МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «ЛЬВІВСЬКА ПОЛІТЕХНІКА»

******

Курсова робота

З дисципліни

“ Прикладне програмування ”

*Виконав:*

*Якимець В. М.*

*Прийняв*

*Процько І.О.*

*Львів – 2021*

**Завдання на курсову роботу**

- навчитись формулювати задачі для програмних проектів, розкладати їх на підзадачі, вибирати методи та алгоритми для їх розв’язання;

- навчитись визначати, порівнювати, вибирати і використовувати сучасні засоби та технології програмування для розв’язування конкретних прикладних задач;

- навчитись реалізовувати, відлагоджувати та розгортати програмні проекти з використанням сучасних технологій розробки;

- навчитись описувати програмні проекти, як кінцеві результати, так і процес виконання.

- розширити реалізацію програмного продукту із комплексної лабораторної роботи, обравши певний тип програмного рішення: настільний (desktop), веб-орієнтований або мобільний застосунок, та піключивши сховище даних.

**Зміст**

[1. Вступ 4](#_Toc90575619)

[2. Постановка задачі 5](#_Toc90575620)

[3. Огляд літератури 6](#_Toc90575621)

[3.1 Предметна область 6](#_Toc90575622)

[3.2 Огляд схожих рішень 7](#_Toc90575623)

[3.3 Можливі варіанти розв’язання поставленої задачі та вибір технологій 7](#_Toc90575624)

[4. Опис етапу проектування 12](#_Toc90575625)

[4.1 Діаграма архітектури проекту 12](#_Toc90575626)

[4.2 Use cases 12](#_Toc90575627)

[4.3 Діаграма бази даних 14](#_Toc90575628)

[4.4 Діаграма шарів бекенд сервісів 15](#_Toc90575629)

[5. Програмне рішення 16](#_Toc90575630)

[5.1 Взаємодія з ui-частиною та базою даних. 16](#_Toc90575631)

[5.2 Структура проекту 17](#_Toc90575632)

[5.2.1 Структура фронтенд частини 17](#_Toc90575633)

[5.2.2 Структура бекенд частини 18](#_Toc90575634)

[6. Опис проведених експериментів 22](#_Toc90575635)

[6.1 Алгоритм розгортання веб-застосунку 22](#_Toc90575636)

[6.2 Система контролю версій та робота із кількох комп’ютерів 22](#_Toc90575637)

[6.3 Приклади користування програмою 24](#_Toc90575638)

[7. Висновки 27](#_Toc90575639)

[8. Список використаних джерел 28](#_Toc90575640)

[9. Додатки 29](#_Toc90575641)

# Вступ

**Мета:** розробити програмне рішення - систему менеджменту кредитів (їх створення, перегляд, редагування, перегляд позичальників, тощо) з ціллю спрощення вибору варіантів кредиту, його оформлення, уникнення черг та натовпу для позичальника.

**Актуальність:** в наш час, коли чи не кожна людина має власний ком’ютер, ноутбук, мобільний телефон, цифрове рішення даної системи однозначно користуватиметься попитом. Такий програмний продукт вирішить велику кількість проблем, а саме: можливість переглядати та порівнювати кредитні пропозиції банків в одному додатку, необхідність відвідування банкових відділень для оформлення кредитів пропаде, що в свою чергу знизить ймовірність утворення натовпу та черг у банкових приміщеннях. Не варто і забувати про факт, що це просто зекономить час як і працівників банку так і позичальників.

**Майбутнє застосування на практиці:** реалізований додаток будуть використовувати обидві сторони: позичальники та банки, варто зазначити, що дане програмне рішення більш націлене на ріст популярності за рахунок зручності в користуванні для позичальників, в свою чергу, якщо таким застосунком будуть користуватись багато потенційних клієнтів, банки будуть майже змушені викладати власні варіанти кредитних пропозицій для постійного приросту клієнтської бази.

# Постановка задачі

Як тип свого застосунку я обрав веб-орієнтований додаток, тому що доступ до Інтернету сьогодні має майже кожна людина, таким чином я матиму змогу охопити якнайбільшу кількість користувачів.

Моє програмне рішення має забезпечувати наступний функціонал, який я поділяю на дві групи:

1. функціонал для позичальника
2. функціонал для банку

Отож, позичальник матиме змогу виконувати наступні дії:

1. Переглядати кредитні пропозиції всіх банків
2. Виконувати пошук-фільтрування кредитів за певними параметрами
3. Оформляти кредит
4. Реєструватись у системі
5. Логуватись у системі
6. Переглядати запозичені кредити
7. Погашати кредити, в тому числі достроково
8. Збільшувати кредитну лінію

В свою чергу оператор банку може:

1. Додавати кредитні пропозиції
2. Переглядати кредитні пропозиції
3. Редагувати, видаляти кредитні пропозиції, якщо даний кредит вже не запозичив користувач
4. Переглядати запозичені кредити користувачами
5. Реєструватись у системі
6. Логуватись у системі

# Огляд літератури

## Предметна область

**Кредит** — кошти й матеріальні цінності, що надаються кредитором у користування позичальнику на певний строк та під відсоток. Кредит поділяють на фінансовий, товарний і кредит під цінні папери, які засвідчують відносини позики.

Також розрізняють основні принципи кредитування:

1. Поверненість позиченої вартості — повернення позичальником кредитору всього обсягу позиченої вартості.
2. Строковість позики — чітко обумовлено термін користування позикою.
3. Платність — позичальник платить кредитору за користування взятою в борг вартістю. Формою плати є відсоток за кредитом.
4. Цільове призначення позики — базова передумова забезпечення інтересів сторін. Позичальник чітко визначає ціль, на яку будуть позичені кошти. Вона погоджується обома сторонами, які вступають у кредитні відносини.
5. Забезпеченість — заходи, що гарантують повернення позички у визначений строк. Як забезпечення може виступати майно, капітал, авторські права тощо.

**Банк** — кредитно-фінансова установа, яка здійснює грошові розрахунки, акумулює грошові кошти та інші цінності, надає кредити та здійснює інші послуги за фінансовими операціями.

## Огляд схожих рішень

Аналогічних за призначенням та функціоналом проектів я не знайшов, проте знайшов схожі, серед таких мобільні додатки, або веб-сайти відомих українських банків: Privat Bank, Monobank, Universal Bank. Головна їхня відмінність, що вони не дають змоги розглядати кредитні пропозиції інших банків, що є цілком логічним рішенням, адже банки не зацікавленні у втраті клієнтів. З іншого боку, у таких застосунках є куди ширший функціонал, оскільки передбачені різні дії із власними коштами включаючи різні види переказів, в тому числі навіть покупка залізничних квитків. Вищезгадані компанії пропонують різні типи рішень своїх застосунків, від мобільних до веб.

## 3.3 Можливі варіанти розв’язання поставленої задачі та вибір технологій

Для написання веб застосунку можна використати різний стек технологій, як і для front-end частини, так і серверної. Питання про вибір бази даних також залишається відкритим.

Почнемо із front-end частини – будь-який сучасний вебсайт не обходиться без html, css, javascript, проте написання динамічного сайту без використання бібліотек чи фреймворків є доволі складною та часозатратною задачею.

До поширених фреймворків та бібліотек належать наступні проекти:

1. React – бібліотека для javascirpt, яка активно розробляється та підтримується Facebook
2. Angular – написаний на typescript компанією Google front-end фреймворк
3. Vue – Javascript фреймворк написаний Еваном Ю

Розглянемо переваги та недоліки кожного з них:

1. React

Плюси:

1. Дуже швидкий завдяки реалізації React Virtual DOM і різним оптимізаціям рендерингу.
2. Redux, найпопулярніша платформа для управління станом додатків у React, її легко вчити та використовувати
3. React реалізує концепції функціонального програмування (FP), створюючи простий у тестуванні та багаторазово використовуваний код.

Мінуси:

1. React відходить від компонентів на основі класів, що може стати перешкодою для розробників, яким комфортніше працювати з об'єктно-орієнтованим програмуванням (ООП).
2. Змішування шаблонів з логікою (JSX) може спантеличити деяких розробників при перших знайомствах з React.

2. Angular

Плюси:

1. Angular використовується разом з Typescript. Він має виняткову підтримку для цього.
2. Angular-language-service – забезпечує інтелектуальні можливості та автозаповнення шаблону HTML-компонента.
3. MVVM (Model-View-ViewModel) патерн, який дозволяє розробникам добре структурувати власний код
4. Структура та архітектура, спеціально створені для великої масштабованості проекту

Мінуси:

1. Різноманітність різних структур (Injectables, Components, Pipes, Modules тощо) ускладнює вивчення порівняно з React і Vue.js, які мають лише «Component».
2. Повільність. З іншого боку, це можна легко вирішити, використовуючи так званий ChangeDetectionStrategy, який допомагає вручну контролювати процес рендерингу компонентів.

3. Vue

Плюси

1. Масштабування. Vue.js може допомогти в розробці досить великих шаблонів багаторазового використання, які можуть бути зроблені майже за той самий час, що і простіші.
2. Крихітний розмір. Vue.js важить близько 20 КБ, зберігаючи при цьому свою швидкість і гнучкість, що дозволяє досягти кращої продуктивності в порівнянні з іншими платформами.
3. Детальна документація. Vue.js має дуже докладну документацію, яка може прискорити процес навчання для розробників та заощадити багато часу на розробку програми, використовуючи лише базові знання HTML та JavaScript.

Мінуси

1. Нестача ресурсів. Vue.js, як і раніше, займає досить невелику частку ринку в порівнянні з React або Angular, що означає, що обмін знаннями в цьому середовищі все ще знаходиться на початковій стадії.
2. Ризик надмірної гнучкості. Іноді у Vue.js можуть виникнути проблеми при інтеграції у величезні проекти, і поки що немає досвіду можливих рішень, але вони обов'язково з'являться найближчим часом.

Врахувавши переваги та недоліки кожного з фреймворків чи бібліотек, я обрав React для написання front-end частини, взявши до уваги те, що у мене є попередній досвід із ним.

Щодо написання серверної частини, то згідно умови завдання, вона має бути написана на Java, тут простіше, оскільки є фреймворк у якого немає конкурентів – Spring, та його похідні компоненти як Spring Boot – який спрощує процес розробки додатків на основі Spring, а саме за допомогою автоконфігурації. Варто згадати, що Spring Boot включає в себе built-in контейнер сервлетів, що дозволяє запускати наші аплікації як звичайні java програми. Потрібно також не забувати, що на основі Spring створена велика кількість фреймворків для побудови сучасних веб-сервісів – серед них: Spring Security – який запроваджує функції автентифікації та авторизації та інші аспекти безпеки для програм, Spring Data JPA – бібліотека, що є додатковим рівнем абстракції поверх ORM реалізації JPA, таким чином запроваджує максимально зручний доступ до бази даних

