КВАЛІФІКАЙНА РОБОТА

КР.ІПЗ – 18.00.000 ПЗ

Група ІПЗс-2017

Федуняк Р.І.

2021

ЗАКЛАД ВИЩОЇ ОСВІТИ
УНІВЕРСИТЕТ КОРОЛЯ ДАНИЛА

Факультет суспільних і прикладних наук
Кафедра інформаційних технологій

на правах рукопису

Федуняк Роман Ігорович

УДК 004.415

Розробка веб-платформи для організації волонтерської діяльності в умовах пандемії Covid-19.

Спеціальність 121 — «Інженерія програмного забезпечення»

Кваліфікаційна робота на здобуття освітнього ступеню бакалавра

Науковий керівник
кандитат технічних
наук, доцент,
Мануляк Ірина Зіновіївна

Івано-Франківськ — 2021

ЗВО «Університет Короля Данила»
Факультет суспільних і прикладних наук
Кафедра інформаційних технологій

Освітній ступінь: «бакалавр»
Спеціальність: 121 «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри

_____
«_____» _____ 202__ року

З А В Д А Н Н Я
НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ

_____
(прізвище, ім'я, по-батькові)

1. Тема роботи
_____
_____
_____
керівник роботи
_____,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом вищого навчального закладу від «__» _____ 201_ року № ____
2. Строк подання студентом роботи
_____
3. Зміст бакалаврської роботи (перелік питань, які потрібно розробити)
_____
_____
_____
_____
_____
_____
_____

4. Дата видачі завдання

_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз предметної області та технологій веб-розробки	27.03.2021	
2	Структура платформи табази даних	20.04.2021	
3	Реалізація та огляд платформи	23.05.2021	
4	Охорона праці	25.05.2021	
5	Оформлення пояснювальної записки	29.05.2021	
6	Оформлення графічного матеріалу та підготовка до захисту роботи	01.06.2021	

Студент _____ _____

(підпис) **(прізвище та інціали)**

Керівник роботи _____ _____

(підпис) **(прізвище та інціали)**

Вихідні дані:

_____

_____

_____

_____

_____

_____

_____

_____
_____
_____
_____

Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень):

Консультанти розділів роботи

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

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

- API – прикладний програмний інтерфейс (Aplication Programming Interface).
- Back-end – код серверної сторони програми.
- CMS – система керування вмістом (Content Management System).
- CORS – Cross-Origin Resource Sharing (спільне використання ресурсів з різних джерел) механізм безпеки сучасних браузерів.
- CRUD – 4 основні функції управління даними (Create, read, update, delete): створення, читання, оновлення і вилучення.
- CSS2 – каскадні таблиці стилів (Cascading Style Sheets).
- DOM – об'єктна модель документа (Document Object Model).
- DSL – предметно-орієнтована мова (Domain-specific language).
- ECMA – некомерційна асоціація європейських виробників комп'ютерів (European Computer Manufacturers Association).
- ECMAScript – стандарт мови програмування JavaScript.
- Front-end – клієнтська сторона, будь-який компонент, яким керує користувач.
- GUI – графічний інтерфейс користувача (Graphical user interface).
- HTML – мова розмітки гіпертексту (HyperText Markup Language).
- HTTP – протокол передачі Даних, что вікорістовується в комп'ютерних мереж. Назва скорочена від HyperText Transfer Protocol, протокол передачі гіпертекстових документів.
- HTTPS – схема URI, що синтаксично ідентична http: схемі, яка зазвичай використовується для доступу до ресурсів Інтернет.
- JSON – текстовий формат обміну даними між комп'ютерами (JavaScript Object Notation).
- JSX – розширення до синтаксису JavaScript (JavaScript Syntax Extension) XML для JavaScript.

- JWT – стандарт токена доступу на основі JSON (JSON Web Token).
- MIME – код, який визначає формат файлу або тип контенту, що передається мережею Інтернет.
- MVC – архітектурний шаблон модель–вигляд–контролер (Model-view-contro ller).
- ODM – об'єктно-документна проекція (Object-document maping), технологія програмування для документо-орієнтованих баз даних.
- OOUI – тип користувацького інтерфейсу, що ґрунтується на парадигмі об’єктно-орієнтованого програмування (Object-oriented user interface).
- ORM – об'єктно-реляційна проекція (Object-relational mapping), технологія програмування для реляційних баз даних.
- PHP – скрипкова мова програмування (Hypertext Preprocessor).
- REST – підхід до архітектури мережевих протоколів, які надають доступ до інформаційних ресурсів (Representational State Transfer).
- RoR – об'єктно-орієнтований програмний фреймворк Ruby on Rails.
- UI – інтерфейс користувача (User interface).
- URI – уніфікований ідентифікатор ресурсів (Uniform Resource Identifier).
- WUI – веб-інтерфейси користувача (Web user interface).
- XML – розширювана мова розмітки (Extensible Markup Language).
- БД – база даних.
- ООП – об’єктно-орієнтоване програмування.
- СКБД – система керування базами даних.
- Футер – нижня частина сторінки.
- Хеш – структура даних.
- Хук – функція, що дозволяє використовувати стан та інші можливості React без написання класу.

ВСТУП

Актуальність теми: люди та компанії створюють сайти для різних цілей: щоб продавати товари та послуги, розміщувати й знаходити інформацію, здобувати знання, спілкуватися з іншими користувачами, розважатися тощо.

Насправді надання інформації або послуг – це засіб залучення відвідувачів до даного ресурсу для здобуття, наприклад, статистичної інформації або ж для показу реклами, якщо це рекламний майданчик.

Дані переваги можна використовувати не тільки для вигоди власників вебресурсів, але й для відвідувачів. В час пандемії люди в спробі захистити своє здоров'я, обмежують себе в спілкуванні вживу. Найбільше страждають незахищені категорії населення, люди з інвалідністю та пенсіонери, часто вони також не мають доступу до інтернету.

Також при розробці сайту важливо обрати сучасні технології, які забезпечуть: швидкодію, якісне розширення функціоналу та довготривалу підтримку. При обранні технологій слід враховувати, що потрібні два види технологій, як для інтерфейсу так і для серверу. Код інтерфейсу описує дизайн. За його допомогою комп'ютери розуміють зовнішній вигляд і розташування елементів на сайті. Щоб написати код інтерфейсу, програмісти використовують мови HTML, CSS і Javascript. Серверний код описує поведінку сайту. За його допомогою сервер визначає, що відбудеться, якщо користувач натисне кнопку чи введе текст. Такого коду не може бути в односторінкових сайтах. Серверний код потрібен, якщо на сайті є можливість створити акаунт, придбати товар чи послугу. Серверний код пишуть на PHP, Java, Ruby тощо.

Розроблений інформаційні вебсайт належить до так званого типу

"альтруїст". Він повинен надавати безкоштовні сервіси, але його теж потрібно обслуговувати, розвивати, а отже, вкладати кошти. Але проекти, які не приносять прибуток, довго не живуть, тому для таких вебсайтів характерне заробляння грошей або на рекламі, підтримці користувачів, або на зборі статистичних даних. На таких сайтах дуже часто пропонують зареєструватися, аби отримати маленький додатковий сервіс.

Зовнішній вигляд сайту є надзвичайно важливим, оскільки, насамперед, сторінки сприймаються візуально. Великі фрагменти тексту виглядають нудно, їх важко читати, вони вимагають від користувача більшого розумового напруження, ніж альтернативні методи подання інформації. Також важливо вміти правильно структурувати дані на сторінці, щоб важлива інформація була першочерговою.

Метою і завданням роботи є створити належні умови і дати можливість користувачам виконувати будь які операції вирішувати різні питання і робити покупки. Правильно зроблений вебсайт із легкістю приведе клієнта до висновку про необхідність покупки товарів, або послуг, або ідей, що пропагуються на ньому. У сучасної людини немає багато часу для ходіння по магазинах. Тому можливість замовлення товарів і послуг, не відходячи від комп'ютера, значно розширює можливості і клієнта, і продавця.

Тому для інформування про поточну ситуацію під час пандемії та методів захисту від хвороби створено багато вебсайтів. Також в різних країнах створюють ресурси на яких здійснюється координація волонтерів, для допомоги не захищеним верствам населення. Саме через це є гостра потреба в створенні такого ж сайту в Україні.

Об’єктом роботи є розроблений вебсайт який є ремейком на вже існуючий на даний момент «volunteer.country».

Предмет роботи є засади функцінування, структура та зміст вебсайту волантерів як документально – інформаційний.

Кваліфікайна робота складається з переліку умовних скорочень, вступу, тьох розділів, висновків, списку використаних джерел. Перелік умовних скорочень включає в себе 38 позицій, дослідження міститься на 74 сторінках, список використаних джерел нараховує 25 позиції.

Загальний обсяг роботи – 84 сторінки.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ТЕХНОЛОГІЙ ВЕБРОЗРОБКИ

1.1 Огляд основних інструментів та технологій веб-розробки

Існує безліч програмного забезпечення, яке дозволяє створити сайт, різноманіття CMS (система керування вмістом) вражає. Але найбільш гнучким засобом залишаються технології на яких базуються CMS і вся веб розробка в цілому, HTML що дозволяє реалізувати розмітку елементів сторінок та мови опису зовнішнього вигляду та стилю сторінок CSS. Також є інші веб-орієнтовані мови програмування та розмітки, але вони розроблені на вищезгаданих. У випадку розробки даної платформи найкращим вибором є саме використання HTML та CSS і більш зручних засобів на їх основі, а не CMS [1].

Для опису використовується спеціалізована мова HTML мова розмітки гіпертексту. Готові зверстані шаблони використовуються в наступних етапах реалізації проекту. На стадії верстки розмітка ділиться на окремі елементи і з використанням технологій HTML і CSS та трансформується в код, який можна переглядати в браузерах.

Таким чином, створений HTML – документ передається програмі (в основному HTML-документи використовуються для створення сайтів, отже, зазвичай, програма для читання HTML це браузер), що обробляє його згідно правил, що в ній запрограмовані. Отже, коли браузер читає HTML – документ, то він зображає на екрані його вміст та структуру, керуючись принципами, що в нього закладені та розмічені мовою HTML [2].

Теги це елементи які є базовими компоненти розмітки HTML. Кожен тег має дві основні властивості: атрибути та контент. Існують визначені правила щодо кожного атрибута та контенту тега, які потрібно виконувати для того, щоб HTML – документ був визнаний валідним та міг бути зчитаним веб-переглядачем.

У тега є початковий елемент, який має вигляд <tag-name>, та кінцевий елемент, який має вигляд </tag-name>. Атрибути тегів вказуються в початковому елементі одразу після назви елемента, контент тега записується між його двома елементами.

Блочна або каскадна верстка веб-сторінок прийшла на заміну табличній. Головна перевага блочної верстки це розділення контенту та вмісту сторінки та їхньої візуальної презентації.

CSS використовується розробникми вебсторінок, щоб стилізувати шрифти, кольори, зображення, розміщення та інші аспекти вигляду сторінки. Одною з головних переваг є можливість розділити контент сторінки (або контент, наповнення HTML, або іншу розмітку) від вигляду документу (що описана в CSS) [3].

Таке розділення може покращити сприйняття та доступність контенту, забезпечити більшу гнучкість та контроль за відображенням контенту в різних умовах, зробити контент більш структурованим та простим, прибрати повтори тощо. CSS також дозволяє адаптувати контент до різних умов відображення (на екрані монітора, мобільного пристрою (КПК), у роздрукованому вигляді, на

екрані телевізора, пристроях з підтримкою шрифту Брайля або голосових браузерах та ін.) [4].

Мова JavaScript містить визначену бібліотеку об'єктів та структур даних (таких як Array, Date та Map) і основний набір елементів мови програмування, таких як умовні та інші оператори, цикли та вирази.

Ядро JavaScript може бути розширене для різних потреб шляхом доповнення його додатковими об'єктами, все завдяки тому, що зазвичай ця мова не має строгої типізації.

На стороні клієнта JavaScript розширює ядро , додаючи об'єкти для керування переглядачем і його DOM – об'єктною моделлю документа [6]. Наприклад, клієнтські розширення дозволяють застосункам розміщувати елементи на HTML та реагувати на дії користувача, такі як клацання миші, введення даних у форму і пересування сторінками та інші.

На стороні сервера JavaScript розширює ядро мови шляхом додавання об'єктів, що стосуються роботи JavaScript на сервері, це специфічні об’єкти. Наприклад, серверні розширення дозволяють застосункам взаємодіяти з базами даних, забезпечувати безперервне виконання потоку інформації від одного запущеного застосунку до іншого, або виконувати маніпуляції з файлами на сервері та багато іншого.

JavaScript стандартизований асоціаіцєю з стандартизації інформаційних та комунікаційних систем – ECMA. Ця стандартизована версія JavaScript, що має назву ECMAScript, однаково працює у всіх застосунках (браурзеах), що підтримують стандарт. Різні компанії можуть використовувати цей відкритий стандарт для розробки своїх реалізацій JavaScript [7].

Для спрощення роботи з рівнем представлення, використовують готові бібліотеки. Одна з найсучасніших та найкращих бібліотек на даний момент є React. Розробляється Facebook, Instagram і спільнотою індивідуальних розробників [8].

React дозволяє розробникам створювати великі вебзастосунки, які використовують дані, котрі в свою чергу змінюються з часом, без перезавантаження сторінки. Його мета полягає в тому, щоб бути швидким, простим, масштабованим. React обробляє тільки користувацький інтерфейс у застосунках. Це відповідає видові у шаблоні модель–вид–контролер (MVC), і може бути використане у поєднанні з іншими JavaScript бібліотеками або в великих фреймворках MVC, таких як Angular. Він також може бути використаний з React на основі надбудов, щоб піклуватися про частини без користувацького інтерфейсу побудови веб-застосунків. Як бібліотеку інтерфейсу користувача React найчастіше використовують разом з іншими подібними бібліотеками, такими як Redux.

React не намагається надати повну "схему додатків". Він безпосередньо спрямований на побудову користувацьких інтерфейсів, і тому не включає в себе безліч інструментів, які деякі розробники вважають необхідними для створення програми. Це дозволяє вибрати будь-які бібліотеки, які розробник вважає за краще виконувати, щоб виконати певних завдань, таких як здійснення доступу до мережі або локальне зберігання даних.

Компоненти React зазвичай написані на JSX. Код написаний на JSX компілюється у виклики методів бібліотеки React. Розробники можуть так само писати на чистому JavaScript. JSX нагадує іншу мову, яку створили у компанії Фейсбук для розширення PHP [8].

Найбільш розповсюдженим з архітектурних шаблонів програмного забезпечення є архітектура клієнт-сервер (рис. 1.1). Клієнт–сервер є домінуючою концепцією у створенні мережних розподілених програм і передбачає взаємодію та обмін даними між ними.

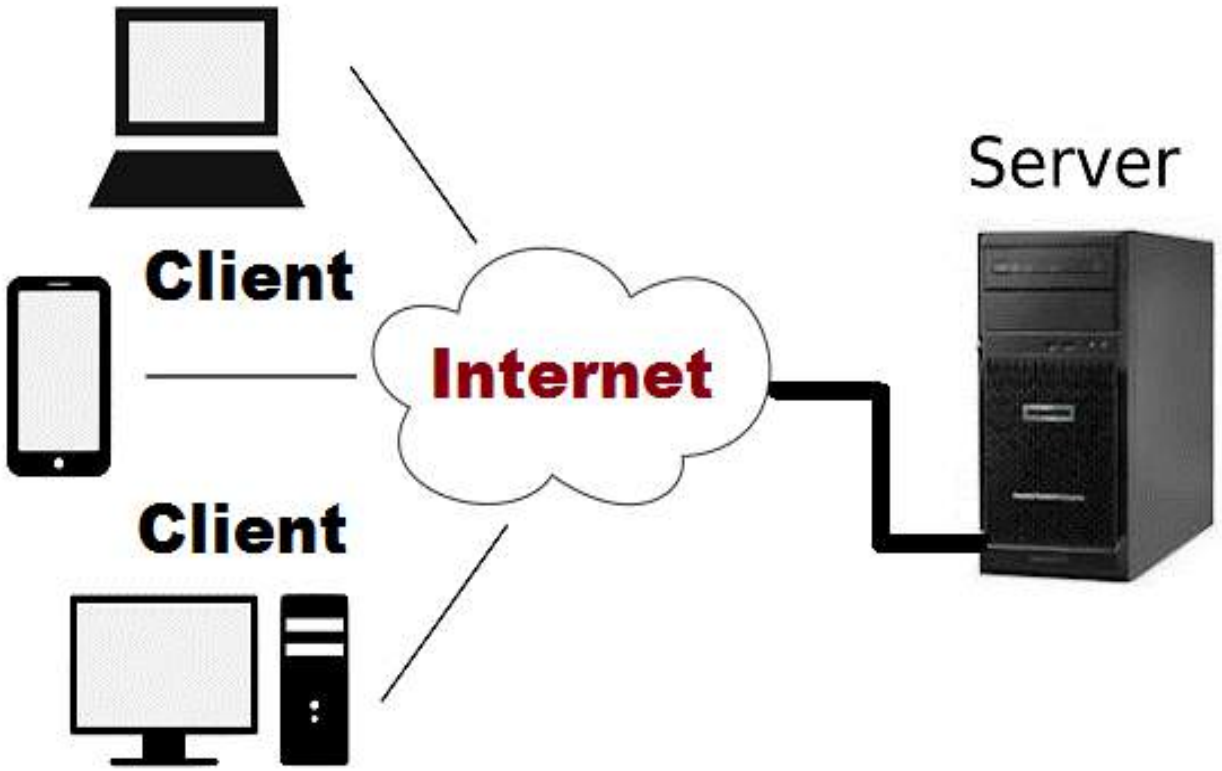

Рисунок 1.1 – Архітектура клієнт-сервер

Основні компоненти:

- набір серверів, що власне надають інформацію або будь-які інші необхідні послуги з обробки даних програмам, що звертаються до них;
- набір клієнтів, які використовують розроблені сервіси, які надаються серверами;
- мережа, яка забезпечує взаємодію між клієнтами та серверами, передачу даних та команд.

Сервери повинні бути незалежними один від одного. Клієнти також функціонують паралельно і незалежно один від одного. Немає прив'язки клієнтів до серверів. Дуже типовою є ситуація, коли один сервер одночасно обробляє запити від багатьох різних клієнтів, з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про

доступні сервери, але не повинні мати жодного уявлення про існування інших клієнтів [9].

