КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп’ютерних наук та кібернетики

Кафедра математичної інформатики

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття ступеня магістра

за спеціальністю 122 Комп'ютерні науки

на тему:

Розробка веб-застосунку для інвестування

Виконав студент 2-го курсу магістратури

Всеволод РЕШЕТНІКОВ _____

(підпис)

Науковий керівник:

асистент, кандидат технічних наук

Олексій ФЕДОРУС _____

(підпис)

Роботу розглянуто й допущено до захисту на засіданні кафедри математичної інформатики

«___» _____ 20___р.

протокол № _____

Завідувач кафедри

Василь ТЕРЕЩЕНКО _____

(підпис)

Київ – 2023

РЕФЕРАТ

Обсяг роботи складає 60 сторінок, в ній містяться 29 ілюстрацій, з яких 14 прикладів коду, додатку з посиланнями на код, використано 27 джерел посилань. Робота складається зі вступу, 3 розділів, висновків, списку використаних джерел.

Об’єктом роботи є побудова веб-застосунку для інвестування у проєкти, у якому компанії можуть реєструватися та додавати свої проєкти, а інвестори можуть знаходити цікаві для себе стартапи для інвестицій.

Цілями виконання роботи є:

– систематизація, закріплення та розширення теоретичних та практичних знань, застосування їх у розв’язанні конкретних фахових задач;

– розвиток навичок самостійної роботи;

– оволодіння методиками проведення досліджень та інших форм роботи з розв'язання поставлених проблем;

– аналіз та побудова бізнес-логіки застосунку, зручного для користувачів, який відповідає вимогам;

– розробка серверної за клієнтської частини застосунку;

– подальший аналіз та майбутній розвиток продукту.

Метою роботи є створення веб-версії застосунку, в якому компанії та інвестори зможуть знаходити одне одного для подальшої співпраці. Таким чином, кожна компанія здатна зареєструвати свій новий продукт на даній платформі з метою залучення інвестиції у стартапи, а кожний інвестор має змогу розглядати пропозиції на ринку та обирати для себе проєкти задля інвестицій та отримання прибутку.

Інструментами розробки є інтегроване середовище IntelliJ IDEA, система контролю версій Git та MySQL Connector для використання бази даних.

Результати роботи: проведений аналіз технологій, обрано мови програмування для серверної та клієнтської частини, обрано також систему управління базою даних, виконана попередня постановка задачі, розроблений дизайн продукту, виконано планування прикладного програмного інтерфейсу, а також саме розробка back-end та front-end частин застосунку, проведено unit-тестування та інтеграційне тестування, інтегровані між собою серверна та клієнтська частини.

У ході роботи були вивчені та використані різні технології, такі як мови програмування Java 8 для розробки серверної частини та Javascript для клієнтської частини застосунку, залучено такі бібліотеки та фреймворки як Spring Framework & Spring Boot, React.js, використовується база даних MySQL та бібліотека Hibernate ORM для простоти написання запитів у коді, а також JSON Web Token (JWT) та Spring Security для забезпечення безпеки запитів. Для тестування використовувалися бібліотеки JUnit та Spring Testing Library.

Також показаний більш детальний процес роботи застосунку завдяки утиліті Postman для відправлення запитів.

ЗМІСТ

ВСТУП

Сьогодні інвестування, як і інші соціально-економічні явища, розвивається за допомогою Інтернету. Високорозвинені країни світу мають професійні інвестиційні фонди, які мають великі обсяги капіталу і можуть здійснювати різні види інвестування. Українські інвестиційні фонди поки не досягли такого рівня розвитку через нестачу інвестиційних коштів і несприятливий інвестиційний клімат. Наукова література недостатньо приділяє увагу розгляданню доходності вітчизняних інвестиційних фондів та їхньої прибутковості та стабільності.

Одним із способів залучення фінансів на проєкти є створення веб-застосунку з використанням інноваційних технологій автоматизації програмних додатків. Застосунок "InApp" розроблено з використанням мови програмування Java та REST API, що допомагає підтримувати будь-яке front-end середовище, а також з застосуванням різних фреймворків та додаткових програм. Інтерфейс користувача розроблено з використанням Javascript, HTML та CSS, а також фреймворком React.js. Основна мета дипломної роботи полягає в розробці такого застосунку, який допоможе втілювати перспективні ідеї в реальність шляхом збору фінансів на проєкти.

Відповідно до цього на шляху досягнення мети будуть вирішені та обґрунтовані такі задачі:

– Обґрунтування вибору мов програмування та середовищ розробки back-end та front-end частин;
– Дослідження існуючих технологій та рішень для створення веб-застосувань через REST API
– Створення use-case діаграми для демонстрування інтерфейсу користувача на шляху розробки продукту
– Створення схеми бази даних для кращого планування back-end, поєднання серверу з системою управління базою даних
– Розробка застосунку, зокрема, back-end та front-end частин на основі мов Java та фреймворків React.js та Spring
– Аналіз розробленого програмного продукту
– Unit-тестування серверної частини коду та мануальне тестування за допомогою утиліти Postman
– Демонстрація роботи застосунку
– Можливості розгорнення застосунку на веб-сервері для публічного доступу
– Подальщі можливості покращення логіки застосунку та інтерфейсу користувача
– Висновки стосовно виконаної роботи

РОЗДІЛ 1. Обґрунтування теми та опис концепції

1.1 Передумови створення застосунку

Розробка веб-застосунку для інвестування - це актуальна тема в сучасному світі, оскільки інвестування стало одним з ключових способів збільшення капіталу. Зростання популярності інвестування в останні роки відбувається в контексті зростання цифрової економіки та доступності інформації, що дозволяє інвесторам шукати нові можливості та диверсифікувати свої інвестиції.

Основними перевагами веб-додатків для інвестування є:

- Зручність та доступність. Користувач може здійснювати операції з будь-якого місця, де є доступ до Інтернету.
- Швидкість та ефективність. Веб-застосунок дозволяє інвесторам швидко здійснювати операції та отримувати актуальну інформацію про свої інвестиції в режимі реального часу.
- Доступ до нових ринків. Завдяки технологіям, інвестори мають доступ до нових ринків, які раніше були недоступними через нестачу інформації, та можуть здійснювати операції з мінімальними комісійними.

1.2 Порівняння з існуючими проєктами

Одним з аналогів веб-застосунку для інвестування є Kickstarter. Kickstarter є онлайн-платформою для збору коштів на різноманітні проєкти, які зазвичай є культурними, технологічними, або науковими ініціативами. Основна мета Kickstarter полягає у зборі коштів на підтримку цих проєктів, а не на отримання прибутку від інвестування.

Однак, веб-застосунок для інвестування відрізняється від Kickstarter за наступними параметрами:

- Орієнтація на прибуток. Основною метою інвестора в веб - застосунку для інвестування є отримання прибутку від своїх інвестицій, а не просто підтримка проєкту.
- Більш широкий спектр інвестиційних можливостей. Веб-застосунок для інвестування може пропонувати інвесторам різноманітні інструменти для інвестування, такі як акції, облігації, фонди тощо. В той же час, Kickstarter зазвичай обмежується підтримкою проєктів з досить конкретними цілями та обмеженим переліком опцій підтримки.
- Більш серйозна система безпеки та захисту даних. Оскільки інвестування має стосуватись значно більших сум грошей, ніж підтримка проєкту на Kickstarter, веб-застосунок повинен забезпечувати високий рівень захисту даних та безпеки операцій.

Отже, розробка веб-застосунку для інвестування є актуальною темою на сьогоднішній день, оскільки дозволяє інвесторам отримувати доступ до нових можливостей та швидко здійснювати операції з мінімальними витратами. Веб-застосунок для інвестування має ряд переваг порівняно з аналогами, такими як Kickstarter, зокрема, орієнтацію на прибуток, більший спектр інвестиційних можливостей та вищий рівень захисту даних та безпеки операцій.

1.3 Побудова основної частини логіки, яку бачить користувач

Спочатку була розроблена Use-case діаграма зі схематичним дизайном, яку можна побачити на рисунку 1

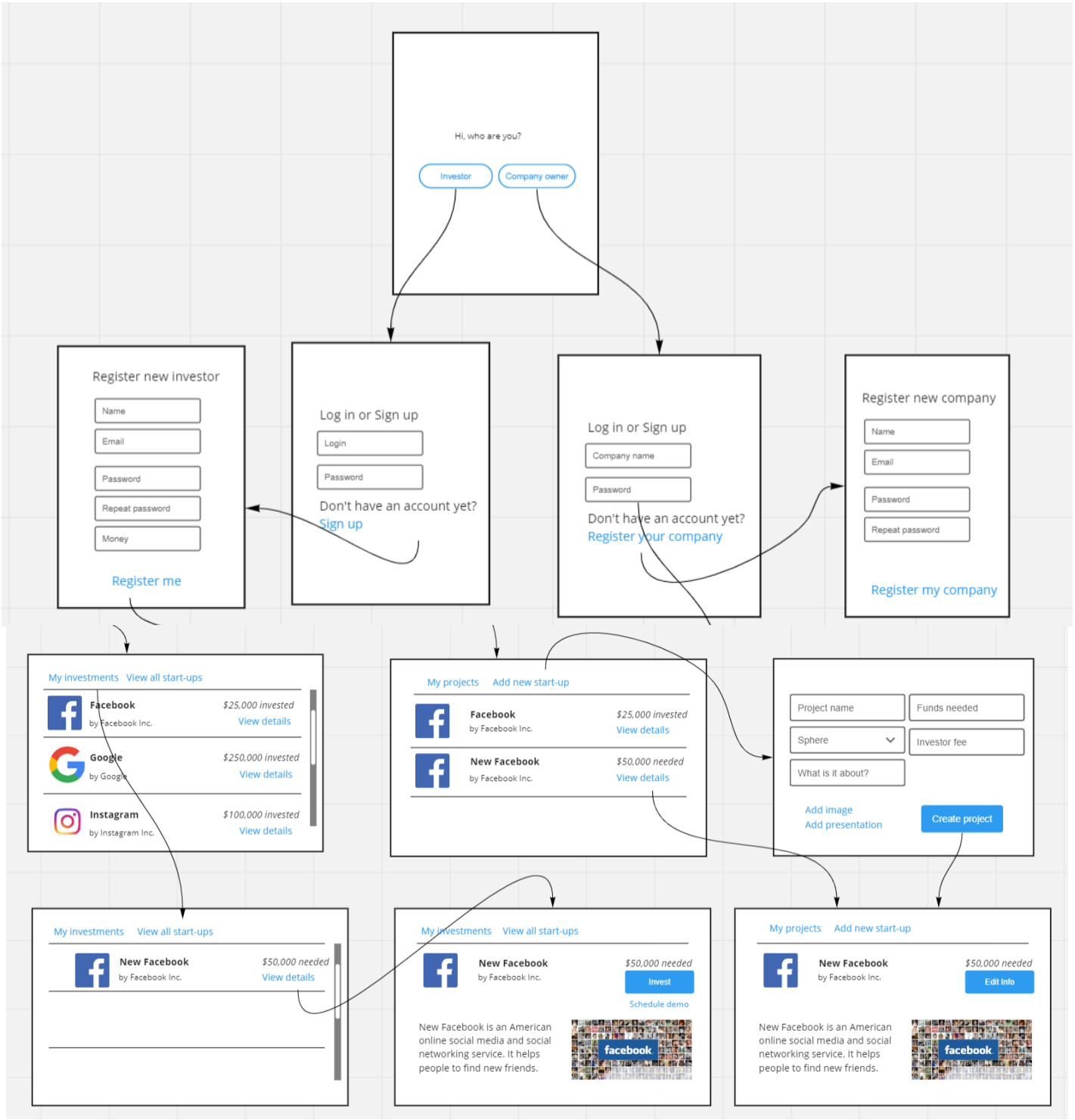

Рисунок 1 – Use-case діаграма

РОЗДІЛ 2. Використання технологій для розробки застосунку

2.1 Back-end

Back-end - це частина програмного забезпечення, яка відповідає за обробку даних, логіку застосунку та комунікацію з базою даних та іншими системами. Back-end забезпечує функціональність та безпеку веб-застосунку та забезпечує співпрацю між користувачем та front-end частиною. [1]

Одними з основних мов програмування для back-end є Java, C++, Python, Ruby, PHP та інші. Для зберігання даних та управління ними використовуються бази даних, такі як MySQL, PostgreSQL, Oracle та інші. Для забезпечення безпеки та захисту використовуються протоколи шифрування, такі як SSL / TLS. [2]

Back-end зазвичай складається з трьох головних компонентів:

- Сервер: це програмне забезпечення, яке відповідає за обробку запитів користувачів та відправку відповідей на ці запити.
- База даних: це програмне забезпечення, яке забезпечує зберігання та управління даними, що використовуються в застосунку.
- API: це інтерфейс програмування застосунку, який дозволяє спілкуватися з іншими застосунками та сервісами.

При розробці back-end зазвичай використовуються такі фреймворки, як Spring, Django, Ruby on Rails та інші, що значно спрощують процес розробки та забезпечують високу продуктивність.

