МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ПРОЄКТУВАННЯ ТА РОЗРОБКА ВЕБЗАСТОСУНКУ З ВИКОРИСТАННЯМ ФРЕЙМВОРКУ RUBY ON RAILS»**

Виконав: студент 4 курсу, групи 6.1210-2пі
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
С.О. Владарський
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, PhD Столярова А.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент доцент кафедри комп'ютерних наук, доцент, к.т.н. Решевська К.С.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2024

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет математичний

Кафедра програмної інженерії

Рівень вищої освіти бакалавр

Спеціальність 121 інженерія програмного забезпечення
(шифр і назва)

Освітня програма програмна інженерія

ЗАТВЕРДЖУЮ
Завідувач кафедри програмної інженерії, к.ф.-м.н., доцент

_____ Лісняк А.О.
(підпис)

“ ____ ” _____ 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Владарському Сергію Олександровичу
(прізвище, ім’я та по-батькові)

1. Тема роботи Проєктування та розробка вебзастосунку з використанням фреймворку Ruby on Rails

керівник роботи Столярова Анастасія Валеріївна, PhD
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 21 » грудня 2023 року № 2180-с

2. Строк подання студентом роботи 03.06.2024 р.

3. Вихідні дані до роботи 1. Постановка задачі.
2. Перелік літератури.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1. Постановка задачі.
2. Основні теоретичні відомості.
3. Проєктування та розробка вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень)
презентація за темою доповіді

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання 25.12.2023 р.

К А Л Е Н Д А Р Н И Й П Л А Н

№	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1.	Розробка плану роботи.	09.01.2024	
2.	Збір вихідних даних.	23.01.2024	
3.	Обробка методичних та теоретичних	20.02.2024	
	джерел.		
4.	Розробка першого та другого розділу.	09.04.2024	
5.	Розробка третього розділу.	20.05.2024	
6.	Оформлення та нормоконтроль	27.05.2024	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	18.06.2024	

Студент _____ С.О. Владарський
(підпис) (ініціали та прізвище)

Керівник роботи _____ А.В. Столярова
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

Нормоконтролер _____ А.В. Столярова
(підпис) (ініціали та прізвище)

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Проєктування та розробка вебзастосунку з використанням фреймворку Ruby on Rails»: 54 с., 35 рис., 2 табл., 19 джерел.

ПРОЄКТУВАННЯ, РОЗРОБКА, ACTIONCABLE, DOCKER, MVP, POSTGRESQL, REDIS, RSPEC, RUBY, RUBY ON RAILS, TDD, WEBSOCKET.

Об’єкт дослідження: процес розробки вебзастосунків.

Предмет дослідження: використання фреймворку Ruby on Rails при розробці соціальних мереж.

Мета роботи: дослідити ефективність використання мови програмування Ruby та фреймворку Ruby on Rails для розробки безпечних, швидких та підтримуваних вебсайтів та розробити вебзастосунок з використанням вказаних інструментів.

Методи дослідження: збір, аналіз вимог, проєктування та конструювання програмного забезпечення.

Результатом проєкту є розроблений вебзастосунок, який є безпечною, оптимізованою та підтримуваною соціальною мережею.

Дослідження Ruby on Rails показало, що фреймворк ідеально підходить для швидкої розробки невеликих MVP проєктів. Для покращення проєкту можливе розширення функціоналу та використання таких технологій як React, AWS S3, ActionCable, Sidekiq background jobs.

Отриманий вебзастосунок може бути використано для подальшого дослідження розробки соціальних мереж та використання фреймворку Ruby on Rails.

SUMMARY

Bachelor’s qualifying paper “Design and Development of a Web Application Using the Ruby on Rails Framework”: 54 pages, 35 figures, 2 tables, 19 references.

ACTIONCABLE, DESIGN, DEVELOPMENT, DOCKER, MVP, POSTGRESQL, REDIS, RSPEC, RUBY, RUBY ON RAILS, TDD, WEBSOCKET.

The object of the study is web application development process.

The subject of the study is the use of the Ruby on Rails framework in the development of social networks.

The purpose of the study is to investigate the effectiveness of using the Ruby programming language and the Ruby on Rails framework for the development of secure, fast and maintainable websites and to develop web application using the specified tools.

The methods of research are collection, analysis of requirements, design and development of software.

The result of the project is a developed web application that is a safe, optimized and supported social network.

Research on Ruby on Rails has shown that the framework is ideal for fast development of small MVP projects. To improve the project, it is possible to expand the functionality and use such technologies as React, AWS S3, ActionCable, Sidekiq background jobs.

The resulting web application can be used for further research into the development of social networks and the use of the Ruby on Rails framework.

ЗМІСТ

ВСТУП

Веброзробка – одна з найрозвиненіших ІТ сфер у сучасному світі. З кожним роком та появою нових технологій комплексність вебзастосунків підвищується. Одними з основних критерій при підборі технологій для розробки є: можливість реалізації визначеного функціоналу, сучасність, складність підтримки та ціна за розробку продукту на даній технології.

Фреймворк Ruby on Rails з'явився у 2004 році та швидко почав набирати популярність серед бекенд розробників. З тих пір він тільки розвивався, надаючи все нові покращення та оптимізації для фреймворку.

Основою проєкту для дослідження можливостей Ruby on Rails було взято розробку соціальної мережі. Користувачі завжди хочуть бути у контакту один з одним, тому соціальні мережі завжди були та будуть популярним продуктом. Комплексність цього проєкту надає можливість в повній мірі виконати всі стадії розробки продукту на фреймворку Ruby on Rails.

Мета роботи: дослідити ефективність використання мови програмування Ruby та фреймворку Ruby on Rails для розробки безпечних, швидких та підтримуваних вебсайтів та розробити вебзастосунок з використанням вказаних інструментів.

Для досягнення поставленої мети необхідно вирішити наступні завдання:

- проаналізувати використання різних інструментів та технологій для розробки вебзастосунків;
- розробити архітектуру застосунку з використанням Ruby on Rails, враховуючи структуру бази даних, моделі, контролери та вигляди;
- розробити схему бази даних та діаграму прецедентів;
- реалізувати основні функціональні можливості вебзастосунку, включаючи автентифікацію, авторизацію, CRUD-операції та інтеграцію з зовнішніми API, якщо необхідно;

– протестувати вебзастосунок, використовуючи юніт-тести для перевірки коректності роботи окремих компонентів вебзастосунку.

Структурно кваліфікаційна робота складається зі вступу, трьох розділів, висновків, переліку посилань. Загальний обсяг роботи становить 54 сторінки. У переліку посилань 19 використаних джерел.

В першому розділі аналізуються обрані технології, наводяться переваги та недоліки кожної з них, робляться висновки щодо доцільності використання кожної окремої технології для розробки застосунків в межах певної предметної області.

В другому розділі висвітлено процес створення архітектури. Для обраної бази даних розробляється даіграма прецедентів та ER-діагарма.

Третій розділ присвячено безпосередньо процесу розроби та тестування вебдодатку. Наведено приклади коректної роботи програми, основні приклади коду, які демонструють ефективність імплементації певного функціоналу.

1 ТЕОРЕТИЧНІ ВІДОМОСТІ ТА АНАЛІЗ ОБРАНИХ ТЕХНОЛОГІЙ

1.1 Технічне завдання

1.1.1 Опис предметної області

Додаток «Соціальна мережа» надає можливість користувачам переглядати та публікувати різноманітний контент, взаємодіяти з іншими користувачами та вести особисті сторінки.

Основні аспекти предметної області включають:

– користувачі: реєстрація, авторизація та профіль користувача, включаючи особисті дані, фотографії;
– пости: публікація текстових повідомлень, фотографій, можливість зміни та видалення;
– взаємодія: коментарі, лайки, підписка на інші профілі, обмін повідомленнями та комунікація в чатах;
– перегляд контенту: можливість переглядати та взаємодіяти з профілями інших користувачів через стрічку новин;
– панель адміністратора: управління користувачами та контентом.

Таким чином, соціальна мережа надає зручний та функціональний інструмент для спілкування, обміну інформацією та взаємодії для своїх користувачів.

1.1.2 Аналіз ринку

Аналіз ринку соціальних мереж є критично важливим для розуміння сучасних тенденцій, конкурентного середовища та визначення можливостей для нової платформи. Розробникам нових платформ необхідно ретельно

аналізувати ринок, враховуючи технічні вимоги, користувацькі очікування та інноваційні функції, щоб успішно конкурувати і задовольнити потреби цільової аудиторії.

Facebook – це одна з найбільших і найпопулярніших соціальних мереж у світі. Платформа дозволяє користувачам створювати профілі, додавати друзів, обмінюватися повідомленнями, публікувати фото та відео, створювати та приєднуватися до груп, а також брати участь у подіях. Facebook має рекламні можливості для бізнесів та розвинену систему персоналізації контенту, що дозволяє користувачам отримувати інформацію, яка найбільше відповідає їхнім інтересам.

Instagram – це провідна платформа для обміну фотографіями та відео. Instagram пропонує користувачам можливість публікувати зображення та короткі відео, використовувати фільтри, створювати історії, а також вести прямі трансляції. Instagram є потужним інструментом для бізнесу, пропонуючи можливість маркетингу, піару та онлайн шопінгу.

Twitter – це платформа для мікроблогів, яка дозволяє користувачам публікувати короткі повідомлення. Користувачі можуть стежити за іншими, коментувати та ретвітити повідомлення, використовуючи хештеги для категоризації контенту. Платформа також відома своєю роллю у громадських дискусіях та політичних дебатах. Після покупки Twitter бізнесменом Ілоном Маском, правила платформи були полегшені, через що люди можуть висловлювати будь-яку контроверсійну думку, не отримуючи ніяких обмежень.

LinkedIn – це найбільша професійна соціальною мережа у світі. Платформа орієнтована на професіоналів та бізнеси, надаючи можливість створювати професійні профілі, встановлювати зв’язки, публікувати статті, вакансії, резюме, шукати роботу, групи однодумців та знаходити потенційних співробітників. У багатьох сучасних сферах, особливо у сфері інформаційних технологій, Linkedin вважається найважливішим інструментом для пошуку роботи або кандидатів.

Mastodon – це відкрита децентралізована соціальна мережа, написана на Ruby on Rails та React [14]. Mastodon працює на основі інстансів, що дає користувачам можливість вибору сервера, на якому вони хочуть створити свій обліковий запис. Кожен інстанс має свої правила та адміністраторів, що створює різноманітні спільноти з різними інтересами та стандартами поведінки. У Mastodon користувачі можуть публікувати «статуси» (пости), взаємодіяти з іншими користувачами, створювати списки, відповіді, додавати статуси у улюблені, закладки, використовувати хештеги та обмінюватися приватними повідомленнями. Децентралізована природа Mastodon надає більшу свободу вибору та контролю над власними даними користувачам, а також створює сприятливі умови для розвитку різноманітних та унікальних спільнот. Mastodon – це продукт з відкритим вихідним кодом, що дає можливість суспільності вносити зміни до коду або використовувати код Mastodon у своїх проєктах.

1.1.3 Вибір стеку

При обговоренні різних стеків технологій для розробки соціальних мереж, варто враховувати різні аспекти, включаючи продуктивність розробки, доступність готових рішень, швидкість виконання, масштабованість та рівень підтримки спільноти. Далі буде наведено порівняння популярних фреймворків (див. табл. 1.1).

Ruby та Ruby on Rails, завдяки своїй зручній синтаксису та великій кількості готових рішень, відомі своєю високою продуктивністю розробки. Пакети та геми для Rails дозволяють швидко реалізувати різноманітні функціональності, що дозволяє швидко прототипувати та розвивати вебдодатки.

Python та Django також є популярними виборами для розробки соціальних мереж. Django має розвинену екосистему, але меншу, ніж у Rails,

проте забезпечує добру продуктивність та масштабованість. З іншого боку, JavaScript та Node.js забезпечують високу швидкість виконання завдяки використанню асинхронного програмування, але їхня екосистема може бути менш розвиненою для розробки вебдодатків зі складною бізнес-логікою.

З Laravel, PHP-фреймворком, також можна досягти великої продуктивності та швидкості розробки, особливо для команд, які знайомі з мовою PHP. Laravel має багату екосистему та добре підтримується активною спільнотою, що робить його привабливим вибором для розробки соціальних мереж. Остаточний вибір між цими стеками буде залежати від конкретних потреб розроблюваного проєкту та власних уподобань у розробці програмного забезпечення.

