

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Інститут інформатики та радіоелектроніки
Факультет комп'ютерних наук та технологій
(повне найменування інституту, факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавра

(ступінь вищої освіти (освітній ступінь))

на тему РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ ДЛЯ ПОШУКУ МІСЦЬ
ВОЛОНТЕРІВ НА ОСНОВІ ТЕХНОЛОГІЙ NODEJS ТА REACT

Виконав: студент 4 курсу, групи КНТ-517
спеціальності _____

123 «Комп'ютерна інженерія»

(код і найменування спеціальності)

Освітня програма (спеціалізація) _____

«Комп'ютерна інженерія»

Поліщук Владислав Олександрович

(прізвище та ініціали)

Керівник Скрупський С.Ю

(прізвище та ініціали)

Рецензент _____

(прізвище та ініціали)

м. Запоріжжя
2021 рік

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»
 (повне найменування вищого навчального закладу)

Інститут, факультет інформатики та радіоелектроніки, комп'ютерних наук і технологій
 Кафедра «Комп'ютерні системи та мережі»
 Ступінь вищої освіти (освітній ступінь) бакалаврський
 Спеціальність 123 Комп'ютерна інженерія
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерна інженерія
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри Кудерметов Р.К.

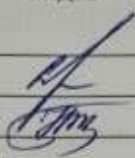
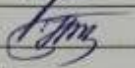
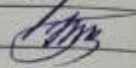
“ ” 2021 року

ЗАВДАННЯ
НА ДИПЛОМНИЙ ПРОЕКТ (РОБОТУ) СТУДЕНТА

Поліщука Владислава Олександровича
 (прізвище, ім'я, по батькові)

- Тема проекту (роботи) Розробка комп'ютерної системи для пошуку місій волонтерів на основі технологій NodeJs та React
 керівник проекту (роботи) Скрупський Степан Юрійович, к. т. н., доцент
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
 затверджені наказом вищого навчального закладу від “17” березня 2021 року № 81
- Строк подання студентом проекту (роботи) 06 травня 2021 року
- Вихідні дані до проекту (роботи) предметна область, загальні відомості про технології, структура веб-додатку, інтегроване середовище розробки,
- Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1) Аналіз предметної області;
2) Проектування комп'ютерної системи;
3) Розробка апаратної специфікації та інфраструктури;
4) Розробка програмного забезпечення системи.
- Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Плакат 1 - Блок – схема алгоритму роботи системи;
Плакат 2 - Порівняльна характеристика використаних технологій;
Плакат 3 - Схема бази даних системи;
Плакат 4 - Демонстрація роботи комп'ютерної системи.

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4	Скрупський С. Ю., к.т.н., доцент		
нормоконтроль	Щербак Н.В., ст. викл.		

7. Дата видачі завдання 01.03.2020 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проекту (роботи)	Строк виконання етапів проекту (роботи)	Примітка
1	Аналіз наявних методів та алгоритмів пошуку об'єктів, перегляд отриманих результатів, виділення недоліків та переваг	01.03.2021 р.	
2	Розробка структури системи	15.03.2021 р.	
3	Розробка алгоритму роботи системи	20.03.2021 р.	
4	Реалізація системи уточненого розпізнавання	01.04.2021 р.	
5	Дослідження системи уточненого розпізнавання	10.04.2021 р.	
6	Оформлення отриманих результатів у ПЗ	01.05.2021 р.	
7	Оформлення графічного матеріалу	10.05.2021 р.	
8	Оформлення допоміжного матеріалу	17.05.2021 р.	

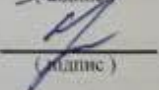
Студент


(підпис)

В.О. Полішук

(ініціали та прізвище)

Керівник проекту (роботи)


(підпис)

С.Ю. Скрупський

(ініціали та прізвище)

РЕФЕРАТ

ПЗ: 65 с., 11 рис., 2 таблиці, 31 лістингів, 33 джерел

JAVASCRIPT, REACT, API, NODEJS, EXPRESS, POSTGRESQL, БАЗА ДАННИХ, КЛІЄНТ-СЕРВЕР, РОЗРОБКА КОМП'ЮТЕРНОЇ СИСТЕМИ

Об'єктом проектування є розробка комп'ютерної системи на базі NodeJS та React технологій.

Метою проектування є розробити веб-додаток для пошуку місій для волонтерів.

Веб-додаток розділяться на клієнтську та серверну частину які будуть написані за допомогою різних технологій Javascript.

Результатом проектування є система, що демонструє веб-додаток для пошуку місій й волонтерів.

Додаток поліпшує пошук волонтерів для благодійних місій

Такий додаток можна застосовувати у багатьох країнах світу, розширювати функціонал та корисність.

ЗМІСТ

Скорочення та умовні позначки	7
Вступ.....	8
1 Аналіз предметної області	9
1.1 Аналіз об'єкту розробки.....	9
1.2 Аналіз інструменту для написання клієнтської частини	10
1.3 Аналіз ide для розробки.....	12
1.4 Аналіз інструментів серверної частини	14
1.5 Аналіз субд та бд.....	16
2 Проектування комп'ютерної системи.....	18
2.1 Вимоги до системи.....	20
2.2 Побудова алгоритму роботи системи	22
2.3 Наповнення сторінок системи	23
2.4 Структура бази даних системи	25
3 Розробка апаратної специфікації та інфраструктури	26
4 Розробка програмного забезпечення системи.....	28
4.1 Створення проекту у ide	28
4.2 Розробка клієнтської частини додатку	30
4.3 Розробка серверної частини.....	45
4.4 Тестування системи	53
Висновки	57
Перелік джерел посилання.....	59
Перелік графічного матеріалу.....	

Плакат 1 - Блок – схема алгоритму роботи системи	
Плакат 2 - Порівняльна характеристика використаних технологій.....	
Плакат 3 - Схема бази даних системи	
Плакат 4 - Демонстрація роботи комп'ютерної системи	

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

КС	- комп'ютерна система
ПЗ	- пояснювальна записка
СУБД	- система управління базами даних
CRUD delete	- створення читання модифікація видалення, create read update
API interface	- інтерфейс програмування додатків, application programming
REST	- передача репрезентативного стану, representational state transfer
IDE environment	- інтегроване середовище розробки, integrated development
MYC	- модель представлення контролер, model view controller

ВСТУП

Дуже швидких оборотів набирає волонтерський рух у світі, це поширена практика добровільної безоплатної допомоги. Останнім часом в Україні волонтерський рух дуже сильно розвинувся. По всій країні існують різні гуртки волонтерів, які допомагають іншим людям або країні.

Деякі волонтери допомагають сиротам, збирають їм нові речі, гроші, цукерки, інші допомагають військовим, хтось прибирає вулиці та парки, або очищую нашу природу.

Волонтерська допомога надається безоплатно за яку людина не отримує коштів. При цьому допомога родичам чи друзям не є волонтерством тому що це просто допомога. Усім подобається допомагати по різному але всіх людей об'єдную цей рух. Важливо пам'ятати, що волонтерство – це взаємовигідна діяльність. Воно допомагає не тільки тим, на кого безпосередньо спрямоване, а й суспільству в цілому, і, звичайно ж, самому волонтеру. Фактом є те що добрі справи та волонтерство роблять людину щасливою [1].

При погляді збоку цей рух видно що йому не вистачає якоїсь структури.

Такий рух потрібно рекламувати, та просувувати у маси, щоб люди до нього приєднувались та також допомагали. Різні волонтерські організації мають свої сторінки у соціальних мережах але в них нема спеціалізованих сайтів на яких такі організації мали б можливість знайти волонтерів, а волонтери знайти місії для допомоги.

Виникає потреба у додатку в якому можна знайти волонтерів та волонтерські організації

Додаток повинен відповідати таким критеріям:

- легка доступність;
- простота у використанні;
- зрозумілий інтерфейс усім користувачам.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз об'єкту розробки

Головною метою проекту є створення легко доступного додатку для пошуку волонтерів та місій. Одні з найчастіше використовуваних додатків в інтернеті є веб-сайти. А в останній час це веб-додатки так як до них є доступ як з мобільних пристроїв так і з комп'ютерів чи ноутбуків. Тому розробка веб-додатку є оптимальною для того щоб знайти найбільшу аудиторію користувачів.

Веб-додаток – розподілений застосунок, в якому клієнтом виступає браузер, а сервером — вебсервер [2].

Вебсервер – це сервер, що приймає HTTP-запити від клієнтів, зазвичай веббраузерів, видає їм HTTP-відповіді [3].

Так як у додатку є клієнтська та серверна частини, таких підхід називається REST API [5].

API – це інтерфейс розробки додатків. Це готовий код, який можна долучити до якогось проекту та він буде працювати так як він є самостійно існуючим та незалежним [4].

REST (Representational State Transfer) використовує HTTP як протокол передачі даних для запитів і відповідей [5]. Це архітектурний стиль який описав Рой Філдинг який використовує HTTP протокол для передачі даних, форматів даних які можна використовувати багато, найчастіше використовують тип JSON. Зв'язок встановлюється за допомогою URL шляху [5].

JSON - це текстовий формат обміну даними між комп'ютерами [6].

Щоб отримати доступу до API серверної частини існує кілька способів, але найбільш популярним рішенням є використання axios, заснованого на Promise HTTP-клієнта [7].

Axios – бібліотека створена для полегшення роботи с HTTP запитами у REST API додатках [7].

На рисунку 1.1 відображена схема REST API , у додатку клієнтська та серверна частини додатку будуть розроблені на JavaScript мові

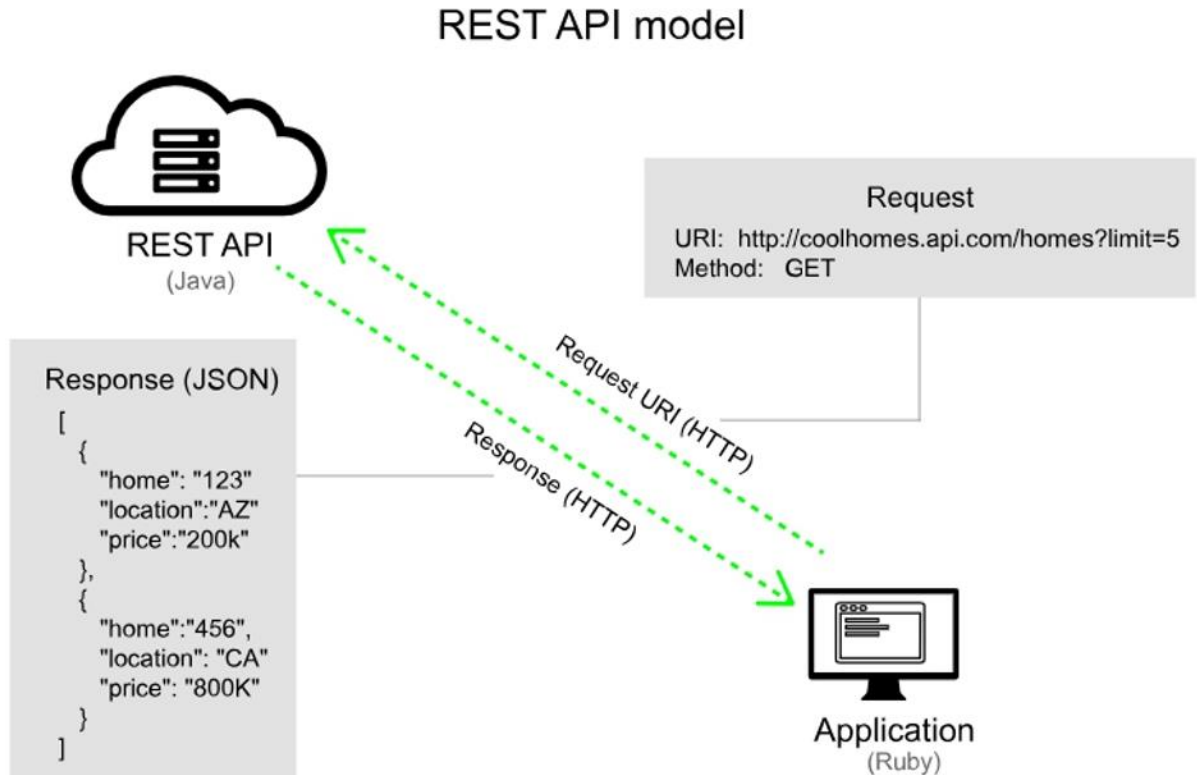


Рисунок 1.1 – Діаграма моделі REST API

Клієнт посилає запити до серверу за допомогою HTTP протоколу, ці запити приймає та обробляє, надає відповідь, найчастіше дані пересилаються у JSON форматі.

1.2 Аналіз інструменту для написання клієнтської частини

Одними з найпопулярніших мов для написання сайтів є JavaScript та PHP.

JavaScript – динамічна, об'єктно-орієнтована, прототипна, мова програмування, найчастіше використовується для написання веб-сторінок [8].

