МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЗАПОРІЗЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

МАТЕМАТИЧНИЙ ФАКУЛЬТЕТ
Кафедра програмної інженерії

КВАЛІФІКАЦІЙНА РОБОТА БАКАЛАВРА

на тему: **«ПРОЕКТУВАННЯ ТА РОЗРОБКА ВЕБДОДАТКА ДЛЯ РОЗМІЩЕННЯ ОГОЛОШЕНЬ З ПОШУКУ ТА НАДАННЯ ВОЛОНТЕРСЬКОЇ ДОПОМОГИ З ВИКОРИСТАННЯМ ANGULAR ТА NODE.JS»**

Виконала: студентка 3 курсу, групи 6.1210-пі-с
спеціальності 121 інженерія програмного забезпечення
(шифр і назва спеціальності)
освітньої програми програмна інженерія
(назва освітньої програми)
А.О. Валяєва
(ініціали та прізвище)

Керівник доцент кафедри програмної інженерії, доцент, к.т.н. Лимаренко Ю.О.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Рецензент директор ТОВ «Голден-Тім» Калін М.В.
(посада, вчене звання, науковий ступінь, прізвище та ініціали)

Запоріжжя – 2023

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)

“ 07 ” 02 2023 р.

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТЦІ

Валяєвій Анні Олегівні
(прізвище, ім’я та по-батькові)

1. Тема роботи Проектування та розробка вебдодатка для розміщення оголошень з пошуку та надання волонтерської допомоги з використанням Angular та Node.js

керівник роботи Лимаренко Юлія Олексіївна, к.т.н., доцент
(прізвище, ім’я та по-батькові, науковий ступінь, вчене звання)

затверджені наказом ЗНУ від « 26 » січня 2023 року № 102-с

2. Строк подання студентом роботи 07.06.2023 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. Проектування та реалізація програмного забезпечення.

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

6. Консультанти розділів роботи

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

7. Дата видачі завдання 07.02.2023 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.	08.02.2023	
2.	Збір вихідних даних.	15.02.2023	
3.	Обробка методичних та теоретичних	22.02.2023	
	джерел.		
4.	Розробка першого та другого розділу.	05.04.2023	
5.	Розробка третього розділу.	10.05.2023	
6.	Оформлення та нормоконтроль	01.06.2023	
	кваліфікаційної роботи бакалавра.		
7.	Захист кваліфікаційної роботи.	23.06.2023	

Студент _____ А.О. Валяєва
(підпис) (ініціали та прізвище)

Керівник роботи _____ Ю.О. Лимаренко
(підпис) (ініціали та прізвище)

Нормоконтроль пройдено

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

РЕФЕРАТ

Кваліфікаційна робота бакалавра «Проектування та розробка вебдодатка для розміщення оголошень з пошуку та надання волонтерської допомоги з використанням Angular та Node.js»: 59 с., 19 рис., 2 табл., 13 джерел.

АВТЕНТИФІКАЦІЯ, АВТОРИЗАЦІЯ, БАЗА ДАНИХ, ВЕБДОДАТОК, КЛІЄНТ-СЕРВЕРНА АРХІТЕКТУРА, ОБ'ЄКТНО-РЕЛЯЦІЙНЕ ВІДОБРАЖЕННЯ, ТОКЕН, ANGULAR, NODE.JS, POSTGRESQL.

Об’єкт дослідження – основні методи та засоби розробки вебдодатків.

Мета роботи: розробка вебдодатка для розміщення оголошень з пошуку та надання волонтерської допомоги.

Метод дослідження – методи збору та аналізу вимог до програмного забезпечення, методи моделювання та проектування програмного забезпечення.

У кваліфікаційній роботі було наведено аналіз предметної області, огляд існуючих систем з предметної області та описано засоби для створення вебдодатків. На основі даних теоретичних відомостей розроблено проект вебдодатка для розміщення оголошень з пошуку та надання волонтерської допомоги. Результатом виконання кваліфікаційної роботи є вебдодаток з клієнт-серверною архітектурою, написаний, використовуючи PostgreSQL, Node.js та Angular.

SUMMARY

Bachelor’s qualifying paper «Design and Development of a Web-resource for Searching and Providing Volunteer Assistance using Angular and Node.js»: 59 pages, 19 figures, 2 tables, 13 references.

ANGULAR, AUTHENTICATION, AUTHORIZATION, CLIENT-SERVER ARCHITECTURE, DATABASE, NODE.JS, OBJECT-RELATIONAL MAPPING, POSTGRESQL, TOKEN, WEB APPLICATION.

The object of the study is main methods and tools of developing web applications.

The aim of the study is development of a web-resource for searching and providing volunteer assistance.

The methods of research are collecting and analyzing software requirements, methods of modeling and designing software.

The qualification paper provided an analysis of the subject area, an overview of existing systems in the subject area, and described tools for creating web applications. On the basis of these theoretical information, a project of a web application for searching and providing volunteer assistance was developed. The result of the qualification work is a web application with a client-server architecture, developed using PostgreSQL, Node.js and Angular.

ЗМІСТ

ВСТУП

Волонтерська допомога залишається актуальною і важливою в сучасному світі. Створення вебдодатка для організації волонтерської допомоги є дуже актуальним і корисним завданням.

Додаток для організації волонтерської допомоги може залучати і координувати волонтерів, надаючи їм зручний спосіб перегляду доступних завдань. Волонтери можуть легко знайти потрібну допомогу і долучитися до проектів. Додаток також може полегшити процес управління волонтерськими проектами. Організатори можуть створювати та оновлювати завдання, встановлювати терміни виконання, відстежувати прогрес, надавати звіти тощо. Додаток може забезпечувати зручні інструменти для комунікації.

Отже, у якості виконання кваліфікаційної роботи було вирішено створити вебдодаток для організації волонтерської допомоги.

Перший розділ кваліфікаційної роботи містить характеристику предметної області. Наведені два приклади додатків для організації волонтерства, описаний їхній функціонал. Описаний стек технологій, за домопогою якого можна розробити вебдодаток: PostgreSQL, Node.js, Angular.

У другому розділі описано технічне завдання, призначення розробки та технічні вимоги до програмного продукту. Наведено діаграму варіантів використання, ER-діаграму бази даних. Описані етапи створення вебдодатка та наведені макети застосунку.

Третій розділ містить особливості реалізації додатка. Описано ініціалізацію та налаштування проектів серверної та клієнтської частин. Наведений опис процесу створення бази даних PostgreSQL та роботи з нею за допомогою об'єктно-реляційного відображення Prisma. Описано процес розробки серверної частини, використовуючи платформу Node.js, фреймворк Express.js та супутні зовнішні пакети. Описано процес розробки клієнтської частини, використовуючи фреймворк Angular та супутні йому бібліотеки.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ОБ'ЄКТУ ДОСЛІДЖЕННЯ

1.1 Характеристика предметної області

Волонтерські платформи – це онлайн-ресурси, які з'єднують волонтерів з організаціями або окремими людьми, що потребують волонтерської допомоги. Вони забезпечують зручну платформу для взаємодії, спілкування та обміну інформацією між волонтерами та людями, що потребують допомоги.

Основна мета волонтерських платформ – взаємодія волонтерів з волонтерськими проектами, що допомагає забезпечити ефективну роботу волонтерського сектору.

Волонтерські платформи забезпечують пошук волонтерських можливостей. Ці системи надають зручні інструменти для пошуку волонтерських проектів, де волонтери можуть шукати волонтерські потреби за різними критеріями, такими як місце розташування, галузь діяльності, тривалість проекту тощо.

Також платформи для волонтерів забезпечують реєстрацію та зберігання профілів волонтерів. Волонтери можуть створювати свої профілі на платформі, вказуючи свої навички, інтереси та досвід. Це допомагає організаціям знайти потрібних волонтерів для своїх проектів.

Волонтерські онлайфн-платформи можуть мати функціонал зворотного зв'язку. Волонтери можуть залишати відгуки та оцінки проектів та організації, з якими вони співпрацювали. Це надає корисну інформацію для інших волонтерів при виборі проектів.

Також додатки для волонтерів допомогають з комунікацією та обміном інформацією між волонтерами та організаціями чи окремими людьми, що потребують допомоги.

Волонтерські платформи є важливим інструментом для організації та

підтримки волонтерської діяльності. Існує кілька причин, чому волонтерські платформи є корисними.

Волонтерські онлайн-платформи надають зручний доступ до волонтерських можливостей. Вони мають зручну централізовану базу даних, де організації можуть розміщувати свої проекти, а потенційні волонтери можуть знайти і обрати проекти, які відповідають їхнім інтересам і навичкам. Це спрощує процес пошуку та залучення волонтерів.

Волонтерські платформи сприяють розширенню потенційної аудиторії, залучаючи волонтерів з усього світу, що дозволяє організаціям залучити більше зацікавлених та різнобічних волонтерів. Це дозволяє розширити географію волонтерської діяльності та залучити людей з різних культур та фахових галузей.

Онлайн-платформи спрощують процес організації. Вони надають інструменти для ефективного управління волонтерськими проектами, такі як системи реєстрації, комунікації з волонтерами, планування розкладу та координації завдань. Це допомагає організаціям ефективно організовувати та керувати волонтерською роботою.

Покращується безпека та надійсніть взаємодії, адже платформи можуть мати механізми перевірки організацій та волонтерів, а також системи відгуків та рейтингів, що допомагають забезпечити безпеку та якість волонтерських проектів. Вони також можуть надавати детальну інформацію про проекти, умови та інші важливі деталі, що сприяють надійності та прозорості.

Також волонтерські онлайн-додатки створюють спільноти волонтерів, де люди можуть обмінюватися досвідом та ідеями. Це стимулює співпрацю, взаємодопомогу та розвиток волонтерського руху.

Отже, волонтерські платформи сприяють підтримці та розвитку волонтерської діяльності, забезпечуючи зручність, доступність, безпеку та ефективність процесу організації та участі у волонтерських проектах.

1.2 Огляд існуючих волонтерських платформ

1.2.1 Вебдодаток організації «Українська Волонтерська Служба»

Українська Волонтерська Служба (УВС) – це незалежна громадська організація, що займається координацією та підтримкою добровольчої діяльності в Україні. Організація була створена з метою надання допомоги людям, які потребують допомоги внаслідок конфлікту, кризових ситуацій, природних лих чи інших негативних подій. УВС залучає та координує волонтерів з різних сфер життя, які готові надавати допомогу в усіх регіонах України.

Діяльність організації охоплює такі напрямки, як: гуманітарна допомога, підтримка уразливих груп населення, волонтерська діяльність та розвиток громадськості. Організація спирається на доброчесність, відданість та проактивність своїх учасників, а також на підтримку громадськості та партнерів для реалізації своїх проектів. Волонтери УВС виявляються в різних сферах життя та діяльності, всі вони разом створюють сильну команду з метою підтримки та розвитку України. УВС активно співпрацює з іншими волонтерськими організаціями, державними установами, місцевими органами влади та іншими зацікавленими сторонами з метою спільної реалізації проектів та програм для благодійних цілей.

Українська Волонтерська Служба (УВС) має на своєму вебдодатці різноманітний функціонал, який допомагає волонтерам та організаціям знаходити та здійснювати волонтерські проекти, спілкуватися та отримувати необхідну інформацію.

Даний вебдодаток має механізми перевірки організацій та волонтерів, а також системи відгуків та рейтингів, що допомагають забезпечити безпеку та якість волонтерських проектів. Вони також надають детальну інформацію про проекти, умови та інші важливі деталі, що сприяють надійності та прозорості взаємодії.

Користувачі можуть створити обліковий запис на сайті та увійти в свій особистий кабінет, де вони матимуть можливість управляти своїми профілями, переглядати і редагувати інформацію про себе, налаштовувати свої волонтерські інтереси та знаходити проекти, що відповідають їхнім потребам.

На онлайн-платформі УВС доступні фільтри та пошукова система, що дозволяє користувачам знаходити волонтерські проекти за категоріями, регіонами, часом тривалості тощо. Користувачі можуть переглядати деталі проектів, такі як: опис, місце, дати, вимоги до волонтерів тощо, і вибирати ті проекти, в яких вони бажають взяти участь.

Додаток УВС може надає можливості для спілкування та обміну інформацією між волонтерами та тими, хто потребує допомоги. Це забезпечується системою приватних повідомлень, де користувачі можуть обговорювати проекти, задавати питання, ділитися досвідом та планувати спільну роботу.

На платформі розміщений календар подій, де відображаються дати та деталі майбутніх волонтерських заходів, тренінгів, зустрічей тощо. Це допомагає волонтерам бути в курсі актуальних подій та зареєструватися на участь.