Таблиця 1.1 – Порівняння стеків для розробки

Особливості	Ruby / Ruby on Rails	Python / Django	JavaScript / Node.js	PHP / Laravel
Продуктивність розробки у невеликій команді	Висока	Середня	Середня	Середня
Екосистема та готові рішення	Багата екосистема, велика кількість готових рішень	Значна кількість бібліотек та пакетів, але менше ніж у Rails	Широкий вибір бібліотек, але менше ніж у Rails та Django	Багата екосистема, зручні інструменти для розробки
Спільнота та підтримка	Активна спільнота, гарна документація	Велика спільнота, документація	Активна спільнота, але менша ніж у Rails та Django	Активна спільнота, зручна документація

Продовження таблиці 1.1

Особливості	Ruby / Ruby on Rails	Python / Django	JavaScript / Node.js	PHP / Laravel
Швидкість виконання	Середня	Середня	Висока	Висока
Масштабованість	Висока	Висока	Висока	Висока
Складність вивчення та крива навчання	Середня	Середня	Висока	Середня
Інструменти фронтенду	ERB, HotWire, CSS	Django Templates, JavaScript, CSS	React, Vue.js, Angular, HTML, CSS	Blade, JavaScript, CSS
Філософія	Конвенція над конфігурацією	Явне краще, ніж неявне	Асинхронне, неблокуюче введення / виведення	Виразний, елегантний синтаксис

Laravel, так само як і Ruby та Ruby on Rails, володіє високою продуктивністю розробки, швидким виконанням, доброю масштабованістю та активною спільнотою. Вибір між Laravel та Ruby on Rails може залежати від ваших особистих вподобань, досвіду, а також конкретних вимог вашого проєкту.

Для розробки проєкту «соціальна мережа» було обрано Ruby on Rails через високу продуктивність для невеликої команди або одного розробника. Ruby on Rails має великий нативний інструментарій для написання фул стак додатку. Філософія «конфевнція над конфігурацією» робить цей продукт ідеальним для швидкої розробки невеликих проєктів, бо все що потрібне для написання MVP (мінімально життєздатного продукту) вже є сконфігуровано у фреймворку.

HoWire та ERB значно облегшує розробку фронтенду, не перевантажуючи розробника JavaScripipt фреймворками. Також буде використано Docker та docker-compose для швидкого підняття та встановлення залежностей проєкту, фреймворк Rspec для написання юніт тестів, Swagger – для написання документації для API.

1.1.4 Вибір СКБД

СКБД – це програмне забезпечення, яке надає структурну можливість організовувати та керувати даними.

У світі розробки програмного забезпечення існує ряд систем керування базами даних (СКБД). Серед найбільш використовуваних СКБД на сьогодні виділяються PostgreSQL, MySQL та MongoDB.

PostgreSQL – це об’єктно-реляційна система керування базами даних, яка використовує мову SQL для зберігання та управління даними [16]. Вона відкрита, безкоштовна та має велику спільноту розробників, що постійно допомагає у її розвитку та підтримці [19]. PostgreSQL підтримує широкий спектр функцій, таких як складні типи даних, транзакції, підтримка відновлення після збоїв та вибіркові індекси.

MySQL – це об’єктно-реляційна СКБД. MySQL відома своєю простотою в установці та використанні, що робить її дуже популярною серед розробників [18]. Однак PostgreSQL має більшу підтримку стандартів SQL, більшу функціональність та розширюваність, що робить її більш потужною для складних проєктів, зокрема, для соціальних мереж. Також PostgreSQL має відкритий вихідний код, що покращує підтримку суспільства .

MongoDB – це документно-орієнтована NoSQL база даних. MongoDB підходить для проєктів, де потрібна гнучкість схеми даних та швидкість запису [17]. У соціальних мережах, де важливо зберігати великі об’єми різноманітних даних, MongoDB може бути привабливою альтернативою для деяких конкретних випадків використання.

Однак, у випадку розробки соціальної мережі на Ruby on Rails, PostgreSQL виявляється більш привабливим вибором через свою реляційну модель, яка часто більш підходить для таких проєктів, де потрібна структурована та сполучена інформація про користувачів та їх взаємодію. Вбудована ActiveRecord є відмінною системою об’єктно-реляційного відображення, що нативно взаємодіє з базою даних.

Однією з переваг PostgreSQL є його розширюваність та гнучкість. Вона має велику кількість розширень та модулів, які дозволяють налаштовувати базу даних під конкретні потреби проєкту. Крім того, PostgreSQL має потужну систему безпеки, що дозволяє контролювати доступ до даних та забезпечувати їхню конфіденційність та цілісність.

Для проєктів, написаних на Ruby on Rails, PostgreSQL є стандартом у стеку.

1.2 Мова програмування Ruby

Ruby – це динамічна, об’єктно-орієнтована мова програмування, створена Юкіхіро Мацуумото [1]. Перший офіційний реліз Ruby з’явився 21 грудня 1995 року. Ruby була розроблена з метою створення мови програмування, яка є простою у використанні та водночас потужною.

У Ruby все є об’єктом, включаючи примітивні типи даних [2]. Це дозволяє використовувати гнучкість об’єктно-орієнтованого програмування (ООП) для написання чистого та підтримуваного коду. Ruby використовує динамічну типізацію, що означає, що типи змінних визначаються під час виконання програми, а не на етапі компіляції.

Вбудований збирач сміття, який автоматично звільняє пам’ять, що більше не використовується програмою. Фаза маркування починається з маркування всіх кореневих об’єктів (об’єктів, які безпосередньо доступні з програми) як сірих. Потім ітеративно обробляються сірі об’єкти, маркуючи

їхні дочірні об’єкти сірими, а оброблені об’єкти стають чорними. Цей процес продовжується, поки не залишиться більше сірих об’єктів [4].

Ruby була створена з ідеєю «щастя розробника», тому її синтаксис легко зрозуміти, читати та писати. Це також сприяє тому, що у Ruby низький поріг входу для новачків.

Підтримка метапрограмування дозволяє програмам модифікувати себе під час виконання за допомогою таких технік як Monkey Patching.

Ruby не підтримує множинне наслідування, тому замість них використовуються модулі (Modules). Модулі – це набір методів та значень, які можна імпортувати в клас.

Для імпорту функціоналу у Ruby використовуються Геми. Геми – це пакети або бібліотеки, які містять готовий код для виконання певних завдань. Геми розповсюджуються через RubyGems та встановлюються за допомогою bundler, що автоматично вирішує всі залежності та конфлікти.

Ruby підтримує багатопотоковість, що дозволяє програмам виконувати кілька задач одночасно [3]. Для цього використовуються Threads та Fibers. Також існують різні бібліотеки та фреймворки для асинхронного програмування, які дозволяють ефективно працювати з асинхронними операціями в Ruby.

Спільнота розробників Ruby активно розвивається і підтримує безліч інструментів, гемів, бібліотек та фреймворків. Також існує велика кількість імплементацій рубі (див. табл. 1.2) на різних мовах програмування та з різними техніками, що по різному впливає на продуктивність програм, хоча MRI все ще найбільш стабільна та використовувана версія [15].

Таблиця 1.2 – Порівняння імплементацій Ruby

Імплементація	**Опис**
MRI (Matz’s Ruby Interpreter)	Офіційна імплементація, написана мовою C. Розробляється головним розробником Ruby, Юкіхіро Мацумото.

Продовження таблиці 1.2

Імплементація	**Опис**
JRuby	Ruby, що працює на віртуальній машині Java (JVM). Вона дозволяє використовувати бібліотеки Java у Ruby-коді та інтегрувати Ruby в Java-додатки.
Rubinius	Це імплементація Ruby, що використовує власний віртуальний стек (крім стандартного механізму стеку JVM або C). Вона була спроєктована для того, щоб бути більш модульною та легко розширюваною.
TruffleRuby	Це ще одна імплементація Ruby, яка працює на віртуальній машині Java (JVM). TruffleRuby використовує технологію GraalVM для виконання Ruby-коду, що дозволяє досягти високої продуктивності та інтеграції з іншими мовами програмування.
MRuby	Легка імплементація Ruby, спроєктована для вбудованого використання. MRuby створена з метою оптимізації для використання в обмежених ресурсах, таких як мікроконтролери та вбудовані системи.

1.3 Ruby on Rails

1.3.1 Історія та архітектура Ruby on Rails

Ruby on Rails – це фреймворк розробки вебдодатків, написаний мовою програмування Ruby, спрямований на зменшення коду та підвищення продуктивності. Його філософія базується на принципах «Не повторюйтеся» (DRY) та «Конвенція над конфігурацією». Rails робить припущення про «найкращий» спосіб робити речі, що може підвищити продуктивність, але також може створити перешкоди для розробників, які відхиляються від “Rails

Way”. Попри це, він забезпечує швидкість розробки та полегшує обслуговування коду [5].

Ruby on Rails був створений Девідом Генсом (David Heinemeier Hansson) у 2003 році як інструмент для швидкої розробки вебдодатків. Перший офіційний реліз Rails (версія 0.5) відбувся у липні 2004 року.

Ruby on Rails використовує архітектуру MVC (або модель-вигляд-контролер) для побудування вебдодатків. MVC – це шаблон проєктування програмного забезпечення, що використовується для організації коду вебдодатків з метою полегшення розробки та підтримки [6].

У MVC система розділена на три основні компоненти: модель, вигляд, контролер.

Модель (Model) – це представлення даних, бізнес логіки та обробки бази даних за допомогою ORM (Об’єктно-реляційне відображення). Ruby on Rails по стандарту використовує ActiveRecord ORM.

Вигляд (View) – це представлення даних користувачу. Вигляд відповідає за відображення інформації, яку видає модель. Ruby on Rails генерує HTML сторінки за допомогою шаблонізатора erb [7] та HotWire. Модульність забезпечується за допомогою Concern [8]. Concern – це модуль, який має спеціальну структуру для використання в Rails, зокрема використовуючи ActiveSupport::Concern для включення в класи моделей та контролерів. Це дозволяє використовувати певні зручності, такі як автоматичне виконання коду при включенні Concern у клас.

Контролер (Controller) – це посередник між моделлю та виглядом. Контролер відповідає за прийом запитів від користувача, взаємодію з моделлю для отримання необхідних даних та передачу цих даних у вигляд для відображення. Крім цього, він також відповідає за обробку подій та логіку переходів між різними сторінками чи функціональними частинами додатку

Завдяки Routes фреймворк отримує URL та направляє виконання запиту певному контролеру.

Міграції – це спосіб управління базою даних за допомогою коду. Це дає

можливість швидко та просто налаштовувати схему та редагувати таблиці.

Багато популярних проєктів було побудовано та підтримується на Ruby on Rails. Основні приклади: Github, Basecamp, Airbnb, Shopify, Twitch.

Станом на 2023 – 5.14% розробників використовують фреймворк Ruby on Rails. Для порівняння, Node.js – 42.65%, Django – 11.47%, Laravel – 7.68% [12].

Рік після релізу (2005) Ruby on Rails отримав бум популярності, оскільки це був новий фреймворк, а програмістам подобалася швидкість розробки та масштабованість системи. Проте, з того часу популярність пішла на спад через велику кількість конкуренції та з появою нових інноваційних оновлень в інших фреймворках.

Користувачі часто питають, чи є Ruby on Rails вмираючим фреймворком. З роками він отримав багато оновлень для покращення життя та спростування розробки, через що зайняв своє місце на ринку як фреймворк для розробки MVP та стартапів, хоч його популярність і не така як була раніше[13].

1.3.2 Концепція «Конвенція понад конфігурацію»

«Конвенція понад конфігурацією» – це основний принцип фреймворку Ruby on Rails [9]. Це означає, що фреймоворк припускає найкращий метод імплементації певного функціоналу та змушує розробника слідувати цим методам, обмежуючи гнучкість конфігурації проєкту.

Наприклад, якщо була створена модель User, то фреймворк автоматично починає посилатися на таблицю users у множинній формі. Це обмежує гнучкість імплементації, але водночас робить вебдодаток набагато легшим у підтримці та читаності. Слідування цим конвенціям пришвидшує написання продуктів та значно полегшує підтримку проєкту командою розробників.

1.4 HotWire

1.4.1 Огляд технології HotWire

HotWire (HTML over wire) – це підхід та набір технологій, таких як Turbo Drive, Turbo Frames, Turbo Streams, Stimulus та Strada, створених та зібраних разом для оптимізації динамічних сторінок без використання JavaScript фреймворків [10].

1.4.2 Компоненти: Turbo і Stimulus

Turbo – це набір технологій, який вирішує проблему оновлення сторінки для оновлення контенту сторінки [11].

Turbo Drive перехоплює натискання юзера на посилання та форми, запобігаючи стандартним протоколам браузеру, робить запрос до серверу та одночасно оновлює контенти сторінки.

