Міністерство освіти і науки України
Національний технічний університет
«Дніпровська політехніка»

Інститут електроенергетики
(інститут)

Факультет інформаційних технологій
(факультет)

Кафедра Програмного забезпечення комп'ютерних систем
(повна назва)

ПОЯСНЮВАЛЬНА ЗАПИСКА
кваліфікаційної роботи ступеня
бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *Вареника Микити Олександровича*
(ПІБ)

академічної групи *122-19-2*
(шифр)

спеціальності *122 Комп'ютерні науки*
(код і назва спеціальності)

освітньої програми *Комп'ютерні науки*
(назва освітньої програми)

на тему: *Розробка системи автоматизованого управління волонтерською діяльністю*

Керівники	**Прізвище, ініціали**	**Оцінка за шкалою**		**Підпис**
		рейтинговою	**інституційною**	
кваліфікаційної роботи	*проф. Мороз Б.І.*			
розділів:				
спеціальний	*проф.Мороз Б.І.*			
економічний	*проф. Вагонова О.Г.*			
Рецензент				
Нормоконтролер	*доц. Гуліна І.Г.*			

Дніпро
2023

Міністерство освіти і науки України
НТУ «Дніпровська політехніка»

ЗАТВЕРДЖЕНО:
завідувач кафедри
програмного забезпечення комп'ютерних систем
(повна назва)
М.О. Алексєєв
(підпис) (прізвище, ініціали)
« » 2023 року

ЗАВДАННЯ
на кваліфікаційну роботу
бакалавра
(назва освітньо-кваліфікаційного рівня)

студента *122-19-2* (група) *Вареника М.О.* (прізвище та ініціали)
тема кваліфікаційної роботи *Розробка системи автоматизованого управління волонтерською діяльністю*

затверджена наказом ректора НТУ «ДП» від *16.05.2023 р.* № *350-с*

Розділ	Зміст виконання	Термін виконання
Спеціальний	***На основі матеріалів проєктно-технологічної практики та інших науково-технічних джерел провести аналіз стану рішення проблеми та постановку задачі. Обґрунтувати вибір та здійснити реалізацію методів вирішення проблеми***	***13.05.2023 р.***
Економічний	***Провести розрахунок трудомісткості розробки програмного забезпечення, витрат на створення ПЗ й тривалості його розробки***	***27.05.2023 р.***

Завдання видав (підпис) *проф. Мороз Б.І.* (посада, прізвище, ініціали)
Завдання прийняв до виконання (підпис) *Вареник М.О.* (прізвище, ініціали)

Дата видачі завдання: *14.01.2023 р.*

Термін подання кваліфікаційної роботи до ЕК: *12.06.2023 р.*

РЕФЕРАТ

Пояснювальна записка: 92с., 28 рис., 3 дод, 20 джерел.

Об’єкт розробки: веб-орієнтований додаток для системи управління волонтерською діяльністю з використанням бібліотек Spring Boot та Angular.

Мета кваліфікаційної роботи: розробка програмного забезпечення для веб-орієнтованого додатку, який буде використовуватися для створення та обробки запитів волонтерської допомоги.

У вступі проводиться аналіз та досліджується поточний стан проблеми, конкретизується мета кваліфікаційної роботи та галузь її застосування, робиться обґрунтування актуальності теми і формується постановка завдання.

У першому розділі проаналізовано предметну галузь, визначено актуальність завдання та призначення розробки, сформульовано постановку завдання, зазначено вимоги до програмної реалізації, технологій та програмних засобів.

У другому розділі проаналізовані наявні рішення, обрано платформу для розробки, виконано проектування і розробку веб-орієнтованої інформаційної системи, описана робота системи, алгоритм і структура його функціонування, а також виклик та завантаження додатку, визначено вхідні і вихідні дані, охарактеризовано склад параметрів технічних засобів.

Практичне значення полягає у створенні веб-орієнтованого додатку, що забезпечує: створення запитів людьми що потребують допомоги, їх обробка волонтерами, коммунікація у чатах що специфічні до кожної окремої проблеми, перегляд усіх наявних запитів, а також прокладення оптимального за задачею коммівояжера шляху проїзду між локаціями запитів.

Актуальність розробки інформаційної системи для автоматизації волонтерської діяльності очевидна в умовах війни, під час якої задля значного пришвидшення надання адресної допомоги все більше використовують інформаційні системи.

Список ключових слів: ІНФОРМАЦІЙНА СИСТЕМА, ВОЛОНТЕРСТВО, БАЗА ДАНИХ, SPRING BOOT, ANGULAR, POSTGRESQL

ABSTRACT

Explanatory note: 92 p., 28 figs., 3 apps, 20 sources.

Development object: a web application for a volunteer management system using Spring Boot and Angular libraries.

The purpose of the qualification work: software development for a web application that will be used to create and process requests for volunteer assistance.

The introduction analyzes and explores the current state of the problem, defines the purpose of the qualification work and its scope, substantiates the relevance of the topic and formulates the tasks.

The first chapter analyzes the subject area, determines the relevance of the problem and the purpose of development, formulates the task statement, defines the requirements for software implementation, technologies and software tools.

The second section analyzes existing solutions, selects a development platform, designs and develops a web-based information system, describes the system operation, algorithm and structure of its functioning, as well as the application call and download, defines input and output data, and characterizes the composition of technical means parameters.

The practical significance lies in the creation of a web application that provides: creating requests by people in need of assistance, processing them by volunteers, viewing all available requests, laying the optimal route between the locations of requests in accordance with the task of the salesman.

The relevance of developing an information system for automating volunteer activities is obvious in the context of war, during which information systems are increasingly used to significantly accelerate the provision of targeted assistance.

Key words: INFORMATION SYSTEM, VOLUNTEERING, DATABASE, SPRING BOOT, ANGULAR, POSTGRESQL

ЗМІСТ

РЕФЕРАТ 3
ABSTRACT 4
ВСТУП 7
РОЗДІЛ 1 8
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ 8
1.1. Загальні відомості з предметної галузі 8
1. 1. 1 Особливості волонтерських систем у сучасних реаліях України 8
1.1.2. Аналіз аналогів 8
1.2. Призначення розробки та галузь застосування 12
1.3. Підстава для розробки 13
1.4. Постановка завдання 13
1.5. Вимоги до програми або програмного виробу 14
1.5.1. Вимоги до функціональних характеристик 14
1.5.2. Вимоги до інформаційної безпеки 14
1.5.3. Вимоги до складу та параметрів технічних засобів 14
1.5.4. Вимоги до інформаційної та програмної сумісності 14
РОЗДІЛ 2 16
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ 16
2.1. Функціональне призначення системи 16
2.2. Опис застосованих математичних методів 16
2.3. Опис використаних технологій та мов програмування 17
2.4. Опис структури програми та алгоритми її функціонування 22
2.4.1. Структура системи 22
2.4.2. Структура бази даних 42
2.5. Обґрунтування та організація вхідних та вихідних даних програми 44
2.6. Опис розробленої системи 46
2.6.1. Використані технічні засоби 46
РОЗДІЛ 3. 59
ЕКОНОМІЧНИЙ РОЗДІЛ 59
3.1. Розрахунок трудомісткості та вартості розробки інформаційної системи 59
3.2. Розрахунок витрат на створення програми 63
ВИСНОВКИ 66
ДОДАТОК А 70
ЛІСТИНГ ПРОГРАМИ 70
ДОДАТОК Б 91
ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ 91
ДОДАТОК В 92
ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ 92

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

БД – база даних.

JWT – json web token.

JSON – javascript object notation.

ORM – object-relational mapping.

HTML – hypertext markup language.

CSS – cascading style sheets.

SQL – structured query language.

URL – uniform resource locator.

ВСТУП

Війни завжди були рушієм прогресу саме через те, що необхідно вирішувати проблеми що стосуються життя та здоров’я людей максимально якісно, швидко та ефективно. Ця війна не стала вийнятком.

Актуальність розробки системи що дозволяє швидко та якісно задовольняти потреби людей через ефективні сучасні механізми комунікацій а також прокладати оптимальний маршрут між місцями знаходження людей у скрутному становищі на основі математичних розрахунків важко перебільшити.

Об'єктом дослідження є веб-орієнтований додаток для управління волонтерською діялністю.

Мета кваліфікаційної роботи: розробка програмного забезпечення інформаційної системи управління волонтерською діялністю задля підвищення ефективності комунікації між волонтерами та люди у потребі та оптимізації маршрутів роботи волонтерів.

Для вирішення поставленої мети необхідно вирішити наступні задачі:

- розглянути особливості подібних систем в Україні:
- сформувати основні вимоги щодо розробленого програмного продукту (вимоги до функціональних характеристик, інформаційної безпеки, складу та параметрів технічних засобів, інформаційної та програмної сумісності, змісту інформаційного наповнення системи);
- здійснити розрахунок трудомісткості та витрат на розробку програмного продукту.

Практичне завдання полягає у створенні веб-орієнтованого додатку, що забезпечує: можливість швидко та легко створити запит на допомогу за геолокацією, швидко комунікувати із волонтером через окремий чат, відстежувати статус виконання замовлення, можливість надати оцінку роботи волонтера, можливість для волонтера отримати оптимальний шлях виконання замовлень згідно з важливих чинників.

РОЗДІЛ 1
АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1. Загальні відомості з предметної галузі

1. 1. 1 Особливості волонтерських систем у сучасних реаліях України

Після початку повномасштабного вторгнення дуже стрімко почали розвиватися усіляки програмні застосунки, що полегшують роботу волонтерського напрямку. Стало очевидним що задля уникнення непорозумінь, махінацій, зменшення часу, який необхідний для надання допомоги, можна використати інформаційні технології як засіб автоматизаціх. Це дозволяє людині у скруті швидше знайти волонтера за допомогою централізації надання цих послуг – якщо людина шукає помічі, вона точно знає куди можна оперативно звернутись.

1.1.2. Аналіз аналогів

Для аналізу обрано сайти UAhelpers [1] та «Волонтерство в Україні» [2], зображення дизайну яких показано на рис.

На обох сайтах наявна головна сторінка, яка включає в себе:

- інформаційні банери (рис. 1.1. – 1.2.);
- усі наявні запити (рис.1.3.);
- усі наявні категорії запитів, у вигляді фільтрів (рис.1.4. та 1.5.);
- створення запиту (рис.1.6.);

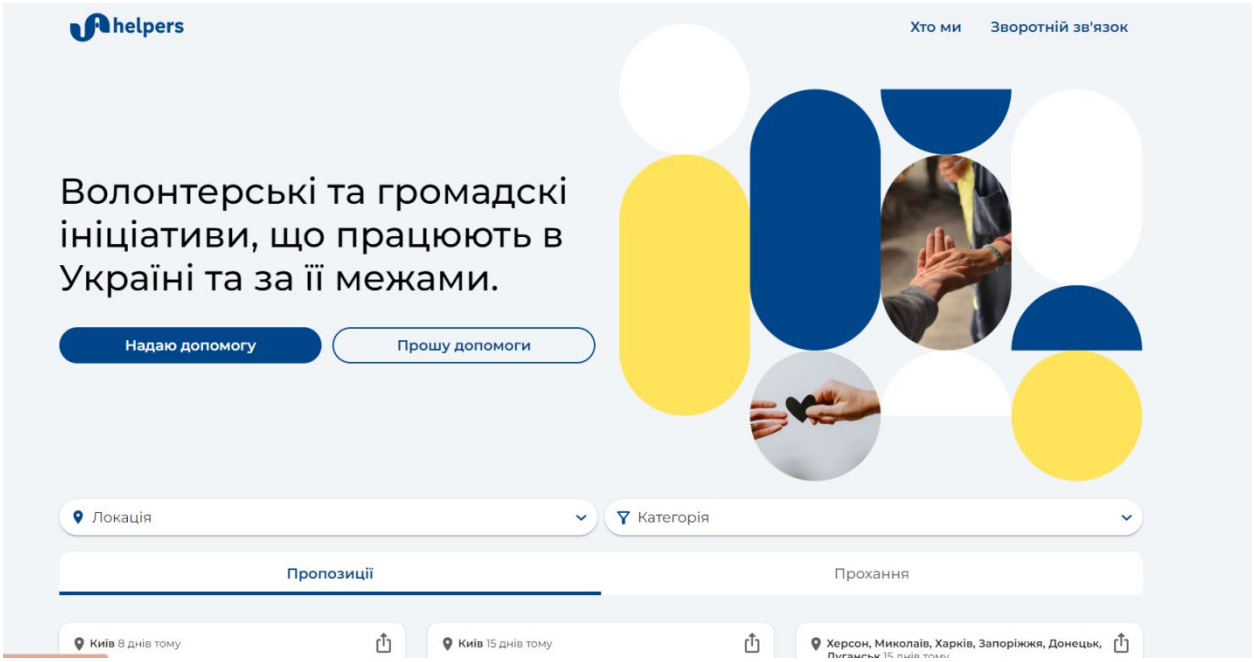

Рис. 1.1. Банери на сайті UAhelpers

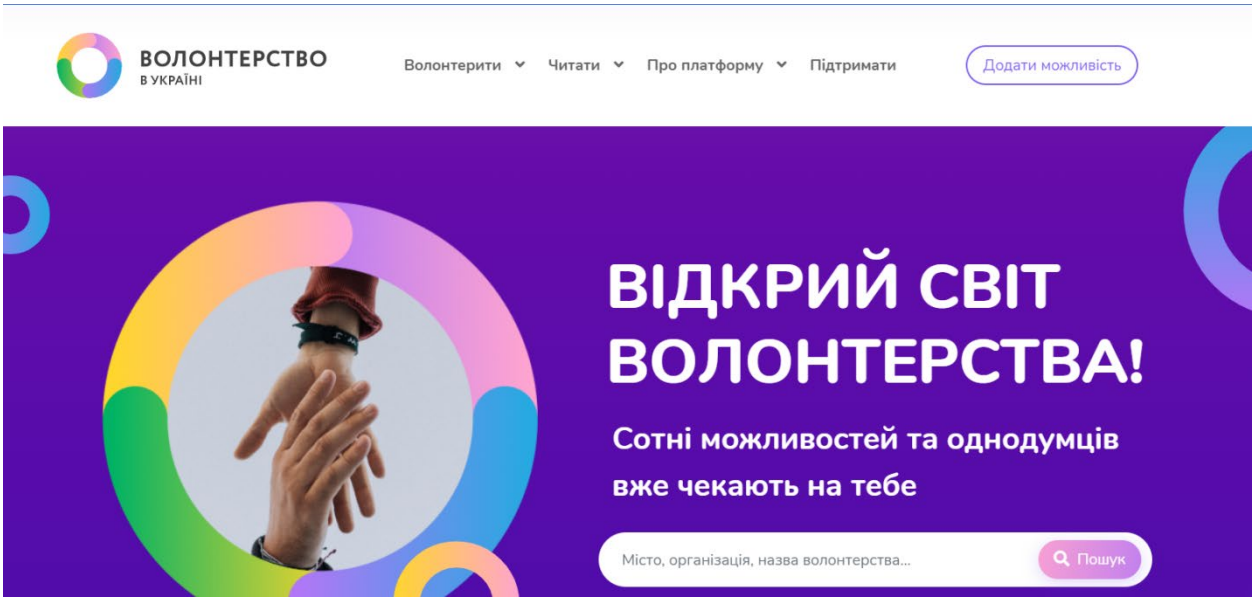

Рис. 1.2. Банери на сайті волонтерство в Україні

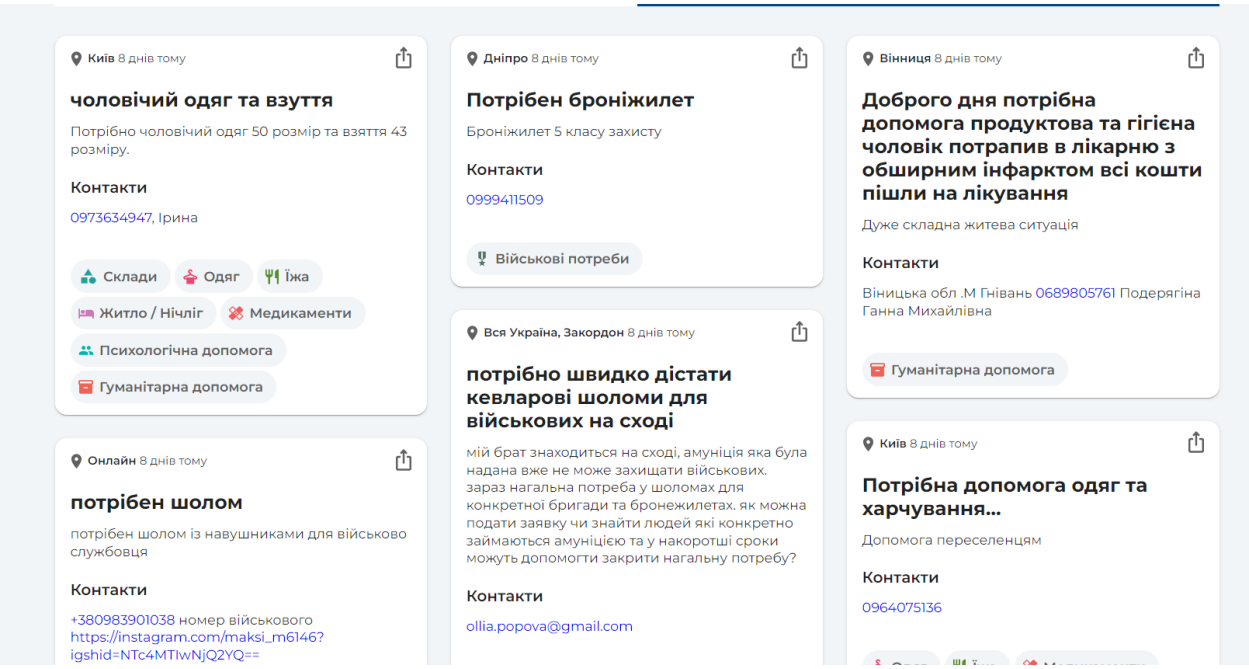

Рис. 1.3. Останні запити на сайті UAhelpers

Рис. 1.4. Категорії на сайті UAhelpers

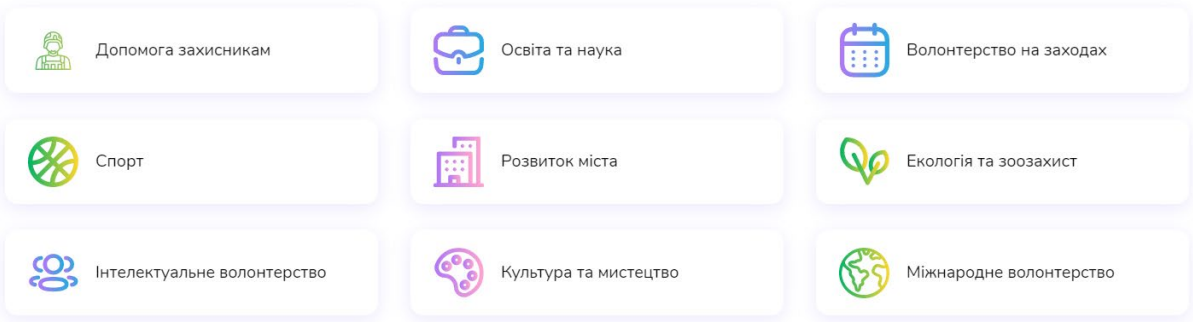

Рис. 1.5. Категорії на сайті Волонтерської Платформи

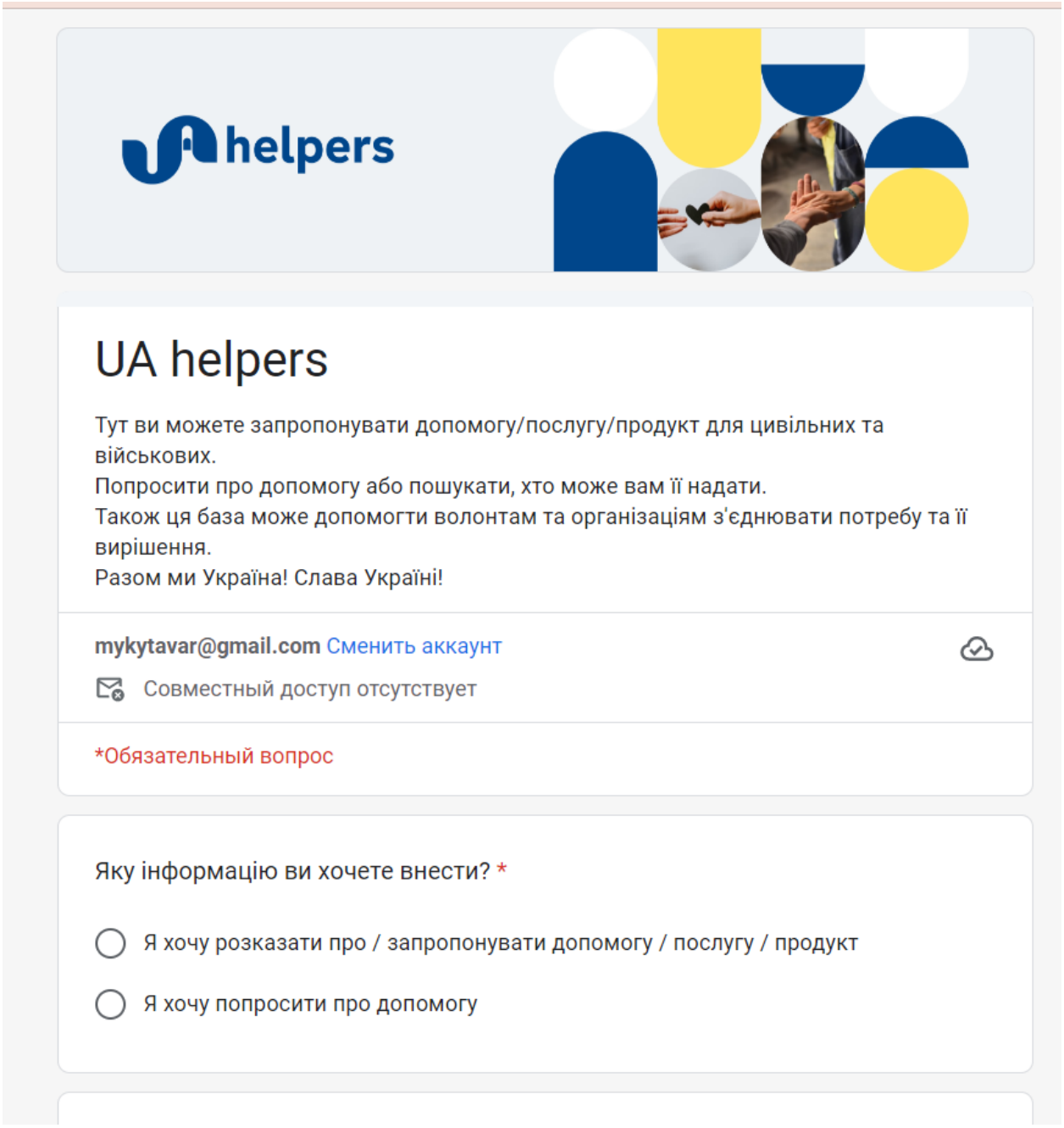

Рис. 1.6. Створення запиту

1.2. Призначення розробки та галузь застосування

Як об’єкт впровадження розроблюваної веб-орієнтованої інформаційної системи для волонтерської діяльності.

