Міністерство освіти і науки України
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ «КИЄВО-МОГИЛЯНСЬКА АКАДЕМІЯ»
Кафедра мультимедійних систем факультету інформатики

Специфіка розробки бекенду на Spring Boot та фронтенду на React

Текстова частина до кваліфікаційної роботи
за спеціальністю «Інженерія програмного забезпечення» 121

Керівник кваліфікаційної роботи
ст. викл. кафедри мультимедійних систем
Борозенний С. О.

_____
(підпис)
«___» _____ 2023 р.

Виконала студентка ІПЗ-4
Андрієнко Є. С.
«___» _____2023 р.

Київ 2023

Тема: Специфіка розробки бекенду на Spring Boot та фронтенду на React

Календарний план виконання роботи:

№ п/п	Назва етапу кваліфікаційної роботи	Термін виконання етапу	Примітка
1.	Отримання завдання на кваліфікаційну роботу	26.10.2022	
2.	Огляд літератури за темою роботи	12.12.2022	
3.	Описання особливостей фреймворку Spring Boot і бібліотеки React	16.01.2023	
4.	Проектування архітектури вебзастосунку та бази даних	13.02.2023	
5.	Реалізація вебзастосунку	06.03.2023	
6.	Розробка рекомендацій щодо використання фреймворку Spring Boot і бібліотеки React	03.04.2023	
7.	Аналіз роботи з керівником і корегування з урахуванням зауважень	10.04.2023	
8.	Оформлення презентації.	05.05.2023	
9.	Захист кваліфікаційної роботи	31.05.2023	

Студентка Андрієнко Єлизавета Сергіївна

Керівник Борозенний Сергій Олександрович

“_____”_____

ЗМІСТ

АНОТАЦІЯ

Кваліфікаційна робота складається з трьох розділів.

Перший розділ присвячено загальному огляду характеристик фреймворку Spring Boot і бібліотеки React, їх порівнянню з аналогами, такими як Quarkus, Micronaut, Vert.x, Helidon, Vue, Angular, Ember. Також досліджено для чого використовують Spring Boot і React та визначено, які перспективи на майбутнє мають ці технології.

Другий розділ спрямовано на проектування архітектури вебзастосунку для обробки заявок, її бази даних і описано технології, які будуть використовуватися при розробці.

У третьому розділі описано початкові кроки для створення full-stack застосунку за допомогою Spring Boot і React, висвітлено основні етапи розробки серверної та клієнтської частин, а також продемонстровано результати при тестуванні програми. Також запропоновано певні рекомендації щодо використання Spring Boot і React.

ВСТУП

Зі зростанням попиту на вебдодатки розробники постійно шукають інструменти та технології, які можуть покращити їх продуктивність та допомогти їм швидко створювати високоякісні продукти. Spring Boot та React є двома такими технологіями, які стали все більш популярними серед розробників через їх простоту використання, масштабованість та надійність.

Комбінація Spring Boot та React надає потужне середовище розробки, яке дозволяє розробникам створювати потужні вебдодатки з високим рівнем функціональності та залучення користувачів. Однак, використання цих технологій вимагає глибокого розуміння їх специфіки та того, як їх можна інтегрувати для створення позитивного користувацького досвіду.

Spring Boot спрощує процес розробки, надаючи попередньо налаштовані налаштування, зменшуючи шаблонний код та дозволяючи розробникам зосередитись на бізнес-логіці. З іншого боку, React пропонує модульний та компонентний підхід до веброзробки, що полегшує повторне використання коду та підтримку сталості в додатку.

Дана тема є актуальною, оскільки дослідження специфіки розробки серверної та клієнтської частини програми, реалізованих за допомогою Spring Boot та React відповідно, може допомогти розробникам покращити свої навички та підвищити свої шанси на успіх у роботі. Також це сприяє розвитку загальної бази знань про розробку вебзастосунків, дозволяє отримати глибше розуміння того, як покращити процес розробки та створити кращі продукти, розробити нові інструменти та техніки, які в подальшому покращать ефективність та якість розробки вебдодатків.

Метою цієї кваліфікаційної роботи є дослідження переваг та недоліків розробки бекенду на Spring Boot і фронтенду на React, а також опис

особливостей застосування даних технологій на практиці. Аналізуючи їх функціональні можливості, архітектурні принципи та рекомендовані практики, робота покликана допомогти розробникам отримати глибше розуміння процесу розробки з використанням вказаних технологій, а також вибрати оптимальний набір інструментів для реалізації повноцінного та ефективного вебдодатку.

Отже, дана робота має значимість як у науковому, так і у практичному аспектах, оскільки надає можливість вивчення інтеграції двох різних технологій для створення більш потужних інструментів та додатків. Вона сприяє поглибленню знань з розробки вебдодатків та надає відомості про інтеграцію різних технологій.

Постановка задачі:

1. Описати основні характеристики фреймворку Spring Boot і бібліотеки React.
2. Порівняти Spring Boot і React з аналогами та проаналізувати майбутні перспективи.
3. Розробити застосунок з бекендом на Spring Boot і фронтендом на React.
4. Розробити рекомендації щодо використання Spring Boot і React.

Об'єкт дослідження: створення вебзастосунку з серверною частиною, реалізованою на Spring Boot, та клієнтською частиною - на React.

Предмет дослідження: особливості, функціональні можливості, архітектурні принципи, інструменти, практики та способи взаємодії між бекендом на Spring Boot та фронтендом на React.

РОЗДІЛ 1. SPRING BOOT і REACT ЯК ПОПУЛЯРНІ ТЕХНОЛОГІЇ ДЛЯ РОЗРОБКИ FULL-STACK ЗАСТОСУНКУ

1.1 Основні характеристики Spring Boot і React

Spring Boot - це фреймворк на основі Java, який спрощує процес створення мікросервісів та вебдодатків [1]. React - це бібліотека JavaScript для створення інтерфейсів користувача [2]. Як Spring Boot, так і React мають унікальні характеристики, які роблять їх корисними у відповідних сферах. Розглянемо головні особливості фреймворку Spring Boot.

1. Spring Initializr

Spring Initializr - це інструмент, який запускає проекти Spring Boot. Ця функція Spring Boot дозволяє створювати проекти за допомогою cURL, кількох IDE та власного Spring CLI. Він не генерує код програми, але надає базову структуру проекту. Spring Initializr дозволяє вибрати проект, мову програмування і додати залежності.

Наприклад, створений проект містить специфікацію збірки Gradle або pom.xml у разі вибору проекту Gradle або Maven відповідно. Крім того, він складається з класу з методом main() для завантаження програми. Існує контекст програми, який використовує автоматичне налаштування Spring Boot і порожній файл властивостей, щоб була можливість додати властивості конфігурації.

2. Spring CLI

Spring Boot CLI - це командний рядок, який можна використовувати для швидкої розробки програми Spring. Spring CLI дозволяє виконувати сценарії Groovy без необхідності багаторазово писати код для повторного використання.

Ця функція Spring Boot містить кілька команд, які можуть допомогти використовувати Initializr для запуску розробки більш традиційного

проекту Java. Наприклад, команда init надає інтерфейс Initializr для створення базового проекту. Отриманий файл zip містить структуру проекту, до якої можна додати власну конфігурацію.

3. Автоконфігурація

Під час створення нового проекту Spring Boot дозволяє вибрати залежності для проекту. На основі цих залежностей функція автоконфігурації завантажує певні конфігурації за замовчуванням. Клас AutoConfiguration містить анотації @Conditional, які активують bean-компоненти за певних обставин. Щоб застосувати автоконфігурацію, потрібно використовувати анотації @EnableAutoConfiguration або @SpringBootApplication.

4. Зовнішня конфігурація

У програмах Spring Boot усі параметри конфігурації зчитуються з файлу ресурсів application.properties або application.yml. Однак існують випадки, коли потрібно перемістити конфігурацію з одного середовища в інше. Саме тоді доведеться налаштувати ці властивості, для чого потрібно буде перебудувати та повторно протестувати програму в усіх середовищах. Додатково, щоразу, коли відбуваються зміни, програму також доведеться повторно розгортати у робочому середовищі.

Щоб подолати цю проблему, Spring Boot дає змогу вивести конфігурацію назовні. Екстерналізація конфігурації означає використання коду програми, який використовується в одному середовищі, у зовнішньому середовищі. Для зовнішньої конфігурації можна використовувати такі файли: файли властивостей (properties), файли YAML, змінні середовища тощо.

5. Actuator

Платформа Actuator - це структура, яку можна використовувати, не встановлюючи іншу програму чи інструмент, для перевірки кінцевих точок та перевірки працездатності програми. Вона допомагає

проаналізувати програму, перевірити, які bean-компоненти налаштовані, кількість викликів певної служби або кількість разів, коли певна служба вийшла з ладу. Spring Actuator забезпечує простий спосіб відстеження стану, показників, інформації тощо.

Нижче наведено головні особливості бібліотеки React.

1. Віртуальний DOM

Ця характеристика React допомагає пришвидшити процес розробки програми та забезпечує гнучкість. Алгоритм полегшує реплікацію вебсторінки у віртуальній пам’яті React. Таким чином, оригінальний DOM представлений віртуальним DOM.

Щоразу, коли програма змінюється або оновлюється, віртуальний DOM знову відображає весь інтерфейс користувача шляхом оновлення компонентів, які було змінено. Це скорочує час і витрати на розробку.

2. JavaScript XML або JSX

Це синтаксис розмітки, який описує зовнішній вигляд інтерфейсу програми. Він робить синтаксис подібним до HTML і використовується розробниками при роботі з інтерфейсом користувача в коді JavaScript. Замість того, щоб штучно відокремити технології, розмістивши розмітку і логіку в окремих файлах, React розділяє відповідальність між вільно зв’язаними одиницями, що містять обидві технології і називаються «компонентами» [3].

3. Одностороння прив'язка даних

React використовує потік даних, який є односпрямованим, змушуючи розробників використовувати функцію зворотного виклику для редагування компонентів і не дозволяючи їм редагувати їх безпосередньо. Управління потоком даних з однієї точки досягається за допомогою компонента архітектури програми JavaScript під назвою Flux. Це дає розробникам кращий контроль над додатком, робить його більш гнучким

і ефективним, особливо, коли проект має динамічні дані, які потрібно оновлювати.

5. Декларативний UI

Декларативний UI означає опис того, як повинен виглядати інтерфейс у певний момент часу, а не прописування кожного кроку вручну для зміни його стану. Тобто розробник визначає бажаний стан UI, а React відповідає за оновлення реального вигляду, враховуючи цей стан. У декларативному підході до побудови інтерфейсу використовуються React-компоненти, які представляють собою невеликі, самодостатні блоки UI. Цей підхід робить код React більш читабельним і дозволяє легше виправляти помилки.

6. Компонентна архітектура

Інтерфейс користувача програми на основі React складається з кількох компонентів, кожен із яких має свою особливу логіку, написану на JavaScript.

Низхідний потік даних у React гарантує, що на батьківську структуру не впливатимуть зміни, внесені до дочірньої структури. Щоб змінити об’єкт, розробник має лише відредагувати його стани та внести відповідні зміни; таким чином буде оновлено лише окремий компонент, а інші залишаться незмінними.

1.2 Порівняння Spring Boot і React з аналогами

1.2.1 Аналоги Spring Boot

Quarkus - це фреймворк Kubernetes Native Java, розроблений для OpenJDK HotSpot і GraalVM.

Quarkus орієнтований на оптимальну продуктивність і невеликий об'єм пам'яті. Він використовує технологію GraalVM для компіляції додатків в нативний код, що забезпечує швидкий запуск. Spring Boot має більше конфігураційних можливостей та велику екосистему. Він використовує

віртуальну машину Java (JVM) для виконання, що забезпечує високу продуктивність, але може мати трохи довший час запуску.

Крім того, Spring Boot має велику спільноту користувачів та підтримку, що сприяє розширенню та масштабуванню додатків. Він надає широкий спектр інтеграційних можливостей та різноманітні модулі для різних потреб. Quarkus тільки набуває популярності, однак також надає набір розширень (extensions), які спрощують інтеграцію з іншими технологіями та фреймворками.

Spring Boot побудований на основі парадигми інверсії керування (IoC) та використовує анотації для опису конфігурації та залежностей. Quarkus базується на реактивній моделі програмування та підтримує асинхронність. Він використовує анотації та декларативні способи для опису конфігурації та роутингу.

Spring Boot має добре розроблену підтримку мікросервісної архітектури. Він надає інструменти для створення, конфігурації та керування мікросервісами, такі як Spring Cloud та Netflix OSS. Quarkus також підтримує мікросервіси та розробку Cloud Native додатків. Він пропонує інтеграцію з Kubernetes і OpenShift, а також вбудовану підтримку для розширення мікросервісної архітектури.