Також додаток має функціонал, який дозволяє волонтерам залишати оцінки та відгуки про проекти, в яких вони брали участь. Це допомагає іншим користувачам прийняти рішення щодо участі у конкретному проекті та забезпечує зворотній зв’язок для організацій.

Це основні функціональні можливості, присутні на онлайн-платформі організації «Українська Волонтерська Служба».

1.2.2 Волонтерська онлайн-платформа «Volunteer World»

Волонтерська онлайн-платформа «Volunteer World»Volunteer World – це онлайн-платформа, яка спеціалізується на знаходженні та підборі волонтерських проектів з усього світу. Ця платформа надає можливість шукати та порівнювати

різні волонтерські програми за різними критеріями, такими як місце розташування, галузь діяльності, тривалість проекту та багато іншого.

Volunteer World пропонує детальну інформацію про кожен волонтерський проект, включаючи опис проекту, вимоги до волонтерів, розклад роботи, вартість та умови проживання, а також відгуки від попередніх волонтерів. Користувачі можуть зареєструватися в системі, створити свій профіль та заявки на участь у волонтерських проектах. Користувачі можуть зберігати свої персональні дані, історію участі в проектах та управляти своїми заявками на волонтерські проекти.

На даній онлайн-платформі користувачі можуть шукати волонтерські проекти за різними критеріями, такими як місце розташування, галузь діяльності, тривалість проекту тощо. Це дозволяє знайти відповідну волонтерську можливість в будь-якій частині світу.

Для кожного волонтерського проекту надається детальний опис, вимоги до волонтерів, тривалість, розклад роботи та умови. Це допомагає користувачам зробити інформований вибір та знайти проект, який найкраще відповідає їхнім можливостям і інтересам.

Volunteer World має систему оглядів та рейтингів, де волонтери можуть залишати свої відгуки про проекти, в яких вони брали участь. Це дозволяє іншим користувачам отримати об’єктивну інформацію про якість та досвід волонтерства в конкретному проекті.

Volunteer World надає підтримку користувачам шляхом надання відповідей на запитання, консультацій щодо волонтерських можливостей та допомоги з процесоми подання на волотерські проекти.

Отже, онлайн-платформа Volunteer World сприяє популяризації волонтерства, забезпечує зручний спосіб зв’язку між волонтерами та організаціями та допомагає волонтерам знайти проекти, які відповідають їхнім інтересам та потребам.

1.3 Засоби для створення вебдодатка

На початку розробки проекту вебдодатка для розміщення оголошень з пошуку та надання волонтерської допомоги постало питання вибору стеку технологій для його створення.

Було вирішено розробити SPA. SPA (Single Page Application) – вебдодаток, для якого весь контент завантажується при першому завантаженні сторінки, і який не перезавантажує сторінки при взаємодії з користувачем [7]. Замість цього, SPA динамічно оновлює вміст сторінки після виконання AJAX-запитів, роблячи ререндеринг потбріних елементів на клієнтському інтерфейсі. SPA забезпечує швидку і зручну навігацію без потреби перезавантаження сторінки при кожній взаємодії з користувачем. SPA є популярним підходом до розробки вебдодатків, оскільки дозволяє забезпечити більш рівномірний досвід користувача та зменшити навантаження на сервер. Серед недоліків SPA – погана SEO оптимізація, оскільки пошукові системи не можуть аналізувати сторінки, яких немає у статичному вигляді.

На противагу SPA, існує Multi Page Application – це вебдодаток, у якому кожна сторінка кожен раз повністю завантажується та генерується на серверній частині [7]. У цьому випадку користувачі переходять з однієї сторінки на іншу, оновлюючи вміст сторінки при кожному переході.

В порівнянні з SPA, MPA може мати менший рівень взаємодії з користувачем і зазвичай потребує більшого обсягу даних, що передаються між сервером та клієнтом, за рахунок багатьох перезавантажень.

Отже, розробку проекту було вирішено провести використовуючи наступний стек технологій:

- об’єктно-реляційна система керування базами даних PostgreSQL;
- платформу Node.js, фреймворк Express.js, об’єктно-реляційне відображення Prisma та допоміжні зовнішні Node.js пакети;
- фреймворк Angular з супутнім йому стеком.

1.3.1 Огляд СУБД PostgreSQL

PostgreSQL – це потужна об’єктно-реляційна система керування базами даних (ОРСКБД), яка є однією з найбільш розповсюджених та популярних систем у світі. PostgreSQL є відкритим програмним забезпеченням та надає високу продуктивність, надійність та безпеку для зберігання та управління даними.

Використання бази даних PostgreSQL має свої переваги. PostgreSQL є потужною та розширюваною системою управління базами даних, яка може працювати з великими обсягами даних та підтримує багато функцій, включаючи транзакції, засоби безпеки та резервне копіювання [5].

Окрім цього, база даних PostgreSQL є безкоштовною та є продуктом з відкритим кодом, що дозволяє розробникам створювати ефективні та надійні додатки без великих витрат на ліцензії та підтримку.

Також PostgreSQL має підтримку багатьох різних типів даних, таких як числа, рядки, дати, географічні дані, зображення та багато інших. Більше того, дана система управління базою даних надає можливості розширення за допомогою різних типів модулів та розширень, що дозволяє легко налаштувати базу даних під власні потреби [5]. Це дозволяє розробникам створювати складні додатки з різноманітними типами даних та використовувати їх для різних цілей, включаючи аналіз даних та створення звітів.

PostgreSQL забезпечує високу продуктивність та швидкість роботи з великими обсягами даних.

PostgreSQL є дуже надійною системою та має ряд вбудованих механізмів, які забезпечують захист даних від втрати та пошкодження. PostgreSQL має різні вбудовані механізми безпеки, такі як шифрування, аутентифікація та авторизація, що дозволяють зберігати дані в безпеці [5].

Крім того, PostgreSQL має велику та активну спільноту користувачів та розробників, які надають допомогу та підтримку у вирішенні проблем та удосконаленні роботи з базою даних.

Хоча PostgreSQL є одним з найпопулярніших та найпотужніших реляційних СУБД, як і будь-який інший продукт, він має свої недоліки. Ось деякі з них:

– PostgreSQL може бути складним для встановлення та налаштування, може вимагати досить потужного обладнання для роботи, зокрема, якщо ведеться робота з великими обсягами даних;
– високі витрати ресурсів на підтримку та розробку – залежно від конфігурації та потреб користувачів, розробка та підтримка PostgreSQL може вимагати великих зусиль розробників і, як наслідок, коштів;
– PostgreSQL має меншу кількість користувачів порівняно з деякими іншими конкурентами, такими як MySQL, що може ускладнити пошук ресурсів для підтримки та розвитку.

Отже, використання системи управління базою даних PostgreSQL є гарним вибором для створення волонтерської платформи у вигляді вебдодатка.

1.3.2 Огляд платформи Node.js

Node.js – це відкрите, кросплатформне середовище виконання JavaScript, яке дозволяє виконувати код JavaScript на сервері [2]. Він базується на двигуні JavaScript V8 від Google та забезпечує можливість побудови швидких та масштабованих додатків на стороні сервера [2]. Node.js дозволяє розробникам використовувати JavaScript як мову програмування для створення серверних додатків, які можуть бути відповідальні за обробку запитів, доступ до баз даних, роботу з мережевими протоколами тощо.

Node.js може використовуватися для створення широкого спектру додатків на стороні сервера, включаючи:

– створення веб-серверів та API: Node.js дозволяє створювати швидкі та ефективні веб-сервери та API з мінімальним навантаженням на ресурси сервера;

- роботу з базами даних: Node.js забезпечує можливість взаємодії з різноманітними базами даних, такими як PostgreSQL, MongoDB та MySQL;
- розробку чат-ботів та інших додатків для обробки повідомлень: платформа надає можливість створювати чат-боти та інші додатки, що обробляють повідомлення;
- створення інструментів командного рядка: Node.js дозволяє створювати інструменти командного рядка, такі як npm, що дозволяють взаємодіяти з сервером через командний рядок;
- розробку розширень для браузерів: Node.js може використовуватися для розробки розширень для браузерів, які можуть допомогти в роботі з веб-сторінками.

Це не повний список використання Node.js, оскільки його можна використовувати для багатьох інших завдань на стороні сервера, що вказує на гнучкість та широкі можливості технології.

Node.js також дозволяє використовувати npm (Node Package Manager). Це менеджер пакетів для JavaScript, який дозволяє розробникам легко встановлювати та керувати залежностями свого проекту [2]. npm забезпечує доступ до мільйонів пакетів, що дозволяє використовувати вже наявний код, щоб прискорити розробку та зменшити кількість коду, який потрібно писати з початком кожного проекту. Основні функції npm:

- управління залежностями: npm дозволяє розробникам встановлювати та оновлювати залежності свого проекту;
- пошук пакетів: npm забезпечує доступ до мільйонів пакетів, що дозволяє розробникам швидко знайти необхідний пакет;
- версіювання пакетів: npm дозволяє розробникам вказувати версію пакету, який потрібно встановити, що дозволяє забезпечити сумісність між різними версіями пакетів;
- публікація пакетів: npm дозволяє розробникам публікувати свої пакети в репозиторії, що дозволяє іншим розробникам використовувати їх;

– створення скриптів: npm дозволяє розробникам створювати скрипти для автоматизації рутинних задач.

Пакетний менеджер npm є важливим інструментом для JavaScript розробників, який дозволяє прискорити розробку, підвищити якість коду та зменшити кількість зусиль, необхідних для управління залежностями проекту.

Node.js використовує події-орієнтовану та неблокуючу архітектуру, що дозволяє йому працювати з багатьма запитами одночасно та забезпечує високу продуктивність додатків [2]. Неблокуюча архітектура Node.js – це підхід, при якому виконання коду не зупиняється при очікуванні відповіді з інших частин системи [2]. В зв’язку з тим, що Node.js є однопотоковим і події-орієнтованим середовищем, він використовує асинхронні запити до інших системних ресурсів, таких як бази даних, мережеві запити, файлова система, що дозволяє іншим частинам системи продовжувати роботу, тоді як Node.js чекає на відповідь. Це робить Node.js особливо ефективним при роботі з багатопоточними операціями, де потрібно обробляти великі об’єми даних. Такий підхід дозволяє досягти високої продуктивності і масштабованості Node.js додатків.

Express.js – це популярний фреймворк для розробки додатків на Node.js. Він дозволяє швидко створювати додатки, забезпечуючи базовий функціонал, такий як маршрутизація, обробка запитів і відповідей, підтримка різних видів HTTP-запитів, робота з шаблонами, підтримка проміжного програмного забезпечення і багато іншого [3]. Крім того, він має велику спільноту розробників, що робить його популярним вибором для створення серверних програм на платформі Node.js.

Prisma – це сучасний ORM (Object-Relational Mapping) для роботи з базами даних у додатках. Prisma надає зручний спосіб взаємодії з базами даних, допомагаючи забезпечити швидкий доступ до даних та автоматично генерувати SQL-запити. Основна особливість Prisma полягає у використанні декларативного DSL (Domain-Specific Language), що дозволяє описувати моделі даних, відношення між ними та зв’язки [4]. Prisma забезпечує автоматичну генерацію SQL-запитів на основі цих описів, що робить роботу з базою даних простішою

та ефективнішою. Prisma підтримує різні типи баз даних, включаючи PostgreSQL, MySQL та SQLite. Інструмент надає розширені можливості, такі як міграції баз даних, управління схемою, підтримка транзакцій та оптимізація запитів [4]. Prisma також надає можливості міграцій баз даних, що дозволяє зручно змінювати схему бази даних без втрати даних. Prisma є популярним вибором для ORM у розробці додатків на платформі Node.js. Також є можливість використовувати Prisma Client, що є генерованим автоматично клієнтом, для виконання запитів до бази даних у Node.js додатка з використанням простого та типобезпечного API.

Отже, використання платформи Node.js, фреймворку Express.js та супутніх допоміжних пакетів є гарним вибором для розробки серверної частини системи.

1.3.3 Огляд фреймворку Angular

Angular – фреймворк для створення застосунків для різних платформ (веб, мобільні засоби, десктоп), розроблена корпорацією Google [8]. Він базується на мові програмування TypeScript та пропонує розробникам широкий спектр інструментів та функцій для створення додатків.

Angular забезпечує механізм для створення компонентів та модулів, що дозволяє розбити додаток на невеликі, повторно використовувані елементи. Він також пропонує розширену систему директив, що дозволяє вбудовувати додаток в розмітку та надавати взаємодію між користувачем та системою. Angular також надає розширену систему обробки подій, що дозволяє взаємодіяти з користувачем шляхом обробки подій.

Angular також підтримує різні підходи до роботи з сервером, такі як HTTP-запити та робота з WebSocket. Він має вбудований механізм роутингу, що дозволяє розробникам зручно обробляти URL-адреси.