Веб-орієнтований додаток повинен забезпечувати такі функції як:

- перегляд усіх запитів;
- перегляд статусу виконання запиту;
- отримання відгуків;
- прокладення маршрутів за задачею Коммівояжера;
- чат між волонтером та людиною у потребі.

Проведений аналіз існуючих технічних рішень сформував основні вимоги, щодо сучасної системи управління волонтерською діяльністю:

- можливість швидкої комунікації з волонтером стосовно конретного запиту на платформі;
- можливість отримання рекомендацій стосовно прокладення можливого шляху з урахуванням різних факторів;
- можливість відстеження статусу заяви.

Аналізуючи вищезазначені вимоги, було зроблено висновок про те, що обов'язково буде реалізовано в проекті:

- головна сторінка сайту має усі запити з можливістю фільтрації за категоріями та локаціями;
- у профілі користувача можна передивитися свої запити а також переглянути статус їх виконання, а також перейти на сторінку окремого запиту;
- на сторінці запиту можна залишити відгук та передивитися точну адресу;
- також волонтер може отримати рекомендації стосовно можливого маршруту.

1.3. Підстава для розробки

Відповідно до ОПП, згідно навчального плану та графіків навчального процесу, в кінці навчання студент виконує кваліфікаційну роботу (проект). Тема роботи узгоджується з керівником проекту, випускаючою кафедрою, та затверджується наказом ректора.

Отже, підставами для розробки (виконання кваліфікаційної роботи) є:

- ОПП за спеціальністю 122 «Комп’ютерні науки»;
- навчальний план та графік навчального процесу;
- наказ ректора Національного технічного університету «Дніпровська політехніка» № 350-с від 16.05.2023 р;
- завдання на дипломний проект на тему «Розробка системи автоматизованого управління волонтерською діяльністю».

1.4. Постановка завдання

В даній кваліфікаційний роботі розглядається створення веб-орієнтованого додатку для управління волонтерською діяльністю.

Основними характеристиками розробки мають бути:

- розгортання бази даних PostgreSQL;
- створення бекенд сервісу для обробки REST запитів та збереження корисного навантаження повідомлень у базі даних, для розрахунку за формулою комівояжера найкращого маршруту між обраними волонтером локацій;
- створення фронтенд додатку за допомогою фреймворку Angular, що буде підтримувати роботу з Google maps для вказання геолокації, а також прокладення на них можливого маршруту.

1.5. Вимоги до програми або програмного виробу

1.5.1. Вимоги до функціональних характеристик

Для досягнення поставленої в роботі мети інформаційна система, що розробляється:

- повинна надавати можливість переглянути особистий профіль;
- повинна надавати доступ переглянути усі наявні запити у системі;
- повинна надавати можливість узяти у обробку запит для волонтера;
- повинна видавати рекомендований маршрут для волонтера;
- повинна дозволяти лише заявнику завершити запит.

1.5.2. Вимоги до інформаційної безпеки

- Аутентифікація повинна бути виконана за допомогою JWT токенів;
- Система повинна не дозволяти csrf атаки;
- Доступ до різних даних визначається роллю користувача;
- Валідація даних виконується на сервері та клієнті.

1.5.3. Вимоги до складу та параметрів технічних засобів

- Вільне місце для БД – як найменьше 1ГБ;
- Мінімальна кількість ядер – 4, для підтримки багатопотоковості;
- Мінімальна кількість оперативної пам’яті – 8 ГБ;
- Відеокарта з мінімальною кількістю оп. пам’яті – 4 ГБ.

1.5.4. Вимоги до інформаційної та програмної сумісності

Для нормального функціонування програми необхідно, щоб програмне забезпечення обчислювальної машини, на якій буде функціонувати веб-орієнтована підсистема, відповідало наступним вимогам:

- операційні системи Windows 10-11, Linux;

– інтернет браузер (Google Chrome, Mozilla FireFox, Microsoft Edge) – останні дві версії.

Для використання бібліотеки Angular 16 знадобляться:

– Node.js версії 16.4.0+;
– версія Typescript 4.9.3+;
– версія RxJS від 6.5.3.

Для використання Spring Boot версії 3.1.0 необхідно встановити:

– Java 17;
– Spring Framework 6.0.9 або вище;
– Maven версії 3.6.3 або вище;

РОЗДІЛ 2
ПРОЕКТУВАННЯ ТА РОЗРОБКА ІНФОРМАЦІЙНОЇ СИСТЕМИ

2.1. Функціональне призначення системи

Веб-орієнтований додаток для управління волонтерської діяльності призначений для організованого пошуку заявок онлайн. Виходячи з функційональних вимог система повинна надавати:

- можливість створювати свої запити як людина у потребі та брати в обробку як волонтер;
- можливість змінювати та відстежувати статус запиту;
- можливість перегляду усіх запитів у системі;
- можливість перегляду усіх запитів що узяті до обробки для волонтера та можливість перегляду усіх створених запитів для людини у потребі;
- можливість для волонтера перегляду запропонованого системою маршруту між локаціями.

Експлуатаційне призначення системи включає:

- полегшити пошук волонтерів та заявок;
- полегшити для волонтера планування маршруту за допомогою рекомендацій;
- можливість відстежувати статус заявки та сконтактувати із відповідальною людиною.

2.2. Опис застосованих математичних методів

Розглянемо граф

$$G = (V, E) \quad (1.1),$$

де V представляє множину міст, а E - множину зважених ребер. Ребро e(u, v) показує зв'язок між вершинами u та v. Відстань між вершинами u та v позначається як d(u, v) і завжди є невід'ємною. Мета полягає в тому, щоб починаючи з міста i, відвідати всі міста множини S по одному разу та

повернутися до початкового міста i. Тобто потрібно знайти найкоротший маршрут, який пройде через всі міста. Припустимо, що ми починаємо з міста 1 і після відвідування деяких міст знаходимося в місті k. Це визначає, які міста будуть найбільш зручними для відвідування далі. Також нам потрібно знати, які міста ми вже відвідали, щоб уникнути повторень. Позначимо g(i, S) як вартість найкоротшого маршруту, який відвідає кожне місто з множини S рівно один раз, починаючи з 1 і закінчуючи в місті i.

Ми розпочнемо з усіх підмножин розміру 2 і обчислимо g(i, S) для кожної такої підмножини, де S - підмножина. Потім обчислимо g(i, S) для підмножин розміру 3 і так далі. Варто зазначити, що у кожній підмножині повинно бути присутнє місто 1. Ми повинні почати з міста 1 і закінчити в місті k. Щоб обрати наступне місто, ми повині вибрати таке k, щоб виконувалась умова:

$$g(i, S) = \min(Cik + g(k, S-\{k\})) \qquad (1.2),$$

де k є елементом множини S.

Тут i - початковий пункт маршруту. [4]

2.3. Опис використаних технологій та мов програмування

Для реалізації веб-орієнтованого додатку було використано наступні технологої:

- Spring (Boot, Security, Data JPA) з використанням Java;
- Angular з використанням Typescript, NgRx, RxJ;
- PostgreSQL;
- HTML, CSS;
- JWT tokens.

Java - це потужна та універсальна мова програмування, яка була розроблена у 1995 році компанією Sun Microsystems (зараз вона належить компанії Oracle). Java є об'єктно-орієнтованою мовою програмування, що означає, що вона підтримує концепцію об'єктів, класів, успадкування, поліморфізму і багато іншого.

Основні особливості мови Java:

– переносимість: однією з головних переваг Java є її здатність працювати на різних платформах без необхідності зміни коду. Це досягається завдяки використанню віртуальної машини Java (JVM), яка перетворює Java-код в байт-код, зрозумілий для будь-якої платформи, що має JVM;
– об'єктно-орієнтований підхід: Java сприяє створенню модульних програм, орієнтованих на об'єкти. Вона підтримує концепцію класів та об'єктів, які дозволяють програмістам легко організовувати та керувати кодом;
– безпека: у Java вбудована система безпеки, що має на меті захистити від підступів та зловживань. Вона використовує механізми, такі як контроль доступу, перевірку типів та обмеження пам'яті, щоб забезпечити безпечну віртуальну середу для виконання програм;
– велика екосистема: Java має велику і активну спільноту розробників, що призводить до широкої наявності бібліотек, фреймворків та інструментів для розробки програм. Це дозволяє програмістам ефективно використовувати готові рішення та прискорювати розробку програм;
– мультипоточність: Java має вбудовану підтримку мультипотоковості, що дозволяє програмістам створювати багатопотокові програми. Це особливо корисно для завдань, які можуть бути виконані паралельно, таких як обробка великих обсягів даних або одночасне обслуговування багатьох користувачів;
– простота використання: Java має простий синтаксис, який легко вивчити та розуміти. Вона надає програмістам багато вбудованих функцій та інструментів для швидкого розробки програм. [3]

Spring Framework — це програмний каркас з відкритим кодом та контейнери з підтримкою інверсії керування для платформи Java.

Основні особливості Spring Framework можуть бути використані будь-яким застосунком Java, але є розширення для створення вебдодатків на платформі Java EE. Попри це, Spring Framework не нав'язує якоїсь конкретної моделі програмування, Spring Framework став популярним у спільноті Java як альтернатива, або навіть доповнення моделі Enterprise JavaBean (EJB).

Spring Boot дозволяє легко створювати автономні додатки на основі Spring промислового рівня, які можна "просто запустити".

Можливості:

- створюйте автономні Spring-програми;
- вбудовуйте Tomcat, Jetty або Undertow безпосередньо (не потрібно розгортати WAR-файли);
- надайте власні "стартові" залежності, щоб спростити конфігурацію збірки;
- автоматичне налаштування бібліотек Spring та сторонніх розробників, коли це можливо;
- надавати готові до виробництва функції, такі як метрики, перевірки працездатності та зовнішні конфігурації;
- абсолютна відсутність генерації коду та відсутність вимог до конфігурації XML [5].

Місія Spring Data полягає у забезпеченні зручної та послідовної моделі програмування на основі Spring для доступу до даних, зберігаючи особливості конкретного сховища даних.

Це спрощує використання технологій доступу до даних, таких як реляційні і нереляційні бази даних, фреймворки для картографічного відображення та хмарні сервіси даних. Spring Data є парасольковим проектом, який включає багато підпроектів, спеціалізованих для різних баз даних. Ці проекти розробляються спільно з компаніями та розробниками, які активно використовують ці захоплюючі технології.

Основні можливості Spring Data включають:

- потужний репозиторій та абстракції відображення об'єктів;
- динамічне створення запитів на основі імен методів репозиторію;
- реалізація базових класів домену, що надають основні функціональності;
- підтримка прозорого аудиту (запис створення та останньої модифікації);

- можливість інтеграції власного коду у репозиторій;
- легка інтеграція з Spring за допомогою JavaConfig та власних файлів конфігурації XML;
- розширена інтеграція з контролерами Spring MVC;
- завдяки цим можливостям розробники можуть швидко та ефективно працювати з базами даних, використовуючи усі переваги Spring Framework;
- експериментальна підтримка міжсховищної персистентності [6].

TypeScript є мовою програмування, яка базується на JavaScript і має сильну систему типів. Вона надає розширений синтаксис, який сприяє кращій інтеграції з редактором і дозволяє виявляти помилки на ранніх етапах розробки.

Однією з переваг TypeScript є те, що код, написаний на ній, перетворюється в стандартний JavaScript, який може бути виконаний у будь-якому середовищі, де працює JavaScript: у веб-браузерах, на платформі Node.js або Deno, а також у вашому додатку.

TypeScript здатний розуміти і інтерпретувати код JavaScript і автоматично виводити типи з нього, що надає вам потужні інструменти без необхідності додаткового написання коду [7].

Angular - це потужний фреймворк для розробки веб-додатків. Він дозволяє створювати високоякісні і динамічні веб-інтерфейси, забезпечуючи розширені можливості для розробників.

Одна з ключових особливостей Angular - це використання TypeScript, сильно типізованої мови програмування, яка розширює стандартний JavaScript і дозволяє виявляти помилки на етапі розробки. Це сприяє покращенню безпеки, підвищенню продуктивності та покращенню підтримки редактора коду.

Основною концепцією Angular є використання компонентної архітектури. Ви будуєте свою програму з роздільних блоків, які називаються компонентами. Кожен компонент включає в себе шаблон HTML для відображення вмісту, стилі для оформлення та логіку, написану на TypeScript. Компоненти взаємодіють один з одним, обмінюючись даними та подіями.

Angular надає багатий набір інструментів і функціональностей, які спрощують розробку веб-додатків. Це включає в себе механізми для роботи з маршрутизацією (навігацією між сторінками), управлінням формами, валідацією даних, роботою з HTTP-запитами, анімаціями та багато іншого.

Крім того, Angular пропонує широкі можливості для тестування, що дозволяє вам перевірити функціональність вашого додатку і забезпечити його якість.

Загалом, Angular є потужним і гнучким фреймворком, який дозволяє розробникам ефективно створювати складні веб-додатки з високою продуктивністю та відмінним користувацьким досвідом [8].

Angular NgRx - це бібліотека станового управління для Angular, яка базується на концепції Redux. Вона надає інструменти та шаблони для ефективного керування станом додатків і дозволяє підтримувати прогнозований, не залежний від стану стан додатку.

Основна ідея NgRx полягає в тому, що стан додатку представляється як незмінний об'єкт, який зберігається в одному централізованому місці - це називається Store. Компоненти можуть взаємодіяти зі Store, відправляючи та підписуючись на дії (actions) та вибірки (selectors).

Дії - це об'єкти, які описують події або зміни, що відбуваються в додатку. Компоненти відправляють дії до Store, і вони потрапляють до редукторів (reducers). Редуктори - це функції, які приймають поточний стан і дію, і повертають новий стан на основі цих даних. Редуктори завжди повинні бути чистими функціями без побічних ефектів.

Селектори - це функції, які дозволяють вибирати певну частину стану зі Store. Вони використовуються для отримання необхідних даних зі стану, які потрібні компонентам для відображення.

NgRx також надає можливість використовувати побічні ефекти (side effects) за допомогою бібліотеки RxJS. Побічні ефекти дозволяють виконувати асинхронні операції, такі як HTTP-запити, обробку локального сховища або взаємодію з зовнішніми сервісами [9].

REST - це архітектурний стиль, який використовується для проектування мережевих додатків. REST базується на принципах, які сприяють побудові масштабованих, легко зрозумілих та зручних для розробки систем.

Основні принципи REST:

- Клієнт-сервер: Система розбивається на дві частини - клієнт та сервер, які взаємодіють між собою через стандартизовані інтерфейси;
- Безстанність: Кожен запит клієнта до сервера містить всю необхідну інформацію для обробки запиту. Сервер не зберігає стан клієнта між запитами, що дозволяє полегшити масштабування системи;
- Кешування: Сервер може повертати відповідь з можливістю кешування, щоб клієнтам не потрібно було постійно звертатися до сервера для отримання даних;
- Єдиносвітний інтерфейс: Клієнти взаємодіють з сервером через єдиний та уніфікований набір інтерфейсів, таких як HTTP методи (GET, POST, PUT, DELETE);
- Структуровані ресурси: Кожний ресурс системи має свій унікальний ідентифікатор та може бути доступний через стандартний URL [20].

2.4. Опис структури програми та алгоритми її функціонування

2.4.1. Структура системи

Структура клієнт-серверних додатків з використанням Spring Boot та Angular може виглядати наступним чином:

Клієнтська частина (Frontend):

- використання Angular: Angular є популярним фреймворком JavaScript, який дозволяє розробляти потужні та динамічні веб-додатки. Angular надає різні інструменти та компоненти для побудови користувацького інтерфейсу;

– користувацький інтерфейс: Angular дозволяє створювати зручний інтерфейс для взаємодії з користувачем, який може включати форми, таблиці, списки та інші елементи;
– взаємодія з сервером: Клієнтська частина може виконувати HTTP запити до сервера, використовуючи Angular HTTP модуль, для отримання або збереження даних.

Серверна частина (Backend):

– використання Spring Boot: Spring Boot є фреймворком для розробки Java-додатків, який спрощує створення серверної частини додатків. Він надає різні модулі та інструменти для швидкої розробки;
– роутинг та контроллери: за допомогою Spring Boot можна визначити маршрути та контролери, які обробляють HTTP запити від клієнта. Контролери виконують необхідні операції над даними та повертають відповіді клієнту;
– бізнес-логіка: у серверній частині може бути реалізована бізнес-логіка додатку, така як обробка даних, перевірка дозволів, взаємодія з базою даних та інші операції.
– взаємодія з базою даних: Spring Boot дозволяє легко використовувати ORM (Object-Relational Mapping) для зв'язку з базою даних, таку як Hibernate або Spring Data JPA.

Загалом, структура клієнт-серверного додатку з використанням Spring Boot та Angular включає різні шари на обох сторонах. Зі сторони бекенду:

– контролери для обробки HTTP запитів та маршрутизації до відповідних обробників;
– сервіси, які містять бізнес-логіку та обробляють операції над даними;
– репозиторії або DAO (Data Access Objects) для взаємодії з базою даних.

На фронтенді (клієнтській частині) можуть бути такі шари:

– компоненти, які представляють окремі елементи інтерфейсу та мають свою логіку;

– сервіси, які виконують HTTP запити до сервера та обробляють дані;
– ефекти, селектори, фасади, дії та редюсери для зберігання та обробки станів системи за допомогою бібліотеки NgRx;
– моделі, які представляють дані, що отримуються від сервера або введені користувачем.