Turbo Frames ділить сторінки на частини та ліново завантажує їх.

Turbo Stream використовує протоколи WebSocket для швидкого та живого оновлення даних на сторінці.

Turbo Native оптимізує кросплатформовий перехід вебсторінок.

Stimulus – це JavaScript фреймворк, призначений для покращення інтерактивності вебсторінок шляхом надання підходу додавання поведінки до елементів HTML.

Stimulus дозволяє розробникам створювати динамічні користувацькі інтерфейси без потреби у складних платформах JavaScript на стороні клієнта.

Використовуючи атрибути даних і контролери, Stimulus дозволяє розробникам інкапсулювати поведінку в окремих компонентах.

1.4.3 Переваги та недоліки Hotwire

Основна задача HotWire – оптимізація та облегшення розробки Full Stack проєкту. Через це втрачається гнучкість інструментарію, як наприклад у React або Vue.js, оскільки, як і з фреймворком Ruby on Rails, HotWire побудований на принципі «конвенції понад конфігурацій».

З акцентом на простоті і сумісності з іншими веб технологіями HotWire відмінний вибір для розробки швидкого продукту в невеликій команді розробників.

1.5 Висновки до розділу 1

Було розглянуто вимоги вебдодатку «соціальна мережа», проаналізовано ринок на вже існуючі схожі рішення, проаналізовано найпопулярніші стеки технологій. Було обрано та приведено теоретичні відомості про Ruby, Ruby on Rails, Hotwire.

2 ПРОЦЕС ПРОЄКТУВАННЯ ВЕБДОДАТКУ

2.1 Діаграма прецедентів

Для візуалізації взаємодії користувача і системи було розроблено діаграму прецедентів (див. рис 2.1).

Діаграма прецедентів (Use Case Diagram) – це діаграма у методології UML (Unified Modeling Language), що використовується для моделювання функціональних вимог системи. Вона відображає взаємодію між користувачами (акторами) та системою, показуючи різні сценарії використання (прецеденти) цієї системи.

Вебдодаток має 4 ролі:

– гість (guest);
– блогер (blogger);
– менеджер (manager);
– адміністратор (admin);

Гість – це незареєстрований користувач. Гість може тільки авторизуватися або зареєструватися.

Блогер – це зареєстрований користувач. Блогер може створювати блогпости, дивитися та коментувати блогпости інших блогерів, ставити лайки під постами та коментарями, писати персональні повідомлення. Блогер також може продивлятися профілі інших користувачів, підписуватися на їх розсилку та потім бачити їх пости у своїй стрічці. Блогеру дозволено змінювати та видаляти свої пости, коментарі та лайки.

Менеджер – це спеціальна роль користувача для адміністрування постів. Цей користувач може продивлятися, змінювати та видаляти пости та коментарі блогерів.

Адміністратор – це найпривілейованіша роль користувача. Адміністратор має доступ до адмін панелі, де він може передивлятися,

створювати та редагувати або видаляти пости та коментарі. Крім цього в адмін панелі відбувається налаштування користувачів. Адміністратор може створювати акаунти, редагувати їх та надавати їм привілеї.

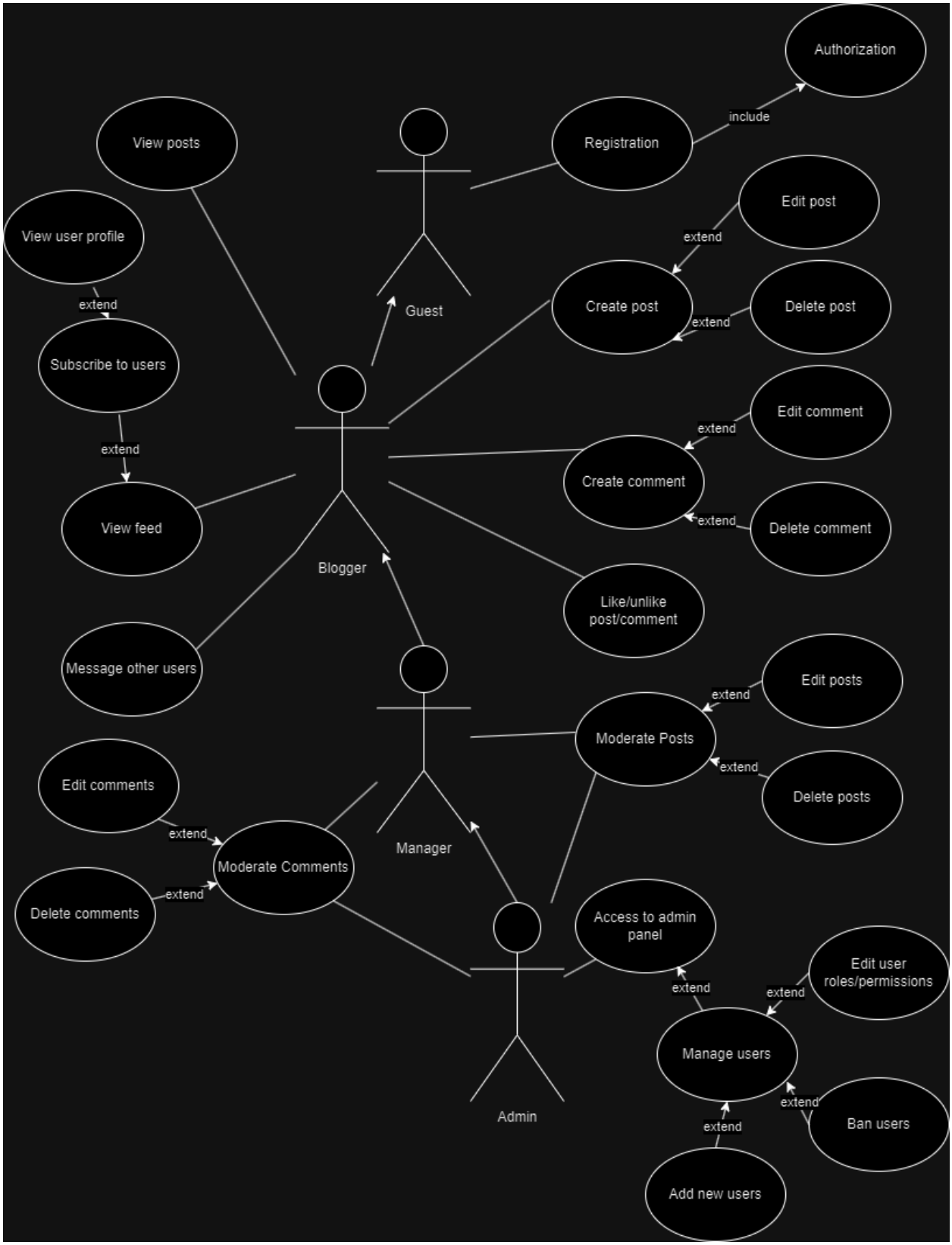

Рисунок 2.1 – Діаграма прецедентів

2.2 Схема бази даних

Схема бази даних, або ER-діаграма (Entity-Relationship Diagram) – це графічне зображення структури бази даних. Вона відображає сутності (таблиці) та їхні взаємозв’язки (зв’язки), що допомагає зрозуміти, як дані організовані та взаємодіють між собою.

ER-діаграма розроблюваного застосунку наведена на рисунку 2.2.

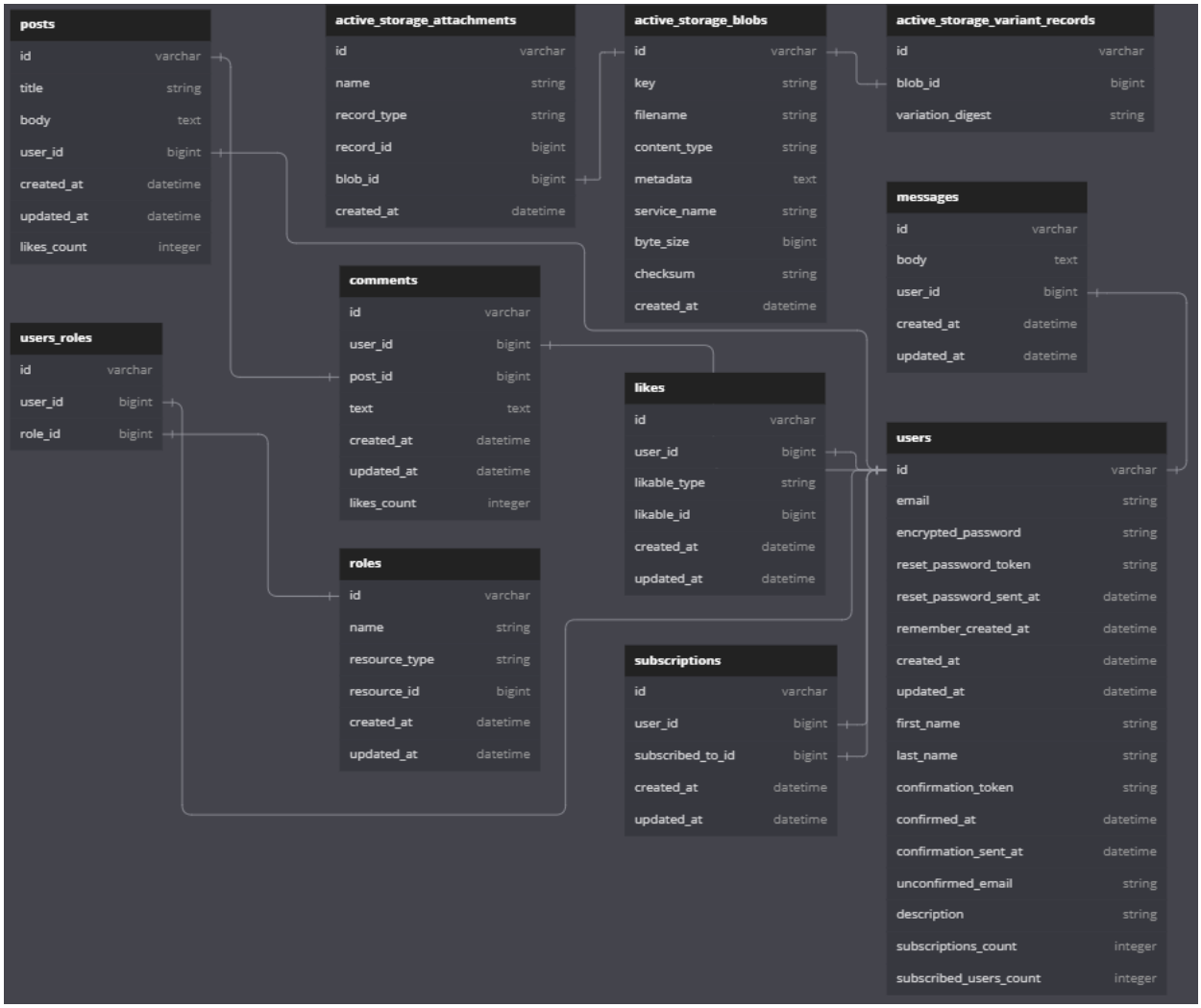

Рисунок 2.2 – Схема бази даних

База даних складається із 11 таблиць:

– users: інформація про користувача, поля та токени для реєстрації Devise;
– posts: інформація про пост, належить користувачу;

- comments: коментарі, належить до поста та користувача;
- likes: лайки, належить користувачу та коментарю або посту з поліморфічним зв’язком;
- subscriptions: підписки, належить до користувача як підписник і той до кого підписуються;
- messages: повідомлення, належить до користувача;
- roles: ролі/дозволи користувачів;
- user_roles: таблиця зв’язку багато-до-багатьох, між користувачами та ролями;
- active_storage_attachments, active_storage_blobs та active_storage_variant_records: таблиці ActiveStorage для інформації про прикріплені файли.

2.3 Висновки до розділу 2

Було наведено та проаналізовано діаграму прецедентів (Use-Case) та схему бази даних проєкту (ER diagram).

3 ПРОЦЕС РОЗРОБКИ ВЕБДОДАТКУ

3.1 Реалізація веб додатку

3.1.1 Структура проєкту

Оскільки філософія Ruby on Rails – це «конвенції понад конфігурацій», при створенні проєкту всі потрібні директорії вже будуть створені та будуть мати певні файли, які потрібно або доповнювати, або створювати схожі за прикладом.