Однією з особливостей Angular є його потужна система тестування, яка дозволяє розробникам створювати автоматичні тести для проектів, перевіряючи

роботу окремих одиниць кодової бази та додаток повністю. Angular підтримує unit та end-to-end тести, інструменти Jasmine и Protractor.

Dependency Injection (DI) – це патерн програмування, який дозволяє використовувати об’єкти, не вказуючи їх прямо, а передаючи відповідальність за створення цих об’єктів спеціальному механізму – контейнеру DI [8]. Angular має вбудовану систему DI, яка забезпечує зручний і простий спосіб керування залежностями в програмі. У Angular DI використовується провайдер (provider), який є конфігураційним об’єктом, що описує, який об’єкт необхідно створити та налаштування його створення. Провайдер може бути створений в різних місцях, наприклад, в компоненті, модулі або сервісі, і може бути використаний в будь-якому місці, де необхідна відповідна залежність [8].

RxJS (Reactive Extensions for JavaScript) – це бібліотека для програмування на JavaScript, яка дозволяє працювати з асинхронними потоками даних і подій з використанням принципів реактивного програмування [9]. RxJS дає можливість композиційної обробки потоків даних з використанням функціонального програмування, де дані розглядаються як потоки подій, які можна обробляти за допомогою операторів. В основі бібліотеки лежить шаблон Observable, як наслідок, бібліотека використовує три сутності – Observable, Observer та Subscriber [10]. Бібліотека RxJS є вбудованою в Angular, що дозволяє працювати зі спостережуваними об’єктами і операторами. Використання RxJS в Angular дозволяє ефективно керувати потоками даних, фільтрувати, групувати і об’єднувати потоки даних.

State management (управління станом) – це підхід до організації додатків, що полягає в тому, щоб зберігати стан додатка в одному місці та забезпечувати єдиний доступ до цього стану з усього додатка [8].

NgRx – це бібліотека стану для Angular, яка базується на принципах Redux [11]. Вона надає засоби для управління станом додатка та спрощує роботу зі складними даними та асинхронними операціями. Основна ідея NgRx полягає в тому, що стан додатка представляється у вигляді ієрархії об’єктів, які називаються store [11]. Кожен об'єкт store містить певний набір даних та функцій,

які дозволяють змінювати ці дані. Для зміни даних в store використовуються дії (actions) та редуктори (reducers). Дії – це об’єкти, які містять тип дії та необов’язкові дані для зміни стану. Редуктори – це чисті функції, які приймають поточний стан та дію і повертають новий стан. NgRx дозволяє підписуватися на зміни даних в store та реагувати на ці зміни, що спрощує роботу зі складними станами та асинхронними операціями.

Zone.js – це бібліотека для JavaScript, яка дозволяє відстежувати і контролювати виконання асинхронних операцій в програмі [8]. В Angular Zone.js використовується для реалізації механізму зон – це механізм, який дозволяє встановлювати контекст виконання для коду в Angular. За допомогою зон Angular може відслідковувати всі асинхронні операції і знати, в якому контексті виконання ці операції виконуються. Також Zone.js дозволяє виконувати в Angular операції в асинхронному режимі, не блокуючи потік виконання програми. Він дозволяє динамічно створювати зони, що дозволяє виконувати додаток в різних контекстах виконання.

Angular базується на архітектурі компонентів. Він не використовує традиційну архітектуру Model-View-Controller (MVC), а замість цього пропонує архітектуру, яка називається Model-View-ViewModel (MVVM) [8]. У архітектурі MVVM, Model представляє дані, з якими взаємодіє користувач. View – це представлення даних для користувача, а ViewModel – це посередник між Model і View. ViewModel приймає дані з Model, і вони можуть бути оброблені із використанням різних логічних операцій, після чого він передає оброблені дані до View для відображення. Angular заснований на компонентах, які є модульними. Компоненти Angular виконують функції View та ViewModel, дозволяючи користувачам розробляти додатки шляхом розділення функціоналу на придатний для повторного використання код. Архітектура Angular, заснована на компонентах, дозволяє швидко реагувати на зміни даних, робить простішою підтримку коду і робить розробку додатків більш продуктивною.

Angular і AngularJS – це дві різні версії фреймворка, розробленого компанією Google для розробки інтерфейсів. AngularJS був випущений в 2010

році, в той час як Angular вийшов у 2016 році і є повністю переробленим фреймворком на основі TypeScript [8]. Основна різниця між Angular і AngularJS полягає у підході до розробки додатків. AngularJS заснований на попередніх підходах до розробки користувацького інтерфейсу, що припускає використання jQuery та більш простих технологій. Angular же пропонує більш продуману і структуровану архітектуру.

Отже, використовувати платформу Angular для розробки сучасного інтерактивного вебдодатка є вдалим рішенням.

2 РОЗРОБКА ПРОЕКТУ

2.1 Технічне завдання

Перед початком створення програмної реалізації вебдодатка для розміщення оголошень з пошуку та надання волонтерської допомоги потрібно сформувати технічне завдання, технічні вимоги до додатка та етапи його розробки.

2.1.1 Найменування і область застосування

Програмний продукт, що розробляється, отримує найменування: «Вебдодаток для розміщення оголошень з пошуку та надання волонтерської допомоги».

Програма призначена для автоматизації взаємодії людей, що надають допомогу з людьми, що її потребують.

2.1.2 Призначення розробки

Даний проект призначений для вирішення наступних завдань:

- реєстрація, вхід та вихід з системи;
- редагування власного профілю;
- відображення інших профілів;
- додавання інших профілів в власний список для відстежування;
- написання та публікація оголошення;
- відображення стрічки з оголошеннями;
- можливість ставити вподобання оголошенням;

– написання коментарів під оголошеннями;
– отримання нотифікацій про певні події.

2.1.3 Технічні вимоги до програмного продукту

На початку роботи над додатом були сформовані наступні технічні вимоги:

– додаток має бути SPA (Single Page Application) типу;
– навігацію та взаємодію з сервером має бути реалізованою за допомогою технології AJAX (Asynchronous JavaScript And XML) без перезавантаження сторінки;
– модуль аутентифікації має бути реалізований за допомогою JWT (JSON Web Token), використовувати систему з access- і refresh-токенів;
– сторінки додатка мають бути захищені від користувачів, що не авторизувалися;
– серверна частина повинна відповідати набору правил REST;
– сервер має віддавати потрібні коди відповідей;
– має бути присутня валідація на веб-інтерфейсі: всі форми, створені у додатка, мають бути провалідованими, має відображатися відповідний інтерфейс, виконання запитів при некоректно введених даних має бути неможливим;
– UI-частина додатка має бути розроблена з використанням однієї з UI-kit бібліотек для Angular;
– всі сторінки додатка мають бути у єдиному стилі;
– має бути реалізований модуль користувачів: кожен зареєстрований користувач має мати профіль, можливість його редагувати та взаємодіяти з профілями інших користувачів;
– має бути реалізований модуль оголошень: кожен зареєстрований користувач має мати можливість опублікувати оголошення та взаємодіяти своїми оголошеннями та оголошеннями інших користувачів;

– має бути реалізований модуль нотифікацій: користувач має отримувати нотифікації про певні події, модуль має бути розроблений використовуючи протокол WebSocket.

2.2 Етапи створення вебдодатка

Процес розробки вебдодатка для розміщення оголошень з пошуку та надання волонтерської допомоги можна розділити на декілька етапів:

– визначення цілей створення проекту, складання технічного завдання та вимог до системи;
– створення макетів сторінок вебдодатка;
– вибір технологій для реалізації;
– проектування бази даних;
– реалізація серверної та клієнтської частин;
– тестування системи;
– деплой додатка.

Визначення технічного завдання є першим та важливим етапом у процесі розробки програмного забезпечення. У технічному завданні наведено опис функціональності додатка, вимог до нього, архітектури системи та іншу технічну інформацію. Технічне завдання дозволяє розробникам зрозуміти бізнес-потреби замовника і створити програмне забезпечення, що відповідає цим вимогам.

Створення макетів сторінок вебдодатка є важливим етапом у процесі розробки, оскільки макети дозволяють визначити та візуалізувати основні елементи та функціональність додатка перед його фактичною розробкою. Також макети дозволяють заздалегіть оцінити зручність використання додатка користувачами та визначити можливі проблеми. Це допомагає забезпечити отримання більш зручного та легкого у використанні додатка.

Наступним етапом є вибір технологій для безпосередньої розробки

додатка. Необхідно визначити, які технології і інструменти будуть використовуватися, наприклад, мови програмування, фреймворки, бібліотеки, бази та допоміжні залежності. Вибір технологій для розробки вебдодатка є важливим етапом, який впливає на якість кінцевого результату розробки. Під час вибору технологій потрібно враховувати такі фактори, як: функціональні вимоги, сумісність з цільовою платформою, масштабованість, стандарти, швидкодія, вартість.

Проектування бази даних допомагає визначити, які дані будуть зберігатися системою, як їх буде організовано та як вони будуть взаємодіяти між собою. Це допомагає забезпечити ефективне та оптимальне зберігання інформації. Проектування БД допомагає забезпечити безпеку даних, яка є важливою складовою будь-якого програмного забезпечення. На цьому етапі може бути виконане визначення прав доступу до даних, шифрування та інші методи захисту. Проектування БД допомагає забезпечити масштабованість системи. Правильна організація даних може допомогти уникнути проблем зі зберіганням та доступом до даних при збільшенні обсягу інформації, яка буде зберігатися в базі даних.

Наступний етап – безпосередньо етап розробки. Розробка вебдодатка з клієнт-серверною архітектурою може бути поділена на дві основні складові частини: бекенд (back-end) та фронтенд (front-end). Бекенд – це та частина вебдодатка, яка відповідає за обробку запитів користувача, збереження та обробку даних, автентифікацію, авторизацію та інші функції, які не пов'язані зі зовнішнім виглядом додатка. Фронтенд – це частина вебдодатка, яка відповідає за інтерфейс та взаємодію з користувачем. Фронтенд забезпечує візуальне представлення даних та можливість взаємодії з ними, забезпечує реактивність та відповідь на дії користувача. Етапи розробки бекенду та фронтенду вебзастосунка можуть бути організовані паралельно або послідовно, залежно від обраної методології розробки та специфіки проекту. Часто проект розпочинають з проектування дизайну та інтерфейсу користувача, після чого розробники можуть почати працювати над бекендом та фронтендом.

Незалежно від того, наскільки добре розроблено додаток, його необхідно

протестувати, щоб переконатися в його правильній роботі та відповідності вимогам бізнесу та потребам користувачів. Тестування допомагає виявити помилки та проблеми, що можуть вплинути на коректну роботу додатка. Це дозволяє вчасно виправити ці проблеми та забезпечити високу якість додатка. Також тестування допомагає забезпечити безпеку додатка – виявлення та виправлення проблем з безпекою допомагає захистити додаток від можливих атак та злому.

Деплой на робочий сервер – остання стадія розробки додатка. Незважаючи на те, що під час розробки додатка часто використовують деплої на сервери для розробників і тестувальників, важливо провести підсумкове тестування вже на робочому сервері. Спочатку треба вибрати і налаштувати серверне середовище. Завантаження вебдодатка на сервер часто здійснюють за допомогою FTP-клієнта. Також для більшості вебдодатків налаштувують доменне ім’я.

Після виконання даних етапів розробки програмного забезпечення додаток вважається готовим до використання користувачами.

2.3 Діаграма прецедентів

Діаграма прецедентів (Use Case Diagram) є одним з інструментів аналізу та проектування програмного забезпечення. Це один з інструментів в UML (Unified Modeling Language), який використовується для моделювання взаємодії між системою та її користувачами. Основною метою діаграми прецедентів є визначення поведінки системи з точки зору її взаємодії з користувачами.

Діаграма прецедентів дозволяє відобразити функціональність системи та поведінку її користувачів через визначення прецедентів, тобто конкретних сценаріїв взаємодії. Діаграма дозволяє ідентифікувати користувачів системи, їх потреби і вимоги. Вона допомагає команді розробників та аналітиків отримати чітке уявлення про те, як система має взаємодіяти з різними типами користувачів. Діаграма прецедентів відображає взаємодію між користувачами та системою і визначає дії, які можуть бути виконані користувачами в системі.

У діаграмі прецедентів зображуються актори (користувачі системи) та прецеденти (функціональність системи). Актори представляють зовнішніх користувачів, а прецеденти описують конкретні дії, які можуть бути виконані акторами.

Діаграма прецедентів була розроблена для платформи для розміщення оголошень з пошуку та надання волонтерської допомоги у формі вебдодатка. Діаграма містить два актори (рис. 2.1).

