КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Факультет комп'ютерних наук та кібернетики

Кафедра математичної інформатики

Дипломна робота

за спеціальністю 122 Комп'ютерні науки

на тему:

Розробка веб-додатку

для інвестування

з використанням хмарних технологій

Виконав студент 4 курсу

Решетніков Всеволод Сергійович

(підпис)

Науковий керівник:

доцент кафедри математичної інформатики,

кандидат фіз.-мат. наук

Деревянченко Олександр Валерійович

(підпис)

Засвідчую, що в цій дипломній роботі немає запозичень з праць інших авторів без відповідних посилань.

Студент

(підпис)

Київ – 2021

РЕФЕРАТ

Обсяг роботи складає 46 сторінок, в ній містяться 29 ілюстрацій, з яких 14 прикладів коду, використано 15 джерел посилань.

Об’єктом роботи є побудова веб-додатку для інвестування у проекти.

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.

Метою роботи є створення додатку, в якому компанії та інвестори зможуть знаходити одне одного для подальшої співпраці.

Результати роботи: проведений аналіз технологій, розробка додатку, тестування, deploy коду на віддалений сервер та інтеграція з front-end частиною.

У ході роботи були вивчені та використані різні технології, такі як мова програмування Java 8, Spring Framework & Spring Boot, Javascript, React.js, HTML, CSS, MySQL, Hibernate ORM, security за допомогою JWT та Spring Security, deploy коду на сервера AWS через посередника heroku.

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

ЗМІСТ

ВСТУП

Сучасна практика інвестування, як і всі явища соціально-економічного життя суспільства, розвивається під впливом мережі Інтернет. У високорозвинених країнах світу інвестиційні фонди є професійними інвесторами, які мають в своєму арсеналі великі обсяги капіталу, що надає їм можливостей для здійснення різноманітних видів інвестування. Українські інвестиційні фонди поки що не досягли такого рівня розвитку й повноцінно не спроможні виконувати заздалегідь покладені на них інвестиційні функції, насамперед, через нестачу інвестиційних коштів й несприятливого інвестиційного клімату в країні. Аналіз наукової літератури довів, що недостатня увага приділяється розгляданню доходності вітчизняних інвестиційних фондів, необхідності забезпечення прибуткової та стабільної діяльності цих небанківських інститутів у майбутньому. Це пов'язано зі специфікою та тенденціями розвитку українського інвестиційного ринку. На даний момент розроблений додаток “**InApp**” є тільки першою версією та має шляхи розвитку в різних напрямках.

Основною метою дипломної роботи є розробка веб-додатку з використанням інноваційних технологій автоматизації програмних додатків. Також створений додаток допоможе втілювати перспективні ідеї в реальність шляхом збору фінансів на проекти. Серверна частина побудована на мові Java через 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.

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

1) Обґрунтування вибору мов програмування та середовищ розробки back-end та front-end частин;
2) Дослідження існуючих технологій та рішень для створення веб-застосувань через REST API
3) Створення use-case діаграми
4) Створення схеми бази даних для кращого планування back-end
5) Розробка додатку
6) Аналіз розробленого програмного продукту;
7) Unit-тестування серверної частини коду
8) Демонстрація роботи додатку

РОЗДІЛ 1. Обґрунтування теми та опис концепції

1.1 Передумови створення додатку

Для багатьох відкривачів своєї справи основною проблемою є залучення коштів до власного бізнесу. Не так просто зібрати потрібну суму для створення чогось революційного. Часто потрібні кошти займають у друзів, також по кредиторам або ж беруть все що є в гаманці. Такі способи не є дуже практичними в наш час та можуть створити більше проблем, ніж користі. Тому для таких цілей потрібно залучати інвестиції до свого проекту.

Залучення коштів від населення інвестиційними фондами можливе лише за умови активізації і поширення фондового ринку, також з використанням прямих інвестицій. Прямі інвестиції — безпосереднє вкладення коштів інвестором в об'єкти інвестування.

Пряме інвестування здійснюють інвестори, які мають достатньо інформації про об'єкт інвестування і знають механізм інвестування. Прямі інвестиції, як правило, здійснюються у формі кредиту без інвестиційних посередників.

1.2 Порівняння з існуючими проектами

Передусім, важливо розуміти, що на ринку вже існують відомі проекти у цій галузі. Найвідоміший сайт для інвестування - kickstarter.com посідає перше місце у світовому рейтингу.

Використаний на сайті спосіб колективного збору коштів називають «краудфандінг». Kickstarter полегшує збір коштів, створивши модель, яка може бути краще традиційних способів інвестування. Той, хто хоче отримати фінансування, повинен зареєструватися і розмістити опис проекту на Kickstarter. Власник проекту повинен вказати термін і

мінімальну кількість коштів, яке необхідно зібрати. Якщо проект не зібрав потрібну кількість коштів до певного терміну, то гроші повертаються спонсорам. При цьому Kickstarter бере 5% від залучених коштів. На відміну від багатьох форумів по збору коштів або інвестицій, Kickstarter не претендує на право власності на проекти і роботи, які вони приймають до публікації на своєму сайті, але мають право використовувати отриману інтелектуальну власність в будь-яких цілях. Проте проекти, що здійснюються на сайті, зберігаються і доступні для громадськості. Після того, як фінансування проектів завершується, завантажена інформація і матеріали не можуть бути відредаговані або видалені з сайту. Таким чином, головна відмінність нашого додатку від kickstarter – те що наш проект не є краудфандинговою платформою – навпаки, ми даємо можливість стартапу мати одного інвестора, що повністю вкладає кошти у проект і має за це певну частку компанії. У випадку краудфандингу люди не отримують частину компанії – лише невеликий подарок, що по суті нічого не коштує. На ринку України є декілька схожих проектів, проте багато з них мають поганий інтерфейс, що не зрозумілий користувачу, або ж дуже широкий спектр галузей, що робить платформу незручною. У нашому випадку ми більш за все розраховуємо на IT галузь, яка і займає на даний момент лідируючі позиції в світі за кількість старт-апів та зростаючих проектів. Зручний інтерфейс та прозорість робить InApp непоганим проектом для залучення інвестицій у різні стартапи IT галузі. До того ж, ніхто не каже про те, що інвестор має бути з України – це можуть бути і відносини з закордонним замовником. Таким чином, ми не тільки допомагаємо власникам нових проектів та інвесторам знайти спільну мову, а й покращуємо потік інвестицій в Україну.

