4 Валідація вхідних запитів та відповідей

Для того щоб впевнитись, що запити та відповіді як зі сторони клієнтської частини так і серверної частини дійшли туди куди треба, було використано сигнатуру. Сама сигнатура формується з тіла запиту та підписується за допомогою алгоритму шифрування MD5[32] та секретного ключа який є унікальним для кожної інтеграції. Після цього результат переводиться у формат Base64(рисунок 3.10).

```
public static String generateSignature(String payload, String secretKey) {
    // Конкатенуємо payload з секретним ключем
    String combined = payload + secretKey;

    // Хешування за допомогою MD5
    byte[] hash = DigestUtils.md5(combined);

    // Перетворення в Base64
    return Base64.getEncoder().encodeToString(hash);
}
```

Рисунок 3.10 – Приклад генерації сигнатури

Далі ця сигнатура передається у заголовку запиту та відповіді. При отриманні відповіді від серверної частини, клієнт повинен згенерувати сигнатуру згідно алгоритму та звірити результат з тим що прийшло у заголовку. Також при надсиланні запиту, згенерована сигнатура повинна бути відправлена у заголовку, що у свою чергу буде вже перевірятись на стороні серверу.

Таким чином, це підвищує захист унеможливлюючи підробку запитів, а також несанкціонованих змін.

3.5 Висновки до розділу

У ході роботи над проєктом було розроблено та реалізовано серверну частину банківського додатку, дотримуючись сучасних принципів програмування та з використанням перевірених технологій.

Розробка структури бази даних на основі MongoDB дозволила забезпечити гнучкість, масштабованість і швидкодію додатку. Використання нереляційної бази даних виявилося оптимальним рішенням для обробки великих обсягів даних і динамічної структури об’єктів, характерної для банківських операцій.

Модуль безпеки та авторизації, побудований на основі Spring Security, дозволив створити надійний механізм для захисту даних і ресурсів системи. Інтеграція з клієнтською частиною була реалізована за допомогою JWT (JSON Web Token), що забезпечує безпечну ідентифікацію користувачів і зручний обмін даними між клієнтом та сервером.

Реалізація бізнес-логіки була виконана через чітко організовану архітектуру, що використовує сервіси та інтерфейси. Такий підхід сприяє модульності, полегшує підтримку та розвиток системи. Він дозволив ефективно впровадити функціональність для управління рахунками, здійснення транзакцій, перегляду історії операцій тощо.

Таким чином, розроблена система відповідає сучасним вимогам у сфері банківських додатків, поєднуючи високу продуктивність, гнучкість у налаштуванні, безпеку та зручність інтеграції з клієнтськими інтерфейсами.

4 ТЕСТУВАННЯ ДОДАТКУ

4.1 Огляд методів тестування

У процесі розробки програмного забезпечення тестування відіграє ключову роль, оскільки забезпечує якість, надійність та безпеку системи. Для цього використовуються різноманітні інструменти та фреймворки, які дозволяють перевірити функціональність, коректність і продуктивність коду. У цьому розділі буде розглянуто два популярних інструменти, що широко застосовуються для тестування різних аспектів програмного забезпечення: Postman, призначений для тестування API, та JUnit, який використовується для модульного тестування Java-додатків.

Postman і JUnit є невід'ємними частинами сучасного процесу розробки, забезпечуючи зручність у створенні тестів, їх автоматизації та інтеграції в CI/CD процеси. У цьому розділі буде наведено детальний опис цих інструментів, їх принципи роботи, переваги та недоліки, що допоможе зрозуміти їхню важливість у тестуванні різних компонентів програмного забезпечення.

4.1.2 Postman

Postman – це один із найпопулярніших інструментів для тестування, налагодження та автоматизації API, що надає зручний інтерфейс для створення та виконання HTTP-запитів, перевірки відповідей сервера та організації процесу тестування. Він є незамінним для розробників і тестувальників, які працюють над забезпеченням якості програмного забезпечення. Postman дозволяє створювати запити різних типів, таких як GET, POST, PUT, DELETE, PATCH тощо, а також налаштовувати їх параметри: заголовки, тіла запитів і додаткові параметри. Після виконання

запитів користувач може переглядати відповіді сервера, включно з HTTP-статусами, заголовками та вмістом тіла відповіді, що дає змогу проводити детальний аналіз роботи API[33].

Postman дозволяє зберігати запити в колекціях для повторного використання, групувати їх за проектами або функціональними можливостями, а також автоматизувати тести. Однією з ключових переваг інструмента є можливість тестування API з різними типами авторизації, такими як JWT токени, Basic Auth, OAuth, що особливо зручно при роботі над проєктами, які вимагають захищеного доступу до ресурсів API (рисунок 4.1).

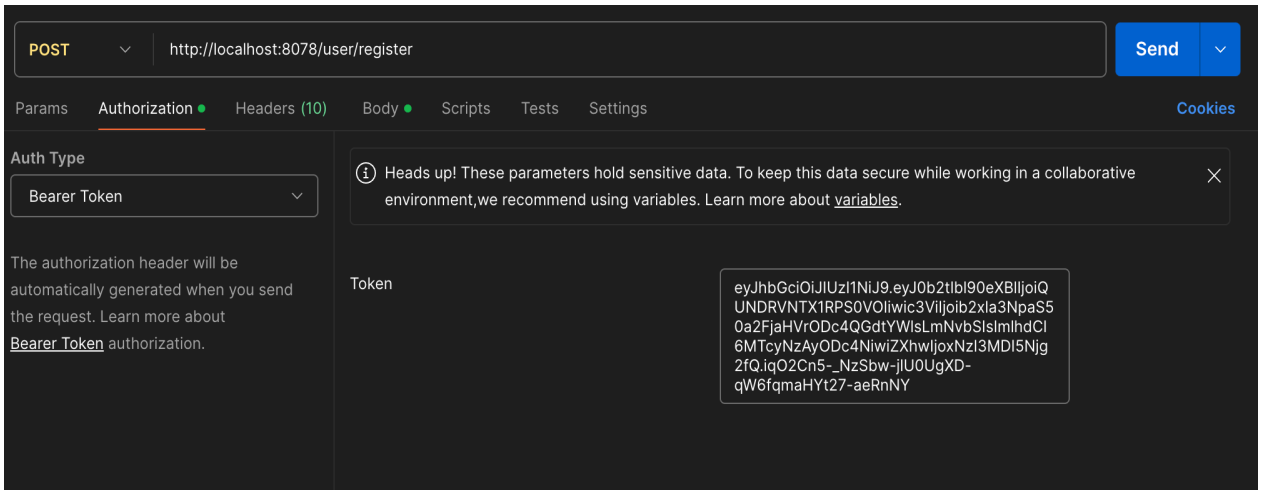

Рисунок 4.1 – Приклад запиту з JWT токеном

Середовище тестування Postman Environment спрощує роботу з динамічними або повторюваними даними завдяки підтримці змінних, які можна використовувати в запитах і колекціях. Інструмент також дозволяє створювати тести на JavaScript для автоматичної перевірки результатів, наприклад, коректності статусу HTTP, структури відповіді або значень окремих параметрів. Ця функціональність сприяє підвищенню ефективності тестування та зменшує кількість ручної роботи.

Додатково Postman можна інтегрувати із системами CI/CD, що забезпечує безперервну інтеграцію та розгортання, а також дозволяє автоматизувати перевірку API в рамках загального процесу розробки. Зручний

обмін даними між членами команди реалізований через функції експортування та імпортування колекцій запитів, що спрощує співпрацю та прискорює виконання завдань у команді.

4.1.2 JUnit

JUnit – основний інструмент для тестування Java-додатків.

Він є одним із найпоширеніших фреймворків для модульного тестування в Java. Він забезпечує розробників потужними інструментами для написання та виконання тестів, перевірки правильності роботи окремих компонентів програми та їх інтеграції. Цей фреймворк базується на принципах тестування, орієнтованого на розробку (TDD – Test-Driven Development), що дозволяє створювати тести ще до реалізації основного функціоналу, забезпечуючи високу якість і стабільність програмного забезпечення(рисунок 4.2) [34].

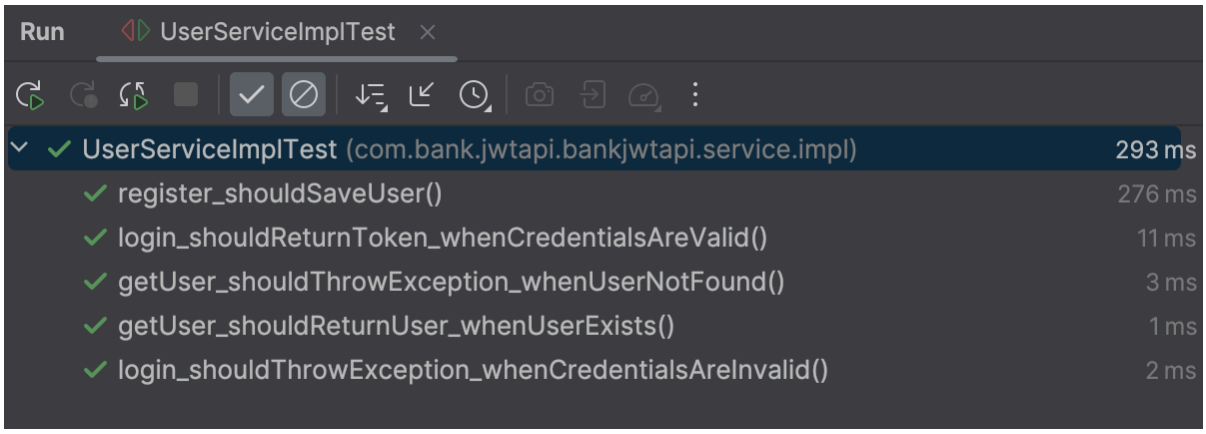

Рисунок 4.2 – Приклад виконання модульних тестів

JUnit дозволяє визначати тести як методи, які позначаються спеціальними анотаціями, такими як @Test, що автоматично ідентифікує їх як тестові сценарії. За допомогою інструментів JUnit розробники можуть перевіряти поведінку окремих методів або класів, використовуючи асерти

(наприклад, assertEquals, assertTrue, assertFalse). Це дає змогу автоматизувати перевірку результатів виконання коду та швидко виявляти помилки. Наприклад, якщо метод повертає несподіване значення, тест негайно повідомить про проблему.

Основна сила JUnit полягає у структурі та зручності організації тестів. Розробники можуть групувати тести у набори (test suites), що дозволяє запускати декілька пов'язаних тестів разом, та застосовувати попередню ініціалізацію середовища за допомогою анотацій, таких як @BeforeEach або @BeforeAll. Це значно спрощує підготовку до тестування складних сценаріїв, коли потрібно створити базові дані чи налаштування.

JUnit підтримує інтеграцію з системами збірки, такими як Maven і Gradle, що дозволяє автоматизувати виконання тестів у процесі компіляції або розгортання(рисунок 4.3). Він також легко інтегрується з інструментами для аналізу покриття коду, такими як JaCoCo, що допомагає оцінити, які частини програми залишаються неперевіреними.

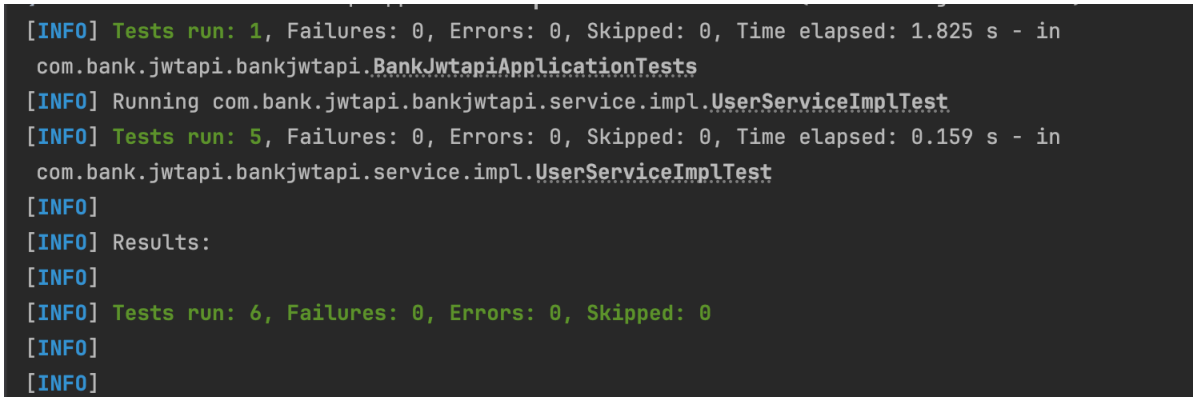

Рисунок 4.3 – Приклад виконання тестів під час компіляції проекту з використанням Maven

Однією з головних переваг JUnit є його висока швидкість виконання тестів, завдяки чому він ідеально підходить для швидких ітерацій у процесі розробки. Крім того, JUnit є відкритим і безкоштовним інструментом, що робить його доступним для широкого кола користувачів.