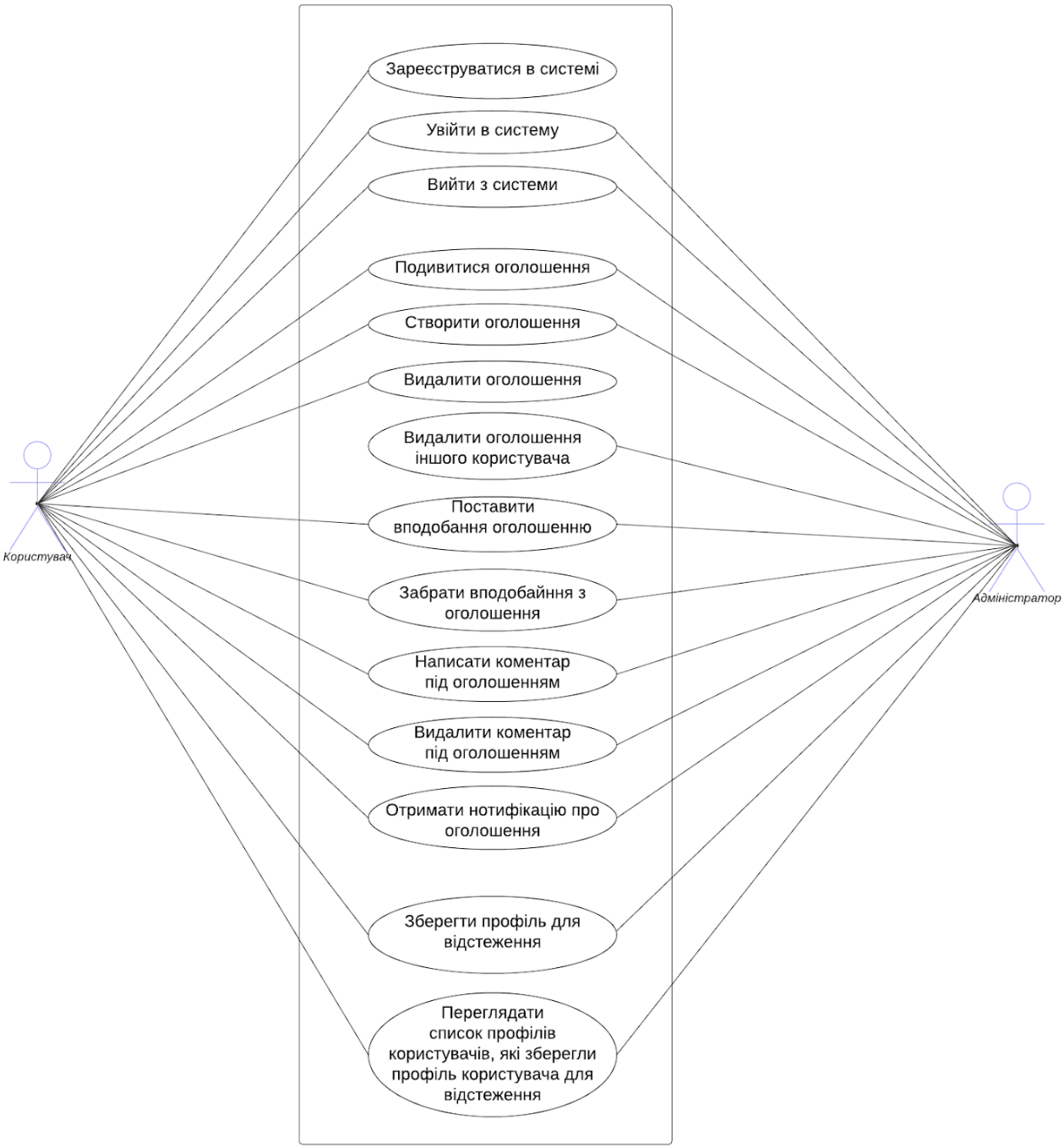

Рисунок 2.1 – Діаграма прецедентів

Прецеденти актора «Користувач» та актора «Адміністратор» наведено у табл. 2.1 та 2.2 відповідно.

Таблиця 2.1 – Прецеденти актора «Користувач»

Актор	Прецедент
Користувач	Зареєструватися в системі Увійти в систему Вийти з системи Подивитися оголошення Створити оголошення Видалити оголошення Поставити вподобання оголошенню Забрати вподобання з оголошення Написати коментар під оголошенням Видалити коментар під оголошенням Отримати нотифікацію про оголошення Зберегти профіль для відстеження Переглядати список профілів користувачів, які зберегли профіль користувача для відстеження

Таблиця 2.2 – Прецеденти актора «Адміністратор»

Актор	Прецедент
Адміністратор	Увійти в систему Вийти з системи Подивитися оголошення Створити оголошення Видалити оголошення Видалити оголошення іншого користувача Поставити вподобання оголошенню Забрати вподобання з оголошення

Продовження табл. 2.2

Актор	Прецедент
	Написати коментар під оголошенням Видалити коментар під оголошенням Отримати нотифікацію про оголошення Зберегти профіль для відстеження Переглядати список профілів користувачів, які зберегли профіль користувача для відстеження

Отже, створена діаграма прецедентів допомогає структурувати потреби користувачів системи та зрозуміти, які функції вона повинна забезпечувати.

2.4 Схема бази даних

Проектування бази даних було виконано відповідно до технічного завдання. Модель бази даних платформи для розміщення оголошень з пошуку та надання волонтерської допомоги складається з восьми сутностей:

- user – дані про користувачів;
- permission – дані про доступи, які можуть мати користувачі;
- ad – дані про оголошення;
- ad_type – дані про типи оголошень;
- ad_comment – дані про коментарі під оголошеннями;
- ad_likes – дані про вподобання оголошень;
- country – дані про країни;
- city – дані про міста країн.

Кожній сутності відповідає таблиця в базі даних. Також є допоміжні таблиці «user_has_permission» та «user_has_user».

Таблиця «user_has_permission» є допоміжною для створення відношення «багато до багатьох» між таблицями «user» та «permission», адже один

користувач може мати декілька доступів і один доступ може належати на кількох користувачів.

Таблиця «user_has_user» є допоміжною для створення відношення «багато до багатьох» в межах таблиці «user», адже один користувач може відстежувати декілька інших користувачів.

Деякі сутності моделі зв’язані відношенням «один до багатьох». Наприклад, сутність «add_comment» має поля «add_id» та «user_id», тобто коментар до оголошення посилається на оголошення, до якого він належить, та до користувача, що залишив цей коментар. Сутність «add_likes» має поля «add_id» та «user_id», тобто вподобання оголошення посилаться на оголошення, до якого вона належить, та до користувача, що поставив це вподобання. Сутність «city» має поля «country_id», тобто запис про місто має посилання на країну.

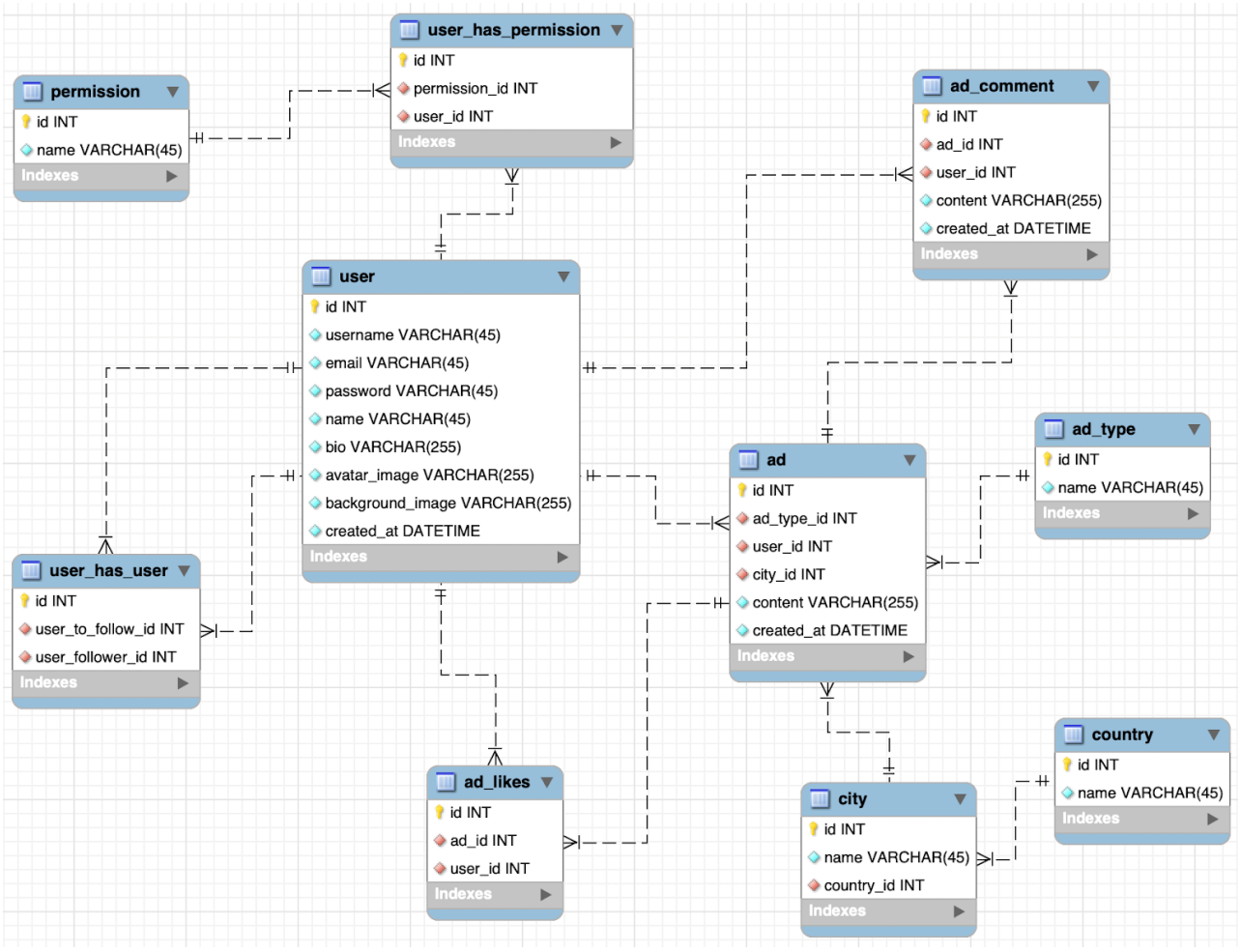

Рисунок 2.2 – ER-діаграма бази даних

2.5 Проектування інтерфейсу

На основі технічного завдання був спроектований інтерфейс системи під веб-платформу.

На рисунку 2.3 наведений макет сторінки реєстрації в системі. Посередині сторінки знаходиться форма для заповнення користувачем, що складається з 4 полів, інформація з яких потрібних для реєстрації, та кнопки підтвердження відправки даних.

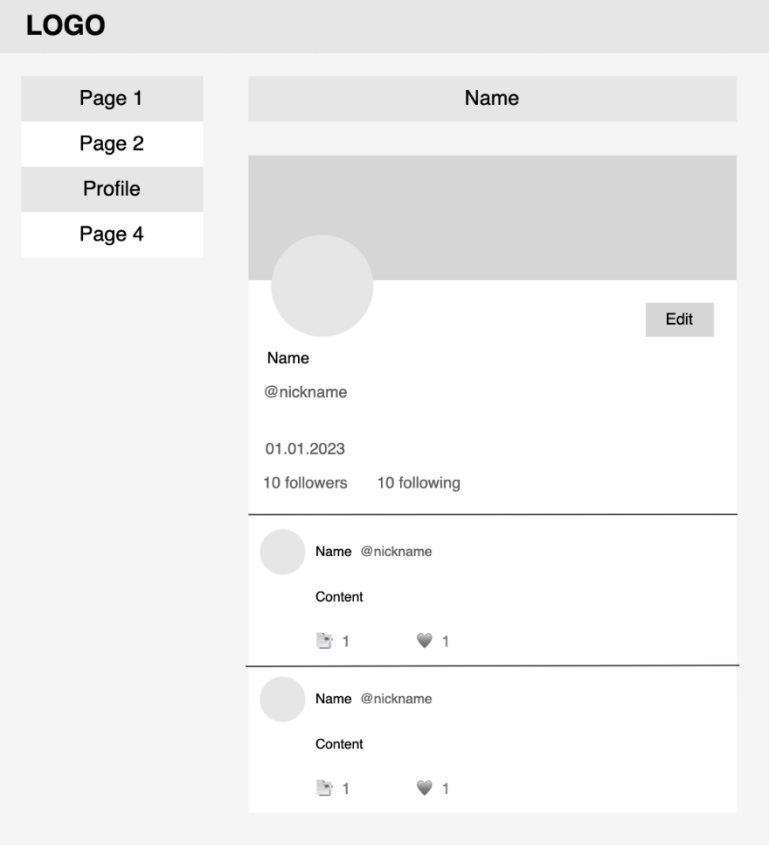

Рисунок 2.3 – Макет сторінки профілю

На рисунку 2.4 наведений макет головної сторінки системи. Сторінка має форму для створення нового оголошення. За формою розміщені опубліковані оголошення поточного та інших користувачів. Кожен блок оголошення має інформацію про користувача, який опублікував оголошення, його аватар, зміст оголошення, кількість коментарів та вподобань. Після натискатання на значок коментарів в блоці оголошення, система перенаправить користувача на окрему

сторінку певного оголошення, де користувач може побачити всі коментарі вибраного оголошення та додати новий.