Ця структура дозволяє розділити відповідальності між клієнтом та сервером, спрощує розробку та підтримку додатків, а також забезпечує масштабованість та швидкодію.

Зі сторони фронтенду можна відзначити наступні компоненти:

– допоміжні компоненти що покликані позбавити кодову базу від повторювань, які поширені між різними компонентами:
  - button.component.ts;
  - request-card.component.ts;
  - request-list.component.ts.
– компоненти, що відповідають основним сторінкам системи:
  - компонент для створення запиту – create-request.component.ts;
  - компонент для відображення головної сторінки для перегляду усіх запитів – requests.component.ts;
  - компонент для входу до системи: login.component.ts;
  - компонент для реєстрації: registration.component.ts;
  - компонент для перегляду даних користувача – profile.component.ts;
  - компонент для перегляду запропонованого маршруту: route.component.ts.

Для поліпшення швидкодії системи підвантаження окремих модулів у системі виконуються «ліниво» - під час старту програми підвантажаються тільки модуль за замовчуванням, а інші частини системи підвантажаються лише після того як до них звернуться через маршрутизацію:

```
const routes: Routes = [
{
```

```
    path: '',
    redirectTo: 'requests',
    pathMatch: 'full'
  },
  {
    path: 'requests',
    loadChildren: () => import('./feature/requests/requests.module').then(m => m.RequestsModule)
  },
  {
    path:'route',
    loadChildren: () => import('./feature/route/route.module').then(m => m.RouteModule),
    canActivate: [VolunteerGuard]
  },
  {
    path: 'create-request',
    loadChildren: () => import('./feature/create-request/create-request.module').then(m =>
m.CreateRequestModule),
    canActivate: [RequesterGuard]
  },
  {
    path: 'registration',
    loadChildren: () => import('./feature/registration/registration.module').then(m =>
m.RegistrationModule)
  },
  {
    path: 'login',
    loadChildren: () => import('./feature/login/login.module').then(m => m.LoginModule)
  },
  {
    path: 'profile',
    loadChildren: () => import('./feature/profile/profile.module').then(m => m.ProfileModule),
    canLoad: [AuthorizedGuard]
  },
  {
    path: '**',
```

```
  loadChildren: () => import('./feature/requests/requests.module').then(m => m.RequestsModule)
 }
]
```

Для того щоб отримати спрощене та централізоване управління станів у системі було використано бібліотеку NgRx. Таким чином відокремлюється логіка обробки даних з сервера та створення запитів від бізнес-логіки. Це досягається шляхом створення подій(actions) що потім відправляється на обробку до фреймворку, який перенаправляє цю подію до наших самописних функцій обробки. Спочатку виконується так званий ефект, що налаштований на обробку окремої події або декількох подій, який робить запит до бекенду а також сам потім повертає подію – що визначає або успіх або помилку для окремої події. Відповідно до цих результуючих подій оновлюєтья внутрішній стан на стороні фронтенду за допомогою редюсерів – функцій що також реагують на події, але заданим чином модифікують дані саме на фронтенді. Отримання даних зі стану виконується за допомогою селекторів.

Приклад оголошення подій:

```
export const requestCreateRequest = createAction('[Request] Create', props<{request:
CreateRequestDto}>())
export const requestCreateRequestSuccess = createAction('[Request] Create success',
props<{ request: Request }>())
export const requestCreateRequestFail = createAction('[Request] Create fail',
props<{ errorMessage: string }>())
```

Приклад відправлення події:

```
createRequest(request: CreateRequestDto) {
 this.store.dispatch(requestCreateRequest({ request }))
}
```

Приклад обробки події у ефекті:

```
createRequest$ = createEffect(() =>

 this.actions$.pipe(
  ofType(requestCreateRequest),
  exhaustMap(((action) => {
```

```
      console.log('create request effect fired')
      return this.requestService.createRequest(action.request)
       .pipe(map(request => {
         return requestCreateRequestSuccess({request: request});
        }),
        catchError((res: HttpErrorResponse) => of(requestCreateRequestFail({errorMessage:
res.message})))
    })
  )));
```

Приклад обробки події у редюсері:

```
export const reducer = createReducer(initialState,
 on(requestCreateRequest, (state) => ({
  ...state
 })),
 on(requestCreateRequestSuccess, (state, {request}) => ({
  ...state,
  allRequests: [...state.allRequests, request]
 })),
 on(requestCreateRequestFail, (state, {errorMessage}) => ({
  ...state,
  errorMessage: errorMessage
 })))
```

Приклад отримання даних у селекторі:

```
export const getAllRequests = createSelector(getRequestState,
 (state: RequestState) => state.allRequests)
```

Для того щоб запобігти доступу користувачів з однією роллю до контенту що доступний лише іншій ролі, зі сторони фронтенду було використано захисні фільтри (guards).

Приклад захисного фільтра для ролі волонтер:

```
@Injectable({
 providedIn: 'root'
})
export class VolunteerGuard implements CanActivate {
```

```
  constructor(private router: Router, private userStoreService: UserStateFacade) {
  }

  canActivate(
    route: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean | UrlTree> |
boolean | UrlTree {
    return this.userStoreService.isVolunteer$.pipe(
      take(1),
      map(isVolunteer => {
        console.log('isAdmin', isVolunteer);
        if(isVolunteer) {
          return true;
        }
        this.router.navigate(['/requests']);
        return false;
      })));
  }

}
```

Для JWT аутентикації потрібно передавати токен у кожному запиті, для чого використовується перехоплювач кожного http запиту, який додає відповідний хедер.

```
        @Injectable({
  providedIn: 'root'
})
export class AuthInterceptorService implements HttpInterceptor {
  constructor(private authFacae: AuthStateFacade) { }
  intercept(request: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {

    console.log('intercepting', request)
    return this.authFacae.getToken$.pipe(take(1), concatMap(token => {
      console.log('token goes', token)
      if (token) {
```

```
      request = request.clone({
        setHeaders: { Authorization: `Bearer ${token}` }
      });

    }

    return next.handle(request).pipe(
      catchError((err) => {
        if (err instanceof HttpErrorResponse) {
          if (err.status === 401) {
            console.log('loging out in interceptor')
            this.authFacae.closeSession()
          }
        }
        return throwError(err);
      })
    )
  }))

  }
}
```

Зі сторони бекенду було створено REST API, що має наступні ендпоїнти:

- POST /users – реєстрація
- POST /users/login – логін
- GET /users/me – отримати дані про себе за токеном
- GET /categories – отримати усі типі категорій запитів
- POST /requests - створити запит
- GET /requests – передивитися усі запити
- POST /requests/take/{id} – узяти запит в обробку для волонтера
- GET /requests/my – отримати усі запити що зв’язані з аккаунтом
- POST /requests/complete/{id} – підтвердити вирішення запиту
- GET /requests/route – отримати рекомендований маршрут

Після отримання запиту, шар контроллерів передає запит до шару фасадів, який виступає у ролі оркестраторів різних сервісів що пов'язані с тією чи іншою сутністю. Шар сервісів робить попередню обробку та передає дані на збереження до шару репозиторіїв, що зберігає дані у базі даних. Шар репозиторію використовує інтерфейси-репозиторії Spring Data JPA, що дозволяє за допомогою аспектів Spring під час виконання програми сгенерувати класи що виконують типові запити до бази даних та обробку відповідей[11].

Приклад контролера:

```
@RestController
@RequiredArgsConstructor
@RequestMapping("/requests")
public class RequestController {
   private final RequestFacade requestFacade;
   private final RouteBuilderFacade routeBuilderFacade;

   @PostMapping
   @ResponseStatus(CREATED)
   public Request createRequest(@RequestBody CreateRequestDto createRequestDto,
                        @RequestHeader("Authorization") String token) throws IOException,
InterruptedException {
      return requestFacade.createRequest(createRequestDto, token);
   }
```

Приклад фасаду:

```
@Component
@AllArgsConstructor
public class RequestFacade {
   private final RequestService requestService;
   private final UserFacade userFacade;

   private final LocationService locationService;
   private final JwtTokenProvider jwtTokenProvider;
   private final RequestRepository requestRepository;

   public Request createRequest(CreateRequestDto createRequestDto, String token) throws
```

```
IOException, InterruptedException {
        Location location = locationService.createLocation(createRequestDto.getLocation());
        User requester = userFacade.tryGetUserByEmail(jwtTokenProvider.getEmail(token));

        return requestService.createRequest(createRequestDto, location, requester);
    }
```

Приклад сервісу:

```
@Service
@AllArgsConstructor
public class RequestService {
    private final RequestRepository requestRepository;

    public Request createRequest(CreateRequestDto createRequestDto,
                                 Location location, User requester) {
        Request request = Request.builder()
                .requester(requester)
                .title(createRequestDto.getTitle())
                .location(location)
                .details(createRequestDto.getDetails())
                .need(Need.valueOf(createRequestDto.getNeedType()))
                .status(Status.NOT_STARTED)
                .build();

        return requestRepository.save(request);
    }
```

Приклад репозиторію:

```
        public interface RequestRepository extends PagingAndSortingRepository<Request, UUID>,
CrudRepository<Request, UUID> {

    Iterable<Request> findAllByVolunteerId(UUID id);

    Iterable<Request> findAllByRequesterId(UUID id);
}
```

Зі сторони бекенду ми оголошуємо обмеження доступу до окремих ендпоїнтів за ролями.

```
@Configuration
@EnableWebSecurity
@AllArgsConstructor
@EnableMethodSecurity
public class SecurityConfiguration {
   private final JwtTokenProvider tokenProvider;
   private final UserDetailsService userDetailsService;


   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      http
            .csrf().disable()
         .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
            .exceptionHandling().authenticationEntryPoint((req, rsp, e) -> {
               System.out.println("Authentication exception: " + e.getMessage());
               rsp.sendError(HttpServletResponse.SC_UNAUTHORIZED, e.getMessage());
            })
            .and()
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(new JwtTokenFilter(tokenProvider),
UsernamePasswordAuthenticationFilter.class)
            .authorizeHttpRequests()
            .requestMatchers(HttpMethod.POST, "/users", "/users/login", "/requests").permitAll()
            .requestMatchers(HttpMethod.POST, "/requests",
"/requests/complete/**").hasRole("REQUESTER")
            .requestMatchers(HttpMethod.POST, "/requests/take/**").hasRole("VOLUNTEER")
            .requestMatchers(HttpMethod.GET, "/requests/route").hasRole("VOLUNTEER")
            .requestMatchers(HttpMethod.GET, "/requests", "/categories").permitAll()
            .requestMatchers(HttpMethod.GET, "/users/me").permitAll()
            .anyRequest().fullyAuthenticated();
```

```
    http.cors();

    return http.build();
  }
```

Система також інтегрована із Google Directions and Places API для прокладення маршрутів на мапах та відображення мап. Для цього спочатку треба імпортувати гугл карти до проекта. Треба вказати ключ виданий на гугл платформі, а також необхідні бібліотеки та мову відображення[10].

```
    <script
src="https://maps.googleapis.com/maps/api/js?key=secret&libraries=
visualization,places&language=en">
```

Потім треба вказати тег гугл мапи для її відображення:

```
    <google-map #mapElement width="400px"
    height="300px"
    [center]="center"
    [zoom]="zoom" (mapClick)="onMapClick($event)">
 <map-marker [position]="markerPosition"></map-marker>
</google-map>
```

Для того щоб визначити точне місце за широтою треба використати клас Geocoder від Google. Приклад використання класу:

```
            onMapClick(event: google.maps.MapMouseEvent) {
 this.geocoder.geocode({location: event.latLng}, (results, status) => {
   console.log('results:', results).
```

Алгоритми функціонування системи представлені у вигляді use case diagrams. Діаграми варіантів використання або моделі варіантів використання - це графічне представлення мовою UML (Unified Modeling Language), яке ілюструє функціональні вимоги до системи з точки зору її користувачів (акторів). Діаграми варіантів використання використовуються для зображення взаємодії між акторами (користувачами) і системою для досягнення конкретних цілей або завдань.

На діаграмі варіантів використання основними елементами є актори та варіанти використання:

Актори представляють користувачів або зовнішні сутності, які взаємодіють з системою. Це можуть бути окремі особи, інші системи або навіть апаратні пристрої. На діаграмі актори зображуються у вигляді паличок.

Варіанти використання представляють конкретні функціональні можливості або завдання, які система виконує для того, щоб приносити користь своїм учасникам. Кожен варіант використання описує певну взаємодію або поведінку між актором і системою. Варіанти використання представлені на діаграмі у вигляді овалів і з'єднані з акторами лініями, які називаються асоціаціями [12].

Use Case діаграма зображена на рис.2.1.

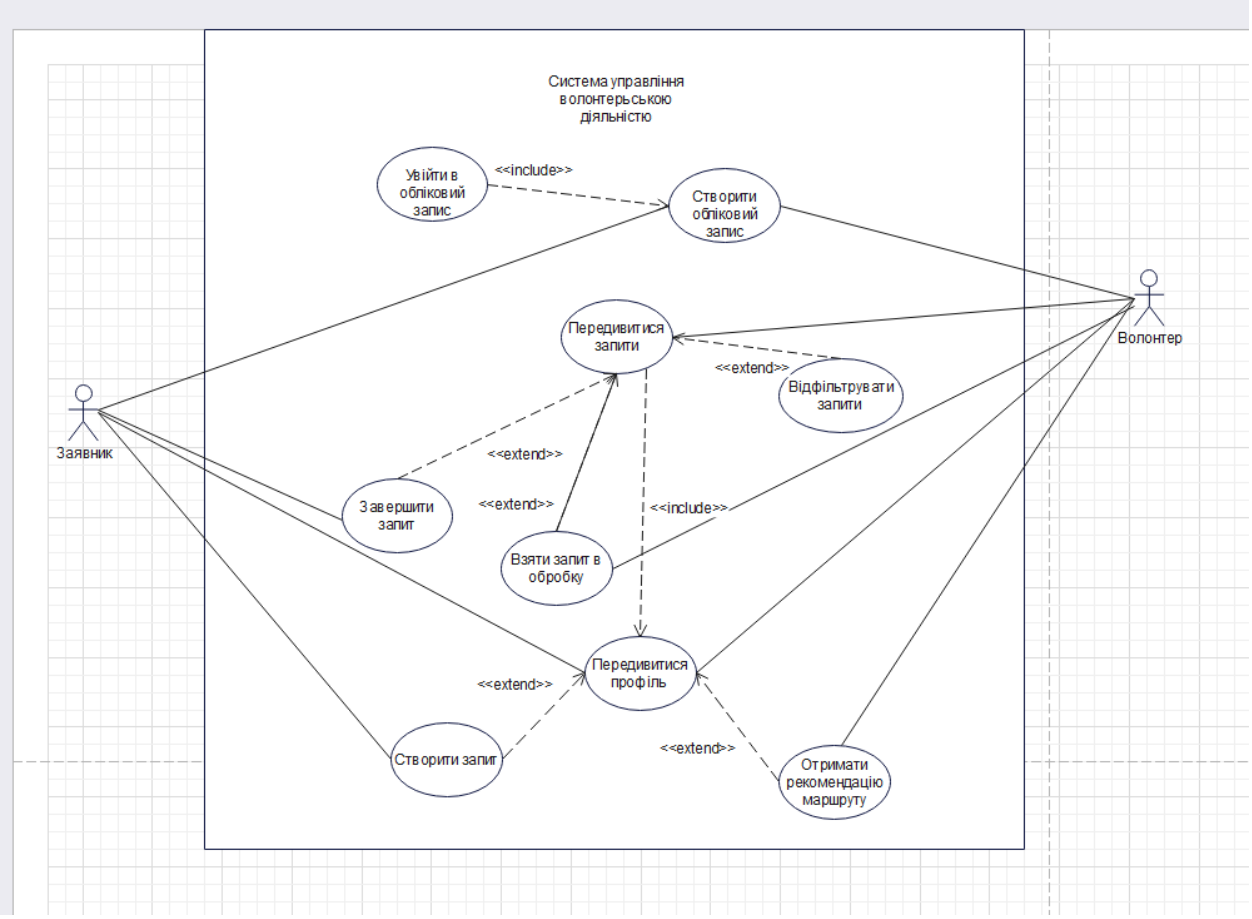

Рис. 2.1. Діаграма Use case

Діаграма активності (також відома як діаграма станів або діаграма активностей) - це візуальна модель, що використовується в рамках уніфікованої мови моделювання (UML) для зображення послідовності дій або активностей у системі, процесі або робочому процесі.

Основні компоненти діаграми активності:

Початковий вузол: Позначає початок активності або процесу. Він зображується у вигляді заповненого кола.

Стан активності: Представляє певну дію або активність у системі. Він зображується у вигляді прямокутника з закругленими кутами. Кожен стан активності може мати вхідні та вихідні потоки.

Розгалужуючий вузол: Представляє точку розгалуження, де потік розгалужується на кілька шляхів на основі певних умов або рішень. Він зображується у вигляді ромба. Кожне розгалуження представляє різні умови або результати рішення.

Вузол об'єднання: Представляє злиття кількох потоків в один потік після точки розгалуження. Він зображується у вигляді ромба з кількома вхідними потоками та одним вихідним потоком.

Вузол розгалуження: Представляє одночасне виконання кількох активностей. Він позначається заповненою смугою, а вихідні потоки з вузла розгалуження показують активності, які виконуються паралельно.

Вузол об'єднання: Представляє синхронізацію кількох одночасних активностей. Він позначається заповненою смугою, а вхідні потоки показують активності, які синхронізуються.

Кінцевий вузол: Позначає кінцеву точку активності або процесу. Він зображується у вигляді заповненого кола з крапкою всередині.

На діаграмі активностей потік активностей зображається стрілками, які з'єднують різні вузли. Стрілки показують послідовність дій та точки розгалуження, а вони можуть мати мітки, що вказують на умови або події, що спричиняють переходи.

Діаграми активностей можуть використовуватися для моделювання широкого спектру систем або процесів, включаючи бізнес-процеси, алгоритми програмного забезпечення, сценарії використання тощо. Вони надають чітку візуалізацію послідовності дій та допомагають стейкхолдерам зрозуміти поведінку системи або процесу.

Загалом, діаграми активностей є цінним інструментом для аналізу, проектування та комунікації динамічних аспектів системи або процесу, що дозволяє стейкхолдерам виявити можливі складнощі, покращити ефективність та забезпечити безперебійне виконання активностей [13].

На діаграмах активностей ми можемо побачити діаграми пов’язані з реєстрацією (рис.2.2.), створенням запиту (рис.2.3.), логіном (рис.2.4), завершенням (рис. 2.5.), обробкою запиту (рис.2.6.) та рекомендацією маршруту (рис. 2.7.).