Найчастіше вебсервер і серверні модулі проміжного рівня розміщуються на одному комп'ютері, хоча і являють собою окремі, але логічно незалежні програмні модулі.

Схожою з клієнт-серверною трьохрівневою архітектурою є шаблон програмування MVC (рис. 1.2).

Це архітектурний шаблон, що використовується під час проектування та розробки програмного забезпечення. Цей шаблон передбачає розподіл системи на три взаємопов'язані частини: модель даних, інтерфейс користувача (вид, вигляд) та модуль керування. Застосовується для відокремлення даних (моделі) від інтерфейсу користувача (виду) так, щоб зміни вигляду мінімально впливали на роботу з даними, а зміни в моделі даних могли здійснюватися без змін в інтерфейсі користувача [10].

Мета шаблону забезпечити гнучкий дизайн програмного забезпечення, який повинен полегшувати подальші розширення чи зміни програм, а також надавати можливість повторного використання окремих компонентів програми. Дуже типовою є подібна ситуація, коли один сервер одночасно обробляє запити від багатьох різних клієнтів; з іншого боку, клієнт може звертатися то до одного сервера, то до іншого. Клієнти мають знати про доступні сервери, але не повинні мати жодного уявлення про існування інших клієнтів. Крім того, використання цього шаблону у великих системах сприяє впорядкованості їхньої структури і робить їх більш менше зрозумілими та за рахунок зменшення їх складності [11].

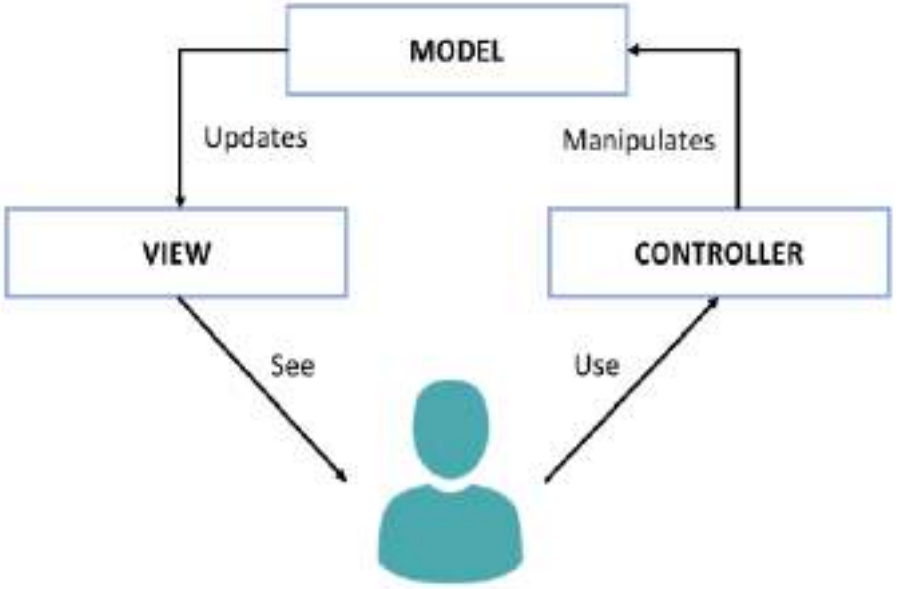

Рисунок 1.2 – Схема роботи MVC

Так як платформа має бути розроблена просто та з великою швидкодією, для цього пригодиться мова програмування Ruby.

Мова вирізняється високою ефективністю розробки програм і поєднала у собі найкращі риси інших мов, таких як: Java, Perl, Python, Eiffel, Smalltalk, Ada і Lisp. Ruby поєднує в собі Perl–подібний синтаксис із об'єктно–орієнтованим підходом мови програмування Smalltalk. Також деякі риси запозичено із мов програмування нижчого рівня Lisp, Python, Dylan та CLU.Ruby об’єктивно –орієнтована, інтерпретована мова програмування. Багатоплатформна реалізація інтерпретатора Ruby поширюється як вільне програмне забезпечення для усіх операційних систем[12].

Для розробки додатку буде використано найпопулярніший об'єктно-орієнтований фреймворк (програмний каркас) для створення веб-додатків, написаний на мові програмування.

Ruby on Rails повністю задовольняє усі вимоги, як поставлені у задачі, так і які диктуються світовими стандартами розробки програмного забезпечення, а також підтримує архітектуру клієнт – сервер та MVC[13] , це нам і потрібно.

Дані, що будуть опрацьовуватись сайтом необхідно організовувати. Сукупність даних, організованих відповідно до певної концепції, яка описує характеристики цих даних і зв'язки між їх елементами, називається базою даних. В загальному випадку базою даних можна вважати будь-який впорядкований набір даних.

Для розробки платформи підійде база даних об'єктно-орієнтована типу, тому що буде використано багато JavaScript та передача даних відбуватиметься у вигляді JSON. А також об'єктно-орієнтовані бази даних зазвичай рекомендовані для тих випадків, коли потрібна високопродуктивна обробка даних, що мають складну структуру. Об'єктно-орієнтованою базою даних, яка містить інтерпретатори для роботи з Ruby on Rails є MongoDB.

Отже для розробки обраний наступний стек технологій:

- JavaScript;
- React;
- JSX;
- Ruby;
- Ruby on Rails;
- MongoDB.

1.2 Аналіз вимог та предметної області

Для того, щоб чітко розуміти, що потрібно реалізувати, потрібно оглянути існуючі рішення. На даний час існує як мінімум один сайт для загальної координації волонтерів в Україні. Для огляду було обрано сайт Українська Волонтерська (Волонтерський центр допомоги – некомерційний проєкт, який об'єднує тих, хто готовий допомогти, з людьми та організаціями, яким потрібна допомога волонтерів) Служба volunteer.country, тому що аудиторія сайту та функціонал дуже схожий з сайтом, що буде розроблятись у цій роботі.

Сайт є багатосторінковим, але не усі сторінки функціонально пов'язані з сайтом, що буде розроблятись. Відповідно огляд сайту буде здійснюватись посторінково, а також сторінки, яким не вистачає одного екрану для відображення будуть відображатись поблоково задля вдалого інтерфейсу і зручності користувача.

Головна сторінка (рис. 1.3) складається з наступних блоків :

- Меню (вверху сторінки).
- Блок заголовку (заголовок, опис та кнопки).
- Інформаційний блок (інформація про волонтерство).
- Блок з списком проектів.
- Досягнення.
- Партнери.

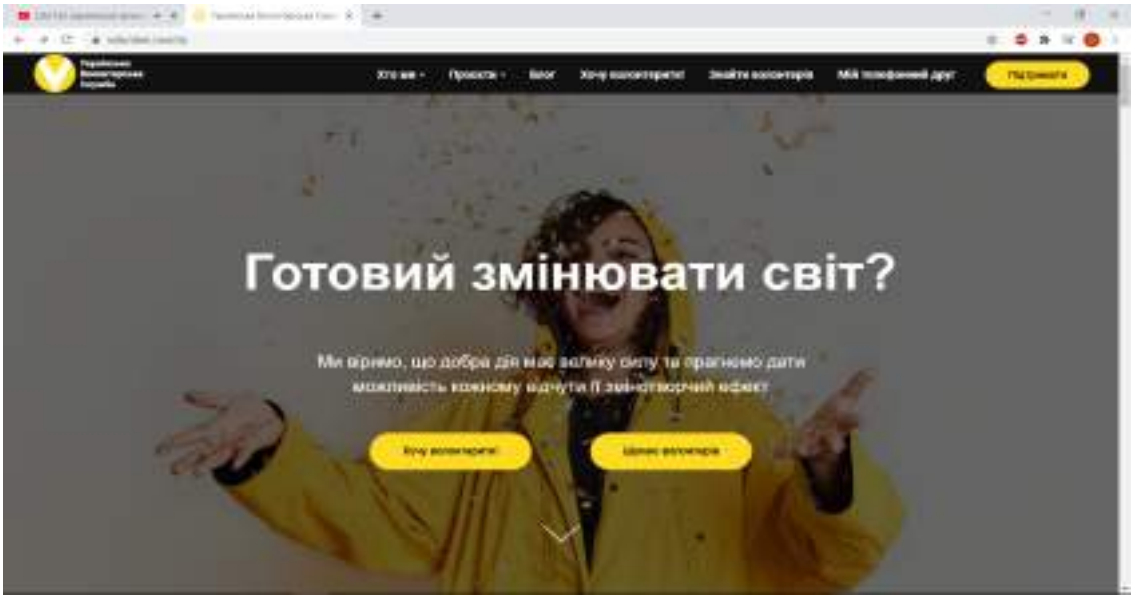

Рисунок 1.3 – Головна сторінка сайту волонтерів

На рисунку зображено меню з логотипом та усіма його пунктами. Також зображено заголовок та кнопки для реєстрації або пошуку волонтерів. Решту блоків з цієї сторінки не буде розглядатись, тому що вони не будуть потрібні при розробці сайту.

З переліку меню буде розглянуто пункти:

- Хочу волонтерити.
- Знайти волонтерів.
- Підтримати.

При переході на сторікну «Хочу волонтерити» (рис. 1.4) відображається заголовок сторінки, меню, так як воно розміщене на усіх сторінках, анкета волонтера, а також деякі блоки з головної сторінки, тому що вони також дублюються на усіх сторінках.

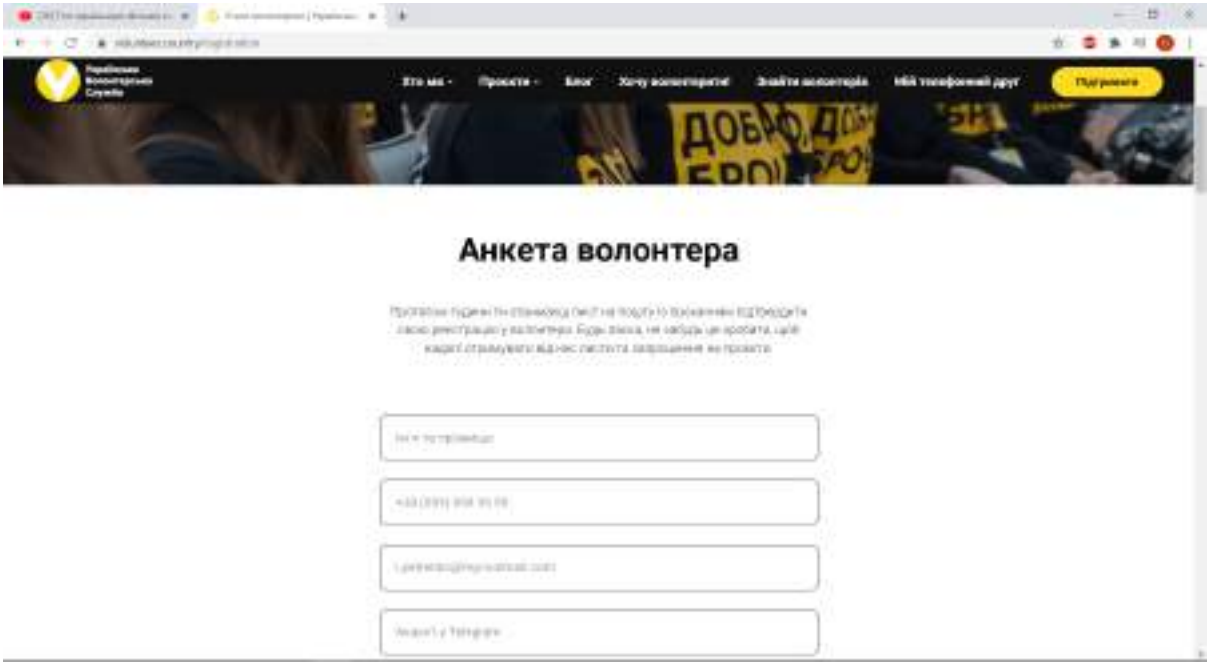

Рисунок 1.4 – Сторінка «Хочу волонтерити»

Найголовнішим елементом даної сторінки, є форма для реєстрації волонтерів. Вона містить короткий опис процедури реєстрації та наступний список полів:

- Ім’я та прізвище.
- Номер телефону.
- Електронна пошта.
- Акаунт в телеграм.
- Область проживання.

- Місто проживання.
- Дата народження.
- Опис навичок.
- Перелік тем волонтерства (можна обрати декілька).
- Перелік місць звідки волонтер дізнався про сайт.
- Згодна на обробку даних.

Деякі з полів форми є обов’язковими, тому при підтвердженні форми з пропущеними даними полями, відобразиться повідомлення про помилку (рис. 1.5). Але якщо усі поля заповнені то виведеться повідомлення про успішну реєстрацію.

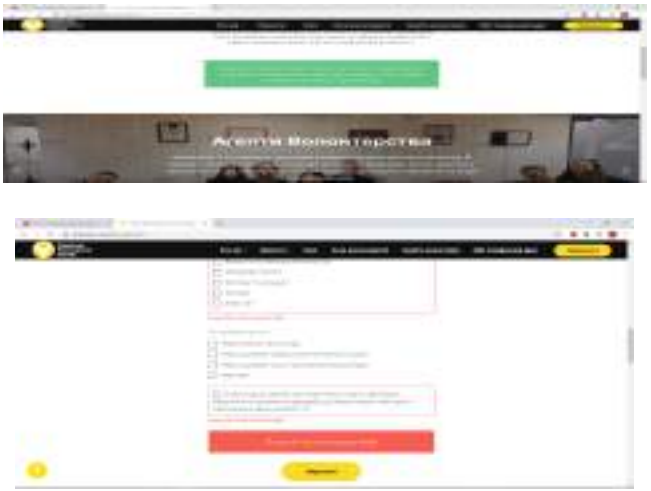

Рисунок 1.5 – Сповіщення форми реєстрації

Дата форма містить і недолік, при обранні дати народження, автоматично обирається поточна дата. Це є не логічним так, тому це потрібно врахувати при розробці сайту.

При переході на сторінку «Знайти волонтерів» відображається сторінка з стандартними для цього сайту блоками, що повторюються а також інформацією про пошук та власне формою пошуку (рис. 1.6).

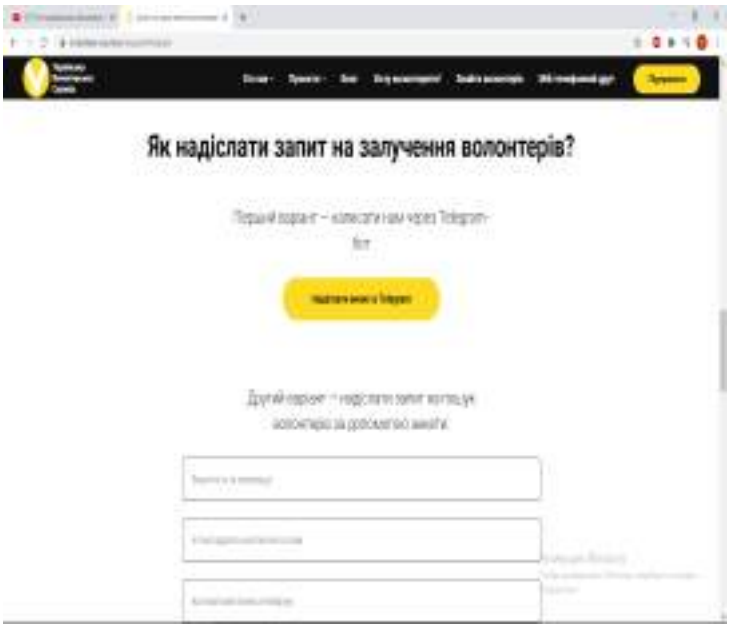

Рисунок 1.6 – Сторінка пошуку волонтерів

Вверху сторінки зображено заголовок, а також інформацію про методи здійснення пошуку волонтерів організації. Форма пошуку складається з наступних елементів:

- Ім’я та прізвище.
- Електронна пошта.
- Номер телефону.
- Назва організації.
- Адреса.
- Поле анонсу заходу з описом події на яку потрібні волонтери.
- Згода на обробку особистих даних.

Деякі поля форми також є обов’язковими, тому також сповіщається про помилку, якщо підтвердити форму з незаповненими полями. Вигляд повідомлень про помилку та успішний запит ідентичні з іншими формами на сайті.

При переході на сторінку «Підтримати» відображається сторінка з кількома загальними блоками. Інформаційним блоком, а також блоком самої ж

підтримки (рис. 1.7).

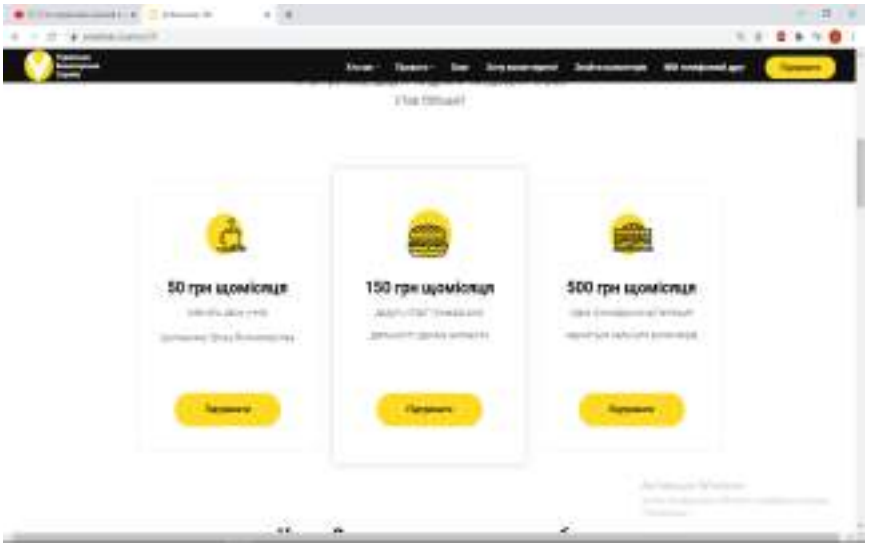

Рисунок 1.7 – Сторінка підтримки

Зображено три блоки з різними іконками, сумую та інформацією. Кожен з блоків описує щомісячну суму підтримки та містить кнопку «підтримати». При натисканні на кнопку відкривається сторінка платіжної системи Liqpay (рис. 1.8), через яку і оформляється підписка на підтримку.