Огляд важливих тек директорії:

- /app/controllers – контролери;
- /app/models – моделі;
- /app/views – вигляди;
- /app/policies – дозволи користувачів;
- /app/assets – скомпільовані ресурси для фронтенду;
- /app/jobs – CRON роботи;
- /config – файли конфігурації проєкту, бази даних та гемів;
- /db – файли міграції, сідів та схеми бази даних;
- /lib – зовнішній код проєкту, який не є гемами та не підходить до використання у concerns;
- /public – публічні ресурси, локальні файли;
- /spec – юніт тести з використанням фреймворку Rspec;
- Gemfile та Gemfile.lock – файли залежностей, які використовує bundler для їх встановлення.

3.1.2 Докеризація проєкту та розгортання

Докеризація проєкту має величезне значення з точки зору розгортання, масштабованості та управління проєктом. За допомогою Docker контейнерів можна створювати ізольовані середовища, що містять усі необхідні залежності та конфігураційні файли. Це дозволяє забезпечити однакові умови роботи як локально для розробників, так і на сервері для продакшену.

Docker спрощує розгортання програмного забезпечення на будь-якому хості, незалежно від його операційної системи або конфігурації, що робить процес розробки та впровадження більш простим та ефективним.

За основу для Dockerfile було взято образ написаний для Ruby з публічного репозиторію образів для Докеру. Цей образ був написаний зі всіма потрібними конфігураціями починаючи з Ubuntu, що робить цей образ найбільш оптимізованим для цього проєкту.

Написаний для проєкту Dockerfile встановлює всі додаткові залежності, робить налаштування директорій та запускає необхідні команди для серверу (див. рис. 3.1).

Dockerfile

```
1   # syntax=docker/dockerfile:1
2   FROM ruby:3.1.2
3   RUN apt-get update -qq && apt-get install -y nodejs postgresql-client
4   WORKDIR /myapp
5   COPY Gemfile /myapp/Gemfile
6   COPY Gemfile.lock /myapp/Gemfile.lock
7   RUN bundle install
8
9   # Add a script to be executed every time the container starts.
10  COPY entrypoint.sh /usr/bin/
11  RUN chmod +x /usr/bin/entrypoint.sh
12  ENTRYPOINT ["entrypoint.sh"]
13  EXPOSE 3000
14
15  # Configure the main process to run when running the image
16  CMD ["rails", "server", "-b", "0.0.0.0"]
```

Рисунок 3.1 – Dockerfile проєкту

Також для оптимізації взаємодії з базою даних було написано docker-compose.yml (див. рис. 3.2). Він підіймає базу даних PostgreSQL та сервер проєкту як контейнери «db» та «web» відповідно (див. рис. 3.3).

docker-compose.yml

```yaml
services:
  db:
    image: postgres
    volumes:
      - ./tmp/db:/var/lib/postgresql/data
    environment:
      POSTGRES_PASSWORD: password
  web:
    build: .
    command: bash -c "rm -f tmp/pids/server.pid && bundle exec rails s -p 3000 -b '0.0.0.0'"
    volumes:
      - .:/myapp
    ports:
      - "3000:3000"
    depends_on:
      - db
```

Рисунок 3.2 – Docker-compose.yml

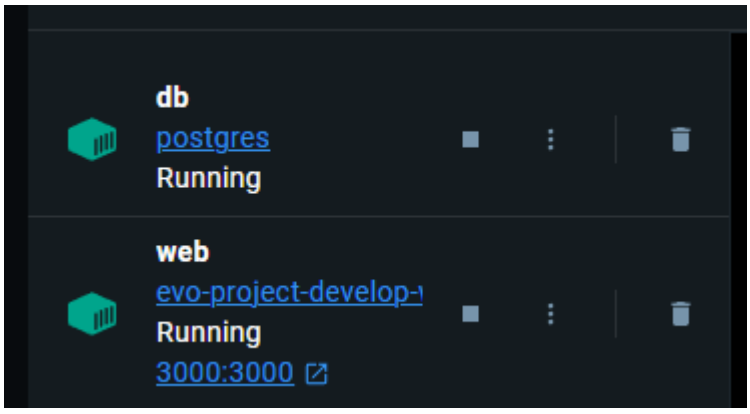

Рисунок 3.3 – Розгорнуті контейнери проєкту

3.1.3 Реєстрація та авторизація за допомогою Devise та JWT

Для реалізації авторизації було обрано рішення Devise. Devise – це найпопулярніший та найстабільніший гем для управління користувачами. Простота встановлення та тонка конфігурація робить його кращим рішенням

для більшсті випадків. Процес встановлення легкий – встановити гем Devise, та створити за допомогою нього модель User. Це створить всі потрібні міграції (див. рис. 3.4) та контролери.

```ruby
class User < ApplicationRecord
  rolify
  after_create :assign_default_role
  devise :database_authenticatable, :registerable,
         :recoverable, :rememberable, :validatable,
         :confirmable

  has_many :posts
  has_many :comments
  has_many :likes
  has_many :subscriptions
  has_many :subscribed_users, through: :subscriptions, source: :subscribed_to
  has_many :messages, dependent: :destroy

  has_one_attached :avatar do |attachable|
    attachable.variant :thumb, resize_to_limit: [100, 100]
  end

  validates :first_name, :last_name, presence: true
```

Рисунок 3.4 – Модель користувача з використанням Devise

Якщо потрібно змінити поля для юзерів, їх можна додати моделі, а потім підправити дозволені параметри у контролері.

До звичайних полів Devise було додано description (опис профіля) та avatar (фото аватар) (див. рис. 3.5).

```ruby
# frozen_string_literal: true

class Users::RegistrationsController < Devise::RegistrationsController
  before_action :configure_account_update_params, only: [:update]

  private

  def configure_account_update_params
    devise_parameter_sanitizer.permit(:account_update, keys: [:avatar, :description])
  end
end
```

Рисунок 3.5 – Конфігурація параметрів користувача у контролері Devise

Для більшості написаного функціоналу було написано також API ендпоінти, з якими Devise нативно не працює, тому було реалізовано JWT токени для реєстрації (див. рис. 3.6).

JWT (JSON Web Token) – це стандарт відкритого обміну даними в форматі JSON, який дозволяє безпечно передавати інформацію між сторонами у вигляді компактного та самостійного засвідчення. Кожен токен містить закодовані дані, такі як ідентифікатор користувача чи інформація про його права доступу, і підпис цих даних, що дозволяє перевірити їх цілісність.

app > controllers > concerns > authenticatable.rb

```ruby
module Authenticatable
    extend ActiveSupport::Concern

    included do
      skip_before_action :verify_authenticity_token

      private

      def authenticate_user_with_token
        token = request.headers['Authorization']&.split(' ')&.last
        decoded_info = JWT.decode(token, Rails.application.credentials.jwt[:AUTH_SECRET]) if token.present?

        if decoded_info && (user = User.find_by(email: decoded_info[0]['email']))
          sign_in(user, store: false)
        else
          head :unauthorized
        end
      end
    end
end
```

Рисунок 3.6 – Concern авторизації за допомогою JWT та Devise для API

3.1.4 Розробка шляхів, моделей, міграцій, контролерів та виглядів

На основі постів розберемо як відбувається розробка функціоналу у Ruby on Rails.

У папці конфігурації можна налаштувати які шляхи програма може приймати (див. рис. 3.7). resources створює шляхи для основних CRUD операцій (створити, прочитати, оновити, видалити) (див. рис. 3.8). Вказуючи only: (тільки) або except: (окрім) можна налаштувати які самі операції мають бути для цього шляху.

```ruby
resources :posts do
  resources :comments, except: [:index, :show, :new] do
    resources :likes, only: [:create, :destroy]
  end
  resources :likes, only: [:create, :destroy]
end
```

Рисунок 3.7 – Ініціалізація шляхів для постів

```
post_comment_likes POST   /posts/:post_id/comments/:comment_id/likes(.:format)          likes#create
 post_comment_like DELETE /posts/:post_id/comments/:comment_id/likes/:id(.:format)      likes#destroy
     post_comments POST   /posts/:post_id/comments(.:format)                           comments#create
 edit_post_comment GET    /posts/:post_id/comments/:id/edit(.:format)                  comments#edit
      post_comment PATCH  /posts/:post_id/comments/:id(.:format)                       comments#update
                   PUT    /posts/:post_id/comments/:id(.:format)                       comments#update
                   DELETE /posts/:post_id/comments/:id(.:format)                       comments#destroy
       post_likes POST   /posts/:post_id/likes(.:format)                               likes#create
        post_like DELETE /posts/:post_id/likes/:id(.:format)                           likes#destroy
            posts GET    /posts(.:format)                                              posts#index
                  POST   /posts(.:format)                                              posts#create
         new_post GET    /posts/new(.:format)                                          posts#new
        edit_post GET    /posts/:id/edit(.:format)                                     posts#edit
             post GET    /posts/:id(.:format)                                          posts#show
                  PATCH  /posts/:id(.:format)                                          posts#update
                  PUT    /posts/:id(.:format)                                          posts#update
                  DELETE /posts/:id(.:format)                                          posts#destroy
```

Рисунок 3.8 – Згенеровані шляхи для постів

У моделях оголошуються основні атрибути, валідації та зв’язки, які потім будуть використовуватись ORM ActiveRecord у роботі з базою даних (див. рис. 3.9).

```ruby
class Post < ApplicationRecord
  resourcify

  belongs_to :user
  has_many :comments, dependent: :delete_all
  has_many :likes, as: :likable
  has_one_attached :image

  validates :title, presence: true, length: { maximum: 100 }
  validates :body, presence: true, length: { maximum: 255 }
  validates :image, blob: { content_type: ['image/png', 'image/jpg', 'image/jpeg']

  scope :search_by_title, ->(search) {
    where("title ILIKE ?", "%#{search}%").includes(:user)
  }

  def self.ransackable_attributes(auth_object = nil)
    ["body","title", "id", "user_id"]
  end

  def self.ransackable_associations(auth_object = nil)
    []
  end
end
```

Рисунок 3.9 – Приклад моделі постів

Основний синтаксис моделі:

– belongs_to *назва моделі* – вказує зв’язок один-до. Якщо зв’язок один-до-одного, то в іншій моделі використовується has_one;
– has_many – зв’язок один-до-багатьох, в іншій моделі повинно бути присутнє belongs_to;
– has_many_and_belongs_to_many або has_many :through – це зв’язок багато-до-багатьох (повинна бути створена таблця, яка буде зберігати зовнішні ключі обох моделей);
– validates – валідація атрибуту за допомогою нативних або створених методів;
– scope – інкапсульований запит до бази даних.

Після генерації моделі генерується міграція (див. рис. 3.10). Без виконання міграції фреймворк буде видавати помилку. Важливо при написанні міграції мати можливість відкату, що при використанні автоматичного синтаксису (def change) автоматично ініціалізується. Проте іноді виникає потреба в написанні інших методів відкату, тому іноді використовується метод up, який описує що робити при міграції, та down, який описує що відбувається при відкаті.

```
class CreatePosts < ActiveRecord::Migration[7.0]
  def change
    create_table :posts do |t|
      t.string :title
      t.text :body
      t.references :user, null: false, foreign_key: true

      t.timestamps
    end
  end
end
```

Рисунок 3.10 – Приклад міграції постів

Коли приходить запит на певний шлях, Rails викликає необхідний контролер і метод з цього контролеру як вказано в шляхах. Контролер

відповідає за зв’язок усіх компонентів програми між собою. На прикладі контролеру постів (див. рис. 3.11) роздивимося основні методи роботи з контролером.

```ruby
class PostsController < ApplicationController
    before_action :authenticate_user!, except: [:index, :show]
    before_action :set_post, only: [:show, :edit, :update, :destroy]

    def index
        @pagy, @posts = pagy(Post.all)
    end

    def new
        @post = Post.new
    end

    def create
        @post = Post.new(post_params)
        @post.user_id = current_user.id

        if @post.save
            redirect_to root_path
        else
            render :new, status: :unprocessable_entity
        end
    end

    def edit
        authorize @post
    end

    def update
        authorize @post
        @post.image.attach(params[:image])
        if @post.update(post_params)
            redirect_to @post
        else
            render :edit, status: :unprocessable_entity
        end
    end

    def destroy
        authorize @post
        @post.destroy
        redirect_to root_path, status: :see_other
    end
```

Рисунок 3.11 – Приклад контролеру посту

Методи new та create відповідають за створення нового запису. new відправляє новостворений пустий об’єкт до вигляду, а create займається керуванням створення нового запису та редіректом на потрібні сторінки. edit та update працюють аналогічно тільки для оновлення запису. Destroy займається видаленням. Метод index викликає колекцію постів, використовуючи пагінацію за допомогою гему Pagy та передає їх для вигляду.

