МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«До захисту допущено»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)
червня 2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
на здобуття освітнього ступеня бакалавр

зі спеціальності 122 - Комп'ютерних наук,
освітньо-професійної програми «Інформатика»
на тему: «Веб-орієнтована система пошуку житла біженцями та переселенцями»
Здобувача групи ІН-91 Бондарєва Андрія Сергійовича

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело.

_____ Андрій БОНДАРЄВ
(підпис)

Керівник,
асистент,
кандидат фізико-математичних наук Ольга ШУТИЛЄВА _____
(підпис)

Суми – 2023

Сумський державний університет
Факультет електроніки та інформаційних технологій
Кафедра комп'ютерних наук

«Затверджую»
В.о. завідувача кафедри
_____ Ігор ШЕЛЕХОВ
(підпис)

ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавра

зі спеціальності 122 - Комп'ютерних наук, освітньо-професійної програми «Інформатика»
здобувача групи ІН-91 Бондарєва Андрія Сергійовича

1. Тема роботи: «Веб-орієнтована система пошуку житла біженцями та переселенцями» затверджую наказом по СумДУ від _____
2. Термін здачі здобувачем кваліфікаційної роботи_____
3. Вхідні дані до кваліфікаційної роботи _____
4. Зміст розрахунково-пояснювальної записки (перелік питань, що їх належить розробити) *1) Аналіз додатків-аналогів, визначення актуальності проблеми, постановка задачі. 2) Огляд технологій, що використовуються для розробки додатку, проектування інформаційної моделі. 3) Розробка серверної та клієнтської частини веб-орієнтованої системи пошуку житла. 4)Тестування 5) Аналіз результатів.*
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)_____
6. Консультанти до проекту (роботи), із значенням розділів проекту, що стосується їх

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв

7. Дата видачі завдання «____» _____ 20 ___ р.

Завдання прийняв до виконання _____ (підпис) Керівник _____ (підпис)

КАЛЕНДАРНИЙ ПЛАН

№ п/п	Назва етапів кваліфікаційної роботи	Термін виконання	Примітка
1	*Проведення огляду додатків аналогів*		
2	*Формування постановки задачі бакалаврської роботи*		
3	*Огляд літератури*		
4	*Проектування бази даних та інформаційної моделі*		
5	*Практична реалізація веб-орієнтованої системи*		
6	*Підбиття підсумків роботи*		

Здобувач вищої освіти _____ (підпис) Керівник _____ (підпис)

АНОТАЦІЯ

Записка: 99 стр., 54 рис., 1 додаток, 19 використаних джерел.

Обґрунтування актуальності теми роботи – тема кваліфікаційної роботи є актуальною через велику проблему з житлом для переселенців та біженців в зв'язку із війною з Росією та через те, що тема присвячена розв'язанню цієї проблеми шляхом розробки веб-орієнтованої системи.

Об'єкт дослідження – процес розробки веб-орієнтованої системи для пошуку житла біженцями та переселенцями за допомогою сучасних технологій веб-розробки

Мета роботи – розробка веб-орієнтованої системи для пошуку житла переселенцями та біженцями з використанням сучасних технологій розробки клієнтської та серверної частини веб-додатку

Методи дослідження – аналіз і вивчення інформації з відповідної літератури та інтернет-джерел, порівняльний аналіз, узагальнення, проектування, опис.

Результати – розроблено інформаційну систему, для створення та пошуку оголошень зі здачі житла постраждалим від війни на безоплатній основі. Проведено тестування розробки вручну.

ІНФОРМАЦІЙНА СИСТЕМА, ВЕБ-СЕРВІС, ПОШУК ЖИТЛА, БІЖЕНЦІ ТА ПЕРЕСЕЛЕНЦІ, JAVASCRIPT, NODEJS, REACT, REDUX, MATERIALUI, REDUX-TOOLKIT, NESTJS, GRAPHQL

ЗМІСТ

Вступ 5
1 АНАЛІТИЧНИЙ ОГЛЯД 7
1.1 Огляд додатків аналогів 7
1.2 Постановка задачі 9
2 Методика вирішення поставлених задач 11
2.1 Технології клієнтської частини веб-додатку 11
2.2 Технології серверної частини веб-додатку 16
2.3 База даних 20
2.4 Проектування інформаційної моделі 20
3 Програмна реалізація 23
3.1 База даних та сутності додатку 23
3.2 Створення серверної частини 24
3.2.1 Базові налаштування 24
3.2.2 User Module 25
3.2.3 Post Module 28
3.2.4 Auth Module 30
3.3 Створення клієнтської частини 31
3.3.1 Структура проекту 31
3.3.2 Redux store 33
3.3.3 Загальні компоненти додатку 35
3.3.4 Модуль реєстрації/авторизації 36
3.3.5 ProfilePage 39
3.3.6 Модуль пов’язаний з постами 40
3.4 Тестування готової веб-системи 45
3.4.1 Тестування серверу 45
3.4.2 Тестування клієнтської частини 49
Висновки 53
Список використаної літератури 54
Додаток А 56
А.1 Серверна частина 56
А.2 Клієнтська частина 69

ВСТУП

За останнє десятиріччя майже всі доступні сервіси та послуги стали доступними в інтернеті. Сфера веб-розробки набула величезної популярності. В зв’язку з повномасштабним вторгненням РФ проблема розміщення біженців та переселенців постала дуже гостро. Саме тому темою бакалаврської роботи було обрано веб-додаток для пошуку житла біженцями та переселенцями і відповідно для розміщення оголошення про готовність надати притулок.

Актуальність обумовлюється тим, що через повномасштабне вторгнення Російської Федерації на територію України значна кількість населення стала біженцями та переселенцями через руйнування їх домівок або перебування їх регіону під окупацією. Разом з цим зріс попит на оренду житла в більш безпечних регіонах. Це призвело до того, що значна кількість біженців та переселенців не можуть дозволити собі оренду житла. Також варто враховувати зменшення кількості робочих місць та матеріальні збитки, які понесли звичайні українці через агресію Росії. Все це обумовлює попит на додаток, який би допоміг всім хто потребує знайти житло і всім хто готовий допомогти надати свої послуги.

Об’єкт дослідження - процес розробки веб-орієнтованої системи для пошуку житла біженцями та переселенцями за допомогою сучасних технологій веб-розробки.

Предмет дослідження – сучасні технології і підходи, які застосовуються під час розробки веб-орієнтованих систем.

Гіпотеза – ґрунтується на припущенні, що розробка веб-додатку з використанням сучасних технологій спростить алгоритм та зменшить час пошуку біженцями та переселенцями тимчасового безкоштовного житла.

Новизна – полягає в створенні веб-орієнтованої системи із застосування найсучасніших технологій веб-розробки, зі зручним та зрозумілим для користувача інтерфейсом.

Структура - дана робота складається зі вступу, аналітичного огляду, постановки задачі, методики вирішення поставлених задач, програмної реалізації, висновків, списку використаних джерел та додатку.

Метою роботи є проектування та розробка веб-додатку для пошуку житла біженцями та переселенцями. А саме проведення аналізу та огляду додатків аналогів, формування постановки задачі, проведення огляду літератури з технологій, що будуть використані, проектування бази даних та інформаційної моделі, а також реалізація додатку та підбиття підсумків.

1 АНАЛІТИЧНИЙ ОГЛЯД

1.1 Огляд додатків аналогів

Для аналізу аналогів було обрано 3 додатки: Airbnb, booking.com, prykhystok.

Airbnb – веб-додаток, що надає можливості по розміщенню оголошення про оренду житла і відповідно можливості по пошуку таких оголошень. Це один з найвідоміших букінг-сайтів, який працює по всьому світу. Додаток має дуже простий, зрозумілий і разом з тим красивий дизайн та інтерфейс. Сторінка посту надає дуже широкий спектр інформації. Також є можливість одразу оплатити і забронювати місця. Функція оплати не є необхідною, враховуючи тему бакалаврської роботи. Наявна система рейтингу оголошень. Airbnb надає дуже широкі можливості фільтрації і пошуку підходящого оголошення. Наявна можливість залишати коментарі. Наявний широкий вибір мов локалізації.