1.3 Побудова основної частини логіки, яку бачить користувач

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

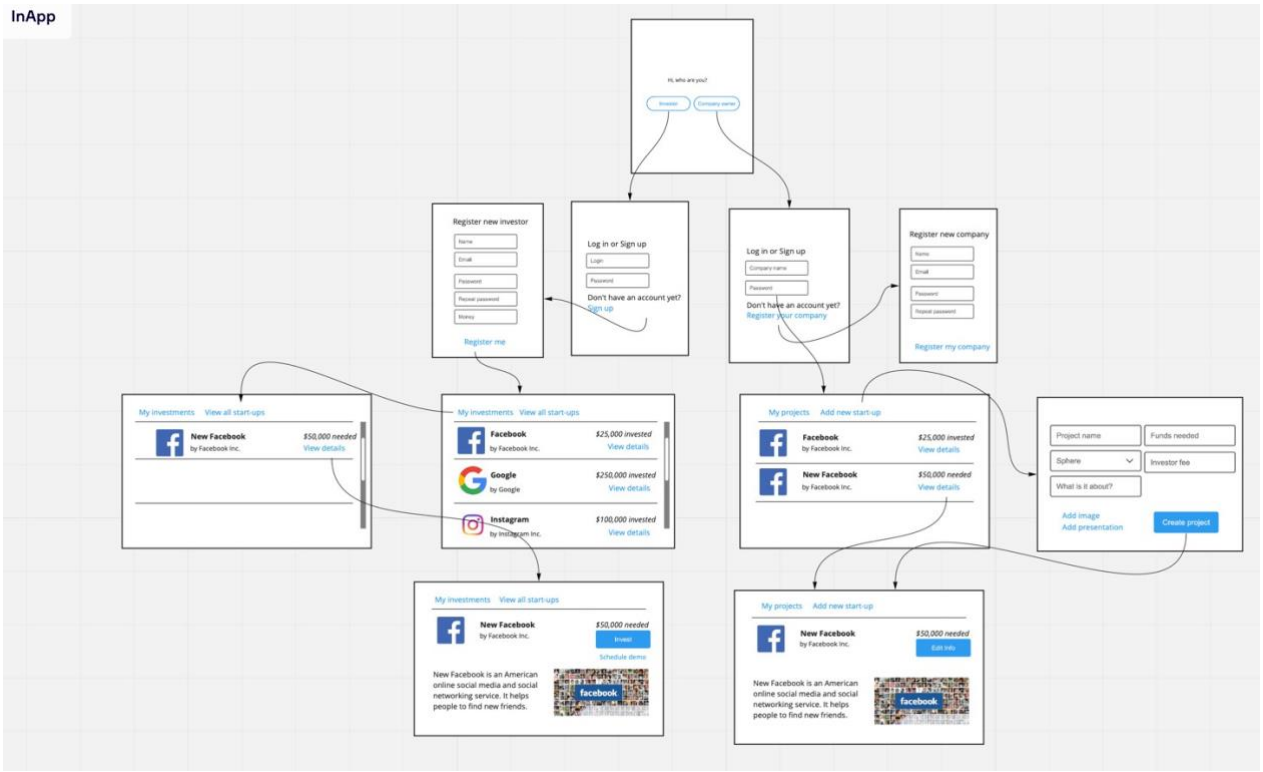

Рисунок 1 – Use-case діаграма

РОЗДІЛ 2. Використання технологій для розробки додатку

2.1 Back-end

Back-end - це серверна частина сервісу: завантаження інформації з сервера сайту, відправлення повідомлень, пошук інформації. Back-end складається з трьох частин:

1) Серверу.
2) Додатку.
3) Бази даних.

Навіть при замовленні авіаквитків або квитків на концерти, при відкритті веб-сайту потрібно взаємодіяти з інтерфейсом. Після введення потрібної інформації додаток зберігає його в базі даних, створеної на сервері. Навіть коли відбувається купівля в інтернет магазині, потрібно заповнювати контактну форму. При введені всіх даних, браузер надсилає запит на серверний бік. Потім сервер повертає інформацію у вигляді зовнішнього коду. Цей код браузер може інтерпретувати і відображати, залежно від того, що знаходиться в базі даних, і зрештою бачимо зміни в контенті на дисплеї. [1]

2.2 Java як основна мова програмування

Основною мовою програмування для створення back-end в проекті є Java. Також з цією мовою програмування використовується фреймворк Spring, який дозволяє використовувати такі шаблони та патерни програмування як MVC. Даний framework легко та зрозуміло реалізує принципи SOLID[2].

Java - це мова програмування зі своєю специфікацією, також це ще і технологія створення додатків. Однією з найбільших переваг Java вважається незалежність від платформи (процесора). Основа технології

Java це байт код. Тобто програма зберігається не в машинних кодах, а в спеціальному коді незалежному від платформи. Для того, щоб цей код міг виконуватися на конкретному процесорі, використовується спеціальна віртуальна Java-машина JVM. Це просто спеціальна програма, яка переводить байт код в машинний код для конкретного процесора [3].

Гарними та сучасними технологіями у Java 8 є Stream API. Завдяки новій версії Java маємо спрощене використання у stream завдяки різним функціям: фільтрам, колекторам та ін.

2.3 Spring Framework та Spring Boot

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

Ключовий елемент Spring - підтримка інфраструктури на рівні програми: основна увага приділяється оформленню додатку, 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.

Особливості Spring Boot:

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

1) Простота управління залежностями

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 додаток. 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 проект 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-додаток, просто додайте залежність spring-boot-starter-web, яка підтягне в проект 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 додаток

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

2)Автоматична конфігурація

Друга серйозна можливість Spring Boot - це автоматична конфігурація додатків.

Після вибору відповідного початкового пакету, Spring Boot намагається автоматично налаштувати Spring-додаток на основі доданих jar-залежностей

Наприклад, якщо ви додаєте Spring-boot-starter-web, Spring Boot автоматично сконфігурує такі зареєстровані біни, як DispatcherServlet, ResourceHandlers, MessageSource

Якщо ви використовуєте 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 додаток, проте Spring також надає можливості створення prototype – (породжуючий патерн, що створює схожі об’єкти, частково копіюючи їх).

```
@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 проекту. 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.

2.4 Spring Security

Також використаний Spring Security[4], який допомагає менеджити JWT (JSON Web Token) - таким чином ми знаємо, коли та хто робить запит і які він має ролі, тобто, чи має права на виконання саме цього запиту до REST API. Це також робить неможливим втараччя ззовні – жоден анонімний користувач не має доступу до виклику API, що стосуються інших проектів чи компаній.

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

Якщо розглядати детальніше, то REST (RESTful) – це архітектурний стиль, що базується на загальниї 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.[5]

Вся взаємодія з сервером зводиться до 4 операцій, тому що 4 операції - це необхідний і достатній мінімум:

1. Отримання даних з серверу - зазвичай у форматі JSON або XML.
2. Додавання нових даних на сервер.
3. Модифікація існуючих даних на сервері.
4. Видалення даних з серверу.

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

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 проекта 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.

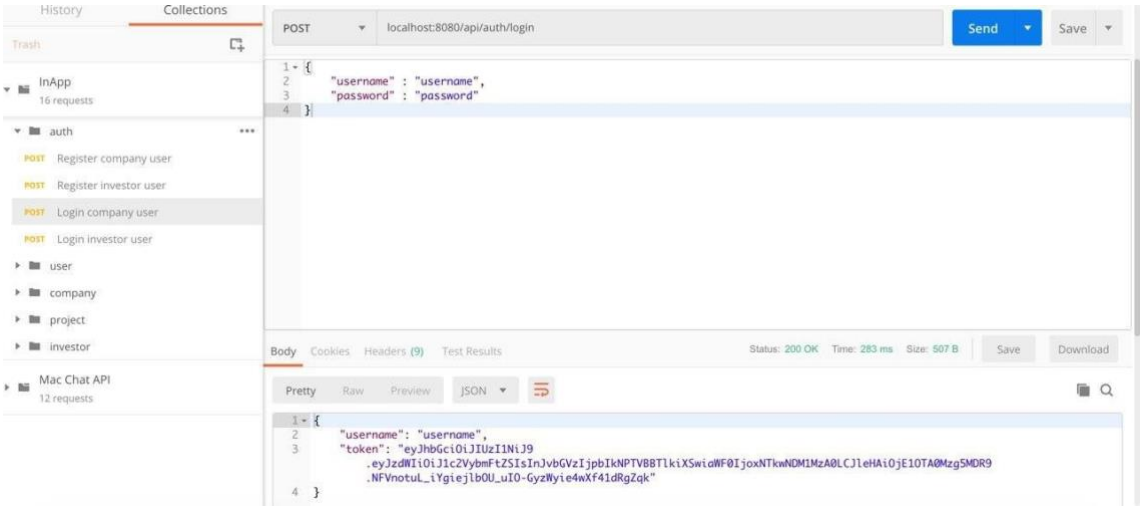

Рисунок 4 - Інтерфейс Postman та приклад взаємодії з даними.

2.5 JSON Web Token Security

Оскільки в додатку 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: [11]

1) Header - системні дані.
2) Payload - основна частина, що є зашифрованою та зберігає у собі такі дані як роль, ім’я користувача, що робить запит
3) Signature - підпис, який неможливо розшифрувати, не знаючи секретного ключа, що схований на сервері.

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.