Важливою частиною роботи з контролерами у Ruby on Rails є дозвіл параметрів (див. рис. 3.12). Задля безпеки фреймворк не довзволяє зберігати будь-які параметри, оскільки юзер може прислати той параметр, який не повинен зберігатися у базі даних в даний момент, що може зашкодити безпеці, викликати баг або привести до небажаного результату.

Колбек before_aciton (а також after_action та around_action), дозволяє викликати якусь частину коду перед методом. У контролері постів колбеки before_action authenticate_user! Перевіряє щою користувач був зареєстрований, перед тим як викликати index та show, а set_post створює об’єкт моделі з баи даних згідно з id.

```
54    private
55
56    def post_params
57        params.require(:post).permit(:title, :body, :image)
58    end
59
```

Рисунок 3.12 – Дозвіл параметрів у контролері

На прикладі форми створення постів розглянемо розробку вигляду (див. рис. 3.13). Вигляди написані на мові для написання шаблонів erb та за допомогою нативних методів HotWire, які оптимізують сторінку. erb – це HTML код із вставленим в нього кодом Ruby, завдяки чому досягається динаміка. За допомогою render відбувається виклик partials. Partials – це якась частина вебсторінки яка повторюється и повинна бути винесена в окремий елемент. В нашому випадку це хедер.

app > views > posts > new.html.erb

```erb
<%= render "shared/header"%>

<h1>New Post</h1>

<% cache(["new_post_form", @post]) do %>
  <div class="mx-3">
    <%= form_with model: @post do |form| %>
      <div class="form-group">
        <%= form.label :title %>
        <%= form.text_field :title, class: "form-control" %>
        <% @post.errors.full_messages_for(:title).each do |message| %>
          <div class="text-danger"><%= message %></div>
        <% end %>
      </div>

      <div class="form-group">
        <%= form.label :body %>
        <%= form.text_area :body, class: "form-control" %>
        <% @post.errors.full_messages_for(:body).each do |message| %>
          <div class="text-danger"><%= message %></div>
        <% end %>
      </div>

      <div class="form-group">
        <%= form.label :image %>
        <%= form.file_field :image, class: "form-control-file" %>
        <% @post.errors.full_messages_for(:image).each do |message| %>
          <div class="text-danger"><%= message %></div>
        <% end %>
      </div>

      <div class="form-group">
        <%= form.submit "Create Post", class: "btn btn-primary" %>
      </div>
    <% end %>
  </div>
<% end %>

<div class="mx-3 my-3">
  <%= link_to "View all posts", posts_path, class: "btn btn-secondary" %>
</div>

```

Рисунок 3.13 – Форма створення постів у вигляді

Метод cache дозволяє закешувати частину сторінки задля покращення продуктивності. Метод form_with довзоляє написати форму для певної моделі, яка буде мати власний CSRF токен, що покращує безпеку, а також буде пришвидшена використанням HotWire. У формі можна динамічно описати усі поля а також викликати вивід помилок. Метод link_to створює посилання на URL, який називається «post_path» у шляхах и виглядає як «/posts». Фоорма

напряму створює об’єкт моделі та відправляє його назад до контролера, де викликається вже метод create.

За допомогою turbo_stream та ActionCable було розроблено чатрум, де користувачі можуть обмінюватися повідомленнями в прямому ефірі. Для реалізація було написано контролер за допомогою Stimulus (див. рис. 3.14).

app > javascript > controllers > JS chat_controller.js > ...

```
import { Controller } from "@hotwired/stimulus"

// Connects to data-controller="chat"
export default class extends Controller {
  static values = { currentUserId: String };

  // Set up observers when the controller connects
  connect() {
    this.setupScrollListener();
    this.setupMessageListener();
  }

  // Disconnect observers when the controller disconnects
  disconnect() {
    this.scrollObserver.disconnect();
    this.messageObserver.disconnect();
  }

  // Set up an observer to scroll the chatbox to the bottom
  setupScrollListener() {
    this.scrollToBottom();

    this.scrollObserver = new MutationObserver(() => {
      this.scrollToBottom();
    });

    this.scrollObserver.observe(this.element, {
      childList: true,
      subtree: true,
    });
  }

  // Scroll to the bottom of the chatbox
  scrollToBottom() {
    this.element.scrollTop = this.element.scrollHeight;
  }

  // Get the message node from a node
  getMessageNode(node) {
    return node.closest(".message") || node.querySelector(".message");
  }
}
```

Рисунок 3.14 – Контролер повідомлень Stimulus

Після цього в виглядах було використано тег turbo_stream_for (див. рис. 3.15).

app > views > chatrooms > show.html.erb

```
<%= turbo_stream_from dom_id(current_user) %>
<div class="container mt-5">
  <div class="card">
    <div class="card-header">
      <h1>Chatrooms</h1>
      <p>Welcome <%= current_user.email %>!</p>
    </div>
    <div class="card-body">
      <div id="chatbox" class="mb-3">
        <%= render partial: "messages/chatbox", locals: { messages: @messages } %>
      </div>
      <%= render partial: "messages/form", locals: { message: @message } %>
    </div>
  </div>
</div>

```

Рисунок 3.15 – Вигляд чатруму з turbo_stream

Таким чином, також було розроблено наступний функціонал:

– коментарі: юзери можуть коментувати пости , змінювати та видаляти їх;
– лайки: юзери можуть ставити та прибирати «лайк» під постами та коментарями інших користувачів(поліморфічний зв’язок);
– підписка на профілі: юзери можуть підписатися на профілі інших користувачів щоб бачити їх пости у себе в стрічці новин;
– чат: за допомогою протоколу WebSocket юзери можуть обмінюватися повідомленнями у кімнатах чату без оновлення сторінки;
– адмін панель: розроблено за допомогою гему active-admin адміністратор може керувати постами, коментарями та користувачами.

3.2 Тестування вебдодатку

3.2.1 Юніт тестування

Написання тестів є важливою частиною циклу розробки будь-якого проєкту, оскільки тести підвищують підтримуваність проєкту та забезпечують відповідність вимогам у процесі розробки. Один з популярних методів розробки – TDD (Test Driver Development). З цим методом спочатку поступово пишуться тести до функціоналу, потім сам функціонал, поки тести не пройдуть. Це приводить до облегшеного процесу розробки та рефакторингу у процесі, оскільки це забезпечує планування функционалу до того як розробник приступив до його написання.

У Ruby on Rails є вбудований фреймврк тестування, але Rspec є найкращим для рубі, бо має більше всього підтримки спільноти, додаткових інструментів та читабельності.

На прикладі постів розберемо процес написання тестів за допомогою Rspec. Тестування моделей відбувається за допомогою перевірки – чи існує зв’язок з іншими моделями та чи валідуються певні поля (див. рис. 3.16).

post_spec.rb

spec > models > post_spec.rb

```ruby
require 'rails_helper'

RSpec.describe Post, type: :model do
  describe 'associations' do
    it { should belong_to(:user) }
    it { should have_many(:comments)  }
    it { should have_many(:likes)  }
  end

  describe 'validations' do
    it { should validate_presence_of(:title) }
    it { should validate_presence_of(:body) }
    it { should validate_length_of(:title).is_at_most(100) }
    it { should validate_length_of(:body).is_at_most(255) }
  end
end

```

Рисунок 3.16 – Юніт тест моделі Post

Метод describe описує шлях який тестується, context – дає опис що має відбуватися.

Для тестів контролерів використовуються техніки мокінгу та стабінгу. Мокінг – це сімукляція створення об’єкта та виконання дій над ним. Стабінг – виклик методів до певних ініціалізованих даних.

У прикладі тестів метода створення контролера постів (див. рис. 3.17) використовується метод let, який створює Mock object.

Метод create – це метод FactoryGirl, бібліотеки яка дозволяє створювати такі об’єкти. Метод before дозволяє виконати певну дію перед кожним юніт тестом. Метод excpect описує що повинно відбуватися, і якщо цього не відбувається, тест провалюється.

```ruby
describe 'POST /posts' do
  context 'when the user is authenticated' do
    let(:user) { create(:user, :confirmed) }

    before { sign_in user }

    context 'with valid parameters' do
      it 'creates a new post' do
        post_params = { post: attributes_for(:post) }

        expect {
          post '/posts', params: post_params
        }.to change(Post, :count).by(1)

        expect(response).to redirect_to(root_path)
      end
    end

    context 'with invalid parameters' do
      it 'does not create a new post' do
        post_params = { post: { title: '', body: '' } }

        expect {
          post '/posts', params: post_params
        }.not_to change(Post, :count)

        expect(response).to have_http_status(:unprocessable_entity)
        expect(response).to render_template(:new)
      end
    end
  end
```

Рисунок 3.17 – Юніт тести контролеру постів

Для більшої частини моделей та контролерів було написано юніт тести. Якщо запустити команду rspec, можна подивитися що тести проходять (бо вони відображені зеленим) (див. рис. 3.18).

```
root@91adbf735b48:/myapp# rspec

Comment
  associations
    is expected to belong to user required: true
    is expected to belong to post required: true
    is expected to have many likes
  validations
    is expected to validate that :text cannot be empty/falsy
    is expected to validate that the length of :text is at most 100

Like
  associations
    is expected to belong to user required: true
    is expected to belong to likable required: true

Post
  associations
    is expected to belong to user required: true
    is expected to have many comments
    is expected to have many likes
  validations
    is expected to validate that :title cannot be empty/falsy
    is expected to validate that :body cannot be empty/falsy
    is expected to validate that the length of :title is at most 100
    is expected to validate that the length of :body is at most 255

Comments
  POST #create
    when a user is authenticated
      with valid parameters
        creates a new comment
      with invalid parameters
        does not create a new comment
    when a user is not authenticated
      redirects to the sign-in page
  GET /posts/:post_id/comments/:id/edit
    when the user is authenticated
      returns a successful response for the user who owns the comment
      renders the edit template for the user who owns the comment
    when the user is authenticated but does not own the comment
      denies access
    when the user is not authenticated
      redirects to the sign-in page
  PUT /posts/:post_id/comments/:id
    when the user is authenticated
      with valid parameters
        updates the comment
      with invalid parameters
        does not update the comment
      when the user is not authorized to update the comment
        denies access
    when the user is not authenticated
      redirects to the sign-in page
  DELETE /posts/:post_id/comments/:id
    when the user is authenticated
      when the user owns the comment
        deletes the comment
      when the user does not own the comment
        denies access
    when the user is not authenticated
      redirects to the sign-in page
```

Рисунок 3.18 – Приклад запуску та проходження юніт тестів

3.2.2 Swagger документація до API

Swagger – це специфікація для опису RESTful API, що надає можливість описувати структуру та функціональні можливості API у форматі, який легко зрозуміти і використовувати як для розробки, так і для тестування. Rswag – це Ruby-гем для інтеграції Swagger з Ruby on Rails додатками. Він дозволяє автоматично генерувати документацію Swagger з коду Ruby, а також використовувати специфікації Swagger для автоматизованого тестування API.

Після конфігурації Rswag автоматично буде додавати тести з папки api до документації (див. рис. 3.19).

```
config.swagger_docs = {
  'v1/swagger.yaml' => {
    openapi: '3.0.1',
    info: {
      title: 'API V1',
      version: 'v1'
    },
    paths: {},
    servers: [
      {
        url: 'http://{defaultHost}',
        variables: {
          defaultHost: {
            default: 'localhost:3000'
          }
        }
      }
    ],
    securityDefinitions: {
      Bearer: {
        type: :apiKey,
        name: 'Authorization',
        in: :header,
        description: 'Your Bearer token'
      }
    },
    components: {
      securitySchemes:{
        bearerAuth: {
          type: 'http',
          scheme: 'bearer',
          bearerFormat: 'JWT'
        }
      }
    }
  }
}
```

Рисунок 3.19 – Конфігурація Rswag

В Authorization можна додати Bearer Token. Наприклад, можна спробувати отримати список постів за допомогою Swagger (див. рис. 3.20).