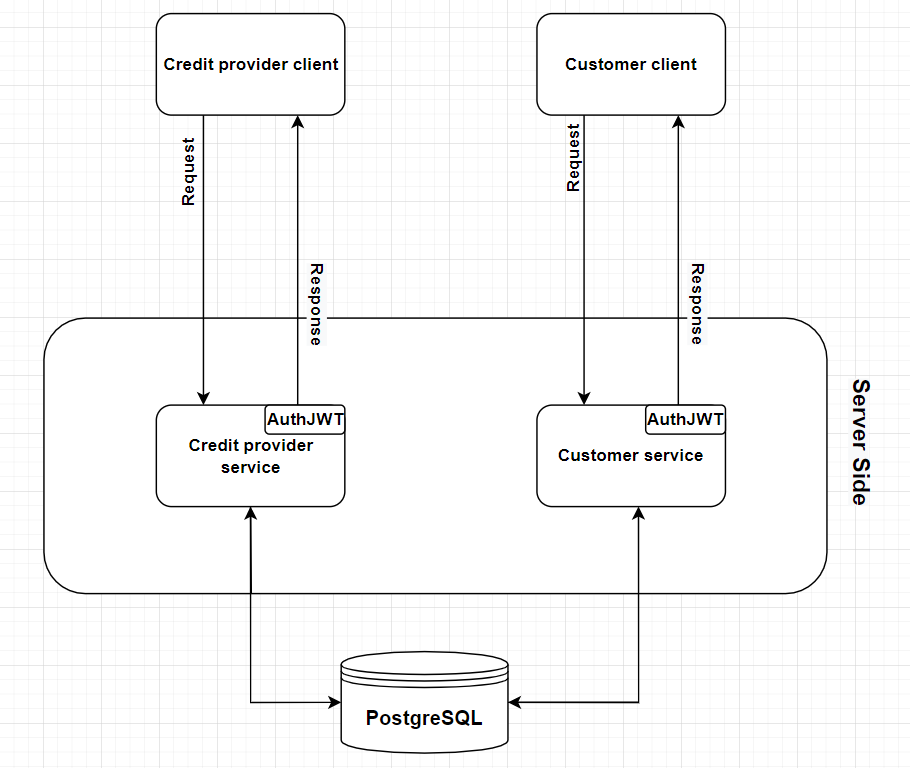
І, зрештою, потрібно розглянути варіанти бази даних для проекту. Серед можливих варіантів NoSQL бази даних та реляційні. Перші розраховані на роботу із великим об’ємом даних, і відповідно на масштабованість за менші кошти (горизонтальну масштабованість) жертвуючи невідповідністю даних при їх добуванні, неефективність при складних запитах та малим комюніті. Реляційні в свою чергу більш поширені, та дозволяють ефективніше виконувати складніші запити проте розраховані на застосунки середнього розміру, яким мій додаток і є, оскільки подальше його застосування розраховане лише на український ринок.

Отож, обравши тип бази даних, потрібно обрати конкретну субд, серед яких є найбільш поширені: PostgreSQL, MySQL, SQLServer. Я вирішив обрати PostgreSQL, тому що дана реалізація має кращі результати у ефективності виконання паралельних запитів, підтримує всі потрібні типи даних, має велике комюніті, та має підтримку нереляційних типів даних: JSON, XML Hstore, Cstore.

# Опис етапу проектування

## 4.1 Діаграма архітектури проекту

У своєму проекті я використав мікросервісний підхід для побудови веб-застосунку, де кожен клієнт звертається до потрібного бекенд-сервісу, а спільною точкою є лише база даних.

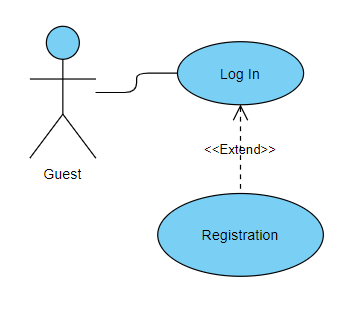
****

*Рис.1 Діаграма архітектури проекту*

## 4.2 Use cases

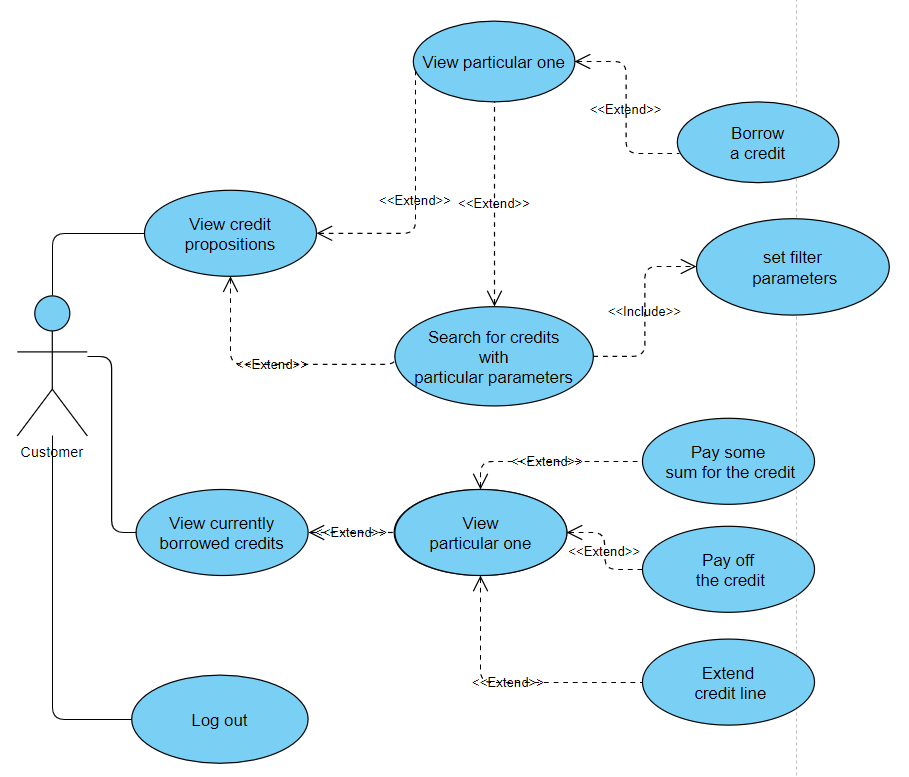
Функціонал застосунку відображено на Use Case діаграмі наведеній нижче:

Use cases для незалогованого юзера:



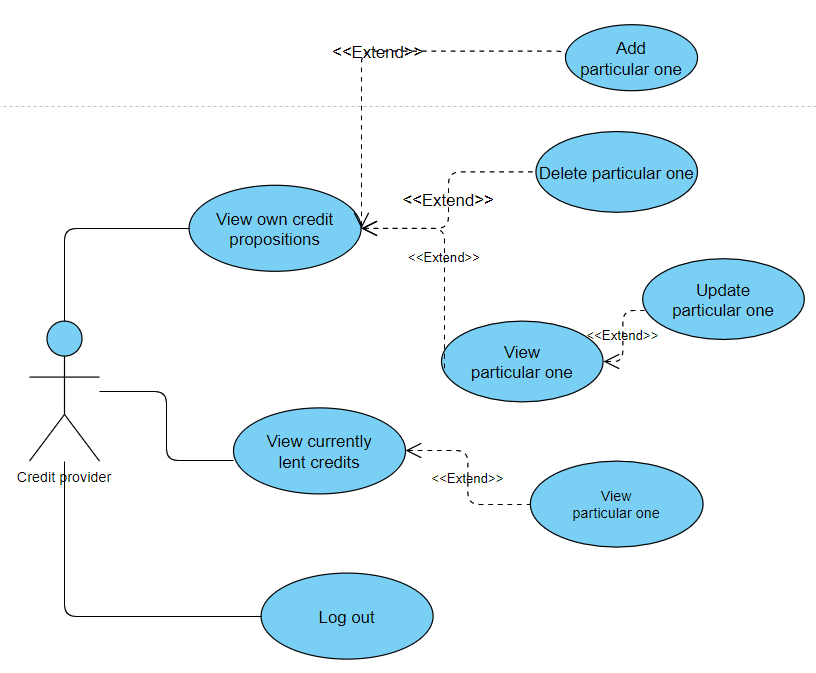
*Рис.2 Діаграма прецендентів для гостя сайту*

Use cases для позичальника:



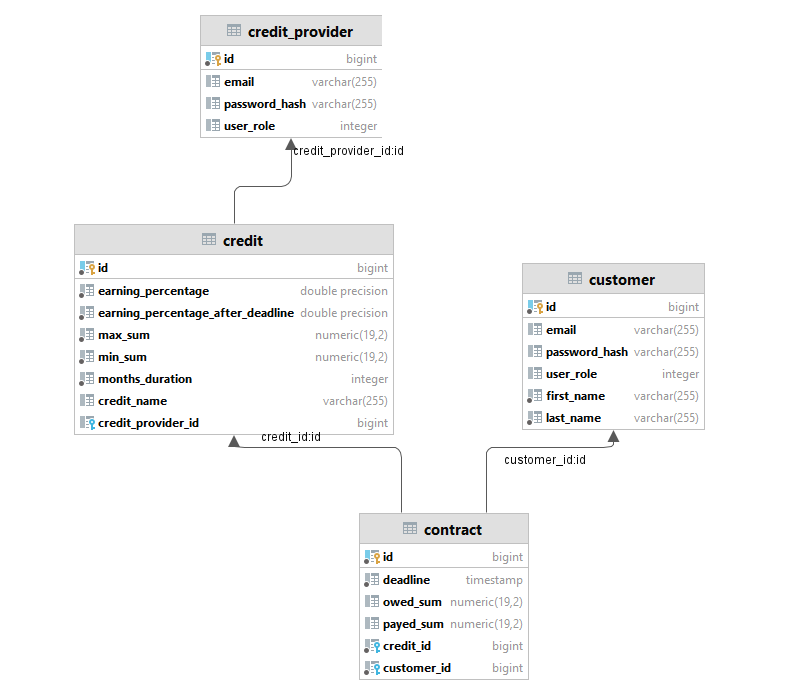
*Рис.3 Діаграма прецендентів для позичальника*

Use cases для оператора банку



*Рис.4 Діаграма прецендентів для оператора банку*

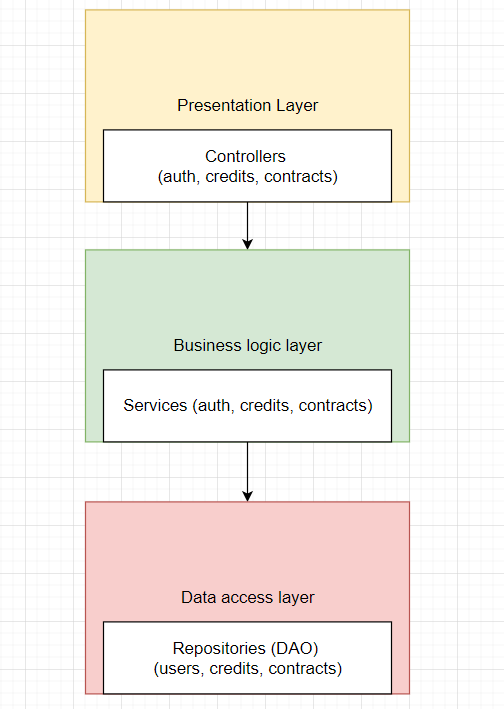
## 4.3 Діаграма бази даних



*Рис.4 Схема БД*

## 4.4 Діаграма шарів бекенд сервісів

Загалом я розподілив бекенд додатку на 3 логічних шари, тому діаграма бекенд-сервісу (як і сервісу позичальника, так і сервісу банк-оператора) виглядає наступним чином:



*Рис.5 Схема бекенд сервісів*

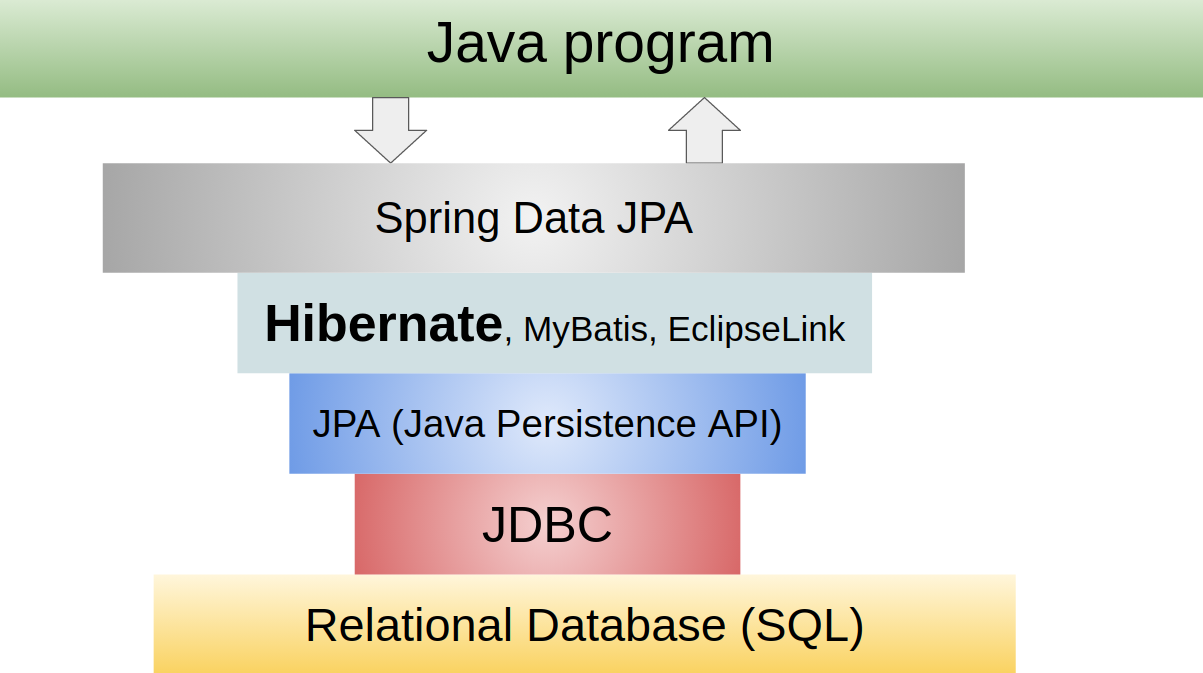
Можемо бачити, що рівень презентації, а саме контроллери використовують сервіси (рівень бізнес-логіки), які в свою чергу використовують класи рівня доступу до даних. За рахунок легкої зв’язності таких компонентів, такий підхід дозволяє нам без великих зусиль замінити будь який з цих шарів програми.

Оскільки діаграма класів є досить великою, основні її частини можна знайти у додатку.

# Програмне рішення

## 5.1 Взаємодія з ui-частиною та базою даних.

Як і вже згадувалось програмна реалізація додатку розроблена із використанням патерну мікросервісів. Графічний інтерфейс побудовано за допомогою React-компонентів, та бібліотеки для легкого менедженту стейту програми Redux. А взаємодія із серверною частиною відбуваєтсья за допомогою асинхронного http клієнту *axios* зі сторони фронт-енд частини, та контроллерів і вбудованого контейнеру сервлетів у спрінг-буті Tomcat зі сторони серверу. Взаємодія із базою даних відбувається через Spring Data JPA, що є рівнем абстракції над Hibernate (по замовчуванню), яка в свою чергу є імплементацією Java Persistence API, яка використовує JDBC driver, як міст між світом Java та світом бази даних. Графічне предствлення можемо бачити на рисунку нижче.

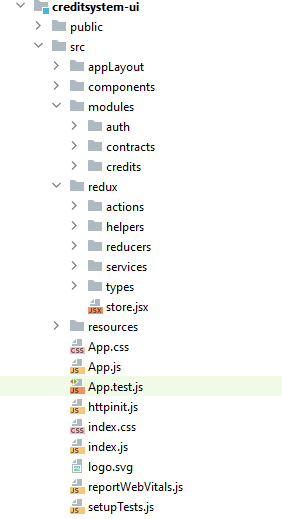


*Рис.6 Схема взаємодії із базою даних*

## Структура проекту

### 5.2.1 Структура фронтенд частини

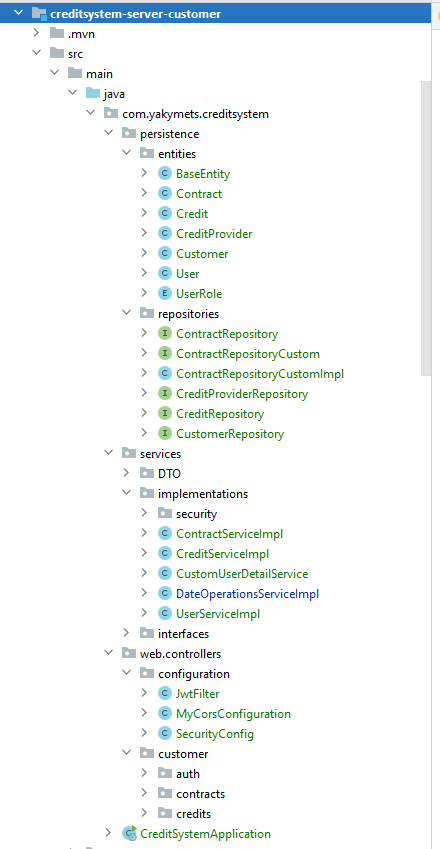
Фронт-енд частина містить наступну будову:



*Рис.7 структура ui частини*

Папка appLayout декларування батьківської розмітки сторінок. Components – папка компонентів які не мають логічного змісту, і не утворюють модулі (висувне меню, заголовок, звичайні допоможні компоненти, які використовуються в інших). Modules – логічні модулі ui частини, сторінки кредитів, каталогу кредитів, заключених контрактів із банками, сторінки авторизації, тощо. Redux – папка для управління станом програми. Певний компонент із конкретного модулю відправляє як повідомлення функцію яка знаходиться в actions, яка зазвичай викликає сервіс, із папки сервіс, який викликає http client – axios, який надсилає http запит до нашого серверу. Після отриманої відповіді – викликається відповідний reducer із папки reducers, який змінює потрібний компонент стану програми.

### 5.2.2 Структура бекенд частини



*Рис.8 структура cерверної частини*

Я розповім лише про структуру серверної частини сервісу кастомера (позичальника), оскільки реалізація сервісу оператора банку є аналогічною.

CreditSystemApplication.java – вхідна точка програми, старт нашого сервісу.

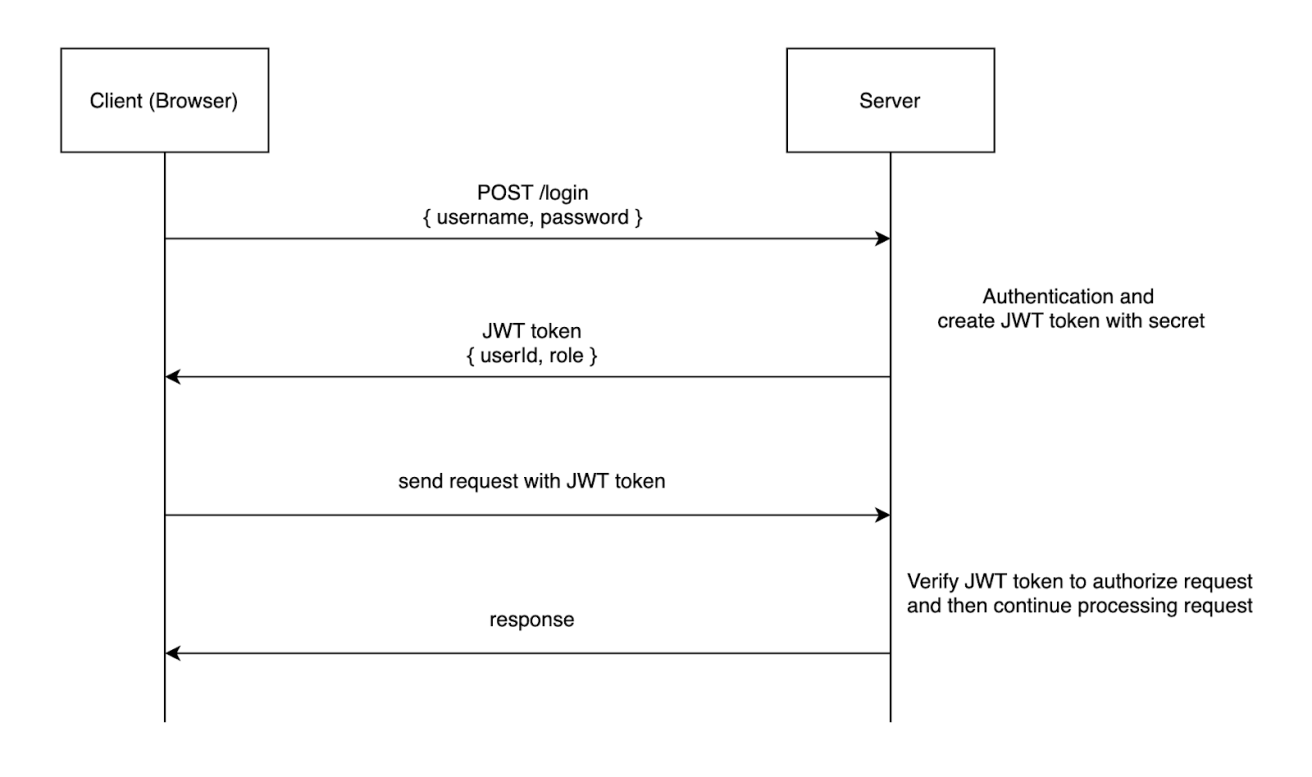
Package web.controllers – пакет наших контроллерів, що приймають запити від http клієнту, та вертають певний результат.

Всередині пакету є внутрішні пакети такі як: пакет contracts призначений для виконання CRUD операцій із заключеними контрактами між банком та позичальником. Пакет credits призначений для виконання CRUD операцій із кредитами.

Цікавим є і пекедж auth, який виконує автентифікацію та авторизацію для нашого клієнта. Я використовую JWT token ля автентифікації та авторизації клієнта. Тому було створено додатковий пакет configuration для задання додаткової конфігурації серверної частини, а саме: поставлено додатковий jwt-filter для запитів що надсилаються із клієнта, таким чином для отримання успішних результатів для більшості запитів потрібно мати jwt token із відповідною закодованою роллю у ньому.

Токен створюється та відсилається на клієнт при виконанні успішного логін процесу. Далі при надсиланні запиту із кліжнту, в кожного реквесту уже є заданий в authorization headers певний токен.