```
@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 – клас AutenticationController – endpoint для авторизації

```
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 JwtTokenProvider створення токену для авторизації

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 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.
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 – class JwtTokenFilter приклад 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 – клас SecurityConfig – надаємо доступ до певних endpoints

2.6 База даних MySQL

В додатку база даних створена за допомогою сервісу MySQL. Програмне забезпечення MySQL являє собою дуже швидкий багатопотоковий, розрахований на багато користувачів надійний SQLсервер баз даних. Сервер MySQL призначений як для критичних за завданнями виробничих систем з великим навантаженням, так і для вбудовування в програмне забезпечення масового поширення. На рисунку 2 зображена схема бази даних. Для нормального функціоналу додатку 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 проекти.
3) Company - роль юзера, яка дозволяє створювати проекти для подальших інвестицій в них.
4) Project - уособлення самого проекту, в який можна інвестувати інвесторам.

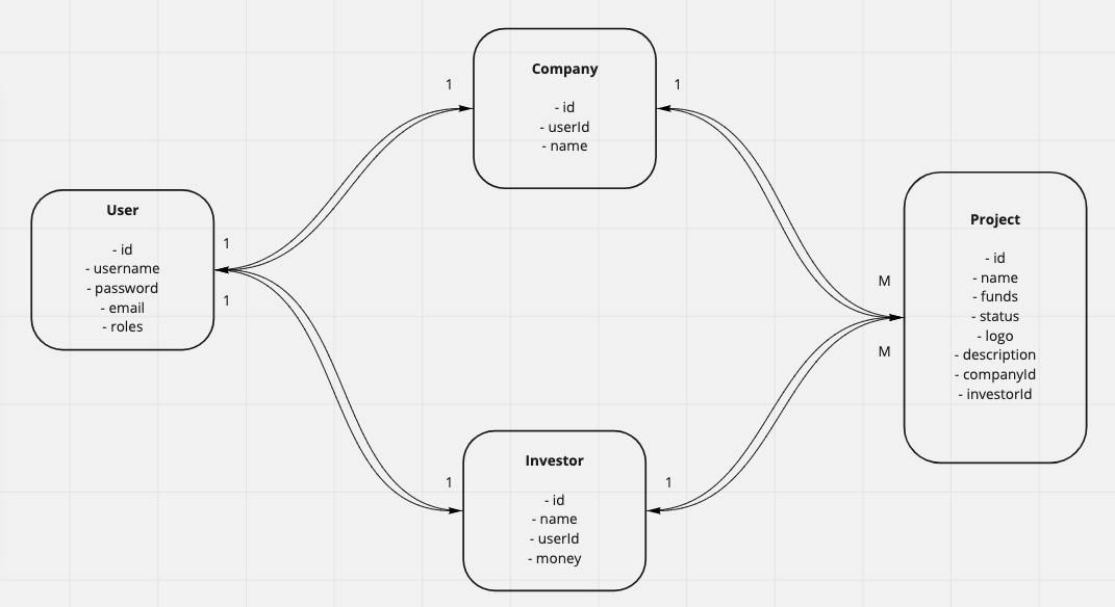

Рисунок 9 - Схема бази даних.

2.6.1 Hibernate як оболочка для даних у коді

Подальша інтеграція Hibernate для використання у Java та Spring - легко та зрозуміло надає таблицю бази даних у серверному коді. [9]

```
@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.

Анотація @Id каже нам про те, що це поле буде унікальним ідентифікатором нашого ресурсу.
@Enumarated вказує на те, що поле має вибиратися з переліку.

@ManyToOne та @JoinColumn відповідають за зв'язок many-to-one (один до багатьох) та за відповідну колонку в базі, яка є зовнішнім ключем для отримання даних. Усього існують три види зв'язку – one-to-one, many-to-one та many-to-many.

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.

@Entity сповішує Hibernate про те, що цей клас є моделлю та має бути пов'язаний з відповідною таблицею у базі, найменування якої міститься у значенні name анотації @Table.

2.6.2 Spring Data замість запитів у базу

Також використовується Spring Data, яка дуже спрощує запити до бази даних, таким чином не потрібно писати direct sql queries, оперування здійснюється лише методами, які вже конвертуються в запити до бази Mysql завдяки Spring Data. [10]

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

Рисунок 11 – використання Spring Data

2.7 Архітектура через патерн MVC

У самій системі застосовано паттерн MVC - він чітко розділяє контролери, представлення об’єктів та їх відображення (response data).

Model-view-controller - архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення [7]. Шаблон проектування MVC передбачає поділ даних програми, призначеного для користувача інтерфейсу і керуючої логіки на три окремих компоненти, таким чином, що модифікація кожного компонента може здійснюватися незалежно. На рисунку 3 зображено шаблон проектування патерну MVC. Компонентами MVC є:

1) Модель - це компонент, який відповідає за дані, а також визначає структуру програми. Наприклад, якщо створюється To-Do додаток, код компонента model визначатиме список завдань і окремі завдання.
2) Уявлення - це компонент, який відповідає за взаємодію з користувачем. Тобто код компонента view визначає зовнішній вигляд програми і способи його використання.
3) Контролер - цей компонент відповідає за зв'язок між model та view. Код компонента controller визначає, як додаток реагує на дії користувача. По суті, це мозок MVC-додатку.

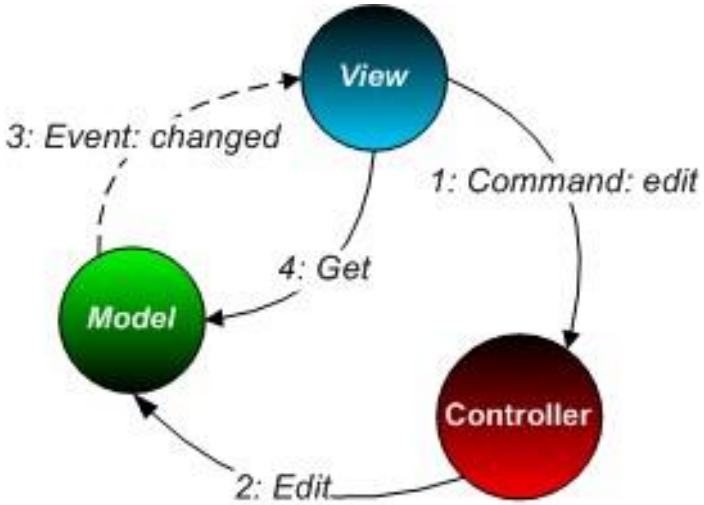

Рисунок 12 - Шаблон проектування MVC.