Рис. 2.2. Діаграма активності: реєстрація

Рис. 2.3. Діаграма активності: створення запиту

Рис. 2.4. Діаграма активності: аутентифікація

Рис. 2.5. Діаграма активності: завершення запиту

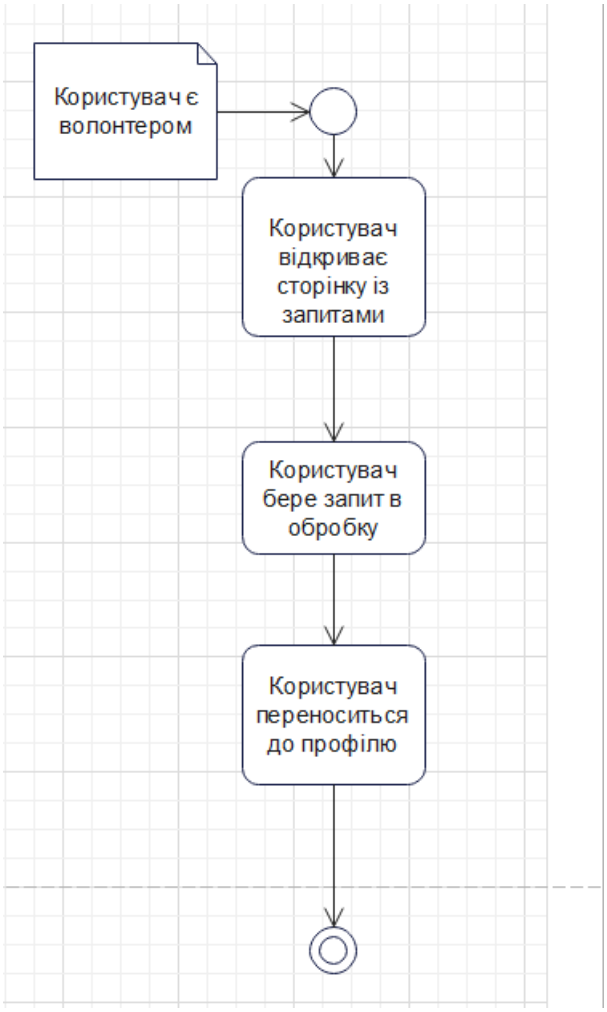

Рис. 2.6. Діаграма активності: обробка запиту

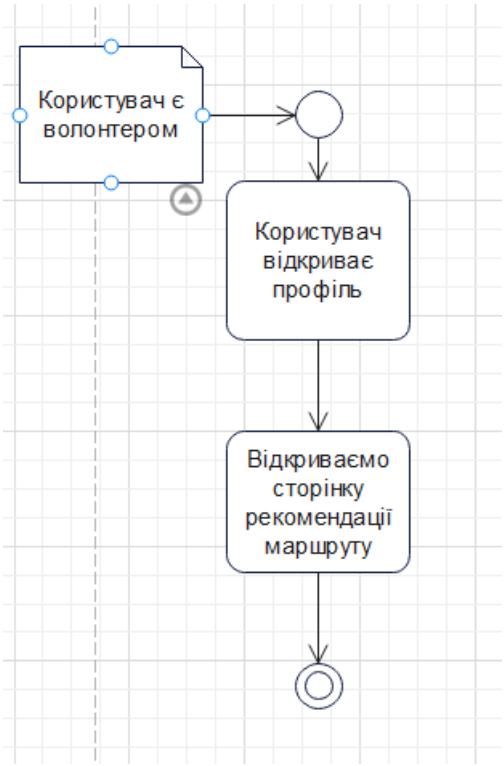

Рис. 2.7. Діаграма активності: рекомендації

2.4.2. Структура бази даних

Як базу даних обрано реляційну PostgreSQL. Для розробленої системи необхідна база даних із чіткою структуру з попередньо визначеною схемою. Основними сутностями у системі є запити (requests), користувачі (users), локації (locations) та ролі (roles).

До сутностей бази даних та їх атрибутів належать:

- таблиця requests відповідає за зберігання інформації про запити заявників:
    - id – первинний ключ, ідентифікатор товару;
    - title – назва запиту;
    - need – вид необхідної допомоги;
    - details – опис запиту;
    - requester_id – ідентифікатор заявника;
    - volunteer_id – ідентифікатор волонтера;
    - location_id – ідентифікатор локації;
    - status – статус запиту.
- таблиця locations відповідає за зберігання конкретної локації:
    - id – первинний ключ, ідентифікатор локації;
    - latitude та longitude -широта та довгота виступають складним первинним ключем;
    - exact_location – рядок що визначає повну адресу локації;
    - nearby_city – рядок що визначає назву населеного пункту (у випадку коли маркер вказано за межами населеного пункту – найближчий населений пункт).
- таблиця roles відповідає за зберігання даних про роль:
    - id - первинний ключ, ідентифікатор ролі;
    - name – назва ролі.
- таблиця users відповідає за зберігання даних про користувача:
    - email – пошта користувача

- id - первинний ключ, ідентифікатор користувача;
- details – деталі користувача;
- role_id – ідентифікатор ролі;
- location_id – ідентифікатор локації;
- password – пароль.

Ролі заносяться до бази даних під час запуску бекенду, у випадку якщо вони ще там не наявні. Локації створюються при вказанні геолокації зі сторони інтерфейсу користувача, якщо локація із такими широтою та довготою не є вже присутніми. Користувач та запит прив’язуються до локації при створенні. До запиту ідентифікатор волонтера додається лише після обранням волонтера заявки. Статус запиту змінюється між NOT_STARTED після створення, IN_PROGRESS після взяття волонтером в обробку та FISHED після завершенням заявником.

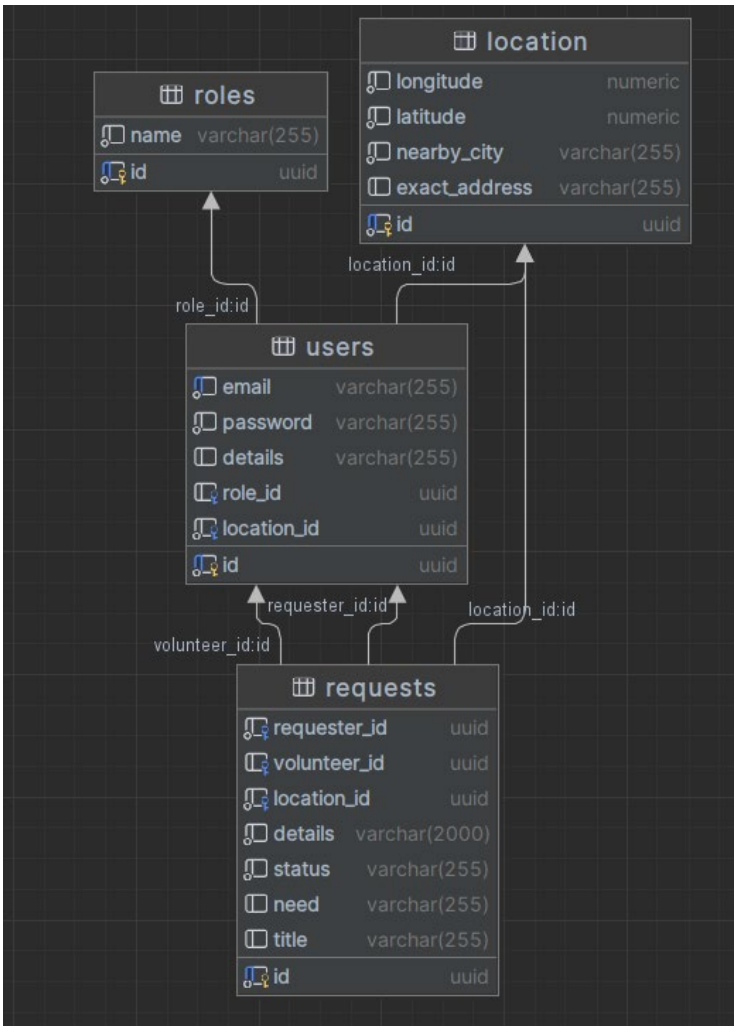

Рис. 2.8. ER діаграма

2.5. Обґрунтування та організація вхідних та вихідних даних програми

Система отримує вхідні дані шляхом запитів до бази даних, вказання геолокації та введення текстових даних зі сторони інтерфейсу користувача, або натиснення кнопок.

Вхідні дані:

- інформація про створюваний запит;
- інформація про користувача;
- геолокація;
- фільтри запитів;
- дані про можливі у системі ролі.

Вихідні дані:

- рекомендація маршруту для волонтера;
- списки усіх запитів;
- список усіх категорій запитів;
- список усіх локацій запитів;
- список запитів що були створені окремим заявником;
- список запитів що були взяті в обробку волонтером;

Отримання вхідних даних за допомогою інтерфейсу користувача відбувається за допомогою реактивних форм фреймворку Angular. Під час входу до системи необхідно заповнити поля email та паролю (рис.2.9).

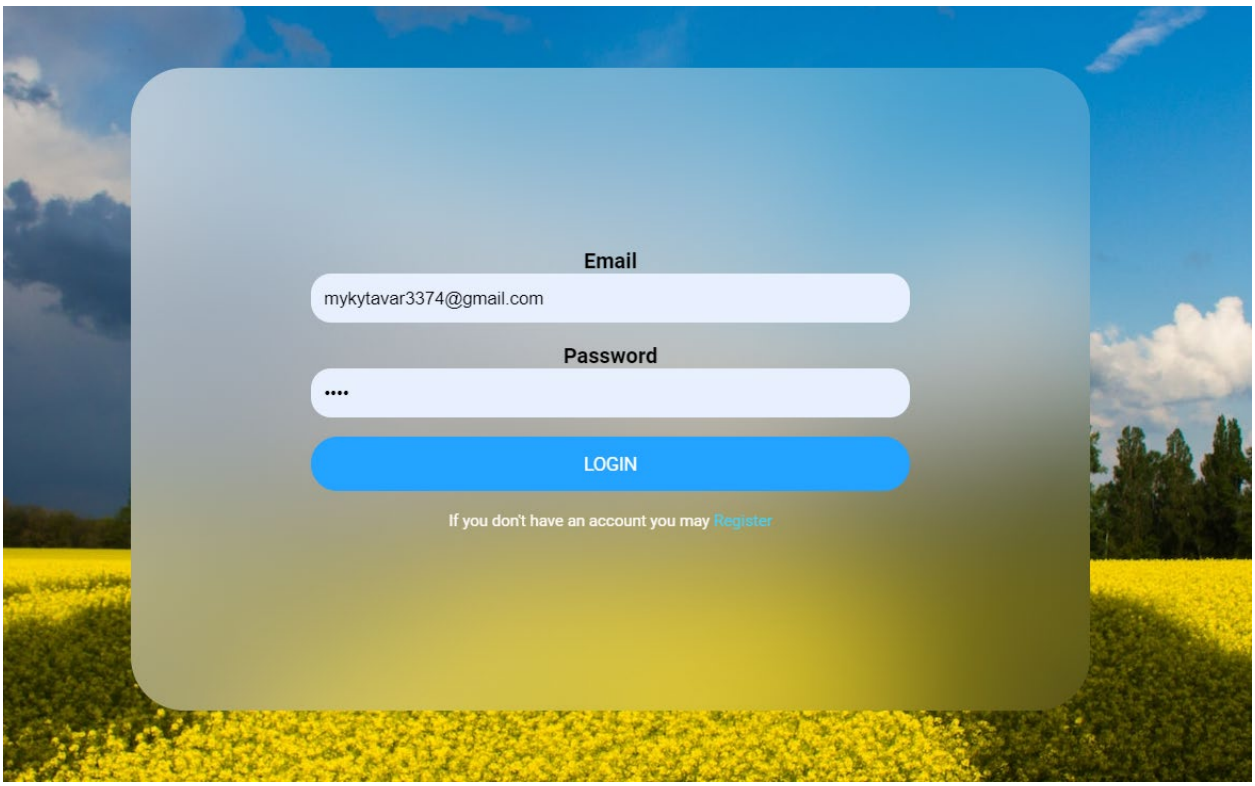

Рис. 2.9. Поля вводу аутентифікації

Під час реєстрації необхідно заповнити поля email, пароль, дані про себе, обрати свою роль та вказати геолокацію (рис.2.10.).

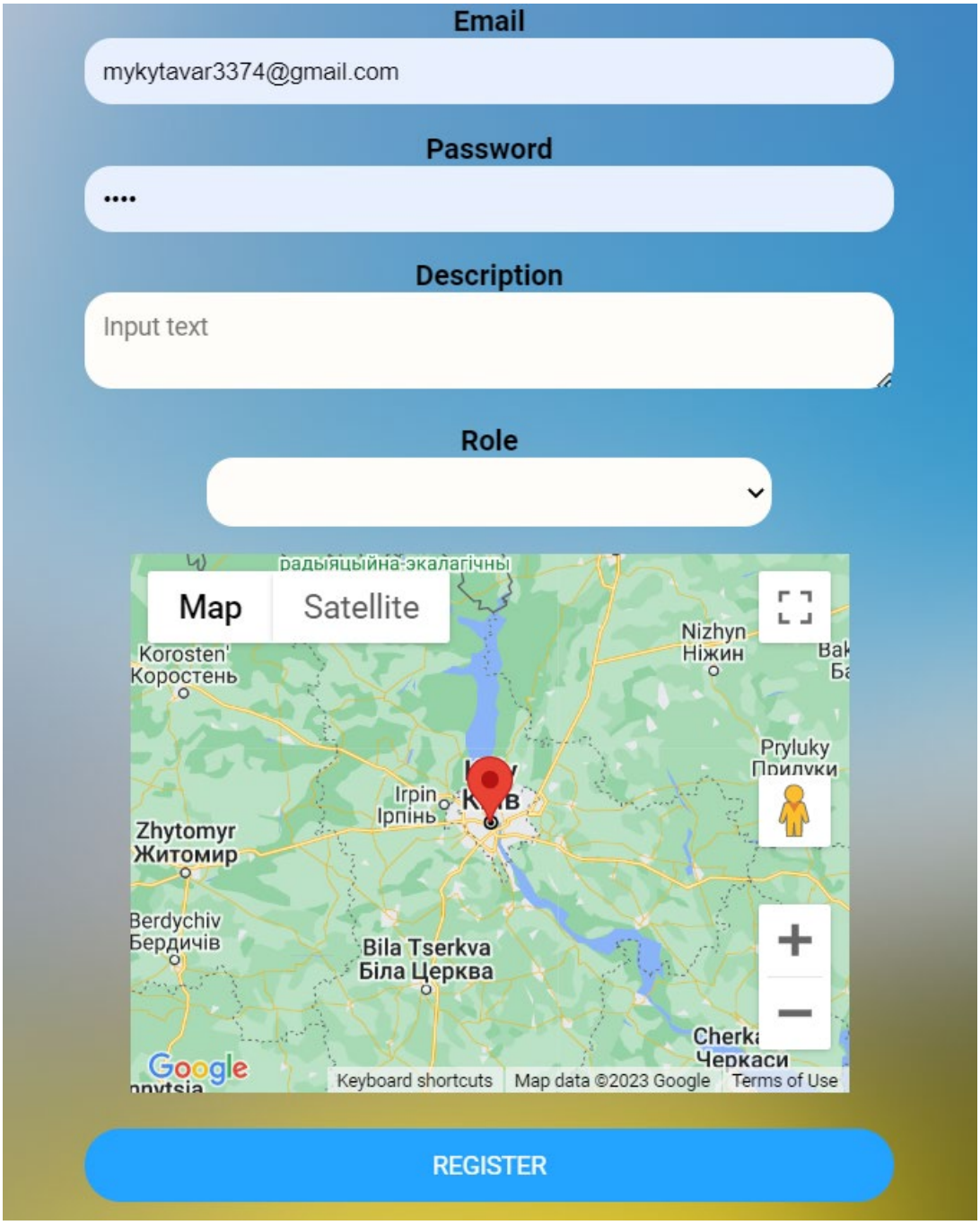

Рис. 2.10 Поля вводу реєстрації

2.6. Опис розробленої системи

2.6.1. Використані технічні засоби

Розробка була виконана за допомогою персонального комп'ютера наступної конфігурації:

– Процесор: 11th Gen Intel(R) Core(TM) i5-11400H @ 2.70GHz

– ОЗУ: 16 ГБ
– SSD диск на 512 ГБ
– Графічна карта: NVIDIA GeForce RTX 3050

2.6.2. Використані програмні засоби

Для проектування, розробки та відладки системи було використано наступні інструменти:

– Intellij Idea 2023 Ultimate;
– WebStorm 2023;
– PgAdmin 4;
– Figma Web;
– Postman;
– Github Copilot;
– Visual Studio Code.

Середовище розробки IntelliJ Idea

IntelliJ IDEA - це інтегроване середовище розробки (IDE) від компанії JetBrains, яке спеціально створене для розробки програмного забезпечення на мові програмування Java, а також підтримує ряд інших мов програмування, таких як Kotlin, Scala, Groovy та інші. IntelliJ IDEA надає розробникам потужний набір інструментів, які спрощують процес написання коду, відлагодження та тестування програмного забезпечення.

Основні функції IntelliJ IDEA включають:

1. Розумне кодування: IntelliJ IDEA розуміє синтаксис Java та інших підтримуваних мов програмування, що дозволяє автоматично генерувати код, підказки та автозавершення.

2. Навігація та пошук: IntelliJ IDEA дозволяє швидко перемикатися між файлами, класами, методами та визначеннями змінних, що полегшує навігацію по проекту.

3. Відлагодження: IntelliJ IDEA має вбудований дебагер, який дозволяє відстежувати виконання коду, переглядати значення змінних та встановлювати точки зупинки.

4. Тестування: IntelliJ IDEA підтримує інтеграцію з популярними фреймворками для тестування, такими як JUnit, TestNG та інші, що дозволяє автоматизувати процес тестування коду.

5. Версійний контроль: IntelliJ IDEA інтегрується з системами контролю версій, такими як Git, Mercurial та Subversion, що дозволяє розробникам легко відстежувати зміни в коді та співпрацювати з командою.

6. Рефакторинг: IntelliJ IDEA надає потужні інструменти для рефакторингу коду, що дозволяє розробникам легко оптимізувати та покращувати структуру коду.

7. Підтримка різних фреймворків та бібліотек: IntelliJ IDEA підтримує різні популярні фреймворки та бібліотеки, такі як Spring, Hibernate, Java EE та інші, що дозволяє розробникам працювати з найсучаснішими технологіями.

8. Кастомізація: IntelliJ IDEA дозволяє налаштувати вигляд та поведінку IDE відповідно до власних потреб та вимог розробника.

IntelliJ IDEA є потужним інструментом для розробки програмного забезпечення на Java та інших мов програмування, який допомагає розробникам підвищити продуктивність та якість коду [17].

Середовище розробки WebStorm

JetBrains WebStorm - це інтегроване середовище розробки (IDE) від компанії JetBrains, яке спеціально створене для розробки веб-додатків на основі JavaScript, HTML і CSS. WebStorm надає розробникам потужний набір інструментів, які спрощують процес написання коду, відлагодження та тестування веб-додатків.

Основні функції WebStorm включають:

1. Розумне кодування: WebStorm розуміє синтаксис JavaScript, HTML і CSS, що дозволяє автоматично генерувати код, підказки та автозавершення.

2. Навігація та пошук: WebStorm дозволяє швидко перемикатися між файлами, функціями та визначеннями змінних, що полегшує навігацію по проекту.

3. Відлагодження: WebStorm має вбудований дебагер, який дозволяє відстежувати виконання коду, переглядати значення змінних та встановлювати точки зупинки.

4. Тестування: WebStorm підтримує інтеграцію з популярними фреймворками для тестування, такими як Jest, Mocha та Karma, що дозволяє автоматизувати процес тестування коду.

5. Версійний контроль: WebStorm інтегрується з системами контролю версій, такими як Git, Mercurial та Subversion, що дозволяє розробникам легко відстежувати зміни в коді та співпрацювати з командою.

6. Рефакторинг: WebStorm надає потужні інструменти для рефакторингу коду, що дозволяє розробникам легко оптимізувати та покращувати структуру коду.

7. Підтримка різних фреймворків та бібліотек: WebStorm підтримує різні популярні фреймворки та бібліотеки, такі як React, Angular, Vue.js та інші, що дозволяє розробникам працювати з найсучаснішими технологіями.

8. Кастомізація: WebStorm дозволяє налаштувати вигляд та поведінку IDE відповідно до власних потреб та вимог розробника.

WebStorm є потужним інструментом для розробки веб-додатків, який допомагає розробникам підвищити продуктивність та якість коду [16].

Середовище проектування Figma

Figma - це інструмент для дизайну інтерфейсів, який працює в режимі онлайн. Він дозволяє дизайнерам та розробникам співпрацювати над проектами в реальному часі, що сприяє підвищенню продуктивності та полегшенню процесу створення дизайну.

Основні переваги Figma включають:

1. Безкоштовний доступ: Figma пропонує безкоштовний план, який дозволяє користувачам створювати необмежену кількість проектів з обмеженою кількістю учасників.

2. Кросплатформеність: Figma працює в браузері, тому користувачі можуть працювати з проектами на різних операційних системах, таких як Windows, macOS та Linux.

3. Співпраця в реальному часі: Figma дозволяє декільком користувачам одночасно працювати над одним проектом, що полегшує обмін ідеями та швидке внесення змін.

4. Вбудовані прототипи: Figma має вбудовані інструменти для створення прототипів, що дозволяє дизайнерам перевіряти свої ідеї та отримувати зворотний зв'язок від команди та клієнтів.

5. Інтеграція з іншими інструментами: Figma може інтегруватися з різними інструментами, такими як Slack, Trello, Jira та інші, що сприяє покращенню робочого процесу.

6. Велика кількість ресурсів: Figma має велику кількість готових шаблонів, компонентів та плагінів, які можуть допомогти дизайнерам прискорити процес створення дизайну.

Figma є потужним інструментом для створення дизайну, який полегшує співпрацю між дизайнерами та розробниками, а також дозволяє швидко та ефективно створювати високоякісні інтерфейси [18].

