Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА
на тему:
«Веб-додаток мікробанку на основі хмарних технологій»

Виконав: студент 2 курсу, групи 2КІ-20м
напряму підготовки (спеціальності)
123 — «Комп'ютерна інженерія»
_____Свіріпи С.М.

Керівник: д.т.н., проф. каф. ОТ

_____ Азаров О.Д.

«____» _____ 2021 р.

Опонент: д.т.н., професор, голова
секції каф МБІС

_____ Яремчук Ю.Є.

«____» _____ 2021 р.

Допущено до захисту
Завідувач кафедри ОТ
д.т.н., проф. Азаров О.Д.
«____» _____ 2021 р.

Вінниця ВНТУ - 2021

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки
Рівень вищої освіти ІІ-й (магістерський)
Галузь знань 12 — Інформаційні технології
Спеціальність 123 — «Комп'ютерна інженерія»
Освітня програма — «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри
обчислювальної техніки
_____ проф., д.т.н. О.Д. Азаров

«___»_____2021 р.

З А В Д А Н Н Я

НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

Свіріпі Святославу Михайловичу

1 Тема роботи «Веб-додаток мікробанку на основі хмарних технологій» керівник роботи Азаров Олексій Дмитрович, д.т.н., професор, затверджені наказом вищого навчального закладу від 24.09.2021 р. №227

2 Строк подання студентом роботи 15.12.2021 р.

3 Вихідні дані — керування даними користувача в мікросервісному середовищі, засоби — середовище програмування IntelleJ IDEA, мови програмування Java, JavaScript, фреймворк Spring та юібліотеки Junit, Hibernate, React, Redux.

4 Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити): вступ, огляд і аналіз існуючих способів та засобів створення програмного забезпечення для мікробанку на основі хмарних технологій, розробка архітектури мікробанку, розробка програмних засобів мікробанку, розрахунок економічної доцільності створення програми мікробанку на основі хмарних технологій, висновки, перелік посилань

6 Консультанти розділів роботи представлені в таблиці 1.

Таблиця 1 — Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1,2,3	Черняк О.І., к. т. н., доцент		
4	Лесько О. Й., к.е.н., професор		

7 Дата видачі завдання 07.09.2021 р.

8 Календарний план наведено в таблиці 2.

Таблиця 2 — Календарний план

№ з/п	Назва етапів виконання магістерської роботи	Строк виконання етапів роботи	Примітка
1.	Інформаційний пошук та огляд літературних джерел	10.07.2021	виконано
2.	Програмна реалізація	14.07.2021	виконано
3.	Підготовка матеріалів пояснювальної записки	17.08.2021	виконано
4.	Перевірка якості оформлення МКР	19.09.2021	виконано
5.	Оформлення економічної частини	20.10.2021	виконано
6.	Оформлення пояснювальної записки і презентації	21.09.2021	виконано
7.	Попередній захист	20.11.2021	виконано

Студент _____ Свіріпа С.М.
(підпис) (прізвище та ініціали)

Керівник магістерської дипломної роботи _____ Азаров О.Д.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

УДК 004.42

Свіріпа С.М. Веб-додаток мікробанку на основі хмарних технологій. Магістерська кваліфікаційна робота зі спеціальності 123 – комп'ютерна інженерія, освітня програма – комп'ютерна інженерія. Вінниця: ВНТУ, 2021. 110с.

На укр.мові. Бібліогр.: 48 назв; рис.: 26 ; табл. 5.

Дана магістерська робота присвячена розробці мікробанку на основі хмарних технологій. В роботі був проведений аналіз відомих технологій та методів створення програмного забезпечення, і було запропоновано розробити архітектуру з використанням мікросервісів та створити 2 ключових мікросервіси.

У роботі було розроблено мікросервісну архітектуру та виконано методи авторизації та взаємодії користувача з мікробанком та розроблена програма, яка реалізує запропоновані підходи.

В магістерській роботі були також виконані економічні розрахунки по визначенню доцільності нового програмного продукту.

ANNOTATION

UDC 004.42

Sviripa SM Microbank web application based on cloud technologies. Master's thesis in specialty 123 – computer engineering, educational program – computer engineering. Vinnytsia: VNTU, 2021. 110p.

In Ukrainian. Bibliogr .: 48 titles; fig .: 26; table 5.

This master's thesis is dedicated to the development of a microbank based on cloud technologies. The paper analyzes the known technologies and methods of software development, and proposed to develop an architecture using microservices and create 2 key microservices.

The microservice architecture was developed in the work and the methods of user authorization and interaction with the microbank were performed and a program was developed that implements the proposed approaches.

In the master's thesis, economic calculations were also performed to determine the feasibility of a new software product.

ЗМІСТ

ВСТУП

Завданням даної дипломної роботи є розробка мікробанку на основі хмарних технологій, що дає можливість користувачам здійснювати повсякденні платежі та переводи коштів між рахунками банку. Для забезпечення широкого впровадження таких мікробанків потрібно розробити програмне забезпечення з покращеним масштабуванням та можливістю модифікації. Існують різні способи написання такого програмного забезпечення [1-4, 45, 49], а також велика кількість мов програмування [5-8, 19, 27-28, 33, 48], які мають необхідний перелік бібліотек [18, 20-22, 32, 33, 36, 37, 42] для реалізації поставленої задачі. Використання даного програмного забезпечення також потребує задіяння технологій, що уніфікувати додаток та надати можливість розгортання на різних платформах [9-13, 17, 26, 29-30, 39-41, 46].

Актуальність даного дослідження полягає у тенденції сучасних технологій до віртуалізації, тобто всі матеріали та накопичені данні переводяться у цифровий формат та зберігаються в хмарних середовищах. Це дозволяє отримати доступ до них в будь-якій точці світу. Те ж саме відноситься і до фінансової сфери. Створюються віртуальні гроші, кошти мігрують з фізичного обігу до мобільного банкінгу [14-16, 24-25, 38-39], що дозволяє використовувати їх за допомогою будь якого пристрою у якого є доступ в глобальну мережу.

Об’єктом дослідження є процес організації сервісів мобільного банкінгу в хмарних середовищах.

Предметом дослідження є архітектура програмного забезпечення для використання в хмарних середовищах з можливістю як горизонтального так і вертикального масштабування та простого внесення нових мікросервісів.

Метою даної дипломної роботи є розробка мікробанку на основі хмарних технологій з можливістю покращеного масштабування та модифікації.

Для досягнення даної мети у дипломній роботі потрібно вирішити такі задачі:

— проаналізувати основні способи та засоби створення програмного забезпечення для мікробанків на основі хмарних технологій;

— розробити мікросервісну архітектуру мікробанку на основі хмарних технологій.

Наукова новизна запропонованого рішення полягає у тому, що на відміну від відомої монолітної архітектури для вибраного Web-додатку пропонується мікросервісна архітектура, яка полягає у створенні декількох мікросервісів кожен з яких відповідає за свої окремі функції. Мікросервіси обмінюються даними між собою за допомогою поставників повідомлень (message broker). Це покращує можливість масштабування та модифікації програмного забезпечення для хмарних технологій.

Практична цінність роботи полягає у тому, що вона може бути використана для розміщення у хмарних середовищах програмного забезпечення невеликих банків кількість яких останнім часом стрімко зростає. Для таких банків особливо актуальною є можливість оперативної зміни контенту в залежності від зміни фінансового законодавства.

Структура та розмір роботи. Робота містить вступ, три розділи, економічну частину, список використаної літератури, додатки. Перший розділ містить коротку інформацію про технології, які використовуються в цій статті. У другому розділі описується архітектура проекту. У третьому розділі описано його розробку та приклади роботи. Список літератури містить 51 літературне джерело. У додатках міститься технічне завдання на роботу з переліком кодів основних програмних модулів.

За результатами дослідження опубліковано тези доповіді: особливості розробки та взаємодії з базою даних мовою Java [Текст] С.М. Свіріпа. Молодь в науці: дослідження, проблеми, перспективи (МН-2021) Тез. доп. — Вінниця, 2021. — Режим доступу https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/paper/view/9145 [1]

1 АНАЛІЗ ІСНУЮЧИХ СПОСОБІВ ТА ЗАСОБІВ СТВОРЕННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МІКРОБАНКУ НА ОСНОВІ ХМАРНИХ ТЕХНОЛОГІЙ

Сьогодні досить важко уявити хоча б одну галузь науки без інформаційних технологій. Наприклад, у сфері банкінгу це онлайн кабінети де ви можете здійснити оплату, перерахунки коштів, отримати заробітню плату, сплатити комунальні послуги і таке інше. На сьогоднішній день є багато засобів для такої реалізації, але в кожного засоба є свої переваги та недоліки. Тому в цьому розділі ми розглянемо допоміжні засоби які розширують функціонал та допоможуть написати швидкодіючий та маштабуючий варіант мікробанку на мові програмування Java.

1.1 Мова програмування Java

Мова програмування Java[6] зародилася в 1991 р. в лабораторіях компанії Sun Microsystems [7].

Основною метою створення Java була потреба в мові програмування, незалежному від платформи (тобто архітектурі) і яку можна було б використовувати для створення програмного забезпечення, вбудованого в різні домашні електронні пристрої, такі як мобільний зв'язок, керування віддаленими пристроями і т.д. .

Незабаром майже всі найпопулярніші веб-браузери того часу змогли безпечно запускати Java-аплети всередині веб-сторінок. У грудні 1998 року Sun Microsystems випустила Java 2 (спочатку називалася J2SE 1.2), що реалізувала кілька змін для різних типів платформ. Наприклад, J2EE був призначений для корпоративних програм і значно скорочений J2ME для пристроїв з обмеженими ресурсами, таких як мобільні телефони. У 2006 році для маркетингових цілей версія J2 була перейменована в Java EE [8], Java ME та Java SE [9] відповідно.

Період становлення Java збігся з розвитком міжнародного інформаційного сервісу World Wide Web[10]. Це було вирішальним у

майбутньому Java, оскільки Інтернет також потребував незалежних від платформи програм. Як наслідок, наголос у розвитку Sun перемістився з побутової електроніки на програмування в Інтернеті.

1.2 Java Framework Spring Boot

Spring Framework [11] (або скорочено Spring) — це універсальний фреймворк із відкритим вихідним кодом [12] для платформи Java. Існує також форк для .NET Framework [13], який називається Spring.NET [13].

Хоча Spring не надав конкретної моделі програмування, вона набула широкого поширення у спільноті Java в основному як альтернатива та заміна моделі Enterprise JavaBeans. Spring дає багато свободи розробникам Java у дизайні; крім того, він надає добре задокументовані та прості у використанні рішення проблеми, що виникають під час створення програм для всього підприємства.

Тим часом, функції ядра Spring застосовні в будь-якій програмі Java, і є багато розширень і вдосконалень для створення веб-застосунків на платформі Java Enterprise. З цих причин Spring став дуже популярним і визнаним розробниками стратегічно важливим фреймворком.

1.3 Таблиця стилів CSS

Каскадні таблиці стилів (скорочено CSS) — це спеціальна мова, яка використовується для опису сторінок, написаних мовою розмітки.

CSS найчастіше використовується для візуального представлення сторінок, написаних на HTML [15] та XHTML, але формат CSS застосовується до інших типів документів XML [16].

Специфікації CSS були створені та розробляються Консорціумом World Wide Web.

CSS має різні рівні та профілі. Наступний рівень CSS базується на попередніх, додаючи нові функції або розширюючи наявні функції. Рівні позначаються як CSS1, CSS2 та CSS3. Профілі – це набір правил CSS одного

або декількох рівнів, призначених для певних типів пристроїв або інтерфейсів. Наприклад, профілі CSS для принтерів, мобільних пристроїв і т.д.

CSS (каскадний або блоковий макет) замінив табличне розташування веб-сторінок. Основною перевагою блокового макета є поділ вмісту сторінки (даних) та їхнє візуальне уявлення.

CSS використовується розробниками веб-сторінок та відвідувачами для визначення кольору, шрифтів, макета та інших аспектів зовнішнього вигляду сторінки. Однією з головних переваг є можливість відокремити вміст сторінки (або вміст, вміст зазвичай HTML, XML або подібну мову розмітки) від типу документа (описаного у CSS).

Цей поділ може покращити сприйняття та доступність вмісту, забезпечити велику гнучкість та контроль над відображенням вмісту в різних середовищах, зробити вміст більш структурованим та простим, усунути дублювання тощо. CSS також дозволяє адаптувати вміст до різних умов відображення (на екрані монітора, мобільного пристрою (КПК), у друкованому вигляді, на екрані телевізора, на пристроях, що підтримують браузер або голосові браузери тощо).

Один і той же документ HTML або XML може відображатися по-різному залежно від CSS.

Стандарт CSS визначає порядок і діапазон стилів, що застосовуються, тобто в якій послідовності і для яких елементів використовуються стилі. Таким чином, використовується каскадний принцип, коли для елементів вказується лише інформація про стилі, що змінилася або не визначена спільними стилями.

1.4 Мова розмітки документів HTML

HTML [15] (HyperText Markup Language) є стандартною мовою розмітки для веб-сторінок в Інтернеті. Більшість веб-сторінок створюються за допомогою HTML (або XHTML). HTML документ обробляється

браузером та відтворюється на екрані у звичайній людській формі.

HTML похідно від SGML, успадкувавши від нього визначення типу документа та ідеології текстової розмітки. Хоча HTML є штучною комп'ютерною мовою, це не мова програмування. HTML, поряд з каскадними таблицями стилів та вбудованими сценаріями, також є трьома основними технологіями для створення веб-сторінок.

HTML реалізує інструменти для:

— створення структурованого документа із зазначенням структурного складу тексту: заголовки, абзаци, списки, таблиці, цитати тощо;

— отримання інформації із всесвітньої мережі за допомогою гіперпосилань;

— створення інтерактивних форм;

— додавання у текст зображення, звук, відео та інші об'єкти.

1.5 Бібліотека Hibernate

Hibernate [18] — бібліотека для мови програмування Java, призначена для вирішення задач об'єктно-реляційного відображення (ORM), найпопулярніша реалізація специфікації JPA. Розповсюджується безкоштовно на правах GNU General Public License.

Дозволяє знизити обсяг низькорівневого програмування під час роботи з реляційними базами даних; може використовуватися як у процесі проектування системи класів і таблиць з нуля, так роботи з існуючими базами даних.

Бібліотека не тільки вирішує проблему зв'язування класів Java з таблицями бази даних (і типів даних Java з типами даних SQL [19]), але також надає інструменти для автоматичного генерування та оновлення набору таблиць, запитів та обробки отриманих даних та може значно скорочувати час. розробки, яка зазвичай витрачається на ручне написання SQL та JDBC-коду [20]. Hibernate автоматизує генерацію запитів SQL та звільняє розробника від ручної обробки результуючого набору даних та

перетворення об'єктів, максимально полегшуючи перенесення (портування) програми до будь-якої бази даних SQL.

Hibernate забезпечує прозору підтримку зберігання POJO [21] (тобто для стандартних об'єктів Java); єдиною суворою вимогою до збереженого класу є наявність конструктора за умовчанням (без параметрів). Методи equals() і hashCode() також слід враховувати для правильної поведінки у деяких додатках.

1.6 JWT токен

Маркер JWT [22] є відкритим стандартом (RFC 7519) для створення маркерів доступу на основі JSON. Зазвичай використовується для передачі даних для автентифікації в програмах клієнт-сервер. Токени створюються сервером, підписуються секретним ключем і передаються клієнту, який використовує цей маркер для перевірки особистості.

Маркер JWT складається з трьох частин: заголовка, корисного навантаження та даних підпису або шифрування. Перші два елементи є об'єктами JSON певної структури. Третій елемент розраховується на основі першого та залежить від обраного алгоритму (у разі беззнакового JWT можна опустити). Токени можуть бути перекодовані в компактне уявлення (JWS/JWE Compact Serialization): до заголовка та корисного навантаження застосовується алгоритм кодування Base64-URL, після чого додається підпис і всі три елементи розділяються точками («.»).

У заголовку вказується необхідна інформація для опису самого токена.

Обов'язковий ключ тут лише один, alg, алгоритм, який використовується для підпису/шифрування (у разі непідписаного JWT використовується значення "none").

Необов'язкові ключі:

— type, тип токена (type), використовується у разі, коли токени змішуються з іншими об'єктами, що мають заголовки JOSE, має значення «JWT»;

— cty, тип вмісту (content type), якщо в токені крім зареєстрованих службових ключів є користувацькі, то цей ключ не повинен бути присутнім, в іншому випадку повинно мати значення «JWT».

Корисне навантаження. У цій секції вказується інформація користувача (наприклад, ім'я користувача та рівень його доступу), а також можуть бути використані деякі службові ключі. Усі вони є необов'язковими:

— iss, чутливий до регістру рядок або URI, який є унікальним ідентифікатором сторони, що генерує токен (issuer);

— sub, чутливий до регістру рядок або URI, який є унікальним ідентифікатором сторони, про яку міститься інформація в даному токені (subject), значення з цим ключем мають бути унікальними в контексті сторони, що генерує JWT;

— aud, масив чутливих до регістру рядків або URI, що є списком одержувачів цього токена, коли сторона, що приймає, отримує JWT з цим ключем, вона повинна перевірити наявність себе в одержувачах — інакше проігнорувати токен (audience);

— exp, час у форматі Unix Time [23], що визначає момент, коли токен стане невалідним (expiration);

— nbf, на противагу ключу exp, це час у форматі Unix Time, що визначає момент, коли токен стане валідним (not before);

— jti, рядок, що визначає унікальний ідентифікатор цього токена (JWT ID);

— iat, час у форматі Unix Time, який визначає момент, коли токен був створений. iat і nbf можуть не збігатися, наприклад, якщо токен був створений раніше, ніж час, коли він має стати валідним.

1.7 Використання JavaScript

JavaScript [24] — це багатопарадигмальна мова програмування. Підтримує об'єктно-орієнтований, імперативний та функціональний стилі. Є

реалізацією мови ECMAScript[25] (стандарт ECMA-262).

JavaScript зазвичай використовується як вбудована мова для доступу до об'єктів програми. Він найбільш широко використовується в браузерах як мова сценаріїв, щоб зробити веб-сторінки інтерактивними.

Основні архітектурні особливості: динамічний набір, слабкий набір, автоматичне керування пам'яттю, програмування прототипів, функції як об'єкти першого класу.

На JavaScript вплинули багато мов, з метою зробити мову схожою на Java, але легкою для непрограмістів. JavaScript не належить жодній компанії або організації, яка відрізняє його від низки мов програмування, які використовуються у веб-розробці. Назва JavaScript є зареєстрованою торговою маркою Oracle Corporation [26].

1.8 Використання веб-сервісу GitHub

GitHub [27] — найбільший веб-сервіс для розміщення IT-проектів та їхньої спільної розробки. Заснована на системі контролю версій Git та розроблена на Ruby on Rails [28] та Erlang [29] GitHub, Inc. (Раніше Logical Awesome).

Сервіс повністю безкоштовний для проектів з відкритим кодом та надає їм усі можливості (включно з SSL [30]), а для приватних проектів пропонуються різні платні тарифні плани.

Система розроблена як набір програм спеціально розроблених для їх використання у сценаріях. Це дозволяє легко створювати спеціалізовані системи контролю версій на основі Git або інтерфейсу користувача. Наприклад, Cogito є таким прикладом інтерфейсу для репозиторіїв Git. І StGit використовує Git для керування колекцією патчів.

Система має ряд інтерфейсів користувача: наприклад, gitk і git-gui поширюються разом з самим Git.

Віддалений доступ до репозиторію Git забезпечує git-daemon, SSH або HTTP-сервер [31]. Служба TCP git-daemon є частиною дистрибутива Git і

разом із SSH є найбільш поширеним та надійним методом доступу. Метод доступу HTTP, незважаючи на ряд обмежень, дуже популярний у контрольованих мережах, оскільки дозволяє використовувати наявні конфігурації мережного фільтра.

4 червня 2018 року Microsoft купила GitHub за 7,5 мільярда доларів.

1.9 Програмне забезпечення IntelliJ IDEA

IntelliJ IDEA [32] — це інтегроване середовище розробки програмного забезпечення для багатьох мов програмування, включаючи Java, JavaScript, Python, розроблене JetBrains.

Перша версія з'явилася в січні 2001 року і швидко стала популярною як перше середовище Java із широким набором інтегрованих рефакторингових інструментів, що дозволило програмістам швидко реорганізувати вихідний код програми. Дизайн середовища орієнтований на продуктивність програмістів, що дозволяє зосередитися на функціональних завданнях, тоді як IntelliJ IDEA бере на себе рутинні операції.

Починаючи з шостої версії, IntelliJ IDEA надає інтегровані інструменти для розробки графічного інтерфейсу користувача. Серед інших функцій середовище добре сумісне з багатьма популярними безкоштовними інструментами розробника, такими як CVS, Subversion, Apache Ant, Maven [33] та JUnit [34]. У лютому 2007 року розробники IntelliJ оголосили про ранню версію плагіна для підтримки програмування Ruby.

Версія IntelliJ IDEA для спільноти підтримує інструменти (у вигляді плагінів) для тестування TestNG та JUnit, CVS, Subversion, Mercurial та Git, інструменти компіляції Maven, Ant, Gradle, Java, Scala, Clojure, Groovy та Dart мови програмування.

Підтримується розробка програм для мобільної платформи Android. Він включає модуль візуального дизайну для графічного інтерфейсу Swing UI Designer, редактор XML, редактор регулярних виразів, систему перевірки коду, систему моніторингу завдань та надбудови для імпорту та експорту

проектів із Eclipse. Доступні інструменти для інтеграції із системами відстеження помилок JIRA, Trac, Redmine, Pivotal Tracker, GitHub, YouTrack, Lighthouse.

Комерційна версія «Ultimate Edition» має ПІДТРИМКУ додаткових мов програмування (наприклад, PHP, Ruby, Python, JavaScript, CoffeeScript, HTML, CSS, SQL), підтримку Java EE, UML [35]-діаграм, розрахунок покриття коду, можливість роботи з фреймворками (Rails, Grails, Google Web Toolkit, Spring, Play Framework та Hibernate), інструментами інтеграції з Perforce, Microsoft Team Foundation Server та Rational ClearCase.

1.10 База даних PostgreSql

PostgreSQL [36] — це об'єктно-реляційна система управління базами даних (СУБД). Це альтернатива як комерційним СУБД (Oracle Database, Microsoft SQL Server), і СУБД з відкритим кодом (MySQL, Firebird, SQLite).

У порівнянні з іншими проектами з відкритим кодом, такими як Apache, FreeBSD або MySQL, PostgreSQL не контролюється однією компанією, і його розвиток можливий завдяки співпраці багатьох людей та компаній, які бажають використовувати цю базу даних та впроваджувати останні розробки.

Сервер PostgreSQL написаний на C. Зазвичай він поширюється як набір текстових файлів із необробленим кодом. Для встановлення необхідно зібрати файли на комп'ютері та скопіювати їх у будь-який каталог. Весь процес докладно описано у документації.

1.11 Бібліотека React

React [37] (іноді React.js або ReactJS) є бібліотекою JavaScript з відкритим вихідним кодом для розробки інтерфейсів користувача.

React розробляється та підтримується Facebook, Instagram та спільнотою окремих розробників та корпорацій.

React можна використовувати для розробки односторінкових та мобільних програм. Його мета — забезпечити високу швидкість, простоту та масштабованість. Як бібліотека для розробки інтерфейсів користувача, React часто використовується з іншими бібліотеками, такими як MobX, Redux [38] і GraphQL [39].

1.11.1 Особливості

Однонаправлена передача даних. Властивості передаються від батьківських компонентів до нащадків. Компоненти отримують властивості як набір незмінних значень, тому компонент може безпосередньо змінювати властивості, але може викликати зміни з допомогою функцій зворотного виклику. Цей механізм називається "властивості вниз, події вгору".

Віртуальний DOM. React використовує віртуальну DOM [40] (virtual DOM). React створює структуру кеша в пам'яті, яка дозволяє обчислити різницю між попереднім та поточним станом інтерфейсу для оптимального оновлення DOM браузера. Таким чином, програміст може працювати зі сторінкою, вважаючи, що вона поновлюється всюди, але бібліотека вирішує, які компоненти сторінки оновлюються.

JSX. JavaScript XML (JSX) — це розширення синтаксису JavaScript, що дозволяє використовувати синтаксис, подібний до HTML, для опису структури інтерфейсу. Зазвичай, компоненти написані за допомогою JSX, але можна використовувати звичайний JavaScript. JSX схожий на іншу мову, створений Facebook для розширення PHP, XHP.

Методи життєвого циклу. Методи життєвого циклу дозволяють розробнику запускати код різних етапах життєвого циклу компонента. приклад:

— shouldComponentUpdate — дозволяє запобігти перемальовці компонента за допомогою false, якщо перефарбування не потрібне;

— componentDidMount — викликається після першого відтворення компонента, що часто використовується щоб отримання даних з віддаленого джерела через API;

— render — найважливіший метод життєвого циклу, кожен компонент повинен мати цей метод, зазвичай викликаний змінами даних компонента для перемальовування даних в інтерфейсі.