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 розробка, яка додається до проекту з додатковою функціональністю в Java з підтримкою пошуку вихідного коду перед Javaкомпіляцією. Бібліотека Lombok також дозволяє звільнитись від написання великих об’ємів коду Java в більшій кількості випадків і дозволить писати прості коди з рівних хеш-кодів і toString, в результаті чого Java стає в написанні майже таким як Kotlin, Scala. Також Lombok дуже простий те легкий для додання до проекту.

Зв'язок front-end з back-end відбувається за архітектурою REST API.

Це означає, що ми можемо використовувати даний API не лише на платформі iOS, а й узагалі на будь-якій, оскільки REST API не повертає сторінку, а лише віддає стан моделі з серверу (у патерні MVC). Таким чином ми повертаємо лише ті частини даних, які в нас запитують і це робить додаток легковісним та швидким.

2.8 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 Simple Storage Service (Amazon S3) - це сервіс зберігання об'єктів, що пропонує кращі в галузі показники продуктивності, масштабованості, доступності та безпеки даних. Цей сервіс використовується для зберігання і захисту будь-яких обсягів даних в різних ситуаціях, наприклад для забезпечення роботи сайтів, мобільних додатків, для резервного копіювання та відновлення, архівації, корпоративних додатків, пристроїв IoT і аналізу великих даних. Amazon S3 пропонує прості у використанні інструменти адміністрування, які дозволяють організувати дані і точно налаштувати обмеження доступу відповідно потребами бізнесу або законодавчими вимогами. До того ж, його можна використовувати безкоштовно у невеликих масштабах.

```
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 використовуємо підключення через приватний та секретний ключі, що конфігуруються у адміністративній панелі Amazon IAM.
Клас AmazonS3ClientBuilder з бібліотеки aws допомагає нам легко і зручно створити підключення до хмарного сервісу S3, а методи putObject() і getObject() відповідно зберегти та отримати потрібний файл. Це означає, що навіть якщо ми змінимо сервер або хмарну технологію, де запускається наш додаток, все одно всі дані будуть збережені в S3 і після запуску нашого додатку на іншому комп'ютері всі файли будуть збережені.

2.9 Front-end (інтерфейс користувача)

Frontend - це публічна частина web-додатків (вебсайтів), з якої користувач може взаємодіяти і контактувати напряму. У Frontend входить відображення функціональних завдань, призначеного для користувача інтерфейсу, що виконуються на стороні клієнта, а також обробка запитів користувачів. По суті, фронтенд - це все те, що бачить користувач при відкритті web-сторінки.

Компоненти фронтенд розробки:

- HTML (HyperText Markup Language) кажучи простими словами - це мова розмітки всіх елементів і документів на сторінці, і їх взаємодію в структурі сторінки.
- CSS (Cascading Style Sheets) - це мова характеристики і стилізації зовнішнього вигляду документа. За допомогою CSS-коду браузер

розуміє, як саме необхідно відображати елементи. CSS створює шрифти, кольори, визначає розташування блоків сайту, і інше. Також адаптує один і той же документ в різних стилях, виводить передачу на екран або для читання голосом.

- JavaScript - мова, створена для того, щоб оживити веб-сторінку. Завдання JavaScript - відгукуватися на дії користувача, обробляти натискання клавіш, переміщення курсора, кліки мишкою. JavaScript також дає можливість вводити повідомлення, посилати запити на сервер, а також завантажує дані без перезавантаження сторінки, і так далі. У своєму проекті я також використовую фреймворк React.js, що має декілька корисних бібліотек, зібраних разом.

2.10 React.js

React – шикоро відомий JavaScript-framework для створення користувацьких інтерфейсів.

React може використовуватися для розробки односторінкових і мобільних додатків. Його мета - надати високу швидкість, простоту і масштабованість. Як бібліотеки для розробки призначених для користувача інтерфейсів React часто використовується з іншими бібліотеками, такими як MobX, Redux і GraphQL [12].

Основні переваги React:

- Декларативність - створювати інтерактивні інтерфейси на React приємно і просто. Достатньо описати, як частини інтерфейсу програми відображаються в різних станах. React буде своєчасно їх оновлювати, коли дані змінюються.

  Декларативні подання зроблять код більш передбачуваним і спростять відловлювання помилок (debugging).
- Заснований на компонентах – дозводяє створювати інкапсульовані компоненти з власним станом, а потім об'єднувати їх в складні інтерфейси. Оскільки логіка компонента написана на JavaScript, а не міститься в шаблонах, можна з легкістю передавати найрізноманітніші дані по всьому додатку.
- Легкість – React бібліотеки займають дуже мало місця на диску та легко конфігуруються завдяки npm (node package manager).

У нашому випадку ми створили single-page-application, головним класом якого буде App.js – завдяки компоненту Router він розуміє, як саме має міняти умовні сторінки (насправді компоненти), не перезавантажуючи при цьому повністю весь додаток.

```
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

Таким чином, ми робимо певний зв’язок між адресою сторінки і відповідним їй компонентом – при реєстрації інвестора ми будемо направляти користувача за посиланням /register/investor, якому відповідає компонент RegisterInvestor.

Так само, наприклад, за посиланням /projects маємо компонент Projects.

Також інсує можливіть робити відносні посилання, наприклад /project/:id/logo, де id буде передано всередину компонента AddProjectLogo для подальшого використання.

Тепер розглянемо сам компонент зсередини.

Для початку визначимо, як саме проходить життєвий цикл кожного компонента - ряд етапів існування. На кожному з етапів викликається певна функція, в якій ми можемо визначити будь-які дії:

1) constructor (props): конструктор, в якому відбувається початкова ініціалізація компонента
2) static getDerivedStateFromProps (props, state): викликається безпосередньо перед рендерингом компонента. Цей метод не має доступу до поточного об'єкту компонента і повинен повертати об'єкт для поновлення об'єкта state або значення null, якщо нічого оновлювати.
3) render (): рендеринг компонента
4) componentDidMount (): викликається після рендеринга компонента. Тут можна виконувати запити до віддалених ресурсів
5) componentWillUnmount (): викликається перед видаленням компонента з DOM

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:

1) static getDerivedStateFromProps (props, state)
2) shouldComponentUpdate (nextProps, nextState): викликається кожен раз при оновленні об'єкта props або state. Як параметр передаються новий об'єкт props і state. Ця функція повинна повертати true (треба робити оновлення) або false (ігнорувати оновлення). За замовчуванням повертається true. Але якщо функція буде повертати false, то тим самим ми відключимо оновлення компонента, а наступні функції не будуть спрацьовувати.
3) render (): рендеринг компонента (якщо shouldComponentUpdate повертає true)
4) getSnapshotBeforeUpdate (prevProps, prevState): викликається безпосередньо перед компонента. Він дозволяє компоненту отримати інформацію з DOM перед можливим оновленням. Повертає як значення якийсь окремий аспект, який передається в якості третьої параметра в метод componentDidUpdate () і може враховуватися в componentDidUpdate при оновленні. Якщо нічого повертати, то повертається значення null
5) componentDidUpdate (prevProps, prevState, snapshot): викликається відразу після поновлення компонента (якщо shouldComponentUpdate повертає true). Як параметри передаються старі значення об'єктів props і state. Третій параметр - значення, яке повертає метод getSnapshotBeforeUpdate. [14]

Також є особливе поле класу, а саме state – воно зберігає поточний стан компонента та при оновленні state буде викликано функцію componentDidUpdate(), у якій ми можемо робити певні запити для поновлення ресурсів, якщо це потрібно.

