НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 р.

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Інженерія програмного забезпечення комп’ютерних систем”
спеціальності 121 “Інженерія програмного забезпечення”

на тему: Веб-застосунок для координації та агрегації громадських ініціатив

Виконав : студент __4__ курсу, групи ІП-04
(шифр групи)

_____Зіновий Богдан Олександрович_____
(прізвище, ім’я, по батькові) (підпис)

Керівник _____асистент Ковальчук Олександр Миронович_____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Консультант (нормоконтроль) доц. Волокита А.М._____
(назва розділу) (посада, вчене звання, науковий ступінь, прізвище та ініціали) (підпис)

Рецензент _____
(посада, науковий ступінь, вчене звання, прізвище та ініціали) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2024 р.

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Освітньо-професійна програма

“Інженерія програмного забезпечення комп’ютерних систем”

спеціальності 121 “Інженерія програмного забезпечення”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
(підпис)
“__” _____2024 р.

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Зінового Богдана Олександровича

1. Тема проєкту *Веб-застосунок для координації та агрегації громадських ініціатив*
керівник проєкту ас. Ковальчук Олександр Миронович.,
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від 27 травня 2024 року №2112-с

2. Термін здачі студентом закінченого проєкту *05 червня 2024 р.*

3. Вихідні дані до проєкту *технічна документація, теоретичні дані.*

4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)

Розділ 1. Системи для агрегації громадських ініціатив.

Розділ 2. Вибір і обгрунтування засобів реалізації веб-застосунку для координації та агрегації громадських ініціатив.

Розділ 3. Деталі розробки системи.

Розділ 4. Представлення та аналіз реалізованого веб-застосунку.

5. Перелік графічного матеріалу (з точним позначенням обов'язкових креслень) діаграма бази даних системи (функціональна схема), компонентна структура системи (структурна схема), алгоритм дій користувача (принципова схема).

6. Консультанта проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	Волокита А. М.		

7. Дата видачі завдання «05» січня 2024 р.

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*19.02.2024-10.03.2024*	
2.	*Вивчення та аналіз завдання*	*11.03.2024-17.03.2024*	
3.	*Розробка архітектури та загальної структури системи*	*18.03.2024-20.03.2024*	
4.	*Розробка структур окремих підсистем*	*21.03.2024-02.04.2024*	
5.	*Програмна реалізація системи*	*03.04.2024-26.04.2024*	
6.	*Оформлення пояснювальної записки*	*29.04.2024-04.06.2024*	
7.	*Захист програмного продукту*	*05.06.2024*	
8.	*Передзахист*	*12.06.2024*	
9.	*Захист*	*19.06.2024*	

Студент-дипломник _____ Богдан ЗІНОВИЙ
(підпис)

Керівник проєкту _____ Олександр КОВАЛЬЧУК
(підпис)

АНОТАЦІЯ

У даному бакалаврському дипломному проєкті було розроблено веб-застосунок для координації та агрегації громадських ініціатив.

Користувачі системи можуть створювати ініціативи з використанням інтерактивної мапи, коментувати звернення інших користувачів, переглядати статистику та використовувати пошук. Адміністратори можуть інформувати користувачів про стан виконання ініціатив та оновлювати їх статус.

Для реалізації клієнтської частини використано такі інструменти: React.js, JavaScript, Leaflet, Nominatim API, Material UI. Для серверної частини: NestJS, TypeScript, TypeORM, PostgreSQL, Azure Blob Storage, JWT.

Ключові слова: громадські ініціативи, веб-застосунок, інтерактивна мапа, React, JavaScript, NestJS, TypeScript, PostgreSQL, Azure Blob Storage.

ANNOTATION

In this Bachelor's Degree project, a web application for coordinating and aggregating public initiatives was developed.

Users can create initiatives using an interactive map, comment on other user’s initiatives, view statistics and use the search tool. Administrators can inform users about the progress of initiatives and update their status.

The client side was implemented using the following tools: React.js, JavaScript, Leaflet, Nominatim API, and Material UI. The server side was built with NestJS, TypeScript, TypeORM, PostgreSQL, Azure Blob Storage, and JWT.

Keywords: public initiatives, web application, interactive map, React, JavaScript, NestJS, TypeScript, PostgreSQL, Azure Blob Storage.

справки	Формат	*Значення*	*Найменування*	Кіл. листів	№ екземпля	*Додаток*
			Документація загальна			
			Знову розроблена			
	А4	***ІАЛЦ.467200.002 ТЗ***	Веб-застосунок для	4		
			координації та агрегації			
			громадських ініціатив			
			Технічне завдання			
	А4	***ІАЛЦ.467200.003 ПЗ***	Веб-застосунок для	94		
			координації та агрегації			
			громадських ініціатив			
			Пояснювальна записка			
	А4	***ІАЛЦ.467200.004 Д1***	Веб-застосунок для	1		
			координації та агрегації			
			громадських ініціатив			
			Діаграма бази даних системи			
			(функціональна схема)			
	А4	***ІАЛЦ.4672008.005 Д2***	Веб-застосунок для	1		
			координації та агрегації			
			громадських ініціатив			
			Компонентна структура			
			системи (структурна схема)			
	А4	***ІАЛЦ.467200.006 Д3***	Веб-застосунок для	1		
			координації та агрегації			
			громадських ініціатив			
			Алгоритм дій користувача			
			(принципова схема)			
	А4	***ІАЛЦ.467200.007 Д4***	Веб-застосунок для	78		
			координації та агрегації			
			громадських ініціатив			
			Текст програмного коду			

					ІАЛЦ.467200.001 ОА		
Зм	Лист	№ докум.	Підп	Дата			
Розробив		Зіновий Б.О.			Веб-застосунок для координації та агрегації громадських ініціатив Опис альбому	Літ.	Аркуш 1 / Аркушів 1
Перевірив		Ковальчук О.М.					КПІ ім. Ігоря Сікорського ФІОТ, ІП-04

ТЕХНІЧНЕ ЗАВДАННЯ
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Веб-застосунок для координації та агрегації громадських ініціатив*»

Київ – 2024

ЗМІСТ

					ІАЛЦ.467200.002 ТЗ		
		№ докум.	Підпис	Дата			
Розробив		Зіновий Б. О.			**Веб-застосунок для координації та агрегації громадських ініціатив** Технічне завдання	Літ.	Аркуш 1 / Аркушів 4
Перевірив		Ковальчук О. М.					
						КПІ ім. Ігоря Сікорського ФІОТ, ІП-04	
Н. Контр.		Волокита А. М.					
Затвердив							

1 НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Дане технічне завдання поширюється на розробку «Веб-застосунку для координації та агрегації громадських ініціатив» у межах теми дипломного проєкта бакалавра.

Областю застосування розроблюваної системи є створення інструменту, який надасть користувачам можливість створювати ініціативи, отримувати зворотній зв’язок щодо стану їх виконання, взаємодіяти з іншими користувачами для координації спільних зусиль та виокремлення найважливіших питань, що турбують громадян.

2 ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки даної системи є завдання для виконання роботи кваліфікаційно-освітнього рівня «бакалавр інженерії програмного забезпечення», який був затверджений факультетом “Інформатики та обчислювальної техніки” кафедрою обчислювальної техніки Національного технічного Університету України «Київський Політехнічний інститут ім. Ігоря Сікорського».

3 МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою та призначенням даної роботи є розробка веб-застосунку для координації та агрегації громадських ініціатив, що надасть користувачам можливість висловлювати власні ініціативи, брати участь у обговоренні ініціатив інших громадян, переглядати статистику про найбільш актуальні потреби та проблеми суспільства.

4 ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки даного дипломного проєкту є офіційні документації, публікації та статті в мережі Інтернет на дану тему, науково-технічна література, що стосуються проектування баз даних, розробки серверної та клієнтської частин застосунку, а також допоміжних інструментів, необхідних для реалізації всіх потрібних функцій системи.

5 ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

Розроблена система має виконувати такі вимоги:

- Простий та зрозумілий інтерфейс системи.
- Надати користувачам можливість створювати та переглядати ініціативи з використанням інтерактивної мапи
- Надати користувачам можливість взаємодіяти зі зверненнями інших громадян
- Надати користувачам можливість виконувати пошук ініціатив
- Надати користувачам можливість переглядати статистику
- Надати адміністраторам системи можливість здійснювати модерацію

5.2. Вимоги до програмного забезпечення

- ОС Windows, Linux або Mac.
- Visual Studio Code версії 1.82 або вище.
- Node.js версії 20.10 або вище.
- NPM версії 10.2.3 або вище.

5.3. Вимоги до апаратної частини

- ЦП не менше ніж 2 ядра архітектури x86-64/ARM.
- RAM не менше ніж 4 ГБ.
- ROM не менше ніж 16 ГБ вільного місця.
- Мережеве з’єднання з Інтернетом

6 ЕТАПИ РОЗРОБКИ

Назва етапів виконання	Термін виконання
Затвердження теми роботи	*19.02.2024-10.03.2024*
Вивчення та аналіз завдання	*11.03.2024-17.03.2024*
Розробка архітектури та загальної структури системи	*18.03.2024-20.03.2024*
Розробка структур окремих частин системи	*21.03.2024-02.04.2024*
Програмна реалізація системи	*03.04.2024-26.04.2024*
Виправлення помилок	*27.04.2024-28.04.2024*
Оформлення пояснювальної записки	*29.04.2024-04.06.2024*

ПОЯСНЮВАЛЬНА ЗАПИСКА
ДО ДИПЛОМНОГО ПРОЄКТУ

на тему: «*Веб-застосунок для координації та агрегації громадських ініціатив*»

Київ – 2024

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

ORM	(Object-Relational Mapping) Об’єктно-реляційне відображення
UI	(User Interface) Користувацький інтерфейс
API	(Application Programming Interface) Прикладний програмний інтерфейс
SPA	(Single Page Application) Односторінковий додаток
REST	(Representational State Transfer) Передача стану представлення
DTO	(Data Transfer Object) Об’єкт передачі даних
npm	(Node Package Manager) Менеджер пакетів для Node.js
JWT	(JSON Web Token) Веб-токен у форматі JSON
JSON	(JavaScript Object Notation) Об'єктна нотація JavaScript

ВСТУП

В сучасному суспільстві важливо мати засоби, за допомогою яких громадяни можуть впливати на своє оточення та привертати увагу до проблем, що їх турбують. Відкритий канал комунікації між громадянами та владою є одним з ключових аспектів демократичного суспільства, в якому кожен має змогу висловити свою позицію та зробити свій внесок у вирішенні важливих питань. В умовах швидкого розвитку технологій, зростає й потреба у створенні платформи, задачею якої була б реалізація такої можливості.

Метою даного дипломного проекту є розробка веб-додатку, який надає людям можливість ініціювати зміни в своєму оточенні, залучати увагу та підтримку до проблем, які варто вирішити. Використовуючи створений веб-додаток, користувачі зможуть позначати проблемні місця на інтерактивній мапі території України, висловлювати свої пропозиції, переглядати та обговорювати ініціативи інших користувачів. Основним завданням розробки є створення ефективного інструменту для агрегації інформації про найважливіші питання і пропозиції громадян, привертаючи до них увагу влади.

Сфера застосування даного веб-додатку є різноманітною, адже він може стати ефективним інструментом для місцевих органів самоврядування у взаємодії з громадою, допоможе соціальним організаціям у координації спільних проектів та ініціатив, а також стане важливим джерелом інформації щодо проблем, потреб та ініціатив громадян.

Отже, розробка веб-додатку для координації та агрегації громадських ініціатив має забезпечити зручний та доступний спосіб взаємодії між громадськістю і владою, сприяючи активному залученню людей у процеси вирішення питань та підтримуючи принципи демократичного суспільства.

РОЗДІЛ 1. СИСТЕМИ ДЛЯ АГРЕГАЦІЇ ГРОМАДСЬКИХ ІНІЦІАТИВ

1.1 Аналіз потреби в системах для координації та агрегації громадських ініціатив

З розвитком сучасного суспільства стає все більш актуальною потреба у створенні програмних засобів, які забезпечать ефективну взаємодію громадян з органами влади стосовно інформування та вирішення актуальних проблем. Як наслідок, зростає попит на створення систем, які б забезпечували зручний та доступний інструмент для агрегації громадських ініціатив, слугували б платформою для обговорення проблем та сприяли активному залученню громадян у рішення, що стосуються їхнього оточення.

Розробка систем для агрегації громадських ініціатив передбачає створення зручного інтерфейсу, який дозволить користувачам додавати нові ініціативи, обговорювати існуючі проблеми та спільно шукати їхні рішення.

Важливою складовою таких систем є можливість відображати географічне розташування ініціатив на карті, що дозволяє користувачам легко знаходити інформацію саме про ті проблеми, які наявні у їхніх регіонах. Також необхідно забезпечити громадян можливістю отримувати актуальну інформацію стосовно статусу їхніх звернень та змін, які відбулися в їхніх містах, а також розуміти, які саме теми турбують суспільство найбільше.

В зв'язку з цим виникає необхідність в аналізі та порівнянні існуючих систем, що вирішують подібні завдання. На сьогоднішній день вже існує кілька рішень, які надають засоби для взаємодії громадян з владою. Кожне з наявних рішень має як свої переваги, так і недоліки, а тому потребує детального аналізу та вибору найбільш оптимального рішення для конкретного випадку.

1.2 Огляд існуючих систем для агрегації громадських ініціатив

1.2.1 Урядовий контактний центр

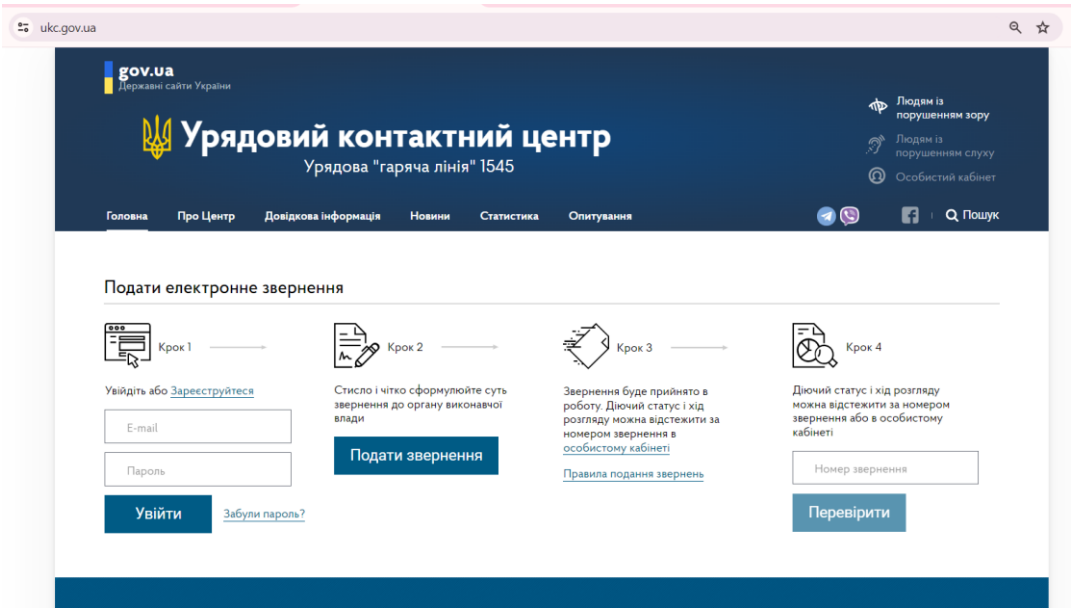


Рисунок 1.1 – Скріншот №1 додатку «Урядовий контактний центр»

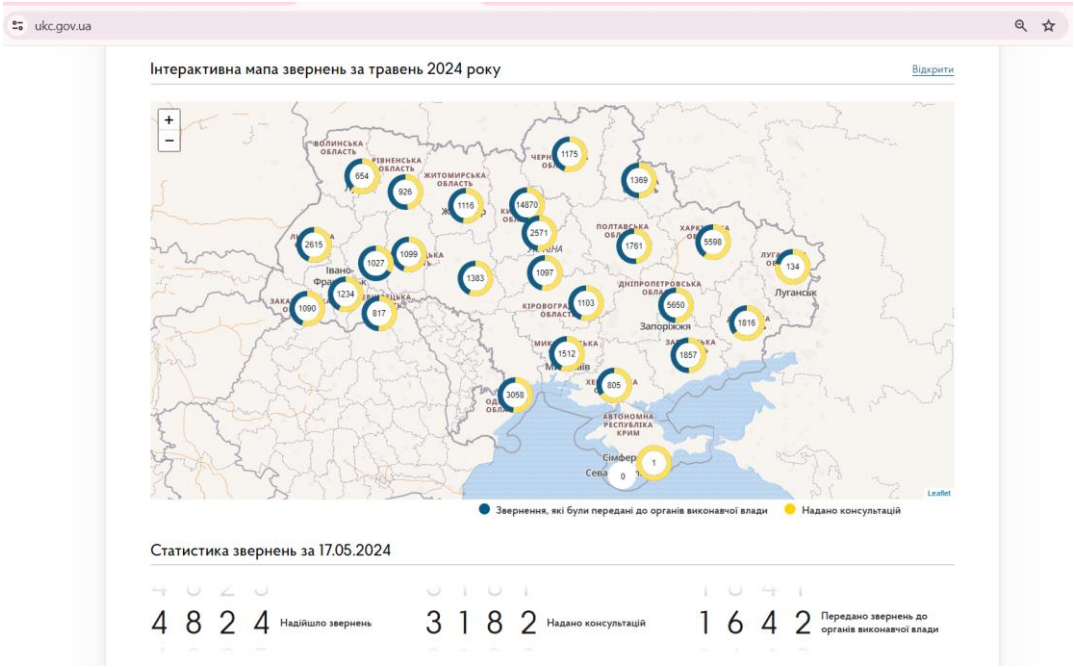


Рисунок 1.2 – Скріншот №2 додатку «Урядовий контактний центр»

Урядовий контактний центр – це державна установа, що створена з метою забезпечити оперативне реагування органів виконавчої влади на звернення громадян та їх взаємодії під час розділу звернень [1].

Сайт надає громадянам України можливість взаємодіяти з органами державної влади шляхом подання скарг, пропозицій та звернень, забезпечуючи громадян можливістю безпосередньо повідомляти про актуальні проблеми владу.

Система дозволяє оперативно інформувати владу про проблеми на місцях, контролювати процес розгляду звернень та їх подальше виконання. Також вона підвищує ефективність комунікації між громадянами та державними органами.

Інтерфейс системи простий і зручний для користувача, тому подача звернень не викликає труднощів навіть для користувачів з мінімальним досвідом роботи з комп’ютером.

Оскільки Урядовий контактний центр має інтеграцію з іншими державними сервісами, це дозволяє централізовано обробляти звернення, відповіді яких надходять в електронному вигляді. Крім того, Урядовий контактний центр постійно вдосконалюється, впроваджуючи нові технології для підвищення ефективності взаємодії з громадянами, впроваджуючи спеціалізовані «гарячі лінії» та надаючи функції для людей з обмеженими можливостями.

Наявні функції:

- Подання звернень через веб-форми;
- Можливість прикріплювати файли (фото, документи);
- Відстеження статусу звернень;
- База знань з відповідями на типові питання;
- Інтеграція з іншими державними сервісами;
- Велика кількість «гарячих ліній» для різних тематик звернень.

1.2.2 Контактний центр Києва

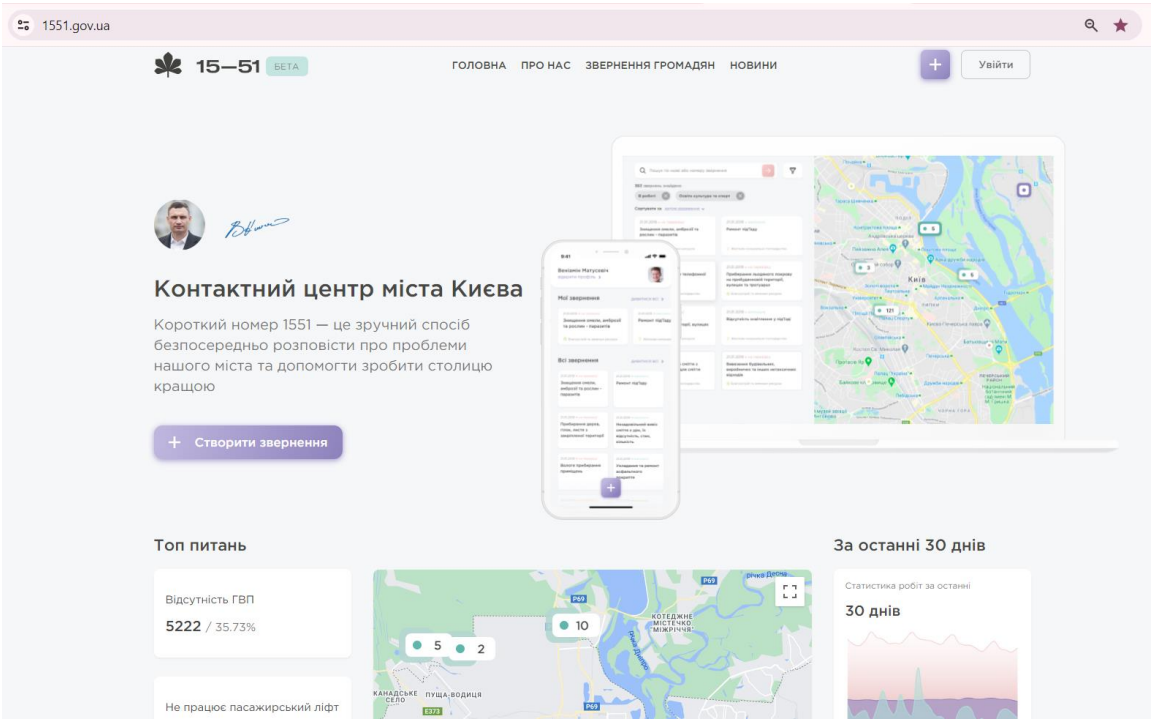


Рисунок 1.3 – Скріншот №1 додатку «Контактний центр міста Києва»

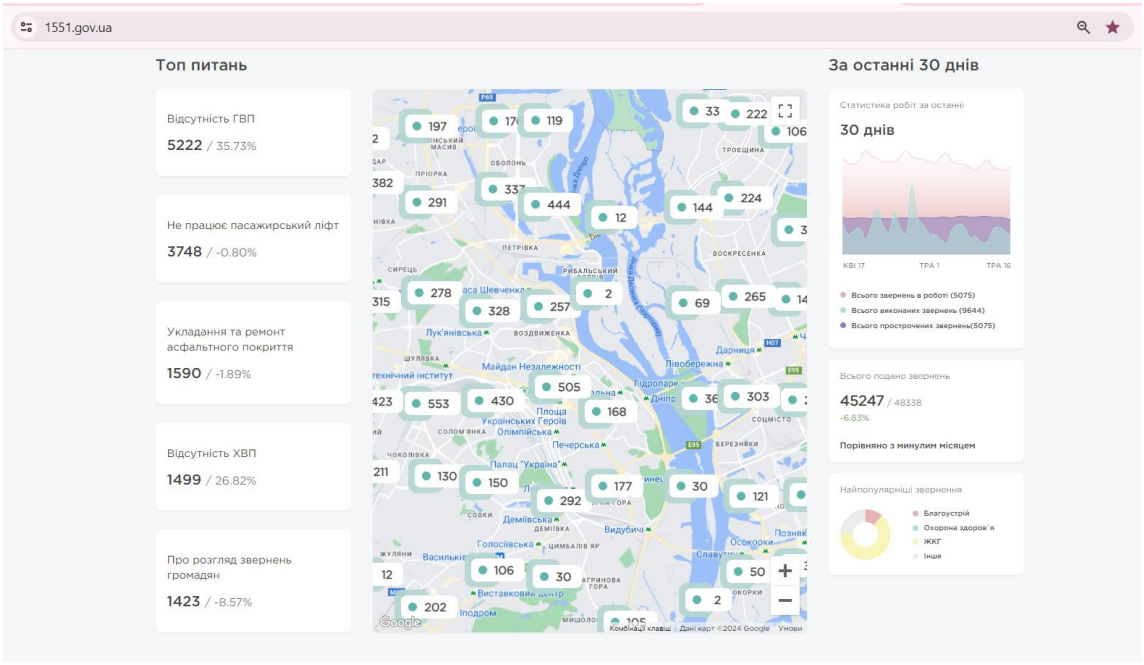


Рисунок 1.4 – Скріншот №2 додатку «Контактний центр міста Києва»

«15-51» - це комунальна бюджетна установа «Контактний центр міста Києва», яка надає зручний спосіб розповісти про проблеми міста та допомогти зробити столицю кращою [2].

Система створена для мешканців Києва, щоб вони могли повідомляти про проблеми столиці, отримувати оперативну допомогу від відповідних служб та очікувати вирішення необхідних питань. Це сприяє підтримці життєдіяльності міста та швидкому реагуванню на потреби його жителів.

Загалом користувачі використовують систему для повідомлень про проблеми з комунальними послугами, незадовільний стан окремих участків доріг, освітленням вулиць тощо. Авторизуватися можна з використанням мобільного телефону. Користувачі додають місце проблеми використовуючи інтерактивну карту та можуть бачити звернення інших громадян. Система надає зворотній зв'язок між громадянами та міськими службами, зазвичай сприяючи оперативному вирішенню проблем.

Інтерфейс веб-додатку інтуїтивно зрозумілий, подання заявок є швидким і простим процесом. Це забезпечує високу залученість користувачів та підвищує ефективність комунікації з міськими службами.

Користувачі отримують сповіщення про статус своїх звернень через смс-повідомлення або електронну пошту. Відповіді та звіти виконання звернень також доступні в особистому кабінеті користувача на сайті.

Наявні функції:

- Можливість подавати звернення через веб-сайт, мобільний додаток або телефон;
- Інтеграція з картами для точного визначення місця проблеми;
- Відстеження статусу звернень в реальному часі;
- Звіти про виконані роботи;
- Наявність статистики.

1.2.3 Карта Відновлення

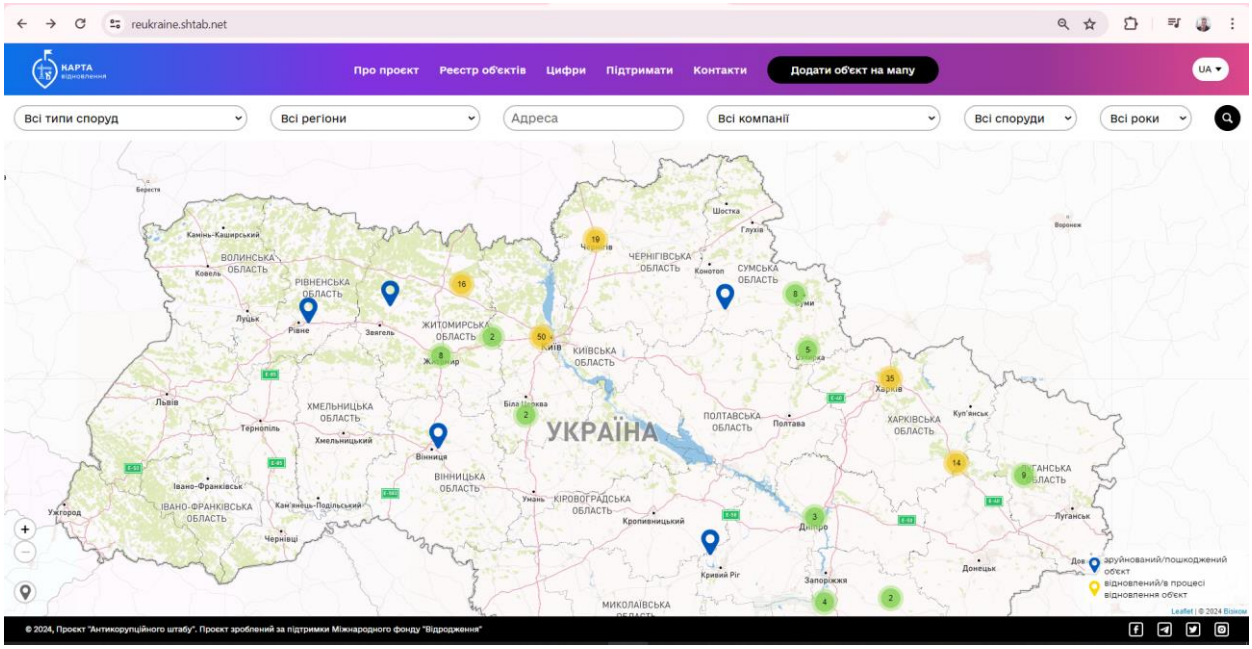


Рисунок 1.5 – Скріншот №1 додатку «Карта Відновлення»

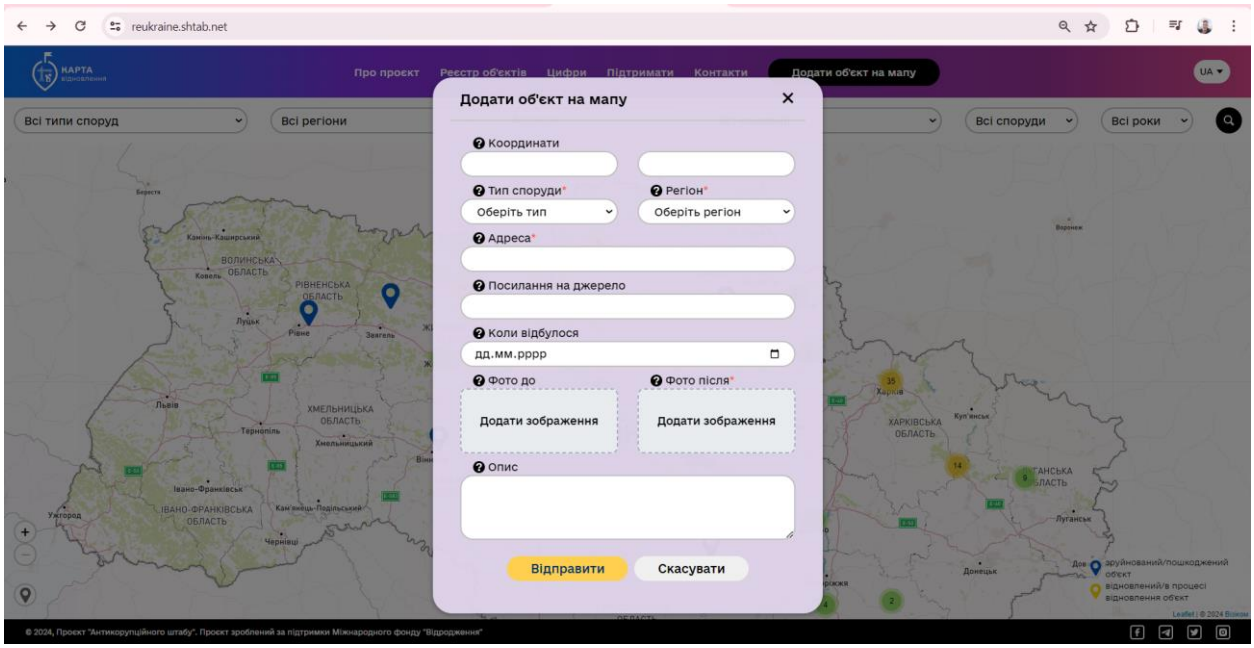


Рисунок 1.6 – Скріншот №2 додатку «Карта Відновлення»

Карта відновлення – це проект, команда якого працює над збором інформації про зруйновані та відновлені об’єкти України. Кожен об’єкт додається на інтерактивну мапу, містить фотографії, опис, історію, період відновлення тощо [3].

Система спрямована на документування та демонстрацію руйнувань внаслідок військових дій в Україні, а також координацію зусиль з відновлення зруйнованої інфраструктури. Вона покликана зосередити зусилля на процесі відновлення та забезпечити його більшу прозорість.

Наявна інформація про масштаби руйнувань та необхідні ресурси для відновлення сприяє координації між різними волонтерськими організаціями та урядовими установами, внаслідок чого ефективніше розподіляються ресурси для відбудови зруйнованих об’єктів.

Використовуючи інтерфейс інтерактивної карти, досить зручно знаходити інформацію про конкретні місця руйнувань та відновлювальні роботи, аналізувати міста, що зазнали найбільшого впливу від бойових дій.

Система допомагає привернути увагу міжнародної спільноти до проблем України, надає актуальну інформацію про стан руйнувань. Також вона використовується для координації допомоги та ефективного розподілу ресурсів серед організацій, що займаються відновленням.

Наявні функції:

- Інтерактивна карта з мітками руйнувань;
- Інформація про статус відновлювальних робіт;
- Можливість подання звітів про нові випадки руйнувань;
- База даних волонтерських та урядових організацій, які беруть участь у відновленні;
- Можливість додавати власні зруйновані об’єкти з фотозвітами.

1.3 Порівняння наявних систем для агрегації громадських ініціатив

1.3.1 Аналіз функцій систем з існуючими рішеннями

Проведемо порівняльний аналіз систем з існуючими рішеннями на основі необхідних функцій, що мають бути реалізовані згідно поставленої задачі до розроблюваного додатку:

Таблиця 1.1 - Порівняльна таблиця функціональності систем з існуючими рішеннями:

Функція	Система 1 (ukc.gov.ua)	Система 2 (1551.gov.ua)	Система 3 (reukraine.shtab.net)
Наявність авторизації	+	+	-
Можливість створювати власні ініціативи / повідомляти проблему	+	+	+
Доступна вся територія України, а не окремий регіон	+	-	+
Наявність інтерактивної мапи	+-	+	+
Наявність пошуку	+	+	+
Охоплює різні тематики	+	+	-
Перегляд звернень інших користувачів	-	+	+
Перегляд статистики	+	+	+

*примітка до Таблиці 1.1 - в Системі 1 наявна інтерактивна мапа, але вона використовується лише для інструмент для відображення статистики звернень до органів виконавчої влади та кількості консультацій по кожній області. Можливість використовувати карту для позначення місця своє звернення чи щоб знайти звернення інших користувачів, відсутня.

Згідно з Таблиці 1.1 можна зробити висновок, що в кожній із наявних систем відсутня принаймні одна функція, яка не реалізовує ідею, закладену в вимогах до розроблюваної системи. В Системі 1 відсутня можливість використовувати інтерактивну карту для додавання ініціатив та перегляду звернень інших громадян. Система 2 реалізовує набір необхідних функцій, але не має можливості формувати звернення поза територією міста Київ. Система 3 обмежена тематикою зруйнованих об’єктів і не надає можливість додавати ініціативи на інші тематики.

1.3.2 Переваги та недоліки розглянутих систем

Виділимо основні переваги та недоліки кожної з розглянутих систем:

Таблиця 1.2 – Переваги та недоліки систем з існуючими рішеннями:

Система	Переваги	Недоліки
ukc.gov.ua	Можливість додавати ініціативи чи повідомляти про проблему з будь-яких куточків України. Можливість відслідковувати статус заяви у своєму особистому кабінеті. Звернення подається напряму до органу виконавчої влади.	Відсутня можливість використовувати інтерактивну мапу для додавання ініціатив та перегляду розташування звернень. Також немає можливості переглядати звернення інших громадян.

Продовження таблиці 1.2:

1551.gov.ua	Зручний та приємний інтерфейс, наявність інтерактивної карти з відображенням звернень інших громадян. Наявний розділ статистики.	Система функціонує виключно на території міста Київ та не надає можливості додавати ініціативи поза межами міста для інших регіонів України.
reukraine.shtab.net	Можливість повідомляти про проблему на інтерактивній карті по території всієї України. Наявність зручного пошуку.	Тематика обмежена лише зруйнованими об’єктами, немає можливості додавати пропозиції, що стосуються інших сфер або висувати свої пропозиції.

Аналізуючи Таблицю 1.2 можна зробити висновок, що кожна з систем має свої певні переваги та виконує функції, для яких вона призначена. Проте, водночас, кожна з них має недоліки, які не дозволяють її вважати універсальною для поставлених задач.

ВИСНОВОК ДО РОЗДІЛУ 1

В ході порівняння існуючих рішень було розглянуто декілька наявних реалізованих систем, виділено їхні переваги та недоліки. В результаті аналізу цих рішень було зроблено висновок, що жодна з розглянутих систем на даному етапі не реалізовує всі необхідні функції одночасно, а отже - не є універсальною та не задовольняє вимог до розроблюваної системи у повній мірі.

Керуючись вищенаведеними фактами, було прийнято рішення розробити власну систему. Основною перевагою розроблюваної системи буде можливість одночасно використовувати 3 функції, а саме:

- додавати ініціативи без обмеження тематики, а не лише в одній визначеній сфері;
- використовувати інтерактивну мапу для додавання та візуального відображення всіх наявних ініціатив;
- можливість додавати ініціативи в межах усієї території України, а не лише в її окремому регіоні.

Таким чином, розроблювана система буде поєднувати в собі переваги розглянутих існуючих рішень та водночас компенсувати кожну з нереалізованих у них функцій.

РОЗДІЛ 2. ВИБІР І ОБҐРУНТУВАННЯ ЗАСОБІВ РЕАЛІЗАЦІЇ ВЕБ-ЗАСТОСУНКУ ДЛЯ КООРДИНАЦІЇ ТА АГРЕГАЦІЇ ГРОМАДСЬКИХ ІНІЦІАТИВ

2.1 Визначення вимог до засобів реалізації

Перед вибором засобів реалізації системи необхідно детально розібратися з усіма викликами, що стоять перед системою та необхідними функціями, які вона буде реалізовувати. Розробка системи координації та агрегації громадських ініціатив вимагає виконання наступних функціональних та нефункціональних вимог.

2.1.1 Функціональні вимоги до розроблюваної системи

До функціональних вимог належать:

1. Авторизація. Користувачі мають пройти реєстрацію та використовувати дані для аутентифікації щоб отримати доступ до інших ресурсів додатку.
2. Додавання нових ініціатив та повідомлень. Користувачі повинні мати можливість легко додавати нові ініціативи та повідомлення про проблеми через зручний інтерфейс. Основна ідея додатку в тому, щоб робити це з використанням інтерактивної карти.
3. Географічна прив'язка ініціатив до місця на карті. Кожна додана ініціатива має мати прив’язку до географічних координат для зручного відображення на карті.
4. Категоризація ініціатив. Користувачі повинні мати можливість вибирати теми та категорії для своїх ініціатив.
5. Коментування, вподобання та обговорення ініціатив інших користувачів. Система повинна підтримувати функцію коментування та

обговорення ініціатив. Також має бути наявна функція вподобання ініціатив та коментарів, щоб мати змогу сортувати їх за популярністю та виділяти найбільш актуальні теми.

6. Відображення ініціатив у вигляді міток на карті. Система повинна відображати ініціативи на інтерактивній карті для наочної демонстрації проблемних місць, щоб користувачі могли проаналізувати, в яких куточках країни подається найбільше звернень.
7. Пошук та фільтрація. Інструменти для ефективного пошуку та фільтрації ініціатив за різними параметрами. Зокрема користувач повинен мати змогу відобразити свої власні ініціативи, список найпопулярніших ініціатив. Також має бути реалізовано пошук за ключовими словами та фільтрацію за певними параметрами, такі як тема ініціативи, період дати в який вона могла бути додана, її статус тощо.
8. Управління файлами зображень. Можливість завантаження та зберігання зображень до ініціатив та коментарів.
9. Відображення статистики. Має бути наявна інформація про статистику звернень, така як найпопулярніші теми, що турбують громадян, міста з найбільшою кількістю звернень, загальна кількість звернень тощо.

Зважаючи на описані вище функціональні вимоги, варто зазначити, що розроблювана система для координації та агрегації громадських ініціатив складатиметься з двох частин – серверної та клієнтської. Клієнтська частина веб-застосунку буде відповідати за зручний інтерфейс додатку для взаємодії з користувачем, а серверна буде зберігати всі дані що стосуються системи, а саме інформацію про користувача, ініціативи тощо. [4]

2.1.2 Нефункціональні вимоги до розроблюваної системи

До нефункціональних вимог належать:

1. Масштабованість. Система повинна підтримувати збільшення навантаження без втрати продуктивності.
2. Надійність та безпека. Високий рівень надійності та захисту даних користувачів.
3. Продуктивність. Висока швидкість роботи інтерфейсу та обробки даних.
4. Юзабіліті. Інтуїтивно зрозумілий UI для користувачів.

Зважаючи на вищезгадані функціональні та нефункціональні вимоги до розроблюваної системи, будуть обрані засоби її реалізації [5].

2.2 Вибір та обгрунтування засобів реалізації для серверної частини застосунку

2.2.1 Задачі, що мають вирішувати обрані засоби реалізації

Перш за все, враховуючи складність застосунку, потенційну кількість окремих сутностей в предметній області та різної бізнес логіки, важливо обрати основний фреймворк для реалізації серверної частини таким, що буде якнайкраще підтримувати модульну архітектуру додатку. Це допоможе розділити всю функціональність застосунку на незалежні окремі модулі, і як наслідок, легше масштабувати застосунок та підтримувати читабельність коду.

Іншим важливим моментом є вибір мови програмування зі строгою типізацією, оскільки строга типізація полегшує процес розробки коду, повідомляючи про помилки ще на процесі компіляції. Також строга типізація

підвищує загальну надійність системи, що є важливим для серверної частини застосунку в контексті роботи з даними.

Для збереження даних про користувача, ініціатив та інших сутностей треба обрати реляційну базу даних, оскільки система матиме виключно структуровані дані, а також структура бази даних не вимагатиме частих змін. Також для спрощення роботи з базою даних варто обрати ORM. Для збереження файлів зображень потрібно обрати сторонній сервіс, який забезпечуватиме високу надійність та доступність збережених даних.

Також для розроблюваного веб-застосунку буде необхідна система авторизації, яка забезпечуватиме авторизований доступ до ресурсів серверу користувача [6].

