

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування інституту, назва факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
бакалавр
(ступінь вищої освіти)

на тему ПРОГРАМНА РЕАЛІЗАЦІЯ ДИНАМІЧНОГО САЙТУ
ДЛЯ МАНДРІВНИКА.

SOFTWARE IMPLEMENTATION OF A DYNAMIC SITE
FOR THE TRAVELER.

Виконав: студент(ка) 4 курсу, групи КНТ-110сп
Спеціальності 121 Інженерія програмного
забезпечення
(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

БОНДАРЕНКО А.Г.
(прізвище та ініціали)

Керівник ЗЕЛЕНЬОВА І.Я.
(прізвище та ініціали)

Рецензент КАЗУРОВА А.Є.
(прізвище та ініціали)

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет ФКНТ

Кафедра програмних засобів

Ступінь вищої освіти бакалавр

Спеціальність 121 Інженерія програмного забезпечення

(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення

(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф .

Сергій СУББОТІН

« » 2023 року

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

БОНДАРЕНКО Антон Геннадійович

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмна реалізація динамічного сайту для мандрівника. Software Implementation of a Dynamic Site for the Traveler

керівник проєкту (роботи) к.т.н., доцент ЗЕЛЕНЬОВА Ірина Яківна

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від « 4 » квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 1 червня 2023 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області. 2. Розробка архітектури програми. 3. Основні рішення розробки. 4. Тестування інформаційного сайту

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЗЕЛЕНЬОВА І.Я., к.т.н., доц.		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання «17» квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи.	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області.	1 тиждень	Розділ 1
3	Вибір мови програмування та інших технологій	2 тиждень	Розділ 2
4	розробки.		
5	Розробка архітектури програми.	2 тиждень	Розділ 3
6	Розробка програми.	3-4 тижні	Розділ 3, 4
7	Тестування та експериментальне дослідження	5 тиждень	Розділ 4
	програмного забезпечення.		
	Оформлення пояснювальної записки та документів	6 тиждень	Додатки
	до неї.		
	Нормоконтроль та рецензування.	7 тиждень	
	Захист роботи.	8 тиждень	

Студент(ка)

_____ Антон БОНДАРЕНКО
(підпис) (Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

_____ Ірина ЗЕЛЕНЬОВА
(підпис) (Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра:
83 с., 43 рис., 1 табл., 2 дод., 23 джерел.

МІСЦЯ ПОДОРОЖІ, ВЕБСАЙТ, МАНДРІВНИК, АДАПТИВНА
ТЕМІЗАЦІЯ, REACT, SPA

Об'єкт дослідження – програмні засоби для сайту «Мандрівник».

Предмет дослідження – комп'ютерні програми для перегляду місця подорожі, пам'яток та інших об'єктів.

Мета роботи – створення програмної системи для інтерактивного перегляду місця подорожі, пам'яток та інших об'єктів з додатковим функціоналом.

Матеріали, методи та технічні засоби: структурне та об'єктно-орієнтоване програмування, Visual Studio Code, мови програмування Javascript, Typescript, Node js, Html, CSS, персональний комп'ютер з процесором Intel Core i5 під управлінням операційної системи Microsoft Windows 11.

Результати. Створено програмний вебсайт, використовуючи реактивний підхід за рахунок бібліотеки React.

Висновки. Розроблено вебсайт для перегляду місця подорожі, пам'яток та інших об'єктів.

Галузь використання – для організаторів приватних туристичних поїздок, для туристичних агенцій, як додатковий сучасний цифровий інструмент для мандрівника.

ABSTRACT

Explanatory note to the bachelor's thesis: 83 pages, 43 figures, 1 table, 2 appendices, 23 sources.

TRAVEL PLACES, WEBSITE, TRAVELER, ADAPTIVE THEMING, REACT, SPA

The object of the research is software for the "Traveler" website.

The subject of research is computer programs for viewing travel destinations, sights and other objects.

The purpose of the work is to create a software system for interactive viewing of travel destinations, sights and other objects with additional functionality.

Materials, methods and technical means: structural and object-oriented programming, Visual Studio Code, programming languages Javascript, Typescript, Node js, Html, CSS, a personal computer with an Intel Core i5 processor running the Microsoft Windows 11 operating system.

The results. A software website was created using a reactive approach using the React library.

Conclusions. A website has been developed for viewing travel destinations, attractions and other objects.

The field of use is for organizers of private tourist trips, for travel agencies, as an additional modern digital tool for the traveler.

ЗМІСТ

	С.
ПЕРЕЛІК СКОРОЧЕНЬ ТР УМОВНИХ ПОЗНАК	8
ВСТУП	9
1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ.....	10
1.1 Аналіз переваг та цікавих вирішень аналогів.....	10
1.2 Аналіз інструментів для розробки	10
1.3 Технічна підготовка до завдання	11
2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЕКТУЄТЬСЯ, СТВОРЕННЯ ПЛАНУ РОЗРОБКИ.....	12
2.1 Вибір середовища розробки	12
2.2 Вибір фреймворка	13
2.3 Структура та функціонування програми	16
2.3.1 Структурна схема сайту	16
2.3.2 Схема функціонування програми	19
2.4 Створення плану розробки.....	20
3 РОЗРОБКА ОСНОВНИХ ПРОГРАМНИХ ЧАСТИН ВЕБСАЙТУ «Мандрівник».	23
3.1 Підготовка до початку розробки.....	23
3.1.1 Встановлення всіх залежностей	23
3.1.2 Створення файлової структури	24
3.2 Реалізація вебсайту «Мандрівник»	28
3.2.1 Головна сторінка.....	28
3.2.2 Сторінка списку місць та фільтрів для них	29
3.2.3 Сторінка з інформацією про знайдене місце	29
3.2.4 Організація динамічних стилів, світлий стиль та темний стиль	31
3.2.5 Сторінки для реєстрації, входу та відновлення пароля	32
4 ТЕСТУВАННЯ ТА ДОСЛІДЖЕННЯ ВИГОТОВЛЕНОЇ ПРОГРАМИ.....	35

4.1 Тестування UI елементів	35
4.2 Тестування у Google Chrome Dev tools	36
4.3 Рекомендації що до використання сайту «Мандрівник»	37
ВИСНОВКИ.....	39
ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ.....	40
ДОДАТОК А – Програмний код компонентів сайту	43
ДОДАТОК Б – Слайди презентації	73

ПЕРЕЛІК СКОРОЧЕНЬ ТР УМОВНИХ ПОЗНАК

API	– Application Programming Interface;
CSS	– Cascading Style Sheets
DOM	– Document Object Model
HTML	– HyperText Markup Language
IDE	– Integrated Development Environment,
JS	– Java script
NPM	– Node Package Manager
SPA	– Single Page Application
UI	– User Interface
WEB	– World Wide Web;

ВСТУП

Сучасною реальністю є широке впровадження у всі сфери життєдіяльності сучасних інформаційних технологій, розгортання на їх основі різноманітної складності локальних і глобальних інформаційних систем, призначених для прискорення обміну інформацією та доступу до різноманітних інформаційних джерел.

За останні декілька десятиків років завдяки бурхливому розвитку Інтернету в програмуванні виділяють окремий напрямок – вебпрограмування, тобто створення скриптів для вебсайту, так як в час стрімкого прогресу просто красиво оформлений текст і картинки на вебсайті вже нікого не здивують. Та й вимоги до сайтів дуже змінилися, і тепер для успішного представництва організації або особи в Інтернеті необхідно надати своїм відвідувачам різні можливості: контактні форми, каталоги продукції, підписки, розсилання, пошук по сайту й багато чого іншого.

Подорож в іншу країну або пошук нових місць для відвідування в своїй країні завжди потрібно для мандрівників. Швидкий пошук, простий і зручний інтерфейс, легкий у розумінні користувача з достатньою кількістю інформації є преорететним у додатку.

Тому реалізація сайту «Мандрівник» є актуальною, поєднавши зручний та красивий інтерфейс, де користувач може знайти інформацію про місце подорожі: ресторани, місця для відвідувань, місця відпочинку, музеї та ін. Додаткова корисна інформація для користувача з огляду на опис місця, фото, посилання на більш детальну інформацію, коментарі та оцінки інших користувачів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Аналіз переваг та цікавих вирішень аналогів

В інтернеті велика кількість сайтів котрі допомагають мандрівникам: допомога в купівлі квитків, допомога в оренді житла, форуми з відгуками, які допомагають побудувати туристичні маршрути і найбільший помічник - це сайти карти.

Найкращий приклад аналогів сайтів – це Google карти [1]. Вони також дають користувачам широкий спектр інформації про місце подорожі. Вони використовують також інші сервіси для надання великого спектру послуг користувачам. Але насамперед вони все одно переважно використовуються як карти, хоч і мають величезний функціонал напрацьований часом. З додаткових сервісів, які використовують карти, що також є помічниками для мандрівників: Booking [2], сервіси для замовлення авіаквитків [3] та ін . Також є можливість перегляду фотографій місць, можливість залишити відгуки, а також функціонал додавання та редагування місць іншими користувачами.

Було важливим зрозуміти структуру сайтів аналогів, їх плюси та мінуси. За рахунок них можна зрозуміти які технології вибрати, який дизайн і структуру сайту використовувати, а також сервіси, які доповнять сайт - щоб збільшити переваги над аналогами.

1.2 Аналіз інструментів для розробки

Аналіз включає перегляд інструментів сайтів аналогів та їх вирішення у верстці сторінок, підбір стеку технологій та технічної підготовка до завдання.

Написання сайту тільки на технологіях: Html, Css, Js зараз не актуальний, сайти які перезавантажують сторінку втратили актуальність, так як є жахливою практикою для розробника:

- коли додаток почне розширюватися – подібна техніка завдасть проблем майбутнім розробкам;
- повторюваний функціонал для схожих блоків у яких відрізняється лише мала частина;
- велика затратність ресурсів при багатосторінковому додатку;
- спрощення взаємодії сайту з backend сервером, швидше оновлення даних без перезавантаження сторінки.

З цього зрозуміло, що стек технологій потребує сучасних вирішень, а саме вибір фреймворку для SPA додатків, що підходить для всіх рішень сайту “Мандрівник”.

1.3 Технічна підготовка до завдання

Сайт «Мандрівник» потребує у таких технологіях:

- динамічні зміни даних та їх централізоване зберігання;
- фреймворк який не вимагає перезавантаження сторінок для багатосторінковості, отримання, відправлення, оновлення даних;
- технологія відправлення та отримання даних через API;
- можливість перевикористання компонентів або темлейтів;
- бібліотеки темізації які підійдуть фреймворку.

2 ВИБІР ТА ОБҐРУНТУВАННЯ СТРУКТУРИ СИСТЕМИ, ЯКА ПРОЕКТУЄТЬСЯ, СТВОРЕННЯ ПЛАНУ РОЗРОБКИ

2.1 Вибір середовища розробки

Досить багато IDE для веброзробок і кожне має свої плюси та мінуси, було виділено кілька:

- Visual Studio Code – редактор вихідного коду розроблений Microsoft [4] для Windows, Linux і macOS, є безкоштовним і має безліч розширень для зручності та підтримки розробки. З мінусів тільки те, що вимагає попереднього налаштування та установки відповідних розширень для розробки;

- Sublime Text – текстовий редактор [5], розроблений Джоном Скіннером, доступний для Windows, Linux та macOS, є частково безкоштовним рішенням, проте є платна ліцензія з доступом для більшого пакета розширень для нього;

- WebStorm – інтегроване середовище розробки на JavaScript [6], CSS та HTML від компанії JetBrains, розроблене на основі платформи IntelliJ IDEA. Доступна для Windows, Linux і macOS, є абсолютно платним рішенням, але з усіма потрібними розширеннями і налаштованими спеціально для веброзробки. Основний плюс - майже не вимагає жодних налаштувань для початку розробки на ньому;

Був обраний Visual Studio Code – з головних його плюсів це:

- офіційне безкоштовне IDE;
- підтримка множини розширень для вільної розробки;
- зручність та варіативність налаштувань;
- зручна робота з терміналом;
- зручне налаштування стилів коду;
- зручна взаємодія та робота з git системою;

2.2 Вибір фреймворка

Фреймворк – програмне забезпечення, що полегшує розробку та об'єднання різних компонентів великого програмного проекту. Основний сенс сучасних вебдодатків, сайтів – реактивність. Її мета надати високу швидкість, простоту та масштабованість проекту. Найпопулярніші та які використовуються: React, Vue 2, Angular. Порівняння яких представлено в табл. 2.1.

Таблиця 2.1 – Порівняння фреймворків

	React	Vue 2	Angular
Реактивність	Оновлення відбувається за рахунок перебудови React DOM і порівняння його попереднього стану DOM, для написання компонентів використовується Jsx синтаксис [7].	Працює на основі ViewModel, для оновлення стану компонентів для мінімізації кількості ререндерів, написання компонентів за рахунок темлейтів та вбудованого додаткового функціоналу від фреймворку [8].	Працює за принципом Model-View-Controller, структурування йде на Модулі, Компоненти та Сервіси, для написання компонентів використовуються шаблони з окремим файлом функціоналу та з можливістю імпорту його в інші компоненти [9].
Екосистема	Існує безліч бібліотек для реакцій, які доповнюють його функціонал, полегшують розробку та можливості зручного розширення програми [7].	Фреймворк з'явився пізніше ніж два інші фреймворки, але він орієнтувався на всі їхні плюси та мінуси. Майже всі основні бібліотеки для React, є і для Vue [8].	Має пристойну кількість різноманітності бібліотек, але через меншу популярність фреймворку їх не така велика різноманітність як у інших [9].

Продовження таблиці 2.1

	React	Vue 2	Angular
Typescript	Опціональний	Опціональний, але використовується в проектах дуже рідко	Написаний на ньому
Навчання та популярність	Реакт із переходом на функціональні компоненти та на хуки став більш простим у використанні та розумінні, але все одно синтаксис та деякі особливості залишаються складними для новачків. Проблема полягає в не найкращій документації [7].	Простий у розумінні для новачків та з дуже добре розписаною документацією для освоєння базових знань і навіть більше [8].	Складний для новачків, перші кроки навчання не даються легко. Документація для фреймворку не найкраща, а так само проблемою спочатку є вбудований Typescript, який не відразу легко дається в розумінні [9].

Також хотілося б виділити порівняння фреймворків у продуктивності: React (рис. 2.1), Vue (рис. 2.2), Angular (рис. 2.3) [10].

Перша прорисовка контенту показує, що Vue та React краще справляються з провантаженням і рендерингом сторінки, ніж Angular, якому зазвичай потрібно більше часу, щоб підготувати сторінку.

Затримкою першого введення (3 пункт) є тимчасовий проміжок між процесом користувача (клік на кнопку, введення даних) і реакцією сторінки на подію, на щастя, усі 3 фреймворка показують винятково позитивні результати за цим параметром [3].

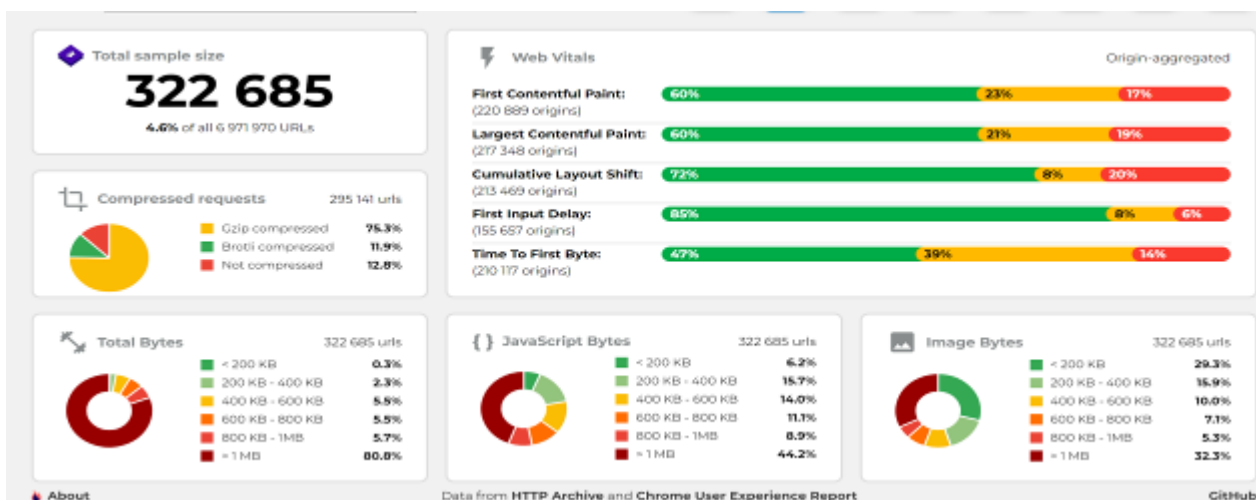


Рисунок 2.1 – Статистика фреймворку React на сайті perf-track.web.ap

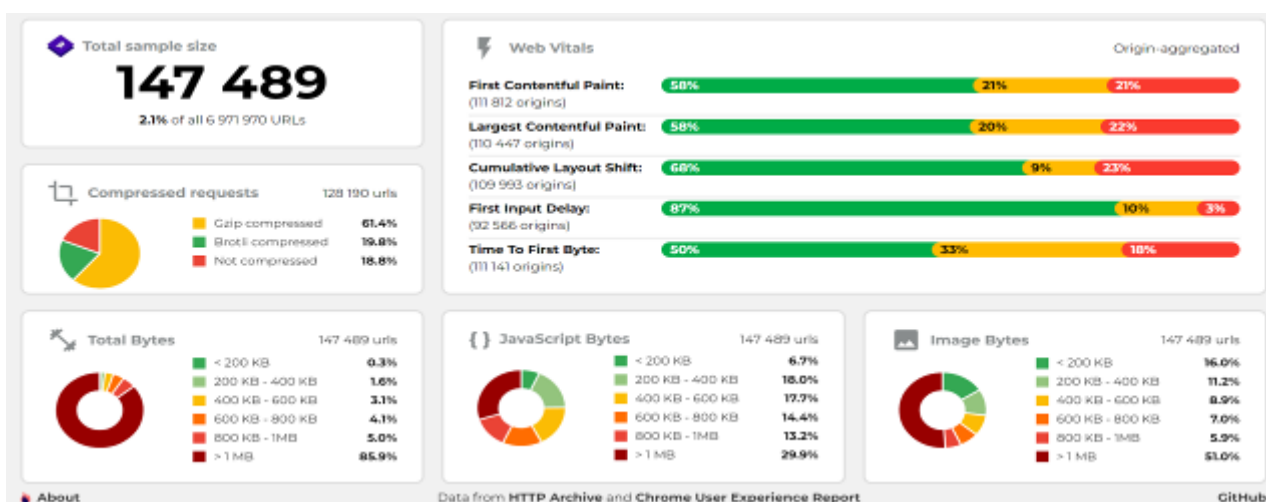


Рисунок 2.2 – Статистика фреймворку Vue на сайті perf-track.web.ap

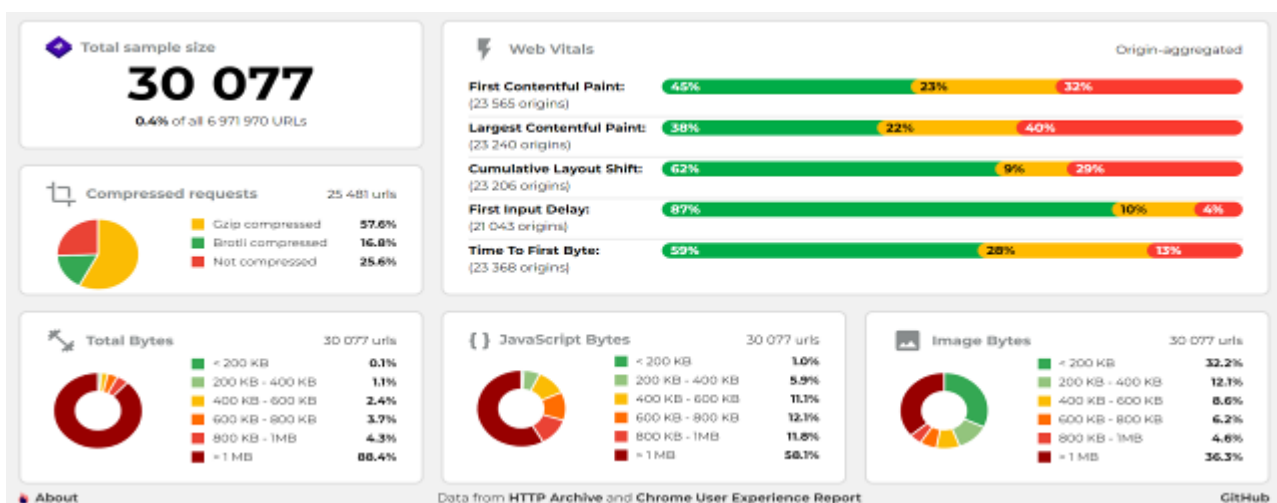


Рисунок 2.3 – Статистика фреймворку Angular на сайті perf-track.web.ap

Також варто зауважити, що понад 70% додатків на Vue завантажують менше 1MB JavaScript для своєї роботи, тоді як інші 2 фреймворки та Angular зокрема зазвичай займають набагато більше пам'яті. Однак, потрібно розуміти, що це може означати, що більшість маленьких проєктів написані на Vue, тоді як Angular використовується для більш потужних проєктів [3].