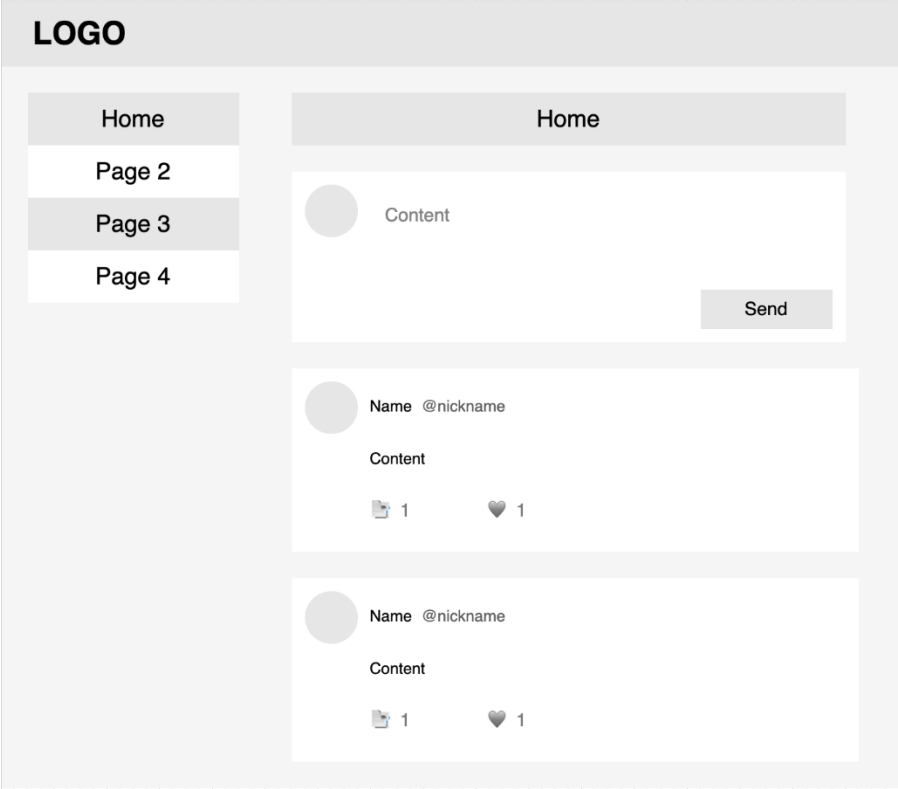

Рисунок 2.4 – Макет головної сторінки

На рисунку 2.5 наведений макет сторінки профілю. Вгорі сторінки відображається обкладинка профілю користувача, його аватар, ім’я, нікнейм, дата реєстрації в системі, кількість інших користувачів, яких відслідковує користувач та кількість користувачів, що відслідковують поточного користувача. Справа розміщена кнопка редагування, натискання якої відкриває модальне вікно з формою, що містить дані користувача, доступні для редагування. Після блока інформації користувача розміщені його опубліковані оголошення. Блок оголошення є ідентичним описаному на головній сторінці системи.

Вебдодаток має два головні розмітки сторінок. Перша використовується сторінками модуля авторизації та представлена на рисунку 2.3. Інша розмітка використовується сторінками, що доступні тільки авторизованому користувачу.

Ця розмітка представлена на рисунку 2.4 та на рисунку 2.5.

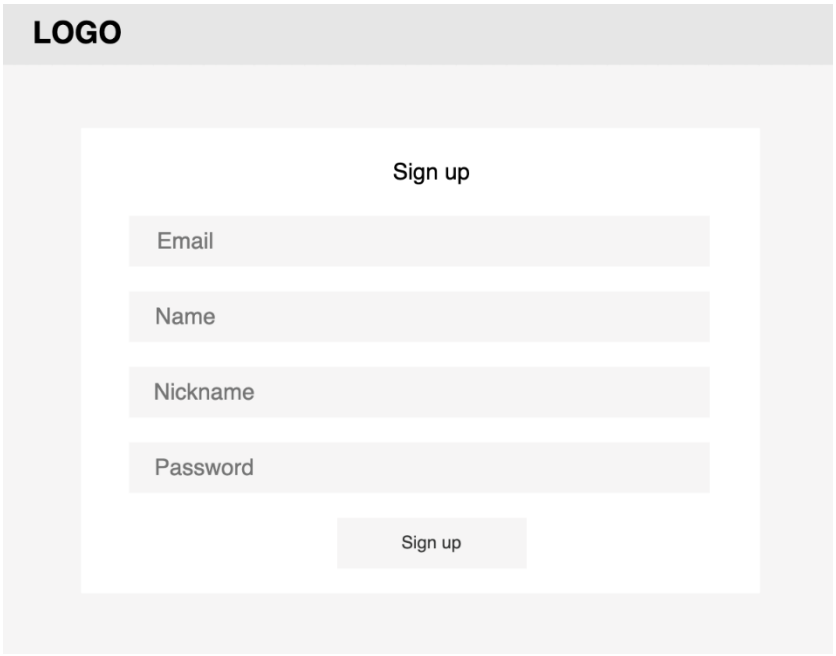

Рисунок 2.5 – Макет реєстрації у системі

Інтерфейс платформи для розміщення оголошень з пошуку та надання волонтерської допомоги було створено відповідно до технічного завдання та технічних вимог.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

На етапі проектування інформаційної системи було вирішено створити SPA засобами PostgreSQL, Node.js, Express.js, Angular та супутніми бібліотеками. Під час розробки використовувалася система контролю версій Git. Було створено окремі репозиторії для серверної та клієнтської частин.

3.1 Розробка серверної частини

Перший етап безпосередньої розробки системи – розробка серверної частини. Очікуваний результат цього етапу – API, що розроблений на платформі Node.js, відповідає вимогам REST, працює з форматом даних JSON та реалізує функціонал описаної на етапі проектування бізнес-логіки.

3.1.1 Ініціалізація та налаштування проекту

Ініціалізація Node.js проекту була виконана за допомогою команди «init» пакетного менеджеру npm. Результат роботи команди – створений файл «package.json», що є основною конфігурацією Node.js проекту, головна інформація якого – необхідні пакети, потрібні для роботи додатка, пакети, необхідні при його розробці, та описання команд (див. рис. 3.1).

Був встановлений зовнішній пакет «nodemon» – інструмент розробки для Node.js, який дозволяє автоматично перезапускати серверний додаток при зміні файлів в проекті [2]. Також були встановлені необхідні для роботи додатка такі зовнішні пакети, як: «express», «jsonwebtoken», «dotenv», «cors», «uuid».

У конфігурації проекту були прописані команди для запуску проекту у режимі розробки та для продакшену.

Був створений та налаштований екземпляр класу застосунку з бібліотеки Express.js та оброблений GET запит за маршрутом за замовчуванням.

Була налаштована CORS policy, яка за замовчуванням робить неможливим звернення до створеного API з інших URL. Код «app.use(cors())» вказує Express.js використовувати вбудоване проміжне програмне забезпечення «cors» з однойменного зовнішнього пакету, який дозволить API відповідати на запити з різних джерел.

```
require('dotenv').config()
const express = require('express')
const cors = require('cors')
const fileUpload = require('express-fileupload')
const router = require('./routes/index')
const path = require('path')

const PORT = process.env.PORT || 5000

const app = express()
app.use(cors())
app.use(express.json())
app.use(express.static(path.resolve(__dirname, 'static')))
app.use(fileUpload({}))
app.use('/api', router)

const start = async () => {
    try {
        app.listen(PORT, () => console.log(`Server has started on port ${PORT}`))
    } catch (e) {
        console.log(e)
    }
}
start()
```

Рисунок 3.1 – Ініціалізація Node.js проекту

До екземпляру застосунку було додане проміжне програмне забезпечення для обробки запитів з використанням JSON формату. Код «app.use(express.json())» вказує Express.js використовувати вбудоване проміжне програмне забезпечення «express.json», яке аналізує тіло запиту в форматі JSON

та перетворює його в об'єкт – тип даних мови JavaScript. Це дозволяє легко отримувати доступ до даних, надісланих у вигляді JSON, в контролерах, які обробляють запити до серверного додатка.

Було додане вбудоване в Express.js проміжне програмне забезпечення «express.static», яке дозволяє обробляти та віддавати статичні файли. Всі файли, розташовані у вказаній папці, будуть доступні за відповідними шляхами на сервері.

В кореневій директорії проекту було створено файл «.env», який дозволяє зберігати конфіденційні змінні середовища. Зовнішній модуль «dotenv» дозволяє завантажити змінні середовища з файлу змінних оточення «.env» і використовувати їх у коді.

За допомогою методу екземпляру застосунку «listen» сервер з його прослуховуванням запитів було запущено на порті, що вказаний у файлі змінних оточення.

3.1.2 Створення бази даних

На основі спроектованої ER-діаграми було почато створення бази даних застосунку, для чого було обрано СУБД PostgreSQL та ORM Prisma.

БД PostgreSQL була запущена за допомогою інструменту Docker. На основі завантаженого образа PostgreSQL з Docker Hub був створений та запущений контейнер за допомогою команди «docker run postgres» з додатковими флагами конфігурації БД, такі як внутрішній та зовнішній порт, ім’я БД та іншими.

У створеному Node.js проекті за допомогою команди «install» пакетного менеджеру NPM були встановлені зовнішні пакети «prisma» та «@prisma/client».

Був створений файл «shema.prisma» з конфігурацією бази даних (рис. 3.2). У створеному файлі була описана СУБД, клієнт бази даних, дані, необхідні для підключення до БД, та опис всіх таблиць. Командою «npx prisma generate» було згенеровано код Prisma Client на основі базових даних та моделей, описаних у

файлі «shema.prisma». Також була виконана команда «npx prisma migrate», результатом роботи якої є міграційний файл.

```
model Ad {
  id          String      @id @default(uuid())
  addType     AdType      @relation(fields: [addTypeId], references: [id])
  addTypeId   String
  user        User        @relation(fields: [userId], references: [id])
  userId      String
  city        City        @relation(fields: [cityId], references: [id])
  cityId      String
  commentOwner AdComment[] @relation()
  likeOwner   AdLike[]    @relation()
  content     String
  createdAt   DateTime    @default(now()) @map(name: "created_at")
}

model AdType {
  id   String @id @default(uuid())
  ad   Ad[]   @relation()
  name String
}

model AdComment {
  id        String   @id @default(uuid())
  ad        Ad       @relation(fields: [adId], references: [id])
  adId      String
  user      User     @relation(fields: [userId], references: [id])
  userId    String
  content   String
  createdAt DateTime @default(now()) @map(name: "created_at")
}

model AdLike {
  id     String @id @default(uuid())
  ad     Ad     @relation(fields: [adId], references: [id])
  adId   String
  user   User   @relation(fields: [userId], references: [id])
  userId String
}
```

Рисунок 3.2 – Опис таблиць модуля оголошень у Prisma

У Node.js проекті був створений екземпляр Prisma Client, який надає зручний спосіб взаємодії з базою даних, абстрагуючи розробника від деталей взаємодії з базою даних.

Результат виконання цього етапу – описана за створеною ER-діаграмою схема бази даних у форматі ORM Prisma, створена за цією схемою база даних PostgreSQL і підключена до серверного застосунку та налаштована для роботи ORM Prisma.

3.1.3 Розробка модуля авторизації

Мета цього етапу – реалізувати автентифікацію, авторизацію на серверній частині додатка та механізм захисту ендпоінтів.

За стратегію автентифікації було обрано використовувати JWT (JSON Web Tokens) з парою токенів (access-токен та refresh-токен) та зчитуванням access-токена у заголовку запита Authorization.

Такий stateless підхід з JWT був обраний на противагу stateful підходу з ідентифікаторами сесій, тому що:

- токени дозволяють реалізувати слабкозв'язну архітектуру;
- не вимагають використання додаткових баз даних як кешу (наприклад, NoSQL СУБД Redis);
- токени не мають прив'язки до веб-середовища і один серверний застосунок може обслуговувати різні клієнти, реалізовані під різні платформи.