Зважаючи на описані вище вимоги до серверної частини застосунку, оберемо засоби реалізації.

2.2.2 NestJS

NestJS - це прогресивний фреймворк для Node.js, який використовує TypeScript та модульну архітектуру, забезпечуючи розробникам інструменти для створення масштабованих і ефективних серверних додатків, реалізовуючи принципи Rest API. [7] Також сам фреймворк реалізовує необхідні патерни проектування, що полегшує процес розробки [8].

Основні патерни та їх використання:

- Модульна архітектура: Дозволяє розділяти функціональність на незалежні модулі, що полегшує масштабування та підтримку коду.
- Інтерцептори: Використовуються для обробки запитів до та після їх виконання, що зручно для логування, обробки помилок.

- Гуарди: Забезпечують захист маршрутів сервера шляхом перевірки авторизації та ролей користувачів, що критично для безпеки додатку.
- Провайдери та ін’єкція залежностей: Дозволяють легко керувати залежностями між компонентами, полегшуючи зв’язність та тестуванню коду.
- Контролери, сервіси та репозиторії: Контролери обробляють HTTP-запити, сервіси містять бізнес-логіку, а репозиторії працюють з базою даних через TypeORM, забезпечуючи чіткий розподіл обов'язків.

Серед конкурентів можна виділити Express.js, але він менш структурований, потребує додаткових налаштувань для забезпечення масштабованості та модульності. Оскільки для розроблюваного застосунку в основу буде покладена саме модульність та уже реалізовані патерни, згадані вище – то NestJS буде найкращим рішенням.

Ці патерни роблять NestJS ідеальним для розроблюваного проекту, забезпечуючи структуру, яка легко підтримується та масштабується, а також дозволяє швидко додавати нові функції, якщо з’явиться така необхідність.

2.2.3 TypeScript

TypeScript додає статичну типізацію до JavaScript, що дозволяє виявляти помилки на етапі компіляції, поліпшуючи якість та підтримуваність коду [9].

JavaScript недоцільно використовувати, оскільки відсутність статичної типізації ускладнює підтримку проєкту та сам процес розробки коду.

Завдяки статичній типізації та сучасним можливостям, TypeScript підвищує надійність та ефективність розробки, забезпечуючи безпеку коду та зручність рефакторингу.

2.2.4 PostgreSQL

PostgreSQL - це потужна реляційна база даних з підтримкою складних запитів, транзакцій та великої кількості одночасних з’єднань. Вона забезпечує надійне зберігання даних та високу продуктивність.

Конкуренти:

- MySQL: менш потужна в обробці складних запитів та менш популярна за PostgreSQL.
- MongoDB: нереляційна база даних, краще підходить для документно-орієнтованих додатків, а оскільки структура бази даних буде структурованою і буде рідко змінюватися – використання нереляційних баз даних буде недоцільним.

PostgreSQL забезпечує високу надійність, масштабованість та підтримку складних реляційних запитів, що є критичним для нашого проекту, дозволяючи обробляти великі обсяги даних та підтримувати транзакційні операції [10].

2.2.5 TypeORM

TypeORM - це ORM для TypeScript та JavaScript, який підтримує PostgreSQL та забезпечує легку інтеграцію з NestJS. Він дозволяє працювати з базами даних у вигляді об'єктів, що значно спрощує розробку.

Серед конкурентів найпопулярнішим є Sequelize: ORM для Node.js, але його основним недоліком є те, що він має меншу підтримку TypeScript порівняно з TypeORM. Також Sequelize має меншу інтеграцію з обраним фреймворком NestJS, що може ускладнити його використання.

TypeORM забезпечує чудову підтримку TypeScript та інтеграцію з NestJS, що робить його ідеальним вибором для серверної частини додатку, спрощуючи роботу з базою даних [11].

2.2.6 Azure Blob Storage

Azure Blob Storage – це надійне та масштабоване хмарне сховище компанії Microsoft для зберігання великих обсягів неструктурованих даних. Сховище забезпечує безпечне зберігання даних з легким до них доступом.

Серед конкурентів можна виділити Amazon S3: Аналогічний сервіс від компанії Amazon, але його використання може бути дорожчим та вимагає додаткового вивчення інфраструктури Amazon, що може затримати процес розробки застосунку.

Також Azure Blob Storage інтегрується з іншими сервісами Azure, що спрощує управління та знижує витрати на інфраструктуру. Він забезпечує високу надійність та доступність даних [12].

2.2.7 JWT

JWT - JSON Web Tokens, забезпечують простий і безпечний метод авторизації користувачів без необхідності використання сторонніх сервісів. Вони дозволяють створювати токени, які містять інформацію про користувача та його права доступу.

Авторизацію з використанням JWT-токенів відносно легко реалізувати, тим не менше, така система забезпечить високий рівень безпеки. JWT є безкоштовними та повністю задовольняють вимоги проекту, дозволяючи ефективно керувати авторизацією користувачів [13].

Як альтернативу можна було б розглянути використання сторонніх сервісів, таких як Auth0, але оскільки JWT уже реалізовує те, що необхідно для простої авторизації в системі та є безкоштовним – перевага віддається йому.

2.3 Вибір та обгрунтування засобів реалізації для клієнтської частини застосунку

2.3.1 Задачі, що мають вирішувати обрані засоби реалізації

Перш за все, варто обрати фреймворк/бібліотеку для створення інтерфейсів користувача. Основними критеріями вибору мають стати відносно простий поріг входу у технологію та гнучкість у побудові архітектури.

Оскільки основною функцією додатку буде використання інтерактивної карти, необхідно обрати інструмент для зручного використання карт. Враховуючи, що користувачі будуть багато взаємодіяти з мапою, важливо мінімізувати витрати на сервіс чи бібліотеку, що надаватиме необхідну функціональність. Не менш важливою є наявність кастомізації міток на карті в залежності від різних типів інформації, наприклад, різні піктограми міток для різних статусів ініціатив.

Також з використанням інструменту для відображення карт, необхідно знайти способи знаходження координат місцевості шляхом геокодування та спосіб обмежити користувачеві можливість додавати ініціативи до географічних координат, що лежать поза територією України.

Враховуючи наведені факти вище, обрано засоби реалізації клієнтської частини веб-додатку для координації та агрегації громадських ініціатив.

2.3.2 React

React - це популярна бібліотека для побудови інтерфейсів користувача, яка забезпечує високу продуктивність та гнучкість завдяки використанню компонентів. Вона дозволяє створювати швидкі та інтерактивні веб-додатки.

React базується на компонентах, що дозволяє легко повторно використовувати код та швидко впроваджувати нові функції.

Конкуренти:

- Angular: більш важкий, повноцінний фреймворк з високим порогом входу.
- Vue.js: легкий фреймворк, але має меншу спільноту розробників.

React забезпечує простоту та гнучкість у розробці інтерфейсів, що робить його ідеальним для нашого проекту. Компонентний підхід дозволяє легко повторно використовувати код та швидко впроваджувати нові функції [14]. Оскільки поріг входу та гнучкість архітектури є основними критеріями вибору React підходить для заданих цілей найкраще.

2.3.3 JavaScript

JavaScript - це мова програмування, яка використовується для створення динамічних і інтерактивних елементів на веб-сторінках. Вона є однією з основних технологій у веб-розробці поряд з HTML та CSS. JavaScript дозволяє додавати функціональність до веб-додатків, таку як обробка подій, маніпуляція DOM, асинхронні запити до серверу тощо [15].

Як альтернативу можна було б розглянути TypeScript - розширення JavaScript, яке додає типізацію та інші покращення. Але для клієнтської частини веб-додатку типізація не є такою критичною, як для серверної частини. Це зумовлено тим, що клієнтський код зазвичай обробляє меншу кількість складної логіки і не має таких високих вимог до надійності та масштабованості. Більш важливою є швидкість розробки та гнучкість у внесенні змін, що забезпечується використанням JavaScript. Також, оскільки JavaScript є «рідною» мовою для браузерів, немає необхідності у додатковій компіляції коду, що прискорює процес розробки та відладки.

Оскільки JavaScript забезпечує простоту у використанні та широкі можливості для розробки веб-додатків, це робить його ідеальним вибором для розроблюваного проєкту.

2.3.4 Leaflet

Leaflet - це легка бібліотека для інтерактивних карт, яка забезпечує високу продуктивність. Вона дозволяє легко інтегрувати карти у веб-додатки. Також бібліотека дозволяє кастомізувати маркери, які додаються на карту, що є необхідним для реалізації функціоналу додатку у повній мірі.

Конкурентом може слугувати Google Maps API, оскільки він більш потужний, але він дорожчий і складніший у налаштуванні.

Leaflet є простим у використанні та ефективним інструментом для відображення карт, що робить його ідеальним для нашого проекту, забезпечує високу продуктивність та легкість інтеграції [16].

Головною перевагою цієї бібліотеки порівняно з Google Maps API є те, що Leaflet – безкоштовний інструмент, а отже при активному використанні інтерактивної карти великою кількістю користувачів не буде стягуватися плата. Цей момент є ключовим у виборі інструменту для взаємодії з картами, оскільки обидві технології мають необхідний функціонал для роботи застосунку.

2.3.5 Nominatim API

Nominatim API забезпечує безкоштовне та надійне геокодування на основі даних OpenStreetMap. Він дозволяє перетворювати адреси у географічні координати та навпаки. Цей інструмент є необхідним для зручного пошуку в додатку населених пунктів. Використовуючи рядок пошуку можна отримати координати запитуваних міст. Також цей сервіс буде використано для

максимально точної перевірки належності обраних координат на карті території України, щоб не можна було ставити мітки на карті поза її кордонами [17].

Конкуренти:

- Google Geocoding API: дорожчий і має обмеження на кількість безкоштовних запитів.
- Mapbox Geocoding API: - потужний, але може бути дорогим у використанні.

Отже, Nominatim API забезпечує економічну ефективність та надійність, що робить його ідеальним вибором для проєкту. Він дозволяє легко здійснювати геокодування без додаткових витрат та обмежень на частоту запитів.

2.3.6 JSON

Використання JSON файлу для зберігання координат України забезпечує легкий доступ та швидке завантаження даних. Це дозволяє зберігати координати у зручному форматі та швидко обробляти їх у додатку.

Як альтернативу можна було б розглянути базу даних з географічною інформацією. Недоліком такого підходу є те, що такі бази даних більш складні у налаштуванні та потребують більше ресурсів сервера. У цьому немає необхідності, тому що завантаживши на клієнтську частину JSON файл невеликого розміру можна повністю виконати функцію відображення кордонів України, без додаткових запитів до сторонніх сервісів та бази даних.

JSON файл забезпечує простоту у використанні та легкість інтеграції з іншими компонентами системи, дозволяє швидко зберігати та обробляти координати, забезпечуючи високу продуктивність додатку. Сам файл з координатами України буде завантажено з відкритих джерел.

2.3.7 Material UI

Material UI - це популярна бібліотека компонентів для React, яка реалізує концепцію Material Design від Google. Вона забезпечує набір готових компонентів, які можуть бути використані для швидкого створення простих та функціональних інтерфейсів користувача, що і буде використано в клієнтській частині розроблюваного для спрощеного створення окремих компонентів інтерфейсів або їх складових в одному мінімалістичному стилі.

Material UI підтримує кастомізацію, що дозволяє легко адаптувати компоненти до дизайну конкретного проекту. Також Material UI забезпечує швидку розробку інтерфейсів користувача та дозволяє створювати сучасні та зручні інтерфейси з мінімальними зусиллями [18].

ВИСНОВОК ДО РОЗДІЛУ 2

В даному розділі було сформовано вимоги до засобів реалізації системи для координації та агрегації громадських ініціатив. Загалом, описано основні функціональні та нефункціональні вимоги до системи.

До функціональних вимог належать:

- Система авторизації
- Додавання нових ініціатив та повідомлень
- Географічна прив'язка ініціатив до місця на карті
- Категоризація ініціатив
- Коментування та вподобання ініціатив інших користувачів
- Відображення ініціатив
- Пошук та фільтрація
- Управління файлами
- Відображення статистики

До нефункціональних вимог системи належать:

- Масштабованість
- Надійність та безпека
- Продуктивність
- Юзабіліті

Було визначено, що веб-застосунок має складатися з клієнтської та серверної частин. До кожної з частин застосунку було сформовано перелік задач та вимог до інструментів реалізації, якими потрібно керуватися при виборі відповідних технологій. В результаті, обрано та обгрунтовано наступні технології та підходи для реалізації системи.

Серверна частина:

- NestJS – прогресивний фреймворк для Node.js, призначений для створення масштабованих і ефективних серверних додатків з модульною архітектурою
- TypeScript – мова програмування, що додає статичну типізацію до JavaScript
- PostgreSQL – швидка та безкоштовна реляційна база даних з відкритим вихідним кодом
- TypeORM – ORM для TypeScript, який забезпечує легку інтеграцію з PostgreSQL та NestJS
- Azure Blob Storage – надійне та масштабоване хмарне сховище для зберігання великих обсягів даних
- JSON Web Token – забезпечують простий та безпечний метод авторизації без використання сторонніх сервісів

Клієнтська частина:

- React – популярна бібліотека для побудови інтерфейсів користувача, яка має високу продуктивність та гнучку архітектуру
- JavaScript – мова програмування
- Leaflet – бібліотека для створення інтерактивної мапи
- Nominatim API – сторонній сервіс для геокодування на основі даних OpenStreetMap
- JSON файл з географічними кордонами України.
- Material UI – бібліотека з готовим набором компонентів для React

Загалом обрані інструменти повністю задовольняють вимоги, що були сформовані на початку розділу. Це свідчить про те, що завдання вибору технологій було виконано успішно.

РОЗДІЛ 3. ДЕТАЛІ РОЗРОБКИ СИСТЕМИ

3.1 Опис реалізації серверної частини веб-застосунку

3.1.1 Загальна структура серверної частини

У попередньому розділі були описані основні функціональні та нефункціональні вимоги до розроблюваної системи, а також обрано інструменти, необхідні для реалізації цих функцій.

Перш за все, у виборі фреймворка для реалізації серверної частини застосунку, однією з найважливіших умов була підтримка модульної архітектури додатку. У основу побудови архітектури серверної частини розроблюваної системи буде покладена модульність. Кожний модуль буде реалізовувати власні функції та бізнес логіку, пов’язану зі своєю сутністю.

Структура серверної частини додатку буде складатися з наступних директорій:

- common
- config
- modules
- systems

Директорія *common* буде використовуватися для збереження різноманітних констант, декораторів, типів даних, перечислюваних об’єктів. Головною особливістю буде те, що весь програмний код, розміщений в цій директорії, буде використовуватися в багатьох програмних модулях одночасно, а отже буде загальним. Використання такого підходу є хорошою практикою, оскільки дозволяє виокремити й позначити загальним код, який буде використовуватися в різних частинах програми. Інший програмний код, який

стосуватиметься безпосередньо якоїсь сутності, буде розміщений у відповідному модулі, пов’язаному з нею.

В директорії *config* буде реалізовано модуль конфігурації. Головним завданням цього сервісу буде зручне отримання окремих змінних оточення та конфігурації бази даних з файлу конфігурацій “.env”.

В директорії *modules* знаходитимуться папки з модулями. Виходячи з описаних функціональних вимог, будуть реалізовані наступні модулі:

- модуль авторизації
- модуль токенів для оновлення сесії
- модуль користувачів
- модуль тем ініціатив
- модуль локацій ініціатив
- модуль регіонів України
- модуль ініціатив
- модуль коментарів
- модуль лайків
- модуль зображень

Кожен з перелічених вище програмних модулів реалізовуватиме маршрутизацію, виконання бізнес логіки, маніпуляції з даними, які стосуватимуться конкретної сутності. Для прикладу, модуль *Users* відповідатиме за роботу з моделлю даних *User*. В модулі міститимуться: файл моделі *User* для бази даних; файл сервісу, в якому будуть реалізовані методи для роботи з цією моделлю, звернення до бази даних, маніпуляції з даними; файл контролера, який визначатиме ендпоінти та їхні обробники, що викликатимуть відповідні методи сервісу; файл з константами для цієї сутності; папка з файлами DTO для передачі даних між підстистемами додатку, такі як DTO для створення нового користувача та оновлення даних про уже існуючого.

Інші модулі серверної частини застосунку будуть реалізовані за аналогічним принципом.

У директорії *systems* знаходитимуться модулі редяційної бази даних та хмарного сховища для збереження зображень.

Окрім цих директорій, до основних файлів серверної частини додатку будуть входити файл конфігурації зі змінними оточення *.env*, файл *app.module.ts* зі створенням головного програмного модулю та файл *main.ts*, який є точкою входу в додаток [19].

3.1.2 Підключення до бази даних

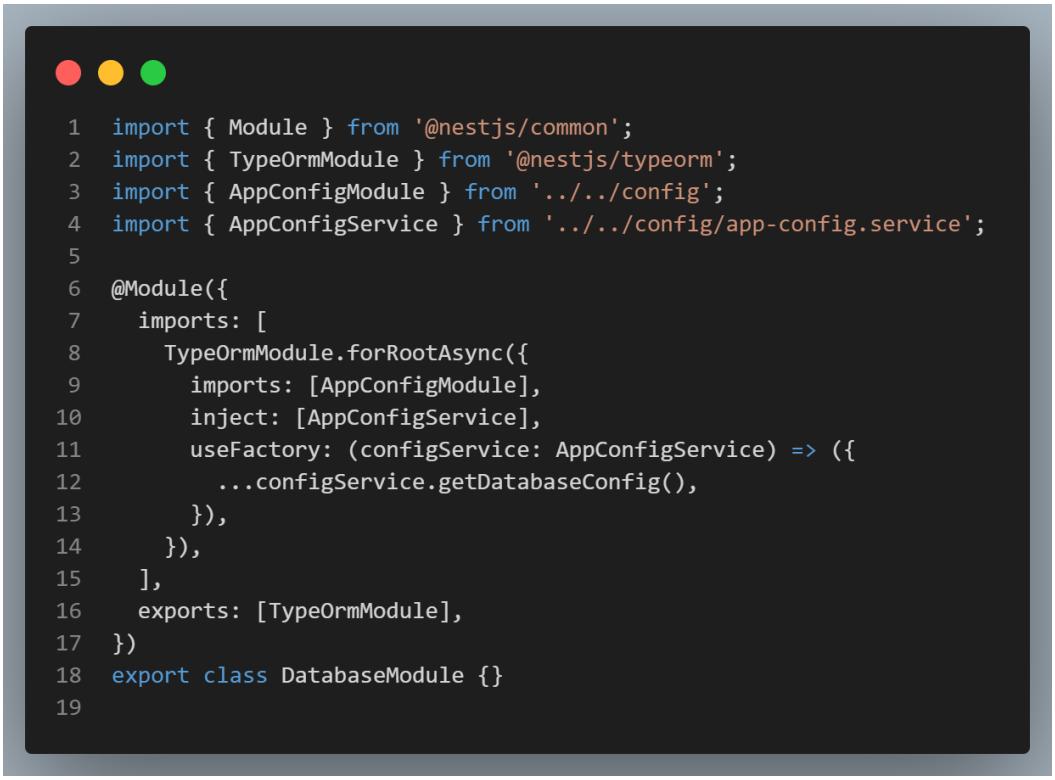
Для збереження даних в додатку буде використано реляційну базу даних PostgreSQL. Для отримання об’єкта конфігурацій для бази даних буде реалізовано модуль *AppConfigService*. Цей сервіс матиме два методи, один з яких буде отримувати змінну оточення з файлу конфігурацій проекта «.env», а інший метод формуватиме необхідний набір змінних оточення для подальшого використання об’єкта конфігурацій для підключення до бази даних PostgreSQL.

Для підключення до бази даних в проекті буде використовуватися TypeORM, а саме метод асинхронного підключення до бази даних *forRootAsync*, що прийматиме об’єкт з налаштуваннями підключення, отриманий з *AppConfigService* сервісу. Цей об’єкт містить визначені дані про назву з’єднання, тип, ім’я хоста, порт, назву бази даних, користувача, пароль та інші допоміжні опції. Також наявна інформація для TypeORM включає визначення формату назв файлів та директорій, з яких автоматично будуть знаходитися та додаватися моделі бази даних та міграції, створені за допомогою TypeORM.

Конфігурація підключення та код для створення модуля бази даних виглядають наступним чином:

```typescript
@Injectable()
export class AppConfigService {
  constructor(private configService: ConfigService) {}

  getDatabaseConfig(): TypeOrmModuleOptions {
    return {
      name: this.get('TYPEORM_CONNECTION_NAME'),
      type: this.get<'postgres'>('TYPEORM_TYPE'),
      host: this.get('TYPEORM_HOST'),
      port: this.get('TYPEORM_PORT'),
      cache: this.get('TYPEORM_CACHE'),
      logging: this.get('TYPEORM_LOGGING'),
      database: this.get<string>('TYPEORM_DATABASE'),
      username: this.get('TYPEORM_USERNAME'),
      password: this.get<string>('TYPEORM_PASSWORD'),
      extra: {
        ssl: this.get('TYPEORM_SSL'),
      },
      dropSchema: this.get('TYPEORM_DROP_SCHEMA'),
      synchronize: this.get('TYPEORM_SYNCHRONIZE'),
      migrationsRun: this.get('TYPEORM_MIGRATIONS_RUN'),
      entities: [join(__dirname, '../modules/**/*.entity.{ts,js}')],
      migrations: [join(__dirname, '../systems/database/migrations/*.{ts,js}')],
    };
  }

  get<T>(key: string): T {
    const value = this.configService.get<T>(key);

    if (!value) {
      throw new Error(key + ' environment variable is not set');
    }

    try {
      return JSON.parse(value as string);
    } catch {
      return value;
    }
  }
}
```

Рисунок 3.1 – Код сервісу «AppConfigService»



```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { AppConfigModule } from '../../config';
import { AppConfigService } from '../../config/app-config.service';

@Module({
  imports: [
    TypeOrmModule.forRootAsync({
      imports: [AppConfigModule],
      inject: [AppConfigService],
      useFactory: (configService: AppConfigService) => ({
        ...configService.getDatabaseConfig(),
      }),
    }),
  ],
  exports: [TypeOrmModule],
})
export class DatabaseModule {}

```

Рисунок 3.2 – Код модуля бази даних «DatabaseModule»

3.1.3 Підключення до хмарного сховища

Використання реляційних баз даних не оптимізовано для зберігання файлів. Також запис і читання файлів в такому випадку можуть знизити продуктивність бази даних та додатку, що її використовує. Ще однією проблемою є нагромадження великого обсягу даних, що значно ускладнює резервне копіювання та обслуговування бази даних. Тому для зберігання зображень в додатку буде використано сервіс Azure Blob Storage, який підходить для зберігання та швидкого доступу до великого обсягу неструктурованих даних.

Для реалізації даного сервісу спершу необхідно встановити бібліотеку *@azure/storage-blob*, використовуючи пакетний менеджер npm. Далі, користуючись сайтом хмарної платформи Microsoft Azure, необхідно авторизуватися, створити та налаштувати новий контейнер для ресурсу

BlobStorage. Після цього, використовуючи отримані дані для підключення до для створеного контейнеру, будуть реалізовані методи для роботи з сервісом в додатку.

Сервіс *StorageService* модуля *storage* матиме методи для завантаження файлу до хмарного сховища в контейнер та його видалення. Як результат, при завантаженні файлу зображення буде отримано шлях, за яким його можна буде відобразити на клієнті [20].

3.1.4 Проектування бази даних

В описі архітектури серверної частини застосунку було сформовано список модулів, які необхідно реалізувати. Для кожного з модулів необхідно мати модель даних – сутність, яка відповідатиме таблиці в базі даних. Сутність TypeORM – це представлення таблиці в базі даних у вигляді об’єкта. Створення таблиць відбувається з використанням опису цих сутностей, які пізніше будуть автоматично підключені до модуля бази даних, задовольняючи умови пошуку назви файлів, визначених в конфігурації бази даних. Опишемо структуру даних кожної з таблиць.

Таблиця *users* призначена для зберігання інформації про користувача. Поля*: id, name, surname, password, email, role, createdAt, updatedAt.* Зв’язки: один користувач може мати багато коментарів, ініціатив та лайків. Також перед створенням та оновленням паролю буде відбуватися його хешування. Це необхідно в цілях безпеки, щоб у базі даних не можна було переглядати незахищені дані, а можна було б лише перевірити відповідність паролю до хешу в базі даних на сервері.

Таблиця *refresh-tokens* використовується для збереження токенів, за допомогою яких можна виконати оновлення сесії користувача, якщо термін дії його токену доступу закінчився. Поля: *id, value, expiresAt, createdAt, updatedAt, userId.* Зв’язки: пов’язана з таблицею *users* зв’язком Багато-до-Одного.

Таблиця *topics* призначена для збереження тем ініціатив. Поля: *id, name, description, createdAt, updatedAt*. Зв’язки: одній темі можуть належати багато різних ініціатив.

Таблиця *regions* буде зберігати назву області та її геокод в міжнародному стандарті ISO3166-2-lvl4, який використовується для зручної фільтрації локацій по вевних областях. Поля: *id, name, geocode, createdAt, updatedAt*. Зв’язки: одному регіону належать багато різних локацій міток на карті.

Таблиця *locations* зберігає географічні координати доданої мітки на карті. Поля: *id, latitude, longitude, createdAt, updatedAt, regionId*. Зв’язки: локація має зв’язки Багато-до-Одного з таблицею *regions* та Один-до-Одного з таблицею *initiatives*.

Таблиця *initiatives* призначена для збереження всієї інформації про ініціативу. Поля: *id, title, description, status, createdAt, updatedAt, locationId, topicId, userId*. Зв’язки: має одного користувача, локацію та тему; може мати кілька зображень, коментарів та лайків.

Таблиця *comments* використовуватиметься для збереження коментарів користувачів та відповідей адміністраторів. Поля: *id, text, createdAt, updatedAt, initiativeId, userId*. Зв’язки: може мати багато лайків та зображень, належить одній ініціативі та одному користувачу.

Таблиця *likes* зберігає інформацію про наявність лайку користувача для коментарів та ініціатив. Поля: *id, createdAt, updatedAt, userId*. Зв’язки: належить одному користувачу, одній ініціативі або коментареві.

Таблиця *images* зберігає дані про завантажене в хмарне сховище зображення, які використовуються для формування посилання на відповідний ресурс з метою отримання файлу зображення. Поля: *id, filename, fileExt, fileNameWithExt, mimetype, filePath, createdAt, updatedAt*. Зв’язки: належить одній ініціативі або коментареві [21].

Також варто зазначити, що з метою нормалізації бази даних таблиці *images* та *likes* мають звязки Багато-до-Багатьох з таблицями *comments* та

initiatives, а отже для них необхідно створення зв’язуючих таблиць *comments-images, comments-likes, initiatives-images, initiatives-likes*. TypeORM автоматично створює ці таблиці якщо зв’язок між таблицями визначити як Багато-до-Багатьох, тому окремо їх створювати немає необхідності [22].

3.1.5 Створення сутностей з використанням TypeORM

Розглянемо створення таблиць з використанням TypeORM. Спершу відбувається імпорт необхідних класів та декораторів з бібліотеки “typeorm”. Далі визначається сутність, використовуючи декоратор *@Entity*, що буде перетворений в таблицю бази даних з визначеною назвою *topics*.

Використання та опис декораторів відповідає за відображення полів в базі даних. Наприклад *@PrimaryGeneratedColumn(“uuid”)* визначає, що поле *id* буде первинним ключем таблиці з автоматично згенерованим айді формату uuid, *@CreateDateColumn* та *@UpdateDateColumn* забезпечують автоматичний запис дати створення та оновлення таблиці в поля *createdAt* та *updatedAt*.

Зв’язок з таблицею *initiatives* встановлюється з використанням декоратора *@OneToMany*. Також при визначенні полів можна передавати об’єкти опцій в дані декоратори. Наприклад, можна вказати, що поле не може бути невизначеним та має бути унікальним, встановивши опції *unique* в *true* та *nullable* в *false* [23].

Приклад коду створення таблиці *topics* з використанням засобів TypeORM:

```typescript
import { ApiHideProperty, ApiProperty } from '@nestjs/swagger';
import {
  BaseEntity,
  Entity,
  Column,
  PrimaryGeneratedColumn,
  CreateDateColumn,
  UpdateDateColumn,
  OneToMany,
} from 'typeorm';
import { InitiativeEntity } from '../initiatives/initiative.entity';

@Entity({ name: 'topics' })
export class TopicEntity extends BaseEntity {
  @ApiProperty({ type: 'string', format: 'uuid' })
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @ApiProperty({ type: 'string' })
  @Column({ length: 64, unique: true, nullable: false })
  name: string;

  @ApiProperty({ type: 'string' })
  @Column({ length: 512 })
  description: string;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @CreateDateColumn({ readonly: true })
  readonly createdAt: Date;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @UpdateDateColumn({ readonly: true })
  readonly updatedAt: Date;

  @ApiHideProperty()
  @OneToMany(() => InitiativeEntity, ({ topic }) => topic, {
    nullable: true,
  })
  initiatives: InitiativeEntity[];
}
```

Рисунок 3.3 – Код створення сутності «TopicEntity»

3.1.6 Деталі реалізації сервісів

Після закінчення проектування бази даних та створення всіх сутностей, необхідно визначити сервіси, які будуть взаємодіяти з цими сутностями та виконувати логіку, необхідну для роботи додатку. Це можуть бути як прості CRUD операції, так і методи зі складнішою логікою та специфічними функціями.

Кожен сервіс буде відповідати за роботу з однією конкретною сутністю, виконуючи запити до бази даних. Але варто зазначити, що будуть випадки, коли в межах одного сервісу потрібно буде маніпулювати даними кількох таблиць одночасно, а отже використовувати в методах сервісу, що реалізовується, методи інших сервісів. Наприклад, у сервісі зображень, а саме в методі створення нового зображення необхідно буде викликати метод сервісу хмарного сховища, який завантажить переданий бінарний файл в контейнер та поверне посилання, за яким можна буде отримати до нього доступ.

Щоб мати змогу використовувати методи одного сервісу в іншому сервісі, і при цьому зберігати розділеність логіки між окремими модулям, в Nest.js існує зручний механізм ін’єкції залежностей, який базується на використанні декоратора *@Injectable()*, що робить клас сервісу доступним для ін'єкції залежностей. Сервіс визначається як залежність іншого сервісу у його конструкторі, де NestJS автоматично підключає потрібні залежності. Модулі об'єднують сервіси та інші провайдери, і для доступу до них в інших модулях використовуються експорти та імпорти модулів [24].

Взаємодія у сервісі з базою даних буде відбуватися з використанням сутності TypeORM, яка буде додаватися в сервіс через механізм ін’єкції залежностей. У цієї сутності будуть наявні зручні методи для роботи з базою даних, а саме будуть використані методи *find, findOne, create, remove* та інші. Ці методи приймають опції, що визначають критерії для даних, які потрібно знайти, оновити, створити чи видалити. Для прикладу, якщо викликати метод

find у сутності *UserEntity* та передати як опції для пошуку ім’я користувача – результат запиту поверне всі об’єкти таблиці *users*, які відповідають критеріям пошуку. Цей підхід є досить зручним, оскільки немає необхідності щоразу створювати SQL-запити вручну і можна працювати з моделями бази даних, використовуючи спрощений синтаксис.

Також ім’я помилок, що будуть виникати при виконанні запитів з некоректними даними до бази даних у сервісах, будуть взяті з визначених назв помилок у об’єкті перечислюваного типу *AppErrorMessagesEnum*.

Важливо зазначити, що майже всі сервіси реалізовуватимуть базовий набір CRUD операцій. Для прикладу, розглянемо сервіс *UsersService*. У ньому будуть реалізовані наступні методи:

- *findAll* - отримання всіх користувачів;
- *findOne* - пошуку конкретного користувача;
- *createOne* - додавання нового користувача;
- *updateOne* - оновлення інформації;
- *deleteOne* - видалення існуючого користувача.

З метою уникнення дублювання тексту опису однакових методів сервісів, описані вище методи будуть вважатися як базові CRUD операції у випадку, якщо інші сервіси реалізовуватимуть такий самий набір операцій [25]. Розглянемо детальніше кожен із сервісів.

Сервіс *UsersService* реалізовує базові CRUD операції.

Сервіс *TopicsService* реалізовує базові CRUD операції.

Сервіс *RegionsService* реалізовує базові CRUD операції.

Сервіс *LocationsService* реалізовує базові CRUD операції.

Сервіс *LikesService* реалізовує методи:

- *findCommentLike* – використовується для перевірки, чи вподобав користувач конкретний коментар;

- *createCommentLike* - створює новий лайк для коментаря, якщо користувач ще не вподобав його;
- *deleteCommentLike* - видаляє лайк коментаря.
- *countLikesByCommentId* - повертає кількість лайків для конкретного коментаря за його *id*.
- *findInitiativeLike* - використовується для перевірки, чи вподобав користувач конкретний коментар;
- *createInitiativeLike* - створює новий лайк для ініціативи, якщо користувач ще не вподобав її раніше;
- *deleteInitiativeLike* - видаляє лайк користувача для ініціативи.
- *countLikesByInitiativeId* - повертає кількість лайків для конкретної ініціативи за її *id.*

Сервіс *ImagesService* реалізовує методи:

- базових CRUD операцій. При цьому варто зазначити, що в методах для створення та видалення зображення використовується виклики методів сервісу StorageService: createOne для завантаження файлу зображення в хмарне сховище та метод deleteOne для видалення пов’язаного файлу.
- *linkImageToInitiative* – додає зв’язок конкретного зображення до певної ініціативи.
- *linkImageToComment* - додає зв’язок конкретного зображення до певного коментаря.
- *getInitiativeImages* - знаходить всі зображення, що належать певній ініціативі.

Сервіс *CommentsService* реалізовує методи:

- базових CRUD операцій. При цьому варто зазначити, що в методах для створення та видалення коментаря використовується виклики методів сервісу *ImagesService*: *linkImageToComment* для прив’язки

завантаженого зображення до конкретного коментаря та метод *deleteOne* для видалення зображень пов’язаних з коментарем при його видаленні.

- *getInitiativeComments* - використовується щоб отримати всі коментарі ініціативи за її «id».

Сервіс *InitiativesService* реалізовує методи:

- базових CRUD операцій. При цьому варто зазначити, що в методі створення ініціативи викликаються метод *createOne* сервісу *LocationsService* для попереднього створення локації та метод *linkImageToInitiative* сервісу *ImagesService* для прив’язки зображення. Також в методі для видалення ініціативи використовуються виклики методів deleteOne сервісів *ImagesService*, *CommentsService* та *LocationsService*, щоб попередньо видалити пов’язані сутності при видаленні самої ініціативи.
- *getUsersInitiatives* – знаходить всі ініціативи, створені поточним користувачем.
- *getMostLiked* – повертає певну кількість ініціатив з найбільшою кількістю вподобань.
- *findByCustomSearch* – приймає об’єкт з полями для пошуку ініціатив та створює запит до бази даних на основі наявності певних переданих параметрів пошуку.
- *changeStatus* – змінює статус ініціативи.
- *getMonthStatistics* – підраховує статистику за останній місяць: кількість ініціатив за статусами, найбільш популярні області за кількістю ініціатив, теми з найбільшою кількістю доданих ініціатив.

Структура коду сервісів на прикладі *TopicsService*:

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
  FindManyOptions,
  FindOptionsWhere,
  FindOneOptions,
  Repository,
} from 'typeorm';
import { BadRequestException, NotFoundException } from '@nestjs/common';
import { TopicEntity } from './topic.entity';
import { AppErrorMessagesEnum } from 'src/common/enums';

@Injectable()
export class TopicsService {
  constructor(
    @InjectRepository(TopicEntity)
    private readonly entityRepository: Repository<TopicEntity>,
  ) {}

  findAll(
    options: FindManyOptions<TopicEntity> = { loadEagerRelations: true },
  ): Promise<TopicEntity[]> {
    return this.entityRepository.find(options).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.TOPICS_NOT_FOUND);
    });
  }

  findOne(
    conditions: FindOptionsWhere<TopicEntity>,
    options: FindOneOptions<TopicEntity> = { loadEagerRelations: true },
  ): Promise<TopicEntity> {
    return this.entityRepository
      .findOneOrFail({
        ...options,
        where: conditions,
      })
      .catch(() => {
        throw new NotFoundException(AppErrorMessagesEnum.TOPIC_NOT_FOUND);
      });
  }

  async createOne(entity: Partial<TopicEntity>): Promise<TopicEntity> {
    const entityToCreate = this.entityRepository.create(entity as TopicEntity);
    const { id } = await this.entityRepository
      .save(entityToCreate)
      .catch(() => {
        throw new BadRequestException(
          AppErrorMessagesEnum.INVALID_REQUEST_DATA,
        );
      });
    return this.findOne({ id } as FindOptionsWhere<TopicEntity>);
  }

  async updateOne(...);

  async deleteOne(...);
}

```

Рисунок 3.4 – Код сервісу «TopicsService»

Вище було розглянуто методи сервісів, що відповідатимуть за основну логіку роботи додатку. Окрім уже описаних сервісів ще існують сервіси *RefreshTokensService* та *AuthService*, що відносяться до модуля авторизації, а тому будуть описані далі в наступному пункті.

3.1.7 Система авторизації

Для повноцінної роботи веб-застосунку необхідно реалізувати систему авторизації. Це необхідно для того, щоб захистити маршрути серверної частини від неавторизованого доступу до даних та забезпечити розподіл дозволів користувачів відповідно до їх ролей. У другому розділі було описано переваги використання JWT токенів та обрано цю технологію для реалізації системи авторизації.

Принцип роботи системи авторизації, побудованої на JWT токенах, представляє собою механізм, за якого при успішній спробі авторизації користувач отримує токен доступу – *accessToken*. Цей токен зберігається клієнтом, наприклад, в *localStorage* браузера та використовується для відправки запитів на сервер у загаловку авторизації *Authorization*. Сервер перевіряє коректність виданого токену доступу з використанням секретного рядка, що застосовувався при підписанні токену. Якщо перевірка проходить успішно – сервер надає доступ до запитуваного ресурсу користувачем, у випадку помилки сервер обробить запит як помилку неавторизованого користувача.

Іншим важливим моментом є використання не лише одного токена доступу, а і токену оновлення, який матиме більший час життя та використовуватиметься для автоматичного оновлення активної сесії клієнта без необхідності повторної авторизації. Такий підхід з використанням пари *refresh* та *access* токенів буде використано для розробки системи авторизації. *refresTokеn* буде зберігатися у спеціальній таблиці в базі даних.

Щоб покращити досвід користування застосунком, прийнято рішення зробити максимальну кількість для двох одночасних підключень для одного користувача, щоб він міг використовувати застосунок з кілької пристроїв. Така можливість реалізовується за допомогою збереження двох *refresh* токенів в базі даних. Натомість, якщо користувач захоче ввійти третій раз – його найдавніша сесія буде перезаписана [26].

Спершу обхідно реалізувати сервіс RefreshTokensService, який буде відповідати за збереження refresh токену в базі даних. Створений сервіс реалізовує наступні методи:

- *createRefreshToken* – створює запис в таблиці токенів з переданим refresh токеном, а також визначає інформацію про час життя токена, дату закінчення дії, користувача якому він належить.
- *deleteExceededRefreshTokens* – видаляє першу додану сесію користувача, якщо при записі нового токена загальна кількість сесій авторизації перевищуватиме максимально визначену кількість.
- інші загальні CRUD операції.

Далі необхідно створити сервіс *AuthService*, що реалізовуватиме методи авторизації користувача. Цей сервіс використовує методи сервісів *UsersService* та *RefreshTokensService* для співставлення введених користувачем даних з даними про користувача в базі даних, створення і видалення resfresh токенів у базі даних відповідно.

Сервіс *AuthService* реалізовує наступні методи:

- *generateTokens* – генерує пару *access* та *refresh* JWT токенів з інформацією про користувача в payload (об’єкт з даними про користувача, якому належить токен). Для генерації та верифікації токенів необхідно встановити бібліотеку *@nestjs/jwt*

- *createUserTokens* – використовує метод для генерації токенів та записує згенерований refresh токен в базу даних, викликаючи метод *createRefreshToken* сервісу *RefreshTokens*.
- *findUser* – використовує метод findOne сервісу UsersService для отримання профілю користувача.
- *signUp* – відповідає за процес реєстрації. У разі успішного створення нового користувача, повертає пару згенерованих токенів.
- *signIn* – відповідає за процес логіну. Метод перевіряє чи існує користувач з введеним email, зіставляє введені користувачем дані з відповідним записом сутності у базі даних, порівнює хеші паролей. У разі успішної авторизації повертає пару згенерованих токенів.
- *refreshTokens* – видаляє попередній refresh токен користувача та повертає нову пару згенерованих токенів.
- *signOut* – видаляє активну сесію з бази даних викликом методу видалення токену з сервісу *RefreshTokensService*.

Після створення сервісів для авторизації, необхідно створити механізм захисту маршрутів від неавторизованого доступу, який буде забезпечувати перевірку валідності токенів авторизації, що надходитимуть у заголовку авторизації. В Nest.js цей механізм реалізовується створенням стратегій авторизації для їхнього використання у захисниках маршрутів.

Спершу необхідно встановити необхідні бібліотеки *@nestjs/passport* та *passport-jwt*. Створення стратегій авторизації відбувається наслідуванням від класу *PassportStrategy* бібліотеки passport-jwt, при цьому вказується назва стратегії та реалізовується власний метод для валідації JWT токена *validate*. Також в конструкторі створюваного класу визначається секретне слово для верифікації токена та місце, звідки необхідно взяти сам токен для перевірки. У даному випадку таким місцем є заголовком авторизації запиту *Authorization*.

Оскільки в додатку використовується пара токенів, відповідно необхідно створити дві стратегії авторизації: *AccessTokenStrategy* та *RefreshTokenStrategy*.

AccessTokenStrategy буде валідувати *accessToken* використовуючи секретне слово сервера та буде використовувати сервіс *UsersService* для знаходження користувача з відповідними даними токена у базі даних. Якщо користувача не знайдено або токен не є валідним чи виданим сервером, метод *validate* не завершиться успішно.

RefreshTokenStrategy також виконує аналогічні дії в методі *validate AccessTokenStrategy*, але вже для *refreshToken* і також виконує додаткову перевірку, чи існує токен в в базі даних, викликаючи відповідний метод сервісу *RefreshTokensService* [27].

Приклад коду створення стратегії авторизації *AccessTokenStrategy*:

```
import { Injectable } from '@nestjs/common';
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { AppConfigService } from 'src/config/app-config.service';
import { AccessJwtPayload } from '../types';
import { UsersService } from 'src/modules/users/users.service';

@Injectable()
export class AccessTokenStrategy extends PassportStrategy(Strategy, 'jwt') {
  constructor(
    private readonly appConfigService: AppConfigService,
    private readonly usersService: UsersService,
  ) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      secretOrKey: appConfigService.get<string>('JWT_ACCESS_SECRET'),
    });
  }

  async validate({ id, role, name }: AccessJwtPayload) {
    const userData = await this.usersService.findOne(
      { id, role, name },
      {
        select: { id: true, role: true, name: true },
        loadEagerRelations: false,
      },
    );

    return userData;
  }
}
```

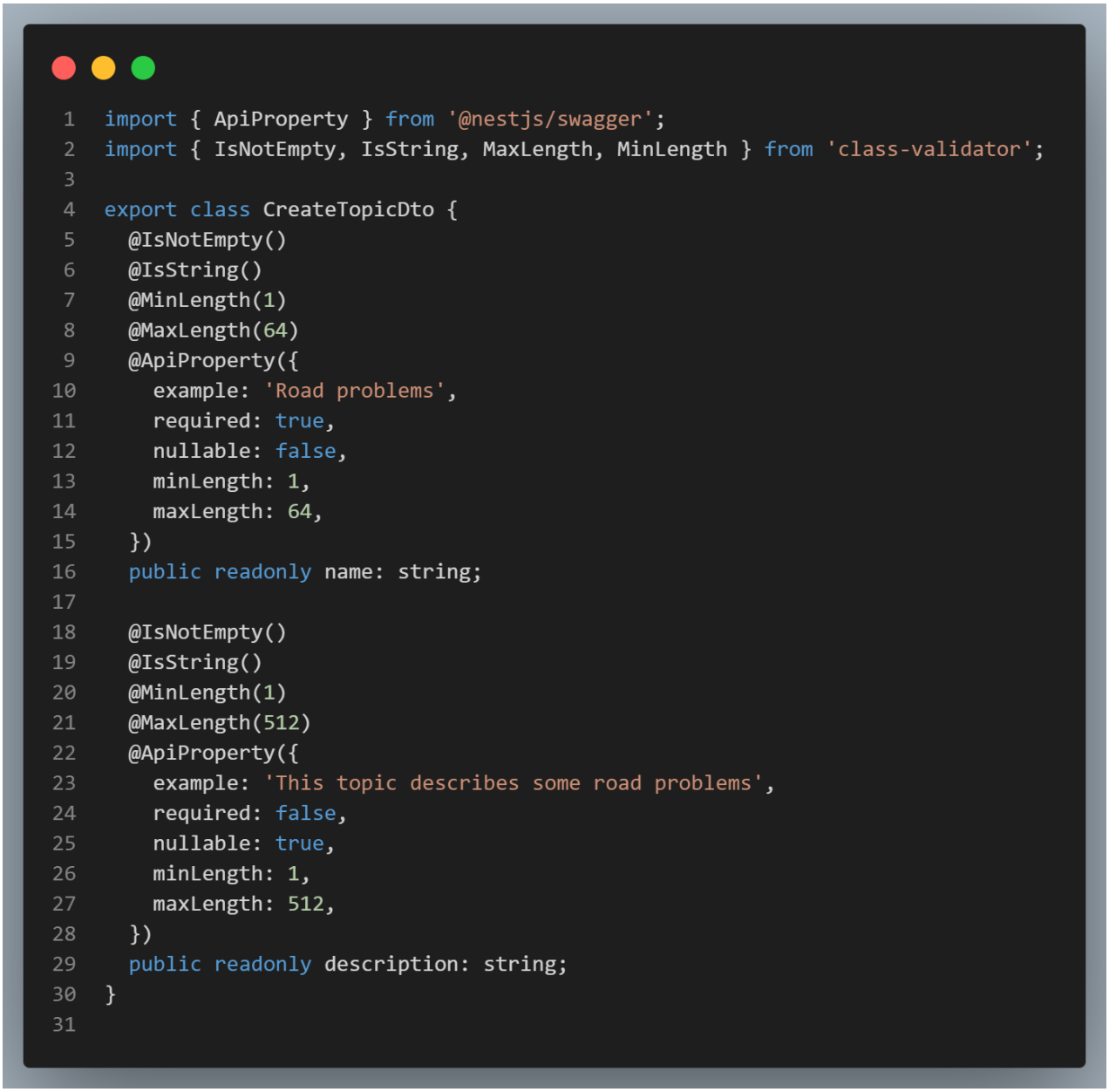
Рисунок 3.5 – Код стратегії авторизації «AccessTokenStrategy»

Створені стратегії підключаються до модуля авторизації та реєструються з використанням *JwtModule* бібліотеки *@nestjs/jwt*, а відповідно вже можуть використовуватися для створення захисників маршрутів. Кожній стратегії авторизації буде відповідати один обробник захисту маршрутів, який реалізовується наслідуванням від базового *AuthGuard* з бібліотеки *@nestjs/passport*, при цьому вказуючи назву створеної стратегії. Зазвичай для більшості маршрутів буде використовуватися створений *@AccessTokenGuard*, а *@RefreshTokenGuard* буде використовуватися лише для маршрутів оновлення токенів та виконання логауту з системи.

3.1.8 Деталі реалізації контроллерів

Після створення сервісів, необхідно створити файли контроллерів та визначити маршрути, за якими буде звертатися клієнтська частина застосунку до сервера. Процес створення контроллера в Nest.js характеризується створенням класу, який буде позначатися декоратором *@Controller()* та передавати маршрут, який буде оброблятися контроллером всередину цього декоратора. В класі контроллера створюються методи обробки маршрутів, самі ж маршрути визначаються всередині декораторів з типом запиту, для прикладу *@Post()* без переданого маршруту визначає, що метод контроллера буде оброблювати POST запит за машрутом контроллера.

Також іншим важливим моментом є використання DTO для визначення типу об’єкта та валідації його полей, що відправляються з клієнтської частини застосунку у запиті. Для встановлення правил валідації класів DTO буде використовуватися бібліотека *class-validator*, яка дозволяє додавати декоратори до полей класу, що будуть перевіряти відповідність певним умовам. Для прикладу, декоратор *@IsString()* очікує, що властивість об’єкта буде рядкового типу, інакше Nest.js одразу завершить виконання запиту помилкою на етапі валідації. Приклад структури DTO для створення теми *CreateTopicDto*:



```
import { ApiProperty } from '@nestjs/swagger';
import { IsNotEmpty, IsString, MaxLength, MinLength } from 'class-validator';

export class CreateTopicDto {
  @IsNotEmpty()
  @IsString()
  @MinLength(1)
  @MaxLength(64)
  @ApiProperty({
    example: 'Road problems',
    required: true,
    nullable: false,
    minLength: 1,
    maxLength: 64,
  })
  public readonly name: string;

  @IsNotEmpty()
  @IsString()
  @MinLength(1)
  @MaxLength(512)
  @ApiProperty({
    example: 'This topic describes some road problems',
    required: false,
    nullable: true,
    minLength: 1,
    maxLength: 512,
  })
  public readonly description: string;
}

```

Рисунок 3.6 – Код класу «CreateTopicDto» для визначення моделі даних створення теми

Користувачі мають отримувати доступ до певних ресурсів на сервері лише за наявності ролі адміністратора, а тому необхідно визначити механізм, що буде захищати певні методи контроллерів, до яких звичайний користувач програми не повинен мати доступу. Наприклад, лише адміністратори повинні мати змогу видаляти коментарі та ініціативи, користувачів, змінювати статус ініціатив та інші сутності системи. Для встановлення переліку ролей,

необхідних для доступу до певного методу контроллера, було створено декоратор *@HasRoles*, що приймає масив ролей користувачів, які можуть отримати доступ до ресурсу та встановлює це значення в метадані методу. Для перевірки відповідності ролі поточного користувача ролям в метаданих контроллера створено захисник маршрутів *RolesGuard*, що використовується у вигляді декоратора при створенні класу контроллера [28].

Сам код методів контроллера не містить складної логіки, адже його основною задачею є виклик методів сервісів з переданими даними, які були отриманих в запиті.

```typescript
@ApiTags('topics')
@Controller('topics')
@ApiBearerAuth()
@UseGuards(AccessTokenGuard, RolesGuard)
@UseInterceptors(ClassSerializerInterceptor)
export class TopicsController {
  constructor(private readonly topicsService: TopicsService) {}

  @Get()
  findAll(): Promise<TopicEntity[]> {
    return this.topicsService.findAll();
  }

  @Get(':id')
  findOne(@Param() conditions: IdParameterDto): Promise<TopicEntity> {
    return this.topicsService.findOne(conditions);
  }

  @HasRoles(UserRoleEnum.ADMIN)
  @Post()
  createOne(@Body() createEntityDto: CreateTopicDto): Promise<TopicEntity> {
    return this.topicsService.createOne(createEntityDto);
  }

  @HasRoles(UserRoleEnum.ADMIN)
  @Patch(':id')
  updateOne(
    @Param() conditions: IdParameterDto,
    @Body() updateEntityDto: UpdateTopicDto,
  ): Promise<TopicEntity> {
    return this.topicsService.updateOne(conditions, updateEntityDto);
  }

  @HasRoles(UserRoleEnum.ADMIN)
  @Delete(':id')
  deleteOne(@Param() conditions: IdParameterDto): Promise<TopicEntity> {
    return this.topicsService.deleteOne(conditions);
  }
}
```

Рисунок 3.7 – Код контроллера «TopicsController»

3.1.9 Маршрутизація

Перелік основних маршрутів сервера, які використовуються в клієнтській частині веб-застосунку:

- *POST "/auth/signup"* – реєстрація нового користувача
- *GET "/auth/profile"* – отримати профіль користувача
- *POST "/auth/signout"* – вийти з облікового запису
- *POST "/auth/signin"* – ввійти в обліковий запис
- *GET "/initiatives/find-by-location/:id"* – знайти ініціативу за її локацією
- *GET "/initiatives"* – отримати перелік всіх ініціатив
- *GET "/initiatives/user/:id"* – отримати ініціативи, створені користувачем
- *GET "/initiatives/find-by-search?..."* – знайти ініціативи за обраними критеріями пошуку
- *GET "/initiatives/most-liked"* – отримати перелік найпопулярніших ініціатив
- *GET "/topics"* – отримати перелік тем ініціатив
- *POST "/images"* – завантажити зображення на сервер
- *POST "/initiatives"* – створення ініціативи
- *GET "/images/initiative/:id"* – отримати зображення ініціативи
- *GET "/likes/initiatives/user-like?userId=...&initiativeId=..."* – перевірка, чи вподобав користувач ініціативу
- *GET "/likes/initiatives/count-likes/:id"* – отримати кількість вподобань ініціативи
- *DELETE "/likes/initiatives/:id"* – забрати власний лайк з ініціативи
- *POST "/likes/initiatives"* – додати власний лайк на ініціативу
- *DELETE "/initiatives/:id"* – видалити ініціативу

- *GET "/regions"* – отримати список областей України
- *GET "/initiatives/month-statistics"* – отримати статистику за місяць
- *GET "/comments/initiative/:id"* – отримати всі коментарі ініціативи
- *GET "/likes/comments/count-likes/:id"* – отримати кількість вподобань коментаря
- *GET "/likes/comments/user-like?userId=...&commentId=..."* – перевірка, чи вподобав користувач певний коментар
- *DELETE "/likes/comments/:id"* – забрати власний лайк з коментаря
- *POST "/likes/comments"* – вподобати коментар
- *DELETE "/comments/:id"* – видалити коментар до ініціативи
- *POST "/comments"* – додати коментар чи відповідь до ініціативи

Варто зазначити, що повна адреса ендпоінтів сервера складається з ім'я хоста сервера та префікса *"/api/v1"*, вище наведено відносні маршрути серверної частини застосунку.

3.1.10 Міграції бази даних

В серверній частині застосунку наявні сутності бази даних, які вимагають визначення початкового набору даних перед використанням додатку. Такими сутностями є теми ініціатив *topics*, області України *regions* та адміністратори системи в таблиці користувачів *users*. Одним з варіантів додати необхідні сутності є виклик методів контроллера для кожної окремої сутності. Такий піхід не є зручним, адже вимагає великої кількості часу та уважності при введені кожного елементу набору даних.

З метою оптимізації часу на додавання об'єктів сутностей в базу даних, було прийнято рішення використати створення міграцій. Міграції створюються з використанням спеціального інтерфейсу від TypeORM. Потрібно визначити назву міграції та два методи, один з яких буде запускати міграцію, а інший –

відміняти зроблені дії. Самі файли міграцій знаходяться в директорії з модулем бази даних та будуть знайдені TypeORM за шаблоном назви файлів, вказаному в конфігурації створення бази даних [29].

Для прикладу, міграції на вставку значень областей України в таблицю *regions* будуть виглядати наступним чином:

```
import { MigrationInterface, QueryRunner } from 'typeorm';

export class InsertRegionsData1627402586275 implements MigrationInterface {
  public async up(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.query(`
            INSERT INTO regions (name, geocode) VALUES
            ('Вінницька область', 'UA-05'),
            ('Волинська область', 'UA-07'),
            ('Дніпропетровська область', 'UA-12'),
            ('Донецька область', 'UA-14'),
            ('Житомирська область', 'UA-18'),
            ('Закарпатська область', 'UA-21'),
            ('Запорізька область', 'UA-23'),
            ('Івано-Франківська область', 'UA-26'),
            ('Київська область', 'UA-32'),
            ('Кіровоградська область', 'UA-35'),
            ('Луганська область', 'UA-09'),
            ('Львівська область', 'UA-46'),
            ('Миколаївська область', 'UA-48'),
            ('Одеська область', 'UA-51'),
            ('Полтавська область', 'UA-53'),
            ('Рівненська область', 'UA-56'),
            ('Сумська область', 'UA-59'),
            ('Тернопільська область', 'UA-61'),
            ('Харківська область', 'UA-63'),
            ('Херсонська область', 'UA-65'),
            ('Хмельницька область', 'UA-68'),
            ('Черкаська область', 'UA-71'),
            ('Чернівецька область', 'UA-77'),
            ('Чернігівська область', 'UA-74'),
            ('АР Крим', 'UA-43'),
            ('Київ', 'UA-30'),
            ('Севастополь', 'UA-40');
        `);
  }

  public async down(queryRunner: QueryRunner): Promise<void> {
    await queryRunner.query(`DELETE FROM regions`);
  }
}

```

Рисунок 3.8 – Код міграцій бази даних на вставку даних до таблиці «regions»

3.2 Опис реалізації клієнтської частини веб-застосунку

3.2.1 Загальна структура клієнтської частини

Клієнтська частина веб-застосунку буде реалізовуватися з використанням бібліотеки React, особливістю якої є використання компонентів. Компонентна структура сприяє повторному використанню коду, дозволяє легко підтримувати модульність, оскільки кожен з компонентів має свої стилі та реалізовує власну логіку. Також використання компонентів полегшує керування станом певного динамічного інтерфейсу користувача, так як відбувається оновлення лише тих частин інтерфейсу, дані яких зазнали змін [30].

Додаток складатиметься з трьох наступних сторінок:

- Сторінка логіну, на яку користувач автоматично перенаправляється при першому зверненні до веб-застосунку. При успішній авторизації користувач перенаправляється на головну сторінку з функціоналом додатку.
- Сторінка реєстрації, на яку потрапляє користувач зі сторінки логіну, якщо він ще не має створеного аккаунту. Після реєстрації користувач перенаправляється на головну сторінку додатку.
- Сторінка додатку, яка буде реалізована в SPA стилі. Це означає, що при взаємодії користувача з додатком, на певні дії користувача сторінка не буде перезавантажуватися або перенапраляти на іншу сторінку, натомість буде лише довантажувати необхідні дані, зберігаючи попередньо завантажену інформації, розмітку та файли [31]. Це забезпечить інтерактивність інтерфейсу та підвищить продуктивність застосунку, оскільки при кожній дії користувача не буде необхідності завантажувати ті ж самі дані для роботи додатку.

3.2.2 Статичні файли

Для реалізації основної функціональності клієнтської частини веб-застосунку потрібно додати набір необхідних файлів, які будуть використовуватися в проекті:

- Зображення логотипу додатку, яке буде розміщене разом з назвою додатку в заголовку вкладки браузера.
- Зображення мітки на карті, яке буде доступне в 4-х кольорових варіаціях, кожна з яких відповідатиме за конкретний статус ініціативи.
- Зображення карти застосунку з мітками, яке використовуватиметься як фон з доданим ефектом розмиття для сторінок логіну та реєстрації, щоб створити відчуття інтерактивності та створити загальне враження про додаток ще на етапі авторизації користувача.
- JSON файл, що складається з географічних координат кордонів України.

Щодо ресурсів, які будуть використані для пошуку відповідних файлів, варто зазначити, що зображення будуть використовуватися такі, що знайдені в мережі Інтернет та не містять обмежень на своє використання, щоб уникнути можливих проблем з порушенням авторських прав. Також JSON-файл буде завантажено з ресурсу *data.humdata.org* – платформи, яка надає доступ до відкритих даних, що стосуються демографії, економіки, інфраструктури країн світу тощо.

3.2.3 Реалізація класу HttpClient

Для взаємодії клієнтської частини з серверною частиною застосунку, необхідно реалізувати механізм, який дозволятиме зручно формувати запити та отримувати відповідь. Для виконання HTTP-запитів у JavaScript наявна вбудована функція *fetch,* яка приймає посилання на ресурс та опції запиту, що буде формуватися до сервера. Важливим моментом є той факт, що всі запити, які відправляються на сервер, окрім запитів що відповідають за виконання входу чи реєстрацію користувача в систему, мають бути авторизовані. При успішній авторизації, клієнтська частина буде записувати дані про профіль користувача та пару токенів авторизації у *localStorage* браузера. Відповідно, при кожній відправці запиту до сервера, необхідно буде виконати дії, пов’язані з отриманням токенів авторизації з *localStorage* браузера та передачі відповідного токену в заголовок авторизації запиту. Також у випадку, якщо термін життя *accessToken* закінчується, необхідно щоразу виконувати оновлення пари токенів з використанням *refreshToken* та повторити запит, використовуючи пару нових токенів.

При прямому використанні такого підходу, описаного вище, відбувається дублювання коду, а оскільки клієнт з сервером будуть досить часто взаємодіяти, це може ускладнити читання та відлагодження коду. З метою спрощення процедури створення авторизованого запиту, було прийнято рішення створити окремий клас *HttpClient*, який буде використовуватися для формування всіх запитів до серверу.

HttpClient реалізовуватиме наступні методи:

- *authorizedFetch()* – виконує авторизований запит за переданим посиланням та опціями запиту, попередньо отримавши необхідний токен доступу з *localStorage браузера.* Якщо при виконанні запиту повертається відповідь *401* з помилкою авторизації, то

викликається метод цього ж класу *refreshToken()* для оновлення пари токенів. Після оновлення запит виконується ще раз, після цього повертаючи результат.

- *unauthorizedFetch()* – дублює виконання логіки функції *fetch().* Використовується для формування запитів на виконання логіну або реєстрації, які не вимагають авторизованого доступу.
- *refreshTokens()* – виконує запит на оновлення токенів та записує нову пару токенів в *localStorage* браузера.

Також варто зазначити, що всі запити включають в себе перетворення відповіді в JSON формат, а відповідно більше немає необхідності викликати такий метод після отримання відповіді сервера в компонентах React, що також зменшує дублювання коду.

```
class HttpClient {
  static async authorizedFetch(url, options = {}) {
    try {
      options.headers = options.headers || {};
      options.headers.Authorization = `Bearer ${localStorage.getItem(
        "accessToken"
      )}`;

      let response = await fetch(url, options);
      if (response.status === 401) {
        const tokenRefreshed = await HttpClient._refreshToken();

        if (tokenRefreshed) {
          options.headers.Authorization = `Bearer ${localStorage.getItem(
            "accessToken"
          )}`;

          response = await fetch(url, options);
        } else {
          throw new Error("Failed to refresh tokens");
        }
      }

      return response.json();
    } catch (error) {
      throw error;
    }
  }

  static async _refreshToken() {
    try {
      const refreshToken = localStorage.getItem("refreshToken");
      if (!refreshToken) {
        return false;
      }

      const response = await fetch(
        "http://localhost:8080/api/v1/auth/tokens/refresh",
        {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
            Authorization: `Bearer ${refreshToken}`,
          },
        }
      );

      if (!response.ok) {
        return false;
      }

      const tokens = await response.json();
      localStorage.setItem("accessToken", tokens.accessToken);
      localStorage.setItem("refreshToken", tokens.refreshToken);

      return true;
    } catch (error) {
      throw error;
    }
  }

  static async unauthorizedFetch(url, options = {}) {...}
}

export default HttpClient;
```

Рисунок 3.9 – Код основних методів класу HttpClient

3.2.4 Сторінки авторизації

Першою сторінкою, на яку потрапляє користувач в додатку, є сторінкою логіну. Якщо користувач ще не має облікового запису в системі, йому пропонується перейти за посиланням реєстрації.

Компонент реєстрації *RegistrationComponent* містить форму з полями, які користувач має заповнити для створення облікового запису. Валідація форми відбувається з використанням бібліотек *formic* та *Yup*. Бібліотека *formic* використовується для керування станом форми [32], а *Yup* – для визначення правил валідації до кожного поля форми та валідації самої схеми даних. Якщо присутня помилка валідації, вона відображаються під відповідним полем введення з текстом самої помилки [33].

Після заповнення форми валідними даними відправляється запит на сервер для реєстрації нового користувача. У разі успішної відповіді сервера, компонент обробляє відповідь, записуючи пару токенів авторизації в *localStorage*. Після цього відправляється запит на отримання профілю авторизованого користувача, а отримані дані записуються об’єкт user в *localStorage*. Після всіх вищеописаних дій, авторизований користувач автоматично перенаправляється на головну сторінку додатку.

Якщо користувач уже має створений обліковий запис, він може скористатися сторінкою для логіну. Компонент для логіну дуже схожий за своєю реалізацією на компонент реєстрації, але користувач не має вводити поля з ім’ям та прізвищем, а сама форма обробляється запитом на логін користувача. Інший алгоритм обробки форми залишається без змін.

Також важливим моментом є використання компонентів бібліотеки *@mui/material*, що значно пришвидшує розробку та поліпшує вигляд користувацького інтерфейсу.

Приклад валідації та обробника відправки форми компонента Логіну:

```
const RegistrationComponent = () => {
  const navigate = useNavigate();

  const validationSchema = Yup.object({
    name: Yup.string()
      .min(1, "Мінімальна довжина 1 символ")
      .max(32, "Максимальна довжина 32 символи")
      .required("Ім'я є обов'язковим"),
    surname: Yup.string()
      .min(1, "Мінімальна довжина 1 символ")
      .max(32, "Максимальна довжина 32 символи")
      .required("Прізвище є обов'язковим"),
    email: Yup.string()
      .email("Невірний формат електронної пошти")
      .min(8, "Мінімальна довжина 8 символів")
      .max(256, "Максимальна довжина 256 символів")
      .required("Електронна адреса є обов'язковою"),
    password: Yup.string()
      .min(8, "Мінімальна довжина 8 символів")
      .max(64, "Максимальна довжина 64 символи")
      .required("Пароль є обов'язковим"),
  });

  const formik = useFormik({
    initialValues: {
      name: "",
      surname: "",
      email: "",
      password: "",
    },
    validationSchema: validationSchema,
    onSubmit: async (values, { setSubmitting, setErrors }) => {
      try {
        const data = await HttpClient.unauthorizedFetch(
          "http://localhost:8080/api/v1/auth/signup",
          {
            method: "POST",
            headers: {
              "Content-Type": "application/json",
            },
            body: JSON.stringify(values),
          }
        );

        localStorage.setItem("accessToken", data.accessToken);
        localStorage.setItem("refreshToken", data.refreshToken);
        await fetchUserProfile(data.accessToken);
        navigate("/app");
      } catch (error) {
        setErrors({ submit: "При реєстрації виникла помилка" });
      } finally {
        setSubmitting(false);
      }
    },
  });

  const fetchUserProfile (){...};

  return (...);
};

export default RegistrationComponent;
```

Рисунок 3.10 – Схема валідації та форма реєстрації в «RegistrationComponent»

3.2.5 Головна сторінка додатку

Після реєстрації або логіну користувач потрапляє на сторінку додатку. Ця сторінка реалізовує всю функціональність веб-застосунку.

Сам компонент *AppComponent* складається з менших компонентів, кожен з яких виконує свою конкретну функцію. Розділення головної сторінки на менші компоненти підтримує принцип модульності, спрощує розробку, підтримуваність, читабельність та тестування коду. Компонент *AppComponent* реалізовує взаємодію складових компонентів сторінки додатку, а саме:

- *MapComponent*
- *SearchPlaceComponent*
- *CreateInitiativeComponent*
- *ViewInitiativeComponent*
- *MenuComponent*
- *SearchInitiativeComponent*
- *CommentsComponent*
- *AdminNotesComponen*
- *StatisticsComponent*
- *LogoutComponent*

Розглянемо детальніше кожен з компонентів, що входить до складу *AppComponent*.

MapComponent – найскладніший компонент додатку, основною функцією якого є створення інтерактивної карти та взаємодія з користувачем у реальному часі. Займає всю площу сторінки. Спершу компонент ініціалізує шар мапи Leaflet, додає візуальне відображення координат України, завантажує інформацію про місця міток всіх наявних ініціатив та відображає їх на карті іконками маркера з відповідними статусами. Також компонент може змінювати набір ініціатив для відображення на карті в залежності від обраної опції в меню,

оброблювати події додавання нової мітки та перегляду існуючої відповідними викликами та передачею даних до інших компонентів.

Основні методи компонента:

- *initializeMap* – створення всіх шарів карти та відображення всіх ініціатив
- *addTileLayer* – додавання зображення карти
- *addGeoJSONLayer* – додавання кордонів України до карти
- *addMapClickHandler* – обробник подій кліку на карту, який передає координати мітки в компонент *CreateInitiativeComponent*
- *getMarkerLocationInfo* – виконує реверсне геокодування мітки за її координатами, отримуючи код країни та області, використовуючи звернення до Nominatim API
- *addMarkerToMap* – створює новий маркер та додає його на карту
- *selectMarker* – метод виділення існуючого маркера
- *removeMarker* – функція видалення маркера
- *clearMarkers* – очищення всіх маркерів на карті
- *handleMenuOptionsChange* – метод, який викликає відповідні обробники в залежності від зміни обраної опції в компоненті меню
- *showAllInitiatives* – отримує масив всіх ініціатив та відображає на карті
- *showUserInitiatives* – отримує масив ініціатив, створених користувачем, та відображає на карті
- *showCustomSearchInitiatives* – отримує масив ініціатив, сформований за критеріями пошуку, та відображає його на карті
- *showMostPopularInitiatives* – отримує масив найпопулярніших ініціатив та відображає їх на карті
- *handleCreateInitiativeClose* – обробник події закриття компоненту створення ініціативи

- *handleSearchPlaceView* – відображає знайдене користувачем місце з використанням компоненту SearchPlaceComponent.

SearchPlaceComponent – компонент для пошуку міста чи конкретної адреси. Розміщується по центру у верхній частині компонента *MapComponent*. Користувач виконує введення інформації для пошуку, обирає один варіант зі списку знайдених варіантів, компонент передає інформацію про межі координат області на карті в *AppComponent*, який в свою чергу передасть її в *MapComponent* для відображення знайденого місця для користувача. Методи компонента:

- *handleSearch* – виконує запит на пошук місць та їхніх координат з використанням Nominatim API
- *handleResultClick* – відправляє результати пошуку в компонент AppComponent
- *clearSearchResult* – очищує масив результатів пошуку

CreateInitiativeComponent – викликається для створення ініціативи при кліку на вільне місце мапи. Відображається на передньомц плані компонента *MapComponent* з правої частини сторінки. Компонент складається з форми, яку користувач має заповнити для створення ініціативи. Також присутнє поле для вибору і завантаження зображення. При успішній валідації та створенні ініціативи, id її локації передається в компонент *AppComponent*, а згодом викликається компонент для перегляду створеної ініціативи *ViewComponent*. Форма створюється з використанням бібліотеки formik, валідується з використанням бібліотеки *Yup*. Також реалізовано наступні методи компонента:

- *fetchTopics* – завантажує перелік тем для вибору теми ініціативи
- *handleImageChange* – завантажує зображення на сервер та записує айді створеної сутності для передачі значення при створенні ініціативи

ViewInitiativeComponent – компонент для перегляду ініціативи, що відображає ініціативу за переданим значенням її локації з компонента *AppComponent*. Компонент розташовується поверх компонента мапи з правої частини сторінки. Реалізовані наступні методи:

- *fetchInitiativeData* – отримує інформацію з сервера про ініціативу, кількість вподобань, зображення.
- *deleteUserLike* – видаляє лайк користувача, якщо він присутній
- *addUserLike* – додає лайк користувача на ініціативу
- *handleLikeButtonClick* – обробник події кліку на кнопку лайку
- *handleDeleteInitiative* – обробник події кліку на кнопку видалення ініціативи. Може викликатися лише у випадку, коли користувач має роль "*Admin*"
- *toggleUserComments* – метод обробки кліку на кнопку коментарів. Викликає або приховує коментарі користувача.
- *toggleAdminComments* – метод обробки кліку на кнопку відповідей адміністратора, що викликає або приховує коментарі адміністраторів.
- *getStatusColor* – функція, що повертає колір тексту в залежності від статусу ініціативи

MenuComponent відображає меню з вибором кількох доступних опцій. В додатку присутні опції для перегляду всіх ініціатив за замовчуванням, перегляду тільки ініціатив створених користувачем, найпопулярніших ініціатив, пошук ініціативи за певними критеріями, перегляд розділу статистики. Сам компонент розміщується зліва у верхній частині екрану у вигляді кнопки, а при кліку на неї відображає список опцій меню. Головним методом є обробник *handleOptionSelect*, який передає обрану опцію користувачем в змінну стану компоненту *AppComponent*, на основі якої будуть викликані відповідні обробники в залежності від обраної опції меню.

SearchInitiativeComponent є компонентом, який відображається зліва знизу сторінки після вибору опції пошуку ініціативи в меню. Сам компонент складається з форми, після заповненя якої надсилається запит на пошук ініціатив за переданими критеріями. Результат пошуку передається в компонент *AppComponent* та використовується в компоненті *MapComponent* для відображення результатів пошуку. Форма створюється з використанням бібліотеки *formik*, валідується з використанням бібліотеки *Yup*. Також реалізовано наступні методи компонента:

- *fetchTopics* – завантажує список існуючих тем
- *fetchRegions* – завантажує список областей України

CommentsComponent – компонент, який відображається в компоненті ViewInitiativeComponent та відповідає за розділ коментарів користувачів. Компонент складається з поля для введення коментаря користувача, вибору опції сортування коментарів та відображення самих коментарів. Компонент складається з наступних методів:

- *fetchComments* – завантажує список коментарів ініціативи та пов'язану з кожним коментарем кількість лайків
- *handleLikeButtonClick* – в залежності від того, чи вподобав користувач конкретний коментар, додає лайк або ж видаляє його
- *handleDeleteComment* – метод видаляє коментар, доступний для виклику лише для користувача з роллю адміністратора
- *handleSortChange* – функція обробки способу сортування коментарів при вибору відповідної опції
- *sortComments* – функція сортування коментарів

AdminNotesComponent – ще один компонент, який відображається в компоненті для перегляду ініціатив та виконує функцію відображення відповідей адміністраторів щодо етапу виконання ініціативи. Також наявна секція для зміни статусу ініціативи. Відмінність від компоненту коментарів

полягає в тому, що додавати коментарі та змінювати статус ініціативи в цьому розділі можуть лише адміністратори додатку, а до коментарів можуть додаватися зображення. Також відсутня функція вподобання відповідей та вибір сортування, оскільки відповіді автоматично сортуються за датою створення. Окрім функцій форми та схеми валідації, компонент містить наступні методи:

- *fetchComments* – завантажує всі коментарі певної ініціативи
- *handleImageChange* – завантажує зображення на сервер
- *sortComments* – сортує відповіді адміністраторів за датою створення

StatisticsComponent – компонент для відображення статистики за місяць. Викликається при виборі відповідної опції меню та розміщується знизу у лівій частині сторінки. Також наявна побудова кругової діаграми для кращої візуалізації співвідношення найпопулярніших тем, використовуючи бібліотеку *react-vis*. В компоненті наявний метод *fetchStatistics* для отримання статистичних даних з сервера, а також код для побудови самої діаграми та обчислення її складових значень.

LogoutComponent – відображає інформацію про користувача. Містить кнопку, при натисканні якої відбувається логаут користувача з системи та перенаправлення на сторінку логіну.

3.2.6 Взаємодія компонентів головної сторінки

Взаємодія всіх описаних вище компонентів відбувається в *AppComponent*. Він використовує змінні стану, функції та хендлери для управління відображення компонентів та їхньою взаємодією [34].

Змінні станів

- *showCreateInitiative* - відповідає за відображення компонента *CreateInitiativeComponent*.

- *showViewInitiative* - відповідає за відображення компонента *ViewInitiativeComponent*.
- *markerLocationInfo* - зберігає інформацію про місце розташування маркера, що буде використовуватися для створення ініціативи в *CreateInitiativeComponent*.
- *locationId* - зберігає id локації, що буде використовуватися для перегляду ініціативи в *ViewInitiativeComponent*.
- *menuOptions* - зберігає обраний пункт меню, який впливає на відображення відповідного компонента
- *searchParams* - зберігає параметри пошуку для компонента *SearchInitiativeComponent*.
- *markerToAdd* - зберігає інформацію про маркер, який потрібно додати.
- *viewedInitiative* - зберігає інформацію про ініціативу, яку було переглянуто.

Функції та хендлери:

- *handleMarkerAdd* – записує інформацію про місце кліку на карті та відображає CreateInitiativeComponent.
- *handleMarkerClick* – записує інформацію про виділений маркер на карті та відображає ViewInitiativeComponent.
- *handleCreateInitiativeClose* – обробляє закриття CreateInitiativeComponent і відображає ViewInitiativeComponent.
- handleViewInitiativeClose – обробляє закриття ViewInitiativeComponent.
- *handleMenuOptionSelect* – записує обрану опцію меню в змінну стану.
- *handleSearch* – записує введені користувачем критерії пошуку ініціативи.

- *handleMenuClose* – приховує компонент меню та встановлює обрану опцію меню для відображення всіх ініціатив за замовчуванням.

Для прикладу буде розглянуто сценарій взаємодії компоненту *MapComponent* та *ViewInitiativeComponent*.

Користувач натискає на маркер на карті.

1. В компоненті *MapComponent* користувач клікає на існуючий маркер на мапі. Відповідний хендлер компонента викликає колбек-функцію *onMarkerClick* та передає *id* обраної локації ініціативи.
2. В компоненті *AppComponent* викликається обробник функції *onMarkerClick – handleMarkerClick*, який записує *id* обраної локації ініціативи для перегляну в змінну *locationId*. Після цього значення *showViewInitiative* встановлюється в true і відображається компонент для перегляду ініціативи *ViewInitiativeComponent*.
3. Компонент *ViewInitiativeComponent* приймає змінну *locationId* та визначає обробник події *onClose* як хендлер функції в компоненті *AppComponent handleViewInitiativeClose*. Коли відбувається закриття компоненту для перегляду ініціатив, спрацьовує хендлер *handleViewInitiativeClose* компоненту *AppComponent*, який встановлює значення *showViewInitiative* в *false*, в результаті чого компонент для перегляду ініціативи закриється і не буде відображатися.

Таким чином, компонент *AppComponent* дозволяє організувати взаємодію між різними частинами програми, реалізовуючи зручний користувацький інтерфейс.

Нижче наведено код створення змінних станів, методи обробки подій та налагоджено взаємодію компонентів у *AppComponent*:

```jsx
const AppComponent = () => {
  const [showCreateInitiative, setShowCreateInitiative] = useState(false);
  const [showViewInitiative, setShowViewInitiative] = useState(false);
  const [markerLocationInfo, setMarkerLocationInfo] = useState(null);
  const [locationId, setLocationId] = useState(null);
  const [menuOptions, setMenuOptions] = useState(null);
  const [searchParams, setSearchParams] = useState({});
  const [markerToAdd, setMarkerToAdd] = useState({ locationId: undefined });
  const [viewedInitiative, setViewedInitiative] = useState({
    locationId: undefined,
  });

  const handleMarkerAdd = (markerLocation) => {
    setMarkerLocationInfo(markerLocation);
    setShowViewInitiative(false);
    setShowCreateInitiative(true);
  };

  const handleMarkerClick = (markerLocationId) => {
    setLocationId(markerLocationId);
    setShowCreateInitiative(false);
    setShowViewInitiative(true);
  };

  const handleCreateInitiativeClose = (markerLocationId) => {
    setShowCreateInitiative(false);
    setLocationId(markerLocationId);
    setShowViewInitiative(true);
    setMarkerToAdd({ locationId: markerLocationId });
  };

  const handleViewInitiativeClose = () => {
    setViewedInitiative({ locationId });
    setShowViewInitiative(false);
  };

  const handleMenuOptionSelect = (option) => {
    setMenuOptions(option);
    if (option !== "search") {
      setSearchParams({});
    }
  };

  const handleSearch = (params) => {
    setSearchParams(params);
  };

  const handleMenuClose = () => {
    setSearchParams({});
    setMenuOptions("all");
  };

  return (...);
};

export default AppComponent;
```

Рисунок 3.11 – Код змінних станів та хендлерів «AppComponent»

```jsx
const AppComponent = () => {
  ...