Було вирішено використовувати React, бо у нього є велика кількість бібліотек з функцілом для створення реактивних вебсторінок. Також вирішено використовувати Typescript [11] для повної типізації проєкту, спрощення подальшої масштабованості проєкту, також він зручно вбудовується у React бібліотеку [7].

Для backend було обрано фреймворк Nest js, заснований на Node js з додаванням Typescript [11], що значно спрощує розробку разом з Frontend. Також Nest js був обраний через відмінний інтерфейс, велику кількість допоміжних бібліотечних файлів для підключення тільки необхідного функціоналу, зручного структурування папок та обробки API-інтерфейсів. Також плюсом те, що він створений на основі Node js, що збільшує роботу з ним, оскільки ресурсів і книг для нього велика кількість [12, 13].

2.3 Структура та функціонування програми

Структура програми – це перший план, як виглядатиме ваш сайт. Структура покаже зразкову взаємодію з користувачем, розміщення функціональних блоків та їх зразкову взаємодію. Було використано сервіс Diagrams [14].

2.3.1 Структурна схема сайту

За схемою головна сторінка (рис.2.4): позиція 1 – шапка сайту, в ній буде розміщуватися логотип і кнопки "Увійти" або кнопка "Профіль", так само буде

кнопка перемикання нічного режиму сайту; позиція 2 – пошукова панель в якій буде проводитись пошук локації; позиція 3 – слайдер іконок категорій місць, користувач може вибрати одну категорію або перейти без вибору до списку; позиція 4 – кнопка отримання геолокації користувача; позиція 5 – картки категорій місць знайденої локації.

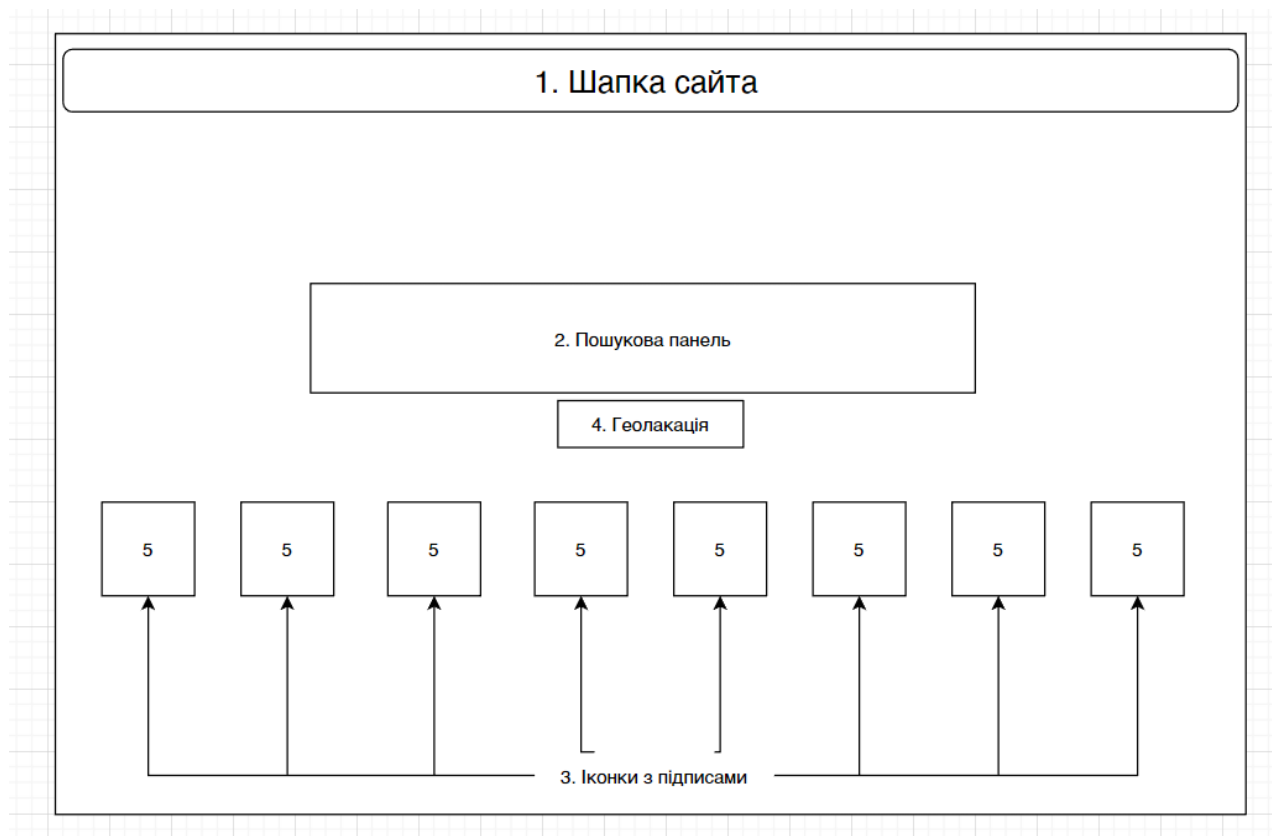


Рисунок 2.4 – Сторінка місць знайденої локації

За схемою сторінка зі списком місць (рис.2.5): позиція 1 – найменування шуканої локації; позиція 2 – список місць, що знайшли сайт за заданими фільтрами; позиція 3 – пагінація списку місць; позиція 4 – віджет прогнозу погоди; позиція 5 – блок фільтрів для місць.

За схемою сторінка місця знайденої локації (рис.2.6): позиція 1 – блок з фото локації якщо така є в базі даних; позиція 2 – інформаційне поле місця якщо данна інформація є в базі даних; позиція 3 – блок карти з точною вказівкою місця на ній; позиція 4 – блок з посиланнями на більшу інформацію о місці.

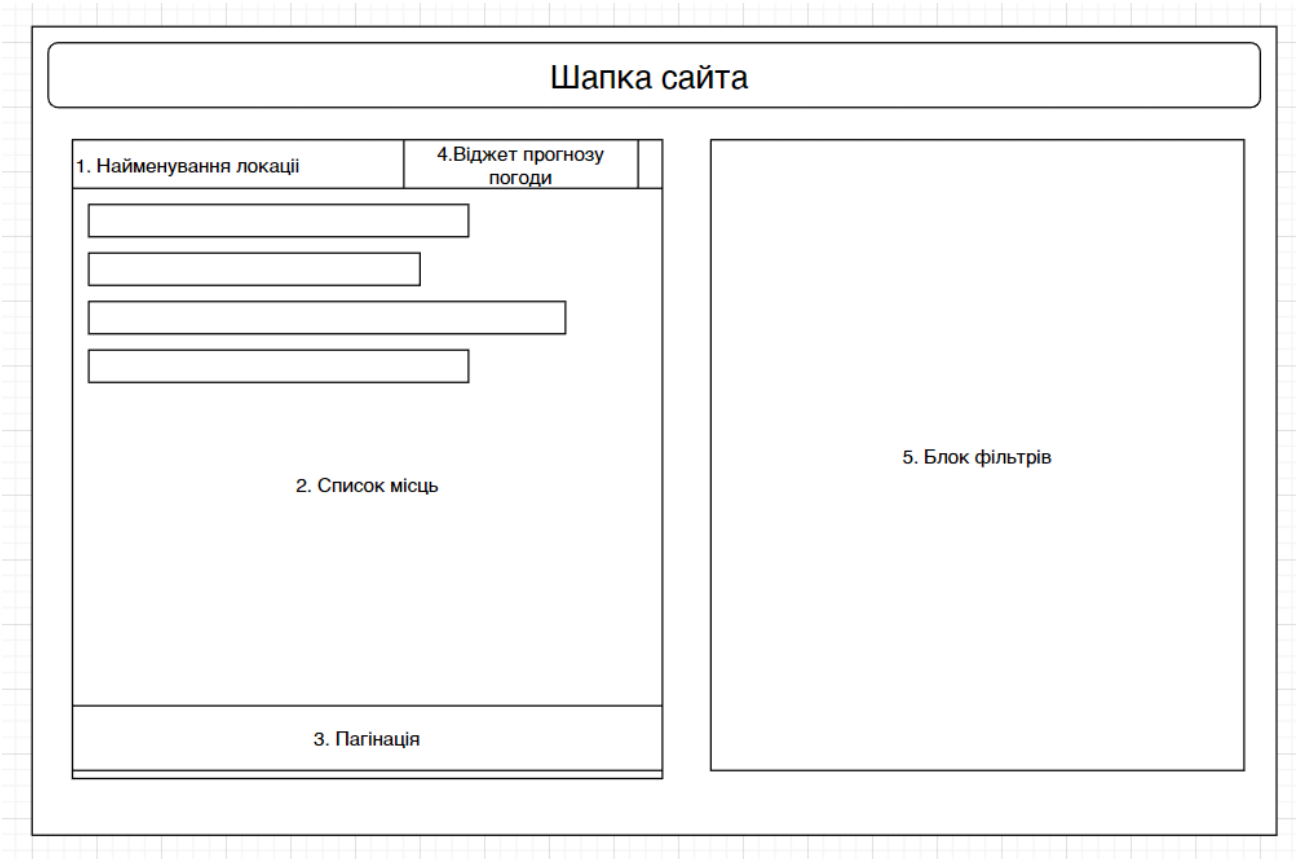


Рисунок 2.5 – Сторінка зі списком місць

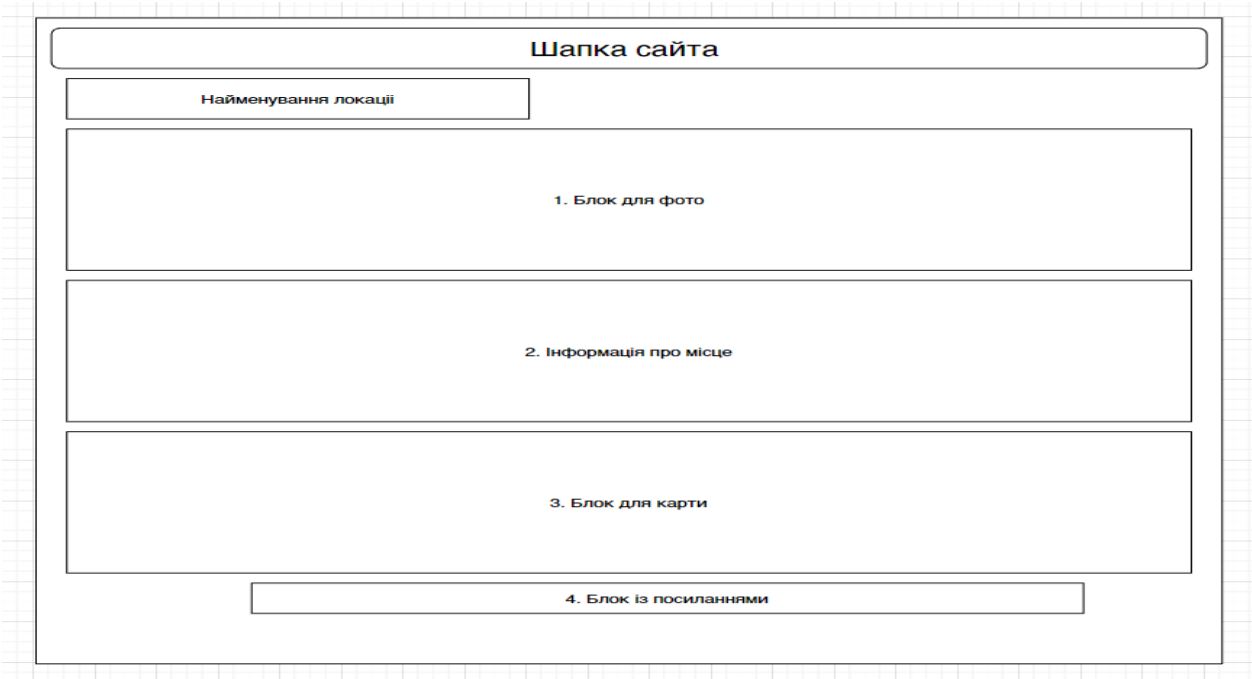


Рисунок 2.6 – Сторінка місця знайденої локації

За схемою сторінка Сторінка входу, реєстрації, відновлення пароля (рис.2.7): позиція 1 – логотип; позиція 2 – поля введення інформації; позиція 3 – кнопка підтвердження.

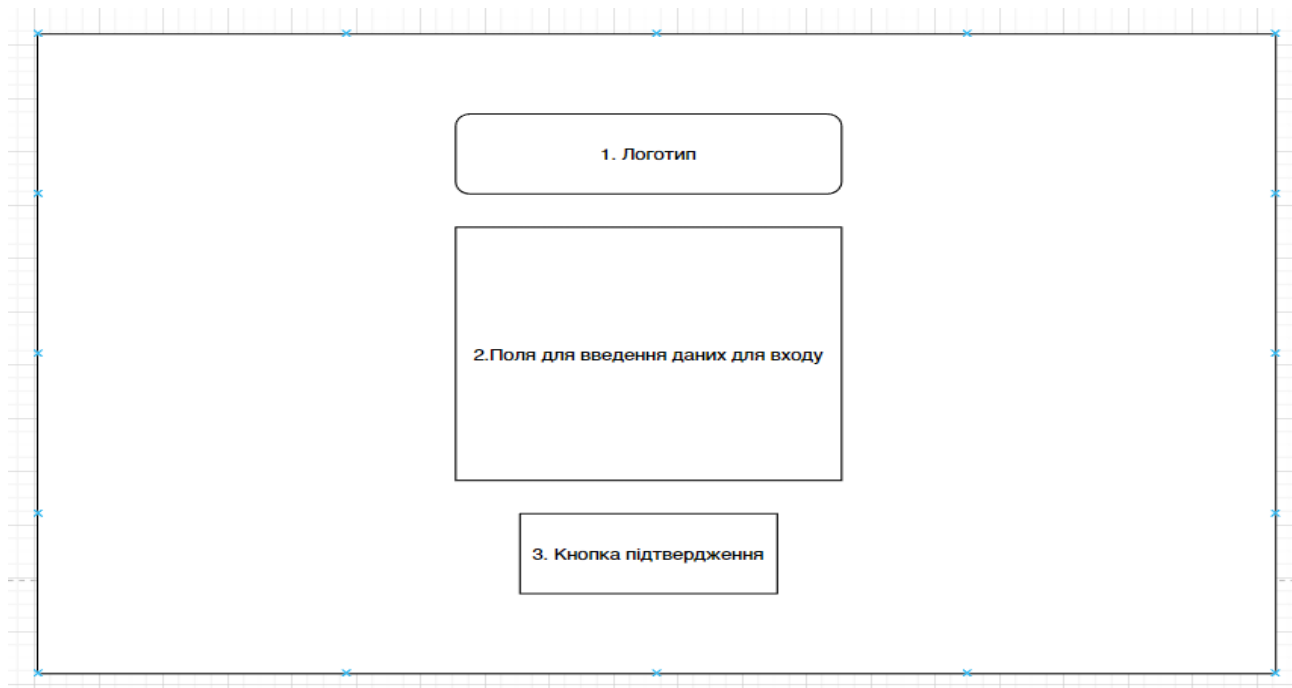


Рисунок 2.7 – Сторінка входу / реєстрації / відновлення пароля

2.3.2 Схема функціонування програми

Перед початком виконання потрібно продумати функціональність сайту.

У рис. 2.8 представлено основну схему функціональності сайту:

- відкриття сайту;
- пошук локації або отримання геолокації користувача;
- знаходимо данні о локації;
- відображаємо категорії знайдених місць;
- далі користувач переходить на сторінку з місцями, за допомогою фільтрів шукає потрібне місце;

- зі списку користувач вибирає потрібну локацію та переглядає інформацію про неї: фото, коротка інформація, розташування на карті та посилання на докладну інформацію, коментарі інших користувачів;
- якщо користувач увійшов на сайт, він може залишити відгуки та оцінку місцю.

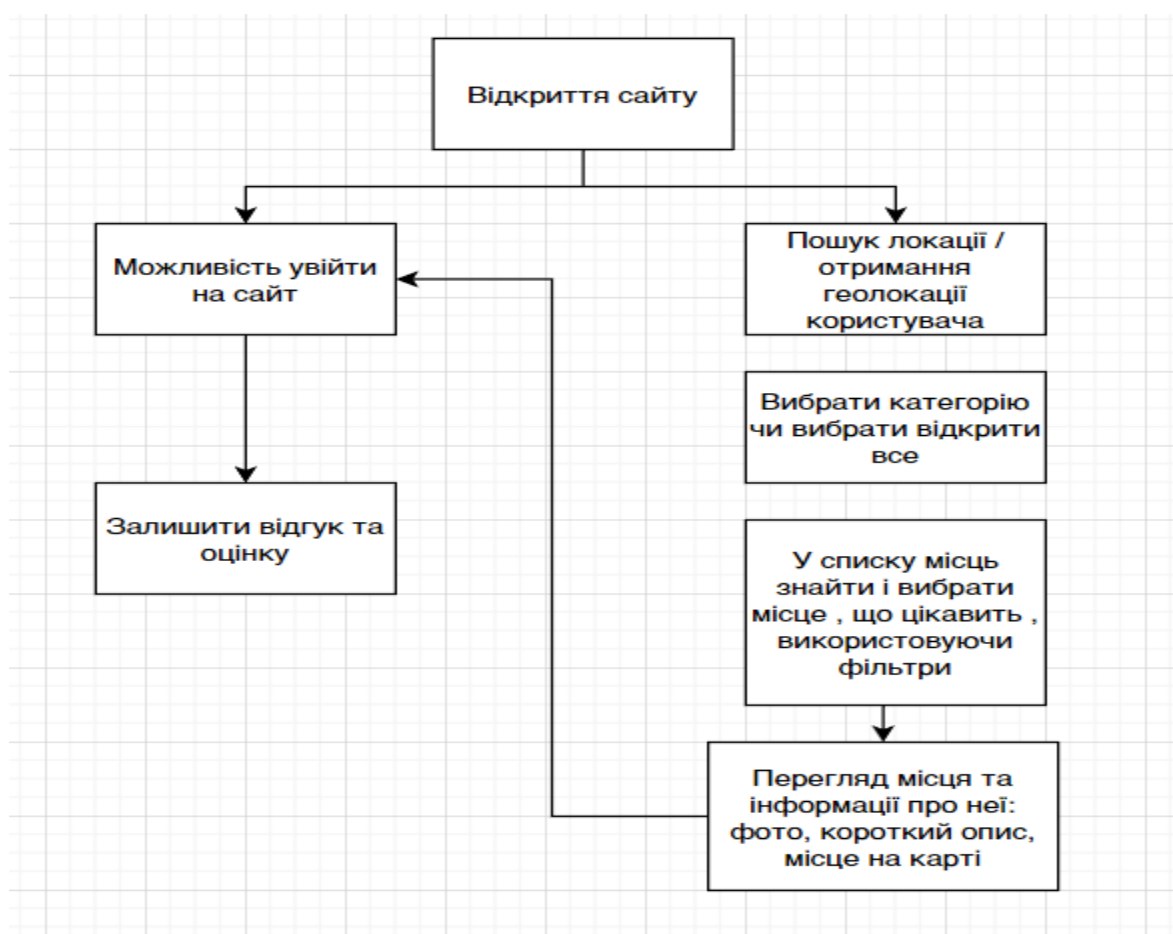


Рисунок 2.8 – Схема функціонування програми

2.4 Створення плану розробки

Для спрощення розробки потрібно складання чіткого плану, процесу розробки – поставлення завдання так, щоб не доводилося повертатися робити якісь виправлення, додавання контенту, зміни у файловій структурі.

Вибір великий але ж було обрано – Jira [15], комерційна система відстеження помилок, призначена для організації взаємодії з користувачами та управління проектами. Процес складання плану:

- повний опис завдання для програми (Рис.2.9);
- розподіл по окремих маленьким завданням (Рис.2.10);
- створення опису кожного маленького завдання;
- розставлення пріоритетів (Рис.2.10).

Дані про місця брати з арі - [OpenTripMap API](#)

- Отримання даних про місця шуканої локації користувачем.
- Пошук по будь-якому місту світу

Локації повинні мати коротку інформацію про себе:

- Фото
- коротка інформація
- Відображення розташування на карті
- Посилання на детальну інформацію про місце

Особливості:

- Можливість повної фільтрації місць локації
- Можливість зміни теми сайту
- Збереження обраних локацій та місць
- Короткий прогноз погоди про місце

Рисунок 2.9 – Повний опис задачі

Канбан дошка [16] (Рис.2.10) потрібна для чіткого розподілу завдань і простого контролю їх виконання, так само надалі це використовується для поліпшення спільної роботи в компаніях, дотримання часу виконання та розуміння точного закінчення ключових завдань, звичайних, багів і т.д., можливістю вибору їхньої пріоритетності.