Рисунок 1.8 – Сторінка платіжної системи

Загалом сайт реалізовано дуже детально та він є повністю функціональним. Але поточна структура сайту для подальшої розробки є надлишковою, тому що вебсайт буде більш вузько направлений. Для подальшої роботи обрано наступні елементи з оглянутого сайту:

- Багатосторінкова структура.
- Форма реєстрації волонтерів.
- Інформаційні блоки.
- Форма пошуку волонтера або запиту на волонтера.
- Сторінка підтримки та оформлення підписки.
- Загальна побудова сторінок.

1.3 Постановка задачі

Метою розробки є створення сайту, який буде забезпечувати координацію роботи волонтерів у допомозі людям похилого віку під час пандемії COVID-19. Також потрібно реалізувати адмінпанель для наповнення контенту та обробки заявок.

Сайт повинен бути розробленим за допомогою стандартних методів та технологій веб-розробки. Основними етапами розробки повинні бути:

- Верстка сторінок.
- Програмування веб на стороні клієнта.
- Програмування серверної частини.
- Розробка баз даних.

Забезпечити швидке завантаження сторінок сайту при великій кількості запитів, для максимальної взаємодії з користувачем.

Інтерфейс повинен бути максимально простим, та не має містити багато елементів.

Для роботи ресурсу не потрібні додаткові встановлення програмного

забезпечення, або конфігурації браузера. Сайт повинен бути максимально простий у користування [15].

Сайт повинен містити наступні блоки та сторінки:

- Головна інформаційна сторінка.
- Меню.
- Сторінка реєстрації волонтерів.
- Сторінка запиту на волонерів.
- Сторінка з завданнями для зареєстрованих волонтерів.
- Форми реєстрації, авторизації, надсилання запиту та завантаження звітів.
- Сторінка грошової підтримки.
- Адмінпанель з відповідними сторінками та блоками, які дозволятимуть координувати дії волонтерів.
- Форма зворотнього зв’язку та відгуків.

Висновок до розділу 1

Отже в розділі ми провели аналіз предметної області та технологій веброзробки. Також розглянули та охарактеризували основні інструменти та технології веброзробки. Зробили аналіз вимог та предметної області, Розкрили постановку задачі, мету розробки.

РОЗДІЛ 2. СТРУКТУРА ПЛАТФОРМИ ТА БАЗА ДАНИХ

2.1 Розробка бази даних

Головні ідеї сучасних інформаційних технологій базуються на концепції баз даних. Відповідно до цієї концепції, основою інформаційних технологій є дані, які повинні бути організовані в бази даних в цілях адекватного відображення мінливого реального світу і задоволення інформаційних потреб користувачів.

Одним з найважливіших і основоположних понять в теорії баз даних є поняття інформації. Під інформацією розуміються будь – які відомості про будь – яку подію, процеси, об'єкти.

Для коректного визначення вимог, необхідних для проектування бази даних відряджень працівників, визначимо такі поняття.

У базі даних інформація повинна бути організована так, щоб забезпечити мінімальну частку її надмірності, оскільки надмірна надмірність даних тягне за собою ряд негативних наслідків: збільшення обсягу інформації, а отже, потреба в додаткових ресурсах для зберігання і обробки, а так само поява помилок при введенні дублюючої інформації, що порушують цілісність бази даних.

В БД повинні зберігатися дані, логічно пов'язані між собою. Для того щоб дані можна було пов'язати між собою, і зв'язати так, щоб ці зв'язки відповідали реально існуючим в даній предметній області, останню піддають
детальному аналізу, виділяючи сутності чи об'єкти. Сутності мають деякі характеристики, звані атрибутами, які теж необхідно зберігати в БД [16].

Визначивши сутності і їхні атрибути, необхідно перейти до виявлення зв'язків, які можуть існувати між деякими сутностями. Зв'язки між сутностями також є частиною даних, і вони також повинні зберігатися в базі даних.

Проектована база даних повинна володіти певними властивостями.

- Цілісність бази даних досягається внаслідок введення обмежень, пов'язані з нормалізацією БД. У кожен момент часу існування БД відомості, що містяться в ній, повинні бути несуперечливі.
- Відновлювання – можливість відновлення БД після збою системи або окремих видів поломок системи.
- Безпека – припускає захист даних від навмисного і ненавмисного доступу, модифікації або руйнування.
- Ефективність, зазвичай оцінюється двома параметрами: мінімальний час реакції на запит користувача і мінімальні потреби в пам'яті, а так же поєднанням цих параметрів.

На першому етапі проектування необхідно визначити сутності. З точки зору заданої предметної області типовим користувачем розроблюваної бази даних будуть волнтери та люди, які потребують допомоги волонтерів.

Основне завдання бази даних саме забезпечити всією необхідною інформацією даних користувачів.

Для досягнення мети і вирішення поставленого завдання виділимо на основі аналізу предметної області наступні сутності: завдання (Task), користувач (User), відгук (Callback).

Перед етапом визначення зв’язків варто також детально описати основні поля, якими повинна володіти кожна сутність, а також типи цих полів. Далі визначимо атрибути або властивості для кожної з сутностей, а так само ключі для кожної сутності. Ключем сутності називається атрибут або набір атрибутів, що дозволяють однозначно ідентифікувати примірник сутності:

Сутність Task містить:

- Ідентифікатор (id), зовнішній ключ.
- Заголовок (title), текстове поле.
- Опис (description), текстове поле.

- Адреса (address), текстове поле.
- Номер телефону (phone), текстове поле.
- Коментар (comment), текстове поле.
- Оцінка (mark), числове поле.
- Тип (type), текстове поле.
- Статус (status), текстове поле.
- Дата створення (created_at), поле типу дата.
- Дата початку (start_date), поле типу дата.
- Дата завершення (end_date), поле типу дата.
- Опубліковано (published), поле булевого типу.

Сутність User містить:

- Ідентифікатор (id) – зовнішній ключ.
- Електронна пошта (email), текстове поле.
- Опис (description), текстове поле.
- Роль (role), текстове поле.
- Пароль (password_digest), текстове поле в зашифрованому вигляді.
- Ім’я (first_name), текстове поле.
- Прізвище (last_name), текстове поле.
- Активований (activated), поле булевого типу.
- Дата створення (created_at), поле типу дата.
- Посилання на зображення користувача (avatar), текстове поле.

Сутність Callback містить:

- Ідентифікатор (id) – зовнішній ключ.
- Електронна пошта (email), текстове поле.
- Тема (topic), текстове поле.
- Дата створення (created_at), поле типу дата.
- Повідомлення (message), текстове поле.

Другий етап – визначення зв'язків між виділеними сутностями. Завдання

повинно містити інформацію про те, хто його створив та хто почав його виконувати, тому важливо імплементувати зв'язок між сутностями User та Task. Для цього в сутність Task додано два поля:

- Поле автора (author) посилання на ідентифікатор користувача (User) – символьного типу.
- Поле волонтера (volonteer) посилання на ідентифікатор користувача (User) – символьного типу.

П'ятий і шостий етапи проектування бази даних за методом сутність – зв'язок вимагають визначити ступінь зв'язку і клас приналежності. Ступінь зв'язку показує скільки безпосередніх зв'язків може мати кожен екземпляр сутності з іншим екземпляром сутності, з яким він пов'язаний.

Розроблену схему бази даних зображено на (рис. 2.1)

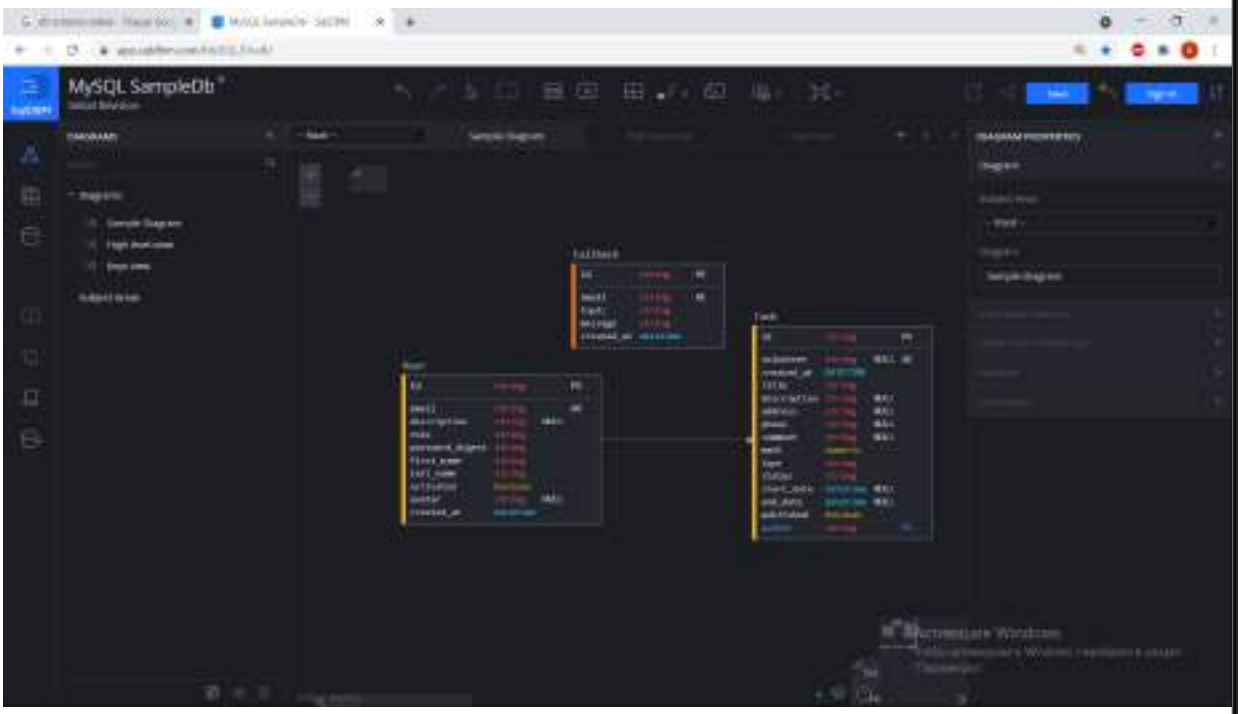

Рисунок 2.1 – Загальна схема бази даних

Наступним типом зв'язку який зустрічається в базі є зв'язок багато до багатьох, який відображає багатозначну залежність об'єктів один від одного.

Такий зв'язок досить складно обробляти, оскільки виникає безліч аномалій обробки. Наприклад, при видаленні одного примірника будь-якого з двох об'єктів, що беруть участь у встановленні зв'язку, можливо каскадне видалення всіх пов'язаних об'єктів іншого об'єкта. Але наявність такого ж правила в зворотну сторону призведе до циклічного видалення даних, що може привести до їх втрати. Саме тому в базах даних реалізуються захисні механізми обмеження посилальної цілісності, але також на рівні реалізації зв'язків між об'єктами варто обмежити можливість застосування зв'язку багато до багатьох.

Клас приналежності може бути обов'язковим і необов'язковим, якщо екземпляри даної суті мають брати участь у зв'язку, то участь називається обов'язковою, а якщо екземпляри можуть не брати участь у зв'язку, то необов'язковою.

Так як використовується фреймворк Ruby on Rails та база даних MongoDB, для створення таблиць можна використати ORM гем mongoid.

Завдяки йому, шляхом простого опису моделей даних будуть створені таблиці бази даних. Таблиці складають основу бази даних – саме в них зберігаються всі дані. Перед усім, повинна бути спланована структура кожної таблиці. Структура таблиць обумовлюється вмістом тих вихідних форм, запитів та звітів, які повинні бути отримані при роботі з базою даних.

При плануванні таблиць необхідно уникати повторення колонок в різних таблицях, тільки якщо вони не слугують для визначення зв'язків між ними.

Таблиця складається з записів (рядків), кожний з яких описує одну сутність. Кожна колонка таблиці – це поле. Поле містить однотипну інформацію, яка визначає тип даних.

Тип даних визначає вид і межі допустимих значень, які можуть бути введені в поле, а також об'єм пам'яті, який виділяється для цього поля, що важливо при проектуванні великих БД.

Створення таблиці та моделі даних User з відповідними полями сутності

(короткий опис моделі, без валідацій та методів, повний опис моделі реалізовано в додатку):

```
class User
  include Mongoid::Document
  include Mongoid::Timestamps

  has_many :created_tasks, inverse_of: :author, class_name: 'Task'

  field :email, type: String
  field :description, type: String
  field :role, type: String, default: CLIENT
  field :password_digest, type: String

  field :first_name, type: String
  field :last_name, type: String
  field :activated, type: Mongoid::Boolean, default: false
  mount_uploader :avatar, UserAvatarUploader
end
```

Створення таблиці та моделі даних Task з відповідними полями сутності (короткий опис моделі, без валідацій та методів, повний опис моделі реалізовано в додатку):

```
class Task
  include Mongoid::Document
  include Mongoid::Timestamps

  belongs_to :author, inverse_of: :created_tasks, class_name: 'User'
  field :volunteer, type: User, default: nil

  field :title, type: String
  field :description, type: String

  field :address, type: String
  field :phone, type: String
  field :comment, type: String
  field :mark, type: Integer, default: 0

  field :type, type: String, default: ANOTHER
  field :status, type: String, default: TO_DO

  field :start_date, type: DateTime
  field :end_date, type: DateTime
  field :published, type: Mongoid::Boolean, default: false
end
```

Створення таблиці та моделі даних Callback з відповідними полями сутності

```
class Callback
  include Mongoid::Document
  include Mongoid::Timestamps

  field :email, type: String
  field :topic, type: String
  field :message, type: String
end
```

Зображений вище код, описує процес створення даних для відповідних сутностей та моделей. Імпортований функціонал бібліотеки mongoid додає функціонал для створення таблиць у базі даних та поле часу створення об’єкту. Також описано типи та значення по замовчуванню, для створення об’єктів.

2.2 Розробка інтерфейсу платформи

В випадку з розробкою вебсатйів інтерфейсом є сукупність засобів, за допомогою яких користувач взаємодіє з сайтом або вебзастосунком через браузер. Однією з базових вимог до розробки вебінтерфейсів є однаковий зовнішній вигляд елементів і однакова функціональність при роботі у різних браузерах [17].

Загалом виділяють такі основні види інтерфейсів:

- Інтерфейс прямої обробки. Це назва загального класу інтерфейсів користувача, що дозволяє користувачам керувати наданими їм об'єктами, з використанням дій, котрі мають відповідати фізичному світу.
- GUI. Графічні інтерфейси користувача, приймають вхідні дані за допомогою периферійних пристроїв, таких як комп'ютерна клавіатура та миша і інші. Забезпечують графічний висновок на моніторі комп'ютера. У GUI – дизайні, широко використовуються як мінімум, два різні принципи: інтерфейси орієнтовані на додатки та

об'єктно-орієнтовані інтерфейси користувача (OOUI).

- Вебінтерфейси користувача або вебінтерфейси (WUI). Це інтерфейси користувача, які приймають вхідні дані та забезпечують виведення, створенням вебсторінок та елементів на сторінках, які передаються засобами Інтернет і переглядаються користувачем за допомогою програми веббраузера. Адміністративні вебінтерфейси для веб-серверів, серверів і мережевих комп'ютерів, часто називаються панелями керування [18].
- Сенсорні екрани. Дисплеї, які приймають введення, дотиком пальців або спеціальних предметів. Використовуються у все великій кількості мобільних пристроїв і багатьох промислових процесах, засобах точок продажу, і машинах, пристроях самообслуговування тощо.
- Апаратні інтерфейси. Фізичні інтерфейси, що присутні на виробах у повсякденному житті від мікрохвильовки, до приладових панелей автомобілів чи кабін літаків. Як правило, є поєднанням кнопко, ричагів, слайдерів, перемикачів і сенсорних екранів.
- Інтерфейси командного рядка, це інтерфейси де користувач надає дані на вхідний сигнал, введенням командного рядка за допомогою комп'ютерної клавіатури, а система забезпечує обробку та виведення, шляхом відбиття тексту на моніторі комп'ютера чи приладу.

Для того, щоб мати змогу створити правильний інтерфейс, потрібно наперед визначити схему розміщення елементів на ньому та схему сторінок загалом [19]. Сайт складається з інформативних сторінок та сторінок на яких користувач зможе здійснювати певні дії, наприклад зареєструватись як волонтер. Сайт складатиметься з наступних сторінок:

- Головна (коротка інформація про сайт).
- Про нас (детальна інформація про цілі та діяльність).
- Знайти волонтера (сторінка пошуку волонтера через створення

завдань).

- Завдання (сторінка зі списком опублікованих авдань).
- Підтримати (сторінка з описом можливостей підтримки сайту).
- Зворотній зв'язок (сторінка на котрій користувач може залишити відгук чи побажання).
- Увійти (сторінка на якій здійснюється авторизація).
- Реєстрація (сторінкан на котрій користувач може зареєструватись).
- Адміністрування користувачів (сторінка на котрій можна здійснювати маніпуляції над акаунтами користувачів).
- Адміністрування завдань (сторінка на котрій можна здійснювати маніпуляції над завданнями).
- Адміністрування зворотнійх зв’язків (сторінка на котрій можна переглядати повідомлення залишені користувачами).

Велика кількість функцій та сторінок потребує розробки зрозумілого інтерфейсу, для цього варто скористатись одним з основних принципів розробки інтерфейсів, а саме розробити усі сторінки та елементи за однаковим загальним шаблоном (рис. 2.2).