Даний процес проілюстровано нижче:



*Рис.9 Auth process в моєму програмному рішенні*

Також можемо бачити CorsConfiguration.java файл, який призначений для налаштування Cross-Origin Resource Sharing – механізму безпеки сучасних браузерів, який дозволяє веб-сторінкам використовувати дані, що знаходяться на інших доменах. У ньому ми просто дозволяємо нашому сокету із браузера стукатись до веб-серверу. Виглядає це наступним чином:

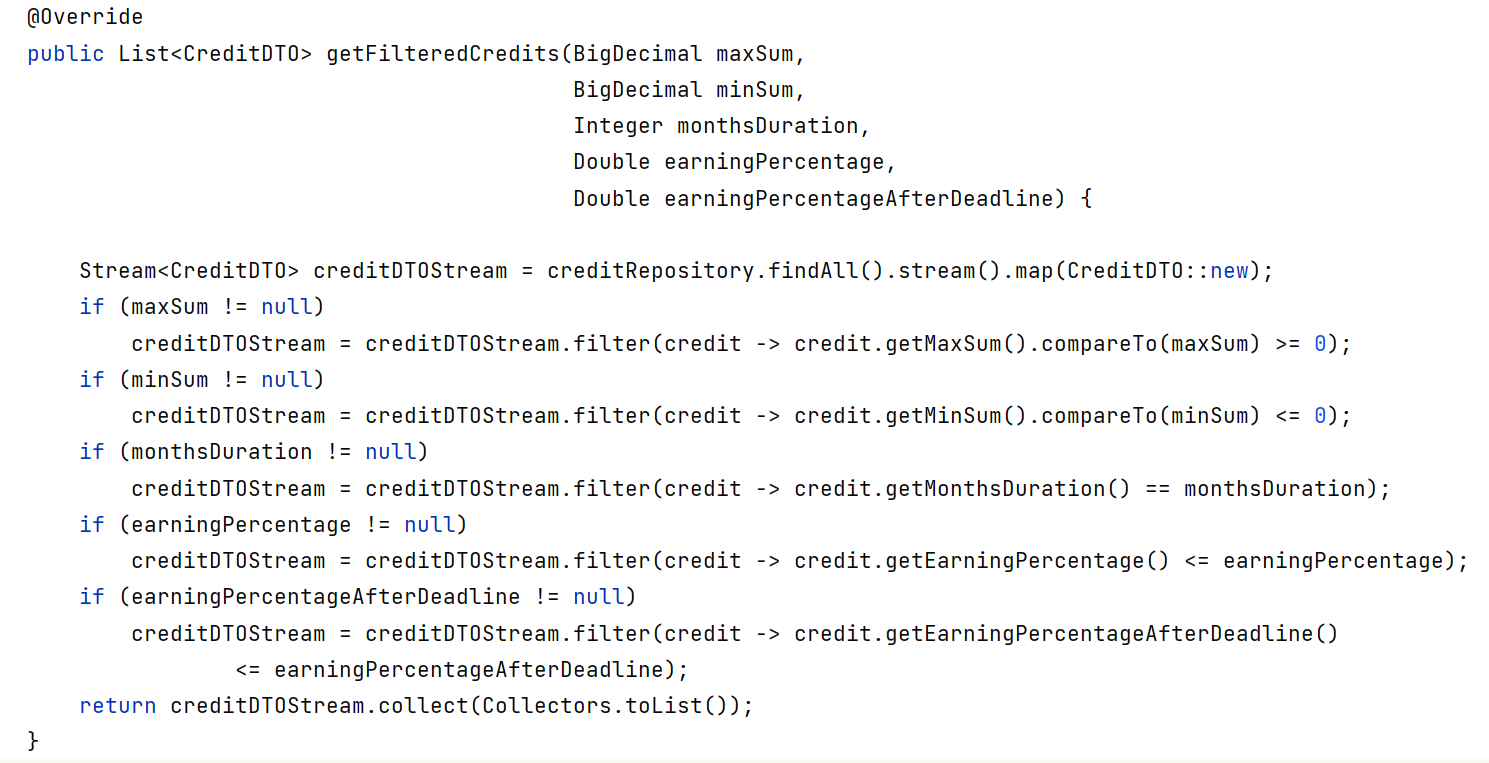


*Рис.10 налаштування CORS*

Пакет services – пакет шару бізнес-логіки, який використовують контроллери, а також, де виконуються основні обчислювальні операції, такі як фільтрування кредитів по параметрах, і відбувається взаємодія із шаром доступу до даних для отримання об’єктних представлень моделей із бази даних а також мапування їх до певних об’єктів які буде буде відсилати наш контроллер, і отримувати та обробляти клієнт.

Пакет persistence – шар доступу до даних (отримання, видалення, редагування, створення кредитів, користувачів, контрактів із сховища та мапування їх на об’єктні моделі)

Також варто згадати функціонал фільтрування, який реалізовано за допомогою Stream API



*Рис.11 фільтрування кредитів по параметрах*

# Опис проведених експериментів

## 6.1 Алгоритм розгортання веб-застосунку

Оскільки додаток обгорнутий у докер-контейнери для його запуску потрібно:

1. Склонуватипроект <https://github.com/YakymetsVova/creditsystem-coursework>
2. Завантажити та встановити докер на свій ПК із сайту <https://docs.docker.com/desktop/windows/install/>
3. Запустити докер
4. Відкрити термінал
5. Перейти у робочу кореневу директорію проекту: *creditsystem/*
6. Виконати команду: docker compose build
7. Виконати команду: docker compose up
8. Після успішного старту всіх контейнерів ui частина для позичальника доступна за посиланням: <http://localhost:3000/>
9. Для того, аби запустити ui частину для оператора банку існує два варіанти:

а) Відкрити git bash, перейти у директорію*creditsystem/creditsystem-ui* виконати команду *REACT\_APP\_API\_URL=*'http://localhost:8080/api/credit-provider' *PORT=3001 npm start*

б) Відкрити файл htttpservice.js у директорії creditsystem/creditsystem-ui/redux/services і змінити'http://localhost:8081/api/customer' на'http://localhost:8080/api/credit-provider' після цього виконати кроки 4-7

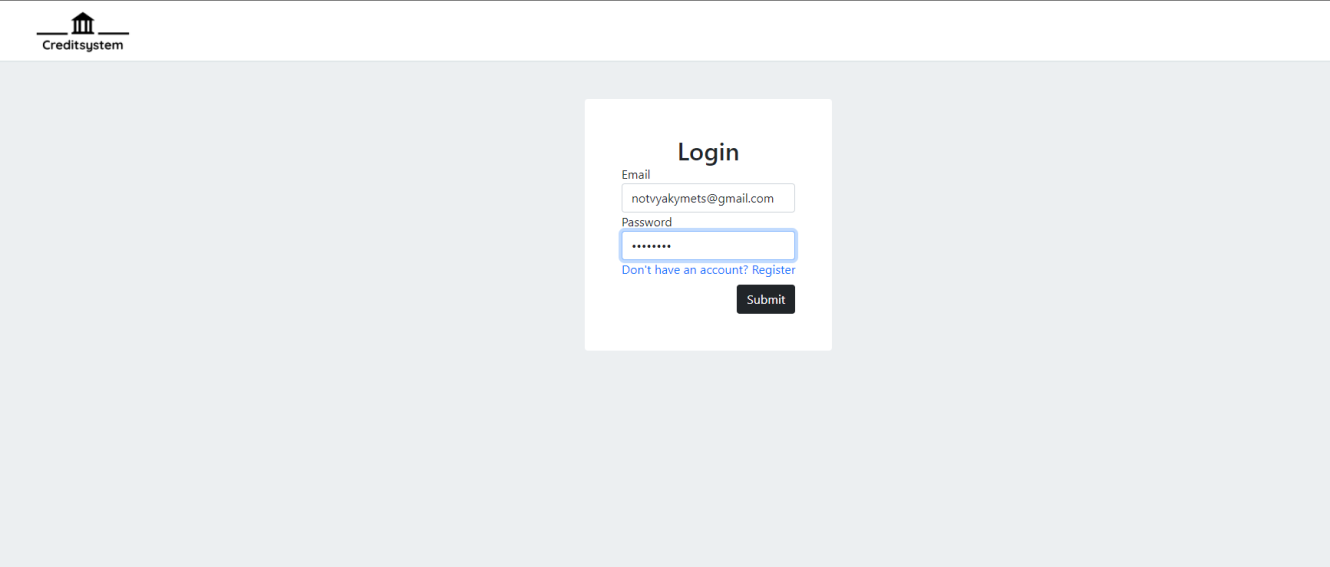
* 1. **Система контролю версій та робота із кількох комп’ютерів**

Для цього проекту використовується система контролю версій git, а проект розміщено на віддаленому репозиторію GitHub: <https://github.com/YakymetsVova/creditsystem-coursework>

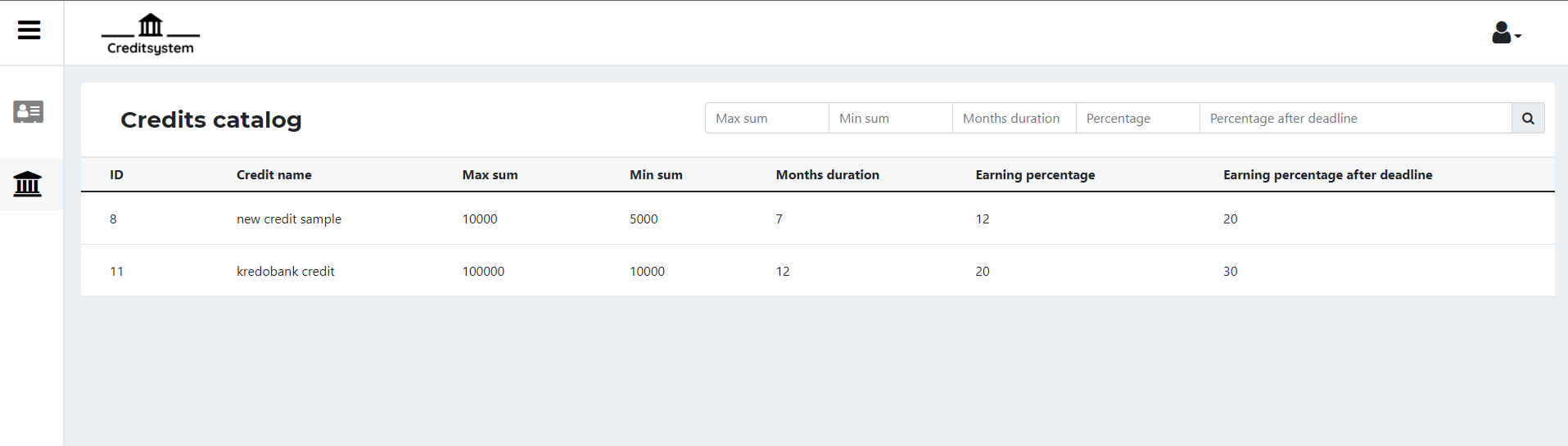
Тому для продовження роботи на іншому пк потрібно спочатку додати та закомітати для відстежування всі зроблені зміни: git commit –am”your message”, запушити зміни на віддалений репозиторій: git push origin your\_branch\_name

Після цього, на іншому пк відкрити термінал, перейти в потрібну директорію та склонувати репозиторій: git clone <https://github.com/YakymetsVova/creditsystem-coursework.git>

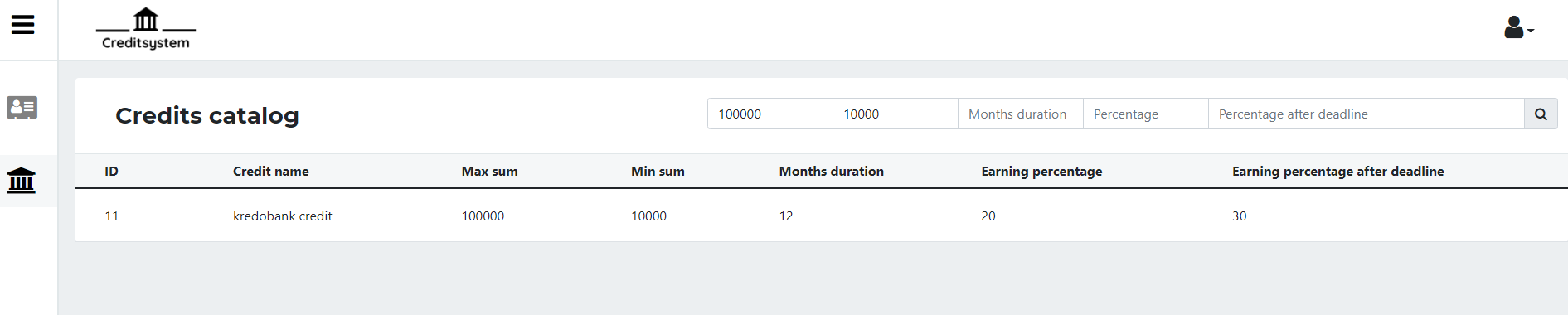
## 6.3 Приклади користування програмою



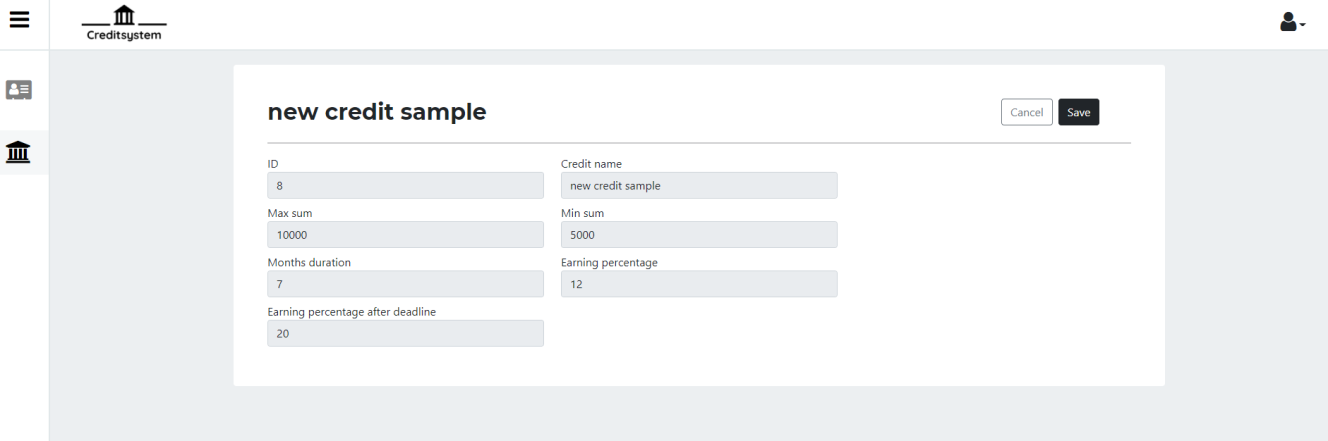
*Рис.12 Логін сторінка для позичальника*