Не просто малювання HTML в браузері. React використовується не тільки для відтворення HTML в браузері. Наприклад, у Facebook є динамічні графіки, які відображаються в тегах <canvas>. Netflix та PayPal використовують ізоморфні завантаження для відображення ідентичного HTML на сервері та клієнті.

React Hooks. Хуки дають змогу використовувати стан та інші функції React без написання класів. Побудова хуків користувача дозволяє розмістити логіку компонентів у багаторазових функціях.

1.12 Redux

Redux — бібліотека JavaScript із відкритим кодом для керування станом програми. Найчастіше використовується у зв'язку із розробкою клієнта React або Angular. Містить ряд інструментів, які значно спрощують передачу даних зберігання через контекст. Творці: Данило Абрамов та Ендрю Кларк.

Redux — це бібліотека з простим API, що передбачає сховище статусу програми. Вона працює за тим самим принципом, що і функція зменшення, одна з концепцій функціонального програмування. Його авторів надихнула багатофункціональна мова програмування Elm.

1.13 Docker

Docker [41] — це інструмент управління ізольованими контейнерами Linux. Docker доповнює набір інструментів LXC API вищого рівня, що дозволяє управляти контейнерами лише на рівні ізоляції окремих процесів.

Зокрема, Docker дозволяє запускати довільні процеси в режимі ізоляції, не турбуючись про вміст контейнера, а потім передавати та клонувати контейнери, створені для цих процесів, на інші сервери, беручи на себе всю роботу зі створення, підтримки та обслуговування контейнерів.

Вихідний код Docker написаний на Go та поширюється під ліцензією Apache 2.0. Набір інструментів ґрунтується на використанні вбудованих механізмів ізоляції на основі ядра Linux на основі простору імен та cgroup. Для створення контейнерів використовують скрипти LXC. Щоб сформувати контейнер, достатньо завантажити базовий образ середовища (команда docker pull base), тоді можна запускати довільні програми в ізольованих середовищах (наприклад, запуск bash можна запустити docker run -i -t base/bin/bash).

Основні можливості Docker:

— можливість розміщувати в ізольованому середовищі різнорідну начинку, що включає різні комбінації файлів, бібліотек, конфігураційних файлів, скриптів, файлів jar, gem, tar тощо;

— підтримка роботи на будь-якому комп'ютері на основі архітектури x86_64 із системою на базі Linux, від ноутбуків до серверів та віртуальних машин, можливість роботи на немодифікованих сучасних ядрах Linux (без патчів) та у звичайних середовищах всіх основних дистрибутивів Linux, включаючи Fedora , RHEL, U , Debian, SUSE, Gentoo та Arch;

— використання полегшених контейнерів для ізоляції процесів від інших процесів та основної системи;

— оскільки контейнери використовують власну автономну файлову систему, не має значення, де, коли та в якому середовищі вони працюють.

— ізоляція на рівні файлової системи: кожен процес виконується в повністю окремій кореневій ФС;

— ізоляція ресурсів: споживання системних ресурсів, таких як споживання пам'яті та навантаження на процесор, може бути обмежено кожним контейнером за допомогою cgroups;

— мережеві ізоляція на рівні мережі, кожен ізольований процес має доступ тільки до простору мережевих імен, пов'язаних з контейнером, включаючи інтерфейс віртуальної мережі та пов'язані IP-адреси;

— коренева файлова система для контейнерів створюється за допомогою механізму копіювання під час запису (окремо зберігаються лише змінені та нові дані), що дозволяє прискорити розгортання, зменшити споживання пам'яті та заощадити місце на диску;

— всі стандартні потоки (stdout/stderr) кожного процесу, що виконується в контейнері, накопичуються та зберігаються у вигляді журналу;

— модифікована файлова система одного контейнера може бути використана як основа для формування нових базових зображень та створення інших контейнерів, без необхідності розробляти шаблони або вручну налаштовувати композицію зображень;

— можливість використання інтерактивної оболонки, до стандартного введення будь-якого контейнера можна прив'язати псевдо-tty для запуску оболонки;

— підтримка використання різних систем зберігання даних, які можна підключати як плагіни, серед підтримуваних драйверів сховища є aufs, мапери пристроїв (використовуються знімки LVM), vfs (на основі копіювання каталогу) та Btrfs, драйвери для ZFS, Gluster та Ceph очікуються;

— можливість створення контейнерів, що містять складні програмні стеки, шляхом зв'язування наявних контейнерів, що містять компоненти сформованого стеку, прив'язка відбувається не шляхом поєднання вмісту, а через забезпечення взаємодії між контейнерами (створення мережевого тунелю).

Docker складається з двох процесів:

— "Демон Docker", що працює на гостьовій машині (якщо це Linux), або всередині середовища VirtualBox boot2docker (якщо це Windows або OS X);

— клієнт, за допомогою якого можна взаємодіяти з демоном;

— «Образ Docker» — містить операційну систему, програму та всі її залежності, зображення в Docker складаються з шарів, якщо нам потрібен образ із веб-сервером, беремо за основу образ із дистрибутивом операційної системи, додаємо залежність веб-сервер, і ми пишемо це як нове зображення, яке матиме два шари — один із ОС, що йде з веб-сервером, зображеннями можна обмінюватися через DockerHub [42];

— Docker-контейнер — це запущений образ, контейнери Docker можна запускати, зупиняти, переміщати та видаляти, ви також можете створити контейнер докерів, який створить зображення з поточного стану контейнера.

1.14 Swagger Specification

The OpenAPI Specification [43] (з англ. «специфікація OpenAPI»; спочатку відома як Swagger Specification) — формалізована специфікація та екосистема безлічі інструментів, що надає інтерфейс між front-end системами, кодом бібліотек низького рівня і комерційними рішеннями у вигляді API. Разом з тим специфікація побудована таким чином, що не залежить від мов програмування, і зручна у використанні як людиною, так і машиною.

Щодо призначення, OpenAPI розглядається як універсальний інтерфейс для користувачів (клієнтів) щодо взаємодії із сервісами (серверами). Якщо спроектована специфікація для деякого сервісу, то на її підставі можна генерувати вихідний код для бібліотек клієнтських додатків, текстову документацію для користувачів, варіанти тестування та інші. Для цих дій є великий набір інструментів для різних мов програмування та платформ.

Спочатку розробка специфікації під назвою Swagger Specification проводилася з 2010 року компанією SmartBear. У листопаді 2015 року SmartBear оголосила, що вона працює над створенням нової організації Open API Initiative за спонсорської підтримки Linux [44] Foundation. 1 січня 2016

специфікація була перейменована в The OpenAPI Specification, а її розвиток ведеться в рамках Open API Initiative.

1.15 Kafka [44]

Сучасні серверні програми складні, багатоярусні та включають безліч компонентів та сервісів. Архітектори програмного забезпечення виділяють на окремі модулі все, що можна: розсилку SMS, системи збору статистики, підсистеми авторизації.

Навіщо? По-перше, можна розбити великі важкі завдання на маленькі шматочки і вирішувати частинами. По-друге, це дозволяє розподілити навантаження і додати стійкості до відмови.

Але таким розподіленим системам треба якось передавати дані між собою. Тут на сцені з'являються системи обміну повідомленнями (брокери повідомлень, диспетчери повідомлень). Це розгалужена система труб, в яку з одного кінця можна кинути, наприклад, контейнер з повідомленнями, а з іншого кінця його хтось отримає і прочитає. До таких систем відносять і Apache Kafka.

Система комунікацій між сервісами, якщо вона грамотно вибудувана, дозволяє компонентам ставити одне одному завдання, повідомляти про зміни в системі та повідомляти зацікавлені частини логіки про свої стани.

Брокер повідомлень Kafka — розподілена система. Його сервери об'єднуються у кластери. Зберігання та пересилання повідомлень йде паралельно на різних серверах, а це дає більшу надійність та відмовостійкість. Навіть при виході з ладу кількох машин повідомлення все ще пересилатимуться і оброблятимуться.

Також сервіс легко масштабується горизонтально. Тобто, для нарощування потужності Apache Kafka достатньо вводити до ладу додаткові сервери. Це найпростіший у реалізації спосіб масштабування систем, але він підходить не для всіх. Наприклад, із базами даних такий підхід не працює —

незрозуміло, що робити із записами в таблицях на нових серверах. Kafka ж спочатку заточили під вибухове зростання продуктивності.

Ще один плюс — консистентність даних. Записи в Apache Kafka зберігаються як журналу комітів. Це виглядає як черга повідомлень, у яку можна додавати записи, а ось видаляти чи модифікувати — ні. Такий підхід дає величезну надійність та простоту зміни будь-яких станів — завжди зрозуміло, що, як і в якій послідовності змінювалося.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МІКРОБАНКУ

В даному розділі описано розробку архітектури програмного забезпечення мікробанку на основі хмарних технологій. Запропонована реалізація архітектури з використанням мікросервісів. Представленні схеми взаємодії мікросервісів між собою, використані бібліотеки для зв’язку даних мікросервісів та описано процес автоматизації розгортання даного програмного забезпечення.

2.1 Розробка структури мікросервісів мікробанку

Розробка даного програмного забезпечення базується на створенні набору невеликих мікросервісів, кожен з яких працює у своєму власному процесі та взаємодіє з бібліотеками, що полегшує взаємодію частин через API-інтерфейси та HTTP-ресурси. Ці мікросервіси побудовані на бізнес-задачі мікробанку і повинні розгортатися незалежно за допомогою повністю автоматизованого механізму. Також повинен існувати мінімальний рівень централізованого управління цими службами, які можуть бути написані різними мовами програмування та використовувати різні технології зберігання даних.

Для досягнення даної мети запропоновано реалізувати архітектуру яка складається з двох мікросервісів admin та transfer-service, кожен з яких використовує однакові класи для взаємодії між собою і тому було вирішено створити додаткові бібліотеки для їх повторного використання. Було створено п’ять бібліотек: configuration-lib, яка відповідає за базову конфігурацію мікросервіса; transfer-lib, призначену для опису сутностей обміну даними з мікросервісом transfer-service; domain-lib, що забезпечує взаємодію з базою даних; security-lib, яка надає можливість коригування прав доступу конкретних користувачів до кінцевих точок мікросервісів та message-lib, що забезпечує надсилання повідомлень на конкретний номер

телефону. На рисунку 2.1 представлено графічне зображення запропонованої архітектури програмного забезпечення мікробанку.

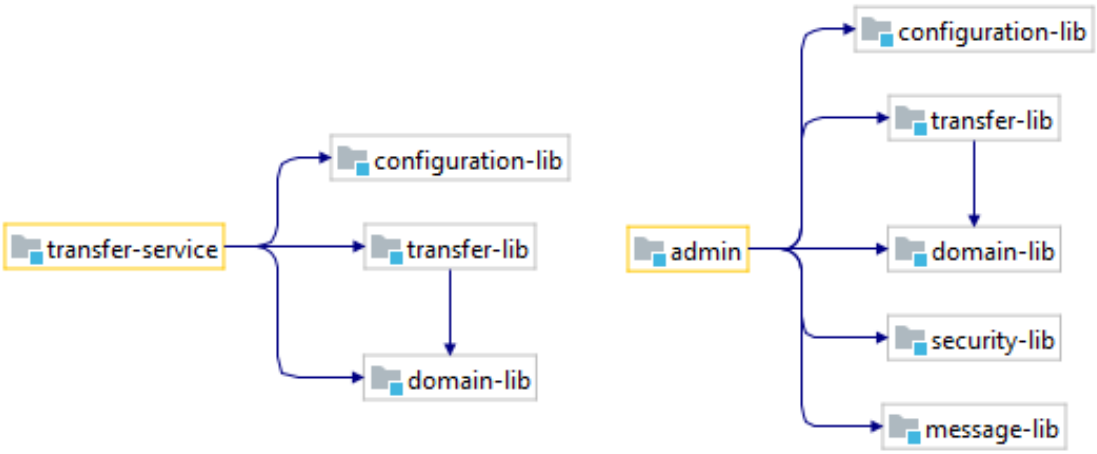


Рисунок 2.1 — Архітектура програмного забезпечення мікробанку на основі мікросервісів

Данні мікросервіси спілкуються між собою за допомогою асинхронного обміну повідомленнями в форматі JSON. Для того що полегшити взаємодію обміну повідомленнями було створено додатковий модуль який виступає в ролі бібліотеки, в ньому розміщені під-модулі з класами даних мікросервісів, що зображені на рисунку 2.2. Це зумовлено повторним використанням коду а також без шовного зв’язку.

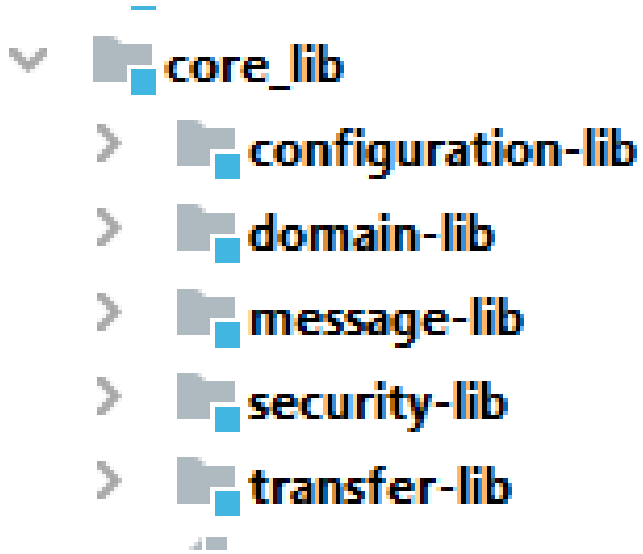


Рисунок 2.2 — Бібліотека мікробанку

Було виділено шість властивостей мікросервісу:

— він не великий;

— він незалежний;

— він будується навколо бізнес-потреби та використовує обмежений контекст (Bounded Context);

— він взаємодіє з іншими мікросервісами через мережу на основі патерну Smart endpoints and dumb pipes;

— його розподілена суть зобов'язує використати підхід Design for failure;

— процеси його розробки та підтримки вимагають автоматизації.

2.1.1 Розробка дизайну мікросервісів

Розробка мікросервісів admin та transfer-service полягає у визначенні об’єму бізнес процесу які мають описати дані мікросервіси. Як індикативні оцінки для створення компактного мікросервіса виділено рекомендації наведені нижче. Розмір мікровервісу повинен бути таким, щоб виконувалася одна з умов:

— один сервіс може розвивати одна команда не більше ніж з дюжини людей;

— контекст (не тільки бізнесу, а й розробки) одного сервісу міститься в голові однієї людини;

— один сервіс може бути повністю переписаний однією командою за короткий термін.

За даними критеріями створено admin та transfer-service мікросервіси, в них містяться класи controller які відповідають за кінцеві точки програмного забезпечення які взаємодіють з зовнішнім API через HTTP протокол, як приклад приведено структура класів з методами admin мікросервіса без відображення допоміжних бібліотек зображено на рисунку 2.3.

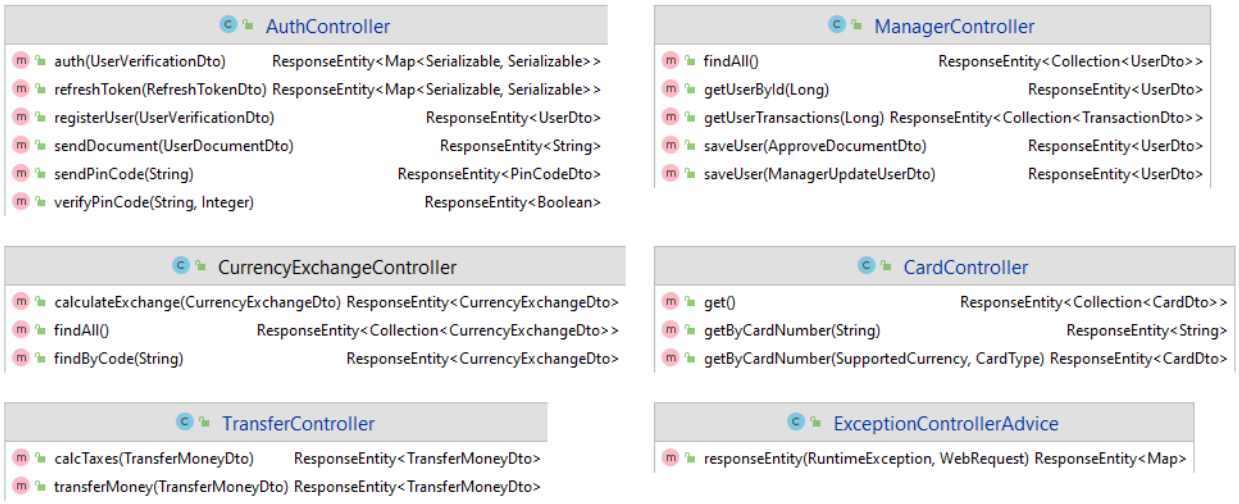


Рисунок 2.3 — Структура admin мікросервісу

2.1.2 Розробка незалежного виконання

Розробка даної архітектури є втілення патернів High Cohesion та Low Coupling. Все, що суперечить цьому відкидлось. Отже admin та transfer-service мають бути незалежними мікросервісами.

Дане програмне забезпечення розробляється з розбиттям на компоненти, які безумовно ґрунтуються на тих же принципах, що і мікросервіси.

В моноліті загальна кодова база відкриває можливості порушення низької пов'язаності. Із-за слабкої дисципліни рано чи пізно код перетворюється на велику кількість зв’язків між своїми компонентами які важко підтримувати та вносити зміни. Під таке формулювання компонента підходять сторонні бібліотеки які використані під час розробки даного програмного забезпечення які наведені на рисунку 2.4.

Розбиття на мікросервіси змушує дотримуватися жорсткого їх поділу, адже вони мають відповідати жорсткішим критеріям незалежності.

Кожен розроблений мікросервіс працює у своєму процесі і тому має свій API. Враховуючи, що інші компоненти можуть використовувати тільки цей API і він віддалений, мінімізація зв'язків стає життєво важливою.

Такий поділ дає явний виграш із погляду незалежного розвитку різних

компонентів.

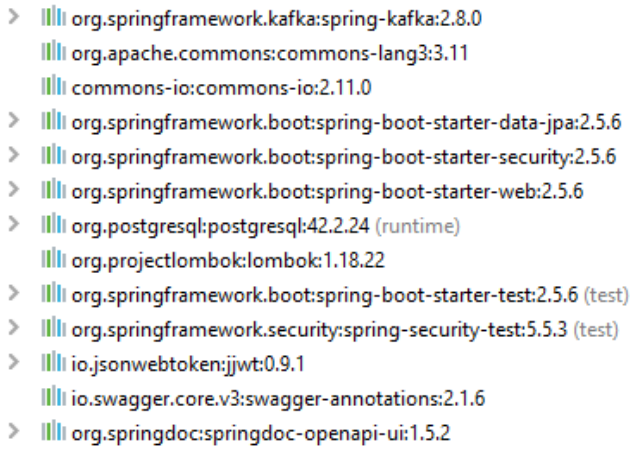


Рисунок 2.4 — Сторонні бібліотеки admin мікросервіса

В даній архітектурі не заборонено використання бібліотек. Їх використання не вітається, оскільки так чи інакше призводить до залежностей між мікросервісами, але все ж таки допускається.

Незалежність мікросервісів дозволяє організувати незалежний життєвий цикл розробки, створювати окремі зборки, тестувати та розгортати.

Оскільки розмір мікросервісів невеликий, очевидно, що під час подальшої розробки програмного забезпечення мікросервісів ставатиме все більше. Керувати ними вручну буде складно. Тому нижче в розділі наведено приклад автоматизації відповідно до Continuous integration і Continuous Delivery.

2.1.3 Бізнес-потреба

Розробка даного програмного забезпечення базується на визначенні функцій за які відповідають мікросервіси.

Щоб визначити принцип зони відповідальності мікросервісу було проаналізовано бізнес-потребу та сформовано основні функції мікросервісів.

Якщо відкинути приставку controller то можна побачити основні функції за які відповідає сервіс, авторизація, дії з банківськими картками,

валюти, перевід коштів та функції для менеджера які зображені на рисунку 2.5. І чим вона компактніша, чим формалізовані її взаємини з іншими областями, тим простіше створити новий мікросервіс. На ньому ґрунтується створення будь-яких інших компонентів. Надалі витримувалась ця зона відповідальності, що обговорювалась у попередньому параграфі.

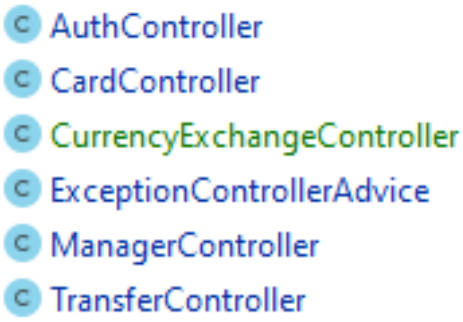


Рисунок 2.5 — Основні бізнес потреби admin мікросервіса

Коли межі мікросервісу задані і він виділений в окрему кодову базу, захистити ці межі від стороннього впливу не важко. Далі всередині мікросервісу був розроблений свій мікросвіт, спираючись на патерн "обмеженого контексту" зображеного на рисунку 2.6, де кожний мікросервіс інтерпретує свій контекст.

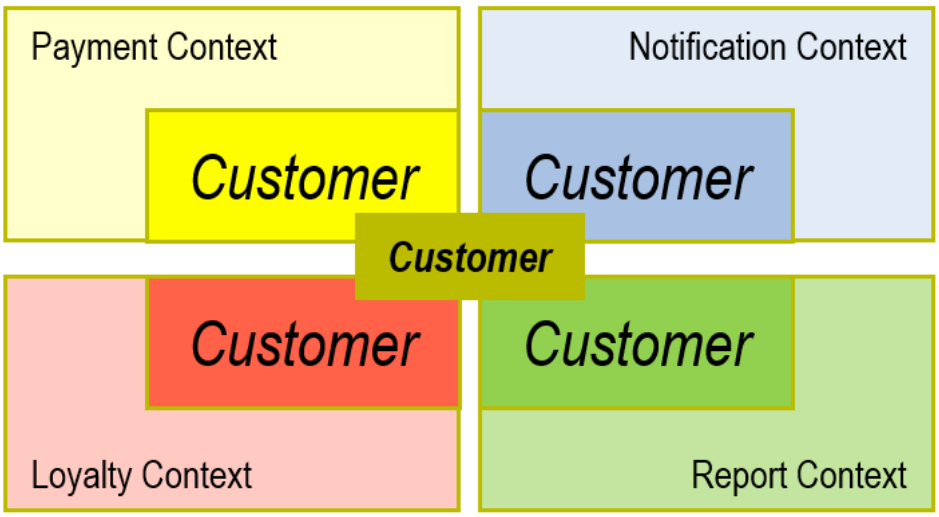


Рисунок 2.6 — Приклад патерну "обмеженого контексту"

Мікросервісна архітектура приводить дану розробку у виграшний стан

оскільки мікросервіси дають багато переваг в пропускній здатності та масштабуванню даного проекту.

2.1.4 Інтеграція. Smart endpoints and dumb pipes

Спочатку для інтеграції був використаний простий текстовий протокол, заснований на HTTP, щоб нівелювати можливу технологічну різницю мікросервісів. REST-подібні протоколи є майже стандартом. Як виняток можуть використовуватися бінарні протоколи типу Java RMI або .NET Remoting.

Було виділено кілька зауважень:

— основною складністю мікросервісів є визначення їх меж. Складність полягає в тому, що локальні виклики методів стають віддаленими. І це впливає на стиль взаємодії, оскільки часті виклики не підходять;

— асинхронна взаємодія є більш ефективна ні синхронна, сервіси повинні взаємодіяти між собою за допомогою event архітектури (Event Driven Architecture), а самі мікросервіси повинні відповідати вимогам Reactive.

Для досягнення даних критеріїв було використано технологію для доставки повідомлень Kafka. За її допомогою дуже просто відправити повідомлення зображене на рисунок 2.7.

```
private final KafkaTemplate<String, TransferMoneyDto> kafkaTemplate;

@PostMapping
public ResponseEntity<TransferMoneyDto> transferMoney(@RequestBody @Valid TransferMoneyDto transferMoneyDto) {
    kafkaTemplate.send( topic: "outside-bank-transfers", transferMoneyDto);
```

Рисунок 2.7 — Відправка повідомлення за допомогою Kafka

Метод send є асинхронним і він повертає клас який ви можете проігнорувати або підписатись на нього і очікувати результат виконання даного асинхронного виклику.

2.1.5 Design for failure для розподіленої системи

Було визначено, що одне з найбільш критичних місць у мікросервісній архітектурі — необхідність розробляти код для розподіленої системи, складові якої взаємодіють через мережу.

А мережа ненадійна за своєю природою. Мережа може просто відмовити, може працювати погано, може раптом перестати пропускати якийсь тип повідомлень, тому що змінилися налаштування файрвола. Десятки причин та видів недоступності.