  return (
    <div
      style={{
        display: "flex",
        height: "100vh",
        overflow: "hidden",
        margin: "0",
      }}
    >
      <MapComponent
        onMarkerAdd={handleMarkerAdd}
        onMarkerClick={handleMarkerClick}
        menuOptions={menuOptions}
        searchParams={searchParams}
        markerToAdd={markerToAdd}
        viewedInitiative={viewedInitiative}
      />
      {showCreateInitiative && (
        <CreateInitiativeComponent
          markerLocationInfo={markerLocationInfo}
          onClose={handleCreateInitiativeClose}
        />
      )}
      {showViewInitiative && locationId && (
        <ViewInitiativeComponent
          locationId={locationId}
          onClose={handleViewInitiativeClose}
        />
      )}
      <LogoutComponent />
      <MenuComponent handleMenuOptionSelect={handleMenuOptionSelect} />
      {menuOptions === "search" && (
        <SearchInitiativeComponent
          onSearch={handleSearch}
          onClose={handleMenuClose}
        />
      )}
      {menuOptions === "statistics" && (
        <StatisticsComponent onClose={handleMenuClose} />
      )}
    </div>
  );
};

export default AppComponent;
```

Рисунок 3.12 – Код взаємодії компонентів у «AppComponent»

3.2.7 Маршрутизація

Маршрутизація додатку має досить просту логіку, оскільки загальна структура додатку складається зі сторінок авторизації та сторінки додатку, що реалізована у стилі односторінкового застосунку [35].

- *“/login”* – обробляється викликом компонента LoginComponent.
- *“/register”* – обробляється викликом компонента *RegistrationComponent*.
- *“/app”* – обробляється викликом компонента *AppComponent*.
- *“/”* – якщо користувач вперше переходить за посиланням додатку, він автоматично перенаправляється на сторінку *“/login”*

ВИСНОВОК ДО РОЗДІЛУ 3

В даному розділі було детально описано реалізаціцію серверної та клієнтської частин веб-застосунку для координації та агрегації громадських ініціатив.

Для серверної частини розглянуто процес підключення реляційної бази даних PostgreSQL до додатку, підключення до хмарного сховища Azure Blob Storage та реалізацію сервісу для взаємодії з ним. Визначено та реалізовано основні програмні модулі, виконано проектування та створення таблиць з використанням сутностей TypeORM. Визначено та реалізовано методи сервісів, що взаємодіять з базою даних та надають необхідну для роботи додатку функціональність. Також було виокремлено маршрути, за якими буде звертатися клієнтська частина застосунку для взаємодії з ресурсами сервера, реалізовано функції контроллерів, що виконуватимуть функцію обробників визначених маршрутів. На завершення, додано файли міграцій бази даних, щоб зменшити час, необхідний для налаштування необхідних сутностей для першого використання додатку.

Окрім основного функціоналу серверної частини застосунку, також описано й реалізовано систему авторизації з використанням JWT-токенів, забезпечено одночасну кількість 2-х сесій для одного користувача. З метою забезпечення безпеки застосунку, додано обробники для захисту маршрутів, що перевіряють наявність авторизації та відповідність ролей користувача вказаним ролям для доступу до ресурсу.

Щодо клієнтської частини веб-застосунку, було визначено та знайдено необхідні для роботи додатку файли зображень та координат України, що будуть розміщуватися в статичній директорії веб-застосунку. Для взаємодії з серверною частиною застосунку розроблено клас *HttpClient*, що реалізовує

методи авторизованого запиту та логіку автоматичного, непомітного для користувача процесу оновлення токенів авторизації, якщо час життя токену доступу минув.

Також визначено необхідні компоненти для реалізації зручного інтерактивного інтерфейсу користувача, описано їхню взаємодію та реалізовано методи, що забезпечують необхідну функціональність застосунку.

Результатом роботи, описаної в цьому розділі, є повністю реалізовані клієнтська та серверна частини веб-застосунку для координації та агрегації громадських ініціатив, з дотриманням функціональних і нефункціональних вимог, та з використанням інструментів, сформованих та обгрунтованих у попередніх розділах дипломної роботи.

РОЗДІЛ 4. ПРЕДСТАВЛЕННЯ ТА АНАЛІЗ РЕАЛІЗОВАНОГО ВЕБ-ЗАСТОСУНКУ

4.1 Демонстрація сценаріїв використання веб-застосунку

В попередньому розділі було реалізовано клієнтську та серверну частини веб-застосунку для координації та агрегації громадських ініціатив. Розглянемо сценарії взаємодії користувача з системою та опишемо основні функції, які вона надає.

При першому зверненні за адресою додатку, користувач автоматично перенаправляється на сторінку для авторизації. Для виконання логіну необхідно ввести імейл та пароль, що були вказані при реєстрації. У разі успішної спроби авторизації користувач буде перенаправлений на сторінку додатку. Якщо введені дані будуть некоректними, то користувач отримає повідомлення про помилку валідації полів форми або невдалу спробу авторизації.

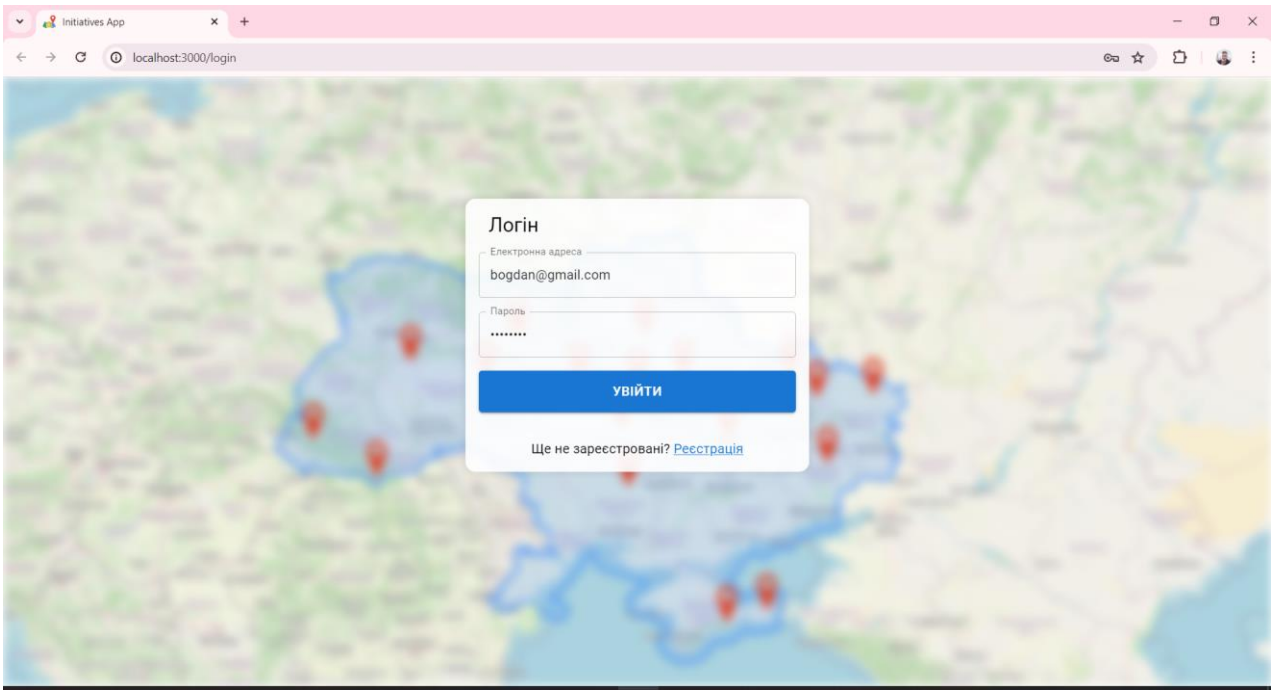


Рисунок 4.1 – Сторінка логіну

Якщо ж користувач ще не має створеного облікового запису в системі, він клікає за посиланням Реєстрація та переходить на сторінку реєстрації. У разі успішного заповнення форми створюється новий аккаунт для користувача системи та відбувається перенаправлення користувача на сторінку додатку.

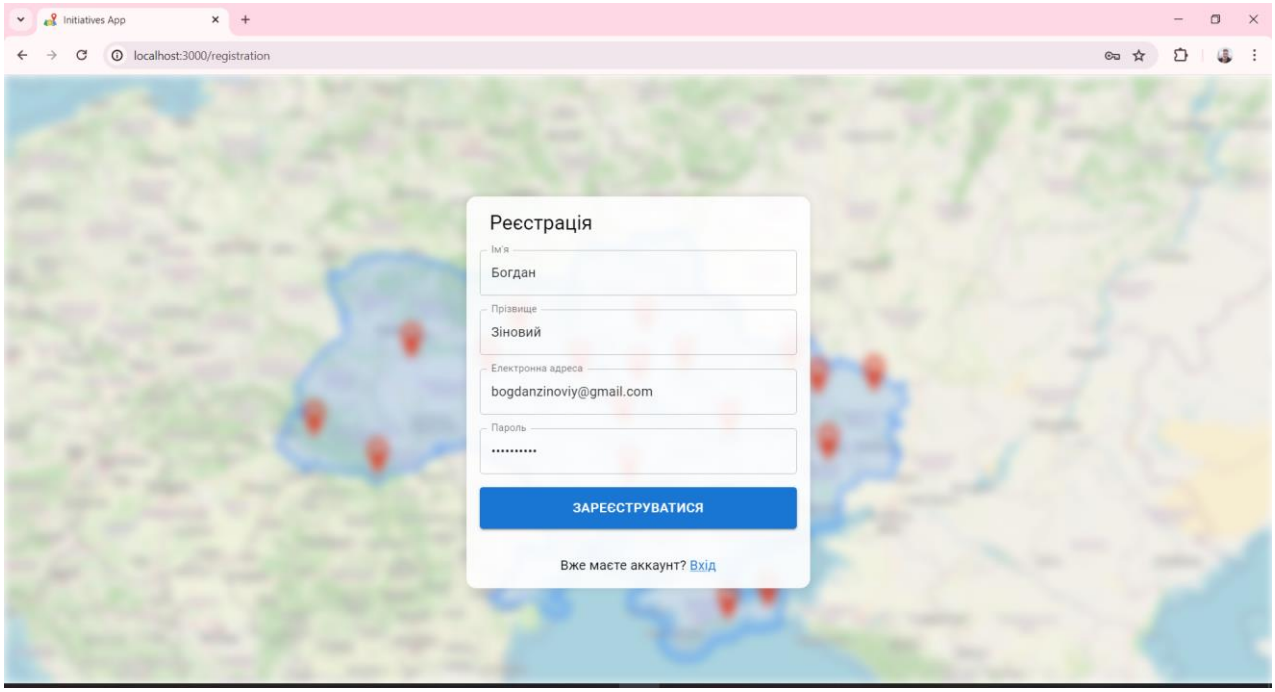


Рисунок 4.2 – Сторінка реєстрації

Головна сторінка додатку виглядає наступним чином. Основними компонентами для взаємодії з користувачем є інтерактивна мапа, що забезпечує механізм створення та перегляду ініціатив, компонент Меню з різними опціями доступних функції, компонент Пошук локації для зручного знаходження міста чи конкретної адреси на карті, компонент перегляду інформації про користувача.

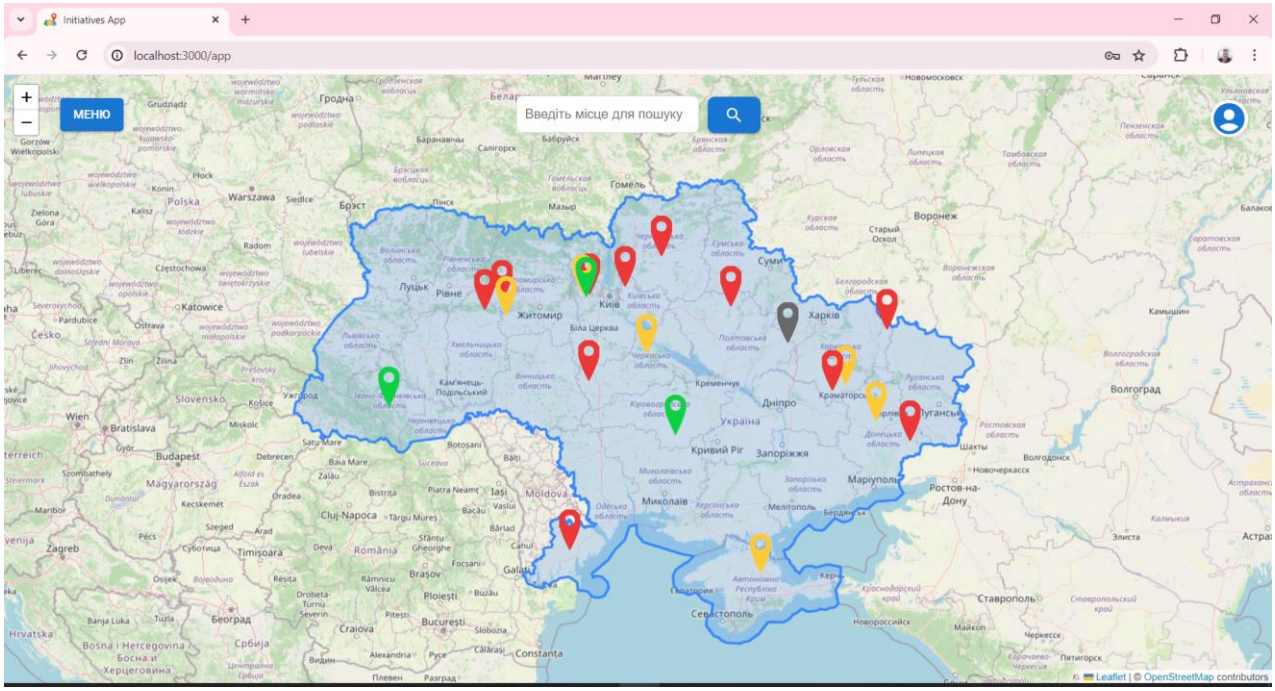


Рисунок 4.3 – Головна сторінка додатку

Для додавання нової ініціативи користувач обирає будь-яке місце на мапі, вільне від міток, та клікає по ньому. Відображається форма для створення ініціативи. Користувач вводить пов’язані дані, завантажує зображення та натискає кнопку для створення ініціативи. У разі проходження валідації, ініціатива додається на мапу, відображається компонент для її перегляду.

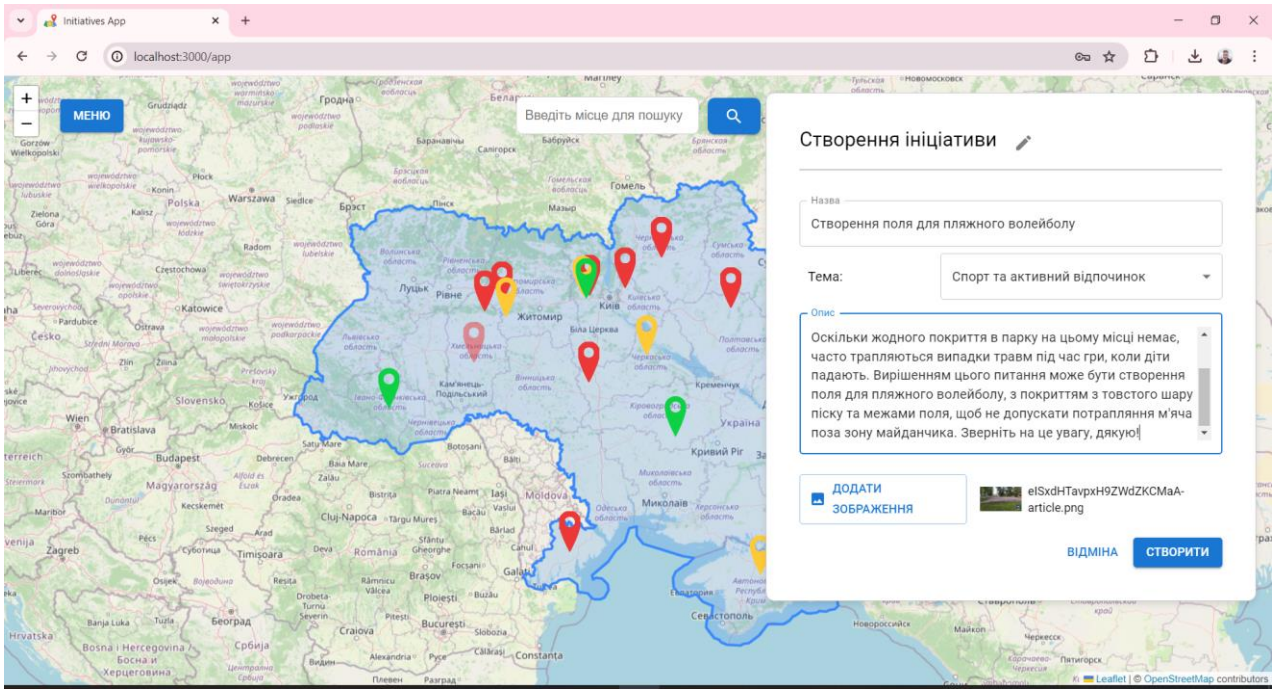


Рисунок 4.4 – Додавання ініціативи

Користувач переглядає створену ініціативу. Також перегляд ініціативи запускається щоразу при кліку на будь-яку існуючу мітку на карті. Відображається введена інформація при створенні ініціативи, її статус.

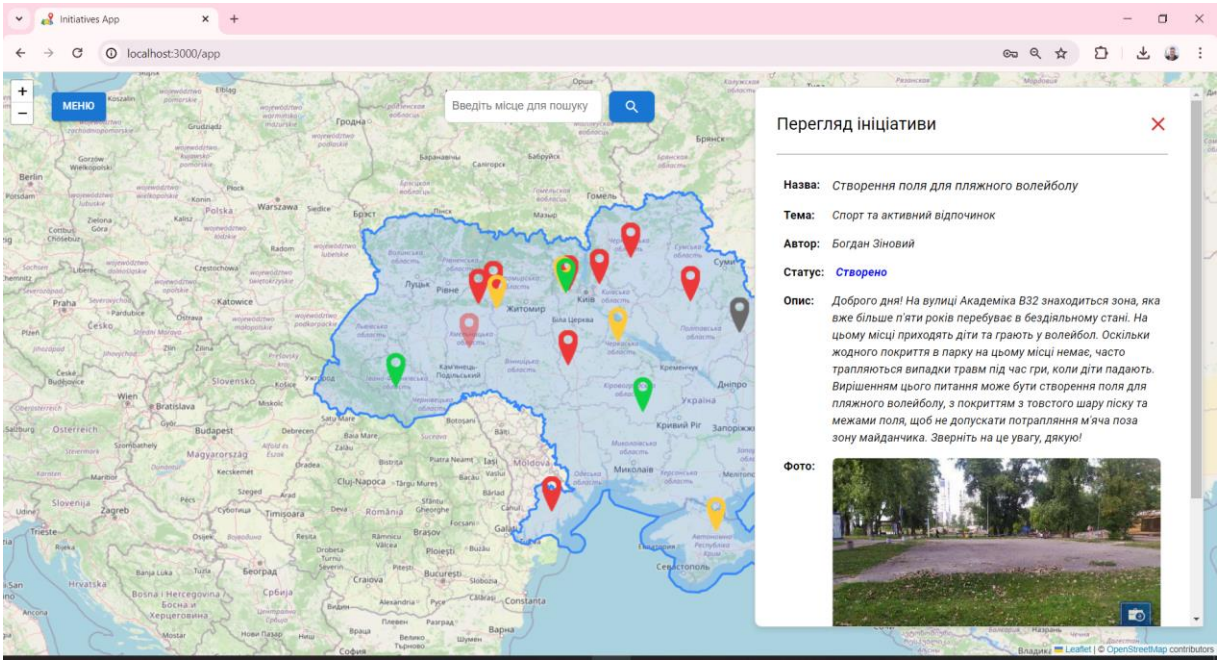


Рисунок 4.5 – Перегляд ініціативи

Також користувач може переглянути кількість вподобань та дату створення ініціативи, додати лайк. Внизу компонента доступні кнопки, що відповідають за відкриття розділу коментарів та відповідей адміністраторів системи.

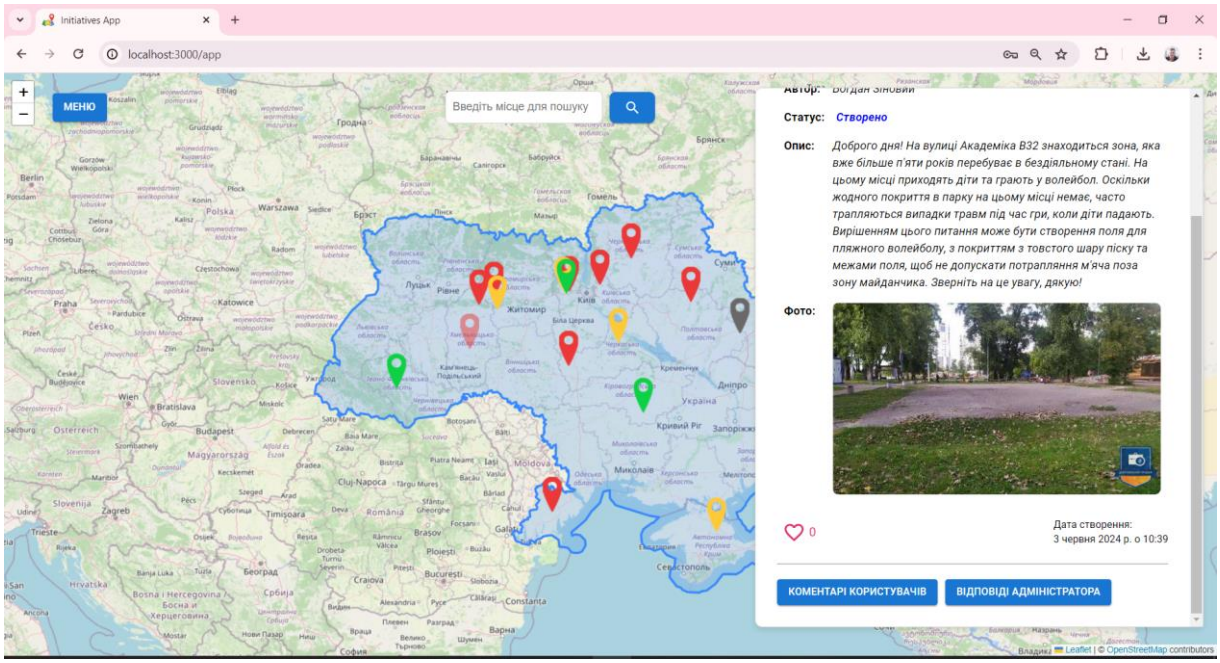


Рисунок 4.6 – Кількість вподобань та перехід до секції коментарів

При кліку на кнопку, що відкриває розділ коментарів користувачів, відображається поле для додавання власного коментаря, а також коментарі інших користувачів. Окрім можливості вподобати коментар, користувач може відсортувати коментарі за популярністю або датою створення.

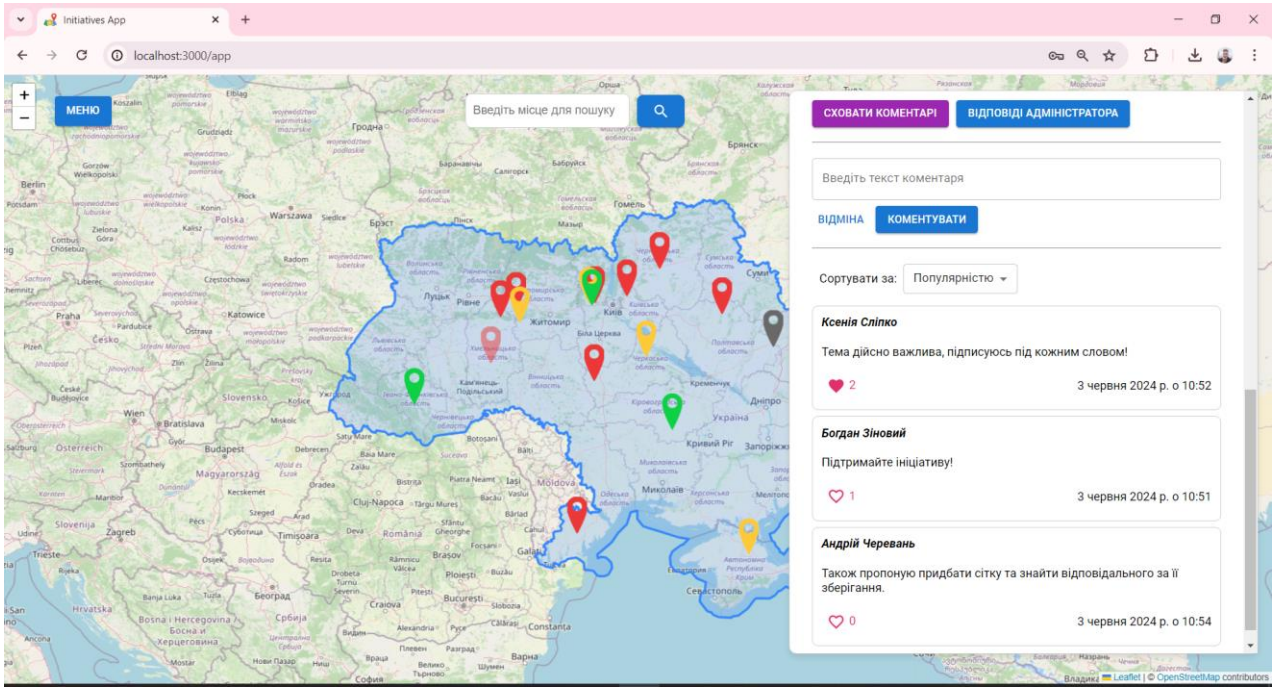


Рисунок 4.7 – Секція коментарів для користувачів з роллю «User»

Користувач з роллю адміністратора при перегляді ініціативи та коментарів користувачів має можливість видаляти ініціативу або якийсь конкретний коментар, реалізовуючи принципи модерації та видалення потенційно неприйнятного для інших користувачів вмісту.

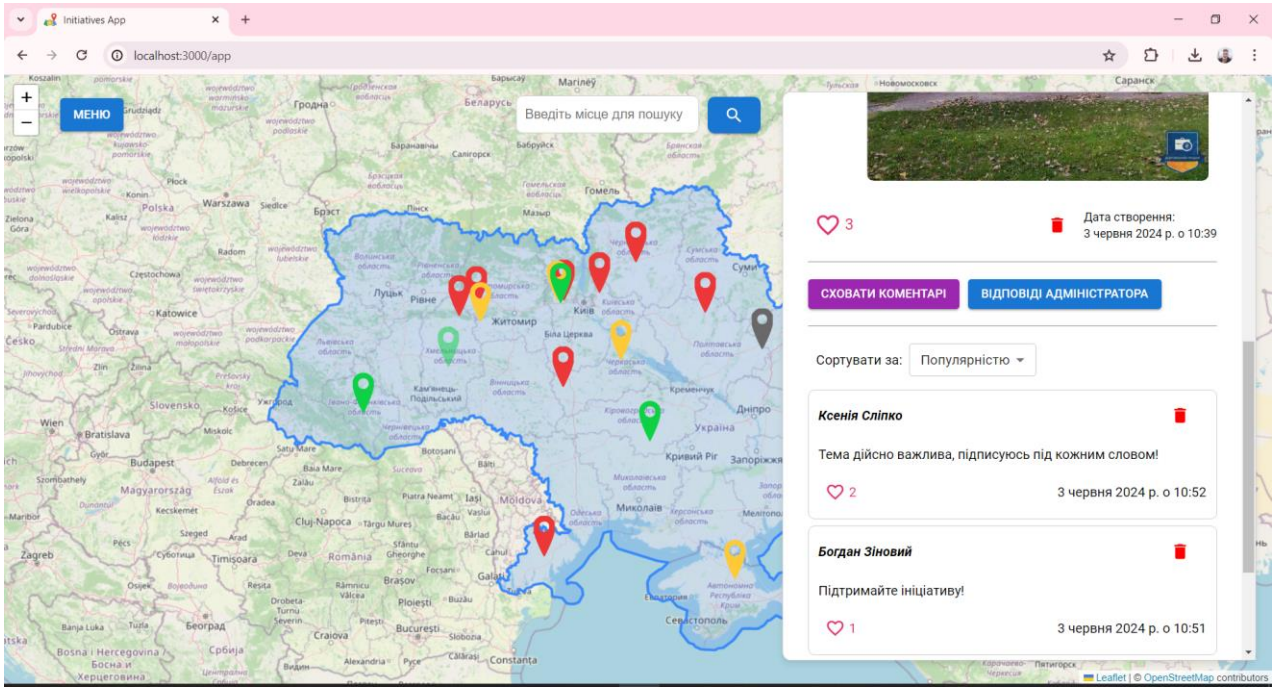


Рисунок 4.8 – Секція коментарів для користувачів з роллю «Admin»

При переході в розділ відповідей адміністратора, звичайний користувач ознайомлюється з інформацією, яка пов’язана зі станом виконання ініціативи та іншими важливими зауваженнями адміністраторів.

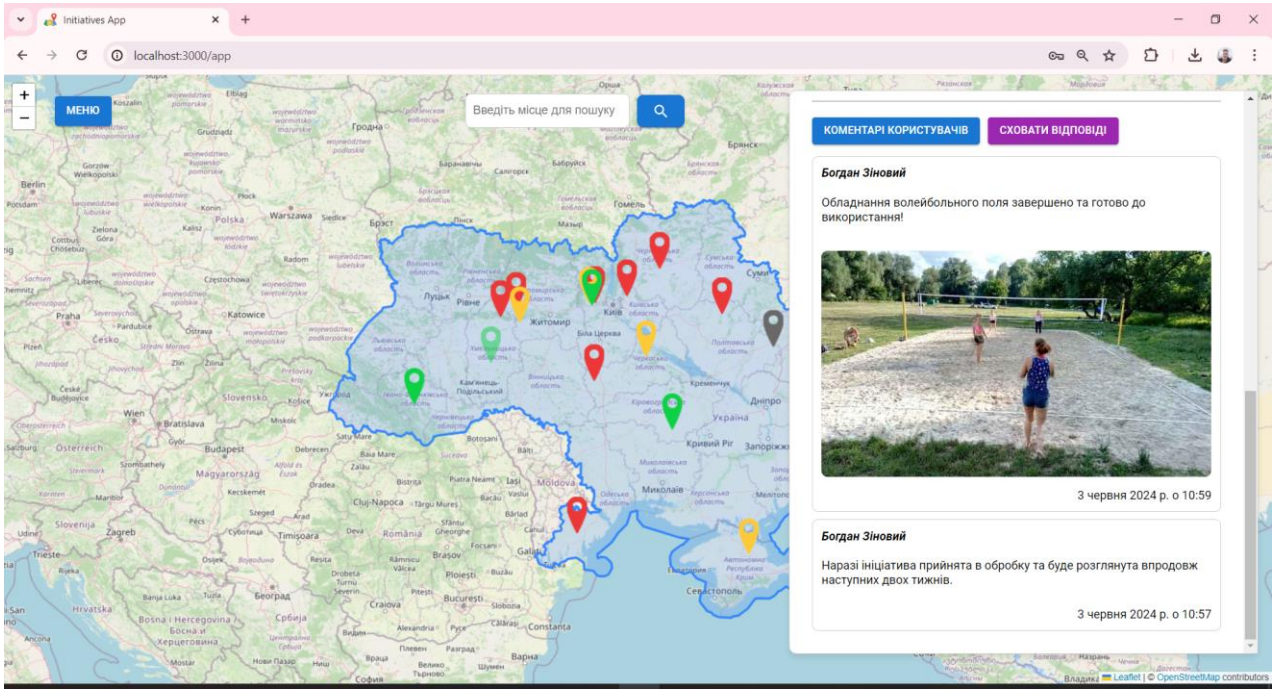


Рисунок 4.9 – Секція відповідей для користувачів з роллю «User»

Якщо користувач має роль адміністратора, для нього додається можливість додавати відповідь з текстом та зображенням, а також змінювати статус ініціативи у відповідному полі.

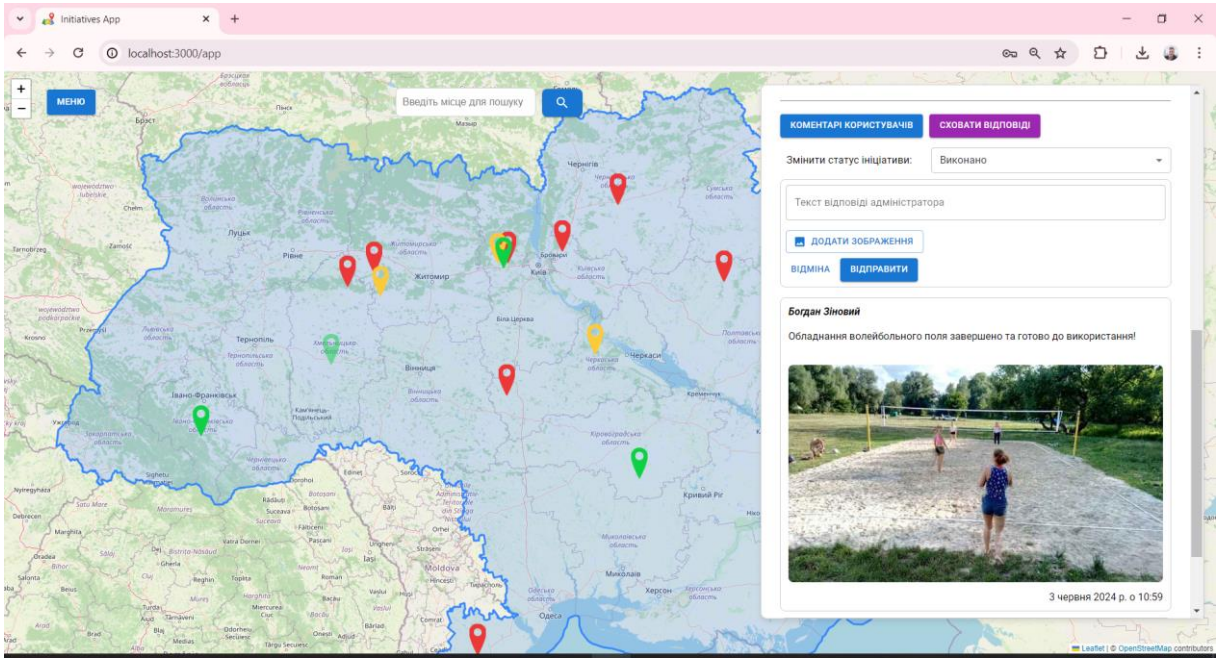


Рисунок 4.10 – Секція відповідей для користувачів з роллю «Admin»

Зверху по центру сторінки розташований компонент для зручного пошуку локації. Користувач вводить місто або конкретну адресу та отримує перелік найбільш релевантних результатів пошуку.

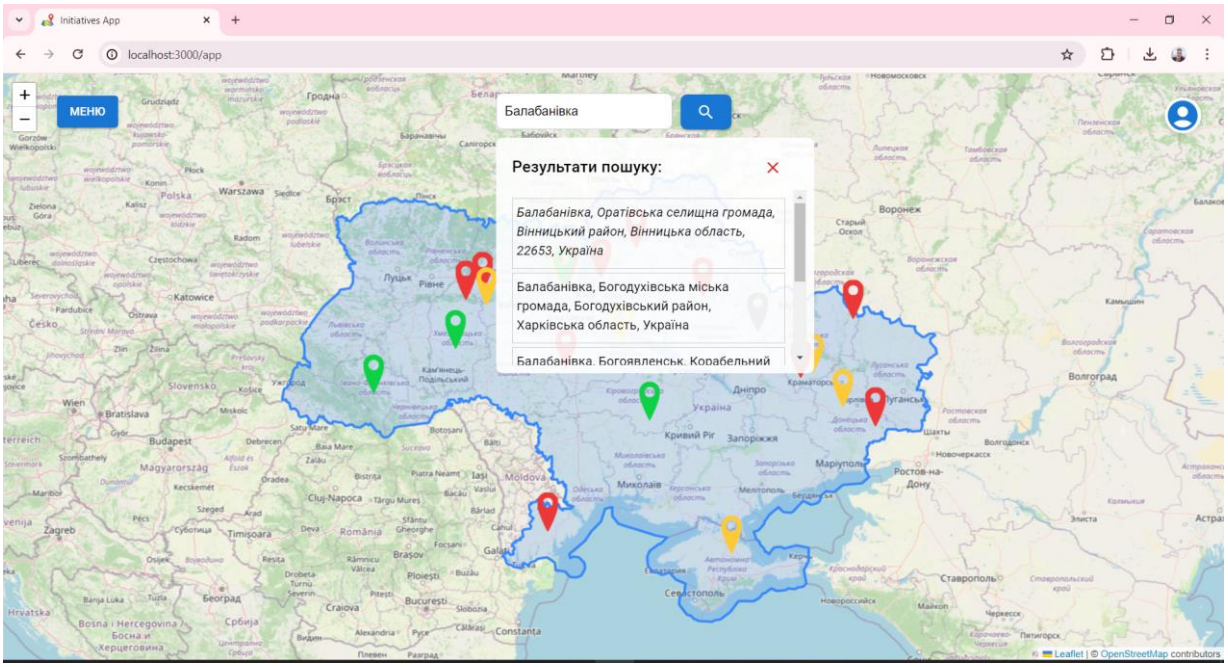


Рисунок 4.11 – Пошук локації на мапі

При кліку на потрібний результат пошуку, на карті буде відображено знайдене місце.

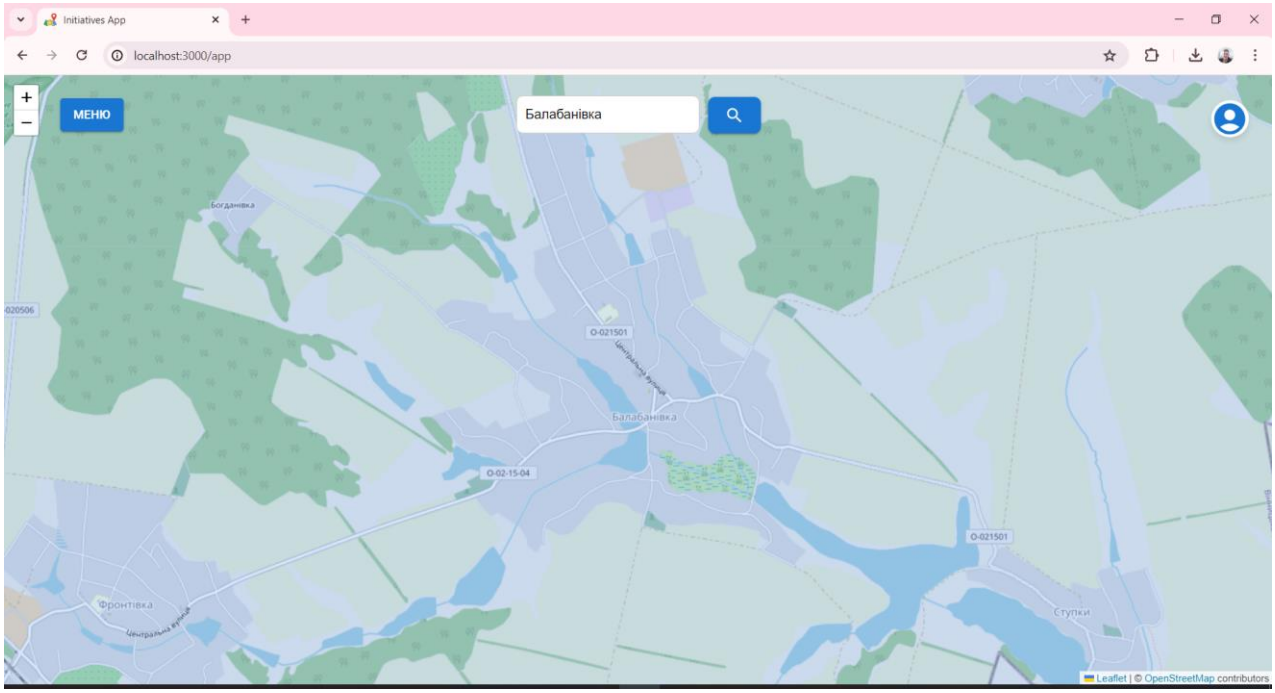


Рисунок 4.12 – Відображення обраного результату пошуку

При виклику меню, користувач обирає одну з опцій випадаючого списку. Наприклад, обравши опцію перегляду найпопулярніших ініціатив, на мапі будуть відображені 10 міток, що стосуються ініціатив з найбільною кількістю вподобань. Також можна обрати опцію, яка залишить на мапі лише ініціативи, додані користувачем. За замовчуванням в меню обрано пункт для відображення всіх ініціатив на мапі.

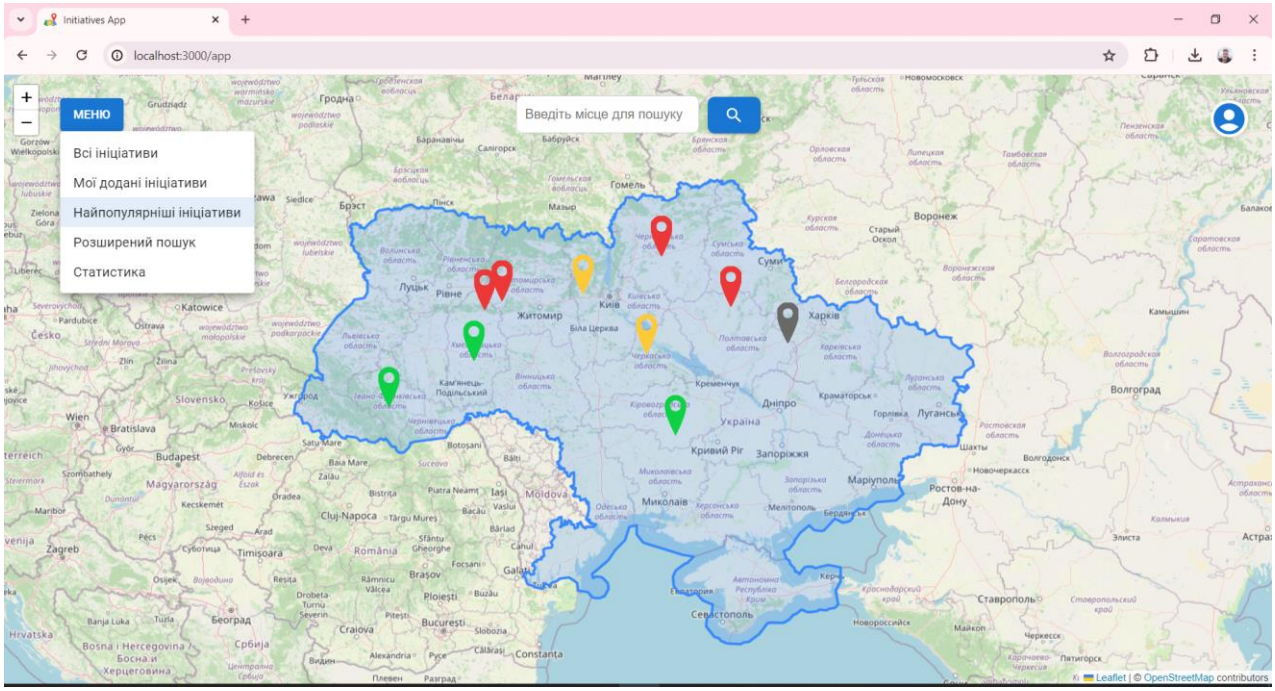


Рисунок 4.13 – Виклик опцій меню

Обравши опцію пошуку ініціатив, користувач вводить інформацію для фільтрації полів, що його цікавлять серед запропонованих. Доступний пошук за такими критеріями: назва ініціативи, тема, її статус, область, проміжок дат створення. При натисканні кнопки пошуку, на мапі відобразяться всі мітки, що відповідають обраним критеріям пошуку.

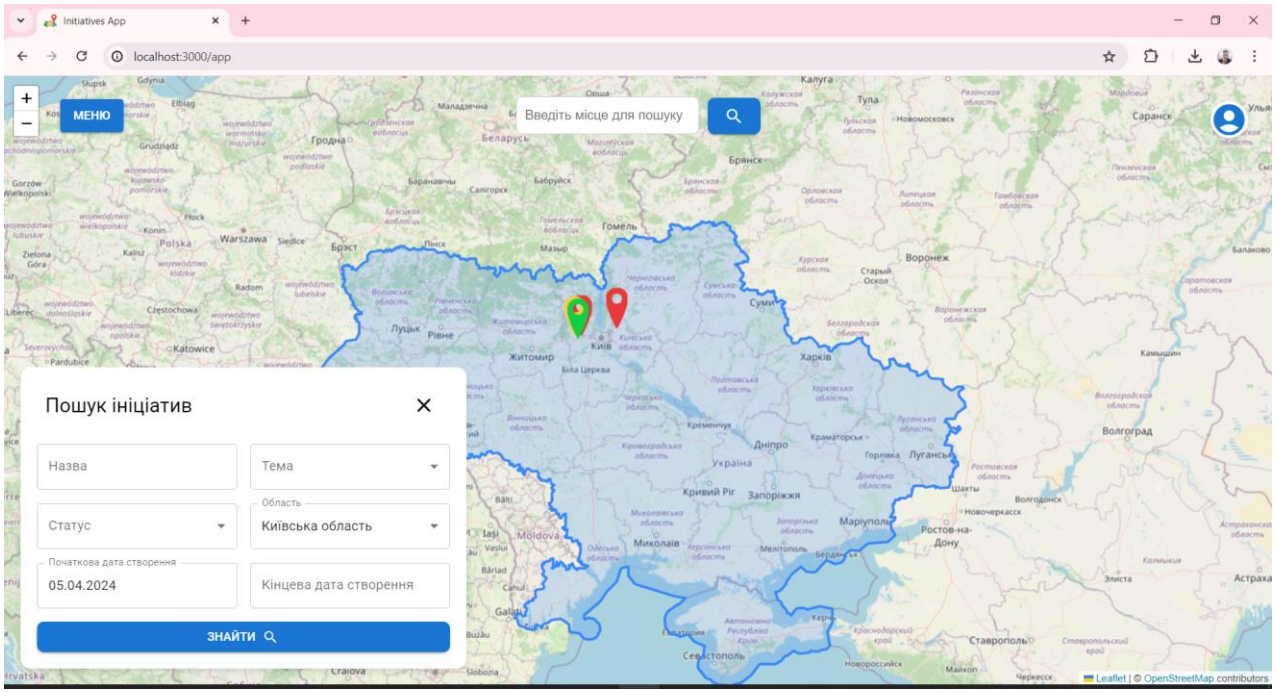


Рисунок 4.14 – Пошук ініціатив

Останньою опцією меню є перегляд статистики. У відповідному компоненті сторінки буде відображатися інформація про найпопулярніші теми, кількість звернень за статусами та області з найбільною кількість звернень впродовж останнього місяць.

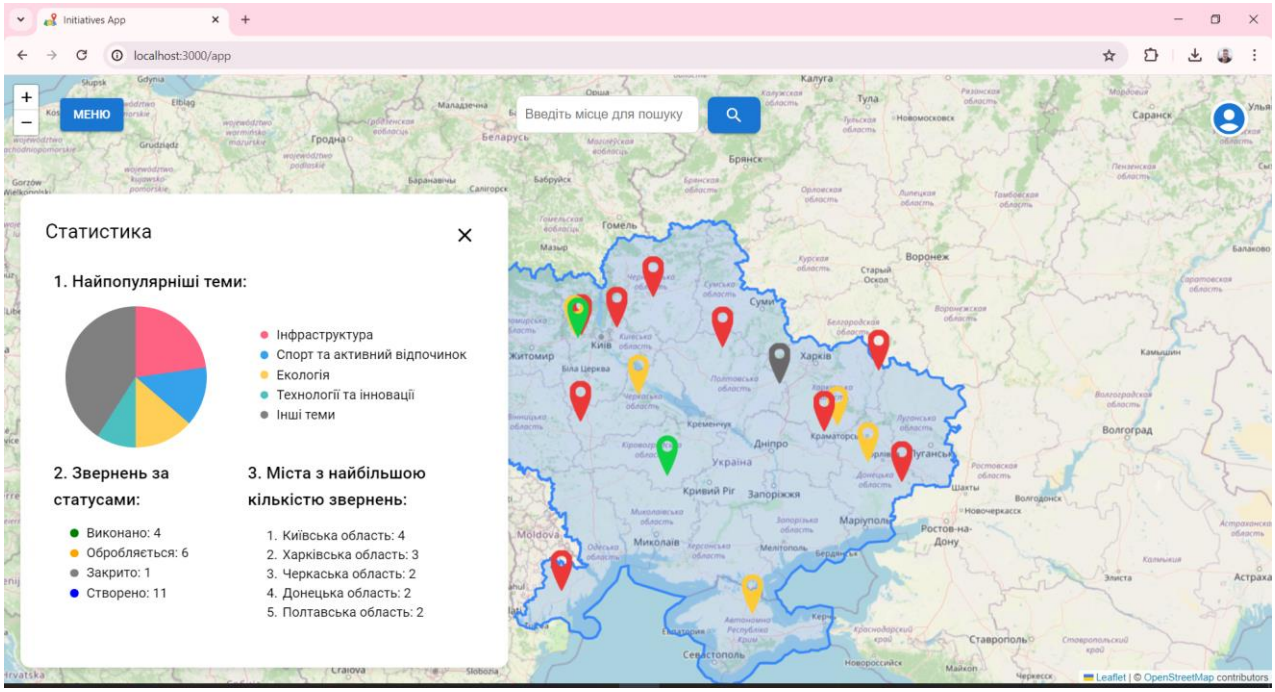


Рисунок 4.15 – Перегляд статистики

Для перегляду інформації користувача потрібно клікнути на кнопку з іконкою користувача. За бажанням, можна виконати логаут з системи, натиснувши на відповідну кнопку в компоненті.

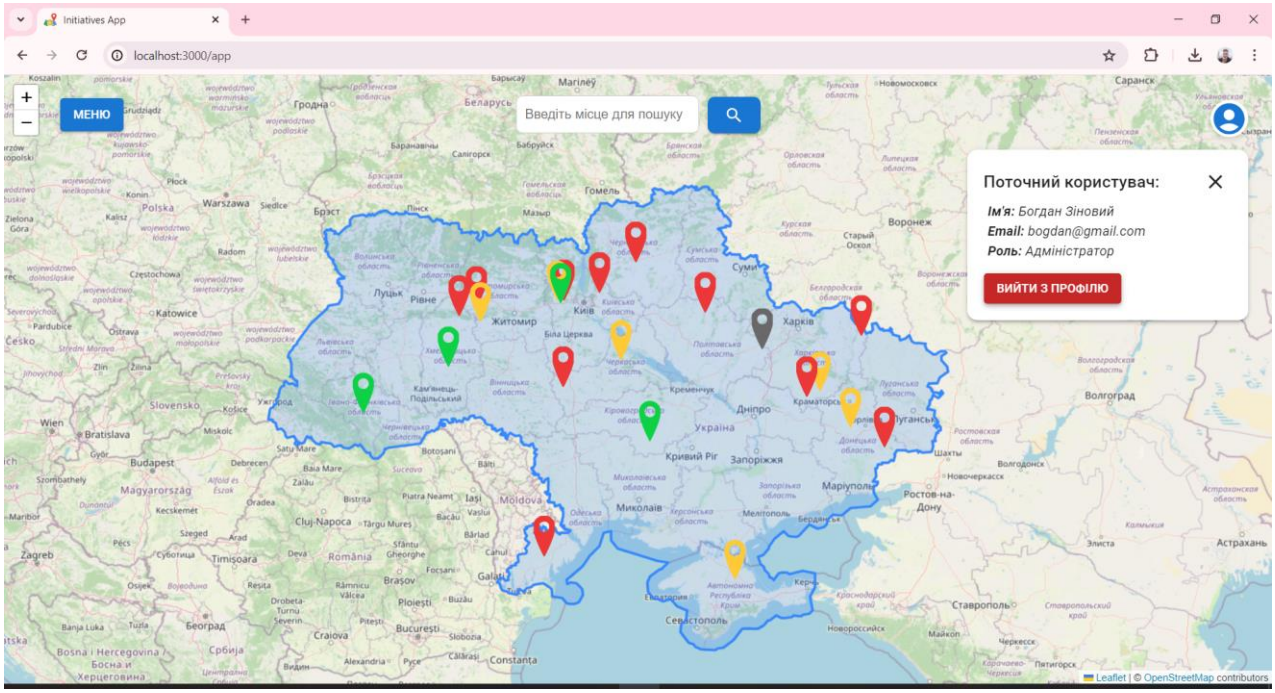


Рисунок 4.16 – Компонент інформації користувача та логаут з системи

Таким чином, веб-застосунок реалізовує всю необхідну функціональність і надає зручний та зрозумілий інтерфейс для взаємодії користувача з системою.

4.2 Порівняння реалізованої системи з конкурентами

Порівнюючи реалізовану систему з уже наявними рішеннями, розглянутими у першому розділі дипломної роботи, можна виокремити моменти, які в розробленому веб-застосунку для координації та агрегації громадських ініціатив, реалізовані краще за конкурентів.

Одним з таких рішень є одночасна реалізація кількох функцій, якої немає у жодного з існуючих аналогів: це можливість додавати ініціативи з використанням мапи, для всієї території України та без обмеження тематики звернення.

Ще однією особливістю застосунку є те, що навідміну від конкурентів, він орієнтований саме на візуалізацію місць звернень на карті та використання інтерактивної мапи для додавання ініціатив та їх перегляду.

Тим не менше, розглянуті аналоги, що також надають можливість створення ініціатив, мають у певній мірі більш стилізований та адаптивний інтерфейс користувача. Також серед переваг аналогів над розробленим продуктом можна виділити повноцінно реалізований механізм взаємодії між поданням звернень та реакцією на них з боку влади. В розробленому застосунку реалізована можливість виконання дій адміністраторів для модерації контенту та інформування про статус звернень, проте немає чіткого зв'язку з органами виконавчої влади. Але варто зазаначити, що основною функцією системи є саме агрегація та координація громадських ініціатив, що виконується в повному обсязі. Цей момент розглянуто лише як потенційний варіант розширення можливостей та функцій системи вцілому.

Таким чином, можна підсумувати, що реалізований веб-застосунок у чомусь поступається наявним рішенням, але тим не менше, він має свої переваги над ними, реалізовуючи набір необхідних функцій з більшим нахилом у сторону агрегації та візуалізації даних.

4.3 Способи покращення веб-застосунку

Насамперед, варто зазначити, що реалізована система для координації та агрегації громадських ініціатив задовольняє всі сформовані на початку розробки функціональні та нефункціональні вимоги. Але в більшості випадків можна зробити програмний продукт більш зручним, продуктивним та визначити основні напрямки його розвитку.

Перш за все, щоб покращити взаємодію користувача з додатком, для таких функцій меню, як пошук ініціативи, перегляд ініціатив користувача чи найпопулярніших ініціатив, зробити не тільки як відображення міток на карті, а ще й у формі списку, щоб користувач одразу міг побачити назву, тему ініціативи та її кількість вподобань. Такий підхід допоможе внести більшої

ясноті у відповідності інформації про ініціативи та їхнє розташування на карті та мати змогу ознайомитися з їхнім змістом без повного завантаження пов’язаної з ініціативої інформації, що також вплине на підвищення продуктивності застосунку.

Другим покращенням може стати зміна підходу до відображення всіх міток на карті, за якого при зменшенні масштабу карти, мітки, що будуть знаходитися близько одна до одної, будуть позначатися як одна мітка з певною кількістю міток, які в ній містяться, а вже при збільшенні масштабу вони будуть відображатися роздільно. Такий підхід також може підвищити продуктивність клієнтської частини застосунку, адже вимагатиме меншого часу та ресурсів для відображення меншої кількості об’єктів на мапі одночасно.

Також ще одним можливим покращенням може стати реалізація окремої клієнтської частини для адмін-версії застосунку. Наразі в реалізованій системі адміністратори мають інструменти для виконання своїх основних функцій, проте набагато зручніше було б створити окремий інтерфейс для взаємодії з контентом з використанням спеціалізованих для них інструментів.

Запропоновані зміни мають покращити загальну продуктивність та зручність користування системою, але, тим не менше, реалізована на даному етапі система уже задовольняє базові вимоги та реалізовує набір функцій, необхідних для повноцінної роботи застосунку.

ВИСНОВОК ДО РОЗДІЛУ 4

У даному розділі дипломної роботи було розглянуто основні сценарії використання розробленої системи. Зокрема описано основні функції веб-застосунку, починаючи від реєстрації, продовжуючи створенням, переглядом, пошуком, коментуванням ініціатив та переглядом статистики, закінчуючи логаутом користувача. Також розглянуто сценарії та додаткові можливості, що надаються користувачам системи, які мають роль адміністратора. Наведено скріншоти роботи застосунку, що демонструють реалізовану функціональність.

Також виконано порівняння реалізованого веб-застосунку з уже наявними рішеннями. У результаті визначено, що розроблена система має як свої переваги порівняно з конкурентами, так і певні недоліки. Головним недоліком є відсутність чіткого механізму взаємодії з органами виконавчої влади. Серед переваг можна виокремити реалізацію одночасного набору функцій, таких як можливість додавати ініціативи на території всієї України, на будь які теми та з використанням інтерактивної мапи. Також особливістю додатку є його акцент в сторону візуалізації даних.

Наостанок, розглянуто основні способи покращення застосунку. Серед перелічених способів були відображення пошуку та певних компонентів меню як списку ініціатив, а не просто відфільтрований набір міток на карті, адаптоване відображення кількості міток на карті в залежності від її масштабу та створення версії клієнтської частини веб-застосунку для адміністраторів системи.

Таким чином, у цьому розділі виконано огляд веб-застосунку для координації та агрегації громадських ініціатив, виконано порівняння реалізованої системи з існуючими рішеннями та сформовано основні напрямки, в яких можна покращити загальний досвід користування застосунком та його продуктивність.

ВИСНОВКИ

В даній дипломній роботі було описано етапи розробки системи для агрегації та координації громадських ініціатив.

У першому розділі було виконано аналіз предметної області та порівняння існуючих рішень, які мають аналогічну або схожу тематику системи. Зокрема розглянуто 3 системи, серед яких: «Урядовий контактний центр», «Контактний центр м. Київ» та «Карта відновлення». Виокремлено переваги та недоліки кожної з систем та визначено критерії, за рахунок яких розроблювана система буде кращою за конкурентів. Такими критеріями визнано більший нахил в сторону візуалізації даних з використанням інтерактивної мапи, а також реалізацію кількох основних функцій, які не реалізовані в жодній із систем одночасно.

В другому розділі було описано загальну архітектуру веб-застосунку, визначено, що він складатиметься з клієнтської та серверної частин. Обрано та обгрунтовано основні засоби реалізації.

В третьому розділі описано весь процес реалізації системи. В пункті, що стосується серверної частини застосунку, спочатку було розглянуто проектування та підключення бази даних, підключення до хмарного сховища для збереження зображень. Визначено перелік програмних модулів та створено сутності бази даних із визначеними зв’язками.

Щодо клієнтської частини застосунку, було описано використані статичні файли проекту, створено клас HttpClient, який реалізовує авторизовані звернення до серверної частини застосунку та обробку відповідей, описано основні компоненти користувацького інтерфейсу, визначено їхню взаємодію та реалізовані функції кожного із них, розглянуто маршрутизацію та навігацію веб-застосунку.

В четвертому розділі наведено приклади сценаріїв використання додатку, скріншоти тестування системи та описано основні функції, доступ до яких мають звичайні користувачі та адміністратори системи. Також виконано фінальне порівняння розробленої системи з наявними рішеннями, розглянутими ще в першому розділі та виокремлено, в чому система поступається конкурентам та за рахунок яких функцій вона має над ними перевагу. Наприкінці, розглянуто кілька можливих способів покращити досвід користування системою та підвищити її загальну продуктивність.

Результатом дипломної роботи є розроблений веб-застосунок для координації та агрегації громадських ініціатив, що задовольняє функціональні та нефункціональні вимоги до системи та надає необхідну функціональність для вирішення поставлених перед нею задач.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Урядовий контактний центр. URL: https://ukc.gov.ua/ (дата звернення: 12.03.2024).
2. Контактний центр міста Київ. 15-51.gov.ua. URL: https://1551.gov.ua/ (дата звернення: 12.03.2024).
3. Карта відновлення та руйнувань. URL: https://reukraine.shtab.net/ (дата звернення: 13.03.2024).
4. Fowler M. Patterns of enterprise application architecture. Boston : Addison-Wesley, 2002. 557 p.
5. Pfleeger S. L., Atlee J. M. Software Engineering: Theory and Practice. 5th ed. Upper Saddle River : Pearson, 2023. 720 p.
6. Smith J. Understanding Backend Web Architecture. Web Dev Journal. 2022. URL: https://webdevjournal.com/articles/backend-architecture (дата звернення: 18.03.2024).
7. NestJS - A progressive Node.js framework. URL: https://docs.nestjs.com/v5/ (дата звернення: 22.03.2024).
8. How NestJS Design Patterns Help Build Modular, Scalable, and Maintainable Applications. Medium. URL: https://medium.com/virtual-force-inc/how-nestjs-design-patterns-help-build-modular-scalable-and-maintainable-applications-801bf1bb5b2c (дата звернення: 24.03.2024).
9. TypeScript vs JavaScript: Which Programming Language Fits Your Project? Intellisoft. URL: https://intellisoft.io/typescript-vs-javascript-which-programming-language-fits-your-project/ (дата звернення: 26.03.2024).
10. PostgreSQL Documentation. URL: https://www.postgresql.org/docs/ (дата звернення: 27.03.2024).

11. TypeORM Documentation. URL: https://typeorm.io/ (дата звернення: 27.03.2024).
12. Azure Blob Storage Documentation. URL: https://learn.microsoft.com/en-us/azure/storage/blobs/ (дата звернення: 01.04.2024).
13. JWT Introduction. URL: https://jwt.io/introduction (дата звернення: 01.04.2024).
14. Strategies to Build React Apps in a Client-side Architecture. The New Stack. URL: https://thenewstack.io/strategies-to-build-react-apps-in-a-client-side-architecture/ (дата звернення: 04.04.2024).
15. MDN Web Docs. JavaScript. URL: https://developer.mozilla.org/en-US/docs/Learn/JavaScript (дата звернення: 06.04.2024).
16. Leaflet Quick Start Guide. URL: https://leafletjs.com/examples/quick-start/ (дата звернення: 10.04.2024).
17. Nominatim API Overview. URL: https://nominatim.org/release-docs/latest/api/Overview/ (дата звернення: 13.04.2024).
18. Material-UI Documentation. URL: https://mui.com/material-ui/ (дата звернення: 07.05.2024).
19. Smolinari. NestJS and Project Structure: What to Do. Dev.to. URL: https://dev.to/smolinari/nestjs-and-project-structure-what-to-do-1223 (дата звернення: 20.04.2024).
20. Manage Azure Blob Storage in NestJS. Dev Genius. URL: https://blog.devgenius.io/manage-azure-blob-storage-in-nestjs-daf5cb5125d4 (дата звернення: 22.04.2024).
21. How to Store Images in a Database. Beekeeper Studio. URL: https://www.beekeeperstudio.io/blog/how-to-store-images-in-a-database (дата звернення: 23.04.2024).
22. Many-to-Many Relations in TypeORM. URL: https://orkhan.gitbook.io/typeorm/docs/many-to-many-relations (дата звернення: 23.04.2024).

23. TypeORM Entities Documentation. URL: https://orkhan.gitbook.io/typeorm/docs/entities (дата звернення: 25.04.2024).
24. Custom Providers in NestJS. URL: https://docs.nestjs.com/fundamentals/custom-providers (дата звернення: 25.04.2024).
25. CRUD: Definition and Operation. DataScientest. URL: https://datascientest.com/en/crud-definition-and-operation (дата звернення: 26.04.2024).
26. Authenticating Users with JWT. Google Cloud. URL: https://cloud.google.com/api-gateway/docs/authenticating-users-jwt (дата звернення: 26.04.2024).
27. Passport-JWT Documentation. URL: https://www.passportjs.org/packages/passport-jwt/ (дата звернення: 26.04.2024).
28. Custom Decorators in NestJS. URL: https://docs.nestjs.com/custom-decorators (дата звернення: 01.06.2024).
29. TypeORM Migrations Documentation. URL: https://typeorm.io/migrations (дата звернення: 06.06.2024).
30. React Architecture Patterns and Best Practices. GeeksforGeeks. URL: https://www.geeksforgeeks.org/react-architecture-pattern-and-best-practices/ (дата звернення: 20.03.2024).
31. MDN Web Docs. Single Page Application (SPA). URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (дата звернення: 16.04.2024).
32. Formik Documentation. URL: https://formik.org/docs/overview (дата звернення: 18.04.2024).
33. Mastering Form Validation in React with Formik and Yup with TypeScript. Medium. URL: https://olaishola.medium.com/mastering-form-validation-

in-react-with-formik-and-yup-with-typescript-9dc4b3885538 (дата звернення: 18.04.2024).

34. React Component Communication. Medium. URL: https://waxlyrical.medium.com/react-component-communication-2d9cf0f5f797 (дата звернення: 23.04.2024).
35. Routing in React. Hygraph. URL: https://hygraph.com/blog/routing-in-react (дата звернення: 25.04.2024).

ДОДАТОК 1

Веб-застосунок для координації та агрегації громадських ініціатив

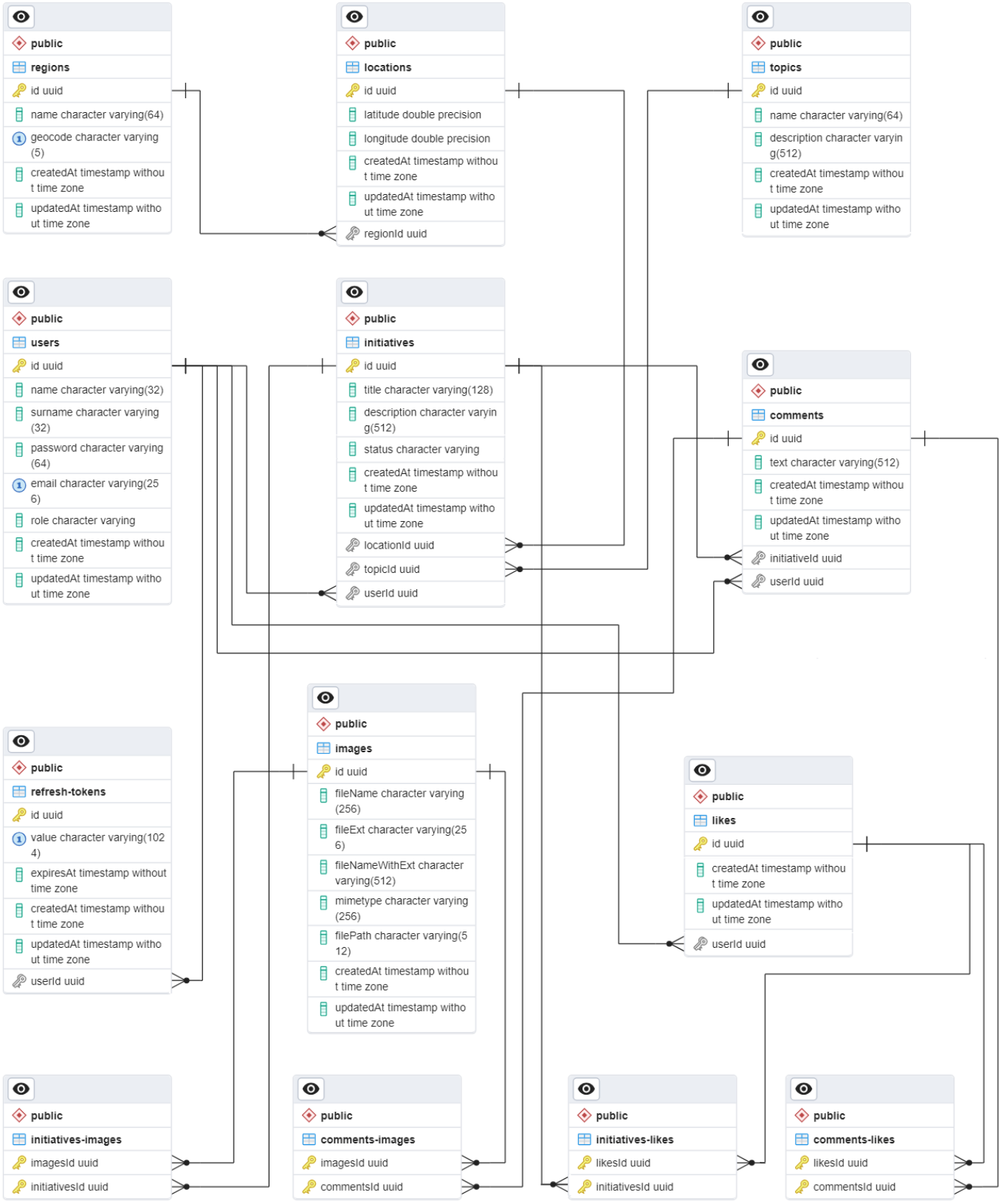
Діаграма бази даних системи

(функціональна схема)

ІАЛЦ.467200.004 Д1

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.004 Д1
		№ докум.	Підпис	Дата	
Розробив		Зіновий Б. О.			**Веб-застосунок для координації та агрегації громадських ініціатив** Діаграма бази даних системи (функціональна схема)
Перевірив		Ковальчук О. М.			
Н. Контр.		Волокита А. М.			
Затвердив					

Літ.	Аркуш	Аркушів
	1	1

КПІ ім. Ігоря Сікорського, ФІОТ, ІП-04

ДОДАТОК 2

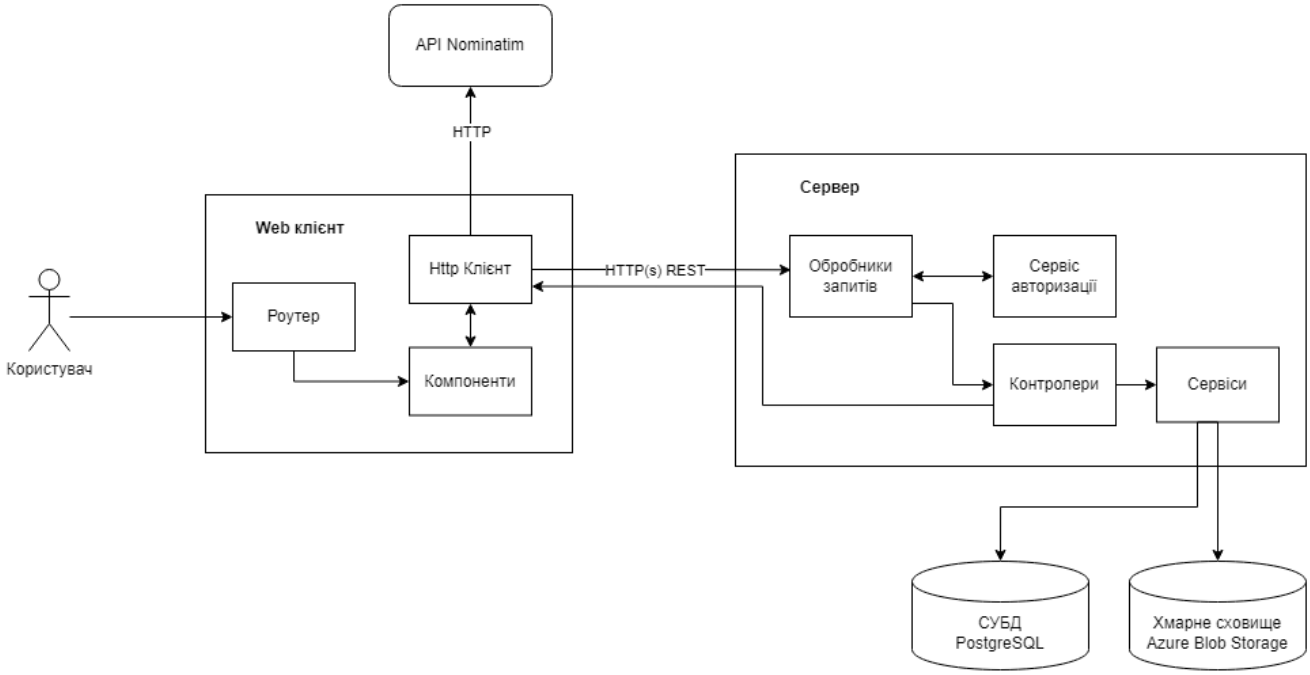
Веб-застосунок для координації та агрегації громадських ініціатив

Компонентна структура системи (структурна схема)

ІАЛЦ.467200.005 Д2

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.005 Д2	
		№ докум.	Підпис	Дата		
Розробив		Зіновий Б. О.			**Веб-застосунок для координації та агрегації громадських ініціатив** Компонентна структура системи (структурна схема)	Літ. / Аркуш / Аркушів: 1 / 1
Перевірив		Ковальчук О. М.				
Н. Контр.		Волокита А. М.				***КПІ ім. Ігоря Сікорського, ФІОТ, ІП-04***
Затвердив						

ДОДАТОК 3

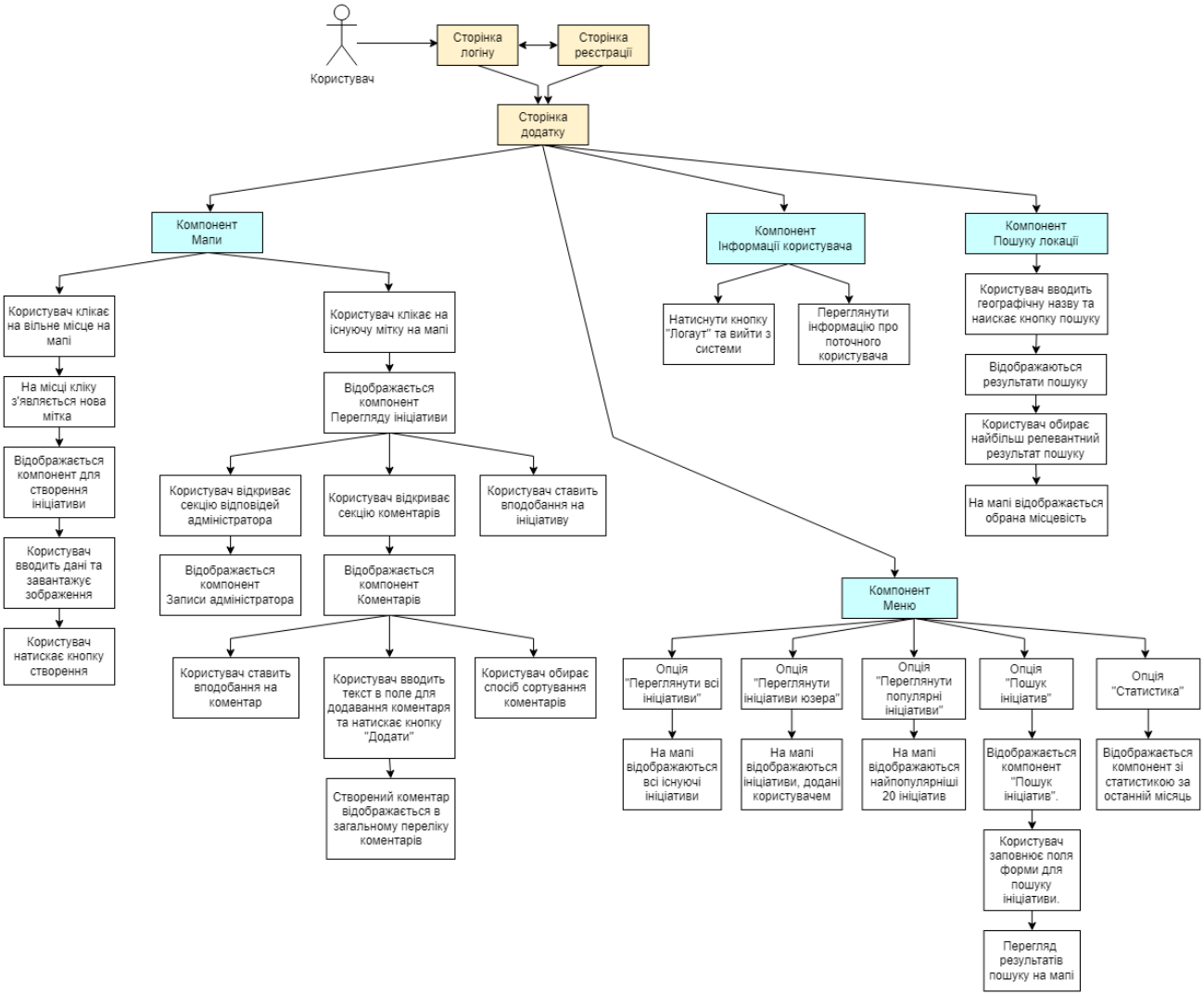
Веб-застосунок для координації та агрегації громадських ініціатив

Алгоритм дій користувача
(принципова схема)

ІАЛЦ.467200.006 Д3

Аркушів 1

Київ 2024 р



					ІАЛЦ.467200.006 Д3	
		№ докум.	Підпис	Дата		
Розробив		Зіновий Б. О.			**Веб-застосунок для координації та агрегації громадських ініціатив** Алгоритм дій користувача (принципова схема)	Літ. / Аркуш 1 / Аркушів 1
Перевірив		Ковальчук О. М.				
Н. Контр.		Волокита А. М.				
Затвердив						***КПІ ім. Ігоря Сікорського, ФІОТ, ІП-04***

ДОДАТОК 4

Веб-застосунок для координації та агрегації громадських ініціатив

Текст програмного коду

ІАЛЦ.467200.007 Д4

Аркушів 78

Київ 2024 р

app.module.ts