Github Copilot

GitHub Copilot - це парний програміст зі штучним інтелектом, який допомагає розробникам писати код швидше і з меншими зусиллями. Він працює на основі OpenAI Codex, генеративної попередньо навченої мовної моделі, створеної OpenAI. GitHub Copilot доступний як розширення для Visual Studio Code, Visual Studio, Neovim та набору інтегрованих середовищ розробки (IDE) JetBrains.

GitHub Copilot навчений на всіх мовах, які з'являються у публічних репозиторіях. Для кожної мови якість пропозицій, які ви отримуєте, може

залежати від обсягу та різноманітності навчальних даних для цієї мови. Наприклад, JavaScript добре представлений у публічних репозиторіях і є однією з мов, які найкраще підтримуються GitHub Copilot. Мови з меншим представництвом у публічних репозиторіях можуть давати менше або меншякісних пропозицій.

Щоб використовувати GitHub Copilot, ви повинні мати активну підписку на GitHub Copilot. Перевірені студенти, викладачі та супровідники популярних проектів з відкритим вихідним кодом на GitHub мають право користуватися Copilot для приватних осіб безкоштовно. Підписку на GitHub Copilot можна оплатити та керувати нею через особистий акаунт на GitHub.com за допомогою Copilot for Individuals, або оплатити та керувати нею централізовано через акаунт організації за допомогою GitHub Copilot for Business.

GitHub Copilot пропонує підказки у стилі автоматичного доповнення, коли ви пишете код. Ви можете отримувати підказки від GitHub Copilot, вводячи власний код або використовуючи надані приклади коду. Якщо у вас увімкнено виявлення дублікатів у GitHub Copilot, ви можете отримати обмежену кількість підказок або не отримати жодної підказки, використовуючи надані приклади коду. У цьому випадку рекомендується почати з написання власного коду, щоб побачити поради від GitHub Copilot [19].

2.6.3. Виклик та завантаження програми

Для виклику системи рекомендуються наступні кроки:

- розгортання серверного додатку на хмарній віртуальній машині, наприклад EC2, на якій мають бути встановлені Java 17, Maven 3.6.3+ та Tomcat;
- налаштувати DNS сервіс, наприклад Route 53;
- розгорнути базу даних у сервісі AWS RDS типу PostgreSQL;
- зберегти Angular додаток у S3 та задеплоїти його на CloudFront.

2.6.4 Опис інтерфейсу користувача

При відкритті додатку користувач опиняється на сторінці реєстраці (рис.2.11.), з якої він може перейти до логіну. Якщо користувач вкаже неправильні дані він отримає відповідні повідомлення (рис. 2.12.).

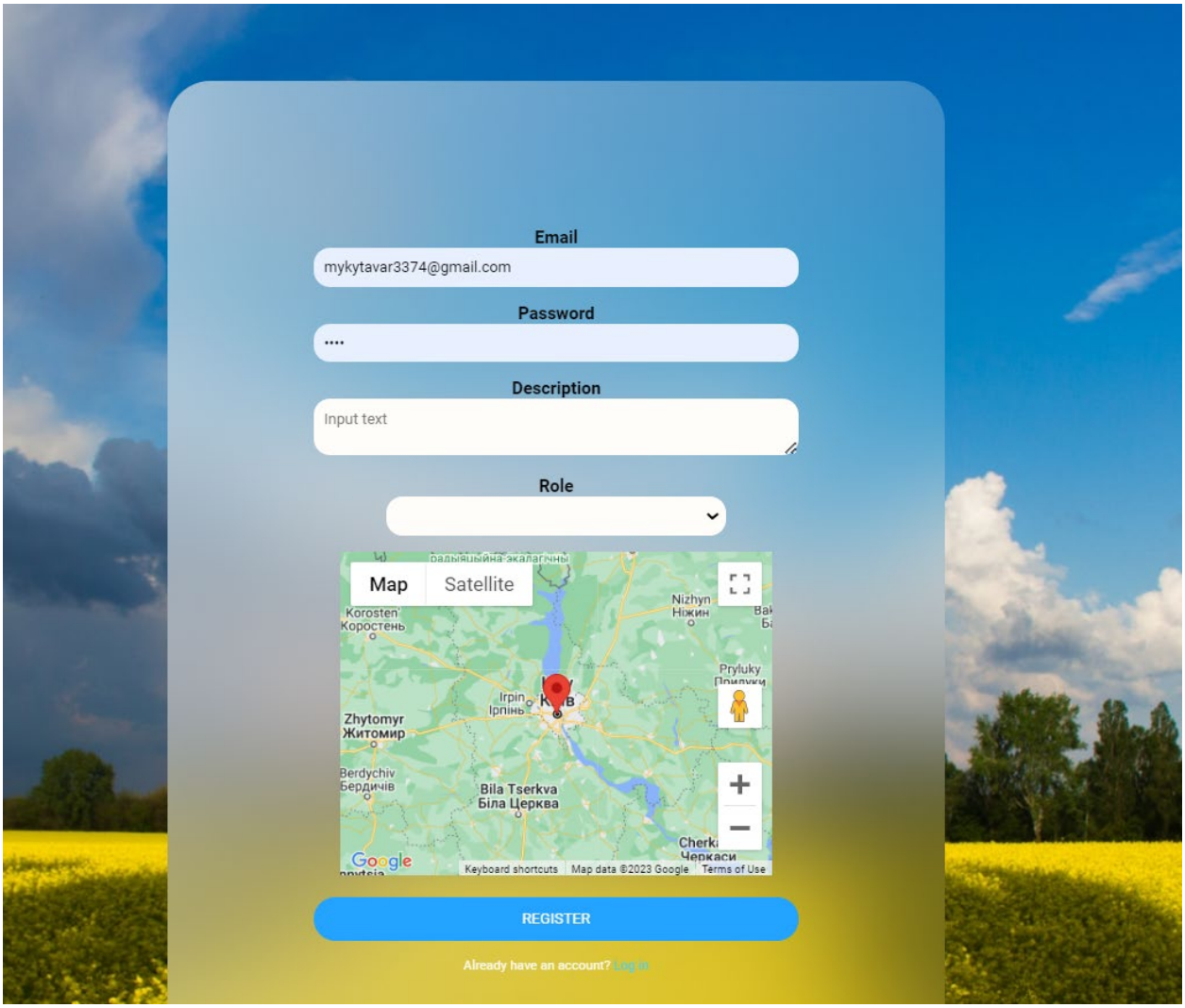

Рис. 2.11. Сторінка реєстрації

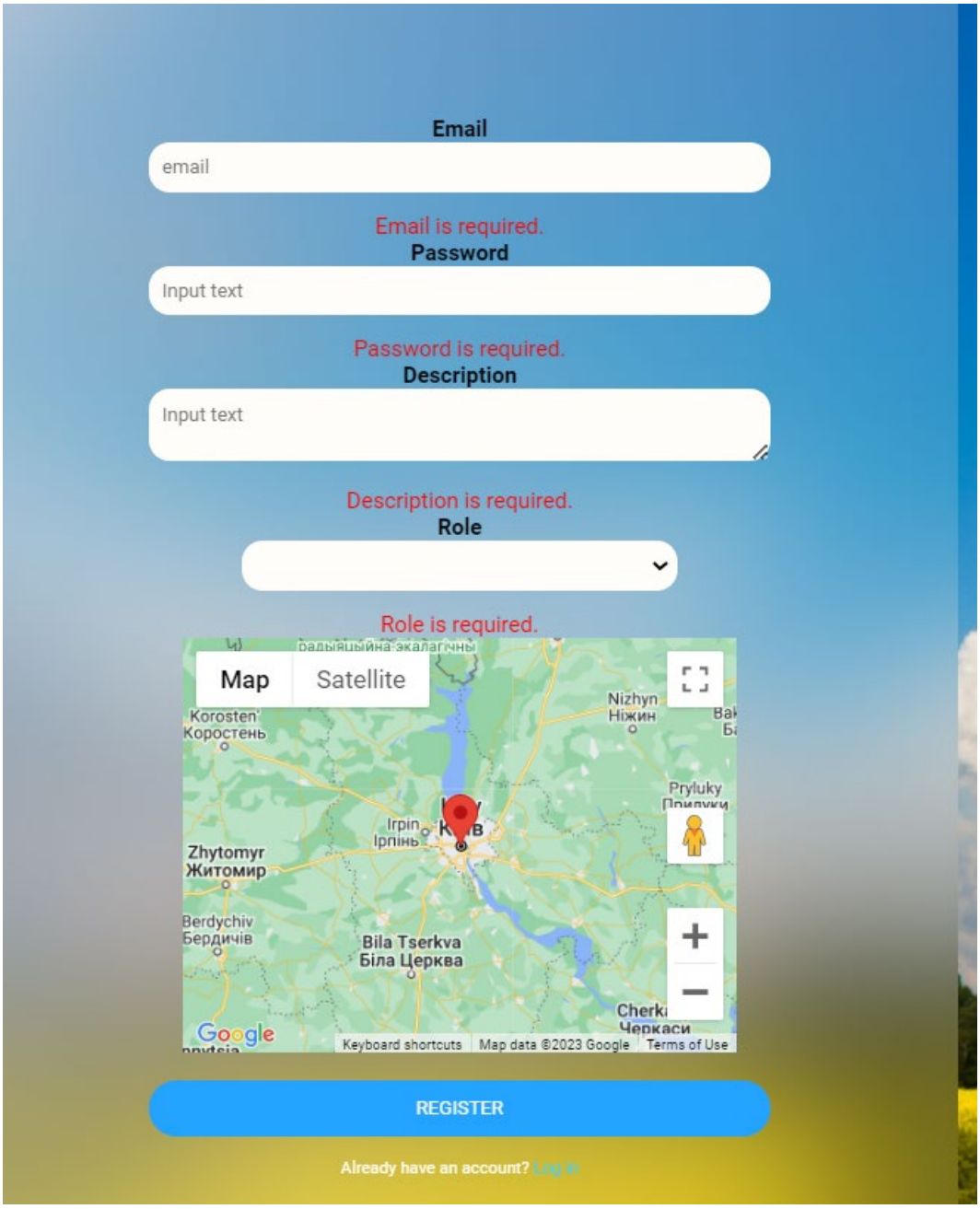

Рис. 2.12. Валідація реєстрації

Якщо користувач вже має аккаунт, він може виконати вхід до системи(рис.2.13). Ця реактивна форма також валідує вхідні дані та виводить відповідні повідомлення (рис.2.14.).

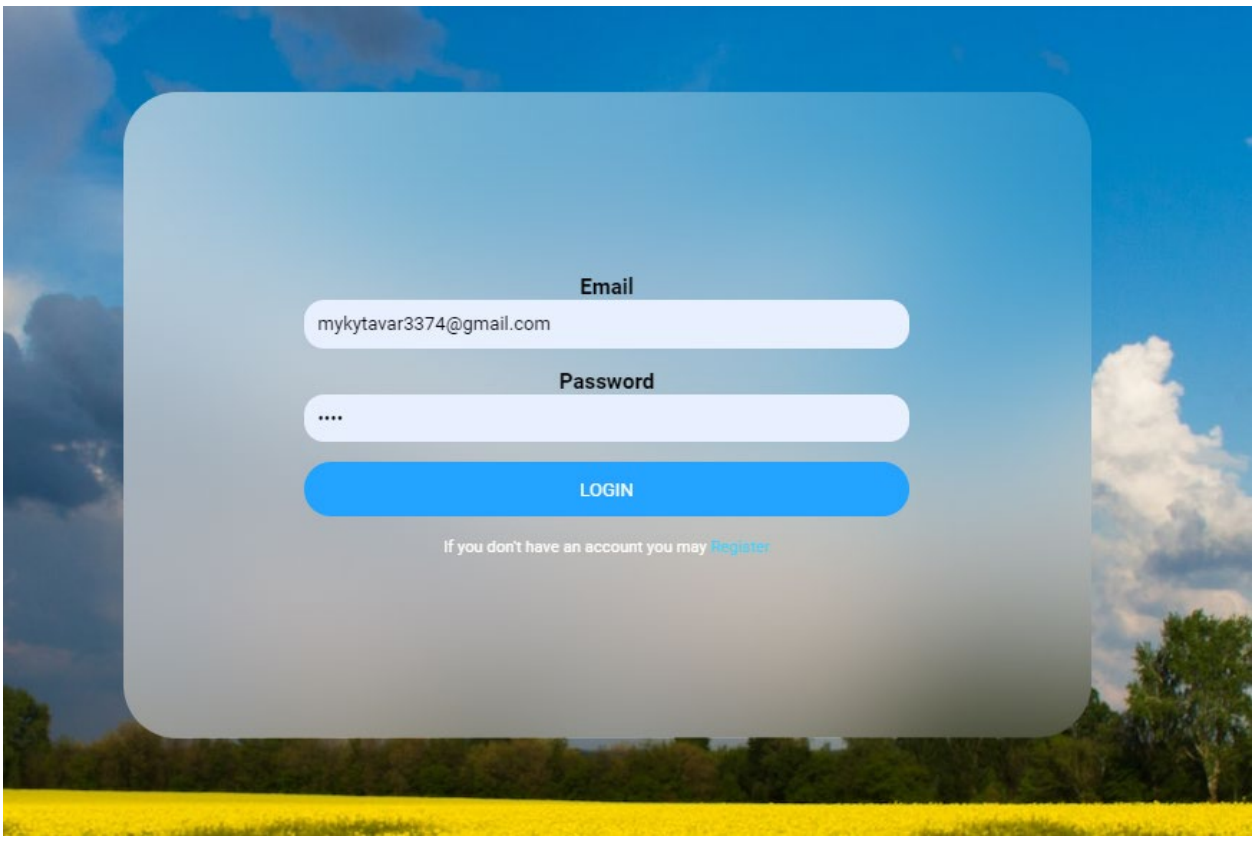

Рис. 2.13. Сторінка аутентифікації

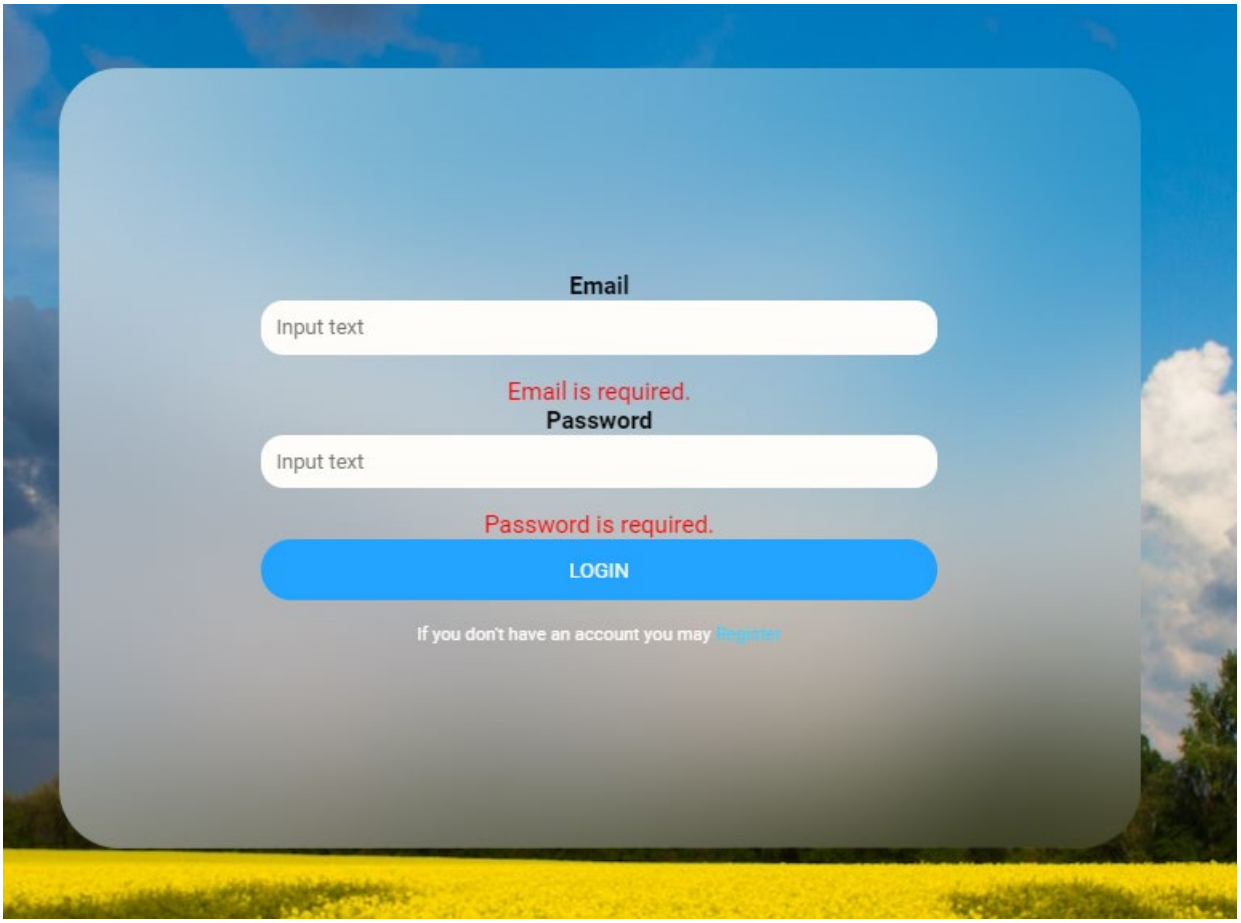

Рис. 2.14. Валідація аутентифікації

Після логіну користувача зустрічає сторінка на якій можна переглянути усі запити (рис.2.15.) та відфільтрувати за населеним пунктом або за типом запиту (рис.2.16.):

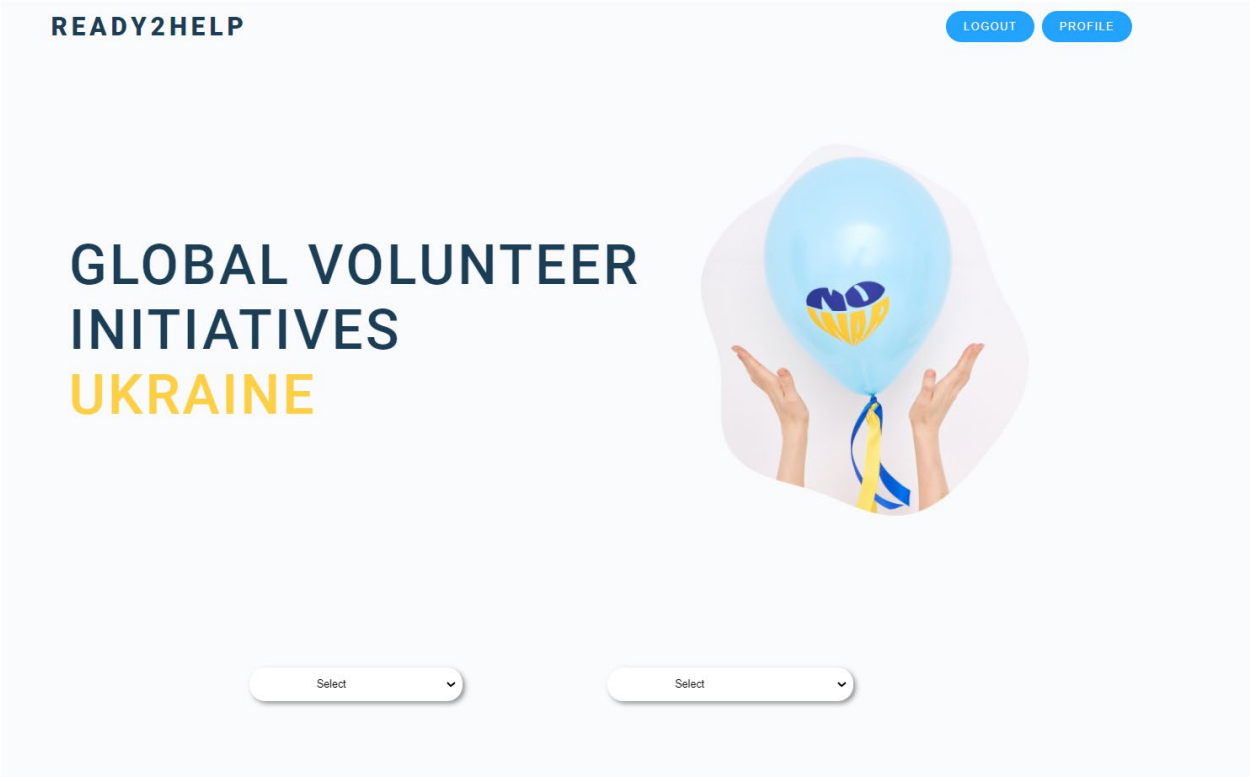

Рис. 2.15. Перша частина головної сторінки

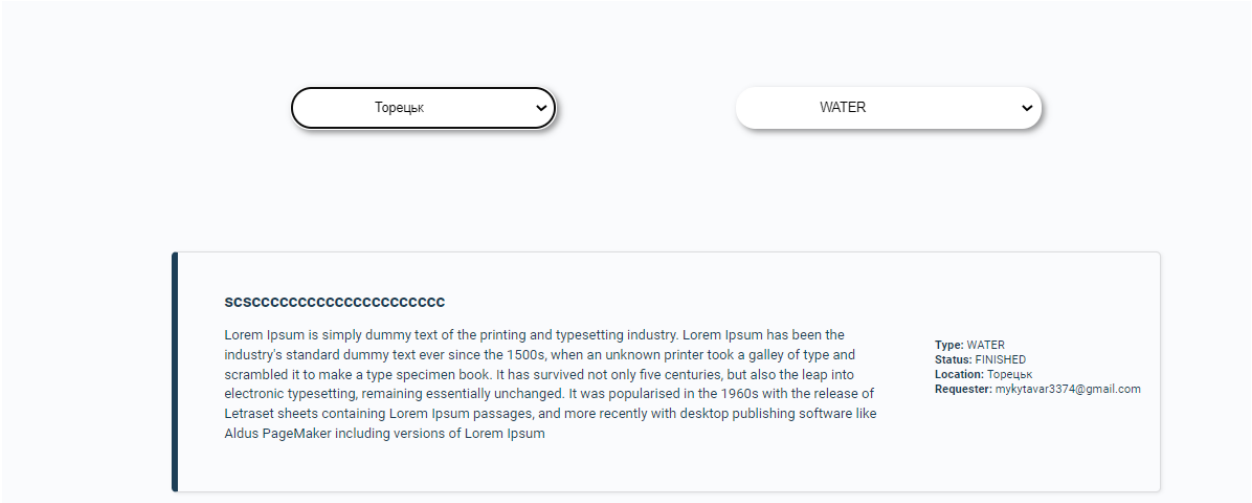

Рис. 2.16. Друга частина головної сторінки

Натиснувши на кнопку PROFILE користувач може перейти до перегляду запитів що пов’язані з ним, і своїх вказаних даних (рис.2.17.)