Серед недоліків stateless підходу – неможливість видалити токен при виході користувача з системи або при витоку даних, адже виданий токен все одно буде дійсний той час, який був вказаний при його генеруванні.

Токен було вирішено відправляти у відповіді на запит логіну та зчитувати у заголовку Authorization при обробці захищених від неавтентифікованих користувачів запитів, тому що такий підхід є більш гнучким для обслуговуванням клієнтів різних платформ. Недолік такого рішення – це те, що

веб-клієнт буде змушений зберігати токен в локальному сховищі браузера, дані з якого можуть бути використані для проведення XSS-атак. Використання кукі для передачі токену є більш захищеним варіантом, але кукі прив'язані до веб-платформи.

Для реалізації JWT аутентифікації в Node.js було встановлено пакет jsonwebtoken, який надає можливість створювати та перевіряти JWT токени. Була написана функція для генерування JWT токену, яка задає вказане корисне навантаження, час дії токену та потребує ключа для шифрування підпису. Також була написана функція, яка за допомогою ключа перевіряє валідність підпису токена та повертає корисне навантаження, наявне у токені. Робота з JWT токеном була винесена у окремий сервіс (див. рис. 3.3).

```
class JWTTokenGenerator {
  private secretKey: string;

  constructor(secretKey: string) {
    this.secretKey = secretKey;
  }

  public generateToken(payload: TokenPayload, expiresInHours: number): string {
    const issuedAt = Math.floor(Date.now() / 1000);
    const expiresAt = issuedAt + expiresInHours * 60 * 60;

    const token = jwt.sign(payload, this.secretKey, {
      expiresIn: expiresInHours * 60 * 60
    });
    return token;
  }

  public verifyToken(token: string): TokenPayload | null {
    return jwt.verify(token, this.secretKey) as TokenPayload || null;
  }
}
```

Рисунок 3.3 – Сервіс для роботи з JWT токеном

Був створений та налаштований екземпляр класу Route з бібліотеки Express.js. Даний клас групує ендпоінти, що відносяться до модулю авторизації,

та задає їм базовий початок шляху. Модуль авторизації містить 3 ендпоінти: «register», «login», «refresh». Кожен з ендпоінтів обробляється відповідним методом контролера авторизації, який в свою чергу звертається до сервісу авторизації.

Ендпоінт типу POST та шляхом «register» очікує в тілі запиту поля, що описують користувача, на основі яких формується та зберігається в базі даних новий об’єкт користувача. Пароль користувача зберігається у базі даних у зашифрованому вигляді, шифрування було виконане за допомогою бібліотеки Bcrypt.

Ендпоінт типу POST та шляхом «login» очікує в тілі запиту два поля – пошту та пароль користувача, які читає з тіла запиту контролер та передає у сервіс авторизації. Відповідний метод сервісу знаходить в базі даних користувача з вказаною поштою та за допомогою бібліотеки Bcrypt перевіряє пароль, отриманий з клієнта, з паролем, збереженим у базі даних. Якщо запис користувача знайдено у базі даних та надано правильний пароль, відбувається генерування JWT токена. Для цього було використано метод «sign» з пакету jsonwebtoken. У корисне навантаження токену був доданий ідентифікатор користувача. Access-токен було створено з часом життя в 1 годину. Також було згенеровано refresh-токен для оновлення access-токену після закінчення часу його дії. Пару токенів було відправлено у форматі JSON як відповідь на даний HTTP-запит.

Вихід користувача з додатка не може бути реалізований в цьому серверному Node.js застосунку, адже був обраний stateless підхід до авторизації, при якому сервер не зберігає видані токони на противагу підходу з використанням ідентифікаторів сесій.

Ендпоінт типу GET та шляхом «refresh» очікує в тілі запиту виданий клієнту раніше refresh-токен та повертає у відповіді новий access-токен.

Було реалізовано перевірку параметрів запитів на валідність та відповідність вимогам. Якщо параметри не відповідають вимогам, запит повертає відповідний статус код та відповідь містить повідомлення з помилкою.

Для захисту ендпоінтів API від неавторизованих користувачів було використано зовнішній пакет Passport.js. Було захищено всі ендпоінти, які відповідають за бізнес-логіку системи. Доступними без токена залишаються тільки роути, що відповідають за реєстрацію та логін. Захищені ендпоінти очікують access-токен з валідним підписом та актуальним часом життя у заголовку Authorization.

3.1.4 Розробка API бізнес-логіки

На даному етапі створений серверний Node.js додаток був розширений реалізацією ендпоінтів, що реалізують бізнес-логіку системи.

При проектуванні API було дотримано правил архітектурного стилю REST. Основні вимоги, які були реалізовані:

- додаток має клієнт-серверну архітектуру: система поділена на серверну частину та клієнтську;
- сервер має тип stateless, з чого випливає, що кожний запит містить всі необхідні дані для його обробки та сервер не зберігає стан клієнта між запитами [6];
- API є уніфікованим: були використані потрібні стандартні HTTP методи (GET, POST, PUT, PATCH, DELETE), запити повертають потрібні статус коди та використовується єдиний формат даних [6].

Головна реалізована сутність – оголошення. Було створено новий роут з префіксом дочірніх маршрутів «ad», контролер та сервіс для роботи з модулем оголошень.

Було розроблено ендпоінт для отримання оголошень за різною фільтрацією: за користувачем, що опублікував оголошення, за типом оголошення, за містом, до якого відноситься оголошення та за датою публікації. Даний запит має тип GET, зчитує фільтри у квері параметрах та повертає масив оголошень.

Другий реалізований ендпоінт – видалення оголошень. Даний запит має тип DELETE та зчитує ідентифікатор оголошення з параметра шляху. Також запит містить перевірку на те, що запит на видалення зроблений або від власника оголошення, або від адміністратора.

Був реалізований ендпоінт на отримання типів оголошень, дані якого потрібні клієнту перед відправкою запита на створення нового оголошення.

Також були створені ендпоінти на отримання списку країн та списку міст певної країни. Ідентифікатор міста потрібний клієнту для формування тіла запита на створення нового оголошення.

Були розроблені ендпоінти для додавання та видалення вподобання оголошення, POST та DELETE відповідно, що зчитують ідентифікатор оголошення з параметрів адреси запиту.

Були розроблені ендпоінти для додавання та видалення коментаря під оголошенням, POST та DELETE відповідно. Запит на додавання коментаря зчитує зміст коментаря з поля з тіла запиту. Запит на видалення зчитує ідентифікатор оголошення з параметрів адреси запиту. Виконання основного функціоналу запиту на видалення коментаря містить перевірку на те, що запит на видалення зроблений або від власника коментаря або від адміністратора.

Також після розробки модуля авторизації було розширено функціонал роботи з користувачами системи.

Був доданий ендпоінт на редагування даних користувача. Запит має тип PATCH, який припускає часткову зміну даних. Реалізований функціонал додавання або зміни головного фото та фото фону профілю. Функціонал завантаження файлів на сервер був реалізований за допомогою зовнішнього пакету Multer.

Був реалізований функціонал можливості користувачеві відстежувати інших користувачів. Було розроблено два ендпоінти: для додавання та видалення відстежування користувача.

У всіх обробках ендпоінтів було використано раніше налаштований інструмент для роботи з базою даних Prisma Client.

При розробці API було використано Postman – інструмент, який дозволяє тестувати, документувати та взаємодіяти з API. Він надає корисні функції для створення та виконання запитів до API, тестування їх, перевірки відповідей та розробки API-документації. Для проекту була створена колекція в Postman, куди було додано розроблені запити з їхніми очікуваними вхідними даними та прописані такі налаштування, як глобальні змінні і середовища.

3.2 Розробка клієнтської частини

Наступний етап розробки проекту – розробка клієнтської частини. Очікуваний результат цього етапу – клієнт для веб-платформи, що є SPA додатком, розробленим, використовуючи фреймворк Angular та супутні йому бібліотеки, написаний, дотримуючись основних кращих практик написання інтерфейсів, реалізує функціонал описаної на етапі проектування бізнес-логіки та має інтерфейс, що відповідає створеним макетам.

3.2.1 Ініціалізація та налаштування проекту

Для роботи з фреймворком Angular було встановлено Angular CLI (Command Line Interface) – інструмент командного рядка, який надає швидкий і зручний спосіб розробки Angular додатків, дозволяє створювати нові компоненти, модулі, сервіси, директиви та інші елементи Angular, а також виконувати різні операції, пов’язані з розробкою, збиранням та тестуванням додатків [8]. За допомогою Angular CLI було створено новий проект. Було створено Angular проект зі структурою файлів і налаштуваннями за замовчуванням, включаючи файли компонентів, модулів та конфігураційний файл «angular.json». Також було встановлено залежності за допомогою команди

«npm install».

При створенні проекту було вказано прапор «--style» або «-s», який визначає тип написання стилів, який буде використовуватись в проекті. Для роботи над проектом було обрано препроцесор Sass (Syntactically Awesome Style Sheets) – препроцесор CSS, який додає додаткові функції та можливості до звичайного CSS, роблячи його більш потужним та гнучким [13]. Sass-файли потребують компіляції у звичайний CSS перед роботою у браузері [13]. Даний препроцесор має два синтаксиси: Sass та SCSS. Обидва синтаксиси забезпечують ті самі функціональність і можливості, але відрізняються безпосередньо синтаксисом. Для роботи над проектом було обрано синтаксис SCSS.

Конфігурація Angular проекту описана у файлу «angular.json», що містить такі налаштування, як шляхи до файлів, налаштування збірок у різних режимах, налаштування модульного та інтеграційного тестування, загальні конфігурації компонентів, окремі налаштування кожного з дочірніх проектів та інші параметри [8].

Налаштування TypeScript в Angular проекті відбувається через файл конфігурації «tsconfig.json», що визначає налаштування компіляції проекту на TypeScript [1]. Він містить налаштування та параметри, які використовуються компілятором TypeScript для генерації JavaScript коду з файлів TypeScript.

При розробці даного Angular проекту було вирішено дотримуватися підходу Feature-Sliced Design. Це підхід до організації структури фронтенд проекту, який спрямований на покращення модульності, розширюваності та повторного використання коду. Цей підхід рекомендує розділяти функціональність проекту на невеликі, самостійні модулі, що забезпечує слабкозв'язність модулів проекту.

До проекту була додана бібліотека NgRx. Основна мета використання NgRx полягає в тому, щоб мати одну істину для стану додатка. Це означає, що всі дані, що впливають на стан додатка, зберігаються в централізованому місці – сторі. Компоненти можуть зчитувати дані зі стору та підписуватись на зміни, а

також викликати дії для зміни стану [11]. Було встановлено пакети NgRx за допомогою npm: пакет @ngrx/store для управління станом додатка і @ngrx/effects для роботи з побічними ефектами.

У проект було встановлено як залежність бібліотеку Tailwind CSS, створені файли конфігурації Tailwind CSS та PostCSS. Tailwind CSS – це бібліотека CSS, яка надає набір готових класів, які можна використовувати для швидкого і зручного написання стилів вебдодатків [12]. Вона побудована на принципі «utility-first», що означає, що вона надає широкий набір класів, які використовуються безпосередньо в HTML-розмітці для стилізації елементів.

3.2.2 Інтеграція з серверною частиною

Спираючись на створену документацію API серверної частини, було виконано інтеграцію клієнтської та серверної частин.

В проект веб-клієнту було підключено модуль HttpClientModule, наданий фреймворком Angular. Цей модуль забезпечує зручний спосіб взаємодії з API та обробки відповідей. HttpClientModule надає клас HttpClient, який має ряд методів для виконання HTTP-запитів, таких як GET, POST, PUT, DELETE та інші. Крім методів, HttpClient також надає можливість встановлення заголовків запиту, обробки параметрів, використання типізації даних та інших опцій для налаштування запитів. HttpClient також підтримує обробку відповідей у форматі JSON, використання потокової передачі даних, обробку помилок та інші функціональні можливості.