Отже, back-end є важливою складовою веб-застосунку, яка забезпечує функціональність та безпеку. Він складається з сервера, бази даних та API та використовується для обробки даних та логіки застосунку. Використання фреймворків спрощує процес розробки та забезпечує високу продуктивність та безпеку. [3]

2.2 Java як основна мова програмування

Java є однією з найпопулярніших мов програмування, що використовуються для розробки веб-додатків. Її висока продуктивність, гнучкість та безпека роблять її зручним вибором для розробників, які прагнуть створити стійкий, надійний та масштабований веб-застосунок. У цій статті ми розглянемо переваги використання мови Java для розробки веб-застосунку та те, як вона реалізує принципи SOLID development. [4]

Переваги використання мови Java для розробки веб-застосунку

- Висока продуктивність: Java має вбудовану оптимізацію пам'яті та може запускати багатопотокові операції, що забезпечує швидку обробку даних та високу продуктивність.
- Безпека: Java має вбудовану систему безпеки, що дозволяє запобігти атакам на веб-застосунок та захистити дані користувачів від зловмисних атак.
- Масштабованість: Java дозволяє створювати веб-додатки, які можуть масштабуватися в залежності від зростання навантаження на сервер.
- Гнучкість: Java має багато фреймворків, які дозволяють розробникам вибирати той, який найкраще підходить для їхніх потреб.
- Широкий спектр бібліотек: Java має велику кількість бібліотек, що дозволяє розробникам швидко та ефективно вирішувати завдання.
- Кросплатформенність: Java працює на різних операційних системах та платформах, що дозволяє запускати веб-додатки на різних пристроях.

Ще однією вагомою перевагою Java є можливість використання великої кількості різноманітних фреймворків та бібліотек, що робить розробку швидшою та ефективнішою. Наприклад, для розробки веб-додатків часто використовують такі фреймворки як Spring та Hibernate. Spring надає можливість швидко розробляти високоякісні додатки, забезпечуючи багато функціональності та можливостей для розширення. Hibernate, у свою чергу, є потужною бібліотекою, яка дозволяє взаємодіяти з базою даних у зручний та простий спосіб.

Java також підтримує принципи SOLID development, що робить її популярною серед розробників. SOLID - це абревіатура від п'яти принципів, що використовуються для розробки програмного забезпечення:

- Single Responsibility
- Open-Closed
- Liskov Substitution
- Interface Segregation
- Dependency Inversion

Ці принципи допомагають розробникам створювати програмне забезпечення, яке легко змінювати та розширювати.

Java підтримує ці принципи шляхом використання інтерфейсів та абстракцій. Це дозволяє створювати код, який може бути легко змінений та розширений у майбутньому, збільшуючи його масштабованість та ефективність. Наприклад, використання інтерфейсів дозволяє розділити код на окремі модулі, які можуть бути розроблені та підтримувані окремо один від одного. [5]

2.3 Spring Framework та Spring Boot

Spring Framework та Spring Boot - це інструменти для розробки веб-додатків на мові Java. Spring Framework є потужною платформою, яка надає велику кількість функцій та інструментів для розробки веб-додатків, а Spring Boot - це додатковий інструмент, який дозволяє зменшити час та зусилля, потрібні для розгортання застосунку. [6]

Особливості Spring Framework:

- Інверсія контролю та впровадження залежностей (Inversion of Control and Dependency Injection). Spring Framework дозволяє розробникам розглядати бізнес-логіку застосунку як набір компонентів, які можна змінювати та розширювати без необхідності вносити зміни в код. Інверсія контролю та впровадження залежностей дозволяє вирішити проблему залежності компонентів один від одного, що зроблює код більш модульним та легко зрозумілим.
- Аспекти (Aspects). Spring Framework надає підтримку для аспектно-орієнтованого програмування, що дозволяє відокремити крос-концерні питання (cross-cutting concerns) від основного коду застосунку.
- ORM (Object-Relational Mapping). Spring Framework надає підтримку ORM, який дозволяє розробникам працювати з базою даних, використовуючи об'єктно-орієнтований підхід, що спрощує роботу з базою даних та зменшує кількість повторюваного коду.
- Модульність та розширюваність. Spring Framework дозволяє розробникам розділити застосунок на модулі та компоненти, що забезпечує легкість розробки та розширення функціоналу застосунку.

В проєкті використовується фреймворк Spring версії Boot 2.0. Даний фреймворк забезпечує комплексну модель розробки і конфігурації застосунку створеному на Java, який використовується на будь-яких платформах.

Spring Framework є фреймворком для розробки додатків на мові Java, який надає широкий спектр можливостей, необхідних для розробки веб-додатків та багатокористувацьких застосунків. Spring Framework забезпечує високу продуктивність, високу міцність і масштабованість.

Spring Framework складається з модулів, що можуть використовуватися для різних завдань розробки. Наприклад, модуль Spring MVC використовується для розробки веб-додатків, модуль Spring Data допомагає в роботі з базами даних, а модуль Spring Security забезпечує безпеку веб-додатків. [7]

Ключовий елемент Spring - підтримка інфраструктури на рівні програми: основна увага приділяється оформленню застосунку, тому розробка зосереджується саме на логіці без зайвих налаштувань в залежності від виконуваного завдання. Можливості в даного фреймворка є такими:

- Впровадження залежностей.
- Аспектно-орієнтоване програмування, включаючи декларативне управління транзакціями.
- Створення Spring MVC web-додатків і RESTful web-сервісів.
- Початкова підтримка JDBC, JPA, JMS.

Spring Boot володіє великим функціоналом, але його найбільш значущими особливостями є: управління залежностями, автоматична конфігурація і вбудовані контейнери сервлетів. [8]

Щоб прискорити процес управління залежностями, Spring Boot неявно упаковує необхідні сторонні залежності для кожного типу додатки на основі Spring і надає їх розробнику за допомогою так званих starter-пакетів (spring-boot-starter-web, spring-boot-starter-data-jpa і т . Д.)

Starter-пакети становлять собою набір зручних дескрипторів залежностей, які можна включити в свій застосунок. Це дозволить отримати універсальне рішення для всіх, пов'язаних зі Spring технологій, позбавляючи програміста від зайвого пошуку прикладів коду та завантаження з них необхідних дескрипторів залежностей.

Наприклад, якщо ви хочете почати використовувати Spring Data JPA для доступу до бази даних, просто включіть в свій проєкт залежність spring-boot-starter-data-jpa і все буде готово (вам не доведеться шукати сумісні драйвери баз даних і бібліотеки Hibernate)

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
    <version>1.5.22.RELEASE</version>
</dependency>
```

Рисунок 2 - використання pom.xml конфігурації для бібліотек

Якщо ви хочете створити Spring web-застосунок, просто додайте залежність spring-boot-starter-web, яка підтягне в проєкт все бібліотеки, необхідні для розробки Spring MVC-додатків, таких як spring-webmvc, jackson-json, validation-api і Tomcat. Іншими словами, Spring Boot збирає всі загальні залежності і визначає їх в одному місці, що дозволяє розробникам просто використовувати їх, замість того, щоб винаходити колесо кожен раз, коли вони створюють новий застосунок.

Отже, при використанні Spring Boot, конфігураційний файл pom.xml містить набагато менше рядків, ніж при використанні його в Spring-

застосунках. Друга серйозна можливість Spring Boot - це автоматична конфігурація додатків.

Після вибору відповідного початкового пакету, Spring Boot намагається автоматично налаштувати Spring-застосунок на основі доданих jar-залежностей. Наприклад, якщо додавати Spring-boot-starter-web, Spring Boot автоматично сконфігурує такі зареєстровані біни, як DispatcherServlet, ResourceHandlers, MessageSource. Якщо використовувати spring-boot-starter-jdbc, Spring Boot автоматично реєструє біни DataSource, EntityManagerFactory, TransactionManager і чистить інформацію для підключень до баз даних із файлу application.properties

Якщо ви не збираєтеся використовувати базу даних та не надавати жодних детальних даних, підключених до ручного режиму, Spring Boot автоматично налаштовує базу даних у пам'яті, без будь-якої додаткової конфігурації з вашої сторони (за наявності H2 або HSQL бібліотеки)

Автоматична конфігурація може бути повністю змінена в будь-який момент за допомогою налаштувань програміста.

Основна ідея фреймворку Spring полягає у використанні бінів (Beans) – по суті біни це прості звичайні Java-об’єкти, проте легкість конфігурації робить їх використання дуже зручним. Саме таким чином контейнер сервлетів Spring Boot реалізовує принцип SOLID Dependency Inversion – у нашому випадку ми створюємо усі біни з використання патерну. Singleton – лише один об’єкт кожного класу на весь застосунок, проте Spring також надає можливості створення prototype – (породжуючий патерн, що створює схожі об’єкти, частково копіюючи їх). [9]

```
@Service
public class CompanyService {

    private CompanyRepository companyRepository;
    private UserService userService;

    @Resource
    public void setCompanyRepository(CompanyRepository companyRepository) {
        this.companyRepository = companyRepository;
    }