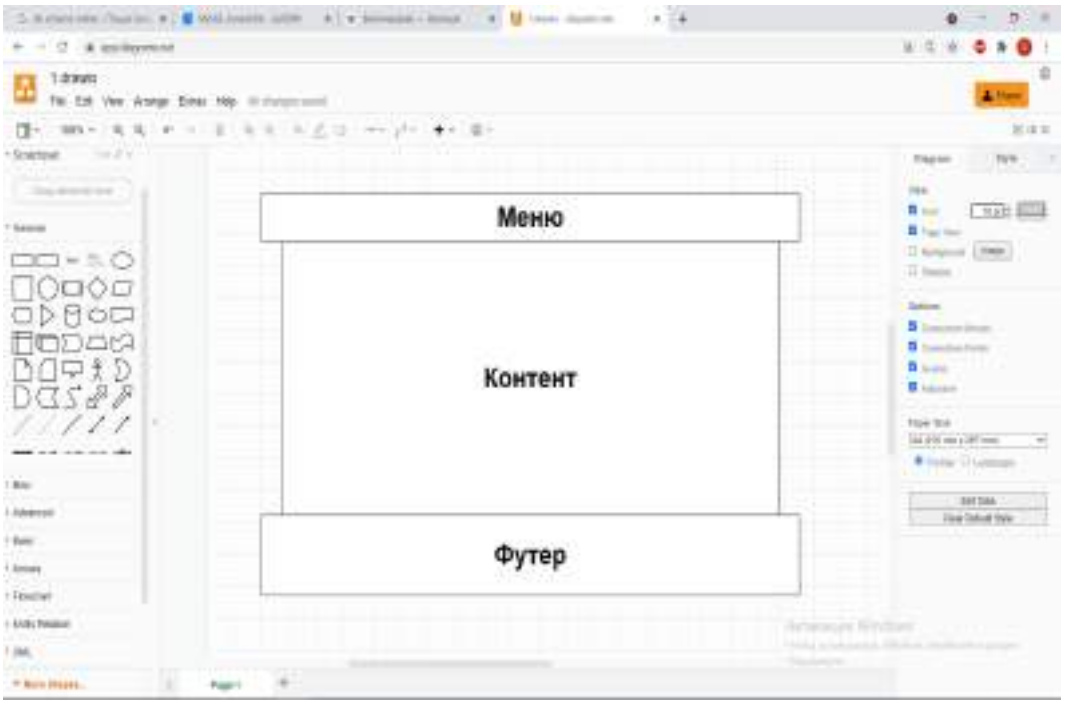

Рисунок 2.2 – Шаблон структури сторінок

На зображеному шаблоні використано розповсюджений підхід у створенні веб інтерфейсів з використанням наступних елементів:

- Меню (хедер), розміщується вверху сторінки.
- Контент (увесь інший, змінний вміст сторінки), займає найбільше місця.
- Футер (в ньому розміщуються навігація та додаткова інформація), розміщується внизу сторінки.

Контент сторінки буде змінюватись відповідно до призначення сторінки, футер завжди залишатиметься незмінним. А от меню для різних користувачів буде різним, це залежить від того чи авторизований користувач чи ні і яка у нього роль. Меню неавторизованого користувача зображено на (рис. 2.3)

Рисунок 2.3 – Загальне меню

На рисунку видно, що перераховані не усі сторінки сайту, це тому що для користувача, який адмініструє систему будуть видимі сторінки адміністрації, відповідно для інших користувачів вони недоступні. Також при вході в систему посилання на сторінки входу та реєстрації зміняться на кнопку виходу з акаунту. Загальна навігація також дублюється у футері (рис. 2.4).

Рисунок 2.4 – Футер сайту

Футер глобально складається з трьох блоків:

- Навігація.
- Контакти.
- Інформація.

В блоці навігації розміщується перелік загальнодоступних сторінок сайту, окрім головної. Блок контактів містить посилання на соціальні мережі. Інформація про власників вебсайту та розробників міститься в третьому блоці роботи.

Контент сторінки також буде перевикористовувати багато загальних однакових елементів та методів їх розміщення на сторінці. Через це буде розглянуто основні види сторінок, які можуть бути перевикористані на інших але зі зміненим вмістом. Головна сторінка зображена на (рис. 2.5)

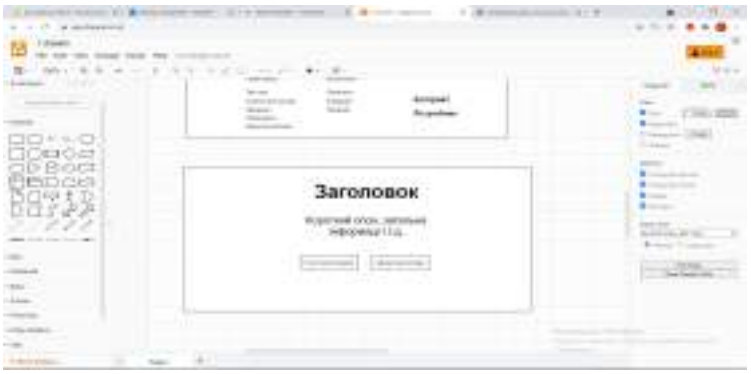

Рисунок 2.5 – Головна сторінка

На сторінці, як і на багатьох інших, міститься заголовок та короткий підзаголовк. Також розміщено дві кнопки, які є посиланнями на сторінки зі створенням завдань та реєстрацією.

Щоб детальніше ознайомитись з сайтом, користувачу потрібна сторінка де можна прочитати усю необхідну інформацію, такою є сторінка «Про нас» (рис. 2.6).

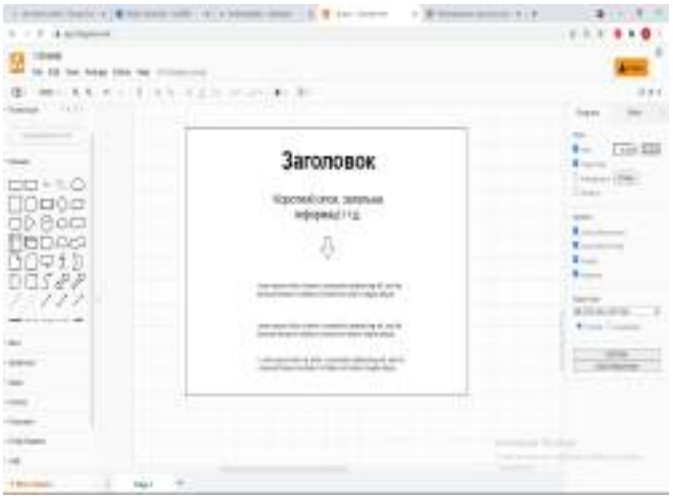

Рисунок 2.6 – Сторінка «Про нас»

На сторінці міститься заголовок та підзаголовок, які є однаковими на усіх сторінках. Також зображено стрілку униз, для того, щоб користувач розумів, що потрібна для нього інформація знаходиться нижче на сторінці.

Наступними йдуть блоки тексту, розподілені на невеликі абзаци. До

сторінок, які є не тільки інформативними, а й на котрих можна виконати певні дії належить сторінка пошуку волонтерів (рис. 2.7).

Рисунок 2.7 – Сторінка пошуку волонтерів

На сторінці містяться заголовок, підзаголовок і змінна кнопка. В залежності від того чи користувач авторизований, кнопка мінятиме вміст, а також направлятиме на різні сторінки. Якщо користувач авторизований, то відкриється форма створення завдання (рис. 2.8).

Рисунок 2.8 – Форма створення завдання

Форма відкривається у вигляді випливаючого вікна, або так званого popup, тому на ній розміщена кнопка закриття вікна. У формі є усі необхідні

поля для створення завдання, також вони є відповідного типу, наприклад наступних HTML типів:

- Текстовий (text).
- Текстове вікно (text area).
- Випадаючий список (select).
- Дата (date).

Також зображену кнопку для підтвердження створення завдання. Якщо користувач неавторизований він не переміститься на сторінку реєстрації (рис. 2.9).

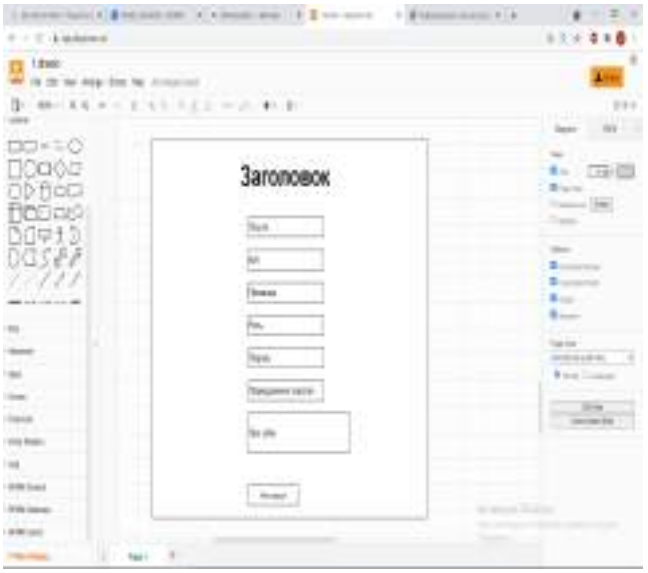

Рисунок 2.9 – Сторінка реєстрації

Сторінка містить заголовок та інтегровану форму з полями, необхідними для створення користувача та пароля для нього. Форма містить поля наступних HTML типів:

- Текстовий (text).
- Текстове вікно (text area).
- Випадаючий список (select).
- Електронна пошта (email),

− Пароль (password).

Під формою розміщено кнопку для підтвердження створення користувача. Якщо ж користувач зареєстрований, то він відкриє сторінку входу в систему (рис. 2.10).

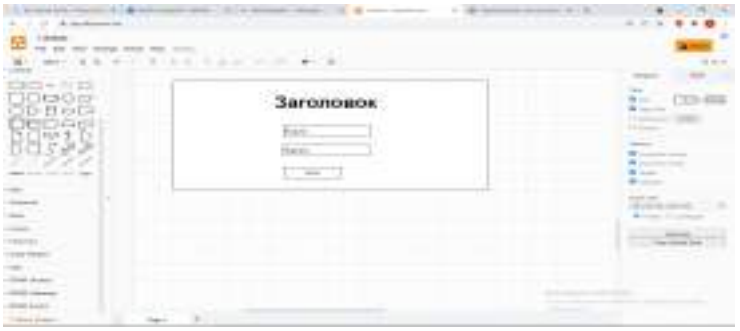

Рисунок 2.10 – Сторінка «Увійти»

Сторінка містить заголовок, а також просту форму для входу, що складається з двох полів та кнопки. Поля email та password типу.

Ще одною загальнодоступною формою є форма зворотного зв’язку (рис. 2.11).

Рисунок 2.11 – Форма зворотного зв’язку

Форма відкривається у popup вікна, тому на ній розміщена кнопка закриття вікна. У формі є три необхідні поля, та кнопка підтвердження

надсилання. Поля відносяться до таких HTML типів:

- Текстовий (text).
- Текстове вікно (text area).
- Електронної пошти (email).

Після створення повідомлення, відкриється інше popup вікно (рис. 2.12). Яке інформує про успішне виконання операції.

Рисунок 2.12 – Вікно сповіщення

Вікно, як і інші popup вікна містить кнопку закриття, а ще заголовок і кнопку підтвердження. Також дане вінок використовується не тільки для сповіщення про успішно виконані операції, а також і для підтвердження дій, як от на сторінці переліку завдань (рис. 2.13).

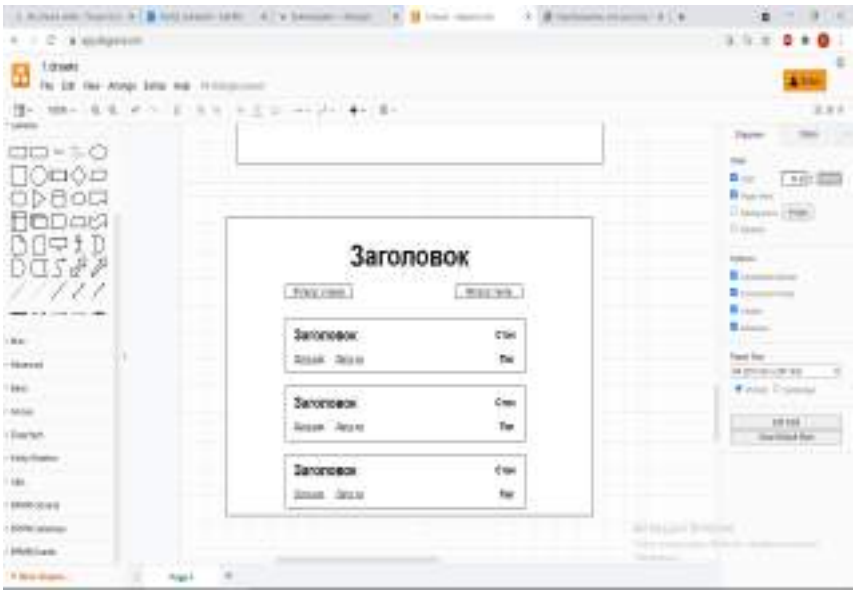

Рисунок 2.13 – Сторінка «Завдання»

На сторінці зображено заголовок, два типи фільтрів по даних з завдання, а також сам список завдань. Список завдань складається з окремих елементів, які крім того можуть відкриватись для ознайомлення з детальною інформацією. Вікдритий елемент завдання зображено на (рис. 2.14)

Рисунок 2.14 – Елемент завдання

Складається даний елемент з текстової інформації оформленої різними стилями та кнопки, яка дозволяє взяти завдання в роботу. Відображається наступна інформація про завдання:

- Заголовок.

- Дата початку.
- Дата закінчення.
- Стан.
- Тип.
- Адреса.
- Коментар.
- Телефон.

Загальний вигляд сторінок адміністрування відрізняється, вони не містять заголовку та складають зі списків інших форматів, які містять кнопки для додаткових операцій. На (рис. 2.15) зображено адміністративну сторінку керування користувачами.

Рисунок 2.15 – Сторінка керування користувачами

На сторінці зображено список користувачів, який крім загальної інформації про користувача містить і кнопки дій над ними. До загальної інформації користувача належать:

- Повне ім’я.
- Роль.
- Стан.

Та наступні кнопки дій:

- Активація або Деактивація, залежить від стану користувача.
- Редагувати.
- Видалити.

Решта сторінок та форм, є ідентичними до представлених макетів. Відрізняють інші макети тільки вмістом інформації тому цінності для огляду в роботі не мають.

Для реалізації інтерфейсу засобами для веброзробки обрано React та Typescript. Це створює можливість для використання бібліотеки Material UI, яка реалізує багато готових рішень для конструювання сторінок та елементів.

Також дана бібліотека містить стилі, які дозволяють розробляти сайти з адаптивною та мобільною верстккою.

Компоненти Material UI працюють ізольовано. Вони самодостатні та додадуть тільки ті стилі, які потрібні для їхнього відображення. Вони не залежать від яких – небудь глобальних стилів, таких як normalize.css [20].

Маючи макети сторінок, можна визначити, які компоненти варто використати [21]. При розробці буде визначено додаткові стилі та використано такі елементи бібліотеки:

- Grid – сітка адаптивного макету Material Design адаптується до розміру екрану і орієнтації, забезпечуючи злагодженість макетів. Адаптивний користувацький інтерфейс Material Design базується на сітці з 12 колонками. Використовує flex box.
- Link – створює блок з посиланням всередині.
- makeStyles – хук, який дозволяє перевизначати стилі елементів та створювати динамічні імена класів.
- Theme – тип теми, з усіма доступними їй елементами, дозволяє перевизначати тему, без додавання нових класів.
- Typography – використовується для оформлення текстів, має обмежену кількість розмірів, які гарно поєднуються між собою.
- Dialog – використовується для створення обгортки popup вікон.
- DialogContent – використовується для розміщення контенту в popup вікно.

- DialogTitle – використовується для заголовку popup вікон.
- Input – використовується для створення полів форм будь-яких типів.
- TextareaAutosize – використовується для створення текстового поля, розміри якого можна змінювати динамічно на сторінці.
- Select – використовується для створення випадаючих списків.
- MenuItem – використовується для створення елементів списку, випадаючого списку.
- Box – використовується для обгортки елементів, по замовчуванню створює div елемент навколо переданого елементу. Але елемент обгортки можна вказувати.
- Button – використовується для створення кнопок різних форматів.
- Collapse – використовується для реалізації анімацій елементів, що обгортає, ефекти анімації чітко визначені наперед.

Також дана бібліотека містить, ще кілька пакетів, зокрема пакет, який містить зображення та логотипи різних форматів. В ході розробки будуть обрані компоненти з даної бібліотеки для відображення стрілок, посилання на головну сторінку та інші.

2.3 Розробка основних функцій платформи

Дана платформа використовує в якості back-end частини додаток на Ruby on Rails, а для front-end використовує React. Є можливість побудувати додаток таким чином, що частина RoR програми, яка відповідає за відображення, буде генерувати HTML сторінки та JS код, який реалізований з використанням React, такі додатки називають монолітами, вони компактні, швидкі, але важко масштабуються, тому підходять для додатків невеликих, та таких, які не планують використовувати інші платформи або часто змінюватись.

Другий варіант використання відображення це використання його в якості

генератора JSON об’єктів, які описуватимуть дані, що відправлятимуться на front-end, даний тип додатків називають REST API [22]. Саме цей варіант задовольняє поставлені вимоги і перспективи платформи, тому варто розробити основні функції, що характерні такій архітектурі, а також базу проекту.

Тому за основний функціонал, як от робота з даними, побудова зв’язків та обробка запитів буде реалізований майже повністю на back-end частині.

В основі REST закладено принципи функціонування інтернету і в тому числі, можливості HTTP. Дані повинні передаватися у вигляді невеликої кількості, та в стандартних форматах (XML, JSON). Будь-який REST протокол повинен підтримувати кешування, та не повинен залежати від мережевого прошарку, також не повинен зберігати інформації про стан між парами запит-відповідь. Стверджується, що такий підхід забезпечує масштабовність системи і дозволяє їй еволюціонувати з новими вимогами [23].

Перша архітектура від якої REST успадковує обмеження – це клієнт-серверна архітектура. Дані обмеження вимагають розділення відповідальності між компонентами, які займаються зберіганням та оновленням даних (сервером), і тими компонентами, які займаються відображенням даних на інтерфейсі користувача та реагують на дії з цим інтерфейсом (клієнтом). Таке розподілення дозволяє компонентам еволюціонувати незалежно.

Наступним обмеженням є те, що взаємодії між сервером та клієнтом не мають стану, тобто кожен запит містить всю необхідну інформацію для його обробки, і не покладається на те, що сервер знає щось з попереднього запиту.

Обмеження «відсутність стану» не означає, що архітектура REST дозволяє будувати лише системи, в яких немає стану. Відсутність стану означає, що сервер не знає про стан клієнта і не повинен запам'ятовувати послідовність здійснених до нього запитів, тому що кожен з них є незалежним. Коли клієнт, наприклад, запитує головну сторінку сайту, сервер відповідає на запитання і забуває про клієнта. Клієнт може залишити сторінку відкритою протягом

кількох років, перш ніж натиснути посилання, і тоді сервер відповість на інший запит. Тим часом сервер може відповідати на запити інших клієнтів, або нічого не робити – для клієнта це не має значення.