Однак JUnit має і деякі обмеження. Наприклад, він переважно орієнтований на модульне тестування, що означає, що для інтеграційних тестів або тестування в реальному середовищі можуть знадобитися додаткові інструменти або фреймворки. Також написання тестів може зайняти багато часу, особливо для складних проєктів із великою кількістю залежностей.

Загалом, JUnit є ефективним рішенням для розробників Java, яке забезпечує високу якість коду, полегшує виявлення помилок і сприяє швидкому та надійному розгортанню програмного забезпечення.

4.2 Тестування з використанням Postman

Для перевірки коректної роботи сервісного додатка необхідно відсилати HTTP запити до сервера, що дозволяє тестувати функціональність API. Як було зазначено в попередньому розділі, для цієї мети використовувався інструмент Postman. Це потужний і зручний інструмент для тестування API, який дозволяє створювати запити різних типів, налаштовувати їх параметри та отримувати відповіді від сервера. Завдяки широкому набору функцій Postman значно спрощує процес тестування, дозволяючи автоматизувати перевірки та організувати ефективний робочий процес для тестувальників і розробників.

У цьому розділі розглянуто, як використовується Postman для перевірки роботи сервісного додатка.

За допомогою цього інструмента було простестовано основні методи в контролерах(Рисунок 4.4).

```
@GetMapping("/users")
public ResponseEntity<List<User>> getUsers() {
    List<User> users = userRepository.findAll();
    return new ResponseEntity<>(users, HttpStatus.OK);
}
```

Рисунок 4.4 - Приклад метода контролера

Методи UserController які будуть тестуватись:

public void addUser(@RequestBody UserDto userDto): очікує дані користувача(рисунок 4.5) та реєструє його у системі(рисунок 4.6).

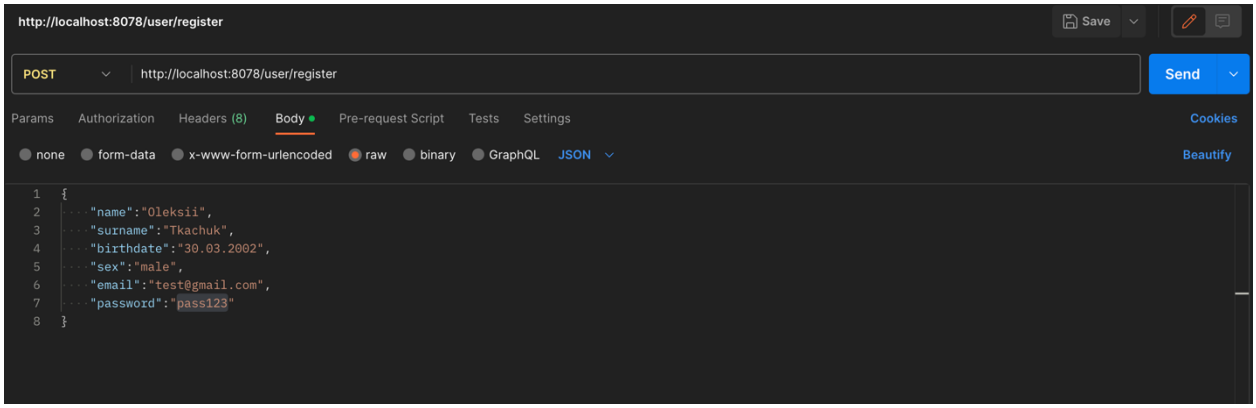

Рисунок 4.5 – Реєстрація користувача

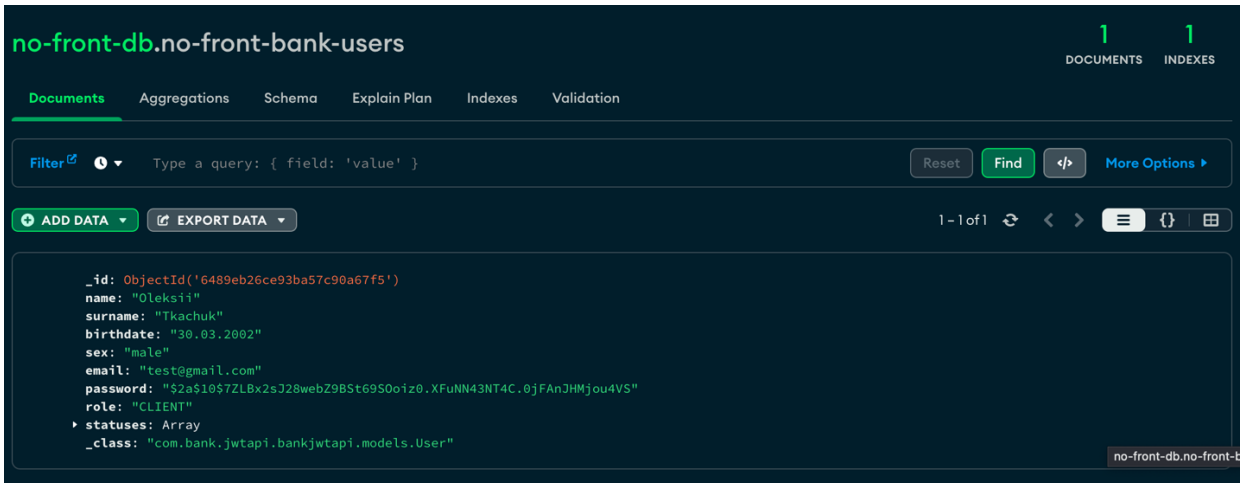

Рисунок 4.6 – Сутність користувача у базі даних

public ResponseEntity<Map<String, String>> login(@RequestBody AuthenticationRequestDto requestDto): очікує дані вже зареєстрованого користувача та виконує вхід в систему та повертає його пошту та JWT токен(рисунок 4.7).

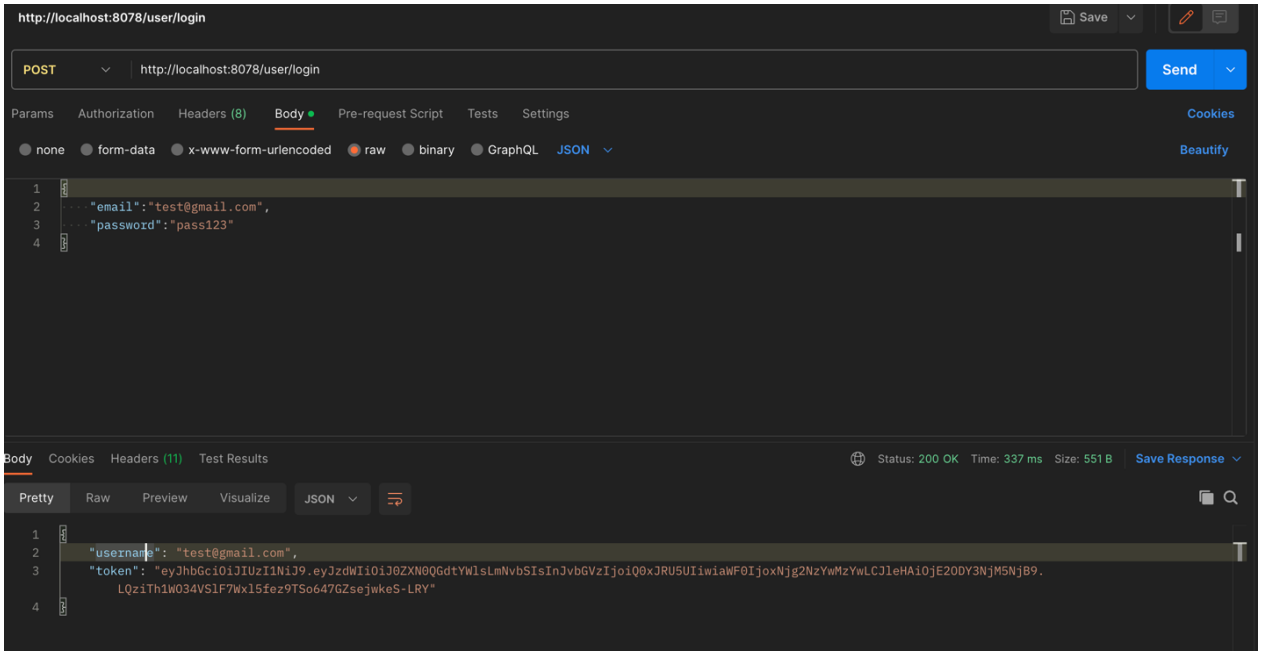

Рисунок 4.7 – Вхід до системи

Методи DebitCardService які будуть тестуватись:

public DebitCard addCard(String userId): створює карту у базу даних та прив’язує її до коритсувача(рисунок 4.8). Повертає унікальний ідентифікатор в базі даних та згенерований номер самої банківської картки(рисунок 4.9).

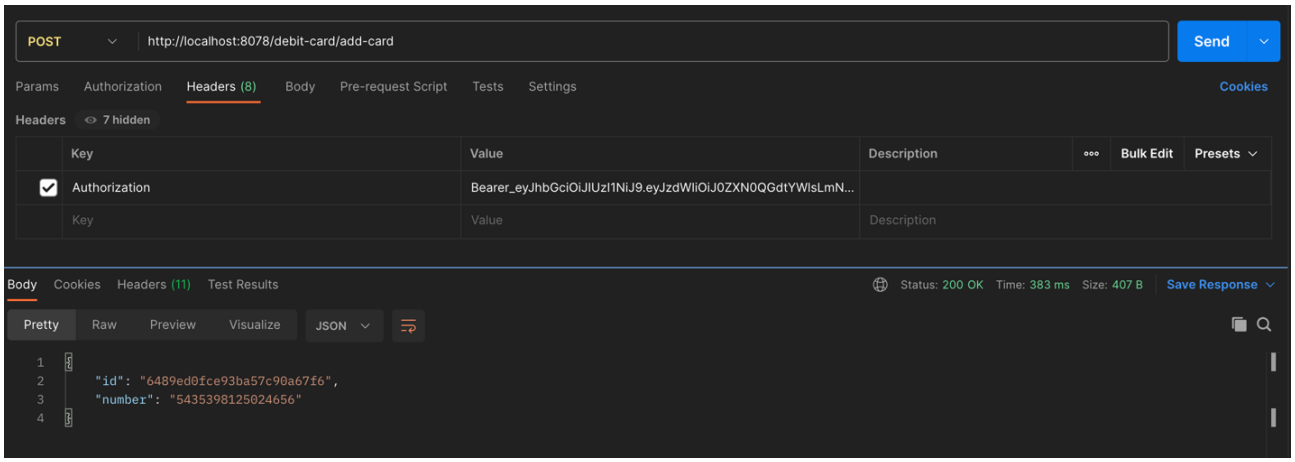

Рисунок 4.8 – Створення банківської картки

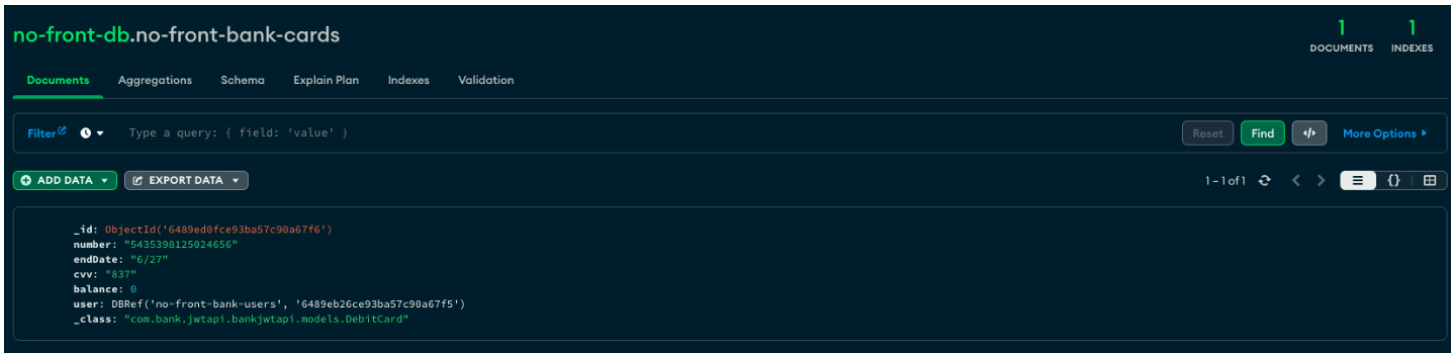

Рисунок 4.9 – Сутність банківської картки у базі даних

public List<DebitCard> cardList(JwtUser jwtUser): повертає список номерів карток, які належать користувачу(рисунок 4.10).