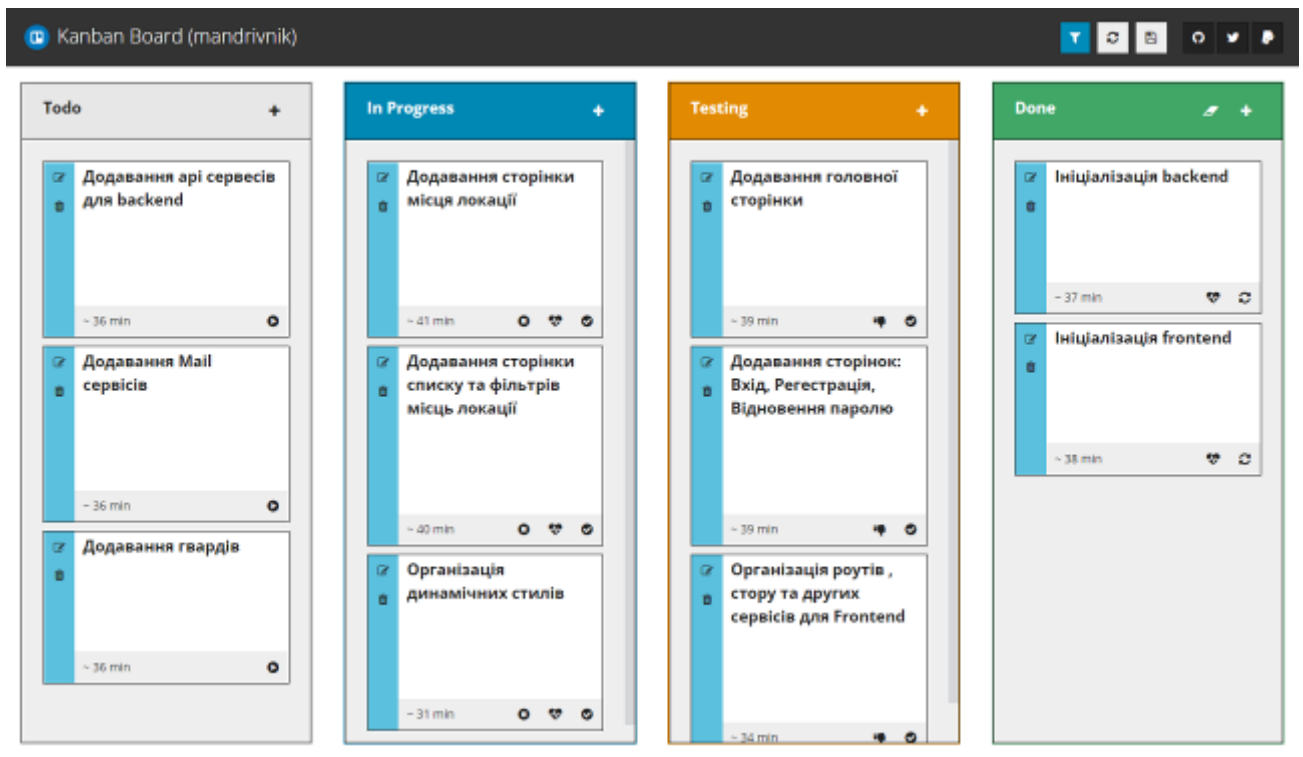


Рисунок 2.10 – Канбан доска для сайту Мандрівник

Також використання канбан дошки допомагає чітко відстежувати виконання поставлених завдань і часу витрачених на них, так як там так само вбудований трекер і можна для кожного завдання встановити чіткий час виконання – скільки вона вимагає.

3 РОЗРОБКА ОСНОВНИХ ПРОГРАМНИХ ЧАСТИН ВЕБСАЙТУ «МАНДРІВНИК».

3.1 Підготовка до початку розробки

На початку розробки потрібно встановити всі залежності, створити початкову структуру файлів, додати всі необхідні початкові стилі, та файли налаштування. Використовуючи React є можливість використовувати функціонал створення програм create-react-app з попередніми налаштуваннями, залежностями та деякими файлами. Для backend Nest також є можливість швидкого встановлення та створення проекту – nest-new. Після цього було додано git систему.

3.1.1 Встановлення всіх залежностей

Для frontend використовуючи можливість create-react-app, частина залежностей вже буде встановлена, але основна частина вимагає установки через команди npm [17]:

- Axios, для спрощеної роботи з API [18];
- Classnames, для спрощеної роботи з класами;
- Date-fns, для форматування та спрощеної роботи з часом;
- Node-sass, це метамова на основі CSS, призначена для збільшення рівня абстракції CSS-коду та спрощення файлів каскадних таблиць стилів;
- Normalize.css, для нормалізації та скидання стандартних стилів;
- React-dnd, для спрощеної роботи із системою drag and drop;
- Swapper, бібліотека slider – для покращення взаємодії й показу фото;
- React-redux [19], бібліотека для роботи з компонентною моделлю React. З ним компоненти автоматично оновлюється за необхідності. Надає API, які дозволяють вашим компонентам взаємодіяти зі сховищем Redux, тому не

потрібно писати цю логіку самостійно. Автоматично реалізує складну оптимізацію продуктивності, тому власний компонент виконує повторну візуалізацію лише тоді, коли необхідні дані дійсно змінилися. Для доповнення було встановлено Redux-toolkit [20], який спрощує роботу та дає нові можливості для Redux;

- React-router-dom, дає можливість працювати програмі з маршрутизацією [21];
- Styled-components, бібліотека, яка дає можливість Використовуючи позначені літерали шаблонів і потужність CSS, styled-components дозволяє вам писати фактичний код CSS для стилізації ваших компонентів всередину. Js коду. Він також видаляє зіставлення між компонентами та стилями, додаючи всі вебпрефікси [22].

Для backend, використовуючи можливість nest-new, велика частина залежностей вже буде встановлена, але частину потрібно установити через npm [17]:

- Mongoose, бібліотека для підключення та використання бази даних MongoDB [23];
- Nodemailer, бібліотека для можливості надсилання листів із готовими темплейтами;
- Ejs, бібліотека для можливості підставляти дані в темплейти листів;
- Passport-Jwt, Bcrypt, бібліотеки для використання захисту запросивши тільки для зареєстрованих користувачів, створення та організації токен захисту API запитів.

3.1.2 Створення файлової структури

Для frontend (рис. 3.1) після встановлення бібліотек було створено основну структуру програми, це спростить подальшу розробку, обслуговування

та оновлення програми. Основна структура складається з конфігураційних файлів та 4 папок:

- node_modules;
- dist;
- public;
- src.

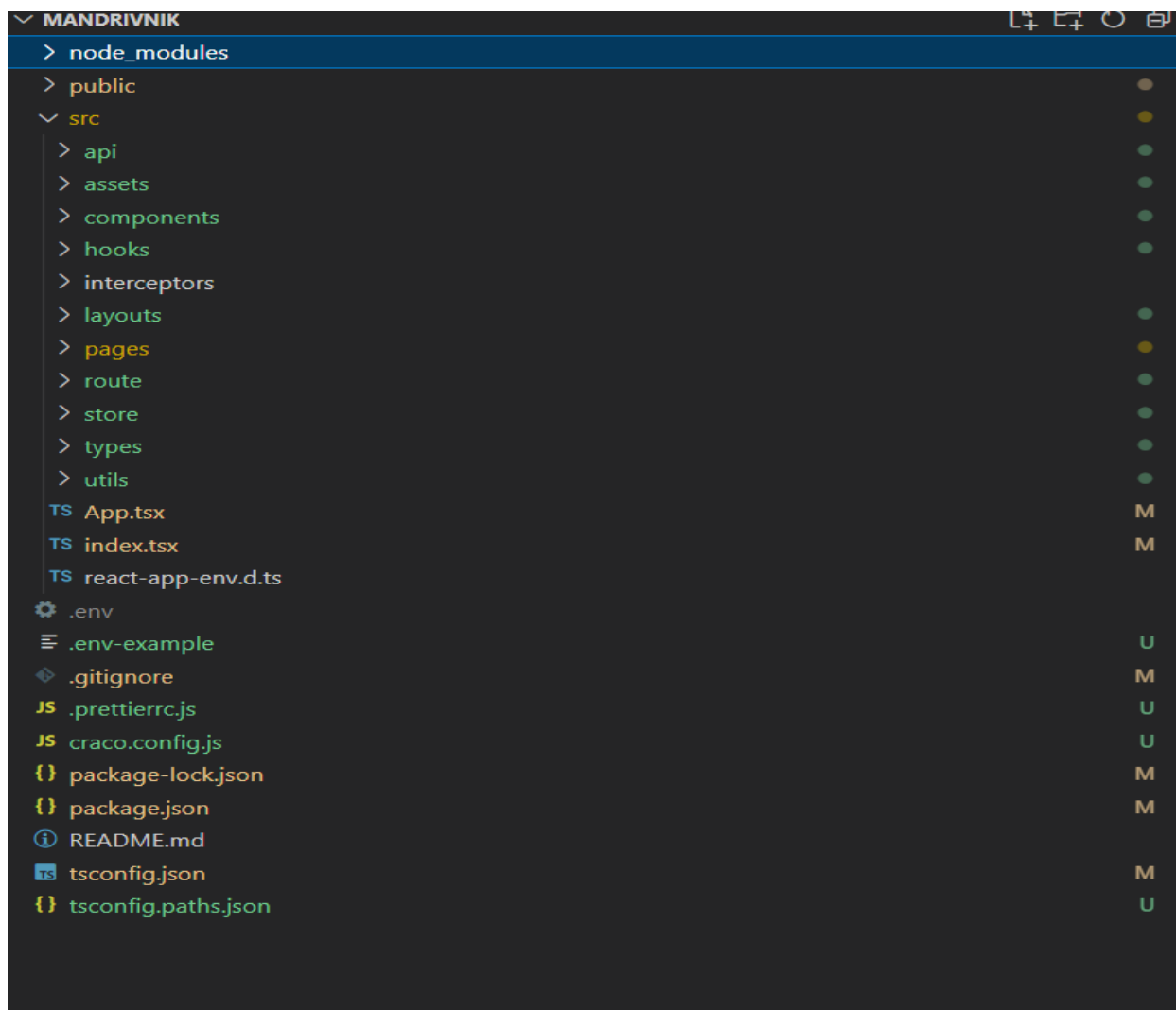


Рисунок 3.1 – Файлова структура frontend сайту Мандрівник

Node_modules папка в якій зберігаються всі встановлені залежності, dist папка з фінальним складанням проекту, обидві папки додані в gitignore так як вони з'являються при установці залежностей і складання проекту відповідно.

Папка `public` потрібна для React як точкою збирання всіх компонентів – там зберігається `index.html`. Папка `src` для розробки – там зберігаються всі файли, які розробить програміст.

Зі структури папок:

- `api` – папка з файлами сервісами API для швидкого доступу та пере використання;
- `app` – папка з файлами інтерфейсами для подальшої типізації проекту;
- `assets` – папка зі стилями, фотографіями та шрифтами;
- `components` – папка з компонентами проекту з якого збирається програма;
- `hooks` – папка з хуками, файлів помічників або функціоналу, що часто пере використовується;
- `pages` – папка зі сторінками проекту;
- `route` – папка з налаштуванням сторінок проектів;
- `store` – папка з налаштуванням сервісу React-Redux, який відповідає за дерево стану програми.

Для backend (рис. 3.2) основна структура складається з конфігураційних файлів та 4 папок таких як і на frontend.

Папка `src` ділиться на контролери та допоміжні файли та папки. Кожен контролер складається з 3 частин:

- `module` – до нього підключаються допоміжні файли, так само проводяться налаштування модулів і сервісів, що підключаються;
- `controller` – Файл, що підключається в модуль, в ньому створюються кінцеві точки API і підключається сервіси;
- `service` – файл у якому виконуються всі основні дії і найчастіше саме там здійснюються взаємодії з базою даних. Також сюди часто підключаються інші послуги для взаємодії между ними.

Comments, Users, Mail контролери, App контролер є основним, куди підключаються інші і підключається frontend при складанні та імпорті проекту.

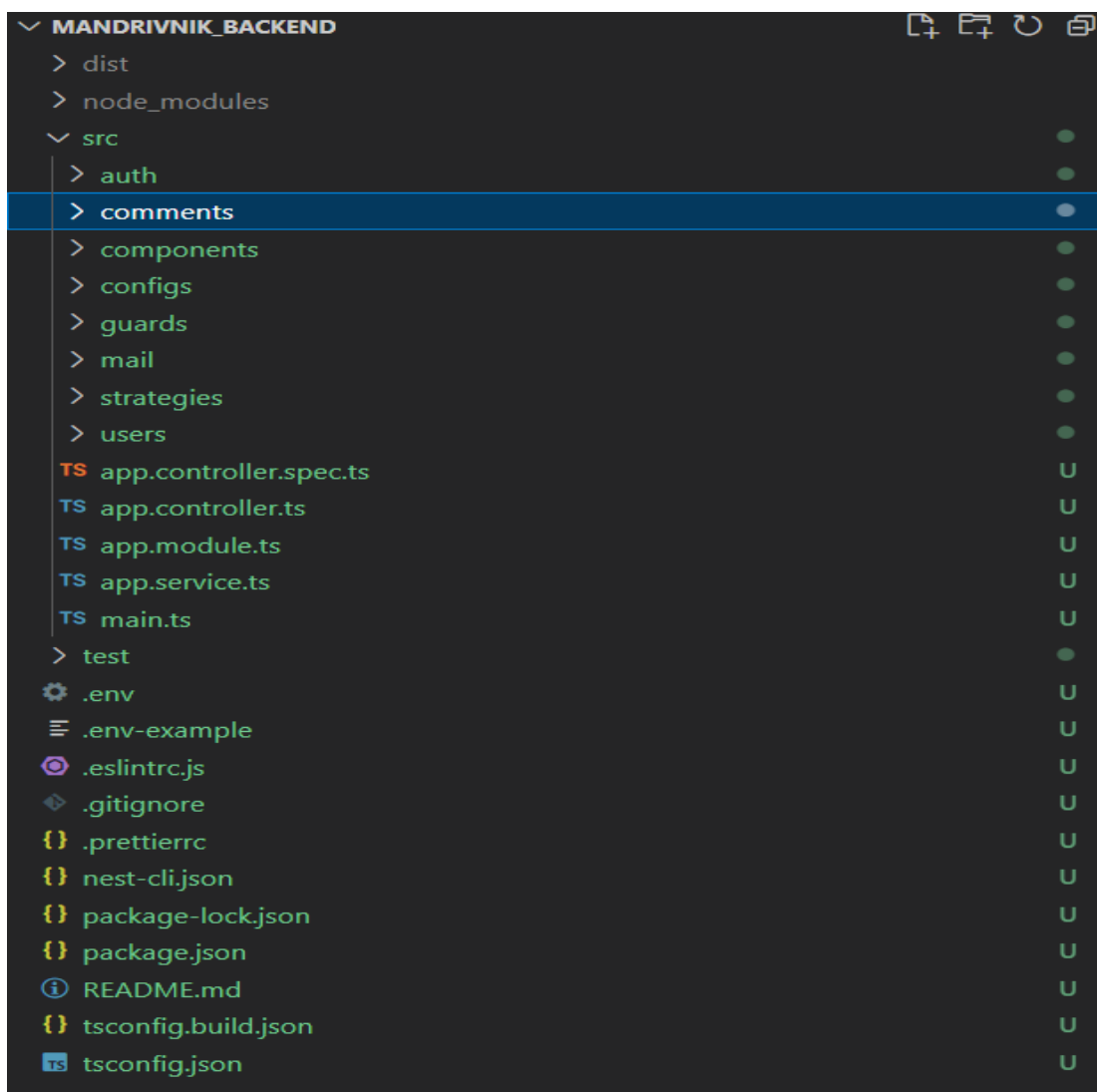


Рисунок 3.2 – Файлова структура backend сайту

Допоміжні файли:

- auth – папка, в якій описані гварди для кінцевих точок запитів для контролерів;
- configs – папка із файлами налаштувань сервісів;
- guards – Файли з повним описом процесу перевірки гвардів для контролерів;
- strategies – покладання стратегій на основі яких працюють гварди.

3.2 Реалізація вебсайту «Мандрівник»

Реалізація вебсайту буде проходити поступово за планом за допомоги канбан дошка з описом кожної дії, що відбувається.

3.2.1 Головна сторінка

Головна сторінка (рис. 3.3) має бути зрозумілою та достатньо інформативною. Користувач відразу бачить по центру екрану слоган і поле для пошуку будь-якого міста світу. Як тільки буде введено значення або натисно кнопку «Моя геолокація», йому відобразиться слайдер зі списком категорій локацій та кнопка «Дивитись усі». Натиснувши на одну з категорій або кнопку усі категорії, користувач перейде на сторінку зі списком і фільтрами знайденої локації. Також на головній сторінці є заголовок в якому є логотип, кнопка входу і світл для зміни теми сайту.

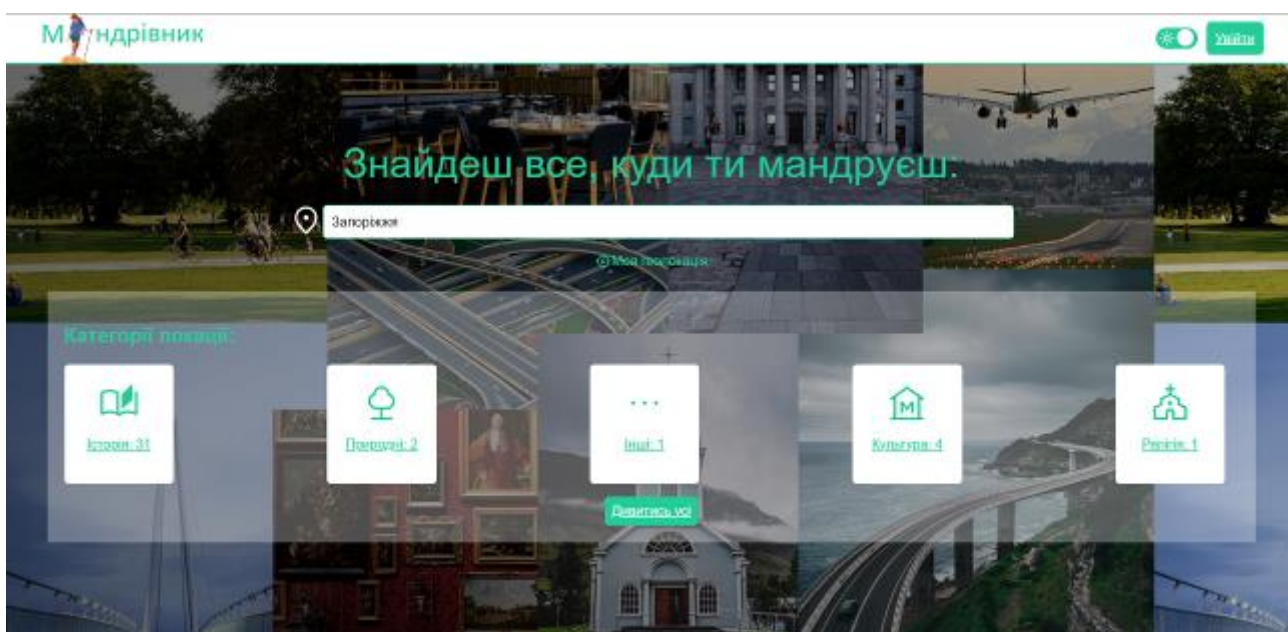


Рисунок 3.3 – Головна сторінка

3.2.2 Сторінка списку місць та фільтрів для них

Після пошуку локації та вибору категорії або всіх категорій відкривається сторінка зі списком усіх місць обраної локації (рис. 3.4) з можливістю фільтрації їх за декількома параметрами: категорії, радіус охоплення місць, рейтинг, який надає API сервіс OpenTripMap. Також додана пагінація для списку місць. У елемента міста відображається назва та їх рейтинг.