Для зміни стану в React передбачено функцію setState.

```
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 проекти 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

Для виконання запитів на back-end ми використовуємо бібліотеку axios, у нашому прикладі ми робимо GET запит за посиланням, що формується зі значення username, яке зберігається у sessionStorage – спеціальному глобальному об’єкті React, в якому ми також зберігаємо token користувача – таким чином навіть якщо користувач закриє сторінку браузера, ці дані не буде видалено і після наступного відкривання сайту від буде залогіненим. Також ми обов’язково передаємо token, який отримали після операції логіну для авторизації та аутентифікації користування на сервері. Основний плюс такого запит – асинхронність, тобто незалажність від інших запитів та частин додатку. Більше переваг буде розказано у наступному розділі. Сам запит повертає спеціальний об’єкт Promise, саме тому код виконається не одразу, а лише по завершенні запиту, оскільки ми викликаємо функцію .then(), інакше б ми працювали з порожнім значенням, оскільки отримання response з серверу вимагає певного часу.
Далі завдяки методу setState() ми змінюємо значення стану компоненту і робимо його поле items рівним відповідним даним з сервера, а isLoading ставимо false, щоб повідомити сторінці про те, що запит завершено.

```
<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 проекти через інший компонент поступово і усі компоненти виходять невеликими, оскільки базуються один на одному.

2.11 Взаємодія front-end та back-end

Front-end не тільки відображає дані з серверу, але і робить певні запити для отримання цих даних та зберігання нових. Усі запити робляться за технологією AJAX з використанням бібліотеки axios. Ajax (Asynchronous Javascript and XML) - підхід до побудови призначених для користувача інтерфейсів веб-додатків, що полягає в «фоновому» обміні даними браузера з веб-сервером. В результаті при оновленні даних веб-сторінка не перезавантажується повністю, і веб-додатки стають швидше і зручніше.

Порівняння стандартного підходу і AJAX:

У класичній моделі веб-додатку:

- Користувач заходить на веб-сторінку і натискає на який-небудь її елемент.
- Браузер формує і відправляє запит серверу.
- У відповідь сервер генерує абсолютно нову веб-сторінку і відправляє її браузеру, після чого браузер повністю перезавантажує всю сторінку.

При використанні AJAX:

- Користувач заходить на веб-сторінку і натискає на який-небудь її елемент.
- Скрипт (на мові JavaScript) визначає, яка інформація необхідна для оновлення сторінки.
- Браузер відправляє відповідний запит на сервер.
- Сервер повертає лише ту частину документа, на яку прийшов запит.
- Скрипт вносить зміни з урахуванням отриманої інформації (без повного перезавантаження сторінки).

Саме такий підхід робить AJAX сучаною та “розумною” технологією, що дозволяє економити час та ресурси.

Переваги:

- економія трафіку - використання AJAX дозволяє скоротити трафік при роботі з веб-додатком завдяки тому, що замість завантаження всієї сторінки достатньо завантажити тільки ту частину, що змінилася або взагалі тільки отримати / передати набір даних в форматі JSON або XML, а потім змінити вміст сторінки за допомогою JavaScript.
- Зменшення навантаження на сервер - при правильній реалізації AJAX дозволяє знизити навантаження на сервер в кілька разів.

- Прискорення реакції інтерфейсу - оскільки завантаження частини, що змінилася, відбувається значно швидше, то користувач бачить результат своїх дій швидше і без перезавантаження сторінки.

РОЗДІЛ 3.Тестування коду

Для тестування використовуються Unit tests з покриттям - Junit 4 та Mockito забезпечують легке тестування без втручання у інші класи, таким чином кожен клас тестується окремо від інших, навіть якщо використовує його функціонал.

Unit тестування - це метод тестування програмного забезпечення, який полягає в окремому тестуванні кожного модуля коду програми. Модулем називають найменшу частину програми, яка може бути протестованою. В об'єктно-орієнтованому програмуванні — метод.

Модульні тести, або unit-тести, розробляють в процесі розробки програмісти та, іноді, тестувальники білої скриньки (white-box testers).[6]

Зазвичай unit-тести застосовують для того, щоб упевнитися, що код відповідає вимогам архітектури та має очікувану поведінку.

JUnit 4 - бібліотека для модульного тестування програмного забезпечення на мові Java [7].

Фреймворк Mockito надає ряд можливостей для створення заглушок замість реальних класів або інтерфейсів під час написання JUnit тестів. При тестуванні коду, перш за все юніт-тестуванням, тестованому елементу часто потрібно надати екземпляри класів, якими він повинен користуватися при роботі. При цьому досить часто вони не повинні бути повно функціональними - навпаки, від них вимагається вести себе тільки заданим чином, так, щоб їх поведінка була простою та цілком передбачуваною. Такі класи і називаються заглушками (stub). Щоб їх отримати, можна створювати альтернативні тестові реалізації інтерфейсів, успадковувати потрібні класи з перевизначенням функціоналу і так далі, але все це досить незручно, досить надлишково та

з значною кількістю помилок. Більш зручне у всіх сенсах рішення - спеціалізовані фреймворки для створення заглушок, саме таким і є Mockito.