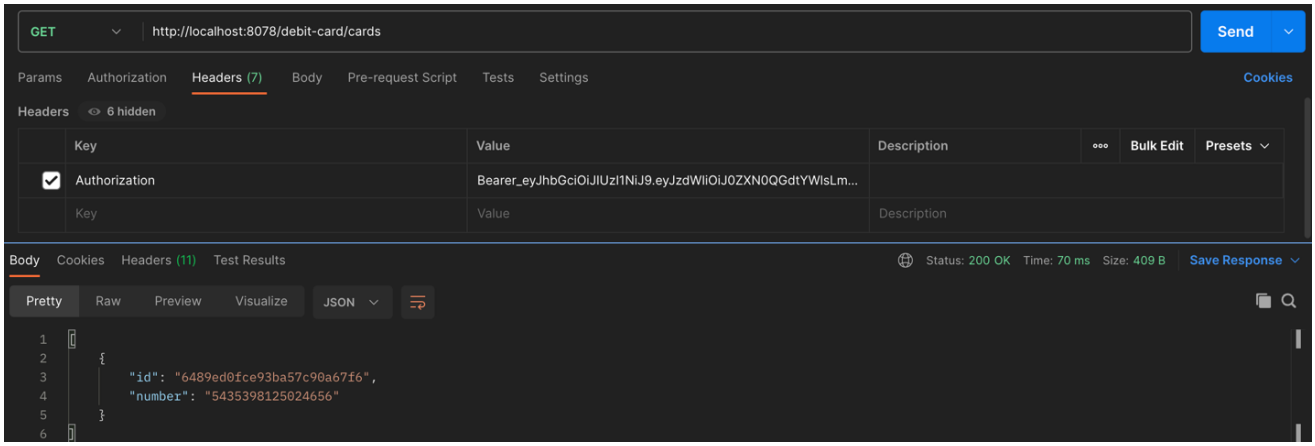

Рисунок 4.10 – Отримання списку карток користувача

Методи LoanService які будуть тестуватись:

public void createLoan(JwtUser jwtUser,@RequestBody LoanDto loanDto): створює позику(рисунок 4.11) та при’язує її до користувача(рисунок 4.12).

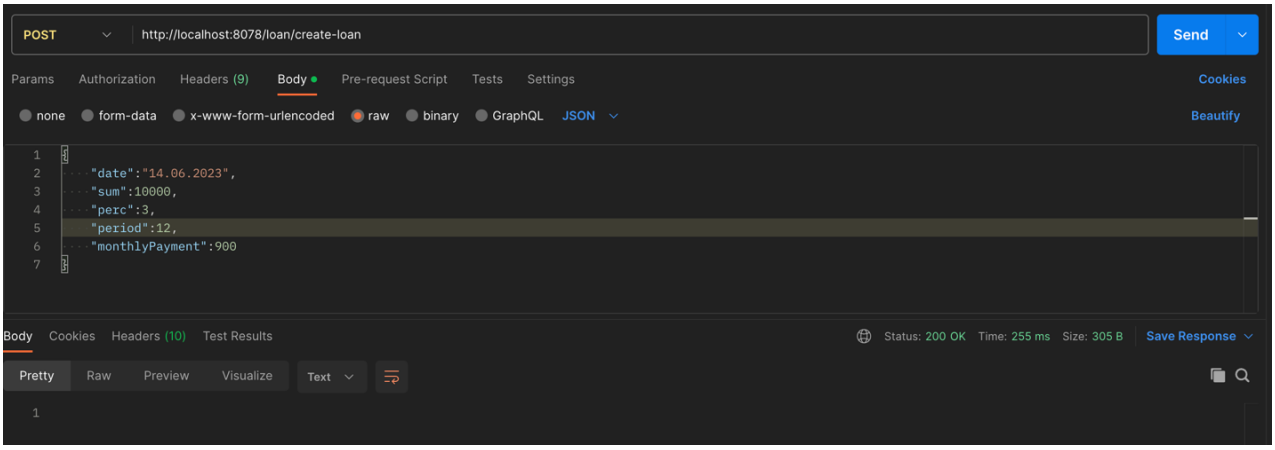

Рисунок 4.11 – Створення позики

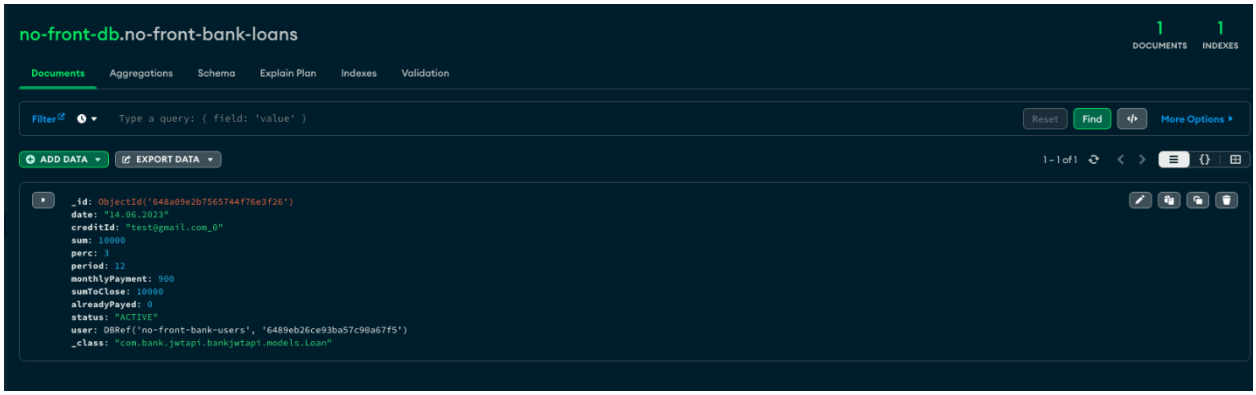

Рисунок 4.12 – Сутність позики у базі даних

public List<Loan> getLoans(JwtUser jwtUser): повертає список всіх позичок користувача(рисунок 4.13).

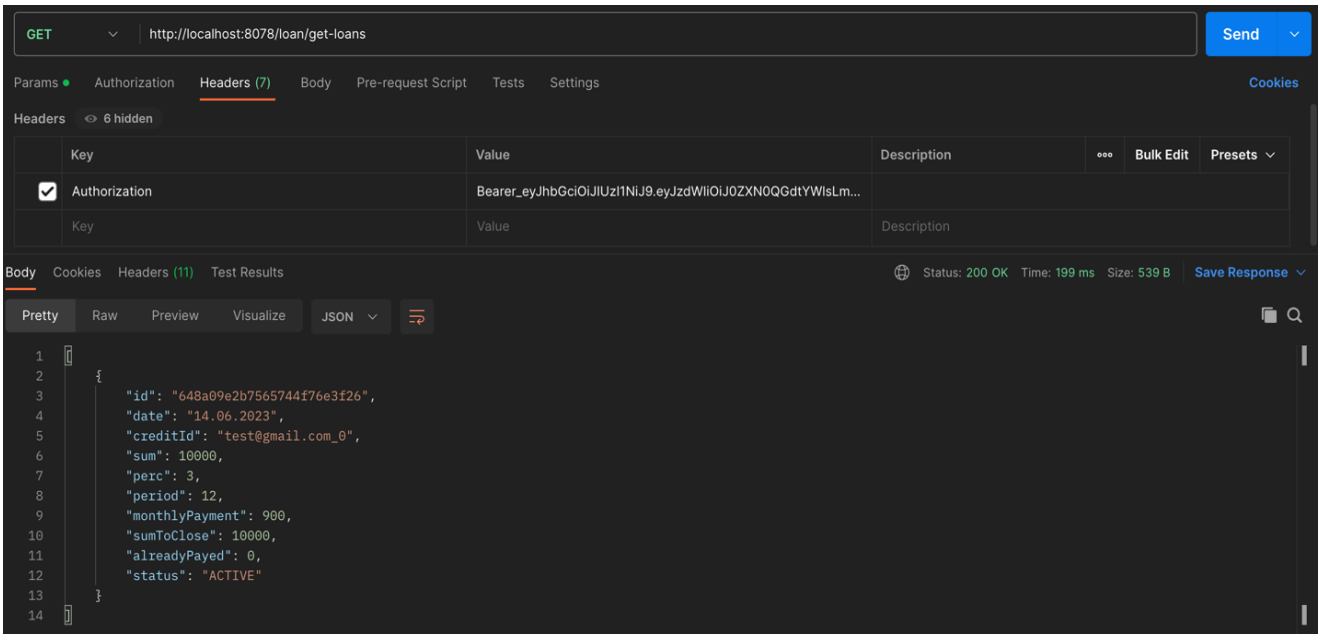

Рисунок 4.13 – Список позик

Результати тестування показали, що розроблене програмне забезпечення відповідає початковим вимогам та забезпечує стабільну роботу додатку конфігурації платформи розіграшів незалежно від програмного забезпечення яке вони використовують, а також більш гнучкому налаштуванні.

4.3 Порівняння з аналогами

Розроблена система банківського додатку має низку переваг і недоліків порівняно з аналогічними системами, такими як традиційні SOAP-базовані сервіси, системи з авторизацією без JWT, та REST API з іншими способами захисту даних (наприклад, OAuth 2.0). Нижче, у таблиці 4.1, наведено порівняльну таблицю.

Характеристика	**Розроблена система (REST API, JWT, MD5)**	**SOAP-базовані сервіси**	**REST API з OAuth 2.0**	**Старі системи без JWT**
Продуктивність	Висока, легка взаємодія	Низька через складність протоколу	Висока	Помірна
Безпека	Висока (JWT, сигнатура MD5)	Висока (завдяки WS-Security)	Дуже висока (OAuth 2.0)	Низька
Масштабованість	Висока	Складна інтеграція через SOAP	Висока	Обмежена
Простота розробки	Середня (потрібне налаштування JWT)	Складна через XML і WS-*	Середня	Висока
Простота інтеграції з клієнтом	Висока (JSON, REST API)	Низька (XML, складний формат)	Висока	Помірна
Аутентифікація	JWT (швидка перевірка токена)	WS-Security (сертифікати)	OAuth 2.0 (багаторівневий)	HTTP Basic Auth
Захист даних	MD5 сигнатури + SSL	SSL + WS-Security	SSL + токени OAuth	Тільки SSL

Продовження таблиці 4.1

Гнучкість бази даних	MongoDB (JSON-структури)	SQL-системи (менш гнучкі)	Різні БД	Залежить від реалізації
Використання в фінансовому секторі	Відповідає сучасним вимогам	Використовується в старих системах	Відповідає стандартам	Обмежене застосування

4.4 Висновки до розділу

Розділ присвячений тестуванню додатку, що є важливим етапом у забезпеченні його надійності та відповідності функціональним вимогам. Було розглянуто основні методи тестування, такі як функціональне, інтеграційне та автоматизоване тестування. Для перевірки роботи API використано Postman, який забезпечив швидкий і зручний інструмент для ручного тестування HTTP-запитів.

JUnit був використаний для автоматизації тестування ключових модулів системи, що дозволило виявити можливі помилки на ранніх етапах розробки. Завдяки поєднанню інструментів Postman та JUnit забезпечено високий рівень перевірки якості додатку, що підвищило його стабільність і відповідність вимогам.