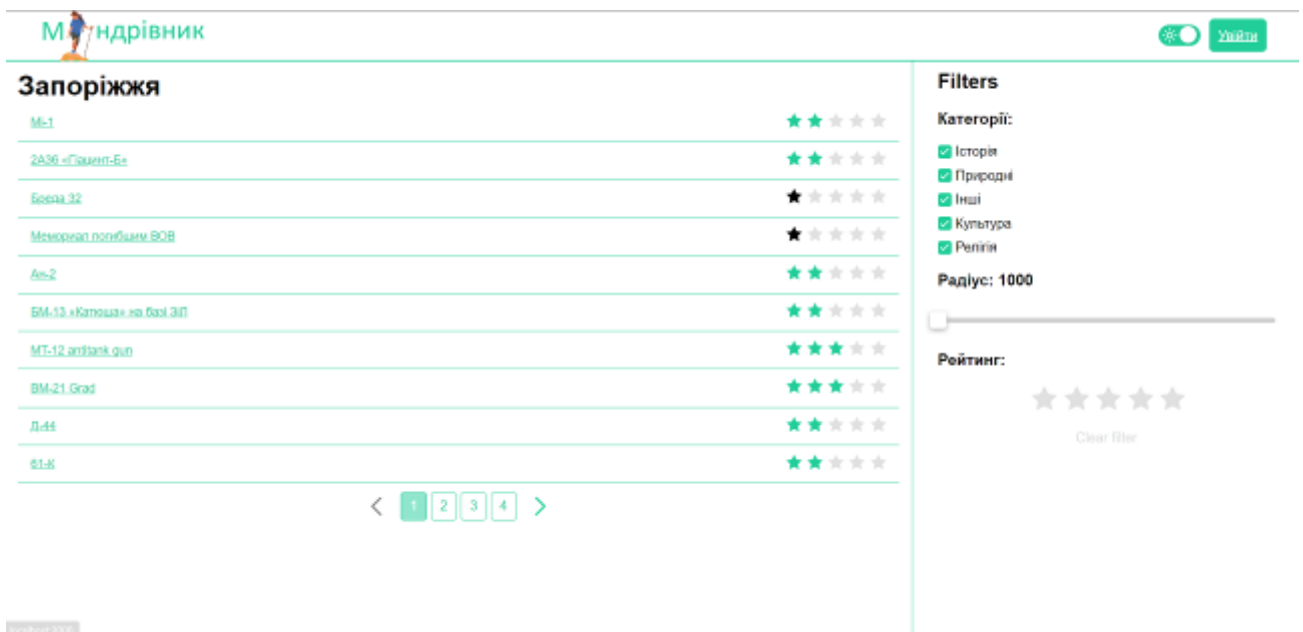
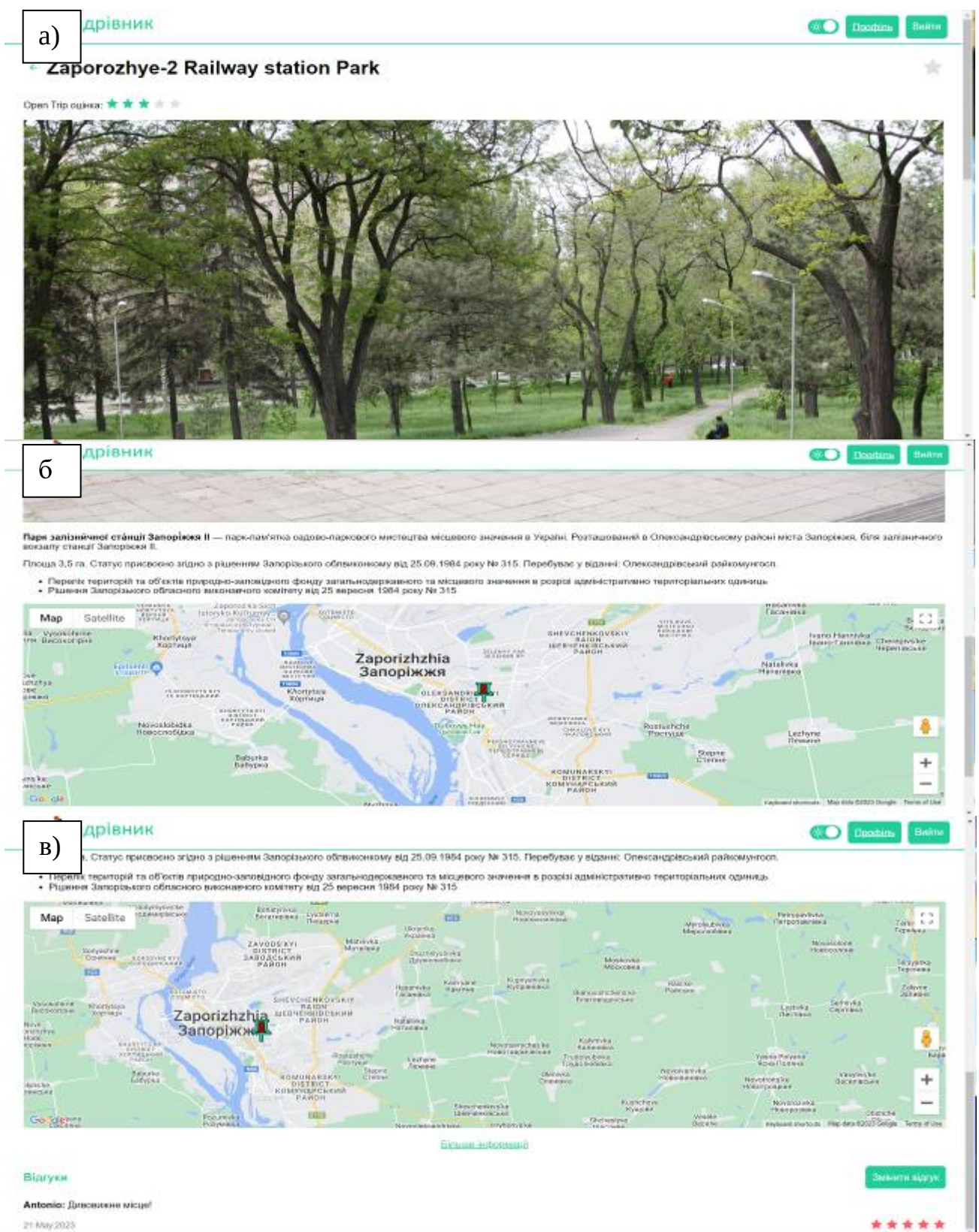


Рисунок 3.4 – Сторінка зі списком локації та фільтрами

3.2.3 Сторінка з інформацією про знайдене місце

Вибравши місце в списку, користувач потрапляє на сторінку цього місця (рис. 3.5) де відображається фотографія, коротка інформація, карта позначеним місцем, посилання на докладнішу інформацію, коментарі інших користувачів.



а – назва, оцінка, фото локації; б – коротка інформація, карта локації; в – посилання на детальну інформацію, коментарі користувачів залишені про дану локацію

Рисунок 3.5 – Сторінка знайденої локації

Потрапляються місця де в базі не збережена фотографія або коротка інформація, але всі точно мають чітку точку на карті. База даних постійно оновлюється та інформація про місця буде додана пізніше.

Більшість інформації українською мовою, але через використання стороннього ресурсу не рідко назви та інша інформація, трапляються англійською.

Зареєстровані користувачі можуть залишати коментарі, вони складаються з таких частин:

- дати написання або зміни;
- коментар;
- імені користувача;
- оцінки користувача.

Але залишати коментарі можуть тільки зареєстровані користувачі, оскільки оцінка повинна бути адекватна і як одна з застережень від захисту ботів.

3.2.4 Організація динамічних стилів, світлий стиль та темний стиль

Організація відображення динамічних стилів досягається за допомогою бібліотеки styled components, є два стилі світлий (рис. 3.6) та темний (рис. 3.7). Дизаїн, кольори були підібрані спеціально щоб користувачеві було комфортно на будь-якому з них.

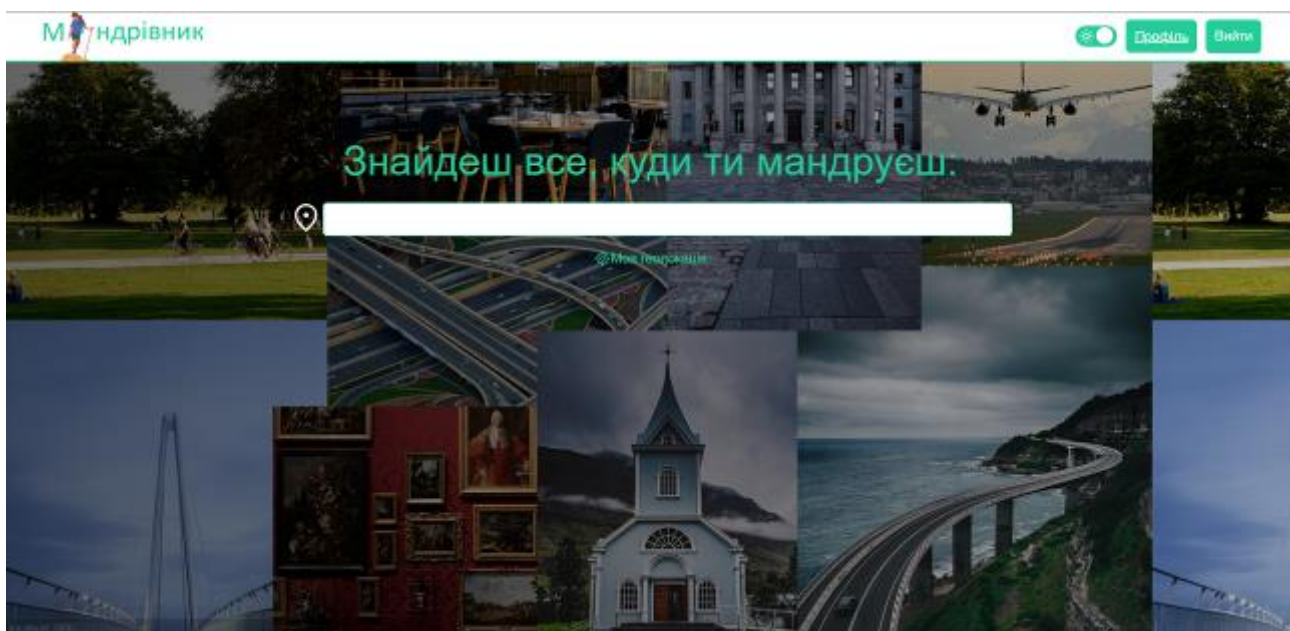


Рисунок 3.6 – За замовчуванням (світлий) стиль

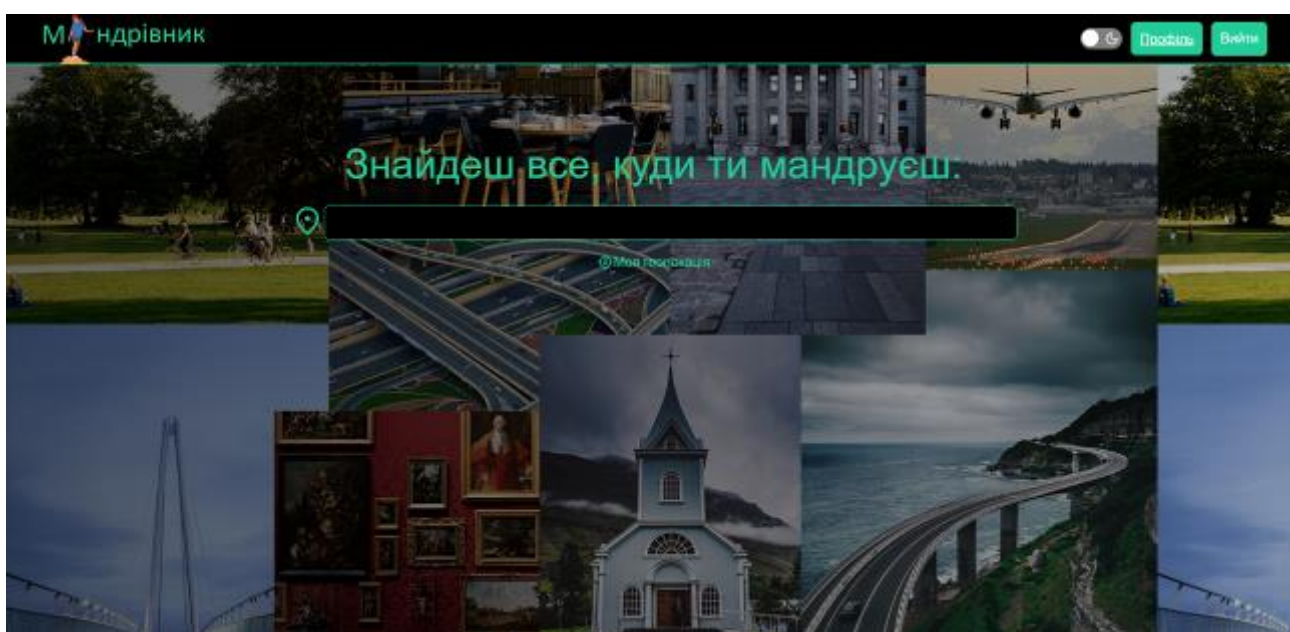


Рисунок 3.7 – Темний стиль

3.2.5 Сторінки для реєстрації, входу та відновлення пароля

На сайті організована можливість для входу (рис. 3.8) для користувачів, для них з'являються такі можливості:

- додавання локацій (рис.3.9) у вибрані місця та швидкий перехід на них (рис. 3.10);
- можливість залишати коментарі та оцінки до кожного місця (рис. 3.11).

Рисунок 3.8 – Сторінка для входу та реєстрації користувача

Рисунок 3.9 – Сторінка профілю користувача та вибрані локації

Для захисту запитів для зареєстрованих користувачів деякі запити працюють лише з бар'єром токеном і звичайні користувачі не можуть отримати доступ до них.

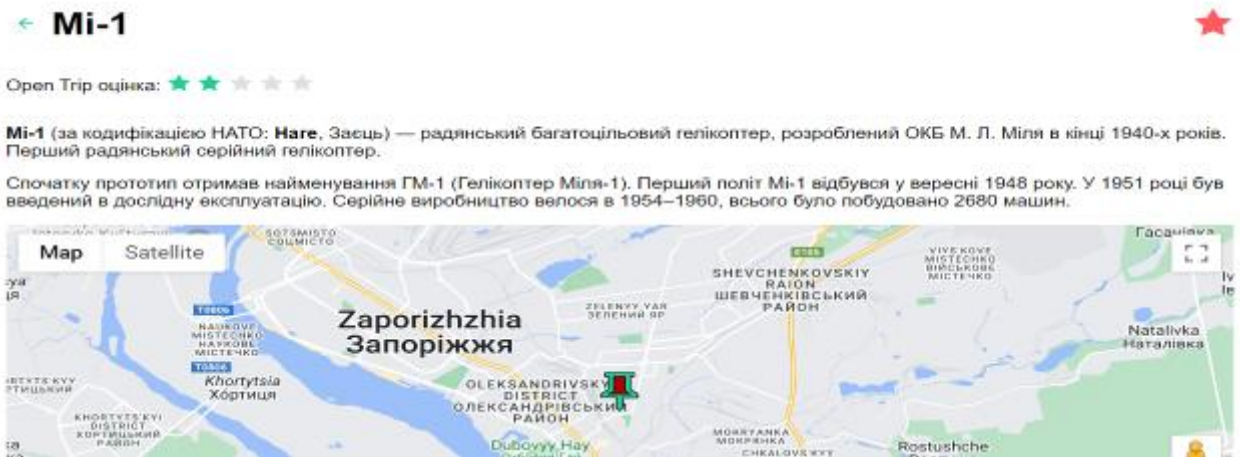


Рисунок 3.10 – Сторінка локації та зірочка вже доданої локації

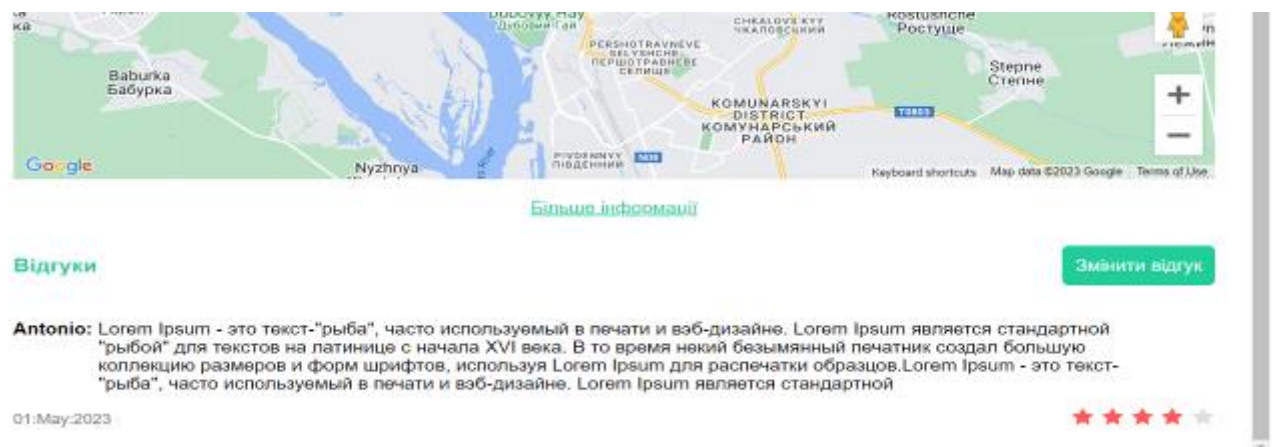


Рисунок 3.11 – Коментарі та оцінки користувачів

Після всіх робіт був проведений рефакторинг коду, а саме: виправлення знайдених помилок, підведення стилізації коду під один стиль – що в подальшому спрощує обслуговування та подальшу розробку коду. Є певні правила яких розробники намагаються дотримуватись, одне з них – це правильна та фіксована стилізація коду, для цього використовувалася плагін Prettier.

4 ТЕСТУВАННЯ ТА ДОСЛІДЖЕННЯ ВИГОТОВЛЕНОЇ ПРОГРАМИ.

4.1 Тестування UI елементів

Тестування візуального відображення – це перша вимога до загального тестування програми. Всі елементи повинні бути юзабельні, всі анімації працювати правильно, всі стилі відображатися коректно у всіх браузерях, і на будь-якій ширені екрану сайт повинен відображатися нормально та без горизонтальних скролів, котрі не були задуманні.

Було проведено тестування ці елементів, а саме:

- доступність натискання;
- правильне та належне функціонування;
- правильні та легко сприйнятливі стилі.

З тестування ці у сайті помилок найденно не було.

Доступність на всіх браузерах потрібна з елементарної причини – кожен користувач працює з браузером, який для нього зручний, у деяких браузерів обробка деяких стилів працює інакше, ніж у інших, що могло б призвести до некоректного відображення вебсайту. З браузерів де було протестовано Microsoft Edge, Opera, Safari і Chrome – не одна з них відмінностей від розробленого не мала.

Розміри екранів повинні бути так само протестовані – дуже багато користувачів які заходять на сайт з телефонів, планшетів, а також телевізорів. З проведених тестів жодних відхилень від первісного дизайну не було і сайт коректно відображається на всіх екранах (рис. 4.1)

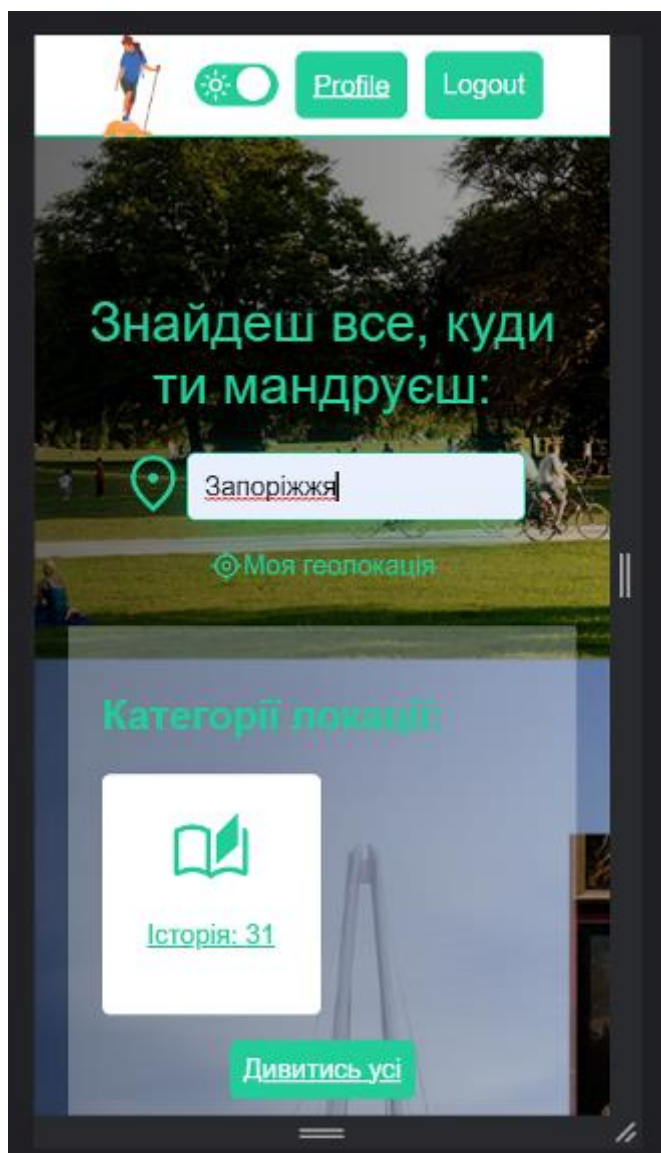


Рисунок 4.1 – Приклад головної сторінки для мобільних телефонів

4.2 Тестування у Google Chrome Dev tools

Основне тестування завантаження сайту було проведено в Google Dev tools системою скріншотів. З показників (рис. 4.2) стало ясно, що завантаження js-коду, css і фото проходить швидко, без помилок і без сильних завантажень на комп'ютер користувача.