У ході аналізу було виділено наступні переваги:

- Сучасний зручний інтуїтивно-зрозумілий дизайн
- Фільтрація оголошень за багатьма різними параметрами
- Можливість забронювати помешкання
- Можливість залишати коментар до оголошення
- Наявність інтерактивної мапи з відображенням пропозицій по оренді на ній
- Наявність зручного пошуку за місцем розташування
- Наявність локалізації на велику кількість популярних мов
- Наявна статистика та рейтинг авторів оголошень

В результаті аналізу додаток Airbnb обрано, як основний на який потрібно спиратися. Було обрано кілька рішень по дизайну спираючись на даний додаток: а саме було вирішено розміщати пости сіткою та для фільтрів, реєстрації, авторизації і створенню постів використано модальні вікна.

Рейтингова система та можливість оплати визначені, як не потрібні функції, оскільки здача житла біженцям та переселенцям повинна бути на безоплатній основі. Рейтингова система не імплементовано з розрахунку на швидку перемогу України у війні і відповідно автори не встигнуть отримати значну кількість відгуків.

Наступним розглянемо аналогічний попередньому додаток booking.com. Це також один із найвідоміших букінг-сайтів у світі. Інтерфейс та дизайн зручний, проте він значно більш навантажений та складніший. Можливості фільтрації також дуже широкі. Також наявна можливість оплати та бронювання місця. Наявна система рекомендацій. Локалізація багатьма мовами. Наявна рейтингова система. Широкі можливості сортування.

Серед переваг виділено наступні:

- Фільтрація оголошень за багатьма різними параметрами
- Можливість забронювати помешкання
- Можливість залишити відгук
- Рекомендації схожих місць та місць поряд
- Дуже широкі можливості пошуку
- Локалізація багатьма мовами
- 10 бальна система оцінювання помешкань
- Сортування за багатьма критеріями

Даний додаток було проаналізовано, але його переваги або не потрібні для теми роботи або реалізація в Airbnb краще підходить. Необхідності в локалізації майже немає, оскільки розроблений додаток для українців. Інтерфейс Airbnb більш зрозумілий. Системи рейтингу, відгуків та оплати не є необхідними, як зазначено раніше.

І останнім розглянемо найбільш близький за темою роботи додаток – prykhystok. Цей додаток якраз і призначений для того щоб біженці могли знайти житло, а інші розмістити для них відповідні пропозиції. Інтерфейс більше подібний до booking.com. Як було вже зазначено рішення по дизайну

та інтерфейсу найкращі в Airbnb. Можливості фільтрації також на гарному рівні. Локалізація наявна. Є можливість поскаржитися на оголошення.

З переваг даного додатка слід відзначити:

- Наявність інструкції по використанню на головній сторінці
- Статистика по пропозиціям в різних регіонах
- Локалізація багатьма мовами
- Наявність широкого вибору варіантів фільтрації
- Наявна можливість поскаржитися на оголошення

Таким чином даний додаток хоч і є найбільш близьким за темою, але все ж таки найкращим прикладом визначено Airbnb.

Явних проблем не було знайдено в жодному додатку. Відсутність тієї чи іншої функції не вважається за недоліки, оскільки при аналізі додатків відзначено лише те, що хотілося б бачити в додатку.

Даний аналіз допоміг усвідомити, які функції обов'язкові в додатку і які було б бажано додати за можливості. Таким чином було визначено, що головними функціями додатку будуть авторизація, відображення списку оголошень, можливість створення, видалення та редагування оголошення та широкий набір для фільтрації даних.

1.2 Постановка задачі

Метою виконання випускної кваліфікаційної роботи є проєктування та розробка веб-додатку для пошуку житла біженцями та переселенцями. Для цього необхідно розробити клієнтську та серверну частину додатку.

Додаток повинен мати наступний функціонал:

- Авторизація та реєстрація за допомогою модального вікна з використанням jwt-токенів
- Перегляд списку всіх оголошень із застосуванням нескінченного скролу
- Перегляд сторінки з повною інформацією про оголошення

- Перегляд профілю з персональними даними та списком власних оголошень
- Фільтрація списку оголошень за багатьма критеріями
- Створення оголошення за допомогою модального вікна
- Редагування оголошення
- Видалення оголошення

2 МЕТОДИКА ВИРІШЕННЯ ПОСТАВЛЕНИХ ЗАДАЧ

2.1 Технології клієнтської частини веб-додатку

Для розробки клієнтської частини веб-додатку було обрано мову програмування Javascript, а саме було використано бібліотеку ReactJS. Також використатно UI-бібліотеку яка інтегрується з React – Material UI. Для маршрутизації використано React Router Dom. Для виконання http-запитів було встановлено бібліотеку axios. Для керування стану додатку використано Redux-Toolkit. Розглянемо технології більш детально.

2.1.1 JavaScript (JS)

JavaScript (JS) – динамічна, об'єктно-орієнтована прототипна мова програмування. Реалізація стандарту ECMAScript. Використовується для створення скриптів на вебсторінках, що надає можливість на боці клієнта спілкуватися з користувачем, оперувати браузером, виконувати асинхронні операції для отримання даних з сервера, змінювати структуру та стилі вебсторінки.

JavaScript класифікують як прототипну (підмножина об'єктно-орієнтованої), скриптову мову програмування з динамічною типізацією. Окрім прототипної, JavaScript також частково підтримує інші парадигми програмування (імперативну та частково функціональну) і деякі відповідні архітектурні властивості, зокрема: динамічна та слабка типізація, автоматичне керування пам'яттю, прототипне наслідування, функції як об'єкти першого класу. Мова JavaScript використовується для:

- написання сценаріїв вебсторінок для надання їм інтерактивності;
- створення односторінкових та прогресивних вебзастосунків (React, AngularJS, Vue.js);
- програмування на боці сервера (Node.js (Express.js));
- стаціонарних застосунків (Electron, NW.js);

- мобільних застосунків (React Native, Cordova);
- сценаріїв в прикладних програмах (наприклад, в програмах зі складу Adobe Creative Suite чи Apache JMeter);
- всередині PDF-документів тощо [1].

JavaScript застосовується для запису послідовних операцій – «сценаріїв» чи «скриптів». Такі послідовності зазвичай інтерпретуються, а не компілюються, а тому не потребують додаткових утилит чи засобів для перетворення в інший рівень кодування.

JavaScript є найпопулярнішою мовою програмування коли мова йде про веб-розробку. Спочатку її можно було виконувати лише в браузері, але згодом з'явилась можливість виконувати її в інших середовищах, наприклад NodeJS[2].

2.1.2 React

React це javascript бібліотека для створення користувацьких інтерфейсів. React спрощує створення інтерфейсів. React дозволяє ефективно перемальовувати частину інтерфейсу в залежності від зміни стану додатку. Декларативний підхід робить код передбачуванішим та робить налагодження зручнішим.

React дозволяє створювати інкапсульовані компоненти, які керують власним станом, а з них будуються складні інтерфейси. Оскільки логіка компонентів написана на JavaScript, замість шаблонів, можна передавати складні дані у додатку і зберігати стан окремо від DOM [3].

React JS – це JavaScript-бібліотека для створення користувацьких інтерфейсів, якими здебільшого користуються в розробці односторінкових застосунків з відкритим кодом [4].

Веб-додаток на React будується за рахунок компонентів, які по суті є блоками з яких будується додаток. Важливою концепцією для розуміння React є props – спосіб передачі даних між компонентами. Ще одним важливим поняттям є state – js-об'єкт який зберігає стан компонента. У react-

компонентах використовується особливий синтаксис jsx – це поєднання html-тегів та javascript. Цей jsx-код React інтерпритує в нативний javascript за допомогою транспілятору – Babel.

Концепція Virtual Dom полягає в тому, що в React створюється віртуальне DOM дерево, яке є просто представленням реального DOM дерева. Суть полягає в тому що, при кожному рендері перемальовується не повне DOM-дерево, а спочатку визначається частина дерева яка буде перемальована, а потім за допомогою оптимізованих алгоритмів порівнюються віртуальне дерево до зміни даних і віртуальне дерево після зміни, і потім перемальовується тільки та частина реального дерева, яка відрізнялася при порівнянні віртуальних.

Також однією з базових концепцій є хуки. Фактично це просто Javascript-функції, які повинні слідувати кільком правилам:

- Викликати їх можна тільки на верхньому рівні інших компонентів. Тобто не можна викликати їх в циклах, вкладених функціях або блоках коду, що виконуються за певної умови. Це гарантує, що при кожному рендері порядок виклику хуків буде незмінним.
- Викликати хуки можна лише всередині функціональних React-компонентів. Це дозволяє бути впевненим, що всю логіку пов’язану з керуванням стану компонента буде чітко видно з коду самого компонента.

У разі необхідності є можливість створювати власні хуки, всередині яких можна викликати інші хуки.