5. ЕКОНОМІЧНИЙ РОЗДІЛ

5.1 Технологічний аудит розробленої системи автоматизації процесу аутентифікації та управління доступом у банківських системах

Як було зазначено раніше, у сучасному світі інформаційні технології стали невід'ємною частиною багатьох сфер життя людини та суспільства, суттєво впливаючи на їх функціонування. Особливо це стосується фінансової галузі, де розвиток цифрових технологій та електронних платежів привів до зростання популярності банківських додатків та віртуальних фінансових послуг. Тому ство-рення безпечних, ефективних і надійних банківських додатків стало важливою і актуальною частиною фінансового сектора економіки.

Але зі зростанням популярності банківських програм постійно виникають все нові і нові виклики, пов’язані насамперед з необхідністю забезпечити захист персональних даних клієнтів і зміст фінансових операцій, що стало критичним завданням для розробників банківських програм.

Тому метою нашої магістерської кваліфікаційної роботи була розробка ав-томатизованої системи, що забезпечить авторизацію, аутентифікацію та безпеку даних користувачів, які здійснюють операції у банківських установах.

Для розв’язання поставленої мети нами було розв’язано такі задачі: проведено аналіз предметної області, а саме проведено аналіз банківських додатків і їх методи авторизації та аутентифікації, функціональні та технічні вимоги, що дало можливість визначити напрями удосконалення автоматизованої системи; розроблено архітектуру проєкту та обрано інструменти та програмні засоби для розробки системи; розроблено серверну частину банківської програми; протестовано автоматизовано систему та порівняно її з аналогами.

В результаті було розроблено серверну частину банківського додатку з використанням REST API та Spring Security, що підвищить рівень безпеки та

забезпечить високий рівень захисту даних клієнтів і їх конфіденційність. Це дозволить банківським установам ефективніше захищати дані клієнтів, а розробникам створювати більш безпечні та надійні додатки.

Для встановлення комерційного потенціалу розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах було запрошено 3-х відомих експертів-фахівців: М.В. Микитюка, М.М. Півеня та Г.В. Горячева.

Запрошені експерти здійснили експертне оцінювання комерційного потенціалу нашої розробки за критеріями, наведеними в таблиці 5.1,

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

Продовження таблиці 5.1

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
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 використовуютьс я у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовую ться 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

Запрошені експерти оцінили комерційний потенціал розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах так, як це наведено в таблиці 5.2:

Таблиця 5.2 – Результати оцінювання комерційного потенціалу розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах (Шкала оцінювання «0»-«1»-«2»-«3»-«4»)

Критерії	Ініціали, прізвище експертів		
	М.В.Микитюк	М.М. Півень	Г.В. Горячев
	Бали, що їх виставили експерти:		
1	4	4	3
2	4	3	4
3	4	3	3
4	4	4	3
5	3	3	4
6	4	4	3
7	3	3	4
8	4	3	3
9	3	4	3
10	3	4	4
11	4	3	3
12	3	4	4
Сума балів	$СБ_1 = 43$	$СБ_2 = 42$	$СБ_3 = 41$
Середньоарифметична сума балів $\overline{СБ}$	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{3} = \frac{43+42+41}{3} = \frac{126}{3} = 42,00$		

Для встановлення комерційного потенціалу розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах звернемося до рекомендацій, які наведено в таблиці 5.3 [35].

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 42,00 балів (із максимально можливих 48-ми балів), то це свідчить, що розроблена нами система автоматизації процесу аутентифікації та управління доступом у банківських системах має рівень комерційного потенціалу, який можна вважати «високим».

Це пояснюється тим, що внесені в нашу розробку інноваційні рішення відкривають нові можливості для розвитку безпекових систем у фінансовій сфері, а удосконалення і розширення функціональності існуючих методів авторизації та аутентифікації за допомогою REST API та Spring Security стануть ключовими чинниками для подальшого підвищення безпеки та відновлення довіри до банківських додатків серед користувачів.

5.2 Розрахунок витрат на розроблення системи автоматизації процесу аутентифікації та управління доступом у банківських системах

При розробленні системи автоматизації процесу аутентифікації та управління доступом у банківських системах були зроблені такі витрати:

А). Основна заробітна плата $З_о$ розробників (дослідників тощо), яка визначається за формулою:

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

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

Для 2024 року приймемо, що:

М = (8000…26000) грн/місяць;

$Т_р$ – число робочих днів в місяці; приймемо $Т_р$ = 23 дні;

t – число днів роботи розробників (дослідників тощо).

Зроблені розрахунки основної заробітної плати розробників (дослідників тощо) зведемо до таблиці 5.4.

Таблиця 5.4 – Основна заробітна плата розробників (дослідників тощо)

Найменування посади виконавця	Місячний посадовий оклад, грн	Оплата за робочий день, грн	Число днів (годин) роботи	Витрати на оплату праці, грн
1. Науковий керівник магістерської роботи	23600	≈ 1026	20 годин	(1026 / 6) × 20 = ≈ 3420 (при 6-годинному робочому дні)
2. Здобувач-магістрант (виконавець)	2000 грн (беремо 8000 грн)	≈347,82	82 дні	28521,24 ≈ 28522
3. Консультант з бухгалтерії ВНТУ	30000	1304,34	1 година	1304,34 / 8 ≈ 163 грн (при 8-годинному робочому дні)
4. Консультант з економічної частини	19200	834,78 ≈ 835	1,5 години	(835 /6) × 1,5 ≈ 209 грн (при 6-годинному робочому дні)
Загалом				$З_о$ = 32314 грн

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

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

Приймемо, що α = 0,1. Тоді для нашого випадку отримаємо:

$$З_д = 0,1 \times 32314 = 3231,4 \approx 3232 \text{ грн.}$$

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:

$НЗН_{зп} = (32314 + 3232) \times 0,22 = 7820,12 \approx 7821$ 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	60000	25	2,8 (при 75% використанні)	2625
2. Приміщення університету, факультету, кафедри	16000	5	2,8 (при 50% використанні)	93,33 ≈ 94
Всього				А = 2719 грн

Д). Витрати на матеріали М розраховуються за формулою:

$$М = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i - \sum_{1}^{n} В_i \cdot Ц_в \quad \text{грн,} \qquad (5.5)$$

де $Н_i$ – витрати матеріалу *i*-го найменування, кг; $Ц_i$ – вартість матеріалу *i*-го най-менування; $К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15); $В_i$ – маса відходів матеріалу *i*-го найменування; $Ц_в$ – ціна відходів матеріалу *i*-го найменування; n – кількість видів матеріалів.

Е). Витрати на комплектуючі К розраховуються за формулою:

$$К = \sum_{1}^{n} Н_i \cdot Ц_i \cdot К_i \quad \text{грн,} \qquad (5.6)$$

де $Н_i$ – кількість комплектуючих *i*-го виду, шт.; $Ц_i$ – ціна комплектуючих *i*-го виду; $К_i$ – коефіцієнт транспортних витрат, $К_i$ = (1,1…1,15); n – кількість видів комплектуючих.

Під час виконання магістерської кваліфікаційної роботи загальні витрати на матеріали та комплектуючі склали укрупнено приблизно 725 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 = 1,28 кВт;

Ф – фактична кількість годин роботи обладнання, годин.

Приймемо, що Ф = 235 годин;

$К_п$ – коефіцієнт використання потужності; $К_п$ < 1 = 0,80.

$К_д$ – коефіцієнт корисної дії, $К_д$ = 0,75.

Тоді витрати на силову електроенергію будуть дорівнювати:

$$В_е = \frac{В \cdot П \cdot Ф \cdot К_п}{К_д} = \frac{4,8 \cdot 1,28 \cdot 235 \cdot 0,8}{0,75} = 1540,09 \approx 1540$$ грн.

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

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

Для нашого випадку отримаємо:

$$В_{інш} = 0,5 \times 32314 = 16157 \text{ грн.}$$

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

$$В = 32314 + 3232 + 7821 + 2719 + 725 + 1540 + 16157 = 64508 \text{ грн.}$$

Л). Загальні витрати на розробку та можливу комерціалізацію розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах розраховуються за формулою:

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

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

Оскільки наша розробка на цей момент часу практично готова до впровадження, то можна умовно прийняти, що, $\beta \approx 0,95$ [1].

Тоді: $B_{заг} = \frac{64508}{0,95} = 67903,15$ грн або приблизно 68 тисяч грн.

Тобто прогнозовані загальні витрати на розробку та можливу комерціалізацію розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах становлять приблизно 68 тисяч грн.

5.3 Розрахунок економічного ефекту від можливої комерціалізації розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах

Проведене дослідження ринку банківських послуг показало, що розроблена нами система автоматизації процесу аутентифікації та управління доступом у банківських системах дозволить значно підвищити ефективність роботи банківського додатку, зменшить ризики помилок і забезпечить надійний захист банківської інформації.

Аналіз місткості ринку аналогічних систем також показує, що на сьогодні в Україні кількість реальних користувачів подібних автоматизованих систем може становити приблизно (70…100) осіб щороку (ІТ відділи банків та інш.). Для початку приймемо величину у 85 осіб. 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 (оскільки вона практично готова до впровадження), 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 + 5 шт. (відносно базового року);

б) 2026 р. – +10 шт. (відносно базового року);

в) 2027 р. – +15 шт. (відносно базового року).

Економічний ефект від можливої комерціалізації розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах пояснюється її значно кращими функціональними можливостями. На сьогодні, аналіз ринку показує, що можлива вартість подібних автоматизованих систем коливається в діапазоні $2000 або приблизно 100 тисяч грн.

А оскільки розроблена нами система автоматизації процесу аутентифікації та управління доступом у банківських системах має значно кращі функціональні можливості, то її можна буде реалізовувати на ринку дещо дорожче, ніж аналогічні за функціями розробки, наприклад, реалізовувати середньому за 105 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 $\Delta Ц_o$ = 105 - 100 = + 5 тисяч грн;

N – основний кількісний показник, який визначає обсяг діяльності у році до впровадження результатів розробки; N = 85 шт.;

ΔN – покращення основного кількісного показника від впровадження результатів розробки. Таке покращення основного кількісного показника становитиме по роках, у 2025 році – + 5 шт., у 2026 році + 10 шт., у 2027 році +15 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; $Ц_o$= 105 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:

$$\Delta П_1 = [5 \cdot 85 + 105 \cdot 5] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) = 324{,}57 \approx 325$$ тисяч грн.

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

$$\Delta П_2 = [5 \cdot 85 + 105 \cdot 10] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) = 503{,}94 \approx 504$$ тисяч грн.

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

$$\Delta П_3 = [5 \cdot 85 + 105 \cdot 15] \cdot 0{,}8333 \cdot 0{,}5 \cdot (1 - \frac{18}{100}) = 683{,}31 \approx 684$$ тис. грн.

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

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

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. Приймемо $\tau = 0{,}10$ (10%);

t – період часу від моменту початку розроблення системи автоматизації процесу аутентифікації та управління доступом у банківських системах 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:

$$ПП = \frac{325}{(1+0{,}1)^2} + \frac{504}{(1+0{,}1)^3} + \frac{684}{(1+0{,}1)^4} \approx 269 + 379 + 468 = 1116 \text{ тисяч грн.}$$

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

Для нашого випадку приймемо, що:

$$PV = (1{,}0 \ldots 5{,}0) \times 68 = 2{,}5 \times 68 = 170 \text{ тисяч грн.}$$

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

$$Е_{абс} = ПП - PV, \quad (5.12)$$

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

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

$$E_{абс} = 1116 – 170 = 946 \text{ тисяч грн.}$$

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

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

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

де $E_{абс}$ – абсолютний ефект вкладених інвестицій; $E_{абс} = 946$ тисяч грн;

PV –теперішня вартість початкових інвестицій PV = 170 тисяч грн;

$T_ж$ – життєвий цикл розробки, роки.

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

Для нашого випадку отримаємо:

$$E_в = \sqrt[4]{1 + \frac{946}{170}} - 1 = \sqrt[4]{1 + 5,5647} - 1 = \sqrt[4]{6,5647} - 1 = 1,60 - 1 = 0,60 \approx 60,0\%.$$

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

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

$$\tau_{мін} = d + f, \quad (5.14)$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2024 році в Україні d = (0,10...0,12), а окремі банки встановлюють ставку за депозитними операціями на рівні до 18%. Приймемо, що $\tau = 18\%$.