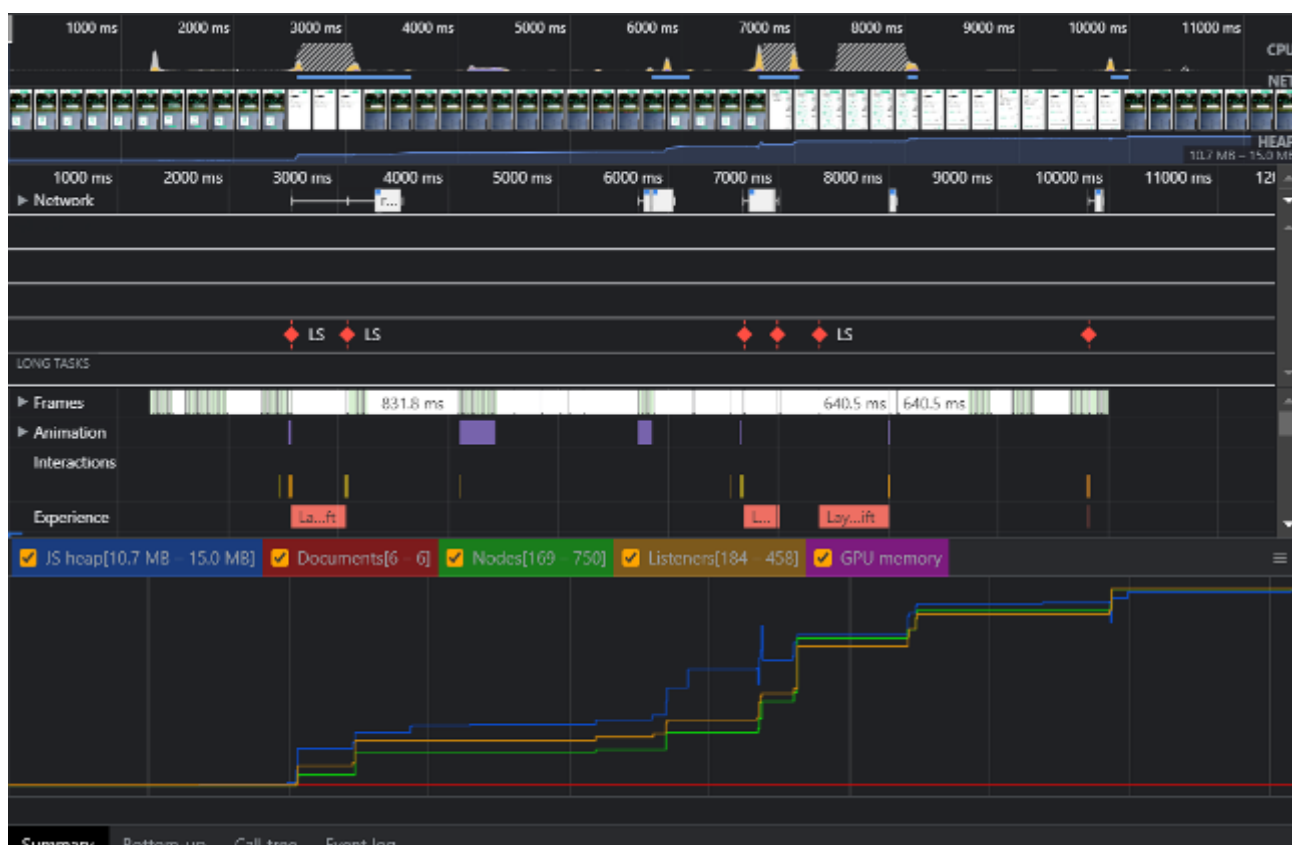


Рисунок 4.2 – Скріншоти програми у Chrome Dev tools

Також можна відзначити про дозавантаження js-коду з сервера хостингу проходить поступово, що зменшує навантаження і можливу кількість збоїв, так як сайт побудований на ньому. З цього всього, розміри файлів js, які є основними для створення вебсторінки, досить невеликі за розміром і легко завантажуються, що є основними для оптимізації швидкодії.

4.3 Рекомендації що до використання сайту «Мандрівник»

Так як зростають вимоги користувачів для сайту, так само зростають і вимоги до пристроїв та швидкості інтернету. Розробники хоч і намагаються щоб вебсайт займав якнайменше місця, але якщо якість інтернету буде дуже поганою, як би не був би оптимізований сайт, він завантажуватиметься набагато довше і працюватиме набагато повільніше.

Для комфортного використання сайту «Мандрівник» є мінімальні вимоги до швидкості інтернету, а саме мінімальна швидкість має бути 3G. За вимогами до пристроїв – відносні, оскільки ті пристрої, які підтримують сучасні браузері, підходять для використання сайту «Мандрівник». Звичайно на старих пристроях він працюватиме повільніше, тому що у сучасних браузерів збільшується завантаження на пристрій.

Для того, щоб використовувати сайт «Мандрівник» потрібно:

- зайти на сайт;
- ввести назву міста або натисніть кнопку «Моє місцезнаходження»;
- вибрати категорію місця або натиснути на кнопку «Вибрати все»;
- на сторінці зі списком локацій знайти потрібну за допомогою фільтрів і вибрати натиснувши на неї;
- на сторінці локації можна знайти фото, коротку інформацію, метознаходження на карті, посилання на докладну інформацію, а також коментарі інших користувачів.

Для того, щоб залишити коментар, потрібно зареєструватися або увійти на сайт:

- натиснути кнопку «Вхід»;
- на сторінці логіну або реєстрації ввести потрібні дані;
- далі на сторінці з обраною локацією з'явиться можливість залишати коментарі.

Також зайшовши на сторінку з'явиться можливість зайти у профіль користувача та додавати локації до обраних.

ВИСНОВКИ

В даній дипломній кваліфікаційній роботі було розроблено вебсайт «Мандрівник» для перегляду локацій у будь-якому місті світу, яка повністю розв'язує поставлене завдання і задовольняє всім поставленим вимогам. В ході розробки була проаналізована предметна область, визначенні вимоги до системи, в результаті чого було розроблене технічне завдання до розробки системи. Проведено аналіз вимог користувачів. Перед проектуванням системи були проаналізовані сучасні технології, і для розробки сайту була обраний такий стек технологій REACT, Style components, Nest js, MongoDB. Проведено концептуальне проектування, в результаті якого визначено структуру застосування та взаємодії з ним. Був розроблений frontend та backend частини для зручного перегляду прогнозу локацій.

Система дозволяє швидко знайти необхідне місто та його локації. Система легка у використанні, завдяки зручному інтерфейсу. Саме завдяки цьому будь-яка людина може оволодіти інтерфейсом та знайти .

Розроблена система була протестована та оптимізована для просування в мережі інтернет.

Таким чином, можна констатувати, що поставлена мета кваліфікаційної роботи бакалавра досягнута.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Google maps картографічний сервіс [Електронний ресурс]. – Режим доступу: <https://www.google.com/maps>;
2. Сервіс для бронування номерів [Електронний ресурс]. – Режим доступу: <https://www.booking.com/>;
3. Сервіс для замовлення квитків [Електронний ресурс]. – Режим доступу: <https://gd.tickets.ua/>;
4. Visual studio code – IDE для створення інтерфейсів користувача – Документація [Електронний ресурс]. – Режим доступу: <https://code.visualstudio.com/docs>;
5. Sublime text– IDE для створення інтерфейсів користувача – Документація [Електронний ресурс]. – Режим доступу: <https://www.sublimetext.com/docs/>;
6. Webstorm – IDE для створення інтерфейсів користувача – Документація [Електронний ресурс]. – Режим доступу: <https://www.jetbrains.com/help/webstorm/creating-jsdoc-comments.html>;
7. React JavaScript-бібліотека для створення інтерфейсів користувача – Документація [Електронний ресурс]. – Режим доступу: <https://ru.reactjs.org/docs/getting-started.html>;
8. Vue – то середовище JavaScript для створення інтерфейсів користувача – Документація [Електронний ресурс]. – Режим доступу: <https://vuejs.org/guide/introduction.html>;
9. Angular – це середовище проектування додатків та платформа розробки для створення ефективних та складних односторінкових додатків – Документація [Електронний ресурс]. – Режим доступу: <https://angular.io/docs>;
10. Порівняльний аналіз сучасних Фреймворків [Електронний ресурс]. – Режим доступу: <https://perf-track.web.app/>;

11. Справочник по TypeScript – Документація [Електронний ресурс]. — Режим доступу: <https://www.typescriptlang.org/docs/handbook/intro.html>;
12. Node.js® — це JavaScript-оточення побудоване на JavaScript — рушієві Chrome V8. Документація [Електронний ресурс]. — Режим доступу: <https://nodejs.org/uk/docs/>;
13. Nest js — Прогресивна платформа на Node.js для створення ефективних, надійних та масштабованих серверних програм. Документація [Електронний ресурс]. — Режим доступу: <https://docs.nestjs.com/>;
14. Diagrams — сервіс для малювання графіків. [Електронний ресурс]. — Режим доступу: <https://app.diagrams.net/>;
15. Jira — програмне забезпечення для відстеження завдань — Документація [Електронний ресурс]. — Режим доступу: <https://jam.dev/jira>;
16. Канбан дошка — сервіс вбудований у VSCode [Електронний ресурс]. — <https://github.com/vscode-kanban/vscode-kanban>;
17. Npm — менеджер пакетів, що входить до складу Node.js. Документація [Електронний ресурс]. — Режим доступу: <https://docs.npmjs.com/>;
18. Axios — це HTTP клієнт на основі обіцянок — Документація [Електронний ресурс]. — Режим доступу: <https://axios-http.com/docs/intro>;
19. React Redux — це офіційний шар прив'язок React UI для Redux. — Документація [Електронний ресурс]. — Режим доступу: <https://react-redux.js.org/introduction/getting-started>;
20. Redux Toolkit — набір інструментів для ефективної розробки Redux. Документація [Електронний ресурс]. — Режим доступу: <https://redux-toolkit.js.org/introduction/><https://docs.npmjs.com/>;
21. React router dom — бібліотека React з маршрутизацією на стороні клієнта— Документація [Електронний ресурс]. — Режим доступу: <https://reactrouter.com/en/main/start/overview>;

22. Styled-components бібліотека React для швидкого та простого створення стилізованих компонентів – Документація [Електронний ресурс]. – Режим доступу: <https://styled-components.com/docs/basics>;

23. Mongoose елегантне об'єктне моделювання mongodb для node.js – Документація [Електронний ресурс]. – Режим доступу: <https://mongoosejs.com/>;

ДОДАТОК А
Програмний код компонентів сайту

Frontend

```
index.tsx
import React from 'react';
import ReactDOM from 'react-dom/client';
import { Provider } from 'react-redux';
import App from './App';
import store from './store/store';
import './api/axios/axios';
const root = ReactDOM.createRoot(document.getElementById('root') as HTMLElement);
```

```
root.render(
  <Provider store={store}>
    <App />
  </Provider>
);
```

```
App.tsx
import React, { useEffect } from 'react';
import { RouterProvider } from 'react-router-dom';
import { ThemeProvider } from 'styled-components';
import router from './route/Route';
import { useAppDispatch, useAppSelector } from './store/hooks';
import { getTheme } from './store/theme/theme-selector';
import { Normalize } from 'styled-normalize';
import { createGlobalStyle } from 'styled-components';
import { changeStateAuth } from './store/user/user-reducer';
import { getUserData } from './store/user/user-actions';
import { Theme } from './store/theme/state';
import { changeTheme } from './store/theme/theme-reducer';
```

```
interface PropsGlobalStyles {
  name: string;
}
```

```
const GlobalStyle = createGlobalStyle<PropsGlobalStyles>`
  @import url('https://fonts.googleapis.com/css2?family=Roboto+Condensed:wght@300;400;700&display=swap');
  html{
    background-color: ${(props) => (props.name === 'black' ? '#ffffff' : 'rgba(128,128,128, 0.5))};
  }
  body {
    height: 100%;
  }
  * {
    box-sizing: border-box;
    font-family: 'Roboto Condensed', sans-serif;
  }
`;
```

```
const App: React.FC = () => {
  const dispatch = useAppDispatch();
  const theme = useAppSelector(getTheme);

  useEffect(() => {
    const theme = localStorage.getItem('theme');
    dispatch(changeTheme(theme === Theme.black ? Theme.black : Theme.white));
  }, []);
```

```

useEffect(() => {
  if (localStorage.getItem('access_token')) {
    dispatch(getUserData());
    dispatch(changeStateAuth(true));
  }
}, []);

return (
  <ThemeProvider theme={theme}>
    <Normalize />
    <GlobalStyle name={theme.name} />
    <RouterProvider router={router} />
  </ThemeProvider>
);
};

export default App;

import Accomodations from '@components/icons/Accomodations';
import Adult from '@components/icons/Adult';
import Amusements from '@components/icons/Amusements';
import Banks from '@components/icons/Banks';
import Cultural from '@components/icons/Cultural';
import Foods from '@components/icons/Foods';
import History from '@components/icons/History';
import IndustrialFacilities from '@components/icons/IndustrialFacilities';
import Natural from '@components/icons/Natural';
import Other from '@components/icons/Other';
import Religion from '@components/icons/Religion';
import Shops from '@components/icons/Shops';
import Sport from '@components/icons/Sport';
import Transport from '@components/icons/Transport';

export enum TypesPlaces {
  Accomodations = 'accomodations',
  Adult = 'adult',
  Amusements = 'amusements',
  Architecture = 'architecture',
  Banks = 'banks',
  Foods = 'foods',
  History = 'historic',
  IndustrialFacilities = 'industrial_facilities',
  Natural = 'natural',
  Other = 'other',
  Religion = 'religion',
  Shops = 'shops',
  Sport = 'sport',
  Transport = 'transport',
  Cultural = 'cultural',
}

export interface PlacesTypes {
  type: TypesPlaces;
  name: string;
  icon: React.ReactElement;
  amount: number;
}

GetTypesByPlaces.tsx
const GetTypesByPlaces = (kinds: string[]) => {

```

```

let placesTypes: PlacesTypes[] = [];

const checkTypeRepetition = (type: TypesPlaces) => {
  return placesTypes.findIndex((place) => place.type === type);
};

kinds.forEach((kind) => {
  switch (true) {
    case kind.includes(TypesPlaces.Accommodations): {
      let indexType = checkTypeRepetition(TypesPlaces.Accommodations);
      if (indexType !== -1) {
        placesTypes[indexType].amount += 1;
        break;
      }
      placesTypes.push({
        type: TypesPlaces.Accommodations,
        name: 'Готелі',
        icon: <Accommodations />,
        amount: 1,
      });
      break;
    }
    case kind.includes(TypesPlaces.Adult): {
      let indexType = checkTypeRepetition(TypesPlaces.Adult);
      if (indexType !== -1) {
        placesTypes[indexType].amount += 1;
        break;
      }
      placesTypes.push({
        type: TypesPlaces.Adult,
        name: 'Місця для дорослих',
        icon: <Adult />,
        amount: 1,
      });
      break;
    }
    case kind.includes(TypesPlaces.Amusements): {
      let indexType = checkTypeRepetition(TypesPlaces.Amusements);
      if (indexType !== -1) {
        placesTypes[indexType].amount += 1;
        break;
      }
      placesTypes.push({
        type: TypesPlaces.Amusements,
        name: 'Розваги',
        icon: <Amusements />,
        amount: 1,
      });
      break;
    }
    case kind.includes(TypesPlaces.Cultural): {
      let indexType = checkTypeRepetition(TypesPlaces.Cultural);
      if (indexType !== -1) {
        placesTypes[indexType].amount += 1;
        break;
      }
      placesTypes.push({
        type: TypesPlaces.Cultural,
        name: 'Культура',
        icon: <Cultural />,
        amount: 1,
      });
    }
  }
});

```

```

});
break;
}
case kind.includes(PlacesTypes.Architecture): {
  let indexType = checkTypeRepetition(PlacesTypes.Architecture);
  if (indexType !== -1) {
    placesTypes[indexType].amount += 1;
    break;
  }
  placesTypes.push({
    type: PlacesTypes.Architecture,
    name: 'Архітектура',
    icon: <Amusements />,
    amount: 1,
  });
  break;
}
case kind.includes(PlacesTypes.Banks): {
  let indexType = checkTypeRepetition(PlacesTypes.Banks);
  if (indexType !== -1) {
    placesTypes[indexType].amount += 1;
    break;
  }
  placesTypes.push({
    type: PlacesTypes.Banks,
    name: 'Банки',
    icon: <Banks />,
    amount: 1,
  });
  break;
}
case kind.includes(PlacesTypes.Foods): {
  let indexType = checkTypeRepetition(PlacesTypes.Foods);
  if (indexType !== -1) {
    placesTypes[indexType].amount += 1;
    break;
  }
  placesTypes.push({
    type: PlacesTypes.Foods,
    name: 'Їжа',
    icon: <Foods />,
    amount: 1,
  });
  break;
}
case kind.includes(PlacesTypes.History): {
  let indexType = checkTypeRepetition(PlacesTypes.History);
  if (indexType !== -1) {
    placesTypes[indexType].amount += 1;
    break;
  }
  placesTypes.push({
    type: PlacesTypes.History,
    name: 'Історія',
    icon: <History />,
    amount: 1,
  });
  break;
}
case kind.includes(PlacesTypes.IndustrialFacilities): {
  let indexType = checkTypeRepetition(PlacesTypes.IndustrialFacilities);

```

```

if (indexType !== -1) {
  placesTypes[indexType].amount += 1;
  break;
}
placesTypes.push({
  type: TypesPlaces.IndustrialFacilities,
  name: 'Промислові об'єкти',
  icon: <IndustrialFacilities />,
  amount: 1,
});
break;
}
case kind.includes(TypesPlaces.Natural): {
  let indexType = checkTypeRepetition(TypesPlaces.Natural);
  if (indexType !== -1) {
    placesTypes[indexType].amount += 1;
    break;
  }
  placesTypes.push({
    type: TypesPlaces.Natural,
    name: 'Природні',
    icon: <Natural />,
    amount: 1,
  });
  break;
}
case kind.includes(TypesPlaces.Religion): {
  let indexType = checkTypeRepetition(TypesPlaces.Religion);
  if (indexType !== -1) {
    placesTypes[indexType].amount += 1;
    break;
  }
  placesTypes.push({
    type: TypesPlaces.Religion,
    name: 'Релігія',
    icon: <Religion />,
    amount: 1,
  });
  break;
}
case kind.includes(TypesPlaces.Shops): {
  let indexType = checkTypeRepetition(TypesPlaces.Shops);
  if (indexType !== -1) {
    placesTypes[indexType].amount += 1;
    break;
  }
  placesTypes.push({
    type: TypesPlaces.Shops,
    name: 'Магазини',
    icon: <Shops />,
    amount: 1,
  });
  break;
}
case kind.includes(TypesPlaces.Sport): {
  let indexType = checkTypeRepetition(TypesPlaces.Sport);
  if (indexType !== -1) {
    placesTypes[indexType].amount += 1;
    break;
  }
  placesTypes.push({

```

```

    type: TypesPlaces.Sport,
    name: 'Спорт',
    icon: <Sport />,
    amount: 1,
  });
  break;
}
case kind.includes(TypesPlaces.Transport): {
  let indexType = checkTypeRepetition(TypesPlaces.Transport);
  if (indexType !== -1) {
    placesTypes[indexType].amount += 1;
    break;
  }
  placesTypes.push({
    type: TypesPlaces.Transport,
    name: 'Транспорт',
    icon: <Transport />,
    amount: 1,
  });
  break;
}
default: {
  let indexType = checkTypeRepetition(TypesPlaces.Other);
  if (indexType !== -1) {
    placesTypes[indexType].amount += 1;
    break;
  }
  placesTypes.push({
    type: TypesPlaces.Other,
    name: 'Имми',
    icon: <Other />,
    amount: 1,
  });
  break;
}
});
return placesTypes;
};

export default GetTypesByPlaces;

```

SignUp.tsx

```

import React from 'react';
import SignUpForm from '@components/forms/SignUpForm';

const SignUp: React.FC = () => {
  return (
    <main>
      <section>
        <SignUpForm />
      </section>
    </main>
  );
};

export default SignUp;

```

SignIn.tsx

```
import SignInForm from '@components/forms/SignInForm';
import React from 'react';
```

```
const SignIn: React.FC = () => {
  return (
    <main>
      <section>
        <SignInForm />
      </section>
    </main>
  );
};
```

```
export default SignIn;
```