    @Resource
    public void setUserService(UserService userService) {
        this.userService = userService;
    }
```

Рисунок 3 – використання Spring Beans

У прикладі вище ми використовуємо анотації для конфігурації впровадження залежностей нашого проєкту. Також можливий підхід конфігурації в XML, проте він вважається застарілим та менш зручним.

Анотація @Service створює бін з іменем CompanyService – для його використання нам слід оголосити відповідний клас у іншому класі та впроваджити залежність. Існує 2 види впровадження залежностей – через конструктор або через сеттер (setter) – у нашому випадку ми використовуємо другий варіант – анотація @Resource знайде в нашому контейнері бінів потрібні singleton-біни та впровадить залежності – саме таким чином Spring Boot реалізує один з основних принципів SOLID – dependency inversion.

Spring Boot є одним з найпопулярніших фреймворків для розробки веб-додатків на мові Java. Його популярність полягає в його легкості використання та швидкості розробки. Spring Boot - це надбудова над Spring Framework, яка забезпечує швидку і легку розробку веб-додатків з мінімальними зусиллями. Spring Boot включає в себе всі основні функції Spring Framework, такі як Spring MVC, Spring Data та Spring Security, і робить

їх доступними за замовчуванням без необхідності складати складні конфігураційні файли. [10]

Spring Boot володіє декількома ключовими перевагами, які роблять його одним з найкращих виборів для розробки веб-додатків на Java. Ось деякі з них:

- Швидкість розробки: Spring Boot надає швидкий та ефективний спосіб розробки веб-додатків. Він надає значну кількість автоматизованих конфігурацій та деяких стандартних функцій, що дозволяє зосередитись на бізнес-логіці застосунку, а не на інфраструктурі.
- Висока продуктивність: Spring Boot оптимізований для швидкості та продуктивності, що робить його ідеальним вибором для розробки високопродуктивних веб-додатків. Він використовує високопродуктивні технології, такі як Spring MVC та Spring Data, які дозволяють прискорити розробку та покращити продуктивність застосунку.
- Гнучкість та розширюваність: Spring Boot дозволяє легко налаштовувати та розширювати функціонал застосунку. Він надає широкий спектр плагінів та розширень, які дозволяють розробникам легко додавати нові функції та можливості до застосунку.
- Легке налаштування проєкту та управління: Spring Boot надає легкість управління та налаштування додатків. Він дозволяє легко налаштувати базу даних, безпеку, логування та інші параметри застосунку. Крім того, він має вбудовану консоль управління, що дозволяє з легкістю моніторити та управляти додатком.
- Спрощення процесу розгортання: Spring Boot дозволяє легко розгортати додатки на різних платформах.

2.4 REST API

Також для створення зв'язку використовується архітектурний стиль REST API, який забезпечує CRUD операції з базою даних. Цей підхід обраний через його простоту, прозорість, легкість та зрозумілість.

REST (Representational State Transfer) - це архітектурний стиль для побудови розподілених веб-систем. REST API - це інтерфейс, який використовує REST-архітектуру. REST API дає змогу взаємодіяти з сервером через стандартні HTTP-методи, такі як GET, POST, PUT, PATCH та DELETE. [11]

Основні принципи REST API:

- Кожен ресурс має бути доступний через унікальний URI (Uniform Resource Identifier)
- Клієнт-серверна архітектура: система повинна бути розділена на клієнтів та сервери, які можуть розвиватися незалежно один від одного.
- Використання стандартних HTTP-методів (GET, POST, PUT, PATCH, DELETE) для взаємодії з ресурсами.
- REST API повинен бути без стану, що означає, що сервер не зберігає жодної інформації про клієнта між запитами.

Однією з ключових переваг REST API є його простота використання та можливість інтеграції з іншими системами. REST API може бути використаний для передачі даних між різними платформами, що робить його ідеальним для побудови мікросервісних архітектур. REST API дозволяє зменшити час розробки, спростити інтеграцію з іншими системами та забезпечити стабільну та ефективну взаємодію між різними сервісами.

Якщо розглядати детальніше, то REST (RESTful) – це архітектурний стиль, що базується на загальних принципах організації взаємодії додатків із серверами, що підтримують протокол HTTP. Особливість REST полягає в

тому, що сервер не запам’ятовує стан користувача між запитами - у кожному окремому запиті передається інформація, ідентифікована користувачем, наприклад, маркер, отриманий через OAuth-авторизацію, та всі параметри, необхідні для виконання операцій. [12]

Взаємодія між клієнтом та сервером у REST API відбувається за допомогою HTTP-запитів. У клієнта є можливість робити запити на сервер, який надає необхідну інформацію у відповідь. Для цього використовуються стандартні HTTP-методи:

GET: запит на отримання ресурсу;

POST: запит на створення ресурсу;

PUT: запит на зміну ресурсу;

PATCH: запит на часткову зміну ресурсу

DELETE: запит на видалення ресурсу.

У відповідь на запит сервер повертає HTTP-статус код, який вказує на стан виконання запиту, та відповідь у форматі, вказаному в HTTP-заголовках запиту (наприклад, JSON або XML). Відповідь може містити необхідну інформацію про запрошений ресурс, його стан, а також може бути повернена інформація про можливі наступні кроки.

Для взаємодії з REST API клієнт використовує URL-адресу запиту та відповідний HTTP-метод. Наприклад, клієнт може зробити запит GET на URL-адресу "https://example.com/api/resource/1" для отримання інформації про ресурс з ідентифікатором 1. [13]

Узагальнюючи, взаємодія між клієнтом та сервером у REST API полягає у виконанні запитів за допомогою HTTP-методів та обміні відповідями у визначеному форматі (наприклад, JSON або XML).

2.5 Spring Security

Spring Security є потужним інструментом для забезпечення безпеки веб-додатків на базі Spring Framework, включаючи REST web-сервіси. Одним зі способів, яким можна використовувати Spring Security для забезпечення аутентифікації та авторизації REST web-сервісів, є використання токенів JWT (JSON Web Tokens). [14]

JWT є стандартом відкритого ключа для створення токенів аутентифікації, які можуть бути передані між різними системами. Вони складаються з трьох частин: заголовок, тіло і підпис. Заголовок містить тип токена і алгоритм шифрування, тіло містить корисну інформацію про користувача, таку як ім'я, ідентифікатор, роль тощо, а підпис підтверджує, що токен був створений з використанням секретного ключа.

Однією з особливостей JWT є те, що вони можуть зберігатись на клієнтському боці, що дозволяє знизити навантаження на сервер при кожному запиті. Крім того, вони можуть бути використані для розподіленої аутентифікації, тобто для забезпечення доступу до кількох додатків з використанням одного токена.

Spring Security надає підтримку JWT, що дозволяє легко використовувати їх для забезпечення безпеки REST web-сервісів. Для цього необхідно налаштувати Spring Security таким чином, щоб він використовував JWT для аутентифікації та авторизації запитів. Це може бути досягнуто за допомогою класу JwtAuthenticationFilter, який перевіряє наявність токена у заголовку запиту і, якщо він присутній, перевіряє його на валідність та наявність прав доступу до запиту.

Використання JWT для автентифікації та авторизації дозволяє зберігати стан сесії на стороні клієнта, що зменшує навантаження на сервер

та дозволяє підтримувати масштабованість системи. JWT зберігає всю необхідну інформацію про користувача у зашифрованому вигляді, що забезпечує безпеку передачі даних.

Для роботи з JWT в Spring Security використовуються фільтри авторизації та аутентифікації. Фільтр аутентифікації (AuthenticationFilter) відповідає за перевірку автентичності та створення JWT, який повертається клієнту. Фільтр авторизації (AuthorizationFilter) перевіряє наявність JWT у запиті та проводить перевірку прав доступу до запиту.

Однією з переваг використання Spring Security є можливість забезпечити безпеку REST API на рівні URL-шляхів, що забезпечує високий рівень гранулярності доступу до ресурсів. Також, Spring Security дозволяє налаштовувати різні рівні безпеки для різних частин застосунку.

Spring Security - це framework, який зосереджений на наданні як аутентифікації, так і авторизації застосункам Java. Як і всі проєкти Spring, реальна сила Spring Security полягає в тому, наскільки легко її можна розширити для задоволення спеціальних вимог. [15]

Особливості

- Повна та розширена підтримка як для аутентифікації, так і для авторизації
- Захист від атак, таких як виправлення сеансу, джек-джей, підробка веб-сайтів тощо
- Інтеграція API сервлетів
- Необов’язкова інтеграція з Spring Web MVC – таким чином, можна використовувати і для REST API, як ми власне і робимо.

Для тестування API в розробці проєкта було використано сервіс Postman. В даному програмному забезпеченні можливо вручну надсилати запити на сервер та продивлятися результат виконання. За допомогою такого сервісу пришвидшується робота з даними та перевіркою сервера на наявність помилок.

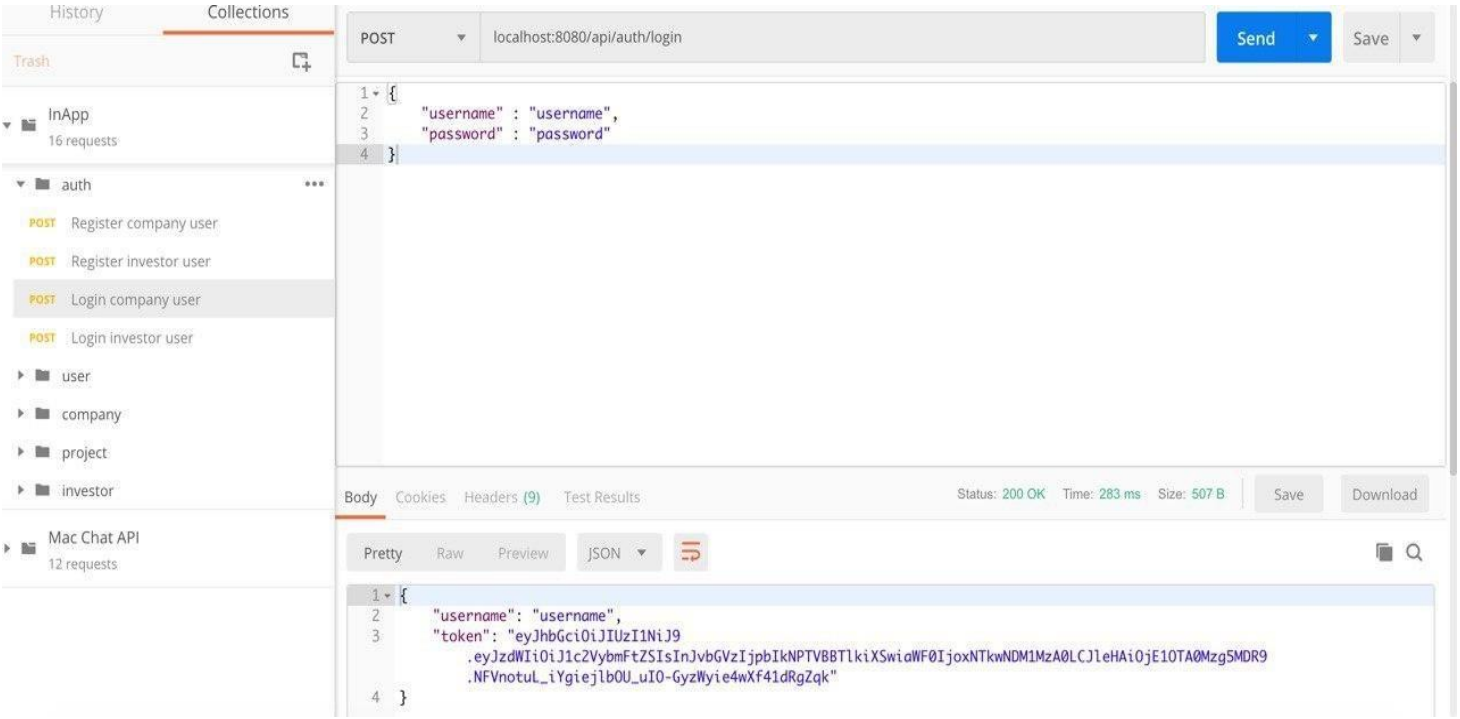

Рисунок 4 - Інтерфейс Postman та приклад взаємодії з даними.

2.6 JSON Web Token Security

Оскільки в застосунку реалізований доступ до бази даних, то маємо забезпечити безпеку, яка реалізована через токен JWT, що використовується для авторизації та аутентифікації. Він містить у собі 3 частини: [16]

1) Header - це JSON об'єкт, який містить метадані про те, як підписується JWT. Зазвичай в ньому вказується алгоритм підпису (наприклад, HMAC, SHA256 або RSA) та тип токену (Bearer).
2) Payload - основна частина, яка представляє собою JSON об'єкт, що містить корисну інформацію, що передається між сторонами. В ньому можуть міститись дані про користувача, які можуть використовуватись для авторизації та іншої інформації, що стосується застосунку.
3) Signature - підпис, який неможливо розшифрувати, не знаючи секретного ключа, що схований на сервері. Signature генерується за допомогою алгоритму підпису, вказаного в header. Підпис складається з base64-кодованого значення header та payload, які додатково підписуються секретним ключем або приватним ключем RSA.

JWT може бути використаний для передачі даних про користувача між клієнтом та сервером в безпечному форматі. Для прикладу, при успішній аутентифікації користувача на сервері, сервер може згенерувати JWT токен та повернути його клієнту. Клієнт може зберігати JWT токен у локальному сховищі та використовувати його при кожному запиті до сервера для авторизації. Сервер може перевірити правильність токену та розшифрувати payload, щоб отримати додаткову інформацію про користувача. Для отримання JWT користувач робить запит на спеціальний endpoint (login). Зрозуміло, що пароль не передається явно, а шифрується за допомогою SHA-256 та у базі зберігається його зашифроване значення. [17]

Якщо користувач з таким ім’ям та паролем міститься у базі данних, то у відповіді від серверу приходить JWT – JSON Web Token і надалі користувач має відправляти усі запити, підставляючи у header Authorization значення

```
@RequestMapping(method = RequestMethod.POST, value = "/login")
public ResponseEntity login(@RequestBody LoginForm loginForm) {
   String username = loginForm.getUsername();
   UserModel user = userService.getUserByUsername(username)
         .orElseThrow(() -> new UsernameNotFoundException("User not found with username: " + username));
   authenticationManager.authenticate(           new
UsernamePasswordAuthenticationToken(username, loginForm.getPassword()));
   String token = jwtTokenProvider.createToken(user);


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


   return ResponseEntity.ok(response);
}
```

Рисунок 5 – endpoint для авторизації

Bearer_{{TOKEN}}, де TOKEN і є значення JWT. Цей токен є валідним певний проміжок часу, що задається на сервері (у даному випадку встановлене значення рівне одній годині). Коли час буде вичерпано, потрібно буде зробити ще один запит за новим токеном для даного користувача. Це зроблено для більшої надійності системи.

```
public String createToken(UserModel userModel) {
   Claims  claims  =  Jwts.claims().setSubject(userModel.getUsername());
claims.put("roles", getRoleNames(userModel.getRoles()));
   Date now = new Date();
   Date validity = new Date(now.getTime() + validityInMilliseconds);


   return Jwts.builder()
.setClaims(claims)
         .setIssuedAt(now)
         .setExpiration(validity)
         .signWith(SignatureAlgorithm.HS256, secret)
.compact();
```

Рисунок 6 – клас створення токену для авторизації

Коли значення JWT приходить на сервер, запит спочатку проходить фільтр, створений для security.

```
@Override
public void doFilter(ServletRequest req, ServletResponse res, FilterChain filterChain)
throws IOException, ServletException {


   String token = jwtTokenProvider.resolveToken((HttpServletRequest) req);
if (token != null && jwtTokenProvider.validateToken(token)) {
      Authentication auth = jwtTokenProvider.getAuthentication(token);


      if (auth != null) {
         SecurityContextHolder.getContext().setAuthentication(auth);
      }
   }


   filterChain.doFilter(req, res);
}
```

Рисунок 7 – коду фільтра для JWT

Таким чином, ми використовуємо патерн Chain of Responsibility, що часто використовується у фільтрах запиту – якщо фільтр не відпрацює, то запит буде вважатися невалідним та поверне виключення. Саме за таким принципом JWT унеможливлює зовнішнє втручання.

Також зрозуміло, що певні endpoints мають бути доступні без токену – наприклад, логін та реєстрація. Саме тому ми налаштовуємо конфігурацію за допомогою Spring Security.