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

Приймемо, що f = 30%, тобто f = 0,3.

Тоді для нашого випадку отримаємо:

$$\tau_{мін} = 0,18 + 0,30 = 0,48 \text{ або } \tau_{мін} = 48\%.$$

Оскільки величина $E_в = 60,0\% > \tau_{мін} = 48\%$, то потенційний інвестор у принципі може бути зацікавлений у комерціалізації розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах.

Далі розраховуємо термін окупності коштів, вкладених у можливу комерціалізацію розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах.

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

$$T_{ок} = \frac{1}{E_в} = \frac{1}{0,60} = 1,67 \text{ років} < 3 \text{ років} \qquad (5.15)$$

що свідчить про потенційну доцільність комерціалізації розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах.

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

Прийнявши рівень інфляції у 20%, отримаємо:

$$ПП = \frac{325}{(1+0,2)^2} + \frac{504}{(1+0,2)^3} + \frac{684}{(1+0,2)^4} \approx 226 + 292 + 330 = 848 \text{ тисяч грн.}$$

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

$$Е_{абс} = 848 – 170 = 678 \text{ тисяч грн.}$$

Внутрішня дохідність $Е_в$ вкладених інвестицій становитиме:

$$Е_в = \sqrt[4]{1+\frac{678}{170}} - 1 = \sqrt[4]{1+3,9882} - 1 = \sqrt[4]{4,9882} - 1 = 1,495 - 1 = 0,495 \approx 49,5\%.$$

Оскільки величина $Е_в = 49,5\% > \tau_{мін} = 48\%$, то потенційний інвестор у принципі може бути зацікавлений у комерціалізації розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах.

Прийнявши рівень інфляції у 30%, отримаємо:

$$ПП = \frac{325}{(1+0,3)^2} + \frac{504}{(1+0,3)^3} + \frac{684}{(1+0,3)^4} \approx 192 + 229 + 239 = 660 \text{ тисяч грн.}$$

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

$Е_{абс} = 660 – 170 = 490$ тисяч грн.

Внутрішня дохідність $Е_в$ вкладених інвестицій становитиме:

$$Е_в = \sqrt[4]{1+\frac{490}{170}} - 1 = \sqrt[4]{1+2,8823} - 1 = \sqrt[4]{3,8823} - 1 = 1,40 - 1 = 0,40 \approx 40,0\%.$$

Оскільки величина $E_в = 40,0\% < \tau_{мін} = 48\%$, то потенційний інвестор у принципі може бути не зацікавлений у комерціалізації нашої розробки. Тому для при-йняття остаточного рішення потрібно провести додаткові обгрунтування і розрахунки (наприклад, знизити рівень прийнятого ризику, підняти ціну реалізації нашої розробки, збільшити попит на нашу розробку тощо).

Зроблені розрахунки у вигляді графіків наведено на рис. 5.1.

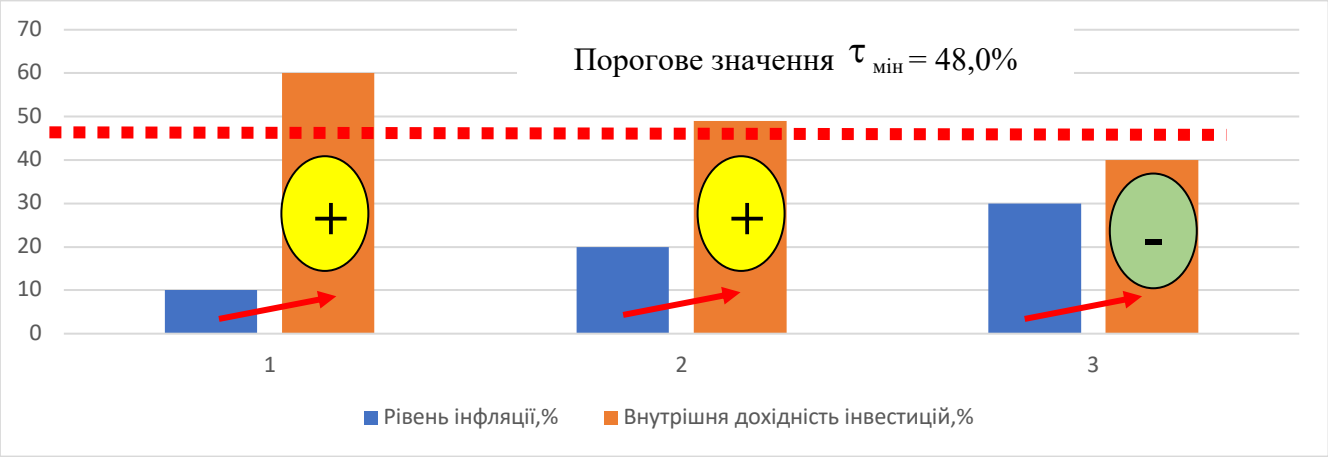

Рисунок 5.1 – Моделювання залежності величини внутрішньої дохідності потенційних інвестицій від рівня інфляції в країні

Результати виконаної економічної частини магістерської кваліфікаційної роботи зведено у таблицю (таблиця 5.6):

Таблиця 5.6 - Результати виконаної економічної частини магістерської кваліфікаційної роботи

Показники	Задані у ТЗ	Досягнуті у магістерській кваліфікаційній роботі	Висновок
1	2	3	4
1. Витрати на розробку	Не більше 70 тисяч грн	68 тисяч грн	Досягнуто
2. Абсолютний ефект від впровадження розробки, тисяч грн	Не менше 900 тисяч грн (за три роки)	946 тисячі грн (при 10% інфляції)	Виконано

Продовження таблиці 5.6

1	2	3	4
3. Внутрішня дохідність інвестицій, %	не менше 48,0%	60,0%	Виконано
4. Термін окупності інвестицій, роки	до 3-ти років	1,67 років	Виконано

Таким чином, основні техніко-економічні показники розробленої нами системи автоматизації процесу аутентифікації та управління доступом у банківських системах, визначені у технічному завданні, повністю виконані.

ВИСНОВКИ

У ході виконання роботи досягнуто поставлену мету — створення серверної частини банківського додатку, яка забезпечує високу продуктивність і безпеку завдяки використанню RESTful інтерфейсу в поєднанні з механізмами аутентифікації та авторизації. Розроблений додаток відповідає сучасним вимогам фінансового сектора, забезпечуючи безпечну та ефективну взаємодію з клієнтською частиною.

У процесі розробки було виконано аналіз предметної області, досліджено актуальні методи аутентифікації та авторизації, серед яких застосовано Spring Security для аутентифікації та використання JWT-токенів для захисту обміну даними. Обґрунтовано вибір архітектури REST API для забезпечення масштабованості та стандартизації взаємодії клієнт-сервер.

Проєктування структури бази даних виконано з використанням MongoDB, що дозволило забезпечити гнучкість у роботі з великими обсягами даних. Базовий модуль бізнес-логіки реалізовано за допомогою сервісів і інтерфейсів, що забезпечує модульність та полегшує майбутню підтримку і розширення системи. Особливу увагу приділено валідації вхідних запитів та відповідей, що реалізовано через генерацію сигнатур із використанням алгоритму MD5 і переведенням у формат Base64. Це дозволяє додатково захищати передані дані від підробок і несанкціонованих змін.

Також умову про витрати на розробку(не більше 70 тисяч гривень) було досягнуто, у розмірі 68 тисяч гривень.

Таким чином, розроблена система повністю відповідає поставленій меті, пропонуючи ефективне, безпечне та масштабоване рішення для банківських додатків.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. RESTful API Design Best URL: https://dzone.com/articles/restful-api-design-best-practices (дата звернення: 27.11.2024)

2. Spring Security URL: https://docs.spring.io/spring-security/site/docs/current/reference/html5/(дата звернення: 26.11.2024)

3. Ткачук О.В., Богач І.В. ВИКОРИСТАННЯ МОБІЛЬНИХ ДОДАТКІВ-АУТЕНТИФІКАТОРІВ ДЛЯ АВТОРИЗАЦІЇ У БАНКІВСЬКІ ДОДАТКИ // Матеріали LIII Всеукраїнської науково-технічної конференції факультету інтелектуальних інформаційних технологій та автоматизації (2024): збірник матеріалів. – Вінниця: ВНТУ, 2023. - URL: https://conferences.vntu.edu.ua/index.php/all-fksa/all-fksa-2024/paper/view/20647

4. Авторизація та автентикація URL: https://www.okta.com/identity-101/authentication-vs-authorization/(дата звернення: 20.11.2024)

5. Stanislav, M. Two-Factor Authentication. Packt Publishing, 2015. 134 c.

6. What Is Two-Factor Authentication (2FA)? How It Works and Example URL: https://www.investopedia.com/terms/t/twofactor-authentication-2fa.asp(дата звернення: 25.11.2024)

7. Biometrics URL: https://www.investopedia.com/terms/b/biometrics.asp#toc-what-is-biometrics (дата звернення: 27.11.2024)

8. David Yambay, Raghavendra Ramachandra, Sébastien Marcel: Handbook of Biometric Anti-Spoofing: Trusted Biometrics under Spoofing Attacks 2014. 273c.

9. One Time Password (OTP, TOTP) : definition, examples URL: https://www.thalesgroup.com/en/markets/digital-identity-and-security/technology/otp(дата звернення: 27.11.2024)

10. What is a Time-based One-time Password (TOTP) URL: https://www.twilio.com/docs/glossary/totp(дата звернення: 27.11.2024)

11. What Is an HMAC-Based One-Time Password (HOTP)? How it Works URL: https://www.1kosmos.com/security-glossary/hmac-based-one-time-password-hotp/(дата звернення: 27.11.2024)

12. Fondo-Ferreiro, P., Gil-Castiñeira, F., González-Castaño, F. J., Candal-Ventureira, D. A Software-Defined Networking Solution for Interconnecting Network Functions in Service-Based Architectures. 2024. 162 c.

13. Porcello, E., Banks, A. Learning GraphQL: Declarative Data Fetching for Modern Web Apps. O'Reilly Media, 2018. 300 c.

14. Robert Englander. Java and SOAP 2002. 276c.

15. Kathy Sierra, Bert Bates, Trisha Gee. Head First Java: A Brain-Friendly Guide 3rd Edition 2022. 752c.

16. Spring Boot URL: https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/ (дата звернення: 26.11.2024)

17. CSRF (Cross Site Request Forgery) https://book.hacktricks.xyz/pentesting-web/csrf-cross-site-request-forgery (дата звернення: 27.11.2024)

18. XSS (Cross Site Scripting) https://book.hacktricks.xyz/pentesting-web/xss-cross-site-scripting (дата звернення: 27.11.2024)

19. Hibernate URL: https://hibernate.org/orm/documentation/ (дата звернення: 27.11.2024)

20. Deuker, C. Shadowed. Houghton Mifflin Harcourt, 2024. 320 c.

21. IDE URL: https://www.codecademy.com/article/what-is-an-ide (дата звернення: 27.11.2024)

22. Git URL: https://git-scm.com/doc (дата звернення: 27.11.2024)

23. Inteliji Idea: https://www.jetbrains.com/idea/ (дата звернення: 27.11.2024)

24. Maven URL: https://maven.apache.org/guides/index.html (дата звернення: 27.11.2024)

25. SLF4J URL: https://www.slf4j.org/manual.html(дата звернення: 27.11.2024)

26. Introduction to MongoDB URL: https://www.mongodb.com/docs/manual/introduction/ (дата звернення: 12.06.2023)

27. MVC URL: https://docs.phalcon.io/4.0/en/mvc (дата звернення: 25.11.2023)

28. Dependency Inversion URL: https://stackify.com/dependency-inversion-principle/(дата звернення: 27.11.2023)

29. Repository URL: https://deviq.com/design-patterns/repository-pattern (дата звернення: 27.11.2023)

30. Security Filter URL: https://docs.spring.io/spring-security/site/docs/3.0.x/reference/security-filter-chain.html (дата звернення: 27.11.2023)

31. JWT URL: https://www.baeldung.com/spring-security-oauth-jwt (дата звернення: 27.11.2023)

32. Russell, J., Cohn, R. Md5. Book on Demand, 2020. 140 с.

33. Postman Tutorial – How to use for API Testing? URL: https://www.guru99.com/postman-tutorial.html(дата звернення: 27.11.2023)

34. Catalin Tudose. JUnit in Action, Third Edition 2020

35. 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.012.02.000 ТЗ

Керівник роботи
к.т.н., доц. каф. АІІТ
Ілона БОГАЧ
«_» _____2024 р.