Root.tsx

```
import React, { useCallback, useEffect, useState } from 'react';
import styled from 'styled-components';
import backgroundMandrivnik from '@assets/mandrivnik.jpg';
import PlaceInput from '@components/PlaceInput';
import PlacesListTypes from '@components/PlacesListTypes';
import { useAppDispatch, useAppSelector } from '@store/hooks';
import { clearCategoriesData, clearLocationData } from '@store/place/place-reducer';
import { getLocation } from '@store/place/place-selector';
import { GeoData } from '@api/api-types';
import Loader from '@components/Loader';
```

```
const Main = styled.main`
  overflow: auto;
  background-image: linear-gradient(90deg, rgba(0, 0, 0, 0.6) 0%, rgba(0, 0, 0, 0.3) 100%),
  url(${backgroundMandrivnik});
`;
```

```
interface sectionProps {
  isTop?: boolean;
}
```

```
const Section = styled.section<sectionProps>`
  padding: 0 50px;
  height: 100vh;
  object-fit: cover;
  padding-top: ${(props) => props.isTop && '150px'};
  @media (max-width: 480px) {
    padding: 0 20px;
    padding-top: ${(props) => props.isTop && '150px'};
  }
`;
```

```
const Title = styled.title`
  display: block;
  font-size: 3rem;
  text-align: center;
  color: ${(props) => (props.theme.white ? '#ffffff' : '#21ce99')};
  @media (max-width: 480px) {
    font-size: 2rem;
  }
`;
```

```
const Home: React.FC = () => {
  const dispatch = useAppDispatch();
```

```

const location: GeoData | null = useAppSelector(getLocation);
const [isLoading, setIsLoading] = useState(false);

useEffect(() => {
  dispatch(clearLocationData());
  dispatch(clearCategoriesData());
}, []);

const changeLoadingState = useCallback((state: boolean) => {
  setIsLoading(state);
}, []);

return (
  <Main>
    <Section isTop>
      <Title>Знайдеш все, куди ти мандруєш:</Title>
      <PlaceInput locationData={location} isLoading={isLoading} changeLoadingState={changeLoadingState} />
      {isLoading && <Loader />}
      {location && <PlacesListTypes />}
    </Section>
  </Main>
);
};

export default Home;

```

RestorePassword.tsx

```

import React from 'react'
import RestoreForm from '@components/forms/RestoreForm'

const ResetPassword: React.FC = () => {
  return (
    <main>
      <section>
        <RestoreForm />
      </section>
    </main>
  )
}

```

export default ResetPassword

ResetPassword.tsx

```

import React from 'react'
import ResetForm from '@components/forms/ResetForm'

const ResetPassword: React.FC = () => {
  return (
    <main>
      <section>
        <ResetForm />
      </section>
    </main>
  )
}

```

export default ResetPassword

Profile.tsx

```

import FavoriteCard from '@components/FavoriteCard';
import Pagination from '@components/Pagination';
import { useAppSelector } from '@store/hooks';
import { getUser } from '@store/user/user-selector';
import React, { useState, useCallback, useMemo } from 'react';
import styled from 'styled-components';

const MainProfile = styled.main`
padding: 0 20px;
padding-top: 60px;
background-color: ${({props}) => (props.theme.name === 'black' ? '#ffffff' : 'rgba(128,128,128, 0.5)')};
height: 100vh;
section {
  &.favorites {
    ul {
      padding: 0;
    }
    li {
      list-style-type: none;
    }
  }
}
`;

const ProfileInput = styled.div`
position: relative;
margin-top: ${({props}) => (props.theme.name === 'black' ? '15px' : '25px')};
max-width: 300px;
height: 40px;
label {
  position: absolute;
  top: ${({props}) => (props.theme.name === 'black' ? '-10px' : '-14px')};
  left: 0;
  background-color: ${({props}) => (props.theme.name === 'black' ? '#ffffff' : 'transparent')};
  font-size: 0.88rem;
}
input {
  height: 100%;
  width: 100%;
  &:disabled {
    background-color: white;
  }
}
`;

const Profile: React.FC = () => {
  const { nick, email, favorites: favoritesIds } = useAppSelector(getUser);
  const [nickname, setNickname] = useState<string>(nick || "");
  const [offset, setOffset] = useState(0);
  const [page, setPage] = useState(1);
  const limit = useMemo(() => 3, []);

  const changePage = useCallback((page: number) => {
    setPage(page);
    setOffset(page * limit);
  }, []);

  const favorites = useMemo(() => favoritesIds.slice(offset, page * limit), [favoritesIds, offset, page, limit]);

  const changeNickName = () => {};

```

```

return (
  <MainProfile>
    <section>
      <h1>Профіль</h1>
      <ProfileInput>
        <label>Псевдонім</label>
        <input value={nickname} onChange={changeNickName} />
      </ProfileInput>
      <ProfileInput>
        <label>Електронна пошта</label>
        <input value={email || ""} readOnly disabled />
      </ProfileInput>
    </section>
    <section className="favorites">
      <h2>Обрані локації</h2>
      <ul>
        {favorites.map((favorite) => (
          <li>
            <FavoriteCard location={favorite} />
          </li>
        ))}
        <Pagination offset={offset} limit={limit} count={favorites.length} changePage={changePage} />
      </ul>
    </section>
  </MainProfile>
);
};

```

export default Profile;

Places.tsx

```

import React, { useCallback, useEffect, useMemo, useState } from 'react';
import { useAppDispatch, useAppSelector } from '@store/hooks';
import {
  getLocationData,
  getPlacesCategoryData,
  getPlacesCountData,
  getPlacesDataByCategory,
} from '@store/place/place-action';
import { GeoData, Place } from '@api/api-types';
import { PathsList } from '@route/Paths';
import { getCategories, getCountPlaces, getLocation, getPlaces } from '@store/place/place-selector';
import { useNavigate, useSearchParams } from 'react-router-dom';
import styled from 'styled-components';
import GetTypesByPlaces, { PlacesTypes } from '@utils/GetTypesByPlaces';
import FilterPlaces from '@components/FilterPlaces';
import Pagination from '@components/Pagination';
import { Link } from 'react-router-dom';
import Stars from '@components/Stars';

const PlacesMain = styled.main`
  padding-top: 60px;
  display: flex;
  height: 100vh;
  background-color: ${({props}) => (props.theme.name === 'black' ? '#ffffff' : 'rgba(128,128,128, 0.5)')});
`;

const PlacesSection = styled.section`
  flex: 1 0 70%;

```

```

max-width: 70%;
padding: 0 15px;
h1 {
  margin: 0;
  margin-top: 10px;
  height: 40px;
}
ul.list-places {
  padding: 0;
  margin: 0;
  max-height: calc(100% - 100px);
  overflow: auto;
  li {
    width: 100%;
    height: 45px;
    border-bottom: 1px solid ${props => (props.theme.name === 'black' ? 'rgb(33, 206, 153)' : 'rgba(0,0,0, 1)')};
    &.not-found {
      height: 100px;
      text-align: center;
      border: none;
      display: flex;
      justify-content: center;
      align-items: center;
    }
  }
}
a {
  padding: 0 15px;
  display: flex;
  justify-content: space-between;
  align-items: center;
  height: 100%;
  color: ${props => (props.theme.name === 'black' ? '#21ce99' : '#000000')};
  font-size: 0.88rem;
  transition: 0.33s ease-in-out background-color;
  &:hover {
    background-color: rgba(33, 206, 153, 0.2);
  }
}
li {
  list-style-type: none;
}
`;

const FiltersSection = styled.section`
  padding: 0 30px;
  flex: 1 0 30%;
  max-width: 30%;
  overflow: auto;
  border-left: 1px solid rgb(33, 206, 153);
  h2 {
    margin-top: 10px;
  }
`;

const Places: React.FC = () => {
  const [offset, setOffset] = useState(0);
  const count = useAppSelector(getCountPlaces);
  const location: GeoData | null = useAppSelector(getLocation);
  const places: Place[] = useAppSelector(getPlaces);
  const categoriesPlaces: string[] = useAppSelector(getCategories);
  const categories: PlacesTypes[] = useMemo(() => GetTypesByPlaces(categoriesPlaces), [categoriesPlaces]);

```

```

const dispatch = useAppDispatch();
const navigate = useNavigate();
const [searchParams, setSearchParams] = useSearchParams();

const limit = useMemo(() => 10, []);

const searchData = useMemo(() => {
  return {
    kinds: searchParams.get('category'),
    rate: searchParams.get('rate'),
    radius: searchParams.get('radius'),
  };
}, [searchParams.get('category'), searchParams.get('rate'), searchParams.get('radius')]);

useEffect(() => {
  if (!location && !searchParams.get('name')) {
    navigate(PathsList.ROOT);
    return;
  }
  if (!location && searchParams.get('name')) {
    dispatch(getLocationData(searchParams.get('name')));
  }
  if (!searchParams.get('radius')) {
    setSearchParams((searchParams) => {
      searchParams.set('radius', '1000');
      return searchParams;
    });
  }
}, []);

useEffect(() => {
  if (location && categories.length === 0 && searchData.radius) {
    dispatch(getPlacesCategoryData({ lat: location.lat, lon: location.lon, radius: Number(searchData.radius) }));
  }
}, [location, searchData.radius, categories]);

useEffect(() => {
  if (location && searchData.radius) {
    dispatch(
      getPlacesCountData({
        lat: location.lat,
        lon: location.lon,
        radius: Number(searchData.radius!),
        kinds: searchData?.kinds || undefined,
        rate: searchData?.rate ? Number(searchData?.rate) : undefined,
      })
    );
    dispatch(
      getPlacesDataByCategory({
        lat: location.lat,
        lon: location.lon,
        radius: Number(searchData.radius!),
        kinds: searchData?.kinds || undefined,
        rate: searchData?.rate ? Number(searchData?.rate) : undefined,
        offset,
        limit,
      })
    );
  }
}, [location, location?.lat, location?.lon, searchData.radius, searchData.kinds, searchData.rate, offset]);

```

```

const changePagePlaces = useCallback((page: number) => {
  setOffset(page * limit);
}, []);

return (
  <PlacesMain>
    <PlacesSection className="places">
      <h1>{location?.name}</h1>
      <ul className="list-places">
        {places.length > 0 ? (
          places.map((place) => (
            <li key={place.xid}>
              <Link to={PathsList.PLACE.replace(':xid', place.xid)}>
                <span>{place.name}</span>
                <Stars isDisable onClick={() => {}} value={place.rate - 1} />
              </Link>
            </li>
          ))
        ) : (
          <li className="not-found">
            <p>За заданими фільтрами не знайдено місць</p>
          </li>
        )}
      </ul>
      <Pagination changePage={changePagePlaces} offset={offset} limit={limit} count={count} />
    </PlacesSection>
    <FiltersSection>
      <h2>Filters</h2>
      <FilterPlaces categories={categories} />
    </FiltersSection>
  </PlacesMain>
);
};

export default Places;

export interface UserAPI {
  getUser: () => Promise<User>;
  changeUserData: (id: string, data: User) => void;
  changeStateFavorite: (data: DataFavorite) => Promise<string>;
}

enum ApiList {
  User = 'user',
  Update = 'update',
  Favorite = 'favorite',
}

const UserService: UserAPI = {
  getUser: () => axios.get<User>(`/${ApiControllers.users}/${ApiList.User}`).then((res) => res.data),
  changeUserData: (id, data) =>
    axios.put(`/${ApiControllers.users}/${ApiList.Update}/${id}`, data).then((res) => res.data),
  changeStateFavorite: (data) =>
    axios.post(`/${ApiControllers.users}/${ApiList.Favorite}`, data).then((res) => res.data),
};

export default UserService;

place.ts

import axios from 'axios';

```

```

import { CreateComment, FullDataPlace, GeoData, GeocodingType, Place, Comment } from './api-types';

export interface PlaceAPI {
  getLocationData: (name: string) => Promise<GeoData>;
  getPlacesCountByLocation: (params: {
    lon: number;
    lat: number;
    radius: number;
    kinds?: string;
  }) => Promise<{ count: number }>;
  getPlacesByLocation: (params: {
    lon: number;
    lat: number;
    radius: number;
    kinds?: string;
    offset?: number;
    limit?: number;
  }) => Promise<Place[]>;
  getPlaceById: (id: string | number) => Promise<FullDataPlace>;
  getCommentsByLocation: (id: string) => Promise<Comment[]>;
  postComment: (date: CreateComment) => Promise<Comment>;
  updateComment: (date: CreateComment) => Promise<Comment>;
  getNameLocation: (lat: number, lon: number) => Promise<GeocodingType[]>;
}

enum ApiList {
  GeoName = '/places/geoname',
  GetByRadius = '/places/radius',
  GetDataById = '/places/xid',
  Comments = '/comments',
}

const PlaceService: PlaceAPI = {
  getNameLocation: async (lat, lon) => {
    return axios
      .get<GeocodingType[]>('https://cors-anywhere.herokuapp.com/http://api.openweathermap.org/geo/1.0/reverse', {
        params: {
          lat: lat,
          lon: lon,
          limit: 5,
          appid: process.env.REACT_APP_OPEN_WEATHER_API_KEY,
        },
      })
      .then((res) => res.data);
  },
  getLocationData: async (name) => {
    let response = await axios.get<GeoData>(process.env.REACT_APP_OPEN_TRIP_API + ApiList.GeoName, {
      params: {
        apikey: process.env.REACT_APP_OPEN_TRIP_KEY,
        name,
      },
    });
    return response.data;
  },
  getPlacesCountByLocation: async (params) => {
    let response = await axios.get<{ count: number }>(process.env.REACT_APP_OPEN_TRIP_API +
    ApiList.GetByRadius, {
      params: { ...params, apikey: process.env.REACT_APP_OPEN_TRIP_KEY, format: 'count' },
    });
    return response.data;
  },
}

```

```

getPlacesByLocation: async (params) => {
  let response = await axios.get<Place[]>(process.env.REACT_APP_OPEN_TRIP_API + ApiList.GetByRadius, {
    params: { ...params, apikey: process.env.REACT_APP_OPEN_TRIP_KEY, format: 'json' },
  });
  return response.data;
},
getPlaceById: async (id) => {
  let response = await axios.get<FullDataPlace>(
    process.env.REACT_APP_OPEN_TRIP_API + ApiList.GetDataById + `/${id}`,
    {
      params: { apikey: process.env.REACT_APP_OPEN_TRIP_KEY, format: 'json' },
    }
  );
  return response.data;
},
getCommentsByLocation: async (id) => axios.get<Comment[]>(ApiList.Comments + `/${id}`).then((res) =>
res.data),
updateComment: async (data) => axios.put<Comment>(ApiList.Comments, data).then((res) => res.data),
postComment: async (data) => axios.post<Comment>(ApiList.Comments, data).then((res) => res.data),
};

```

```
export default PlaceService;
```

```
login.ts
```

```
import axios from 'axios';
```

```
import { ApiControllers, UserData, LoginData, UserResponse } from './api-types';
```

```

export interface LoginAPI {
  signIn: (data: LoginData) => Promise<UserResponse>;
  signUp: (data: LoginData) => Promise<UserData>;
  postRefreshToken: () => Promise<{ access_token: string }>;
  resetPassword: (data: LoginData) => void;
  changePassword: (data: LoginData) => void;
  verifyUser: () => void;
}

```

```

enum ApiList {
  SignIn = 'sign-in',
  Signup = 'sign-up',
  Refresh = 'refresh',
}

```

```

const LoginService: LoginAPI = {
  signIn: async (data) => {
    return await axios
      .post<UserResponse>(`/ ${ApiControllers.users}/${ApiList.SignIn}`, data)
      .then((response) => response.data);
  },
  signUp: async (data) => {
    return await axios
      .post<UserData>(`/ ${ApiControllers.users}/${ApiList.Signup}`, data)
      .then((response) => response.data);
  },
  postRefreshToken: async () => {
    return await axios
      .get<{ access_token: string }>(`/ ${ApiControllers.users}/${ApiList.Refresh}`, {
        headers: { Authorization: `Bearer ${localStorage.getItem('refresh_token')}` },
      })
      .then((response) => response.data);
  },
  resetPassword: () => {},
}

```

```

    changePassword: () => {},
    verifyUser: () => {},
  };

export default LoginService;

api-types.ts

export enum ApiControllers {
  users = 'users',
}

export interface DataFavorite {
  location: string;
  user_id: string;
}
export interface DataFavoriteAnswer {
  state: boolean;
  location: string;
}

export interface User {
  id?: string;
  _id?: string;
  nick: string;
  email: string;
  favorites?: string[];
}

export interface UserResponse {
  user: User;
  access_token: string;
}

export interface LoginData {
  email?: string;
  nick?: string;
  password?: string;
  password_confirm?: string;
}

export interface UserData {
  nick: string;
  email: string;
  password: string;
  password_confirm: string;
}

export interface GeoData {
  name: string;
  country: string;
  lat: number;
  lon: number;
  population: number;
  timezone: string;
  status?: string;
}

export interface Place {
  dist: number;
  kinds: string;

```

```

name: string;
osm: string;
point: { lat: number; lon: number };
rate: number;
wikidata: string;
xid: string;
}

```

```

export interface FullDataPlace {
  xid: string;
  name: string;
  address: {
    city: string;
    road: string;
    house: string;
    state: string;
    county: string;
    country: string;
    postcode: string;
    country_code: string;
    house_number: string;
  };
  rate: string;
  osm: string;
  wikidata: string;
  kinds: string;
  sources: {
    geometry: string;
    attributes: [string, string];
  };
  preview: {
    source: string;
    height: number;
    width: number;
  };
  image: string;
  otm: string;
  wikipedia: string;
  wikipedia_extracts: {
    title: string;
    text: string;
    html: string;
  };
  point: {
    lon: number;
    lat: number;
  };
}

```

```

export interface Comment {
  message: string;
  date_message: string;
  rate: number;
  location_id: string;
  nick: string;
  user_id: string;
}

```

```

export interface CreateComment {
  message: string;
  date_message: number;
}

```

```

    rate: number;
    location_id: string;
    user_id: string;
  }

export interface FavoriteUser {
  location: string;
  user_id: string;
}

export interface GeocodingType {
  id?: number;
  name: string;
  local_names?: any;
  lat: number;
  lon: number;
  country: string;
  state?: string;
}

```

Backend

Main.ts

```

import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';

async function bootstrap() {
  const app = await NestFactory.create(AppModule, {
    cors: true,
  });
  await app.listen(3003);
}
bootstrap();

```

app.service.ts

```

import { Injectable } from '@nestjs/common';

@Injectable()
export class AppService {
  getHello(): string {
    return 'Hello World!';
  }
}

```

App.module.ts

```

import { Module } from '@nestjs/common';
import { ConfigModule } from '@nestjs/config';
import { MongooseModule } from '@nestjs/mongoose';
import { AppController } from './app.controller';
import { AppService } from './app.service';
import { UsersModule } from './users/users.module';
import { MailModule } from './mail/mail.module';
import { CommentsModule } from './comments/comments.module';

@Module({
  imports: [

```

```

    ConfigModule.forRoot({
      isGlobal: true,
    }),
    MongooseModule.forRoot(
      `mongodb+srv://${process.env.DATABASE_LOGIN}:${process.env.DATABASE_PASSWORD}@${process.env.DATABASE_CLUSTER}.${process.env.DATABASE_CONFIG}`,
    ),
    UsersModule,
    MailModule,
    CommentsModule,
  ],
  controllers: [AppController],
  providers: [AppService],
})
export class AppModule {}

```

app.controller.ts

```

import { Controller, Get } from '@nestjs/common';
import { AppService } from './app.service';

@Controller()
export class AppController {
  constructor(private readonly appService: AppService) {}

  @Get()
  getHello(): string {
    return this.appService.getHello();
  }
}

```

users.module.ts

```

import { Logger, Module } from '@nestjs/common';
import { PassportModule } from '@nestjs/passport';
import { UsersService } from './users.service';
import { UsersController } from './users.controller';
import { MongooseModule } from '@nestjs/mongoose';
import { User, UserSchema } from './schemas/users.schemas';
import { JwtModule } from '@nestjs/jwt';
import { AppJwtStrategy } from 'src/strategies/app-jwt.strategy';

@Module({
  imports: [
    MongooseModule.forFeature([{ name: User.name, schema: UserSchema }]),
    PassportModule,
    JwtModule.register({
      secret: process.env.JWT_SECRET_ETERNAL_TOKEN,
      signOptions: { expiresIn: '1h' },
    }),
    Logger,
  ],

  providers: [UsersService, AppJwtStrategy],
  controllers: [UsersController],
  exports: [UsersService],
})
export class UsersModule {}

```

users.service.ts

```

import { BadRequestException, Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Model, Types } from 'mongoose';
import * as bcrypt from 'bcrypt';
import { CreateUserDto } from '../dto/create-user.dto';
import { UpdateUserDto } from '../dto/update-user.dto';
import { User, UserDocument } from '../schemas/users.schemas';
import { JwtService } from '@nestjs/jwt';
import { AuthenticateUserDto } from '../dto/authenticate-user.dto';
import { FavoriteUserDto } from '../dto/favorite-user.dto';

@Injectable()
export class UsersService {
  constructor(
    @InjectModel(User.name) private model: Model<UserDocument>,
    private jwtService: JwtService,
  ) {}

  async createUser(userData: CreateUserDto) {
    const user = await this.model.findOne({ email: userData.email });
    if (!!user) {
      throw new BadRequestException({ error: 'User already registered' });
    }
    if (userData.password !== userData.password_confirm) {
      throw new BadRequestException({ error: 'Password mismatch' });
    }
    const salt = await bcrypt.genSalt(Number(process.env.SALT_ROUNDS));
    const hash = await bcrypt.hash(userData.password, salt);
    const createdUser = new this.model(
      Object.assign(userData, {
        password: hash,
        isVerify: false,
        isConnect: false,
      }),
    );
    await createdUser.save();
    return {
      user: {
        id: createdUser.id,
        nick: createdUser.nick,
        email: createdUser.email,
      },
      access_token: this.jwtService.sign(
        {
          email: user.email,
          id: user._id,
        },
        { expiresIn: '1d' },
      ),
    };
  }

  async login(userData: AuthenticateUserDto) {
    const user = await this.model.findOne({ email: userData.email });
    if (!user) {
      throw new BadRequestException({ error: 'User not registered' });
    }
    if (!bcrypt.compare(user.password, userData.password)) {
      throw new BadRequestException({ error: 'Wrong password' });
    }
  }
}