Таким чином, наприклад дані про стан сесії (користувача, який автентифікувався) зберігаються на клієнті і передаються з кожним запитом. Це покращує масштабовність, бо сервер після закінчення обробки запиту може звільнити всі ресурси, задіяні для цієї операції, без жодного ризику втратити цінну інформацію. Також спрощується моніторинг і зневадження, бо для того аби розібратись у тому, що відбувається в певному запиті, досить подивитись лише на цей запит. Збільшується надійність, бо помилка в одному запиті не зачіпає інші [23].

Мінусом цього обмеження є те, що знижується продуктивність через те, що в кожен запит тепер доводиться додавати дані сесії з клієнта. Також збереження стану на різних клієнтах важче підтримувати, бо реалізації клієнтів можуть відрізнятись, тоді як середовище сервера повністю під контролем розробника.

Додатковим обмеженням стилю REST є те, що системи, написані в цьому стилі, повинні підтримувати кешування, тобто дані, які передаються сервером, повинні містити інформацію про те, чи можна їх кешувати, і якщо можна, то як довго. Це дозволяє збільшувати продуктивність, уникаючи зайвих запитів, але також зменшує надійність системи через те, що дані в кеші можуть застаріти.

Всі компоненти в архітектурі REST підтримують однорідний інтерфейс. Це зменшує зв'язність між компонентами і сервісами які вони надають і дозволяє нескладно змінювати компоненти при потребі. Це досягається кількома точнішими обмеженнями:

- Ідентифікація ресурсів: URI запит повинен точно визначати ресурс, що очікується та якого формату повинна бути відповідь. У відповідь на запит клієнту надається представлення ресурсу. Ресурси концептуально

відділені від представлення. Наприклад, сервер може надсилати дані із бази даних в форматі HTML, XML або JSON, жоден з яких не являється типом даних, що зберігаються всередині сервера.

- Маніпуляція ресурсами через представлення: якщо клієнт має представлення ресурсу, включаючи метадані, він має достатньо інформації, щоб модифікувати або видаляти цей ресурс.
- Самоописові повідомлення: кожне повідомлення має включати достатньо інформації, щоб обробити повідомлення. Наприклад, обробник (parser), що використовується для розшифровки даних, може бути вказаний в списку MIME типів.
- Гіпермедіа як рушій стану застосунку: клієнт, що має доступ до REST сервісу, повинен отримати всі наявні сервіси та методи за допомогою гіперпосилань.

Наступним обмеженням для REST є поділ на шари абстракції. Кожен компонент потрапляє в якийсь шар, і спілкується лише з компонентами в шарі під ним або в шарі над ним. Обмеження знання системи одним шаром зменшує складність компонентів [23].

Останнім архітектурним обмеженням в REST є те що клієнти повинні дозволяти розширювати свою функціональність дозволяючи завантаження додаткового коду в формі аплетів чи скриптів. Щоправда це необов'язкове обмеження.

Компоненти REST системи спілкуються, передаючи один одному представлення ресурсу в форматі, що обирається з оновлюваного набору стандартних форматів даних. Формат обирається динамічно відповідно до бажань компонента-клієнта і можливостей сервера. Чи представлення має той самий формат, що й сам ресурс, чи є результатом якогось перетворення – це деталь реалізації, яка ховається за інтерфейсом.

Для того, щоб посилатись на ресурси, використовуються ідентифікатори

ресурсів. Компонент, який надав ресурсу ідентифікатор і дозволяє звертатись до нього за цим ідентифікатором, відповідає за збереження функції приналежності незмінною.

Відповідно до документації RoR, згенерувати проект з архітектурою REST можна за допомогою стандартних команд. Це в свою чергу зробить наступне:

- Запустить додаток з обмеженою кількістю проміжних сервісів.
- ApplicationController буде унаслідуватись від ActionController::API замість ActionController::Base.
- Пропустить генерацію файлів відображення.

Генерація відбувається за допомогою наступної команди:

```
rails new project-api --api -T
```

Використання аргументу --api вказує, що ми генеруємо саме проект з потрібною API архітектурою. А аргумент –T вказує генератору пропустити підключення стандартного фреймворку для тестів. Тести можна реалізувати за допомогою інших гемів, наприклад RSpec.

Також, для роботи основних функцій додатку потрібно декілька гемів, залежності на які описані в Gemfile. Вміст Gemfile проекту:

```
source 'https://rubygems.org'
git_source(:github) { |repo| "https://github.com/#{repo}.git" }
ruby '2.7.3'
gem 'rails'
gem 'puma', '~> 5.0'
gem 'bootsnap', '>= 1.4.4', require: false

group :development do
  gem 'byebug', platforms: [:mri, :mingw, :x64_mingw]
  gem 'listen', '~> 3.3'
  gem 'rubocop', require: false
  gem 'spring'
end

gem 'tzinfo-data', platforms: [:mingw, :mswin, :x64_mingw, :jruby]
gem "mongoid", "~> 7.2"
```

```
gem "rack-cors", "~> 1.1"
gem "mini_magick", "~> 4.11"
gem "carrierwave-mongoid", "~> 1.3"
gem "jbuilder", "~> 2.11"
gem "jwt", "~> 2.2"
gem "bcrypt", "~> 3.1"
gem "faker", "~> 2.17"
```

Деякі з наведених гемів додаються автоматично і є частиною додатку за замовчуванням, тому варто коротко описати тільки ті, які було додано для певних функцій. Також не будуть описані геми які використовують в середовищі розробки і виділені в окремий блок.

- Puma – швидкий та простий HTTP 1.1 сервер для Ruby за стосунків.
- Mongoid – є ODM (об’єктно-орієнтований маппер) фреймворк для роботи з MongoDB в Ruby.
- Rack-cors – реалізує підтримку Cross-Origin Resource Sharing (CORS) для Rack сумісних веб-застосунків.
- Jbuilder – надає просту DSL для визначення JSON, що дозволяє упорядковувати величезні хеш структури.
- Jwt – Ruby імплементація OAuth JSON Web Token (JWT) стандарту.
- Bcrypt – гем для збереження і генерації безпечних паролів.
- Faker – бібліотека, яка генерує неправдиві дані, для тестів наприклад.

Після встановлення даних гемів варто описати моделі даних. Опис моделей даних (основних полів без валідацій) наведено вище.

Наступним кроком є створення неправдивих але валідних даних для заповнення бази. Це дає можливість відтестувати і наповнити базу даними без участі реальних користувачів і не витрачати на це багато часу, заповнюючи базу в ручну. Створення даних відбуваєть у файлів seeds.rb:

```
def create_user (
  role,
  email = Faker::Internet.email,
  password = Faker::Internet.password,
  first_name = Faker::Name.first_name,
  last_name = Faker::Name.last_name,
```

```
  activated = Faker::Boolean.boolean,
  avatar = seed_image('avatar.jpg'),
  description = Faker::Quotes::Shakespeare.hamlet_quote
)
  User.create! do |user|
    user.email = email
    user.role = role
    user.password = password
    user.first_name = first_name
    user.last_name = last_name
    user.activated = activated
    user.avatar = avatar
    user.description = description
  end
end
```

Описано тільки два методи з файлу, решту коду можна переглянути в додатках. Наведені методи створюють об’єкт завдання та користувача. Об’єкти створюються як в базі так і в додатку. Частина з полів заповнюються неправдивими даними, частина вручну, або розраховується на основі отриманих параметрів.

Наступні два кроки пов’язані з обробкою даних та формуванням даних для front-end частини у форматі JSON. Обробка даних відбувається вбільшості у контролерах, але так як функцій дуже багато, вони є громіcткими, через велику кількість методів. Аналогічно і з формуванням даних. Код усіх контролерів там файлів формування JSON файлів наведено у додатках.

Найважливішою функцією платформи є авторизація, без якої неможлива правильна робота застосунку. Варто оглянути частину розробки функціоналу пов’язану саме з цим. Контролер який відповідає за авторизацію користувача в системі, називається AuthenticationController, він містить тільки один action – login. Також реалізовано допоміжну приватну функцію для обробки параметрів, яка називається login_params. Ця функція витягує з запиту необхідні параметри, як от пароль та пошту. Також в даному файлі визначено константу в котрій збережено значення часу життя точену – JWT_TOKEN_LIVE_TIME. А створення самого точену відбувається в моделі користувача у відповідному

методі, за допомогою гему JWT.

```
def login
  @user = User.where(email: params[:email]).first
  authenticated = false
  begin
    authenticated = @user&.authenticate(params[:password])
  rescue Exception => e
    puts "invalid logging #{e}"
  end
  authenticated ?
    render(json: {
      token: @user.jwt_token(JWT_TOKEN_LIVE_TIME),
      role: @user.role
    }, status: :ok)
  :
    render(json: { error: 'unauthorized' }, status: :unauthorized)
end
```

Action login шукає користувача по пошті, після цього, якщо користувача знайдено відбувається перевірка паролю через функцію authenticate і гем bcrypt. Якщо користувача пароль співпадає, то генерується відповідь на запит. Об’єкт відповіді описується прямо у action, бо не потребує додаткових опрацювань. Він складається з поля token, яке містить токен з зашифрованих даних користувача на основі ключа застосунку, а також поля role, яке описує роль користувача і допоможе на fron-end частині визначити, які сторінки показувати користувачу, а які ні. Об’єкт надсилається зі HTTP статусом 200. Якщо користувача не знайдено, або пароль не підійшов, то у відповідь надсилається об’єкт, що містить одне текстове поле error, та статус 401.

Для того, щоб описати шлях за яким знайти ресурси описані в програмі, потрібно визначити їх у файлі routes.rb. За допомогою спеціального синтаксису він генерує шляхи для відповідних HTTP методів. Також можна визначати дані шляхи вручну. Вміст файлу наведено нижче.

```
Rails.application.routes.draw do
  resources :tasks
  resources :callbacks , only: %i[create, index]
  resources :users
  post '/login', to: 'authentication#login'
```

```
end
```

Оператор resources генерує шляхи усіх методів CRUD, тобто post, get, update, patch, delete, для вказаного контролера [24]. У випадку даної платформи це необхідно для ресурсу користувачів та завдань. А ресурс зворотного зв’язку матиме тільки два action-а вказаних через параметр only у масиві. Шлях для авторизації визначений вручну і працюватиме через метод post.

Основні функції back-end частини розроблені з основними правилами та обмеженнями архітектури REST. В даному розділі було оглянуто основні особливості архітектури та роботи back-end частини платформи. Здійснено огляд реалізації таких елементів REST як: ресурс, URI ресурсу та обробка запитів. Описано генерацію даних платформи, яка необхідна для розробки та огляду системи.

Висновок до розділу 2

В другому розділі ми розглянули розробку структури платформи та бази даних, а саме: розробку бази даних, розробку інтерфейсу платформи, розробку основних функцій.

РОЗДІЛ 3. РЕАЛІЗАЦІЯ ТА ОГЛЯД ПЛАТФОРМИ

3.1 Розробка адмінпанелі

Адмінпанель частково розроблена на двох частинах платформи, front-end та back-end, в залежності від призначення функціоналу. Логіки роботи з даними БД знаходиться на back-end частині, а інтерфейс та засоби звернення до back-end на front-end частині. Основні функції адмінпанелі:

- CRUD операції над користувачами та завданнями.
- Активація користувача.
- Публікація завдання.
- Перегляд та видалення повідомлень зворотного зв’язку.
- Валідація форм.
- Аутентифікація з використанням JWT.
- Перевірка ролі користувача на відповідність.
- Підтримка ODM (Mongoid).

Розробка аутентифікації і шляхів до ресурсів описано в попередньому розділі, тому в даному розділі буде розглядатись частково front-end реалізація та деякі action-и контролерів з back-end частини, наприклад CRUD операції над завданнями. Далі наведено фрагменти файлу tasks_controller.rb в якому містяться функції обробки запитів і даних пов’язаних з завданнями.

```
def index
    if @current_user.present? && @current_user.is_admin?
      @tasks = Task.all
    else
      @tasks = Task.published_tasks
    end
    render :index, status: :ok
  end
```

Дана функція відповідає за повернення списку завдань. Для різних користувачів повернеться різний список, якщо користувача нема, або він не є адміністратором, то повернеться список опублікованих завдань. Якщо ж користувач існує і його роль відповідає адміністратору, то йому повернуться усі можливі завдання, що є в системі. Формування даних відбувається за допомогою index.json.jbuilder.

```
def create
  @task = Task.new(task_params)
  @task.author = @current_user

  if @task.save
    render :show, status: :ok
  else
    render json: {messages: @task.errors}, status: :unprocessable_entity
  end
end
```

Action реалізовує функціонал створення завдання. Нове завдання створюється за допомогою допоміжного методу task_params, який отримує з запиту усі необхідні поля. Також Завданню призначається автор, а саме поточний користувач, який здійснив запит. Якщо завдання успішно зберігається, то у відповідь генерується об’єкт створеного завдання, через генератор show.json.jbuilder. Якщо ж завдання не вдалось зберегти, то повертається об’єкт, який містить поле message з помилками, що виникли при створенні. Статус ж проваленого запиту буде не 200, а 402, що означає, дані не вдалось обробити.

```
def destroy
  @task = Task.where(id: params[:id])
  if @task.destroy
    render json: {}, status: :ok
  else
    render json: {}, status: :not_found
  end
end
```

Видалення завдання відбувається наступним чином. Спочатку по ідентифікатору завдання, воно шукається в базі. Якщо завдання знайдено та

успішно видалено, то у відповідь повертається порожній об’єкт зі статусом 200. Якщо ж щось пішло не так, або завдання не знайдено, то повертається порожній об’єкт зі статусом 404.

```
def update
  @task = Task.where(id: params[:id]).first

  if @task.update(task_params)
    render :show, status: :ok
  else
    render json: {messages: @task.errors}, status: :unprocessable_entity
  end
end
```

Оновлення завдання дуже схоже на його створення. Тільки для початку потрібно знайти завдання в базі по його ідентифікатору. Потім допоміжний метод, який витягає усі параметри передає їх в метод оновлення самого завдання. Якщо оновлення успішне, то повертається об’єкт оновленого завдання (генерується через show.json.jbuilder). Якщо ж щось пішло не так, тоді у відповідь надсилається об’єкт з полем message, що містить помилки, та статус 402.

```
def change_publish
  @task = Task.where(id: params[:id]).first

  if @task.update(published: !@task.published)
    render :show, status: :ok
  else
    render json: {messages: @task.errors}, status: :unprocessable_entity
  end
end
```

Метод change_publish реалізовує публікацію або архівування завдання. По суті це те саме оновлення, тільки замість полів з отриманих параметрів, оновлюється тільки поле published. При оновленні поля використовується інвертоване попереднє значення. Повернення і формування об’єктів та статусів, аналогічне з оновленням.

Оглянуті методи роботи з даними, які обробляють запити з front-end є майже ідентичними з такими ж методами інших ресурсів, таких як користувачі та повідомлення зворотного зв’язку. Тому огляд їхніх контролерів в даному розділі не здійснено.

Більшість CRUD операцій, а саме тільки ті, які доступні з адмінпанелі повинні бути авторизованими. Для цього перед кожною обробкою запиту відбувається перевірка токену.

```
def authorize_request
  header = request.headers['Authorization']
  header = header.split(' ').last if header
  @current_user = nil
  begin
    @decoded = JsonWebToken.decode(header)
    @current_user = User.where(id: @decoded[:user][:id]).first
```

Метод authorize_request, що міститься в ApplicationController реалізовує перевірку токена. Для перевірки спочатку потрібно отримати з запиту відповідний заголовок, в даному випадку це Authorization. Після отримання з заголовку значення токену він розшифровується.

В розшифрованому токені міститься хеш користувача з його ідентифікатором, саме по ньому шукається користувач в базі. Після цього глобальній змінній поточного користувача @current_user присвоюється значення знайденого користувача. Якщо не вдалось розшифрувати токен, або користувача немає, то повертається об’єкт з полем errors в якому міститься повідомлення про помилку, та статус 401.

Огляд обробки запитів здійснено, але варто оглянути і розробку функціоналу здійснення запитів та інтерфейсу front-end частини. Так як адмінпанель призначена для використання користувачем з роллю superadmin, варто оглянути як здійснюється контроль над цим в інтерфейсі користувача.

Для адміністратора повинен бути відкритий доступ до окремих сторінок.

Тобто він повинен мати окреме меню та маршрутизацію на нові сторінки. Додавання нового меню для адміністратора реалізовано у вайлі App.ts.

```
const App: React.FC<TProps> = ({ userRole }) => {
  const classes = useStyles()
  return (
    <BrowserRouter>
      <Container maxWidth="xl" className={classnames(classes.root, 'App')}>
        {userRole === 'superadmin' ? <AdminHeader /> : <Header />}
        <Router />
        {userRole !== 'superadmin' && <Footer />}
      </Container>
    </BrowserRouter>
    )
}
```

У наведеному фрагменті коду описана загальна структура додатку на стороні інтерфейсу, а саме компонент App. Компонент отримує єдиний парметр (props), а саме роль користувача userRole. Передається цей параметр з браузерного сховища redux, часткова реалізація якого буде оглянута в наступному підрозділі. З безлічі елементів, які описують структуру компонента потрібно розглянути ті, в котрих міститься меню, а саме Header та AdminHeader. Дані компоненти відображаються відповідно до умови, якщо роль користувача є адміністратором, тоді відображається меню для адмінпанелі, якщо ні то звичайне меню користувача.

```
{userRole === 'superadmin' && (
  <>
    <Route exact path={userAdminPath} render={() => <AdminUsers />} />
    <Route exact path={tasksAdminPath} render={() => <AdminTasks />} />
                <Route exact path={callbackAdminPath} render={() => <div
className="callbacks admin" />} />
    <Route path="*" render={() => <Redirect to={tasksAdminPath} />} />
  </>
)}
```

Можливість маршрутизації сторінок адмінпанелі тільки коли користувач має потрібну роль є необхідністю. Дана функціональність реалізована у файлі

Router.ts в якому містяться усі маршрутизації платформи.

У наведеному фрагменті коду реалізовується перевірка ролі користувача, якщо вона відповідає ролі адміністратора, тоді маршрутизації на сторінки адмінпанелі будуть доступними.