Виконавець:
ст. гр. 1АКІТР-23М
Олексій ТКАЧУК
«__» _____2024 р.

Вінниця ВНТУ 2024

1. Назва та галузь застосування

Магістерська кваліфікаційна робота: «Автоматизація процесу аутентифікації та управління доступом у банківських системах». Галузь застосування – інформаційні технології.

2. Підстава для розробки

Розробку системи здійснювати на підставі наказу по університету№ 247 від ___ 2023 та завдання до магістерської кваліфікаційної роботи, складеного та затвердженого кафедрою «Автоматизації та інтелектуальних інформаційних технологій»

3. Мета та призначення розробки

Метою роботи є покращення продуктивності та безпеки банківських додатків.

4. Джерела розробки

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, 2023. – 26 с.

2. John K. Ousterhout, A Philosophy of Software Design (2nd Edition), 2021, 196c.

3. Spring Security URL: https://spring.io/projects/spring-security(дата звернення: 27.11.2023)

4. Refactoring Guru URL: https://refactoring.guru/uk/design-patterns (дата звернення: 27.11.2023)

5. Показники призначення

Основні технічні вимоги та мінімальні системні вимоги до програми:

- Серверна платформа: Spring Boot 2.x або вище.Процесор: Core 2 Duo
- База даних: MongoDB 4.x або вище.
- Інструменти: Java 11+ (JDK), Maven або Gradle для управління залежностями.
- Інтеграційні технології: RESTful API, JWT для аутентифікації.
- Хешування: MD5 із конкатенацією секретного ключа для генерації сигнатур.

Результати роботи програми:

- реєстрація та авторизація користувачів;

- збереження інформації в базі даних;
- обробка транзакцій;
- відкриття банківських карток;

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» має бути виконаний до _____..
2. Розділ 2 «Технології та інструменти розробки» має бути виконаний до _____..
3. Розділ 3 «Проектування та реалізація системи» має бути виконаний до _____.
4. Розділ 4 «Тестування додатку» має бути виконаний до _____..
5. Економічний розділ має бути виконаний до _____..

8. Порядок контролю та приймання

1. Рубіжний контроль провести до _____.
2. Попередній захист магістерської кваліфікаційної роботи провести до _____.
3. Захист магістерської кваліфікаційної роботи провести в період з _____.до _____..

Додаток Б

(обов’язковий)

ІЛЮСТРАТИВНА ЧАСТИНА

АВТОМАТИЗАЦІЯ ПРОЦЕСУ АУТЕНТИФІКАЦІЇ ТА УПРАВЛІННЯ ДОСТУПОМ У БАНКІВСЬКИХ СИСТЕМАХ

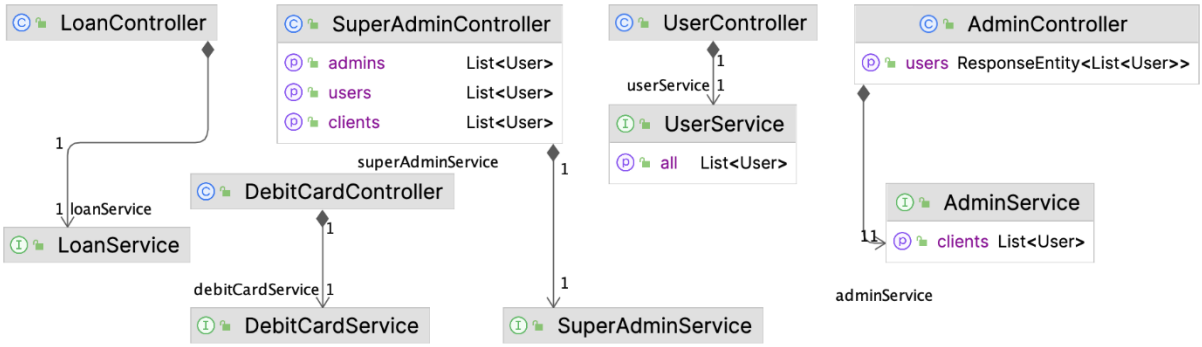

Рисунок Б.1 - Діаграма взаємодії контролерів та сервісів

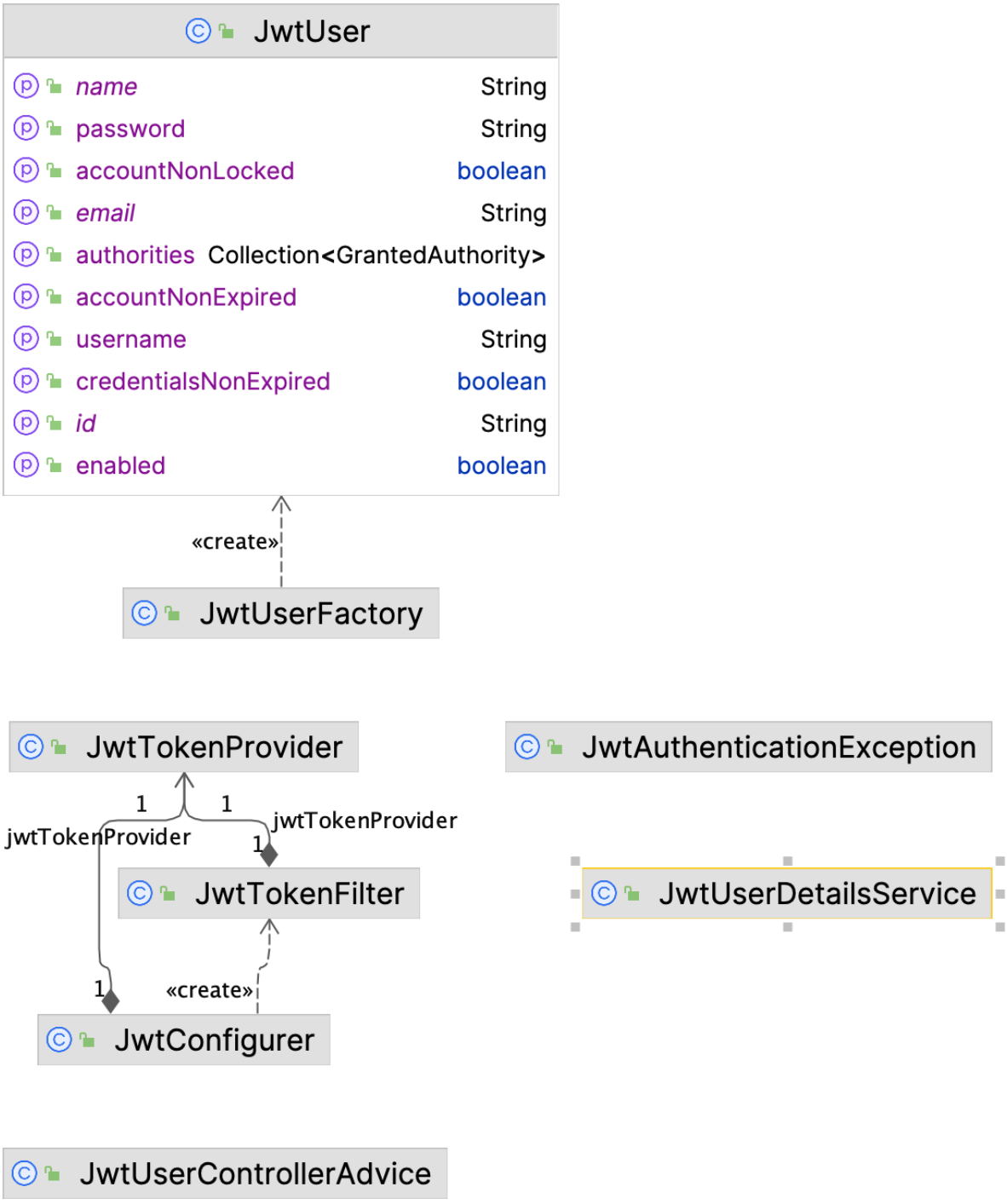

Рисунок Б.2 - Діаграма роботи Spring Security

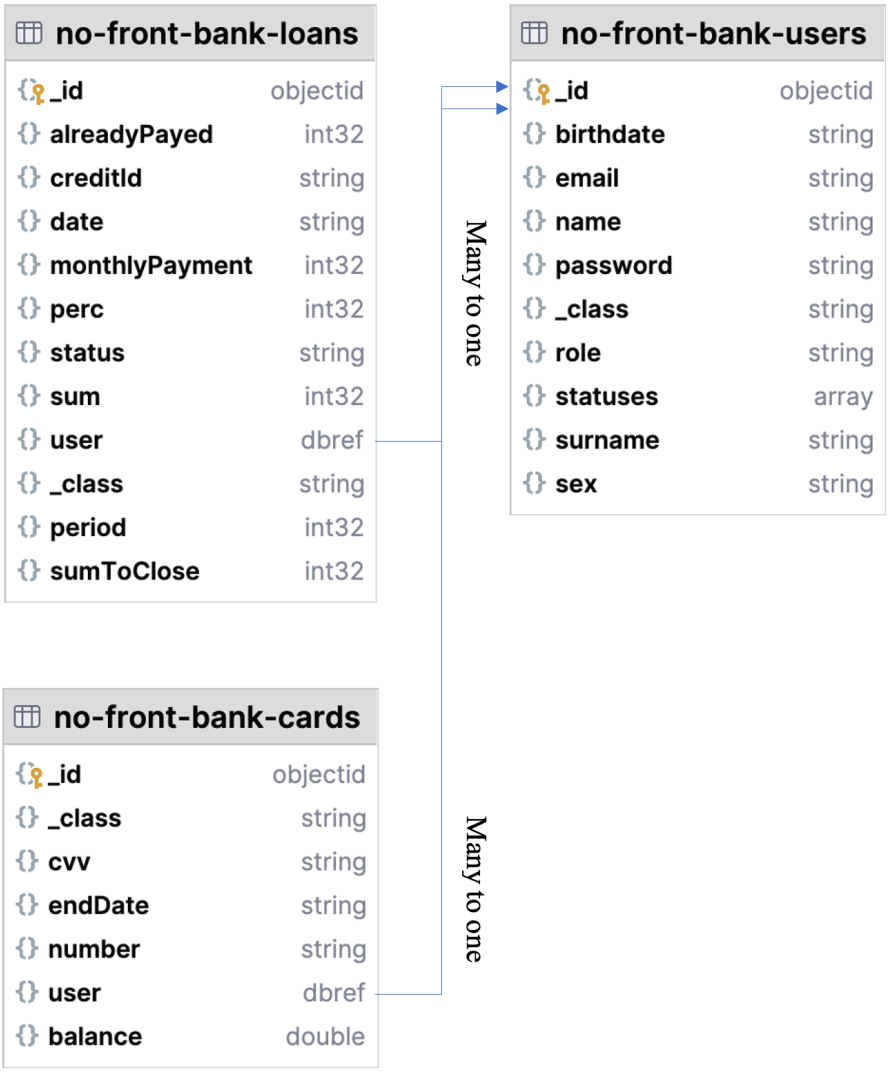

Рисунок Б.3 - Діаграма бази даних (UML)