Micronaut досить молодий фреймворк порівняно з Spring Boot. Оскільки він заснований на «cloud-native» підході, багато функцій безпосередньо інтегровано у фреймворк, тоді як Spring Boot потребує залежності від сторонніх хмарних служб або бібліотек.

Якщо порівнювати легкість створення та запуску нової програми, і Micronaut, і Spring Boot пропонують кілька зручних методів для цього. Наприклад, у випадку обох фреймворків можна створити програму з інтерфейсом командного рядка. Крім того, можна використовувати Spring Initializr для Spring Boot або подібний інструмент для Micronaut під назвою Launch.

Підтримка мови майже ідентична для Spring Boot і Micronaut. Для обох фреймворків можна вибрати між Java, Groovy або Kotlin, а також Gradle або Maven.

Щодо сервлетів, у випадку Spring Boot, програма за замовчуванням використовуватиме Tomcat. Однак Spring Boot можна налаштувати на використання Jetty або Undertow. Програми Micronaut працюватимуть на HTTP-сервері Netty, який можна замінити на Tomcat, Jetty або Undertow.

Spring Boot пропонує п’ять стратегій авторизації: базову, авторизацію через форму, JWT, SAML і LDAP. Micronaut має ті самі параметри, крім SAML. Обидва фреймворки забезпечують підтримку OAuth2. З точки зору фактичного застосування безпеки, обидва фреймворки дозволяють використовувати анотації для захисту методів.

Vert.x - це реактивний фреймворк, який базується на подіях (event-driven) та неблокуючому введенні/виведенні (non-blocking I/O). Основними частинами Vert.x є Verticles і Event Bus. Verticles - це фрагменти коду, які виконуються. Event Bus - це «нервова система» будь-якої програми, яка використовує Vert.x і дозволяє Verticles комунікувати одна з одною [4].

Vert.x спрямований на максимальну продуктивність та швидкість виконання. Неблокуюча модель дозволяє ефективно використовувати ресурси та обробляти багатопоточність. Spring Boot зосереджується на зручності розробки та широкій функціональності. Він може мати більше витрат на швидкість, оскільки використовує більш традиційну багатопоточну модель.

Порівняно з Spring Boot, вибір розширень та інструментів для інтеграції з іншими технологіями може бути обмежений.

Helidon - набір бібліотек для мікросервісів. Він створений і підтримується Oracle. Для його використання підходить Java і Kotlin, тоді як Spring Boot підтримує також Groovy. Він представлений у двох варіантах: Helidon SE і Helidon MP.

Helidon SE - це набір інструментів, що забезпечує реактивні потоки, асинхронне та функціональне програмування та API у вільному стилі. Він пропонує реактивний API для потоків, обміну повідомленнями та баз даних. Можна створювати ендпоінти на основі технологій GraphQL, gRPC, WebSockets і серверів HTTP. Далі можна захистити їх за допомогою спеціального доповнення безпеки.

Helidon MP є реалізацією специфікації MicroProfile. Серед функцій, згаданих для версії SE, тут є підтримка JPA, служб RESTful, безпеки за допомогою JWT і тривалих дій (long-running actions) на основі шаблонів SAGA [5]. Крім того, ця версія призначена для роботи в хмарі; можна розгорнути програму на OCI або Kubernetes.

Так як Helidon є досить новим фреймворком порівняно з Spring Boot, то і документація та кількість інформації, яку можна знайти в інтернеті, буде відповідно меншою.

1.2.2 Аналоги React

VueJS - це фреймворк Javascript, який має відкритий код, часто використовується для розробки односторінкових програм.

З точки зору продуктивності Vue швидший за React. Vue також робить розробку програм більш простою, оскільки пропонує плавний процес інтеграції без впливу на всю систему, незалежно від того, чи йдеться про односторінкову програму чи складний вебінтерфейс. Як і більшість альтернатив React, Vue має менший розмір бібліотеки та пропонує кращу продуктивність.

React базується на віртуальному DOM і оновлює тільки ті частини інтерфейсу, які змінилися, що робить його ефективним при великій кількості компонентів. Vue використовує реактивну систему, яка автоматично відслідковує зміни в стані і оновлює відповідні компоненти. Це робить його зручним для роботи зі станом та реагування на зміни.

React використовує для проектування Javascript XML, відомий як JSX, тоді як Vue дозволяє використовувати шаблони та плагіни JSX і HTML.

AngularJS - це фреймворк із відкритим кодом, який допомагає впорядковувати та спрощувати код Javascript для створення динамічних онлайн, десктопних і мобільних додатків.

На відміну від React, Angular використовує TypeScript, який є строго типізованою версією Javascript. Він має свій синтаксис та вимагає певного розуміння ООП-принципів. Розробка в Angular базується на компонентах та шаблонах.

Оскільки Angular є повноцінним фреймворком, що включає в себе багато вбудованих функцій та модулів, розмір кінцевого додатку може бути більшим. Angular має більш складну структуру, що може вплинути на продуктивність, особливо при великих проектах. React, натомість, є бібліотекою, а не повноцінним фреймворком. Це дозволяє більшу гнучкість та можливість вибору додаткових бібліотек та інструментів для потреб проекту. Розмір кінцевого додатку може бути меншим, а продуктивність - вищою. Однак тестування та налагодження відбувається за замовчуванням у Angular, тоді як у React потрібні додаткові компоненти для тестування та налагодження вашого проекту.

Ember - це фреймворк, який включає всі технічні можливості інтерфейсу, наприклад перегляд широкого діапазону станів програми, завдяки маршрутизації. Ember пропонує рендеринг на стороні клієнта, підтримку URL-адрес, механізм створення шаблонів.

Подібно до Angular Ember включає в себе багато вбудованих функцій та модулів, тому дозволяє розробникам швидше стартувати проект і мати стандартизовану структуру.

На відміну від React, Ember базується на моделі MVVM (Model-View-ViewModel). Також Ember пропонує двостороннє зв’язування даних, а React - одностороннє.

До особливостей фреймворку Ember можна віднести автоматичну оновлюваність стану (automatic state updates). Ember має вбудовану систему спостереження, яка автоматично оновлює стан додатку, коли дані змінюються. Це зменшує необхідність вручну оновлювати стан та дозволяє легко синхронізувати дані між компонентами. Також Ember має вбудований модуль Ember Data, який спрощує роботу з серверними даними та взаємодіє з API. Він надає шар абстракції для роботи з даними, включаючи запити, збереження, оновлення та видалення даних.

1.3 Використання Spring Boot і React, перспективи на майбутнє

Важливою тендецією у майбутньому розвитку Spring Boot є інтеграція з хмарними технологіями. Оскільки все більше компаній переходять до хмарної інфраструктури, Spring Boot, ймовірно, продовжуватиме розвиватися, щоб краще інтегруватися з хмарними технологіями, такими як Kubernetes, Docker і AWS. Проект Spring Cloud, який надає інструменти для створення хмарних додатків за допомогою Spring, є ознакою цієї тенденції.

Реактивне програмування набуває популярності завдяки своїй здатності обробляти великі обсяги даних і складні програми, керовані подіями. Ймовірно, Spring Boot продовжить розвиватися, щоб підтримувати реактивне програмування, спираючись на реактивні функції, які вже присутні у Spring 5.

Spring Boot вже є модульною системою з широким набором компонентів, які можна використовувати за потреби. Однак у міру того, як фреймворк продовжує розвиватися, цілком імовірно, що більше уваги приділятиметься тому, щоб зробити його більш модульним, дозволяючи розробникам вибирати лише ті компоненти, які їм потрібні для їх застосування.

Хоча Spring Boot вже є швидкою та ефективною системою, завжди є місце для вдосконалення. Майбутні версії фреймворку, ймовірно, включатимуть оптимізацію для подальшого підвищення продуктивності, наприклад, скорочення часу запуску та використання пам’яті.

Зі швидким розвитком Spring Boot стає очевидним, що надалі буде покращуватись документація, навчальні посібники та інструменти розробки для того, щоб залишатись простим у використанні та зрозумілим для вивчення.

React переважно використовується для створення програм на стороні клієнта, однак рендеринг на стороні сервера стає все більш важливим для продуктивності та SEO. Тому доволі імовірно, що майбутні версії React включатимуть удосконалення візуалізації на стороні сервера, щоб зробити її простішою та ефективнішою.

React вже відомий своєю продуктивністю, але оскільки вебдодатки стають все більш складними, завжди є місце для вдосконалення. Майбутні версії React, ймовірно, включатимуть оптимізації для подальшого покращення продуктивності, такі як зменшення розміру пакета та збільшення швидкості візуалізації.

React має великий набір інструментів і бібліотек, і він продовжуватиме зростати в майбутньому. У міру появи нових технологій React повинен розвиватися для інтеграції з ними. Наприклад, зростання популярності WebAssembly може спричинити появу у React функцій, які полегшать роботу з модулями WebAssembly.

РОЗДІЛ 2. ПРОЕКТУВАННЯ ПРОГРАМИ

2.1 Архітектура програми

На рис. 2.1 зображено схему взаємодії бекенду та фронтенду програми.

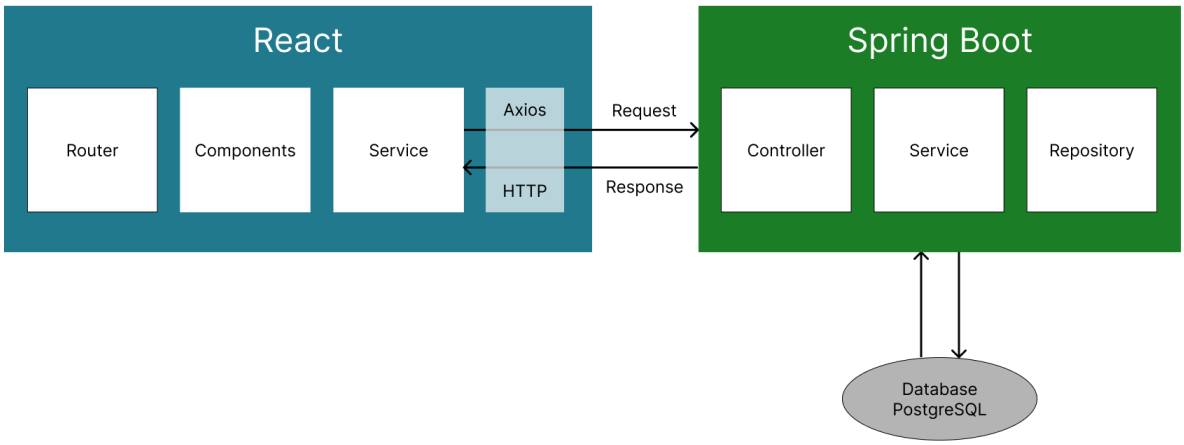

Рисунок 2.1 - Взаємодія бекенду та фронтенду

На фронтенді Router використовується для маршрутизації і навігації між різними сторінками додатку. Він відповідає за визначення шляхів URL та зв'язує їх з відповідними компонентами.

Components є будівельними блоками фронтенду, які відповідають за рендеринг та взаємодію з користувачем. Вони можуть містити логіку, стилізацію та передавати дані до інших компонентів.

Service використовується для взаємодії з бекендом. Він відправляє HTTP-запити до відповідних ендпоінтів бекенду на Spring Boot, використовуючи бібліотеку Axios, і отримує відповіді.

У бекенді Controller відповідає за обробку HTTP-запитів, вхідних параметрів та передачу даних до інших компонентів. Він мапує URL-шляхи на методи контролера, обробляє отримані дані та повертає HTTP-відповіді з результатами.

Service містить бізнес-логіку додатку. Він обробляє отримані дані з контролера, виконує операції з базою даних (через Repository), виконує розрахунки та маніпулює даними для підготовки відповідей.

Repository взаємодіє з базою даних PostgreSQL за допомогою Spring Data JPA . Він виконує операції зчитування, створення, оновлення та видалення даних з бази даних. Spring Boot спрощує взаємодію з базою даних шляхом автоматичного генерування SQL-запитів на основі методів, анотованих у репозиторіях.

Така архітектура дозволяє розділити відповідальності між серверною та клієнтською частинами додатку і забезпечити більшу гнучкість та масштабованість системи.