Повний огляд коду сторінок адмінпанелі в інтерфейсі важко реалізувати в рамках пояснювальної записки, тому що кількість коду є надто великою. Тому буде здійснено огляд методів для виклику API та окремі фрагменти обробки даних на сторінках.

```
export const getTasks = (): Promise<AxiosResponse<ITask[]>> => {
  const request = createRequest()
  return request.get('tasks')
}
```

Функція getTasks та інші запити до API використовують пакет axios, який дозволяє створювати асинхронні запити до віддалених точок. Функція createRequest використовує axios з певними налаштуваннями для створення запиту, додає заголовки з токеном та вказує хост запиту. Функція для отримання списку завдань не потребує параметрів, а повертає специфічно огорнуту відповідь, яку потрібно обробити на випадок виникнення помилок. Запит відправляється типу get на шлях host/tasks.

```
export const createTask = (task: ITask): Promise<AxiosResponse<ITask>> => {
  const request = createRequest()
  return request.post<IUserData>('tasks', task)
}
```

Функція створення завдання як і інші, що пов’язані з запитами до API використовує createRequest. Як параметр отримує об’єкт завдання. Створює запит на URI host/tasks post типу з параметрами, які складаються з полів завдання. Повертає об’єкт завдання.

```
export const updateTask = (task: Partial<ITask>): Promise<AxiosResponse<ITask>>
=> {
```

```
  const request = createRequest()
  return request.patch<IUserData>('tasks', task)
}
```

Функція оновлення отримує параметром частковий об’єкт завдання, так як не усі поля можуть змінюватись. При створенні запиту використовує URI для завдань, тип запиту patch та поля завдання як параметри. Повертає об’єкт завдання.

Функція видалення завдання, отримує один параметр, який є текстовим ідентифікатором завдання. При створенні запиту використовує URI завдань, метод delete та ідентифікатор як параметр. Нічого не повертає.

Інші запити до API реалізовані майже ідентичним чином, тільки відрізняються параметрами відповідно до ресурсу а також URI.

В компонентах сторінок, що відображають сутності в адмінпанелі використовується хук для оновлення даних з сервера.

```
React.useEffect(() => {
  const storeTasks = async () => {
    const response = await getTasks()
    if (response.status === 200 && response.data) {
      saveTasksList(response.data)
    }
  }
  storeTasks()
}, [saveTasksList])
```

Даний хук ефекту запускається при початковому рендері сторінки з завданнями. В середині створена асинхронна функція, яка викликається при спрацюванні ефекту. Дана функція здійснює запит до сервера і при успішному виконанні (код статусу дорівнює 200) і наявності даних у відповіді, зберігає список завдань у сховище redux. Аналогічним чином здійснюється обробка інших запитів до сервера, але міститись вона може не тільки у хуках ефекту, але й в функціях, що викликаються при певних подіях на сторінці. Після оновлення даних у сховищі redux сторінка оновлюється.

Адмінпанель коректно працює, усі дані занесені через неї, правильно

відображаються на сторінках сайту. Реалізований функціонал адмінпанелі сповна задовольняє поставлені завдання та загальні вимоги, для адміністрування та модерування сайту.

3.2 Розробка панелі користувача

Основний функціонал, що реалізований на back-end частині платформи детально оглянуто в попередніх розділах. До даного функціоналу входить:

- Аутентифікація.
- CRUD операції над даними.
- Налаштування роботи додатку відповідно до REST архітектури.
- Створення URI для ресурсів.
- Моделі даних.

Саме тому надалі буде розглянуто тільки розроблення функціоналу панелі користувача, що знаходиться на front-end частині платформи. Щоб при кожній дії, або зміні даних в інтерфейсі платформи не доводилось здійснювати новий запит, а також, щоб зберігати дані користувача, такі як токен і роль, у браузері використано redux.

Для збереження даних також використовуються стан компонента React, але дане сховище є ізольованим, також можна використовувати React Context але у нього є свої недоліки. Одним з таких недоліків є те, що компоненти реагують на зміну будь-якого елементу в сховищі, навіть якщо він не використовується в самому компонентів. Зайві рендери погано впливають на продуктивність додатку.

Redux є універсальним засобом розробки і може бути використаним в зв’язці з різними бібліотеками та фреймворками. Дана технологія забезпечує єдине сховище для усіх даних, а функції, що надають дозволяють реалізувати довільну логіку реагування на зміни в стані сховища [25].

Також для того, щоб працювати саме з React потрібно встановити пакет react – redux, який впроваджує елементи взаємодії з компонентами. Redux не зберігає дані в сховище браузера, а тримає їх в самому додатку в JS, а для того, щоб користувач кожен раз після оновлення сторінки не авторизовувався, потрібно зберігати дані і в браузері, а саме в local storage. Для цього підходить компонент, що автоматично зберігає дані з redux у сховище браузера, а саме redux-persist. Не потрібно зберігати усі дані, а тільки дані користувача, пакет автоматично їх оновлює і при завантаженні сторінци вони знаходяться у сховищі браузера.

Опис та налаштування сховища відбувається у папці store проекту. Саме тут буде описано дії з даними, для оновлення їх та отримання. Також оисано налаштування redux-persist і структури даних сховища. Налаштування сховища відбувається у однойменному з папкою файлі і виглядає наступним чином.

```
const persistConfig = {
  key: 'root',
  storage,
  whitelist: ['userData'],
}
const persistedReducer = persistReducer<any>(persistConfig, rootReducer)
const composedEnhancer = composeWithDevTools()
const configureStore = () =>
     createStore<IState, any, unknown, unknown>(persistedReducer, initialState,
composedEnhancer)

const store = configureStore()

export const storePersistor = persistStore(store)
export default store
```

У файлі описано ключ для даних в локальному сховищі даних, також яку саме частину даних з redux потрібно опрацьовувати userData. Наступним формується новий редюсер даних, з використанням основного редюсера та пакету redux-persist. Далі додано функціонал для роботи зі спеціальним розширенням в браузері, для того щоб при розробці була можливість відслідковувати дані, що зберігаються. На базі створених даних та налаштувань,

а ще даних ініціалізації стану, створюється сховище, а також спеціальний метод для функціонування redux – persist.

Основний редюсер (rootReducer) формується з об'єднаних редюсерів, а саме усіх сутностей, які потрібно зберігати. Для полегшення написання reducer – ів та дій (actions) використано пакет react–act. Цей пакет описує створення дій та редюсерів правильним шляхом та надає відповідний набір функції для роботи з ними. Використання даного пакету спрощує роботу та покращує читабельність коду. Фрагмент файлу reducers.ts наведено далі.

```
const tasks = createReducer<ITask[]>({}, initialTasksList)
tasks.on(addTask, (state, payload) => [...state, payload])
tasks.on(updateTask, (state, payload) => [...state.map((e) => (e.id === payload.id ?
{ ...e, ...payload.task } : e))])
tasks.on(removeTask, (state, payload) => [...state.filter((e) => e.id !== payload)])
tasks.on(saveTasksList, (state, payload) => [...payload])

export const rootReducer = combineReducers<IState>({
  userData: userData,
  tasksList: tasks,
  usersList: users,
  callbacksList: callbacks,
})
```

Для створення редюсера та логіки обробки даних завдань спочатку створюється порожній редюсер, в яких передається стан ініціалізації даного фрагменту сховища. Також передається generic тип даних, що будуть опрацьовані. До створеного редюсера відповідні дії додаються методом on, який отримує два параметри: дію та функцію обробки даних. Функція обробки даних, отримує такі параметри як поточний стан (state) та дані (payload), які потрібно додати або замінити у сховищі. Для завдань створено і описано наступні види обробки даних (згідно дій):

- Додавання завдання в масив завдань.
- Оновлення завдання з масиву.
- Видалення завдання з масиву.
- Збереження нового списку завдань.

Основний редюсер (rootReducer) створюється методом combineReducer, який отримує об’єкт з вказаними редюсерами для кожного елементу даних в сховищі, також передається generic тип сховища.

Для виклику внесення змін у сховище використовуються actions (дії), які описують, що повинно передатись для обробки змін. При їх використанні з Typescript (TS) варто створити відповідний тип з усіма типами дій, це дозволить в подальшому зручно використовувати їх у компонентах додатку. Усі дії та їх типи описуються у файлі actions.ts.

```
export const addTask = createAction<ITask>('add task')
export const updateTask = createAction<{ id: string; task: Partial<ITask> }>(
  'update task',
  (id: string, task: Partial<ITask>) => ({
    id,
    task,
  }),
)
export const removeTask = createAction<string>('remove task')
export const saveTasksList = createAction<ITask[]>('save tasks list')

  const removeTaskType = removeTask.getType()

interface IRemoveTaskAction {
  readonly type: typeof removeTaskType
}

export type ActionsTypes =
  | IRemoveTaskAction
  | IAddTaskAction
  | IUpdateTaskAction
  | ISaveTasksListAction
```

Дії створюються функцію (createAction) пакету redux-act, яка отримує generic тип даних, опис дії та за потреби опис обробки даних дії. У наведеному фрагменті файлу описані дії пов’язані з завданнями, а також частково описано створення типів дій, які в подальшому будуть використані для підключення у компонентах. Усі дії об’єднані у один тип (ActionsTypes), що буде використовуватись при передачі дій в пропси (props) компоненту. Actions по суті є константами, які описують певні дії.

Для підключення стану до компоненті спочатку потрібно підключити react–redux до самого застосунку. Це саме стосується і пакету redux–persist. Підключення даних функцій відбувається у файлі src/intex.ts.

```
ReactDOM.render(
  <Provider store={store}>
    <PersistGate loading={null} persistor={storePersistor}>
      <React.StrictMode>
        <App />
      </React.StrictMode>
    </PersistGate>
  </Provider>,
  document.getElementById('root'),
)
```

Підключення відбувається обгортанням компоненту додатку у відповідні компоненти Provider та PersistGate. Як пропси компоненти приймають створені раніше, у файлі конфігурації, функції та саме сховище. Наступним кроком є підключення сховища до окремих компонентів, як от підключення до компоненту, що відображає завдання.

```
const mapStateToProps = (state: IState) => ({
  tasks: selectTasks(state),
})
const mapDispatchToProps = (dispatch: Dispatch<ActionsTypes>) => ({
  saveTasksList: (data: ITask[]) => dispatch(saveTasksList(data)),
})
const connector = connect(mapStateToProps, mapDispatchToProps)
type TProps = ConnectedProps<typeof connector>
export const AdminTasks = connector(TasksPage)
```

У наведеному фрагменті коду описується підключення і передавання, у якості пропсів, списку завдань та дії, яка збереже новий список у сховище. mapStateToProps метод що викликається кожен раз, коли відбувається оновлення store і саме він передає необхідні значення з store в компонент. Тобто компонент, реагує і оновлює UI кожен раз, коли tasks оновились. На оновлення інших полів сховища не реагує. За допомогою mapStateToProps можно чітко визначити, які поля цікавлять компонент. У даному випадку значення отримується за допомогою функції селектора, що витягує необхідне поле зі сховища.

mapDispatchToProps функція передає в компонент методи для оновлення необхідного поля в store. Щоб не викликати метод dispatch напряму з компонента. Передає в props метод виклик, якого Приведе до виклику dispatch і оновить відповідні поля в сховищі. Так виклик змін виглядає краще та простіше. В даному випадку saveTasksList оновлює в store список завдань tasksList, які потім передають в цей ж компонент.

Функція connect зв’язує mapStateToProps і mapDispatchToProps з компонентом і передає потрібні поля і методи в нього. Повертає компонент обгортку для переданого компоненту.

Також описано тип props, який передається при описі компоненту, він базується на типі компоненту обгортки. І власне здійснене обгортання компоненту TasksPage.

З використанням redux панель користувача реалізовує продуктивну роботу з даними, що отримуються з серверами, але також важливими є використання функціональних компонентів, які є швидшими за класові і гарна декомпозиція. Огляд фрагменту коду компоненту Tasks.ts, який описує роботу та відображення загальнодоступно сторінки з завданнями, наведено нижче.

Функція userStyles перетворює методами JSS стилі та назви класів для компонента у об’єкт, яким можна вказати відповідні стилі і класи для елементів. Дана функція розроблена на базі makeStyles пакету material-ui.

Хук useState використовується для створення внутрішнього стану функціональних компонентів. В даному випадку для списку завдань та значень фільтрів завдань. Отримує значення для ініціалізації як параметр, також generic тип даних.

Хук useEffect використовується для обробки даних і виклику рендеру компоненту. В даному випадку реагує на зміни стану, а саме списку завдань та фільтрів, після чого наново перемальовує компонент з оновленими даними. Також опрацьовує дані та здійснює зміни стану списку завдань.

З використанням компонентів Material UI створено фільтр стану завдань. Для правильного розміщення елементу використовується Grid, а для відображення заголовку та станів Typography. Динамічно визначається додаткове ім’я класу у випадку, якщо обрано певний стан. Даний клас змінює вигляд обраного фільтру на сторінці. Props onClick відповідає за подію натискання на елемент, тому в неї передана функція, яка встановлює фільтр стану у потрібне значення.

Метод map з якого повертається компонент Task, відповідає за відображен ня завдань зі списку. На кожну одиницю даних створюється окремий елемент з переданими в props завданням.

Оглянуто основні особливості роботи панелі користувача на front-end частині. Інший функціонал та компоненти реалізовані схожим чином з використанням пакетів redux, material-ui та наведених вище підходів.

3.3 Огляд основних функцій платформи

Реалізовану платформа запускається локально у два етапи: запускається сервер API (back-end), після цього запускається сервер React додатку (front-end). Для запуску і розміщення на виділеному сервері краще використовувати Docker, а саме docker-compose, де можна описати усі сервіси та взаємодію між ними. Огляд здійснюється на запущеному локально проекті у середовищі development. Щодо даних використаних для заповнення БД у оглядових цілях, використано розроблені в seeds, тому дані виглядатимуть в більшості незрозумілими. При запуску інтерфейсу відкривається головна сторінка (рис. 3.1).

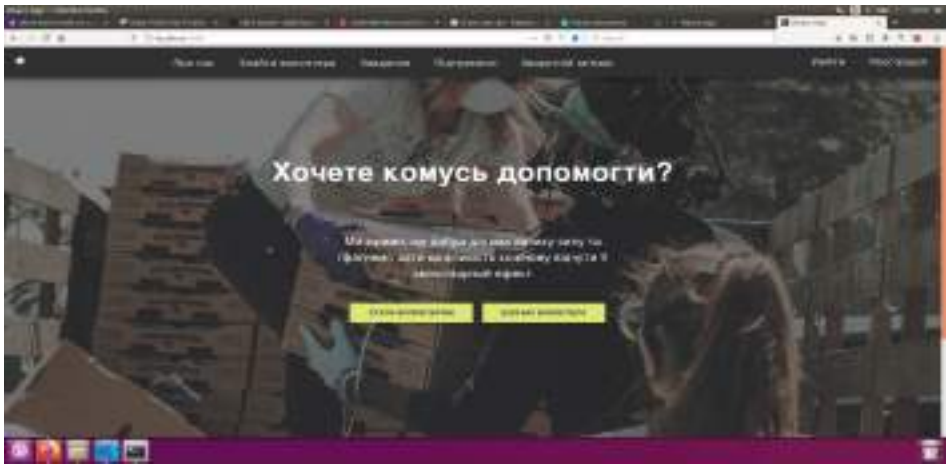

Рисунок 3.1 – Головна сторінка сайту

Сторінка реалізовано згідно макету, підібрано однорідну гармонійну кольорову гамму. Для фону на сторінках використано фото, які знаходяться у вільному доступі. Також розроблено меню з відповідними пунктами та кнопки у контенті сторінки, які виконують функції описані у вимогах. Футер сторінки не вмістився на поточний рисунок, тому його зображено на (рис. 3.2)

Рисунок 3.2 – Футер сторінок сайту

Даний блок розроблено згідно вимог і макету. Розміщається внизу кожної сторінки та містить потрібні блоки. Навігацію по сайту можна здійснювати і через футер, а також можна ознайомитися з посиланнями на соціальні мережі. Навігація містить не усі сторінки, а також не містить сторінок адміністрації сайту. Наступною в порядку меню є інформаційна сторінка «Про нас» (рис. 3.3).

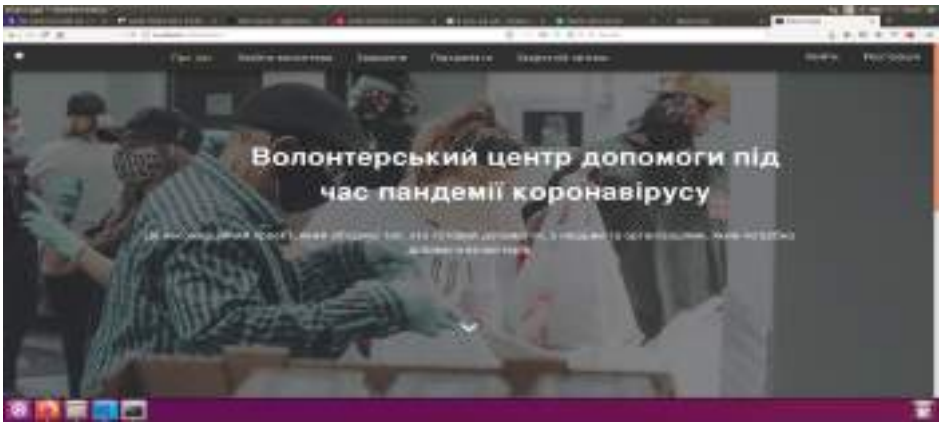

Рисунок 3.3 – Головний екран сторінки «Про нас»

На сторінці відсутні інтерактивні елементи, окрім меню, яке є на усіх сторінках. Також у меню відповідний пункт виділено як активний. Заголовок та підзаголовок інформують про сайт та організацію. Стрілка вниз підказує користувачу, що нижче можна ознайомитись з інформацією, після гортання сторінки (рис. 3.4).

Рисунок 3.4 – Продовження сторінки «Про нас»

Після того як користувач горнув сторінку вниз можна побачити текстові блоки у вигляді абзаців, які містять тільки текстову інформацію. Також можна побачити частину футера і те, що меню є фіксованим відносно позиції на сторінці, тому користувач не втратить його з поля зору. При переході на сторінку «Знайти волонтера» (рис. 3.5) можна ознайомитись з розробленою

формою та модальним вікном.