PHP – скриптова мова програмування, була створена для генерації HTML-сторінок на стороні веб-сервера [9].

Так, як розробляється динамічний клієнт-серверний веб-додаток, для цієї задачі підходить JavaScript. Для пришвидшення розробки веб-додатків дуже популярним зараз є використання бібліотек та фреймворків.

Для написання клієнтської сторони найпопулярніші є такі бібліотеки:

- Angular;
- React;
- Vue.

React - це бібліотека JavaScript, розроблена Facebook в 2013 році, яка відмінно підходить для створення сучасних односторінкових додатків будь-якого розміру і масштабу [10].

React дає змогу поєднати html-розмітку та JavaScript код у одне ціле, для цього використовується JSX-розмітка яку підтримую React з коробки

JSX – це розмітка UI дизайну яку можна одразу наповнити програмуючою логікою, це прискорює розробку, так як не потрібно спочатку писати html-розмітку та окремо під'єднувати до неї JavaScript код [11].

React заснований на створюванні компонентів які приймають вхідні данні та вкінці віддають розмітку яку обробляє браузер. Компоненти потрібно використовувати повторно щоб не дублювати код – це одне з головних правил React

React також може працювати на сервері, використовуючи Node.js і на мобільних платформах, використовуючи React Native [11].

Angular - це фреймворк від компанії Google для створення клієнтських додатків [12].

Vue - JavaScript-фреймворк з відкритим вихідним кодом для створення користувацьких інтерфейсів [13].

З порівняльної таблиці 1.1 фаворитами для розробки є Angular та React через великий розмір суспільства і популярність. Проте, так, як веб-додаток пошуку волонтерів це не дуже великий проект, перевага залишається за React через легкість вивчення та кращу гнучкість ніж Angular.

Таблиця 1.1 – Порівняння бібліотек та фреймворків React, Angular та Vue

Порівняльні параметри	Angular	React	Vue
Популярність	Висока	Висока	Середня
Різноманітність різних структур	Так	Ні	Ні
Легкість вивчення	Низька	Висока	Середня
Розмір суспільства	Великий	Великий	Низький
Гнучкість	Мала	Середня	Занадто велика

Таким чином ми проаналізували найпопулярніші бібліотеки та фреймворки для створення клієнтської частини веб-додатків та обрали бібліотеку React

1.3 Аналіз IDE для розробки

Для розробки веб-додатків можна використовувати багато IDE або текстових редакторів:

- Atom;
- Visual Studio Code (VSCode);
- WebStorm;

- Sublime Text;
- Netbeans.

Найпопулярнішими з віще описаних є WebStorm IDE та VSCode.

IDE – це інтегрована середовище розробки (Integrated Development Environment), текстовий редактор, маючий комплекс засобів для створення програмного забезпечення [14].

WebStorm IDE – інтегроване середовище розробки для JavaScript, HTML та CSS від компанії JetBrains, розроблена на основі платформи IntelliJ IDEA. WebStorm забезпечує автодоповнення, аналіз коду, навігацію по коду, рефакторинг, зневадження та інтеграцію з системами управління версіями [15].

Visual Studio Code – засіб для створення, редагування та зневадження сучасних вебзастосунків і програм для хмарних систем. Visual Studio Code розповсюджується безкоштовно і доступний у версіях для платформ Windows, Linux і OS X [16].

У WebStorm наявний дуже багатий функціонал який прискорює розробку додатків, та покращує продуктивність праці розробників якого нема у VSCode. У WebStorm є можливість більш зручно переглядати структуру проекту та робити маніпуляції з ними, наприклад перейменування файлів, він автоматично підставить нову назву у всіх місцях де цей файл використовується. Також, така сама можливість є зі змінними. Можна запускати Веб-додаток одразу у WebStorm після налаштування та він має дуже зручний інтерфейс для роботи.

Обидва інструменти мають можливість працювати з розподіленою системою Git.

Система контролю версій - це система, що записує зміни у файл або набір файлів протягом деякого часу, так що ви зможете повернутися до певної версії пізніше [17].

GIT – одна з таких систем для управління та зберігання проекту [17].

Git дуже полегшує зберігання та управління проектом у команді через те що у кожного є повна версія проекту з історією якою можна керувати та передавати.

Кожне середовище розробки має свої переваги та недоліки, виходячи з порівняльної таблиці 1.2 для веб-додатку краще використовувати WebStorm через те що він має більше можливостей що прискорює та полегшує розробку

Таблиця 1.2 – Порівняння IDE WebStorm та VScode.

Характеристики	IDE WebStorm	VSCode
Більша кількість можливостей	Так	Ні
Швидкість	Висока	Висока
Вузько спеціалізований напрям	Так	Так
Підтримка декількох мов розробки	Так	Так
Потреби потужності системи	Потребує великого об'єму ресурсів	Потребує великого об'єму ресурсів
Додаткові плагіни	Так	Так
Платне чи безплатне програмне забезпечення?	Платне	Безкоштовне
Наявність системи контролю версій GIT	Так	Так

1.4 Аналіз інструментів серверної частини

Для створення серверної частини веб-додатку також використовується мова Javascript, вона показала себе як високо продуктивна мова для серверів. JavaScript дуже універсальна мова програмування для неї існують безліч платформ навіть для

створення серверної сторони веб-додатків, таким чином вона є оптимальною для роботи у цій сфері, однією з таких платформ є Nodejs.

NodeJs - платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript [31].

У наші дні це найпопулярніша платформа для розробки ефективних та масштабованих REST API. Також за допомогою NodeJs можна створювати додатки для персональних комп'ютерів чи навіть для створення мереж.

Раніше на серверах використовувалась схема в якій клієнт відправляв запит до сервера, а сервер відправляв відповідь та припиняв з'єднання. Для того щоб сервер міг обробляти наступний запит йому потрібно було завершити попередній.

Для вирішення цієї проблеми робили багато поточні сервера котрі приймали декілька запитів одразу. Але неможливо додавати безліч потоків так як падає ефективність та зростає ціна

NodeJs це серверна платформа котра працює на рушії Google Chrome V8, який компілює JavaScript код до машинного. Node.js використовує подієво-орієнтовану модель і неблокуючу архітектуру вводу/виводу, що робить його легким та ефективним.

Таким чином на сервері не блокується основний потік операціями вводу або виводу, та код є асинхронним.

Насправді в NodeJs асинхронність реалізована за допомогою двох черг у циклі в яких виконуються задачі. Тому NodeJs є фаворитом для розробки серверної частини веб-додатків.

Для розробки серверної частини використовується веб-фреймворк Express, котрий є найпопулярнішим серверним фреймворком з багатою кількістю можливостей та дуже гарною гнучкістю [15]. Він спрощує створення сервера і серверної логіки, зокрема, обробка відправлених форм, робота з CORS та інше.

1.5 Аналіз СУБД та БД

Кожен додаток який працює з користувачами повинен десь зберігати данні про них. Для таких цілей створюється база даних кожного додатку в який зберігається та редагуються статичні дані про користувачів та інше. Дані зберігаються на зовнішніх носіях інформації. Усі маніпуляції з базою даних виконую СУБД.

СУБД - являє собою комплекс ПЗ, за допомогою якого можна створювати БД і проводити над ними різні операції: оновлювати, видаляти, вибирати, редагувати. СУБД гарантує збереження, цілісність, безпеку зберігання даних і дозволяє видавати доступ до адміністрування БД [18].

СУБД забезпечує:

- роботу з даними, розміщеними на зовнішніх накопичувачах;
- роботу з даними, що знаходяться в ОЗУ із застосуванням дискового кешу;
- ведення звітності стосовно: резервування, редагування, бекапу даних;
- підтримку різних мов баз даних (для роботи і визначення конкретних типів даних).

SQL – мова структурованих запитів, який був розроблений для роботи з БД, а саме, щоб отримувати, додавати, змінювати дані [20].

Дуже багата кількість СУБД використовую SQL. Найпоширеніші СУБД які використовуються це MySQL, PostgreSQL, Oracle, Microsoft SQL Server

MySQL – одна з найпоширеніших систем керування базами даних. Вона використовується, в першу чергу, для створення динамічних веб-сторінок, оскільки має чудову підтримку з боку різноманітних мов програмування [20].

Конкурентом є PostgreSQL котра також використовує мову запитів SQL, але має більші можливості.

PostgreSQL - широко розповсюджена система керування базами даних з відкритим кодом. Прототип був розроблений в Каліфорнійському університеті Берклі в 1987 році під назвою POSTGRES, та розвивається з того часу [19].

Ця СУБД має дуже високу надійність та продуктивність, гарну масштабованість, підтримую реплікації та транзакції, а також вбудовану мову PL/pgSQL.

Він має можливість створювати, зберігати та видавати складні структури даних через те що це об'єктно-реляційна БД.

Підтримується багатьма мовами програмування в тому числі й JavaScript.

В порівнянні з MySQL має дуже зрозумілу документацію в якій можна знайти дуже багато відповідей на запитання.

PostgreSQL розвивається групою програмістів, а MySQL компанією Oracle що призупиняє розвиток другого [20].

Для проекту кращим варіантом буде використання PostgreSQL через його більшу кількість можливостей та продуктивність.

Таблиця 1.3 – Порівняння СУБД PostgreSQL та MySQL

Порівняльні характеристики	PostgreSQL	MySQL
Тип СУБД	Об'єктно-реляційна	Реляційна
Має менші обмеження пам'яті	Так	Ні
Підтримка усіх стандартів SQL	Так	Ні
Має більшу продуктивність	Так	Ні
Більш швидкий розвиток	Так	Ні

Продовження таблиці 1.3

Порівняльні характеристики	PostgreSQL	MySQL
Легкість ПЗ	Середня	Висока
Гнучкість	Висока	Середня

2 ПРОЕКТУВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

Після визначення з форматом реалізації системи, а саме , web – додаток, необхідно обрати вид його макету на основі представлених функцій, переліку платформ на яких його будуть переглядати та аналізу цільової аудиторії та основного алгоритму роботи системи.

Сходячи з основної задачі системи – звести замовника (компаній, які розгортають волонтерські проекти) та виконавця (заінтересовану особу у наданні допомозі у вирішенні поставленого питання), система повинна включати в себе такі функції:

- можливість переглянути наявні пропозиції, що до виконання;
- можливість додавати, видаляти або редагувати нові волонтерські пропозиції - завдання»;
- можливість переглядати перелік обраних завдань;
- можливість переглянути детальну інформацію та статистику завдань.

Волонтерська діяльність – це справа у котрій не має обмежень за віку. У цьому напрямі можуть знайти себе як і молоді люди так і люди похилого віку. Але зважаючи на те що система працюватиме в інтернеті, діапазон користувачів звужиться.

За даними компанії InMind, користувачі Інтернету у віці 15-29 років у великих містах України становлять п'ятдесят п'ять відсотків, від тридцяти до сорока чотирьох років – тридцять п'ять відсотків, старше сорока п'яти років – десять відсотків. У містах з населенням менше п'ятдесяти тисяч. Чоловік ці цифри відповідно: сорок чотири відсотки, двадцять три відсотка і чотири, в селах – тридцять відсотків, одинадцять відсотків і два відсотка [31].

InMind – це постійно діючий сайт, що об'єднує людей, яким цікаво ділитися своєю думкою, беручи участь в онлайн опитуваннях за різними темами. Наша спільнота нараховує більше 100 000 осіб та ця цифра постійно зростає. Серед нас – люди різного віку, соціальних груп та інтересів, люди з сел, великих та малих міст, зі всіх регіонів України [31].

На основі вище описаного, можна зробити висновок що більша частина користувачів системи будуть віком від вісімнадцяти років, адже більшість видів допомоги зможуть надати лише повнолітні особи, до сорока п'яти років, адже у людей старшого віку, найчастіше, не має знань що до користуванням сучасними технологіями у сфері інтернету. Тож, система повинна мати простий та інтуїтивно зрозумілий інтерфейс та повинна бути реалізована у спокійній кольоровій гаммі і не вістити великої кількості анімованих елементів, що викликають перевантаження сторінки для читання користувачем.

Кожен з візитерів сайту є потенціальним замовником або виконавцем, то ж система повинна мати розділення інтерфейсу користувача на ці дві групи.

Найчастіше люди виходять до мережі Інтернет через браузер комп'ютера, ноутбука або смартфона, тож макет системи повинен коректно відображатись на усіх діагоналях екранів, мати адаптивне верстання для її відображення на мобільних пристроях та інших.

Адаптивної називається така верстка, при якій сайт коректно відображається на різних пристроях завдяки тому, що елементи динамічно підлаштовуються під різні дозволи екрану [21].

2.1 Вимоги до системи

Система повинна відповідати усім сучасним вимогам розробки web – додатків та усім вимогам обраної тематики системи.