2.1.3 Redux

Redux це бібліотека для керування і зберігання стану в Javascript додатках. Її дуже зручно використовувати разом з React.

Ця бібліотека використовується для підтримки та оновлення даних у різних додатках для спільного використання кількома компонентами, залишаючись при цьому незалежним від них.

Redux вирішує в React проблему props-drilling тобто ланцюг із передачі даних вниз по ієрархії додатку. За допомогою Redux існує можливість зробити дані стану незалежними від компонентів. Компонент також має змогу оновити дані і отримати їх зі сховища. Також Redux вирішує проблему коли, необхідно підняти дані вгору по ієрархії додатку.

Управління станами – це, по суті, спосіб полегшити комунікацію та обмін даними між компонентами. [5-6].

Розглянемо ключові сутності Redux:

- Store – глобальне сховище, яке зберігає глобальний стан всього додатку. Він зберігає стан, надає доступ до стану, оновлює його та зберігає список слухачів, тобто всіх хто спостерігає за станом і отримує інформацію про кожну його зміну.
- Actions – це прості Javascript об’єкти, які мають 2 обов’язкові поля: type – містить назву Action та payload – містить додаткові дані, які ми хочемо передати.
- Reducers – функції, які приймають попередній стан додатка та Action як аргументи та повертають оновлений стан. Вони мають бути чистими функціями.

Далі наведена схема, яка показує принцип роботи Redux.

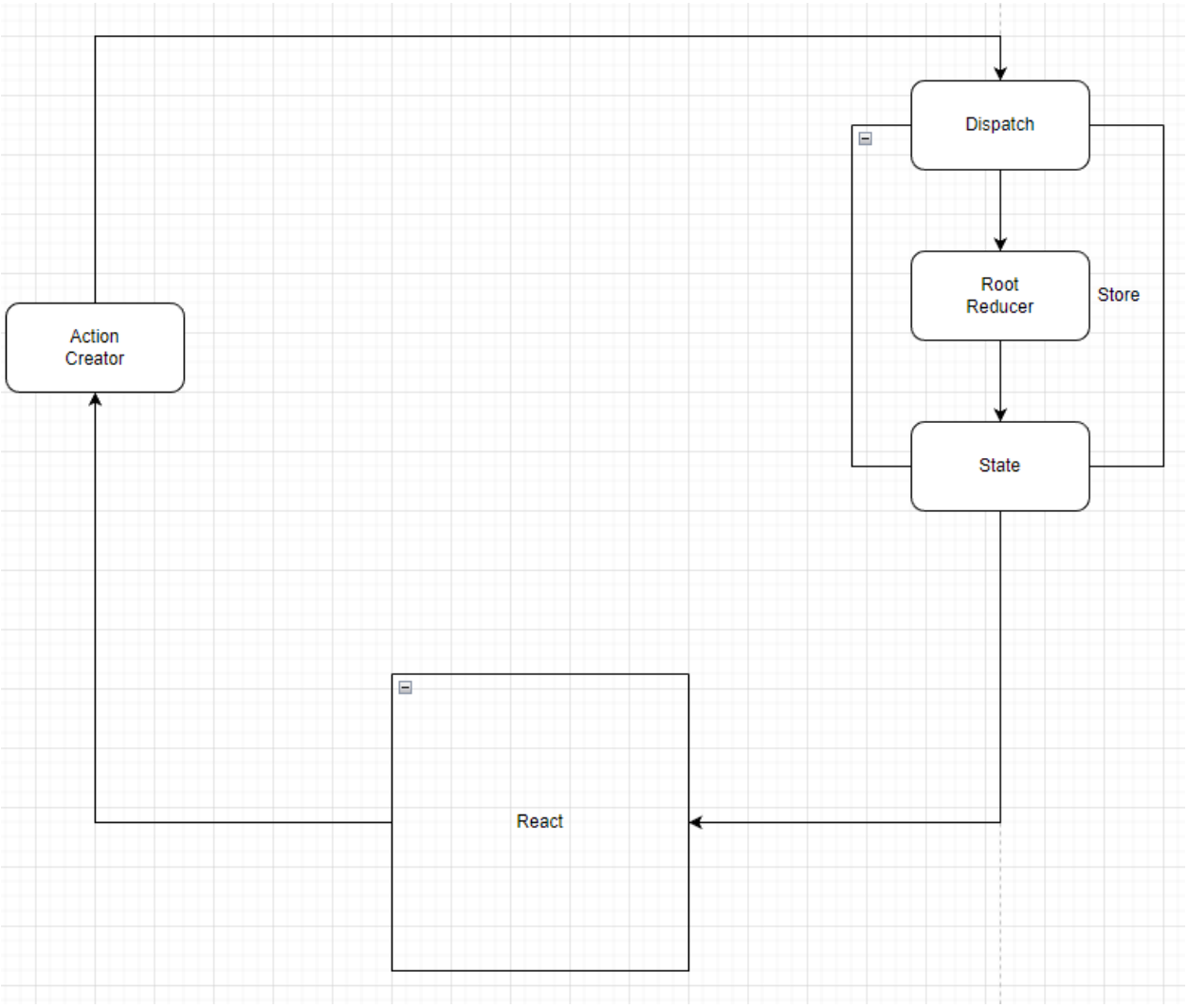

Рисунок 2.1 – Схема роботи Redux

Дана схема показує весь процес роботи Redux. React, коли йому потрібно змінити стан, викликає Action Creator – функцію, яка повертає Action, потім диспатчить цей Action, за допомогою функції dispatch, яка передає цей Action, до всіх Reducers, в яких в залежності від типу Action, виконується зміна стану додатку, яка спричиняє ререндер частини додатку, яка залежить від стану.

Для використання Redux разом з React є окрема бібліотека React-Redux, яка є додатковим шаром для коректної роботи Redux в React.

У даному додатку було використано готове рішення – Redux Toolkit. Це бібліотека, яка надає зручне готове рішення з використання Redux в React. Спочатку він був створений, щоб допомогти вирішити три поширені проблеми, пов'язані з Redux:

- Налаштування Store в Redux занадто складне
- Для коректної роботи Redux, необхідно встановити багато додаткових пакетів та бібліотек.
- Redux вимагає занадто багато шаблонного коду [7].

2.1.4 Material UI

Material UI – це UI-бібліотека готових стилізованих React-компонентів з відкритим вихідним кодом, яка реалізує Material Design від Google. [8]

Дана бібліотека надає широкий спектр базових компонентів для створення додатку. Також надається велика кількість прикладів по використанню та налаштуванню компонентів. Безкоштовно надаються навіть кілька готових шаблонів типових сторінок веб-додатків, наприклад шаблон лендінгу або адмін-панелі. Дана бібліотека надає доступ до API всіх базових компонентів, а також посібник по використанню та стилізації. Також є можливість працювати з низькорівневими хуками, що надає ще ширші можливості з налаштування компонентів.

2.2 Технології серверної частини веб-додатку

Для створення серверної частини було обрано Node.JS, а саме фреймворк NestJS. API побудовано за допомогою GraphQL, відповідно для роботи з ним використано Apollo. Для авторизації та аутентифікації користувачів використовуються jwt-токени.

2.2.1 Node.JS

Node.js – це однопоточне кросплатформове середовище виконання з відкритим вихідним кодом і середовище для виконання JavaScript, платформа з відкритим кодом для виконання високопродуктивних мережевих застосунків, написаних мовою JavaScript, надав можливість виконувати JavaScript-скрипти на сервері та відправляти користувачеві результат їхнього

виконання. Платформа Node.js перетворила JavaScript на мову загального використання з великою спільнотою розробників.

Node.js має наступні властивості:

- асинхронна одно-нитева модель виконання запитів;
- неблокуючий ввід/вивід;
- система модулів CommonJS;
- рушій JavaScript Google V8;

Для керування модулями використовується пакетний менеджер npm (node package manager). [9]

Найчастіше Node.js застосовують як web сервер і саме в цій ролі розкриваються переваги подійно-орієнтованої моделі, що не блокує введення/виведення архітектури.[10]

Особливості роботи Node.js:

- Асинхронність – кілька різних процесів можуть виконуватись паралельно, не блокуючи один одного.
- Керованість подіями – Node.js активно використовує функції зворотнього зв'язку(callback-функції), а також обробники подій. Зазвичай такі функції можна передавати в інші, щоб виконати їх після виконання певного коду, тобто після певної події.
- Один потік – код виконується в одному стекі викликів
- Сумісність з усіма основними ОС, в тому числі мобільних.