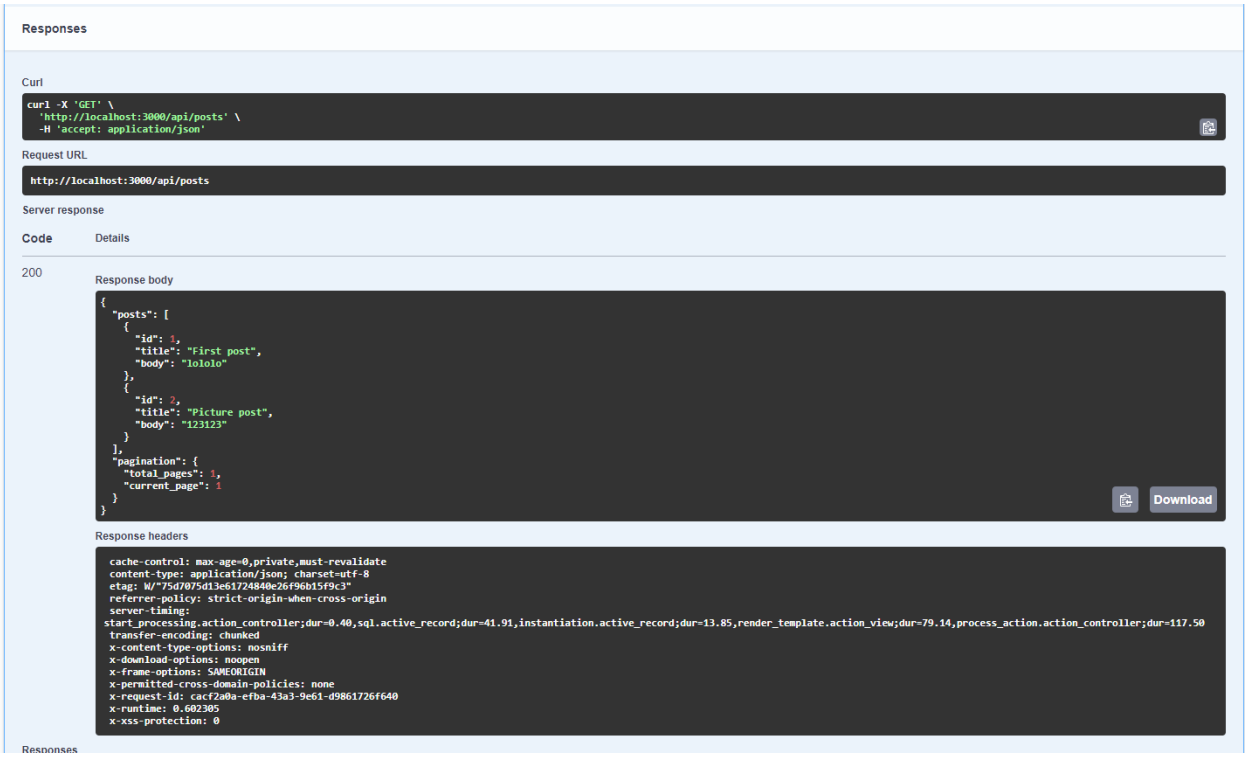

Рисунок 3.20 – Отримання відповіді на API запит у Swagger

3.3 Огляд результату

Для всього функціоналу було написано API ендпоінти. Також було додано фронтенд на bootstrap для візуалізації.

Незареєстрований користувач – це гість, який може тільки продивлятися записи. Щоб отримати доступ до функціоналу блогера, користувач повинен зареєструватися (див. рис. 3.21).

Для авторизації в API потрібно висилати JWT токен, який отримується при реєстрації (див. рис. 3.22).

Після цього цей токен можна вставляти в хедер Authorization для доступу до інших API (див. рис 3.23).

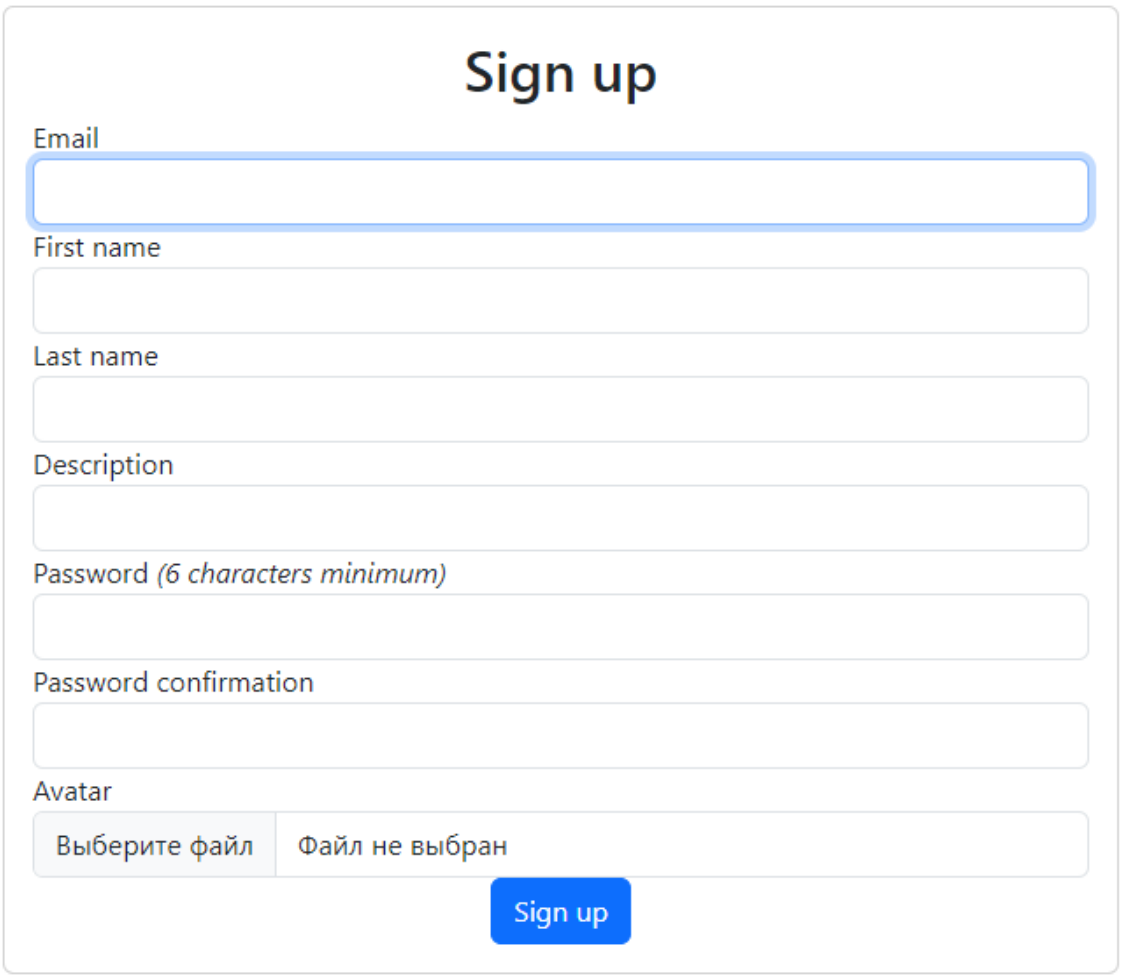

Рисунок 3.21 – Форма реєстрації

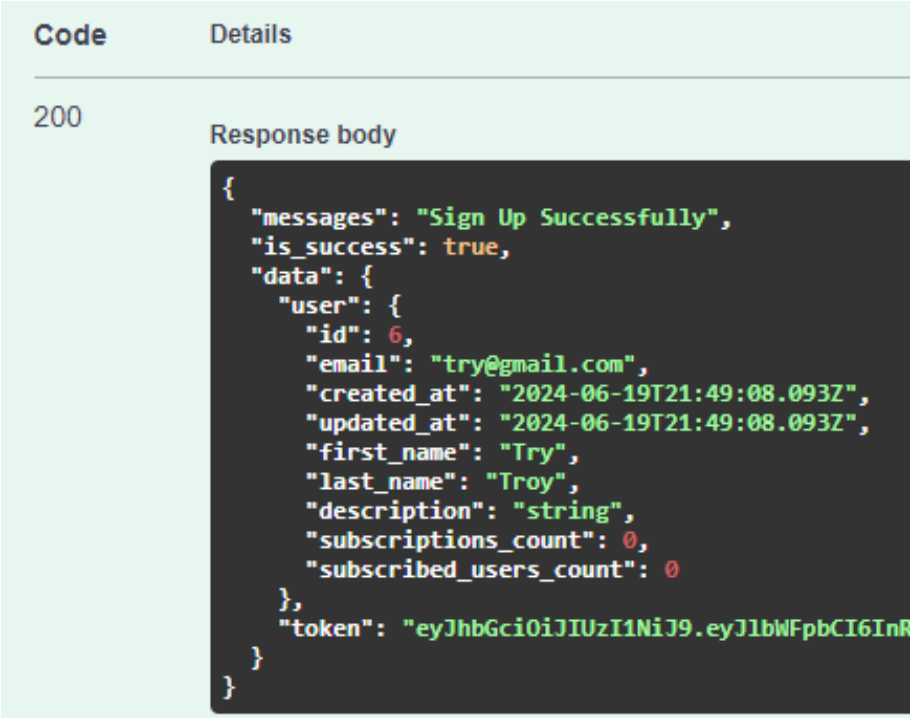

Рисунок 3.22 – API відповідь на реєстрацію

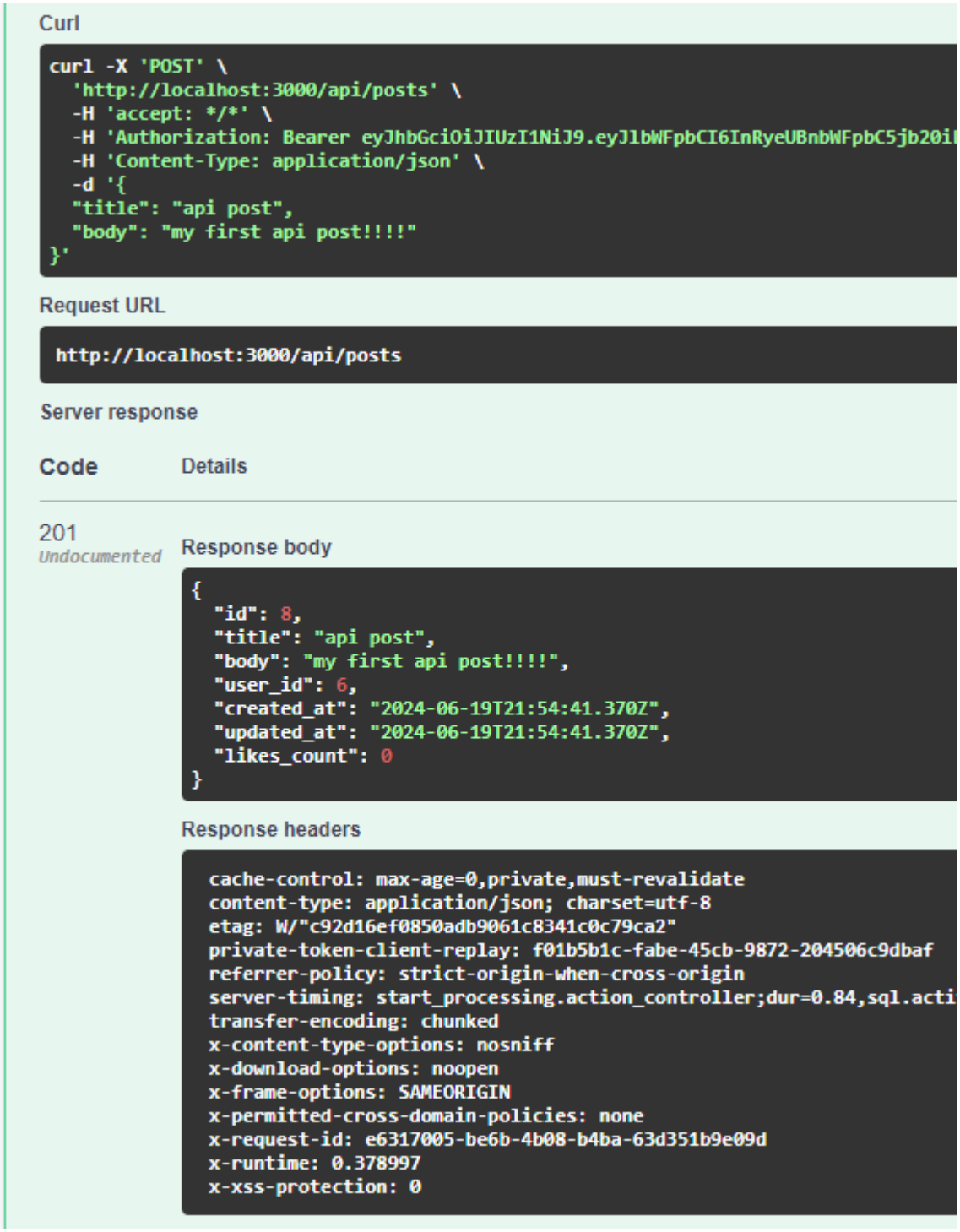

Рисунок 3.23 – Приклад створення поста за через API з JWT авторизацією

Хедер вебсайту має інформацію про поточного користувача, можливість зайти, вийти з профілю, зареєструватися та подивитися більше інформації про свій профіль.Main page – це головна сторінка вебсайту, на якій користувачі можуть побачити пости інших користувачів. Пошук дає можливість знайти пост або іншого користувача по назві або по імені (див. рис 3.24).