```

```

const access_token = await this.jwtService.sign(
  {
    email: user.email,
    id: user._id,
  },
  { expiresIn: '1d', secret: process.env.JWT_SECRET_ETERNAL_TOKEN },
);
const refresh_token = await this.jwtService.sign(
  {
    email: user.email,
    id: user._id,
  },
  { expiresIn: '100d', secret: process.env.JWT_SECRET_REFRESH_TOKEN },
);
return {
  user: {
    id: user.id,
    nick: user.nick,
    email: user.email,
  },
  access_token,
  refresh_token,
};
}

async refreshToken(token: string) {
  console.log(token);
  const { id }: any = this.jwtService.decode(token.slice(7));
  const user = await this.findById(id);
  if (!user) {
    throw new BadRequestException({ error: 'User not find' });
  }
  return {
    access_token: this.jwtService.sign(
      {
        email: user.email,
        id: user._id,
      },
      { expiresIn: '1d' },
    ),
  };
}

async deleteUser(id: string) {
  await this.model.findByIdAndDelete(id);
}

async updateUser(id: string, data: UpdateUserDto) {
  return await this.model.findByIdAndUpdate(id, {
    $set: { ...data },
  });
}

async findAll() {
  return await this.model.find().exec();
}

async findById(id: string | Types.ObjectId) {
  return await this.model.findById(id);
}

```

```

async getUserByToken(token: string) {
  const { id }: any = this.jwtService.decode(token.slice(7));
  const user = await this.findById(id);
  delete user.password;
  return user;
}

async findByName(name: string) {
  return await this.model.find({ name }).exec();
}

async findByNick(nick: string) {
  return await this.model.findOne({ nick });
}

async findByEmail(email: string) {
  return await this.model.findOne({ email });
}

async changeStateLocationFavorite(data: FavoriteUserDto) {
  const user = await this.model.findById(data.user_id);
  const location_index = user.favorites.findIndex(
    (favorite) => favorite === data.location,
  );
  if (location_index !== -1) {
    user.favorites.splice(location_index, 1);
    user.markModified('favorites');
    await user.save();
    return data.location;
  }
  user.favorites.push(data.location);
  user.markModified('favorites');
  await user.save();
  return data.location;
}
}

```

users.controller.ts

```

import {
  Controller,
  Get,
  Put,
  Body,
  Param,
  Delete,
  Post,
  UseGuards,
  Headers,
} from '@nestjs/common';
import { UserDocument } from '../schemas/users.schemas';
import { UsersService } from './users.service';
import { UpdateUserDto } from '../dto/update-user.dto';
import { CreateUserDto } from '../dto/create-user.dto';
import { CreateUserData } from '../dto/user-data.dto';
import { JwtAuthGuard } from 'src/auth/jwt-auth-guard';
import { AuthenticateUserDto } from '../dto/authenticate-user.dto';
import { JwtRefreshGuard } from 'src/auth/jwt-refresh-guard';
import { FavoriteUserDto } from '../dto/favorite-user.dto';

@Controller('users')

```

```

export class UsersController {
  constructor(private readonly service: UsersService) {}

  @UseGuards(JwtAuthGuard)
  @Get('/user')
  findById(@Headers() { authorization }): Promise<UserDocument> {
    return this.service.getUserByToken(authorization);
  }

  @Post('/sign-up')
  sign_up(@Body() data: CreateUserDto): Promise<CreateUserData> {
    return this.service.createUser(data);
  }

  @Post('/sign-in')
  sign_in(@Body() data: AuthenticateUserDto) {
    return this.service.login(data);
  }

  @Post('/recovery')
  recovery() {
    return "";
  }

  @Post('/recovery-password')
  recovery_password() {
    return "";
  }

  @UseGuards(JwtAuthGuard)
  @Put('/update/:id')
  update(@Param('id') id: string, @Body() dateUser: UpdateUserDto) {
    this.service.updateUser(id, dateUser);
  }

  @UseGuards(JwtRefreshGuard)
  @Get('/refresh')
  refresh(@Headers() { Authorization }) {
    return this.service.refreshToken(Authorization);
  }

  @UseGuards(JwtAuthGuard)
  @Delete()
  delete(@Param('id') id: string) {
    this.service.deleteUser(id);
  }

  @UseGuards(JwtAuthGuard)
  @Post('/favorite')
  changeStateFavoriteLocation(@Body() data: FavoriteUserDto) {
    return this.service.changeStateLocationFavorite(data);
  }
}

app-jwt.strategy.ts

import { ExtractJwt, Strategy } from 'passport-jwt';
import { PassportStrategy } from '@nestjs/passport';
import { Injectable } from '@nestjs/common';

@Injectable()

```

```

export class AppJwtStrategy extends PassportStrategy(Strategy, 'app-jwt') {
  constructor() {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: process.env.JWT_SECRET_ETERNAL_TOKEN,
    });
  }

  validate(payload: any) {
    return { email: payload.email, id: payload.id };
  }
}

```

mail-jwt.strategy.ts

```

import { ExtractJwt, Strategy } from 'passport-jwt';
import { PassportStrategy } from '@nestjs/passport';
import { Injectable } from '@nestjs/common';

@Injectable()
export class MailJwtStrategy extends PassportStrategy(Strategy, 'mail-jwt') {
  constructor() {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: process.env.JWT_SECRET_MAIL_TOKEN,
    });
  }

  validate(payload: any) {
    return { email: payload.email, id: payload.id };
  }
}

```

refresh-jwt.strategy.ts

```

import { ExtractJwt, Strategy } from 'passport-jwt';
import { PassportStrategy } from '@nestjs/passport';
import { Injectable } from '@nestjs/common';

@Injectable()
export class RefreshJwtStrategy extends PassportStrategy(
  Strategy,
  'refresh-jwt',
) {
  constructor() {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: process.env.JWT_SECRET_REFRESH_TOKEN,
    });
  }

  validate(payload: any) {
    return { email: payload.email, id: payload.id };
  }
}

```

Mail.service.ts

```

import { MailerService } from '@nestjs-modules/mailer';
import { Injectable } from '@nestjs/common';
import { join } from 'path';
import { SendMail } from './interfaces/send-mail';

@Injectable()
export class MailService {
  constructor(private readonly mailerService: MailerService) {}

  sendMail({ to, subject, context, template }: SendMail) {
    this.mailerService.sendMail({
      to: to,
      subject: subject,
      template: join(__dirname, './templates', template),
      context: context,
    });
    return;
  }
}

```

Mail.module.ts

```

import { Module } from '@nestjs/common';
import { MailerModule } from '@nestjs-modules/mailer';
import { MailService } from './mail.service';
import { ConfigModule, ConfigService } from '@nestjs/config';
import { getMailConfig } from 'src/configs/mail.configs';

@Module({
  imports: [
    MailerModule.forRootAsync({
      imports: [ConfigModule],
      inject: [ConfigService],
      useFactory: getMailConfig,
    }),
  ],
  providers: [MailService],
})
export class MailModule {}

export interface SendMail {
  to: string;
  subject: string;
  context: any;
  template: string;
}

```

Auth.guard.ts

```

import {
  CanActivate,
  ExecutionContext,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { Request } from 'express';
import { jwtConstants } from './constants';

@Injectable()
export class AuthGuard implements CanActivate {

```

```

constructor(private jwtService: JwtService) {}

async canActivate(context: ExecutionContext): Promise<boolean> {
  const request = context.switchToHttp().getRequest();
  const token = this.extractTokenFromHeader(request);
  if (!token) {
    throw new UnauthorizedException();
  }
  try {
    const payload = await this.jwtService.verifyAsync(token, {
      secret: jwtConstants.external,
    });
    // □ We're assigning the payload to the request object here
    // so that we can access it in our route handlers
    request['user'] = payload;
  } catch {
    throw new UnauthorizedException();
  }
  return true;
}

private extractTokenFromHeader(request: Request): string | undefined {
  const [type, token] = request.headers.authorization?.split(' ') ?? [];
  return type === 'Bearer' ? token : undefined;
}
}

```

comments.module.ts

```

import { Module } from '@nestjs/common';
import { PassportModule } from '@nestjs/passport';
import { CommentsService } from './comments.service';
import { CommentsController } from './comments.controller';
import { MongooseModule } from '@nestjs/mongoose';
import { JwtModule } from '@nestjs/jwt';
import { Comment, CommentSchema } from './schemas/comments.schemas';
import { UsersModule } from 'src/users/users.module';
import { AppJwtStrategy } from 'src/strategies/app-jwt.strategy';

@Module({
  imports: [
    UsersModule,
    MongooseModule.forFeature([{ name: Comment.name, schema: CommentSchema }]),
    PassportModule,
    JwtModule.register({
      secret: process.env.JWT_SECRET_ETERNAL_TOKEN,
      signOptions: { expiresIn: '1h' },
    }),
  ],
  providers: [CommentsService, AppJwtStrategy],
  controllers: [CommentsController],
})
export class CommentsModule {}

```

comments.service.ts

```

import { Injectable } from '@nestjs/common';
import { InjectModel } from '@nestjs/mongoose';
import { Comment, CommentDocument } from './schemas/comments.schemas';
import { Model } from 'mongoose';
import { UsersService } from 'src/users/users.service';

```

```

import { AddCommentDto } from './dto/add-comment.dto';
import { ChangeCommentDTO } from './dto/change-comment.dto';

@Injectable()
export class CommentsService {
  constructor(
    @InjectModel(Comment.name) private model: Model<CommentDocument>,
    readonly userService: UsersService,
  ) {}

  async create(commentData: AddCommentDto) {
    const comment = await new this.model(commentData);
    const user = await this.userService.findById(commentData.user_id);
    user.messages.push(comment.id);
    user.markModified('messages');
    await user.save();
    await comment.save();

    return {
      message: comment.message,
      date_message: comment.date_message,
      rate: comment.rate,
      location_id: comment.location_id,
      nick: user.nick,
      user_id: user.id,
    };
  }

  async getAll(id: string) {
    const comments = await this.model.find({ location_id: id });
    const users_comments = await Promise.all(
      comments.map(async (comment) => {
        const user = await this.userService.findById(comment.user_id);
        return {
          message: comment.message,
          date_message: comment.date_message,
          rate: comment.rate,
          location_id: comment.location_id,
          nick: user.nick,
          user_id: user.id,
        };
      })
    );
    return users_comments;
  }

  async changeComment({
    id_location,
    id_user,
    comment: commentText,
    rate,
  }: ChangeCommentDTO) {
    const comment = await this.model.findOneAndUpdate(
      {
        location_id: id_location,
        user_id: id_user,
      },
      { message: commentText, rate: rate },
    );
    const user = await this.userService.findById(comment.user_id);
    await comment.save();
  }
}

```

```

    return {
      message: comment.message,
      date_message: comment.date_message,
      rate: comment.rate,
      location_id: comment.location_id,
      nick: user.nick,
      user_id: user.id,
    };
  }
}

```

comments.controller.ts

```

import {
  Controller,
  Post,
  Get,
  Body,
  Param,
  UseGuards,
  Put,
} from '@nestjs/common';
import { CommentsService } from './comments.service';
import { JwtAuthGuard } from 'src/auth/jwt-auth-guard';
import { AddCommentDto } from './dto/add-comment.dto';
import { ChangeCommentDTO } from './dto/change-comment.dto';
@Controller('comments')
export class CommentsController {
  constructor(private readonly service: CommentsService) {}

  @UseGuards(JwtAuthGuard)
  @Get(':id')
  async getCommentsByIdLocation(@Param('id') id: string) {
    return await this.service.getAll(id);
  }

  @UseGuards(JwtAuthGuard)
  @Put()
  async changeCommentUser(@Body() data: ChangeCommentDTO) {
    return this.service.changeComment(data);
  }

  @UseGuards(JwtAuthGuard)
  @Post()
  addCommentByIdLocation(@Body() data: AddCommentDto) {
    return this.service.create(data);
  }
}

export const CommentSchema = SchemaFactory.createClass(Comment);

export class AddCommentDto {
  message: string;
  date_message: number;
  rate: number;
  location_id: string;
  user_id: string;
}

export class ChangeCommentDTO {

```

```
id_location: string;  
id_user: string;  
comment: string;  
rate: number;  
}
```

```
export class CommentDto {  
  message: string;  
  date_message: string;  
  rate: number;  
  location_id: string;  
  user_id: string;  
}
```

ДОДАТОК Б
Слайди презентації



ЗАПОРІЗЬКА ПОЛІТЕХНІКА
НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ



Факультет комп'ютерних наук і технологій

Дипломний проект на тему:
«Програмна реалізація динамічного сайту для мандрівника»

Виконав: студент групи 110сп
БОНДАРЕНКО А.
Керівник: Доцент, к.т.н ЗЕЛЕНЬОВА І.

1/20

Рисунок Д.2 – Слайд 1

- Об'єкт дослідження – програмні засоби для сайту «Мандрівник».
- Предмет дослідження – комп'ютерні програми для перегляду місця подорожі, пам'яток та інших об'єктів.
- Мета роботи – створення програмної системи для інтерактивного перегляду місця подорожі, пам'яток та інших об'єктів.

2/20

Рисунок Д.2 – Слайд 2

Аналіз переваг та цікавих вирішень аналогів

-  Купівля квитків.
-  Допомога в оренді житла.
-  Форуми з відгуками.
-  Які допомагають побудувати туристичні маршрути.
-  Карти.

3/20

Рисунок Д.2 – Слайд 3

Аналіз інструментів для розробки.

Потреби сучасного сайту:

- Розширюваність
- Компонентний підхід
- Проста взаємодія з backend

Технічна підготовка до завдання

Вибір технології з такими підходами:

- Динамічні зміни даних та їх централізоване зберігання.
- Фреймворк з підтримкою SPA.
- Технологія відправлення даних через API.
- Можливість перевикористання компонентів або темлейтів.
- Бібліотеки темізації.

4/20

Рисунок Д.2 – Слайд 4

СТРУКТУРА ТА ПЛАН РОЗРОБКИ

- Вибір середовища розробки.
- Вибір фреймворка.
- Схема функціонування програми.

5/20

Рисунок Д.2 – Слайд 5

Вибір середовища розробки.

	<u>Безкоштовне чи платне IDE</u>	<u>Windows, Linux та macOS</u>	<u>Плагіни</u>
<u>Visual Studio Code</u>	Безкоштовне	На всіх	Вбудовуються, легко встановлюються
<u>Subline Text</u>	Безкоштовне / Платне	На всіх	Вбудовуються, але важко встановлюються та частково платні
<u>WebStorm</u>	Платне	На всіх	Заздалегідь вбудовані

Було обрано Visual Studio Code

6/20

Рисунок Д.2 – Слайд 6

Вибір середовища розробки.

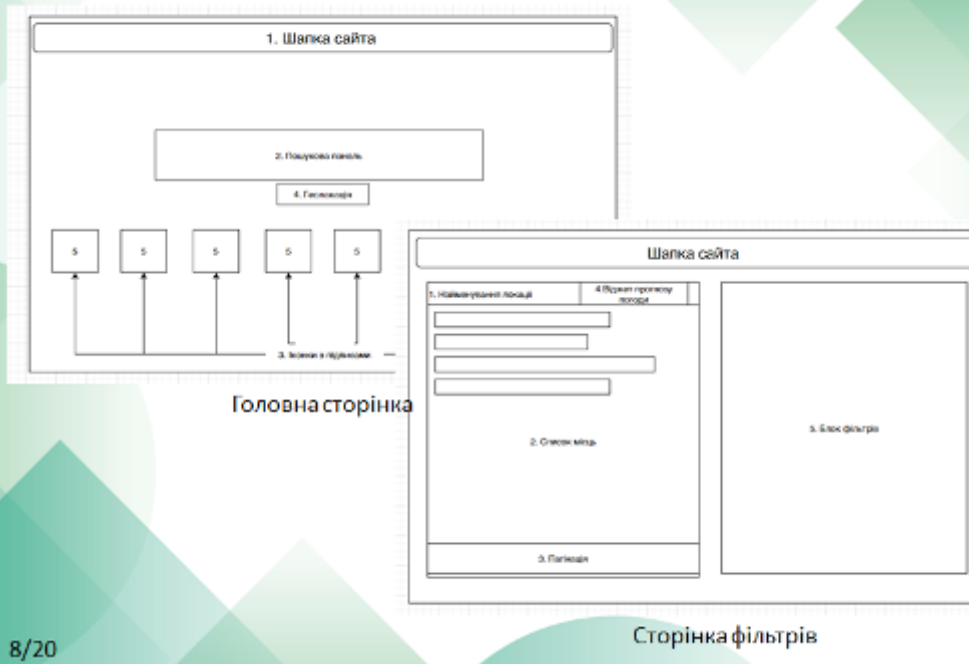
	Реактивність	Екосистема	Typescript	Навчання та популярність
React	Віртуальний DOM, користувач стежить за рендерингом	Велика кількість бібліотек	Опціональний	Функціональні компоненти та хуки надали більшої популярності, деякий синтаксис більш складний
Vue 2	Реактивність на основі ViewModel, фреймворк стежить за рендерингом	Велика кількість бібліотек	Опціональний, в проектах дуже рідко	Простий у розумінні для новачків та з дуже добре розписаною документацією
Angular	Shadow DOM	Достатня кількість але менше, ніж у інших	Написаний на ньому	Складний для новачків, typescript одна з причин

7/20

Було обрано React

Рисунок Д.2 – Слайд 7

Схеми функціонування програми.