```
import { Module } from '@nestjs/common';
import { DatabaseModule } from './systems/database';
import { UsersModule } from './modules/users';
import { AuthModule } from './modules/auth';
import { LocationsModule } from './modules/locations';
import { TopicsModule } from './modules/topics';
import { CommentsModule } from './modules/comments';
import { InitiativesModule } from './modules/initiatives';
import { ImagesModule } from './modules/images';
import { RegionsModule } from './modules/regions';
import { LikesModule } from './modules/likes';

@Module({
  imports: [
    DatabaseModule,
    UsersModule,
    AuthModule,
    LocationsModule,
    RegionsModule,
    TopicsModule,
    CommentsModule,
    InitiativesModule,
    LikesModule,
    ImagesModule,
  ],
  controllers: [],
  providers: [],
})
export class AppModule {}
```

app-config.service.ts

```
import { Injectable } from '@nestjs/common';
import { ConfigService } from '@nestjs/config';
import { TypeOrmModuleOptions } from '@nestjs/typeorm';
import { join } from 'path';

@Injectable()
export class AppConfigService {
  constructor(private configService: ConfigService) {}

  getDatabaseConfig(): TypeOrmModuleOptions {
    return {
      name: this.get('TYPEORM_CONNECTION_NAME'),
      type: this.get<'postgres'>('TYPEORM_TYPE'),
      host: this.get('TYPEORM_HOST'),
      port: this.get('TYPEORM_PORT'),
      cache: this.get('TYPEORM_CACHE'),
      logging: this.get('TYPEORM_LOGGING'),
      database: this.get<string>('TYPEORM_DATABASE'),
      username: this.get('TYPEORM_USERNAME'),
      password: this.get<string>('TYPEORM_PASSWORD'),
      extra: {
        ssl: this.get('TYPEORM_SSL'),
      },
      dropSchema: this.get('TYPEORM_DROP_SCHEMA'),
      synchronize: this.get('TYPEORM_SYNCHRONIZE'),
```

```
      migrationsRun: this.get('TYPEORM_MIGRATIONS_RUN'),
      entities: [join(__dirname, '../modules/**/*.entity.{ts,js}')],
      migrations: [join(__dirname, '../systems/database/migrations/*.{ts,js}')],
    };
  }

  get<T>(key: string): T {
    const value = this.configService.get<T>(key);

    if (!value) {
      throw new Error(key + ' environment variable is not set');
    }

    try {
      return JSON.parse(value as string);
    } catch {
      return value;
    }
  }
}
```

database.module.ts

```
import { Module } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { AppConfigModule } from '../../config';
import { AppConfigService } from '../../config/app-config.service';

@Module({
  imports: [
    TypeOrmModule.forRootAsync({
      imports: [AppConfigModule],
      inject: [AppConfigService],
      useFactory: (configService: AppConfigService) => ({
        ...configService.getDatabaseConfig(),
      }),
    }),
  ],
  exports: [TypeOrmModule],
})
export class DatabaseModule {}
```

storage.service.ts

```
import {
  BlobDeleteResponse,
  BlobServiceClient,
  BlobUploadCommonResponse,
  BlockBlobParallelUploadOptions,
} from '@azure/storage-blob';
import { v4 as uuidv4 } from 'uuid';
import { Injectable, BadRequestException } from '@nestjs/common';
import { join, parse } from 'node:path';
import { STORAGE_DEFAULT_UPLOAD_OPTIONS } from './storage.constants';
import { ImageEntity } from 'src/modules/images/image.entity';
import { AppErrorMessagesEnum } from 'src/common/enums';
import { AppConfigService } from 'src/config/app-config.service';

@Injectable()
export class StorageService {
  private readonly client: BlobServiceClient;
  private readonly blobContainerName: string;
```

```
constructor(private readonly appConfigService: AppConfigService) {
  const connectionString = this.appConfigService.get<string>(
    'AZURE_BLOB_STORAGE_CONNECTION_STRING',
  );

  this.client = BlobServiceClient.fromConnectionString(connectionString);
  this.blobContainerName = this.appConfigService.get<string>(
    'AZURE_BLOB_STORAGE_CONTAINER_NAME',
  );
}

async createOne(
  file: any,
  pathToFile: string,
  options?: BlockBlobParallelUploadOptions,
): Promise<Partial<ImageEntity>> {
  const { originalname: filename, mimetype } = file;

  const uploadOptions = options ?? {
    blobHTTPHeaders: { blobContentType: mimetype },
  };

  const { name, ext } = parse(filename);
  const randomUuid = uuidv4();
  const filePath = join(pathToFile, `${randomUuid}${ext}`).replace(
    /\\/g,
    '/',
  );

  await this.uploadOne(file, filePath, uploadOptions);

  return {
    fileName: name,
    fileNameWithExt: filename,
    fileExt: ext,
    filePath,
    mimetype,
  } as Partial<ImageEntity>;
}

async uploadOne(
  file: any,
  filePath: string,
  options: BlockBlobParallelUploadOptions = STORAGE_DEFAULT_UPLOAD_OPTIONS,
): Promise<BlobUploadCommonResponse> {
  const containerClient = this.client.getContainerClient(
    this.blobContainerName,
  );
  const blockBlobClient = containerClient.getBlockBlobClient(filePath);

  return blockBlobClient.uploadData(file.buffer, options).catch((err) => {
    console.log(err);
    throw new BadRequestException(AppErrorMessagesEnum.IMAGE_UPLOAD_ERROR);
  });
}

async deleteOne(filePath: string): Promise<BlobDeleteResponse> {
  const containerClient = this.client.getContainerClient(
    this.blobContainerName,
  );
  const blockBlobClient = containerClient.getBlockBlobClient(filePath);

  return blockBlobClient.delete().catch(() => {
    throw new BadRequestException(AppErrorMessagesEnum.IMAGE_DELETION_ERROR);
```