2.2 Архітектура бази даних

Побудова ER-моделі даних (рис. 2.2)

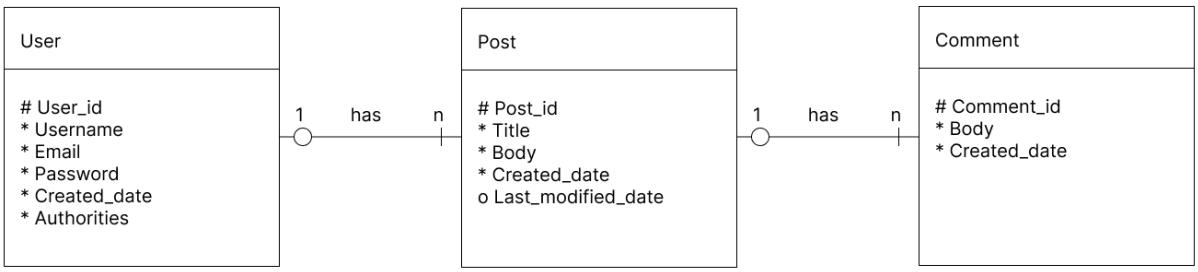

Рисунок 2.2 - ER-модель даних

Опис таблиць та компонентів:

1. «User» - таблиця, що відповідає даним про користувача системи. Містить атрибути:

- User_id – первинний ключ, ідентифікатор користувача в базі даних, що є унікальним
- Username – ім’я користувача, обов’язковий
- Роль – роль користувача, обов’язковий атрибут
- Email – електронна пошта користувача, обов’язковий атрибут

- Password – пароль користувача у зашифрованому вигляді, обов’язковий атрибут
- Created_date – дата реєстрації користувача, обов’язковий атрибут
- Authorities – роль користувача, обов’язковий атрибут. Можливі значення: user, admin

2. «Post» - таблиця, що відповідає даним про пост користувача. Містить атрибути:

- Post_id – первинний ключ, ідентифікатор посту в базі даних, що є унікальним
- Title – назва посту, обов’язковий атрибут
- Body – зміст посту, обов’язковий атрибут
- Created_date – дата створення посту користувачем, обов’язковий атрибут
- Last_modified_date – дата останньої зміни посту користувачем, необов’язковий атрибут

3. «Comment» - таблиця, що відповідає даним про коментар посту. Містить атрибути:

- Comment_id – первинний ключ, ідентифікатор коментаря в базі даних, що є унікальним
- Body – зміст коментаря, обов’язковий атрибут
- Created_date – дата створення коментаря користувачем, обов’язковий атрибут

Після побудови ER-моделі виконується перехід до реляційної моделі даних із дотриманням властивостей реляції.

Реляційна модель даних - це логічна модель даних із відношеннями, які є таблицями й атрибутами - рядки.

Властивостями реляції є:

- назви атрибутів та відношення повинні бути унікальними;
- кортежі повинні бути унікальними;

- порядок слідування атрибутів та кортежів у відношенні не має значення.

Реляційна модель даних показана на рис. 2.3 – 2.7.

User				
Ключ	Ім'я атрибуту	Тип атрибуту	NULL / NOT NULL	Пояснення (перехід від ER-моделі) FK: ON DELETE, ON UPDATE
PK	User_id	Integer	NOT NULL	Первинний ключ. Атрибут простий обов'язковий «Ідентифікатор користувача»
	Username	String	NOT NULL	Атрибут простий обов'язковий «Ім'я користувача»
	Email	String	NOT NULL	Атрибут простий обов'язковий «Електронна пошта користувача»
	Password	String	NOT NULL	Атрибут простий обов'язковий «Пароль користувача»
	Created_date	DateTime	NOT NULL	Атрибут простий обов'язковий «Дата реєстрації користувача»

Рисунок 2.3 - Сутність "User"

Authority				
Ключ	Ім'я атрибуту	Тип атрибуту	NULL / NOT NULL	Пояснення (перехід від ER-моделі) FK: ON DELETE, ON UPDATE
PK	Name	String	NOT NULL	Первинний ключ. Атрибут простий обов'язковий «Назва ролі»

Рисунок 2.4 - Сутність "Authority"

User_authority				
Ключ	Ім'я атрибуту	Тип атрибуту	NULL / NOT NULL	Пояснення (перехід від ER-моделі) FK: ON DELETE, ON UPDATE
PPK, FK1	User_id	Integer	NOT NULL	Зв'язок з таблицею User: user_id. Зв'язок обов'язковий з обох боків. ON DELETE CASCADE, ON UPDATE NO ACTION Автоматично видаляти запис з User_authority при видаленні користувача, заборонити оновлювати ідентифікатор користувача.
PPK, FK2	Authority_name	String	NOT NULL	Зв'язок з таблицею Authority: Authority_name. Зв'язок обов'язковий з обох боків. ON DELETE CASCADE, ON UPDATE CASCADE Автоматично видаляти запис з User_authority при видаленні ролі, автоматично оновлювати запис в User_authority при зміні назви ролі.

Рисунок 2.5 - Сутність "User_authority"

Post				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL / NOT NULL	Пояснення (перехід від ER-моделі) FK: ON DELETE, ON UPDATE
PK	Post_id	Integer	NOT NULL	Первинний ключ. Атрибут простий обов’язковий «Ідентифікатор поста»
	Title	String	NOT NULL	Атрибут простий обов’язковий «Назва поста»
	Body	String	NOT NULL	Атрибут простий обов’язковий «Зміст поста»
	Created_date	DateTime	NOT NULL	Атрибут простий обов’язковий «Дата створення поста»
	Last_modified_date	DateTime	NULL	Атрибут простий необов’язковий «Дата останньої зміни поста»
FK	User_id	Integer	NOT NULL	Зв’язок з таблицею User: User _id. Зв’язок обов’язковий з боку Post. ON DELETE NO ACTION, ON UPDATE NO ACTION Заборонити видаляти пост при видаленні користувача, заборонити оновлювати ідентифікатор користувача.

Рисунок 2.6 - Сутність "Post"

Comment				
Ключ	Ім’я атрибуту	Тип атрибуту	NULL / NOT NULL	Пояснення (перехід від ER-моделі) FK: ON DELETE, ON UPDATE
PK	Comment_id	Integer	NOT NULL	Первинний ключ. Атрибут простий обов’язковий «Ідентифікатор коментаря»
	Body	String	NOT NULL	Атрибут простий обов’язковий «Зміст коментаря»
	Created_date	DateTime	NOT NULL	Атрибут простий обов’язковий «Дата створення коментаря»
FK	Post_id	Integer	NOT NULL	Зв’язок з таблицею Post: Post_id. Зв’язок обов’язковий з боку Comment. ON DELETE CASCADE, ON UPDATE NO ACTION Автоматично видаляти коментар при видаленні поста, заборонити оновлювати ідентифікатор поста.

Рисунок 2.7 - Сутність "Comment"

2.3 Огляд технологій для реалізації

Maven - це інструмент, який використовується для створення та керування будь-яким проектом на основі Java [7]. Основна мета Maven полягає в

тому, щоб розробник міг швидко зрозуміти поточний стан розробки. Для досягнення цієї мети Maven вирішує декілька проблемних сфер, включаючи спрощення процесу створення проекту, забезпечення єдиної системи збірки, надання якісної інформації про проект та підтримку кращих практик розробки.

Використання Maven не звільняє розробника від необхідності знати основні механізми, але воно допомагає уникнути багатьох деталей. Maven будує проект, використовуючи свою об'єктну модель проекту (POM) та набір плагінів. Він надає корисну інформацію про проект, таку як журнал змін, отриманий безпосередньо з системи керування версіями, перехресні посилання на джерела, керовані проектом, списки розсилки, використовувані проектом залежності та звіти про модульне тестування, включаючи покриття.

PostgreSQL - це об’єктно-реляційна база даних з відкритим вихідним кодом, яка використовує та розширює мову SQL у поєднанні з багатьма функціями, які безпечно зберігають і масштабують найскладніші дані [8]. Головними перевагами PostgreSQL є перевірена архітектура, надійність, цілісність даних, розширюваність та набір потужних функцій. Вона працює на всіх основних операційних системах, відповідає вимогам ACID з 2001 року (Atomicity, Consistency, Isolation, Durability, тобто атомарність, узгодженість, ізоляція та надійність) та має розширення, такі як популярний географічний розширювач бази даних PostGIS.

PostgreSQL намагається дотримуватися стандарту SQL, якщо це не суперечить традиційним функціям або не призводить до недоліків в архітектурі. Багато функцій, які потрібні стандарту SQL, підтримуються, хоча іноді з іншим синтаксисом або функціями. З появою версії 15 у жовтні 2022 року PostgreSQL відповідає принаймні 170 з 179 обов'язкових функцій для відповідності SQL:2016 Core, що робить його єдиною реляційною базою даних, яка відповідає цьому стандарту.

Було доведено, що PostgreSQL має високу масштабованість як для обробки великого обсягу даних, так і для обслуговування багатьох користувачів одночасно.

Spring Security - це фреймворк, який забезпечує аутентифікацію, авторизацію та захист від типових атак [9]. Цей фреймворк є стандартом для захисту програм на основі Spring і надає як імперативну, так і реактивну підтримку захисту. У Spring Boot є стартовий модуль spring-boot-starter-security, який об'єднує необхідні залежності, пов'язані з Spring Security.

Якщо Spring Security присутній в classpath (шлях для пошуку класів), то вебдодатки захищені за замовчуванням. Щоб додати захист на рівні методу для вебпрограми, також можна використовувати анотацію @EnableGlobalMethodSecurity з необхідними налаштуваннями.

Основні функції, доступні за замовчуванням, включають:

- Компонент UserDetailsService (або ReactiveUserDetailsService для програм на основі WebFlux) зі сховищем у пам'яті та одним користувачем зі згенерованим паролем.
- Вхід на основі форми або базова безпека HTTP (в залежності від заголовка "Accept" у запиті) для всієї програми, включаючи кінцеві точки (endpoint), що можуть бути на шляху до класів.
- DefaultAuthenticationEventPublisher для публікації подій автентифікації.

Подібно до програм Spring MVC, програми на основі WebFlux можна захистити, додавши залежність spring-boot-starter-security. Конфігурація безпеки за замовчуванням реалізована в ReactiveSecurityAutoConfiguration та UserDetailsServiceAutoConfiguration. ReactiveSecurityAutoConfiguration імпортує WebFluxSecurityConfiguration для налаштування веббезпеки, а UserDetailsServiceAutoConfiguration налаштовує автентифікацію, що також є актуальним для звичайних програм. Щоб повністю вимкнути конфігурацію безпеки вебпрограми за замовчуванням, потрібно додати

бін типу WebFilterChainProxy (це не вимикає конфігурацію UserDetailsService або безпеку Actuator). Щоб також вимкнути конфігурацію UserDetailsService, треба додати бін типу ReactiveUserDetailsService або ReactiveAuthenticationManager.

Правила доступу можна налаштувати, додавши настроюваний SecurityWebFilterChain. Spring Boot надає зручні методи, які можна використовувати для перевизначення правил доступу до кінцевих точок (endpoint) та статичних ресурсів. EndpointRequest можна використовувати для створення ServerWebExchangeMatcher на основі властивості management.endpoints.web.base-path.

JSON Web Token (JWT) - це JSON-об'єкт, визначений стандартом RFC 7519, який слугує як самодостатній спосіб безпечної передачі інформації між сторонами [10]. JWT складається з трьох частин: заголовка (header), корисного навантаження (payload) та підпису (signature).

Заголовок часто містить тип токена (JWT) та використовуваний алгоритм підпису, такий як HMAC SHA256 або RSA.

Корисне навантаження містить заяви про сутність (зазвичай користувача) та додаткові дані. Існують три типи заяв: зареєстровані, публічні та приватні.

Зареєстровані заяви - це набір попередньо визначених заяв, які не є обов'язковими, але рекомендовані для надання корисних і сумісних заяв. Деякі з них: "exp" (термін дії), "sub" (тема) та інші.

Публічні заяви можуть бути визначені за бажанням сторін, які використовують JWT. Проте для уникнення колізій рекомендується визначати їх в реєстрі вебтокенів IANA JSON або як URI з простором імен, що містить стійкий до колізій ідентифікатор.

Приватні заяви - це спеціальні заяви, створені для обміну інформацією між сторонами, які домовилися про їх використання, і не є ні зареєстрованими, ні публічними заявами.

Node.js - це кросплатформне середовище виконання JavaScript із відкритим кодом. Node.js запускає двигун V8 JavaScript, ядро Google Chrome, поза браузером, що дозволяє Node.js бути продуктивним [11].

Оскільки Node.js працює в одному процесі, це дозволяє не створювати новий потік для кожного запиту. За допомогою набору асинхронних примітивів вводу/виводу, які містяться у стандартній бібліотеці Node.js, можна уникнути блокування коду JavaScript. Бібліотеки Node.js використовують неблокуючі парадигми, тому блокування є винятком, а не нормою. Замість блокування потоку і зайняття процесора під час операцій введення-виведення, якими є мережеве зчитування, доступ до бази даних або файлової системи, Node.js відкладає операції до отримання відповіді. Таким чином Node.js обробляє тисячі одночасних з'єднань з одним сервером без потреби управління паралельністю потоків, що може спричиняти помилки.

Однією з переваг Node.js є можливість працювати як на стороні сервера, так і на стороні клієнта без потреби використовувати іншу мову програмування.