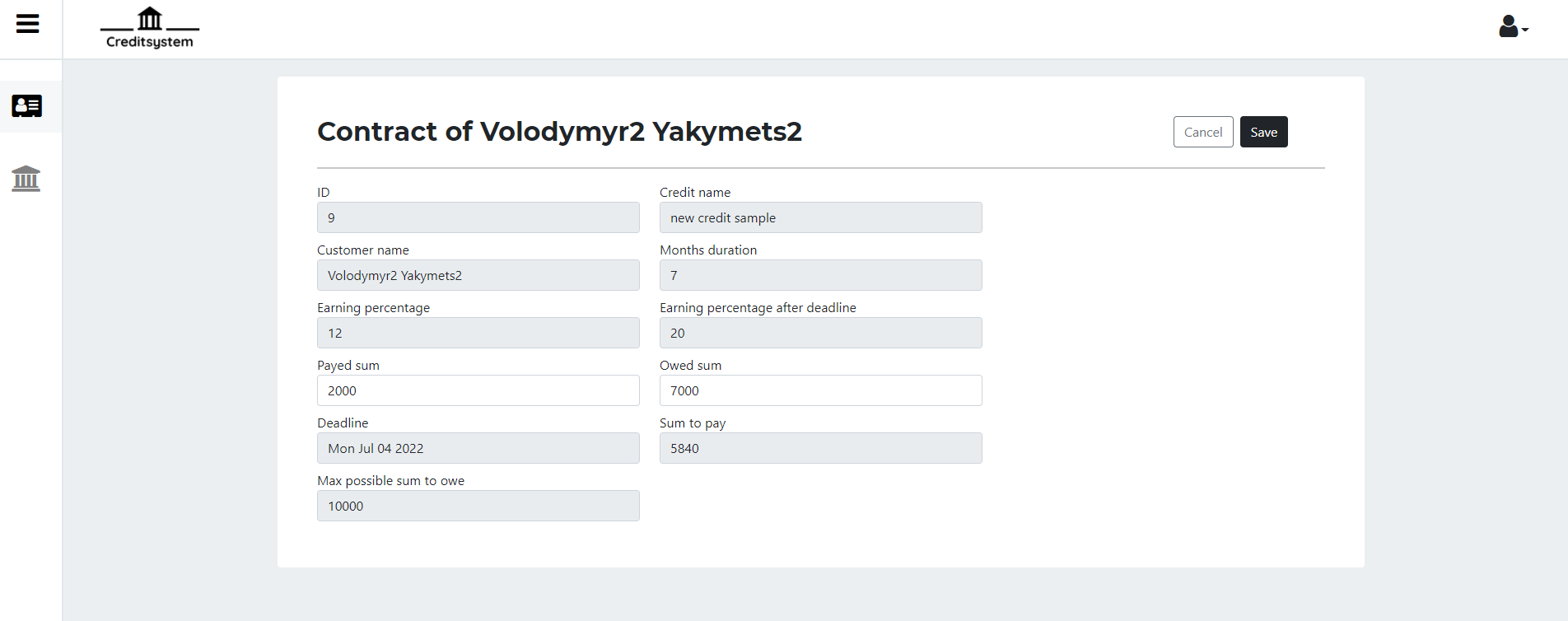
*Рис.13 Сторінка кредитних пропозицій для позичальника*



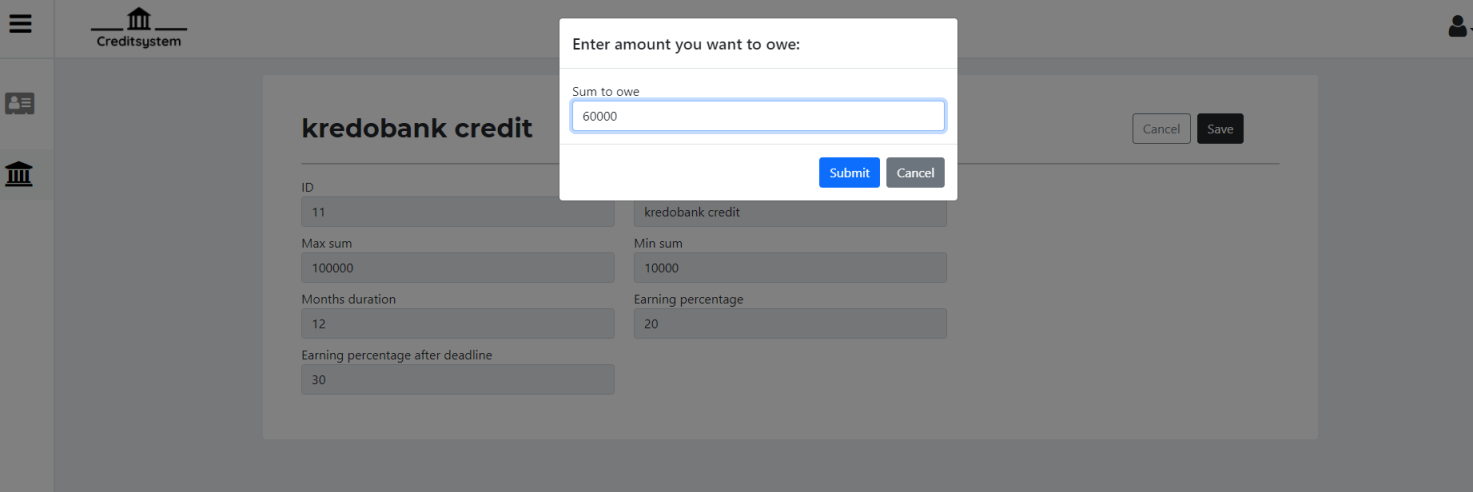
*Рис.14 Сторінка відфільтрованих кредитних пропозицій*



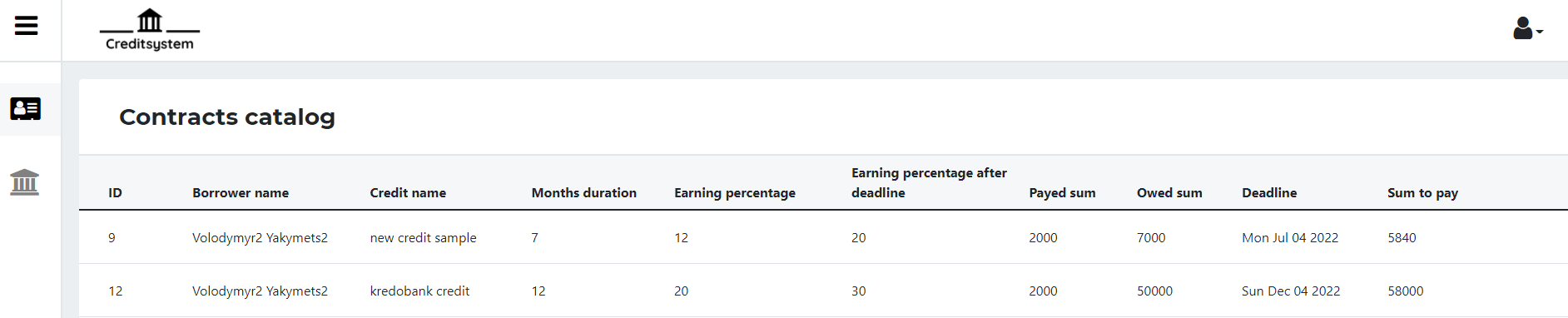
*Рис.15 Сторінка із детальною інформацією про кредити*



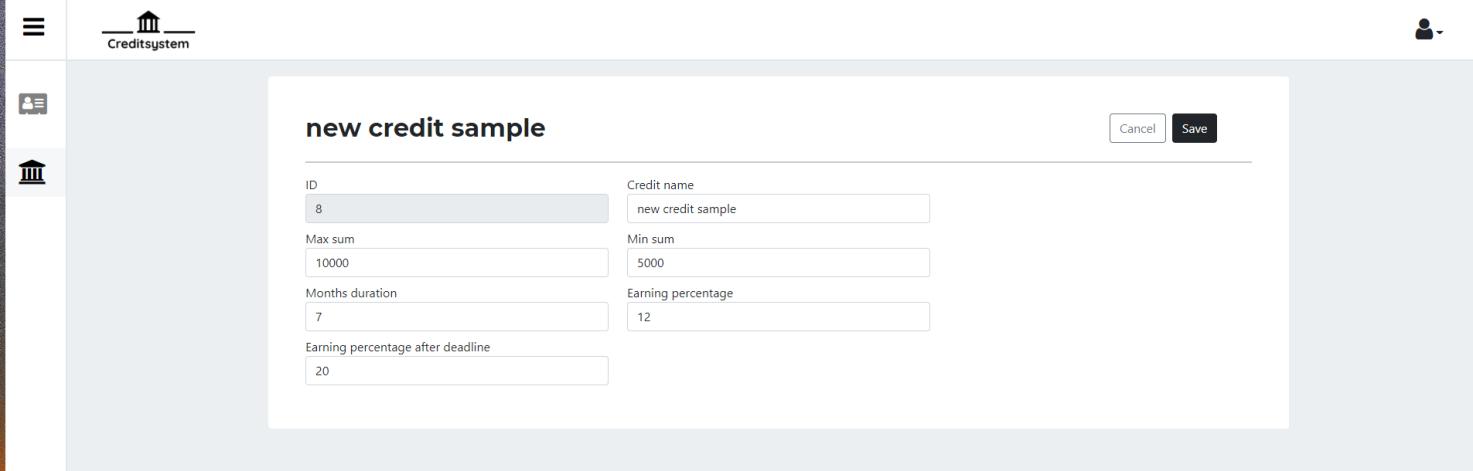
*Рис.16 Сторінка заключеного контракту*



*Рис.17 Сторінка заключання договору (позики)*



*Рис.18 перегляд набраних кредитів*



*Рис.19 сторінка редагування кредитних умов для оператора банку*

# Висновки

В результаті виконання курсової роботи я вирішив поставлену задачу, використавши підхід описаний у розділах 4-5. Програмна система має змогу бути модернізованою, за рахунок контейнеризації додатку, бекенд-сервіси є масштабованими, і ми з легкістю можемо створювати їх репліки.

Варто звернути увагу на функціонал доступний для користувача, та передбачити інтегрування із певною платіжною системою.