2.2.2 GraphQL

GraphQL – це мова запитів і маніпуляції даними з відкритим кодом для API і середовище виконання для обслуговування запитів з наявних даних.

GraphQL надає підхід розробки веб API і його можна порівнювати і протиставляти REST та іншим архітектурам вебсервісів. Він дозволяє клієнтам визначати структуру потрібних даних і таку саму структуру повертає сервер, таким чином запобігаючи передачі надлишкових даних, але це впливає на дієвість вебкешування результатів запитів. Гнучкість і багатість мови запитів,

що може бути не потрібна для простих API. Він складається з системи типів, мови запитів і семантики виконання, статичної валідації і інтроспекції.

GraphQL підтримує читання, писання (змінювання) і підписування на зміни даних (оновлення в реальному часі – зазвичай втілені за допомогою Webhook).

Сервери GraphQL доступні на багатьох мовах включно з Haskell, JavaScript, Perl, Python, Ruby, Java, C#, Scala, Go, Elixir, Erlang, PHP, R і Clojure.[11-12]

З основних переваг GraphQL можна відзначити наступні:

- 1 endpoint – на відміну від REST всі запити надсилаються на один endpoint
- Клієнт самостійно обирає, які дані запросити і які поля отримувати
- Строга типізація
- Зручна обробка помилок
- GraphQL надає зручні можливості по тестуванню та по документуванню

Nest (NestJS) – це фреймворк для створення ефективних, масштабованих серверних додатків Node.js. Він використовує прогресивний JavaScript, побудований на TypeScript і повністю підтримує його (але при цьому дозволяє розробникам кодувати на чистому JavaScript) і поєднує в собі елементи ООП (об'єктно-орієнтованого програмування), ФП (функціонального програмування) і ФРП (функціонально-реактивного програмування). У своїй основі Nest використовує надійні фреймворки HTTP-серверів, такі як Express (за замовчуванням), а за бажанням може бути налаштований на використання Fastify! Nest забезпечує рівень абстракції над цими поширеними фреймворками Node.js (Express/Fastify), але також відкриває їх API безпосередньо розробнику. Це дає розробникам свободу використовувати безліч сторонніх модулів, доступних для базової платформи.

За останні роки, завдяки Node.js, JavaScript став загальноприйнятим в Інтернеті як для інтерфейсних, так і для бекенд-додатків. Однак, незважаючи

на те, що для Node (і серверного JavaScript) існує безліч функціональних бібліотек, помічників та інструментів, жодна з них не вирішує головну проблему – архітектуру. Nest надає готову архітектуру додатків, яка дозволяє розробникам і командам створювати додатки, що добре тестуються, масштабуються, слабко пов'язані і легко підтримуються. Архітектура значною мірою запозичена з Angular. [13]

Серед переваг NestJS виділяють наступні:

- Розширюваність. Через чітку «ангулярну» архітектуру додатки, що використовують NestJS дуже легко розширюються та масштабуються
- Додатки на NestJS легко тестувати
- Використувається Typescript «з коробки».

2.2.3 JWT-токени

JSON Web Token (JWT) – це відкритий стандарт (RFC 7519), який визначає компактний і автономний спосіб безпечної передачі інформації між сторонами у вигляді об'єкта JSON. JWT можуть бути підписані за допомогою секретного ключа (за допомогою алгоритму HMAC) або пари публічних/приватних ключів за допомогою RSA або ECDSA.[14-15]

Найчастіше ці токени використовуються для авторизації. Після того як користувач увійшов у систему йому приходить такий токен, який дозволяє йому отримувати доступ до певних маршрутів, сервісів, функцій для яких цей токен є обов’язковим.

Токен складається з 3 частин: заголовку, вмісту (payload) та підпису. Заголовок містить інформацію про тип токена та алгоритм, який було використано при створенні підпису. Вміст містить дані, які ми хочемо передати. Підпис використовується, щоб переконатися, що дані не було змінено під час надсилання даних. Всі 3 частини токена розділені між собою крапкою

2.3 База даних

Оскільки і клієнтська і серверна частина розроблюється на Javascript то доцільно використати і СКБД, яка підтримує формат JSON, тому для роботи з базою даних було обрано MongoDB Atlas. MongoDB Atlas – це мультихмарний сервіс баз даних від тих самих людей, які створили MongoDB. [16]

MongoDB в свою чергу це – документо-орієнтована система керування базами даних (СКБД) з відкритим вихідним кодом, яка не потребує опису схеми таблиць. MongoDB займає нішу між швидкими і масштабованими системами, що оперують даними у форматі ключ/значення, і реляційними СКБД, функціональними і зручними у формуванні запитів.

Код MongoDB написаний на мові C++. MongoDB підтримує зберігання документів в JSON-подібному форматі, має досить гнучку мову для формування запитів, може створювати індекси для різних збережених атрибутів, ефективно забезпечує зберігання великих бінарних об'єктів, підтримує журналювання операцій зі зміни і додавання даних в БД, може працювати відповідно до парадигми Map/Reduce, підтримує реплікацію і побудову відмовостійких конфігурацій. [17]

У нашому випадку було створено лише 1 базу даних, яка в свою чергу містить 2 колекції: user – зберігає дані про користувачів і post – зберігає дані про оголошення.

2.4 Проектування інформаційної моделі

Для наочного представлення роботи інформаційної системи створено діаграму варіантів використання (Use Case Diagram), що є графічним зображенням можливої взаємодії користувача з системою.

Діаграма прецедентів (або діаграма варіантів використання) – в UML, діаграма, на якій зображено відношення між акторами та прецедентами в системі.

Діаграма прецедентів показує різні варіанти використання та різні типи користувачів системи і часто супроводжується іншими типами діаграм. Варіанти використання представлені колами або еліпсами. Актори (дійові особи) часто зображуються у вигляді паличок.

Діаграма прецедентів є графом, що складається з множини акторів, прецедентів (варіантів використання) обмежених межею системи (прямокутник), асоціацій між акторами та прецедентами, відношень серед прецедентів, та відношень узагальнення між акторами. Діаграми прецедентів відображають елементи моделі варіантів використання.

У мові UML є кілька стандартних видів відношень між акторами і варіантами використання:

- асоціації (association relationship)
- включення (include relationship)
- розширення (extend relationship)
- узагальнення (generalization relationship) [18].

Розроблена діаграма представлена на рисунку 2.2:

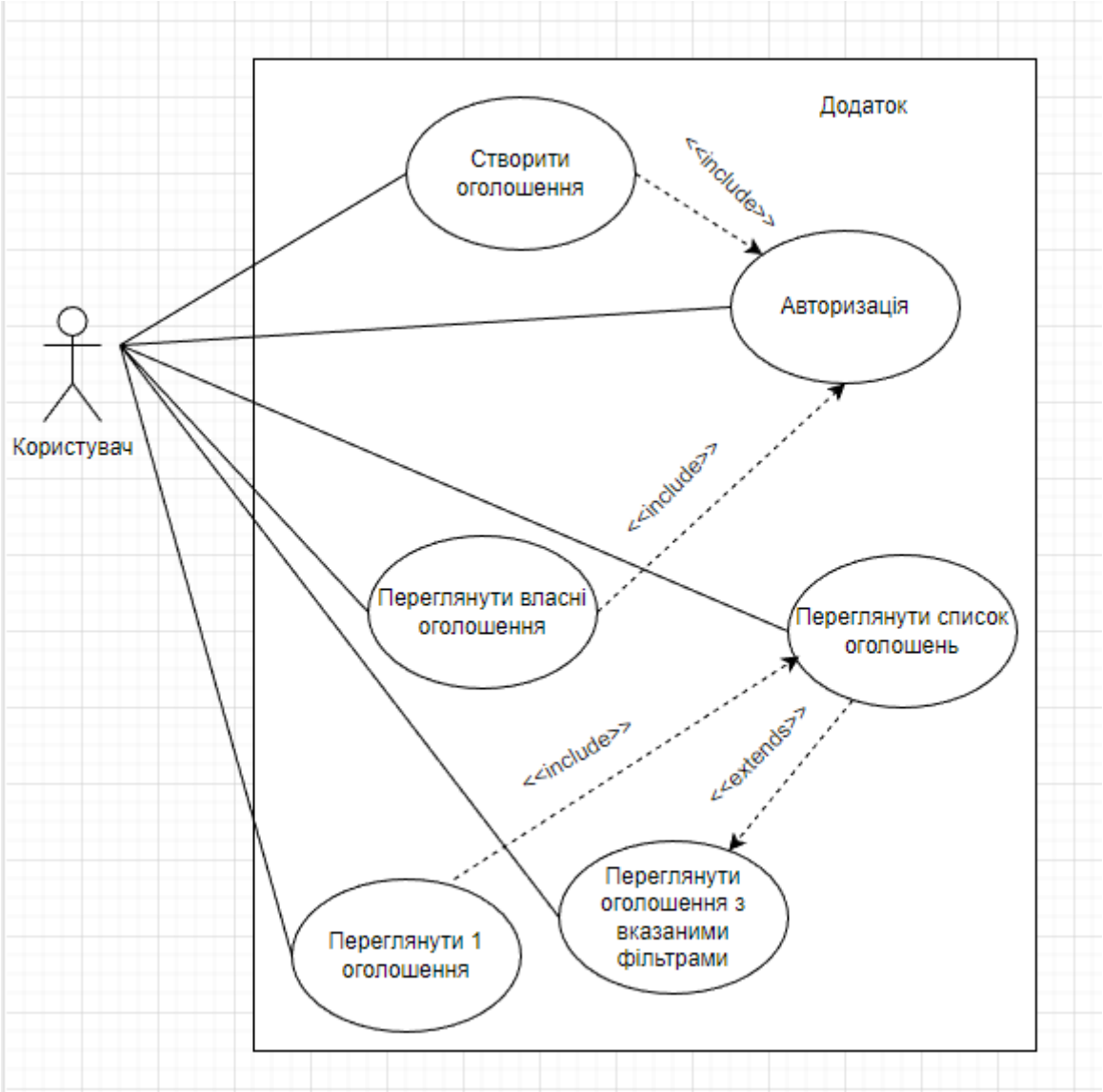

Рисунок 2.2 – Діаграма варіантів використання

Суть діаграми прецедентів полягає в тому, що проєктована система подається у вигляді множини сутностей чи акторів, що взаємодіють із системою за допомогою так званих варіантів використання. Варіант використання використовують для описання послуг, які система надає актору. Іншими словами, кожен варіант використання визначає деякий набір дій, який виконує система під час діалогу з актором. При цьому нічого не говориться про те, яким чином буде реалізовано взаємодію акторів із системою.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 База даних та сутності додатку

У веб-додатку є 2 основні сутності і відповідні типи даних– user та post. Їх опис представлено в таблицях 3.1 та 3.2

Таблиця 3.1 – Інформація про тип User

Тип поля	Назва	Призначення
string	uid	Унікальний ідентифікатор
string	email	Електронна пошта
string	phone	Номер телефону
string	password	Пароль

Таблиця 3.2 – Інформація про тип Post

Тип поля	Назва	Призначення
string	id	Унікальний ідентифікатор
string[]	clientCategories	Категорії для оренди
string	city	Місто
string	region	Область/регіон
number	countOfPlaces	Кількість спальних місць
string	title	Заголовок
string	description	Опис
string	createdAt	Дата створення
string	periodOfTime	Час на який здається
string	type	Тип пропозиції (житло, спальне місце, кімната)
string	user	Id автора

У базі даних відповідно існують колекція user, де зберігаються користувачі та колекція post, де зберігаються оголошення у форматі BSON. Для підключення серверу до БД використовується бібліотека TypeORM, а саме відповідний модуль імпортується в кореневому модулі серверу (Рис.3.1):

```
@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'mongodb',
      url: 'mongodb+srv://Bondarev14:Bondarev14@bachelorcluster.pqyr14m.mongodb.net/?retryWrites=true&w=majority',
      synchronize: true,
      useUnifiedTopology: true,
      entities: [User, Post],
      useNewUrlParser: true,
      logging: true,
    }),
```

Рисунок 3.1 – Кореневий модуль сервера, який відповідає за з'єднання з БД

На рисунку 3.2 наведено вигляд адміністративної панелі бази даних:

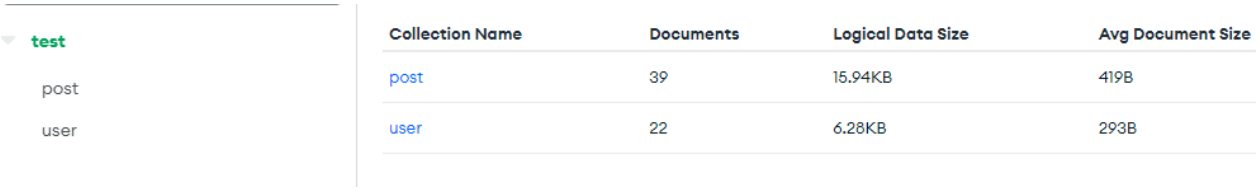

Рисунок 3.2 – Адміністративна панель бази даних

3.2 Створення серверної частини

Для серверної частини було обрано сучасний надбудову над expressJS – NestJS.

3.2.1 Базові налаштування

Для початку потрібно ініціалізувати базовий сервер NestJS, для цього небхідно виконати наступну команду для встановлення необхідних пакетів (Рис.3.3). Ця команда встановить все необхідне для роботи з GraphQL та запуску серверу. Далі автоматично створюється файл з якого запускається додаток з наступним кодом (Рис.3.4).

```
PS D:\web\BachelorWork> npm i @nestjs/graphql @nestjs/apollo @apollo/server graphql
```

Рисунок 3.3 – Команда для встановлення NestJS + GraphQL

```
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes(new ValidationPipe());
  app.use(graphqlUploadExpress({ maxFileSize: 1000000, maxFiles: 10 }));
  app.enableCors();
  await app.listen(4000);
}
bootstrap();
```

Рисунок 3.4 – Код файлу main.ts

AppModule в даному прикладі це кореневий модуль проекту, який має наступний код (Рис.3.5).

```
@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'mongodb',
      url: 'mongodb+srv://Bondarev14:Bondarev14@bachelorcluster.pqyr14m.mongodb.net/?retryWrites=true&w=majority',
      synchronize: true,
      useUnifiedTopology: true,
      entities: [User, Post],
      useNewUrlParser: true,
      logging: true,
    }),
    GraphQLModule.forRoot<ApolloDriverConfig>({
      driver: ApolloDriver,
      autoSchemaFile: true,
    }),
    UserModule,
    PostModule,
  ],
})
export class AppModule {}
```

Рисунок 3.5 – Код AppModule

Як видно тут є налаштування з’єднання з базою даних, підключення модуля для роботи з GraphQL та імпортуються модулі про які буде йти мова далі.

3.2.2 User Module

Цей модуль відповідає за всі дії з користувачами, а саме за їх створення, за входження користувача до системи, отримування даних про користувача за

його id, за адресою його електронної пошти. Сам файл модуля не містить нічого не звичного і просто імпортує відповідний сервіс. Розглянемо більш детально код сервісу користувачів та GraphQL resolver. Сервіс це клас, що містить в собі різні методи для керування користувачів. Розглянемо 2 методи для прикладу. Спершу рисунок 3.6

```
async getUser(uid: string): Promise<User> {
  return this.userRepository.findOneBy({
    uid,
  });
}
```

Рисунок 3.6 – Код функції getUser

Це метод для отримання користувача з бази даних за вказаним id. Це проста javascript функція, яка повертає Promise, саме тому ми вказуємо, що це асинхронна функція за допомогою async. Функція повертає результат виконання цього промісу. У якості параметру приймає рядок, який є ідентифікатором користувача і викликає функцію, яка шукає в відповідному репозиторії запис з таким самим uid.

Тепер розглянемо більш складний метод, який створює нового користувача (Рис.3.7).

```
async createUser(createUserInput: CreateUserInput) {
  const { email, phone, password } = createUserInput;
  const isUserExists = await this.getUserByEmail(email);
  try {
    if (isUserExists) {
      throw new Error('CONFLICT, Email`s already used');
    } else {
      const saltOrRounds = 10;
      const hashPassword = await bcrypt.hash(password, saltOrRounds);
      const user = await this.userRepository.create({
        uid: uuid(),
        email,
        phone,
        password: hashPassword,
        posts: [],
      });
      await this.userRepository.save(user);
      return this.authService.generateUserCredentials(user);
    }
  } catch (err) {
    throw new HttpException(
      'CONFLICT, Email`s already used',
      HttpStatus.CONFLICT,
    );
  }
}
```

Рисунок 3.7 – Код функції createUser

Дана функція приймає 1 аргумент createUserInput, користувацького типу CreateUserInput (Рис.3.8):

```
@InputType()
export class CreateUserInput {
  @IsEmail()
  @Field()
  email: string;
  @IsPhoneNumber('UA')
  @Field()
  phone: string;
  @IsStrongPassword()
  @Field()
  password: string;
}
```

Рисунок 3.8 – Опис InputType CreateUserInput

Про те що це окремий тип даних вказує декоратор @InputType. Даний тип має 3 поля: пошта, номер телефону та пароль. Всі ці 3 поля є типу string. Також на кожен з них накладено декоратор для валідації. Для того щоб вказати, що це поля використовується декоратор @Field.