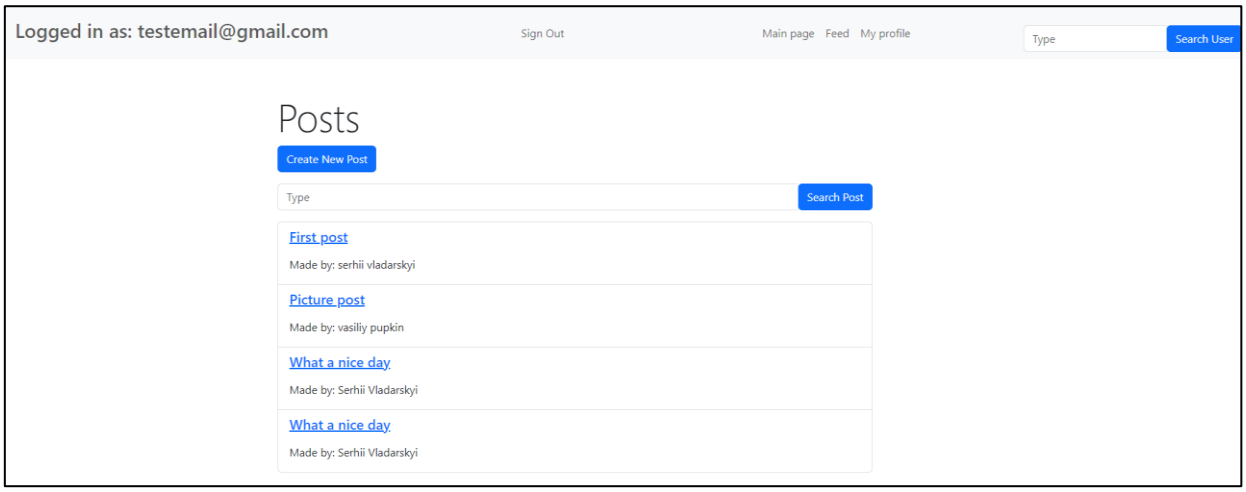

Рисунок 3.24 – Інтерфейс головної сторінки та хедеру вебсайту

Користувач може взаємодіяти з іншими користувачами через пости, коментарі та лайки (див. рис 3.25). Є можливість змінити або видалити свої пости та коментарі. Менеджери можуть видаляти записи інших користувачів.

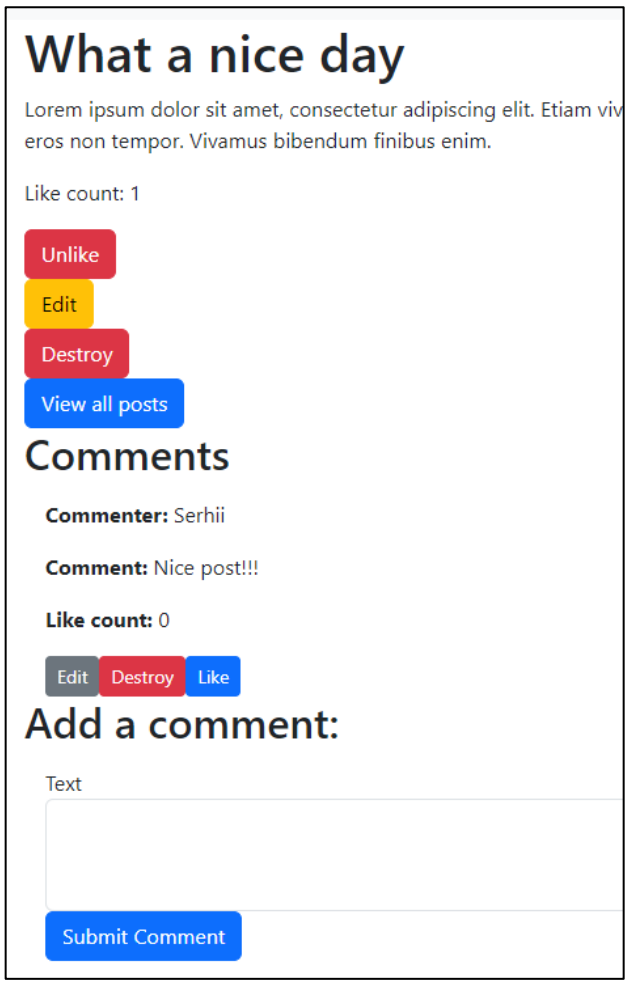

Рисунок 3.25 – Візуалізація функціоналу постів

Є функція пошуку постів та користувачів (див. рис 3.26). У профілі користувачів можна побачити інформацію про них, їх пости та підписатися на них (див. рис. 3.27).

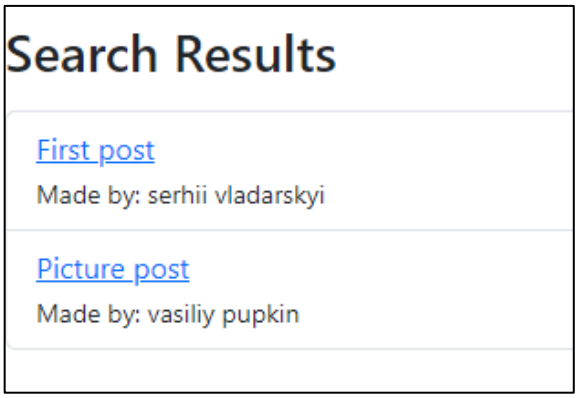

Рисунок 3.26 – Результати пошуку постів за запитом «post»

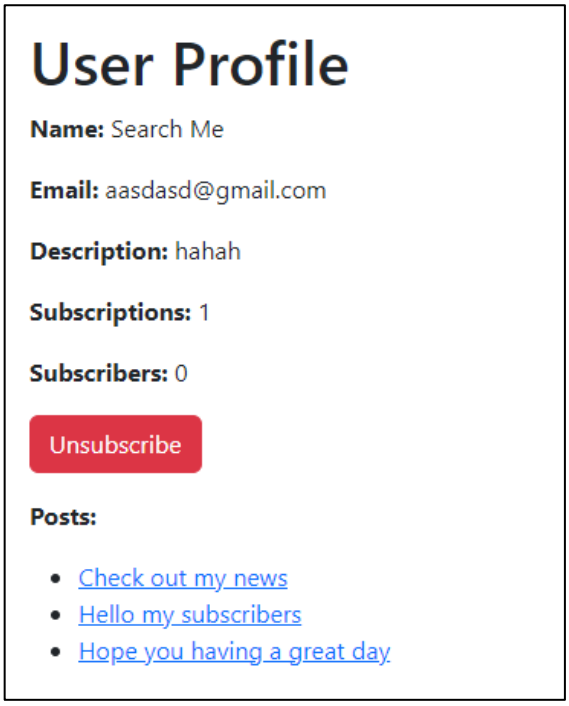

Рисунок 3.27 – профіль користувача

Якщо підписатися на користувача, у стрічці можна буде побачити їх пости (див. рис. 3.28).

Адмін панель доступна тільки адміністраторам. Там вони можуть адмініструвати пости, коментарі та користувачів (див. рис. 3.29). Також тут надаються ролі користувачам (див. рис. 3.30).

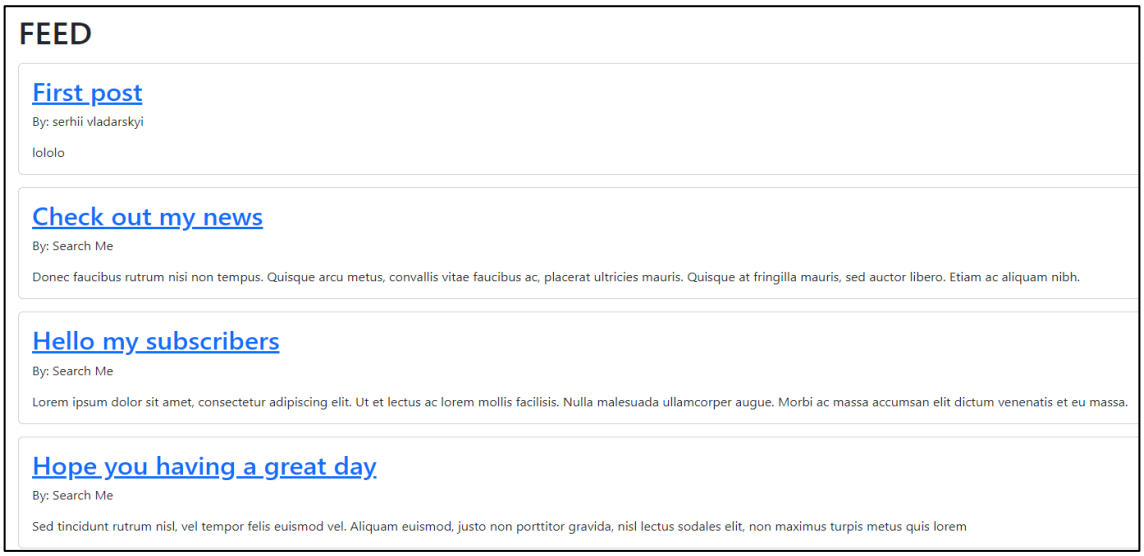

Рисунок 3.28 – Стрічка постів

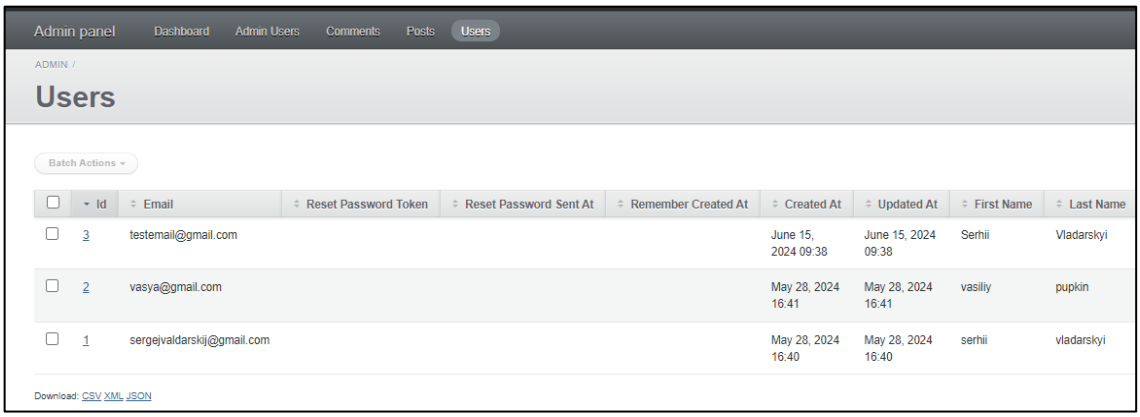

Рисунок 3.29 – Адмін панель

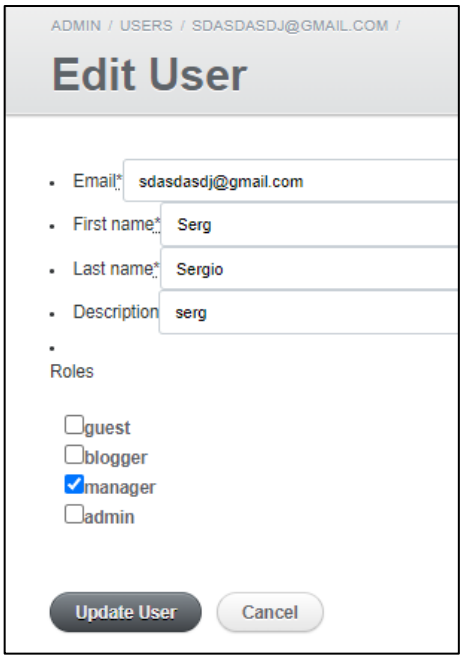

Рисунок 3.30 – Редагування користувача у панелі адміністратора

Усі файли завантажуються на AWS S3 (див. рис 3.31). AWS S3 (Amazon Simple Storage Service) – це масштабована, високошвидкісна вебслужба хмарного сховища, призначена для онлайн-резервного копіювання та архівування даних і програм у вебсервісах Amazon. Він забезпечує зберігання об’єктів через інтерфейс вебсервісу та може зберігати будь-який обсяг даних у будь-який час з будь-якого місця в Інтернеті.