```
    });
  }
}
```

access-token.strategy.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
  FindManyOptions,
  FindOptionsWhere,
  FindOneOptions,
  Repository,
} from 'typeorm';
import { BadRequestException, NotFoundException } from '@nestjs/common';
import { TopicEntity } from './topic.entity';
import { AppErrorMessagesEnum } from 'src/common/enums';

@Injectable()
export class TopicsService {
  constructor(
    @InjectRepository(TopicEntity)
    private readonly entityRepository: Repository<TopicEntity>,
  ) {}

  findAll(
    options: FindManyOptions<TopicEntity> = { loadEagerRelations: true },
  ): Promise<TopicEntity[]> {
    return this.entityRepository.find(options).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.TOPICS_NOT_FOUND);
    });
  }

  findOne(
    conditions: FindOptionsWhere<TopicEntity>,
    options: FindOneOptions<TopicEntity> = { loadEagerRelations: true },
  ): Promise<TopicEntity> {
    return this.entityRepository
      .findOneOrFail({
        ...options,
        where: conditions,
      })
      .catch(() => {
        throw new NotFoundException(AppErrorMessagesEnum.TOPIC_NOT_FOUND);
      });
  }

  async createOne(entity: Partial<TopicEntity>): Promise<TopicEntity> {
    const entityToCreate = this.entityRepository.create(entity as TopicEntity);
    const { id } = await this.entityRepository
      .save(entityToCreate)
      .catch(() => {
        throw new BadRequestException(
          AppErrorMessagesEnum.INVALID_REQUEST_DATA,
        );
      });
    return this.findOne({ id } as FindOptionsWhere<TopicEntity>);
  }

  async updateOne(
    conditions: FindOptionsWhere<TopicEntity>,
    entity: Partial<TopicEntity>,
  ): Promise<TopicEntity> {
    const entityToUpdate = await this.findOne(conditions);
```

```
    const updatedEntity = this.entityRepository.merge(
      entityToUpdate,
      entity as TopicEntity,
    );
    const { id } = await this.entityRepository.save(updatedEntity).catch(() => {
      throw new BadRequestException(AppErrorMessagesEnum.INVALID_REQUEST_DATA);
    });
    return this.findOne({ id } as FindOptionsWhere<TopicEntity>);
  }

  async deleteOne(
    conditions: FindOptionsWhere<TopicEntity>,
  ): Promise<TopicEntity> {
    const entity = await this.findOne(conditions, {
      loadEagerRelations: false,
    });

    return this.entityRepository.remove(entity).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.TOPIC_NOT_FOUND);
    });
  }
}
```

refresh-token.strategy.ts

```
import { PassportStrategy } from '@nestjs/passport';
import { ExtractJwt, Strategy } from 'passport-jwt';
import { Request } from 'express';
import { Injectable } from '@nestjs/common';
import { AppConfigService } from 'src/config/app-config.service';
import { RefreshJwtPayload } from '../types';
import { UsersService } from 'src/modules/users/users.service';
import { RefreshTokensService } from 'src/modules/refresh-tokens/refresh-tokens.service';

@Injectable()
export class RefreshTokenStrategy extends PassportStrategy(
  Strategy,
  'jwt-refresh',
) {
  constructor(
    private readonly appConfigService: AppConfigService,
    private readonly usersService: UsersService,
    private readonly refreshTokensService: RefreshTokensService,
  ) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      secretOrKey: appConfigService.get<string>('JWT_REFRESH_SECRET'),
      passReqToCallback: true,
    });
  }

  async validate(
    req: Request,
    { id, role, name, surname, refreshTokenId }: RefreshJwtPayload,
  ) {
    const userData = await this.usersService.findOne(
      { id, role, name, surname },
      {
        select: { id: true, role: true, name: true, surname: true },
        loadEagerRelations: false,
      },
    );
    const tokenData = await this.refreshTokensService.findOne(
      {
```

```
        id: refreshTokenId,
        user: { id: userData.id },
      },
      {
        select: { id: true },
        loadEagerRelations: false,
      },
    );

    console.log(tokenData);
    return { ...userData, refreshTokenId: tokenData.id };
  }
}
```

access-token.guard.ts

```
import { Injectable } from '@nestjs/common';
import { AuthGuard } from '@nestjs/passport';

@Injectable()
export class AccessTokenGuard extends AuthGuard('jwt') {}
```

roles.guard.ts

```
import { Injectable, CanActivate, ExecutionContext } from '@nestjs/common';
import { Reflector } from '@nestjs/core';
import { UserRoleEnum } from 'src/common/enums';

@Injectable()
export class RolesGuard implements CanActivate {
  constructor(private reflector: Reflector) {}

  canActivate(context: ExecutionContext): boolean {
    const requiredRoles = this.reflector.get<UserRoleEnum[]>(
      'roles',
      context.getHandler(),
    );
    if (!requiredRoles) {
      return true;
    }
    const { user } = context.switchToHttp().getRequest();

    return requiredRoles.includes(user.role);
  }
}
```

auth.service.ts

```
import * as bcrypt from 'bcrypt';
import { v4 as uuidv4 } from 'uuid';
import {
  ConflictException,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { SignUpDto, SingInDto } from './dto';
import { UsersService } from '../users/users.service';
import { RefreshTokensService } from '../refresh-tokens/refresh-tokens.service';
import { AuthTokens, AccessJwtPayload, RefreshJwtPayload } from './types';
import { AppConfigService } from 'src/config/app-config.service';
import { UserEntity } from '../users/user.entity';
import { FindOptionsWhere } from 'typeorm';
import { RefreshTokenEntity } from '../refresh-tokens/refresh-token.entity';
```

```
import { AppErrorMessagesEnum } from 'src/common/enums';

@Injectable()
export class AuthService {
  private readonly jwtAccessSecret: string;
  private readonly jwtAccessExpiresIn: string;
  private readonly jwtRefreshSecret: string;
  private readonly jwtRefreshExpiresIn: string;

  constructor(
    private readonly usersService: UsersService,
    private readonly refreshTokensService: RefreshTokensService,
    private readonly jwtService: JwtService,
    private readonly appConfigService: AppConfigService,
  ) {
    this.jwtAccessSecret =
      this.appConfigService.get<string>('JWT_ACCESS_SECRET');
    this.jwtAccessExpiresIn = this.appConfigService.get<string>(
      'JWT_ACCESS_EXPIRES_IN',
    );
    this.jwtRefreshSecret =
      this.appConfigService.get<string>('JWT_REFRESH_SECRET');
    this.jwtRefreshExpiresIn = this.appConfigService.get<string>(
      'JWT_REFRESH_EXPIRES_IN',
    );
  }

  async signUp(signUpDto: SignUpDto): Promise<AuthTokens> {
    const userExists = await this.usersService
      .findOne({ email: signUpDto.email })
      .catch(() => null);

    if (userExists) {
      throw new ConflictException(AppErrorMessagesEnum.USER_ALREADY_EXISTS);
    }

    const user = await this.usersService.createOne(signUpDto);

    const tokens = await this.createUserTokens(user);

    return tokens;
  }

  async signIn(signInDto: SingInDto): Promise<AuthTokens> {
    const user = await this.usersService
      .findOne({ email: signInDto.email })
      .catch(() => {
        throw new UnauthorizedException(AppErrorMessagesEnum.UNAUTHORIZED);
      });

    const passwordMatches = await bcrypt.compare(
      signInDto.password,
      user.password,
    );
    if (!passwordMatches)
      throw new UnauthorizedException(AppErrorMessagesEnum.UNAUTHORIZED);

    const tokens = await this.createUserTokens(user);

    await this.refreshTokensService.deleteExceededRefreshTokens({
      user: { id: user.id },
    });

    return tokens;
```

```
}

async signOut(refreshTokenId: FindOptionsWhere<RefreshTokenEntity>) {
  return this.refreshTokensService.deleteOne(refreshTokenId);
}

async refreshTokens(
  condition: FindOptionsWhere<UserEntity>,
  refreshTokenId: FindOptionsWhere<RefreshTokenEntity>,
): Promise<AuthTokens> {
  const user = await this.usersService.findOne(condition);

  await this.refreshTokensService.deleteOne(refreshTokenId);

  const tokens = await this.createUserTokens(user);

  return tokens;
}

async generateTokens(
  user: UserEntity,
  refreshTokenId: string,
): Promise<AuthTokens> {
  const accessTokenPayload: AccessJwtPayload = {
    id: user.id,
    name: user.name,
    surname: user.surname,
    role: user.role,
  };

  const refreshTokenPayload: RefreshJwtPayload = {
    ...accessTokenPayload,
    refreshTokenId,
  };

  const [accessToken, refreshToken] = await Promise.all([
    this.jwtService.signAsync(accessTokenPayload, {
      secret: this.jwtAccessSecret,
      expiresIn: this.jwtAccessExpiresIn,
    }),
    this.jwtService.signAsync(refreshTokenPayload, {
      secret: this.jwtRefreshSecret,
      expiresIn: this.jwtRefreshExpiresIn,
    }),
  ]);

  const tokens: AuthTokens = {
    accessToken,
    refreshToken,
  };

  return tokens;
}

async createUserTokens(user: UserEntity): Promise<AuthTokens> {
  const refreshTokenId = uuidv4();
  const tokens = await this.generateTokens(user, refreshTokenId);
  await this.refreshTokensService.createRefreshToken(
    { id: user.id },
    { value: tokens.refreshToken, id: refreshTokenId },
  );

  return tokens;
}
```

```
  async findUser(
    conditions: FindOptionsWhere<UserEntity>,
  ): Promise<UserEntity> {
    return this.usersService.findOne(conditions);
  }
}
```

auth.controller.ts

```
import * as bcrypt from 'bcrypt';
import { v4 as uuidv4 } from 'uuid';
import {
  ConflictException,
  Injectable,
  UnauthorizedException,
} from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import { SignUpDto, SingInDto } from './dto';
import { UsersService } from '../users/users.service';
import { RefreshTokensService } from '../refresh-tokens/refresh-tokens.service';
import { AuthTokens, AccessJwtPayload, RefreshJwtPayload } from './types';
import { AppConfigService } from 'src/config/app-config.service';
import { UserEntity } from '../users/user.entity';
import { FindOptionsWhere } from 'typeorm';
import { RefreshTokenEntity } from '../refresh-tokens/refresh-token.entity';
import { AppErrorMessagesEnum } from 'src/common/enums';

@Injectable()
export class AuthService {
  private readonly jwtAccessSecret: string;
  private readonly jwtAccessExpiresIn: string;
  private readonly jwtRefreshSecret: string;
  private readonly jwtRefreshExpiresIn: string;

  constructor(
    private readonly usersService: UsersService,
    private readonly refreshTokensService: RefreshTokensService,
    private readonly jwtService: JwtService,
    private readonly appConfigService: AppConfigService,
  ) {
    this.jwtAccessSecret =
      this.appConfigService.get<string>('JWT_ACCESS_SECRET');
    this.jwtAccessExpiresIn = this.appConfigService.get<string>(
      'JWT_ACCESS_EXPIRES_IN',
    );
    this.jwtRefreshSecret =
      this.appConfigService.get<string>('JWT_REFRESH_SECRET');
    this.jwtRefreshExpiresIn = this.appConfigService.get<string>(
      'JWT_REFRESH_EXPIRES_IN',
    );
  }

  async signUp(signUpDto: SignUpDto): Promise<AuthTokens> {
    const userExists = await this.usersService
      .findOne({ email: signUpDto.email })
      .catch(() => null);

    if (userExists) {
      throw new ConflictException(AppErrorMessagesEnum.USER_ALREADY_EXISTS);
    }

    const user = await this.usersService.createOne(signUpDto);
```

```
  const tokens = await this.createUserTokens(user);

  return tokens;
}

async signIn(signInDto: SingInDto): Promise<AuthTokens> {
  const user = await this.usersService
    .findOne({ email: signInDto.email })
    .catch(() => {
      throw new UnauthorizedException(AppErrorMessagesEnum.UNAUTHORIZED);
    });

  const passwordMatches = await bcrypt.compare(
    signInDto.password,
    user.password,
  );
  if (!passwordMatches)
    throw new UnauthorizedException(AppErrorMessagesEnum.UNAUTHORIZED);

  const tokens = await this.createUserTokens(user);

  await this.refreshTokensService.deleteExceededRefreshTokens({
    user: { id: user.id },
  });

  return tokens;
}

async signOut(refreshTokenId: FindOptionsWhere<RefreshTokenEntity>) {
  return this.refreshTokensService.deleteOne(refreshTokenId);
}

async refreshTokens(
  condition: FindOptionsWhere<UserEntity>,
  refreshTokenId: FindOptionsWhere<RefreshTokenEntity>,
): Promise<AuthTokens> {
  const user = await this.usersService.findOne(condition);

  await this.refreshTokensService.deleteOne(refreshTokenId);

  const tokens = await this.createUserTokens(user);

  return tokens;
}

async generateTokens(
  user: UserEntity,
  refreshTokenId: string,
): Promise<AuthTokens> {
  const accessTokenPayload: AccessJwtPayload = {
    id: user.id,
    name: user.name,
    surname: user.surname,
    role: user.role,
  };

  const refreshTokenPayload: RefreshJwtPayload = {
    ...accessTokenPayload,
    refreshTokenId,
  };

  const [accessToken, refreshToken] = await Promise.all([
    this.jwtService.signAsync(accessTokenPayload, {
      secret: this.jwtAccessSecret,
```

```
        expiresIn: this.jwtAccessExpiresIn,
      }),
      this.jwtService.signAsync(refreshTokenPayload, {
        secret: this.jwtRefreshSecret,
        expiresIn: this.jwtRefreshExpiresIn,
      }),
    ]);

    const tokens: AuthTokens = {
      accessToken,
      refreshToken,
    };

    return tokens;
  }

  async createUserTokens(user: UserEntity): Promise<AuthTokens> {
    const refreshTokenId = uuidv4();
    const tokens = await this.generateTokens(user, refreshTokenId);
    await this.refreshTokensService.createRefreshToken(
      { id: user.id },
      { value: tokens.refreshToken, id: refreshTokenId },
    );

    return tokens;
  }

  async findUser(
    conditions: FindOptionsWhere<UserEntity>,
  ): Promise<UserEntity> {
    return this.usersService.findOne(conditions);
  }
}
```

comment.entity.ts

```
import { ApiProperty } from '@nestjs/swagger';
import {
  BaseEntity,
  Entity,
  Column,
  PrimaryGeneratedColumn,
  CreateDateColumn,
  UpdateDateColumn,
  JoinColumn,
  ManyToOne,
  ManyToMany,
} from 'typeorm';
import { UserEntity } from '../users/user.entity';
import { InitiativeEntity } from '../initiatives/initiative.entity';
import { ImageEntity } from '../images/image.entity';
import { LikeEntity } from '../likes/like.entity';

@Entity({ name: 'comments' })
export class CommentEntity extends BaseEntity {
  @ApiProperty({ type: 'string', format: 'uuid' })
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @ApiProperty({ type: 'string' })
  @Column({ length: 512 })
  text: string;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
```

```
  @CreateDateColumn({ readonly: true })
  readonly createdAt: Date;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @UpdateDateColumn({ readonly: true })
  readonly updatedAt: Date;

  @ApiProperty({ type: () => LikeEntity, isArray: true })
  @ManyToMany(() => LikeEntity, ({ comments }) => comments, {
    cascade: true,
  })
  likes: LikeEntity[];

  @ApiProperty({ type: () => ImageEntity, isArray: true })
  @ManyToMany(() => ImageEntity, ({ comments }) => comments, {
    eager: true,
  })
  images: ImageEntity[];

  @ApiProperty({ type: () => InitiativeEntity })
  @ManyToOne(() => InitiativeEntity, ({ comments }) => comments)
  @JoinColumn()
  initiative: InitiativeEntity;

  @ApiProperty({ type: () => UserEntity })
  @ManyToOne(() => UserEntity, ({ comments }) => comments, {
    eager: true,
  })
  @JoinColumn()
  user: UserEntity;
}
```

comments.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
  FindManyOptions,
  FindOptionsWhere,
  FindOneOptions,
  Repository,
} from 'typeorm';
import { BadRequestException, NotFoundException } from '@nestjs/common';
import { CommentEntity } from './comment.entity';
import { CreateCommentDto } from './dto';
import { AppErrorMessagesEnum } from 'src/common/enums';
import { ImageEntity } from '../images/image.entity';
import { ImagesService } from '../images/images.service';

@Injectable()
export class CommentsService {
  constructor(
    private readonly imagesService: ImagesService,
    @InjectRepository(CommentEntity)
    private readonly entityRepository: Repository<CommentEntity>,
  ) {}

  findAll(
    options: FindManyOptions<CommentEntity> = { loadEagerRelations: true },
  ): Promise<CommentEntity[]> {
    return this.entityRepository.find(options).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.COMMENTS_NOT_FOUND);
    });
  }
```

```
findOne(
  conditions: FindOptionsWhere<CommentEntity>,
  options: FindOneOptions<CommentEntity> = { loadEagerRelations: true },
): Promise<CommentEntity> {
  return this.entityRepository
    .findOneOrFail({
      ...options,
      where: conditions,
    })
    .catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.COMMENT_NOT_FOUND);
    });
}

getInitiativeComments(
  conditions: FindOptionsWhere<CommentEntity>,
  options: FindManyOptions<CommentEntity> = { loadEagerRelations: true },
): Promise<CommentEntity[]> {
  return this.entityRepository
    .find({
      ...options,
      where: conditions,
    })
    .catch(() => {
      return [];
    });
}

async createOne(entity: CreateCommentDto): Promise<CommentEntity> {
  const { userId, initiativeId, imageId } = entity;

  const comment = this.entityRepository.create({
    ...entity,
    user: { id: userId },
    initiative: { id: initiativeId },
  });

  try {
    const createdComment = await this.entityRepository.save(comment);
    if (imageId) {
      await this.addImage({ id: imageId }, createdComment);
    }

    return this.findOne({ id: createdComment.id });
  } catch (error) {
    throw new BadRequestException(AppErrorMessagesEnum.INVALID_REQUEST_DATA);
  }
}

async updateOne(
  conditions: FindOptionsWhere<CommentEntity>,
  entity: Partial<CommentEntity>,
): Promise<CommentEntity> {
  const entityToUpdate = await this.findOne(conditions);
  const updatedEntity = this.entityRepository.merge(
    entityToUpdate,
    entity as CommentEntity,
  );
  const { id } = await this.entityRepository.save(updatedEntity).catch(() => {
    throw new BadRequestException(AppErrorMessagesEnum.INVALID_REQUEST_DATA);
  });

  return this.findOne({ id } as FindOptionsWhere<CommentEntity>);
```

```
  }

  async deleteOne(
    conditions: FindOptionsWhere<CommentEntity>,
  ): Promise<CommentEntity> {
    const entity = await this.findOne(conditions, {
      loadEagerRelations: true,
    });

    if (!entity) {
      throw new NotFoundException(AppErrorMessagesEnum.COMMENT_NOT_FOUND);
    }

    console.log(entity);

    if (entity.images.length) {
      const deleteImagePromises = entity.images.map((image) =>
        this.imagesService.deleteOne({ id: image.id }),
      );

      await Promise.all(deleteImagePromises);
    }

    return this.entityRepository.remove(entity).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.COMMENT_NOT_FOUND);
    });
  }

  async addImage(
    conditions: FindOptionsWhere<ImageEntity>,
    comment: CommentEntity,
  ): Promise<ImageEntity> {
    return this.imagesService.linkImageToComment(conditions, comment);
  }
}
```

location.entity.ts

```
import { ApiProperty } from '@nestjs/swagger';
import {
  BaseEntity,
  Entity,
  Column,
  PrimaryGeneratedColumn,
  CreateDateColumn,
  UpdateDateColumn,
  OneToOne,
  ManyToOne,
  JoinColumn,
} from 'typeorm';
import { InitiativeEntity } from '../initiatives/initiative.entity';
import { RegionEntity } from '../regions/region.entity';

@Entity({ name: 'locations' })
export class LocationEntity extends BaseEntity {
  @ApiProperty({ type: 'string', format: 'uuid' })
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @ApiProperty({ type: 'number' })
  @Column({ type: 'double precision', nullable: false })
  latitude: number;

  @ApiProperty({ type: 'number' })
```

```
  @Column({ type: 'double precision', nullable: false })
  longitude: number;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @CreateDateColumn({ readonly: true })
  readonly createdAt: Date;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @UpdateDateColumn({ readonly: true })
  readonly updatedAt: Date;

  @ApiProperty({ type: () => InitiativeEntity })
  @OneToOne(() => InitiativeEntity)
  initiative: Partial<InitiativeEntity>;

  @ApiProperty({ type: () => RegionEntity })
  @ManyToOne(() => RegionEntity, ({ locations }) => locations, {
    eager: true,
  })
  @JoinColumn()
  region: RegionEntity;
}
```

locations.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
  FindManyOptions,
  FindOptionsWhere,
  FindOneOptions,
  Repository,
} from 'typeorm';
import { BadRequestException, NotFoundException } from '@nestjs/common';
import { LocationEntity } from './location.entity';
import { AppErrorMessagesEnum } from 'src/common/enums';
import { CreateLocationDto } from './dto';
import { RegionsService } from '../regions/regions.service';

@Injectable()
export class LocationsService {
  constructor(
    private readonly regionsService: RegionsService,
    @InjectRepository(LocationEntity)
    private readonly entityRepository: Repository<LocationEntity>,
  ) {}

  findAll(
    options: FindManyOptions<LocationEntity> = { loadEagerRelations: true },
  ): Promise<LocationEntity[]> {
    return this.entityRepository.find(options).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.LOCATIONS_NOT_FOUND);
    });
  }

  findOne(
    conditions: FindOptionsWhere<LocationEntity>,
    options: FindOneOptions<LocationEntity> = { loadEagerRelations: true },
  ): Promise<LocationEntity> {
    return this.entityRepository
      .findOneOrFail({
        ...options,
        where: conditions,
      })
```

```
      .catch(() => {
        throw new NotFoundException(AppErrorMessagesEnum.LOCATION_NOT_FOUND);
      });
  }

  async createOne(entity: CreateLocationDto): Promise<LocationEntity> {
    const region = await this.regionsService.findOne({
      geocode: entity.geocode,
    });

    const entityToCreate = this.entityRepository.create({
      ...entity,
      region: { id: region.id },
    });

    const { id } = await this.entityRepository
      .save(entityToCreate)
      .catch(() => {
        throw new BadRequestException(
          AppErrorMessagesEnum.INVALID_REQUEST_DATA,
        );
      });
    return this.findOne({ id } as FindOptionsWhere<LocationEntity>);
  }

  async deleteOne(
    conditions: FindOptionsWhere<LocationEntity>,
  ): Promise<LocationEntity> {
    const entity = await this.findOne(conditions, {
      loadEagerRelations: false,
    });

    return this.entityRepository.remove(entity).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.LOCATION_NOT_FOUND);
    });
  }
}
```

image.entity.ts

```
import { ConfigService } from '@nestjs/config';
import { ApiProperty } from '@nestjs/swagger';
import { Expose } from 'class-transformer';
import * as path from 'path';
import { AppConfigService } from 'src/config/app-config.service';
import {
  PrimaryGeneratedColumn,
  CreateDateColumn,
  UpdateDateColumn,
  BaseEntity,
  Column,
  Entity,
  ManyToMany,
  JoinTable,
} from 'typeorm';
import { InitiativeEntity } from '../initiatives/initiative.entity';
import { CommentEntity } from '../comments/comment.entity';

const appConfigService = new AppConfigService(new ConfigService());

@Entity('images')
export class ImageEntity extends BaseEntity {
  @ApiProperty()
  @PrimaryGeneratedColumn('uuid')
```

```
  id: string;

  @ApiProperty({ type: 'string', maxLength: 256, required: true })
  @Column({ type: 'varchar', length: 256 })
  fileName: string;

  @ApiProperty({ type: 'string', maxLength: 256, required: true })
  @Column({ type: 'varchar', length: 256 })
  fileExt: string;

  @ApiProperty({ type: 'string', maxLength: 512, required: true })
  @Column({ type: 'varchar', length: 512 })
  fileNameWithExt: string;

  @ApiProperty({ type: 'string', maxLength: 256, required: true })
  @Column({ type: 'varchar', length: 256 })
  mimetype: string;

  @ApiProperty({ type: 'string', maxLength: 512, required: true })
  @Column({ type: 'varchar', length: 512 })
  filePath: string;

  @Expose()
  @ApiProperty({ readOnly: true })
  get src(): string {
    if (!this.filePath) return null;
    const filePath = path
      .join(appConfigService.get('CDN'), this.filePath)
      .replace(/\\/g, '/');

    return new URL(filePath).toString();
  }

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @CreateDateColumn({ readonly: true })
  readonly createdAt: Date;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @UpdateDateColumn({ readonly: true })
  readonly updatedAt: Date;

  @ApiProperty({ type: () => InitiativeEntity, isArray: true })
  @ManyToMany(() => InitiativeEntity, ({ images }) => images)
  @JoinTable({ name: 'initiatives-images' })
  initiatives: InitiativeEntity[];

  @ApiProperty({ type: () => CommentEntity, isArray: true })
  @ManyToMany(() => CommentEntity, ({ images }) => images)
  @JoinTable({ name: 'comments-images' })
  comments: CommentEntity[];
}
```

images.service.ts

```
import {
  BadRequestException,
  Injectable,
  NotFoundException,
} from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
  FindManyOptions,
  FindOptionsWhere,
  FindOneOptions,
```

```
  Repository,
} from 'typeorm';
import { ImageEntity } from './image.entity';
import { StorageService } from 'src/systems/storage/storage.service';
import { STORAGE_ROOT_DIRECTORY } from 'src/systems/storage/storage.constants';
import { InitiativeEntity } from '../initiatives/initiative.entity';
import { AppErrorMessagesEnum } from 'src/common/enums';
import { imagesMimetypes } from 'src/common/constants';
import { CommentEntity } from '../comments/comment.entity';

@Injectable()
export class ImagesService {
  constructor(
    @InjectRepository(ImageEntity)
    private readonly imageEntityRepository: Repository<ImageEntity>,
    private readonly storageService: StorageService,
  ) {}

  async createOne(
    file: Express.Multer.File,
    data?: Partial<ImageEntity>,
    pathToFile = STORAGE_ROOT_DIRECTORY,
  ) {
    if (!imagesMimetypes.includes(file.mimetype)) {
      throw new BadRequestException(AppErrorMessagesEnum.IMAGE_UNACCEPTED_TYPE);
    }

    console.log(file);

    const uploadedFileEntity = await this.storageService.createOne(
      file,
      pathToFile,
    );
    const entityLike = this.imageEntityRepository.create(uploadedFileEntity);
    const entityToCreate = this.imageEntityRepository.merge(entityLike, data);
    const { id } = await this.imageEntityRepository
      .save(entityToCreate)
      .catch(() => {
        throw new BadRequestException(
          AppErrorMessagesEnum.INVALID_REQUEST_DATA,
        );
      });
    return this.findOne({ id });
  }

  async linkImageToInitiative(
    conditions: FindOptionsWhere<ImageEntity>,
    initiativeEntity: InitiativeEntity,
  ): Promise<ImageEntity> {
    const entityToUpdate = await this.findOne(conditions);
    entityToUpdate.initiatives = [initiativeEntity];

    const { id } = await this.imageEntityRepository
      .save(entityToUpdate)
      .catch(() => {
        throw new BadRequestException(
          AppErrorMessagesEnum.INVALID_REQUEST_DATA,
        );
      });
    return this.findOne({ id } as FindOptionsWhere<ImageEntity>);
  }

  async linkImageToComment(
    conditions: FindOptionsWhere<ImageEntity>,
```

```
    commentEntity: CommentEntity,
  ): Promise<ImageEntity> {
    const entityToUpdate = await this.findOne(conditions);
    entityToUpdate.comments = [commentEntity];

    const { id } = await this.imageEntityRepository
      .save(entityToUpdate)
      .catch(() => {
        throw new BadRequestException(
          AppErrorMessagesEnum.INVALID_REQUEST_DATA,
        );
      });
    return this.findOne({ id } as FindOptionsWhere<ImageEntity>);
  }

  findOne(
    conditions: FindOptionsWhere<ImageEntity>,
    options: FindOneOptions<ImageEntity> = { loadEagerRelations: true },
  ): Promise<ImageEntity> {
    return this.imageEntityRepository
      .findOneOrFail({
        ...options,
        where: conditions,
      })
      .catch(() => {
        throw new NotFoundException(AppErrorMessagesEnum.IMAGE_NOT_FOUND);
      });
  }

  findAll(
    options: FindManyOptions<ImageEntity> = { loadEagerRelations: true },
  ): Promise<ImageEntity[]> {
    return this.imageEntityRepository.find(options).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.IMAGES_NOT_FOUND);
    });
  }

  getInitiativeImages(
    conditions: FindOptionsWhere<ImageEntity>,
    options: FindManyOptions<ImageEntity> = { loadEagerRelations: true },
  ): Promise<ImageEntity[]> {
    return this.imageEntityRepository
      .find({
        ...options,
        where: conditions,
      })
      .catch(() => {
        return [];
      });
  }

  async deleteOne(
    conditions: FindOptionsWhere<ImageEntity>,
  ): Promise<ImageEntity> {
    const entity = await this.findOne(conditions);
    await this.storageService.deleteOne(entity.filePath);
    return this.imageEntityRepository.remove(entity).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.IMAGE_NOT_FOUND);
    });
  }
}
```

initiative.entity.ts

```
import { ApiHideProperty, ApiProperty } from '@nestjs/swagger';
import {
  BaseEntity,
  Entity,
  Column,
  PrimaryGeneratedColumn,
  CreateDateColumn,
  UpdateDateColumn,
  JoinColumn,
  OneToOne,
  ManyToOne,
  ManyToMany,
  OneToMany,
} from 'typeorm';
import { LocationEntity } from '../locations/location.entity';
import { TopicEntity } from '../topics/topic.entity';
import { CommentEntity } from '../comments/comment.entity';
import { UserEntity } from '../users/user.entity';
import { ImageEntity } from '../images/image.entity';
import { InitiativeStatusEnum } from 'src/common/enums';
import { LikeEntity } from '../likes/like.entity';

@Entity({ name: 'initiatives' })
export class InitiativeEntity extends BaseEntity {
  @ApiProperty({ type: 'string', format: 'uuid' })
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @ApiProperty({ type: 'string' })
  @Column({ length: 128 })
  title: string;

  @ApiProperty({ type: 'string' })
  @Column({ length: 512 })
  description: string;

  @ApiProperty({
    enum: InitiativeStatusEnum,
    default: InitiativeStatusEnum.CREATED,
  })
  @Column({
    enum: InitiativeStatusEnum,
    default: InitiativeStatusEnum.CREATED,
    nullable: false,
  })
  status: InitiativeStatusEnum;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @CreateDateColumn({ readonly: true })
  readonly createdAt: Date;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @UpdateDateColumn({ readonly: true })
  readonly updatedAt: Date;

  @ApiProperty({ type: () => LocationEntity })
  @OneToOne(() => LocationEntity, {
    eager: true,
    onDelete: 'CASCADE',
  })
  @JoinColumn()
  location: Partial<LocationEntity>;
```

```
  @ApiProperty({ type: () => TopicEntity })
  @ManyToOne(() => TopicEntity, ({ initiatives }) => initiatives, {
    eager: true,
  })
  @JoinColumn()
  topic: TopicEntity;

  @ApiProperty({ type: () => UserEntity })
  @ManyToOne(() => UserEntity, ({ initiatives }) => initiatives, {
    eager: true,
  })
  @JoinColumn()
  user: UserEntity;

  @ApiHideProperty()
  @OneToMany(() => CommentEntity, ({ initiative }) => initiative)
  comments: CommentEntity[];

  @ApiProperty({ type: () => ImageEntity, isArray: true })
  @ManyToMany(() => ImageEntity, ({ initiatives }) => initiatives)
  images: ImageEntity[];

  @ApiProperty({ type: () => LikeEntity, isArray: true })
  @ManyToMany(() => LikeEntity, ({ initiatives }) => initiatives, {
    cascade: true,
  })
  likes: LikeEntity[];
}
```

initiatives.service.ts

```
import {
  BadRequestException,
  Injectable,
  NotFoundException,
} from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
  FindManyOptions,
  FindOptionsWhere,
  FindOneOptions,
  Repository,
} from 'typeorm';
import { ImageEntity } from './image.entity';
import { StorageService } from 'src/systems/storage/storage.service';
import { STORAGE_ROOT_DIRECTORY } from 'src/systems/storage/storage.constants';
import { InitiativeEntity } from '../initiatives/initiative.entity';
import { AppErrorMessagesEnum } from 'src/common/enums';
import { imagesMimetypes } from 'src/common/constants';
import { CommentEntity } from '../comments/comment.entity';

@Injectable()
export class ImagesService {
  constructor(
    @InjectRepository(ImageEntity)
    private readonly imageEntityRepository: Repository<ImageEntity>,
    private readonly storageService: StorageService,
  ) {}

  async createOne(
    file: Express.Multer.File,
    data?: Partial<ImageEntity>,
    pathToFile = STORAGE_ROOT_DIRECTORY,
  ) {
```

```
  if (!imagesMimetypes.includes(file.mimetype)) {
    throw new BadRequestException(AppErrorMessagesEnum.IMAGE_UNACCEPTED_TYPE);
  }

  console.log(file);

  const uploadedFileEntity = await this.storageService.createOne(
    file,
    pathToFile,
  );
  const entityLike = this.imageEntityRepository.create(uploadedFileEntity);
  const entityToCreate = this.imageEntityRepository.merge(entityLike, data);
  const { id } = await this.imageEntityRepository
    .save(entityToCreate)
    .catch(() => {
      throw new BadRequestException(
        AppErrorMessagesEnum.INVALID_REQUEST_DATA,
      );
    });
  return this.findOne({ id });
}

async linkImageToInitiative(
  conditions: FindOptionsWhere<ImageEntity>,
  initiativeEntity: InitiativeEntity,
): Promise<ImageEntity> {
  const entityToUpdate = await this.findOne(conditions);
  entityToUpdate.initiatives = [initiativeEntity];

  const { id } = await this.imageEntityRepository
    .save(entityToUpdate)
    .catch(() => {
      throw new BadRequestException(
        AppErrorMessagesEnum.INVALID_REQUEST_DATA,
      );
    });
  return this.findOne({ id } as FindOptionsWhere<ImageEntity>);
}

async linkImageToComment(
  conditions: FindOptionsWhere<ImageEntity>,
  commentEntity: CommentEntity,
): Promise<ImageEntity> {
  const entityToUpdate = await this.findOne(conditions);
  entityToUpdate.comments = [commentEntity];

  const { id } = await this.imageEntityRepository
    .save(entityToUpdate)
    .catch(() => {
      throw new BadRequestException(
        AppErrorMessagesEnum.INVALID_REQUEST_DATA,
      );
    });
  return this.findOne({ id } as FindOptionsWhere<ImageEntity>);
}

findOne(
  conditions: FindOptionsWhere<ImageEntity>,
  options: FindOneOptions<ImageEntity> = { loadEagerRelations: true },
): Promise<ImageEntity> {
  return this.imageEntityRepository
    .findOneOrFail({
      ...options,
      where: conditions,
```

```
      })
      .catch(() => {
        throw new NotFoundException(AppErrorMessagesEnum.IMAGE_NOT_FOUND);
      });
  }

  findAll(
    options: FindManyOptions<ImageEntity> = { loadEagerRelations: true },
  ): Promise<ImageEntity[]> {
    return this.imageEntityRepository.find(options).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.IMAGES_NOT_FOUND);
    });
  }

  getInitiativeImages(
    conditions: FindOptionsWhere<ImageEntity>,
    options: FindManyOptions<ImageEntity> = { loadEagerRelations: true },
  ): Promise<ImageEntity[]> {
    return this.imageEntityRepository
      .find({
        ...options,
        where: conditions,
      })
      .catch(() => {
        return [];
      });
  }

  async deleteOne(
    conditions: FindOptionsWhere<ImageEntity>,
  ): Promise<ImageEntity> {
    const entity = await this.findOne(conditions);
    await this.storageService.deleteOne(entity.filePath);
    return this.imageEntityRepository.remove(entity).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.IMAGE_NOT_FOUND);
    });
  }
}
```

like.entity.ts

```
import { ApiProperty } from '@nestjs/swagger';
import {
  BaseEntity,
  Entity,
  PrimaryGeneratedColumn,
  JoinColumn,
  ManyToOne,
  ManyToMany,
  JoinTable,
  CreateDateColumn,
  UpdateDateColumn,
} from 'typeorm';
import { UserEntity } from '../users/user.entity';
import { CommentEntity } from '../comments/comment.entity';
import { InitiativeEntity } from '../initiatives/initiative.entity';

@Entity({ name: 'likes' })
export class LikeEntity extends BaseEntity {
  @ApiProperty({ type: 'string', format: 'uuid' })
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
```

```
  @CreateDateColumn({ readonly: true })
  readonly createdAt: Date;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @UpdateDateColumn({ readonly: true })
  readonly updatedAt: Date;

  @ApiProperty({ type: () => UserEntity })
  @ManyToOne(() => UserEntity, ({ likes }) => likes)
  @JoinColumn()
  user: UserEntity;

  @ApiProperty({ type: () => InitiativeEntity, isArray: true })
  @ManyToMany(() => InitiativeEntity, ({ likes }) => likes)
  @JoinTable({ name: 'initiatives-likes' })
  initiatives: InitiativeEntity[];

  @ApiProperty({ type: () => CommentEntity, isArray: true })
  @ManyToMany(() => CommentEntity, ({ likes }) => likes)
  @JoinTable({ name: 'comments-likes' })
  comments: CommentEntity[];
}
```

likes.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { FindOptionsWhere, FindOneOptions, Repository } from 'typeorm';
import { BadRequestException, NotFoundException } from '@nestjs/common';
import { LikeEntity } from './like.entity';
import { CreateCommentLikeDto, CreateInitiativeLikeDto } from './dto';
import { IdParameterDto } from 'src/common/dto';
import { AppErrorMessagesEnum } from 'src/common/enums';

@Injectable()
export class LikesService {
  constructor(
    @InjectRepository(LikeEntity)
    private readonly likeRepository: Repository<LikeEntity>,
  ) {}

  findCommentLike(
    conditions: FindOptionsWhere<LikeEntity>,
    options: FindOneOptions<LikeEntity> = { loadEagerRelations: true },
  ): Promise<LikeEntity> {
    return this.likeRepository
      .findOneOrFail({
        ...options,
        where: conditions,
      })
      .catch(() => {
        throw new NotFoundException(
          AppErrorMessagesEnum.COMMENT_LIKE_NOT_FOUND,
        );
      });
  }

  async createCommentLike(entity: CreateCommentLikeDto): Promise<LikeEntity> {
    const { userId, commentId } = entity;

    const likeInDb = await this.likeRepository.findOne({
      where: {
        user: { id: userId },
        comments: { id: commentId },
```

```
    },
    relations: ['user', 'comments'],
  });

  if (!likeInDb) {
    const like = this.likeRepository.create({
      user: { id: userId },
      comments: [{ id: commentId }],
    });

    try {
      return await this.likeRepository.save(like);
    } catch (error) {
      throw new BadRequestException(
        AppErrorMessagesEnum.INVALID_REQUEST_DATA,
      );
    }
  } else {
    throw new BadRequestException(AppErrorMessagesEnum.INVALID_REQUEST_DATA);
  }
}

async deleteCommentLike(
  conditions: FindOptionsWhere<LikeEntity>,
): Promise<LikeEntity> {
  const entity = await this.findCommentLike(conditions, {
    loadEagerRelations: false,
  });

  return this.likeRepository.remove(entity).catch(() => {
    throw new NotFoundException(AppErrorMessagesEnum.COMMENT_LIKE_NOT_FOUND);
  });
}

async countLikesByCommentId(IdParameterDto: IdParameterDto): Promise<number> {
  try {
    const count = await this.likeRepository.count({
      where: {
        comments: { id: IdParameterDto.id },
      },
    });

    return count;
  } catch (error) {
    throw new BadRequestException(AppErrorMessagesEnum.INVALID_REQUEST_DATA);
  }
}

findInitiativeLike(
  conditions: FindOptionsWhere<LikeEntity>,
  options: FindOneOptions<LikeEntity> = { loadEagerRelations: true },
): Promise<LikeEntity> {
  return this.likeRepository
    .findOneOrFail({
      ...options,
      where: conditions,
    })
    .catch(() => {
      throw new NotFoundException(
        AppErrorMessagesEnum.INITIATIVE_LIKE_NOT_FOUND,
      );
    });
}
```

```
  async createInitiativeLike(
    entity: CreateInitiativeLikeDto,
  ): Promise<LikeEntity> {
    const { userId, initiativeId } = entity;

    const likeInDb = await this.likeRepository.findOne({
      where: {
        user: { id: userId },
        initiatives: { id: initiativeId },
      },
      relations: ['user', 'initiatives'],
    });

    if (!likeInDb) {
      const like = this.likeRepository.create({
        user: { id: userId },
        initiatives: [{ id: initiativeId }],
      });

      try {
        return await this.likeRepository.save(like);
      } catch (error) {
        throw new BadRequestException(
          AppErrorMessagesEnum.INVALID_REQUEST_DATA,
        );
      }
    } else {
      throw new BadRequestException(AppErrorMessagesEnum.INVALID_REQUEST_DATA);
    }
  }

  async deleteInitiativeLike(
    conditions: FindOptionsWhere<LikeEntity>,
  ): Promise<LikeEntity> {
    const entity = await this.findInitiativeLike(conditions, {
      loadEagerRelations: false,
    });

    return this.likeRepository.remove(entity).catch(() => {
      throw new NotFoundException(
        AppErrorMessagesEnum.INITIATIVE_LIKE_NOT_FOUND,
      );
    });
  }

  async countLikesByInitiativeId(id: string): Promise<number> {
    try {
      const count = await this.likeRepository.count({
        where: {
          initiatives: { id },
        },
      });
      return count;
    } catch (error) {
      throw new BadRequestException(AppErrorMessagesEnum.INVALID_REQUEST_DATA);
    }
  }
}
```

topic.entity.ts

```
import { ApiHideProperty, ApiProperty } from '@nestjs/swagger';
import {
  BaseEntity,
```

```
  Entity,
  Column,
  PrimaryGeneratedColumn,
  CreateDateColumn,
  UpdateDateColumn,
  OneToMany,
} from 'typeorm';
import { InitiativeEntity } from '../initiatives/initiative.entity';

@Entity({ name: 'topics' })
export class TopicEntity extends BaseEntity {
  @ApiProperty({ type: 'string', format: 'uuid' })
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @ApiProperty({ type: 'string' })
  @Column({ length: 64, unique: true, nullable: false })
  name: string;

  @ApiProperty({ type: 'string' })
  @Column({ length: 512 })
  description: string;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @CreateDateColumn({ readonly: true })
  readonly createdAt: Date;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @UpdateDateColumn({ readonly: true })
  readonly updatedAt: Date;

  @ApiHideProperty()
  @OneToMany(() => InitiativeEntity, ({ topic }) => topic, {
    nullable: true,
  })
  initiatives: InitiativeEntity[];
}
```

topics.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
  FindManyOptions,
  FindOptionsWhere,
  FindOneOptions,
  Repository,
} from 'typeorm';
import { BadRequestException, NotFoundException } from '@nestjs/common';
import { TopicEntity } from './topic.entity';
import { AppErrorMessagesEnum } from 'src/common/enums';

@Injectable()
export class TopicsService {
  constructor(
    @InjectRepository(TopicEntity)
    private readonly entityRepository: Repository<TopicEntity>,
  ) {}

  findAll(
    options: FindManyOptions<TopicEntity> = { loadEagerRelations: true },
  ): Promise<TopicEntity[]> {
    return this.entityRepository.find(options).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.TOPICS_NOT_FOUND);
```

```
    });
  }

  findOne(
    conditions: FindOptionsWhere<TopicEntity>,
    options: FindOneOptions<TopicEntity> = { loadEagerRelations: true },
  ): Promise<TopicEntity> {
    return this.entityRepository
      .findOneOrFail({
        ...options,
        where: conditions,
      })
      .catch(() => {
        throw new NotFoundException(AppErrorMessagesEnum.TOPIC_NOT_FOUND);
      });
  }

  async createOne(entity: Partial<TopicEntity>): Promise<TopicEntity> {
    const entityToCreate = this.entityRepository.create(entity as TopicEntity);
    const { id } = await this.entityRepository
      .save(entityToCreate)
      .catch(() => {
        throw new BadRequestException(
          AppErrorMessagesEnum.INVALID_REQUEST_DATA,
        );
      });
    return this.findOne({ id } as FindOptionsWhere<TopicEntity>);
  }

  async updateOne(
    conditions: FindOptionsWhere<TopicEntity>,
    entity: Partial<TopicEntity>,
  ): Promise<TopicEntity> {
    const entityToUpdate = await this.findOne(conditions);
    const updatedEntity = this.entityRepository.merge(
      entityToUpdate,
      entity as TopicEntity,
    );
    const { id } = await this.entityRepository.save(updatedEntity).catch(() => {
      throw new BadRequestException(AppErrorMessagesEnum.INVALID_REQUEST_DATA);
    });
    return this.findOne({ id } as FindOptionsWhere<TopicEntity>);
  }

  async deleteOne(
    conditions: FindOptionsWhere<TopicEntity>,
  ): Promise<TopicEntity> {
    const entity = await this.findOne(conditions, {
      loadEagerRelations: false,
    });

    return this.entityRepository.remove(entity).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.TOPIC_NOT_FOUND);
    });
  }
}
```

region.entity.ts

```
import { ApiHideProperty, ApiProperty } from '@nestjs/swagger';
import {
  BaseEntity,
  Entity,
  Column,
```

```
  PrimaryGeneratedColumn,
  CreateDateColumn,
  UpdateDateColumn,
  OneToMany,
} from 'typeorm';
import { LocationEntity } from '../locations/location.entity';

@Entity({ name: 'regions' })
export class RegionEntity extends BaseEntity {
  @ApiProperty({ type: 'string', format: 'uuid' })
  @PrimaryGeneratedColumn('uuid')
  id: string;

  @ApiProperty({ type: 'string' })
  @Column({ length: 64, unique: true, nullable: false })
  name: string;

  @ApiProperty({ type: 'string', uniqueItems: true })
  @Column({ length: 5, unique: true, nullable: false })
  geocode: string;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @CreateDateColumn({ readonly: true })
  readonly createdAt: Date;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @UpdateDateColumn({ readonly: true })
  readonly updatedAt: Date;

  @ApiHideProperty()
  @OneToMany(() => LocationEntity, ({ region }) => region, {
    nullable: true,
  })
  locations: LocationEntity[];
}
```

regions.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
  FindManyOptions,
  FindOptionsWhere,
  FindOneOptions,
  Repository,
} from 'typeorm';
import { BadRequestException, NotFoundException } from '@nestjs/common';
import { RegionEntity } from './region.entity';
import { AppErrorMessagesEnum } from 'src/common/enums';