Тому мікросервіси можуть раптом перестати відповідати, можуть почати відповідати повільніше, ніж зазвичай. І кожен віддалений виклик має це враховувати. Повинен правильно обробляти різні варіанти відмови, вміти чекати, вміти повертатися до нормальної роботи під час відновлення контрагента приклад реалізації зображено на рисунку 2.8.

```
@Scheduled(fixedRate = 3_600_000)
@Retryable( value = RuntimeException.class,
        maxAttempts = 6, backoff = @Backoff(delay = 5000))
public void scheduleFixedDelayTask() {
    final ResponseEntity<List> forEntity = template.getForEntity(BANK_URL, List.class);
    Map<String, CurrencyExchangeDto> currencyExchangeDtoMap = new HashMap<>();
    if (forEntity.getStatusCode().is2xxSuccessful()) {
        forEntity.getBody().forEach(o -> {
            Map<String, Object> raw = (Map)o;
            currencyExchangeDtoMap.put(
                    raw.get("cc").toString(),
                    new CurrencyExchangeDto((String) raw.get("cc"), ONE, valueOf((Double) raw.get("rate")), raw.get("txt").toString())
            );
        });
    } else if (forEntity.getStatusCode().is5xxServerError()) {
        throw new RuntimeException();
    }
    currentCurrencyExchange = Collections.unmodifiableMap(currencyExchangeDtoMap);
```

Рисунок 2.8 — Приклад повторного надсилання запиту

Анотація Retryable буде задіюватись тільки в тих випадка коли даний метод отримає помилку як зазначено в параметрі value, після цього код буде повторно виконаний максимум 6 разів з інтервалом в 5 секунд.

Додатковий рівень складності приносить подібна архітектура. А налагодження такої системи — не одного мікросервісу, а системи, де багато потоків різноспрямованих невпорядкованих подій, навіть важко уявити. І

навіть якщо кожен із мікросервісів буде бездоганним з погляду бізнес-логіки, цього мало.

І оскільки складність таких систем є дуже високою, то проблему вирішено так:

— вона просто відповідає необхідним дисфункціям, але в ній можуть бути помилки, що незначно впливають на її стійкість і продуктивність;

— повинне бути повне покриття коду unit тестами, інтеграційними та тестами продуктивності;

— має бути інтелектуальний моніторинг, який не лише моментально показує непрацюючі місця, а й сигналізує про погіршення стану системи з прогнозуванням можливих збоїв;

— повинне бути просунуте розподілене логування, що дозволяє оперативно проводити розслідування, і часто за результатами виправляються приховані помилки.

Все це корисно і для будь-якого моноліту, але для мікросервісів така інфраструктура — питання життя та смерті.

Оскільки основною проблемою виникнення помилок є невірно введені данні, тому це було вирішено в першу чергу приклад обробки невірних вхідних даних зображено на рисунку 2.9.

```
@Override
public UserDto findById(Long id) {
    return mapper.toDto(userRepository.findById(id).orElseThrow(USER_NOT_FOUND), UserDto.class);
}
```

Рисунок 2.9 — Оброблення вхідних даних

Викликається метод для отримання користувача за його індентифікатором в базі даних, якщо користувача не знайдено формується помилка з коротким описом. В протилежному випадку отриманий користувач з бази даних трансформується, тобто повертається інша сутність з

обмеженими полями для того щоб не повертати секретні дані. На цьому положенні формуються усі методи, для початку потрібно обробити вхідні данні та якщо щось введено не вірно потрібно повернути помилку і всі змінені дані повертаються в попередній стан. Оскільки основна ідея не допустити довільної модифікації даних.

2.1.6 Автоматизація запуску

Було використано контейнери які використовують такий підхід: вони надають схожий з віртуальними машинами рівень ізоляції, але завдяки правильному залученню низькорівневих механізмів основної операційної системи роблять це в рази меншим навантаженням.

Даний додаток мікробанку не є виключенням, для досягнення автоматизації розгортання використано технологію Docker. Для цього лише потрібно встановити програмний продукт, та написати скрипт для автоматизованого запуску в середині Docker контейнера.

Docker має свою базу з готовими контейнерами для того щоб вам не потрібно було завантажувати такі додатки і прописувати скрипти розгортання вручну. Також великим плюсом є технологія docker compose, вона передбачає опис декількох контейнерів в одному файлі, а також їхній зв'язок і черга запуску якщо наприклад один контейнер залежить від іншого.

В даному файлі ви можете описати основні залежності проекта які зображені на рисунку рисунок 2.10.

```
zookeeper:
  image: confluentinc/cp-zookeeper:latest
  environment:
    ZOOKEEPER_CLIENT_PORT: 2181
    ZOOKEEPER_TICK_TIME: 2000
  ports:
    - "22181:2181"

kafka:
  image: confluentinc/cp-kafka:latest
  depends_on:
    - zookeeper
  ports:
    - "29092:29092"
  environment:
    KAFKA_BROKER_ID: 1
    KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
    KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://kafka:9092,PLAINTEXT_HOST://localhost:29092
    KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: PLAINTEXT:PLAINTEXT,PLAINTEXT_HOST:PLAINTEXT
    KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
    KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 1
```

Рисунок 2.9 — Конфігураційний файл для запуску Kafka в Docker

В скрипті можна задавати змінні які буде використовувати дане програмне забезпечення, вони описані в таблиці 2.1.

Таблиця 2.1 — Перелік параметрів та їх опис

Параметр	Опис
Zookeeper	Назва контейнера
Image	Посилання на готовий контейнер в репозиторії Docker
Environment	Змінні середовища, використовуються додатком для розгортання
Ports	Порти на яких буде доступний додаток всередині контейнера та зовні
Depends_on	Залежності на інші контейнера, контейнер буде очікувати поки всі інші не запустяться

3 РОЗРОБКА ПРОГРАМНИХ ЗАСОБІВ МІКРОБАНКУ

3.1 Постановка завдання

Під час створення сайту слід визначити план, за яким ми працюватимемо. Якщо ви пропустите або не виконаєте якісний елемент, можливі проблеми призведуть до значних витрат часу на виправлення або доопрацювання сайту. Тому при розробці сайту необхідно використовувати мову програмування Java:

— визначитись зі серодою розробки наприклад INTELLIJ IDEA;

— проаналізувати які сутності в будуть використовуватись в пз та набір полів в цих сутностях;

— обрати базу даних;

— створити веб-сторінки для CRUD операцій з транзакціями;

— мова сайту — українська;

— вкладка для адміністрування користувачів;

— підтримка браузерами IE 6.0 і вище, Chrome, Mozilla, Opera;

— підтримати валідацію по номеру телефону;

— завантаження документів;

— швидке доповнення фунціоналу;

— можливість роботи з хмарними технологіями;

— підтримка великої кількості користувачів;

3.2 Створення проекту за допомогою Spring Boot

На початку створення проекту на основі фреймворку Spring Boot необхідно зайти на сайт start.spring.io/, форму буде показано як на малюнку 3.1.

Слід заповнити поля груп, артефакт, пошук залежностей. У полі групи в цьому полі необхідно вказати назву веб-програми, наприклад vtc. Поле артефакту відповідає за назву основного класу нашої програми.

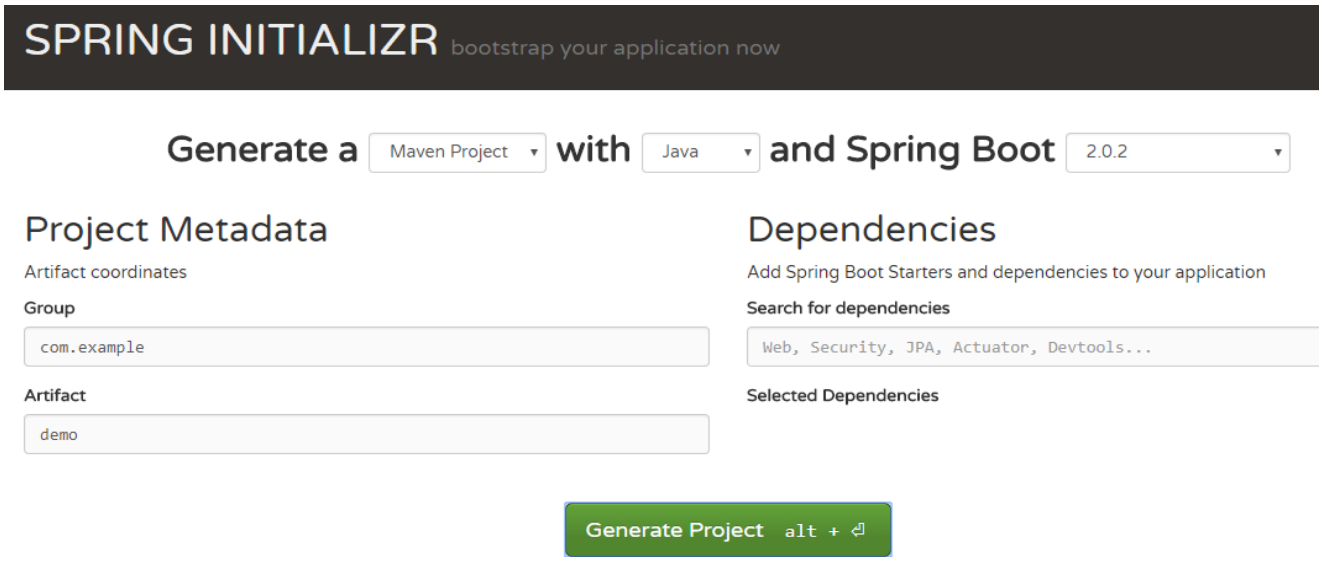


Рисунок 3.1 — Приклад сайту для створення початкового проекту

У полі пошуку залежностей вам потрібно вибрати які залежності ми будемо використовувати у нашому веб-програмі, наприклад: JPA, SECURITY, WEB, POSTGRESQL, KAFKA. У полях генерування залиште maven і Java, скачайте проект і запустіть його у середовищі розробки, як показано малюнку 3.2.

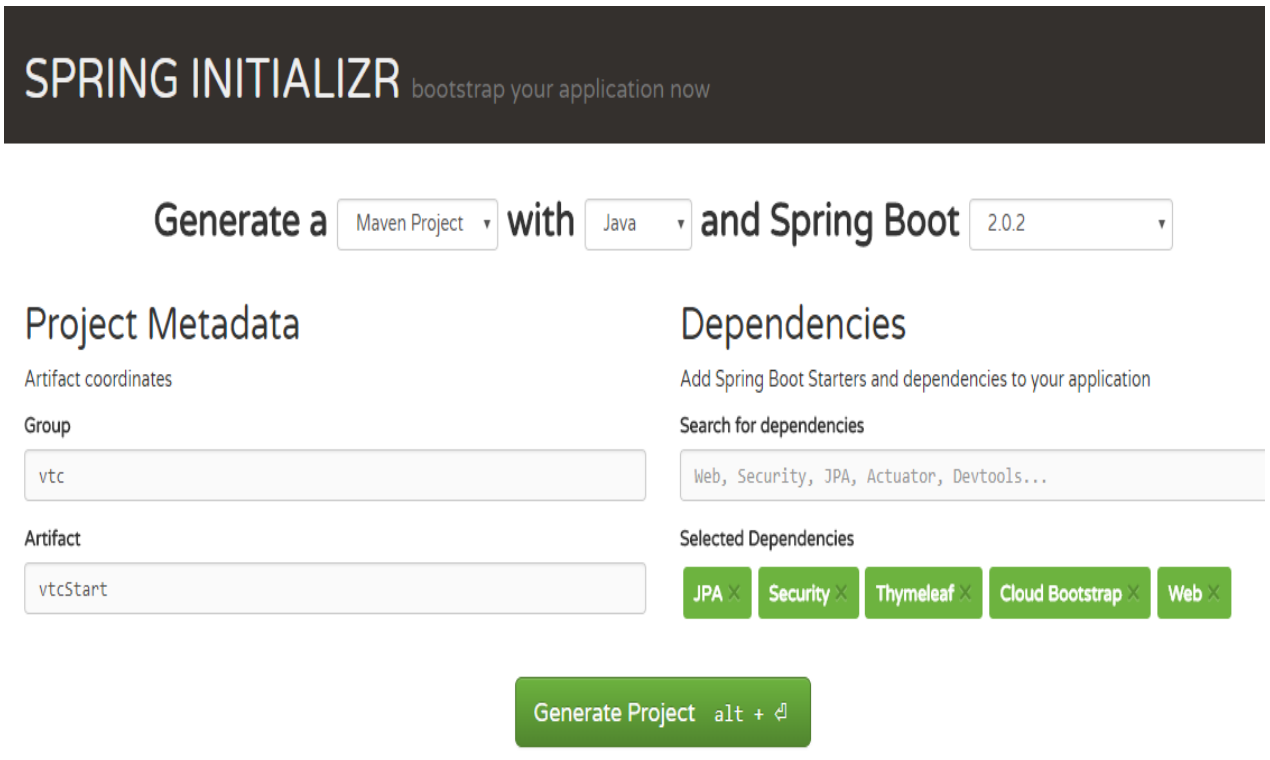


Рисунок 3.2 — Приклад готового стартового проекту на Spring Boot

3.3 Розробка бази даних

3.3.1 Розробка DB-діаграми

Розробку бази даних було розпочато зі створення DB-діаграми з сутностей які були розроблені при аналізі бізнес задачі на рисунку 3.3.

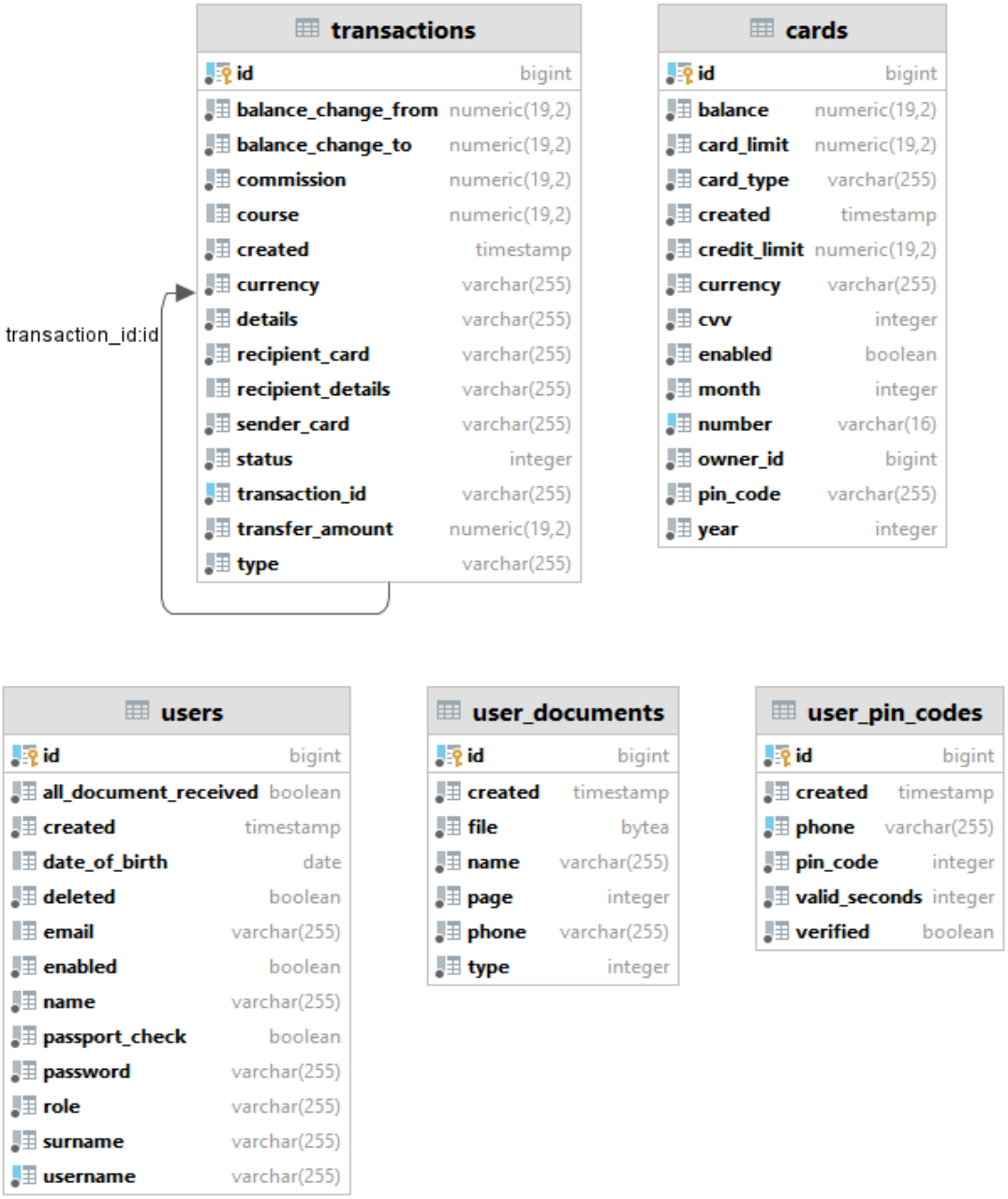


Рисунок 3.3 — Приклад розробленої DB-діаграми під-час початку розробки проекту

При розробці веб-додатка була вдосконалена діаграма БД, внесено корективи до бази даних, фактично додано велику кількість полів під час розробки програми.

Після того, як усі таблиці зображень на DB-діаграмі були створені відповідно до рисунка 3.4, була написана логічна частина та створені всі шаблони HTML, вирішено додати перевірку за номером телефону, будь-яка компанія, що надає такі послуги, може бути додана до проекту просто реалізувавши інтерфейс для відправки SMS.

Після редагування деяких помилок та додавання пропонованого функціоналу до проекту остаточна БД-діаграма виглядає згідно з рисунку 3.4.

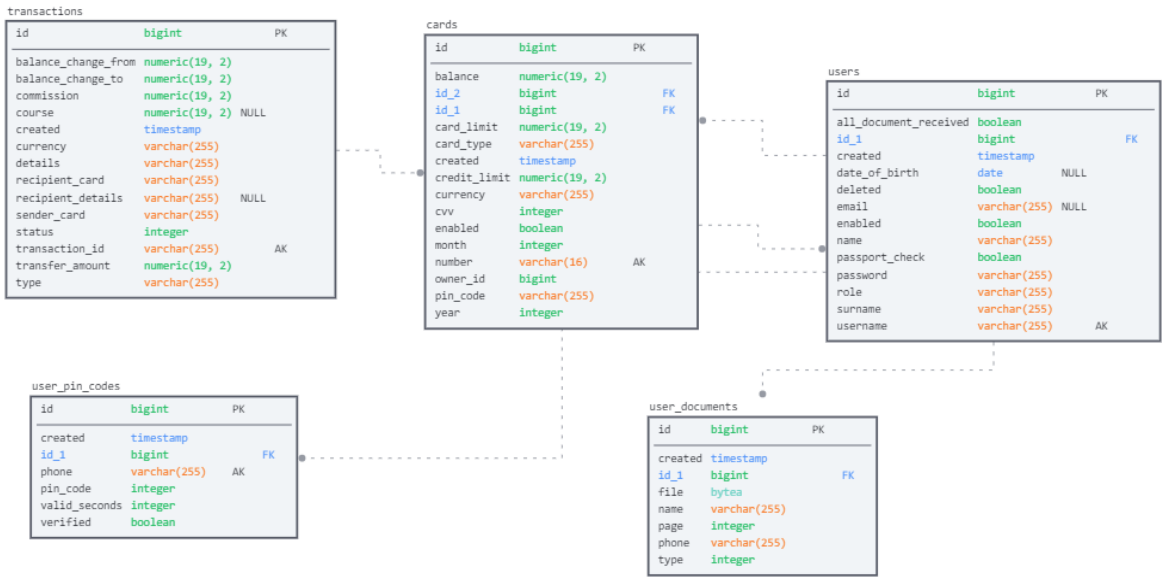


Рисунок 3.4 — Приклад розробленої DB-діаграми в кінцевому результаті розробки проекту

3.3.2 Створення сутностей

На основі аналізу предметної галузі формується перелік програмних сутностей та атрибутів сутностей, представлених класами та їх учасниками (таблиця 3.1).

Після розробки діаграми БД робота перемістилася в середу розробки IntelleJ IDEA, що відкрила раніше створений проект за допомогою Spring

Boot, який створив пакет Entity, і додав класи для кожної сутності з діаграмою ER, в кожному Клас визначив поля, що ця сутність повинна містити, наприклад, у лістингу 3.1 показано поля для об'єкта Pincode.

Таблиця 3.1 — Перелік сутностей та їх атрибутів

Сутність	Атрибути
Транзакії	Зміна баланса з, зміна баланса на, комісія, курс, час створення, валюта, деталі, отримувач, деталі отримувача, відправник, статус, номер транзакції, сума,тип
Картка	Баланс, ліміт, тип, кредитний ліміт, валюта, свв, включена, місяць, власник, пінкод, рік
Користувач	Отримано всі документи, створено, дата народження, видалений, пошта, ввімкнений, імя, паспорт перевірений, роль, прізвище, по батькові.
Пінкод	Створено, телефон, пінкод, дійсний протягом, перевірений
Документи	Створено, файл, імя, сторінка, телефон, тип

Лістинг 3.1 — Приклад полів сутності Пінкод

```
@Table(name = "user_pin_codes")
@Entity
public class PinCode extends AbstractEntity{
@Id
@GenericGenerator(name="increment", strategy = "increment")
@GeneratedValue(generator = "increment")
private Long id;
@Column(nullable = false, unique = true)
private String phone;
@Column(nullable = false)
```

```
private Integer pinCode;
@Column(nullable = false)
private Timestamp created;
@Column(nullable = false)
private Integer validSeconds;
@Column(nullable = false)
private Boolean verified;
private transient String text;
```

Усі поля починаються із символу @, тобто анотації – примітки, з допомогою яких програміст вказує, що робити з частинами коду, крім виконання програми.

Відповідність інструкції:

— @Entity з цією анотацією компілятор бачить, що цей клас відноситься до сутності;

— @Table ця анотація вказує, що цей клас слід передати в бібліотеку сплячого режиму, що працює з базою даних, і створює таблицю з ім'ям, зазначеним у дужках;

— @Column вказує на сплячий режим, що у створеній таблиці потрібно створити поле з назвою, яку ми вказали у дужках.

Використовуючи зв'язок між типами даних, Hibernate автоматично визначає типи даних у полях бази даних.

3.3.3 Створення зв’язків для сутності Users

Клас сутності User містить 13 полів та 2 зв’язка з іншими таблицями. Код класу сутності з створенням зв’язків наведено нижче лістинг 3.2 .

Лістинг 3.2 — Приклад створення зв’язків

```
@Column(nullable = false)
private Timestamp created;
```

```
@Column(nullable = false)
private Boolean passportCheck;
@Column(nullable = false)
private Boolean enabled;
@Column(nullable = false)
private Boolean deleted;
@Column(nullable = false)
private Boolean allDocumentReceived;
@Enumerated(EnumType.STRING)
@Column(nullable = false)
private Roles role;
```

Відповідність анотацій та додатові які можуть використовуватись:

— @ManyToOne за допомогою цієї анотації компілятор бачить, що цей клас потрібно зв'язати з таблицею за допомогою посилання «багато одного»;

— @JoinColumn ця інструкція вказує, на яку таблицю слід посилатися;

— @OneToMany вказує, що ця таблиця має бути пов'язана з іншою таблицею за допомогою «один до багатьох»;

— fetch = FetchType.LAZY — вказує на те, що це з'єднання має будуватися за допомогою відкладеної ініціалізації, тобто після звернення до сутності;

— fetch = FetchType.EAGER — це з'єднання має будуватися з агресивною ініціалізацією, тобто під час компіляції проекту.

— @ Enumerated — ця анотація вказує що тип поля буде енумератором.

3.3.4 Локальний сервер PostgreSQL

Після створення діаграми ER та класів сутностей встановлено локальний сервер PostgreSQL. Ви можете завантажити PostgreSQL з офіційного сайту http://www.postgresql.org/download/windows/. У процесі встановлення програми на кроці 4 установки необхідно ввести логін та пароль, цей логін та пароль необхідно заповнити, вам потрібно буде ввести їх для входу з правами адміністратора. Далі потрібно вказати порт, на якому працюватиме екземпляр PostgreSQL, залиште значення за замовчуванням. Наступним кроком є визначення кодування бази даних, залишеного за замовчуванням.

Після інсталяції була запущена програма PostgreSQL, за замовчуванням після інсталяції програма створює локальний сервер, як показано на малюнку 3.5, локальний сервер бази даних готовий до роботи.