Були написані сервіси, що відповідають безпосередньо за інтеграцію з API (приклад методу для отримання оголошень наведено на рис. 3.4). Методи, що звертаються до ендпоінтів, були поділені на сервіси за принципом зв’язності за спільною сутністю і за принципом єдиної відповідальності.

```
import { Injectable } from '@angular/core';
import { HttpClient, HttpParams } from '@angular/common/http';
import { Observable } from 'rxjs';
import { Ad } from '@core/models ';

interface GetAdsParams {
   userId: string;
   adTypeId: string;
   cityId: string;
}

@Injectable({
  providedIn: 'root'
})
export class AdApiService {
  private apiUrl = 'ad';

  constructor(private http: HttpClient) {}

  public get({ userId, adTypeId, cityId }: GetAdsParams): Observable<Ad[] > {
    let params = new HttpParams();
    params = params.set('user', userId);
    params = params.set('adType', adTypeId);
    params = params.set('city', cityId);
    return this.http.get(`${this.apiUrl}`, { params });
  }
}
```

Рисунок 3.4 – Сервіс інтеграції з API

Методи класу HttpClient повертають об’єкти типу Observable. Observable є частиною пакету RxJS і є асинхронним потоком даних, на який можна підписатися для отримання результатів запиту (див. рис. 3.5).

Було винесено базовий URL сервера за допомогою інтерсептора в Angular, для чого було використано клас HttpClientInterceptor. Інтерсептори дозволяють перехоплювати та обробляти HTTP-запити перед їхньої відправкою та після отримання відповіді сервера. Було написано клас BaseUrlInterceptor, що успадковується від класу HttpInterceptor, та перевизначено метод перехоплення запитів. Створений інтерсептор був доданий у AppModule до списку провайдерів HTTP_INTERCEPTORS. Як результат, при виконанні HTTP-запитів з

використанням HttpClient, базова URL буде автоматично додана до запиту.

```
export class AdPageComponent implements OnInit {
  public ads: Ad[] = [];

  constructor(private adApiService: AdApiService) { }

  public ngOnInit() {
    this.fetchItems();
  }

  public fetchItems() {
    this.adApiService.get().subscribe(
      (ads: Ad[]) => {
        this.ads = ads;
      },
      (error) => {
        console.error('Error:', error);
      }
    );
  }
}
```

Рисунок 3.5 – Використання сервісу інтеграції з API

3.2.3 Розробка модуля авторизації

На цьому етапі розробки клієнту було розроблено модуль авторизації клієнтського додатка: створені шаблони інтерфейсу, виконана інтеграція з серверною частиною, реалізований захист сторінок від неавторизованих користувачів.

При намірі користувача увійти в систему (див. рис. 3.6), з веб-інтерфейсу збираються необхідні дані та виконується запит до API, який у разі валідних даних повертає згенерований JWT токен. Отриманий токен вирішено зберігати у об’єкті localStorage, який забезпечує можливість зберігати дані на боці клієнта в браузері.

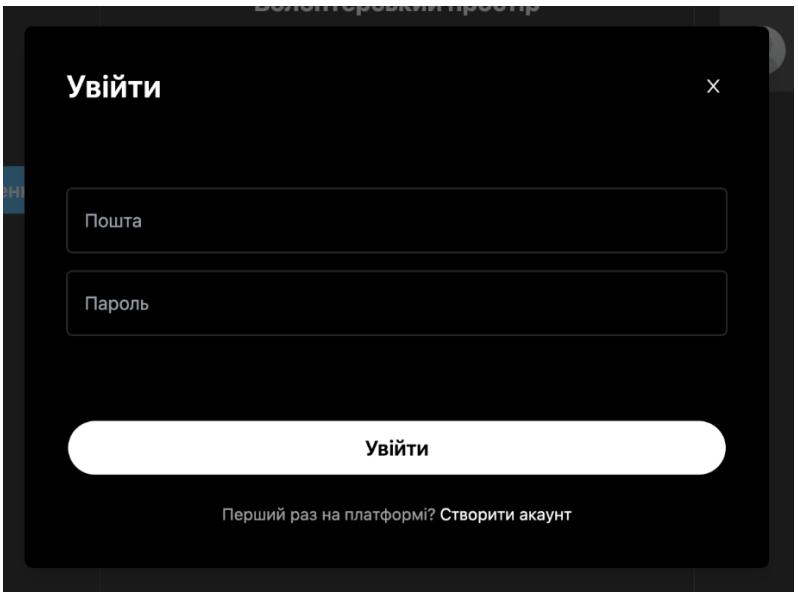

Рисунок 3.6 – Вхід до системи

При першому завантаженні вебдодатка зчитується поле з localStorage, під яким зберігається токен, і у разі наявності токена, він зберігається в стейт, що забезпечує неперервність сесії користувача при закритті вкладки браузера.

При відсутності токена, система відображає інтерфейс створення акаунту (див. рис. 3.7).

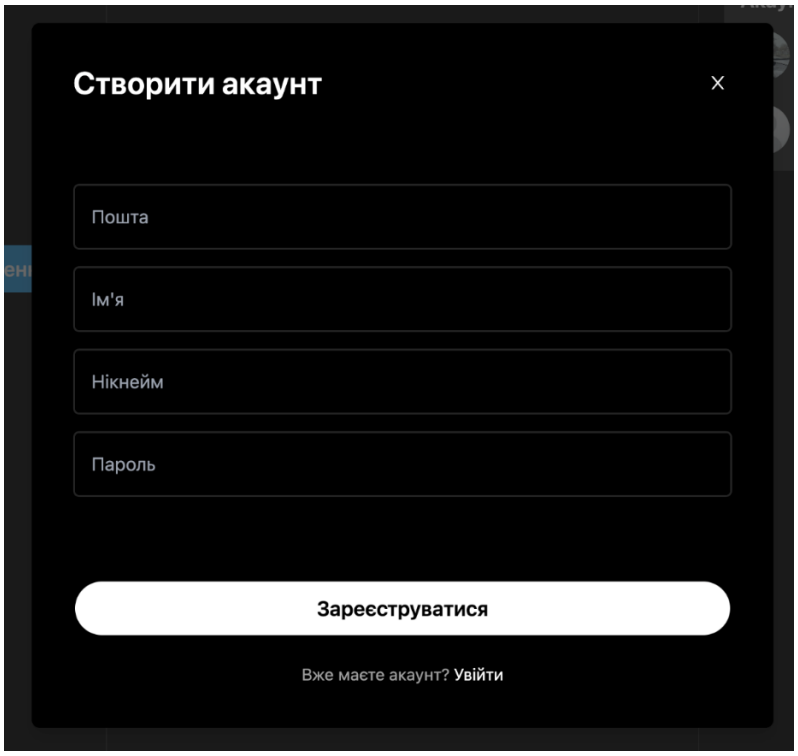

Рисунок 3.7 – Створення акаунту

При намірі користувача вийти з системи, з стану додатка та з локального сховища браузера видаляється токен.

Всі запити до розробленого API, крім запитів на реєстрацію та логін у систему, очікують JWT токен для аутентифікації клієнта. Клієнт має передавати виданий сервером токен у заголовку Authorization – це HTTP заголовок, який використовується для передачі інформації про авторизацію користувача при виконанні HTTP-запитів.

Було винесено додавання токену в заголовок Authorization запитів за допомогою інтерсептора. Було написано клас AuthTokenInterceptor (див. рис. 3.8). Створений інтерсептор був доданий на рівні AppModule до списку провайдерів. При виконанні HTTP-запитів, які потребують аутентифікації клієнта, токен, збережений у стані Angular додатка, автоматично буде доданий заголовок Authorization з типом авторизації Bearer та токеном.

```
export class AuthTokenInterceptor implements HttpInterceptor {
   constructor(private store: Store<AppState>) { }

   public intercept(request: HttpRequest<any>, next: HttpHandler):
Observable<HttpEvent<any>> {
      return this.store.pipe(
         select(getToken),
         take(1),
         switchMap((token) => {
            if (token) {
               request = request.clone({
                  setHeaders: {
                     Authorization: `Bearer ${token}`
                  }
               });
            }

            return next.handle(request);
         })
      );
   }
}
```

Рисунок 3.8 – Сервіс для роботи з JWT токеном

Всі маршрути клієнтського додатка, крім маршрутів модулю авторизації, були захищені від неавторизованих користувачів. Для реалізації цього функціоналу було використано інструмент Guards, наданий фреймворком Angular.

Guards – це класи, які використовуються для захисту та контролю доступу до маршрутів, компонентів або ресурсів в Angular додатка [8]. Вони виконують перевірку на певні умови перед тим, як дозволити користувачеві перейти до певного маршруту або отримати доступ до певного ресурсу. Вони можуть бути використані для реалізації сценаріїв контролю доступу, таких як: перевірка авторизації, перевірка ролей користувача та перевірка валідності токенів.

Були реалізовані наступні типи даних класів: CanActivate, що використовується для визначення, чи може користувач здійснити навігацію до певного маршруту, та CanActivateChild – аналогічний CanActivate, але використовується для визначення, чи може користувач здійснити навігацію до дочірніх маршрутів (див. рис. 3.9).

```
export class AuthGuard implements CanActivate {
   constructor(private store: Store<AppState>, private router: Router) { }

   public canActivate(route: ActivatedRouteSnapshot, state: RouterStateSnapshot):
Observable<boolean> {
      return this.store.pipe(
         select(isLoggedIn),
         take(1),
         map((loggedIn) => {
            if (loggedIn) {
               return true;
            }
            this.router.navigate(['/login']);
            return false;
         })
      );
   }
}
```

Рисунок 3.9 – Захист маршрутів від неавтифікованих користувачів

CanDeactivate Guard перевіряє, чи може користувач покинути поточний маршрут. Він виконується перед виходом з маршруту і дозволяє заборонити перехід, якщо виконуються певні умови (наприклад, незбережені зміни у формі).

Resolve Guard використовується для завантаження даних перед переходом до маршруту. Він дозволяє отримати дані з сервера або інших джерел перед відображенням компонента маршруту.

CanLoad Guard визначає, чи може користувач завантажити модуль перед переходом до маршруту. Він виконується перед завантаженням модуля і дозволяє заборонити завантаження, якщо виконуються певні умови.

3.2.4 Розробка користувацького інтерфейсу

Після ініціалізації, налаштування проекту та розробки модуля авторизації було виконано розробку основного функціоналу додатка.

Був розроблений стан додатка, використовуючи встановлені на етапі ініціалізації проекту пакети бібліотеки NgRx (див. рис. 3.10). Створено необхідну структуру директорій для NgRx, дотримуючись рекомендованої структури, яка включає файли для дій (actions), редукторів (reducers), селекторів (selectors), ефектів (effects) та станів (states). Створено дії, які представляють різні події або дії в додатку. Дії мають типи (types) і можуть мати додаткові дані, які передаються разом з дією (payload).

```
export const loadAds = createAction('[Ad] Load Ads');
export const loadAdsSuccess = createAction('[Ad] Load Ads Success', props<{ ads: Ad[] }>());
export const loadAdsFailure = createAction('[Ad] Load Ads Failure', props<{ error: string }>());

export const adReducer = createReducer(
  initialState,
  on(AdActions.loadAds, (state) => ({ ...state, loading: true })),
  on(AdActions.loadAdsSuccess, (state, { ads }) => ({ ...state, ads, loading: false })),
  on(AdActions.loadAdsFailure, (state, { error }) => ({ ...state, error, loading: false })),
);
```

Рисунок 3.10 – Створення стану оголошень

Створено редуктори для обробки дій і оновлення стану додатка. Редуктори приймають поточний стан і дію і повертають новий стан, враховуючи зміни, внесені дією. Створено селектори, які дозволяють отримувати певні частини стану з загального стану додатка. Селектори є функціями, які приймають стан і повертають певні дані зі стану. Створено ефекти, які дозволяють виконувати побічні дії, такі як взаємодія зі зовнішнім API, асинхронні запити, обробка даних з локального сховища та інші дії, необхідні для управління станом додатка (див. рис. 3.11). Було імпортовано файли редукторів та ефектів та додано їх до налаштувань імпортів зовнішніх модулів пакетів NgRx.

Було розроблено сторінку профіля користувача. Створено компонент для сторінки профілю: виконано команду «ng generate component profile» для автоматичного генерування нового Angular компонента. Створені файли для логіки, розмітки, стилів та модульного тестування компонента. Було визначено маршрут для сторінки профілю. Створено гілка стану додатка, що відповідає за дані користувача. Написаний відповідний сервіс, що звертається до створеної гілки стану. З даним сервісом взаємодіють компоненти, що відносяться до модуля користувача Angular додатка.

```
export class AdEffects {
   loadAds$ = createEffect(() =>
      this.actions$.pipe(
         ofType(AdActions.loadAds),
         mergeMap(() =>
            this.adApiService.get().pipe(
               map((ads) => AdActions.loadAdsSuccess({ ads })),
               catchError((error) => of(AdActions.loadAdsFailure({ error: error.message })))
            )
         )
      )
   );

   constructor(private actions$: Actions, private adService: adApiService) { }
}
```

Рисунок 3.11 – Ефект взаємодії з API

Сторінка профілю користувача має функціонал редагування (рис. 3.12). Редагування відбувається у модальному вікні. Користувач може змінити інформацію про себе та додати або змінити фото свого аватару та фото фону свого профайлу.