@Injectable()
export class RegionsService {
  constructor(
    @InjectRepository(RegionEntity)
    private readonly entityRepository: Repository<RegionEntity>,
  ) {}

  findAll(
    options: FindManyOptions<RegionEntity> = { loadEagerRelations: true },
  ): Promise<RegionEntity[]> {
    return this.entityRepository.find(options).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.REGIONS_NOT_FOUND);
    });
  }
```

```
  findOne(
    conditions: FindOptionsWhere<RegionEntity>,
    options: FindOneOptions<RegionEntity> = { loadEagerRelations: true },
  ): Promise<RegionEntity> {
    return this.entityRepository
      .findOneOrFail({
        ...options,
        where: conditions,
      })
      .catch(() => {
        throw new NotFoundException(AppErrorMessagesEnum.REGION_NOT_FOUND);
      });
  }

  async createOne(entity: Partial<RegionEntity>): Promise<RegionEntity> {
    const entityToCreate = this.entityRepository.create(entity as RegionEntity);
    const { id } = await this.entityRepository
      .save(entityToCreate)
      .catch(() => {
        throw new BadRequestException(
          AppErrorMessagesEnum.INVALID_REQUEST_DATA,
        );
      });
    return this.findOne({ id } as FindOptionsWhere<RegionEntity>);
  }

  async deleteOne(
    conditions: FindOptionsWhere<RegionEntity>,
  ): Promise<RegionEntity> {
    const entity = await this.findOne(conditions, {
      loadEagerRelations: false,
    });

    return this.entityRepository.remove(entity).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.REGION_NOT_FOUND);
    });
  }
}
```

refresh-token.entity.ts

```
import { ApiProperty } from '@nestjs/swagger';
import {
  BaseEntity,
  Entity,
  Column,
  ManyToOne,
  JoinColumn,
  UpdateDateColumn,
  CreateDateColumn,
  BeforeInsert,
  PrimaryColumn,
} from 'typeorm';
import { UserEntity } from '../users/user.entity';
import * as bcrypt from 'bcrypt';
import { SALT_ROUNDS } from 'src/common/constants';

@Entity({ name: 'refresh-tokens' })
export class RefreshTokenEntity extends BaseEntity {
  @ApiProperty({ type: 'string', format: 'uuid' })
  @PrimaryColumn('uuid')
  id: string;
```

```
  @ApiProperty({ type: 'string', maxLength: 1024, uniqueItems: true })
  @Column({ length: 1024, unique: true })
  value: string;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @Column({ readonly: true })
  expiresAt: Date;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @CreateDateColumn({ readonly: true })
  readonly createdAt: Date;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @UpdateDateColumn({ readonly: true })
  readonly updatedAt: Date;

  @BeforeInsert()
  async hashRefreshToken() {
    if (this.value) {
      this.value = await bcrypt.hash(this.value, SALT_ROUNDS);
    }
  }

  @ApiProperty({ type: () => UserEntity })
  @ManyToOne(() => UserEntity, {
    onDelete: 'CASCADE',
  })
  @JoinColumn()
  user: UserEntity;
}
```

refresh-tokens.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
  FindManyOptions,
  FindOptionsWhere,
  FindOneOptions,
  Repository,
} from 'typeorm';
import { BadRequestException, NotFoundException } from '@nestjs/common';
import { RefreshTokenEntity } from './refresh-token.entity';
import * as ms from 'ms';
import { AppConfigService } from 'src/config/app-config.service';
import { UsersService } from '../users/users.service';
import { UserEntity } from '../users/user.entity';
import { AppErrorMessagesEnum } from 'src/common/enums';
import { MAX_USER_SESSIONS_QUANTITY } from 'src/common/constants';

@Injectable()
export class RefreshTokensService {
  constructor(
    @InjectRepository(RefreshTokenEntity)
    private readonly entityRepository: Repository<RefreshTokenEntity>,
    private readonly appConfigService: AppConfigService,
    private readonly usersService: UsersService,
  ) {}

  findAll(
    options: FindManyOptions<RefreshTokenEntity> = { loadEagerRelations: true },
  ): Promise<RefreshTokenEntity[]> {
    return this.entityRepository.find(options).catch(() => {
      throw new NotFoundException(
```

```
      AppErrorMessagesEnum.REFRESH_TOKENS_NOT_FOUND,
    );
  });
}

findOne(
  conditions: FindOptionsWhere<RefreshTokenEntity>,
  options: FindOneOptions<RefreshTokenEntity> = { loadEagerRelations: true },
): Promise<RefreshTokenEntity> {
  return this.entityRepository
    .findOneOrFail({
      ...options,
      where: conditions,
    })
    .catch(() => {
      throw new NotFoundException(
        AppErrorMessagesEnum.REFRESH_TOKEN_NOT_FOUND,
      );
    });
}

async createOne(
  entity: Partial<RefreshTokenEntity>,
): Promise<RefreshTokenEntity> {
  const entityToCreate = this.entityRepository.create(
    entity as RefreshTokenEntity,
  );
  const { id } = await this.entityRepository
    .save(entityToCreate)
    .catch(() => {
      throw new BadRequestException(
        AppErrorMessagesEnum.INVALID_REQUEST_DATA,
      );
    });
  return this.findOne({ id } as FindOptionsWhere<RefreshTokenEntity>);
}

async updateOne(
  conditions: FindOptionsWhere<RefreshTokenEntity>,
  entity: Partial<RefreshTokenEntity>,
): Promise<RefreshTokenEntity> {
  const entityToUpdate = await this.findOne(conditions);
  const updatedEntity = this.entityRepository.merge(
    entityToUpdate,
    entity as RefreshTokenEntity,
  );
  const { id } = await this.entityRepository.save(updatedEntity).catch(() => {
    throw new BadRequestException(AppErrorMessagesEnum.INVALID_REQUEST_DATA);
  });
  return this.findOne({ id } as FindOptionsWhere<RefreshTokenEntity>);
}

async deleteOne(
  conditions: FindOptionsWhere<RefreshTokenEntity>,
): Promise<RefreshTokenEntity> {
  const entity = await this.findOne(conditions, {
    loadEagerRelations: false,
  });

  return this.entityRepository.remove(entity).catch(() => {
    throw new NotFoundException(AppErrorMessagesEnum.REFRESH_TOKEN_NOT_FOUND);
  });
}
```

```
  async createRefreshToken(
    condition: FindOptionsWhere<UserEntity>,
    tokenData: Partial<RefreshTokenEntity>,
  ): Promise<RefreshTokenEntity> {
    const user = await this.usersService.findOne(condition);

    const tokenLifetime = this.appConfigService.get('JWT_REFRESH_EXPIRES_IN');
    const tokenDuration = ms(tokenLifetime);
    const currentDateTime = new Date();
    const expiresAt = new Date(currentDateTime.getTime() + tokenDuration);

    const refreshTokenModel: Partial<RefreshTokenEntity> = {
      user,
      ...tokenData,
      expiresAt,
    };

    return this.createOne(refreshTokenModel);
  }

  async deleteExceededRefreshTokens(
    condition: FindOptionsWhere<RefreshTokenEntity>,
  ) {
    const exceededTokens = await this.entityRepository.find({
      where: condition,
      order: {
        createdAt: 'DESC',
      },
      skip: MAX_USER_SESSIONS_QUANTITY,
    });

    const tokenIdsToDelete = exceededTokens.map((token) => token.id);
    if (tokenIdsToDelete.length) {
      await this.entityRepository.delete(tokenIdsToDelete);
    }
  }
}
```

user.entity.ts

```
import { ApiProperty, ApiHideProperty } from '@nestjs/swagger';
import {
  BaseEntity,
  Entity,
  Column,
  PrimaryGeneratedColumn,
  CreateDateColumn,
  UpdateDateColumn,
  BeforeInsert,
  BeforeUpdate,
  OneToMany,
} from 'typeorm';
import { Exclude } from 'class-transformer';
import * as bcrypt from 'bcrypt';
import { UserRoleEnum } from 'src/common/enums';
import { SALT_ROUNDS } from 'src/common/constants';
import { CommentEntity } from '../comments/comment.entity';
import { InitiativeEntity } from '../initiatives/initiative.entity';
import { LikeEntity } from '../likes/like.entity';

@Entity({ name: 'users' })
export class UserEntity extends BaseEntity {
  @ApiProperty({ type: 'string', format: 'uuid' })
  @PrimaryGeneratedColumn('uuid')
```

```
  id: string;

  @ApiProperty({ type: 'string', maxLength: 32 })
  @Column({ length: 32, nullable: false })
  name: string;

  @ApiProperty({ type: 'string', maxLength: 32 })
  @Column({ length: 32, nullable: false })
  surname: string;

  @ApiHideProperty()
  @Exclude()
  @Column({ length: 64, nullable: false })
  password: string;

  @ApiProperty({ type: 'string', maxLength: 256, uniqueItems: true })
  @Column({ length: 256, unique: true, nullable: false })
  email: string;

  @ApiProperty({ enum: UserRoleEnum, default: UserRoleEnum.USER })
  @Column({ enum: UserRoleEnum, default: UserRoleEnum.USER, nullable: false })
  role: UserRoleEnum;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @CreateDateColumn({ readonly: true })
  readonly createdAt: Date;

  @ApiProperty({ type: 'string', readOnly: true, format: 'date-time' })
  @UpdateDateColumn({ readonly: true })
  readonly updatedAt: Date;

  @BeforeInsert()
  @BeforeUpdate()
  async hashPassword() {
    if (this.password) {
      this.password = await bcrypt.hash(this.password, SALT_ROUNDS);
    }
  }

  @ApiHideProperty()
  @OneToMany(() => CommentEntity, ({ user }) => user, {
    nullable: true,
  })
  comments: CommentEntity[];

  @ApiHideProperty()
  @OneToMany(() => InitiativeEntity, ({ user }) => user, {
    nullable: true,
  })
  initiatives: InitiativeEntity[];

  @ApiHideProperty()
  @OneToMany(() => LikeEntity, ({ user }) => user, {
    nullable: true,
  })
  likes: LikeEntity;
}
```

users.service.ts

```
import { Injectable } from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import {
  FindManyOptions,
```

```
  FindOptionsWhere,
  FindOneOptions,
  Repository,
} from 'typeorm';
import { BadRequestException, NotFoundException } from '@nestjs/common';
import { UserEntity } from './user.entity';
import { AppErrorMessagesEnum } from 'src/common/enums';

@Injectable()
export class UsersService {
  constructor(
    @InjectRepository(UserEntity)
    private readonly entityRepository: Repository<UserEntity>,
  ) {}

  findAll(
    options: FindManyOptions<UserEntity> = { loadEagerRelations: true },
  ): Promise<UserEntity[]> {
    return this.entityRepository.find(options).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.USERS_NOT_FOUND);
    });
  }

  findOne(
    conditions: FindOptionsWhere<UserEntity>,
    options: FindOneOptions<UserEntity> = { loadEagerRelations: true },
  ): Promise<UserEntity> {
    return this.entityRepository
      .findOneOrFail({
        ...options,
        where: conditions,
      })
      .catch(() => {
        throw new NotFoundException(AppErrorMessagesEnum.USER_NOT_FOUND);
      });
  }

  async createOne(entity: Partial<UserEntity>): Promise<UserEntity> {
    const entityToCreate = this.entityRepository.create(entity as UserEntity);
    const { id } = await this.entityRepository
      .save(entityToCreate)
      .catch(() => {
        throw new BadRequestException(
          AppErrorMessagesEnum.INVALID_REQUEST_DATA,
        );
      });

    return this.findOne({ id } as FindOptionsWhere<UserEntity>);
  }

  async updateOne(
    conditions: FindOptionsWhere<UserEntity>,
    entity: Partial<UserEntity>,
  ): Promise<UserEntity> {
    const entityToUpdate = await this.findOne(conditions);
    const updatedEntity = this.entityRepository.merge(
      entityToUpdate,
      entity as UserEntity,
    );
    const { id } = await this.entityRepository.save(updatedEntity).catch(() => {
      throw new BadRequestException(AppErrorMessagesEnum.INVALID_REQUEST_DATA);
    });
    return this.findOne({ id } as FindOptionsWhere<UserEntity>);
  }
```

```
  async deleteOne(
    conditions: FindOptionsWhere<UserEntity>,
  ): Promise<UserEntity> {
    const entity = await this.findOne(conditions, {
      loadEagerRelations: false,
    });

    return this.entityRepository.remove(entity).catch(() => {
      throw new NotFoundException(AppErrorMessagesEnum.USER_NOT_FOUND);
    });
  }
}
```

create-user.dto.ts

```
import { ApiProperty } from '@nestjs/swagger';
import {
  IsEmail,
  IsNotEmpty,
  IsString,
  MaxLength,
  MinLength,
} from 'class-validator';

export class CreateUserDto {
  @IsNotEmpty()
  @IsString()
  @MinLength(1)
  @MaxLength(32)
  @ApiProperty({
    example: 'name',
    required: true,
    nullable: false,
    minLength: 1,
    maxLength: 32,
  })
  public readonly name: string;

  @IsNotEmpty()
  @IsString()
  @MinLength(1)
  @MaxLength(32)
  @ApiProperty({
    example: 'surname',
    required: true,
    nullable: false,
    minLength: 1,
    maxLength: 32,
  })
  public readonly surname: string;

  @IsNotEmpty()
  @IsString()
  @MinLength(8)
  @MaxLength(64)
  @ApiProperty({
    example: 'password1234',
    required: true,
    nullable: false,
    minLength: 8,
    maxLength: 64,
  })
  public readonly password: string;
```

```
  @IsNotEmpty()
  @IsString()
  @IsEmail()
  @MinLength(8)
  @MaxLength(256)
  @ApiProperty({
    example: 'example@mail.com',
    required: true,
    nullable: false,
    minLength: 8,
    maxLength: 256,
  })
  public readonly email: string;
}
```

users.controller.ts

```
import {
  Body,
  ClassSerializerInterceptor,
  Controller,
  Delete,
  Get,
  Param,
  Patch,
  Post,
  UseGuards,
  UseInterceptors,
} from '@nestjs/common';
import { UsersService } from './users.service';
import { UserEntity } from './user.entity';
import { IdParameterDto } from 'src/common/dto';
import { CreateUserDto, UpdateUserDto } from './dto';
import { ApiBearerAuth, ApiTags } from '@nestjs/swagger';
import { AccessTokenGuard, RolesGuard } from '../auth/guards';
import { HasRoles } from '../auth/decorators';
import { UserRoleEnum } from 'src/common/enums';

@ApiTags('users')
@Controller('users')
@ApiBearerAuth()
@UseGuards(AccessTokenGuard, RolesGuard)
@UseInterceptors(ClassSerializerInterceptor)
export class UsersController {
  constructor(private readonly usersService: UsersService) {}
  @HasRoles(UserRoleEnum.ADMIN)
  @Get()
  findAll(): Promise<UserEntity[]> {
    return this.usersService.findAll();
  }

  @HasRoles(UserRoleEnum.ADMIN)
  @Get(':id')
  findOne(@Param() conditions: IdParameterDto): Promise<UserEntity> {
    return this.usersService.findOne(conditions);
  }

  @HasRoles(UserRoleEnum.ADMIN)
  @Post()
  createOne(@Body() createUserDto: CreateUserDto): Promise<UserEntity> {
    return this.usersService.createOne(createUserDto);
  }
```

```
  @HasRoles(UserRoleEnum.ADMIN)
  @Patch(':id')
  updateOne(
    @Param() conditions: IdParameterDto,
    @Body() updateUserDto: UpdateUserDto,
  ): Promise<UserEntity> {
    return this.usersService.updateOne(conditions, updateUserDto);
  }

  @HasRoles(UserRoleEnum.ADMIN)
  @Delete(':id')
  deleteOne(@Param() conditions: IdParameterDto): Promise<UserEntity> {
    return this.usersService.deleteOne(conditions);
  }
}
```

HttpClient.js

```
class HttpClient {
  static async authorizedFetch(url, options = {}) {
    try {
      options.headers = options.headers || {};
      options.headers.Authorization = `Bearer ${localStorage.getItem(
        "accessToken"
      )}`;

      let response = await fetch(url, options);
      if (response.status === 401) {
        const tokenRefreshed = await HttpClient._refreshToken();

        if (tokenRefreshed) {
          options.headers.Authorization = `Bearer ${localStorage.getItem(
            "accessToken"
          )}`;

          response = await fetch(url, options);
        } else {
          throw new Error("Failed to refresh tokens");
        }
      }

      return await response.json();
    } catch (error) {
      throw error;
    }
  }

  static async unauthorizedFetch(url, options = {}) {
    try {
      let response = await fetch(url, options);

      if (!response.ok) {
        throw new Error(response.error);
      }

      return await response.json();
    } catch (error) {
      throw error;
    }
  }

  static async _refreshToken() {
    try {
      const refreshToken = localStorage.getItem("refreshToken");
```

```
      if (!refreshToken) {
        return false;
      }

      const response = await fetch(
        "http://localhost:8080/api/v1/auth/tokens/refresh",
        {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
            Authorization: `Bearer ${refreshToken}`,
          },
        }
      );

      if (!response.ok) {
        return false;
      }

      const tokens = await response.json();
      localStorage.setItem("accessToken", tokens.accessToken);
      localStorage.setItem("refreshToken", tokens.refreshToken);

      return true;
    } catch (error) {
      throw error;
    }
  }
}

export default HttpClient;
```

App.js

```
import React from "react";
import {
  BrowserRouter as Router,
  Routes,
  Route,
  Navigate,
} from "react-router-dom";
import RegistrationComponent from "./components/auth/RegistrationComponent";
import LoginComponent from "./components/auth/LoginComponent";
import AppComponent from "./components/app/AppComponent";

const App = () => {
  return (
    <Router>
      <Routes>
        <Route path="/login" element={<LoginComponent />} />
        <Route path="/registration" element={<RegistrationComponent />} />
        <Route path="/app" element={<AppComponent />} />
        <Route path="/" element={<Navigate to="/login" replace />} />
      </Routes>
    </Router>
  );
};

export default App;
```

RegistrationComponent.ts

```
import React from "react";
import { useFormik } from "formik";
```

```
import * as Yup from "yup";
import Button from "@mui/material/Button";
import TextField from "@mui/material/TextField";
import Typography from "@mui/material/Typography";
import Link from "@mui/material/Link";
import { useNavigate } from "react-router-dom";
import HttpClient from "../../HttpClient";

const RegistrationComponent = () => {
  const navigate = useNavigate();

  const validationSchema = Yup.object({
    name: Yup.string()
      .min(1, "Мінімальна довжина 1 символ")
      .max(32, "Максимальна довжина 32 символи")
      .required("Ім'я є обов'язковим"),
    surname: Yup.string()
      .min(1, "Мінімальна довжина 1 символ")
      .max(32, "Максимальна довжина 32 символи")
      .required("Прізвище є обов'язковим"),
    email: Yup.string()
      .email("Невірний формат електронної пошти")
      .min(8, "Мінімальна довжина 8 символів")
      .max(256, "Максимальна довжина 256 символів")
      .required("Електронна адреса є обов'язковою"),
    password: Yup.string()
      .min(8, "Мінімальна довжина 8 символів")
      .max(64, "Максимальна довжина 64 символи")
      .required("Пароль є обов'язковим"),
  });

  const formik = useFormik({
    initialValues: {
      name: "",
      surname: "",
      email: "",
      password: "",
    },
    validationSchema: validationSchema,
    onSubmit: async (values, { setSubmitting, setErrors }) => {
      try {
        const data = await HttpClient.unauthorizedFetch(
          "http://localhost:8080/api/v1/auth/signup",
          {
            method: "POST",
            headers: {
              "Content-Type": "application/json",
            },
            body: JSON.stringify(values),
          }
        );

        localStorage.setItem("accessToken", data.accessToken);
        localStorage.setItem("refreshToken", data.refreshToken);
        await fetchUserProfile(data.accessToken);
        navigate("/app");
      } catch (error) {
        setErrors({ submit: "При реєстрації виникла помилка" });
      } finally {
        setSubmitting(false);
      }
    },
  });
```

```
const fetchUserProfile = async (accessToken) => {
  try {
    const data = await HttpClient.authorizedFetch(
      "http://localhost:8080/api/v1/auth/profile",
      {
        method: "GET",
      }
    );

    localStorage.setItem("user", JSON.stringify(data));
  } catch (error) {
    formik.setErrors({ submit: error.message });
  }
};

return (
  <div style={{ position: "relative" }}>
    <div
      style={{
        position: "fixed",
        top: 0,
        left: 0,
        width: "100%",
        height: "100%",
        backgroundImage: "url(/background.png)",
        backgroundSize: "cover",
        filter: "blur(6px)",
        zIndex: -1,
      }}
    />
    <div
      style={{
        maxWidth: "calc(100vw / 4)",
        margin: "auto",
        marginTop: "20vh",
        borderRadius: "12px",
        padding: "16px",
        boxShadow: "0 0 10px rgba(0, 0, 0, 0.1)",
        backgroundColor: "rgba(255, 255, 255, 0.9)",
        zIndex: 1,
      }}
    >
      <Typography
        variant="h5"
        style={{ marginBottom: "16px", marginLeft: "12px" }}
      >
        Реєстрація
      </Typography>
      <form onSubmit={formik.handleSubmit}>
        <TextField
          label="Ім'я"
          variant="outlined"
          fullWidth
          name="name"
          value={formik.values.name}
          onChange={formik.handleChange}
          onBlur={formik.handleBlur}
          error={formik.touched.name && Boolean(formik.errors.name)}
          helperText={formik.touched.name && formik.errors.name}
          style={{ marginBottom: "16px" }}
        />
        <TextField
          label="Прізвище"
          variant="outlined"
```

```
        fullWidth
        name="surname"
        value={formik.values.surname}
        onChange={formik.handleChange}
        onBlur={formik.handleBlur}
        error={formik.touched.surname && Boolean(formik.errors.surname)}
        helperText={formik.touched.surname && formik.errors.surname}
        style={{ marginBottom: "16px" }}
      />
      <TextField
        label="Електронна адреса"
        type="email"
        variant="outlined"
        fullWidth
        name="email"
        value={formik.values.email}
        onChange={formik.handleChange}
        onBlur={formik.handleBlur}
        error={formik.touched.email && Boolean(formik.errors.email)}
        helperText={formik.touched.email && formik.errors.email}
        style={{ marginBottom: "16px" }}
      />
      <TextField
        label="Пароль"
        type="password"
        variant="outlined"
        fullWidth
        name="password"
        value={formik.values.password}
        onChange={formik.handleChange}
        onBlur={formik.handleBlur}
        error={formik.touched.password && Boolean(formik.errors.password)}
        helperText={formik.touched.password && formik.errors.password}
        style={{ marginBottom: "16px" }}
      />
      <Button
        type="submit"
        variant="contained"
        color="primary"
        fullWidth
        style={{ marginBottom: "16px", height: "50px", fontSize: "1rem" }}
        disabled={formik.isSubmitting}
      >
        Зареєструватися
      </Button>
      {formik.errors.submit && (
        <Typography
          variant="body1"
          style={{ color: "red", marginTop: "16px" }}
        >
          {formik.errors.submit}
        </Typography>
      )}
    </form>
    <Typography
      variant="body1"
      style={{ textAlign: "center", marginTop: "16px" }}
    >
      Вже маєте аккаунт?{" "}
      <Link href="/login" underline="always">
        Вхід
      </Link>
    </Typography>
  </div>
```

```
    </div>
  );
};

export default RegistrationComponent;
```

AppComponent.ts

```
import React, { useState } from "react";
import MapComponent from "./MapComponent";
import CreateInitiativeComponent from "./CreateInitiativeComponent";
import ViewInitiativeComponent from "./ViewInitiativeComponent";
import LogoutComponent from "../auth/LogoutComponent";
import MenuComponent from "./MenuComponent";
import SearchInitiativeComponent from "./SearchInitiativeComponent";
import "../../App.css";
import StatisticsComponent from "./StatisticsComponent";

const AppComponent = () => {
  const [showCreateInitiative, setShowCreateInitiative] = useState(false);
  const [showViewInitiative, setShowViewInitiative] = useState(false);
  const [markerLocationInfo, setMarkerLocationInfo] = useState(null);
  const [locationId, setLocationId] = useState(null);
  const [menuOptions, setMenuOptions] = useState(null);
  const [searchParams, setSearchParams] = useState({});
  const [markerToAdd, setMarkerToAdd] = useState({ locationId: undefined });
  const [viewedInitiative, setViewedInitiative] = useState({
    locationId: undefined,
  });

  const handleMarkerAdd = (markerLocation) => {
    setMarkerLocationInfo(markerLocation);
    setShowViewInitiative(false);
    setShowCreateInitiative(true);
  };

  const handleMarkerClick = (markerLocationId) => {
    setLocationId(markerLocationId);
    setShowCreateInitiative(false);
    setShowViewInitiative(true);
  };

  const handleCreateInitiativeClose = (markerLocationId) => {
    setShowCreateInitiative(false);
    setLocationId(markerLocationId);
    setShowViewInitiative(true);
    setMarkerToAdd({ locationId: markerLocationId });
  };

  const handleViewInitiativeClose = () => {
    setViewedInitiative({ locationId });
    setShowViewInitiative(false);
  };

  const handleMenuOptionSelect = (option) => {
    setMenuOptions(option);
    if (option !== "search") {
      setSearchParams({});
    }
  };

  const handleSearch = (params) => {
    setSearchParams(params);
  };
```

```
  const handleMenuClose = () => {
    setSearchParams({});
    setMenuOptions("all");
  };

  return (
    <div
      style={{
        display: "flex",
        height: "100vh",
        overflow: "hidden",
        margin: "0",
      }}
    >
      <MapComponent
        onMarkerAdd={handleMarkerAdd}
        onMarkerClick={handleMarkerClick}
        menuOptions={menuOptions}
        searchParams={searchParams}
        markerToAdd={markerToAdd}
        viewedInitiative={viewedInitiative}
      />
      {showCreateInitiative && (
        <CreateInitiativeComponent
          markerLocationInfo={markerLocationInfo}
          onClose={handleCreateInitiativeClose}
        />
      )}
      {showViewInitiative && locationId && (
        <ViewInitiativeComponent
          locationId={locationId}
          onClose={handleViewInitiativeClose}
        />
      )}
      <LogoutComponent />
      <MenuComponent handleMenuOptionSelect={handleMenuOptionSelect} />
      {menuOptions === "search" && (
        <SearchInitiativeComponent
          onSearch={handleSearch}
          onClose={handleMenuClose}
        />
      )}
      {menuOptions === "statistics" && (
        <StatisticsComponent onClose={handleMenuClose} />
      )}
    </div>
  );
};

export default AppComponent;
```

CommentsComponent.ts

```
import React, { useState, useEffect } from "react";
import Button from "@mui/material/Button";
import Typography from "@mui/material/Typography";
import FavoriteIcon from "@mui/icons-material/Favorite";
import FavoriteBorderIcon from "@mui/icons-material/FavoriteBorder";
import IconButton from "@mui/material/IconButton";
import DeleteIcon from "@mui/icons-material/Delete";
import TextField from "@mui/material/TextField";
import Select from "@mui/material/Select";
import MenuItem from "@mui/material/MenuItem";
```

```
import FormControl from "@mui/material/FormControl";
import { useFormik } from "formik";
import * as Yup from "yup";
import HttpClient from "../../HttpClient";

const CommentsComponent = ({ initiativeId }) => {
  const [comments, setComments] = useState([]);
  const [sortBy, setSortBy] = useState("likes");
  const user = JSON.parse(localStorage.getItem("user"));
  const isAdmin = user.role === "Admin";

  useEffect(() => {
    fetchComments();
  }, [initiativeId]);

  const fetchComments = async () => {
    try {
      const commentsData = await HttpClient.authorizedFetch(
        `http://localhost:8080/api/v1/comments/initiative/${initiativeId}`
      );

      const userCommentsData = commentsData.filter(
        (comment) => comment.user.role === "User"
      );

      const likePromises = userCommentsData.map((comment) => {
        const likePromise = HttpClient.authorizedFetch(
          `http://localhost:8080/api/v1/likes/comments/count-likes/${comment.id}`
        )
          .then((data) => {
            comment.likesCount = +data;
          })
          .catch((error) => {
            console.error("Error fetching likes count:", error);
          });

        const userLikePromise = HttpClient.authorizedFetch(
          `http://localhost:8080/api/v1/likes/comments/user-
like?userId=${user.id}&commentId=${comment.id}`
        )
          .then((data) => {
            comment.userLike = data.id ? data : null;
          })
          .catch((error) => {
            console.error("Error fetching user like for comment:", error);
          });

        return Promise.all([likePromise, userLikePromise]);
      });

      await Promise.all(likePromises);

      sortComments(userCommentsData, sortBy);
      setComments(userCommentsData);
    } catch (error) {
      console.error("Error fetching comments:", error);
    }
  };

  const handleLikeButtonClick = (commentId, userLike) => {
    if (userLike) {
      HttpClient.authorizedFetch(
        `http://localhost:8080/api/v1/likes/comments/${userLike.id}`,
        {
```

```
        method: "DELETE",
      }
    )
      .then(() => {
        setComments((prevComments) =>
          prevComments.map((comment) =>
            comment.id === commentId
              ? {
                  ...comment,
                  likesCount: comment.likesCount - 1,
                  userLike: null,
                }
              : comment
          )
        );
      })
      .catch((error) => console.error("Error removing like:", error));
  } else {
    HttpClient.authorizedFetch(
      "http://localhost:8080/api/v1/likes/comments",
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({ userId: user.id, commentId }),
      }
    )
      .then((data) => {
        setComments((prevComments) =>
          prevComments.map((comment) =>
            comment.id === commentId
              ? {
                  ...comment,
                  likesCount: comment.likesCount + 1,
                  userLike: data,
                }
              : comment
          )
        );
      })
      .catch((error) => console.error("Error adding like:", error));
  }
};

const handleDeleteComment = async (commentId) => {
  try {
    await HttpClient.authorizedFetch(
      `http://localhost:8080/api/v1/comments/${commentId}`,
      {
        method: "DELETE",
      }
    );
    setComments((prevComments) =>
      prevComments.filter((comment) => comment.id !== commentId)
    );
  } catch (error) {
    console.error("Error deleting comment:", error);
  }
};

const handleSortChange = (event) => {
  setSortBy(event.target.value);
  setComments((prevComments) => {
```

```
    const commentsToSort = [...prevComments];
    sortComments(commentsToSort, event.target.value);
    return commentsToSort;
  });
};

const sortComments = (commentsData, sortBy) => {
  if (sortBy === "date") {
    commentsData.sort(
      (a, b) => new Date(b.createdAt) - new Date(a.createdAt)
    );
  } else if (sortBy === "likes") {
    commentsData.sort((a, b) => b.likesCount - a.likesCount);
  }
};

const formik = useFormik({
  initialValues: {
    newCommentText: "",
  },
  validationSchema: Yup.object({
    newCommentText: Yup.string()
      .trim()
      .min(1, "Коментар має містити хоча б один символ")
      .max(512, "Максимальна довжина коментаря - 512 символів")
      .required("Коментар не може бути порожнім"),
  }),
  onSubmit: (values, { setSubmitting, resetForm }) => {
    HttpClient.authorizedFetch("http://localhost:8080/api/v1/comments", {
      method: "POST",
      headers: {
        "Content-Type": "application/json",
      },
      body: JSON.stringify({
        userId: user.id,
        initiativeId,
        text: values.newCommentText,
      }),
    })
      .then((data) => {
        setComments((prevComments) => [
          ...prevComments,
          { likesCount: 0, userLike: null, ...data },
        ]);
        resetForm();
        sortComments();
      })
      .catch((error) => console.error("Error adding comment:", error))
      .finally(() => setSubmitting(false));
  },
});

return (
  <div style={{ marginTop: "16px" }}>
    {!isAdmin && (
      <div>
        <hr style={{ marginTop: "20px", marginBottom: "20px" }} />
        <form onSubmit={formik.handleSubmit}>
          <TextField
            label="Введіть текст коментаря"
            name="newCommentText"
            value={formik.values.newCommentText}
            onChange={formik.handleChange}
            onBlur={formik.handleBlur}
```

```
          multiline
          fullWidth
          variant="outlined"
          error={
            formik.touched.newCommentText &&
            Boolean(formik.errors.newCommentText)
          }
          helperText={
            formik.touched.newCommentText && formik.errors.newCommentText
          }
        />
        <div
          style={{
            display: "flex",
            marginTop: "8px",
          }}
        >
          <Button
            variant="text"
            color="primary"
            onClick={() => formik.resetForm()}
            style={{ marginRight: "8px" }}
            disabled={formik.isSubmitting}
          >
            Відміна
          </Button>
          <Button
            variant="contained"
            color="primary"
            type="submit"
            disabled={formik.isSubmitting}
          >
            Коментувати
          </Button>
        </div>
      </form>
    </div>
  )}
  <hr style={{ marginTop: "20px", marginBottom: "20px" }} />
  {comments.length > 0 ? (
    <div
      style={{
        marginTop: "16px",
        marginBottom: "16px",
        display: "flex",
        alignItems: "center",
      }}
    >
      <Typography
        variant="body1"
        style={{
          marginLeft: "10px",
          marginRight: "8px",
        }}
      >
        Сортувати за:
      </Typography>
      <FormControl style={{ height: "40px" }}>
        <Select
          value={sortBy}
          onChange={handleSortChange}
          variant="outlined"
          style={{ height: "100%" }}
        >
```

```
          <MenuItem value="date">Датою створення</MenuItem>
          <MenuItem value="likes">Популярністю</MenuItem>
        </Select>
      </FormControl>
    </div>
  ) : (
    <div style={{ marginTop: "20px" }}>
      ( Коментарі до цієї ініціативи поки відсутні )
    </div>
  )}
  <div style={{ marginTop: "16px" }}>
    {comments.map((comment) => (
      <div
        key={comment.id}
        style={{
          border: "1px solid #ccc",
          padding: "12px",
          borderRadius: "8px",
          marginBottom: "8px",
          display: "flex",
          flexDirection: "column",
        }}
      >
        <div
          style={{
            fontWeight: "600",
            fontStyle: "italic",
            marginBottom: "4px",
            display: "flex",
            alignItems: "center",
          }}
        >
          {comment.user.name} {comment.user.surname}
          {isAdmin && (
            <Button
              onClick={() => handleDeleteComment(comment.id)}
              style={{ marginLeft: "auto", color: "red" }}
            >
              <DeleteIcon />
            </Button>
          )}
        </div>
        <p>{comment.text}</p>
        <div style={{ display: "flex", alignItems: "center" }}>
          <IconButton
            onClick={() =>
              handleLikeButtonClick(comment.id, comment.userLike)
            }
            style={{ color: "#e1306c" }}
          >
            {comment.userLike ? (
              <FavoriteIcon style={{ fontSize: "24px" }} />
            ) : (
              <FavoriteBorderIcon style={{ fontSize: "24px" }} />
            )}
            <Typography
              variant="body2"
              style={{ marginLeft: "5px", fontSize: "16px" }}
            >
              {comment.likesCount}
            </Typography>
          </IconButton>

          <span style={{ marginLeft: "auto" }}>
```

```
                {new Date(comment.createdAt).toLocaleString("uk-UA", {
                  timeZone: "Europe/Kyiv",
                  day: "numeric",
                  month: "long",
                  year: "numeric",
                  hour: "numeric",
                  minute: "numeric",
                })}
              </span>
            </div>
          </div>
        ))}
      </div>
    </div>
  );
};

export default CommentsComponent;
```

CreateInitiativeComponent.ts

```
import React, { useState, useEffect } from "react";
import Grid from "@mui/material/Grid";
import Typography from "@mui/material/Typography";
import TextField from "@mui/material/TextField";
import Button from "@mui/material/Button";
import Select from "@mui/material/Select";
import MenuItem from "@mui/material/MenuItem";
import CreateIcon from "@mui/icons-material/Create";
import ImageIcon from "@mui/icons-material/Image";
import { useFormik } from "formik";
import * as Yup from "yup";
import HttpClient from "../../HttpClient";

const CreateInitiativeComponent = ({ markerLocationInfo, onClose }) => {
  const [topics, setTopics] = useState([]);
  const [image, setImage] = useState(null);
  const user = JSON.parse(localStorage.getItem("user"));

  useEffect(() => {
    fetchTopics();
  }, []);

  const fetchTopics = async () => {
    try {
      const data = await HttpClient.authorizedFetch(
        "http://localhost:8080/api/v1/topics"
      );
      setTopics(data);
    } catch (error) {
      console.error("Error fetching topics:", error);
    }
  };

  const handleImageChange = async (event) => {
    const imageFile = event.target.files[0];

    if (!imageFile) {
      return;
    }

    const formData = new FormData();
    formData.append("file", imageFile);
```

```
  try {
    const result = await HttpClient.authorizedFetch(
      "http://localhost:8080/api/v1/images",
      {
        method: "POST",
        body: formData,
      }
    );
    setImage(result);
    console.log("Created image ID:", result.id);
  } catch (error) {
    console.error("Error uploading image:", error);
  }
};

const formik = useFormik({
  initialValues: {
    title: "",
    description: "",
    selectedTopic: "",
  },
  validationSchema: Yup.object({
    title: Yup.string()
      .trim()
      .min(1, "Назва має містити хоча б один символ")
      .max(128, "Максимальна довжина назви - 128 символів")
      .required("Назва не може бути порожньою"),
    description: Yup.string()
      .trim()
      .min(1, "Опис має містити хоча б один символ")
      .max(512, "Максимальна довжина опису - 512 символів")
      .required("Опис не може бути порожнім"),
    selectedTopic: Yup.string().required("Тема має бути обов'язково обрана"),
  }),
  onSubmit: async (values) => {
    const data = {
      title: values.title,
      description: values.description,
      topicId: values.selectedTopic,
      userId: user.id,
      location: markerLocationInfo,
      imageId: image ? image.id : undefined,
    };
    console.log(data);

    try {
      const result = await HttpClient.authorizedFetch(
        "http://localhost:8080/api/v1/initiatives",
        {
          method: "POST",
          headers: {
            "Content-Type": "application/json",
          },
          body: JSON.stringify(data),
        }
      );
      console.log(result);
      alert("Ініціатива створена успішно!");
      onClose(result.location.id);
    } catch (error) {
      console.error("Error creating initiative:", error);
    }
  },
});
```

```
return (
  <div
    style={{
      position: "absolute",
      zIndex: 999,
      padding: "40px",
      top: "20px",
      right: "20px",
      width: "calc(100% / 3)",
      flex: "none",
      overflowY: "auto",
      maxHeight: "85vh",
      borderRadius: "10px",
      backgroundColor: "#ffffff",
      border: "1px solid #ccc",
      boxShadow: "0px 4px 8px rgba(0, 0, 0, 0.1)",
    }}
  >
    <Typography variant="h5">
      Створення ініціативи
      <CreateIcon
        style={{ verticalAlign: "bottom", marginLeft: "20px", color: "grey" }}
      />
    </Typography>

    <hr style={{ marginTop: "20px", marginBottom: "20px" }} />
    <form onSubmit={formik.handleSubmit}>
      <TextField
        label="Назва"
        variant="outlined"
        fullWidth
        margin="normal"
        name="title"
        value={formik.values.title}
        onChange={formik.handleChange}
        onBlur={formik.handleBlur}
        error={formik.touched.title && Boolean(formik.errors.title)}
        helperText={formik.touched.title && formik.errors.title}
      />
      <Grid container alignItems="center">
        <Grid item xs={4}>
          <Typography marginLeft="10px" variant="body1">
            Тема:
          </Typography>
        </Grid>
        <Grid item xs={8}>
          <Select
            name="selectedTopic"
            value={formik.values.selectedTopic}
            onChange={formik.handleChange}
            onBlur={formik.handleBlur}
            fullWidth
            variant="outlined"
            error={
              formik.touched.selectedTopic &&
              Boolean(formik.errors.selectedTopic)
            }
          >
            <MenuItem value="" disabled>
              ( Оберіть тему )
            </MenuItem>
            {topics.map((topic) => (
              <MenuItem key={topic.id} value={topic.id}>
```

```
            {topic.name}
          </MenuItem>
        ))}
      </Select>
      {formik.touched.selectedTopic && formik.errors.selectedTopic && (
        <Typography color="error" variant="body2">
          {formik.errors.selectedTopic}
        </Typography>
      )}
    </Grid>
  </Grid>
  <TextField
    label="Опис"
    variant="outlined"
    fullWidth
    multiline
    rows={6}
    margin="normal"
    name="description"
    value={formik.values.description}
    onChange={formik.handleChange}
    onBlur={formik.handleBlur}
    error={
      formik.touched.description && Boolean(formik.errors.description)
    }
    helperText={formik.touched.description && formik.errors.description}
  />
  <div
    style={{ display: "flex", alignItems: "center", marginTop: "16px" }}
  >
    <input
      type="file"
      accept="image/*"
      onChange={handleImageChange}
      style={{ display: "none" }}
      id="upload-button"
    />
    <label htmlFor="upload-button">
      <Button
        variant="outlined"
        component="span"
        startIcon={<ImageIcon />}
      >
        Додати зображення
      </Button>
    </label>
    {image && (
      <div
        style={{
          marginLeft: "16px",
          display: "flex",
          alignItems: "center",
        }}
      >
        <img
          src={image.src}
          alt={image.fileNameWithExt}
          style={{
            maxWidth: "50px",
            maxHeight: "50px",
            marginRight: "8px",
          }}
        />
        <Typography variant="body2">{image.fileNameWithExt}</Typography>
```

```
            </div>
          )}
        </div>
        <div style={{ textAlign: "right", marginTop: "16px" }}>
          <Button variant="text" color="primary" onClick={() => onClose(null)}>
            Відміна
          </Button>

          <span style={{ marginRight: "10px" }} />
          <Button variant="contained" color="primary" type="submit">
            Створити
          </Button>
        </div>
      </form>
    </div>
  );
};

export default CreateInitiativeComponent;
```