Система повинна містити:

- можливість відображення на різних пристроях;
- повинна мати високу швидкість завантаження;
- можливість авторизації користувачів;
- наявність особистого кабінету для кожного з типів користувачів;
- можливість додавання, редагування або видалення інформативних блоків що до завдання.

Вимоги до реєстрації користувачів:

- можливість реєстрації через поширені сервіси обліку користувачів такі, як Google або Facebook;
- можливість зміни типу користувача.

Вимоги до особистого кабінету замовника послуг:

- можливість додавання, видалення або редагування завдань;
- можливість надати інформацію що до організації;

Вимоги до особистого кабінету виконавця (волонтера):

- можливість перегляду обраних завдань;
- можливість редагувати особисті дані;
- можливість змінити пароль

Вимоги до сторінки з завданням:

- надання інформації у блочній структурі;
- можливість сортування блоків;
- наявність фільтрів пошуку по сторінці;
- можливість перегляду детальної інформації кожного з блоків.

Вимоги до інформаційного вмісту блоків:

- тематичне зображення або логотип завдання;
- назва завдання;
- оцінка завдання;
- країна замовника;
- ім'я компанії замовника;
- статус замовлення.

Вимоги до дизайну:

- використання спокійної палітри кольорів;
- мінімалістичний дизайн;
- інформативність;
- наявність інтуїтивного інтерфейсу;
- адаптивність відображення(коректність) на різних пристроях.

Вимоги оптимізації:

- можливість відображення на різних пристроях;
- повинна мати високу швидкість завантаження.

Вимоги оптимізації:

- можливість відображення на різних пристроях;
- одночасне підтримання великої кількості запитів
- повинна мати високу швидкість завантаження.

Вимоги до надійності системи:

- система повинна зберігати робочий стан цілодобову у продовж всього часу її існування.
- Надавати відповіді на запити клієнтів у будь який час

2.2 Побудова алгоритму роботи системи

Перше що зустрічає користувач при зверненні до системи, це привітальну або головну сторінку, на якій користувачеві надається вступна інформація та блок з кнопками, які надають вибір користувачеві: зареєструватись у системі або створити новий обліковий запис.

Якщо користувач обирає авторизуватись то йому відкривається вікно з полями для вводу логіну (пошти) та пароллю, також, надається можливість входу до системи за обліковим записом Facebook та можливість відновлення доступу до системи через скидання пароллю.

Якщо користувач обрав зареєструвались у системі то відкриється вікно з полями вводу особистих даних користувача.

Надалі слідкує перевірка валідності наданих даних користувачем, та за відсутності помилок користувач реєструється у базі даних системи та переходить до сторінки з вибором своєї діяльності у системі (волонтер чи замовник) або одразу потрапляє на головну сторінку, якщо він вже був зареєстрован.

Після того, як користувач зробив вибір, його перекидає на сторінку з редагуванню профілю, надалі перевіряється валідність наданих даних та користувач потрапляє на головну сторінку сайту, де може за допомоги запиту знайти завдання на виконання від компаній.

Якщо користувач обрав «замовник» то після реєстрації своєї компанії його відкривається спливаюче вікно з пропозицією відразу перейти до створення замовлення (місії). Якщо користувач погоджується то його перекидає на сторінку створення місій, якщо ж ні – його перекидає на головну сторінку. Після створення місії можна перейти на сторінку цієї місії подивитись, редагувати інформації про місії можна с особистого кабінету. Та волонтери зможуть знайти цю місію та приєднатися до неї.

Блок- схема алгоритму роботи системи зображено на рисунку 2.1.

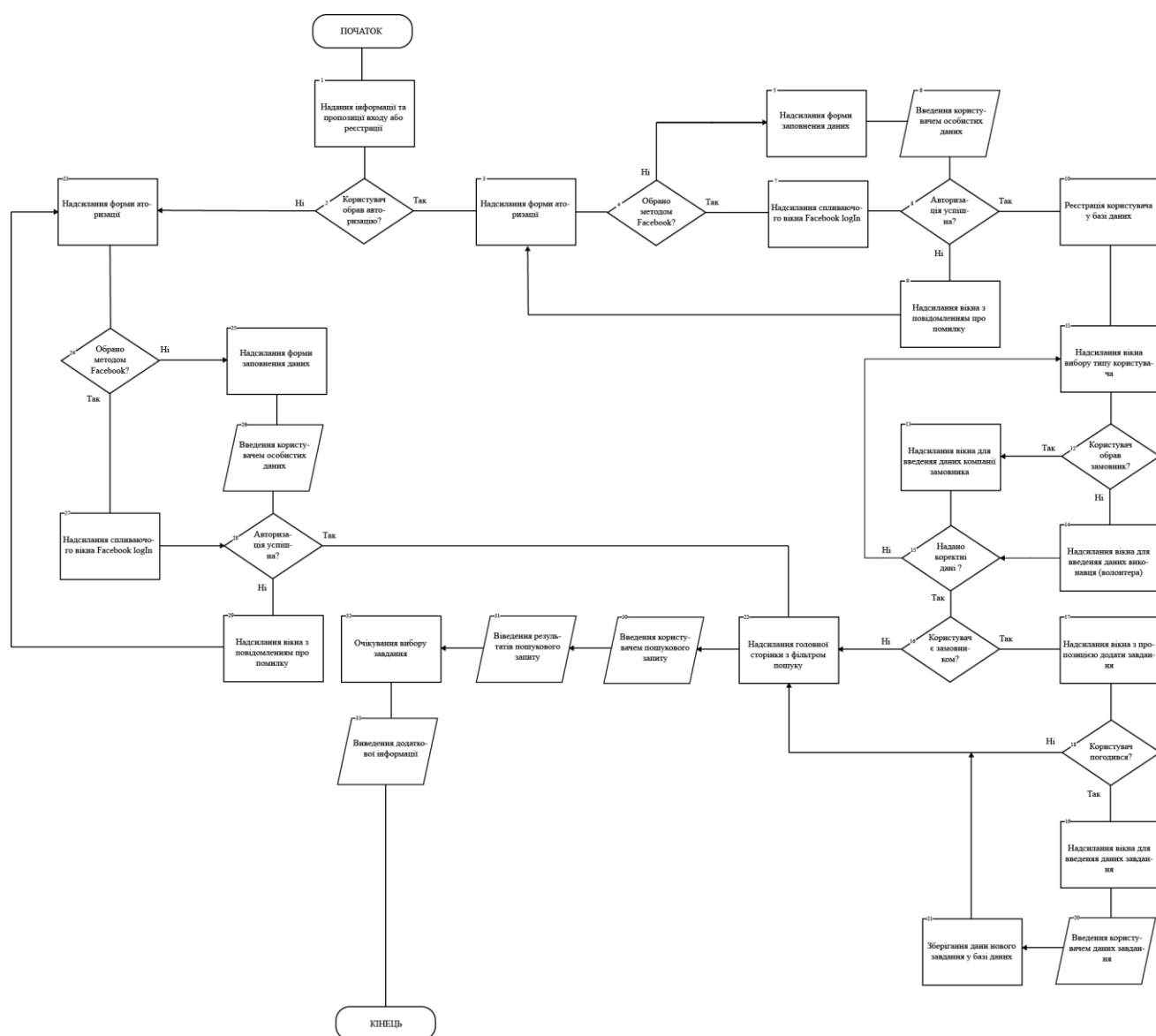


Рисунок 2.1 – Блок-схема роботи системи

2.3 Наповнення сторінок системи

Спираючись на усе вище описане необхідно створити макети сторінок системи.

Сторінки будуть адаптовані до використання адаптивного верстання, матимуть стриманий дизайн та інтуїтивний інтерфейс.

На основі описаних функцій та алгоритму роботи системи, вона повинна містити 4 сторінок та два вікна:

Головку сторінку, яка містить у собі:

- блок з капками реєстрації або входу;
- фоновим зображенням;
- блоком з інформацією;

Вікно реєстрації, яке містить у собі:

а) поле вводу:

- 1) логіну;
- 2) паролю;
- 3) імені;
- 4) фамілії.

б) кнопку реєстрації через Facebook;

в) кнопку відновлення паролю;

г) кнопку «реєстрація».

д) блок з полями фільтру;

е) блок відображення знайдених місій на карті;

ж) блок відображення знайдених місій;

з) блок зміни сторінки.

Вікно входу, яке містить у собі:

- поле вводу логіну;
- поле вводу пароля;
- кнопку входу через Facebook;
- кнопку відновлення паролю;
- кнопку «вхід».

Таким чином розробляється наповнення для усіх сторінок веб-додатку.

2.4 Структура бази даних системи

База даних повинна складатися з трьох таблиць, дивитись рисунок 2.2:

- таблиця User з даними о користувачах
- таблиця Mission з даними о місіях
- таблиця Structure для збереження інформації о організаціях

Таблиця User містить такі поля:

- firstName:string,lastName:string,email:string, dateOfBirthday:string,gender:string,location:string,phone:string,avatar:string
- firstName – ім'я користувача
- lastName – фамілія
- email – електрона адреса
- dateOfBirthday – дата народження
- gender – стать
- phone – мобільний телефон
- avatar – Фото користувача
- location – місце знаходження
- role – роль користувача

Таблиця Mission містить такі поля:

- name – назва місії
- location – місце знаходження місії
- start_date – початок місії
- end_date – кінець місії
- summury – опис місії
- volunteer_task – що потрібно робити волонтеру
- photos – фотографії місії
- duration – кількість днів
- day_for_request – кількість днів за котрі потрібно подати заявку на місії

Таблиця Structure містить такі поля:

- name – назва організації
- website – сайт організації
- location – місце знаходження
- description – опис організації
- avatar – фото організації

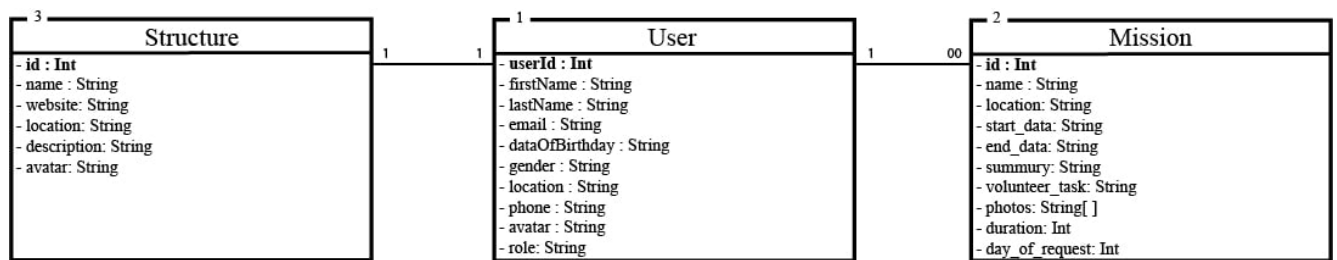


Рисунок 2.2 – Схема таблиць бази даних

Отже, після аналізу необхідних полів та створення на їх основі структури таблиць бази даних, було створено макет, за яким буде реалізовано базу даних системи.

3 РОЗРОБКА АПАРАТНОЇ СПЕЦИФІКАЦІЇ ТА ІНФРАСТРУКТУРИ

Вимоги до апаратної специфікації комп'ютерної системи залежать від цілей які вона повинна виконувати. Комп'ютерна система веб-додатку для пошуку волонтерів є клієнт-серверним додатком у якій є вимоги до сервера та до клієнту.

Щоб користувач міг використовувати дану систему йому потрібен клієнт веб-браузер. Веб-браузер це інструмент для відображення html документів який є як на персональних комп'ютерах так і у смартфонах.

Браузери які підтримє дана система :

- Google Chrome;
- Mozilla Firefox;
- Opera;
- Safari;
- та мобільні версії цих браузерів.

Для стабільної роботи клієнту потрібен постійний зв'язок з мережею інтернету для відправлення запитів та отримання відповіді від сервера.

Для стабільної швидкої та надійної роботи серверної частини додатку необхідно забезпечити сервер постійним безперебійним джерелом енергії швидкісним інтернетом с гарною роздільною здатністю інтернету.

Основні вимоги які повинен виконувати сервер:

- Сервер повинен мати локальне з'єднання за БД для швидкої обробки запитів;
- Мати пропускну здатність інтернету 100 Мбіт/с для одночасної роботи з 25 клієнтами;
- Для обробки запитів та запису до БД потрібно мати мінімум 4-х ядерний процесор та 16 ГБ оперативної пам'яті;
- Місце на жорсткому диску для зберігання інформації від 1000 клієнтів, необхідний мінімум 1 ТБ, для можливості подальшого масштабування додатку.

Таким чином для повноцінної роботи системи клієнтська частина повинна підтримувати технологію веб-браузеру та мати з'єднання с інтернетом. Серверна частина повинна працювати безперебійно зі стабільною пропускнуою здатністю інтернету, мати локальне з'єднання з БД та відповідати технічним характеристикам для роботи з декількома клієнтами одночасно.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ СИСТЕМИ

4.1 Створення проекту у IDE

Після аналізу додатку, хто буде ним користуватися, визначенням технічних завдань додатку можна розпочинати розробку. Вона починається зі створювання проекту в IDE чи у командному рядку. Для створювання React додатків існую спеціальний інструмент `create-react-app`, який надає змогу просто та швидко створити перший проект.