```
@Override
protected void configure(HttpSecurity http) throws Exception {
      http
         .httpBasic().disable()
         .csrf().disable().sessionManagement()
         .sessionCreationPolicy(SessionCreationPolicy.STATELESS)
         .and()
         .authorizeRequests()
         .antMatchers(AUTH_ENDPOINT).permitAll()
         .antMatchers(ADMIN_ENDPOINT).hasRole("ADMIN")
         .anyRequest().authenticated()
         .and()
         .apply(new JwtConfigurer(jwtTokenProvider));
}
```

Рисунок 8 – доступ до певних endpoints

2.7 База даних MySQL

В застосунку база даних створена за допомогою сервісу MySQL. MySQL - це реляційна система управління базами даних (RDBMS), яка забезпечує зберігання, організацію та доступ до даних. Вона є однією з найпопулярніших баз даних у світі, використовується великою кількістю веб-додатків та розподілених систем.

Структура MySQL бази даних включає в себе таблиці, поля, індекси та ключі. Для створення бази даних та її структури можна використовувати SQL-запити. MySQL підтримує велику кількість типів даних, включаючи числа, рядки, дати та більш складні типи, такі як JSON. Також MySQL забезпечує різні рівні доступу до даних, які дозволяють забезпечити безпеку та обмеження доступу до даних.

Для зв'язку з MySQL базою даних використовуються різні мови програмування, такі як Java, Python, PHP тощо. Для Java-додатків зазвичай використовуються різні JDBC-драйвери для зв'язку з базою даних. [18]

MySQL - це одна з найпопулярніших систем управління реляційними базами даних, яка використовується для зберігання та організації даних. Вона дозволяє ефективно керувати великими обсягами інформації та забезпечує широкий спектр інструментів для адміністрування баз даних. MySQL є безкоштовною та відкритою системою, яка доступна для встановлення на більшість операційних систем.

MySQL має багато переваг у порівнянні з іншими системами управління базами даних, особливо для веб-додатків. Основні переваги використання MySQL: [19]

- Відкритий код: MySQL є відкритою системою, що дозволяє розробникам змінювати та адаптувати її під свої потреби.

- Швидкодія: MySQL є дуже швидкою системою управління базами даних, що дозволяє ефективно опрацьовувати великі обсяги даних.
- Надійність: MySQL є дуже надійною системою з високим рівнем стабільності та безпеки.
- Простота використання: MySQL дуже простий та легкий у використанні, що дозволяє швидко навчитися працювати з ним.

На рисунку 2 зображена схема бази даних. Для нормального функціоналу застосунку була створена база даних з 4 таблиць, а саме:

1) User - уособлює в собі звичайного користувача, який має певну роль, яка відносить його або до компаній, або до інвесторів.
2) Investor - роль юзера, яка дозволяє інвестувати в проєкти.
3) Company - роль юзера, яка дозволяє створювати проєкти для подальших інвестицій в них.
4) Project - уособлення самого проєкту, в який можна інвестувати інвесторам.

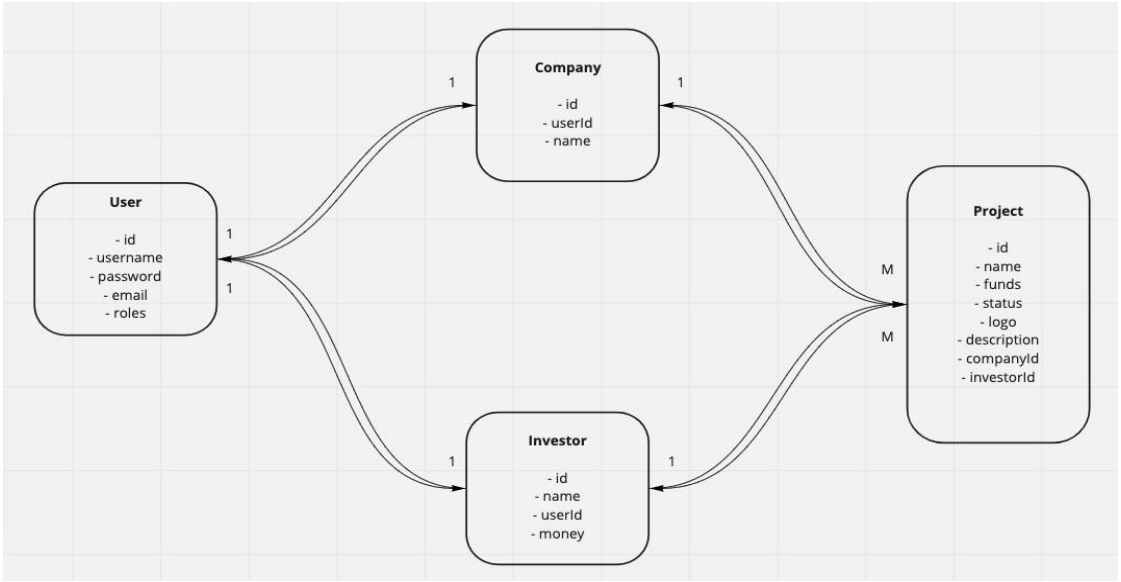

Рисунок 9 - Схема бази даних.

2.7.1 Hibernate як оболочка для даних у коді

Hibernate - це фреймворк для роботи з базами даних, який надає можливість взаємодії з базою даних за допомогою об'єктів Java. Hibernate є популярним вибором для взаємодії з базами даних у проєктах, які використовують Java. Hibernate використовує маппінг об'єктів на таблиці баз даних для забезпечення доступу до даних.

Основною концепцією Hibernate є ORM (Object-Relational Mapping), тобто відображення об'єктів на реляційну модель даних бази даних. ORM дозволяє використовувати об'єктно-орієнтований підхід до роботи з базою даних, що зменшує кількість SQL-запитів та спрощує розробку.

Для роботи з Hibernate використовують анотації, які дозволяють вказати, яким чином мапити об'єкти на таблиці баз даних. Основні анотації Hibernate включають:

- @Entity: вказує, що клас є сутністю, яка пов'язана з таблицею в базі даних.
- @Table: вказує ім'я таблиці бази даних, яка пов'язана з класом сутності.
- @Id: вказує, що поле є унікальним ідентифікатором для сутності.
- @Enumerated: вказує, що поле є перерахуванням і потрібно використовувати відповідний тип даних в базі даних.
- @ManyToOne: вказує, що зв'язок між двома таблицями є багато-до-одного.
- @JoinColumn: вказує ім'я стовпця в базі даних, яке пов'язує дві таблиці з відносинами багато-до-одного.

Hibernate дозволяє використовувати такі типи відносин, як багато-до-одного, один-до-багатьох, один-до-одного, багато-до-багатьох. Для перших двох використовується зовшішній ключ як окреме поле в таблиці, для останнього нам потрібно створювати окрему таблицю зв’язку для підтримки

даних у базі нормалізованими. Він також підтримує кешування для покращення роботи запитів до БД. [20]

```
@Entity
@Table(name = "project")
@Data
public class ProjectModel {


    @Id
    @GeneratedValue(strategy = GenerationType.AUTO)
private Long id;     private String name;
    private int funds;
    @Enumerated(EnumType.STRING)
private StatusModel status;
    private String description;
    @ManyToOne
    @JoinColumn(name = "company_id", nullable = false)
private CompanyModel company;
    @ManyToOne
    @JoinColumn(name = "investor_id")
private InvestorModel investor;
}
```

Рисунок 10 – використання Hibernate для ProjectModel

Як бачимо, ми використовуємо конфігурацію через анотації для пов’язання коду моделі та даних у базі, пояснення до яких наведено вище.

Додатковою перевагою використання анотацій є можливість зменшення кількості коду, що потрібен для взаємодії з базою даних. Hibernate автоматично генерує SQL-запити для створення таблиць, збереження, вибірки та оновлення записів, що зменшує час на розробку та підтримку коду. Однією з основних переваг використання Hibernate є те, що він дозволяє відокремити логіку бізнес-логіки від деталей роботи з базою даних. Hibernate автоматично перетворює об'єкти Java у відповідні записи бази даних та навпаки, що дозволяє зосередитися на розробці функціональності застосунку, а не на деталях взаємодії з базою даних. [21]

2.7.2 Spring Data замість запитів у базу

Spring Data - це підпроєкт Spring Framework, який надає абстракції для роботи з базами даних з використанням JPA або Hibernate. Spring Data

спрощує взаємодію з базою даних і зменшує кількість коду, який потрібно написати для роботи з даними.

Основні переваги Spring Data:

- Автоматичне створення репозиторіїв: Spring Data створює репозиторії автоматично на основі інтерфейсу, який ми створюємо. Наприклад, якщо ми створимо інтерфейс з методом findById(), Spring Data автоматично згенерує код для пошуку запису за ідентифікатором.
- Підтримка різних типів баз даних: Spring Data підтримує багато різних баз даних, таких як MySQL, PostgreSQL, MongoDB та інші.
- Абстракція бази даних: Spring Data надає абстракцію для роботи з базами даних, що дозволяє працювати з даними без прив'язки до конкретної бази даних.
- Підтримка JPA і Hibernate: Spring Data підтримує інтерфейс JPA, що дозволяє використовувати Hibernate як провайдера JPA.
- Підтримка пагінації: Spring Data надає можливість використовувати пагінацію для роботи з великими обсягами даних.
- Підтримка кешування: Spring Data надає можливість кешування даних, що зменшує час запитів до бази даних.
- Підтримка пошуку даних: Spring Data надає можливість використовувати різні методи для пошуку даних, такі як findBy, countBy, deleteBy і т.д.

Використання Spring Data дозволяє значно спростити роботу з базою даних і зменшити кількість коду, який потрібно написати для роботи з даними. Spring Data є однією з популярних бібліотек Spring, яка дозволяє спростити процес доступу до даних в базі даних за допомогою використання різних технологій доступу до даних. Spring Data забезпечує швидкий та простий спосіб взаємодії з базою даних, завдяки чому програмістам не

потрібно багато часу витрачати на налаштування підключення до бази даних та виконання запитів. [22]

```
@Repository
public interface ProjectRepository extends JpaRepository<ProjectModel, Long> {
    List<ProjectModel> getProjectsByCompanyUserUsername(String username);
Optional<ProjectModel> getProjectById(Long id);
}
```

Рисунок 11 – використання Spring Data

2.8 Архітектура через патерн MVC

Архітектурний патерн Model-View-Controller (MVC) є одним з найпопулярніших та найбільш використовуваних патернів у веб-розробці. Він дозволяє розділити компоненти програми на три окремі частини, що забезпечує їхню максимальну незалежність один від одного. Кожна з цих частин виконує своє завдання:

Модель (Model) - це складова, яка відповідає за обробку даних. Вона представляє собою джерело даних, що зберігає, виконує пошук, оновлення та видалення даних, інкапсулюючи всі взаємодії з базою даних. Модель може включати в себе декілька класів, що забезпечують різноманітні операції з даними.

Представлення (View) - це складова, яка відповідає за відображення даних користувачу. Вона може включати в себе HTML-сторінки, які містять вбудований код для відображення даних з Моделі. Представлення також можуть бути відповідальні за збір введення користувача та передачу його Моделі. Представлення може включати HTML, CSS та JavaScript.

Контролер (Controller) - це складова, яка відповідає за обробку запитів користувача та взаємодію з Моделлю та Представленням. Контролер отримує запити від користувачів через Представлення, ініціює взаємодію з Моделлю, отримує необхідні дані та передає їх на Представлення для відображення. [23]

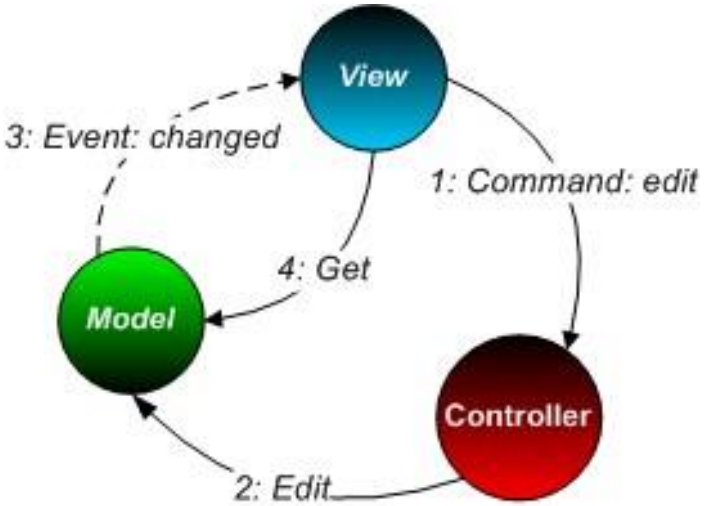

Рисунок 12 - Шаблон проєктування MVC.

Однією з основних переваг використання паттерну MVC є розділення веб-застосунку на окремі компоненти, що дозволяє зробити код більш зрозумілим та легшим для розуміння. Крім того, така архітектура дозволяє ефективніше працювати в команді, коли різні розробники відповідають за різні компоненти.

Ще однією перевагою патерну MVC є його гнучкість та можливість змінювати окремі компоненти без впливу на решту системи. Наприклад, якщо потрібно змінити спосіб відображення даних, можна змінити тільки представлення без впливу на модель або контролер. До переваг патерну MVC можна віднести також зручне тестування коду, оскільки кожна з складових може бути протестована окремо, а також зменшення залежності між компонентами, що робить код більш гнучким і зручним для розширення.

Однак, необхідно враховувати деякі недоліки цього підходу. Наприклад, під час розробки більш складних додатків можуть виникнути проблеми з розподілом завдань між різними компонентами, що може призвести до збільшення складності проєкту в цілому. Крім того, при

використанні патерну MVC виникає необхідність у використанні додаткових інструментів для забезпечення зв'язку між різними складовими програми.

Один із зручних способів організації патерну MVC - використання Spring MVC, який надає готові інструменти для реалізації кожної з його складових. Spring MVC використовує Front Controller pattern, де всі запити до застосунку спочатку обробляються контролером, який визначає, яку модель потрібно відобразити на відповідній сторінці.

Spring MVC розділяє застосунок на 3 складові: контролери, сервіси і представлення. Контролери відповідають за приймання запитів від користувача і виклик сервісів для обробки запиту. Сервіси містять бізнес-логіку застосунку і забезпечують інформацію для представлення. Представлення відображає дані користувачу.

Однією з особливостей Spring MVC є використання анотацій, що робить його дуже гнучким. Наприклад, анотація @Controller дозволяє вказати клас, який буде обробляти запити до застосунку. @RequestMapping вказує шлях до методу, який буде виконуватися при обробці запиту. @RequestParam використовується для отримання параметрів запиту, @PathVariable для отримання значень шляху запиту тощо. [24]

Також перевагою використання патерну MVC є можливість змінювати логіку інтерфейсу користувача, не змінюючи бізнес-логіки застосунку. Це дозволяє розробникам змінювати або оновлювати інтерфейс користувача, не впливаючи на функціональність програми.

У загальному, патерн MVC є досить потужним інструментом, який може забезпечити ефективну та гнучку розробку веб-додатків.

Для спрощеного подання даних у коді використовується популярна бібліотека Lombok, яка допомагає повністю відмовитися від гетерів та сетерів у класах-моделях та різних формах, що передаються з фронт-енду.

Lombok - це бібліотека для Java, яка дозволяє спростити розробку за допомогою автоматичної генерації коду. Вона забезпечує анотації для автоматичного створення геттерів, сеттерів, конструкторів, методів equals, hashCode і toString. Це зменшує кількість написаного коду і дозволяє розробникам швидше створювати програми.

Основні переваги використання Lombok включають:

- Скорочення кількості коду. Завдяки Lombok можна зменшити кількість написаного коду, оскільки більшість стандартних методів можна автоматично згенерувати за допомогою анотацій.
- Забезпечення більш чистого коду. Через використання Lombok розробники можуть уникнути дублювання коду, зберігаючи чистоту і читабельність коду.
- Покращення продуктивності розробки. Завдяки автоматизованому створенню коду розробники можуть швидше і ефективніше створювати програми.

Найбільш популярні анотації, які використовуються в Lombok, включають @Getter, @Setter, @NoArgsConstructor, @AllArgsConstructor, @ToString, @EqualsAndHashCode. Наприклад, за допомогою анотації @Getter можна автоматично згенерувати геттер для поля класу.

Зв'язок між front-end та back-end здійснюється за допомогою архітектури REST API. Це дозволяє використовувати цей API не тільки на певній конкретній платформі (наприклад, iOS або веб-сторінка у браузері), а на будь-якій, оскільки REST API не повертає цілі сторінки, а лише надсилає стан моделі з серверу, використовуючи патерн MVC. Таким чином, логіку застосунку можна розширювати на різні платформи.

2.9 Зберігання файлів у Amazon S3

Оскільки наші користувачі будуть завантажувати фото своєї компанії, то нам потрібно десь зберігати файли. Не дуже зручно зберігати їх прямо на сервері, краще для цього ми будемо використовувати сторонній хмарний сервіс – Amazon S3 bucket. Amazon S3 (Simple Storage Service) є хмарним сервісом зберігання об'єктів, що надає можливість зберігати та отримувати дані з будь-якого місця в Інтернеті. S3 можна використовувати для зберігання та отримання файлів, таких як зображення, відео, документи, а також для зберігання статичного контенту веб-сайту, такого як стилі, скрипти та зображення.

S3 пропонує велику кількість функціональностей, що роблять його корисним сервісом для різних сценаріїв використання. Для зберігання даних використовуються так звані "ведра" (buckets), кожен з яких може містити велику кількість об'єктів. Об'єкти можуть бути захищені за допомогою різних методів шифрування та доступу, що дозволяє зберігати дані в безпечності.

Додатково, S3 має можливість автоматичного масштабування, що дозволяє зберігати великі об'єми даних і автоматично розширюватися при необхідності. Також S3 має різноманітні інструменти для моніторингу та аналізу використання сервісу, що дозволяє ефективно використовувати ресурси та знижувати витрати. До того ж, його можна використовувати безкоштовно у невеликих масштабах.

Одним з найбільших переваг Amazon S3 є його доступність. Це означає, що користувач може звертатися до своїх даних в будь-який час та з будь-якого місця в Інтернеті. Додатково, S3 має велику кількість інтеграцій з іншими сервісами Amazon Web Services (AWS), що дозволяє розширювати функціональність за допомогою інших сервісів. [25]