Mockito дозволяє створити одним рядком коду так званий mock будьякого класу, що в підсумковому варіанті схоже на зразок основи для потрібної заглушки. Для такого mock відразу після створення характерна деяка поведінка за замовчуванням, всі методи повертають заздалегідь відомі значення - зазвичай це null або 0. Можна перевизначити цю поведінку бажаним чином, проконтролювати її з потрібним ступенем детальності звернення до неї і так далі. В результаті mock і стає заглушкою з необхідними властивостями.

```
@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 додатку на віддалений сервер

Оскільки ми хочемо зробити додаток доступним для всіх у мережі Інтернет, ми можемо встановити наш код при виконанні на віддаленому сервері. Для цього було обрано хостинг Heroku, який може зробити безкоштовний код розгортання на сервері та подальше його використання. Сам код функціонує в мережі за адресою https://in-app-front.herokuapp.com/

Існує п’ять основних ключових характеристик хмарних серверів: самообслуговування на вимогу (on-demand self-service), широкий доступ до мережі (broad network access), об’єднання ресурсів (resource pooling), швидка еластичність (rapid elasticity) та вимірювана послуга (measured service). Рішення повинно мати ці п’ять характеристик, щоб вважатися справжнім хмарним рішенням. Існує чотири моделі розгортання хмар: загальнодоступна (public), приватна (private), громадська (community) та гібридна (hybrid). Кожна модель розгортання визначається відповідно до того, де знаходиться інфраструктура навколишнього середовища. Існує три основні моделі хмарних служб: Програмне забезпечення як послуга (Software as a Service), Платформа як послуга (Platform as a Service) та Інфраструктура як послуга (Infrastructure as a Service). SaaS була оригінальною моделлю хмарного сервісу, але хмара продовжувала рости та розширюватися. Зараз доступний широкий спектр моделей послуг.

Є багато факторів, що підштовхують організації до використання хмарниї технологій для розгортання додатків, а також багато факторів, що утримують організації від них. Кожна організація повинна оцінити

хмарні пропозиції для себе, щоб побачити, що найкраще відповідає її потребам. У нашому випадку ми не хочемо турбуватися про інфраструктуру серверів, тому обираємо легкий у розумінні та підтримці безкоштовний сервіс Heroku, що є посередником між нами та AWS Service.

Heroku - постачальник хмарних послуг та платформа для розробки програмного забезпечення, що забезпечує швидке та ефективне створення, розроблення та масштабування веб-додатків. Він має 140 вбудованих додаткових функції, від опосередкування до служб безпеки аналітичних інструментів, які використовуються для таких елементів, як надбудови для моніторингу, кешування та розсилки, а також для мережевих підключень.