Для його роботи та роботи React в цілому потрібно встановити NodeJs платформу. На офіційному сайті NodeJs [32] потрібно завантажити та встановити останню версію даного ПЗ або використати спеціальну утиліту NVM, за допомогою якого можна встановлювати та управляти різними версіями NodeJs. Він включає у себе NPM та NPX.

NPM (Node Package Manager) – це інструмент для встановлення пакетів NodeJs, та найбільший у світі реєстр ПЗ [33].

NPX – новий інструмент для виконання та встановлення пакетів ПЗ.

Для того щоб створити новий проект у командному рядку потрібно ввести необхідну команду, текст якої наведено у лістингу 4.1.

Лістинг 4.1 – Команда створення нового проекту

```
npx create-react-app project_name
```

Також це можливо при створенні проекту в IntelliJ IDE WebStorm. Потрібно вибрати розділ Javascript, далі React , ввести необхідні дані для створення, його назву . Вибрати шлях де буде розміщений проект, версію Node , пакет за допомогою якого створиться проект та натиснути Finish. На рисунку 4.1 відображене меню вибору типу проекту в IntelliJ IDE WebStorm.

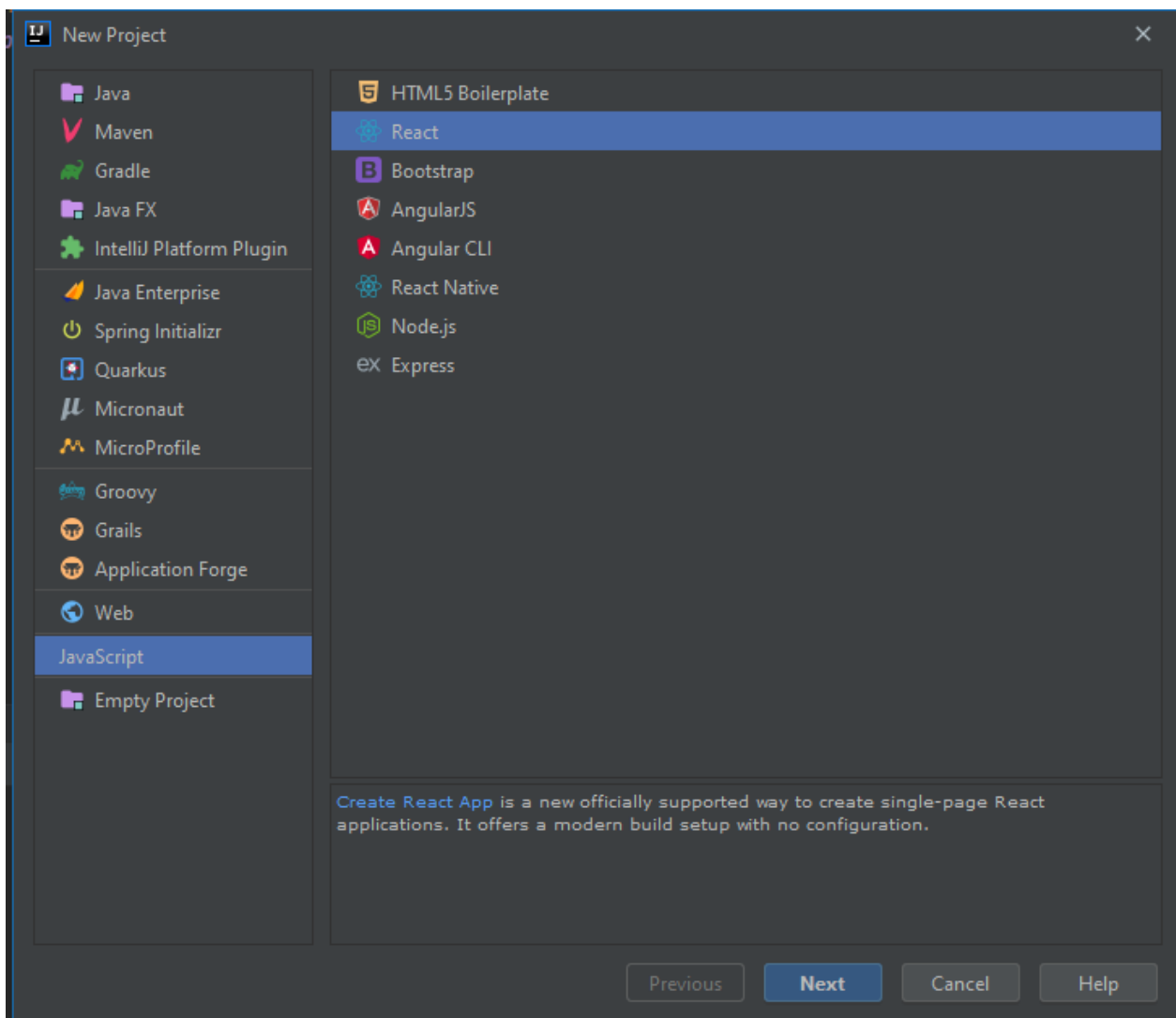


Рисунок 4.1 – Вікно створення проекту в IntelliJ IDE WebStorm.

Після цього буде створений проект React з початковою структурою, що представляє собою дві папки `public` та `src`. У каталозі `public` знаходиться наш головний файл `index.html`, саме цей файл відкриває браузер. У каталозі `src` знаходиться файл `index.js` який під'єднує React до `index.htm` та `App.js` – це головний React компонент який відображаються у нашому додатку та з якого починається розробка. React додаток створюється з компонентів. Всі файли які потрібні для роботи додатку потрібно створювати у папці `src`.

4.2 Розробка клієнтської частини додатку

React створює односторінковий додаток тому файл `index.html` всього один. Усе виконує Javascript він загрузає React додаток на сторінку та може змінювати його.

У додатку повинні бути сторінки для них створюється папка `pages` у каталозі в якій будуть знаходитись компоненти сторінок.

Для того щоб було декілька сторінок використовується бібліотека `react-router-dom` яка надає змогу зробити багату кількість сторінок

Розробка додатку починається з головного файлу `App.js`, в якому потрібно розмістити роутер додатку, шапку та підвал сайту.

Для того щоб у додатку була можливість використовувати історію браузера використовуються метод `createBrowserHistory` з бібліотеки `history`, та під'єднується до Router, який приймає `props history`.

В кожному файлі де використовуються React компоненти потрібно імпортувати бібліотеку React за допомогою коду який наведено у лістингу 4.2 У лістингу 4.3. наведено код компоненту `App.js`.

Лістинг 4.2 - Імпорт React

```
import React from 'react'
```

Лістинг 4.3 – Головний компонент App

```
class App extends PureComponent {
  render() {
    return (
      <Router history={history}>
        <Header />
        <AppModel />
        <Footer />
      </Router>
    )
  }
}
```

React компонент – клас який наслідую клас `Component` або `PureComponent`, має конструктор який приймає пропси - тобто вхідні дані, можна створити стан компоненту, функцію `render` - яка повертаю розмітку та методи життєвого циклу. Або також компонент може бути виконаний звичайною функцією яка приймає вхідні дані та повертаю розмітку, але в таких компонентів немає життєвого циклу, ця проблема була вирішена за допомогою React хуків.

React хук – нововведення, яке дозволяє використовувати стан і інші можливості React без написання класів, це функції в яких можна використовувати стан та працювати з ним за допомогою спеціальних інструментів React, таких як `useState`, `useEffect` та інші [23].

Приклад використання `useState` наведено у лістингу 4.4.

Лістинг 4.4 – Використання `useState` у React

```
import React, { useState } from 'react';

function Example() {
  const [count, setCount] = useState(0);
```

У компоненті `AppModel` знаходяться усі сторінки додатку, за відображення кожної сторінки в залежності від шляху на сайті відповідає `Router`, компонент `Route` приймає шлях та компонент, який буде відображати за цим шляхом.

Для прискорення роботи додатку усі сторінки потрібно загрузати за допомогою функції `lazy`, та відображати в середині компоненту `Suspense`, він надає змогу зачекати доки виконається динамічний імпорт за допомогою `lazy`, а у цей час відображати компонент який приймає у полі `fallback`.

У лістингу 4.5 наведено код компоненту `AppModal` з першою сторінкою `HomePage`.

Лістинг 4.5 – Компонент `AppModal`

```
import React, { lazy, Suspense } from 'react'

const HomePage = lazy(() => import('pages/Home'))
```

```

const routes = {
  home: "/"
}

const AppModel = () => (
  <Suspense fallback={<PreLoader />}>
    <Switch>
      <PublicRoute exact path={routes.home} component={HomePage}
    />
    </Switch>
  </Suspense>
)

```

Всередині додатку потрібно зберігати данні про користувачів та інше. Для цих задач використовуються бібліотека Redux , яка надає змогу з будь-якого місця додатку отримати данні у Redux. Він працює на основі ред'юсерів – це чисті функції які приймають сховище та віддають його новий стан.

Чиста функція - це функція котра повинна задовольняти двом умовам:

- кожного рази повертати один і той самий результат за одних вхідних параметрів;
- не мати побічних ефектів як наприклад прямої зміни вхідних параметрів [22]. Приклад такого ред'юсери наведено у лістингу 4.6.

Лістинг 4.6 – Код створення store за допомогою Redux

```

const initialState = {
  isInitiated: false,
  isSignInModalOpen: false,
}

const appReducer = (state = { ...initialState }, { type, payload }) => {
  switch (type) {
    case APP_INIT: {
      return { ...state, isInitiated: true }
    }
    case APP_SET_SIGN_IN_MODAL_STATE: {
      return {
        ...state,
        isSignInModalOpen: payload,
      }
    }
  }
}

```

```

    default:
      return state
  }
}
export default appReducer

```

Усі ред'юсери необхідно з'єднувати, цим займається функція `combineReducers()` з бібліотеки `Redux`. Застосування цієї функції наведено у лістингу 4.7.

Лістинг 4.7 – Код застосування функції `combineReducers`

```

import { combineReducers } from 'redux'
import { connectRouter } from 'connected-react-router'
import appReducer from '../modules/app/store/reducer'

const createRootReducer = (history) => combineReducers({
  app: appReducer,
  router: connectRouter(history),
})

export default createRootReducer

```

`Redux` необхідно підключити до проекту для цього створюється папка `store` з файлом `index.js` у якому створюється сховище даних.

Код створення `store` наведено у лістингу 4.8.

Лістинг 4.8 – Код створення `store` за допомогою `Redux`

```

import { createStore, compose, applyMiddleware } from 'redux'
import { createBrowserHistory } from 'history'
import thunk from 'redux-thunk'
import { routerMiddleware } from 'connected-react-router'
import createRootReducer from 'store/reducers/root-reducer'

const composeEnhancers = typeof window === 'object'
&& window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__
  ? window.__REDUX_DEVTOOLS_EXTENSION_COMPOSE__({
  }) : compose
const middlewares = [thunk]
const history = createBrowserHistory()
const store = createStore(
  createRootReducer(history),
  composeEnhancers(

```

```

    applyMiddleware(
      routerMiddleware(history),
      ...middlewares,
    ),
  ),
)

export { store, history }

```

Після цього будь який компонент можна під'єднати до сховища за допомогою функції `connect`, вона приймає першим параметром функцію для передачі даних зі сховища до компоненту, та другий об'єкт який складається з функцій дій зі сховищем, вони також передаються до компоненту. Приклад застосування функції `connect` наведено у лістингу 4.9.

Лістинг 4.9 – з'єднання сховища з компонентом за допомогою `connect()`

```

const mapStateToProps = (state) => ({
  missionsList: getMissionsList(state),
})
const actionsToProps = {
  getMissionsList: getMissionsListAction,
}
export default connect(mapStateToProps,
  actionsToProps)(HomePage)

```

Для зв'язку з сервером додатку потрібно налаштувати створення запитів до API. Бібліотека для цього була вибрана `axios`. Вона створює та оперує HTTP запитам з клієнтської сторони. Для нього створена папка `api` у каталозі проєкту. Нижче у лістингу 4.10 наведено код обробника запитів.

Лістинг 4.10 – Код обробника запитів

```

import axios from 'axios'
import { API_URL } from 'constants/main'

const defaultOptions = { isVolunteering: false }

const createRequestHandler = (options = {}) => {
  const { isVolunteering } = {
    ...defaultOptions,
    ...options,
  }
}

```

```

const urlPrefix = isVolunteering ? 'volunteering' : ''
const baseUrl = new URL(urlPrefix, API_URL)
const instance = axios.create({ baseURL: baseUrl.href })

return instance
}

export const baseRequestHandler = createRequestHandler()
export const volunteeringRequestHandler = createRequestHandler({
  isVolunteering: true })

```

Для сайту потрібно розробити шапку, в якій буде знаходитись логотип сайту та кнопки авторизації чи меню для аутентифікованих користувачів.