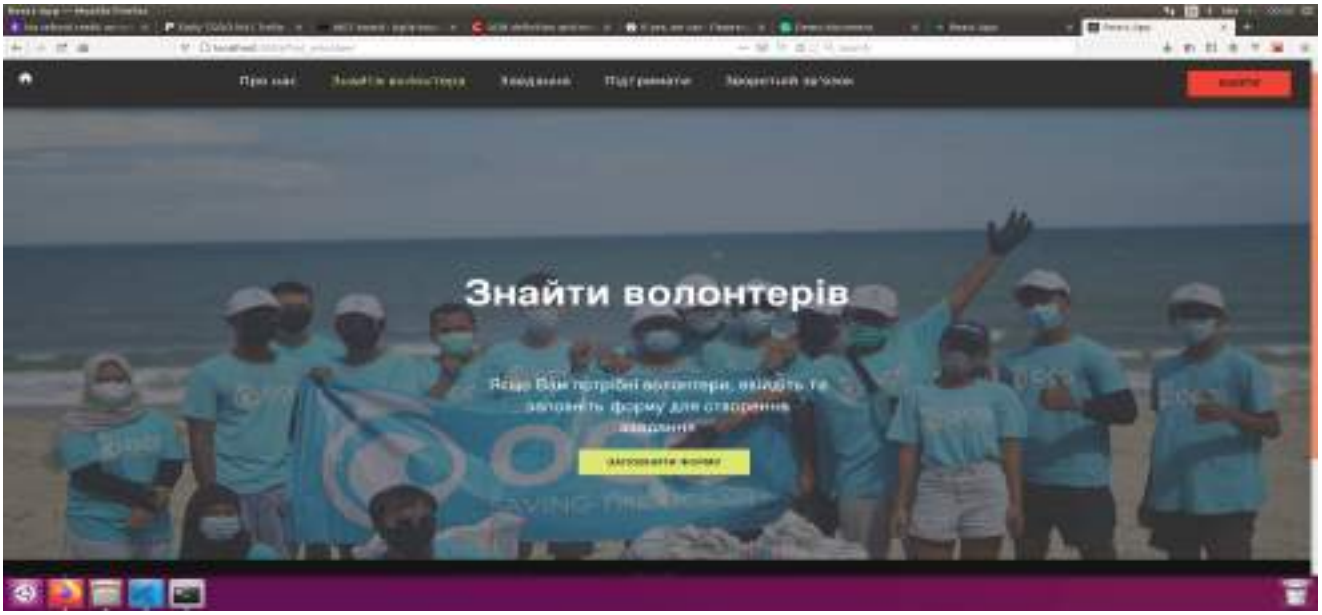

Рисунок 3.5 – Сторінка «Знайти волонтера»

На рисунку зображено відповідну сторінку пошуку волонтерів та створення завдань, але у другому стані платформи. Перший стан, коли користувач ще не авторизований на сайті, тоді сама кнопка міститиме інший текст і вестиме на іншу сторінку – входу в систему. Також ми можемо тут побачити частину футера і те, що меню є фіксованим відносно позиції на сторінці, тому користувач не втратить його з поля зору. А другий стан, коли користувач авторизований, натискання кнопки відкриває модальне вікно з формою для створення завдання (рис. 3.6). Також помітні зміни і в самому меню, пункти входу та реєстрації замінені на кнопку виходу з платформи. Окрім кнопки на сторінці містяться стандартні для сайту заголовок та підзаголовок.

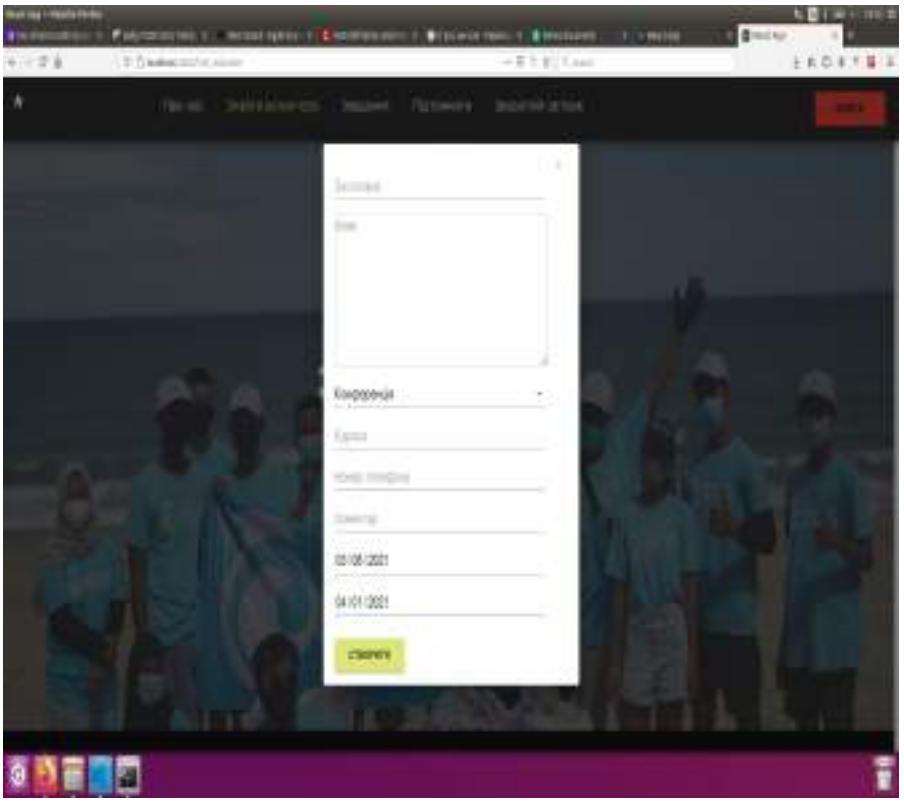

Рисунок 3.6 – Форма створення завдання

Форма розміщується у модальному вікні, яке крім форми містить кнопку закриття вікна. Дане модальне вікно затемнює фон та закривається після успішного підтвердження форми. Поля форми розроблено та розміщено згідно вимог, поля валідуються на обов’язковість та формат. На формі видно, що частина полів заповнені, а частина ні (заповнені поля типу та дати). Поля форми містять відповідні підказки (placeholder), які описують що це за поле. Форма підтверджується натисканням кнопки після цього дані відправляються на сервер і якщо завдання успішно створено, то відкривається інше popup вікно, яке про це інформує (рис. 3.7).

Рисунок 3.7 – Вікно оповіщення

Зображений popup є простим і містить тільки три елементи: заголовок, кнопку підтвердження, кнопку закриття. По суті кнопка підтвердження і закриття в даному випадку виконують таку ж функцію, а саме закриття вікна. Вікно зображене на рисунку оповіщає про успішне надсилання відгуку, або зворотного зв’язку, але аналогічні використовуються і на інших сторінках. Наступною в меню є сторінка перегляду завдань (рис. 3.8).

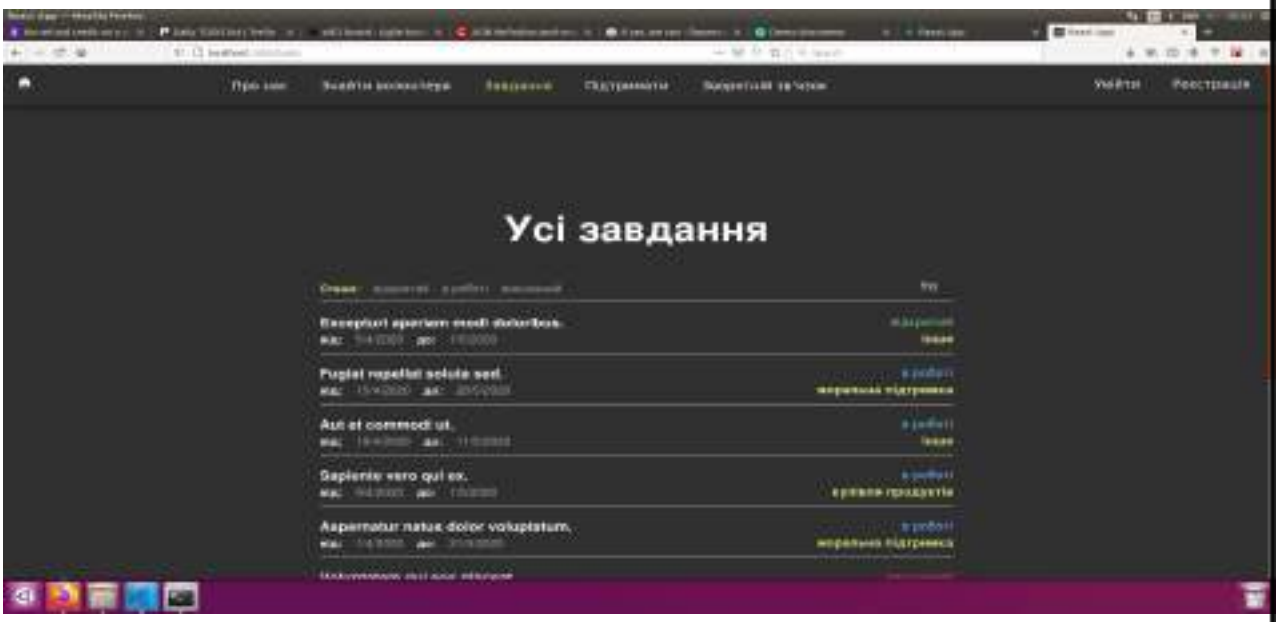

Рисунок 3.8 – Сторінка перегляду завдань

Сторінка показана у стані коли користувач не авторизований, тому відкрити завдання для перегляду детальнішої інформації є неможливим. Користувач може переглянути список завдань, кожне з яких містить наступну інформацію:

- Заголовок.
- Дата початку.
- Дата закінчення.
- Статус.
- Тип.

Кожен вид статусу виділений окремим кольором, це дозволяє користувачу звернути увагу на те чи може він взяти завдання в роботу чи воно вже виконане. Також реалізовано функціонал фільтрування завдань по статусу та типу, щоб спростити пошук волонтерам (рис. 3.9),

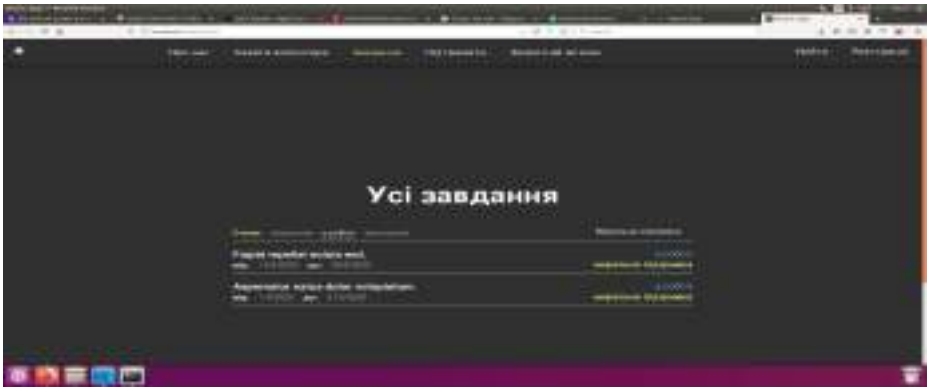

Рисунок 3.9 – Функція фільтрування завдань

На рисунку зображено фрагмент сторінки з відфільтрованими завданнями. Чітко видно, які фільтри обрані, стан фільтрується натисканням на елемент, а для фільтрування по типу, потрібно вибрати значення з випадаючого списку. Для фільтрування не потрібно натискати додаткових кнопок, це здійснюється динамічно при зміні фільтру. Щоб відмінити фільтрування, для стану потрібно натиснути на обраний фільтр ще раз, а для типу, обрати елемент «Усі». Обирати кілька значень одного фільтру наразі не можливо. Наступна сторінка, це сторінка «Підтримати» (рис. 3.10).

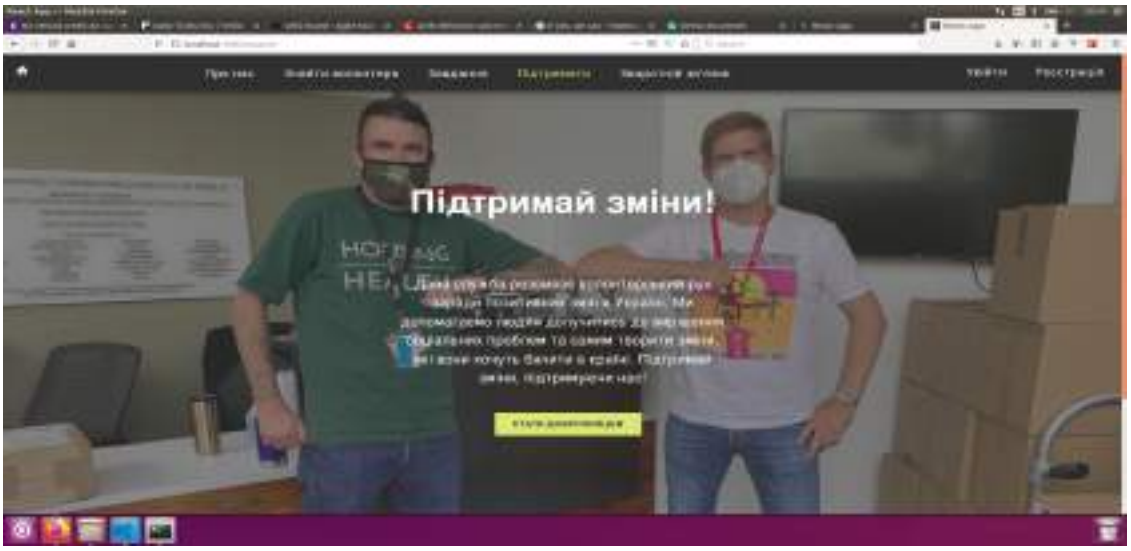

Рисунок 3.10 – Сторінка «Підтримати»

На рисунку видно, що це стандартна для платформи сторінка з заголовком та підзаголовком. Також додано кнопку для бажаючих підтримати організацію. Після натискання на кнопку користувача буде направлено на сторінку оплати. Сторінка не інтегрована в платформу і використовує просте посилання на оплату конкретній організації або на карту. Своїми враженнями або побажаннями користувачі можуть поділитись на сторінці «Зворотній зв’язок», яка є схожою за структурою з поточною сторінкою, але там розроблено форму для того, щоб залишити повідомлення (рис. 3.11).

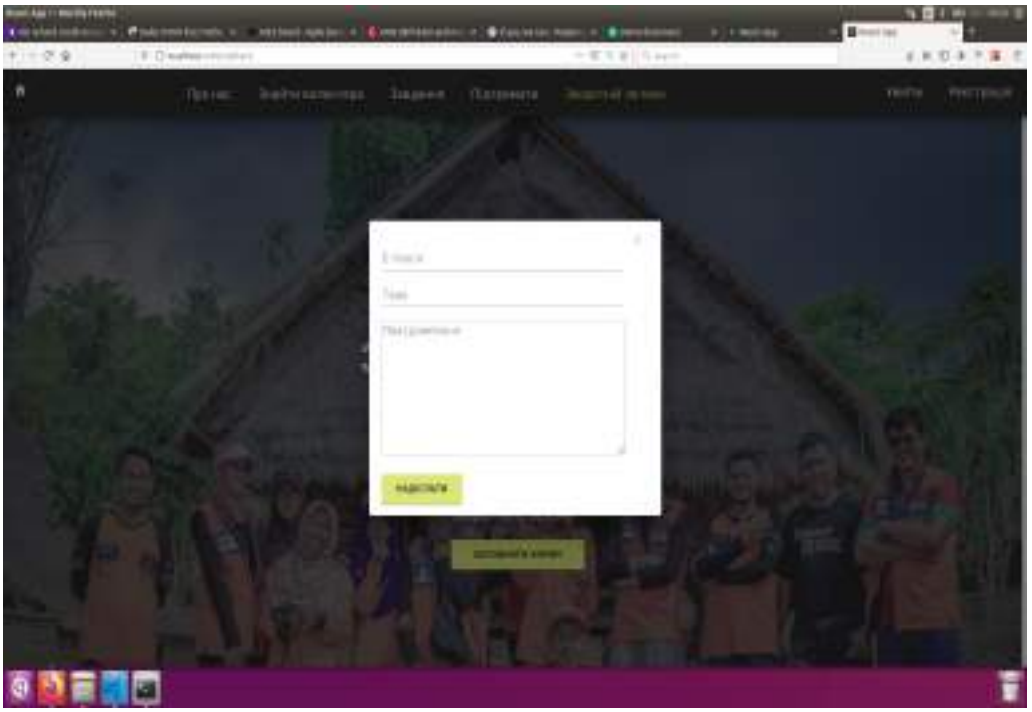

Рисунок 3.11 – Форма зворотного зв’язку

Форму розроблено з використанням popup вікна, яке як і інші закриває натисканням відповідної кнопки, а також при успішному надсиланні даних форми. Також типовим є те, що при успішному запиті з’явиться вікно, що інформуватиме про це. У форми є три поля: електронна пошта, тема, повідомлення та кнопка. Поля валідуються згідно з форматом та обов’язковістю кожного з них.

Останні дві сторінки загальнодоступного меню пов’язані з реєстрацією та авторизацією, логічно розглянути їх в порядку взаємодії користувача з системою, тому першою буде сторінка «Реєстрація» (рис. 3.12).