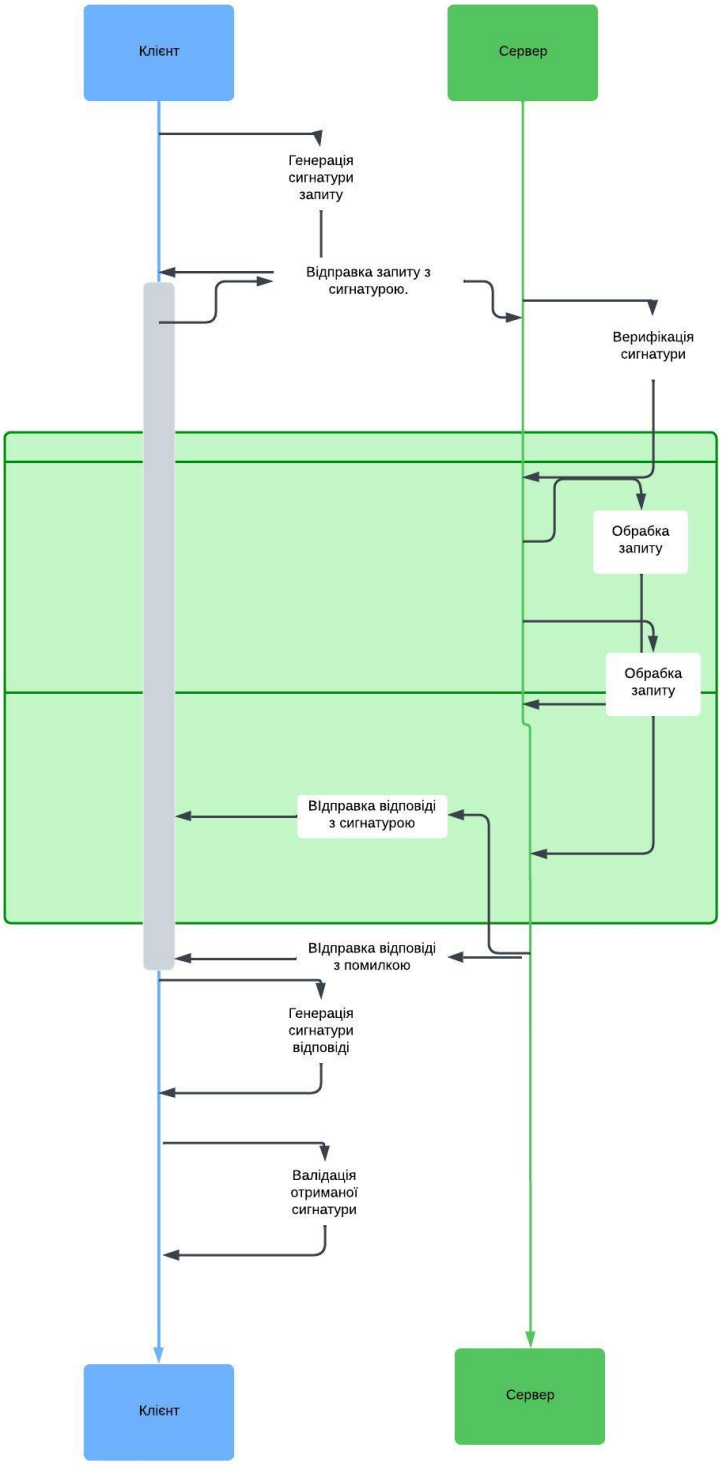

Рисунок Б.4 – Діаграма верифікації запитів та відповідей

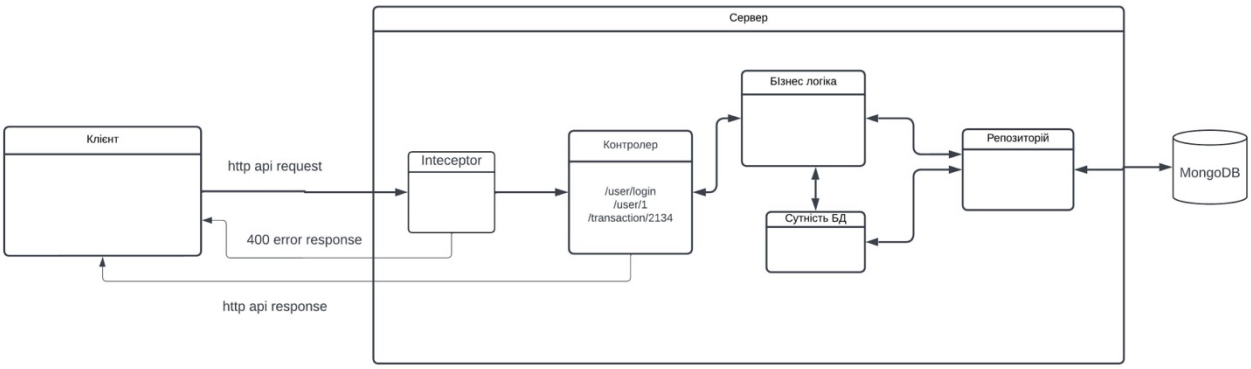

Рисунок Б.5 – Схема взаємодії клієнта з сервером

Продовження додатка Б

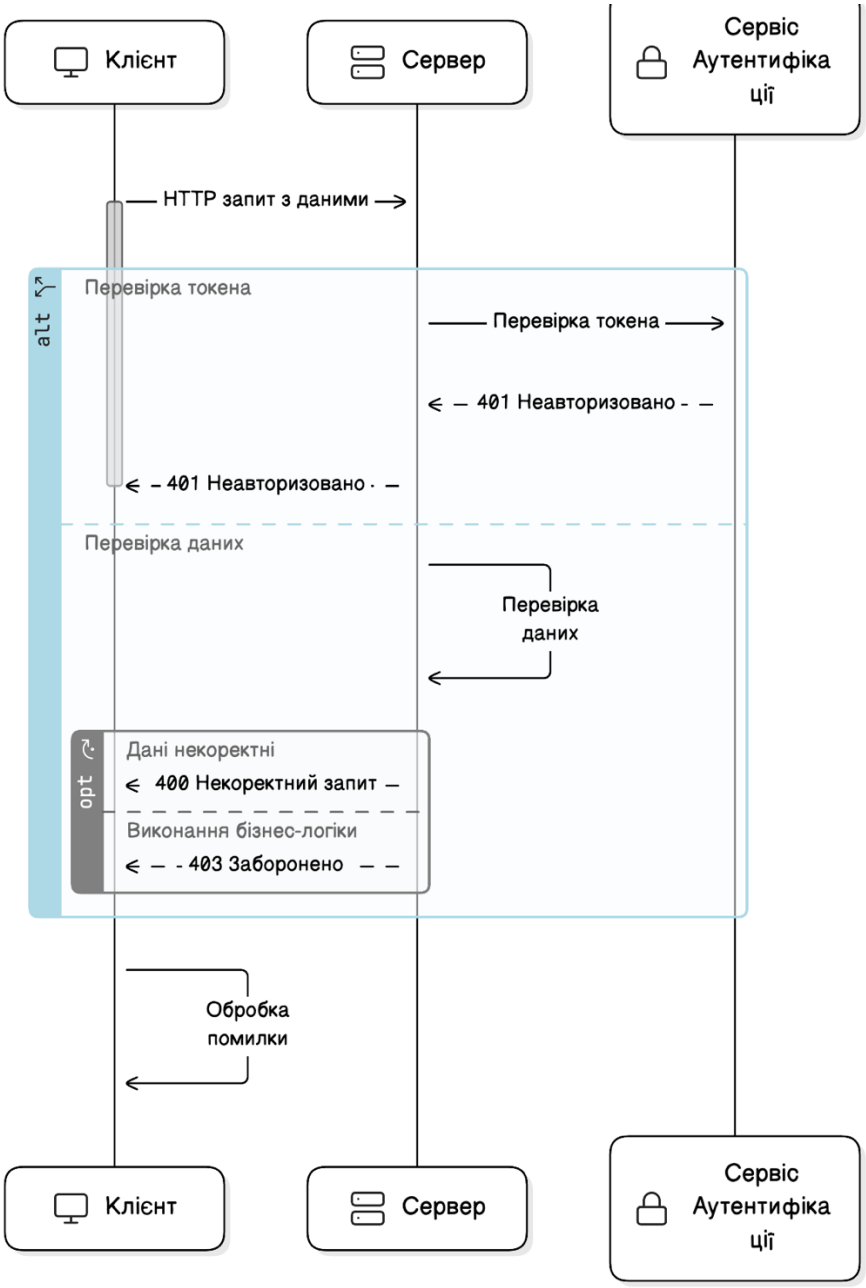

Рисунок Б.6 – Діаграма обробки помилок на сервері

Додаток В (обов'язковий)

Лістинг основних функцій

```
package com.bank.jwtapi.bankjwtapi.service;

import org.apache.commons.codec.digest.DigestUtils;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import org.springframework.web.servlet.HandlerInterceptor;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.Base64;

@Component
public class SignatureVerificationInterceptor implements HandlerInterceptor {

    @Value("${secret.key}")
    private String SECRET_KEY; // Replace with your secret key
    private static final String SIGNATURE_HEADER = "signature";

    @Override
    public boolean preHandle(HttpServletRequest request, HttpServletResponse response, Object handler) throws Exception {
        String receivedSignature = request.getHeader(SIGNATURE_HEADER);
        if (receivedSignature == null || receivedSignature.isEmpty()) {
            response.setStatus(HttpServletResponse.SC_UNAUTHORIZED);
            response.getWriter().write("Missing or empty signature");
            return false;
        }

        String payload = getPayloadFromRequest(request);

        String expectedSignature = generateSignature(payload, SECRET_KEY);

        if (!receivedSignature.equals(expectedSignature)) {
            response.setStatus(HttpServletResponse.SC_BAD_REQUEST);
            response.getWriter().write("Invalid signature");
            return false;
        }

        return true;
    }

    private String getPayloadFromRequest(HttpServletRequest request) {
        try (java.io.BufferedReader reader = request.getReader()) {
            StringBuilder sb = new StringBuilder();
            String line;
            while ((line = reader.readLine()) != null) {
                sb.append(line);
            }
            return sb.toString();
        } catch (Exception e) {
            throw new RuntimeException("Error reading request payload", e);
        }
    }

    private String generateSignature(String payload, String secretKey) {
        String combined = payload + secretKey;
        String md5Hash = DigestUtils.md5Hex(combined);
        return Base64.getEncoder().encodeToString(md5Hash.getBytes());
    }
}
```

BankJwtApiApplication:

```
package com.bank.jwtapi.bankjwtapi;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class BankJwtApiApplication {

    public static void main(String[] args) {
        SpringApplication.run(BankJwtapiApplication.class, args);
    }
}
```

JwtTokenProvider:

```
package com.bank.jwtapi.bankjwtapi.security.jwt;

import com.bank.jwtapi.bankjwtapi.models.Role;
import io.jsonwebtoken.*;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Component;

import javax.servlet.http.HttpServletRequest;
import java.util.Date;

@Component
public class JwtTokenProvider {

    @Value("${application.jwt.secretKey}")
    private String secretKey;

    @Value("${jwt.token.expired}")
    private long tokenExpirationAfterDays;

    @Autowired
    private UserDetailsService userDetailsService;


    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }

    public String createToken(String username, Role role) {

        Claims claims = Jwts.claims().setSubject(username);
        claims.put("roles", getRoleName(role));

        Date now = new Date();
        Date validateDate = new Date(now.getTime() + tokenExpirationAfterDays);

        return Jwts.builder()
                .setClaims(claims)
                .setIssuedAt(now)
                .setExpiration(validateDate)
                .signWith(SignatureAlgorithm.HS256, secretKey)
                .compact();
    }

    public Authentication getAuthentication(String token) {
        UserDetails userDetails = this.userDetailsService.loadUserByUsername(getUsername(token));
        return new UsernamePasswordAuthenticationToken(userDetails, "", userDetails.getAuthorities());
    }

    public String getUsername(String token) {
        return Jwts.parser()
                .setSigningKey(secretKey)
                .parseClaimsJws(token)
                .getBody()
                .getSubject();
    }

    public String resolveToken(HttpServletRequest request) {
        String bearerToken = request.getHeader("Authorization");
        if (bearerToken != null && bearerToken.startsWith("Bearer_"))
            return bearerToken.substring(7, bearerToken.length());
        return null;
    }

    public boolean validateToken(String token) {
            Jws<Claims> claims = Jwts.parser().setSigningKey(secretKey).parseClaimsJws(token);
        return !claims.getBody().getExpiration().before(new Date());
    }
```

```
    private String getRoleName(Role userRole) {
        return userRole.name();
    }
}
```