А також в ньому буде знаходитись мобільне меню яке буде замінювати меню для великих екранів. Щоб розділити відображення одно чи іншого компоненту в залежності в розміру екрану використовуються бібліотека react-responsiv.

Вона дозволяє створювати обгортки для компонентів приймаючи максимальних або мінімальний розмір до якого або від якого потрібно відображати компонент всередині.

Приклад використання цієї бібліотеки наведено у лістингу 4.11 відображення мобільного меню.

Лістинг 4.11 – Відображення мобільного меню тільки на малих екранах

```

<MQ.SM_DOWN>
  <div>
    <Icon
      extension="png"
      iconName="icon_menu"
      size="20x20"
      onClick={onMenuClick}
    />
  </div>
  <MobileMenu
    user={user}
    onMainClick={onMenuClick}
    mainToggle={mainToggle}
  />
</MQ.SM_DOWN>

```

Для того щоб це меню могло відкриватись та закриватись потрібно зберігати стан та змінювати його по кліку на іконку. Створена функція хук `_useToggle`, яка віддає стан кнопки та функцію яка змінює стан на протилежний.

Код функції дивитись у лістингу 4.12, а використання у шапці профіля у лістинку 4.13.

Лістинг 4.12 – Функція хук `_useToggle`, та її використання

```
import { useCallback, useState } from 'react'

const _useToggle = (initialState = false) => {
  const [state, setState] = useState(initialState)
  const toggle = useCallback(() => setState((state) => !state),
    [])

  return [state, toggle]
}

export default _useToggle
```

Лістинг 4.13 – Використання функції `_useToggle` у шапці сайту

```
const [mainToggle, onMainClick] = _useToggle(false)
const onMenuClick = () => {
  onMainClick()
  if (!mainToggle)
document.body.classList.add(classes.offScroll)
  else document.body.classList.remove(classes.offScroll)
}
```

З правої частини шапки потрібно розмістити кнопки для авторизації чи меню для авторизованих користувачів. В залежності від того є користувач чи ні відображаються потрібний компонент це наведено у лістингу 4.14.

Лістинг 4.14 – Умовне відображення в React

```
{user && user.userId ? (
  <AuthorizedActions userData={user} />
) : (
  <UnauthorizedActions />
)}
```

Компонент `UnauthorizedActions` повертає дві кнопки для реєстрації та аутентифікації які є тригерами для відкриття модальних вікон форм реєстрації та авторизації.

Кнопка авторизацію наведена у рисунку 4.2. Кнопки виконані компонентом `Button` код якого наведено у лістингу 4.15.

Лістинг 4.15 – Компонент `Button`

```
import React from 'react'
import PropTypes from 'prop-types'
import clsx from 'clsx'
import SpriteIcon from 'components/icons/SpriteIcon'
import classes from './Button.module.scss'

const Button = (
  {
    Btn,
    color,
    variant,
    text,
    size,
    spacing,
    onClick,
    disabled,
    type,
    nameIcon,
    sizeIcon,
  },
  => (
    <button
      // eslint-disable-next-line react/button-has-type
      type={type}
      className={clsx(classes[color], spacing, classes[size],
        classes[Btn], classes[variant], 'd-flex align-items-center')}
      onClick={onClick}
      disabled={disabled}>
      {nameIcon && <SpriteIcon name={nameIcon}
        className={classes.icon} size={classes[sizeIcon]} />}
      <span className={classes.text}>
        {text}
      </span>
    </button>
  )
)
```



Рисунок 4.2 – Кнопка авторизації

Модальні вікна зроблені за допомогою react-bootstrap а саме компонента Modal, та його основі створений компонент ModalWrapper, їх використання надане у лістингу 4.16

Лістинг 4.16 – Компонент UnauthorizedActions

```
import ModalWrapper from 'components/common/Modal'
import SignIn from 'modules/auth/SignIn'
import SignUp from 'modules/auth/SignUp'
import classes from
'components/partials/Header/ActionsBlock/Unauthorized/index.module.s
css'

import Button from '../../../../ui/Button/base'
const UnauthorizedActions = () => (
  <div className={classes.wrapper}>
    <ModalWrapper
      modalName="sign-in"
      headerTitle="Sign In"
      buttonClasses="mr-4"
      buttonSpacing="mr-4"
      buttonSize="small"
      onlySignInModal
      trigger={ (
        <Button
          type="button"
          text="Sign In"
          spacing="mr-4"
          size="small"
        />
      ) }>
      <SignIn />
    </ModalWrapper>

    <ModalWrapper
      modalName="sign-up"
      headerTitle="Sign Up"
      buttonText="Sign Up"
      buttonSize="small"
```

```

trigger={ (
  <Button
    type="button"
    text="Sign Up"
    spacing="mr-4"
    size="small"
  />
)}>
  <SignUp />
</ModalWrapper>
</div>
)
export default UnauthorizedActions

```

У модальних вікнах відображаються форми реєстрації та авторизації. Макет модального вікна форми авторизації зображено на рисунку 4.3.

Рисунок 4.3 – Макет модального вікна форми авторизації

Для збереження даних які вводить користувач під час заповнення форм використовується бібліотека Formik, вчасності хук useFormik, який дає змогу до використання всіх можливостей бібліотеки [25]. Він приймає початковий стан форми, схему для валідації даних та функцію підтвердження. Код використання при створенні форми авторизації наведено у лістингу 4.17.

Лістинг 4.17 – Використання useFormik у компоненті SignIn

```
const formikSignIn = useFormik({
  initialValues: {
    [formFields.email]: '',
    [formFields.password]: '',
  },

  validationSchema: Yup.object({
    [formFields.email]: Yup.string().email().required(),
    [formFields.password]: Yup.string().min(8).required(),
  }),

  onSubmit: (values, { resetForm }) => {
    signIn(values)
    resetForm()
    closeThisModal()
  },
})
```

У верхній частині форми знаходиться кнопка авторизації через Фейсбук. Вона реалізована за допомогою бібліотеки react-facebook-login, яка надає розробника зручний спосіб підключити авторизації через фейсбук. Код компоненту наведено у лістингу 4.18.

Лістинг 4.18 – Компонент FacebookAuth

```
import React from 'react'
import PropTypes from 'prop-types'
import FacebookLogin from 'react-facebook-login/dist/facebook-
login-render-props'
import Icon from 'components/common/Icon'
import { FACEBOOK_APP_ID } from 'constants/main'
import classes from './index.module.scss'

const FacebookAuth = ({ type, callback, onClickFunction }) => (
  <FacebookLogin
```

```

    appId={FACEBOOK_APP_ID}
    version={3.2}
    redirectUri={window.location.href.split('?')[0]}
    scope="public_profile, email"
    fields="email,name"
    callback={callback}
    onClick={onClickFunction}
    render={({renderProps}) => (
      <button
        type="button"
        className={classes.facebookButton}
        onClick={renderProps.onClick}>
        <Icon iconName="facebook-v2" />
        <span className={classes.facebookButtonText}>
          {type === 'signin' && 'Sign In '}
          {type === 'signup' && 'Sign Up '}
          via Facebook
        </span>
      </button>
    )}
  />
)

FacebookAuth.propTypes = {
  type: PropTypes.oneOf(['signin', 'signup']),
  callback: PropTypes.func,
  onClickFunction: PropTypes.func,
}
export default FacebookAuth

```

Наголовніше у цьому компоненті це функція `callback` яку компонент визиваю при завершенні входу через Фейсбук та ми можемо відправити дані о користувачі на сервер та авторизувати або зареєструвати його у нашому додатку.

Для авторизації був створений `AuthService` в якому знаходяться методи авторизації та роботи с токеном, код дивитись у лістингу 4.19. Для того щоб користувача при кожному заході на сайт не авторизувався, токен зберігається у сховищі браузера `LocalStorage`, воно існують, доки браузер не буде видалений. Але у токена є свій час життя для безпеки користувача.

Лістинг 4.19 – AuthService

```
import { baseRequestHandler, volunteeringRequestHandler } from
'api/http-client/request-handler'
const tokenKey = 'authToken'
const AuthService = {
  getToken() {
    return localStorage.getItem(tokenKey)
  },
  setToken(token) {
    if (!token) {
      return localStorage.removeItem(tokenKey)
    }
    return localStorage.setItem(tokenKey, token)
  },
  hasToken() {
    return !!this.getToken()
  },
  async signUp(formData) {
    const { authorization } = await
volunteeringRequestHandler.post('register', formData).then(({
result }) => result)
    this.setToken(`${authorization.token_type}
${authorization.access_token}`)
  },
  async signIn(formData) {
    const { authorization } = await
volunteeringRequestHandler.post('login', formData).then(({
result }) => result)
    this.setToken(`${authorization.token_type}
${authorization.access_token}`)
  },

  signOut() {
    this.setToken()
  },

  forgotPassword(formData) {
    return baseRequestHandler.post('forgot-password', formData)
  },

  resetPassword(formData) {
    return baseRequestHandler.post('reset-password', formData)
  },

  async facebookAuth(formData) {
    const { authorization } =
volunteeringRequestHandler.post('social/login',
formData).then(({ result }) => result)
    this.setToken(`${authorization.token_type}
${authorization.access_token}`)
  },
}
```

```

}

export default AuthService

```

Також до обробника запитів додатку додається переглядач запитів, котрий до кожного запиту додає токен. У HTTP протоколі для передачі токена у заголовках існує поле `Authorization` у яке зберігається токен та відправляється на сервер, код дивитись у лістингу 4.20.

Лістинг 4.20 – Додавання переглядача запитів до `api`

```

const requestInterceptor = (config) => {
  const headers = { ...config.headers }

  if (!headers.Authorization && AuthService.hasToken()) {
    headers.Authorization = AuthService.getToken()
  }

  if (config.method === 'get') {
    body[0].classList.add('loading-overlay')
  }

  return ({
    ...config,
    headers,
  })
}

instance.interceptors.request.use(requestInterceptor)

```

Після успішної авторизації виконується функція `getCurrentProfile`, яка робить запит на отримання даних користувача на сервер та зберігає до `Redux` сховища, код функції `getCurrentProfile` надан у лістингу 4.21.

Лістинг 4.21 – Функція `getCurrentProfile`

```

export const syncCurrentUserAction = () => async (dispatch) => {
  const payload = {}
  payload.user = await CurrentUserService.getCurrentUser()

  if (payload.user.isProfileFilled) {
    payload.profile = await
    CurrentUserService.getProfileById(payload.user.userId)
  }
}

```

```

    dispatch({
      type: SET_CURRENT_USER_AND_PROFILE,
      payload,
    })
    return payload
  }
  const getCurrentProfile = async (dispatch) => {
    const { user } = await dispatch(syncCurrentUserAction())
    if (!user.isProfileFilled) {
      dispatch(replace(routes.createProfile))
    } else {
      dispatch(replace(formatRoute(routes.dashboardProfile, {
tab: 'personal-info' }))))
    }
  }
}

```

Після створення реєстрації, авторизації та зберігання даних про користувача у Redux. Створюються інші сторінки на основі компонентів. На головній сторінці зверху форма для швидкого пошуку місій яка відправляє користувача на сторінку пошуку місій, нижче приклади місій сайту та підвал. Дивитись рисунку 4.3.