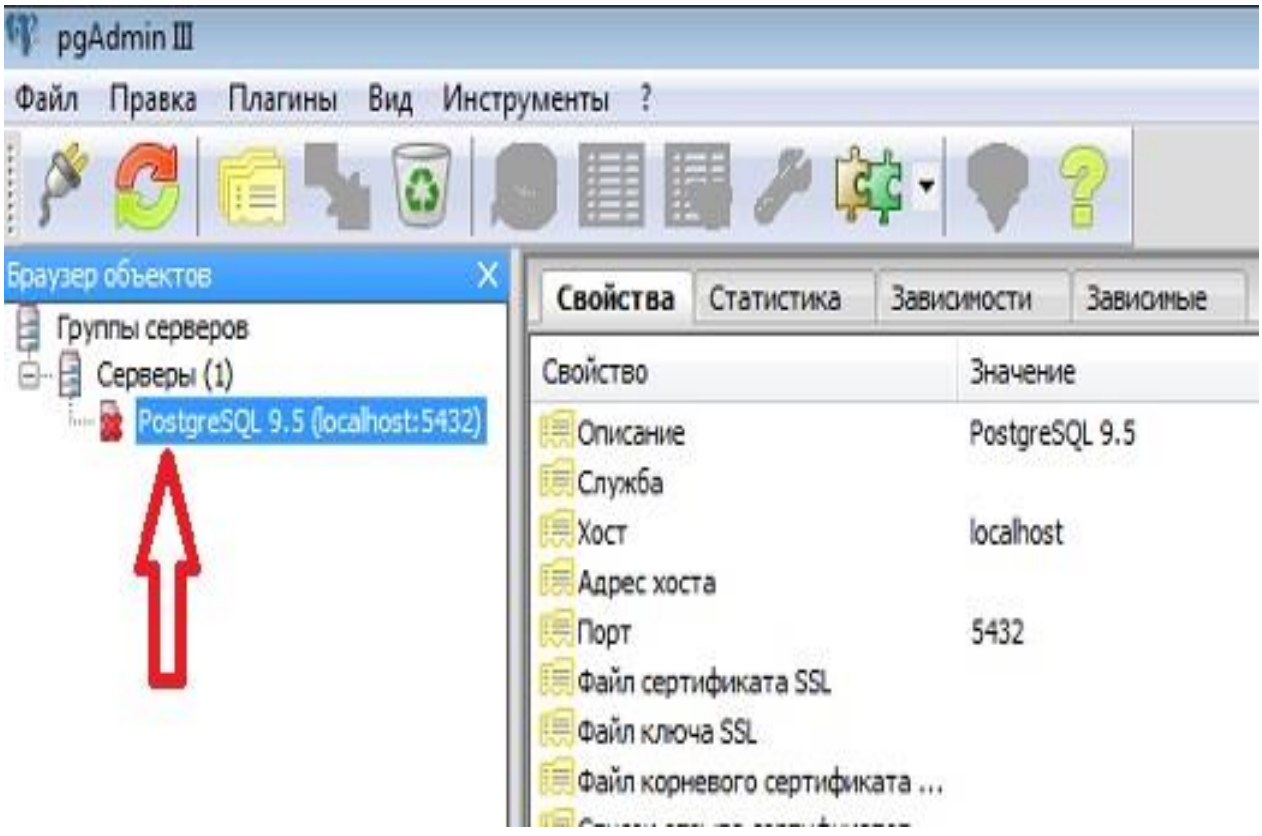


Рисунок 3.5 — Програма для керування базою даних

3.3.5 Налаштування бази даних та створення таблиць за допомогою Hibernate

Після встановлення локального сервера PostgreSQL, щоб середовище розробки IntelliJ IDEA працювало належним чином, потрібно налаштувати файл application.properties відповідно до рисунку 3.6, він розміщується в папці resources модулю core_lib, та підмодулю configuration lib, це було зроблено для уникнення дублювання одних і тих же налаштувань для різних мікросервісів.

```
# JDBC Driver name.
spring.datasource.driver-class-name=org.postgresql.Driver
# Database connection string URL.
# Postgre SQL Server name: linux_test
spring.datasource.url=jdbc:postgresql://localhost:5432/postgres
# Database user's name.
spring.datasource.username=postgres
# Database user's password.
spring.datasource.password=postgres
# Number of ms to wait before throwing an exception if no connection is a
spring.datasource.tomcat.max-wait=10000
# Maximum number of active connections that can be allocated from this po
spring.datasource.tomcat.max-active=50
# Validate the connection before borrowing it from the pool.
spring.datasource.tomcat.test-on-borrow=true

# JPA Configuration
spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect
spring.jpa.generate-ddl=true
```

Рисунок 3.6 — Налаштування файлу app.properties

Обов'язково вкажіть кілька полів. Поле spring.datasource.driver-class-name це поле вказує, де отримати драйвер для PostgreSQL. Поле spring.datasource.url — вказує, куди звертатися для доступу до бази даних. Поля spring.datasource.username та spring.datasource.password — це логін та пароль, які ми ввели при установці Postgre.

Після установки файлу перейдіть на вкладку бази даних і натисніть на вкладку «Додати», виберіть Джерело даних і виберіть PostgreSQL. З'явиться вікно для редагування даних, як показано на малюнку 3.7.

Всі дані були взяті з програми PostgreSQL, яка була встановлена раніше, натисніть на створений сервер і клацніть властивості цього сервера, URL створюється автоматично після введення даних.

Після цього ви можете переглядати діаграми та таблиці, доступні вашому користувачеві, а також редагувати та переглядати дані.

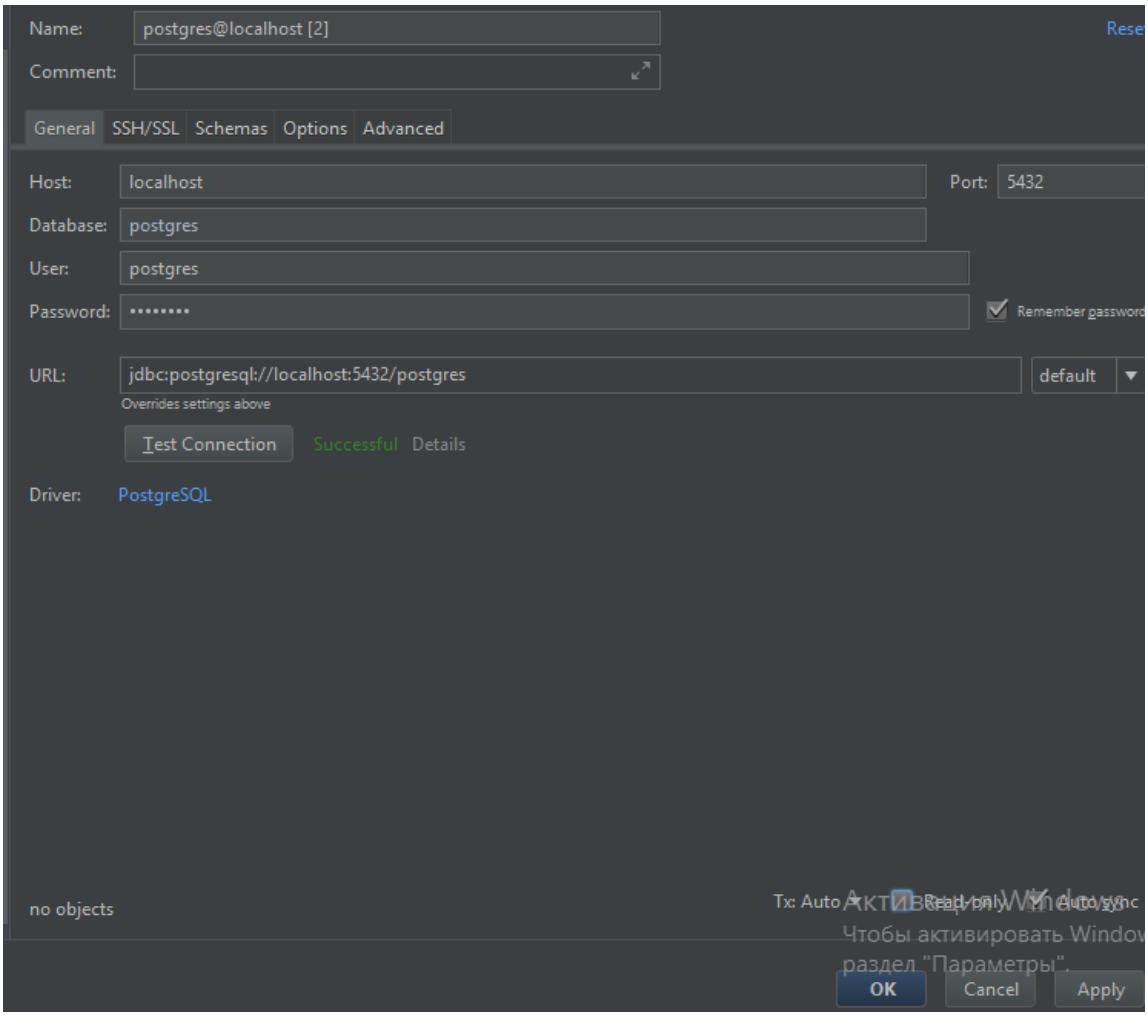


Рисунок 3.7 — Форма для встановдення зєднання з базою даних

Після успішного підключення бази даних, створюємо таблиці для цього натискаємо на кнопку компіляції, за допомогою hibernate під-час компіляції таблиці в базі даних створюються автоматично, hibernate знаходить класи помічені анотаціями Entity та Table, якщо в класах не має додаткових анотацій для створення полів таблиці, таблиці створюються автоматично.

3.4 Розробка запитів та логіки роботи додатку

3.4.1 Створення запитів до бази даних

Запити Java Spring створюються за допомогою Spring Data Jpa. Що вам потрібно Spring Data Jpa — якщо вам потрібно швидко створити рівень репозиторію в проекті на основі JPA, призначеного в основному для операцій CRUD, і ви не хочете створювати абстрактні dao, їх інтерфейси реалізації, Spring Data JPA — це хороший вибір.

І тому потрібен клас сутності, наприклад User. Для початку була створена папка Repository відповідно до рисунка 3.8, що містить інтерфейси для доступу до основних операцій CRUD.

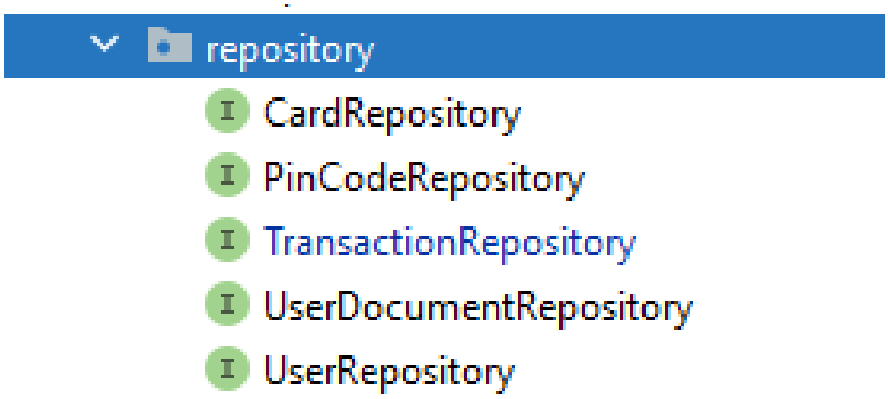


Рисунок 3.8 — Папка Repository

Створюємо UserRepository лістинг 3.3 який успадковуємо від JpaRepository який приймає два параметри клас сутності та другим параметром тип поля ID в класі сутності.

Лістинг 3.3 — Приклад створеного UserRepository

```
public interface UserRepository extends JpaRepository <Users, Long>  {
    Users findByUsername (String login);  Users findByPhone (String phone);List <Users> findAll(); }
```

Щоб створити будь-який запит, якого немає в стандартному наборі методів інтерфейсу, потрібно вказати тип об'єкта, який в кінцевому рахунку

внесе назву методу, я використовую такі ключові слова, як: "знайти", "порядок", ім. я І змінних і так далі. Розробники Spring Data JPA спробували розглянути більшість можливих варіантів, які можуть знадобитися.

Наприклад другий метод, User вказуємо, що зрештою отримаємо об'єкт типу User, findByUsername — у назві цього методу зазначено, що потрібно знайти користувача за логіном, цей метод приймає один параметр стрічки.

3.4.2 Створення бізнес логіки

Для того щоб напряму не взаємодіяти з базою даних, ми створюємо бізнес логіку, через яку ми будем взаємодіяти з базою даних подальшому.

Для цього було створено папку Service у відповідності рисунку 3.9, де для кожного класу сутностей було створено інтерфейс та клас який реалізовував цей інтерфейс з набором основних CRUD операцій та реалізації додаткових методів які були написані в інтерфейсі репозиторія.

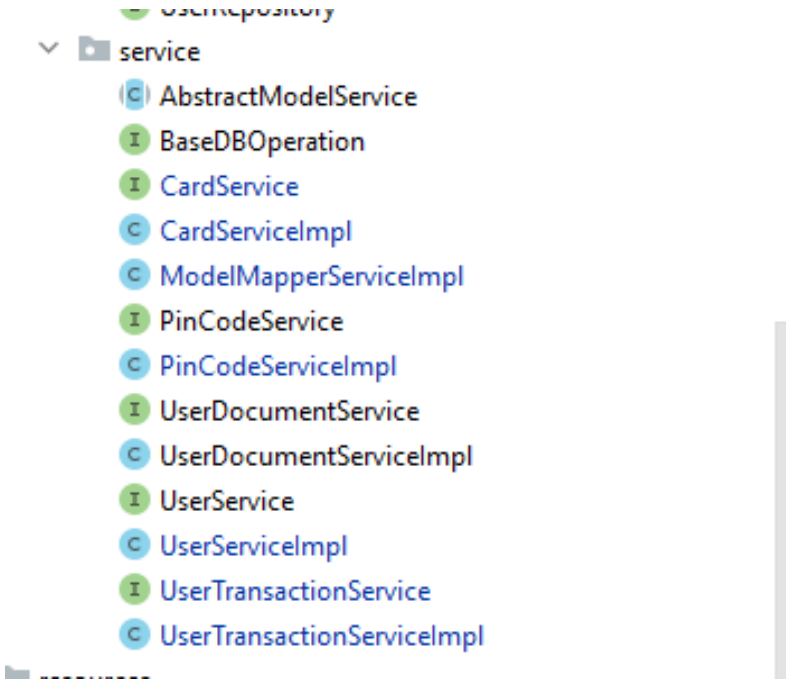


Рисунок 3.9 — Папка Service

Розглянемо створення бізнес логіки на прикладі реалізації класу сутності User. Для цього було створено папку з назвою User в якому було

створено інтерфейс UserService та клас UserServiceImpl який реалізує методи інтерфейсу наведено в лістингу 3.4.

Лістинг 3.4 — Приклад реалізації UserServiceImpl

```
@Service
public class UserServiceImpl extends AbstractModelService implements UserService {
    private static final String FILL_AFTER_DOC_CHECK = "FILL_AFTER_DOC_CHECK";
    private static final Date DEF_DATE_OF_BIRTH = Date.valueOf(LocalDate.of(1900, 1, 1));
    private final UserRepository userRepository;
    private final BCryptPasswordEncoder passwordEncoder;
    private final PinCodeRepository pinCodeRepository;
    @Override
    public UserDto createUser(User user) {
        final LocalDateTime now = LocalDateTime.now();
        pinCodeRepository.findByPhone(user.getUsername())
                .filter(PinCode::getVerified).orElseThrow(WRONG_PIN_CODE);
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        user.setCreated(Timestamp.valueOf(now));
        user.setDeleted(false);
        user.setEnabled(true);
        user.setRole(User.Roles.ROLE_USER_WITHOUT_PASSPORT_CHECK);
        user.setPassportCheck(false);
        user.setAllDocumentReceived(false);
        user.setName(FILL_AFTER_DOC_CHECK);
        user.setSurname(FILL_AFTER_DOC_CHECK);
        user.setDateOfBirth(DEF_DATE_OF_BIRTH);
        return mapper.toDto(userRepository.save(user), UserDto.class);}
```

Наведений вище код представляє методи CRUD. Розглянемо метод addUser, що використовується для компіляції об'єкта сутності та передачі його методу save, що використовується для збереження цього об'єкта у базі даних. Цей метод передає два об'єкти та поле стрічки. Потім об'єкт bCryptPasswordEncoder, який має власний метод, який приймає один аргумент, шифрує пароль. Наступні три рядки коду створюють посилання на об'єкт і знаходять об'єкт за допомогою параметра, переданого цьому методу roleId, знаходять цей об'єкт і додають його до раніше створеного об'єкта. Потім запишіть в об'єкт User об'єкт, що передається, і знайдений об'єкт, збережіть і поверніть збережений об'єкт.

3.4.3 Налаштування Kafka

Оскільки додаток базується на мікросервісній архітектурі тому було створено 2 сервіса, admin та transfer. Щоб забезпечити обмін повідомленнями між ними було використано технологію для передачі повідомлень Kafka. Для того щоб використовувати її для початку вам потрібно встановити декілька додатків zookeeper та kafka. Найпростіше це зробити за допомогою docker потрібно лише скопіювати інструкції наведені в попередньому розділі.

Після створення файлу docker-compose.yml з наведеними вище інструкціями потрібно лише виконати команду *docker compose up* в даній дерективі. Після старту сервісів можна приступати до розробки обміну повідомленнями в нашому додатку. Потрібно підключити залежність в maven наведену на рисунку 3.11 для того щоб використовувати функціонал kafka з коробки.

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
    <version>2.8.0</version>
</dependency>
```

Рисунок 3.11 — Залежність в maven на kafka

Наступним кроком потрібно налаштувати цю бібліотеку, задати всі потрібні змінні. Для цього створюємо клас конфігурування як це показано на рисунку 3.12, та здаємо базові налаштування, адрес на якому ми запустили наш сервіс в docker, та topic – грубо кажучи це таблиці в які записуються повідомлення та зчитуються

```
@Configuration
public class KafkaConfig {
    @Value(value = "localhost:29092")
    private String bootstrapAddress;

    @Bean
    public KafkaAdmin kafkaAdmin() {
        Map<String, Object> configs = new HashMap<>();
        configs.put(AdminClientConfig.BOOTSTRAP_SERVERS_CONFIG, bootstrapAddress);
        return new KafkaAdmin(configs);
    }

    @Bean
    public NewTopic bankTransfersFromOutside() { return new NewTopic( name: "outside-bank-transfers",  numPartitions: 1, (short) 1); }

    @Bean
    public NewTopic bankTransfersFromInside() { return new NewTopic( name: "inside-bank-transfers",  numPartitions: 1, (short) 1); }

    @Bean
    public NewTopic bankTax() { return new NewTopic( name: "bank-transfers-tax",  numPartitions: 1, (short) 1); }
}
```

Рисунок 3.12 — Базові налаштування kafka

Також щоб писати повідомлення в дані таблиці вам потрібно налаштувати формат даних повідомлень. Для цього створимо ще один клас для конфігурування зображеного на рисунку 3.13.

```
@Configuration
public class KafkaProducerConfig {
    @Value(value = "localhost:29092")
    private String bootstrapAddress;

    @Bean
    public <T extends AbstractDto> ProducerFactory<String, T> producerFactory() {
        Map<String, Object> configProps = new HashMap<>();
        configProps.put(
                ProducerConfig.BOOTSTRAP_SERVERS_CONFIG,
                bootstrapAddress);
        configProps.put(
                ProducerConfig.KEY_SERIALIZER_CLASS_CONFIG,
                String.class);
        configProps.put(
                ProducerConfig.VALUE_SERIALIZER_CLASS_CONFIG,
                JsonSerializer.class);
        return new DefaultKafkaProducerFactory<>(configProps);
    }

    @Bean
    public <T extends AbstractDto> KafkaTemplate<String, T> kafkaTemplate(ProducerFactory<String, T> producerFactory) {
        return new KafkaTemplate<>(producerFactory);
    }
}
```

Рисунок 3.13 — Формат повідомлень kafka

Після даних налаштувань ви можете відправити повідомлення як наведено в лістингу 3.5

Лістинг 3.5 — Приклад відправки повідомлення