SecurityBankConfig:

```
package com.bank.jwtapi.bankjwtapi.config;

import com.bank.jwtapi.bankjwtapi.security.JwtUserDetailsService;
import com.bank.jwtapi.bankjwtapi.security.jwt.JwtConfigurer;
import com.bank.jwtapi.bankjwtapi.security.jwt.JwtTokenProvider;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.crypto.password.PasswordEncoder;

import static com.bank.jwtapi.bankjwtapi.models.Role.*;

@Configuration
public class SecurityBankConfig extends WebSecurityConfigurerAdapter {

    private final JwtTokenProvider jwtTokenProvider;
    private final PasswordEncoder passwordEncoder;
    private final JwtUserDetailsService userDetailsService;

    @Autowired
    public SecurityBankConfig(JwtTokenProvider jwtTokenProvider, PasswordEncoder passwordEncoder, JwtUserDetailsService
userDetailsService) {
        this.jwtTokenProvider = jwtTokenProvider;
        this.passwordEncoder = passwordEncoder;
        this.userDetailsService = userDetailsService;
    }

    @Bean
    @Override
    public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManagerBean();
    }

    @Override
    public void configure(HttpSecurity http) throws Exception {
        http
                .httpBasic().disable()
                .csrf().disable()
                .sessionManagement()
                    .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
                .and()
                .authorizeRequests()
                .antMatchers("/user/register", "/user/login", "/pay-item/**")
                    .permitAll()
                .antMatchers("/admin/*")
                    .hasRole(String.valueOf(ADMIN))
                .antMatchers("/super_admin/*")
                    .hasRole(String.valueOf(SUPERADMIN))
                .anyRequest().authenticated()
                .and()
                .apply(new JwtConfigurer(jwtTokenProvider));

    }
}
```

JwtUser:

```
package com.bank.jwtapi.bankjwtapi.security.jwt;

import com.fasterxml.jackson.annotation.JsonIgnore;
import lombok.Data;
import org.springframework.security.core.GrantedAuthority;
import org.springframework.security.core.userdetails.UserDetails;
import java.util.Collection;
@Data
public class JwtUser implements UserDetails {

    private final String id;
    private final String password;
```

```
    private final String email;
    private final String name;
    private final boolean enabled;
    private final Collection<? extends GrantedAuthority> authorities;

    public JwtUser(String id, String password, String email, String name, boolean enabled, Collection<? extends GrantedAuthority> authorities) {
        this.id = id;
        this.password = password;
        this.email = email;
        this.name = name;
        this.enabled = enabled;
        this.authorities = authorities;
    }

    @Override
    public Collection<? extends GrantedAuthority> getAuthorities() {
        return authorities;
    }

    @JsonIgnore
    @Override
    public String getPassword() {
        return password;
    }

    @Override
    public String getUsername() {
        return email;
    }

    @JsonIgnore
    @Override
    public boolean isAccountNonExpired() {
        return true;
    }

    @JsonIgnore
    @Override
    public boolean isAccountNonLocked() {
        return true;
    }

    @JsonIgnore
    @Override
    public boolean isCredentialsNonExpired() {
        return true;
    }

    @JsonIgnore
    @Override
    public boolean isEnabled() {
        return enabled;
    }
}
```

UserServiceImpl:

```
package com.bank.jwtapi.bankjwtapi.service.impl;

import com.bank.jwtapi.bankjwtapi.dto.AuthenticationRequestDto;
import com.bank.jwtapi.bankjwtapi.dto.UserDto;
import com.bank.jwtapi.bankjwtapi.exceptions.UserNotFoundException;
import com.bank.jwtapi.bankjwtapi.exceptions.UserNotRelatedException;
import com.bank.jwtapi.bankjwtapi.models.Role;
import com.bank.jwtapi.bankjwtapi.models.Status;
import com.bank.jwtapi.bankjwtapi.models.User;
import com.bank.jwtapi.bankjwtapi.repos.UserRepository;
import com.bank.jwtapi.bankjwtapi.security.jwt.JwtTokenProvider;
import com.bank.jwtapi.bankjwtapi.service.UserService;
import lombok.RequiredArgsConstructor;
import org.springframework.http.ResponseEntity;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.BadCredentialsException;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.AuthenticationException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import java.util.ArrayList;
```

```
import java.util.HashMap;
import java.util.List;
import java.util.Map;

@Service
@RequiredArgsConstructor
public class UserServiceImpl implements UserService {

    private final UserRepository userRepository;
    private final BCryptPasswordEncoder passwordEncoder;
    private final AuthenticationManager authenticationManager;
    private final JwtTokenProvider jwtTokenProvider;

    @Override
    public void register(UserDto userDto) {
        List<Status> userStatuses = new ArrayList<>();
        userStatuses.add(Status.ACTIVE);
        userRepository.save(User.builder()
                    .name(userDto.getName() != null ? userDto.getName(): "UserName")
                    .surname(userDto.getSurname() != null ? userDto.getSurname(): "UserSurname")
                    .birthdate(userDto.getBirthdate())
                    .sex(userDto.getSex())
                    .email(userDto.getEmail())
                    .password(passwordEncoder.encode(userDto.getPassword()))
                    .role(Role.CLIENT)
                    .statuses(userStatuses)
                .build());
    }

    @Override
    public User getUser(String email) throws UserNotFoundException {
        return userRepository.findByEmail(email).orElseThrow(() -> new UserNotFoundException("User with id " + email + " does not exist"));
    }

    @Override
    public ResponseEntity<Map<String, String>> login(AuthenticationRequestDto authenticationRequestDto) {
        try {
            String username = authenticationRequestDto.getEmail();
            authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(username, authenticationRequestDto.getPassword()));
            User user = getUser(username);

            String token = jwtTokenProvider.createToken(username, user.getRole());

            Map<String, String> response = new HashMap<>();
            response.put("username", username);
            response.put("token", token);

            return ResponseEntity.ok(response);
        } catch (AuthenticationException e) {
            throw new BadCredentialsException("Invalid username or password");
        } catch (UserNotFoundException e) {
            throw new RuntimeException(e);
        }
    }

    @Override
    public User registerAdmin(UserDto userDto) {
        Status userStatus = Status.ACTIVE;
        List<Status> userStatuses = new ArrayList<>();
        userStatuses.add(userStatus);

        return userRepository.save(User.builder()
                .name(userDto.getName() != null ? userDto.getName(): "UserName")
                .surname(userDto.getSurname() != null ? userDto.getSurname(): "UserSurname")
                .birthdate(userDto.getBirthdate())
                .sex(userDto.getSex())
                .email(userDto.getEmail())
                .password(passwordEncoder.encode(userDto.getPassword()))
                .role(Role.ADMIN)
                .statuses(userStatuses)
                .build());
    }

    @Override
    public User findById(String id) throws UserNotFoundException {
        return userRepository.findById(id).orElseThrow(() -> new UserNotFoundException("User with id" + id + "not found"));
    }

    @Override
```

```
    public List<User> getAll() {
        return userRepository.findAll();
    }

    @Override
    public List<User> getAllByRole(Role role) {
        return userRepository.findAllByRole(role);
    }


    @Override
    public void delete(String email, String id) throws UserNotFoundException, UserNotRelatedException {
        User user = getUser(email);
        if (!user.getId().equals(id)){
            throw new UserNotRelatedException("Users do not match");
        }
        userRepository.deleteByEmail(email);
    }

}
```

DebitCardServiceImpl:

```
package com.bank.jwtapi.bankjwtapi.service.impl;

import com.bank.jwtapi.bankjwtapi.dto.DebitCardDto;
import com.bank.jwtapi.bankjwtapi.exceptions.CardNotFoundException;
import com.bank.jwtapi.bankjwtapi.exceptions.UserNotFoundException;
import com.bank.jwtapi.bankjwtapi.models.DebitCard;
import com.bank.jwtapi.bankjwtapi.models.User;
import com.bank.jwtapi.bankjwtapi.repos.DebitCardRepository;
import com.bank.jwtapi.bankjwtapi.repos.UserRepository;
import com.bank.jwtapi.bankjwtapi.service.DebitCardService;
import com.bank.jwtapi.bankjwtapi.service.UserService;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;

import java.time.LocalDateTime;
import java.util.List;
import java.util.Random;

@Service
@AllArgsConstructor
public class DebitCardServiceImpl implements DebitCardService {

    private final DebitCardRepository debitCardRepository;
    private final UserService userService;

    private final UserRepository userRepository;

    @Override
    public void withdrawFromCard(DebitCardDto paymentDto, String userId) throws Exception, CardNotFoundException {
        User user = userService.findById(userId);
        if (paymentDto.getSum() == null){
            throw new Exception();
            //TODO add exception
        }
        DebitCard debitCard = debitCardRepository.findByNumberAndUser(paymentDto.getCardNumber(), user).orElseThrow(() -> new
CardNotFoundException("Card not found"));
        debitCard.setBalance(debitCard.getBalance() - new Double(paymentDto.getSum()));
        debitCardRepository.save(debitCard);
    }

    @Override
    public DebitCard addCard(String userId) throws UserNotFoundException {
        User user = userService.findById(userId);
        System.out.println(user);
        return debitCardRepository.save(DebitCard.builder()
                .number(generateCardNumber())
                .balance(0.0)
                .cvv(generateCVV())
                .endDate(generateEndDate())
                .user(user)
                .build());

    }

    @Override
    public List<DebitCard> userCardsList(String userId) throws UserNotFoundException {
```

```
        return debitCardRepository.findAllByUser(userService.findById(userId));
    }

    @Override
    public DebitCard userCard(String number, String userId) throws UserNotFoundException, CardNotFoundException {
        return debitCardRepository.findByNumberAndUser(number, userService.findById(userId)).orElseThrow(() -> new
CardNotFoundException("Card not found"));
    }

    @Override
    public Double changeBalance(DebitCardDto debitCardDto, String userId) throws UserNotFoundException, CardNotFoundException {

        DebitCard card = debitCardRepository.findByNumberAndUser(debitCardDto.getCardNumber(),
userService.findById(userId)).orElseThrow(() -> new CardNotFoundException("Card not found"));
        double balance = card.getBalance();
        if (debitCardDto.getAction().equals("add"))
            balance += new Double(debitCardDto.getSum());
        else if (debitCardDto.getAction().equals("subtract") || balance - new Double(debitCardDto.getSum()) >= 0)
            balance -= new Double(debitCardDto.getSum());
        card.setBalance(balance);
        debitCardRepository.save(card);
        return balance;
    }

    @Override
    public DebitCard upBalance(String userId, String cardNumber) throws CardNotFoundException {
        DebitCard debitCard = debitCardRepository.findByNumberAndUserId(cardNumber, userId).orElseThrow(() -> new
CardNotFoundException("Card not found"));
        return null;
    }

    private String generateCardNumber(){
        Random random = new Random(System.currentTimeMillis());
        int randomNumberLength = 12;
        StringBuilder builder = new StringBuilder("5435");

        do {
            for (int i = 0; i < randomNumberLength; i++) {
                int digit = random.nextInt(10);
                builder.append(digit);
            }
        }
        while (debitCardRepository.existsByNumber(builder.toString()));
        return builder.toString();
    }

    private String generateEndDate(){
        return LocalDateTime.now().getMonthValue() + "/" + (LocalDateTime.now().getYear() - 2000 + 4);
    }

    private String generateCVV(){
        Random random = new Random(System.currentTimeMillis());
        StringBuilder cvv = new StringBuilder();
        do {
            for (int i = 0; i < 3; i++) {
                cvv.append(random.nextInt(10));
            }

        }
        while (debitCardRepository.existsByCvv(String.valueOf(cvv)));
        return cvv.toString();
    }
}
```

Додаток Г (Обов'язковий)

ПРОТОКОЛ

ПЕРЕВІРКИ КВАЛІФІКАЦІЙНОЇ РОБОТИ НА НАЯВНІСТЬ ТЕКСТОВИХ ЗАПОЗИЧЕНЬ

Назва роботи: Автоматизація процесу аутентифікації та управління доступом у банківських системах

Тип роботи: магістерська кваліфікаційна робота

Підрозділ: кафедра Автоматизації та інтелектуальних інформаційних технологій, факультет інтелектуальних інформаційних технологій та автоматизації

Показники звіту подібності Turnitin

Оригінальність 97% Схожість 3%

Аналіз звіту подібності (відмітити потрібне):
☑ 1. Запозичення, виявлені у роботі, оформлені коректно і не містять ознак плагіату.
☐ 2. Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності роботи і відсутності самостійності її виконання автором. Роботу направити на розгляд експертної комісії кафедри.
☐ 3. 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 _____ Маслій Р.В.
(підпис) (прізвище, ініціали)

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

Автор роботи _____ Ткачук О.О.
(підпис) (прізвище, ініціали)

Керівник роботи _____ Богач І. В.
(підпис) (прізвище, ініціали)