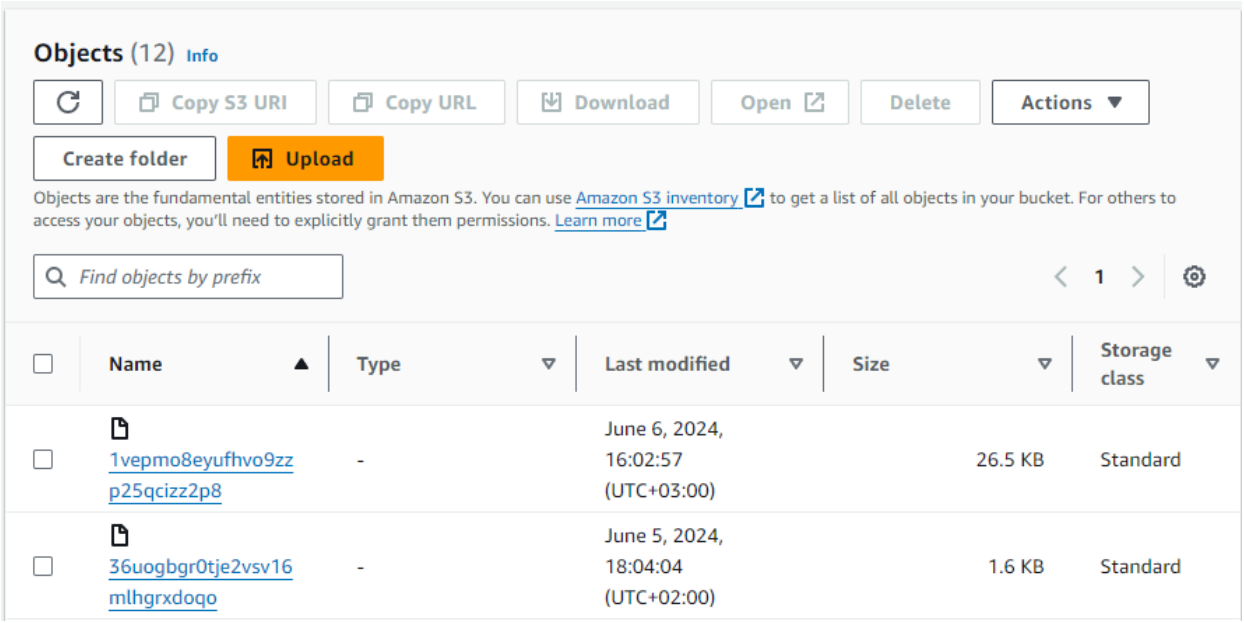

Рисунок 3.31 – Інформація про об’єкти у бакеті AWS S3

Бакет S3 схожий на каталог у хмарі S3, куди можна завантажити будь-яку кількість файлів. Кожен файл у бакеті є об’єктом. Кожен об’єкт ідентифікується унікальним ключем, призначеним користувачем.

Для використання AWS S3 було створено обліковий запис AWS та сегмент S3 за допомогою AWS SDK. AWS пропонує безкоштовний план якщо не перевищути кількість запитів та записів до бакету.

До проєкту було додано гем aws-sdk-s3 для взаємодії з сегментом S3. Налаштування відбувається у storage.yml (див. рис 3.32). Rails Active Storage потім може використовувати цю конфігурацію для завантаження файлів на S3. Важливою частиною конфігурації є використання ключів із Rails.application.credentials. Ці ключі лежать у спеціальному файлі, який

захищений за допомогою master.key, який був згенерований при ініціалізації проєкту. Цей алгоритм дозволяє використовувати секретні ключи та токени безпечно.

```
amazon:
  service: S3
  access_key_id: <%= Rails.application.credentials.dig(:aws, :access_key_id) %>
  secret_access_key: <%= Rails.application.credentials.dig(:aws, :secret_access_key) %>
  region: eu-central-1
  bucket: svdiploma-project-bucket
```

Рисунок 3.32 – Конфігурація AWS S3

Користувачі можуть обмінюватися повідомленнями у чатрумі (див. рис. 3.33). Оновлення йде в прямому ефірі за допомогою Stimulus та ActionCable.

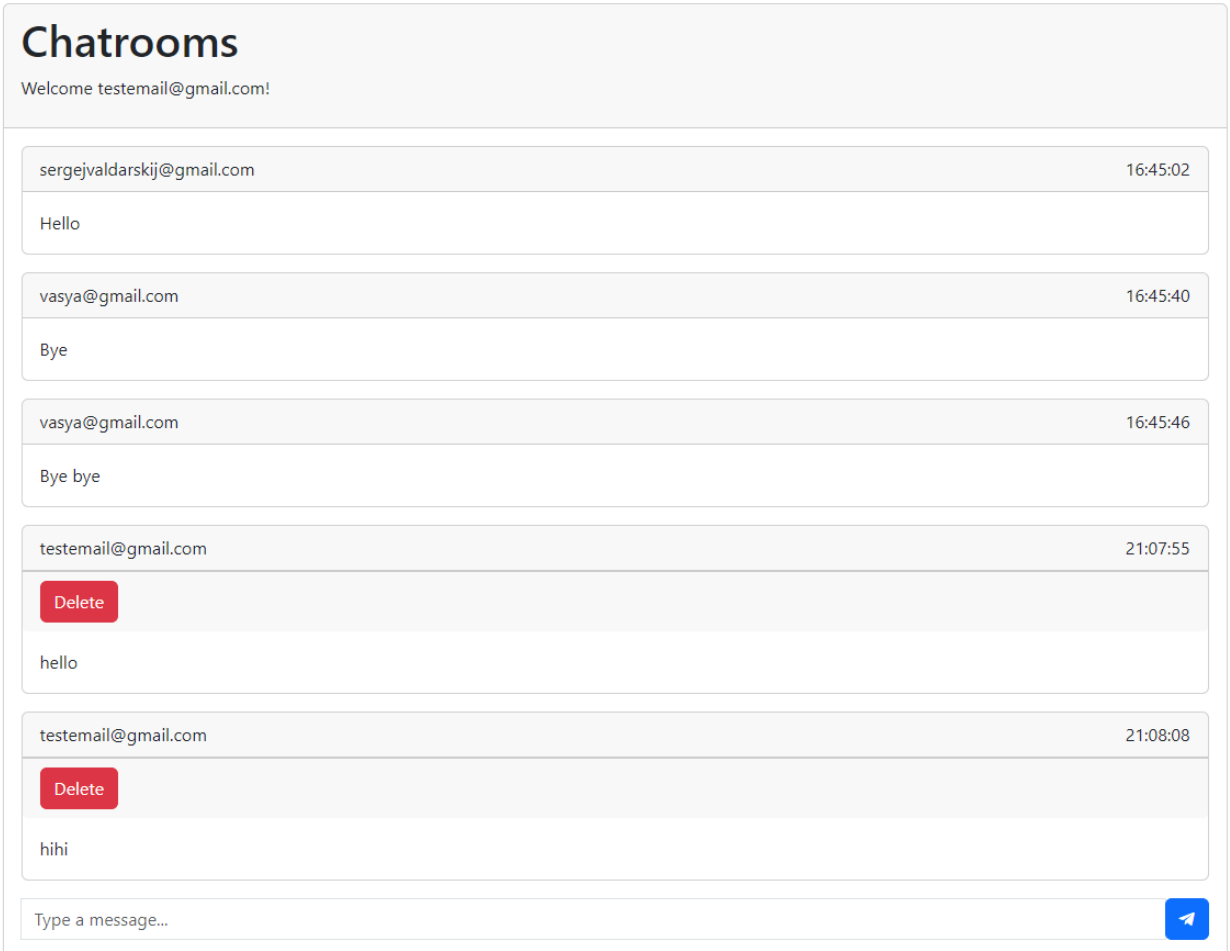

Рисунок 3.33 – Чат

3.4 Висновки до розділу 3

Було проаналізовано процес розробки, тестування та кінцевий результат соціальної мережі. Для розробки бекенду було використано Ruby on Rails та PostgreSQL. Для розробки користувацького інтерфейсу було використано erb, bootstrap та HotWire. Для тестування і документації було використано Rspec та Swagger.

ВИСНОВКИ

У результаті проведеної роботи було створено додаток «Соціальна мережа» на фреймворку Ruby on Rails.

Проєкт було розроблено як фулстак так і API додаток, тобто було створені ендпоінти з інтерфейсом, які генерують документ, так і з API, які відправляють дані json.

На етапі проєктування було створено та проаналізовано діаграму прецедентів та схему бази даних.

Було розроблено функціонал згідно з вимогами, що включали в себе пости, коментарі, лайки, стрічка новин, чат та пошук. Було додано багатофункціональну систему авторизації та автентифікації за допомогою Devise та JWT. Було розроблено адміністративний функціонал. Користувачі поділяються на ролі, мають свої дозволи, а адміністратор може керувати всім контентом вебсайту та користувачами.

Для розробки функціонально простого інтерфейсу було використано шаблонізацію за допомогою erb та bootstrap.

Весь код проєкту покрито тестами та документацією, що значно спростить подальше розширення функціоналу.

Розробка даного проєкту на Ruby on Rails показала, що цей фреймворк підходить для швидкої розробки невеликих проєктів та стартапів, оскільки поєднує в собі простоту синтаксису, налаштування та використання вбудованих рішень.

Проєкт досяг ланки MVP (мінімально життєздатний продукт). Його можна використовувати для подальшого розширення функціоналу та використання у комерційних цілях. Для покращення інтерфейсної частини проєкту рекомендується розробити додаток на фронтенд фреймворку (наприклад, React).

ПЕРЕЛІК ПОСИЛАНЬ

1. About Ruby. URL: https://www.ruby-lang.org/en/about/ (дата звернення: 19.02.2024).
2. Programming Ruby: The Pragmatic Programmer’s Guide. URL: https://ruby-doc.com/docs/ProgrammingRuby/ (дата звернення: 25.02.2024).
3. Ruby-Doc.org: Documenting the Ruby Language. URL: https://ruby-doc.org/ (дата звернення: 28.02.2024).
4. Karsh P. Deep Dive into Garbage Collection in Ruby 3. URL: https://patrickkarsh.medium.com/deep-dive-into-garbage-collection-in-ruby-3-incremental-garbage-collection-29775d48e253 (дата звернення: 01.03.2024).
5. Ruby on Rails Guides. URL: https://guides.rubyonrails.org/ (дата звернення: 04.03.2024).
6. Okolo D. Ruby on Rails: MVC and you! URL: https://dev.to/dumebii/model-view-controller-in-rails-a-deep-dive-into-the-mvc-architecture-4oi1 (дата звернення: 10.03.2024).
7. Layouts and Rendering in Rails. URL: https://guides.rubyonrails.org/layouts_and_rendering.html (дата звернення: 14.03.2024).
8. ActiveSupport: Concern – Rails API. URL: https://api.rubyonrails.org/v7.1.3.2/classes/ActiveSupport/Concern.html (дата звернення: 18.03.2024).
9. Rails core principles – Learn Ruby on Rails – BigBinary Academy. URL: https://courses.bigbinaryacademy.com/learn-rubyonrails/rails-core-principles/ (дата звернення: 22.03.2024).
10. Hotwire: HTML Over The Wire. URL: https://hotwired.dev/ (дата звернення: 26.03.2024).
11. What is the difference between Turbo and Stimulus, and what exactly is Hotwire? URL: https://www.ducktypelabs.com/turbo-vs-stimulus/ (дата

звернення: 26.03.2024).

12. Most used web frameworks among developers 2023. URL: https://www.statista.com/statistics/1124699/worldwide-developer-survey-most-used-frameworks-web/ (дата звернення: 26.03.2024).
13. Ruby on Rails is still relevant – here's why. URL: https://syndicode.com/blog/why-is-ruby-still-our-choice-in-2020-2/ (дата звернення: 27.03.2024).
14. What is Mastodon? URL: https://docs.joinmastodon.org/ (дата звернення: 28.03.2024).
15. Eustachy M. What is a Ruby implementation? URL: https://mathieu-eustachy.com/articles/what-is-a-ruby-implementation?locale=en (дата звернення: 13.03.2024).
16. Documentation – PostgreSQL. URL: https://www.postgresql.org/docs/ (дата звернення: 27.03.2024).
17. What Is MongoDB? URL: https://www.mongodb.com/company/what-is-mongodb (дата звернення: 28.03.2024).
18. MySQL 8.4 Reference Manual :: 1.2.1 What is MySQL? URL: https://dev.mysql.com/doc/refman/8.4/en/what-is-mysql.html (дата звернення: 28.03.2024).
19. PostgreSQL: About. URL: https://www.postgresql.org/about/ (дата звернення: 28.03.2024).