```
private static final AmazonS3 s3client = AmazonS3ClientBuilder
        .standard()
        .withCredentials(new AWSStaticCredentialsProvider(new
BasicAWSCredentials(KEY, SECRET)))
        .withRegion(Regions.EU_CENTRAL_1)
        .build();

public void saveProjectLogo(Long projectId, MultipartFile logo) {
    try {
        File tempFile = File.createTempFile("project_logo_" + projectId,
".jpg");
        Files.write(tempFile.toPath(), logo.getBytes());
        s3client.putObject(BUCKET_NAME, "logos/" + projectId + ".jpg",
tempFile);
    } catch (IOException e) {
        e.printStackTrace();
    }
}
```

Рисунок 13 – використання Amazon S3

Ми можемо безпечно підключитись до хмарного сервісу S3, використовуючи приватний та секретний ключ, які можна налаштувати в адміністративній панелі Amazon IAM. Для створення підключення до S3 ми можемо скористатись класом AmazonS3ClientBuilder з бібліотеки AWS, а для збереження та отримання файлів - методами putObject() і getObject(). Якщо ми змінимо сервер або хмарну технологію, де запускається наш застосунок, то всі дані будуть збережені в S3, тому після запуску нашого застосунку на іншому комп’ютері ми зможемо без проблем отримати всі файли. Таким чином, використання Amazon S3 забезпечує стійкість та надійність збереження даних.

2.10 Front-end (інтерфейс користувача)

Front-end - це частина веб-розробки, яка займається розробкою і реалізацією користувацького інтерфейсу веб-застосунку або веб-сайту. Front-end складається з трьох основних компонентів: HTML, CSS та JavaScript.

1. HTML (HyperText Markup Language) використовується для створення структури веб-сторінок і описування їх вмісту. HTML використовує теги для відображення тексту, зображень, відео та інших компонентів на веб-сторінках.
2. CSS (Cascading Style Sheets) використовується для візуального оформлення веб-сторінок. CSS описує, як HTML елементи повинні відображатися на екрані, включаючи кольори, розміри, шрифти та інші аспекти дизайну.
3. JavaScript використовується для динамічного взаємодії з користувачем та забезпечення інтерактивності веб-сторінок. JavaScript дозволяє створювати анімацію, обробляти події, валідувати форми та здійснювати комунікацію з сервером за допомогою AJAX-запитів.

Додатково, front-end може включати в себе використання фреймворків та бібліотек, таких як React, Angular, Vue, для полегшення та прискорення процесу розробки веб-додатків. Ці інструменти забезпечують можливість створення більш складних інтерфейсів, використання компонентів та повторного використання коду, що знижує час розробки та сприяє забезпеченню більшої якості коду та зручності його супроводу.

2.11 React.js

React - це відкрита JavaScript-бібліотека для створення інтерфейсів користувача, що розробляється Facebook. React дозволяє створювати багатофункціональні, динамічні інтерфейси, які можна легко підтримувати і розширювати.

Основна перевага React полягає в його можливості забезпечувати більш простий і ефективний спосіб розробки віджетів (components) і складних інтерфейсів. Для цього React використовує концепцію Virtual DOM, яка дозволяє створювати та оновлювати компоненти, не перезавантажуючи сторінку. Це робить React більш ефективним та швидким в порівнянні з традиційними методами розробки.

До інших переваг React можна віднести:

- Модульність і простота: React дозволяє розбити складні інтерфейси на прості компоненти, які можна легко керувати та модифікувати.
- Багатоплатформність: React може працювати на різних платформах, таких як веб, мобільні пристрої та настільні комп’ютери.
- Швидкість: завдяки концепції Virtual DOM, React може відображати зміни на екрані без перезавантаження сторінки, що робить роботу з ним швидшою та більш ефективною.
- Широкі можливості: React є дуже гнучким та може бути використаний в різних сферах, включаючи розробку мобільних додатків, веб-сайтів, ігор, а також забезпечення роботи інших бібліотек та фреймворків.
- Підтримка великої спільноти: React має велику спільноту розробників, яка створює та підтримує різні інструменти та додатки.

Ми створили односторінковий застосунок (single-page-application), головним компонентом якого є App.js. Завдяки використанню компонента Router, наш застосунок може змінювати вміст віртуальних сторінок, тобто

компонентів, не перезавантажуючи застосунок повністю. Router є основним маршрутизатором нашого застосунку, який відповідає за відображення потрібних компонентів в залежності від шляху (URL) у браузері. Це забезпечує більш швидке та зручне взаємодію з користувачем, оскільки він може переходити між компонентами без необхідності повного перезавантаження застосунку.

```
function App() {
  return (
      <div className="App global-page">
        <Router history={browserHistory}>
          <Route path="/" component={Main}/>
          <Route path="/investor/register" component={RegisterInvestor}/>
          <Route path="/company/register" component={RegisterCompany}/>
          <Route path="/login" component={Login}/>
          <Route path="/projects" component={Projects}/>
          <Route path="/all-projects" component={AllProjects}/>
          <Route path="/project" component={AddProject}/>
          <Route path="/projects/:id/logo" component={AddProjectLogo}/>
        </Router>
      </div>
  );
}
```

Рисунок 14 – компонент Router і Route

Це означає, що ми можемо створити зв'язок між адресою сторінки та певним компонентом, який буде відображатися на цій сторінці. Наприклад, якщо ми хочемо відобразити компонент RegisterInvestor на сторінці /register/investor, ми можемо зв'язати цю адресу з відповідним компонентом. Таким чином, коли користувач перейде за адресою /register/investor, на сторінці відобразиться компонент RegisterInvestor.

Можна також створювати відносні посилання, які містять динамічні параметри. Наприклад, посилання /project/:id/logo може відображати компонент AddProjectLogo для певного проєкту з ідентифікатором id.

Параметр id може бути переданий у вигляді змінної всередину компонента AddProjectLogo для подальшої обробки. Завдяки цьому, ми

можемо створювати динамічні сторінки, які можуть відображати різний контент залежно від параметрів в адресному рядку.

Кожен компонент в React має свій життєвий цикл, який складається з декількох етапів.

- Першим етапом є конструктор (constructor), який викликається при створенні нового екземпляру компонента і дозволяє провести початкову ініціалізацію.
- Другий етап - static getDerivedStateFromProps (статичний метод, який викликається безпосередньо перед рендерингом компонента) - дозволяє оновлювати стан компонента на основі нових пропсів, переданих йому з батьківського компонента.
- Третім етапом є рендеринг компонента (render), в результаті якого повертається дерево React елементів, яке буде відображатись на сторінці.
- Після виконання рендерингу компонента викликається метод componentDidMount, в якому можна виконати запити до віддалених ресурсів, налаштувати таймери або підписки на події.
- І останній етап - componentWillUnmount, який викликається перед видаленням компонента з DOM і дає можливість відписатися від підписок, звільнити пам'ять та виконати інші дії, необхідні для коректної роботи застосунка.

Крім цих основних етапів або подій життєвого циклу, також є ще ряд функцій, які викликаються при оновленні стану після початкового рендеринга компонента, якщо в компоненті відбуваються поновлення:

– static getDerivedStateFromProps(props, state) - викликається перед рендерингом компонента при наступних оновленнях стану. Цей метод повинен повернути об'єкт для поновлення об'єкта state або

значення null, якщо нічого оновлювати. Цей метод викликається тільки при оновленні props або state.

- shouldComponentUpdate(nextProps, nextState) - викликається при кожному оновленні props або state. Ця функція повинна повертати true (треба робити оновлення) або false (ігнорувати оновлення). За замовчуванням повертається true. Якщо ця функція поверне false, то наступні функції в життєвому циклі компонента не будуть викликані.
- render() - якщо shouldComponentUpdate поверне true, то викликається ця функція для рендерингу компонента.
- componentDidUpdate(prevProps, prevState, snapshot) - викликається відразу після поновлення компонента (якщо shouldComponentUpdate поверне true). Цей метод може бути використаний для виконання дій після поновлення компонента, наприклад, для виконання запитів до сервера або зміни стану компонента на основі нових props або state.

У React, state - це особливе поле, по суті, об'єкт, який зберігає дані, які можуть змінюватися протягом життєвого циклу компонента. State зазвичай використовується для зберігання даних, які впливають на відображення компонента та які можуть змінюватися в результаті взаємодії користувача з компонентом.

При зміні значень в об'єкті state за допомогою методу setState, React перерендерює компонент, який використовує цей state, з оновленим значенням. При цьому виконується метод componentDidUpdate(), який дозволяє реагувати на зміни стану та виконувати певні дії після оновлення компонента.

Стан компонента є приватним, тобто може бути змінено лише внутрішніми методами компонента, такими як setState. Змінити стан ззовні компонента неможливо. Крім того, для компонентів, які не використовують

state, можна використовувати функціональні компоненти, які зазвичай є більш простими та швидшими виконувати, ніж класові компоненти.

```
export default class Projects extends Component {
    constructor(props) {
        super(props);
        this.state = {
            items: [],
            isLoading: true
        }
    }
    componentDidMount() {
        getProjectsForCompany()
            .then((response) => {
                this.setState({items: response.data, isLoading: false})
                let status = localStorage.getItem("notificationStatus");
                let message = localStorage.getItem("notificationMessage");
                if (status && message) {
                    this.showNotification(status, message);
                }
            })
            .catch(error => {
                console.log("Error *** : " + error);
            });
    }
```

Рисунок 15 – оновлення state компонента

Як бачимо, наш компонент Projects має конструктор, що ставить значення state таким: items: [], isLoading: true. Ці поля будуть використовуватися пізніше у створенні сторінки та відображенні даних.

Функція componentDidMount виконується одразу після першого завантаження сторінки – саме в цьому місці React рекомендує отримувати ресурси, як ми і робимо, викликаючи функцію getProjectsForCompany – оскільки на цій сторінці ми відображаємо саме проєкти компанії.