```
@RestController
@AllArgsConstructor
@RequestMapping("/api/transfer")
public class TransferController
{
   private final KafkaTemplate<String, TransferMoneyDto> kafkaTemplate;
   private final CardService cardService;
   @PostMapping
   public
ResponseEntity<TransferMoneyDto> transferMoney(@RequestBody @Valid
TransferMoneyDto transferMoneyDto) {
      cardService.findByCardOwner(((CustomUserDetails) SecurityHelper
.getUserDetails()).getUserId()).stream()
.filter(cardDto-> cardDto.
getNumber().equals(transferMoneyDto.getOwnerCard().getNumber())
| |cardDto.getNumber().equals(transferMoneyDto .getCard().getNumber()))
.findFirst()
 .orElseThrow(USER_TOKEN_AND_CARD_MISMATCH);
 kafkaTemplate.send("outside-bank-transfers", transferMoneyDto);
 return
ResponseEntity.ok(transferMoneyDto);
}
```

Але і для того щоб отримати це повідомлення потрібно виконати декілька налаштувань схожі на ці що були продемонстровані вище.

3.5 Схема класів які були розроблені

В результаті даної дипломної роботи було створено 2 сервіса наведених на, 5 бібліотек, 88 класів рисунку 3.14, та написано більше чим 2300 стрічок коду.

Це все було зроблено для того щоб досягти максимального перевиконання коду, уникнення дублювання та легкості заміни чи доповнення логіки без виникнення ускладнень, а також для можливості масштабування додатку при великому навантаженню.

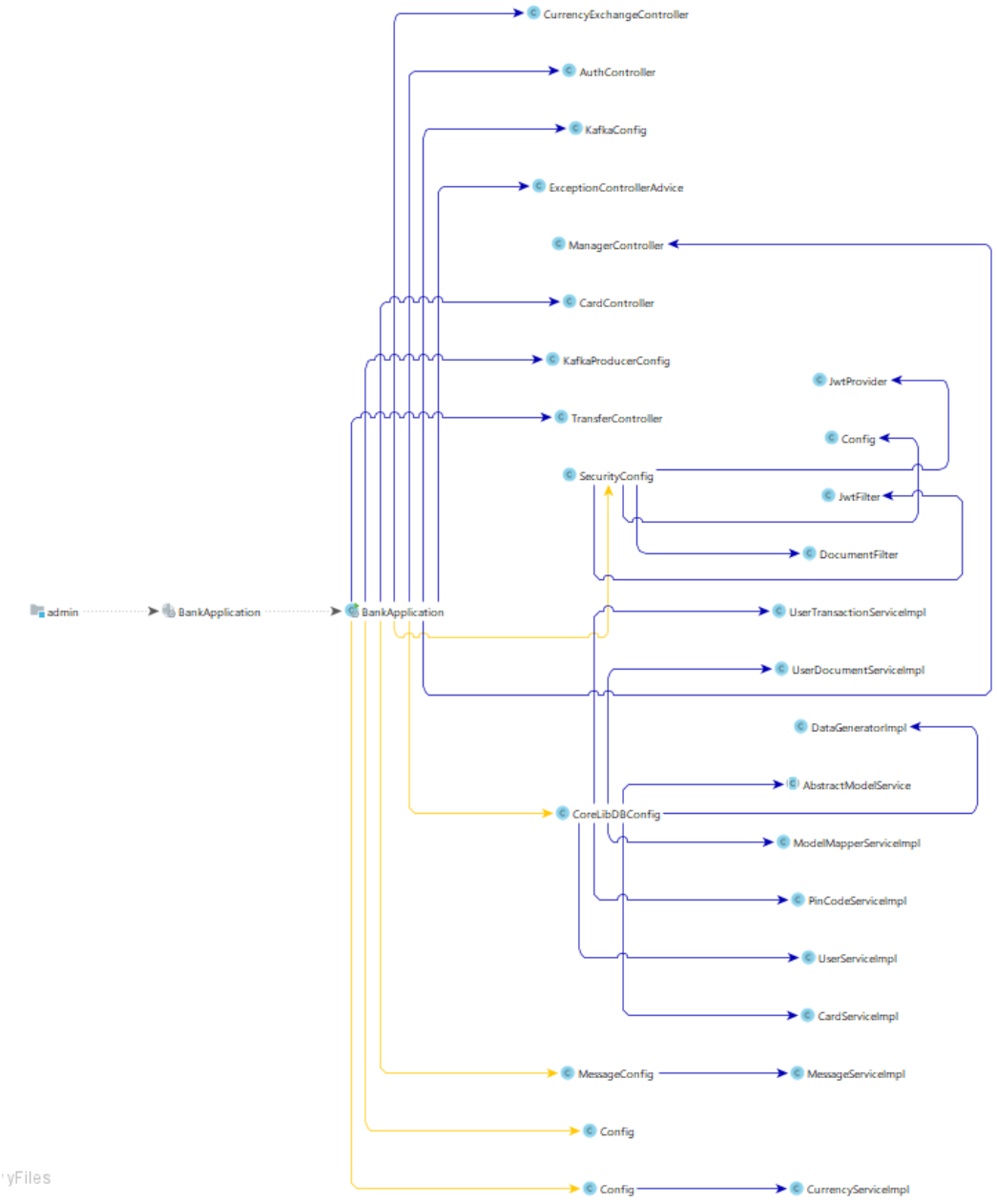


Рисунок 3.13 — Приклад зв’язків для класів для admin сервісу

Як наведено вище кожний клас відповідає за свою окрему роль, і виконує лише одну функцію, що дозволяє більш швидко аналізувати зв’язки і знаходити неполадки в коді або доповняти його. Кожний з цих класів де є приставка Impl реалізує інтерфейс, це допомагає встановлювати зв’язки між класами за допомогою інтерфейсів, великий плюс в цьому що в любий момент часу можна написати власну реалізацію і підмінити стару без будь якої шкоди залежним класам.

3.6 Перевірка роботи програми

Перевірка роботи програмного забезпечення буде здійснюватися за допомогою утиліти для надсилання HTTP запитів Postman.

Для того щоб розпочати працювати в програмному забезпеченні потрібно зареєструватись, для цього кроку знадобиться телефон та декілька запитів на сервер в певній послідовності як зображено на рисунку 3.14

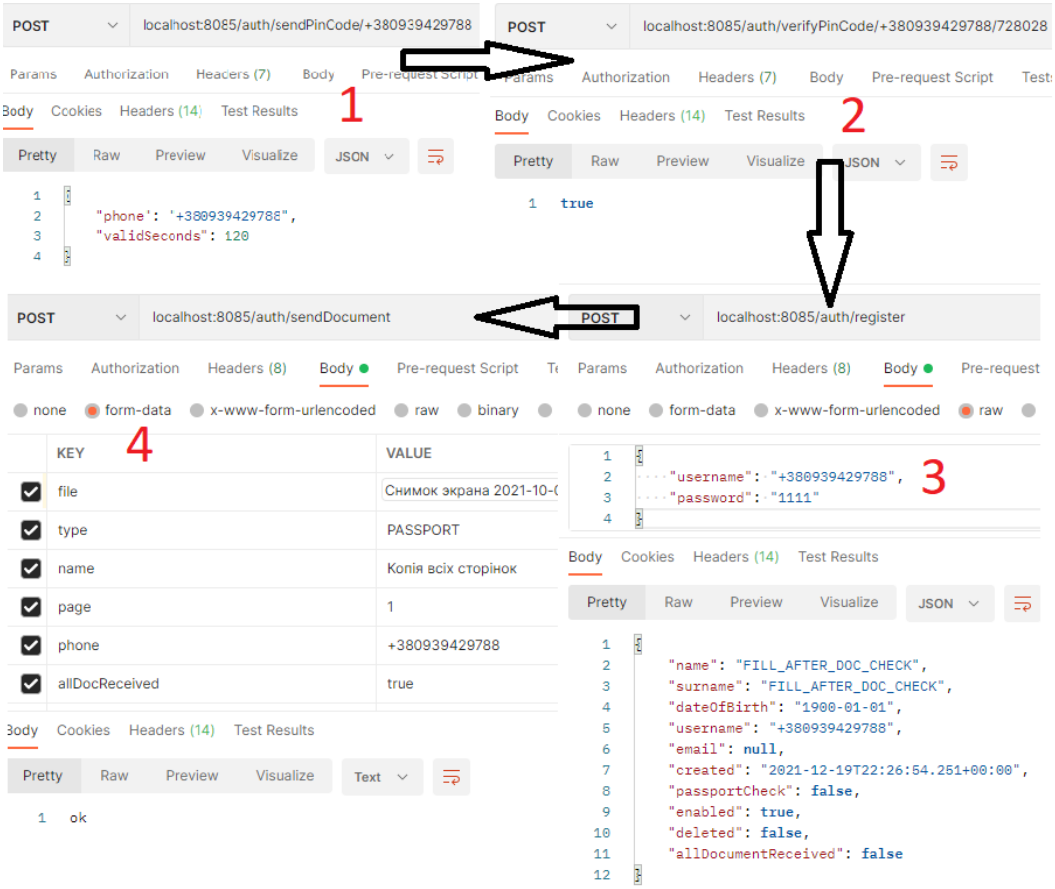


Рисунок 3.14 — Приклад реєстрації нового користувача

Першим запитом потрібно сформувати запит в якому потрібно ввести адресу серверу url адрес кінцевої точки та номер телефону для якого буде здійснюватися реєстрація, після відправки цього запиту надійде відповідь у якій вказано номер телефону на який було відправлено смс з кодом та скільки даний код буде валідним. Після отримання смс потрібно виконати наступний крок, ввести номер телефону для якого буде здійснюватися реєстрація та 6 цифровий код і надіслати на вказану адресу на рисунку вище. Якщо введений код є вірний, потрібно продовжити реєстрацію та ввести пінкод як показано на рисунку 3.14, у відповідь надійде шаблон користувача з заповненими декількома полями а саме номером телефону, пікодом та датою створення всі решта полів буде заповнена після того як буде надіслано файли які підтверджують вашу особистість та перевірені менеджером, тільки після цих кроків кабінет буде створено повністю.

Щоб ввійти менеджеру або користувачу в систему потрібно виконати декілька кроків, а саме відправити форму з номером телефону та паролем на адрес вказаний на рисунку 3.15, у відповідь на цю форму прийде токен який користувач повинен відправляти у кожному запиті для можливості взаємодіяти з інтерфейсом програмного забезпечення мікробанку.

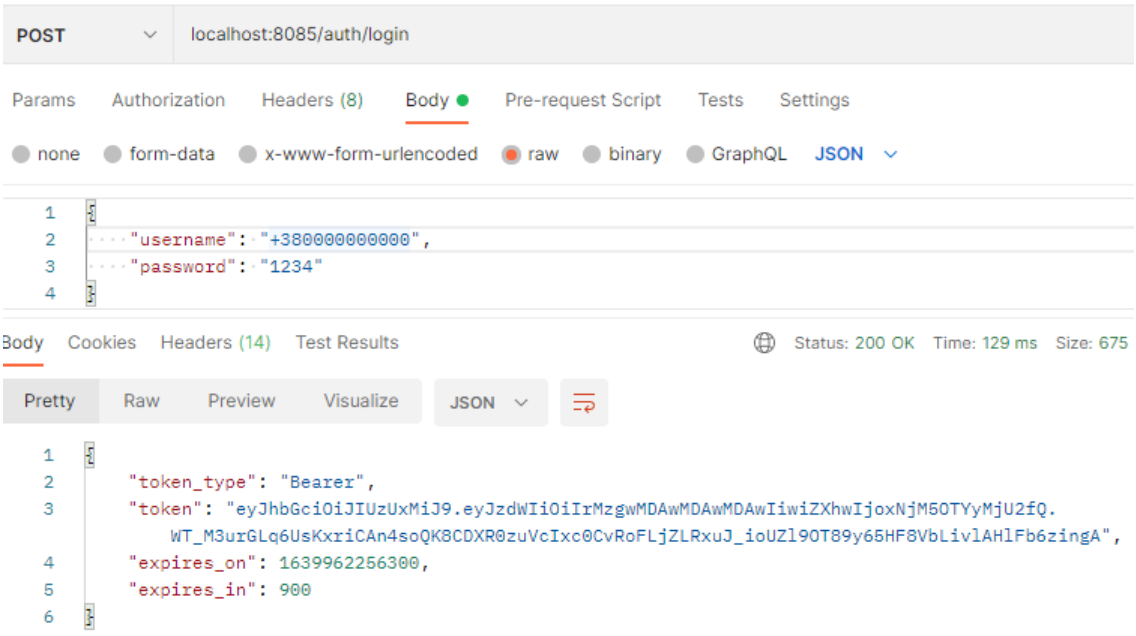


Рисунок 3.15 — Вхід в систему мікробанку

Після входу в систему менеджер повинен перевірити документи нового користувача та ввести всі дані згідно документів та відправити їх з статусом чи користувач має право на використання мікробанку як показано на рисунку 3.16.

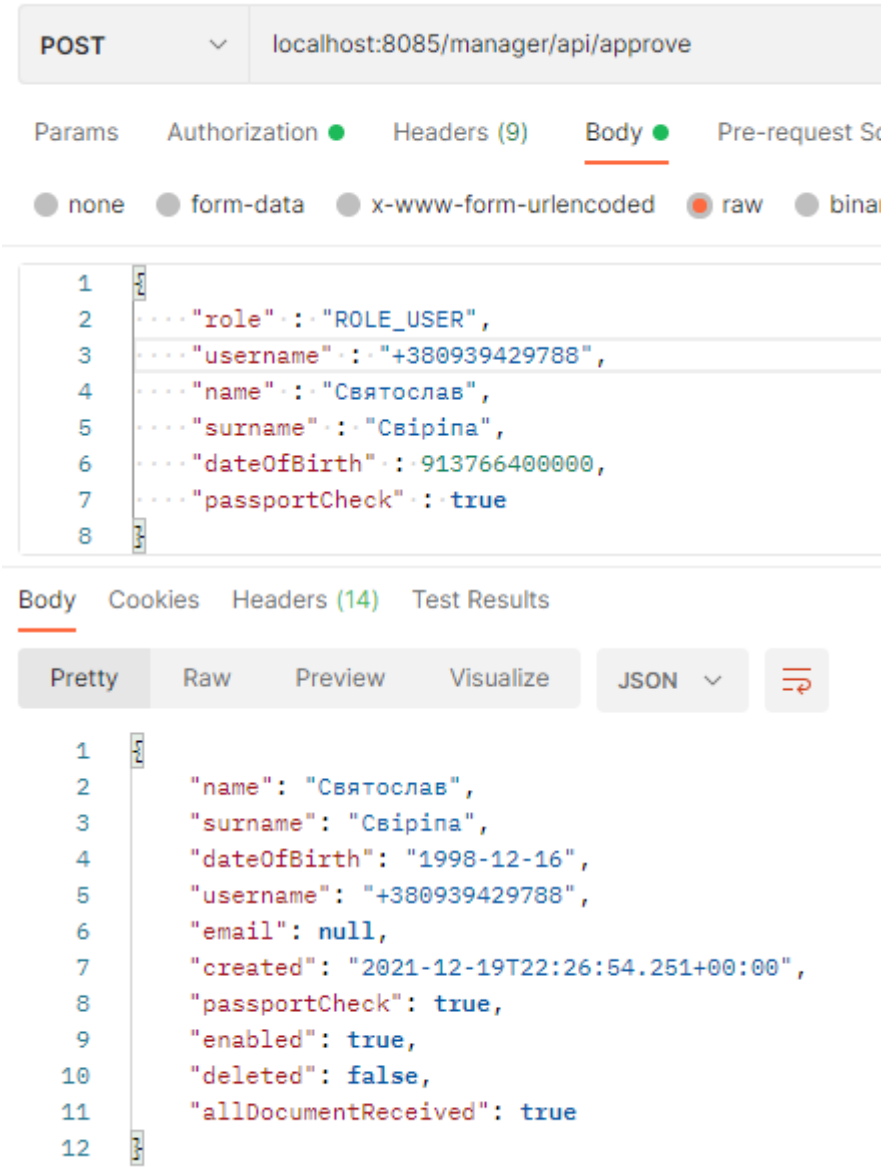


Рисунок 3.16 — Вхід в систему мікробанку

У відповідь на цей запит менеджер отримає всі поля які доступні для його перегляду.

Після того як менеджер перевірив документи користувач отримає смс сповіщення про зміну статусу його особистого кабінету в мікробанку. Тепер

користувач може створити картки та переводити кошти між своїми або іншими картками чи рахунками цього чи іншого банку.

Доступні для вибору три валюти: євро, долар та гривня, а також п’ять типів карток: чорна, платинум, біла, залізного банку та валютна. Щоб створити нову картку потрібно вибрати валюту та тип картки після цього надіслати запит на сервер як показано на рисунку 3.17.

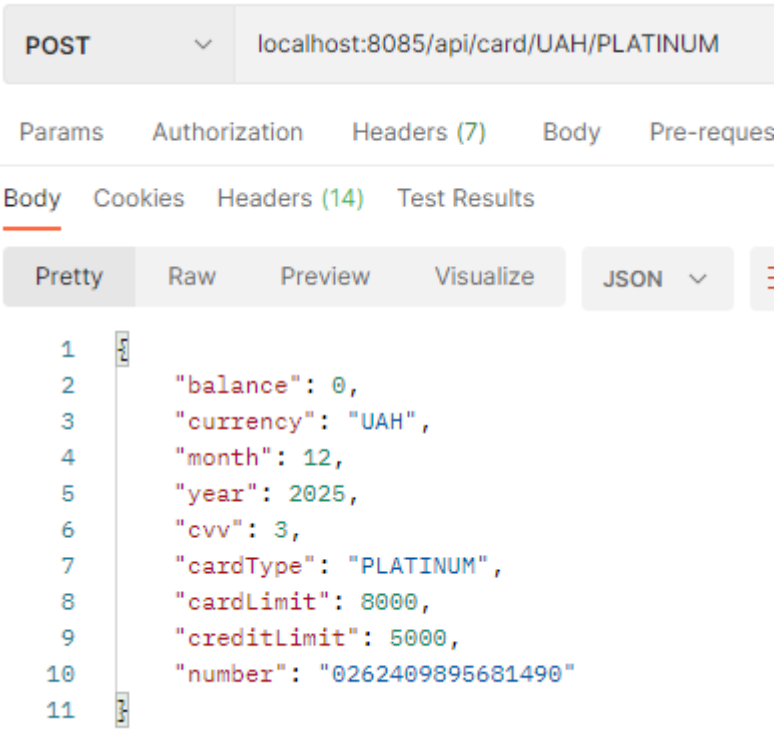


Рисунок 3.17 — Створення картки мікробанку

Після того як була створена картка користувач може отримувати та відправляти платежі. Підтримується три типи отримання платежів такі як: готівкою, з іншої картки та гугл пей. Для переказу коштів підтримується два типи платежів: на іншу картку та поповнення мобільного телефону.

Для здійснення транзакції як приклад отримання платежу за допомогою готівки, потрібно виконати декілька речей: вказати тип транзакції, суму переказу, валюту та картку отримувача як показано на рисунку 3.18. Після виконання даного запиту на мікросервіс переводу коштів надійде запит на поповнення катки, мікросервіс створить транзакцію з в

очікувані та після підтвердження переведе в отримано та зарахує кошти на рахунок отримувача. Переведення коштів, перегляд рахунку на картці та транзакції які були проведені на картці показано на рисунку 3.18

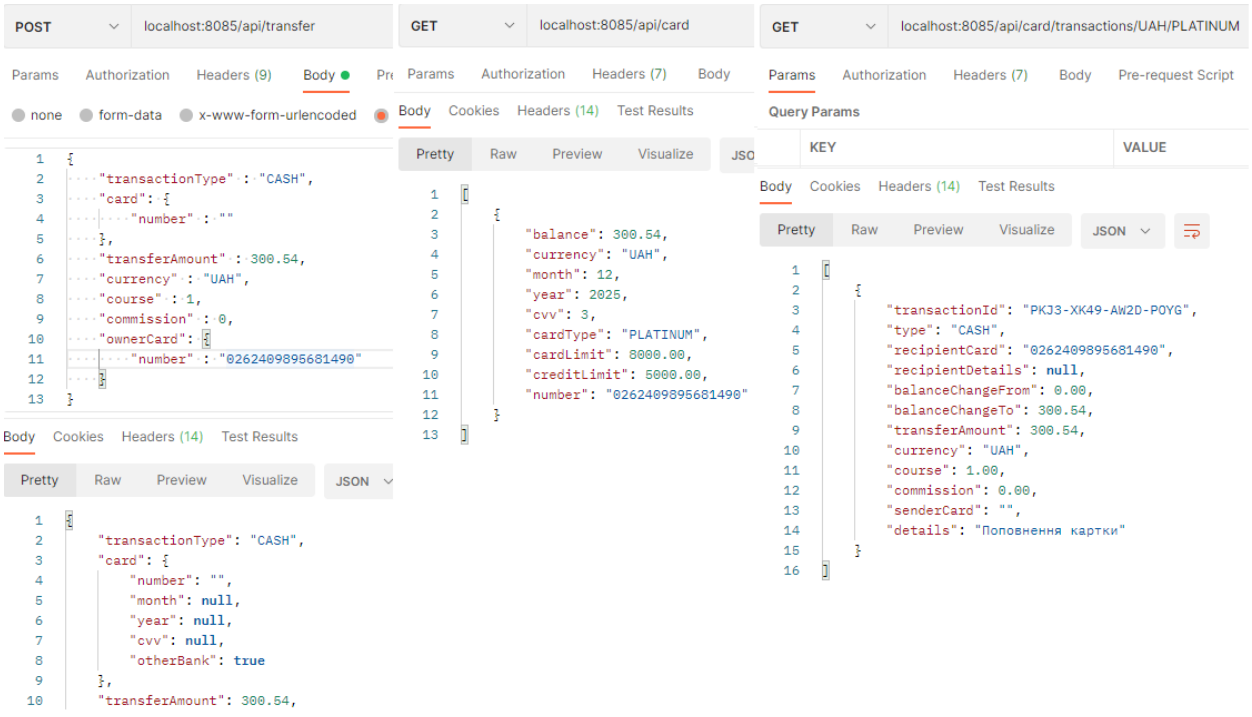


Рисунок 3.18 — Переказ коштів на картку мікробанку

4 РОЗРАХУНОК ЕКОНОМІЧНОЇ ДОЦІЛЬНОСТІ СТВОРЕННЯ ПРОГРАМНОГО ЗАСОБУ МІКРОБАНКУ НА ОСНОВІ ХМАРНИХ ТЕХНОЛОГІЙ

Дослідження завжди дорогі. Ці витрати на виробництво та реалізацію товарів необхідно постійно зменшувати, оскільки це прогрес будь-якого виробництва. На основі економічних розрахунків можна продемонструвати рентабельність та ефективність впровадження результатів досліджень у виробництво, тобто комерціалізація наукових досліджень. Дана магістерська кваліфікаційна робота відноситься до розряду прикладної науково-технічної роботи. Прогнозується виведення науково-технічної розробки на ринок із залученням потенційним інвестора. Дана послідовність приведена на рисунку 4.1.

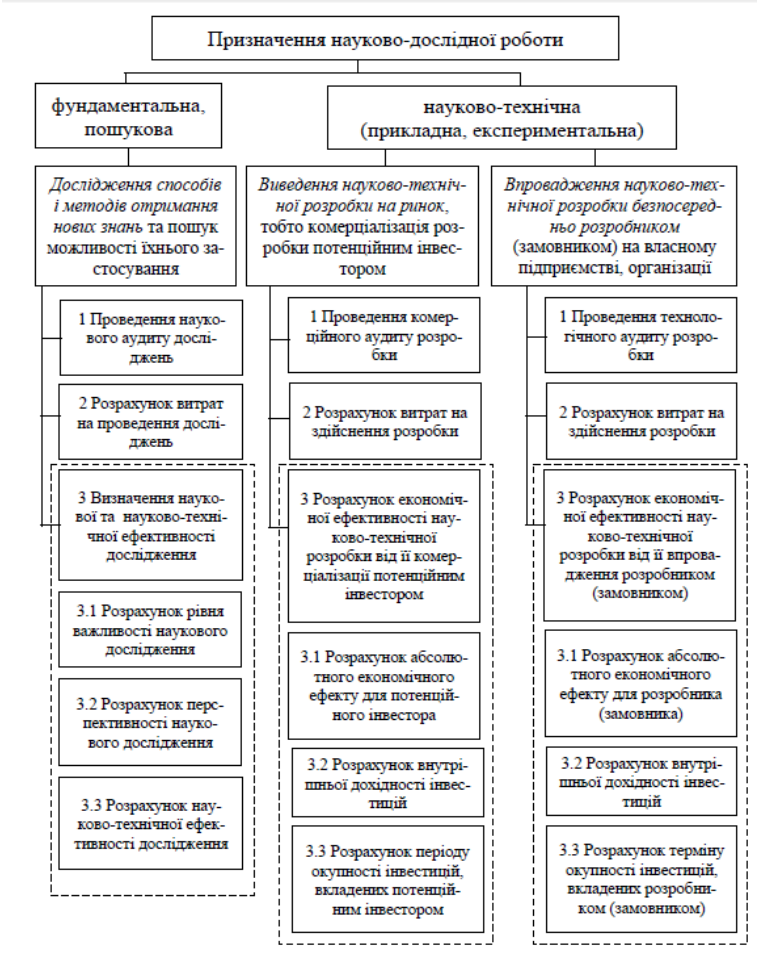


Рисунок 4.1 — Складові економічної частини магістерської кваліфікаційної роботи

Економічна частина цієї магістерської роботи буде поділена на такі елементи. Усі подальші економічні розрахунки будуть розглянуті у згаданих розділах економічної частини.

4.1 Проведення комерційного та технологічного аудиту науково-технічної розробки

Метою оцінки потенціалу комерційного розвитку є оцінка потенціалу комерційного розвитку, що випливає з науково-технічних досліджень. За результатами оцінки робляться висновки про напрямки (особливості) організації в майбутньому її впровадження з урахуванням встановленої оцінки. Комерційний потенціал інвестицій буде оцінюватись відповідно до дванадцяти критеріїв, наведених у таблиці 4.1.

Таблиця 4.1 — Оцінювання комерційного потенціалу розробки

Критерії оцінювання та бали (за 5-бальною шкалою)					
Критерій	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	Багато 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

Продовження таблиці 4.1

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
Ринкові перспективи					
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
Практична здійсненність					
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
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.	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси.Джерела фінансування є	Потрібні незначніфінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років.Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років.Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років.Термін окупності інвестицій менше 3-х років

Закінчення таблиці 4.1

12	Необхідно регламентні документи та велика кількість дозвільних документів на виробництво продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	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

На основі таблиці різні експерти, у нашому випадку керівник магістерської роботи та інші незалежні експерти з різних профілів програмування, визначають різні результати. Результати цієї оцінки комерційного потенціалу узагальнено у таблиці 4.2.

Таблиця 4.2 — Результати оцінювання комерційного потенціалу розробки

Критерії	Експерт (ПІБ, посада)		
	1 Кисюк Д.В. сенйор джава розробник	2 Крупельницький Л.В., к.т.н., доц. кафедри ОТ	3 Черняк О.І., к.т.н., доц. кафедри ОТ
	Бали:		
1.Технічна здійсненність концепції	3	2	3
2.Ринкові переваги (наявність аналогів)	3	2	4
3. Ринкові переваги (ціна продукту)	3	3	3
4.Ринкові переваги (технічні властивості)	2	3	3
5. Ринкові переваги (експлуатаційні витрати)	2	3	3
6. Ринкові перспективи (розмір ринку)	3	3	4
7. Ринкові перспективи (конкуренція)	2	3	3
8. Практична здійсненність (наявність фахівців)	3	2	2
9. Практична здійсненність (наявність фінансів)	2	2	3

Закінчення таблиці 4.2

10.Практична здійсненність (необхідність нових матеріалів)	1	4	2
11. Практична здійсненність (термін реалізації)	3	4	3
12. Практична здійсненність (розробка документів)	2	3	3
Сума балів	$СБ_1 = 31$	$СБ_1 = 34$	$СБ_1 = 36$
Середньо-арифметична сума балів $СБ_С$	$СБ_С = \frac{\sum_1^3 СБ_i}{3} = \frac{31 + 34 + 36}{3} = 33{,}6$		

Відповідно до таблиці 4.2, а також відповідно до рекомендацій, наведених у таблиці 4.3, можна зробити висновок про рівень потенціалу комерційного розвитку.

Таблиця 4.3 — Рівні комерційного потенціалу 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

З урахуванням середніх арифметичних балів $СБ_С$ = 33,6, які були визначені експертами, можна зробити висновок, що рівень комерційного потенціалу цієї розробки буде вище середнього.

Програмний ресурс Raiffaisen було використано для порівняння властивостей. Це програмне забезпечення має більш широкий спектр застосування та дещо вищий відсоток злагодженої роботи.

Нова розробка, навпаки, є вузькоспрямованою і тому має більшу швидкість. Окрім того, інтерфейс не перевантажений зайвою інформацією,

що значно полегшує роботу, а також має велику швидкодію і легке оновлення функціоналу.

Також, ще один аналог розробленого програмного забезпечення — Privat24, який призначений для роботи з різними платіжними сервісами.

Порівння розробки з її аналогами приведено в таблиці 4.4.

Таблиця 4.4 — Порівняння характеристик розробки із аналогом

Показники	Розробка	Аналог1	Аналог2
Функціонал	7	9	9
Швидкодія	9	7	9
Надійність	9	8	8
Метод розповсюдження	7	7	6
Інтерфейс, простота використання	8	8	7

Продукт буде просуватися за допомогою реклами в соціальних мережах, пошукових системах та багатьох інших джерелах Інтернету. Використовуючи аналітику цих сервісів, можна буде націлити рекламу на цільову групу захисників інформації.

Продукт також може бути використаний в банківській справі, оплати послуг в соціальних мережах, комп’ютерних ігор та інших сфер.

Новизна дослідження полягає в тому, що вперше буде запропоновано та впроваджено поєднання хмарних обчислень та банківської справи, що дозволило збільшити швидкість та надійність обробки платежів в реальному часі.

Виходячи з результатів цього порівняння, можна з упевненістю сказати, що новий дизайн є конкурентоспроможним, оскільки в деяких аспектах в ньому переважає один з найкращих аналогів на ринку.

Даний рівень було досягнуто за рахунок покращення та/або розширення функціональних можливостей нової науково-технічної розробки порівняно з аналогічними розробками, існуючими в цей час на ринку.

4.3 Розрахунок витрат на здійснення науково-дослідної роботи

У магістерській роботі розглядається програмне забезпечення для розпізнавання осіб на основі зображення особи, тому значну частину витрат складають витрати на розробку, а не на виробництво та відтворення. Відповідно, є певна специфіка розрахунків.

4.3.1 Витрати на оплату праці

Основна заробітна плата розробників, що працюють над проектом, визначена у формулі:

$$З_о = \sum_{i=1}^{k} \frac{M_{ni} \cdot t_i}{T_p}, \quad (4.1)$$

k — кількість посад дослідників, залучених до процесу досліджень;

M_{ni} — місячний посадовий оклад конкретного дослідника, грн;

T_p — середня кількість робочих днів в місяці, $T_р$= 22 дні;

t_i— кількість днів роботи конкретного дослідника, днів;

Над створенням розробки працював менеджер проекту та інженер програмного забезпечення, тому ми виконаємо для них усі необхідні розрахунки, і вносимо їх до таблиці 4.5:

$$Зо.к. = \frac{25000 \cdot 10}{22} = 11\,363{,}63(грн).$$

$$Зо.в. = \frac{20000 \cdot 44}{22} = 40\,000\ (грн).$$

Витрати на основну заробітну плату робітників за відповідними найменуваннями робіт відсутні, тобто $З_р = 0$.

Таблиця 4.5 — Витрати на заробітну плату дослідників

Найменування посади	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник проекту	25000	1 136,36	10	11 363,6
Старший інженер-програміст	20000	909,09	44	39 999,96
Всього				51 363,56

Додаткова винагорода ($З_{дод.}$) усіх розробників та працівників, які брали участь у цьому етапі роботи, обчислюється як 10% від суми основної заробітної плати дослідників та робітників за формулою:

$$З_{дод.} = (З_о + З_р) \cdot \frac{Н_{дод.}}{100\%}, \quad (4.2)$$

$Н_{дод.}$ — норма нарахування додаткової заробітної плати.

$$З_{дод.к.} = \frac{10 \cdot 11\,363{,}6}{100} = 1\,136{,}36\ (грн),$$

$$З_{дод.в.} = \frac{10 \cdot 39\,999{,}96}{100} = 3\,999{,}99\ (грн),$$

$$З_{дод} = З_{дод.к.} + З_{дод.в.} = 5\,136{,}35 (грн).$$

4.3.2 Відрахування на соціальні заходи

Заробітна плата робітників відсутня, тому $З_р = 0$. Нарахування на заробітну плату дослідників та нарахування на заробітну плату працівників, які брали участь у цьому етапі роботи, розраховується як 22% від суми

основної та додаткової заробітної плати дослідників і робітників за формулою:

$$З_н = (З_о + З_р + З_{дод}) \cdot \frac{Н_{зп}}{100\%} \quad (4.3)$$

$Н_{зп}$ — норма нарахування на заробітну плату.

$$З_н = (51\,363{,}56 + 0 + 5\,136{,}35) \cdot \frac{22\%}{100\%} = 12\,495{,}98 \text{ (грн.)}$$

4.3.3 Сировина та матеріали

Витрати на матеріали (М), у вартісному вираженні розраховуються окремо по кожному виду матеріалів за формулою:

$$М = \sum_{j=1}^{n} Н_j \cdot Ц_j \cdot К_j - \sum_{j=1}^{n} В_j \cdot Ц_{вj}, \quad (4.4)$$

де $Н_j$ — кількість матеріалу j-го виду, шт.;

n — кількість видів матеріалу.

$Ц_j$ — ціна матеріалу j-го виду, грн;

$К_j$ — коефіцієнт транспортних витрат, $К_j$ = 1,15;

$В_j$ — маса відходів j-го найменування, кг;

$Ц_{вj}$ — вартість відходів j-го найменування, грн/кг.;

Результати розрахунків занесено до таблиці 4.6.

4.3.4 Розрахунок витрат на комплектуючі

Оскільки кінцевий продукт, який ми створюємо — це програмний інструмент, це не спричиняє жодних витрат на компоненти та $К_в = 0$.

Таблиця 4.6 — Витрати на матеріали

Найменування комплектуючих	Ціна за 1 штуку, грн	Кількість матеріалу, штук	Величина відходів, кг	Ціна відходів, грн/кг	Вартість витраченого матеріалу, грн.
Ручка	30,00	1	0,06	1,80	29,89
Карта пам’яті	850,00	1	0	0,00	850,00
Пачка офісного папіру	115,00	1	0,5	57,50	86,25
Azure cloud	1500,00	1	0	0	1500,00
Всього (з урахуванням транспортних витрат)					2 836,06

4.3.5 Спецустаткування для наукових (експериментальних) робіт

Спецустаткування для проведення експериментальних робіт по створенню програмного продукту по мікробанку на основі хмарних технологій не має потреби залучати.

4.3.6 Програмне забезпечення для наукових (експериментальних) робіт

Програмне забезпечення для створення програмного продукту мікробанку на основі хмарних технологій використовується таке, що є у вільному розповсюдженні, тому витрати на придбання такого забезпечення відсутні. Це мова програмування Java та програмні продукти із бібліотек із відкритим кодом Apache, Spring.

4.3.7 Амортизація обладнання, програмних засобів та приміщень

В спрощеному вигляді амортизаційні відрахування по кожному виду обладнання, приміщень та програмному забезпеченню тощо можуть бути розраховані з використанням прямолінійного методу амортизації за формулою:

$$А_{обл} = \frac{Ц_б}{Т_в} \cdot \frac{t_{вик}}{12,} \qquad (4.5)$$

Таблиця 4.7 — Амортизаційні відрахування по кожному виду обладнання

Найменування обладнання	Балансова вартість, грн.	Строк корисного використання, років	Термін використання, місяців.	Амортизаційні відрахування, грн
ЕОМ	15000	2	2	1 250
Приміщення	190000	20	2	1 583,32
Всього				2 833,32

4.3.8 Паливо та енергія для науково-виробничих цілей

Витрати на силову електроенергію ($В_е$) розраховують за формулою:

$$В_е = \sum_{i=1}^{n} \frac{W_{yi} \cdot t_i \cdot Ц_е \cdot К_{впі}}{\eta_i}, \quad (4.6)$$

W_{yi} — встановлена потужність обладнання на певному етапі розробки, кВт;

t_i — тривалість роботи обладнання на етапі дослідження, год;

$Ц_е$ — вартість 1 кВт-години електроенергії, грн; (вартість електроенергії визначається за даними енергопостачальної компанії), $Ц_е = 4{,}62$ [];

$К_{впі}$ — коефіцієнт, що враховує використання потужності, $К_{впі}$ <1; обираємо $К_{впі}$= 0,7;

η_i — коефіцієнт корисної дії обладнання, η_i<1.

$$В_е = \sum_{i=1}^{1} \frac{0{,}6 \cdot 352 \cdot 4{,}62 \cdot 0{,}7}{0{,}8} = 853{,}77 \text{ (грн.)},$$

Проведені розрахунки необхідно звести до таблиці 4.8.

Таблиця 4.8 — Витрати на електроенергію

Найменування обладнання	Встановлена потужність, кВт	Тривалість роботи, год	Сума, грн
ЕОМ	0,6	352	853,77
Всього			853,77

4.3.9 Службові відрядження

Під час розробки програмного забезпечення відрядження штатних працівників, працівників організацій, які працюють за договорами цивільно-правового характеру, магістрів, зайнятих розробленням досліджень, відрядження, пов'язані з проведенням випробувань машин та приладів, а також витрати на відрядження на наукові з'їзди, конференції, наради, пов'язані з виконанням конкретних досліджень, не плануються.

4.3.10 Витрати на роботи, які виконують сторонні підприємства, установи і організації

Витрати за статтею «Витрати на роботи, які виконують сторонні підприємства, установи і організації» не плануються, так як у цьому не має потреби.

4.3.11 Інші витрати

Витрати за статтею «Інші витрати» розраховуються як 50…100% від суми основної заробітної плати дослідників та робітників за формулою:

$$I_в = (З_о + З_р) \cdot \frac{Н_{ів}}{100\%} \qquad (4.7)$$