Node.js також дозволяє розробнику самостійно вибирати, яку версію ECMAScript використовувати, змінюючи версію Node.js.

Npm скорочена форма Node Package Manager) - є найбільшим у світі реєстром програмного забезпечення. Він використовується для обміну пакетами і також діє як утиліта командного рядка для проектів Node.js, дозволяючи встановлювати пакети, керувати залежностями та версіями.

Зазвичай npm складається з трьох компонентів [12]:

1. Вебсайт корисний для пошуку пакетів для проекту, створення та налаштування профілів для керування та доступу до приватних і загальнодоступних пакетів.
2. Інтерфейс командного рядка (CLI) запускається з терміналу комп’ютера для взаємодії з пакетами та сховищами npm.

3. Реєстр - це велика загальнодоступна база даних проектів і метаінформації JavaScript. Можна використовувати будь-який підтримуваний реєстр npm.

Npm дозволяє легко адаптувати пакети коду для власних програм або використовувати їх без змін. Можна запускати пакети без завантаження за допомогою npx, обмежувати код для певних розробників та створювати організації для координації обслуговування пакетів, кодування та розробників. Крім того, за допомогою npm можна оновлювати програми після оновлення основного коду.

Webpack - це збірник статичних модулів для сучасних програм JavaScript [13].

Коли розробляється вебпрограма, яка складається з багатьох файлів і використовує сучасний JavaScript та інші препроцесори, наприклад SASS, браузери не завжди повністю підтримують ці технології. Через це потрібні додаткові інструменти, такі як Webpack, для перетворення коду зі сучасного JavaScript у JavaScript ES5, перетворення коду з SASS в CSS та виконання інших завдань. Після налаштування Webpack для фронтенду програми, він автоматично виконуватиме повторювані завдання. В результаті отримуємо дві версії проекту: вихідний код, який розробник пише з використанням сучасного JavaScript та інших зручних інструментів, і перетворений код для публікації на хостинг.

Основні поняття, пов'язані з використанням Webpack:

- Точка входу (entry point) визначає внутрішній граф залежностей. Webpack аналізує точку входу, щоб встановити, які модулі та бібліотеки пов'язані прямо чи непрямо. Кожна залежність перетворюється на файл, який називається "бандл" (bundle).
- Точка виходу (output) вказує, куди Webpack повинен зберігати зібрані бандли і як назвати ці файли (за замовчуванням, це "./dist").

- Завантажувачі (loaders) дозволяють Webpack обробляти різні типи файлів, необхідні для вашої програми, включаючи JavaScript. Завантажувачі перетворюють файли у модулі, які можна включити до графа залежностей програми і, отже, до бандлу.
- Плагіни (plugins) дозволяють виконувати додаткові завдання після збирання бандлу. Ці завдання можуть стосуватися як самого бандлу, так і іншого коду. Для використання плагіна потрібно його підключити за допомогою require() і додати до масиву плагінів. Більшість плагінів можна налаштувати за допомогою параметрів. Оскільки плагіни можуть використовуватися багаторазово для різних цілей, їх можна створювати кілька окремих екземплярів, використовуючи new.

РОЗДІЛ 3. РОЗРОБКА ВЕБЗАСТОСУНКУ

3.1 Аналіз функціональних вимог

Ролі користувачів у системі: гість (незареєстрований), користувач, адміністратор.

Технічні вимоги користувача до функціональності системи:

1. Реєстрація.

a. Наявність полів: ім’я, електронна пошта, пароль.

b. Динамічна валідація полів при заповненні форми.

Виведення підказок користувачу у разі неправильно заповнених полів. Наприклад, якщо пароль занадто короткий, користувач бачить відповідне повідомлення.

Реєстрація та перехід на головну сторінку відбувається тільки після введення коректних даних.

2. Аутентифікація.

Наявність власної електронної пошти та паролю у зареєстрованого користувача для входу в систему та доступу до своїх даних.

3. Створення поста.

Наявність форми для введення назви, опису поста. Автоматичне присвоєння дати.

4. Перегляд постів та коментарів.

Можливість переглядати пости та коментарі до них користувачами системи (аутентифікація не обов’язкова).

5. Редагування та видалення постів.

Можливість редагувати та видаляти власні пости аутентифікованим користувачем.

6. Коментування.

Можливість додавати коментарі аутентифікованим користувачем.

7. Перегляд користувачів.

Можливість адміністратора переглядати всіх користувачів, зареєстрованих у системі.

8. Вихід із системи.

3.2 Реалізація проекту

Програма складається з трьох основних частин: клієнт, сервер, база даних. Проект поділений на папки backend та frontend.

3.2.1 Структура серверної частини

Після генерації заготовки Spring Boot додатку з додаванням стандартних залежностей за допомогою Spring Initializr було створено проект, розбитий на два основні пакети: java та resources (рис. 3.1).

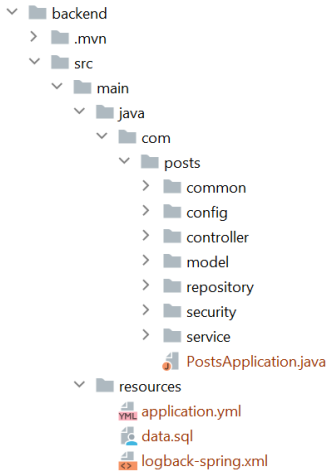

Рисунок 3.1 - Структура серверної частини

Пакет java містить всю реалізацію серверної частини програми.

Папка common (рис. 3.2) складається з підпапки exception, в якій містяться класи винятків, їх обробки, та підпапки утиліт, в якій міститься AuthoritiesConstants (перелік констант - назв ролей користувачів у системі).

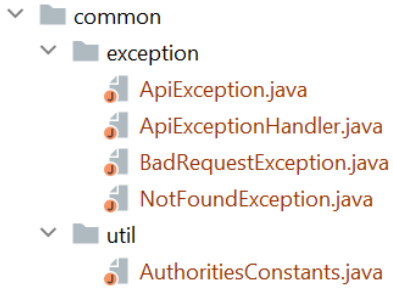

Рисунок 3.2 - Структура пакету common

config – пакет, який містить конфігурацію для Spring Security, що використовується при обробці запитів реєстрації, аутентифікації та авторизації користувача системи.

controller – пакет складається з контролерів, які обробляють запити та повертають у відповідь моделі з атрибутами за допомогою ResponseEntity<>. При створенні класів контролерів використовувалась анотація @RestController (рис. 3.3).

```
@RestController
@RequestMapping("/api")
public class PostController {
```

Рисунок 3.3 - Створення контролерів

Для обробки запитів використовувалась анотація @RequestMapping на рівні класів та анотації @GetMapping, @PostMapping, @PutMapping, @DeleteMapping, які є складеними анотаціями @RequestMapping і вказують на певний HTTP-метод (get, post, put, delete відповідно). Наприклад, @GetMapping діє як скорочений варіант для @RequestMapping(method = RequestMethod.GET) (рис. 3.4)

```
@GetMapping(value = "/posts")
public ResponseEntity<List<PostDto>> getPostList(Pageable pageable) {
    logger.debug("Request to get post list : {}", pageable);
    Page<Post> pagePost = postService.findAllByOrderByCreatedDateDescPageable(pageable);
    Page<PostDto> pagePostDto = pagePost.map(post -> new PostDto((post)));
    return new ResponseEntity<>(pagePostDto.getContent(), HttpStatus.OK);
}
```

Рисунок 3.4 - Обробка запиту GET

model – складається з чотирьох інших пакетів та класу BaseModel (рис. 3.5).

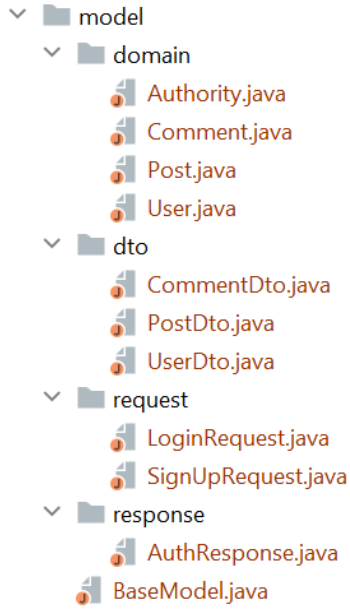

Рисунок 3.5 - Структура пакету model

Пакет domain містить класи, які описують таблиці в базі даних за допомогою анотації @Entity та автоматично створюють їх за допомогою анотації класу @Table і анотації поля класу @Column, що може задавати ім'я колонки таблиці, довжину і тд. Приклад реалізації такого класу наведено на рис. 3.6.

```
@Entity
@Table(name = "post")
@Getter
@Setter
public class Post {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    @Column(name = "post_id")
    private Long id;

    @Column(name = "title", nullable = false)
    @Length(min = 1)
    private String title;

    @Column(name = "body", columnDefinition = "TEXT")
    private String body;
```

Рисунок 3.6 - Клас Post

Пакет dto включає об’єкти передачі даних, що передають лише ті дані, якими потрібно поділитися з інтерфейсом користувача.

Пакети request і response містять моделі для передачі запиту та відповіді при реєстрації та авторизації користувача.

repository – включає репозиторії для роботи з таблицями в базі даних. Репозиторії - це інтерфейси, які дозволяють зменшити кількість шаблонного коду, необхідного для впровадження рівнів доступу до даних для різних сховищ постійності. У якості аргументів типу приймається клас домену, яким потрібно керувати, а також тип ідентифікатора класу домену. JpaRepository - це спеціальне розширення JPA (Java Persistence API) Repository. Він надає повний API CrudRepository та PagingAndSortingRepository, тобто містить API для основних CRUD операцій, а також для розбиття сторінок і сортування.

Наприклад, UserRepository (рис. 3.7) містить абстрактні методи для знаходження користувача за ідентифікатором, електронною поштою та перевіряє, чи існує користувач за його електронною поштою.

```
@Repository
public interface UserRepository extends JpaRepository<User, Long> {

    Optional<User> findOneById(Long id);
    Optional<User> findByEmail(String email);
    Optional<User> findOneWithAuthoritiesByEmail(String lowercaseEmail);
    Boolean existsByEmail(String email);
}
```

Рисунок 3.7 - Інтерфейс UserRepository

security – пакет, який містить файли, необхідні для налаштування Spring Security та аутентифікації за допомогою JWT.

Розглянемо схему головних класів для здійснення Spring Security та Jwt аутентифікації, яка зображена на рис. 3.8

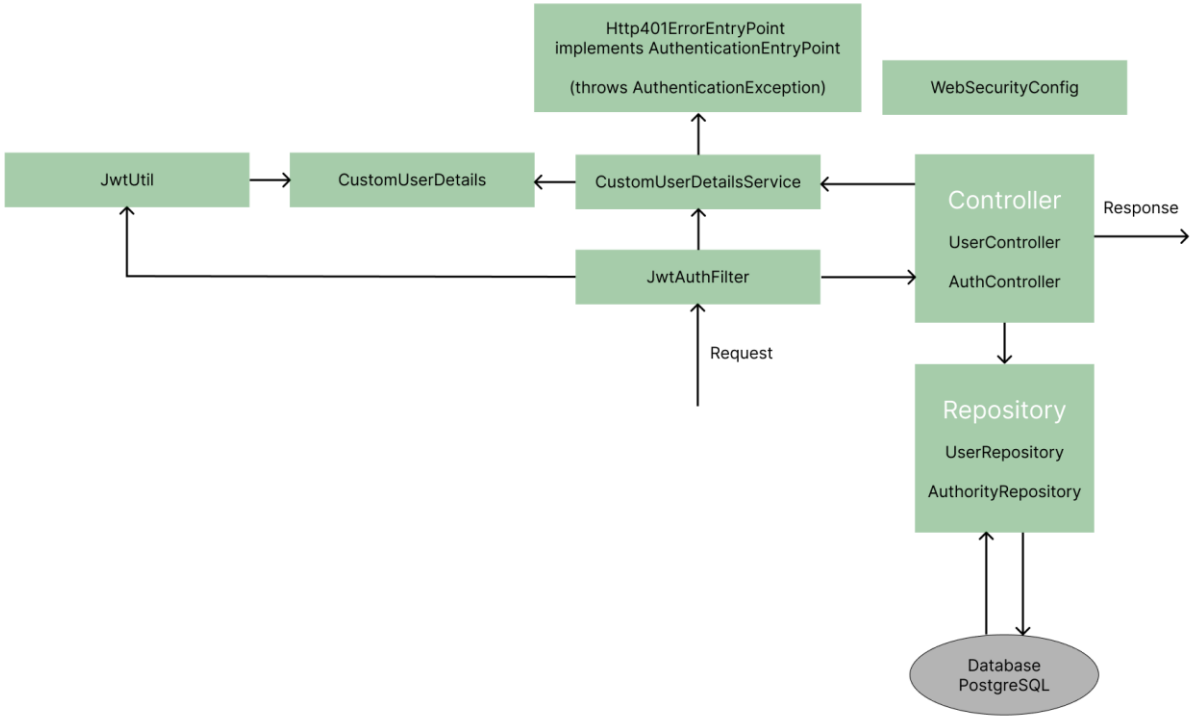

Рисунок 3.8 - Схема класів для реалізації аутентифікації за допомогою Spring Security і JWT