Повернемося до самої функції. Було використано деструктуризуюче присвоювання для того, щоб записати дані з вхідного аргумента у відповідні змінні. Потім виконується перевірка на те, чи існує користувач з такою електронною поштою в базі даних. Якщо існує такий користувач то викликається відповідна помилка, а якщо ні, то пароль шифрується і створюється новий користувач з вхідними даними і зашифрованим паролем. Новий користувач зберігається, а потім за допомогою authService, про який буде йтися далі генерується jwt-токен для нового користувача і саме цей токен і повертається функцією. За таким же принципом даний сервіс має ще декілька функцій.

Розглянемо тепер user.resolver. Резолвери (Resolvers) надають інструкції для перетворення операції GraphQL (запиту(query) чи мутації (mutation)) в дані. Вони повертають дані у тій самій формі, яку ми вказали в нашій схемі (Schema) – або синхронно, або у вигляді промісу (Promise), яка перетворюється

на результат цієї форми. Пакет @nestjs/graphql генерує карту резолвера автоматично, використовуючи метадані, надані декораторами, які використовуються для анотування класів. [19]

Тобто фактично резолвери це класи, які містять інструкції по тому, як виконувати мутації та запити, у вигляді функцій на які повісили декоратори. Розглянемо приклади функції (Рис.3.9).

```
@Query((returns) => UserType)
user(@Args('uid') uid: string) {
  return this.userService.getUser(uid);
}
```

Рисунок 3.9 – Код резолвера user

Декоратор @Query вказує, що це саме запит, а в дужках вказується, який тип даних повинен повертатися, в даному випадку це UserType, тобто тип користувача. Потім вказується назва запиту (user), а вже в дужках вказуються аргументи, до кожного аргументу необхідно застосувати декоратор @Args, потім в дужках після нього вказати назву. В самій функції викликається метод getUser з userService (той що в прикладі вище), у якості аргументу передаємо аргумент, який отримали в цій функції і повертаємо результат виконання функції з сервісу. Для мутацій все аналогічно, лише замість декоратора @Query до функції застосовується декоратор @Mutation.

3.2.3 Post Module

Цей модуль для роботи з оголошенням. Загальний принцип такий самий як і з модулем користувачів. Звернути увагу потрібно лише на окремі функції (Рис.3.10).

```
async getPagePosts(take: number, skip: number, filters: GetPostsFilters) {
  const options: FindManyOptions<Post> = {
    skip,
    take,
    order: {},      You, 11 hours ago • Uncommitted changes
    where: {},
  };
  if (filters.city) options.where['city'] = filters.city;
  if (filters.periodOfTime)
    options.where['periodOfTime'] = filters.periodOfTime;
  if (filters.region) options.where['region'] = filters.region;
  if (filters.countOfPlaces !== undefined)
    options.where['countOfPlaces'] = { $gte: filters.countOfPlaces };
  if (filters.type) options.where['type'] = filters.type;
  if (filters.clientCategories && filters.clientCategories.length !== 0) {
    options.where['clientCategories'] = { $all: filters.clientCategories };
  }
  const [result] = await this.postRepository.findAndCount(options);
  return result;
}
```

Рисунок 3.10 – Код функції getPagePosts

Ця функція отримує у якості аргументів кількість постів, яку потрібно отримати та кількість постів, яку потрібно пропустити, а також фільтри, які необхідно застосувати. Тобто ця функція повертає пости для 1 сторінки із заданими фільтрами.

Також варто розглянути частину методу для створення посту, а саме код, який відповідає за збереження зображень (Рис.3.11).

```
const id = uuid();
const { createReadStream, filename } = await photoUrl;
const name = `${id}${filename}`;       You, 1 second ago • Uncommitted chan
const promise = new Promise<string>(async (resolve, reject) => {
  createReadStream()
    .pipe(createWriteStream(`D:/web/BachelorWork/Client/public/${name}`))
    .on('finish', () => resolve(name))
    .on('error', () =>
      reject(
        new HttpException('Could not save image', HttpStatus.BAD_REQUEST),
      ),
    );
});
const result = await promise;
```

Рисунок 3.11 – Код функції createPost, який відповідає за збереження зображення

У даному випадку використаний метод createWriteStream, який записує дані за вказаним URL, в даному випадку в публічну папку із зображеннями.

3.2.4 Auth Module

Даний модуль відповідає за формування jwt-токенів, їх перевірку та перевірку паролів. Варто розглянути 2 методи сервісу. Спочатку рисунок 3.12:

```
async validateUser(email: string, password: string): Promise<any> {
  const user = await this.userService.getUserByEmail(email);
  if (user) {
    if (await bcrypt.compare(password, user.password)) {
      delete user.password;
      return user;
    }
  }
  return null;
}
```

Рисунок 3.12 – Код функції validateUser

Даний метод отримує пошту та пароль як аргументи, потім знаходиться користувач за цією поштою і далі порівнюється пароль цього користувача та шифрований пароль, який передався як аргумент.

Другий метод відповідає за створення jwt-токену (Рис.3.13).

```
async generateUserCredentials(user: User) {
  const payload = {
    email: user.email,
    phone: user.phone,
    uid: user.uid,
    sub: user._id,
  };

  return {
    access_token: this.jwtTokenService.sign(payload),
  };
}
```

Рисунок 3.13 – Код функції generateCredentials

У даній функції як аргумент передається об’єкт типу User, потім з цих даних генерується payload для токену і потім повертається підписаний токен.

3.3 Створення клієнтської частини

Основна технологія для створення клієнтської частини веб-системи є React. Для типізації використано Typescript.

3.3.1 Структура проекту

При створенні проекту було використано ViteJS (Рис.3.14):

```
PS D:\web\BachelorWork> npm create vite@latest client -- --template react-ts
```

Рисунок 3.14 – Команда для створення проекту за допомогою Vite з налаштуваннями для шаблону React і Typescript

У результаті було створено кілька файлів з попереднім налаштування: package.json, package-lock.json, vite.config.ts, tsconfig.json (Рис.3.15).

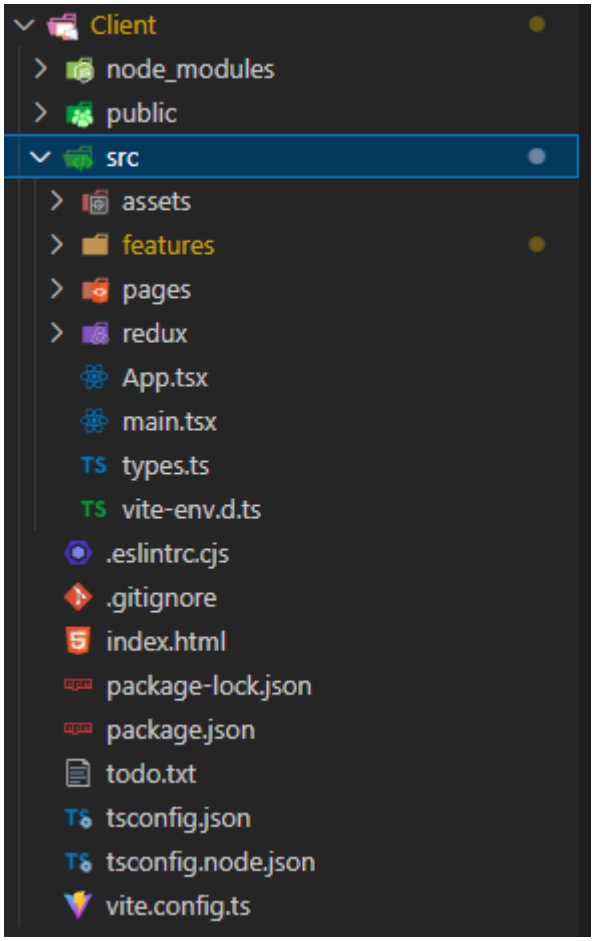

Рисунок 3.15 – Вигляд директорії проекту

Було створено папки проект: assets – для медіа файлів, features – для окремих фіч та функціоналу, pages – містить файли, що відповідають сторінкам додатку, redux – містить базові файли налаштування сховища Redux. Автоматично було створено точку входу проекту, а саме файл index.html (Рис.3.16):

```html
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Pink Willow</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Рисунок 3.16 – Index.html

Саме в div з id = root вмонтовуються всі компоненти. Це відбувається в файлі main.tsx, який також було згенеровано автоматично і доповнено потрібними налаштуваннями (Рис.3.17):