Також хорошим рішенням для хостингу сервісів буде використання одного чи кількох клауд провайдерів, таких як: Microsoft Azure, Google Cloud Platfor, Amazon Web Services.

Звісно, виконавши дану роботу я покращив свої навички у побудові мікросервісних систем, написання фронт-енд додатків, RESTful веб сервісів, та роботою із базою даних, суттєво покращив DevOps навички, контейнеризувавши свої сервіси.

# Список використаних джерел

1. Spring Framework [Електронний ресурс] – режим доступу до ресурсу:

<https://spring.io/projects/spring-framework>

1. Spring Data [Електронний ресурс] – режим доступу до ресурсу:

<https://spring.io/projects/spring-data>

1. Spring Security [Електронний ресурс] – режим доступу до ресурсу:

<https://spring.io/projects/spring-security>

1. Angular vs React vs Vue [Електронний ресурс] – режим доступу до ресурсу <https://www.codeinwp.com/blog/angular-vs-vue-vs-react/>
2. Docker [Електронний ресурс] – режим доступу до ресурсу: <https://docs.docker.com/get-started/>
3. Банківський кредит [Електронний ресурс] – режим доступу до ресурсу: [https://uk.wikipedia.org/wiki/Банківський\_кредит](https://uk.wikipedia.org/wiki/%D0%91%D0%B0%D0%BD%D0%BA%D1%96%D0%B2%D1%81%D1%8C%D0%BA%D0%B8%D0%B9_%D0%BA%D1%80%D0%B5%D0%B4%D0%B8%D1%82)
4. Privat24 [Електронний ресурс] – режим доступу до ресурсу: <https://next.privat24.ua/>
5. Monobank [Електронний ресурс] – режим доступу до ресурсу: <https://www.monobank.ua/?lang=uk>
6. Банк [Електронний ресурс] – режим доступу до ресурсу: <https://en.wikipedia.org/wiki/Bank>
7. Microservice oriented architecture [Електронний ресурс] – режим доступу до ресурсу:

<https://microservices.io/patterns/microservices.html>

# Додатки

Вихідний код контроллерів кастомера:

*credits.CreditController.java*

package com.yakymets.creditsystem.web.controllers.customer.credits;  
  
import com.yakymets.creditsystem.services.DTO.CreditDTO;  
import com.yakymets.creditsystem.services.interfaces.CreditService;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.web.bind.annotation.\*;  
  
import java.math.BigDecimal;  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/customer/credits")  
public class CreditController {  
 private final CreditService creditService;  
 Logger logger = LoggerFactory.*getLogger*(CreditController.class);  
  
 @Autowired  
 public CreditController(CreditService creditService) {  
 this.creditService = creditService;  
 }  
  
 @GetMapping  
 @ResponseStatus(HttpStatus.*OK*)  
 public List<CreditDTO> getAll(@RequestParam(required = false, name = "max-sum") BigDecimal maxSum,  
 @RequestParam(required = false, name = "min-sum") BigDecimal minSum,  
 @RequestParam(required = false, name = "months-duration") Integer monthsDuration,  
 @RequestParam(required = false, name = "earning-percentage") Double earningPercentage,  
 @RequestParam(required = false, name = "earning-percentage-after-deadline") Double earningPercentageAfterDeadline) {  
 List<CreditDTO> credits = creditService.getFilteredCredits(maxSum, minSum, monthsDuration, earningPercentage, earningPercentageAfterDeadline);  
 logger.info("Got all credits for customer");  
 return credits;  
 }  
  
 @GetMapping("/{id}")  
 @ResponseStatus(HttpStatus.*OK*)  
 public CreditDTO get(@PathVariable(value = "id") Long userId) {  
 CreditDTO creditById = creditService.getCreditById(userId);  
 logger.info("Got credit for customer");  
 return creditById;  
 }  
}

*contracts.ContractController.java*

package com.yakymets.creditsystem.web.controllers.customer.contracts;  
  
import com.yakymets.creditsystem.services.DTO.ContractDTO;  
import com.yakymets.creditsystem.services.interfaces.ContractService;  
import com.yakymets.creditsystem.services.interfaces.JwtProvider;  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
import java.util.NoSuchElementException;  
  
@RestController  
@RequestMapping("/api/customer/contracts")  
public class ContractControllerCustomer {  
 private final ContractService contractService;  
 private final JwtProvider jwtProvider;  
 Logger logger = LoggerFactory.*getLogger*(ContractControllerCustomer.class);  
  
 @Autowired  
 public ContractControllerCustomer(ContractService contractService, JwtProvider jwtProvider) {  
 this.contractService = contractService;  
 this.jwtProvider = jwtProvider;  
 }  
  
 @GetMapping  
 @ResponseStatus(HttpStatus.*OK*)  
 public List<ContractDTO> getAll(@RequestHeader("Authorization") String token) {  
 String email = jwtProvider.getEmailFromToken(token.substring(7));  
 List<ContractDTO> contractsByUserEmail = contractService.getContractsByUserEmail(email, false);  
 logger.info("Got contracts by user email");  
 return contractsByUserEmail;  
 }  
  
 @GetMapping("/{id}")  
 @ResponseStatus(HttpStatus.*OK*)  
 public ContractDTO get(@PathVariable(value = "id") Long id) {  
 ContractDTO contract = contractService.getContract(id);  
 logger.info("Got contract");  
 return contract;  
 }  
  
 @PostMapping  
 @ResponseStatus(HttpStatus.*CREATED*)  
 public ContractDTO create(@RequestHeader("Authorization") String token, @RequestBody ContractDTO contractDTO) {  
 String email = jwtProvider.getEmailFromToken(token.substring(7));  
 ContractDTO addContract = contractService.addContract(contractDTO, email);  
 logger.info("Created contract");  
 return addContract;  
 }  
  
 @PutMapping("/{id}")  
 @ResponseStatus(HttpStatus.*OK*)  
 public ContractDTO update(@PathVariable(value = "id") Long id, @RequestBody ContractDTO contractDTO) {  
 ContractDTO updateContract = contractService.updateContract(contractDTO, id);  
 logger.info("updated contract");  
 return updateContract;  
 }  
  
 @ExceptionHandler(NoSuchElementException.class)  
 @ResponseStatus(HttpStatus.*NOT\_FOUND*)  
 public ResponseEntity<String> handleNoSuchElementFoundException(  
 NoSuchElementException exception  
 ) {  
 logger.error("No such contract");  
 return ResponseEntity  
 .*status*(HttpStatus.*NOT\_FOUND*)  
 .body(exception.getMessage());  
 }  
  
*// @DeleteMapping("/delete/{id}")  
// @ResponseStatus(HttpStatus.OK)  
// public ResponseEntity<Long> delete(@PathVariable(value = "id") Long id) {  
// return new ResponseEntity<>(contractService.deleteContract(id), HttpStatus.OK);  
// }*}

Вихідний код контроллерів оператора банку:

*contracts.ContractController.java*

package com.yakymets.creditsystem.web.controllers.creditprovider.contracts;  
  
import com.yakymets.creditsystem.services.DTO.ContractDTO;  
import com.yakymets.creditsystem.services.interfaces.ContractService;  
import com.yakymets.creditsystem.services.interfaces.JwtProvider;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/credit-provider/contracts")  
public class ContractControllerCreditProvider {  
 private final ContractService contractService;  
 private final JwtProvider jwtProvider;  
  
 @Autowired  
 public ContractControllerCreditProvider(ContractService contractService, JwtProvider jwtProvider) {  
 this.contractService = contractService;  
 this.jwtProvider = jwtProvider;  
 }  
  
 @GetMapping  
 @ResponseStatus(HttpStatus.*OK*)  
 public List<ContractDTO> getAll(@RequestHeader("Authorization") String token) {  
 String email = jwtProvider.getEmailFromToken(token.substring(7));  
 return contractService.getContractsByUserEmail(email, true);  
 }  
  
 @GetMapping("/{id}")  
 @ResponseStatus(HttpStatus.*OK*)  
 public ContractDTO get(@PathVariable(value = "id") Long id) {  
 return contractService.getContract(id);  
 }  
}

*сredits.CreditController.java*

package com.yakymets.creditsystem.web.controllers.creditprovider.credits;  
  
import com.yakymets.creditsystem.services.DTO.CreditDTO;  
import com.yakymets.creditsystem.services.interfaces.CreditService;  
import com.yakymets.creditsystem.services.interfaces.JwtProvider;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.http.HttpStatus;  
import org.springframework.http.ResponseEntity;  
import org.springframework.web.bind.annotation.\*;  
  
import java.util.List;  
  
@RestController  
@RequestMapping("/api/credit-provider/credits")  
public class CreditControllerCreditProvider {  
 private final CreditService creditService;  
 private final JwtProvider jwtProvider;  
  
 @Autowired  
 public CreditControllerCreditProvider(CreditService creditService, JwtProvider jwtProvider) {  
 this.creditService = creditService;  
 this.jwtProvider = jwtProvider;  
 }  
  
 @GetMapping  
 @ResponseStatus(HttpStatus.*OK*)  
 public List<CreditDTO> getAll(@RequestHeader("Authorization") String token) {  
 System.*out*.println(token.substring(7));  
 String email = jwtProvider.getEmailFromToken(token.substring(7));  
 return creditService.getAllCreditsByCreditProviderEmail(email);  
 }  
  
 @GetMapping("/{id}")  
 @ResponseStatus(HttpStatus.*OK*)  
 public CreditDTO get(@PathVariable(value = "id") Long id) {  
 return creditService.getCreditById(id);  
 }  
  
 @PutMapping("/{id}")  
 @ResponseStatus(HttpStatus.*OK*)  
 public CreditDTO update(@PathVariable(value = "id") Long id, @RequestBody CreditDTO creditDTO) {  
 return creditService.updateCredit(id, creditDTO);  
 }  
  
 @DeleteMapping("/{id}")  
 @ResponseStatus(HttpStatus.*OK*)  
 public ResponseEntity<Long> delete(@PathVariable(value = "id") Long id) {  
 return new ResponseEntity<>(creditService.deleteCredit(id), HttpStatus.*OK*);  
 }  
  
 @PostMapping  
 @ResponseStatus(HttpStatus.*CREATED*)  
 public CreditDTO addCredit(@RequestHeader("Authorization") String token, @RequestBody CreditDTO creditDTO) {  
 String email = jwtProvider.getEmailFromToken(token.substring(7));  
 return creditService.createCredit(creditDTO, email);  
 }  
}

Контроллер автентифікації

package com.yakymets.creditsystem.web.controllers.creditprovider.auth;  
  
  
import com.yakymets.creditsystem.services.DTO.LoginDTO;  
import com.yakymets.creditsystem.services.DTO.RegisterDTO;  
import com.yakymets.creditsystem.services.DTO.TokenDTO;  
import com.yakymets.creditsystem.services.interfaces.JwtProvider;  
import com.yakymets.creditsystem.services.interfaces.UserService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.web.bind.annotation.PostMapping;  
import org.springframework.web.bind.annotation.RequestBody;  
import org.springframework.web.bind.annotation.RequestMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@RestController  
@RequestMapping("/api/credit-provider/auth")  
public class AuthCreditProviderController {  
  
 private final UserService userService;  
  
 private final JwtProvider jwtProvider;  
  
 @Autowired  
 public AuthCreditProviderController(UserService userService, JwtProvider jwtProvider) {  
 this.userService = userService;  
 this.jwtProvider = jwtProvider;  
 }  
  
 @PostMapping("/login")  
 public TokenDTO login(@RequestBody LoginDTO loginDTO) {  
 LoginDTO user = userService.findUserByEmailAndPassword(loginDTO.getEmail(), loginDTO.getPassword());  
 return jwtProvider.generateToken(user.getEmail());  
 }  
  