JwtAuthFilter попередньо обробляє HTTP-запит, надає метод doFilterInternal() (рис. 3.9), який реалізує синтаксичний аналіз та перевірку JWT, завантаження деталей користувача (за допомогою CustomUserDetailsService).

```
@Override
protected void doFilterInternal(HttpServletRequest request, HttpServletResponse response,
                                FilterChain filterChain) throws ServletException, IOException {
    try {
        String jwt = getJwtFromRequest(request);

        if (StringUtils.hasText(jwt) && jwtUtil.validateToken(jwt)) {
            String userId = jwtUtil.getUserIdFromToken(jwt);

            UserDetails userDetails = userDetailsService.loadUserByUsername(userId);
            UsernamePasswordAuthenticationToken authentication = new UsernamePasswordAuthenticationToken
                    (userDetails, null, userDetails.getAuthorities());
            authentication.setDetails(new WebAuthenticationDetailsSource().buildDetails(request));

            SecurityContextHolder.getContext().setAuthentication(authentication);
        }
    } catch (Exception ex) {
        System.out.println("Could not set user authentication in security context" + ex);
    }
    filterChain.doFilter(request, response);
}
```

Рисунок 3.9 - Реалізація методу doFilterInternal() у класі JwtAuthFilter

Клас CustomUserDetailsService (рис. 3.10) має метод завантаження користувача за ідентифікатором та електронною поштою користувача та повертає об’єкт CustomUserDetails, який Spring Security може використовувати для аутентифікації та валідації.

```
@Service
public class CustomUserDetailsService implements UserDetailsService {

    @Autowired
    UserRepository userRepository;

    @Override
    @Transactional
    public UserDetails loadUserByUsername(String email) throws UsernameNotFoundException {
        User user = userRepository.findByEmail(email)
                .orElseThrow(() -> new UsernameNotFoundException("User not found with email : " + email));
        return CustomUserDetails.create(user);
    }

    @Transactional
    public UserDetails loadUserById(Long id) {
        User user = userRepository.findById(id)
                .orElseThrow(() -> new ApiException("User not found ", HttpStatus.NOT_FOUND));
        return CustomUserDetails.create(user);
    }
}
```

Рисунок 3.10 - Клас CustomUserDetailsService

CustomUserDetails містить необхідну інформацію (ідентифікатор користувача, ім'я, електронну пошту, пароль, роль) для створення об’єкта аутентифікації.

JwtUtil - утиліта для генерації токена (для звичайного користувача і адміністратора), перевірки аутентифікації (за допомогою UsernamePasswordAuthenticationToken), вилучення інформації про користувача з токена, а також валідації токена, його терміну дії.

Метод для створення токена зображений на рис. 3.11.

```
public String createToken(Authentication authentication, Boolean rememberMe) {
    String authorities = authentication.getAuthorities().stream()
            .map(GrantedAuthority::getAuthority)
            .collect(Collectors.joining(","));

    long now = (new Date()).getTime();
    Date validity;
    if (rememberMe) {
        validity = new Date(now + this.tokenValidityInMillisecondsForRememberMe);
    } else {
        validity = new Date(now + this.tokenValidityInMilliseconds);
    }
    CustomUserDetails customUserDetails = (CustomUserDetails) authentication.getPrincipal();

    return Jwts.builder()
            .setId(Long.toString(customUserDetails.getId()))
            .setSubject(customUserDetails.getEmail())
            .setIssuedAt(new Date())
            .claim(AUTHORITIES_KEY, authorities)
            .signWith(SignatureAlgorithm.HS512, secretKey)
            .setExpiration(validity)
            .compact();
}
```

Рисунок 3.11 - Реалізація методу createToken() у класі JwtUtil

Клас Http401ErrorEntryPoint імплементує AuthenticationEntryPoint, він перехоплює помилку аутентифікації.

Репозиторій виконує роботу з базою даних та імпортує дані в контролер. Контролер отримує й обробляє запит після того, як він був відфільтрований у JwtAuthFilter.

Конфігурація WebSecurityConfig (міститься в пакеті config) призначена для визначення безпеки Spring, кодування паролів і аутентифікації токена JWT. Також тут налаштовується cors, csrf, керування сесіями, правила для захищених ресурсів.

service – пакет сервісів, який реалізує бізнес-логіку програми окремо від контролерів. Такі класи використовують анотацію @Service і містять методи для роботи з моделями. Нижче наведено приклад сервісу PostService, в якому реалізовано створення нового поста (рис. 3.12).

```
@Service
@Transactional
public class PostService {

    @Autowired
    private PostRepository postRepository;

    public Optional<Post> findForId(Long id) { return postRepository.findById(id); }

    public PostDto registerPost(PostDto postDto, CustomUserDetails customUserDetails) {
        Post newPost = new Post();
        newPost.setTitle(postDto.getTitle());
        newPost.setBody(postDto.getBody());
        newPost.setCreatedBy(customUserDetails.getName());
        newPost.setCreatedDate(LocalDateTime.now());
        newPost.setUser(new User(customUserDetails.getId()));
        return new PostDto(postRepository.saveAndFlush(newPost));
    }
```

Рисунок 3.12 - Клас PostService

PostsApplication - клас, який запускає весь серверну частину, після запуску програма розгортається на вбудованому сервері за замовчуванням Tomcat. Пакет resources містить конфігуріційний файл application.yml.

Поширеною практикою у Spring Boot є використання зовнішньої конфігурації для визначення властивостей додатку. Це дозволяє використовувати той самий код програми в різних середовищах. Можна використовувати файли властивостей (properties), файли YAML, змінні середовища та аргументи командного рядка. За замовчуванням Spring Boot може отримати доступ до конфігурацій, встановлених у файлі application.properties, який використовує формат ключ-значення. Однак YAML є більш зручним форматом для визначення даних ієрархічної конфігурації.

У цьому файлі визначено параметри для JWT аутентифікації, такі як тривалість валідності токена, і вказаний секретний ключ, який використовується для алгоритму хешування (рис. 3.13). Секретний ключ поєднується із заголовком (header) і корисним навантаженням (payload) для створення унікального хешу. Перевірити цей хеш можна лише якщо є секретний ключ.

```yaml
security:
  jwt:
    secret: 5867a8374e7c0f6284b177b48faf89e1c79d72d8
    token-validity-in-seconds: 86400
    token-validity-in-seconds-for-remember-me: 2592000
```

Рисунок 3.13 - Налаштування для JWT у файлі application.yml

Також у файлі application.yml відбувається підключення до бази даних PostgreSQL і H2 (рис. 3.14).

```yaml
datasource:
  type: com.zaxxer.hikari.HikariDataSource
  url: jdbc:postgresql://localhost:5432/postsdb
  username: postgres
  password: [redacted]
  driver-class-name: org.postgresql.Driver
h2:
  console:
    enabled: true
jpa:
  database-platform: org.hibernate.dialect.H2Dialect
  database: H2
  show-sql: true
  format_sql: true
  properties:
    hibernate.id.new_generator_mappings: true
    hibernate.cache.use_second_level_cache: false
    hibernate.cache.use_query_cache: false
    hibernate.generate_statistics: true
    hibernate.dialect: org.hibernate.dialect.PostgreSQLDialect
    hibernate.jdbc.lob.non_contextual_creation: true
  hibernate:
    ddl-auto: update
  defer-datasource-initialization: true
```

Рисунок 3.14 - Налаштування підключення до бази даних у файлі application.yml

У секції datasource вказується пул підключень до бази даних, url, ім’я користувача, пароль і назву класу драйвера для PostgreSQL. Далі ми вмикаємо доступ до консолі бази даних H2.

У секції jpa визначається база даних для роботи. Параметр show-sql: true дозволяє виводити в консоль всі операції, які виконуються над базою даних. У properties вказуються додаткові властивості для JPA провайдера. Встановлюючи ddl-auto: update таблиці в базі даних будуть створюватись

автоматично на основі сутностей, які визначені в проекті, і кожен раз, коли буде оновлюватися поле в класі Entity, воно автоматично буде оновлюватися в базі даних.

3.2.2 Структура клієнтської частини

У пакеті frontend реалізована клієнтська частина проекту (рис. 3.15).

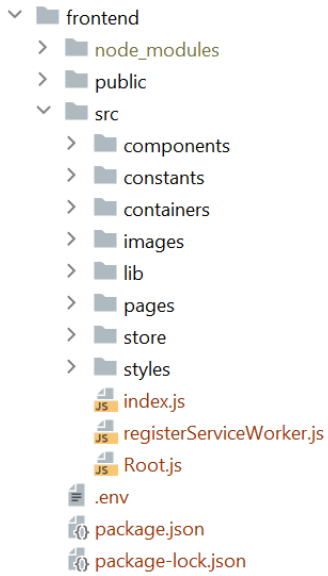

Рисунок 3.15 - Структура пакету frontend

При створенні React проекту автоматично були згенеровані пакети node_modules, public та src.

Пакет node_modules містить різні бібліотеки, які можуть бути використаними у проекті.

У пакеті public міститься файл index.html – шаблон сторінки, необхідний для коректної роботи програми.

Файл package.json – це файл керування версіями. Його основне призначення – зберігати список залежностей (бібліотек), необхідних проекту node.js для роботи. Він також включає іншу метаінформацію, у тому числі скрипти, дані про автора та ліцензію, опис та властивості проекту.

Пакет components (рис. 3.16) складається з набору презентаційних компонентів інтерфейсу користувача, таких як форми реєстрації та

аутентифікації, список постів, список коментарів, поля для створення нових постів і коментарів, панель управління адміністратора, а також загальні компоненти (верхній блок - header, нижній блок - footer, шаблон основного вмісту сторінки). Ці компоненти отримують дані через пропси (props) і просто відображають їх з використанням певних стилів, але не змінюють отримані дані.

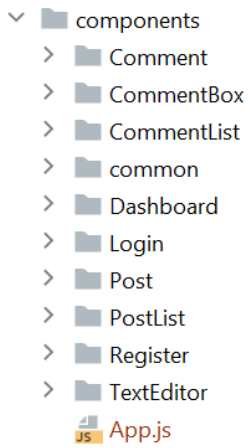

Рисунок 3.16 - Структура пакету components

У файлі App.js використовується React Router - декларативний фреймворк маршрутизації, який дозволяє налаштовувати маршрути за допомогою стандартних компонентів React. Switch огортає динамічні маршрути, а компонент Route налаштує маршрути та вказує компонент, який має відтворюватися (рис. 3.17).

```
render() {
  return (
    <div>
      <Route path="/login" component={LoginContainer} />
      <Route path="/register" component={RegisterContainer} />
      <Route path="/dashboard" component={DashboardContainer} />
      <Switch>
        <Route path="/pages/:page" component={PostListPage} />
        <Route path="/posts/:id" component={PostPage} />
        <Route path="/editor/:id?" component={EditorPage} />
        <Route path="/" component={PostListPage} />
        <Route component={NotFoundPage} />
      </Switch>
    </div>
  );
}
};
```

Рисунок 3.17 - Налаштування шляхів у файлі App.js

Для створення деяких компонентів (наприклад Post) використовується елемент <Fragment> з бібліотеки react, який дозволяє згрупувати декілька компонентів в один (рис. 3.18).

```
return (
  <Fragment>
    <div className={cx("post-wrapper")}>
      <div className={cx("post-header")}>
        <h2 className={cx("post-title")}>
          <Link to={"/posts/" + post.get("id")}>{post.get("title")}</Link>
          {...}
        </h2>
          {...}
        <hr />
      </div>
      <div className={cx('post-body')}>{renderHTML(post.get("body"))}</div>
    </div>
    <hr></hr>
    <CommentList
      isAuthenticated={isAuthenticated}
      loading={false}
      comments={comments}
      writeComment={writeComment}
      postId={post.get("id")}>
    </CommentList>
  </Fragment>
)
};
```

Рисунок 3.18 - Приклад використання елементу Fragment

Певні компоненти визначені як класи які розширюють Component.

Клас LoginModal містить конструктор, в якому через метод super(props) компонент отримує доступ до даних, які використовуються в інших компонентах, для коректної аутентифікації користувача. Функція handleSubmit отримає дані форми (логін, пароль і параметр rememberMe, введені користувачем), якщо перевірка форми пройшла успішно (рис. 3.19).

```
constructor(props) {
  super(props);

  this.state = {
    loginError: false
  };
}

handleSubmit = (event, errors, { username, password }) => {
  const { handleLogin } = this.props;
  handleLogin(username, password);
};
```

Рисунок 3.19 - Обробка форми для аутентифікації користувача

Пакет constants - це пакет для файлів, які містять константи, необхідні для клієнтської частини проекту.