```
export const getProjectsForCompany = () => {
    return
axios.get(`http://${url}/api/companies/${sessionStorage.getItem("username")}/pr
ojects`, {
        headers: {
            'Content-Type': 'application/json',
            'Authorization': `Bearer_${sessionStorage.getItem("token")}`
        }
    })
};
```

Рисунок 16 – використання axios-запиту

У React для виконання запитів до back-end сервера використовується бібліотека axios. У даному прикладі ми робимо GET запит, в якому формується посилання зі значенням username, що зберігається в sessionStorage. За допомогою цього механізму, дані про користувача зберігаються навіть після закриття сторінки браузера. Також передаємо token, отриманий після операції логіну, для авторизації та аутентифікації користувача на сервері.

Перевага такого запиту полягає у тому, що він є асинхронним, тобто не залежить від інших запитів та частин застосунку. Код виконається тільки після завершення запиту, оскільки використовується метод .then(). Це дозволяє отримати дані з сервера, які повертаються у спеціальному об’єкті Promise.

Після отримання даних з сервера, використовуючи метод setState(), ми змінюємо значення стану компоненту, зробивши поле items рівним відповідним даним з сервера. Також змінюємо значення поля isLoading на false, щоб повідомити сторінці про те, що запит було завершено. Це дозволяє користувачам взаємодіяти з додатком більш плавно та зручно завдяки VDOM технології, впровадженій у React.

```
<Col sm={{span: 10, offset: 1}}>
    {this.state.items.map((project) => {
        return (
            <div className="user-preview" key={project.id}>
```

Рисунок 17 – цикли в React

Завдяки використанню JavaScript та React ми можемо використовувати декларативний підхід та писати цикли прямо всередині HTML коду – таким чином ми відображаємо усі дані зі списку через інший компонент поступово і усі компоненти виходять невеликими за об’ємом коду, оскільки базуються один на одному.

2.12 Взаємодія front-end та back-end

На фронтенді не тільки відображаються дані, що приходять з сервера, але і здійснюються запити до сервера для отримання нової інформації або зберігання нових даних. Для цього використовують технологію AJAX з використанням бібліотеки axios. AJAX - це підхід до розробки веб-додатків, який полягає у взаємодії браузера та сервера в фоновому режимі без перезавантаження сторінки. Завдяки цьому користувач може взаємодіяти з веб-додатком без перерви в роботі. Це робить веб-додатки швидшими та зручнішими для користувача.

AJAX (Asynchronous JavaScript and XML) дозволяє веб-застосункам взаємодіяти з сервером без перезавантаження сторінки. Запити на сервер виконуються асинхронно, що означає, що користувач може продовжувати взаємодіяти з додатком, тоді як запит виконується у фоновому режимі. Запити виконуються за допомогою JavaScript, а дані можуть передаватися у різних форматах, не обов'язково XML.

Технологія AJAX зробила можливим створення веб-додатків з більш взаємодійним та динамічним інтерфейсом, який не потребує повного перезавантаження сторінки для оновлення даних. AJAX використовується в багатьох сучасних веб-застосунках, включаючи соціальні мережі, онлайн-магазини тощо.

Порівняння стандартного підходу і AJAX:

У класичній моделі веб-застосунку:

- Після завантаження на веб-сторінки користувач обирає елемент
- Це змушує браузер сповістити сервер про дію користувача через запит

- У відповідь сервер генерує нову веб-сторінку і відправляє її браузеру, після чого перезавантажена сторінка відображається користувачу.

При використанні AJAX:

- Після завантаження на веб-сторінки користувач обирає елемент
- Скрипт (на мові JavaScript) обирає інформацію, що необхідна для оновлення стану сторінки.
- Браузер відправляє відповідний запит на сервер.
- Сервер повертає лише ті частини документу, які містилися в запиті
- Скрипт оновлює дані на сторінці, лише змінюючи певні її частини, при цьому не перезавантажуючи її повністю

Саме такий підхід робить AJAX сучаною та “розумною” технологією, що дозволяє економити час користувача та ресурси сервера. [26]

Переваги використання AJAX:

- Зменшення трафіку - AJAX дозволяє економити трафік завдяки можливості завантажити лише ту частину сторінки, що змінилася, або передати набір даних в форматі JSON або XML, і після цього змінити вміст сторінки за допомогою JavaScript.
- Зниження навантаження на сервер - правильна реалізація AJAX дозволяє значно знизити навантаження на сервер.
- Підвищення швидкості відгуку інтерфейсу - завантаження лише тієї частини сторінки, що змінилася, дозволяє користувачеві бачити результат своїх дій швидше і без перезавантаження сторінки.

РОЗДІЛ 3.Тестування коду

Для проведення тестування використовуються Unit тести з покриттям JUnit 4 та Mockito, які дозволяють легко тестувати окремі частини коду програми без втручання у інші класи. Це дозволяє проводити тестування кожного класу окремо від інших, навіть якщо він використовує функціонал інших класів. Unit тестування є методом тестування програмного забезпечення, який полягає у тестуванні кожного модуля коду програми окремо. Модулем може бути будь-яка частина програми, яка може бути протестована, наприклад метод в об'єктно-орієнтованому програмуванні. Модульні тести, або unit-тести, розробляються програмістами та тестувальниками білої скриньки в процесі розробки програмного забезпечення. Ці тести зазвичай використовуються для перевірки відповідності коду вимогам архітектури та очікуваної поведінки. JUnit 4 є бібліотекою для проведення модульного тестування програмного забезпечення на мові Java. [27].

Фреймворк Mockito допомагає створювати заглушки класів або інтерфейсів, які можна використовувати замість реальних елементів при юніт-тестуванні коду. Заглушки повинні мати задану поведінку і не обов'язково повинні бути функціонально вірними. Mockito дозволяє створювати моки будь-якого класу одним рядком коду, з зазначеною поведінкою за замовчуванням. Поведінка цих моків може бути перевизначена, щоб вони поводилися відповідно до потреб тестування. В результаті, з допомогою Mockito можна легко створити потрібні заглушки для юніт-тестів з мінімальною кількістю помилок та зусиль.

Крім можливості створення заглушок (mock-об'єктів), фреймворк Mockito має декілька інших корисних функцій для спрощення тестування.

Один з найважливіших функціоналів - це перевірка викликів методів. Mockito дозволяє перевірити, скільки разів і з якими аргументами був викликаний певний метод. Це дозволяє перевірити правильність взаємодії між об'єктами під час тестування.

Mockito також дозволяє створювати Spy-об'єкти - об'єкти, які повністю копіюють функціональність реальних об'єктів, але можуть бути перевизначені деякі методи. Крім того, Mockito може бути інтегрований з іншими фреймворками тестування, такими як JUnit та TestNG. Це дозволяє легко і зручно використовувати фреймворк у вашому тестовому проєкті. Нарешті, варто відзначити, що Mockito - це досить легкий у використанні та зрозуміти фреймворк. Він має простий API та хорошу документацію, що дозволяє легко розпочати роботу з ним.

```
@InjectMocks
private CompanyService companyService;


@Mock
private CompanyRepository companyRepository;
@Mock
private UserService userService;


@Mock
private CompanyModel companyModel;
@Mock
private UserModel userModel;


@Before
public void setUp() {
when(userModel.getUsername()).thenReturn(COMPANY_USERNAME);
}


@Test
public void shouldGetCompanyByUserUsername() {
when(companyRepository.getCompanyByUserUsername(COMPANY_USERNAME))
.thenReturn(Optional.of(companyModel));


    Optional<CompanyModel> actual =
companyService.getCompanyByUserUsername(COMPANY_USERNAME);