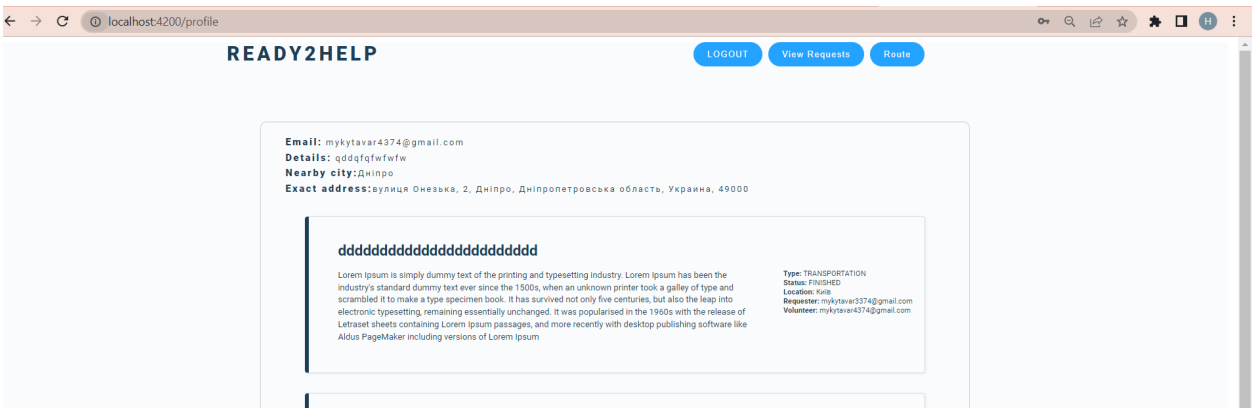

Рис. 2.17. Сторінка профілю

Користувач із роллю заявник може створити запит вказавши локацію (рис.2.18.). На цій сторінці також діє валідація (рис.2.19.).

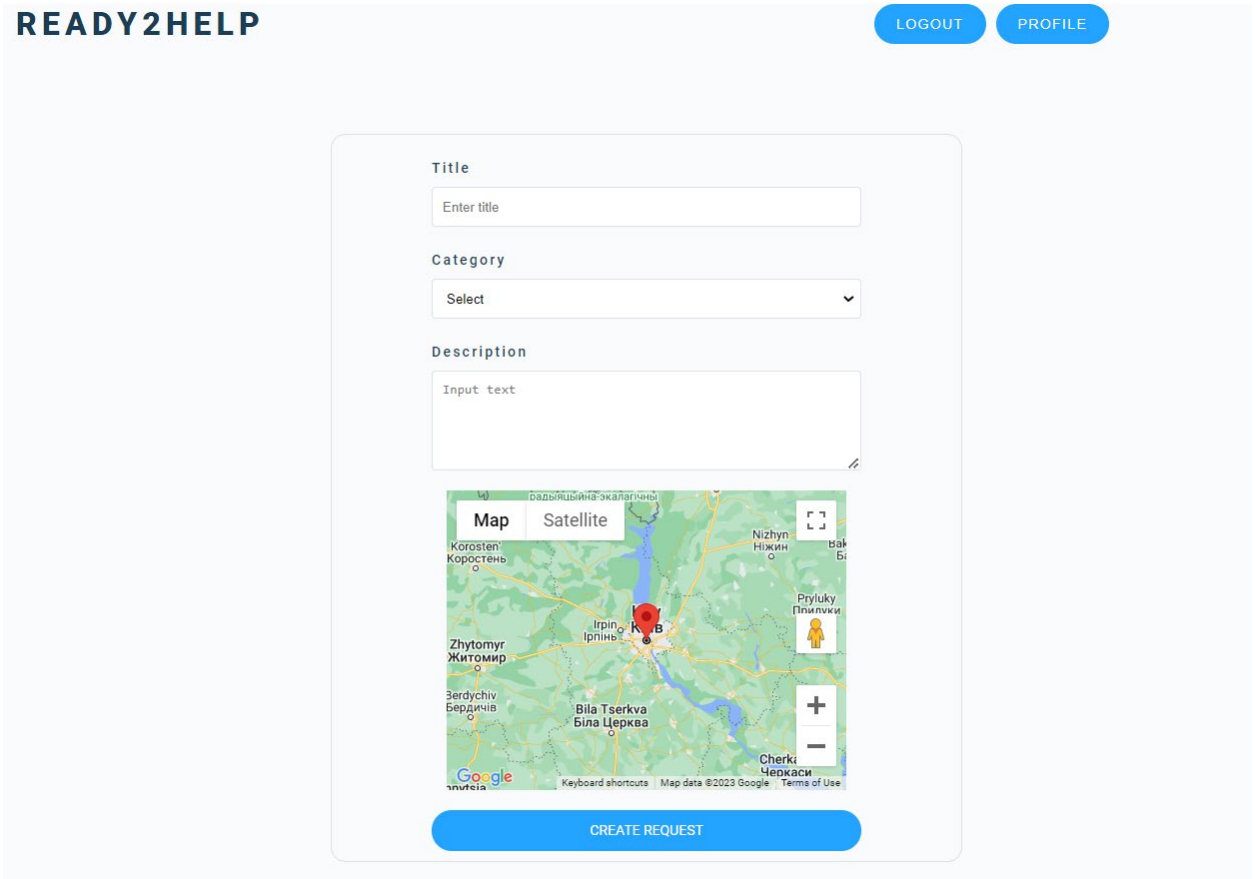

Рис. 2.18. Сторінка створення запиту

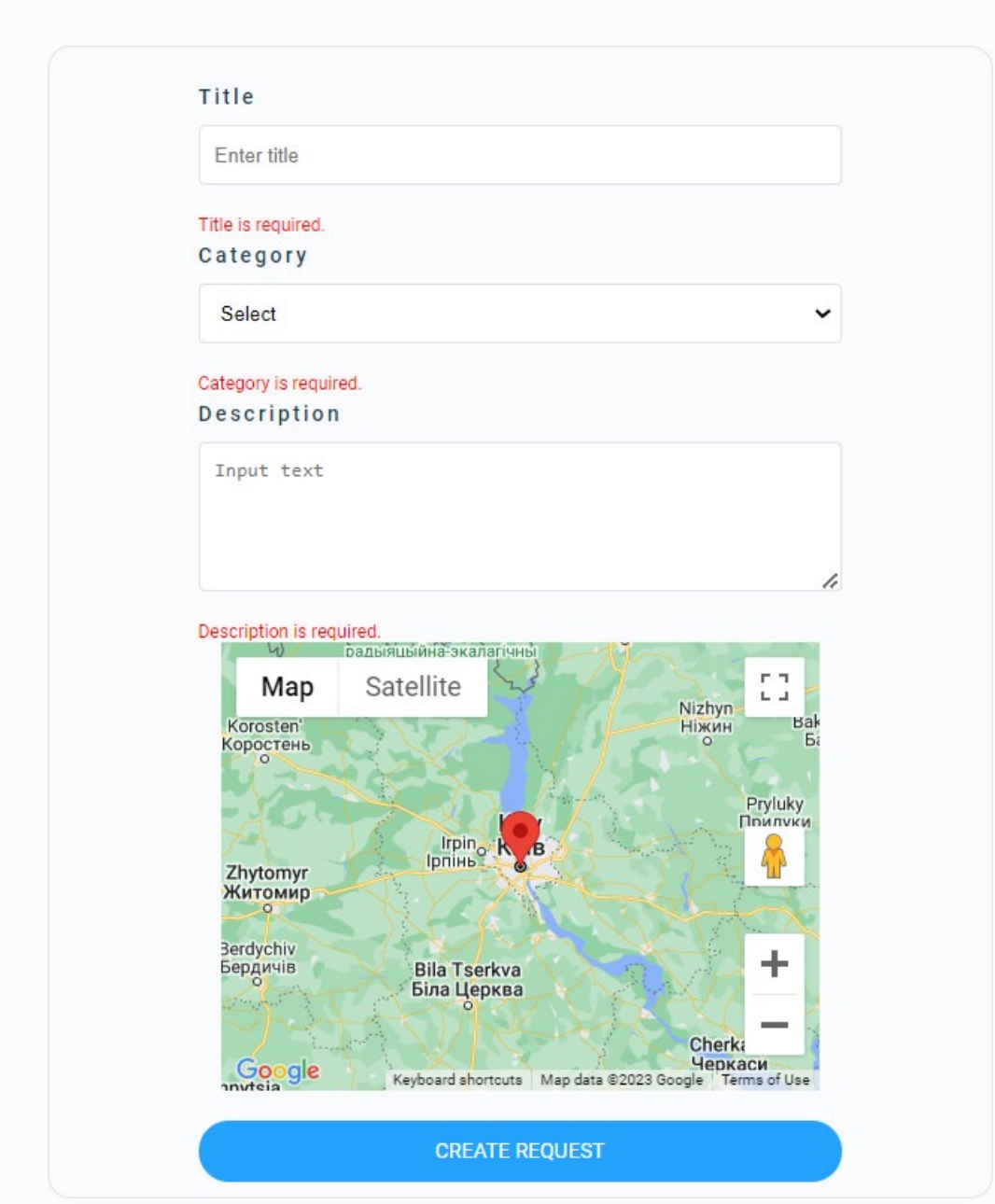

Рис. 2.19. Валідація полів запиту

Коли запити створені, і вони ще не в обробці, волонтер може взяти його на себе натиснувши на кнопку TAKE (рис.2.20.).

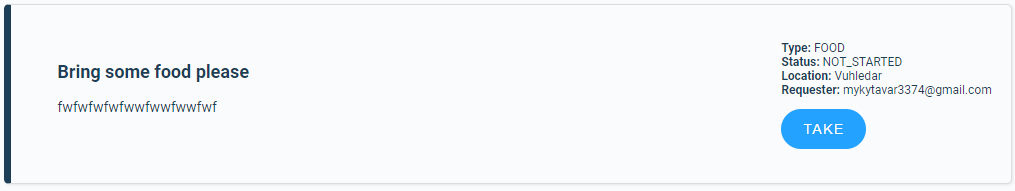

Рис. 2.20. Створений запит

Волонтер може попросити сформувати рекомендований маршрут. Домашня локація волонтера виступає точкою завершення маршруту (рис.2.21.).

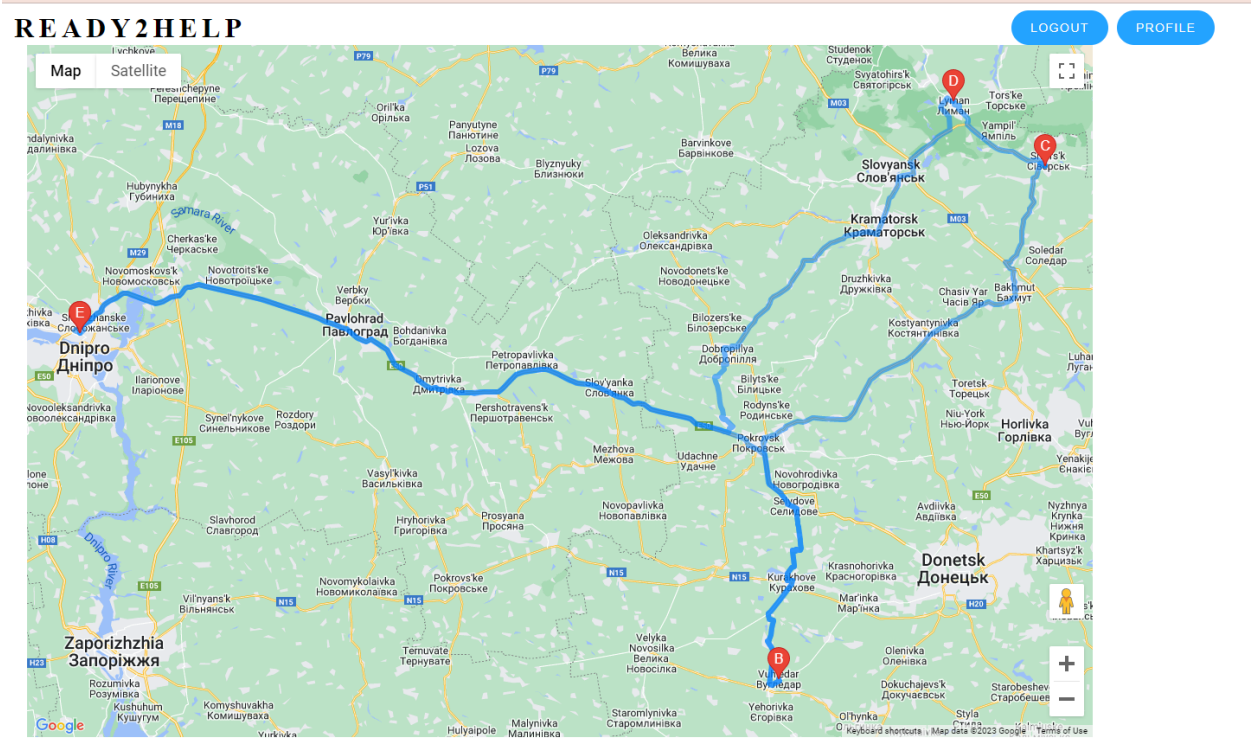

Рис. 2.21. Рекомендований маршрут

Коли запит було виконано, тільки заявник може це підвердити та завершити його натиснувши на кнопку COMPLETE (рис.2.22.).

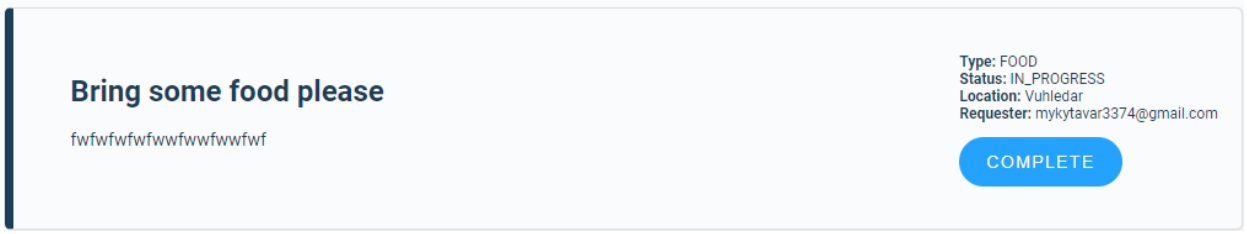

Рис. 2.22. Завершений запит

РОЗДІЛ 3.

ЕКОНОМІЧНИЙ РОЗДІЛ

3.1. Розрахунок трудомісткості та вартості розробки інформаційної системи

Вхідні дані:

- передбачуване число операторів – 2400;
- коефіцієнт корекції програми в ході її розробки – 0, 1;
- коефіцієнт складності програми – 1,3;
- годинна заробітна плата програміста – 512,2 грн/год;
- коефіцієнт збільшення витрат праці в наслідок недостатнього опису задачі – 1,25;
- коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності – 1.2;
- вартість машино-години ЕОМ – 12,3 грн/год;
- кількість розробників – 1.

Заробітна плата за годину Middle Java Software Engineer була вирахувана згідно «Української спільноти програмістів (DOU)» [14]. Станом на грудень 2022 року медіанна зарплата Middle Java Software Engineer з двома роками досвіду дорівнювала 2470$. При курсі валют НБУ на кінець грудня 2022 року один американський долар дорівнює 36,5 грн, тому середня зарплата в гривнях дорівнює 90 155 грн. При стандартному графіку (176 годин/місяць) зарплата за годину буде становити близько 512,2 грн.

Оскільки за годину використаний ігровий ноутбук потребує 0,18 кВт/год, а вартість електроенергії за кВт/год складає 1,68 грн, то за годину комп’ютер споживає електроенергії на $0{,}18 \cdot 1{,}68 = 0{,}30$ грн. Амортизація комп’ютерного обладнання становить 12 грн.

Нормування праці в процесі створення ПЗ істотно ускладнено в силу творчого характеру праці програміста. Тому трудомісткість розробки ПЗ може бути розрахована на основі системи моделей з різною точністю оцінки.

Трудомісткість розробки ПЗ можна розрахувати за формулою:

$$t=t_o+t_u+t_a+t_n+t_{отл}+t_д \quad (3.1)$$

де t_o- витрати праці на підготовку й опис поставленої задачі (приймається 50 людино-годин);

t_u - витрати праці на дослідження алгоритму рішення задачі;

t_a- витрати праці на розробку блок-схеми алгоритму;

t_n- витрати праці на програмування по готовій блок-схемі;

$t_{отл}$- витрати праці на налагодження програми на ЕОМ;

$t_д$ - витрати праці на підготовку документації.

Складові витрати праці визначаються через умовне число операторів у програмному забезпеченні, яке розробляється.

Умовне число операторів (підпрограм):

$$Q=q*C*(1+p) \quad (3.2)$$

де q - передбачуване число операторів (2432);

C - коефіцієнт складності програми (1,3);

p - коефіцієнт корекції програми в ході її розробки (0,1).

За формулою (3.2) умовне число операторів в програмі становить:

$$Q = 2432 * 1,3 * (1+0,05) = 3319,68$$

Витрати праці на вивчення опису задачі t_u визначається з урахуванням уточнення опису і кваліфікації програміста:

$$t_u = \frac{Q*B}{(75..85)*k}, \text{ людино-годин} \quad (3.3)$$

де B - коефіцієнт збільшення витрат праці внаслідок недостатнього опису задачі (1,25);

k - коефіцієнт кваліфікації програміста, обумовлений від стажу роботи з даної спеціальності. При стажі роботи 2 роки він складає 1.2.