У пакеті containers подібно до пакету components знаходяться компоненти, однак їх основною функцією є передача даних та їх поведінки презентаційним компонентам, які вони містять.

Наприклад, у класі LoginContainer викликається метод AuthActions.login з файлу store/modules/auth.js для подальшої обробки отриманих даних про користувача (рис. 3.20).

```
handleLogin = async (username, password, rememberMe = false) => {
  const { AuthActions } = this.props;
  try {
    await AuthActions.login(username, password);
  } catch (e) {
    console.log("error log :" + e);
  }
};
```

Рисунок 3.20 - Реалізація функції handleLogin у компоненті LoginContainer

images – містить зображення, які використовуються у застосунку.

Пакет lib містить фасади для різних бібліотек, які застосовуються у проекті. Тут визначений файл api.js для бібліотеки axios, який створює новий API поверх API axios, який потім використовується у програмі.

Наприклад, для доступу до певного посту використовуться url:

export const getPost = (postId) => axios.get(`${API_BASE_URL}/api/posts/${postId}`);

де API_BASE_URL визначена як константа у файлі constants/index.js.

Файл storage.js створений для керування вебсховищами, що знаходяться у браузері користувача та призначені для зберігання даних.

Файли в папці pages вказують маршрути програми. Кожен файл у цій папці містить свій маршрут і стан, вони використовують шаблон сторінки PageTemplate, заданий у файлі components/common/PageTemplate/ PageTemplate.js. На рис. 3.21 продемонстровано PostPage (сторінку посту).

```
const PostPage = ({match}) => {
  const { id } = match.params;
  console.log('post page')
  return (
    <PageTemplate header={<HeaderContainer/>}>
      <PostContainer id={id}/>
    </PageTemplate>
  );
};

export default PostPage;
```

Рисунок 3.21 - Реалізація сторінки PostPage

Пакет store містить проміжне програмне забезпечення (middleware), яке виконує обробку запитів. У файлі config.js визначені основні конфігурації, необхідні для використання бібліотеки redux-pender. Вона допомагає слідкувати за станами програми та керувати асинхронними діями. Пакет modules об'єднує модулі, які описують обробку дій відповідних об'єктів.

Наприклад, для отримання списку постів у контейнері PostListContainer.js було попередньо визначено стан для обробки різних результатів дії GET_POST_LIST (рис. 3.22).

```
export default connect(
  state => ({
    posts: state.post.get("posts"),
    loading: state.pender.pending["post/GET_POST_LIST"],
    error: state.pender.failure["post/GET_POST_LIST"],
    success: state.pender.success["post/GET_POST_LIST"],
    isAuthenticated: state.auth.get("isAuthenticated")
  }),
  dispatch => ({
    PostActions: bindActionCreators(postActions, dispatch)
  })
)(PostContainer);
```

Рисунок 3.22 - Реалізація з'єднання даних з пакету store з компонентом PostListContainer за допомогою методу connect() бібліотеки redux

Далі створюється відповідна дія у файлі modules/post.js за допомогою метода бібліотеки redux-actions createAction():

const GET_POST_LIST = 'post/GET_POST_LIST';

export const getPostList = createAction(GET_POST_LIST, api.getPosts);

У методі handleActions вказується різна поведінка програми залежно від результату дії (рис. 3.23).

```
  ...pender({
    type: GET_POST_LIST,
    onSuccess: (state, action) => {
      const { data: content } = action.payload;
      console.log("GET_POST_LIST onSuccess")
      return state.set('posts', fromJS(content))
    },
    onFailure: (state, action) => {
      console.log("GET_POST_LIST onFailure")
      return state;
    },
    onPending: (state, action) => {
      console.log("GET_POST_LIST onPending")
      return state;
    }
  }),
```

Рисунок 3.23 - Обробка запиту для отримання списку постів у методі handleActions() бібліотеки redux-pender

Пакет styles складається з файлів з роширенням scss, які задають стилі компонентів. SCSS (Sassy Cascading Style Sheets) — це препроцесори CSS, які додають спеціальні функції, такі як змінні, міксіни (mixins), селектори вкладення та успадкування. Вони запобігають дублюванню викликаних властивостей з тим самим значенням.

Файл base.scss задає основні стилі, які використовуються у всьому проекті. Також кожен компонент у папці components має свій файл з розширенням scss.

3.3 Результати тестування програми

На рис. 3.24 зображена головна сторінка, яку бачить користувач при запуску програми. Це список всіх постів, які були опубліковані.

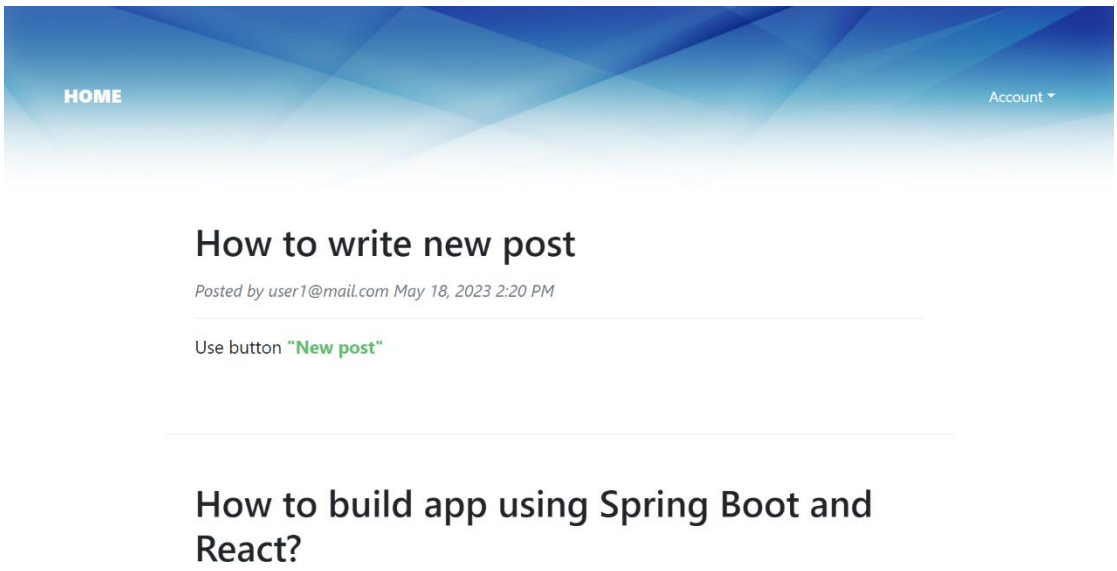

Рисунок 3.24 - Головна сторінка застосунку

Зверху є меню навігації. Пункт «Home» дозволяє переходити на головну сторінку.

«Account» – це випадаючий список кнопок «Sign in» та «Sign up», які дозволяють увійти в існуючий акаунт або зареєструватись у системі (рис. 3.25).

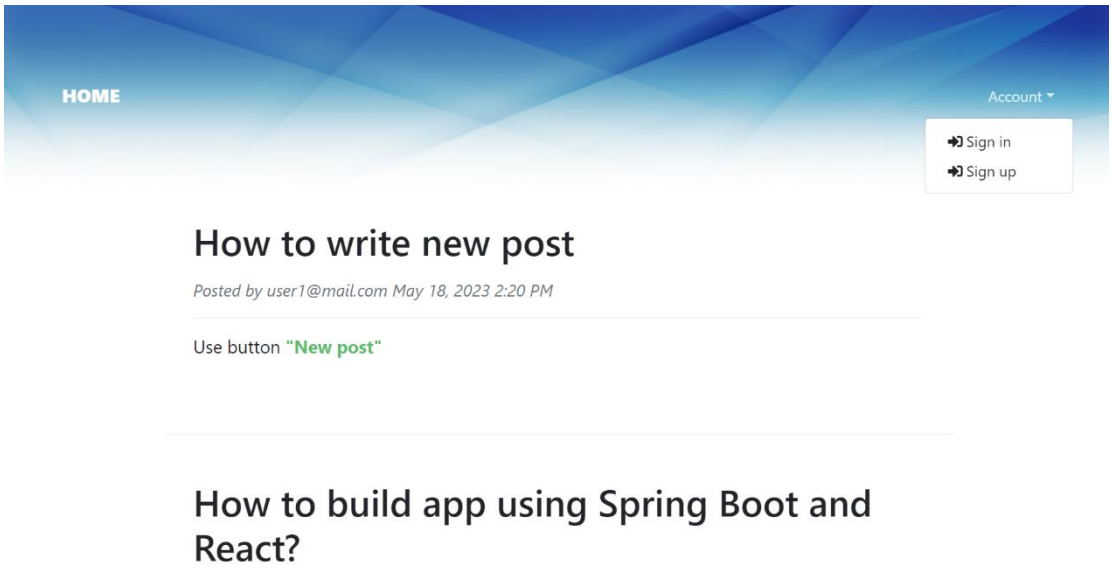

Рисунок 3.25 - Кнопки «Sign in» та «Sign up»,

Коли користувач натисне «Sign up», з'явиться вікно з формою для реєстрації, в якій потрібно обов'язково заповнити всі поля, а саме: ім'я, електронна пошта, пароль (рис. 3.26).

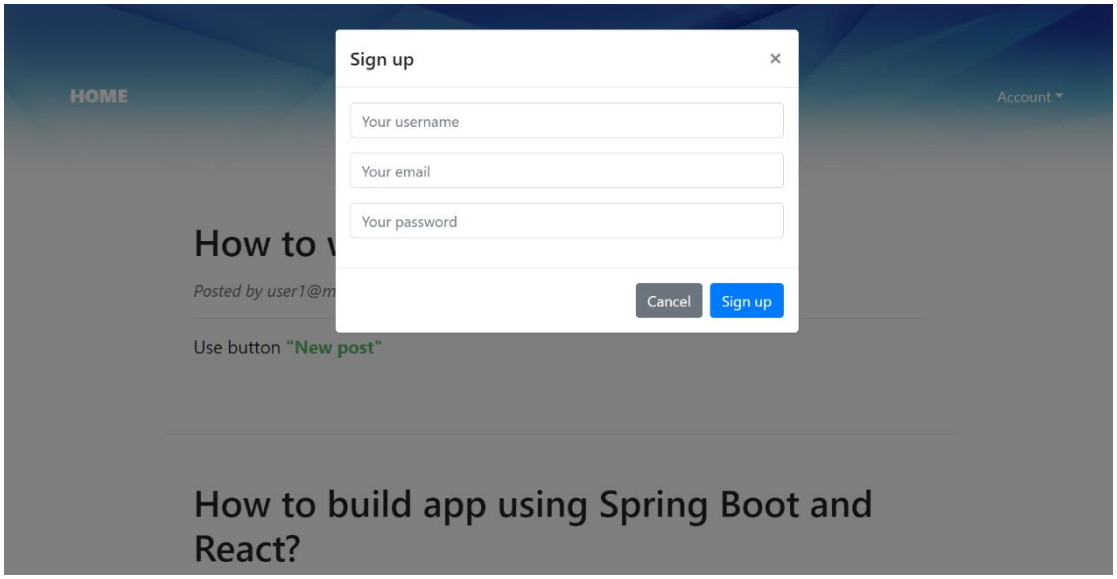

Рисунок 3.26 - Форма для реєстрації користувача

Якщо поля залишити порожніми, з'являться повідомлення про помилку (рис. 3.27).

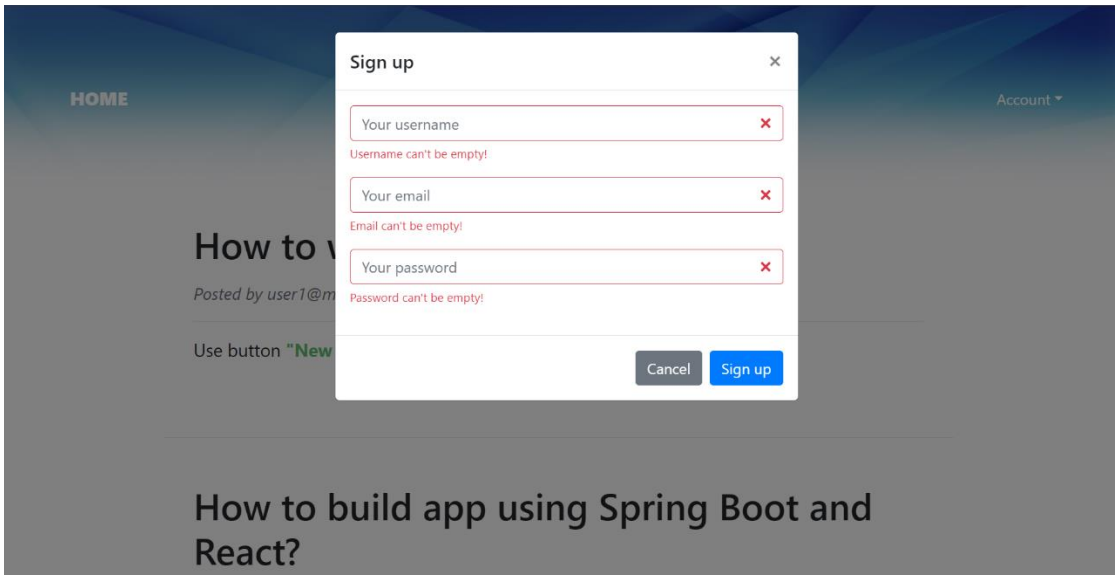

Рисунок 3.27 - Повідомлення про порожні поля

Якщо ввести дані вже існуючого користувача, з’явиться повідомлення про помилку (рис. 3.28).