    assertThat(actual).isPresent();
    assertThat(actual.get()).isEqualTo(companyModel);
}
```

Рисунок 18 – використання Junit та Mockito для тестування класу CompanyService

РОЗДІЛ 4. Публікація застосунку на віддалений сервер

Ми хочемо зробити наш застосунок доступним для всіх у мережі Інтернет, це можливо зробити через публікацію на віддаленому сервері. Для цього можна обрати хостинг Heroku, який дозволяє розгорнути безкоштовний код на сервері та використовувати його подальше. Хмарні сервери мають п'ять ключових характеристик, які включають самообслуговування на вимогу, широкий доступ до мережі, об'єднання ресурсів, швидку еластичність та вимірювану послугу. Хмарні рішення повинні мати ці характеристики, щоб вважатися справжніми. Є чотири моделі розгортання хмар, які відрізняються в залежності від місця розташування інфраструктури навколишнього середовища. Також є три основні моделі хмарних служб: Програмне забезпечення як послуга, Платформа як послуга та Інфраструктура як послуга. Хоча SaaS була оригінальною моделлю хмарного сервісу, зараз доступний широкий спектр моделей послуг.

Багато компаній використовують хмарні технології для розгортання своїх додатків, але також є багато чинників, що зупиняють їх від цього. Кожна організація повинна оцінити різні варіанти, щоб знайти найкращий для своїх потреб. У цьому випадку, Heroku - безкоштовна платформа для розробки програмного забезпечення, яка дозволяє легко створювати, розробляти та масштабувати веб-додатки. Вона має багато вбудованих додаткових функцій, які допомагають в процесі розробки, таких як надбудови для моніторингу, кешування та розсилки, а також для мережевих підключень. Heroku також надає діагностичні сервіси, щоб допомогти у вирішенні проблем під час виконання програмного забезпечення, і управляє інфраструктурою автоматично. Heroku є посередником між користувачами та AWS Service та належить компанії Salesforce.

РОЗДІЛ 5. Демонстрація роботи застосунку

Тепер розглянемо, як працюють окремо back-end та front-end

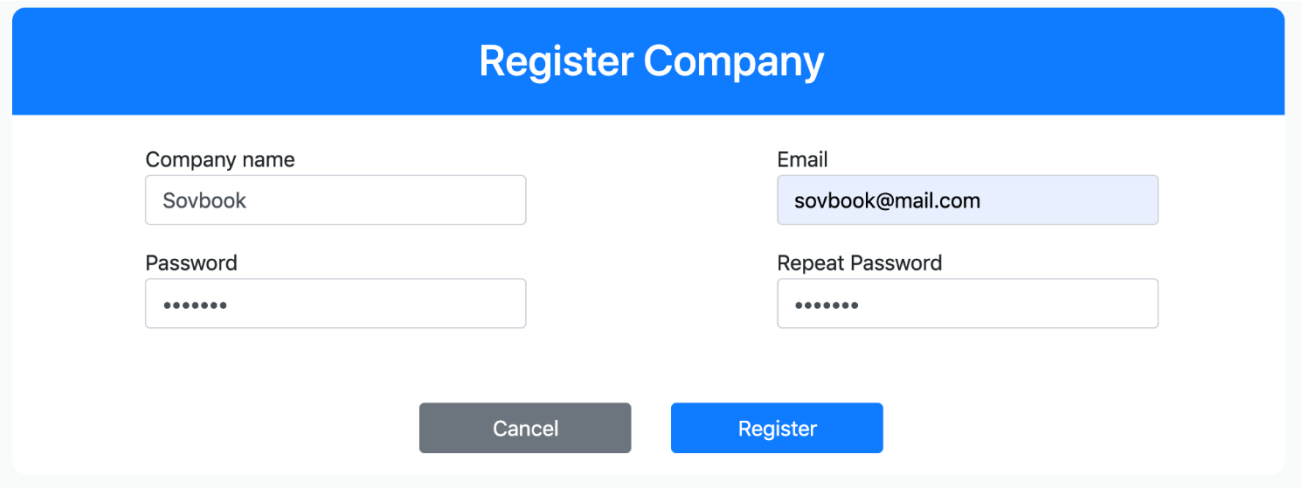

Рисунок 19 – реєстрація компанії.

На даному екрані розміщено 4 поля для заповнення:

логін, пароль, email та роль.

Тепер ми можемо зайти під компанією.

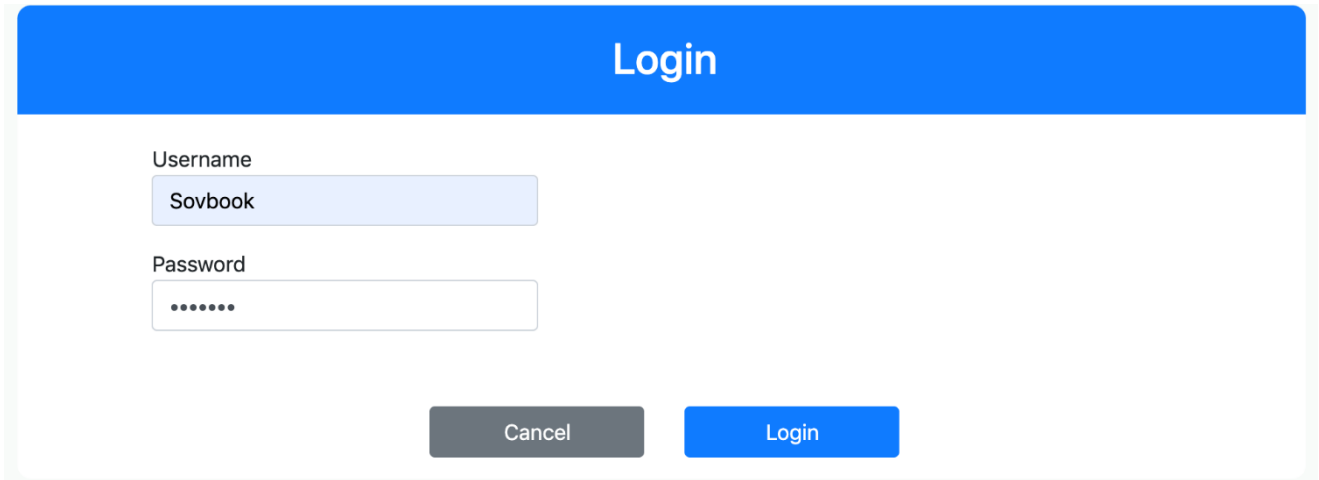

Рисунок 20 - логін

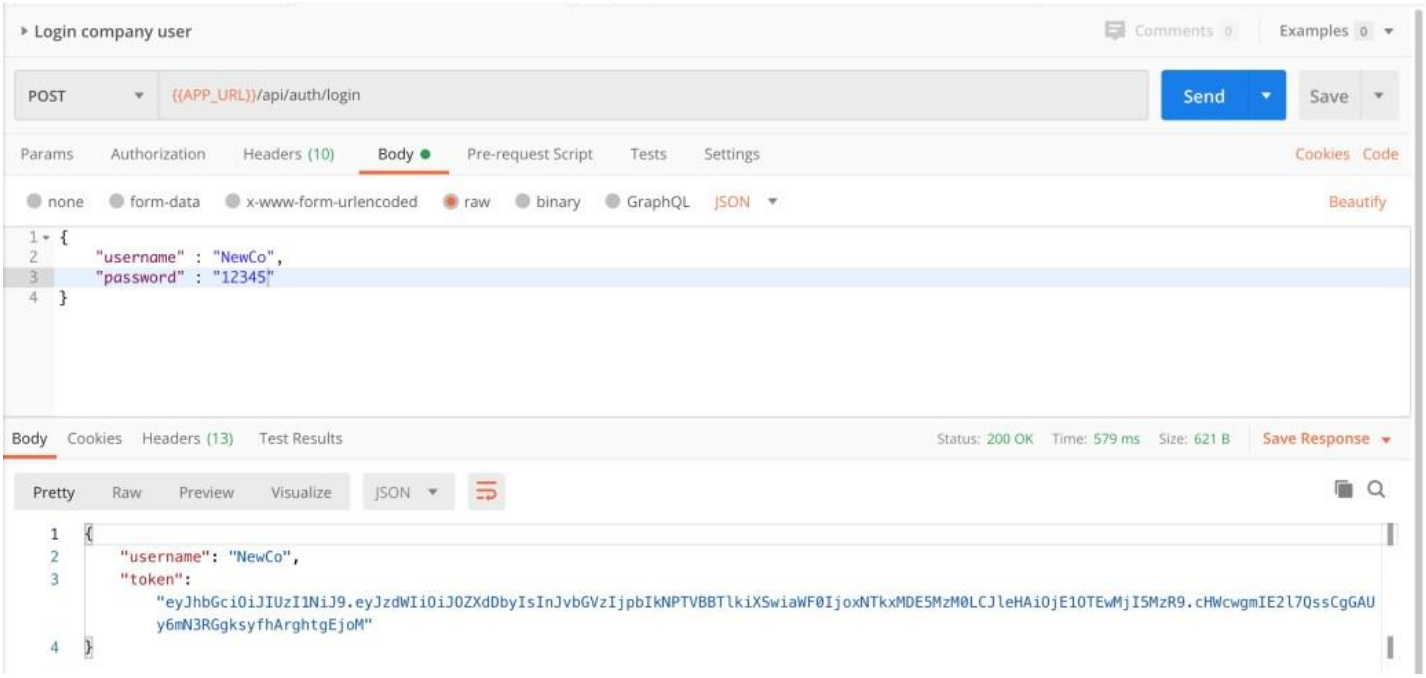

Рисунок 21 – логін компанії.

Для більш детального аналізу покажемо відповідний запит, який відправимо у Postman – утиліти для тестування REST API. Відповіддю на цей запит маємо токен JWT для авторизації у подальших запитах.

Відповідно під даним користувачем створюємо проєкт

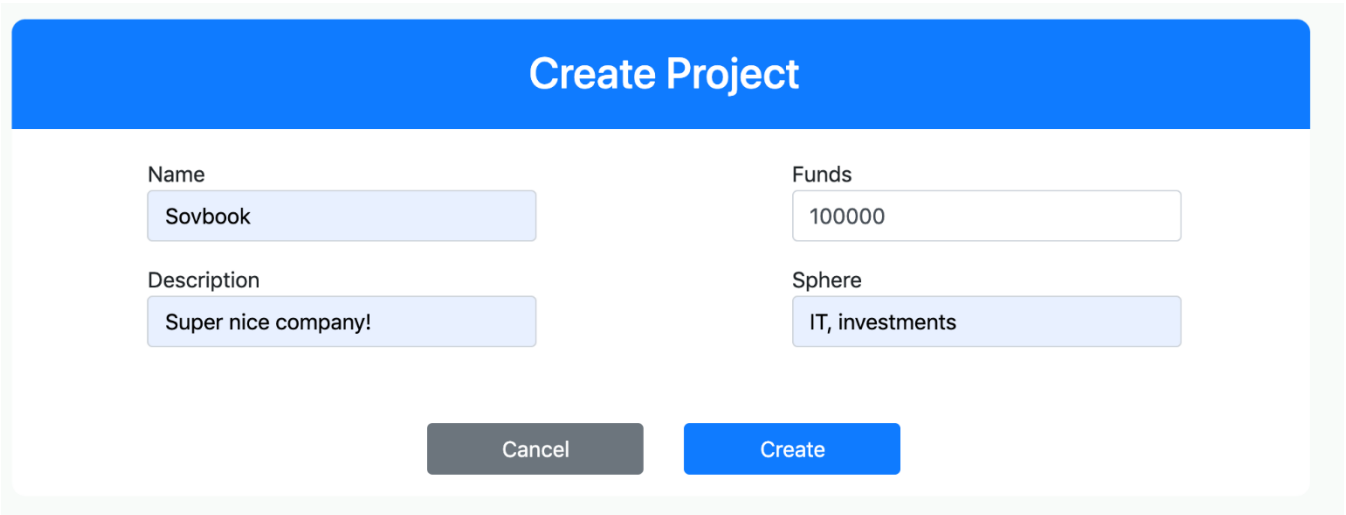

Рисунок 22 – створити компанію

Якщо ж ми посилаємо запит без токену, маємо 403 помилку

Рисунок 23 – спроба створити компанію без токену

Це наглядно демонструє права доступу до застосунку – ми можемо зробити запит лише якщо авторизовані.

Аналогічно створюємо проєкт

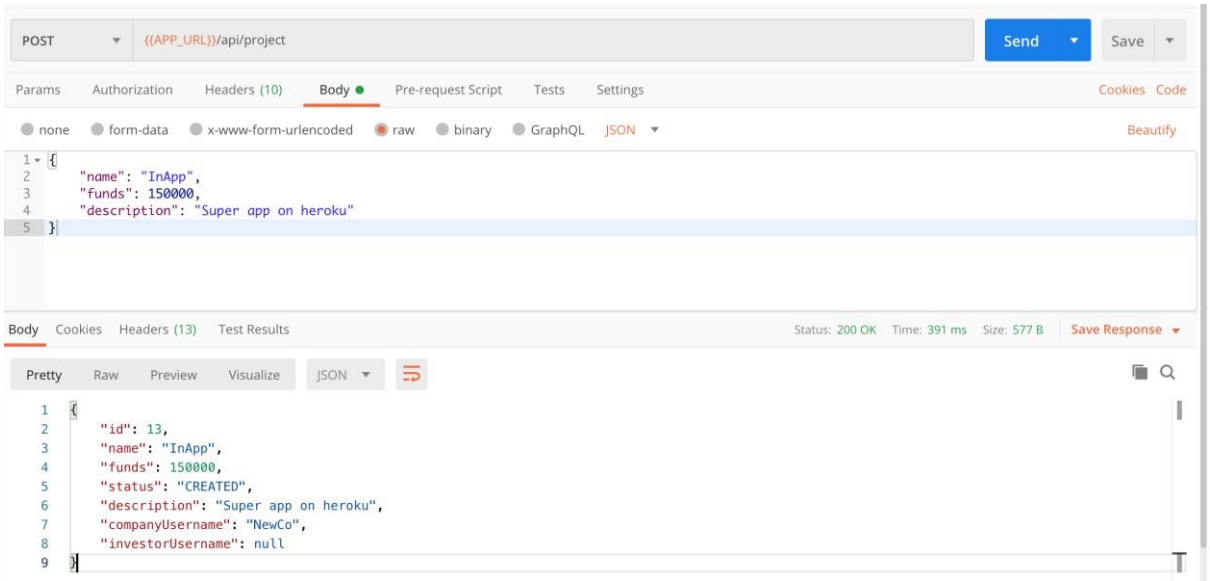

Рисунок 24 – відповідь від серверу після створення проєкту

Наступним кроком нас просять завантажити логотип компанії

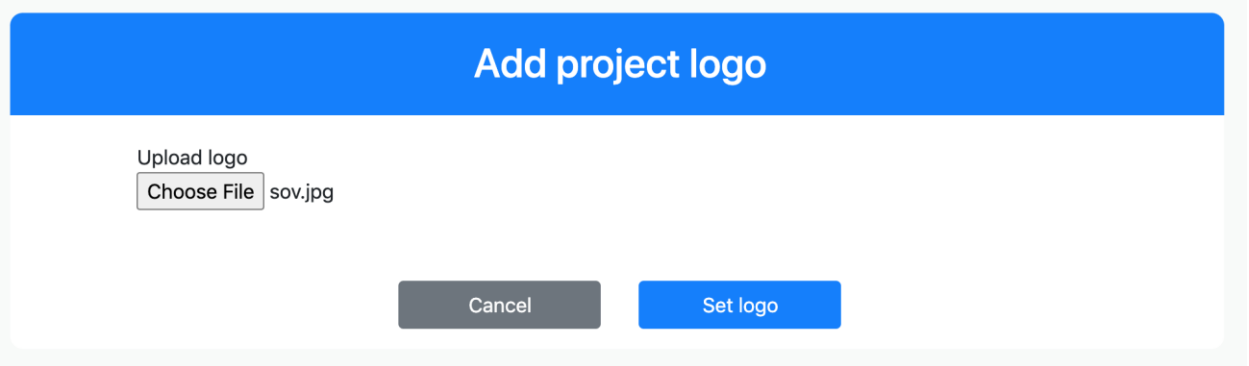

Рисунок 25 – завантажуємо логотип

Після цього наш проєкт буде створено і ми зможемо побачити його у списку наших проєктів

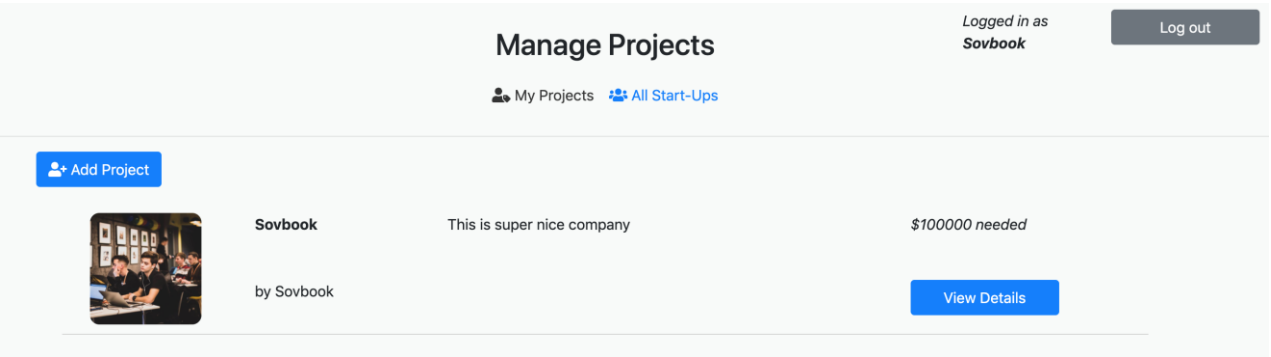

Рисунок 26 – Наші проєкти

Так само створимо інвестора, залогінимо його та проінвестуємо у цей проєкт, покажемо лише останній запит на інвестування.

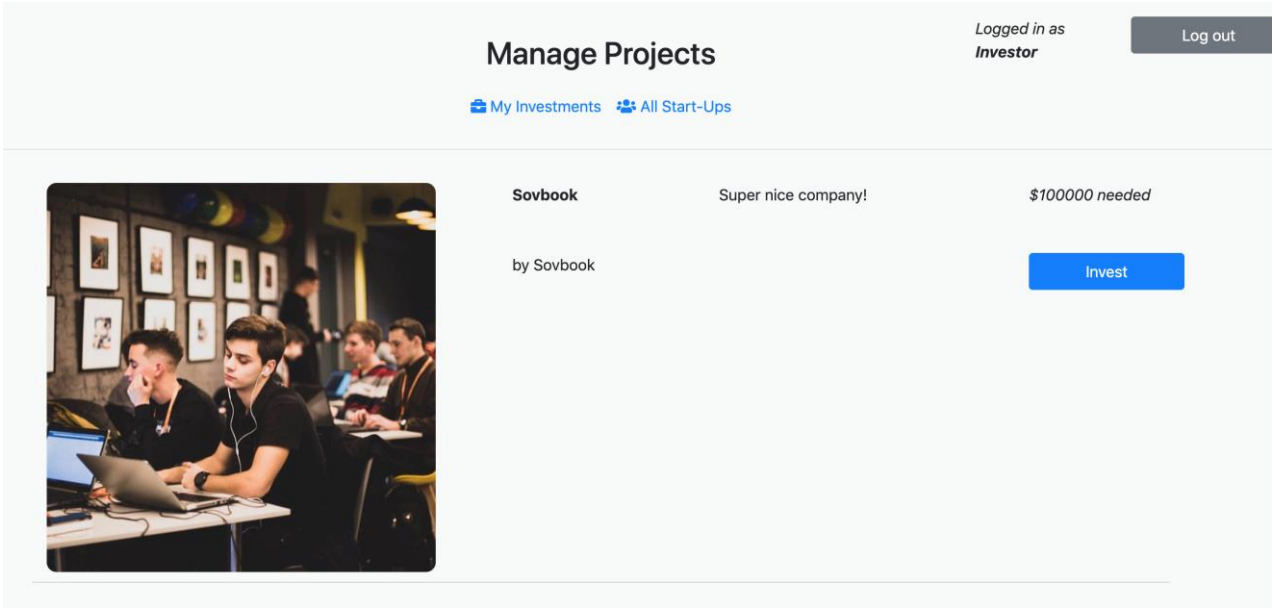

Рисунок 27 – інвестування у конкретний проєкт

Після цього можемо бачити цей проєкт у списку наших проєктів

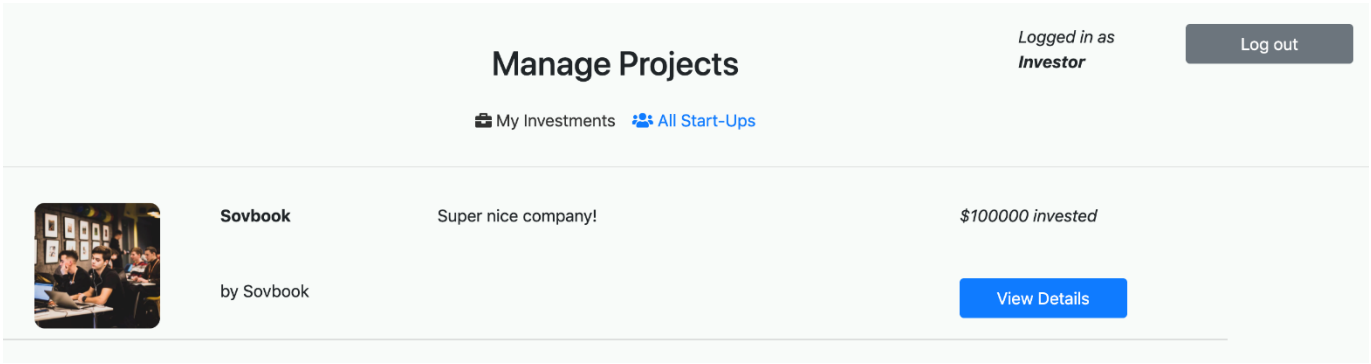

Рисунок 28 – проєкти, в які ми проінвестували

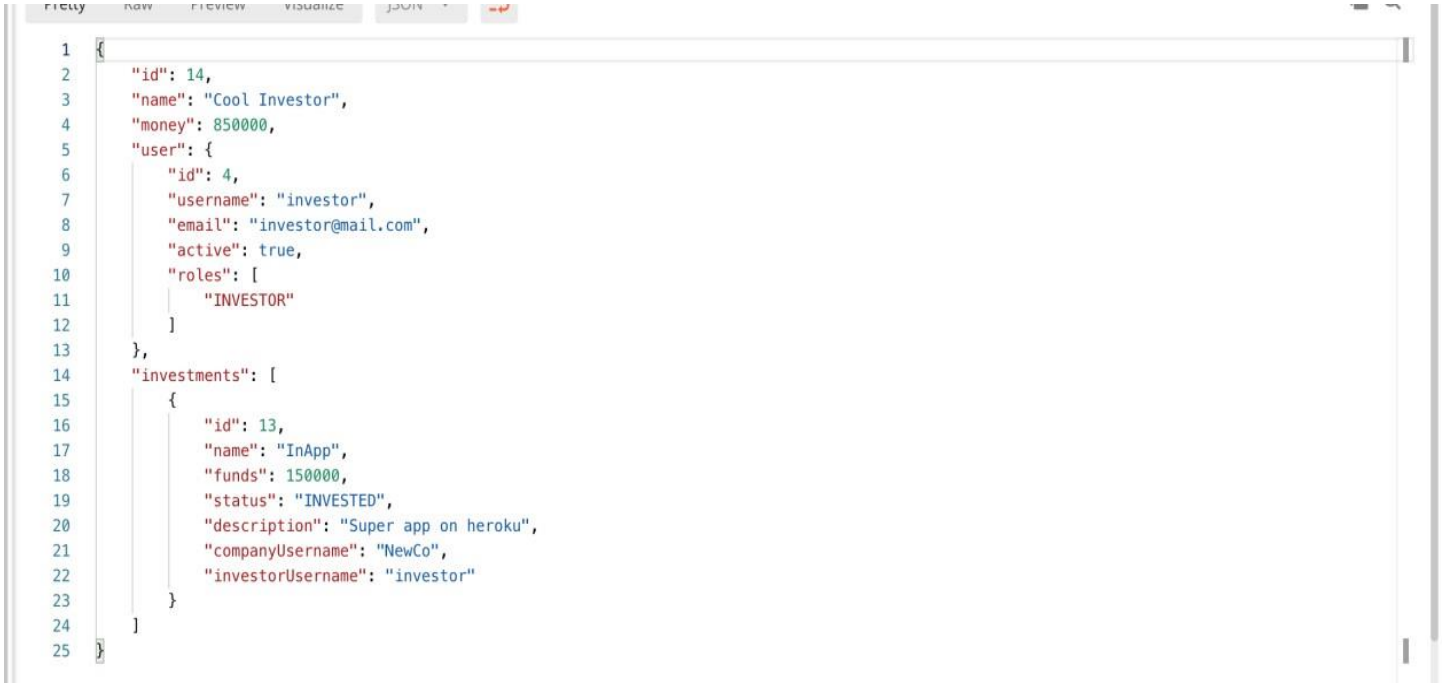

Рисунок 29 – відповідь від серверу після інвестування в проєкт

Таким чином, ми можемо бачити, що застосунок повністю функціонує та виконує свою задачу – користувачі можуть реєструватися як компанії або приватні інвестори та знаходити зв’язок між собою для подальшої співпраці.

Застосунок підтримує можливість авторизації і аутентифікації та не дозволяє анонімним користувачам робити запити на створення проєкту, інвестування в проєкт. Поточна версія не є остаточною та буде доповнюватися.

ВИСНОВОК

Виконання даної роботи показує, що створення інвестиційного програмного забезпечення для розвитку економічних підприємств та інвестицій в перспективні галузі є корисним. Під час розробки дипломного проєкту були використані сучасні технології, такі як мова програмування Java 8, Spring Framework, JWT security, JUnit & Mockito, Postman, React.js, HTML, CSS, які є дуже популярними в програмуванні та мають великий попит. Були використані основні принципи проєктування програмного забезпечення, такі як SOLID, MVC та REST.

Під час розробки застосунку було проаналізовано ринок IT-галузі, порівнюючи новий продукт з конкурентами, знаходячи недоліки інших продуктів та впроваджувано бізнес-логіку у проєкт.