8/20

Сторінка фільтрів

Рисунок Д.2 – Слайд 8

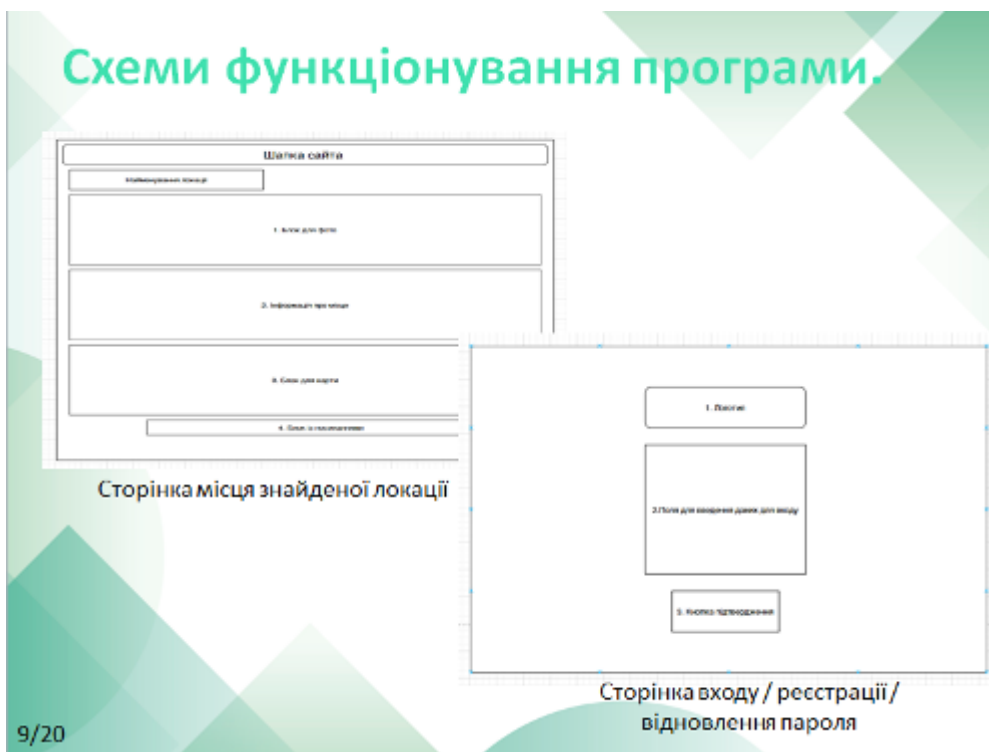


Рисунок Д.2 – Слайд 9



Рисунок Д.2 – Слайд 10

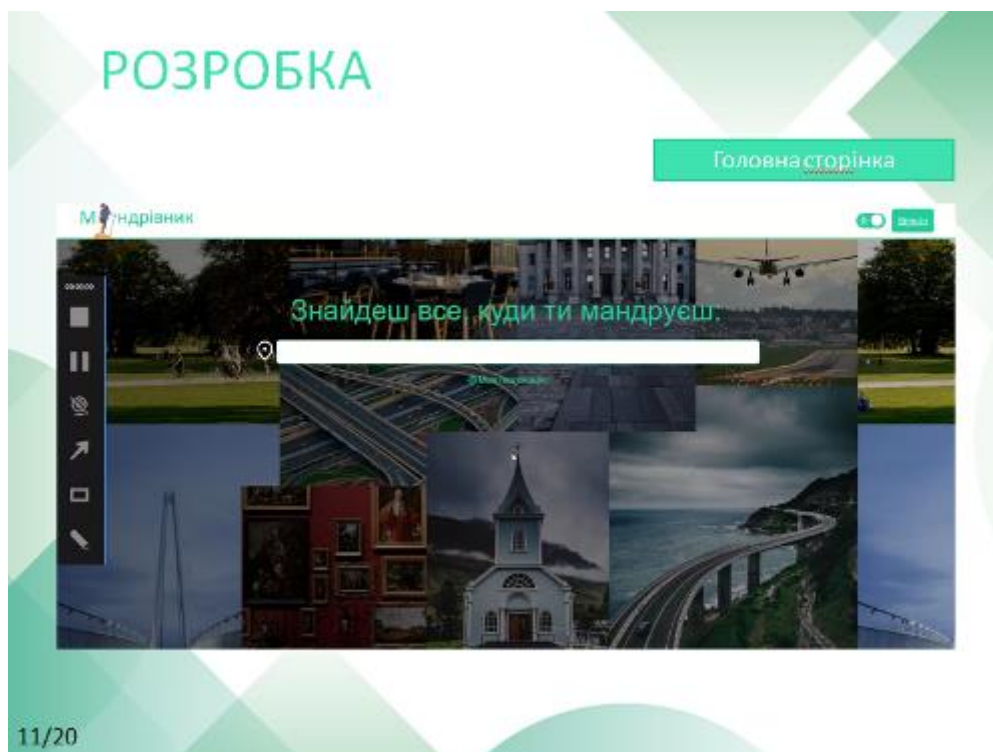


Рисунок Д.2 – Слайд 11

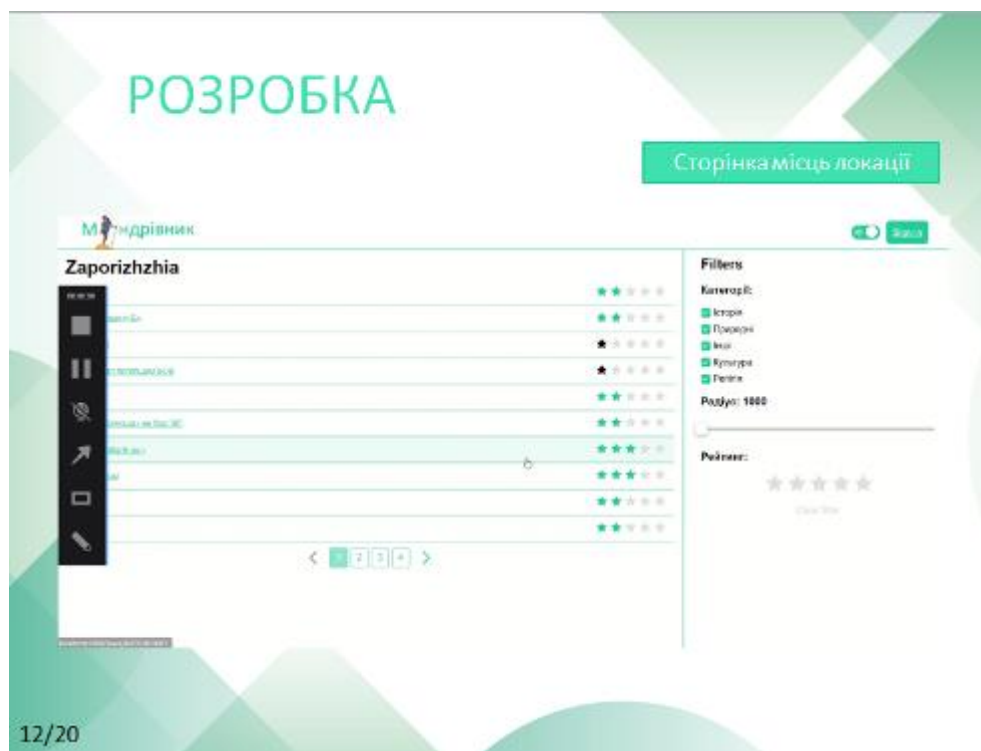


Рисунок Д.2 – Слайд 12

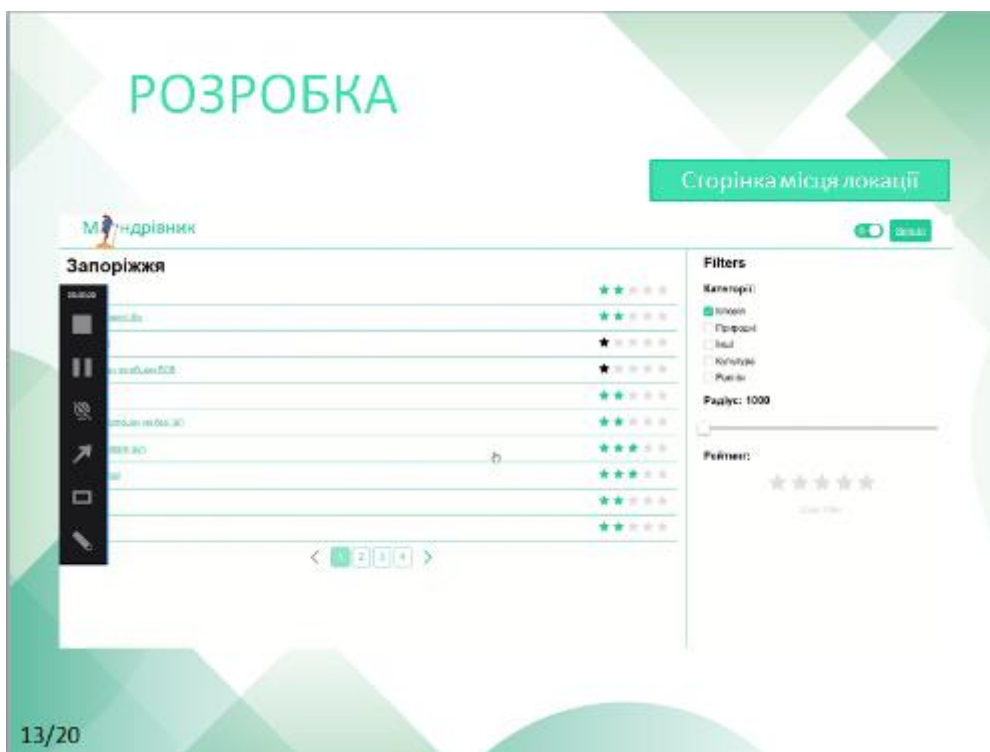


Рисунок Д.2 – Слайд 13

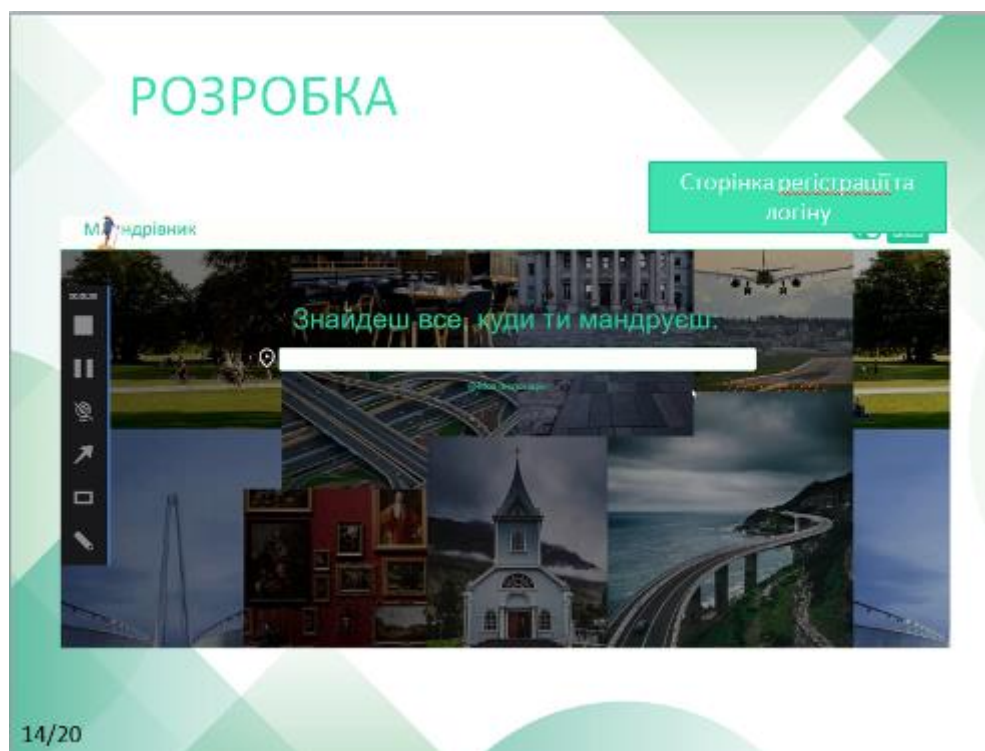


Рисунок Д.2 – Слайд 14



Рисунок Д.2 – Слайд 15

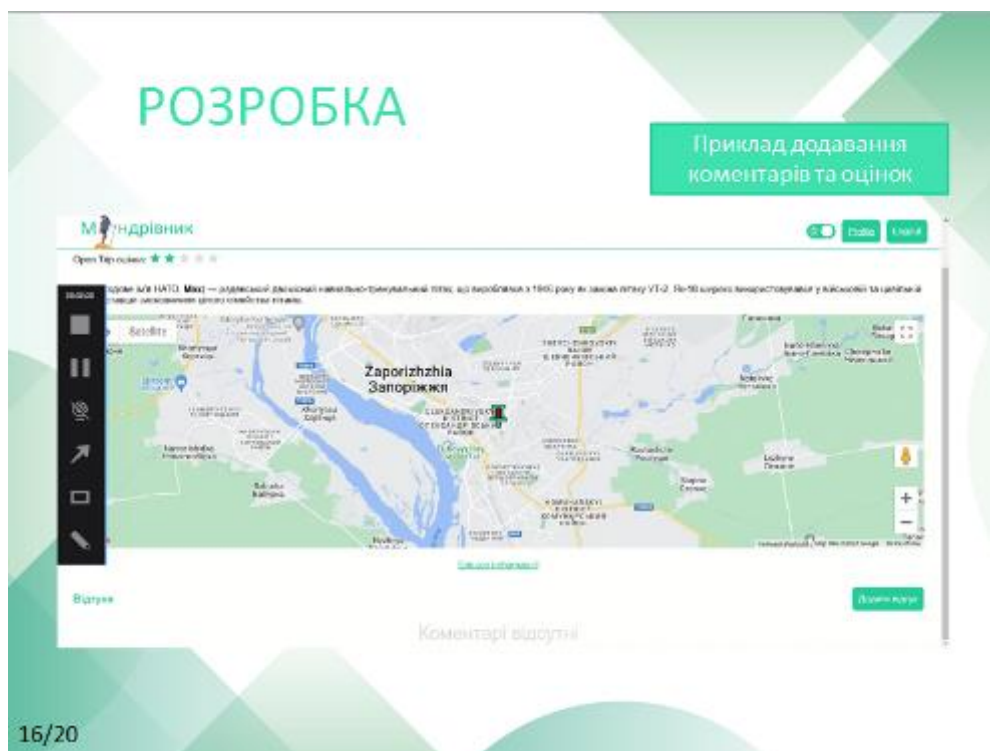


Рисунок Д.2 – Слайд 16

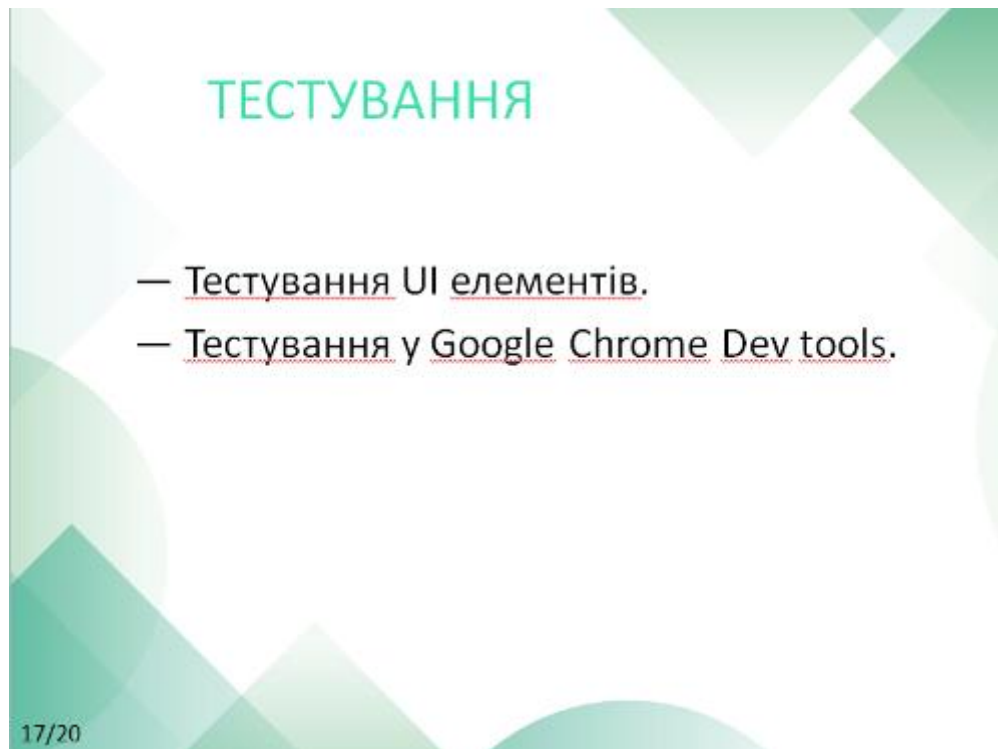


Рисунок Д.2 – Слайд 17

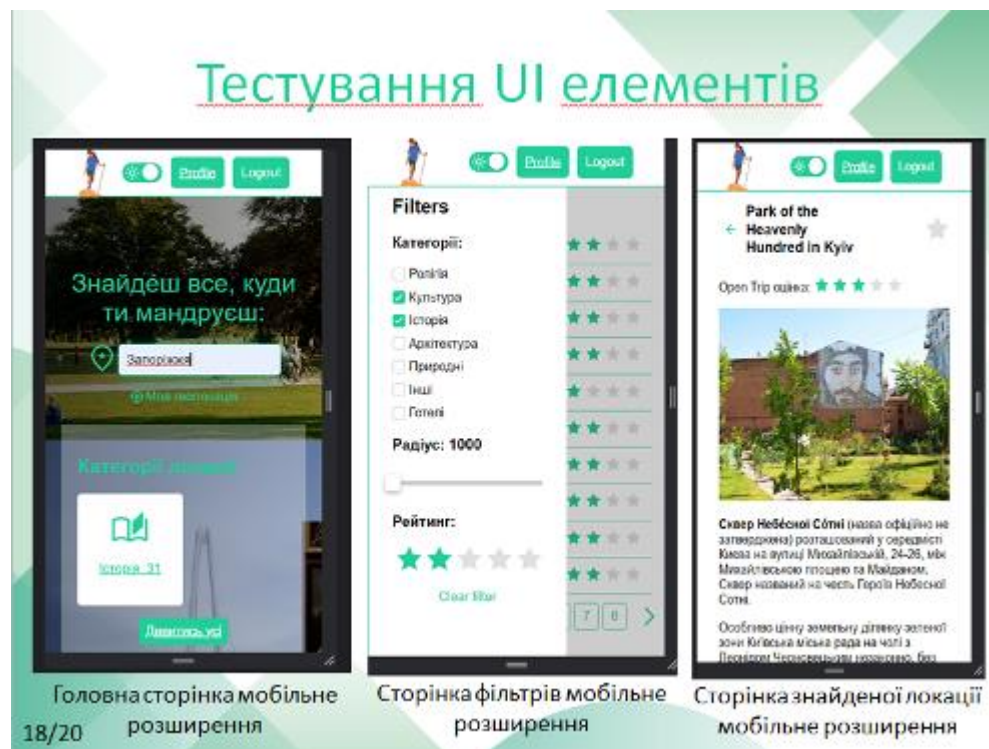


Рисунок Д.2 – Слайд 18

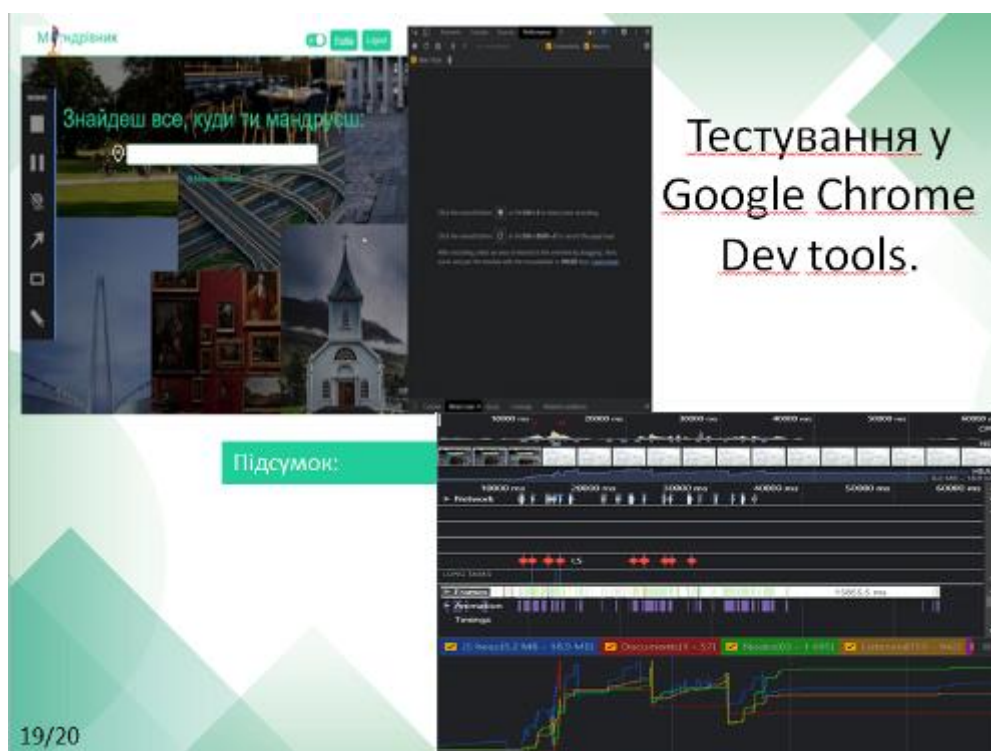


Рисунок Д.2 – Слайд 19

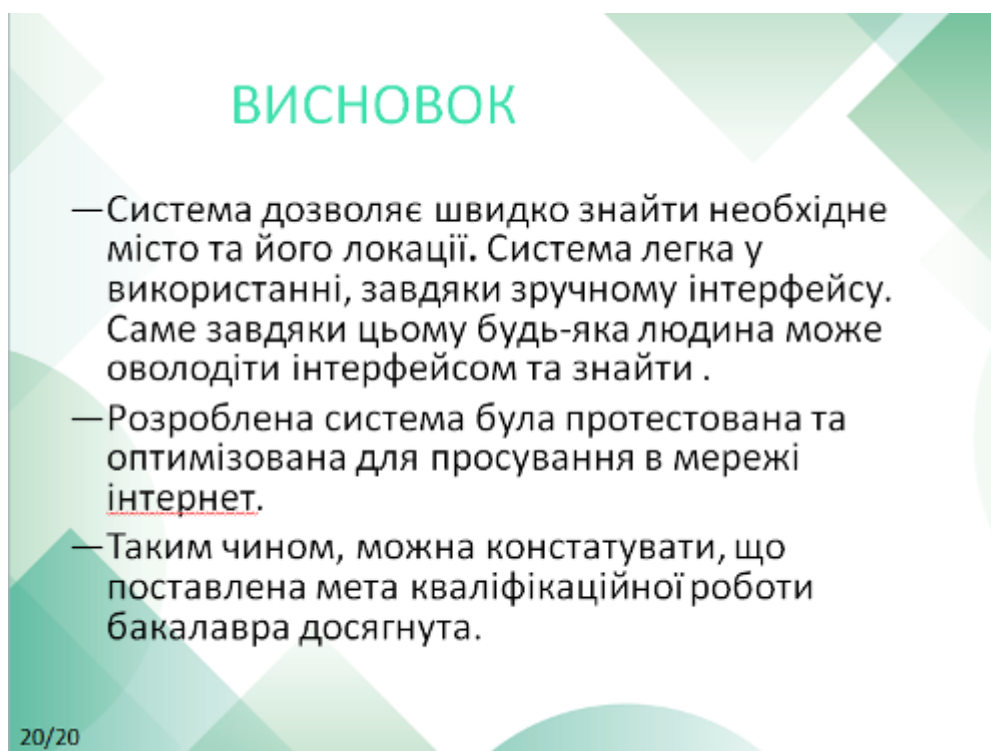


Рисунок Д.2 – Слайд 20