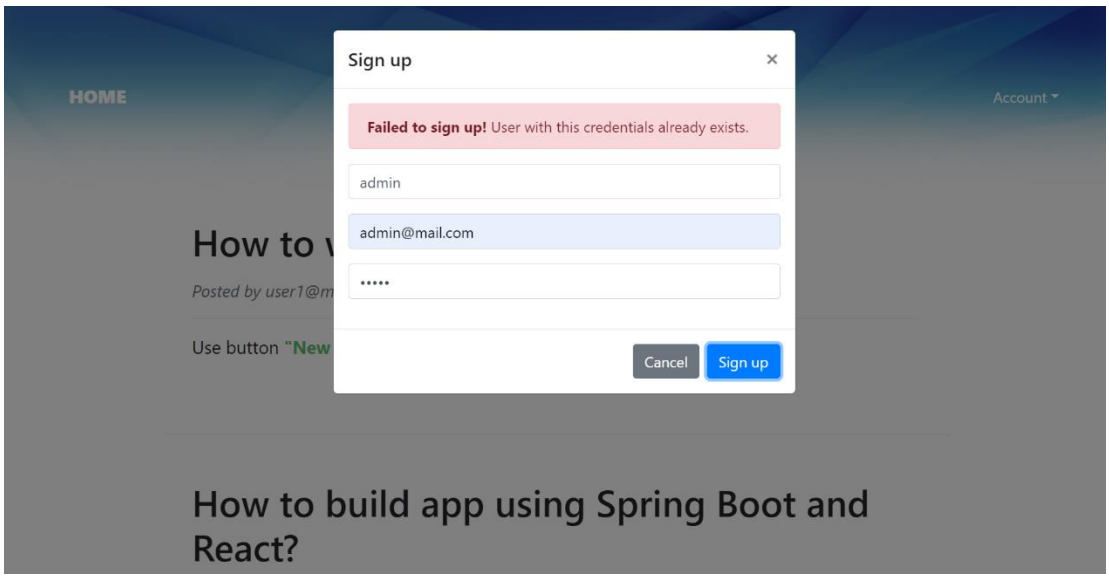

Рисунок 3.28 - Помилка при спробі ввести дані зареєстрованого акаунту

Схоже вікно з формою відкриється після натиску кнопки «Sign in» (рис. 3.29).

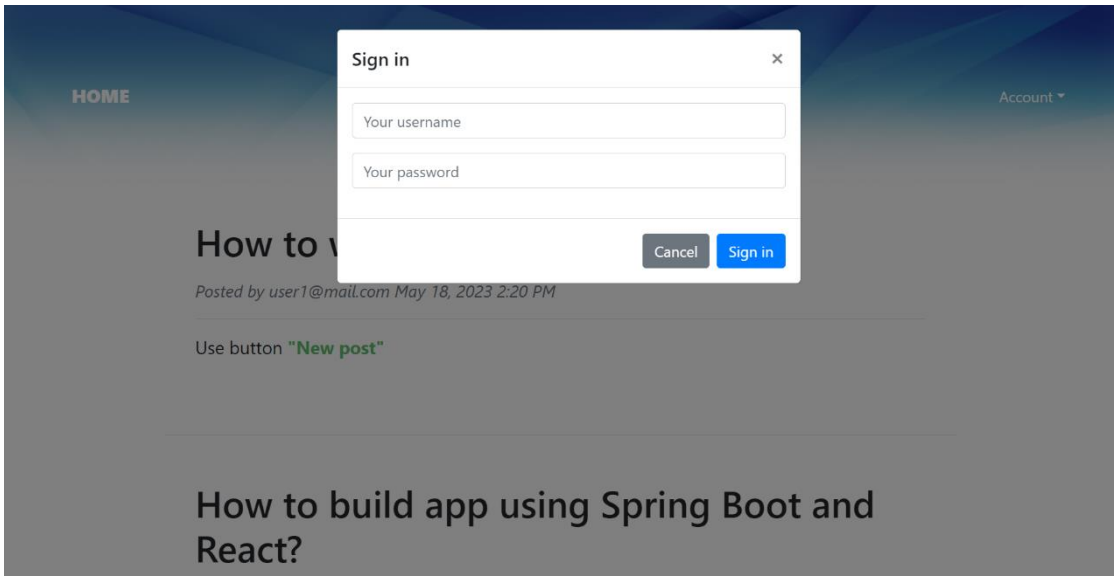

Рисунок 3.29 - Форма для входу користувача в систему

Натиснувши на назву поста або на кнопку «Read more», відкриється сторінка даного поста, де можна переглянути весь його зміст (рис. 3.30), а також коментарі користувачів до нього (рис. 3.31).

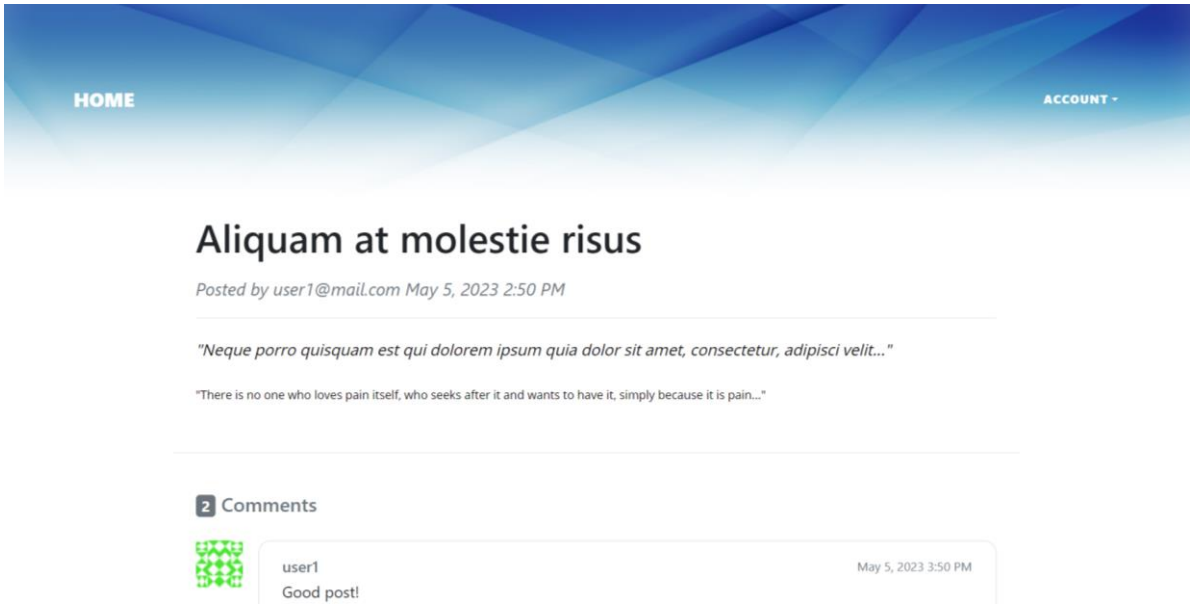

Рисунок 3.30 - Сторінка поста

Рисунок 3.31 - Список коментарів поста

Коли користувач у системі авторизується в системі, він зможе написати власний коментар (рис. 3.32).

Рисунок 3.32 - Створення нового коментаря

Також авторизований користувач може створити новий пост за допомогою кнопки кнопка «New post», яка з’являється на головній сторінці (рис. 3.33).

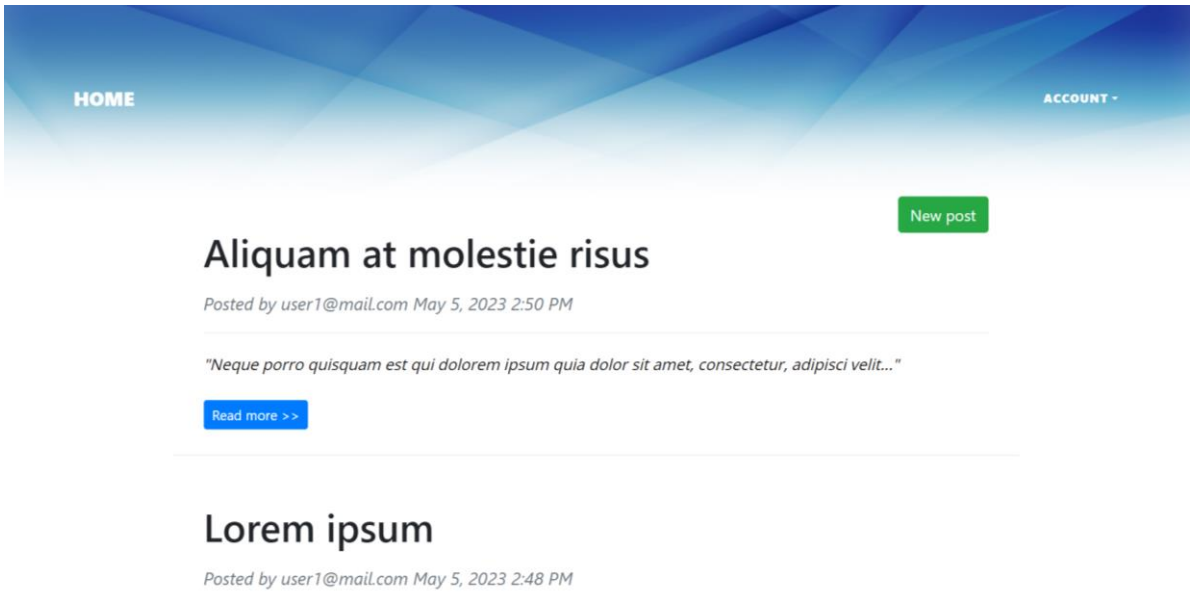

Рисунок 3.33 - Кнопка «New post»

Форма створення нового поста зображена на рис. 3.34.

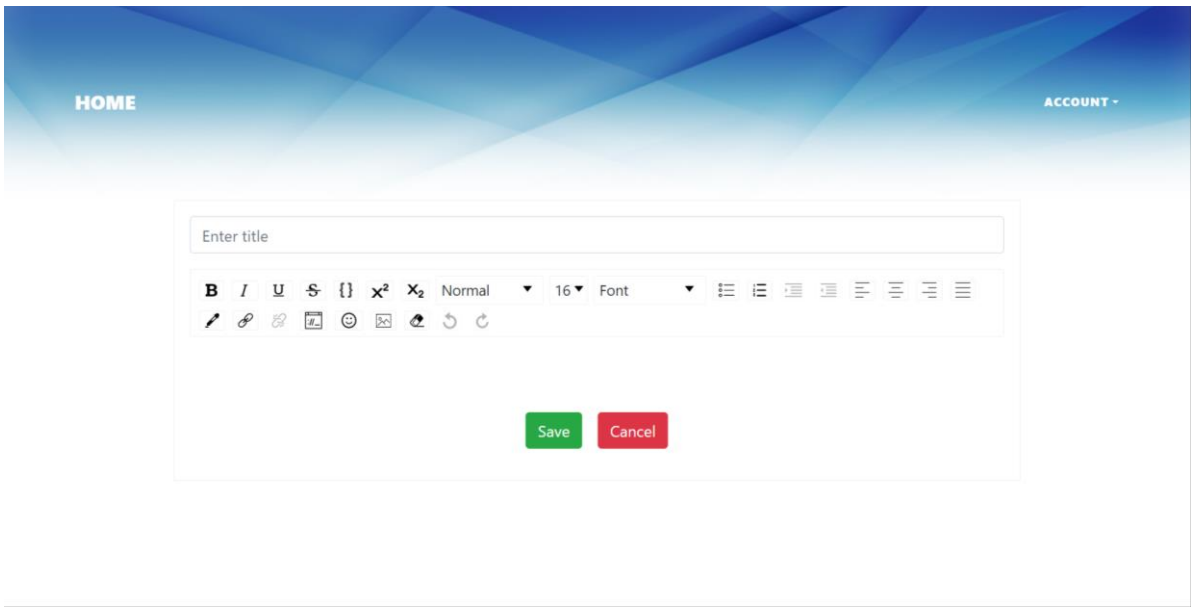

Рисунок 3.34 - Сторінка створення нового поста

Обов’язковою для вводу є лише назва поста, вміст можна лишити порожнім.