З технічного боку, розроблено web-застосунок з back-end та front-end частинами, використано різні мови програмування, спроєктовано архітектуру серверної частини, розроблено REST API для взаємодії з сервером, написано інтерфейс користувача у стилі single-page-application та поєднано серверну та клієнтську частини. Застосунок може бути загорнуто у хмарний сервіс та опубліковано на віддаленому сервері для подальшого використання застосунку іншими людьми. Також було використано різноманітні бібліотеки для покращення функціональності та простоти коду. Проведено тестування застосунку – спочатку виконано ізольоване тестування серверної частини, а після подальшої інтеграції проведено інтегроване тестування усього застосунку разом. В цілому ціль роботи досягнуто - розроблено web-застосунок, який в подальшому можна розвивати, доповнюючи створений проєкт його новим функціоналом та покращуючи дизайн.

Перелік джерел

[1] - "What Is Backend Development?" [Електронний ресурс] // : Eric An on CareerFoundry – Режим доступу до ресурсу https://careerfoundry.com/en/blog/web-development/what-is-backend-development/

[2] - "Backend Developer Roadmap - 2021" [Електронний ресурс] // : Kamran Ahmed on GitHub – Режим доступу до ресурсу https://github.com/kamranahmedse/developer-roadmap/blob/master/backend.png

[3] - "What is Backend Development and Why it Matters?" [Електронний ресурс] // : Andrei Olaru on Stackify – Режим доступу до ресурсу https://stackify.com/what-is-backend-development/

[4] - Java SE 11 & JDK 11 - Features and Enhancements. [Електронний ресурс] // Oracle – Режим доступу до ресурсу https://www.oracle.com/java/technologies/javase/11-relnote-issues.html

[5] - SOLID Principles of Object Oriented Design Explained with Examples [Електронний ресурс] // JournalDev – Режим доступу до ресурсу https://www.journaldev.com/2389/solid-principles-object-oriented-design

[6] - "Getting started with Spring Framework" - [Електронний ресурс] // – Режим доступу до ресурсу https://spring.io/guides/gs/spring-framework/

[7] - "Spring Framework Tutorial" - [Електронний ресурс] // – Режим доступу до ресурсу https://www.tutorialspoint.com/spring/index.htm

[8] - Spring Boot Documentation - [Електронний ресурс] // – Режим доступу до ресурсу https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/

[9] - "Getting started with Spring Boot" by Baeldung [Електронний ресурс] // – Режим доступу до ресурсу https://www.baeldung.com/spring-boot-start

[10] - "Why Spring Boot is Popular among Java Developers?" - [Електронний ресурс] // Mithun Sridharan – Режим доступу до ресурсу https://dzone.com/articles/why-spring-boot-is-popular-among-java-developers

[11] - "What is REST API?" [Електронний ресурс] // Mulesoft – Режим доступу до ресурсу https://www.mulesoft.com/resources/api/what-is-rest-api-design

[12] – Що таке REST – архітектура - [Електронний ресурс] // Вікіпедія – Режим доступу до ресурсу - https://ru.wikipedia.org/wiki/REST

[13] - Architectural Styles and the Design of Network-based Software Architectures. Doctoral dissertation, University of California, Irvine.;. RESTful web services. O'Reilly Media, Inc. // Fielding, R. T., Richardson, L., (2007)

[14] - "Introduction to Spring Security": [Електронний ресурс] // DZone – Режим доступу до ресурсу https://dzone.com/articles/introduction-to-spring-security

[15] - "Spring Boot Security: Authentication with JWT" [Електронний ресурс] // Okta Developer – Режим доступу до ресурсу https://developer.okta.com/blog/2018/10/31/jwts-with-spring-boot-security-in-5-steps

[16] - Introduction to JSON Web Tokens [Електронний ресурс] – Режим доступу до ресурсу - https://jwt.io/introduction/

[17] - "JSON Web Token (JWT) Authentication with Spring Boot" - [Електронний ресурс] – Режим доступу до ресурсу https://auth0.com/blog/implementing-jwt

[18] - MySQL Official Documentation – [Електронний ресурс] – Режим доступу до ресурсу https://dev.mysql.com/doc/

[19] - JDBC API Tutorial and Reference - [Електронний ресурс] – Режим доступу до ресурсу https://docs.oracle.com/en/java/javase/16/docs/api/java.sql/java/sql/package-summary.html

[20] - Java Persistence with Hibernate. Manning Publications // Bauer, C., & King, G. (2005).

[21] - High-Performance Java Persistence, 2016-2021 [Електронний ресурс] // Vlad Mihalcea – Режим доступу до ресурсу https://vladmihalcea.com/

[22] - "Spring Data: Modern Data Access for Enterprise Java" // Mark Pollack, Oliver Gierke, Thomas Risberg, and Jon Brisbin (2012)

[23] - "Design Patterns: Model View Controller (MVC) Pattern" Tutorials Point: [Електронний ресурс] – Режим доступу до ресурсу https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm

[24] – Spring MVC Tutorial with a Hands-On Step by Step Approach [Електронний ресурс] – Режим доступу до ресурсу https://www.digitalocean.com/community/tutorials/spring-mvc-tutorial

[25] - Офіційна документація Amazon S3 - [Електронний ресурс] – Режим доступу до ресурсу https://docs.aws.amazon.com/s3/index.html

[26] - "JavaScript and JQuery: Interactive Front-End Web Development" // Jon Duckett

[27] – Бібліотека Junit в Java – короткий посібник – [Електронний ресурс] – Режим доступу до ресурсу https://blog.ithillel.ua/articles/unit-testy-v-java.-kratkoe-rukovodstvo

[28] - Методичні вказівки з підготовки та оформлення курсових та дипломних робіт для студентів факультету комп’ютерних наук та кібернетики / Л. Л. Омельчук, А. Б. Ставровський – К.: Київський національний університет імені Тараса Шевченка, 2017 – 47 с.

Додаток – посилання на код на GitHub:

1) https://github.com/Sevatkaaa/InAppMag- back-end
2) https://github.com/Sevatkaaa/in-app-front-mag- front-end