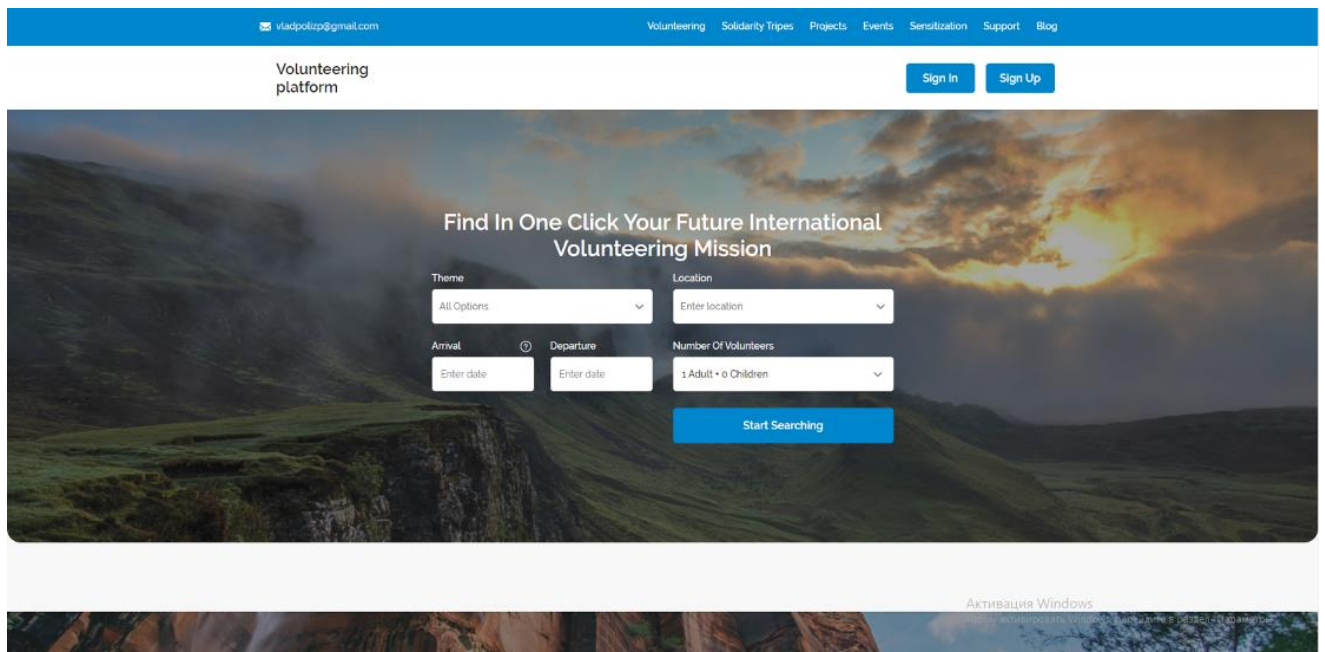


Рисунок 4.3 – Головна сторінка сайту

Також, створюється сторінка створення структури на яку користуувач потрапляє після реєстрації як організація, сторінка профіля, налаштування та місії.

Вкінці розробки клієнтської частини був отриманий односторінковий сайт написаний за допомогою технологій React та Redux, з використанням допоміжних бібліотек для роботи з формами, запитами до сервера та адаптивної верстки сайту, авторизації за допомогою Facebook та інші.

4.3 Розробка серверної частини

Найголовніша ціль роботи сервера, це відповідати на запити які надходять від клієнтів та зберігати дані у БД. Для цих цілей була обрана середа NodeJs.

Розробка серверу на NodeJs починається з команди `npm init`, для того щоб ініціалізувати роботу npm, після цього створиться файл `package.json` зі вмістом, який дивитись у лістингу 4.22.

Лістинг 4.22 – `package.json` після команди `npm init`

```
{
  "name": "dip",
  "version": "1.0.0",
  "description": "",
  "main": "index.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Після ініціалізації npm доступні його команди для робота з ним у проекті.

Для початку, необхідно встановити веб-фреймворк Express, для цього потрібно виконати команду `npm install express`, створити папку `src`, в якій будуть знаходитись усі каталоги та файли серверної частини. Створюємо файл `index.js`, який є головним для сервера.

Для зручного написання коду, та щоб можна було робити імпорти за допомогою ES6 imports замість CommonJs imports треба підключити до серверної частини проекту Babel.

Babel – це набір інструментів, який в основному використовується для перетворення коду ECMAScript 2015+ у поліфіл JavaScript для старих браузерів чи середовищ [26].

Поліфіл - це фрагмент коду (у мережах - зазвичай JavaScript), який дозволяє використовувати сучасну функціональність у більш старих браузерах, які не підтримують її [27].

Для цього потрібно його встановити за допомогою npm та створити файл .babelrc з його налаштуванням. Також для зручності встановлюється пакет nodemon, це інструмент який автоматично перезапускає сервер після змін у коді.

Після цих перших налаштувань можна написати index.js, в якому створюється об'єкт типу express, в нашому випадку це app, до функції use передаються middleware cors, щоб з'єднання не блокувалося браузером. Також для того щоб використовувати у тілі запитів Json структуру потрібно використовувати express.json().

Middleware - це функції, які мають доступ до об'єкта запиту (req), об'єкту відповіді (res) і до наступної функції тимчасової роботи в циклі "запит-відповідь" додатка [28]. Код index.js дивитись у лістингу 4.23.

Лістинг 4.23 – Головний файл сервера index.js

```
import express from 'express';
import router  from './routers/index.js';
import cors    from 'cors';

const PORT = process.env.NODE_PORT || 5000;
const app = express();

app.use( cors() );
app.use( express.json() );
app.use( '/api', router );

app.listen( PORT, () => console.log( `Example app listening on
port ${PORT}!` ) );
```

Усі шляхи які буде прослуховувати сервер створюються у папці routes

Один роутер, який прослуховує шлях створюється за допомогою функції `express.Router()`;

Кожен роутер може прослуховувати підшляхи запитів прослуховування кінцевих шляхів здійснюється за допомогою об'єкту роутера `route` у методах `get`, `post`, `put` та `delete` для різних дій з одним шляхом.

Таким чином у файлі `routes/index.js` створюється основний роутер який приймає запити, код наведено у лістингу 4.24.

Лістинг 4.24 – Код основного роутера сервера

```
import express                from 'express';
import * as ErrorHandlers     from
'../../middlewares/errorHandler/index.js';
import userRouter             from './user.js';
import authenticationRoute     from './authentication.js';
import { checkAuthorization } from
'../../middlewares/authorization';

const router = express.Router();

router.use( authenticationRoute );
router.use( checkAuthorization );
router.use( userRouter );

router.use( ErrorHandlers.handleValidationError );
router.use( ErrorHandlers.handleApplicationError );
router.use( ErrorHandlers.handleSequelizeError );

export default router;
```

Для збереження даних користувачів була обрана БД PostgreSQL. Спочатку треба її встановити для цього потрібно виконати інструкцію з установки з офіційного сайту.

Потрібно створити базу даних для цього використовується команда `createdb_name`. Для з'єднання БД з сервером використовуються ORM Sequelize.

ORM (Object-relational mapping Об'єктно-реляційна проекція) — технологія програмування, яка зв'язує бази даних з концепціями об'єктно-орієнтованих мов програмування [29].

Sequelize - це ORM-бібліотека для додатків на Node.js, яка здійснює зіставлення таблиць в БД і відносин між ними з класами [30]. За допомогою цієї бібліотеки можна працювати в БД з кожною таблицею як з об'єктами.

Для роботи з цією бібліотекою потрібно встановити її за допомогою команди код якої дивитись у лістинг 4.25

Лістинг 4.25 – Код команди встановлення Sequelize

```
npm install --save sequelize sequelize-cli.
```

Sequelize-cli потрібен нам для того щоб можна було виконувати sequelize команди у командному рядку. Для роботи з БД Postgres потрібно встановити додатки для зв'язку за допомогою команди `npm install --save pg pg-hstore`.

Після цього у корні проекту потрібно виконати команду `sequelize init`, яка створить 4 каталоги: `models`, `migrations`, `seeders` та `config`. У каталозі `model` знаходяться усі моделі БД та файл `index.js` за допомогою якого відбувається з'єднання з базою даних код цього файлу наведено у лістингу 4.26.

Лістинг 4.26 – Файл `models/index.js`

```
import path      from 'path';
import Sequelize from 'sequelize';

const basename = path.basename( __filename );
const env = process.env.NODE_ENV || 'development';
const config = require( __dirname + '/../config/config.json'
)[env];
const db = {};

let sequelize;
if (config.use_env_variable) {
  sequelize = new Sequelize(
process.env[config.use_env_variable], config );
} else {
  sequelize = new Sequelize( config.database, config.username,
config.password, config );
}
```

```

fs
  .readdirSync( __dirname )
  .filter( file => {
    return (file.indexOf( '.' ) !== 0) && (file !== basename)
    && (file.slice( -3 ) === '.js');
  } )
  .forEach( file => {
    const model = sequelize['import']( path.join( __dirname,
file ) );
    db[model.name] = model;
  } );

Object.keys( db ).forEach( modelName => {
  if (db[modelName].associate) {
    db[modelName].associate( db );
  }
} );

db.sequelize = sequelize;
db.Sequelize = Sequelize;

module.exports = db;

```

У файлі config/config.json знаходяться налаштування для св'язку с БД.

Для створення таблиці у БД за допомогою sequelize потрібно створити міграцію та модель для використання її на сервері. Щоб створити модель та міграцію потрібно виконати команду у командному рядку, текст якої наведено у лістингу 4.27.

Лістинг 4.27 – Команда створення моделі та міграції

```

npx sequelize-cli model:generate --name User --attributes f
irstName:string,lastName:string,email:string,dateOfBirth:string,
gender:string,location:string,phone:string,avatar:string

```

Таким чином sequelize створить файл user.js у каталозі models, та міграцію у каталозі migrations. Приклад коду файлу міграції наведено у лістингу 4.28 , а моделі у лістингу 4.29

Лістинг 4.28 – Міграція таблиці користувача

```

'use strict';
const nameAttribute = {
  type: Sequelize.STRING,

```

```

    allowNull: false,
  };

module.exports = {
  up: (queryInterface, Sequelize) => {
    return queryInterface.createTable('User', {
      id: {
        allowNull: false,
        autoIncrement: true,
        primaryKey: true,
        type: Sequelize.INTEGER
      },
      firstName: nameAttribute,
      lastName: nameAttribute,
      email: {
        type: Sequelize.STRING,
        allowNull: false,
        unique: true,
      },
      phone: {
        type: Sequelize.STRING
      },
    });
  },
  down: (queryInterface, Sequelize) => {
    return queryInterface.dropTable('User');
  }
};

```

Лістинг 4.29 – Приклад коду Моделі користувача

```

'use strict';
import { NAME_PATTERN, SALT_ROUND } from '../constants';

import bcrypt from 'bcrypt';

export default (sequelize, DataTypes) => {
  const User = sequelize.define( 'User', {
    firstName: {
      type: DataTypes.STRING,
      allowNull: false,
      validate: {
        is: NAME_PATTERN,
      }
    },

    lastName: {
      type: DataTypes.STRING,
      allowNull: false,
      validate: {
        is: NAME_PATTERN,

```

```

    }
  },
  phone: {
    type: DataTypes.STRING,
    allowNull: false,
    validate: {
      isPhone: true
    }
  }
}, {} );
User.associate = function (models) {};
return User;

```

Усі необхідні асоціації налаштовуються всередині методу моделі `associate`.

Після цього можна запустити міграції для створення таблиці у БД за допомогою команди `prx sequelize-cli db:migrat`. Буде створена таблиця `Users`, та зараз можна працювати у `Nodejs` з цією таблицею через модель `User`.

CRUD операції над користувачем знаходяться у роутері `userRouter` який наведено нижче у лістингу 4.30.

Лістинг 4.30 – `userRouter`

```

import express from 'express';
import UserController from '../controllers/user.js';
const userRouter = express.Router();
userRouter.route( '/user(/:id)?' )
  .post( UserController.createUser )
  .get( UserController.getUserById )
  .patch( UserController.updateUserById )
  .delete( UserController.deleteUserById );

export default userRouter;

```

Код `UserController` наведено у лістингу 4.31

Лістинг 4.31 – `UserController`

```

import { User } from '../models';

class UserController {
  createUser = async (req, res, next) => {
    try {
      const userData = (await User.create( req.body )).get();

```