За формулою (3.3) отримуємо витрати праці на вивчення опису завдання:

$$t_u = \frac{3319{,}68 * 1.25}{85*1.2} = 40,68 \text{ людино-години}$$

Витрати праці на розробку алгоритму рішення задачі визначаються за формулою:

$$t_a = \frac{Q}{(20...25)*k}, \text{ людино-годин,} \quad (3.4)$$

$$t_a = \frac{3319{,}68}{(25)*1.2} = 110,656 \text{ людино-годин,}$$

Витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{Q}{(20...25)*k}, \text{ людино-годин,} \quad (3.5)$$

Маючи формулу (3.5) витрати на складання програми по готовій блок-схемі:

$$t_n = \frac{3319,68}{(25) * 1.2} = 110,656 \text{ людино-годин}$$

Витрати праці на налагодження програми на ЕОМ:

– за умови автономного налагодження одного завдання:

$$t_{\text{отл}} = \frac{Q}{(4..5)*k}, \quad (3.6)$$

Витрати праці на налагодження програми на ЕОМ за умови автономного налагодження одного завдання за формулою (3.6):

$$t_{\text{отл}} = \frac{3319{,}68}{5*1.2} = 553,28, \text{ людино-годин},$$

– за умови комплексного налагодження завдання:

$$t_{\text{отл}}^{k} = 1{,}5 * t_{\text{отл}}, \quad (3.7)$$

Витрати праці на налагодження програми на ЕОМ за умови комплексного налагодження завдання за формулою (3.7):

$$t_{\text{отл}}^{k} = 1{,}5 * 553,28 = 829,92, \text{ людино-годин},$$

2. Витрати праці на підготовку документації визначаються за формулою:

$$t_{\text{д}} = t_{\text{др}} + t_{\text{до}}, \quad (3.8)$$

де $t_{\text{др}}$ - трудомісткість підготовки матеріалів і рукопису:

$$t_{\text{др}} = \frac{Q}{(15..20)*k}, \quad (3.9)$$

$t_{\text{до}}$ - трудомісткість редагування, печатки й оформлення документації:

$$t_{\text{до}} = 0{,}75 * t_{\text{др}}, \quad (3.10)$$

Маючи формули (3.8), (3.9) та (3.10) обчислюємо витрати праці на документацію:

$$t_{\text{др}} = \frac{3319{,}68}{20*1.2} = 138,32 \text{ , людино-годин,}$$

$$t_{\text{до}} = 0{,}75 * 138,32 = 103,74 \text{ людино-годин,}$$

$$t_{\text{д}} = 138{,}32 + 103{,}74 = 242,06\text{, людино-годин,}$$

Повертаючись до формули (3.1), отримаємо повну оцінку трудомісткості розробки програмного забезпечення:

$$t = 50 + 40{,}68 + 110{,}65 + 110{,}65 + 553{,}28 + 242{,}06 = 1107{,}32\text{, людино-годин.}$$

3.2. Розрахунок витрат на створення програми

Витрати на створення ПЗ КПО включають витрати на заробітну плату виконавця програми $З_{ЗП}$ і витрат машинного часу, необхідного на налагодження програми на ЕОМ:

$$К_{ПО} = З_{ЗП} + З_{МВ}\text{, грн.} \qquad (3.11)$$

Заробітна плата виконавців визначається за формулою:

$$З_{ЗП} = t * С_{ПР}\text{, грн,} \qquad (3.12)$$

де t - загальна трудомісткість, людино-годин (833,2);

$С_{ПР}$ - середня годинна заробітна плата програміста, грн/година.

З урахуванням того, що середня годинна зарплата програміста становить 512,2 грн / год, за формулою (3.12) отримуємо:

$$З_{ЗП} = 1107{,}32 * 512{,}2 = 567\ 169{,}304 \text{ грн},$$

Вартість машинного часу, необхідного для налагодження програми на ЕОМ, визначається за формулою:

$$З_{МВ} = t_{отл} * С_{Мч}, \text{ грн}, \quad (3.13)$$

де $t_{отл}$ - трудомісткість налагодження програми на ЕОМ, год (705,25 год);

$С_{Мч}$ - вартість машино-години ЕОМ, грн/год (12,3 грн/год).

Підставивши в формулу (3.13) відповідні значення, обчислюємо вартість необхідного для налагодження машинного часу:

$$З_{МВ} = 829{,}92 * 12{,}3 = 10\ 208{,}016, \text{ грн},$$

Звідси, за формулою (3.21), витрати на створення програмного продукту:

$$К_{ПО} = 567\ 169{,}304 + 10\ 208{,}016 = 577\ 377 \text{ грн},$$

Очікуваний період створення програмного застосунку:

$$T = \frac{t}{B_k * F_p}, \text{ міс}, \quad (3.14)$$

де B_k - число виконавців (1);

F_p - місячний фонд робочого часу (при 40 годинному робочому тижні F_p =176 годин).

За формулою 3.14 очікуваний період створення програмного забезпечення:

$$T = \frac{1107{,}32}{1*176} \approx 6{,}29 \text{ міс}. \quad (3.14)$$

Висновки: розробка системи управління волонтерською діяльністю має коштувати 577 377 грн без додаткових витрат.

Згідно отриманих даних при розрахунках, очікуваний час розробки становить 1107, 32 години, або 6,29 місяці, з урахуванням стандартного робочого графіку. Результат, що отримано, залежить від незначного коефіцієнта кваліфікації розробника, та у свою чергу включає час на виконання досліджень та розробку концепції для втілення поставленої задачі, надалі програмування за створеною концепцією, тестування програмного продукту та впровадження документації.

ВИСНОВКИ

Метою кваліфікаційної роботи було створення сайту управління волонтерською діяльністю, що допоможе людям у потребі швидше знаходити волонтерів, а волонтерам допоможе витрачати меньше часу на планування дороги.

Актуальність теми полягає у тому що Україна має багато організацій, громадських ініціатив та небайдужих людей, які займаються волонтерською діяльністю. Однак, часто буває важко зорієнтуватися в тому, які проекти потребують найбільшої уваги та як краще розподілити ресурси. Система управління волонтерською діяльністю допомагає збирати інформацію про потреби в різних регіонах, що дозволяє ефективніше координувати дії волонтерів.

З практичної точки зору, завдяки системі управління волонтерською діяльністю, можна виявити потреби різних регіонів та виділити пріоритетні напрямки допомоги. Рекомендація маршрутів для волонтерів дозволяє ефективно планувати їх переміщення, щоб максимально задовольнити потреби в різних місцях.

Розробка виконана за допомогою таких технологій: Angular, HTML, CSS, Typescript, NgRx, RxJs, Spring Boot, Spring Data JPA, Java 17, Google Directions and Places API, PostgreSQL.

Як можливий шлях майбутнього розвитку можна зазначити створення системи оцінок волонтерів, за якої, перед тим як волонтер міг би приступити до виконання завдання, заявник міг би передивитись його профіль та відгуги та прийняти особисте рішення стосовно подальшої співпраці. Також гарним варіантом було б надання не тільки рекомендації маршруту на мапі, а й покрокову інформацію стосовно переміщення по маршруту, з інформацією про повороти і інш. Таку функціональність можна розробити також за допомогою Google Directions API.

Під час виконання економічного розділу було обраховано повну оцінку трудомісткості розробки програмного забезпечення, що становило 1107,32 людино-годин. Заробітна плата виконавців за період виконання проекту визначена як 567 169,304 грн. Вартість машинного часу обраховано як 10 208 грн. Загальні витрати визначено як 577 377 грн.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. UaHelpers база волонтерської допомоги та запитів на допомогу URL /https://uahelpers.com/ (дата звернення 01.04.2023)
2. Волонтерська платформа / URL https://platforma.volunteer.country (дата звернення 01.04.2023)
3. The Java Language Specification, Java 17 Edition / URL https://docs.oracle.com/javase/specs/jvms/se17/html/jvms-1.html#jvms-1.1 (дата звернення 11.04.2023)
4. Travelling salesman problem / URL https://medium.com/ivymobility-developers/traveling-salesman-problem-9ab623c88fab (дата звернення 15.04.2023)
5. Spring Boot Documentation / URL https://spring.io/projects/spring-boot (дата звернення 17.04.2023)
6. Spring Data Documentation / URL https://spring.io/projects/spring-data (дата звернення 17.04.2023)
7. Typescript Documentation / URL https://www.typescriptlang.org/ (дата звернення 17.04.2023)
8. What is Angular? / URL https://angular.io/guide/what-is-angular
9. Ngrx and Angular 2 Tutorial: Building a Reactive Application URL / https://www.toptal.com/angular-js/ngrx-angular-reaction-application (дата звернення 20.04.2023)
10. Google Maps Platform Directions Service / URL https://developers.google.com/maps/documentation/javascript/directions?_gl=1*1ri2zr9*_ga*MTU2ODE5MzE4Ny4xNjg1NTQxMTk4*_ga_NRWSTWS78N*MTY4NTU0MTE5Ny4xLjEuMTY4NTU0MTY1Ni4wLjAuMA (дата звернення 01.05.2023)
11. The persistence layer with Spring Data JPA / URL https://www.baeldung.com/the-persistence-layer-with-spring-data-jpa (дата звернення 01.05.2023)

12. What is a Use Case Diagram? / URL https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/ (дата звернення 05.05.2023)

13. UML Activity Diagram / URL https://www.lucidchart.com/pages/uml-activity-diagram (дата звернення 01.05.2023)

14. Зарплатне опитування DOU / URL https://jobs.dou.ua/salaries/?period=2022-12&position=Middle%20SE&technology=Java&experience=2(дата звернення 10.05.2023)

15. Скільки електроенергії споживає комп'ютер — як дізнатися: 3 способи / URL https://www.moyo.ua/ua/news/skolko_elektroenergii_potreblyaet_kompyuter_kak_uznat_3_sposoba_.html (дата звернення 15.05.2023)

16. WebStorm The smartest JavaScript IDE / URL https://www.jetbrains.com/webstorm/ (дата звернення 01.05.2023)

17. IntelliJ IDEA – the Leading Java and Kotlin IDE / URL https://www.jetbrains.com/idea/ (дата звернення 01.05.2023)

18. The power of Figma as a design tool / URL https://www.toptal.com/designers/ui/figma-design-tool (дата звернення 03.05.2023)

19. About Github Copilot for individuals / URL https://docs.github.com/en/copilot/overview-of-github-copilot/about-github-copilot-for-individuals (дата звернення 03.05.2023)

20. What is REST / URL https://restfulapi.net/ (дата звернення 05.05.2023)

ДОДАТОК А

ЛІСТИНГ ПРОГРАМИ

VolunteerManagementApplication.java

```
import com.example.demo.domain.Role;
import com.example.demo.repositories.RoleRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.boot.context.event.ApplicationReadyEvent;
import org.springframework.context.event.EventListener;
import org.springframework.stereotype.Component;
import java.util.List;
@SpringBootApplication
public class VolunteerManagementApplication {
    public static void main(String[] args) {
        SpringApplication.run(VolunteerManagementApplication.class, args);
    }
}
@Component
class Initializer {
    @Autowired
    private RoleRepository roleRepository;
    @EventListener(ApplicationReadyEvent.class)
    public void initialize() {
        if (!(roleRepository.existsByName("REQUESTER") && roleRepository.existsByName("VOLUNTEER"))) {
            roleRepository.saveAll(List.of(new Role("REQUESTER"), new Role("VOLUNTEER")));
        }
    }
}
```

User.java

```
package com.example.demo.domain;

import jakarta.persistence.*;

import lombok.AllArgsConstructor;

import lombok.Builder;

import lombok.Getter;

import lombok.Setter;

import org.hibernate.annotations.GenericGenerator;

import java.util.UUID;

@Entity(name = "users")

@Getter

@Builder

@Setter

@Table(name = "users", schema = "volunteer_management")

@AllArgsConstructor

public class User {

    @Id

    @GeneratedValue(generator = "uuid2")

    @GenericGenerator(name = "uuid2", strategy = "uuid2")

    @Column(name = "id")
```

```
    private UUID id;
    @Column(name = "email", unique = true)
    private String email;
    @Column(name = "password")
    private String password;
    @Column(name = "details")
    private String details;
    @ManyToOne
    private Role role;
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "location_id")
    private Location location;
    public User() {  }
}
```

Location.java

```
@Entity(name = "location")
@Table(name = "location", schema = "volunteer_management")
@Builder
@Getter
@AllArgsConstructor
public class Location {
    @Id
    @GeneratedValue(generator = "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "uuid2")
    @Column(name = "id")
    private UUID id;
    @Column(name = "longitude")
    private BigDecimal longitude;
    @Column(name = "latitude")
    private BigDecimal latitude;
    @Column(name = "nearby_city")
    private String nearby_city;
    @Column(name = "exact_address")
    private String exact_address;
    @OneToOne(mappedBy = "location")
    @JsonBackReference
    private User user;
    @OneToOne(mappedBy = "location")
    @JsonBackReference
    private Request request;
    public Location() { }
}
```

Need.java

```
package com.example.demo.domain;
public enum Need {
   TRANSPORTATION, WATER, FOOD, MEDICINE, TRANSLATION
}
```

Request.java

```
package com.example.demo.domain;
import com.fasterxml.jackson.annotation.JsonManagedReference;
import jakarta.persistence.*;
import lombok.*;
import org.hibernate.annotations.GenericGenerator;
import java.util.UUID;
@Entity(name = "requests")
@Table(name = "requests", schema = "volunteer_management")
@Builder
@NoArgsConstructor
@AllArgsConstructor
@Getter
@Setter
public class Request {
   @Id
   @GeneratedValue(generator = "uuid2")
   @GenericGenerator(name = "uuid2", strategy = "uuid2")
   @Column(name = "id")
   private UUID id;
   @ManyToOne
   @JoinColumn(name = "requester_id")
   private User requester;
   @Column(name = "title")
   private String title;
   @ManyToOne
   @JoinColumn(name = "volunteer_id")
   private User volunteer;
   @Column(name = "details", length = 2000)
   private String details;
   @Enumerated(EnumType.STRING)
   @Column(name = "status")
   private Status status;
   @Enumerated(EnumType.STRING)
   @Column(name = "need")
   private Need need;
```

```
    @OneToOne(cascade = CascadeType.ALL)
    @JoinColumn(name = "location_id")
    @JsonManagedReference
    private Location location;
}
```

Role.java

```
package com.example.demo.domain;
import jakarta.persistence.*;
import lombok.Getter;
import lombok.NoArgsConstructor;
import org.hibernate.annotations.GenericGenerator;
import org.springframework.security.core.authority.SimpleGrantedAuthority;
import java.util.UUID;
@Entity(name = "roles")
@Table(name = "roles", schema = "volunteer_management")
@Getter
@NoArgsConstructor
public class Role {
    @Id
    @GeneratedValue(generator = "uuid2")
    @GenericGenerator(name = "uuid2", strategy = "uuid2")
    @Column(name = "id")
    private UUID id;
    @Column(name = "name")
    private String name;
    public Role(String name) {
        this.name = name;
    }
    public static SimpleGrantedAuthority getAuthority(User user) {
        return new SimpleGrantedAuthority("ROLE_" + user.getRole().getName());
    }
}
```

SecurityConfiguration.java

```
package com.example.demo.configuration;
import com.example.demo.jwt.JwtTokenFilter;
import com.example.demo.jwt.JwtTokenProvider;
import jakarta.servlet.http.HttpServletResponse;
import lombok.AllArgsConstructor;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.http.HttpMethod;
```

```
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.AuthenticationProvider;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;
import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.password.PasswordEncoder;
import org.springframework.security.web.SecurityFilterChain;
import org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
import org.springframework.web.cors.CorsConfiguration;
import org.springframework.web.cors.CorsConfigurationSource;
import org.springframework.web.cors.UrlBasedCorsConfigurationSource;
import java.util.Arrays;
@Configuration
@EnableWebSecurity
@AllArgsConstructor
@EnableMethodSecurity
public class SecurityConfiguration {
   private final JwtTokenProvider tokenProvider;
   private final UserDetailsService userDetailsService;
   @Bean
   public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
      http
            .csrf().disable()
            .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS).and()
            .exceptionHandling().authenticationEntryPoint((req, rsp, e) -> {
               System.out.println("Authentication exception: " + e.getMessage());
               rsp.sendError(HttpServletResponse.SC_UNAUTHORIZED, e.getMessage());
            })
            .and()
            .authenticationProvider(authenticationProvider())
            .addFilterBefore(new JwtTokenFilter(tokenProvider), UsernamePasswordAuthenticationFilter.class)
            .authorizeHttpRequests()
            .requestMatchers(HttpMethod.POST, "/users", "/users/login", "/requests").permitAll()
            .requestMatchers(HttpMethod.POST, "/requests", "/requests/complete/**").hasRole("REQUESTER")
            .requestMatchers(HttpMethod.POST, "/requests/take/**").hasRole("VOLUNTEER")
            .requestMatchers(HttpMethod.GET, "/requests/route").hasRole("VOLUNTEER")
            .requestMatchers(HttpMethod.GET, "/requests", "/categories").permitAll()
            .requestMatchers(HttpMethod.GET, "/users/me").permitAll()
```

```
            .anyRequest().fullyAuthenticated();
        http.cors();
        return http.build();}
    @Bean
    public CorsConfigurationSource corsConfigurationSource() {
        final CorsConfiguration config = new CorsConfiguration();
        config.setAllowedOrigins(Arrays.asList("http://localhost:4200"));
        config.setAllowedMethods(Arrays.asList("GET", "POST", "OPTIONS", "DELETE", "PUT", "PATCH"));
        config.setAllowCredentials(true);
        config.setAllowedHeaders(Arrays.asList("Authorization", "Cache-Control", "Content-Type"));
        final UrlBasedCorsConfigurationSource source = new UrlBasedCorsConfigurationSource();
        source.registerCorsConfiguration("/**", config);
        return source;
    }
    @Bean
    public AuthenticationProvider authenticationProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();
        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
        return authProvider;
    }
    @Bean
    public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {
        return config.getAuthenticationManager();
    }
    @Bean
    public PasswordEncoder passwordEncoder() {
        return new MyPasswordEncoder();
    }
    private static class MyPasswordEncoder implements PasswordEncoder {
        @Override
        public String encode(CharSequence charSequence) {
            return charSequence.toString();
        }
        @Override
        public boolean matches(CharSequence charSequence, String s) {
            return true;
        }
    }
}
```

CategoriesController.java

```
package com.example.demo.controller;
```

```
import com.example.demo.domain.Need;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.ResponseStatus;
import org.springframework.web.bind.annotation.RestController;
import java.util.Arrays;
import java.util.List;
import static org.springframework.http.HttpStatus.OK;
@RestController
@RequiredArgsConstructor
@RequestMapping("/categories")
public class CategoriesController {
   @GetMapping
   @ResponseStatus(OK)
   public List<String> getCategories() {
      return Arrays.stream(Need.values()).map(Enum::name).toList();
   }
}
```

RequestController.java

```
package com.example.demo.controller;
import com.example.demo.domain.Location;
import com.example.demo.domain.Request;
import com.example.demo.dto.CreateRequestDto;
import com.example.demo.service.RequestFacade;
import com.example.demo.service.RouteBuilderFacade;
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.*;
import java.io.IOException;
import java.util.List;
import java.util.UUID;
import static org.springframework.http.HttpStatus.CREATED;
import static org.springframework.http.HttpStatus.OK;
@RestController
@RequiredArgsConstructor
@RequestMapping("/requests")
public class RequestController {
   private final RequestFacade requestFacade;
   private final RouteBuilderFacade routeBuilderFacade;

   @PostMapping
   @ResponseStatus(CREATED)
```

```
    public Request createRequest(@RequestBody CreateRequestDto createRequestDto,
                                 @RequestHeader("Authorization") String token) throws IOException, InterruptedException {
        return requestFacade.createRequest(createRequestDto, token);
    }
    @GetMapping
    @ResponseStatus(OK)
    public Iterable<Request> getAllRequests() {
        return requestFacade.getAllRequests();
    }
    @PostMapping("/take/{id}")
    @ResponseStatus(OK)
    public Request takeRequest(@PathVariable UUID id, @RequestHeader("Authorization") String token) throws
IOException, InterruptedException {
        return requestFacade.takeRequest(token, id);
    }
    @GetMapping("/my")
    @ResponseStatus(OK)
    public Iterable<Request> getMyRequests(@RequestHeader("Authorization") String token) throws IOException,
InterruptedException {
        return requestFacade.getMyRequests(token);
    }
    @PostMapping("/complete/{id}")
    @ResponseStatus(OK)
    public Request completeRequest(@PathVariable UUID id, @RequestHeader("Authorization") String token) throws
IOException, InterruptedException {
        return requestFacade.completeRequest(token, id);
    }
    @GetMapping("/route")
    @ResponseStatus(OK)
    public List<Location> getRoute(@RequestHeader("Authorization") String token) throws IOException,
InterruptedException {
        return routeBuilderFacade.createRoute(token);
    }
}
```