```tsx
const theme = createTheme({
  palette: {
    primary: {
      main: '#DA0B64',
    },
    secondary: {
      main: '#0bda80',
    },
  },
});

ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(

    <Provider store={store}>
      <ThemeProvider theme={theme}>
        <App />
      </ThemeProvider>
    </Provider>
)
```

Рисунок 3.17 – main.tsx

Тут було налаштовано тему всього додатку засобами MaterialUI, також, як було написано раніше, в root вмонтовуються компонент App, обгорнутий провайдером сховища Redux, цей компонент містить маршрутизацію всіх сторінок додатку, тобто фактично і є основним компонентом. Також створено файл types.tsx, який містить основні інтерфейси і типи даних, які є універсальними і використовуються у багатьох модулях проекту (Рис.3.18):

```
export interface IUser {
    email?: string,
    phone?: string,
    uid?: string,
    password?: string,
    posts?: IPost[],
}


export interface IPost {
    id?: string,
    photoUrl?: string,
    clientCategories?: Array<string>,
    city?: string,
    region?: string,
    countOfPlaces?: number,
    title?: string,
    description?: string,
    createdAt?: string,
    periodOfTime?: string,
    type?: string,
    user?: string,
}
```

Рисунок 3.18 – types.tsx

Тут представлені інтерфейси, які є відображеннями 2 головних сутностей роботи: посту та користувача.

3.3.2 Redux store

Файли для налаштування Redux store та всього, що необхідно для його ефективного використання, знаходиться в папці з відповідною назвою Redux (Рис.3.19):

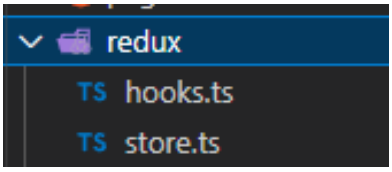

Рисунок 3.19 – Вміст папки redux

Як видно є лише 2 файли. Розглянемо основний файл – store.ts (Рис.3.20):

```
export const store = configureStore({
  reducer: {
    auth: authReducer,
    posts: postsReducer,
    profile: profileReducer,
  },
})

export type RootState = ReturnType<typeof store.getState>
export type AppDispatch = typeof store.dispatch
```

Рисунок 3.20 – Налаштування стору

У даному файлі створюється та експортується сховище, для його створення використані редьюсери, які описані в файлах зі словом slice в назві, кожен такий файл знаходиться в папці features і всередині папки з назвою конкретної фічи до, якої має безпосереднє відношення. Всі вони будуть розглянуті далі. Також з цього файлу експортуються типи для функції dispatch та тип загального стану додатку. У файлі hooks.ts зберігаються хуки, які будуть використані по всьому проекту (Рис 3.21):

```
export const useAppDispatch: () => AppDispatch = useDispatch
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector
```

Рисунок 3.21 – hooks.ts

Вони використовують вище описані типи і будуть використані замість звичайних useDispatch та useSelector , які надає бібліотека React-Redux.

3.3.3 Загальні компоненти додатку

Спочатку розглянемо App компонент, який відповідає за маршрутизацію всього проекту (Рис 3.22). Даний компонент містить маршрутизацію: за шляхом «/» рендериться компонент HomePage, за шляхом «/profile» компонент ProfilePage, за «post/:postId» - PostPage, postID – параметр url, який у даному випадку являє собою id поста. Усі компоненти згадані вище – сторінки додатку, про них буде йти мова далі. Також в App ми можемо побачити хук useEffect, який викликається лише 1 раз і він бере з локального сховища токен і зберігає його в редакс сховищі, це необхідно, щоб при перезавантаженні сторінки користувачу не потрібно було знову логінитися.

Рисунок 3.22 – App.tsx

Також варто звернути увагу на Header-component.tsx. У залежності від наявності jwt-токена в редакс сховищі будуть рендиритися різні компоненти: якщо токен є, тобто користувач авторизований, буде відображено кнопку виходу та кнопку створення оголошення, а якщо ні – то кнопку авторизації.

3.3.4 Модуль реєстрації/авторизації

Процес реєстрації та авторизації відбувається наступним чином – користувач натискає на відповідну кнопку і відкривається модальне вікно з відповідною формою. Варто зазначити, що для стилізації даного і всіх наступних модулів використані компоненти MUI. Після введення даних відбувається авторизація/реєстрація користувача. Найважливішим для розуміння принципу реєстрації/авторизації є файл register-slice.ts, всі файли пов’язані безпосередньо з цим функціоналом знаходяться в папці register всередині папки features. Розглянемо вміст файлу register-slice.ts (Рис.3.23).

Рисунок 3.23 – Створення слайсу всередині файлу register-slice.ts

Тут показано базовий принцип створення redux-toolkit слайсу. Кожен слайс відповідає за окрему частину глобального стану додатку, даний слайд наприклад відповідає за реєстрацію та авторизацію. Reducer з цього файлу було експортовано до фалу з глобальним сховищем.

Окремо описується початковий стан. Інтерфейс для нього винесено в окремий файл з типами, які відносяться до цієї фічі. Потім безпосередньо

створюється слайс за допомогою відповідної функції. Як параметри передаються ім’я слайсу, початковий стан та ред’юсери: у даному випадку їх 3. ToggleLoading – при діспатчі змінює значення поля loading стану слайсу на протилежний, setToken – встановлює токен, який передано в payload, logout – очищює локальне сховище від токена та встановлює значення токена в стані як null. В кінці файлу експортуються action для кожного ред’юсеру, їх генерує сам Redux Toolkit (Рис.3.24):

```
export const { toggleLoading, setToken, logout } = authSlice.actions
```

Рисунок 3.24 – Експорт actions із слайсу register-slice.ts

Також описуються та експортуються селектори для отримання окремих значень стану (Рис.3.25):

```
export const selectRegisterLoading = (state: RootState) => state.auth.loading
export const selectRegisterSuccess = (state: RootState) => state.auth.success
export const selectToken = (state: RootState) => state.auth.token
```

Рисунок 3.25 – Експорт селекторів із слайсу register-slice.ts

Вони використовуються, коли необхідно отримати відповідне значення стану в будь-якому місці додатку. Як значення за замовчення експортується AuthSlice.Reducer, саме він і імпортується в файлі з глобальним сховищем редаксу. Частина цього файлу, яка відповідає за процес реєстрації та авторизації це – AsyncThunk – функції, які використовуються для виконання асинхронних операцій, в нашому випадку запитів на сервер. У цьому файлі їх 2 – для авторизації та реєстрації. Спершу розглянемо для реєстрації (Рис.3.26). Для створення використовуються відповідна функція, яка приймає назву та функцію-коллбек. Функція в даному випадку за допомогою бібліотеки axios надсилає Post запит на сервер в якому надсилає GraphQL мутацію для реєстрації, як змінні в запит передаються значення аргументу, який приймає

функція-коллбек. Функція очікує на результат виконання промісу, який надсилає запит і потім повертає результат:

```
export const register = createAsyncThunk(
  "auth/register",
  async ( userData: IUser ) => {
    const response = await axios.post("http://localhost:4000/graphql", {
      query: `
        mutation createUser($email: String!, $phone: String!, $password: String!){
          createUser(createUserInput: {
            email: $email,
            phone: $phone,
            password: $password
          }){
            access_token
          }
        }
        `,
      variables: {
        email: userData.email,
        phone: userData.phone,
        password: userData.password
      }
    });
    return response.data.data.createUser;
  }
);
```

Рисунок 3.26 – Async thunk для реєстрації із слайсу register-slice.ts

Для подальшої обробки даних в слайсі визначаються екстра ред’юсери (Рис.3.27):