Користувач може редагувати або видаляти лише свої пости (рис. 3.35).

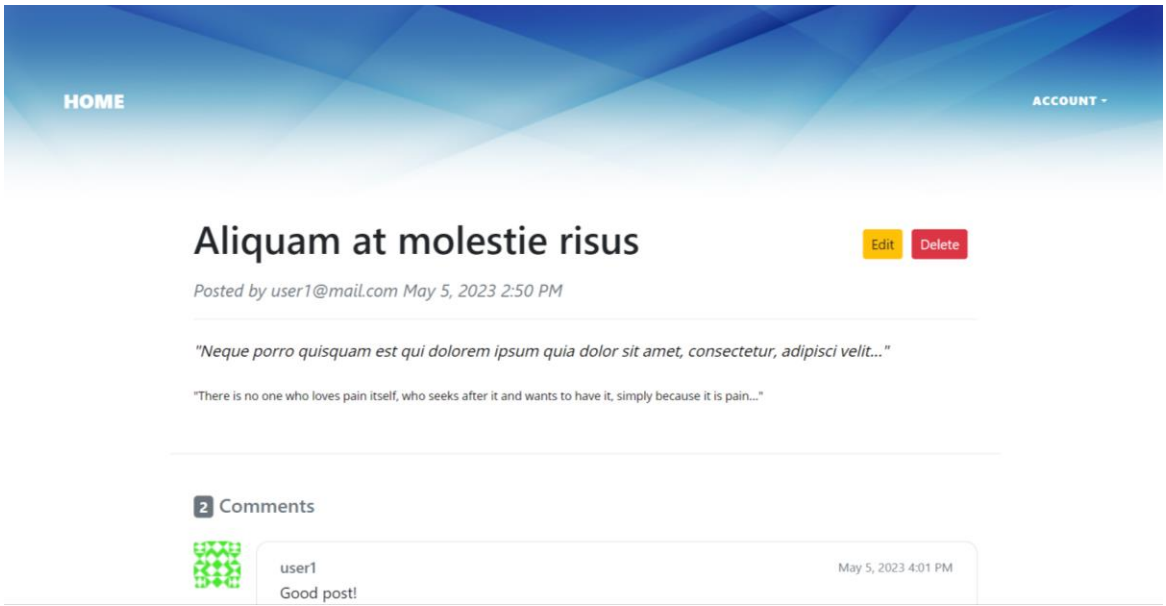

Рисунок 3.35 - Кнопки «Edit» та «Delete»

Якщо у системі авторизувався користувач з роллю Admin, у меню з’явиться пункт «Dashboard». На сторінці Dashboard адміністратор може переглядати список всіх користувачів, зареєстрованих у системі (рис. 3.36).

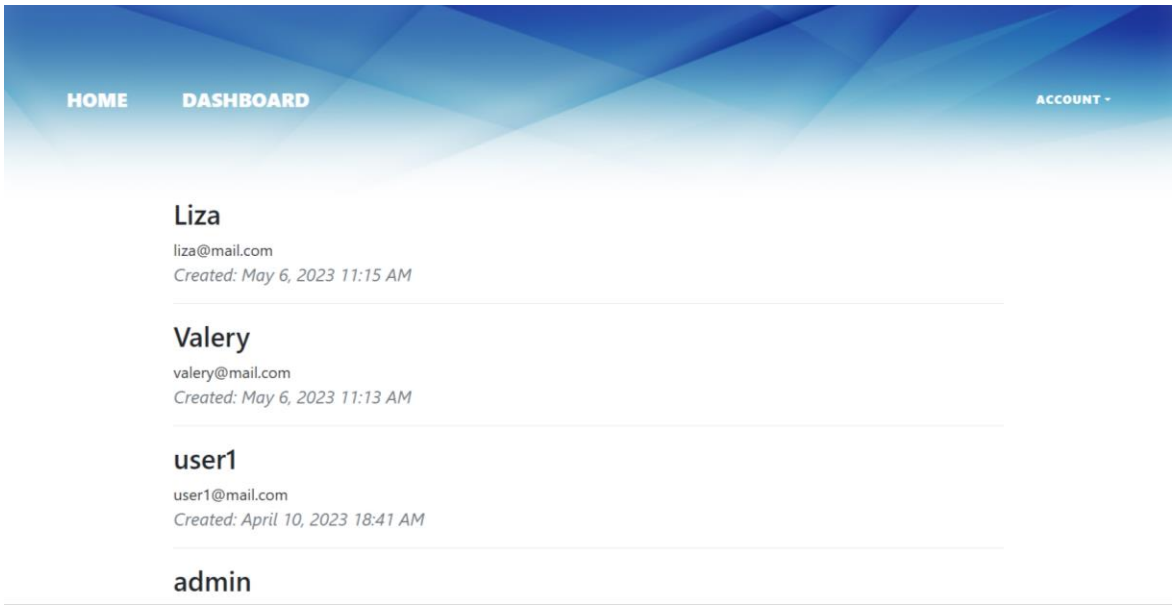

Рисунок 3.36 - Панель управління для адміністратора зі списком користувачів

Для виходу із програми потрібно натиснути кнопку «Sign Out», яка з’явиться у пункті меню «Account» одразу після авторизації користувача.

3.4 Рекомендації щодо використання Spring Boot і React

Для створення Spring Boot проекту варто скористатися Spring Initializer, який дозволяє вказати початкові налаштування та одразу додати необхідні залежності. Spring Initializer гарантує, що всі залежності є правильними та перевіреними.

Важливою частиною будь-якої програми є чітка та зрозуміла структура. Всі модульні частини повинні бути відокремлені папками, наприклад: контролери, сервіси, конфігурації і т.д. Головне - дотримуватися одного стилю. Це забезпечить легшу навігацію для розробників. Також важливо зберігати клас запуску (вхідну точку програми) в основному каталозі верхнього рівня (зазвичай src).

Спростити код та зробити його більш функціональним можна завдяки використанню автоконфігурації, яка є однією з особливостей Spring Boot. Автоконфігурація активується, коли певний файл jar виявляється на шляху до класів. Найпростіший спосіб його використання - це Spring Boot Starters. За допомогою цих стартерів конфігурації легко інтегровані та коректно працюють разом, оскільки всі вони перевірені

Певні класи конфігурації можна виключити з автоконфігурації за допомогою таких атрибутів анотації:

@EnableAutoConfiguration(exclude = {ClassNotToAutoconfigure.class})

При розробці бекенду на Spring Boot слід розмежовувати логіку контролерів від бізнес-логіки. Контролери повинні обробляти HTTP-рівень програми, це не варто перекладати на сервіси. Крім того контролери мають бути stateless, оскільки за замовчуванням вони є синглтонами, і будь-який стан може спричинити помилки.

Один із способів захистити бізнес-логіку від коду Spring Boot — це використання ін’єкції конструктора. Також вона запобігає створенню циклічних залежностей між компонентами (beans).

Spring Data JPA спрощує реалізацію пагінації та сортування, підвищує продуктивність, особливо у разі отримання великого набору даних. Для цбого репозиторії повинні розширювати JpaRepository, щоб мати змогу передавати Pageable (інтерфейс для PageRequest, де можна вказати кількість і номер сторінки, а також атрибут сортування) і повертати об’єкт Page. У контролері повертається Page як тіло в ResponseEntity, і передається Pageable як додатковий вхідний параметр.

Зі сторони клієнта викликається кінцева точка (endpoint) з пагінацією та сортуванням за допомогою параметрів запиту.

Поширеною практикою є використання бібліотеки Lombok для уникнення повторів коду при створенні моделей. Наприклад: анотації @Data, @NoArgsConstructor, @AllArgsConstructor дозволять позбутися геттерів/сеттерів і конструкторів у сутностях і об’єктах значень, а також шаблонного коду Java, такого як: toString(), equals() і hashCode() [14].

Також бажано забезпечити послідовний спосіб обробки винятків. Spring Boot пропонує два основні підходи:

- використання HandlerExceptionResolver для глобальної стратегії обробки винятків;
- використання анотації @ExceptionHandler в контролері в деяких конкретних сценаріях.

У клієнтській частині наявність великих компонентів, які виконують декілька функцій, ускладнює підтримку цих компонентів і заважає використанню переваг оптимізації React. Варто створювати невеликі компоненти, які візуалізують певні частини сторінки або додають нову поведінку до інтерфейсу користувача. Це підвищить продуктивність. Крім того, компоненти стануть більш придатними для повторного використання у програмі.

Використання бібліотеки Redux дозволить додатку масштабуватися, надаючи розумний спосіб керування станом за допомогою моделі

односпрямованого потоку даних. Також Redux забезпечить коректне прив’язування інтерфейсу користувача для проекту React. Він постійно оновлюється з будь-якими змінами API, щоб забезпечити належну роботу компонентів. Крім того, бібліотека сприяє внутрішній оптимізації продуктивності, що дозволяє повторно відтворювати компоненти лише тоді, коли це дійсно потрібно, і підтримує правильну архітектуру застосунку.

Для визначення декількох url-адрес програми застосовується бібліотека React Router. Зазвичай вона використовується для розробки односторінкових вебдодатків. Router надає синхронну url-адресу в браузері з даними, які відображатимуться на вебсторінці. Завдяки бібліотеці можна переходити між сторінками додатку, не перезавантажуючи сторінку.

У компонентах програми React метод render відтворює лише один кореневий вузол всередині нього за раз. Проте у разі потреби додати кілька елементів до компонента рекомендується використати синтаксис Fragment, замість тегу div. Оскільки div створить додатковий вузол DOM, це іноді призводить до неправильного форматування HTML-виводу, наприклад під час роботи з CSS Flexbox і Grid. Таким чином використання Fragment не призводить до порушенням макету сторінки, займає менше пам'яті і пришвидшує виконання коду.

ВИСНОВКИ

У ході виконання кваліфікаційної роботи було визначено основні функціональні можливості фреймворку Spring Boot і бібліотеки React. На основі розглянутих теоретичних відомостей виконано порівняльний аналіз Spring Boot та React з їх аналогами: Quarkus, Micronaut, Vert.x, Helidon і Vue, Angular, Ember відповідно.

Було описано технічні засоби розробки програми, такі як інструмент для автоматичної побудови проектів Maven, бази даних PostgreSQL та H2, фреймворк Spring Security, платформа Node.js, менеджер пакетів npm, збірник модулів Webpack. Вони сприяли ефективному вирішенню всіх необхідних функціональних вимог.

Для аналізу практичних аспектів розробки бекенду на Spring Boot і фронтенду на React розглянута клієнт-серверна архітектура. У результаті був розроблений вебзастосунок, який підтримує можливість аутентифікації, має розподіл ролей на адміністратора та звичайного користувача, виконує базові CRUD-операції.

Насамкінець визначено майбутні перспективи та розроблено рекомендації щодо практичного застосування Spring Boot та React при розробці full-stack додатку.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. What is Java Spring Boot? [Електронний ресурс]. Режим доступу:

https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-java-spring-boot/

2. React [Електронний ресурс]. Режим доступу:

https://yarnpkg.com/package/react-is

3. JSX [Електронний ресурс]. Режим доступу:

https://uk.legacy.reactjs.org/docs/introducing-jsx.html

4. Vert.x [Електронний ресурс]. Режим доступу:

https://vertx.io/docs/vertx-core/java/

5. Helidon [Електронний ресурс]. Режим доступу:

https://www.oracle.com/a/ocom/docs/technical-brief--helidon-report.pdf

6. VueJS [Електронний ресурс]. Режим доступу:

https://vuejs.org/guide/essentials/reactivity-fundamentals.html

7. Maven [Електронний ресурс]. Режим доступу:

https://maven.apache.org/what-is-maven.html

8. Postgresql [Електронний ресурс]. Режим доступу:

https://www.postgresql.org/about/

9. Spring Security [Електронний ресурс]. Режим доступу:

https://docs.spring.io/spring-security/reference/index.html

10. JWT [Електронний ресурс]. Режим доступу:

https://jwt.io/introduction

11. NodeJS [Електронний ресурс]. Режим доступу:

https://nodejs.dev/en/learn/introduction-to-nodejs/

12. Npm [Електронний ресурс]. Режим доступу:

https://docs.npmjs.com/about-npm

13. Webpack [Електронний ресурс]. Режим доступу:

https://webpack.js.org/concepts/

14. Project Lombok [Електронний ресурс]. Режим доступу:

https://projectlombok.org/features/Data