$Н_{ів}$ — норма нарахування за статтею «Інші витрати».

$$I_{в.к.} 11363{,}6 \cdot \frac{50\%}{100\%} = 5681{,}8(\text{грн.}),$$

$$I_{в.в.} = 39999{,}96 \cdot \frac{50\%}{100\%} = 19\,999{,}98\ (\text{грн.}),$$

$$I_в = I_{в.к.} + I_{в.в.} = 25\,681{,}78\ (\text{грн.}).$$

4.3.12 Накладні (загальновиробничі) витрати

Витрати за статтею «Накладні (загальновиробничі) витрати» розраховуються як 100…150% від суми основної заробітної плати дослідників та робітників за формулою:

$$В_{нзв} = (З_о + З_р) \cdot \frac{Н_{нзв}}{100\%} \quad (4.8)$$

$Н_{нзв}$ — норма нарахування за статтею «Накладні (загальновиробничі) витрати».

Беремо норму нарахування 100%.

$$В_{нзв.к.} = 11363{,}6 \cdot \frac{100\%}{100\%} = 11363{,}6(грн.),$$

$$В_{нзв.в.} = 39999{,}96 \cdot \frac{100\%}{100\%} = 39999{,}96(грн.),$$

$$В_{нзв} = В_{нзв.к.} + В_{нзв.в.} = 51\,363{,}56\,(грн.).$$

Витрати на проведення науково-дослідної роботи розраховуються як сума всіх попередніх статей витрат за формулою:

$$В_{заг} = З_о + З_р + З_{дод} + З_н + М + К_в + В_{спец} + В_{прг} + А_{обл} + В_е + В_{св} + В_{сп} + І_в + В_{нзв}. \quad (4.9)$$

У нашому випадку:

$$З_р = 0, К_в = 0, В_{спец} = 0,\ В_{прг} = 0, В_{св} = 0, В_{сп} = 0, \text{тому отримаємо:}$$

$$В_{заг} = З_о + З_{дод} + З_н + М + А_{обл} + В_е + І_в + В_{нзв}$$
$$= 51\,363{,}56 + 5\,136{,}35 + 12\,495{,}98 + 2\,836{,}06 + 2\,833{,}32 + 853{,}77$$
$$+ 25\,681{,}78 + 51\,363{,}56 = 152\,564{,}38(грн).$$

Загальні витрати ЗВ на завершення науково-дослідної (науково-технічної) роботи та оформлення її результатів розраховуються за формулою:

$$ЗВ = \frac{В_{заг}}{\eta}, \quad (4.10)$$

η — коефіцієнт, який характеризує етап (стадію) виконання науково-дослідної роботи. Обираємо його 0,5.

$$ЗВ = \frac{152\,564{,}38}{0{,}5} = 305\,128{,}76 \text{ (грн)}.$$

4.4 Розрахунок економічної ефективності науково-технічної розробки за її можливої комерціалізації потенційним інвестором

Розробка чи суттєве вдосконалення програмного засобу (програмного забезпечення, програмного продукту) для використання масовим споживачем.

Для всіх наведених випадків можливе збільшення чистого прибутку у потенційного інвестора $\Delta П_i$ для кожного із років, протягом яких очікується отримання позитивних результатів від можливого впровадження та комерціалізації науково-технічної розробки, розраховується за формулою:

$$\Delta П_i = (\pm \Delta Ц_o \cdot N \cdot Ц_o \cdot \Delta N)_i \cdot \lambda \cdot \rho \cdot \left(1 - \frac{\vartheta}{100}\right) \qquad (4.11)$$

$\pm \Delta Ц_o$ — зміна основного якісного показника від впровадження результа-тів науково-технічної розробки в аналізованому році;

N — основний кількісний показник, який визначає величину попиту на аналогічні чи подібні розробки у році до впровадження результатів нової науково-технічної розробки;

$Ц_o$ — основний якісний показник, який визначає ціну реалізації нової на-уково-технічної розробки в аналізованому році, $Ц_o = Ц_б \pm \Delta Ц_o$;

$Ц_б$ — основний якісний показник, який визначає ціну реалізації існуючої (базової) науково-технічної розробки у році до впровадження результатів;

ΔN — зміна основного кількісного показника від впровадження результа-тів науково-технічної розробки в аналізованому році;

Λ — коефіцієнт, який враховує сплату потенційним інвестором податку на додану вартість. У 2021 році ставка податку на додану вартість становить 20%, а коефіцієнт Λ=0,8333;

ρ — коефіцієнт, який враховує рентабельність інноваційного продукту (послуги). Рекомендується брати ρ=0,26;

ϑ — ставка податку на прибуток, який має сплачувати потенційний інвестор, у 2021 році ϑ=18%.

В результаті впровадження результатів наукових розробок поліпшується якість програмного забезпечення, що дозволяє подорожчати за його впровадження, а кількість потенційних користувачів ресурсу збільшиться — у перший рік — на 200 одиниць, на другий рік — ще на 350 одиниць, на третій рік — ще 800 штук.

Ми прогнозуємо щорічний приріст чистого прибутку компанії від впровадження результатів наукових розробок щодо вихідного стану.

Збільшення чистого прибутку підприємства $\Delta П_i$ за перший рік складе:

$$\Delta П_1 = [1800 \cdot 1 + (3500 + 1800) \cdot 200] \cdot 0,8333 \cdot 0,26 \cdot \left(1 - \frac{18\%}{100\%}\right) = 188\,571 \text{ (грн)}.$$

Збільшення чистого прибутку компанії $\Delta П_i$ на другий рік (порівняно з базовим, тобто роком, що передує впровадженню результатів наукових досліджень) складе:

$$\Delta П_2 = [1800 \cdot 1 + (3500 + 1800) \cdot (200 + 350)] \cdot 0,8333 \cdot 0,26 \cdot \left(1 - \frac{18\%}{100\%}\right) = 518\,197,40 \text{ (грн)}.$$

Збільшення чистого прибутку підприємства $\Delta П_i$ на третій рік складе:

$$\Delta П_3 = [1800 \cdot 1 + (3500 + 1800) \cdot (200 + 350 + 800)] \cdot 0,8333 \cdot 0,26 \cdot \left(1 - \frac{18\%}{100\%}\right) = 1\,271\,473,93 \text{ (грн)}.$$

Далі розраховують приведену вартість збільшення всіх чистих прибутків ПП, що їх може отримати потенційний інвестор від можливого впровадження та комерціалізації науково-технічної розробки:

$$ПП = \sum_{i=1}^{T} \frac{\Delta П_i}{(1+\tau)^t}, \quad (4.12)$$

$\Delta П_i$ — збільшення чистого прибутку у кожному з років, протягом яких виявляються результати впровадження науково-технічної розробки, грн;

T — період часу, протягом якого очікується отримання позитивних результатів від впровадження та комерціалізації науково-технічної розробки, роки;

τ — ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні, τ = 0,05…0,15. Обираємо τ 0,1;

t — період часу (в роках) від моменту початку впровадження науково-технічної розробки до моменту отримання потенційним інвестором додаткових чистих прибутків у цьому році.

$$ПП = \frac{188\,571}{(1+0{,}1)^1} + \frac{518\,197{,}40}{(1+0{,}1)^2} + \frac{1\,271\,473{,}93}{(1+0{,}1)^3}$$
$$= 171\,428{,}18 + 428\,262{,}31 + 955\,277{,}18 = 1\,554\,967{,}67 \text{ (грн.)}$$

Далі розраховують величину початкових інвестицій PV, які потенційний інвестор має вкласти для впровадження і комерціалізації науково-технічної розробки. Для цього можна використати формулу:

$$PV = k_{інв} \cdot ЗВ, \quad (4.13)$$

де $k_{інв}$ — коефіцієнт, що враховує витрати інвестора на впровадження науково-технічної розробки та її комерціалізацію. Це можуть бути витрати на підготовку приміщень, розробку технологій, навчання персоналу, маркетингові заходи тощо; зазвичай $k_{інв}$=2...5, але може бути і більшим. Обираємо даний коефіцієнт 2;

ЗВ — загальні витрати на проведення науково-технічної розробки та оформлення її результатів, грн.

$$PV = 2 \cdot 305\,128{,}76 = 610\,257{,}52 \text{ (грн.)}$$

Тоді абсолютний економічний ефект $E_{абс}$ або чистий приведений дохіддля потенційного інвестора від можливого впрова-дження та комерціалізації науково-технічної розробки становитиме:

$$E_{абс} = ПП - PV \qquad (4.14)$$

ПП — приведена вартість зростання всіх чистих прибутків від можливого впровадження та комерціалізації науково-технічної розробки, грн;

PV — теперішня вартість початкових інвестицій, грн.

$$E_{абс} = 1\,554\,967{,}67 - 610\,257{,}52 = 944\,710{,}15 \text{ (грн.)}$$

Внутрішня економічна дохідність інвестицій $E_в$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки, розраховується за формулою:

$$E_в = \sqrt[Т_ж]{1 + \frac{E_{абс}}{PV}} - 1, \qquad (4.15)$$

$E_{абс}$ — абсолютний економічний ефект вкладених інвестицій, грн;

PV – теперішня вартість початкових інвестицій, грн;

$Т_ж$ — життєвий цикл науково-технічної розробки, тобто час від початку її розробки до закінчення отримування позитивних результатів від її впро-вадження, роки.

$$E_в = \sqrt[3]{1 + \frac{944\,710{,}15}{610\,257{,}52}} - 1 = 0{,}366$$

Далі визначають бар'єрну ставку дисконтування τ_{min}, тобто мінімальну внутрішню економічну дохідність інвестицій, нижче якої кошти у впровадження науково-технічної розробки та її комерціалізацію вкладатися не будуть.

Мінімальна внутрішня економічна дохідність вкладених інвестицій $\tau_{мін}$визначається за формулою:

$$\tau_{мін} = d + f, \quad (4.16)$$

d— середньозважена ставка за депозитними операціями в комерційних банках; в 2021 році в Україні **d**=0,9...0,12. Обираємо **d** 0,1;

f — показник, що характеризує ризикованість вкладення інвестицій; зазвичай величина **f**=0,05...0,5, але може бути і значно вищою. Обираємо 0,19;

$$\tau_{мін} = d + f = 0,1 + 0,19 = 0,29\%$$

Величина $E_в > \tau_{мін}$, отже інвестор може бути зацікавлений у фінансуванні цього дослідження.

Далі розраховуємо період окупності інвестицій $T_{ок}$, які можуть бути вкладені потенційним інвестором у впровадження та комерціалізацію науково-технічної розробки:

$$T_{ок} = \frac{1}{E_в}, \quad (4.17)$$

$E_в$ — внутрішня економічна дохідність вкладених інвестицій.

$$T_{ок} = \frac{1}{0,366} = 2,73 \text{ року}$$

Оскільки $T_{ок}$= 2,73 року тоді розвиток доречний.

ВИСНОВКИ

В першому розділі було проаналізовано вибрану мову програмування Java, фреймворк Spring та допоміжні бібліотеки для розробки серверної частини даного програмного забезпечення. Для розробки візуальної частини було використано мову програмування JavaScript та залежні бібліотеки.

В другому розділі було розроблено архітектурний дизайн мікросервісів, визначено бізнес межі даного програмного забезпечення та інтеграція між мікросервісами. Було створено скрипт для автоматизованого розгортання даного проекта в контейнері.

В третьому розділі було розроблено програмне забезпечення на основі мікросервісів, описано кроки розробки діаграми бази даних налаштування зв’язків між мікросервісами та наведено приклад роботи даного програмного забезпечення.

У четвертому розділі магістерської кваліфікаційної роботи було виконано обґрунтування доцільності розробки нового наукового вирішення представленої проблеми по мікробанку на основі хмарних технологій, здійснено розрахунок потрібних економічних затрат, що необхідні для реалізації запропонованих засобів і позначено комерційні переваги впровадження створеного програмного продукту.

Завдяки створеному сайту з’явилася можливість більш ефективно оновлювати додаток підвищена продуктивність та легка адаптація до великих навантажень.

Таким чином, поставлена мета була досягнута.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЯ

1. Оцифровка. URL: https://ru.wikipedia.org/wiki/Оцифровка (дата звернення: 10.07.2021);
2. Маштабування. URL: https://uk.wikipedia.org/wiki/Масштабовність (дата звернення: 10.07.2021);
3. Поставщик повідомлень. URL: https://ru.wikipedia.org/wiki/Брокер_сообщений (дата звернення: 10.07.2021);
4. Паттерн. URL: https://ru.wikipedia.org/wiki/Паттерн (дата звернення: 10.07.2021);
5. Java. URL: https://uk.wikipedia.org/wiki/Java (дата звернення: 10.07.2021);
6. Sun Microsystems. URL: https://uk.wikipedia.org/wiki/Sun_Microsystems (дата звернення: 10.07.2021);
7. Java ee. URL: https://uk.wikipedia.org/wiki/Java_EE (дата звернення: 10.07.2021);
8. Java se. URL: https://uk.wikipedia.org/wiki/Java_SE (дата звернення: 10.07.2021);
9. WWW. URL: https://uk.wikipedia.org/wiki/Всесвітнє_павутиння (дата звернення: 10.07.2021);
10. Spring. URL: https://uk.wikipedia.org/wiki/Spring_Framework (дата звернення: 20.10.2021);
11. .NET. URL: https://uk.wikipedia.org/wiki/.NET_Framework (дата звернення: 20.10.2021);
12. Framework. URL: https://ru.wikipedia.org/wiki/Фреймворк (дата звернення: 20.10.2021);
13. Spring. URL: https://www.springframework.net/ (дата звернення: 20.10.2021);
14. CSS. URL: https://uk.wikipedia.org/wiki/CSS (дата звернення: 20.10.2021);
15. HTML. URL: https://uk.wikipedia.org/wiki/HTML (дата звернення:

20.10.2021);

16. XML. URL: https://uk.wikipedia.org/wiki/XML (дата звернення: 20.10.2021);

17. Script. URL: https://ru.wikipedia.org/wiki/Сценарный_язык (дата звернення: 20.10.2021);

18. Hibernate. URL: https://ru.wikipedia.org/wiki/Hibernate_(библиотека) (дата звернення: 20.10.2021);

19. SQL. URL: https://uk.wikipedia.org/wiki/SQL (дата звернення: 10.07.2021);

20. JDBC. URL: https://uk.wikipedia.org/wiki/Java_Database_Connectivity (дата звернення: 20.10.2021);

21. POJO. URL: https://uk.wikipedia.org/wiki/POJO (дата звернення: 20.10.2021);

22. JWT. URL: https://uk.wikipedia.org/wiki/JSON_Web_Token (дата звернення: 20.10.2021);

23. UNIX. URL: https://uk.wikipedia.org/wiki/Час_Unix (дата звернення: 20.10.2021);

24. Java Script. URL: https://ru.wikipedia.org/wiki/JavaScript (дата звернення: 20.10.2021);

25. ECMAScript. URL: https://uk.wikipedia.org/wiki/ECMAScript (дата звернення: 20.10.2021);

26. GitHub. URL: https://uk.wikipedia.org/wiki/GitHub (дата звернення: 20.10.2021);

27. Ruby. URL: https://uk.wikipedia.org/wiki/Ruby_on_Rails (дата звернення: 20.10.2021);

28. Erlang. URL: https://uk.wikipedia.org/wiki/Erlang (дата звернення: 20.10.2021);

29. SSL. URL: https://uk.wikipedia.org/wiki/SSL (дата звернення: 10.11.2021);

30. SSH. URL: https://uk.wikipedia.org/wiki/SSH (дата звернення:

10.11.2021);

31. IDEA. URL: https://uk.wikipedia.org/wiki/IntelliJ_IDEA (дата звернення: 10.11.2021);

32. Maven. URL: https://uk.wikipedia.org/wiki/Apache_Maven (дата звернення: 10.11.2021);

33. JUnit. URL: https://uk.wikipedia.org/wiki/JUnit (дата звернення: 10.11.2021);

34. UML. URL: https://uk.wikipedia.org/wiki/Unified_Modeling_Language (дата звернення: 10.11.2021);

35. PostgreSQL. URL: https://uk.wikipedia.org/wiki/PostgreSQL (дата звернення: 10.11.2021);

36. React. URL: https://uk.wikipedia.org/wiki/React (дата звернення: 10.11.2021);

37. Redux. URL: https://uk.wikipedia.org/wiki/Redux (дата звернення: 10.11.2021);

38. GraphQL. URL: https://uk.wikipedia.org/wiki/GraphQL (дата звернення: 10.11.2021);

39. DOM. URL: https://coderoad.ru/21965738/Что-такое-Virtual-DOM (дата звернення: 10.11.2021);

40. Docker. URL: https://uk.wikipedia.org/wiki/Docker (дата звернення: 10.11.2021);

41. Docker hub. URL: https://hub.docker.com/ (дата звернення: 10.11.2021);

42. OpenAPI. URL: https://uk.wikipedia.org/wiki/OpenAPI (дата звернення: 10.11.2021);

43. Ефективна Java – Джошуа Блох, 2013. 952 ст.

44. Clean Code: A Handbook of Agile Software Craftsmanship – Robert C. Martin, 2013. 212 ст.

45. Head First. Паттерни проектування - Ерік Фрімен, Елізабет Робсон, 2018. 582 ст.

46. Spring в действии – Крейг Уоллс, 2018. 712 ст.

47. Test Driven: TDD and Acceptance TDD для Java Developers – Lasse Koskela, 2012. 378 ст.

48. Алгоритмы на Java – Роберт Седжвик, Кевин Уэйн, 2018. 252 ст.

49. Head First Object-Oriented Analysis and Design – Brett D. McLaughlin, 2016. 444 ст.

50. Java. Полное руководство – Герберт Шилдт, 2015. 752 ст.

51. SCJP Sun Certified Programmer for Java 6 Exam 310-065, 2015. 352 ст.

52. особливості розробки та взаємодії з базою даних мовою Java https://conferences.vntu.edu.ua/index.php/all-fitki/all-fitki-2020/paper/view/9145

ДОДАТОК А

Міністерство освіти та науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра обчислювальної техніки

ЗАТВЕРДЖУЮ
Завідувач кафедри ОТ
_____проф., д.т.н. О. Д. Азаров
«___» _____ 2021 р.

ТЕХНІЧНЕ ЗАВДАННЯ

на виконання магістерської кваліфікаційної роботи
«Веб-додаток мікробанку на основі хмарних технологій»
08-23.МКР.025.00.000 ТЗ

Науковий керівник: д.т.н., професор
_____ Азаров О. Д.
Магістрант групи 2КІ-20м
_____ Свіріпа С.М.

Вінниця 2021

1 Підстава для виконання магістерської кваліфікаційної роботи (МКР)

1.1 Актуальність даного дослідження полягає у тенденції сучасних технологій до віртуалізації, тобто всі матеріали та накопичені данні переводяться у цифровий формат та зберігаються в хмарних середовищах. Це дозволяє отримати доступ до них в будь-якій точці світу. Те ж саме відноситься і до фінансової сфери. Створюються віртуальні гроші, кошти мігрують з фізичного обігу до мобільного банкінгу, що дозволяє використовувати їх за допомогою будь якого пристрою у якого є доступ в глобальну мережу.

1.2 Наказ про затвердження теми магістерської кваліфікаційної роботи.

2 Мета і призначення МКР

2.1 Мета магістерської роботи полягає можливості покращеного масштабування та модифікації.

2.2 Призначення розробки — виконання магістерської кваліфікаційної роботи.

3 Вихідні дані для виконання МКР

Виконати розробку програмного забезпечення для мікробанку на основі хмарних технологій провести його тестування. Схеми архітектури програмного забезпечення мікробанку на основі хмарних технологій, лістинги програми представити в додатках до роботи.

4 Вимоги до виконання МКР

МКР повинна задовольняти такі вимоги:

— запропонувати нову архітектуру для мікробанку на основі хмарних технологій;

— розробити архітектуру для мікробанку на основі хмарних технологій;

— вхідні дані — виклики кінцевих точок програмного забезпечення

для переводу коштів між рахунками в мікросервісному середовищі;

— результат роботи, а саме цифрове зображення, на якому представлені символи текстового документа.

5 Етапи МКР та очікувані результати наведені в таблиці А.1

Таблиця А.1 — Етапи виконання роботи

№	Назва етапу	Термін виконання		Очікувані результати
		початок	кінець	
1	Аналіз завдання. Вступ	07.09.21	9.09.21	Вступ
2	Аналіз літературних джерел для розпізнавання особи	10.09.21	16.09.21	розділ 1
3	Розробка технічного завдання	17.09.21	18.09.21	Технічне завдання
3	Розробка структури системи розпізнавання особи за зображенням обличчя	19.09.21	21.10.21	Розділ 2, розробка структури
4	Розробка програми, проектування програмного продукту	22.10.21	31.10.21	Розділ 3, розробка програми
5	Практична реалізація, результати.	01.11.21	16.11.21	Розділ 3
6	Розробка економічної частини	17.11.21	30.11.21	Розділ 4
7	Оформлення пояснювальної записки	01.12.21	15.12.21	ПЗ, презентація

6 Матеріали, що подаються до захисту МКР — пояснювальна записка МКР, ілюстративні та графічні матеріали, протокол попереднього захисту МКР на кафедрі, відзив наукового керівника, відзив рецензента, протоколи складання державних екзаменів, анотації до МКР українською та іноземною мовами, довідка про відповідність оформлення МКР діючим вимогам.

7 Порядок контролю виконання та захисту МКР

Виконання етапів розрахункової та графічної документації МКР контролюється науковим керівником згідно зі встановленими термінами. Захист МКР відбувається на засіданні Державної екзаменаційної комісії, затвердженою наказом ректора.

8 Вимоги до оформлення МКР

Вимоги викладені в ДСТУ 3008:2015 та положення ВНТУ про МКР-2021.

Технічне завдання до виконання отримав _____ Свіріпа С.М.

ДОДАТОК Б

Повний лістинг створеного ПЗ

AbstractModelService class