Інструмент може запропонувати вам вбудовані діагностичні сервіси, додані під час виконання. Більше того, вам не потрібно думати про інфраструктуру, оскільки вона управляється автоматично самим програмним забезпеченням. Heroku належить компанії Salesforce.

РОЗДІЛ 5. Демонстрація роботи додатку

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

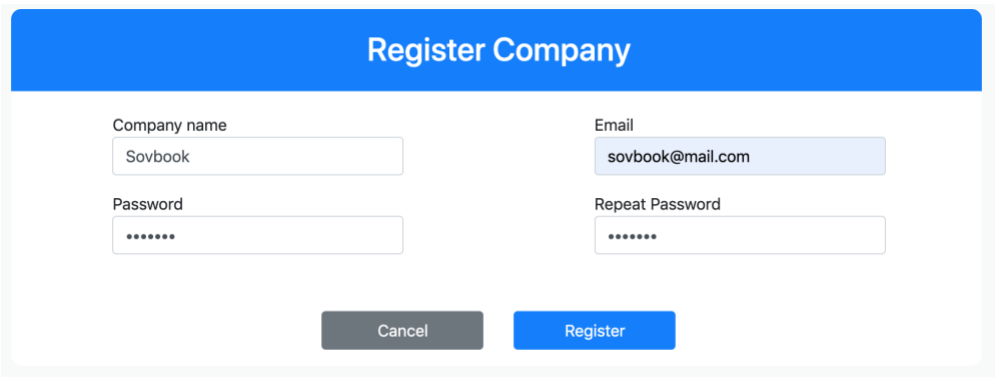

Рисунок 19 – реєстрація компанії.

На даному екрані розміщено 4 поля для заповнення:

логін, пароль, email та роль.

Тепер ми можемо зайти під компанією.

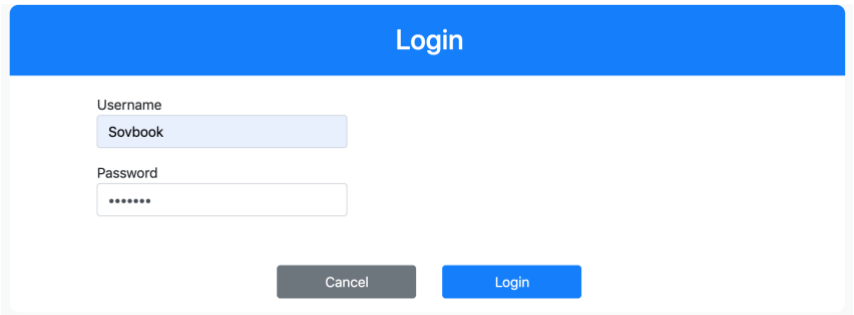

Рисунок 20 - логін

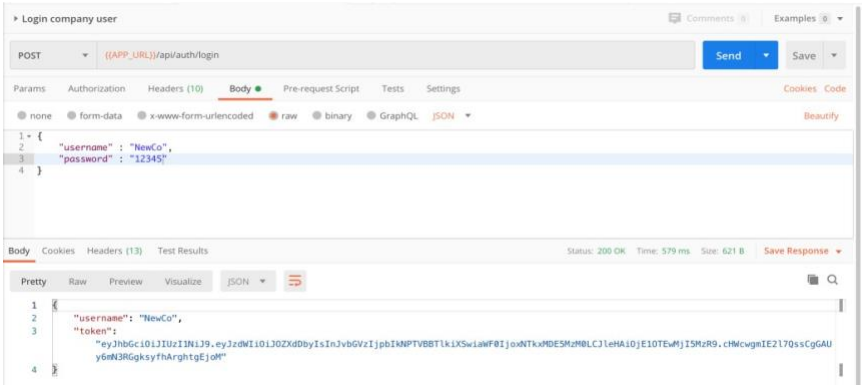

Рисунок 21 – логін компанії.

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 проект

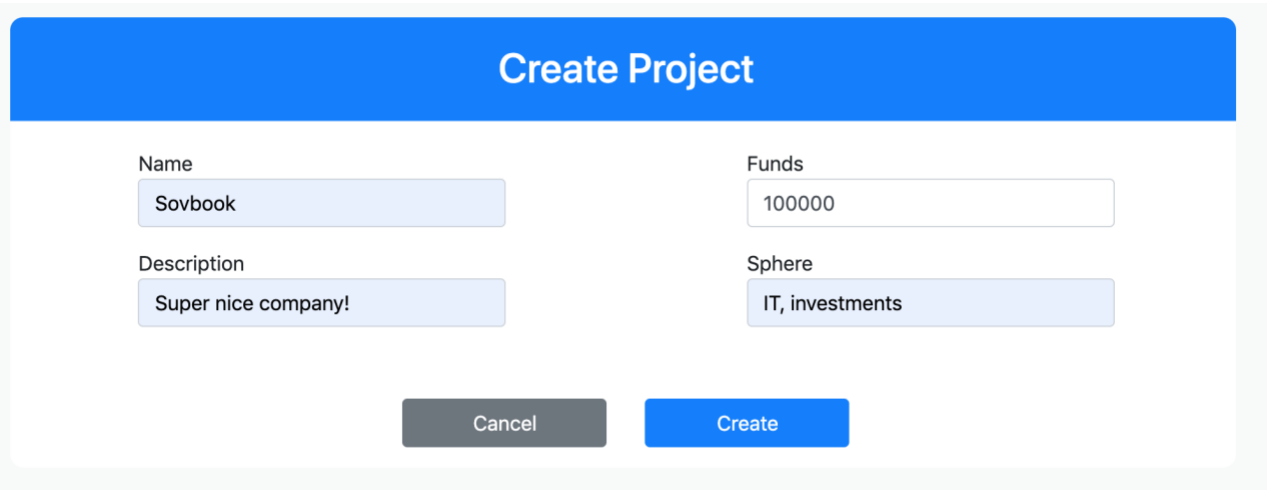

Рисунок 22 – створити компанію, використовуючи JWT токен

Якщо ж ми посилаємо запит без токену, маємо 403 помилку

Рисунок 23 – спроба створити компанію без токену

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

Аналогічно створюємо проект

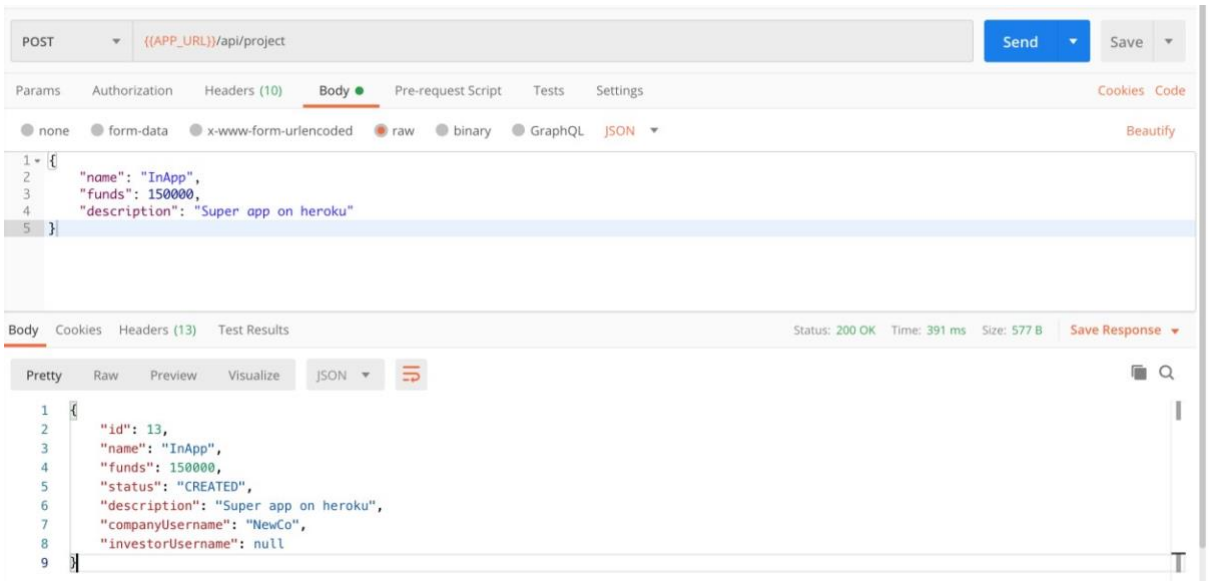

Рисунок 24 – відповідь від серверу після створення проекту

Наступним кроком нас просять завантажити логотип компанії

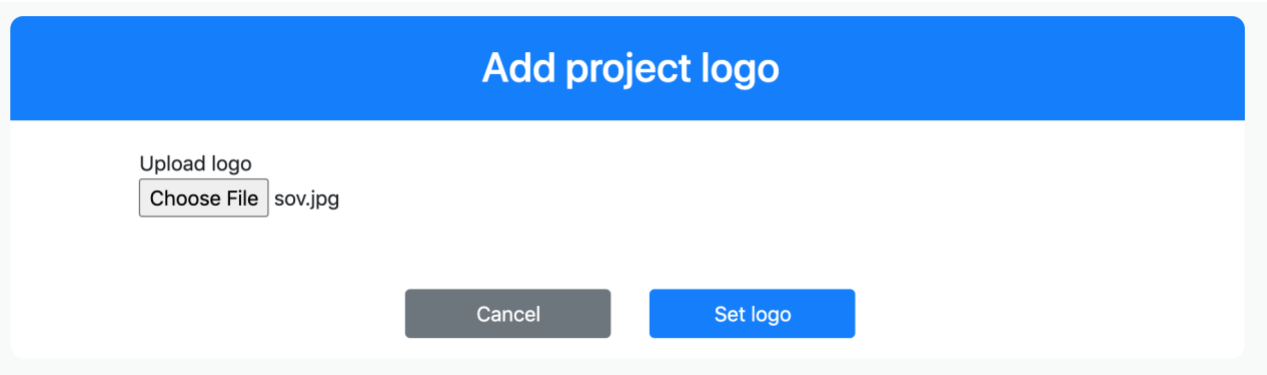

Рисунок 25 – завантажуємо логотип

Після цього наш проект буде створено і ми зможемо побачити його у списку наших проектів

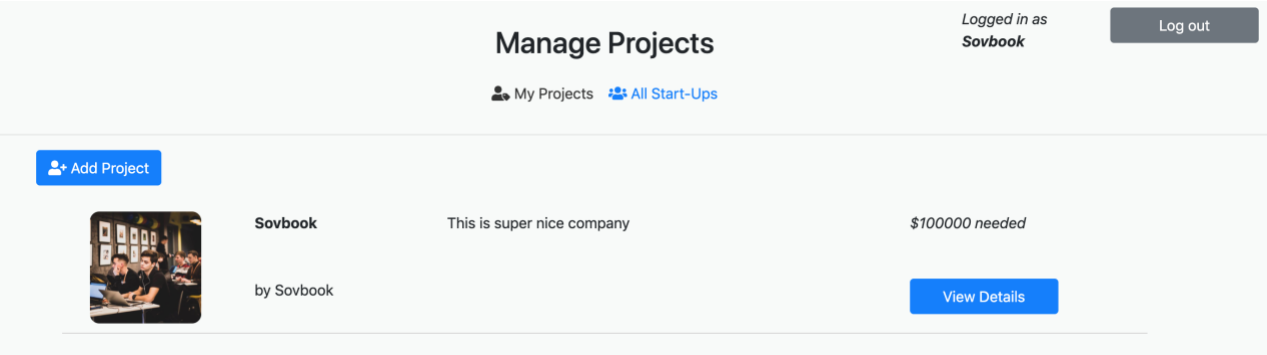

Рисунок 26 – Наші проекти

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

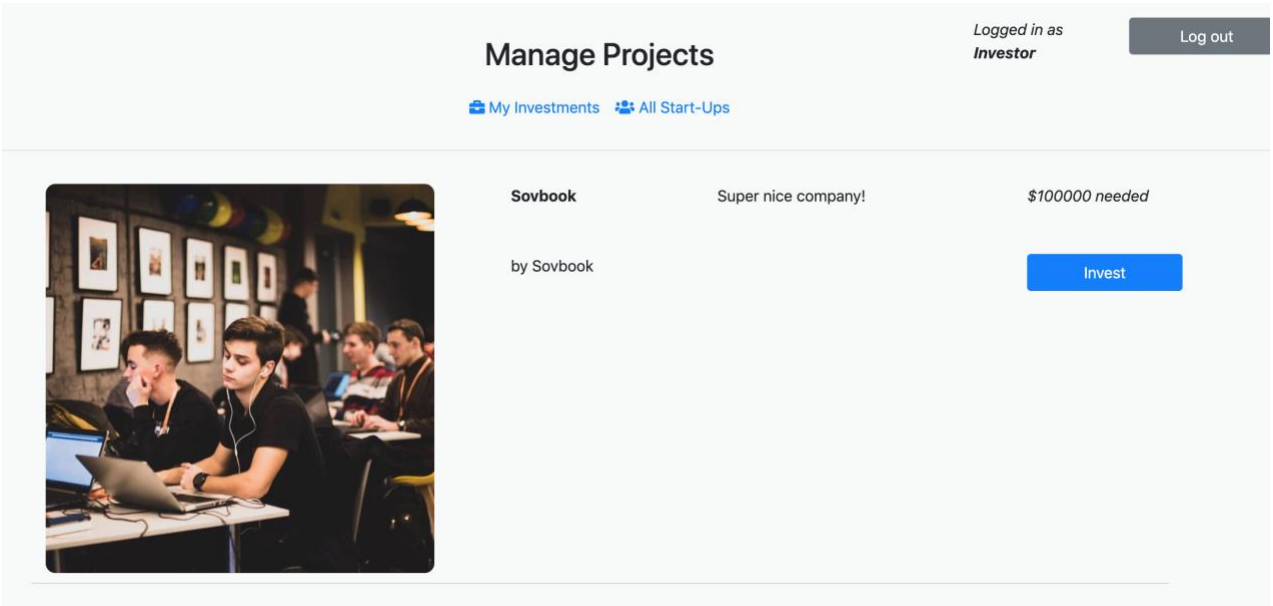

Рисунок 27 – інвестування у конкретний проект

Після цього можемо бачити цей проект у списку наших проектів

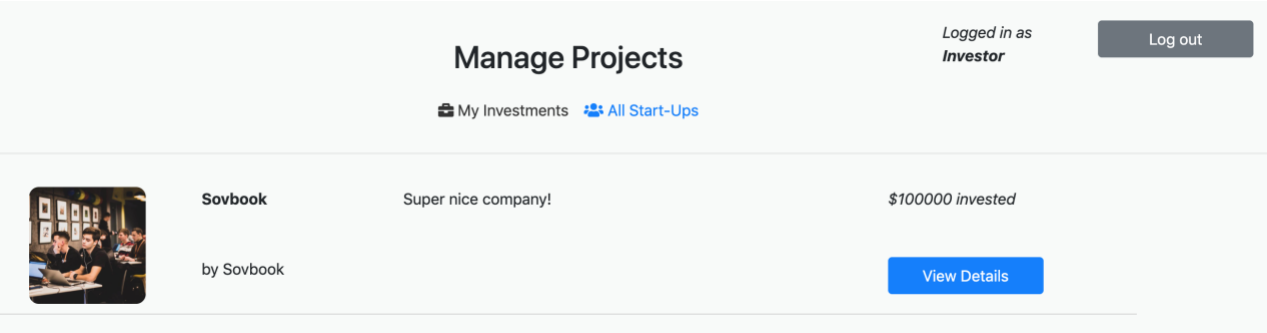

Рисунок 28 – проекти, в які ми проінвестували

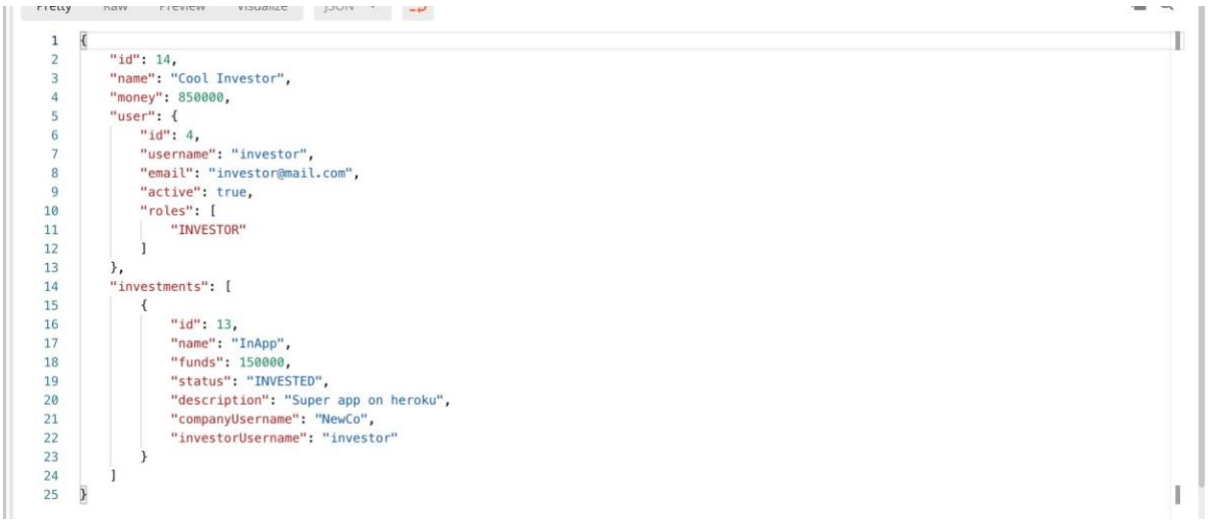

Рисунок 29 – відповідь від серверу після інвестування в проект

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

Додаток підтримує можливість авторизації і аутентифікації та не дозволяє анонімним користувачам робити запити на створення проекту, інвестування

в проект. Поточна версія не є остаточною та буде доповнюватися і розширюватися.

ВИСНОВОК

Виконання даної роботи підтверджує доцільність створення інвестиційного програмного забезпечення для розвитку економічних підприємств та інвестицій в перспективні галузі.

В ході розробки дипломного проекту було використано новітні технології, а саме мови програмування Java 8, Spring Framework, JWT security, JUnit & Mockito, Postman, React.js, HTML, CSS, які стрімко розвиваються та мають великий попит в розробці програмного забезпечення, основні принципи в проектуванні програмного забезпечення SOLID, MVC, REST, без яких складно уявити формування сьогоденних додатків.

Під час розробки додатку я навчився аналізувати ринок IT-галузі, порівнювати свій продукт з конкурентами, знаходити недоліки інших продуктів та впроваджувати бізнес-логіку у свій проект. На даний момент ринок інвестування в IT-проекти в Україні лише починає розвиватися, саме тому я обрав цю тему для подальшої праці.