```

        delete userData.password;
        req.user = userData;
        next();
    } catch (e) {
        next( e );
    }
};

deleteUserById = async (req, res, next) => {
    try {
        const deletedRowCount = await User.destroy( {
where: { id:req.params.id }
        } );
        if (deletedRowCount) {
            res.send(deletedRowCount);
        }
    } catch (e) {
        next( e );
    }
};

getUserById = async (req, res, next) => {
    try {
        res.send( await User.findByPk( req.params.id, {
            attributes: {
                exclude: ['password']
            }
        } ) );
    } catch (e) {
        next( e );
    }
};

updateUserById = async (req, res, next) => {
    try {
        const [updatedRowCount, updatedRows] = await
User.update(req.body, {
            where: {
                id:req.params.id,
            },
            returning: true,
        })
        if (updatedRowCount) {
            delete updatedRows[0].password;
            return res.send( updatedRows[0] );
        }
    } catch (e) {
        next( e );
    }
};
}

export default new UserController();

```

За аналогією до користувача створюються інші роутери та моделі для зберігання та редагування у БД місій, організацій та іншого.

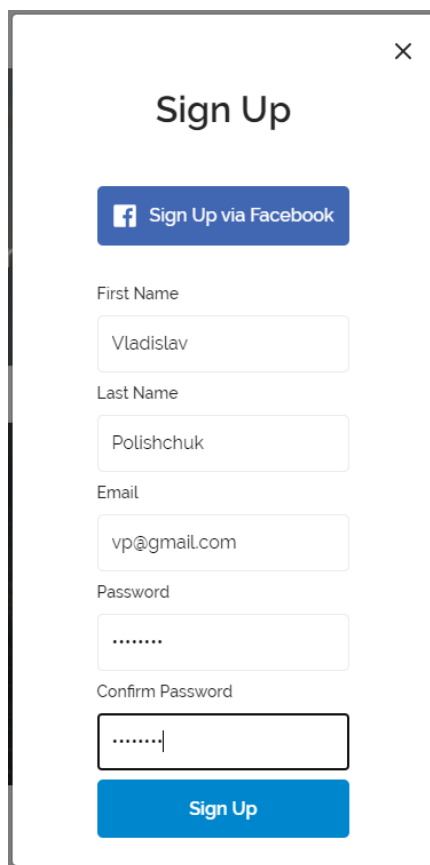
В результаті ми отримали серверну частину веб-додатку для пошуку місій та волонтерів написану за допомогою технологій NodeJS, веб-фреймворку Express, з використанням БД PostgreSQL та ORM Sequelize.

4.4 Тестування системи

Щоб провести тестування системи потрібно перевірити роботу сторінки додатку такі як реєстрація створення структури, пошук та створення місії.

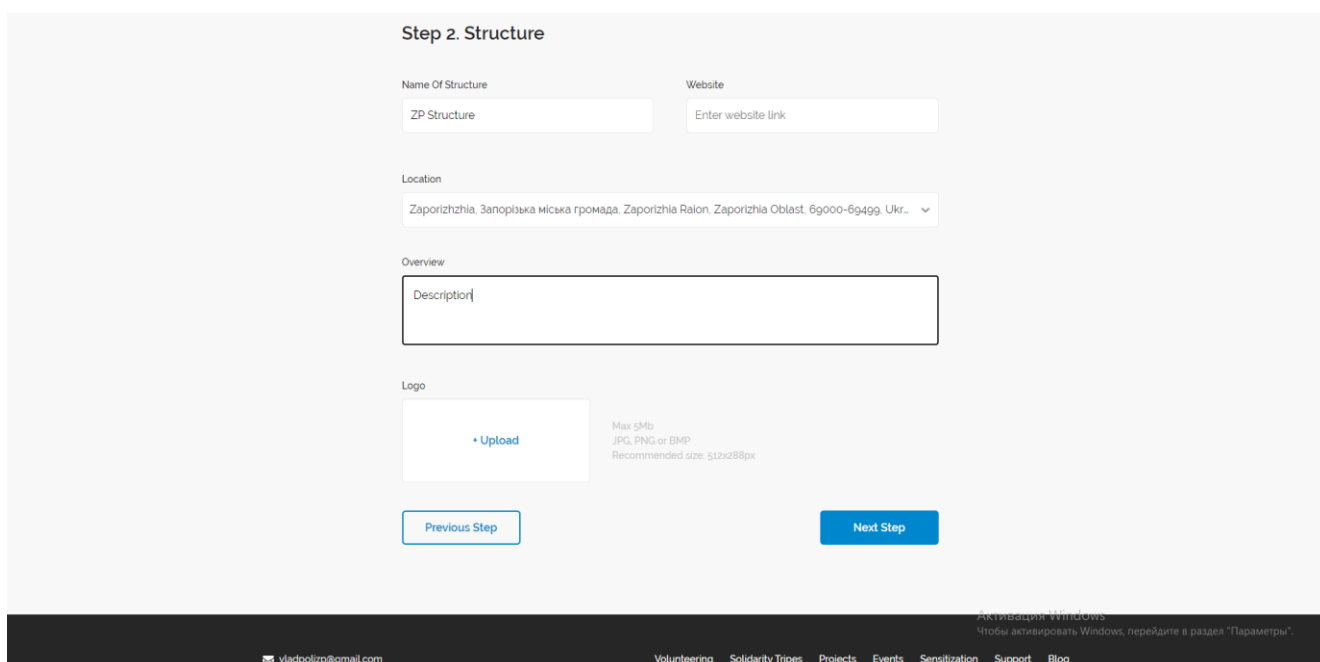
Після успішного тестування системи, бачимо що вікно реєстрації з заповненими полями наведено у рисунку 4.4.

Після успішної реєстрації користувача та вибору ролі волонтер або організація, потрібно заповнити форму з особистими даними та форму інформації про організацію, у якій потрібно заповнити назву організації, шлях до веб-сайту за наявності, місце знаходження організації, опис організації та за наявності фото організації, ця форма зображена на рисунку 4.5.



A mobile-style sign-up form titled "Sign Up" with a close button (X) in the top right corner. It features a blue button for "Sign Up via Facebook". Below this are input fields for "First Name" (containing "Vladislav"), "Last Name" (containing "Polishchuk"), "Email" (containing "vp@gmail.com"), "Password" (masked with dots), and "Confirm Password" (also masked with dots). A blue "Sign Up" button is at the bottom.

Рисунок 4.4 – Вікно реєстрації користувача



A web form titled "Step 2: Structure" for creating an organization. It includes input fields for "Name Of Structure" (containing "ZP Structure") and "Website" (containing "Enter website link"). A "Location" dropdown menu is set to "Zaporizhzhia, Запорізька міська громада, Zaporizhzhia Raion, Zaporizhzhia Oblast, 69000-69499, Ukr.". There is a large "Overview" text area with "Description" as a placeholder. A "Logo" section features an "Upload" button and text specifying "Max 5MB", "JPG, PNG or BMP", and "Recommended size: 512x288px". Navigation buttons "Previous Step" and "Next Step" are at the bottom. The footer contains a contact email "vladpolizp@gmail.com", a list of site sections (Volunteering, Solidarity Trips, Projects, Events, Sensitization, Support, Blog), and a Windows activation notice.

Рисунок 4.5 – Форма інформації про організацію

Після завершення реєстрації організації можна створити місію, а переглянути створені місії можна на сторінці особистого кабінету, ця сторінка наведена на рисунку 4.6.

Для того щоб переглянути усі місії які є на сайті потрібно зайти на сторінку пошуку місій, на якій розміщено фільтр місій, карта з маркерами знайдених місій та їх список, демонстрація сторінки пошуку наведена на рисунку 4.7

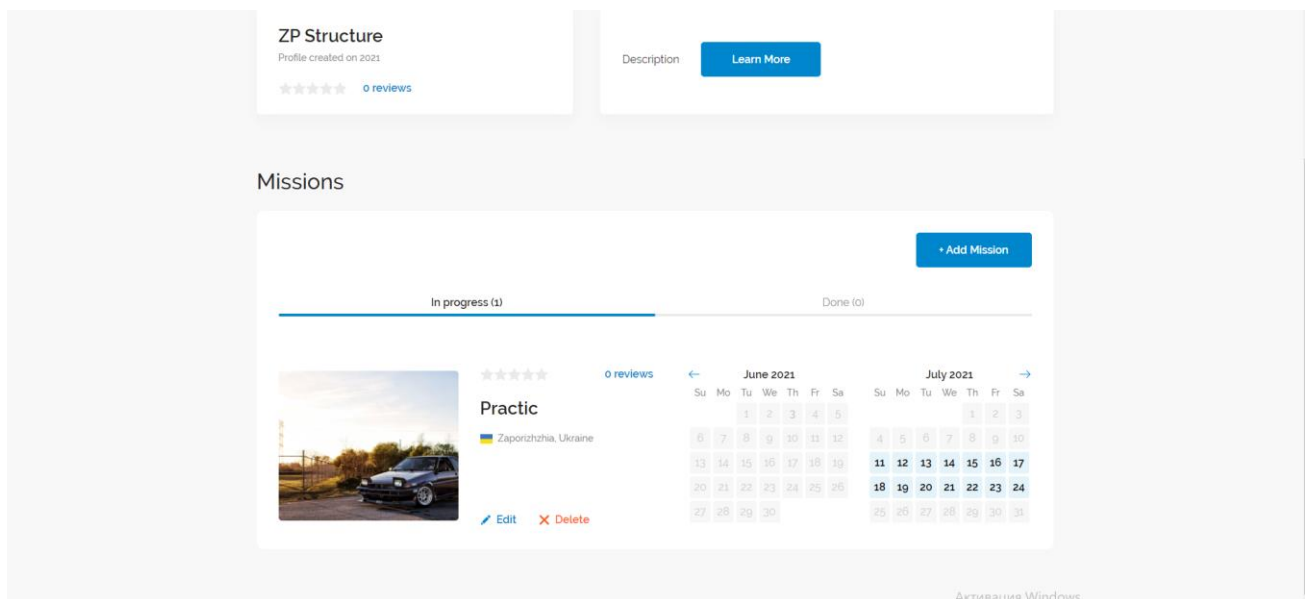


Рисунок 4.6 – Сторінка створених місій

Таким чином користувач може зайти на сайт та знайти волонтерську місію до якої захоче приєднатися та допомагати людям.

Сторінка меню системи має вигляд, при відпрацюванні адаптивного верстання для коректного відкриття на мобільних пристроях (аналогічним чином відображаються і інші сторінки системи), дивитись рисунок 4.8.

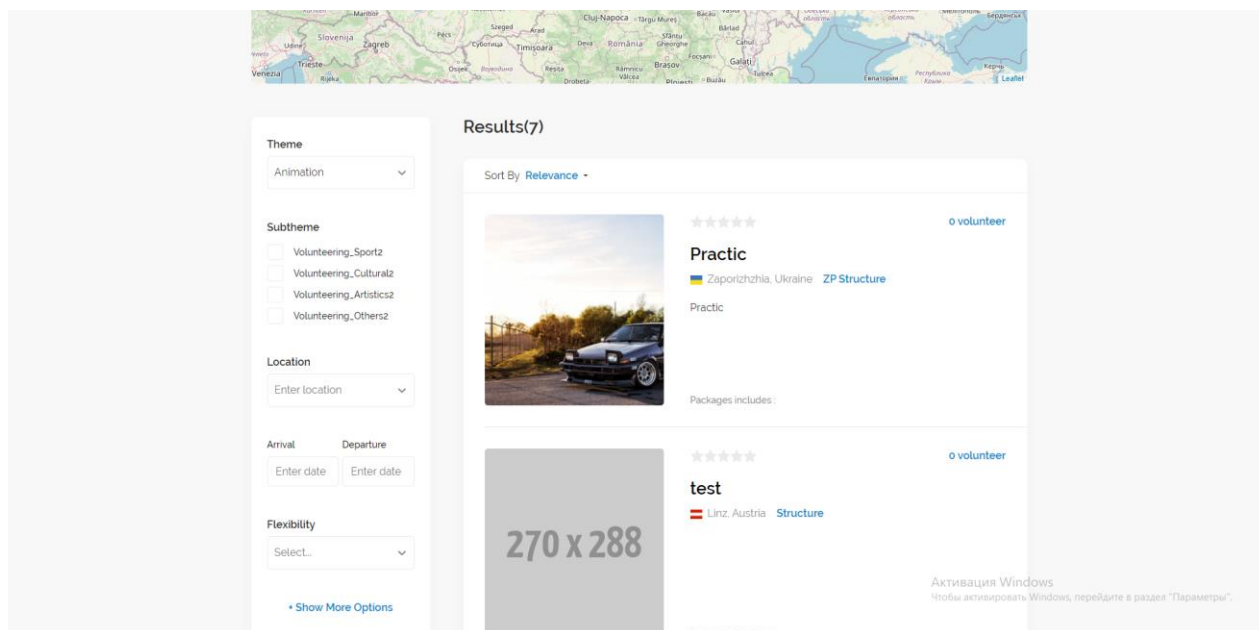


Рисунок 4.7 – Сторінка пошуку місій

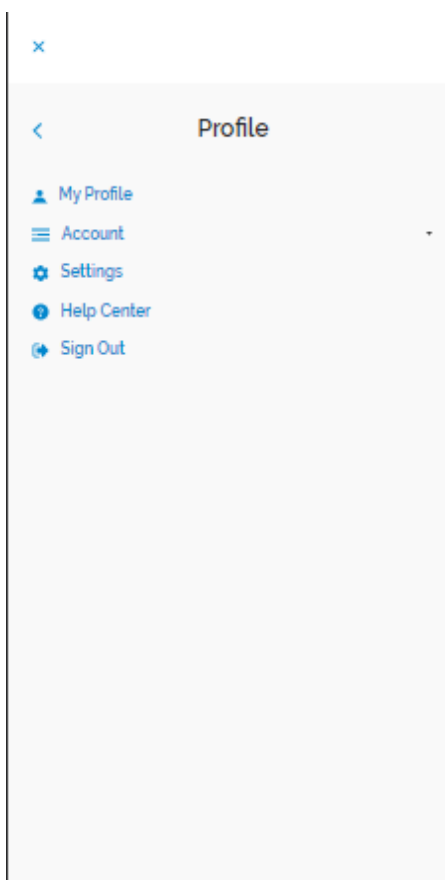


Рисунок 4.8 – Сторінка меню у мобільній версії

ВИСНОВКИ

Таким чином, в результаті була проаналізована предметна область для комп'ютерної системи яка буде створюватись та актуальність.

На даний момент в Україні не існують повністю цифрового додатку для пошуку місій в якому можна зареєструватися як користувач.

Були описані вимоги до комп'ютерної системи на основі яких проводились аналізи технологій які потрібно використовувати. Для охоплення найбільшої кількості аудиторії система представляє собою веб-додаток з клієнтською та серверною частинами.

На основі аналізу для клієнтської частини була обрана мова програмування JavaScript, так як це найзручніший інструмент для створення клієнтської частини.

Після вибору мови програмування був проведений аналіз інструментів що прискорюють та поліпшують розробку, для розробки була обрана бібліотека React, через її переваги, та те, що вона найбільш підходить до виконання цілей клієнтської частини додатку.

На основі аналізу двох найпопулярніших IDE, через більшу кількість переваг була обрана IDE WebStorm.

У аналізі для написання серверної частини додатку була обрана технологія NodeJS як найновіший інструмент для написання серверних частин на мові JavaScript. Маршрутизація та приймання запитів відбувається за допомогою веб-фреймворку Express.

Проаналізувавши бази даних серед найпопулярніших PostgreSQL та MySQL, була обрана БД PostgreSQL через те що вона більш продуктивніша ніж MySQL, та не має її недоліків. Для зв'язку серверної частини з БД була використана бібліотека Sequelize, яка надає можливість працювати з таблицями БД, як з об'єктами у JavaScript.

Після завершення аналізу усіх частин системи були сформовані макети сторінок веб-додатку, схема роботи додатку та схема БД.

За макетами була створена клієнтська частина веб-додатку зі сторінками, вікнами реєстрації та функціоналом, а також серверна частина з таблицями для функціонування додатку.