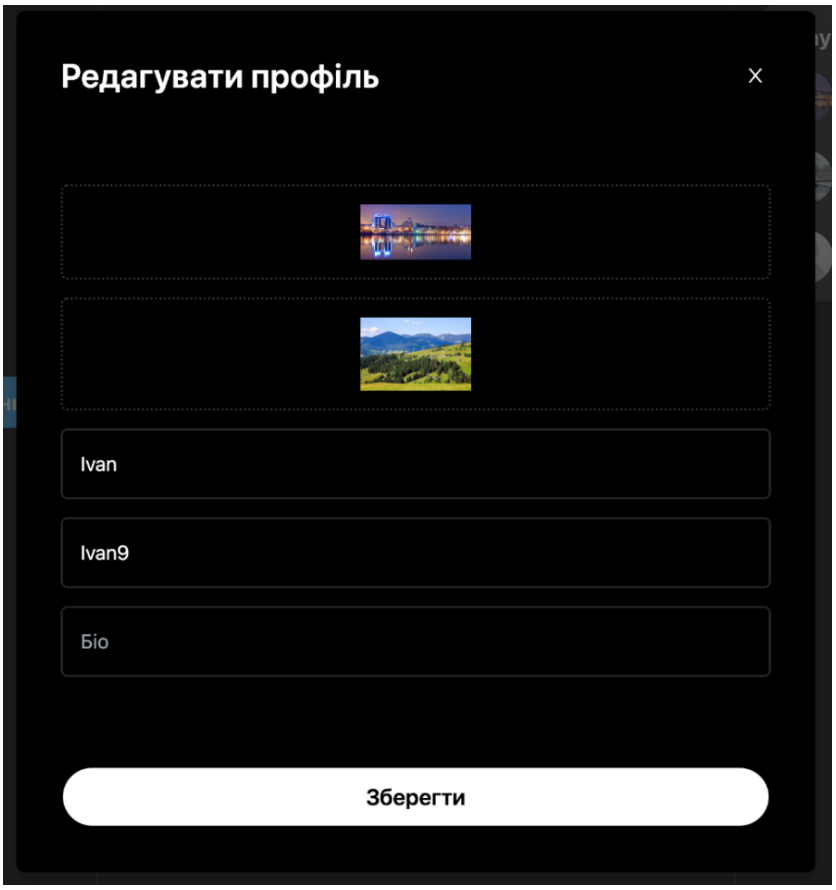

Рисунок 3.12 – Редагування профіля

Було розроблено головну сторінку веб-клієнта, яка містить форму для створення нового оголошення та стрічку з іншими оголошеннями (див. рис. 3.13). Блок оголошення має функціонал додавання або видалення вподобання та публікації нового коментаря або видалення свого наявного коментаря.

Також відображається список користувачів, зареєстрованих у системі. Натиснувши на певного користувача, додаток виконує редірект користувача на відповідний профіль (див. рис. 3.14). Сторінка стороннього профілю має функціонал додавання або видалення користувача у свій список для відстеження.

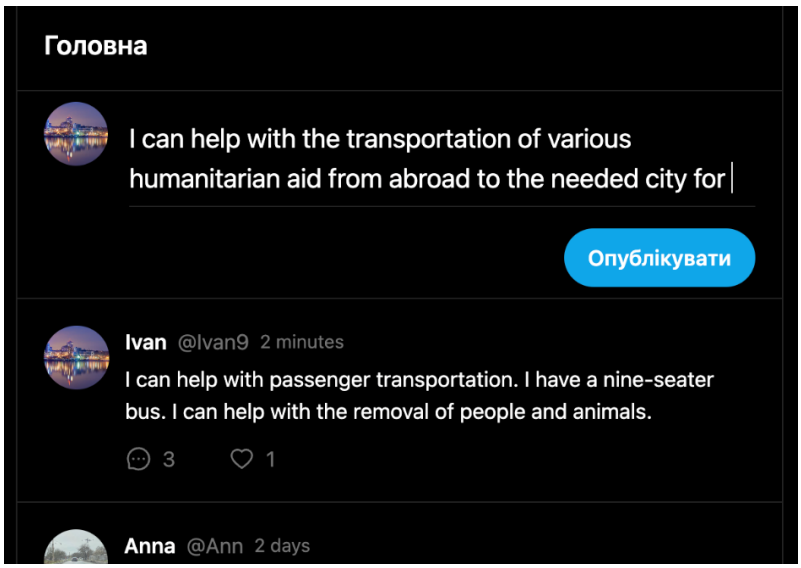

Рисунок 3.13 – Публікація оголошення

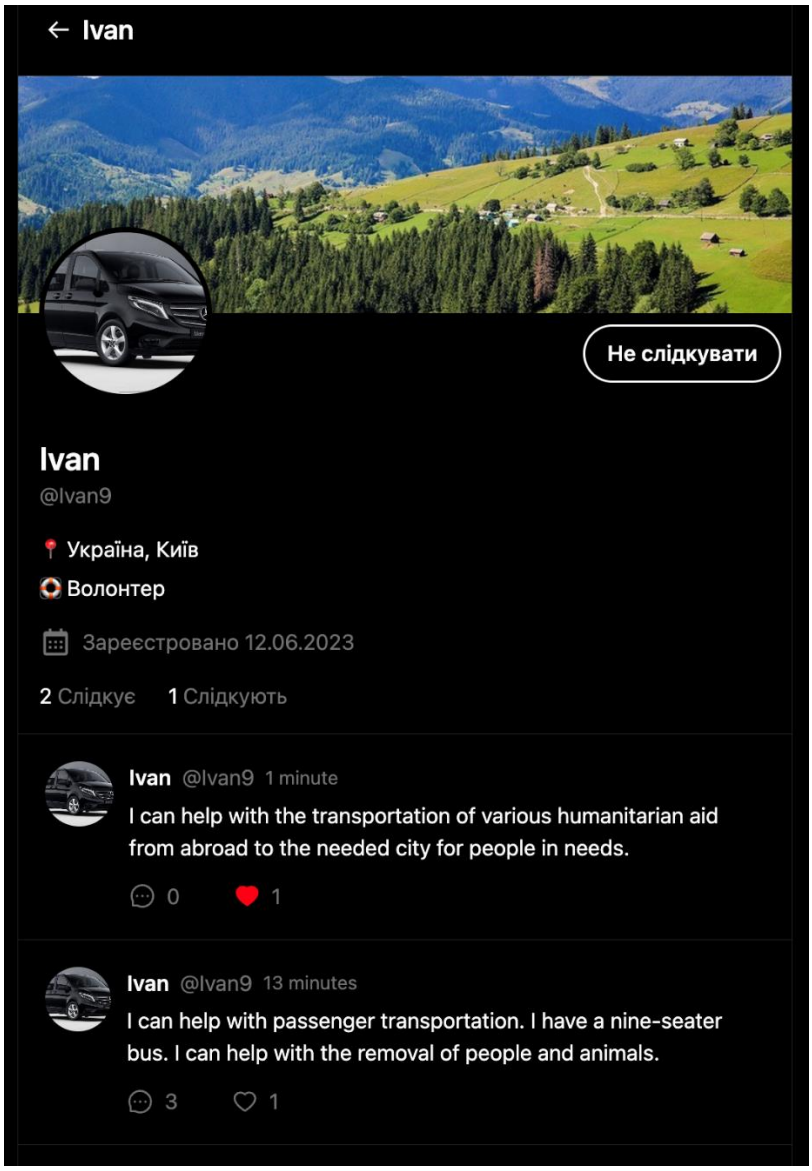

Рисунок 3.14 – Профіль користувача

Повторювані UI-елементи були винесені у окремі компоненти, що полегшує управління та повторне використання логіки, шаблонів і стилів компонентів. Було створено новий модуль в Angular проекті, який містить компоненти UI-кіту. В середині модуля UI-кіту було створено окремі компоненти для кожної виділеної частини UI.

Також було винесено лейаути сторінок додатка в окремі компоненти, що спрощує управління та підтримку інтерфейсу. В компонентах лейаутів розміщено загальні елементи сторінок. У шаблонах лейаутів було реалізовано можливість вбудовувати дочірні компоненти у лейаут залежно від поточного шляху маршрутизації.

Для обробки форм веб-інтерфейсу було використано пакет @angular/forms, наданий фреймворком Angular, який надає засоби для створення та обробки форм. Був обраний реактивний підхід до форм у Angular, який базується на використанні модуля ReactiveFormsModule та класів FormControl, FormGroup та FormArray для керування станом та поведінкою форм. За допомогою реактивних форм дані автоматично синхронізуються між шаблоном і компонентом. Реактивні форми надають засоби для валідації введених даних. Було додано валідатори до полів форми, реалізована перевірка їх на валідність, обробка помилок та відображення повідомлення про помилки у шаблоні.

Для зменшення початкового розміру додатка і покращення швидкості його завантаження було реалізовано lazy loading – підхід, який дозволяє завантажувати модулі асинхронно лише тоді, коли вони потрібні, а не заздалегідь при завантаженні додатка [8]. Розроблений функціонал було поділено на окремі модулі, які будуть завантажуватися при потребі. Визначено маршрути для завантаження модулів. Замість прямого імпорту модулів було вказано шляхи до модулів і використано функцію loadChildren. Завдяки цьому лише необхідні ресурси завантажуються, коли користувач потрапляє на певну сторінку або виконує певні дії, що допомагає зберегти ресурси і зменшити час першого завантаження додатка.

Для основних класів розробленого клієнтського проекту було написано

модульні тести. Модульні тести допомагають виявляти помилки на ранніх етапах розробки та забезпечують якість програмного забезпечення. Для написання модульних тестів використано фреймворк тестування Jasmine. Jasmine надає можливості для опису тестів та встановлення очікуваних результатів [8].

Для кожного модуля, компонента або сервісу було створено специфікаційний файл з префіксом «spec». В специфікаційних файлах написані тести, які перевіряють правильність роботи окремих функцій, методів або компонентів. Використано функції «describe» та «it» для організації тестів і функцію «expect» для встановлення очікуваних результатів. За допомогою тестового ранера Karma було виконано створені модульні тесті у браузері.

ВИСНОВКИ

В результаті виконання кваліфікаційної роботи було створено програмну реалізацію вебдодатка для розміщення оголошень з пошуку та надання волонтерської допомоги.

Для розробки вебдодатка використовувався наступний стек технологій:

- платформа Node.js, мова JavaScript, фреймфорк Express.js;
- СУБД PostgreSQL, ORM Prisma, бібліотека Prisma Client;
- мова TypeScript, фреймворк Angular, бібліотеки RxJS, NgRx і Tailwind CSS.

У ході роботи були виконані такі завдання:

- проведення аналізу предметної області, визначення функціональності аналогічних систем;
- визначення мети розробки системи, складання технічного завдання та вимог до системи;
- розробка схеми API та створення макетів інтерфейсу;
- проектування та створення бази даних, використовуючи СУБД PostgreSQL та ORM Prisma;
- розробка RESTful API, використовуючи платформи Node.js та фреймворку Express.js;
- розробка клієнтського додатка, використовуючи фреймворка Angular;
- деплой додатка;
- тестування додатка.

Розроблений додаток відповідає сформованому технічному завданню та технічним вимогам.

ПЕРЕЛІК ПОСИЛАНЬ

1. TypeScript Documentation. URL: https://www.typescriptlang.org (дата звернення: 21.04.2023).
2. Node.js Documentation. URL: https://nodejs.org/dist/latest-v18.x/docs/api (дата звернення: 19.03.2023).
3. Express.js Documentation. URL: http://expressjs.com (дата звернення: 21.03.2023).
4. Prisma Documentation. URL: https://www.prisma.io (дата звернення: 17.04.2023).
5. PostgreSQL Documentation. URL: https://www.postgresql.org/docs (дата звернення: 10.04.2023).
6. RESTful web API design. URL: https://learn.microsoft.com/en-us/azure/architecture/best-practices/api-design (дата звернення: 15.04.2023).
7. Single page apps in depth. URL: http://singlepageappbook.com/goal.html (дата звернення: 05.03.2023).
8. Angular Documentation. URL: https://angular.io/docs (дата звернення: 05.05.2023).
9. The RxJS library. URL: https://angular.io/guide/rx-library (дата звернення: 21.04.2023).
10. Introduction to RxJS. URL: https://rxjs.dev/guide/overview (дата звернення: 21.04.2023).
11. NgRx Documentation. URL: https://ngrx.io (дата звернення: 21.04.2023).
12. Tailwind CSS Documentation. URL: https://tailwindcss.com (дата звернення: 17.04.2023).
13. Sass Documentation. URL: https://sass-lang.com (дата звернення: 17.04.2023).