З технічної точки зору я навчився розробляти web-додаток з back-end та front-end частиною, закріпив практичні навички та розвинув свої здібності у формуванні невеликого стартапу, використав різін мови програмування, спроектував архітектуру серверної частини, розробив REST API для взаємодії з сервером, написав інтерфейс користувача у стилі single-page-application та поєднав серверну та клієнтську частини, загорнув додаток у хмарний сервіс та провів публікацію коду на віддалений сервер для подальшого використання додатку іншими людьми, а також впровадив різноманітні бібліотеки для покращення функціональності та простоти коду. Також я провів тестування додатку – спочатку я ізольовано тестував серверну частину, а після подальшої інтеграції провів інтегроване тестування усього додатку разом. В цілому я навчився розробляти web-додаток і планую в подальшому розвивати створений проект, доповнюючи його новим функціоналом та покращуючи дизайн.

Перелік джерел

1. [1] – Що таке back-end - електроний режим доступу - https://otus.ru/nest/post/943/
2. [2] – SOLID-патерни - електроний режим доступу - https://en.wikipedia.org/wiki/SOLID
3. [3] – Що таке JVM - електроний режим доступу - https://tproger.ru/blogs/jvm-insides/
4. [4] – Spring Security – official Spring documentation – електроні дані https://spring.io/projects/spring-security
5. [5] – Що таке REST – архітектура - Вікіпедія - вільна енциклопедія - https://ru.wikipedia.org/wiki/REST
6. [6] – Unit-тестування – веб портал для програмістів – електроні дані - https://habr.com/ru/post/169381/
7. [7] – Бібліотека Junit в Java – короткий посібник – електроні дані - https://blog.ithillel.ua/articles/unit-testy-v-java.-kratkoe-rukovodstvo
8. [8] – Design Patterns – MVC Pattern – електроні дані - https://www.tutorialspoint.com/design_pattern/mvc_pattern.htm
9. [9] – Object/Relational Mapping – Hibernate – електроний режим доступу - https://hibernate.org/orm/
10. [10] – Spring Data – Spring web-documentation – електроні дані - https://spring.io/projects/spring-data
11. [11] – Introduction to JSON Web Tokens – електроні дані - https://jwt.io/introduction/
12. [12] – Фреймворк React – Вікіпедія – елеткроні дані - https://ru.wikipedia.org/wiki/React
13. [13] – React – офіційна онлайн-ддокументація – електроні дані - https://ru.reactjs.org/

14. [14] – Життєвий цикл компонента React – електроні дані - https://metanit.com/web/react/2.6.php

15. 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/InApp - back-end
2) https://github.com/Sevatkaaa/in-app - front-end
3) Код back-end, що функціонує на хмарному сервері https://in-app-rest.herokuapp.com/
4) Код front-end, що функціонує на хмарному сервері https://in-app-front.herokuapp.com/