MapComponent.ts

```
import React, { useEffect, useState, useRef } from "react";
import "leaflet/dist/leaflet.css";
import L from "leaflet";
import SearchPlaceComponent from "./SearchPlaceComponent";
import { InitiativeStatuses } from "../../constants";
import HttpClient from "../../HttpClient";

const MapComponent = ({
  onMarkerAdd,
  onMarkerClick,
  menuOptions,
  searchParams,
  markerToAdd,
  viewedInitiative,
}) => {
  const mapRef = useRef(null);
  const activeMarkerRef = useRef(null);
  const [markers, setMarkers] = useState([]);
  const user = JSON.parse(localStorage.getItem("user"));
  const [currentAddedMarker, setCurrentAddedMarker] = useState(null);
  const isUser = user.role === "User";

  useEffect(() => {
    initializeMap();
  }, []);

  useEffect(() => {
    console.log("log");
    handleMenuOptionsChange();
  }, [menuOptions, searchParams]);

  useEffect(() => {
    handleCreateInitiativeClose();
  }, [markerToAdd]);

  useEffect(() => {
    handleInitiativeViewClose();
  }, [viewedInitiative]);

  const handleInitiativeViewClose = async () => {
    if (viewedInitiative.locationId) {
      removeMarker(viewedInitiative.locationId);
```

```
      try {
        const initiativeData = await HttpClient.authorizedFetch(
          `http://localhost:8080/api/v1/initiatives/find-by-
location/${viewedInitiative.locationId}`
        );
        const { latitude, longitude } = initiativeData.location;
        const marker = addMarkerToMap(
          [latitude, longitude],
          initiativeData.location.id,
          initiativeData.status
        );
        setMarkers((prevMarkers) => [...prevMarkers, marker]);
      } catch (error) {
        console.error("Error fetching initiative data:", error);
      }
    }
  };

  const initializeMap = () => {
    if (!mapRef.current) {
      mapRef.current = L.map("map").setView([49.0139, 31.2856], 6);

      addTileLayer(mapRef.current);
      showAllInitiatives();
      addGeoJSONLayer(mapRef.current);
      if (isUser) addMapClickHandler(mapRef.current);
    }
  };

  const addTileLayer = (map) => {
    L.tileLayer("https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png", {
      attribution:
        '&copy; <a href="https://www.openstreetmap.org/copyright">OpenStreetMap</a>
contributors',
    }).addTo(map);
  };

  const addGeoJSONLayer = (map) => {
    HttpClient.unauthorizedFetch("/ukraine.json")
      .then((data) => {
        L.geoJSON(data).addTo(map);
      })
      .catch((error) => {
        console.error("Error while loading GeoJSON:", error);
      });
  };

  const addMapClickHandler = (map) => {
    map.on("click", async function (e) {
      const latlng = [e.latlng.lat, e.latlng.lng];
      const { countryCode, geocode } = await getMarkerLocationInfo(latlng);
      if (countryCode === "UA") {
        const marker = addMarkerToMap(latlng);
        setCurrentAddedMarker((prevMarker) => {
          if (prevMarker) {
            prevMarker.remove();
          }
          return marker;
        });
        selectMarker(marker);
        const markerLocationInfo = {
          latitude: latlng[0],
          longitude: latlng[1],
          geocode,
```

```
        };
        onMarkerAdd(markerLocationInfo);
      } else {
        alert("Sorry, but you can add marker only on Ukraine state");
      }
    });
  };

  const getMarkerLocationInfo = async (latlng) => {
    const [lat, lng] = latlng;
    const nominatimEndpoint =
`https://nominatim.openstreetmap.org/reverse?format=json&lat=${lat}&lon=${lng}`;

    try {
      const data = await HttpClient.unauthorizedFetch(nominatimEndpoint);
      const countryCode = data.address.country_code.toUpperCase();
      const geocode = data.address["ISO3166-2-lvl4"];

      return { countryCode, geocode };
    } catch (error) {
      console.log("Error getting marker location info");
    }
  };

  const addMarkerToMap = (latlng, locationId, status) => {
    let iconUrl = "/marker-created.png";

    if (status) {
      const statusElement = InitiativeStatuses.find((el) => el.name === status);
      iconUrl = statusElement.iconUrl;
    }

    const marker = L.marker(latlng, {
      icon: new L.icon({
        iconUrl,
        iconSize: [26, 50],
        iconAnchor: [13, 50],
        popupAnchor: [13, 50],
      }),
      locationId,
    })
      .addTo(mapRef.current)
      .on("click", function (event) {
        selectMarker(event.target);
        setCurrentAddedMarker((prevMarker) => {
          if (prevMarker) {
            prevMarker.remove();
          }
          return null;
        });
        onMarkerClick(event.target.options.locationId);
      });

    return marker;
  };

  const selectMarker = (marker) => {
    if (activeMarkerRef.current) {
      activeMarkerRef.current.setOpacity(1);
    }
    activeMarkerRef.current = marker;
    marker.setOpacity(0.5);
  };
```

```
const removeMarker = (locationId) => {
  setMarkers((prevMarkers) => {
    prevMarkers.forEach((marker) => {
      if (marker.options.locationId === locationId) {
        marker.remove();
      }
    });

    return prevMarkers;
  });
};

const clearMarkers = () => {
  console.log(markers);
  setMarkers((prevMarkers) => {
    prevMarkers.forEach((marker) => {
      marker.remove();
    });
    return [];
  });
};

const handleMenuOptionsChange = () => {
  const menuOptionFunctions = {
    all: showAllInitiatives,
    user: showUserInitiatives,
    popular: showMostPopularInitiatives,
    search: showCustomSearchInitiatives,
  };

  if (menuOptions in menuOptionFunctions) {
    menuOptionFunctions[menuOptions]();
  }
};

const showAllInitiatives = () => {
  clearMarkers();
  HttpClient.authorizedFetch("http://localhost:8080/api/v1/initiatives")
    .then((data) => {
      data.forEach((initiative) => {
        const location = initiative.location;
        const marker = addMarkerToMap(
          [location.latitude, location.longitude],
          location.id,
          initiative.status
        );
        setMarkers((prevMarkers) => [...prevMarkers, marker]);
      });
    })
    .catch((error) => {
      console.error("Error while loading markers:", error);
    });
};

const showUserInitiatives = () => {
  clearMarkers();
  HttpClient.authorizedFetch(
    `http://localhost:8080/api/v1/initiatives/user/${user.id}`
  )
    .then((data) => {
      data.forEach((initiative) => {
        const location = initiative.location;
        const marker = addMarkerToMap(
          [location.latitude, location.longitude],
```

```
          location.id,
          initiative.status
        );
        setMarkers((prevMarkers) => [...prevMarkers, marker]);
      });
    })
    .catch((error) => {
      console.error("Error while loading markers:", error);
    });
};

const showCustomSearchInitiatives = () => {
  clearMarkers();
  const queryParams = new URLSearchParams(searchParams);
  HttpClient.authorizedFetch(
    `http://localhost:8080/api/v1/initiatives/find-by-search?${queryParams.toString()}`
  )
    .then((data) => {
      data.forEach((initiative) => {
        const location = initiative.location;
        const marker = addMarkerToMap(
          [location.latitude, location.longitude],
          location.id,
          initiative.status
        );
        setMarkers((prevMarkers) => [...prevMarkers, marker]);
      });
    })
    .catch((error) => {
      console.error("Error while loading markers:", error);
    });
};

const showMostPopularInitiatives = () => {
  clearMarkers();
  HttpClient.authorizedFetch(
    `http://localhost:8080/api/v1/initiatives/most-liked`
  )
    .then((data) => {
      data.forEach((initiative) => {
        const location = initiative.location;
        const marker = addMarkerToMap(
          [location.latitude, location.longitude],
          location.id,
          initiative.status
        );
        setMarkers((prevMarkers) => [...prevMarkers, marker]);
      });
      console.log(markers);
    })
    .catch((error) => {
      console.error("Error while loading markers:", error);
    });
};

const handleCreateInitiativeClose = () => {
  if (currentAddedMarker) {
    setCurrentAddedMarker((prevMarker) => {
      if (markerToAdd.locationId) {
        prevMarker.options.locationId = markerToAdd.locationId;
        setMarkers((prevMarkers) => [...prevMarkers, prevMarker]);
      } else {
        prevMarker.remove();
      }
```

```
        return null;
      });
    }
  };

  const handleSearchPlaceView = async (result) => {
    if (result) {
      const boundingbox = result.boundingbox.map(parseFloat);
      const southwest = [boundingbox[0], boundingbox[2]];
      const northeast = [boundingbox[1], boundingbox[3]];
      const bounds = [southwest, northeast];
      mapRef.current.fitBounds(bounds);
    }
  };

  return (
    <div style={{ position: "relative", width: "100%", height: "100vh" }}>
      <div
        id="map"
        style={{
          zIndex: 0,
          position: "absolute",
          top: 0,
          left: 0,
          width: "100%",
          height: "100%",
        }}
      ></div>
      <div
        style={{ zIndex: 1, position: "absolute", top: "20px", left: "440px" }}
      >
        <SearchPlaceComponent onSearch={handleSearchPlaceView} />
      </div>
    </div>
  );
};

export default MapComponent;
```

MenuComponent.ts

```
import React, { useState, useRef, useCallback } from "react";
import Typography from "@mui/material/Typography";
import Menu from "@mui/material/Menu";
import MenuItem from "@mui/material/MenuItem";
import Button from "@mui/material/Button";

const MenuComponent = ({ handleMenuOptionSelect }) => {
  const [selectedOption, setSelectedOption] = useState("all");
  const [anchorEl, setAnchorEl] = useState(null);
  const menuButtonRef = useRef(null);

  const handleOptionSelect = useCallback(
    (option) => {
      setSelectedOption(option);
      handleMenuOptionSelect(option);
      setAnchorEl(null);
    },
    [handleMenuOptionSelect]
  );

  const handleMenuOpen = (event) => {
    setAnchorEl(event.currentTarget);
```

```
  };

  const handleMenuClose = () => {
    setAnchorEl(null);
  };

  return (
    <div
      style={{
        position: "fixed",
        top: "20px",
        left: "60px",
        zIndex: 999,
        textAlign: "left",
      }}
    >
      <Button
        ref={menuButtonRef}
        onClick={handleMenuOpen}
        variant="contained"
        color="primary"
        style={{ margin: "8px", height: "40px" }}
      >
        Меню
      </Button>
      <Menu
        anchorEl={anchorEl}
        open={Boolean(anchorEl)}
        onClose={handleMenuClose}
      >
        <MenuItem
          selected={selectedOption === "all"}
          onClick={() => handleOptionSelect("all")}
        >
          <Typography variant="body1">Всі ініціативи</Typography>
        </MenuItem>
        <MenuItem
          selected={selectedOption === "user"}
          onClick={() => handleOptionSelect("user")}
        >
          <Typography variant="body1">Мої додані ініціативи</Typography>
        </MenuItem>
        <MenuItem
          selected={selectedOption === "popular"}
          onClick={() => handleOptionSelect("popular")}
        >
          <Typography variant="body1">Найпопулярніші ініціативи</Typography>
        </MenuItem>
        <MenuItem
          selected={selectedOption === "search"}
          onClick={() => handleOptionSelect("search")}
        >
          <Typography variant="body1">Розширений пошук</Typography>
        </MenuItem>
        <MenuItem
          selected={selectedOption === "statistics"}
          onClick={() => handleOptionSelect("statistics")}
        >
          <Typography variant="body1">Статистика</Typography>
        </MenuItem>
      </Menu>
    </div>
  );
};
```

```
export default MenuComponent;
```

ViewInitiativeComponent.ts

```
import React, { useState, useEffect } from "react";
import Typography from "@mui/material/Typography";
import Button from "@mui/material/Button";
import FavoriteIcon from "@mui/icons-material/Favorite";
import IconButton from "@mui/material/IconButton";
import FavoriteBorderIcon from "@mui/icons-material/FavoriteBorder";
import CloseIcon from "@mui/icons-material/Close";
import Grid from "@mui/material/Grid";
import DeleteIcon from "@mui/icons-material/Delete";
import CommentsComponent from "./CommentsComponent";
import AdminNotesComponent from "./AdminNotesComponent";
import { InitiativeStatuses } from "../../constants";
import HttpClient from "../../HttpClient";

const ViewInitiativeComponent = ({ locationId, onClose }) => {
  const [initiative, setInitiative] = useState(null);
  const [likesInfo, setLikesInfo] = useState({ userLike: null, likesCount: 0 });
  const [showUserComments, setShowUserComments] = useState(false);
  const [showAdminNotes, setShowAdminNotes] = useState(false);
  const user = JSON.parse(localStorage.getItem("user"));
  const isAdmin = user.role === "Admin";

  useEffect(() => {
    fetchInitiativeData();
  }, [locationId]);

  const fetchInitiativeData = async () => {
    try {
      const initiativeData = await HttpClient.authorizedFetch(
        `http://localhost:8080/api/v1/initiatives/find-by-location/${locationId}`
      );
      const imagesData = await HttpClient.authorizedFetch(
        `http://localhost:8080/api/v1/images/initiative/${initiativeData.id}`
      );
      const likesData = await HttpClient.authorizedFetch(
        `http://localhost:8080/api/v1/likes/initiatives/user-
like?userId=${user.id}&initiativeId=${initiativeData.id}`
      );
      const likesCountData = await HttpClient.authorizedFetch(
        `http://localhost:8080/api/v1/likes/initiatives/count-likes/${initiativeData.id}`
      );

      setInitiative({ ...initiativeData, images: imagesData });
      setLikesInfo({
        userLike: likesData.id ? likesData : null,
        likesCount: +likesCountData,
      });
    } catch (error) {
      console.error("Error fetching initiative data:", error);
    }
  };

  const deleteUserLike = async () => {
    try {
      await HttpClient.authorizedFetch(
        `http://localhost:8080/api/v1/likes/initiatives/${likesInfo.userLike.id}`,
        {
          method: "DELETE",
        }
```

```
    );
    setLikesInfo((prevInfo) => ({
      ...prevInfo,
      userLike: null,
      likesCount: prevInfo.likesCount - 1,
    }));
  } catch (error) {
    console.error("Error deleting like:", error);
  }
};

const addUserLike = async () => {
  try {
    const data = await HttpClient.authorizedFetch(
      "http://localhost:8080/api/v1/likes/initiatives",
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
        },
        body: JSON.stringify({
          userId: user.id,
          initiativeId: initiative.id,
        }),
      }
    );
    setLikesInfo((prevInfo) => ({
      ...prevInfo,
      userLike: data,
      likesCount: prevInfo.likesCount + 1,
    }));
  } catch (error) {
    console.error("Error adding like:", error);
  }
};

const handleLikeButtonClick = async () => {
  try {
    if (likesInfo.userLike) {
      await deleteUserLike();
    } else {
      await addUserLike();
    }
  } catch (error) {
    console.error("Error handling like:", error);
  }
};

const handleDeleteInitiative = async (initiativeId) => {
  try {
    await HttpClient.authorizedFetch(
      `http://localhost:8080/api/v1/initiatives/${initiativeId}`,
      {
        method: "DELETE",
      }
    );
    onClose();
  } catch (error) {
    console.error("Error deleting initiative:", error);
  }
};

const toggleUserComments = () => {
  setShowAdminNotes(false);
```

```
  setShowUserComments(!showUserComments);
};

const toggleAdminComments = () => {
  setShowUserComments(false);
  setShowAdminNotes(!showAdminNotes);
};

const getStatusColor = () => {
  const status = InitiativeStatuses.find(
    (status) => status.name === initiative.status
  );
  return status ? status.color : "black";
};

return (
  <div
    style={{
      position: "absolute",
      zIndex: 999,
      padding: "30px",
      top: "20px",
      right: "20px",
      width: "calc(100% / 3)",
      flex: "none",
      overflowY: "auto",
      maxHeight: "85vh",
      borderRadius: "10px",
      backgroundColor: "#ffffff",
      border: "1px solid #ccc",
      boxShadow: "0px 4px 8px rgba(0, 0, 0, 0.1)",
    }}
  >
    {!initiative && <div>Loading...</div>}
    {initiative && (
      <>
        <Grid container alignItems="center">
          <Grid item xs={11}>
            <Typography variant="h5">Перегляд ініціативи</Typography>
          </Grid>
          <Grid item xs={1}>
            <Button color="error" onClick={onClose}>
              <CloseIcon style={{ fontSize: "32px" }} />
            </Button>
          </Grid>
        </Grid>

        <hr style={{ marginTop: "20px", marginBottom: "20px" }} />
        <Grid
          container
          spacing={2}
          style={{ fontSize: "1.05em", padding: "10px" }}
        >
          <Grid item xs={1.5}>
            {" "}
            <Typography variant="body1" fontWeight="bold">
              Назва:
            </Typography>
          </Grid>
          <Grid item xs={10.5}>
            {" "}
            <Typography variant="body1" fontStyle="italic" fontSize="1.05em">
              {initiative.title}
            </Typography>
```

```
</Grid>
<Grid item xs={1.5}>
  <Typography variant="body1" fontWeight="bold">
    Тема:
  </Typography>
</Grid>
<Grid item xs={10.5}>
  <Typography variant="body1" fontStyle="italic">
    {initiative.topic ? initiative.topic.name : "Select topic"}
  </Typography>
</Grid>
<Grid item xs={1.5}>
  <Typography variant="body1" fontWeight="bold">
    Автор:
  </Typography>
</Grid>
<Grid item xs={10.5}>
  <Typography variant="body1" fontStyle="italic">
    {initiative.user.name + " " + initiative.user.surname}
  </Typography>
</Grid>
<Grid item xs={1.5}>
  {" "}
  <Typography variant="body1" fontWeight="bold">
    Статус:
  </Typography>
</Grid>
<Grid item xs={10.5}>
  {" "}
  <Typography
    variant="body1"
    fontStyle="italic"
    fontWeight="bold"
    marginLeft="5px"
    style={{ color: getStatusColor() }}
  >
    {initiative.status}
  </Typography>
</Grid>
<Grid item xs={1.5}>
  <Typography variant="body1" fontWeight="bold">
    Опис:
  </Typography>
</Grid>
<Grid item xs={10.5}>
  <Typography variant="body1" fontStyle="italic">
    {initiative.description}
  </Typography>
</Grid>
<Grid item xs={1.5}>
  {" "}
  <div>
    {initiative.images && (
      <Typography variant="body1" fontWeight="bold">
        Фото:
      </Typography>
    )}
  </div>
</Grid>
<Grid item xs={10.5}>
  {" "}
  <div>
    {initiative.images &&
      initiative.images.map((image, index) => (
```

```
            <img
              key={index}
              src={image.src}
              alt={`Initiative img ${index + 1}`}
              style={{
                maxWidth: "100%",
                maxHeight: "400px",
                height: "auto",
                width: "auto",
                borderRadius: "8px",
              }}
            />
          ))}
      </div>
    </Grid>
  </Grid>
  <div
    style={{
      display: "flex",
      alignItems: "center",
      marginTop: "15px",
      justifyContent: "space-between",
    }}
  >
    <IconButton
      onClick={handleLikeButtonClick}
      style={{ marginRight: "10px", color: "#e1306c" }}
    >
      {likesInfo.userLike ? (
        <FavoriteIcon style={{ fontSize: "32px" }} />
      ) : (
        <FavoriteBorderIcon style={{ fontSize: "32px" }} />
      )}
      <Typography
        variant="body2"
        style={{ marginLeft: "5px", fontSize: "18px" }}
      >
        {likesInfo.likesCount}
      </Typography>
    </IconButton>
    {isAdmin && (
      <Button
        onClick={() => handleDeleteInitiative(initiative.id)}
        style={{ marginLeft: "auto", color: "red" }}
      >
        <DeleteIcon />
      </Button>
    )}
    <div>
      <Typography variant="body2">Дата створення:</Typography>
      <Typography marginLeft="auto" variant="body2">
        {new Date(initiative.createdAt).toLocaleString("uk-UA", {
          timeZone: "Europe/Kyiv",
          day: "numeric",
          month: "long",
          year: "numeric",
          hour: "numeric",
          minute: "numeric",
        })}
      </Typography>
    </div>
  </div>
  <hr style={{ marginTop: "20px", marginBottom: "20px" }} />
  <Button
```

```
            variant="contained"
            color={showUserComments ? "secondary" : "primary"}
            onClick={toggleUserComments}
          >
            {showUserComments ? "Сховати коментарі" : "Коментарі користувачів"}
          </Button>
          <Button
            style={{ marginLeft: "10px" }}
            variant="contained"
            color={showAdminNotes ? "secondary" : "primary"}
            onClick={toggleAdminComments}
          >
            {showAdminNotes ? "Сховати відповіді" : "Відповіді адміністратора"}
          </Button>
          {showUserComments && (
            <CommentsComponent
              initiativeId={initiative.id}
              style={{ marginTop: "40px" }}
            />
          )}
          {showAdminNotes && (
            <AdminNotesComponent
              initiativeId={initiative.id}
              style={{ marginTop: "40px" }}
            />
          )}
        </>
      )}
    </div>
  );
};

export default ViewInitiativeComponent;
```

SearchInitiativeComponent.ts

```
import React, { useState, useEffect } from "react";
import Button from "@mui/material/Button";
import SearchIcon from "@mui/icons-material/Search";
import TextField from "@mui/material/TextField";
import MenuItem from "@mui/material/MenuItem";
import CloseIcon from "@mui/icons-material/Close";
import Grid from "@mui/material/Grid";
import Typography from "@mui/material/Typography";
import { useFormik } from "formik";
import * as Yup from "yup";
import { format, parse, isValid } from "date-fns";
import { InitiativeStatuses } from "../../constants";
import HttpClient from "../../HttpClient";

const SearchInitiativeComponent = ({ onSearch, onClose }) => {
  const [themes, setThemes] = useState([]);
  const [regions, setRegions] = useState([]);

  useEffect(() => {
    fetchThemes();
    fetchRegions();
  }, []);

  const fetchThemes = async () => {
    try {
      const data = await HttpClient.authorizedFetch(
        "http://localhost:8080/api/v1/topics"
      );
```

```
    setThemes(data);
  } catch (error) {
    console.error("Error fetching themes:", error);
  }
};

const fetchRegions = async () => {
  try {
    const data = await HttpClient.authorizedFetch(
      "http://localhost:8080/api/v1/regions"
    );
    setRegions(data);
  } catch (error) {
    console.error("Error fetching regions:", error);
  }
};

const validationSchema = Yup.object().shape({
  title: Yup.string()
    .max(128, "Назва не може бути більше 128 символів")
    .test(
      "notEmpty",
      "Назва не може складатися лише з пустих символів",
      (value) => {
        return !value || value.trim().length > 0;
      }
    ),
  createdAtStart: Yup.string().test(
    "isValidDate",
    "Дата повинна бути у форматі дд.мм.рррр",
    (value) => {
      if (!value) return true;
      const date = parse(value, "dd.MM.yyyy", new Date());
      return isValid(date);
    }
  ),
  createdAtEnd: Yup.string().test(
    "isValidDate",
    "Дата повинна бути у форматі дд.мм.рррр",
    (value) => {
      if (!value) return true;
      const date = parse(value, "dd.MM.yyyy", new Date());
      return isValid(date);
    }
  ),
});

const formik = useFormik({
  initialValues: {
    title: "",
    topicId: "",
    status: "",
    regionId: "",
    createdAtStart: "",
    createdAtEnd: "",
  },
  validationSchema,
  onSubmit: (values) => {
    const queryParams = {};
    for (const key in values) {
      if (values[key]) {
        if (key === "createdAtStart" || key === "createdAtEnd") {
          const date = parse(values[key], "dd.MM.yyyy", new Date());
          queryParams[key] = format(date, "yyyy-MM-dd");
```

```
        } else {
          queryParams[key] = values[key];
        }
      }
    }
    onSearch(queryParams);
  },
});

return (
  <div
    style={{
      position: "fixed",
      bottom: "20px",
      left: "20px",
      backgroundColor: "#ffffff",
      boxShadow: "0px 4px 8px rgba(0, 0, 0, 0.1)",
      zIndex: 998,
      textAlign: "left",
      padding: "20px",
      borderRadius: "12px",
      maxWidth: "500px",
    }}
  >
    <div
      style={{
        display: "flex",
        justifyContent: "space-between",
        marginBottom: "20px",
      }}
    >
      <Typography variant="h5" padding="10px">
        Пошук ініціатив
      </Typography>
      <Button color="inherit" onClick={onClose}>
        <CloseIcon style={{ fontSize: "28px" }} />
      </Button>
    </div>
    <form onSubmit={formik.handleSubmit} padding="10px">
      <Grid container spacing={2}>
        <Grid item xs={6}>
          <TextField
            name="title"
            value={formik.values.title}
            onChange={formik.handleChange}
            onBlur={formik.handleBlur}
            label="Назва"
            placeholder="Введіть назву"
            fullWidth
            error={formik.touched.title && Boolean(formik.errors.title)}
            helperText={formik.touched.title && formik.errors.title}
          />
        </Grid>
        <Grid item xs={6}>
          <TextField
            select
            name="topicId"
            value={formik.values.topicId}
            onChange={formik.handleChange}
            onBlur={formik.handleBlur}
            label="Тема"
            fullWidth
          >
            <MenuItem value="">Всі теми</MenuItem>
```

```
    {themes.map((theme) => (
      <MenuItem key={theme.id} value={theme.id}>
        {theme.name}
      </MenuItem>
    ))}
  </TextField>
</Grid>
<Grid item xs={6}>
  <TextField
    select
    name="status"
    value={formik.values.status}
    onChange={formik.handleChange}
    onBlur={formik.handleBlur}
    label="Статус"
    fullWidth
  >
    <MenuItem value="">Всі статуси</MenuItem>
    {InitiativeStatuses.map((status) => (
      <MenuItem key={status.name} value={status.name}>
        {status.name}
      </MenuItem>
    ))}
  </TextField>
</Grid>

<Grid item xs={6}>
  <TextField
    select
    name="regionId"
    value={formik.values.regionId}
    onChange={formik.handleChange}
    onBlur={formik.handleBlur}
    label="Область"
    fullWidth
  >
    <MenuItem value="">Вся територія України</MenuItem>
    {regions.map((region) => (
      <MenuItem key={region.id} value={region.id}>
        {region.name}
      </MenuItem>
    ))}
  </TextField>
</Grid>
<Grid item xs={6}>
  <TextField
    label="Початкова дата створення"
    value={formik.values.createdAtStart}
    onChange={formik.handleChange}
    onBlur={formik.handleBlur}
    name="createdAtStart"
    placeholder="дд.мм.рррр"
    fullWidth
    error={
      formik.touched.createdAtStart &&
      Boolean(formik.errors.createdAtStart)
    }
    helperText={
      formik.touched.createdAtStart && formik.errors.createdAtStart
    }
  />
</Grid>
<Grid item xs={6}>
  <TextField
```

```
              label="Кінцева дата створення"
              value={formik.values.createdAtEnd}
              onChange={formik.handleChange}
              onBlur={formik.handleBlur}
              name="createdAtEnd"
              placeholder="дд.мм.рррр"
              fullWidth
              error={
                formik.touched.createdAtEnd &&
                Boolean(formik.errors.createdAtEnd)
              }
              helperText={
                formik.touched.createdAtEnd && formik.errors.createdAtEnd
              }
            />
          </Grid>
          <Grid item xs={12}>
            <Button
              variant="contained"
              color="primary"
              type="submit"
              style={{ borderRadius: "8px" }}
              fullWidth
              endIcon={<SearchIcon style={{ marginRight: "5px" }} />}
            >
              Знайти
            </Button>
          </Grid>
        </Grid>
      </form>
    </div>
  );
};

export default SearchInitiativeComponent;
```

SearchPlaceComponent.ts

```
import React, { useState } from "react";
import Typography from "@mui/material/Typography";
import Button from "@mui/material/Button";
import SearchIcon from "@mui/icons-material/Search";
import CloseIcon from "@mui/icons-material/Close";
import HttpClient from "../../HttpClient";

const SearchPlaceComponent = ({ onSearch }) => {
  const [query, setQuery] = useState("");
  const [searchResults, setSearchResults] = useState([]);

  const handleSearch = async () => {
    if (query.trim() !== "") {
      try {
        const data = await HttpClient.unauthorizedFetch(

`https://nominatim.openstreetmap.org/search?countrycodes=UA&q=${query}&format=json&limit=4`
        );
        setSearchResults(data);
      } catch (error) {
        console.error(
          "Error fetching data from nominatim.openstreetmap.org:",
          error
        );
      }
    }
```

```
};

const handleResultClick = (result) => {
  onSearch(result);
  //clearSearchResults();
  setSearchResults([]);
};

const clearSearchResults = () => {
  setSearchResults([]);
};

return (
  <div style={{ marginTop: "5px", marginLeft: "180px" }}>
    <input
      type="text"
      value={query}
      onChange={(e) => setQuery(e.target.value)}
      placeholder="Введіть місце для пошуку"
      style={{
        padding: "10px",
        width: "200px",
        height: "24px",
        borderRadius: "8px",
        border: "1px solid #ccc",
        marginRight: "10px",
        fontSize: "1em",
      }}
    />
    <Button
      variant="contained"
      color="primary"
      onClick={handleSearch}
      style={{ borderRadius: "8px" }}
    >
      <SearchIcon style={{ height: "32px" }} />
    </Button>
    <div style={{ marginTop: "10px", width: "400px" }}>
      {searchResults.length > 0 && (
        <div
          style={{
            borderRadius: "8px",
            backgroundColor: "rgba(255, 255, 255, 0.9)",
            padding: "10px",
          }}
        >
          <div
            style={{
              display: "flex",
              padding: "10px",
              justifyContent: "space-between",
            }}
          >
            <Typography variant="h6">Результати пошуку:</Typography>
            <Button color="error" onClick={clearSearchResults}>
              <CloseIcon />
            </Button>
          </div>
          <div
            style={{
              borderRadius: "8px",
              padding: "10px",
              maxHeight: "200px",
              overflowY: "auto",
```

```
            }}
          >
            {searchResults.map((result) => (
              <React.Fragment key={result.place_id}>
                <div
                  onClick={() => handleResultClick(result)}
                  style={{
                    cursor: "pointer",
                    marginBottom: "5px",
                    border: "1px solid #ccc",
                    paddingBottom: "5px",
                  }}
                >
                  <Typography
                    style={{ fontStyle: "normal", padding: "5px" }}
                    onMouseEnter={(e) => {
                      e.target.style.fontStyle = "italic";
                    }}
                    onMouseLeave={(e) => {
                      e.target.style.fontStyle = "normal";
                    }}
                  >
                    {result.display_name}
                  </Typography>
                </div>
              </React.Fragment>
            ))}
          </div>
        </div>
      )}
    </div>
  </div>
);
};

export default SearchPlaceComponent;
```

StatisticsComponent.ts

```
import React, { useState, useEffect } from "react";
import Typography from "@mui/material/Typography";
import Button from "@mui/material/Button";
import { RadialChart } from "react-vis";
import CloseIcon from "@mui/icons-material/Close";
import { InitiativeStatuses } from "../../constants";
import HttpClient from "../../HttpClient";

const StatisticsComponent = ({ onClose }) => {
  const [statistics, setStatistics] = useState(null);

  useEffect(() => {
    fetchStatistics();
  }, []);

  const fetchStatistics = async () => {
    try {
      const data = await HttpClient.authorizedFetch(
        "http://localhost:8080/api/v1/initiatives/month-statistics"
      );

      setStatistics({ ...data });
    } catch (error) {
      console.error("Error fetching statistics:", error);
    }
```

```
};

if (!statistics) {
  return null;
}

const statusCounts = statistics.statusCounts.map((element) => {
  console.log(element);
  const status = InitiativeStatuses.find(
    (status) => status.name === element.status
  );

  return {
    ...element,
    color: status.color ? status.color : "black",
  };
});

const totalCount = statistics.statusCounts.reduce(
  (acc, item) => acc + parseInt(item.count),
  0
);

const sortedThemes = statistics.topThemes.sort(
  (a, b) => parseInt(b.count) - parseInt(a.count)
);

const colors = ["#FF6384", "#36A2EB", "#FFCE56", "#4BC0C0"];

let chartData = sortedThemes.slice(0, 4).map((item, index) => ({
  label: item.topic_name,
  angle: parseInt(item.count),
  color: colors[index],
}));

const otherCount =
  totalCount - chartData.reduce((acc, data) => acc + data.angle, 0);

if (otherCount > 0) {
  chartData.push({
    label: "Інші теми",
    angle: otherCount,
    color: "#808080",
  });
}

return (
  <div
    style={{
      position: "fixed",
      bottom: "20px",
      left: "20px",
      backgroundColor: "#ffffff",
      boxShadow: "0px 4px 8px rgba(0, 0, 0, 0.1)",
      textAlign: "left",
      padding: "20px",
      borderRadius: "12px",
      width: "550px",
    }}
  >
    <div
      style={{
        marginLeft: "10px",
        marginTop: "10px",
```

```
      display: "flex",
      justifyContent: "space-between",
    }}
  >
    <Typography variant="h5">Статистика</Typography>
    <Button color="inherit" onClick={onClose}>
      <CloseIcon style={{ fontSize: "28px" }} />
    </Button>
  </div>
  <div style={{ padding: "20px" }}>
    <div>
      <Typography variant="h6">1. Найпопулярніші теми:</Typography>
      <div style={{ height: "200px", position: "relative" }}>
        <RadialChart
          data={chartData}
          width={200}
          height={200}
          showLabels={false}
          colorType="literal"
        />
        <div
          style={{
            position: "absolute",
            top: "50%",
            left: "250px",
            transform: "translateY(-50%)",
            display: "flex",
            flexDirection: "column",
            alignItems: "flex-start",
            justifyContent: "center",
          }}
        >
          {chartData.map((dataItem, index) => (
            <div
              key={index}
              style={{
                display: "flex",
                alignItems: "center",
                marginBottom: "5px",
              }}
            >
              <div
                style={{
                  width: "10px",
                  height: "10px",
                  backgroundColor: dataItem.color,
                  marginRight: "5px",
                  borderRadius: "50%",
                }}
              ></div>
              <span
                style={{
                  overflow: "hidden",
                  textOverflow: "ellipsis",
                  whiteSpace: "nowrap",
                  marginLeft: "5px",
                }}
              >
                {dataItem.label}
              </span>
            </div>
          ))}
        </div>
      </div>
```

```
        </div>
        <div style={{ display: "flex" }}>
          <div style={{ width: "270px" }}>
            <Typography variant="h6">2. Звернень за статусами:</Typography>
            <div style={{ marginLeft: "20px", marginTop: "15px" }}>
              {statusCounts.map((item, index) => (
                <div key={index} style={{ marginBottom: "5px" }}>
                  <span
                    style={{
                      display: "inline-block",
                      width: "10px",
                      height: "10px",
                      backgroundColor: item.color,
                      marginRight: "5px",
                      borderRadius: "50%",
                    }}
                  ></span>
                  <span style={{ marginLeft: "5px" }}>
                    {item.status}: {item.count}
                  </span>
                </div>
              ))}
            </div>
          </div>
          <div style={{ marginLeft: "40px", width: "380px" }}>
            <Typography variant="h6">
              3. Міста з найбільшою кількістю звернень:
            </Typography>
            <ol>
              {statistics.topCities.map((item, index) => (
                <li key={index} style={{ padding: "2px" }}>
                  {item.region_name}: {item.count}
                </li>
              ))}
            </ol>
          </div>
        </div>
      </div>
    </div>
  );
};

export default StatisticsComponent;
```

LogoutComponent.ts

```
import React, { useState, useEffect, useRef } from "react";
import Typography from "@mui/material/Typography";
import Button from "@mui/material/Button";
import { useNavigate } from "react-router-dom";
import IconButton from "@mui/material/IconButton";
import CloseIcon from "@mui/icons-material/Close";
import AccountCircleIcon from "@mui/icons-material/AccountCircle";
import Paper from "@mui/material/Paper";
import Box from "@mui/material/Box";
import HttpClient from "../../HttpClient";

const LogoutComponent = () => {
  const user = JSON.parse(localStorage.getItem("user"));
  const navigate = useNavigate();
  const [isUserInfoVisible, setIsUserInfoVisible] = useState(false);
  const wrapperRef = useRef(null);

  const handleLogout = async () => {
```

```
  try {
    const refreshToken = localStorage.getItem("refreshToken");
    await HttpClient.unauthorizedFetch(
      "http://localhost:8080/api/v1/auth/signout",
      {
        method: "POST",
        headers: {
          "Content-Type": "application/json",
          Authorization: `Bearer ${refreshToken}`,
        },
      }
    );

    localStorage.removeItem("accessToken");
    localStorage.removeItem("refreshToken");
    localStorage.removeItem("user");
    navigate("/login");
  } catch (error) {
    console.error("Logout failed:", error);
  }
};

const handleProfileClick = () => {
  setIsUserInfoVisible(true);
};

const handleCloseClick = () => {
  setIsUserInfoVisible(false);
};

const handleClickOutside = (event) => {
  if (wrapperRef.current && !wrapperRef.current.contains(event.target)) {
    setIsUserInfoVisible(false);
  }
};

useEffect(() => {
  if (isUserInfoVisible) {
    document.addEventListener("mousedown", handleClickOutside);
  } else {
    document.removeEventListener("mousedown", handleClickOutside);
  }

  return () => {
    document.removeEventListener("mousedown", handleClickOutside);
  };
}, [isUserInfoVisible]);

return (
  <div
    style={{
      position: "fixed",
      top: "30px",
      right: "30px",
      zIndex: 998,
      textAlign: "left",
    }}
  >
    <IconButton
      onClick={handleProfileClick}
      style={{
        width: "45px",
        height: "45px",
        backgroundColor: "#ffffff",
```

```
      boxShadow: "0px 4px 8px rgba(0, 0, 0, 0.1)",
      borderRadius: "50%",
    }}
  >
    <AccountCircleIcon
      style={{
        width: "45px",
        height: "45px",
        color: "#1976d2",
      }}
    />
  </IconButton>

  {isUserInfoVisible && (
    <Paper
      ref={wrapperRef}
      elevation={3}
      style={{
        position: "absolute",
        top: "60px",
        right: "0px",
        padding: "20px",
        borderRadius: "12px",
        width: "300px",
      }}
    >
      <Box
        display="flex"
        justifyContent="space-between"
        alignItems="center"
      >
        <Typography variant="h6">Поточний користувач:</Typography>
        <IconButton
          color="inherit"
          onClick={handleCloseClick}
          style={{
            minWidth: "24px",
            padding: "5px",
          }}
        >
          <CloseIcon style={{ fontSize: "28px" }} />
        </IconButton>
      </Box>
      <div style={{ padding: "5px" }}>
        <Typography variant="body1">
          <span style={{ fontWeight: "bold", fontStyle: "italic" }}>
            Ім'я:
          </span>{" "}
          <span style={{ fontStyle: "italic" }}>
            {user && `${user.name} ${user.surname}`}
          </span>
        </Typography>
        <Typography variant="body1">
          <span style={{ fontWeight: "bold", fontStyle: "italic" }}>
            Email:
          </span>{" "}
          <span style={{ fontStyle: "italic" }}>{user && user.email}</span>
        </Typography>
        <Typography variant="body1">
          <span style={{ fontWeight: "bold", fontStyle: "italic" }}>
            Роль:
          </span>{" "}
          <span style={{ fontStyle: "italic" }}>
            {user && user.role === "User" ? "Користувач" : "Адміністратор"}
```

```
              </span>
            </Typography>
          </div>

          <Button
            variant="contained"
            color="error"
            onClick={handleLogout}
            style={{ marginTop: "10px", display: "block" }}
          >
            Вийти з профілю
          </Button>
        </Paper>
      )}
    </div>
  );
};

export default LogoutComponent;
```