```
@Service
@AllArgsConstructor
public abstract class AbstractModelService {
   protected final ModelMapperServiceImpl mapper;
}
```

BaseDBOperation class

```
public interface BaseDBOperation<T, R> {
   R save(T t);
   R edit(T t);
   R findById(Long id);
   boolean delete(T t);
   boolean delete(Long id);
   Collection<R> findAll();
   Collection<R> findAll(Specification<T> specification);
}
```

CardServiceImpl class

```
package com.violence.domain.service;

import com.violence.domain.data.dto.CardDto;
import com.violence.domain.data.entity.Cards;
import com.violence.domain.data.entity.User;
import com.violence.domain.enums.CardType;
import com.violence.domain.enums.SupportedCurrency;
import com.violence.domain.repository.CardRepository;
import com.violence.domain.repository.UserRepository;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.sql.Timestamp;
import java.time.LocalDateTime;
import java.util.Collection;
import java.util.Optional;

import static com.violence.domain.exception.ExceptionHelper.*;
```

```
@Service
public class CardServiceImpl extends AbstractModelService implements
CardService {
    private final CardRepository cardRepository;
    private final UserRepository userRepository;
    private final DataGenerator dataGenerator;

    public CardServiceImpl(ModelMapperServiceImpl mapper, CardRepository
cardRepository, UserRepository userRepository, DataGenerator dataGenerator) {
        super(mapper);
        this.cardRepository = cardRepository;
        this.userRepository = userRepository;
        this.dataGenerator = dataGenerator;
    }

    @Override
    public CardDto save(Cards cards) {
        return mapper.toDto(cardRepository.save(cards), CardDto.class);
    }

    @Override
    public CardDto edit(Cards cards) {
        return null;
    }

    @Override
    public CardDto findById(Long id) {
        return null;
    }

    @Override
    public boolean delete(Cards cards) {
        cards.setEnabled(false);
        cardRepository.save(cards);
        return true;
    }

    @Override
    public boolean delete(Long id) {
        final Cards byId = cardRepository.getById(id);
        byId.setEnabled(false);
        cardRepository.save(byId);
        return false;
```

```
    }

    @Override
    public Collection<CardDto> findAll() {
        return null;
    }

    @Override
    public Collection<CardDto> findAll(Specification<Cards> specification) {
        return null;
    }

    @Override
    public Collection<CardDto> findByCardOwner(Long ownerId) {
        return mapper.toDto(cardRepository.findByOwnerId(ownerId),
CardDto.class);
    }

    @Override
    public CardDto findByCardNumber(String cardNumber) {
        return
mapper.toDto(cardRepository.findByNumber(cardNumber).orElseThrow(CARD_
NOT_FOUND), CardDto.class);
    }

    @Override
    public String findOwnerByCardNumber(String cardNumber) {
        return cardRepository.findByNumber(cardNumber)
                .map(cards ->
userRepository.findById(cards.getOwnerId()).map(User::getFullName).orElse(car
dNumber))
                .orElse(cardNumber);
    }

    @Override
    public CardDto createCard(SupportedCurrency supportedCurrency, CardType
cardType, Long userId) {
        if
(cardRepository.findByCurrencyAndCardTypeAndOwnerId(supportedCurrency,
cardType, userId).isPresent()) {
            throw ENTITY_ALREADY_EXIST.get();
        }
        final Timestamp now = Timestamp.valueOf(LocalDateTime.now());
        final User owner =
```

```
userRepository.findById(userId).orElseThrow(USER_NOT_FOUND);

        Cards cards = new Cards();
        cards.setBalance(BigDecimal.ZERO);
        cards.setCardLimit(BigDecimal.valueOf(8000));
        cards.setCardType(cardType);
        cards.setCreated(now);
        cards.setCreditLimit(BigDecimal.valueOf(5000));
        cards.setCurrency(supportedCurrency);
        cards.setEnabled(true);
        cards.setCvv("003");
        cards.setMonth(now.toLocalDateTime().getMonthValue());
        cards.setYear(now.toLocalDateTime().plusYears(4).getYear());
        cards.setPinCode(owner.getPassword());
        cards.setOwnerId(owner.getId());
        cards.setNumber(dataGenerator.randomCardNumber());

        return mapper.toDto(cardRepository.save(cards), CardDto.class);
    }

    @Override
    public Cards findByCurrencyAndCardTypeAndOwnerId(SupportedCurrency
currency, CardType cardType, Long ownerId) {
        return cardRepository.findByCurrencyAndCardTypeAndOwnerId(currency,
cardType, ownerId).orElseThrow(CARD_NOT_FOUND);
    }
}
```

ModelMapperServiceImpl class

```
package com.violence.domain.service;

import com.violence.domain.data.dto.AbstractDto;
import com.violence.domain.data.entity.AbstractEntity;
import lombok.AllArgsConstructor;
import org.modelmapper.ModelMapper;
import org.springframework.stereotype.Service;

import java.util.Collection;
import java.util.Objects;
import java.util.stream.Collectors;

@Service
@AllArgsConstructor
```

```
public class ModelMapperServiceImpl {
   private final ModelMapper modelMapper;

   public <T extends AbstractEntity, R extends AbstractDto> T toEntity(R
abstractDto, Class<T> aClass) {
      return Objects.isNull(abstractDto) ? null : modelMapper.map(abstractDto,
aClass);
   }

   public <T extends AbstractEntity, R extends AbstractDto> R toDto(T
abstractEntity, Class<R> aClass) {
      return Objects.isNull(abstractEntity) ? null :
modelMapper.map(abstractEntity, aClass);
   }

   public <T extends AbstractDto, R extends AbstractDto> R toDto(T abstractDto,
Class<R> aClass) {
      return Objects.isNull(abstractDto) ? null : modelMapper.map(abstractDto,
aClass);
   }

   public <T extends AbstractEntity, R extends AbstractDto> Collection<R>
toDto(Collection<T> abstractEntity, Class<R> aClass) {
      return Objects.isNull(abstractEntity) ? null : abstractEntity.stream().map(t ->
toDto(t, aClass)).collect(Collectors.toList());
   }

   public <T extends AbstractEntity, R extends AbstractDto> Collection<T>
toEntity(Collection<R> abstractDto, Class<T> aClass) {
      return Objects.isNull(abstractDto) ? null : abstractDto.stream().map(t ->
toEntity(t, aClass)).collect(Collectors.toList());
   }
}
```

UserServiceImpl class

```
package com.violence.domain.service;

import com.violence.domain.data.dto.ApproveDocumentDto;
import com.violence.domain.data.dto.UserDto;
import com.violence.domain.data.entity.CustomUserDetails;
import com.violence.domain.data.entity.PinCode;
import com.violence.domain.data.entity.User;
import com.violence.domain.repository.PinCodeRepository;
```

```
import com.violence.domain.repository.UserRepository;
import org.springframework.data.jpa.domain.Specification;
import org.springframework.security.core.userdetails.UserDetails;
import
org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;

import javax.annotation.PostConstruct;
import java.sql.Date;
import java.sql.Timestamp;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.Collection;

import static
com.violence.domain.exception.ExceptionHelper.USER_NOT_FOUND;
import static
com.violence.domain.exception.ExceptionHelper.WRONG_PIN_CODE;

@Service
public class UserServiceImpl extends AbstractModelService implements
UserService {
   private static final String FILL_AFTER_DOC_CHECK =
"FILL_AFTER_DOC_CHECK";
   private static final Date DEF_DATE_OF_BIRTH =
Date.valueOf(LocalDate.of(1900, 1, 1));
   private final UserRepository userRepository;
   private final BCryptPasswordEncoder passwordEncoder;
   private final PinCodeRepository pinCodeRepository;

   public UserServiceImpl(ModelMapperServiceImpl mapper, UserRepository
userRepository, BCryptPasswordEncoder passwordEncoder, PinCodeRepository
pinCodeRepository) {
       super(mapper);
       this.userRepository = userRepository;
       this.passwordEncoder = passwordEncoder;
       this.pinCodeRepository = pinCodeRepository;
   }

   @Override
   public UserDetails loadUserByUsername(String s) throws
UsernameNotFoundException {
       final User user =
```

```
userRepository.findByUsername(s).orElseThrow(USER_NOT_FOUND);
        return new CustomUserDetails(user);
    }

    @Override
    public UserDto createUser(User user) {
        final LocalDateTime now = LocalDateTime.now();
        pinCodeRepository.findByPhone(user.getUsername())
                .filter(PinCode::getVerified).orElseThrow(WRONG_PIN_CODE);
        user.setPassword(passwordEncoder.encode(user.getPassword()));
        user.setCreated(Timestamp.valueOf(now));
        user.setDeleted(false);
        user.setEnabled(true);
        user.setRole(User.Roles.ROLE_USER_WITHOUT_PASSPORT_CHECK);
        user.setPassportCheck(false);
        user.setAllDocumentReceived(false);
        user.setName(FILL_AFTER_DOC_CHECK);
        user.setSurname(FILL_AFTER_DOC_CHECK);
        user.setDateOfBirth(DEF_DATE_OF_BIRTH);
        return mapper.toDto(userRepository.save(user), UserDto.class);
    }

    @Override
    public UserDto save(User user) {
        return mapper.toDto(userRepository.save(user), UserDto.class);
    }

    @Override
    public UserDto edit(User user) {
        return mapper.toDto(userRepository.saveAndFlush(user), UserDto.class);
    }

    @Override
    public UserDto findById(Long id) {
        return
mapper.toDto(userRepository.findById(id).orElseThrow(USER_NOT_FOUND),
UserDto.class);
    }

    @Override
    public boolean delete(User user) {
        userRepository.delete(user);
        return userRepository.existsById(user.getId());
    }
```

```
    @Override
    public boolean delete(Long id) {
        userRepository.deleteById(id);
        return userRepository.existsById(id);
    }

    @Override
    public Collection<UserDto> findAll() {
        return mapper.toDto(userRepository.findAll(), UserDto.class);
    }

    @Override
    public Collection<UserDto> findAll(Specification<User> specification) {
        return null;
    }

    @Override
    public User checkUserCredentials(String phoneNumber, String password) {
        return userRepository.findByUsername(phoneNumber)
                .filter(user -> passwordEncoder.matches(password, user.getPassword()))
                .orElseThrow(USER_NOT_FOUND);
    }

    @Override
    public User findByUserName(String username) {
        return
userRepository.findByUsername(username).orElseThrow(USER_NOT_FOUND);
    }

    @Override
    public UserDto approveDocument(ApproveDocumentDto documentDto) {
        return
userRepository.findByUsername(documentDto.getUsername()).map(user -> {
            user.setPassportCheck(documentDto.getPassportCheck());
            user.setSurname(documentDto.getSurname());
            user.setName(documentDto.getName());
            user.setRole(documentDto.getRole());
            user.setDateOfBirth(new Date(documentDto.getDateOfBirth()));
            return mapper.toDto(userRepository.save(user), UserDto.class);
        }).orElseThrow(USER_NOT_FOUND);
    }

    @PostConstruct
```

```
    public void createAdminUser() {
        if (!userRepository.findByUsername("+380000000000").isPresent()) {
            User user = new User();
            final LocalDateTime now = LocalDateTime.now();
            user.setPassword(passwordEncoder.encode("1234"));
            user.setCreated(Timestamp.valueOf(now));
            user.setDeleted(false);
            user.setEnabled(true);
            user.setRole(User.Roles.ROLE_ADMIN);
            user.setPassportCheck(true);
            user.setAllDocumentReceived(true);
            user.setName(FILL_AFTER_DOC_CHECK);
            user.setSurname(FILL_AFTER_DOC_CHECK);
            user.setDateOfBirth(DEF_DATE_OF_BIRTH);
            user.setUsername("+380000000000");
            user.setEmail("bank@gmail.com");

            userRepository.save(user);
        }
    }
}
```

UserTransactionServiceImpl class

```
package com.violence.domain.service;

import com.violence.domain.data.dto.TransactionDto;
import com.violence.domain.repository.TransactionRepository;
import org.springframework.stereotype.Service;

import java.util.Collection;
import java.util.Set;

@Service
public class UserTransactionServiceImpl extends AbstractModelService
implements UserTransactionService {
    private final TransactionRepository transactionRepository;

    public UserTransactionServiceImpl(ModelMapperServiceImpl mapper,
TransactionRepository transactionRepository) {
        super(mapper);
        this.transactionRepository = transactionRepository;
```

```
    }

   @Override
    public Collection<TransactionDto> findBySenderCard(String card) {
        return mapper.toDto(transactionRepository.findAllBySenderCard(card),
TransactionDto.class);
    }

    @Override
    public TransactionDto findByTransactionId(String transactionId) {
        return
mapper.toDto(transactionRepository.findFirstByTransactionId(transactionId),
TransactionDto.class);
    }

    @Override
    public Collection<TransactionDto> findBySenderCardIn(Set<String>
senderCars) {
        return
mapper.toDto(transactionRepository.findAllBySenderCardIn(senderCars),
TransactionDto.class);
    }

    @Override
    public Collection<TransactionDto> findAllForCard(String cardNumber) {
        return
mapper.toDto(transactionRepository.findAllBySenderCardOrRecipientCard(cardN
umber, cardNumber), TransactionDto.class);
    }
}
```

ReplenishmentServiceImpl class

```
package com.violence.transfer.service.service;

import com.violence.domain.data.BaseCardDetails;
import com.violence.domain.data.dto.CardFormDto;
import com.violence.domain.data.entity.Cards;
import com.violence.domain.data.entity.Transactions;
import com.violence.domain.enums.TransactionStatus;
import com.violence.domain.enums.WithdrawalType;
import com.violence.domain.exception.ExceptionHelper;
import com.violence.domain.repository.CardRepository;
import com.violence.domain.repository.TransactionRepository;
```

```
import com.violence.domain.repository.UserRepository;
import com.violence.domain.service.DataGenerator;
import com.violence.domain.service.ModelMapperServiceImpl;
import com.violence.transfer.dto.TransferMoneyDto;
import com.violence.transfer.service.CurrencyService;
import lombok.AllArgsConstructor;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

import java.math.BigDecimal;
import java.sql.Timestamp;
import java.time.LocalDateTime;
import java.util.Optional;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.TimeUnit;

import static com.violence.domain.enums.TransactionStatus.OK;
import static java.util.Optional.of;

@Service
@AllArgsConstructor
public class ReplenishmentServiceImpl implements ReplenishmentService {
    private final CardRepository cardRepository;
    private final TransactionRepository transactionRepository;
    private final UserRepository userRepository;
    private final CurrencyService currencyService;
    private final OtherBankTransferService otherBankTransferService;
    private final KafkaTemplate<String, TransferMoneyDto> kafkaTemplate;
    private final ModelMapperServiceImpl mapper;
    private final DataGenerator dataGenerator;

    @Override
    @Transactional
    public void doFromAnotherCard(TransferMoneyDto transferMoneyDto,
boolean isInsideTransfer) {

transferMoneyDto.setCourse(currencyService.findByCode(transferMoneyDto.getC
urrency()).getAmount());
        final Cards ownerCard =
cardRepository.findByNumber(transferMoneyDto.getOwnerCard().getNumber()).
orElseThrow(ExceptionHelper.CARD_NOT_FOUND);
        final Optional<Cards> card =
cardRepository.findByNumber(transferMoneyDto.getCard().getNumber());
```

```
        final Transactions.TransactionsBuilder baseTransaction =
getBaseTransactionForOwner(transferMoneyDto);
        if (card.isEmpty()) {
            otherBankTransferService.doReplenishment(transferMoneyDto,
baseTransaction.build().getTransactionId());

baseTransaction.recipientDetails(transferMoneyDto.getCard().getNumber());
        } else {

baseTransaction.recipientDetails(userRepository.getById(card.get().getOwnerId()).
getFullName());
            of(isInsideTransfer).filter(a -> a)
                .ifPresentOrElse(a-> baseTransaction.status(OK),
                    () ->
                        CompletableFuture.supplyAsync(() ->
                            kafkaTemplate.send("inside-bank-transfers",
convertToWithdrawal(transferMoneyDto, ownerCard, card.get())),
                            CompletableFuture.delayedExecutor(5,
TimeUnit.SECONDS))
                    );
        }
        final Transactions transactions = baseTransaction
                .status(OK)
                .details("Поповнення картки")
                .balanceChangeFrom(ownerCard.getBalance())

.balanceChangeTo(ownerCard.getBalance().add(transferMoneyDto.getTransferAm
ount().multiply(transferMoneyDto.getCourse())))
                .build();
        transactionRepository.save(transactions);
        ownerCard.setBalance(transactions.getBalanceChangeTo());
        cardRepository.save(ownerCard);
    }

    @Override
    @Transactional
    public void doGooglePay(TransferMoneyDto transferMoneyDto, boolean
isInsideTransfer) {

    }

    @Override
    @Transactional
    public void doAccordingToTheDetailsInUkraine(TransferMoneyDto
```

```
transferMoneyDto, boolean isInsideTransfer) {

    }

    @Override
    @Transactional
    public void doCash(TransferMoneyDto transferMoneyDto, boolean
isInsideTransfer) {

transferMoneyDto.setCourse(currencyService.findByCode(transferMoneyDto.getC
urrency()).getAmount());
        final Cards ownerCard =
cardRepository.findByNumber(transferMoneyDto.getOwnerCard().getNumber()).
orElseThrow(ExceptionHelper.CARD_NOT_FOUND);
        final Transactions.TransactionsBuilder baseTransaction =
getBaseTransactionForOwner(transferMoneyDto);
        otherBankTransferService.doReplenishment(transferMoneyDto,
baseTransaction.build().getTransactionId());
        final Transactions transactions = baseTransaction
                .details("Поповнення картки")
                .balanceChangeFrom(ownerCard.getBalance())

.balanceChangeTo(ownerCard.getBalance().add(transferMoneyDto.getTransferAm
ount().multiply(transferMoneyDto.getCourse())))
                .build();
        transactionRepository.save(transactions);
        ownerCard.setBalance(transactions.getBalanceChangeTo());
        cardRepository.save(ownerCard);
    }

    @Override
    @Transactional
    public void doPayoneer(TransferMoneyDto transferMoneyDto, boolean
isInsideTransfer) {
        doFromAnotherCard(transferMoneyDto, isInsideTransfer);
    }

    private Transactions.TransactionsBuilder
getBaseTransactionForOwner(TransferMoneyDto transferMoneyDto) {
        return Transactions.builder()
                .transactionId(dataGenerator.randomTransactionId())
                .status(TransactionStatus.WAITING_FOR_APPROVE)
                .type(transferMoneyDto.getTransactionType().getType())
                .currency(transferMoneyDto.getCurrency())
```

```
            .recipientCard(transferMoneyDto.getOwnerCard().getNumber())

.course(currencyService.findByCode(transferMoneyDto.getCurrency()).getAmoun
t())
            .senderCard(transferMoneyDto.getCard().getNumber())
            .transferAmount(transferMoneyDto.getTransferAmount())
            .commission(BigDecimal.ZERO)
            .created(Timestamp.valueOf(LocalDateTime.now()));
    }

    private TransferMoneyDto convertToWithdrawal(TransferMoneyDto
transferMoneyDto, Cards ownerCard, Cards card) {
        final TransferMoneyDto transferMoneyDto1 = new TransferMoneyDto();
        transferMoneyDto1.setCourse(transferMoneyDto.getCourse());
        transferMoneyDto1.setCard(mapper.toDto(ownerCard, CardFormDto.class));

transferMoneyDto1.setTransferAmount(transferMoneyDto.getTransferAmount());
        transferMoneyDto1.setCurrency(transferMoneyDto.getCurrency());

transferMoneyDto1.setTransactionType(WithdrawalType.CARD_TRANSFER);
        transferMoneyDto1.setOwnerCard(mapper.toDto(card, CardFormDto.class));
        transferMoneyDto1.setCommission(BigDecimal.ZERO);
        return transferMoneyDto1;
    }
}
```

WithdrawalServiceImpl class

```
package com.violence.transfer.service.service;

import com.violence.domain.data.BaseCardDetails;
import com.violence.domain.data.dto.CardFormDto;
import com.violence.domain.data.entity.Cards;
import com.violence.domain.data.entity.Transactions;
import com.violence.domain.enums.ReplenishmentType;
import com.violence.domain.enums.TransactionStatus;
import com.violence.domain.enums.WithdrawalType;
import com.violence.domain.exception.ExceptionHelper;
import com.violence.domain.repository.CardRepository;
import com.violence.domain.repository.TransactionRepository;
import com.violence.domain.repository.UserRepository;
import com.violence.domain.service.DataGenerator;
import com.violence.domain.service.ModelMapperServiceImpl;
import com.violence.transfer.dto.TaxDto;
```

```
import com.violence.transfer.dto.TransferMoneyDto;
import com.violence.transfer.service.CurrencyService;
import lombok.AllArgsConstructor;
import org.springframework.kafka.core.KafkaTemplate;
import org.springframework.stereotype.Service;

import java.math.BigDecimal;
import java.sql.Timestamp;
import java.time.LocalDateTime;
import java.util.Optional;
import java.util.TreeMap;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.TimeUnit;

import static com.violence.domain.enums.TransactionStatus.OK;
import static java.util.Optional.of;
import static java.util.concurrent.CompletableFuture.delayedExecutor;
import static java.util.concurrent.CompletableFuture.supplyAsync;