Останнім етапом розробки було тестування системи та виправлення недоліків. Після їх виправлення та проведення тестів додатку, його можна вважати завершеним.

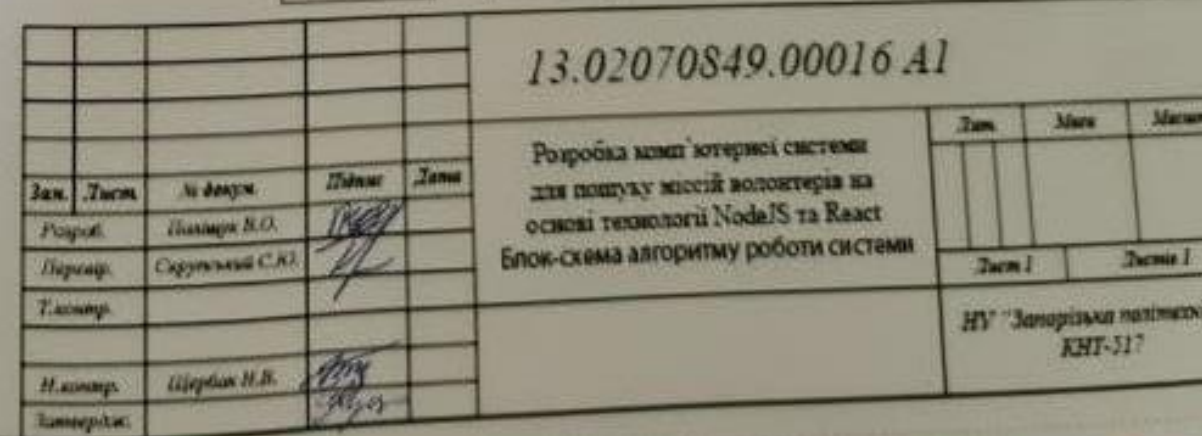
Таким чином ми отримали комп'ютерну систему котра надає змогу шукати волонтерів та місії для них у браузері, неважливо на ПК або у смартфоні.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

- 1 Волонтерський рух в Україні URL: https://pidru4niki.com/16330826/pravo/volonterskiy_ruh_ukrayini (дата звернення: 27.04.2021).
- 2 Веб-додаток URL: <https://uk.wikipedia.org/wiki/Вебзастосунок> (дата звернення: 29.04.2021).
- 3 Вебсервер URL: <https://uk.wikipedia.org/wiki/Вебсервер> (дата звернення: 20.02.2021).
- 4 Что такое API URL: <https://habr.com/ru/sandbox/52599/> (дата звернення: 29.04.2021).
- 5 Что такое REST API? URL: <https://starkovden.github.io/what-is-rest-api.html> (дата звернення: 29.04.2021).
- 6 Json URL: <https://uk.wikipedia.org/wiki/JSON> (дата звернення: 01.05.2021).
- 7 Используем axios для доступа к API: <https://ru.vuejs.org/v2/cookbook/using-axios-to-consume-apis.html> (дата звернення: 01.05.2021).
- 8 JavaScript URL: <https://uk.wikipedia.org/wiki/JavaScript> (дата звернення: 01.05.2021).
- 9 PHP URL: <https://uk.wikipedia.org/wiki/PHP> (дата звернення: 01.05.2021).
- 10 React URL: <https://ru.reactjs.org/> (дата звернення: 02.05.2021).
- 11 JSX URL: <https://ru.reactjs.org/docs/introducing-jsx.html> (дата звернення: 02.05.2021).
- 12 Angular URL: <https://metanit.com/web/angular2/1.1.php> (дата звернення: 02.05.2021).

- 13 Vue URL: <https://ru.vuejs.org/v2/guide/index.html> (дата
звернення: 02.05.2021).
- 14 IDE URL: <https://timeweb.com/ru/community/articles/5-luchshih-ide-1> (дата звернення: 02.05.2021).
- 15 WebStorm URL: <https://uk.wikipedia.org/wiki/WebStorm> (дата
звернення: 03.05.2021).
- 16 Visual Studio Code URL:
https://uk.wikipedia.org/wiki/Visual_Studio_Code (дата звернення: 03.05.2021).
- 17 Про систему контролю версій URL: <https://git-scm.com/book/uk/v2/Вступ-Про-систему-контролю-версій> (дата звернення: 03.05.2021).
- 18 Система управління базами даних URL: <https://itglobal.com/ru-ru/company/glossary/subd-sistema-upravleniya-bazami-dannyh/> (дата звернення: 04.05.2021).
- 19 PostgreSQL URL: <https://uk.wikipedia.org/wiki/PostgreSQL> (дата
звернення: 04.05.2021).
- 20 Сравнение MySQL и PostgreSQL URL:
<https://hyperhost.ua/info/ru/sravnenie-mysql-i-postgresql> (дата звернення: 05.05.2021).
- 21 Адаптивная верстка URL: <https://tproger.ru/translations/responsive-web-design-tips/> (дата звернення: 06.05.2021).
- 22 Что такое чистые функции в JavaScript? URL:
<https://habr.com/ru/post/437512> (дата звернення: 15.05.2021).
- 23 Введение в хуки URL: <https://ru.reactjs.org/docs/hooks-state.html> (дата
звернення: 15.05.2021).
- 24 Express URL: <https://expressjs.com/ru/> <https://ru.reactjs.org/docs/hooks-state.html> (дата звернення: 16.05.2021).
- 25 Formik URL: <https://formik.org/docs/api/useFormik> (дата звернення: 16.05.2021).

- 26 Babel URL: <https://babeljs.io/docs/en/index.html> (дата звернення: 19.05.2021).
- 27 Полифил URL: <https://developer.mozilla.org/ru/docs/Glossary/Polyfill> (дата звернення: 19.05.2021).
- 28 Middleware URL: <https://expressjs.com/ru/guide/using-middleware.html> (дата звернення: 20.05.2021).
- 29 ORM URL: https://uk.wikipedia.org/wiki/Об'єктно-реляційне_відображення (дата звернення: 20.05.2021).
- 30 Sequelize URL: <https://metanit.com/web/nodejs/9.1.php> (дата звернення: 20.05.2021).
- 31 Про нас InMind URL: <https://ask.inmind.ua/> (дата звернення: 20.05.2021).
- 32 Node Головна сторінка URL: <https://nodejs.org/uk/> (дата звернення: 25.05.2021).
- 33 NPM URL : <https://docs.npmjs.com/about-npm> (дата звернення: 25.05.2021).



Порівняльна характеристика використаних технологій

Порівняння СУБД PostgreSQL та MySQL

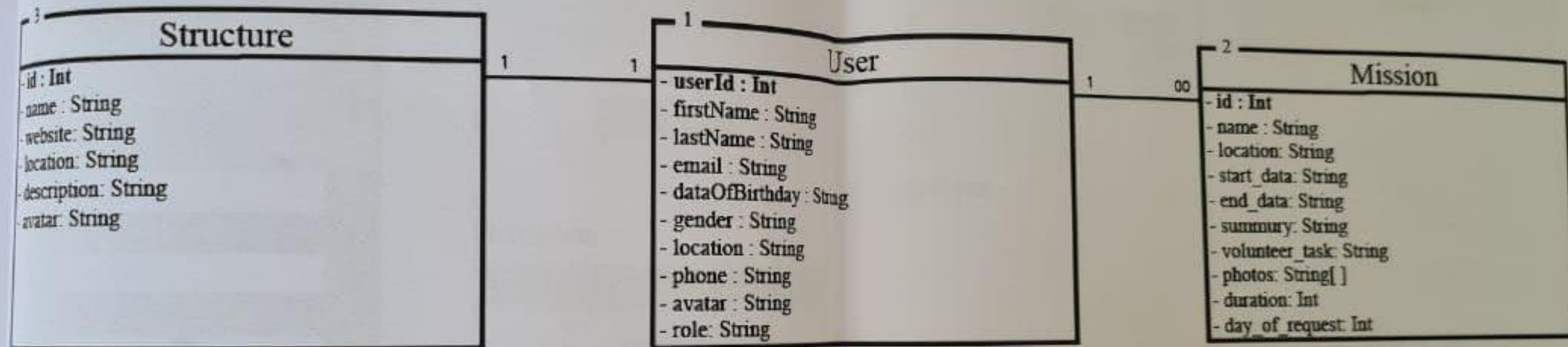
Порівняльні характеристики	PostgreSQL	MySQL
Тип СУБД	Об'єктно-реляційна	Реляційна
Має менші обмеження пам'яті	Так	Ні
Підтримка усіх стандартів SQL	Так	Ні
Має більшу продуктивність	Так	Ні
Більш швидкий розвиток	Так	Ні
Легкість ПЗ	Середня	Висока

Порівняння бібліотек та фреймворків React, Angular та Vue

Порівняльні параметри	Angular	React	Vue
Популярність	Висока	Висока	Середня
Різноманітність різних структур	Так	Ні	Ні
Легкість вивчення	Низька	Висока	Середня
Розмір суспільства	Великий	Великий	Низький
Гнучкість	Мала	Середня	Занадто велика

13.02070849.00016 ПЛІ													
Розробка клієнтської системи для пошуку місць волонтерів на основі технологій NodeJS та React													
Порівняльна характеристика використаних технологій													
Зам.	Лист	№ докум.	Підпис	Дата									
Розроб.		Поліщук В.О.											
Перевір.		Середюк С.Ю.											
Т.монтаж.													
Н.монтаж.		Щербак Н.В.											
Затверджен.													
<table border="1"> <tr> <td>Лист</td> <td>Місяць</td> <td>Місяць</td> </tr> <tr> <td></td> <td></td> <td></td> </tr> <tr> <td colspan="2">Лист 1</td> <td>Листів 1</td> </tr> </table>					Лист	Місяць	Місяць				Лист 1		Листів 1
Лист	Місяць	Місяць											
Лист 1		Листів 1											
НУ "Запорізька політехніка" КНТ-517													

Схема Базы Данных системы



13.02070849.00016 ПЛ2				
Зам.	Лист	№ докум.	Подпис	Дата
Разработ.		Полыгин В.О.		
Провер.		Скрябинский С.Ю.		
Утвержд.				
И. п. зам.		Щербин Н.В.		
Завершил:				
Разработчик: для запуска миссий волонтеров на основе технологии NodeJS та React Схема базы данных системы				
Лист 1 из 1 Лист 1 из 1				
ИУ "Автоматизация подготовки" КИТ-517				

Демонстрація роботи системи

Домашня сторінка

Volunteering platform

Find In One Click Your Future International Volunteering Mission

Theme

Location

Arrival Departure

Number Of Volunteers

Start Searching

Сторінка пошуку місій

Volunteering platform

ZP Structure

Missions

In progress (1)

Practic

Сторінка особистого кабінету

Volunteering platform

Results(20)

Practic

Сторінка мобільного меню

Profile

My Profile

Account

Settings

Help Center

Sign Out

Зам.	Лист.	М. докум.	Підпис	Дата
Розроб.		Поліщук В.О.		
Перевір.		Сарученко С.Ю.		
Т.м.м.м.				
Н.м.м.м.		Щербак Н.В.		
Затверджен.				

13.02070849.00016 ПЛЗ

Розробка мобільної системи для пошуку місій волонтерів на основі технологій NodeJS та React Демонстрація роботи системи

Дат.	Міся.	Місяц.
Дат. 1	Дат. 1	
НУ "Закарпатська національна		
КНТ-517		