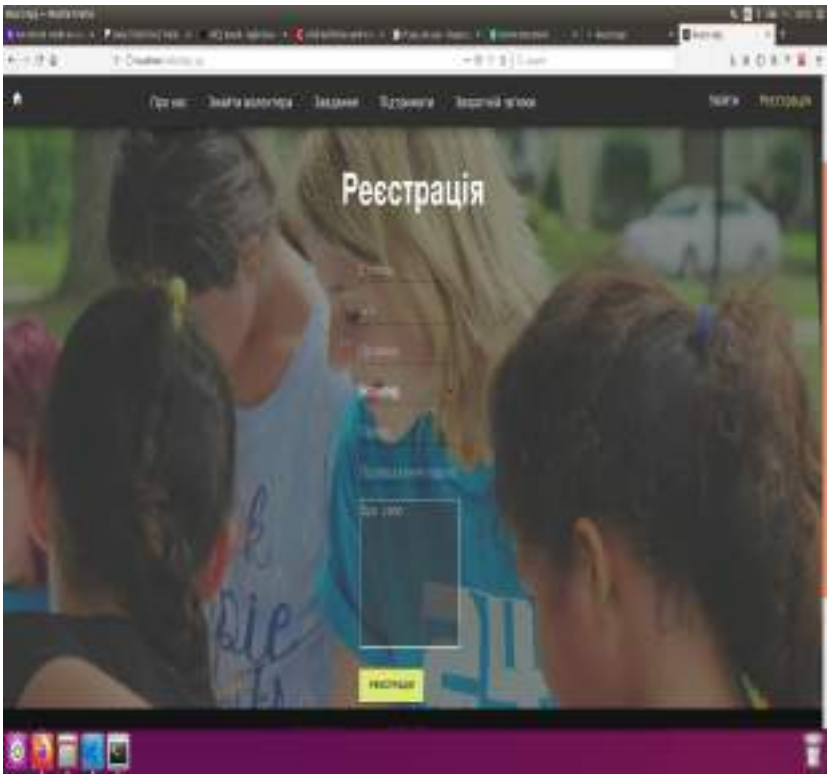

Рисунок 3.12 – Сторінка реєстрації

На рисунку зображено фрагмент сторінки реєстрації. В контенті сторінки розміщено два головних елементи, це форма та заголовок. В даному випадку як і на сторінці входу користувача в систему (рис. 3.13) форма відображається зразу, а не в окремому вікні так як це головні функції сторінки і тут не потрібна додаткова інформація. Форма містить поля різних типів, для окремих потреб системи під час реєстраці, наприклад поля типу password та email. З випадаючого списку можна обрати роль з якою користувач хоче зареєструватисть, як користувач, що потребує волонтерів або як волонтер. По замовчуванню вказано роль волонтера. Кожне поле містить підказку про його призначення. Також поля валідуються і реалізовано перевірку на спів падіння полів при підтвердженні паролю. Після успішного надсилання заявки на реєстрацію, користувачу відобразиться повідомлення про реєстрацію в системі. Коли користувача буде активовано він зможе увійти в систему.

Рисунок 3.13 – Фрагмент сторінки входу

На рисунку зображено сторінку входу, а саме форму входу. Форма містить два поля: електронну пошту та пароль. Якщо користувач успішно увійшов, то його перекине на сторінку, яка відповідає його ролі. Для звичайних користувачів це головна сторінка, а для адміністраторів це сторінка з адмінпанелі для керування завданнями (рис. 3.14).

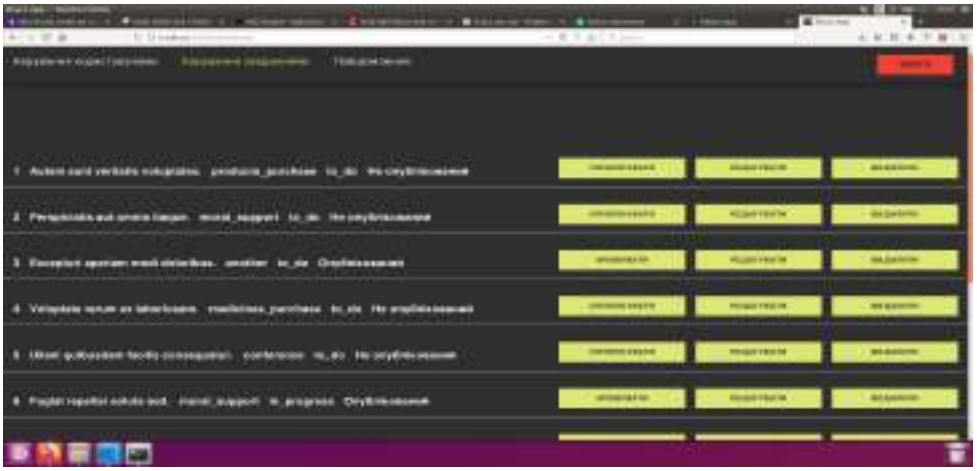

Рисунок 3.14 – Сторінка керування завданнями

Відповідно до сторінки зображеної на рисунку зрозуміло, що користувач авторизований, тому що наявна кнопка виходу. Також розроблений функціонал перевірки ролі відображає адміністратору інше меню. В даному меню тільки три пункти для адміністрації завдань, користувачів та зворотного з’вязку.

Завдання ж відображаються у вигляді списку. Кожен елемент списку

містить:

- Порядковий номер.
- Заголовок.
- Тип.
- Стан.
- Поле, що відображає чи завдання опубліковане.

Також містяться кнопки дій, для керування завданнями. Детальніше переглянути завдання адміністратор може при нас натисканні на кнопку «Редагувати», де може як і виправити щось так і просто прочитати інформацію. Дана форма схожа до форми створення завдання. Кнопка «Опублікувати –Архівувати» міняє текст у відповідності до стану публікації завдання. Кнопка видалення видаляє завдання, але перед тим появляється вікно, схоже на вікно успішного повідомлення, яке перепитує у користувача чи підтверджує він дію.

Решта сторінок та елементів дуже схожі до оглянутих як і функціоналом і структурою, так і виглядом, тому оглянуті сторінки сповна описують роботу розробленої платформи.

Платформа розроблена згідно поставлених вимог, розроблених макетів та архітектури REST API. Сповна реалізує потрібний функціонал та є зручною та швидкою у користуванні, оглянуті сторінки та функціонал це підтверджують.

Висновок до розділу 3

У підсумку ми розглянули реалізацію та огляд платформи, проаналізували розробку адмінпанелі, розробку панелі користувача. Розкрили основні функції та завдання платформи.

РОЗДІЛ 4. ОХОРОНА ПРАЦІ

4.1 Основні шкідливі фактори, що впливають на оператора ПК

Фізичні:

- підвищений рівень шуму на робочому місці (від вентиляторів блоку живлення процесорів та аудіоплат);
- підвищене значення напруги а електричному ланцюзі, замикання якого може статися через тіло людини;
- підвищений рівень статичної електрики;
- недостатня концентрація негативних іонів у повітрі робочої зони;
- підвищений рівень електромагнітного випромінювання;
- підвищена напруженість електричного поля;

Психофізіологічні:

- фізичні перевантаження статичної (опорно-м’язова система) та динамічної (кисті рук) дії;
- нервово-психічні перевантаження, перенапруження зорового аналізатора, розумове перенапруження, монотонність праці, емоційні перевантаження.

Для захисту від електромагнітних, електростатичних та інших полів можуть застосовуватися спеціальні технічні засоби, що мають відповідний сертифікат щодо їх захисних властивостей.

4.2 Освітлення приміщень

Робота користувачів комп'ютерів характеризується значним напруженням зорового аналізатора, тому виключно важливе значення має забезпечення раціонального освітлення робочих місць. Зоровий дискомфорт може бути викликаний:

- неправильною орієнтацією робочого місця відносно світлових отворів (вікон);
- неадекватними світловими характеристиками світильників (та/або) неправильним їх просторовим розташуванням відносно робочих місць;
- сліпучою дією яскравих предметів, що знаходяться в полі зору користувача (пряма близькість).

У забезпеченні максимально комфортних умов зорової роботи вагома роль належить оптимізації кількісних та якісних показників освітлення. Однак ці показники суттєво залежать від специфіки використання ПК. Якщо користувач постійно працює за ПК, то до такого робочого місця висуваються одні світлотехнічні вимоги.

Сприятливі умови роботи забезпечують як високу продуктивність праці, так і позитивно впливають на психологічний стан людини, на її працездатність і здоров'я. Особливо важливе біологічне і гігієнічне значення для людини має природне освітлення, тому при проектуванні виробничих приміщень важливо передбачити наявність відкритих природніх джерел освітлення для комфортних умов праці.

Освітлення в приміщеннях може здійснюватися природнім та штучним освітленням. При недостатньому природному освітленні використовують загальне освітлення. Останнє представляє собою освітлення, при якому водночас використовують природне і штучне освітлення. Загальним називають освітлення, світильники якого освітлюють всю площу приміщення. Місцевим

називають локальне освітлення, призначене для освітлення певного робочого місця.

При виробничій потребі дозволяється експлуатувати ПК у приміщеннях без природного освітлення за узгодженням з органами державного нагляду за охороною праці та органами і установами санітарно-епідеміологічної служби. В таблиці 4.1 наведені допустимі рівні освітленості.

Таблиця 4.1

Рівні освітленості у приміщеннях з ЕОМ

Виробниче приміщення	Розряд та під розряд зорової роботи	Освітленість, лк				Тип світильників
Приміщення для експлуатації ЕОМ	III в	Загальна	Комбінована	Аварійна	Евакуаційна	ЛПО-12 або ЛСП-12 «Кососвет» з лампами типу ЛД-40
		200	500	10	-	

НесприятливийвпливназоровуроботукористувачаПКможездійснюватидзеркальневідбиттянаекраніяскравихелементівнеправильнорозташованихсвітильників, або ділянок стелі чи вікна, на які подають сонячні промені. Такі дзеркальні відбиття при відносно невеликій яскравості екрана ПК, можуть викликати практично повну втрату контрасту зображення.

4.3 Електробезпека

Згідно стандартів, приміщення, де експлуатуються ЕОМ І ПЕОМ відносяться до приміщень без підвищеної небезпеки ураження людини електричним струмом. Устаткування для обслуговування, ремонту та

налагодження роботи їх, електропроводи і кабелі мають відповідати електробезпеці зони та мати апаратуру захисту від струму короткого замикання. Необхідно забезпечити неможливість виникнення джерела загорання внаслідок короткого замикання та перевантаження проводів шляхом переходу на негорючу ізоляцію. При одночасному використанні більше п’яти ПЕОМ на помітному місці встановлюється аварійний резервний вимикач, який в разі небезпеки повністю обезструмлює електричну мережу (крім освітлення). Електромережі для підключення ВДГ, ЕОМ і ПЕОМ оснащуються справжніми штепсельними з’єднаннями та електророзетками, які крім контактів фазового і нульового робочого провідників, мають спеціальні контакти для підключення нульового захисного провідника, що під’єднаний раніше ніж вони [26].

Штепсельні з’єднання або електричні розетки для напруги 12 і 36 В мають бути пофарбовані в колір, що відрізняється від їх кольору для напруги 127 і 220 В.

Розрахуємо горизонтальний заземлюючий пристрій. Вихідні дані:

1. Захищуваний об’єкт – комп’ютерний центр.

2. Напруга мережі – 220 В.

3. Виконання мережі – з глухо заземленою нейтраллю.

4. Тип заземлювального пристрою – горизонтальний зі стрічкової сталі.

5. Розміри заземлювачів і довжина стрічки $R_{\text{с}} = 55$ м, ширина стрічки $b_{\text{с}} = 0{,}04$ м.

6. Розташування заземлювачів – паралельне.

7. Відстань між паралельним заземлювальними стрічками $L_{\text{с}} = 5.5$ м.

8. Глибина закладання заземлювачів $h_{\text{з}} = 0{,}8$ м.

9. Ґрунт – суглинок.

10. Характеристика ґрунту: склад – однорідний; вологість – нормальна; агресивність – нормальна.

11. Кліматична зона – III.

12. Природні заземлювачі відсутні.

Визначаємо $R_{Д}$ – допустиме нормативне значення питомого опору, Ом, розтіканню струму в заземлювальному пристрої. Згідно з ПУЕ, ПТЕ та ПТБ $R_{Д} = 4$ Ом.

Визначаємо $\rho_{ТАБЛ}$ – наближене значення опору ґрунту $\rho_{ТАБЛ} = 100$ Ом.

Визначаємо $K_{С.Г}$ – коефіцієнт сезонності для ґрунту нормальної вологості при кліматичній зоні III для горизонтального заземлення довжиною $L_{с} = 5.5$ м. Тоді $K_{С.Г.} = 2$.

Розрахунковий питомий опір ґрунту для горизонтальних заземлювачів обчислюють за формулою 4.1:

$$\rho_{РОЗР.Г} = \rho_{ТАБЛ.} \cdot K_{С.Г.} \qquad (4.1)$$

Звідси:

$$\rho_{РОЗР.Г} = 100 \cdot 2 = 200 \text{ Ом.}$$

Визначаємо $R_{Г}$ – теоретичний опір розтікання струму в одному горизонтальному заземлювачі:

$$R_{Г} = 0{,}366 \cdot \frac{\rho_{РОЗР.Г}}{L_{С}} \lg \frac{2L_{С}}{h_{З} b_{С}}. \qquad (4.2)$$

Згідно формули 4.4:

$$R_{\Gamma} = 0{,}366 \cdot \frac{200}{55} \lg \frac{2 \cdot 55^2}{0{,}8 \cdot 0{,}04} = 8{,}35 \text{ Ом.}$$

Тепер визначаємо теоретичну кількість горизонтальних заземлювачів без врахування $\eta_{\text{в.г}}$ – коефіцієнта використання, тобто $\eta_{\text{в.г}} = 1$.

$$n_{\Gamma} = \frac{R_{\Gamma}}{R_{\text{Д}} \cdot \eta_{\text{в.г}}}. \qquad (4.3)$$

Звідси:

$$n_{\Gamma} = \frac{8{,}35}{4 \cdot 1} = 2{,}08.$$

В розрахунках кількість горизонтальних заземлювачів n_{Γ}=2шт.

Коефіцієнт використання паралельно вкладених горизонтальних заземлювачів при $n_{\Gamma} = 2$, $L_{\text{С}} = 55$ м, становить 0,78.

Розраховуємо необхідну кількість горизонтальних заземлювачів (стрічок), $n_{\text{Г.Н}}$

$$n_{\text{Г.Н}} = \frac{R_{\Gamma}}{R_{\text{Д}} \cdot \eta_{\text{в.г}}} = \frac{8{,}35}{4 \cdot 0{,}78} = 2{,}68.$$

Звідси $n_{\text{г.н}}$=3шт.

Тепер коли усі необхідні дані є відомі підставляємо дані в формулу 4.4:

$$R_{\text{РОЗР.Г}} = \frac{R_{\text{Г}}}{n_{\text{Г.Н}} \cdot \eta_{\text{В.Г}}}. \quad (4.4)$$

Звідси:

$$R_{\text{РОЗР.Г}} = \frac{8,35}{3 \cdot 0,78} \approx 3,6 \text{ Ом.}$$

Отриманий розрахунковий опір розтікання струму відповідає вимогам прописаних у державних стандартах.

Висновок до розділу 4

В цьому розділу проаналізовано шкідливі та небезпечні фактори в роботі за комп'ютером, було розглянуто на питання з безпечної експлуатації, пожежної безпеки.

ВИСНОВКИ

В кваліфікаційній роботі розроблено сайт, який забезпечує координацію роботи волонтерів у допомозі людям які потребують допомоги під час пандемії COVID-19. Також реалізовано адмінпанель для наповнення контенту та обробки заявок.

Дану систему розроблено відповідно до потреб потенційних користувачів.

Потреби користувачів будувались на основі порівняльного аналізу інформаційних ресурсів. Розроблено макети згідно яких створювався інтерфейс системи, було застосовано усі правила і обмеження побудови ефективних інтерфейсів.

Сайт розроблений за допомогою методів та технологій веб-розробки. З огляду вимоги специфікації, клієнтська сторона створена за допомогою бібліотеки реактивних інтерфейсів React, а серверна сторона за допомогою фреймворку Ruby on Rails. Архітектуру проекту було обрано на основі сучасних трендів та вимог до платформи. Елементи системи побудовано згідно з вимогами архітектурного підходу REST API.

Розробка сайту зумовила розширення знань в області створення веб-додатків. А саме розроблення архітектури з використанням новітніх стандартів, та побудову ефективних сервісів на базі актуальних технологій

Детально описано усі етапи розробки сайту, від планування архітектури та бази даних до створення макетів та інтерфейсів. Здійснено огляд функціоналу розробленої системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. CMS // [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Система_керування_вмістом.

2. Підручники HTML і CSS // [Електронний ресурс] – Режим доступу:

https://htmlbook.at.ua/.

3. CSS Tutorial // [Електронний ресурс] – Режим доступу: https://www.w3schools.com/css/default.asp.

4. MDN web docs - Основи CSS [Електронний ресурс] – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Learn/CSS_basics.

5. JavaScript // [Електронний ресурс] – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/JavaScript.

6. DOM Living Standard // [Електронний ресурс] – Режим доступу: https://dom.spec.whatwg.org/.

7. ECMAScript // [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/ECMAScript.

8. Посібник React // [Електронний ресурс] – Режим доступу: https://uk.reactjs.org/tutorial/tutorial.html.

9. Клієнт-серверна архітектура та ролі серверів // [Електронний ресурс] – Режим доступу: https://medium.com/@IvanZmerzlyi/клієнт-серверна-архітектура-та-ролі-серверів-9893d8048229.

10. Wikipedia. MVC // [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/модель-вид-контроллер.

11. The DCI Architecture: A New Vision of Object-Oriented Programming / James O. Coplien – W.: Artima, 2009. – 80 с.

12. Ruby is // [Електронний ресурс] – Режим доступу: https://www.ruby-lang.org/en/.

13. Imagine what you could build if you learned Ruby on Rails // [Електронний ресурс] – Режим доступу: https://rubyonrails.org/.

14. Завадський І. О. Основи баз даних / І. О. Завадський. – Київ: ПП І.О. Завадський, 2011. – 192 с.

15. Постановка задачи при проектировании сайта. – Режим доступу:

http://site-manager.ua/webmasters/lib/110/.

16. Організація баз даних: практичний курс: Навч. посіб. для студ. / А. Ю. Берко, О. М. Верес; Нац. ун-т «Львів. політехніка». — Л., 2003. — 149 с.

17. Гото К. Веб-редизайн / К. Гото, Э. Котлер – М.: Символ-Плюс, 2006. – 416 с.

18. The role of natural language in a multimodal interface / Cohen, Philip R. – UIST, 1992. – 180 с.

19. Структура веб-сайту // [Електронний ресурс] – Режим доступу: https://naurok.com.ua/urok-struktura-veb-saytiv-riznovidi-veb-saytiv-riznovidi-veb-storinok-33309.html.

20. Frequently Asked Questions // [Електронний ресурс] – Режим доступу: https://material-ui.com/getting-started/faq/.

21. Прототип сайту – як він допомагає оцінювати і розробляти сайти [Електронний ресурс] – Режим доступу до ресурсу: http://evergreens.com.ua/ua/articles/prototype-site.html.

22. Front-end та back-end // [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/Front_end_та_back_end.

23. Введение в REST API — RESTful веб-сервисы // [Електронний ресурс] – Режим доступу: https://habr.com/ua/post/483202/.

24. What is CRUD? // [Електронний ресурс] – Режим доступу: https://www.codecademy.com/articles/what-is-crud.

25. Getting Started with Redux // [Електронний ресурс] – Режим доступу: https://redux.js.org/introduction/getting-started.