@Service
@AllArgsConstructor
public class WithdrawalServiceImpl implements WithdrawalService {
    private static final BigDecimal OTHER_BANK_TRANSFER_COMMISSION
= BigDecimal.valueOf(0.02);
    private static final BigDecimal FOR_CREDIT_TRANSFER_COMMISSION =
BigDecimal.valueOf(0.04);
    private final CardRepository cardRepository;
    private final TransactionRepository transactionRepository;
    private final UserRepository userRepository;
    private final CurrencyService currencyService;
    private final OtherBankTransferService otherBankTransferService;
    private final KafkaTemplate<String, TransferMoneyDto> kafkaTemplate;
    private final ModelMapperServiceImpl mapper;
    private final DataGenerator dataGenerator;

    @Override
    public void doCardTransfer(TransferMoneyDto transfer, boolean
isInsideTransfer) {
        final Cards ownerCard =
cardRepository.findByNumber(transfer.getOwnerCard().getNumber()).orElseThro
w(ExceptionHelper.CARD_NOT_FOUND);
        final Optional<Cards> card =
cardRepository.findByNumber(transfer.getCard().getNumber());
        final Transactions.TransactionsBuilder baseTransaction =
```

```
getBaseTransactionForOwner(transfer);
      BigDecimal commission = BigDecimal.ZERO;
      if (card.isEmpty()) {
         commission =
transfer.getTransferAmount().multiply(OTHER_BANK_TRANSFER_COMMISS
ION);
         otherBankTransferService.doWithdrawal(transfer,
baseTransaction.build().getTransactionId());
         baseTransaction.recipientDetails(transfer.getCard().getNumber());
      } else {

baseTransaction.recipientDetails(userRepository.getById(card.get().getOwnerId()).
getFullName());
         of(isInsideTransfer).filter(a -> a)
               .ifPresentOrElse(a -> baseTransaction.status(OK),
                     () ->
                           supplyAsync(() ->
                                 kafkaTemplate.send("inside-bank-transfers",
convertToReplenishment(transfer, ownerCard, card.get())),
                                 delayedExecutor(5, TimeUnit.SECONDS)
                           )
               );
      }
      final BigDecimal balanceChangeTo =
ownerCard.getBalance().subtract(transfer.getTransferAmount()).subtract(commissi
on);
       Transactions transactions;
      if (balanceChangeTo.compareTo(BigDecimal.ZERO) >= 0) {
         transactions = baseTransaction
               .details("Переказ на картку")
               .status(OK)
               .commission(commission)
               .balanceChangeFrom(ownerCard.getBalance())
               .balanceChangeTo(balanceChangeTo)
               .build();
      } else {
         transactions = baseTransaction
               .details("Недостатньо коштів")
               .status(TransactionStatus.CANCEL)
               .commission(commission)
               .balanceChangeFrom(ownerCard.getBalance())
               .balanceChangeTo(ownerCard.getBalance())
               .build();
      }
```

```
        transactionRepository.save(transactions);
        ownerCard.setBalance(transactions.getBalanceChangeTo());
        cardRepository.save(ownerCard);
    }

    @Override
    public void doMobilePayment(TransferMoneyDto transfer, boolean
isInsideTransfer) {
        final Cards ownerCard =
cardRepository.findByNumber(transfer.getOwnerCard().getNumber()).orElseThro
w(ExceptionHelper.CARD_NOT_FOUND);
        final Transactions.TransactionsBuilder baseTransaction =
getBaseTransactionForOwner(transfer);
        BigDecimal commission = BigDecimal.ZERO;
        final BigDecimal balanceChangeTo =
ownerCard.getBalance().subtract(transfer.getTransferAmount()).subtract(commissi
on);
        Transactions transactions;
        if (balanceChangeTo.compareTo(BigDecimal.ZERO) >= 0) {
            transactions = baseTransaction
                    .details("Поповнення телефону")
                    .status(OK)
                    .commission(commission)
                    .balanceChangeFrom(ownerCard.getBalance())
                    .balanceChangeTo(balanceChangeTo)
                    .build();
        } else {
            transactions = baseTransaction
                    .details("Недостатньо коштів")
                    .status(TransactionStatus.CANCEL)
                    .commission(commission)
                    .balanceChangeFrom(ownerCard.getBalance())
                    .balanceChangeTo(ownerCard.getBalance())
                    .build();
        }

        transactionRepository.save(transactions);
        ownerCard.setBalance(transactions.getBalanceChangeTo());
        cardRepository.save(ownerCard);
    }

    @Override
    public void doIBANPayment(TransferMoneyDto transferMoneyDto, boolean
isInsideTransfer) {
```

```
    }

    @Override
    public void doUrgentTransfer(TransferMoneyDto transferMoneyDto, boolean
isInsideTransfer) {

    }

    @Override
    public TaxDto calculateCommission(TaxDto taxDto) {

        return null;
    }

//    private Cards calculateCommissionCoefficient(Cards ownerCard,
BaseCardDetails card, BigDecimal transferAmount) {
//        if (card.isOtherBank()) {
//            BigDecimal commission;
//            final BigDecimal afterTransfer =
ownerCard.getBalance().subtract(transferAmount);
//            if (afterTransfer.compareTo(BigDecimal.ZERO) >= 0) {
//                commission =
transferAmount.multiply(OTHER_BANK_TRANSFER_COMMISSION);
//                if (afterTransfer.subtract(commission).compareTo(BigDecimal.ZERO)
< 0) {
//                    ownerCard.setBalance(BigDecimal.ZERO);
//                    ownerCard.setCreditLimit(ownerCard.getCreditLimit().subtract());
//                }
//            }
//        } else {
//
//        }
//    }

    private Transactions.TransactionsBuilder
getBaseTransactionForOwner(TransferMoneyDto transferMoneyDto) {
        return Transactions.builder()
                .transactionId(dataGenerator.randomTransactionId())
                .status(TransactionStatus.WAITING_FOR_APPROVE)
                .type(transferMoneyDto.getTransactionType().getType())
                .currency(transferMoneyDto.getCurrency())
                .recipientCard(transferMoneyDto.getOwnerCard().getNumber())
```

```
.course(currencyService.findByCode(transferMoneyDto.getCurrency()).getAmoun
t())
                .senderCard(transferMoneyDto.getCard().getNumber())
                .transferAmount(transferMoneyDto.getTransferAmount())
                .commission(BigDecimal.ZERO)
                .created(Timestamp.valueOf(LocalDateTime.now()));
    }

    private TransferMoneyDto convertToReplenishment(TransferMoneyDto
transferMoneyDto, Cards ownerCard, Cards card) {
        final TransferMoneyDto transferMoneyDto1 = new TransferMoneyDto();
        transferMoneyDto1.setCourse(transferMoneyDto.getCourse());
        transferMoneyDto1.setCard(mapper.toDto(ownerCard, CardFormDto.class));

transferMoneyDto1.setTransferAmount(transferMoneyDto.getTransferAmount());
        transferMoneyDto1.setCurrency(transferMoneyDto.getCurrency());

transferMoneyDto1.setTransactionType(ReplenishmentType.FROM_ANOTHER_
CARD);
        transferMoneyDto1.setOwnerCard(mapper.toDto(card, CardFormDto.class));
        transferMoneyDto1.setCommission(BigDecimal.ZERO);
        return transferMoneyDto1;
    }
}
```

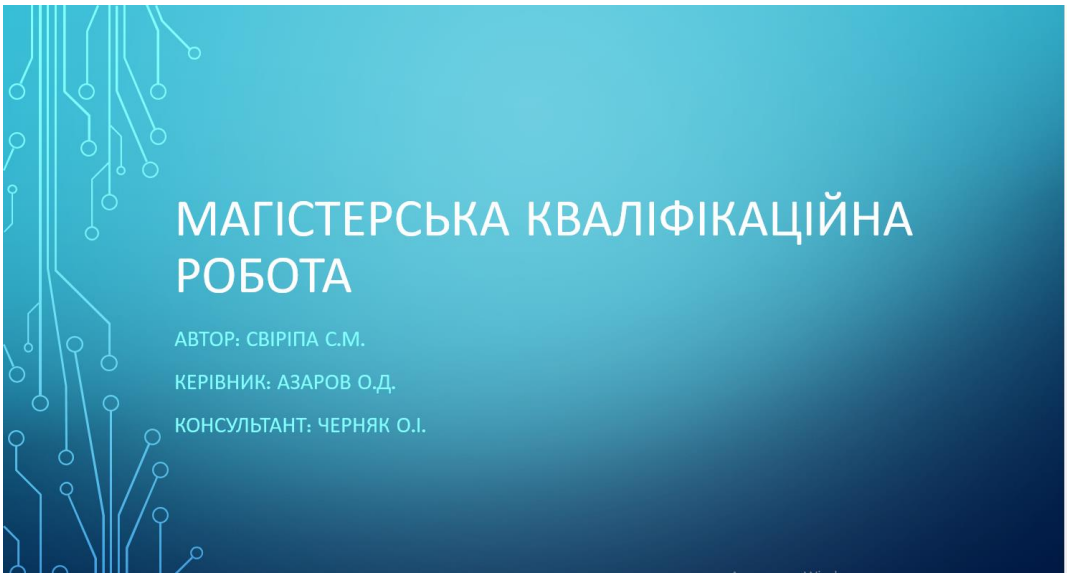
OtherBankTransferServiceImpl class

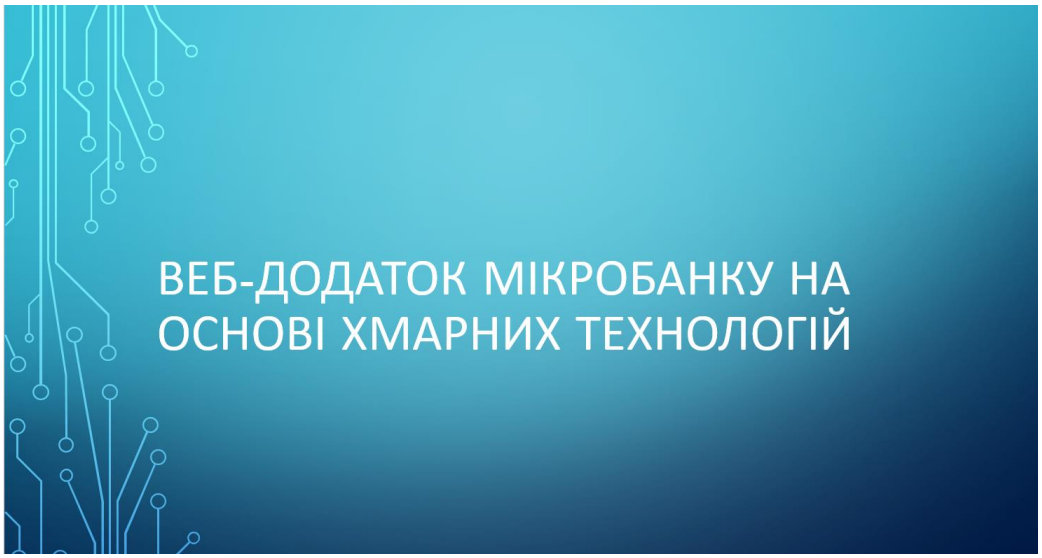
```
package com.violence.transfer.service.service;

import com.violence.domain.data.entity.Transactions;
import com.violence.domain.enums.TransactionStatus;
import com.violence.domain.repository.TransactionRepository;
import com.violence.transfer.dto.TransferMoneyDto;
import org.springframework.scheduling.annotation.Async;
import org.springframework.stereotype.Service;

import java.util.Random;
import java.util.concurrent.CompletableFuture;
import java.util.concurrent.Executor;
import java.util.concurrent.TimeUnit;

@Service
public class OtherBankTransferServiceImpl implements
OtherBankTransferService {
```

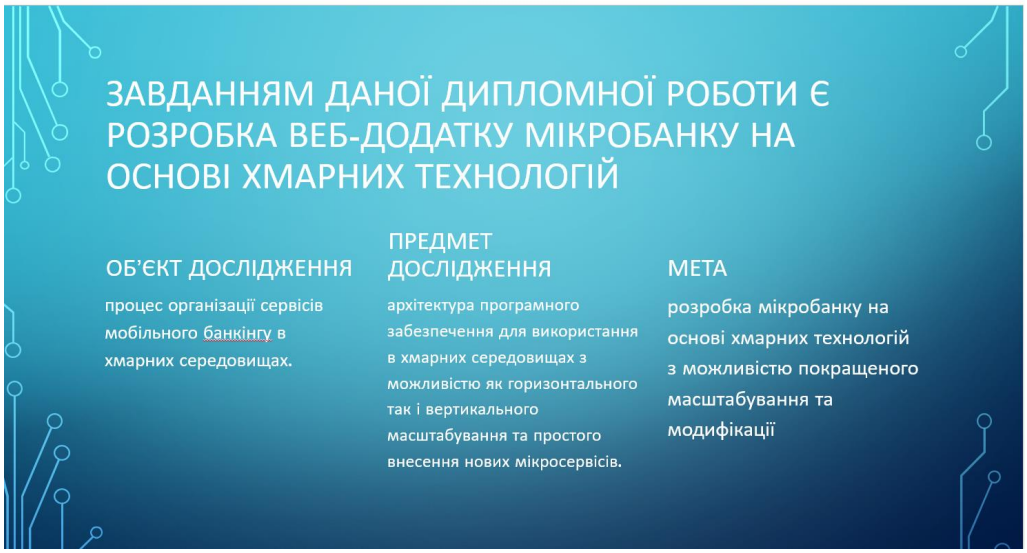
```
    private final Random random = new Random();
    private final TransactionRepository transactionRepository;

    public OtherBankTransferServiceImpl(TransactionRepository
transactionRepository) {
        this.transactionRepository = transactionRepository;
    }

    @Async
    @Override
    public void doReplenishment(TransferMoneyDto transferMoneyDto, String
transactionId) {
        final Executor executor =
CompletableFuture.delayedExecutor((random.nextInt(6) + 1) * 2,
TimeUnit.SECONDS);
        CompletableFuture.supplyAsync(() -> {
            final Transactions firstByTransactionId =
transactionRepository.findFirstByTransactionId(transactionId);
            firstByTransactionId.setStatus(TransactionStatus.OK);
            return transactionRepository.save(firstByTransactionId);
        }, executor);
    }

    @Override
    @Async
    public void doWithdrawal(TransferMoneyDto transferMoneyDto, String
transactionId) {
        doReplenishment(transferMoneyDto, transactionId);
    }
}
```

ДОДАТОК В

Презентація







ЗАВДАННЯМ ДАНОЇ ДИПЛОМНОЇ РОБОТИ Є РОЗРОБКА ВЕБ-ДОДАТКУ МІКРОБАНКУ НА ОСНОВІ ХМАРНИХ ТЕХНОЛОГІЙ

ОБ'ЄКТ ДОСЛІДЖЕННЯ

процес організації сервісів мобільного банкінгу в хмарних середовищах.

ПРЕДМЕТ ДОСЛІДЖЕННЯ

архітектура програмного забезпечення для використання в хмарних середовищах з можливістю як горизонтального так і вертикального масштабування та простого внесення нових мікросервісів.

МЕТА

розробка мікробанку на основі хмарних технологій з можливістю покращеного масштабування та модифікації

MICROSERVICE ARCHITECTURE

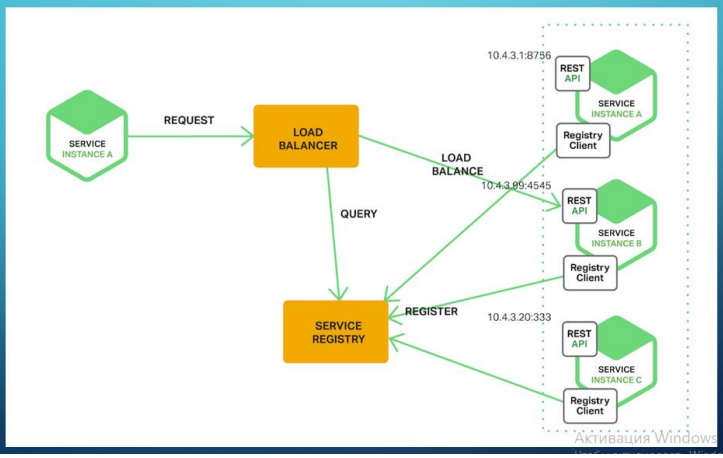
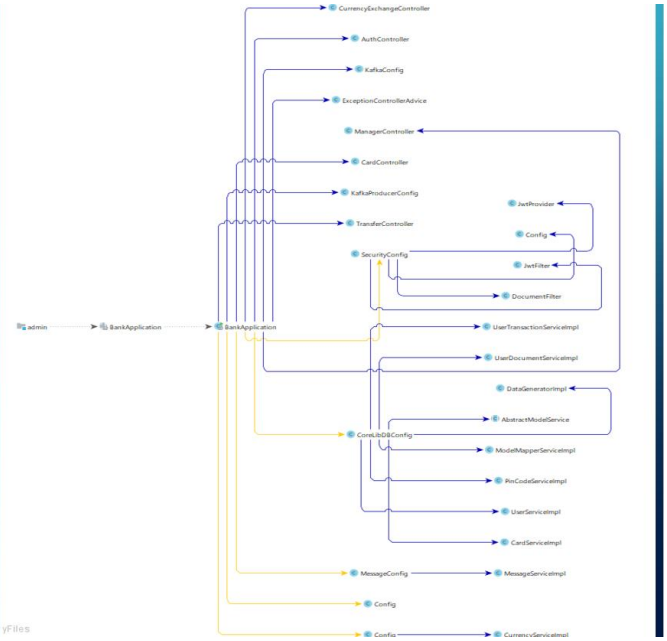
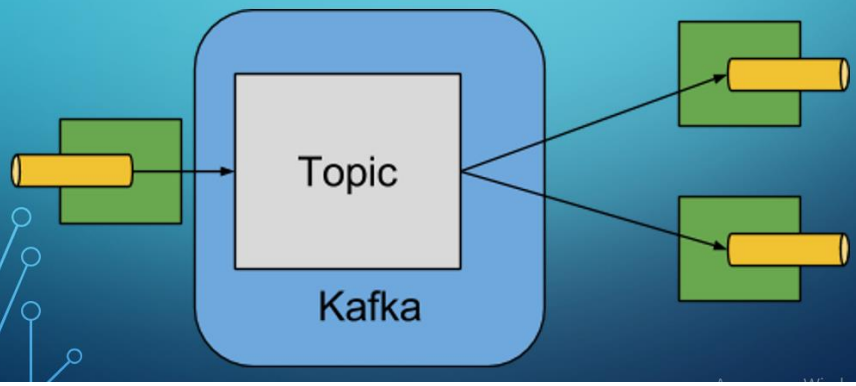


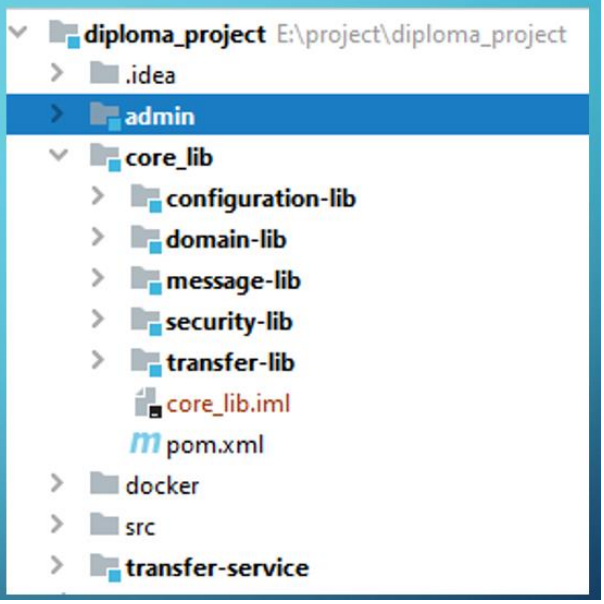
СХЕМА АДМІН МІКРОСЕРВІСА



РОБОТА MSG BORKER KAFKA



СТРУКТУРА ПРОЕКТУ

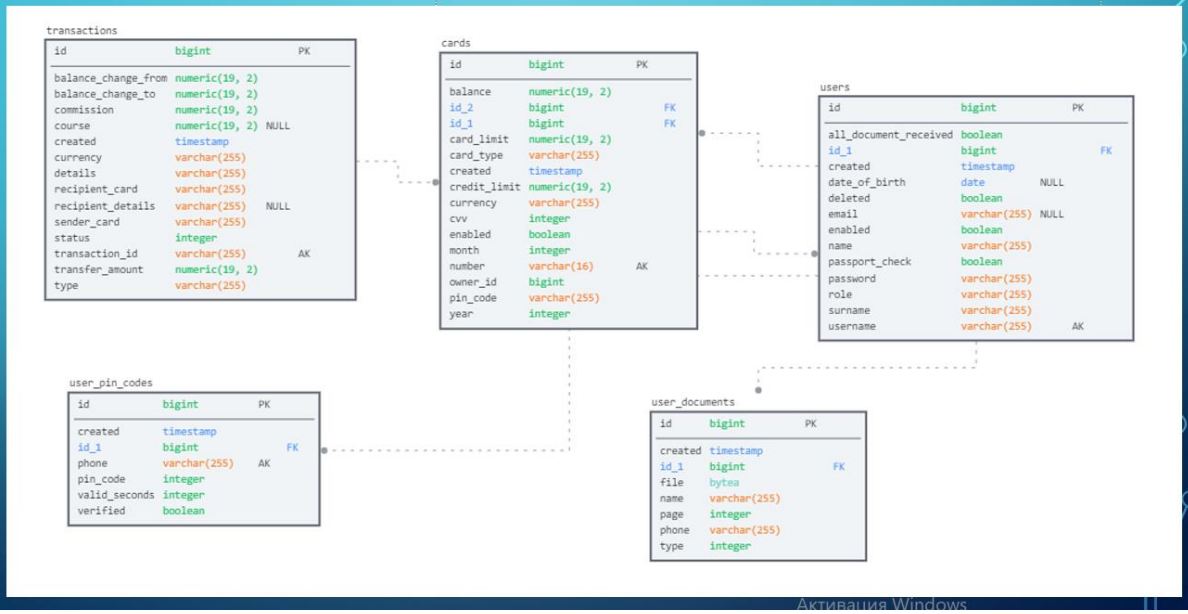


БІБЛІОТЕКИ ЯКІ ВИКОРИСТОВУЮТЬСЯ В ПРОЕКТІ

```
<dependency>
    <groupId>org.springframework.kafka</groupId>
    <artifactId>spring-kafka</artifactId>
    <version>2.8.0</version>
</dependency>
<dependency>
    <groupId>org.apache.commons</groupId>
    <artifactId>commons-lang3</artifactId>
    <version>3.11</version>
</dependency>
<dependency>
    <groupId>commons-io</groupId>
    <artifactId>commons-io</artifactId>
    <version>2.11.0</version>
</dependency>
```

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-security</artifactId>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-web</artifactId>
</dependency>
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.springframework.security</groupId>
    <artifactId>spring-security-test</artifactId>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

СХЕМА БАЗИ ДАНИХ



В ДАНІЙ МАГІСТЕРСЬКІЙ КВАЛІФІКАЦІЙНІЙ РОБОТІ БУЛО РОЗГЛЯНУТО ВЕБ-ДОДАТОК МІКРОБАНКУ НА ОСНОВІ ХМАРНИХ ТЕХНОЛОГІЙ. ДАНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМУЄ API ЗАПИТИ ДЛЯ ВЗАЄМОДІЇ З ВІРТУАЛЬНОЮ КАРТКОЮ КОРИСТУВАЧА.

ДОДАТОК Г
ПРОТОКОЛ ПЕРЕВІРКИ НАВЧАЛЬНОЇ (КВАЛІФІКАЦІЙНОЇ) РОБОТИ

Назва роботи:_____Веб-додаток мікробанку на основі хмарних технологій

Тип роботи:____магістерська кваліфікаційна робота_____

(кваліфікаційна роботи, курсовий проект (робота), реферат, аналітичний огляд, інше (зазначити))

Підрозділ _Кафедра обчислювальної техніки_____

(кафедра, факультет (інститут), навчальна група)

Науковий керівник ___Азаров О. Д., професор_____

(прізвище, ініціали, посада)

Показники звіту подібності

Plagiat.pl (StrikePlagiarism)		Unicheck	
КП1			
КП2		Оригінальність	89,8
Тривога/Білі знаки	/	Схожість	10,2

Аналіз звіту подібності (відмінити подібне)

- ✓ Запозичення, виявлення у роботі, оформлені коректно і не містять ознак плагіату.
- □ Виявлені у роботі запозичення не мають ознак плагіату, але їх надмірна кількість викликає сумніви щодо цінності і відсутності самостійності її автора. Робот направити на доопрацювання.
- □ 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.

Заявляю, що ознайомлений(-на) з повним звітом подібності, який був згенерований Системою щодо роботи (додається)

Автор _____ ___ Свіріпа С.М._____

(підпис) (прізвище, ініціали)

Опис прийнятого рішення

___Ступінь оригінальності роботи відповідає вимогам, що висуваються до МКР__

Особа, відповідальна за перевірку_____ ___Захарченко С.М. __

(підпис) (прізвище, ініціали)

Експерт _____ _____

(за потреби) (підпис) (прізвище, ініціали)