UserController.java

```
package com.example.demo.controller;
import com.example.demo.domain.User;
import com.example.demo.dto.AuthRequestDto;
import com.example.demo.dto.AuthResponseDto;
import com.example.demo.dto.UserDto;
import com.example.demo.service.UserFacade;
import lombok.RequiredArgsConstructor;
```

```
import org.springframework.web.bind.annotation.*;
import static org.springframework.http.HttpStatus.CREATED;
import static org.springframework.http.HttpStatus.OK;
@RestController
@RequiredArgsConstructor
@RequestMapping("/users")
public class UserController {
    private final UserFacade userFacade;
    @PostMapping
    @ResponseStatus(CREATED)
    public AuthResponseDto createUser(@RequestBody UserDto userDto) {
        return userFacade.createUser(userDto);
    }
    @PostMapping("login")
    @ResponseStatus(OK)
    public AuthResponseDto login(@RequestBody AuthRequestDto authRequestDto) {
        return userFacade.login(authRequestDto);
    }
    @GetMapping("me")
    @ResponseStatus(OK)
    public User getFromToken(@RequestHeader("Authorization") String token) {
        return userFacade.getUserByToken(token);
    }
}
```

LocationService.java

```
package com.example.demo.service;
import com.example.demo.domain.Location;
import com.example.demo.dto.LocationDto;
import com.example.demo.repositories.LocationRepository;
import lombok.RequiredArgsConstructor;
import org.springframework.stereotype.Service;
@Service
@RequiredArgsConstructor
public class LocationService {
    private final LocationRepository locationRepository;
    public Location createLocation(LocationDto locationDto) {
        var byLongitudeAndLatitude = locationRepository.findByLongitudeAndLatitude(locationDto.getLongitude(),
locationDto.getLatitude());
        return byLongitudeAndLatitude.orElseGet(() -> locationRepository.save(Location.builder()
                .longitude(locationDto.getLongitude())
                .latitude(locationDto.getLatitude())
                .nearby_city(locationDto.getNearby_city())
```

```
            .exact_address(locationDto.getExact_address())
            .build()));
    }
}
```

RequestFacade.java

```
package com.example.demo.service;
import com.example.demo.domain.*;
import com.example.demo.dto.CreateRequestDto;
import com.example.demo.jwt.JwtTokenProvider;
import com.example.demo.repositories.RequestRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Component;
import org.springframework.transaction.annotation.Transactional;
import java.util.UUID;
@Component
@AllArgsConstructor
public class RequestFacade {
    private final RequestService requestService;
    private final UserFacade userFacade;
    private final LocationService locationService;
    private final JwtTokenProvider jwtTokenProvider;
    private final RequestRepository requestRepository;
    @Transactional
    public Request createRequest(CreateRequestDto createRequestDto, String token)  {
        Location location = locationService.createLocation(createRequestDto.getLocation());
        User requester = userFacade.tryGetUserByEmail(jwtTokenProvider.getEmail(token));
        return requestService.createRequest(createRequestDto, location, requester);
    }
    public Iterable<Request> getAllRequests() {
        return requestService.getAllRequests();
    }
    public Iterable<Request> getMyRequests(String token) {
        User user = userFacade.tryGetUserByEmail(jwtTokenProvider.getEmail(token));
        if (user.getRole().getName().equals("VOLUNTEER")) {
            return requestRepository.findAllByVolunteerId(user.getId());
        }
        return requestRepository.findAllByRequesterId(user.getId());
    }
    public Request takeRequest(String token, UUID requestId) {
        User user = userFacade.tryGetUserByEmail(jwtTokenProvider.getEmail(token));
        Request request = requestRepository.findById(requestId).orElseThrow();
        request.setVolunteer(user);
```

```
        request.setStatus(Status.IN_PROGRESS);
        return requestRepository.save(request);
    }
    public Request completeRequest(String token, UUID requestId) {
        User user = userFacade.tryGetUserByEmail(jwtTokenProvider.getEmail(token));
        Request request = requestRepository.findById(requestId).orElseThrow();
        request.setStatus(Status.FINISHED);
        return requestRepository.save(request);
    }
}
```

RequestService.java

```
package com.example.demo.service;
import com.example.demo.domain.*;
import com.example.demo.dto.CreateRequestDto;
import com.example.demo.repositories.RequestRepository;
import lombok.AllArgsConstructor;
import org.springframework.stereotype.Service;
@Service
@AllArgsConstructor
public class RequestService {
    private final RequestRepository requestRepository;
    public Request createRequest(CreateRequestDto createRequestDto,
                                 Location location, User requester) {
        Request request = Request.builder()
                .requester(requester)
                .title(createRequestDto.getTitle())
                .location(location)
                .details(createRequestDto.getDetails())
                .need(Need.valueOf(createRequestDto.getNeedType()))
                .status(Status.NOT_STARTED)
                .build();
        return requestRepository.save(request);
    }
    public Iterable<Request> getAllRequests() {
        return requestRepository.findAll();
    }
}
```

RouteBuilderFacade.java

```
@Service
```

```
@RequiredArgsConstructor
public class RouteBuilderFacade {
    private final RequestFacade requestFacade;
    private final UserFacade userFacade;
    private double[][] distanceMatrix;
    public List<Location> createRoute(String token) {
        List<Location> locations = getLocations(token);
        if(locations.size() <= 2) {
            return locations;
        }
        distanceMatrix = new double[locations.size()][locations.size()];
        for (int i = 0; i < locations.size() ; i++) {
            for (int j = 0; j < locations.size() ; j++) {
                if (i == j) {
                    distanceMatrix[i][j] = 0;
                } else {
                    distanceMatrix[i][j] = getDistance(locations.get(i), locations.get(j));
                }
            }
        }
        TspCalculator main = new TspCalculator(distanceMatrix);
        List<Integer> optimalPath = main.getTour();
        List<Location> optimalPathRequests = new ArrayList<>();
        for (Integer integer : optimalPath) {
            optimalPathRequests.add(locations.get(integer));
        }
        return optimalPathRequests;
    }
    private double getDistance(Location firstLocation, Location secondLocation) {
        final double earthRadius = 6371;
        double dLat = Math.toRadians(secondLocation.getLatitude().doubleValue() -
                firstLocation.getLatitude().doubleValue());
        double dLon = Math.toRadians(secondLocation.getLongitude().doubleValue() -
                firstLocation.getLongitude().doubleValue())
        double a = Math.sin(dLat / 2) * Math.sin(dLat / 2) +
                Math.cos(Math.toRadians(firstLocation.getLatitude().doubleValue())) *
                        Math.cos(Math.toRadians(secondLocation.getLatitude().doubleValue())) *
                        Math.sin(dLon / 2) * Math.sin(dLon / 2);
        double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
        return earthRadius * c;
    }
    private List<Location> getLocations(String token) {
```

```
        User userByToken = userFacade.getUserByToken(token);
        Iterable<Request> myRequestsIterable = requestFacade.getMyRequests(token);
        List<Request> myRequestsList = new ArrayList<>();
        myRequestsIterable.forEach(myRequestsList::add);
        var locations = new ArrayList<>(myRequestsList.stream()
                .filter(request -> request.getStatus().equals(Status.IN_PROGRESS))
                .map(Request::getLocation)
                .toList());
        locations.add(0, userByToken.getLocation());
        System.out.println("locations:" + locations);
        return locations;
    }
}
```

UserFacade.java

```
package com.example.demo.service;
import com.example.demo.domain.Location;
import com.example.demo.domain.Role;
import com.example.demo.domain.User;
import com.example.demo.dto.AuthRequestDto;
import com.example.demo.dto.AuthResponseDto;
import com.example.demo.dto.UserDto;
import com.example.demo.exception.ResourceNotFoundException;
import com.example.demo.jwt.JwtTokenProvider;
import com.example.demo.repositories.RoleRepository;
import com.example.demo.repositories.UserRepository;
import lombok.AllArgsConstructor;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.stereotype.Component;
import java.util.Map;
import java.util.UUID;
@Component
@AllArgsConstructor
public class UserFacade {
    private UserService userService;
    private LocationService locationService;
    private RoleRepository roleRepository;
    private UserRepository userRepository;
    private final JwtTokenProvider tokenProvider;

    private final AuthenticationManager authenticationManager;
    public AuthResponseDto createUser(UserDto userDto) {
```

```
        Location location = locationService.createLocation(userDto.getLocation());
        String roleName = userDto.getRoleName();
        Role role = roleRepository.findByName(roleName).orElseThrow(() ->
                new ResourceNotFoundException(Map.of("Role with this name does not exist", roleName)));
        User user = User.builder()
                .email(userDto.getEmail())
                .password((userDto.getPassword()))
                .details(userDto.getDetails())
                .role(role)
                .location(location)
                .build();
        user = userService.createUser(user);
        return new AuthResponseDto(tokenProvider.createToken(user.getEmail(), user.getRole()), user.getId().toString());
    }
    public User getUserByToken(String token) {
        String email = tokenProvider.getEmail(token);
        return tryGetUserByEmail(email);
    }
    public User tryGetUserById(UUID userId) {
        return userRepository.findById(userId)
                .orElseThrow(() -> new ResourceNotFoundException(Map.of("Such user does not exist", userId.toString())));
    public AuthResponseDto login(AuthRequestDto requestDto) {
        authenticationManager.authenticate(new UsernamePasswordAuthenticationToken(requestDto.getEmail(),
                requestDto.getPassword()));
        User user = tryGetUserByEmail(requestDto.getEmail());
        return new AuthResponseDto(tokenProvider.createToken(user.getEmail(), user.getRole()), user.getId().toString());
    }
    public User tryGetUserByEmail(String email) {
        return userRepository.findByEmail(email)
                .orElseThrow(() -> new ResourceNotFoundException(Map.of("User with such email does not exist",
email)));
    }
}
```

app.module

```
import {CUSTOM_ELEMENTS_SCHEMA, InjectionToken, NgModule} from '@angular/core';
import {BrowserModule} from '@angular/platform-browser';

import {AppComponent} from './app.component';
import {NgOptimizedImage} from "@angular/common";
import {SharedModule} from './shared/shared.module';
import {RegistrationModule} from "./feature/registration/registration.module";
import {ProfileModule} from './feature/profile/profile.module';
```

```
import {RequestsModule} from "./feature/requests/requests.module";
import {AppRoutingModule} from './app-routing.module';
import {StoreModule} from "@ngrx/store";
import {effects, reducers} from "./store";
import {EffectsModule} from "@ngrx/effects";
import {AuthEffects} from "./store/auth/auth.effects";
import {HTTP_INTERCEPTORS, HttpClientModule} from "@angular/common/http";
import {AuthInterceptorService} from "./auth/interceptor/token.interceptor";
export const WindowToken = new InjectionToken('Window');
export function windowProvider() {
  return window;
}
@NgModule({
  schemas: [CUSTOM_ELEMENTS_SCHEMA],
  declarations: [
    AppComponent
  ],
  imports: [
    StoreModule.forRoot(reducers),
    EffectsModule.forRoot(effects),
    BrowserModule,
    NgOptimizedImage,
    SharedModule,
    RegistrationModule,
    ProfileModule,
    RequestsModule,
    AppRoutingModule,
    HttpClientModule
  ],
  providers: [{provide: WindowToken, useFactory: windowProvider}, AuthEffects, {
    provide: HTTP_INTERCEPTORS,
    useClass: AuthInterceptorService,
    multi: true,
  }],
  bootstrap: [AppComponent]
})
export class AppModule {
}
```

app.routing.module

```
import {NgModule} from '@angular/core';
import {RouterModule, Routes} from "@angular/router";
```

```
import {VolunteerGuard} from "./auth/guards/volunteer.guard";
import {RequesterGuard} from "./auth/guards/requester.guard";
import {AuthorizedGuard} from "./auth/guards/authorized.guard";
const routes: Routes = [
  {
   path: '',
   redirectTo: 'requests',
   pathMatch: 'full'
  },
  {
   path: 'requests',
   loadChildren: () => import('./feature/requests/requests.module').then(m => m.RequestsModule)
  },
  {
   path:'route',
   loadChildren: () => import('./feature/route/route.module').then(m => m.RouteModule),
   canActivate: [VolunteerGuard]
  },
  {
   path: 'create-request',
   loadChildren: () => import('./feature/create-request/create-request.module').then(m => m.CreateRequestModule),
   canActivate: [RequesterGuard]
  },
  {
   path: 'registration',
   loadChildren: () => import('./feature/registration/registration.module').then(m => m.RegistrationModule)
  },
  {
   path: 'login',
   loadChildren: () => import('./feature/login/login.module').then(m => m.LoginModule)
  },
  {
   path: 'profile',
   loadChildren: () => import('./feature/profile/profile.module').then(m => m.ProfileModule),
   canLoad: [AuthorizedGuard]
  },
  {
   path: '**',
   loadChildren: () => import('./feature/requests/requests.module').then(m => m.RequestsModule)
  }
]
@NgModule({
```

```
  declarations: [],
  imports: [
    RouterModule.forRoot(routes)
  ],
  exports: [
    RouterModule
  ]
})
export class AppRoutingModule {
}
```

create-request.component.ts

```
import {Component, OnInit} from '@angular/core';
import {Observable} from "rxjs";
import {Router} from "@angular/router";
import {CategoryStateFacade} from "../../store/categories/category.facade";
import {FormBuilder, FormGroup, Validators} from "@angular/forms";
import {RequestStateFacade} from "../../store/request/request.facade";
import {CreateRequestDto} from "../../CreateRequestDto";
import {LocationDto} from "../../LocationDto";
import {AuthStateFacade} from "../../store/auth/auth.facade";
@Component({
  selector: 'app-create-request',
  templateUrl: './create-request.component.html',
  styleUrls: ['./create-request.component.css', '../../../assets/reset.css']
})
export class CreateRequestComponent implements OnInit {
  categories$: Observable<string[]> = new Observable<string[]>();
  center: google.maps.LatLngLiteral = {lat: 50.450001, lng: 30.523333};
  markerPosition: google.maps.LatLngLiteral = this.center;
  geocoder = new google.maps.Geocoder();
  isAuthenticated$ = new Observable<boolean>();
  zoom: number = 7;
  requestCreationForm!: FormGroup;
  login: string = 'LOGIN';
  logout: string = 'LOGOUT';
  profile: string = 'PROFILE';
  location: LocationDto = {
    latitude: 50.450001,
    longitude: 30.523333,
    nearby_city: 'Kyiv',
    exact_address: 'Kyiv'
  }
```

```
constructor(private fb: FormBuilder, private categoryFacade: CategoryStateFacade, private router: Router,
        private requestFacade: RequestStateFacade, private authFacade: AuthStateFacade) {
  console.log('CreateRequestComponent#constructor called')
  this.buildForm();
}
buildForm(): void {
  this.requestCreationForm = this.fb.group({
    details: ['', [Validators.required, Validators.minLength(10)]],
    category: ['', [Validators.required]],
    title: ['', [Validators.required, Validators.minLength(10)]]
  });
}

get title() {
  return this.requestCreationForm.get('title');
}
get details() {
  return this.requestCreationForm.get('details');
}
get category() {
  return this.requestCreationForm.get('category');
}
createRequest() {
  console.log('CreateRequestComponent#createRequest called')
  console.log(this.requestCreationForm)
  if (this.requestCreationForm.valid) {
    console.log('CreateRequestComponent#createRequest form is valid')
    const requestDto = {
      details: this.requestCreationForm.controls['details'].value,
      needType: this.requestCreationForm.controls['category'].value,
      location: {
        latitude: this.markerPosition.lat,
        longitude: this.markerPosition.lng,
        nearby_city: this.location.nearby_city,
        exact_address: this.location.exact_address
      },
      title: this.requestCreationForm.controls['title'].value
    } as CreateRequestDto;
    this.requestFacade.createRequest(requestDto);
  }
}
ngOnInit(): void {
```

```
    this.categories$ = this.categoryFacade.getCategories$;
    this.isAuthenticated$ = this.authFacade.isAuthorized$;

  onMapClick(event: google.maps.MapMouseEvent) {
    this.geocoder.geocode({location: event.latLng}, (results, status) => {
      console.log('results:', results)
      if (status == 'OK') {
        // @ts-ignore
        this.location.latitude = event.latLng.lat();
        // @ts-ignore
        this.location.longitude = event.latLng.lng();
        // @ts-ignore
        this.location.nearby_city = results[0].address_components.filter((component: any) =>
          component.types.includes('locality')
        ).at(0).long_name
        // @ts-ignore
        let result = results[0].address_components.filter((component: any) =>
          component.types.includes('route')
        )
        // @ts-ignore
        this.location.exact_address = results[0].formatted_address
        console.log('this.location:', this.location)
        this.markerPosition = {
          // @ts-ignore
          lat: event.latLng.lat(),
          // @ts-ignore
          lng: event.latLng.lng()
        };
      } else {
        alert('Geocode was not successful for the following reason: ' + status);
      }
    });
  }
  logoutReq() {
    this.authFacade.logout();
  }
  navigateToProfile() {
    this.router.navigate(['/profile']);
  }
}
```

router.component.ts

```
import {Component, ElementRef, OnInit, Renderer2, ViewChild} from '@angular/core';
```

```
import {map, Observable, of, tap} from "rxjs";
import {RequestStateFacade} from "../../store/request/request.facade";
import {Location} from "../../../Location";
import {DOCUMENT} from '@angular/common';
import {Inject} from '@angular/core';
import {Router} from "@angular/router";
import {AuthStateFacade} from "../../store/auth/auth.facade";
@Component({
  selector: 'app-route',
  templateUrl: './route.component.html',
  styleUrls: ['./route.component.css', '../../../assets/reset.css']
})
export class RouteComponent implements OnInit {
  login: string = 'LOGIN';
  logout: string = 'LOGOUT';
  profile: string = 'PROFILE';
  @ViewChild("mapElement")
  public mapElementRef: ElementRef | undefined;
  @ViewChild("directionsRenderer")
  public directionsRendererRef: ElementRef | undefined;
  isAuthenticated$ = new Observable<boolean>();
  zoom: number = 7
  directionsService = new google.maps.DirectionsService();
  center: google.maps.LatLngLiteral = {lat: 51.673858, lng: 7.815982};
  polylinePoints$: Observable<google.maps.DirectionsWaypoint[]> = new
Observable<google.maps.DirectionsWaypoint[]>()
  directions$: Observable<google.maps.DirectionsResult> = new Observable<google.maps.DirectionsResult>()
  constructor(private requestFacade: RequestStateFacade,
              @Inject(DOCUMENT) private document: Document,
              private renderer: Renderer2, private router: Router
    , private authFacade: AuthStateFacade) {
  }
  ngOnInit(): void {
    console.log('route ngOnInit')
    // @ts-ignore
    let locations: google.maps.DirectionsWaypoint[] = [];

    this.isAuthenticated$ = this.authFacade.isAuthorized$;
    this.requestFacade.route$.subscribe((route: Location[]) => {
      console.log('route', route);
      this.center = {lat: route[0].latitude, lng: route[0].longitude};
      route.forEach(location => {
```

```
      locations.push({
        location: {
          lat: location.latitude,
          lng: location.longitude
        },
        stopover: true
      })
    })
    // @ts-ignore
    locations.shift()
    //@ts-ignore
    locations.pop()
    let request = {
      origin: {lat: route[0].latitude, lng: route[0].longitude},
      destination: {lat: route[route.length - 1].latitude, lng: route[route.length - 1].longitude},
      waypoints: locations,
      travelMode: google.maps.TravelMode.DRIVING,
      optimizeWaypoints: true
    }
    this.directionsService.route(request, (result, status) => {
      console.log('result', result)
      console.log('status', status)
      if (status == 'OK') {
        console.log('result', result)
        // @ts-ignore
        this.directions$ = of(result)
      }
    })
  }
 )
}
logoutReq() {
  this.authFacade.logout();
}
navigateToProfile() {
  this.router.navigate(['/profile']);
}
```

ДОДАТОК Б

ВІДГУК КЕРІВНИКА ЕКОНОМІЧНОГО РОЗДІЛУ

ДОДАТОК В

ПЕРЕЛІК ФАЙЛІВ НА ДИСКУ

Ім’я файлу	**Опис**
Пояснювальні документи	
Диплом_Вареник.doc	Пояснювальна записка до кваліфікаційної роботи. Документ Word.
Диплом_ Вареник.pdf	Пояснювальна записка до кваліфікаційної роботи в форматі PDF.
Програма	
Program.rar	Архів. Містить коди програми
Презентація	
Презентація_Вареник.ppt	Презентація кваліфікаційної роботи.