 @PostMapping("/register")  
 public Long register(@RequestBody RegisterDTO registerDTO) {  
 return userService.saveUser(registerDTO);  
 }  
}

JWT Filter

package com.yakymets.creditsystem.web.controllers.configuration;  
  
import com.yakymets.creditsystem.services.interfaces.JwtProvider;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;  
import org.springframework.security.core.context.SecurityContextHolder;  
import org.springframework.security.core.userdetails.UserDetails;  
import org.springframework.security.core.userdetails.UserDetailsService;  
import org.springframework.stereotype.Component;  
import org.springframework.web.filter.GenericFilterBean;  
  
import javax.servlet.FilterChain;  
import javax.servlet.ServletException;  
import javax.servlet.ServletRequest;  
import javax.servlet.ServletResponse;  
import javax.servlet.http.HttpServletRequest;  
import java.io.IOException;  
  
import static org.springframework.http.HttpHeaders.*AUTHORIZATION*;  
import static org.springframework.util.StringUtils.*hasText*;  
  
@Component  
public class JwtFilter extends GenericFilterBean {  
  
 private final JwtProvider jwtProvider;  
 private final UserDetailsService customUserDetailsService;  
  
 @Autowired  
 public JwtFilter(JwtProvider jwtProvider, UserDetailsService userDetailsService) {  
 this.jwtProvider = jwtProvider;  
 this.customUserDetailsService = userDetailsService;  
 }  
  
 @Override  
 public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse, FilterChain filterChain) throws IOException, ServletException {  
 String token = getTokenFromRequest((HttpServletRequest) servletRequest);  
 if (token != null && jwtProvider.isValidToken(token)) {  
 String email = jwtProvider.getEmailFromToken(token);  
 UserDetails customUserDetails = customUserDetailsService.loadUserByUsername(email);  
 UsernamePasswordAuthenticationToken auth = new UsernamePasswordAuthenticationToken(customUserDetails, null, customUserDetails.getAuthorities());  
 SecurityContextHolder.*getContext*().setAuthentication(auth);  
 }  
 filterChain.doFilter(servletRequest, servletResponse);  
 }  
  
 private String getTokenFromRequest(HttpServletRequest request) {  
 String bearer = request.getHeader(*AUTHORIZATION*);  
 if (*hasText*(bearer) && bearer.startsWith("Bearer ")) {  
 return bearer.substring(7);  
 }  
 return null;  
 }  
}

SecurityConfig

package com.yakymets.creditsystem.web.controllers.configuration;  
  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.context.annotation.Bean;  
import org.springframework.context.annotation.Configuration;  
import org.springframework.security.config.annotation.web.builders.HttpSecurity;  
import org.springframework.security.config.annotation.web.builders.WebSecurity;  
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;  
import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;  
import org.springframework.security.config.http.SessionCreationPolicy;  
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;  
import org.springframework.web.cors.CorsConfiguration;  
import org.springframework.web.cors.CorsConfigurationSource;  
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;  
  
import java.util.Arrays;  
import java.util.Collections;  
  
  
@Configuration  
@EnableWebSecurity  
public class SecurityConfig extends WebSecurityConfigurerAdapter {  
  
 private final JwtFilter jwtFilter;  
  
 @Autowired  
 public SecurityConfig(JwtFilter jwtFilter) {  
 this.jwtFilter = jwtFilter;  
 }  
  
 @Override  
 protected void configure(HttpSecurity http) throws Exception {  
 http  
 .cors().and()  
 .httpBasic().disable()  
 .csrf().disable()  
 .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.*STATELESS*)  
 .and()  
 .authorizeRequests()  
 .antMatchers("api/credit-provider/\*\*").hasRole("CreditProvider")  
 .and()  
 .addFilterBefore(jwtFilter, UsernamePasswordAuthenticationFilter.class);  
 }  
  
 @Override  
 public void configure(WebSecurity web) {  
 web.ignoring().antMatchers("/api/credit-provider/auth/\*\*");  
 }  
  
 @Bean  
 protected CorsConfigurationSource corsConfigurationSource() {  
 UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();  
 CorsConfiguration corsConfiguration = new CorsConfiguration();  
 corsConfiguration.setAllowedMethods(Arrays.*asList*(  
 "GET",  
 "HEAD",  
 "POST",  
 "PUT",  
 "PATCH",  
 "DELETE",  
 "OPTIONS",  
 "TRACE"));  
 corsConfiguration.setAllowedOrigins(Collections.*singletonList*("\*"));  
 corsConfiguration.setAllowedHeaders(Collections.*singletonList*("\*"));  
 corsConfiguration.setMaxAge(1800L);  
 source.registerCorsConfiguration("/\*\*", corsConfiguration);  
 return source;  
 }  
  
  
}

CreditsService

package com.yakymets.creditsystem.services.implementations;  
  
import com.yakymets.creditsystem.persistence.entities.Credit;  
import com.yakymets.creditsystem.persistence.entities.CreditProvider;  
import com.yakymets.creditsystem.persistence.repositories.CreditProviderRepository;  
import com.yakymets.creditsystem.persistence.repositories.CreditRepository;  
import com.yakymets.creditsystem.services.DTO.CreditDTO;  
import com.yakymets.creditsystem.services.interfaces.CreditService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.math.BigDecimal;  
import java.util.List;  
import java.util.NoSuchElementException;  
import java.util.stream.Collectors;  
import java.util.stream.Stream;  
  
@Service  
public class CreditServiceImpl implements CreditService {  
  
 private final CreditRepository creditRepository;  
 private final CreditProviderRepository creditProviderRepository;  
  
 @Autowired  
 public CreditServiceImpl(CreditRepository creditRepository, CreditProviderRepository creditProviderRepository) {  
 this.creditRepository = creditRepository;  
 this.creditProviderRepository = creditProviderRepository;  
 }  
  
 @Override  
 public List<CreditDTO> getAllCredits() {  
 return creditRepository.findAll().stream().map(CreditDTO::new).collect(Collectors.*toList*());  
 }  
  
 @Override  
 public CreditDTO getCreditById(Long id) {  
 return new CreditDTO(creditRepository.findById(id).orElseThrow(NoSuchElementException::new));  
 }  
  
 @Override  
 public List<CreditDTO> getAllCreditsByCreditProviderEmail(String email) {  
 return creditRepository.findAllByCreditProviderEmail(email)  
 .stream().map(CreditDTO::new)  
 .collect(Collectors.*toList*());  
 }  
  
 @Override  
 public List<CreditDTO> getFilteredCredits(BigDecimal maxSum,  
 BigDecimal minSum,  
 Integer monthsDuration,  
 Double earningPercentage,  
 Double earningPercentageAfterDeadline) {  
  
 Stream<CreditDTO> creditDTOStream = creditRepository.findAll().stream().map(CreditDTO::new);  
 if (maxSum != null)  
 creditDTOStream = creditDTOStream.filter(credit -> credit.getMaxSum().compareTo(maxSum) <= 0);  
 if (minSum != null)  
 creditDTOStream = creditDTOStream.filter(credit -> credit.getMinSum().compareTo(minSum) >= 0);  
 if (monthsDuration != null)  
 creditDTOStream = creditDTOStream.filter(credit -> credit.getMonthsDuration() == monthsDuration);  
 if (earningPercentage != null)  
 creditDTOStream = creditDTOStream.filter(credit -> credit.getEarningPercentage() <= earningPercentage);  
 if (earningPercentageAfterDeadline != null)  
 creditDTOStream = creditDTOStream.filter(credit -> credit.getEarningPercentageAfterDeadline() <= earningPercentageAfterDeadline);  
 return creditDTOStream.collect(Collectors.*toList*());  
 }  
  
 @Override  
 public CreditDTO updateCredit(Long id, CreditDTO creditDTO) {  
 Credit credit = creditRepository.findById(id).orElseThrow(NoSuchElementException::new);  
 credit.setName(creditDTO.getName());  
 credit.setMinSum(creditDTO.getMinSum());  
 credit.setMaxSum(creditDTO.getMaxSum());  
 credit.setEarningPercentage(creditDTO.getEarningPercentage());  
 credit.setEarningPercentageAfterDeadline(creditDTO.getEarningPercentageAfterDeadline());  
 credit.setMonthsDuration(creditDTO.getMonthsDuration());  
 creditRepository.save(credit);  
 return new CreditDTO(credit);  
 }  
  
 @Override  
 public Long deleteCredit(Long id) {  
 creditRepository.deleteById(id);  
 return id;  
 }  
  
 @Override  
 public CreditDTO createCredit(CreditDTO creditDTO, String creditProviderEmail) {  
 Credit credit = new Credit();  
 CreditProvider creditProvider = creditProviderRepository.findCreditProviderByEmail(creditProviderEmail);  
 credit.setName(creditDTO.getName());  
 credit.setMinSum(creditDTO.getMinSum());  
 credit.setMaxSum(creditDTO.getMaxSum());  
 credit.setEarningPercentage(creditDTO.getEarningPercentage());  
 credit.setEarningPercentageAfterDeadline(creditDTO.getEarningPercentageAfterDeadline());  
 credit.setMonthsDuration(creditDTO.getMonthsDuration());  
 creditProvider.addCredit(credit);  
 creditProviderRepository.save(creditProvider);  
 return new CreditDTO(credit);  
 }  
}

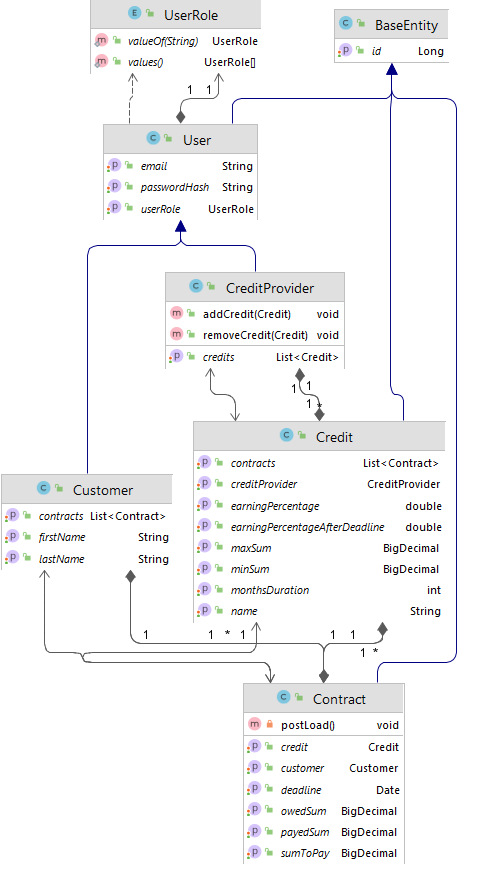
ContractsService

package com.yakymets.creditsystem.services.implementations;  
  
import com.yakymets.creditsystem.persistence.entities.Contract;  
import com.yakymets.creditsystem.persistence.entities.Credit;  
import com.yakymets.creditsystem.persistence.entities.Customer;  
import com.yakymets.creditsystem.persistence.repositories.ContractRepository;  
import com.yakymets.creditsystem.persistence.repositories.ContractRepositoryCustom;  
import com.yakymets.creditsystem.persistence.repositories.CreditRepository;  
import com.yakymets.creditsystem.persistence.repositories.CustomerRepository;  
import com.yakymets.creditsystem.services.DTO.ContractDTO;  
import com.yakymets.creditsystem.services.interfaces.DateOperationsService;  
import com.yakymets.creditsystem.services.interfaces.ContractService;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.math.BigDecimal;  
import java.util.Date;  
import java.util.List;  
import java.util.NoSuchElementException;  
import java.util.stream.Collectors;  
  
@Service  
public class ContractServiceImpl implements ContractService {  
 private final CustomerRepository customerRepository;  
 private final CreditRepository creditRepository;  
 private final ContractRepository contractRepository;  
 private final DateOperationsService dateOperationsService;  
 private final ContractRepositoryCustom contractRepositoryCustom;  
  
 @Autowired  
 public ContractServiceImpl(CustomerRepository customerRepository, CreditRepository creditRepository, ContractRepository contractRepository, DateOperationsService dateOperationsService, ContractRepositoryCustom contractRepositoryCustom) {  
 this.customerRepository = customerRepository;  
 this.creditRepository = creditRepository;  
 this.contractRepository = contractRepository;  
 this.dateOperationsService = dateOperationsService;  
 this.contractRepositoryCustom = contractRepositoryCustom;  
 }  
  
 @Override  
 public ContractDTO addContract(ContractDTO contractDTO, String borrowerEmail) {  
 Customer customer = customerRepository.findCustomerByEmail(borrowerEmail);  
 Credit credit = creditRepository.getById(contractDTO.getCredit().getId());  
 Contract contract = new Contract();  
 contract.setCredit(credit);  
 contract.setCustomer(customer);  
 contract.setPayedSum(new BigDecimal(0));  
 contract.setDeadline(dateOperationsService.addMonths(new Date(), credit.getMonthsDuration()));  
 contract.setOwedSum(contractDTO.getOwedSum());  
 contractRepository.save(contract);  
 return new ContractDTO(contract);  
 }  
  
 @Override  
 public ContractDTO updateContract(ContractDTO contractDTO, Long contractId) {  
 Contract contract = contractRepository.findById(contractId).orElseThrow(NoSuchElementException::new);  
 contract.setOwedSum(contractDTO.getOwedSum());  
 contract.setPayedSum(contractDTO.getPayedSum());  
 contractRepository.save(contract);  
 return new ContractDTO(contract);  
 }  
  
*// @Override  
// public Long deleteContract(Long contractId) {  
// return null;  
// }* @Override  
 public ContractDTO getContract(Long id) {  
 return new ContractDTO(contractRepository.findById(id).orElseThrow(NoSuchElementException::new));  
 }  
  
 @Override  
 public List<ContractDTO> getContractsByUserEmail(String userEmail, boolean isCreditProvider) {  
 if (isCreditProvider) {  
 return contractRepositoryCustom.findAllByCreditProviderEmail(userEmail)  
 .stream().map(ContractDTO::new).collect(Collectors.*toList*());  
 }  
 return contractRepository.findAllByCustomerEmail(userEmail).stream().map(ContractDTO::new).collect(Collectors.*toList*());  
 }  
  
  
  
}

JWTProvider Service

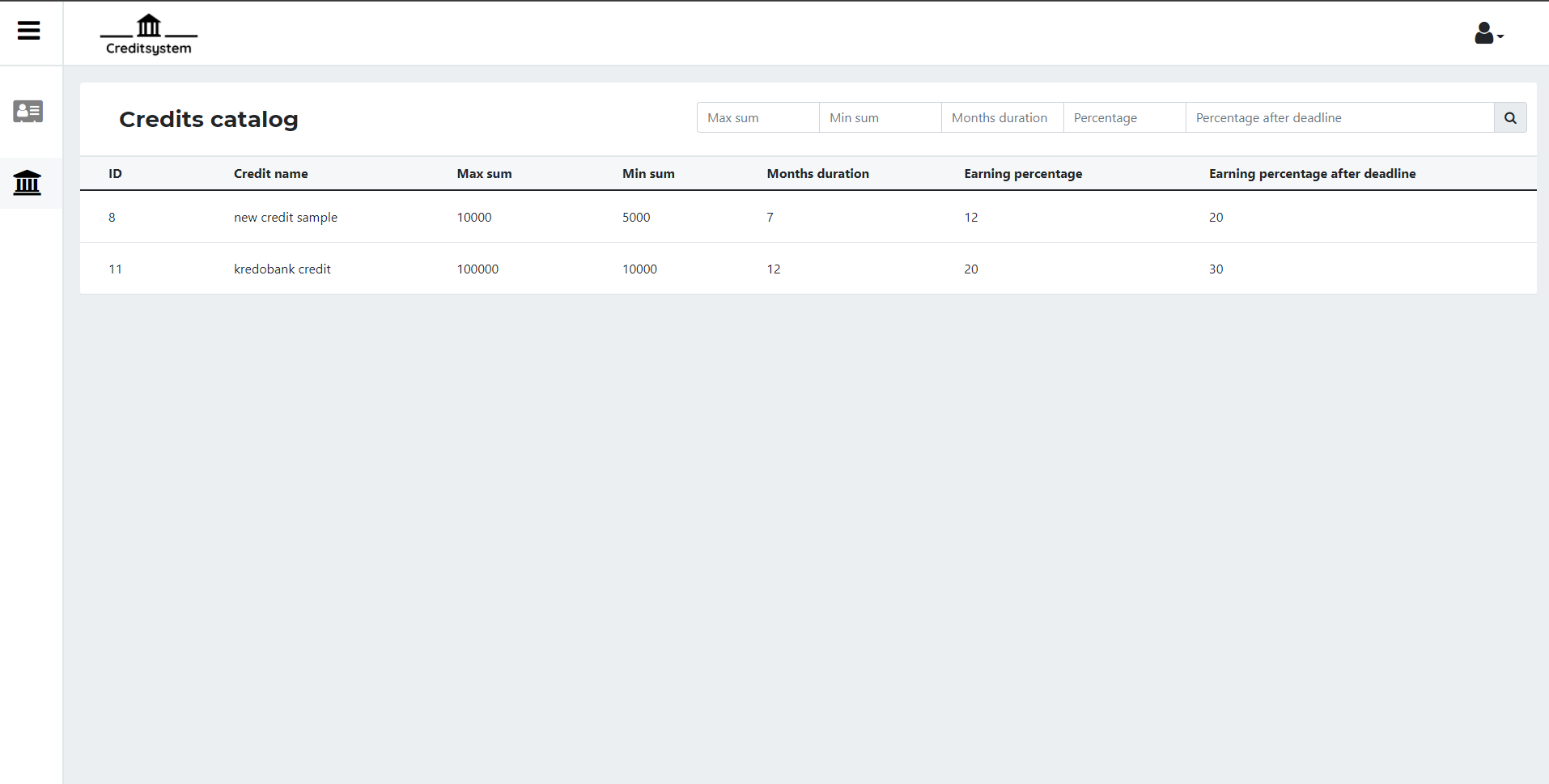
package com.yakymets.creditsystem.services.implementations.security;  
  
import com.yakymets.creditsystem.persistence.entities.UserRole;  
import com.yakymets.creditsystem.services.DTO.TokenDTO;  
import com.yakymets.creditsystem.services.interfaces.JwtProvider;  
import com.yakymets.creditsystem.services.interfaces.UserService;  
import io.jsonwebtoken.Claims;  
import io.jsonwebtoken.Jwts;  
import io.jsonwebtoken.SignatureAlgorithm;  
import org.springframework.beans.factory.annotation.Autowired;  
import org.springframework.stereotype.Service;  
  
import java.time.LocalDate;  
import java.time.ZoneId;  
import java.util.Date;  
  
@Service  
public class JwtProviderImpl implements JwtProvider {  
  
 private final String jwtSecret = "somesupersecretvalue";  
 private final UserService userService;  
  
 @Autowired  
 public JwtProviderImpl(UserService userService) {  
 this.userService = userService;  
 }  
  
 @Override  
 public TokenDTO generateToken(String email) {  
 Date date = Date.*from*(LocalDate.*now*().plusDays(15).atStartOfDay(ZoneId.*systemDefault*()).toInstant());  
 UserRole userRole = userService.findUserByEmail(email).getUserRole();  
 String token = Jwts.*builder*()  
 .setSubject(email)  
 .setExpiration(date)  
 .signWith(SignatureAlgorithm.*HS512*, jwtSecret)  
 .claim("userRole", userRole.name())  
 .compact();  
 return new TokenDTO(token);  
 }  
  
 @Override  
 public boolean isValidToken(String token) {  
 try {  
 Jwts.*parser*().setSigningKey(jwtSecret).parseClaimsJws(token);  
 return true;  
 } catch (Exception exception) {  
 return false;  
 }  
 }  
  
 @Override  
 public String getEmailFromToken(String token) {  
 Claims claims = Jwts.*parser*().setSigningKey(jwtSecret).parseClaimsJws(token).getBody();  
 return claims.getSubject();  
 }  
}

Class diagram for enttites

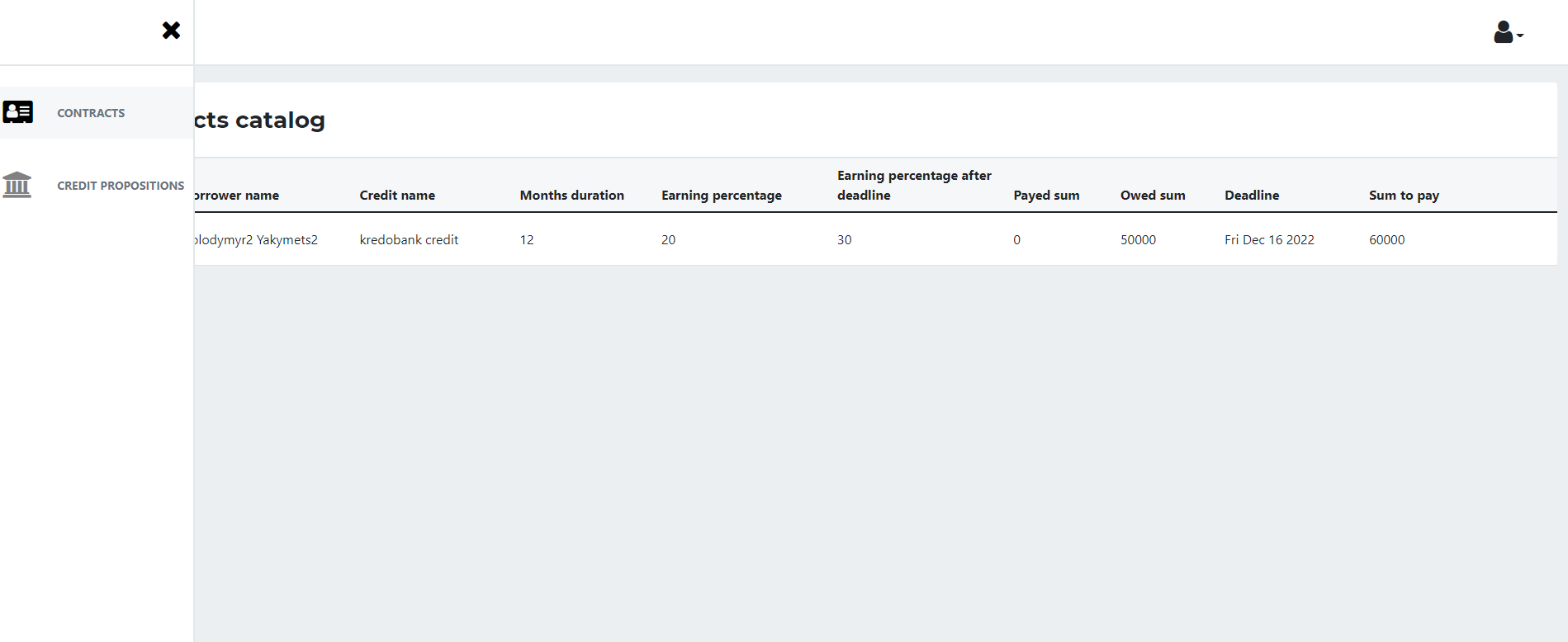


*Рис.20 Діграма класів для об’єктного представлення моделей бази даних*

Скріншоти програми



*Рис.21 Скріншот програми*



*Рис.22 Скріншот програми*