```
extraReducers: (builder) => {
  builder.addCase(register.pending, (state) => {
    state.loading = true;
  })
  builder.addCase(register.fulfilled, (state, action) => {
    state.token = action.payload.access_token;
    if(state.token !== null){
      window.localStorage.setItem('token', state.token)
    }
    state.loading = false;
    state.success = true;
  })
```

Рисунок 3.27– Екстра ред’юсери для обробки даних отриманих після реєстрації

Для кожного стану виконання функції, а саме: pending – означає, що функція в процесі виконання, обробки асинхронної операції; rejected – означає, що функція повернула помилку; fulfilled – означає, що функція виконалась повністю, визначається окрема поведінка. В даному випадку під виконання функції значення поля loading стає істинним і в додатку в цей час відображається loader-component, щоб показити користувачу, що реєстрації в процесі. Для виконаного стану функції вказано, що дані, які повернув запит заносяться в поле token, loading стає хибним, success стає істинним і якщо токен не null, то відповідне значення заноситься в localStorage. Це весь процес реєстрації – відправлення запиту і отримання токену із серверу і його подальше зберігання. Принцип авторизації аналогічний, різниця лише в GraphQL мутації.

3.3.5 ProfilePage

Сторінка профілю містить основу інформацію про користувача та перелік всіх створених ним постів. Файл profile-slice.ts містить ред'юсери для зміни значення поля loading та для отримання з jwt-токена даних про користувача і запис їх у відповідні поля (Рис.3.28):

```
decodeToken: (state, action) => {
  const token = action.payload;
  if(token !== null){
    const userData: IUser = jwt_decode(token)
    state.user.email = userData.email;
    state.user.phone = userData.phone;
    state.user.uid = userData.uid;
  }
}
```

Рисунок 3.28 – Ред'юсер для декодування jwt-токену в profile-slice.ts

Для цього застосовується функція jwt_decode(token); Asynk Thunk в цьому модулі одна – для отримання всіх постів для поточного користувача. Для цього в ній відправляється відповідний запит до сервера (query).

3.3.6 Модуль пов’язаний з постами

Розглянемо спочатку частину, що відповідає за створення постів, вона доступна лише після авторизації. Форма відкривається в модальному вікні після натискання відповідної кнопки на хедері. Після натиснення кнопки «Створити» на формі, діспатчиться async thunk, який відповідає за відправку даних з форми, які передаються як параметр, на сервер (Рис.3.29):

```
export const createPost = createAsyncThunk(
  "posts/createPost",
  async ( props: createPostProps ) => {
    const response = await axios.post("http://localhost:4000/graphql", {
      query: `
      mutation createPost($photoUrl: String!, $clientCategories: [String!]
        createPost(createPostInput: {
          photoUrl: $photoUrl,
          clientCategories: $clientCategories,
          city: $city,
          region: $region,
          countOfPlaces: $countOfPlaces,
          title: $title,
          description: $description,
          periodOfTime: $periodOfTime,
          type: $type,
          user: $user,
        }){
          city,
          countOfPlaces
        }
      }
      `,
      variables: {
        photoUrl: props.data.photoUrl,
        clientCategories: props.data.clientCategories,
        city: props.data.city,
        region: props.data.region,
        countOfPlaces: props.data.countOfPlaces,
        title: props.data.title,
        description: props.data.description,
        periodOfTime: props.data.periodOfTime,
        type: props.data.type,
        user: props.data.user,
      }
    },
    );
    return response.data.data.createPost;
  }
);
```

Рисунок 3.29 – Async thunk для створення посту в post-slice.ts

Такий вигляд має ця функція. Вона надсилає GraphQL мутацію. Розглянемо тепер відображення всіх постів. Сітка з постами виконана за допомогою MUI Grid – який імплементує Grid – розмітку CSS для позиціонування тегів. Сітка знаходиться на головній сторінці і доступна для перегляду навіть не авторизованим користувачам. Сітка має наступний код компоненти (Рис.3.30):

```
const CardsGrid = () => {

   const countOfItemsOnPage = useAppSelector(selectCountOfItemsOnPage);
   const page = useAppSelector(selectPage);
   const posts = useAppSelector(selectPosts);
   const loading = useAppSelector(selectPostsLoading);
   useFetchInitialPosts({page, countOfItemsOnPage});
   useInfiniteScrollForPosts({page, countOfItemsOnPage, loading});

   if(loading)
      return <Loader/>;

   return (
      <Grid container spacing={3}>
         {posts.map((item) => <Grid item xs={2} key={item.id}>
            <PostCard post={item}/>
         </Grid>)}
      </Grid>
   );
}

export default CardsGrid;
```

Рисунок 3.30 – CardsGrid компонент

Як видно тут використано 2 користувацьких хуки. Розглянемо їх більш детально, вони винесені в папці цієї фічі в окреми файл hooks.ts (Рис.3.31):

```
export const useFetchInitialPosts = (props: TUseFetchInitialPostsProps) => {
   const filters = useAppSelector(selectPostsFilters);
   const {page, countOfItemsOnPage} = props;
   const dispatch = useAppDispatch();
   useEffect(() => {
      dispatch(getPosts({take: countOfItemsOnPage, skip: (page - 1) * countOfItemsOnPage, filters}));
   },[filters])
}

export const useInfiniteScrollForPosts = (props: TUseInfiniteScrollForPosts) => {
   const {page, countOfItemsOnPage, loading} = props;
   const filters = useAppSelector(selectPostsFilters);
   const dispatch = useAppDispatch();
   const handleScroll = () => {
      if (window.innerHeight + document.documentElement.scrollTop !== document.documentElement.offsetHeight || loading) {
        return;
      }
      dispatch(getPosts({take: countOfItemsOnPage, skip: (page - 1) * countOfItemsOnPage, filters}));
    };

    useEffect(() => {
      window.addEventListener('scroll', handleScroll);
      return () => window.removeEventListener('scroll', handleScroll);
    }, [loading, filters]);
}
```

Рисунок 3.31 – Користувацькі хуки для отримування постів

Описано 2 користувацьких хуки: useFetchInitialPosts для отримання початкових постів та useInfiniteScrollForPosts для отримання постів на сторінці та реалізації нескінченного скролу, тобто коли користувач дійде до останнього посту сторінки то хук спрацює і здіспатчить async thunk, який надсилає запит на отримання постів на сторінці. Для обох хуків при отриманні постів застосовуються фільтри при надсиланні запита. В обох хуках використано наступну функцію для отримання постів (Рис.3.32):

```
export const getPosts = createAsyncThunk(
  "posts/getPosts",
  async ( data: getPostsProps ) => {
    const response = await axios.post("http://localhost:4000/graphql", {
      query: `
        query postsOnPage($take: Float!, $skip: Float!, $filters: GetPostsFilters!){
          postsOnPage(take: $take, skip: $skip, filters: $filters){
            id,
            photoUrl,
            clientCategories,
            city,
            region,
            countOfPlaces,
            title,
            description,
            createdAt,
            periodOfTime,
            type,
            user{
              email,
              phone
            }
          }
      }
      `,
      variables: {
        take: data.take,
        skip: data.skip,
        filters: data.filters,
      },
    });
    return response.data.data.postsOnPage;
  }
);
```

Рисунок 3.32 – Async thunk для отримування постів на сторінці з заданими фільтрами

Дана функція надсилає запит, в якому передаються значення take – кількість постів, яку потрібно отримати, skip – кількість постів, яку необхідно пропустити в базі даних (це необхідно, щоб отримувати пости для конкретної сторінки), filters – фільтри, за якими пости обираються з бд. Вигляд ред'юсера для цієї async thunk (Рис.3.33).

```
extraReducers: (builder) => {
  builder.addCase(getPosts.fulfilled, (state, action) => {
    state.posts = [...state.posts, ...action.payload];
    state.loading = false;
    state.success = true;
    state.page++;
  })
```

Рисунок 3.33 – Екстра ред’юсер для обробки отриманих постів на сторінці з заданими фільтрами

Пости які повертає сервер додаються до попередньо отриманих, номер сторінки збільшується на 1. Це також частина реалізації нескінченного скролу. Таким чином кожного разу, коли користувач доходить до останнього поста, надсилається новий запит на отримання постів для наступної сторінки, отримані в результаті цього запиту пости додаються до попередніх і користувач бачить на 1 сторінку постів більше.

Для даного модуля файл з типами виявився найбільшим з усіх аналогічних (Рис.3.34). В цьому файлі описані інтерфейси для фільтрів та для стану слайсу. Описані типи даних для props хуків описаних раніше та компоненти, яка представляє 1 пост.

```
export interface IPostsState {
    posts: IPost[],
    page: number,
    countOfItemsOnPage: number,
    loading: boolean,
    success: boolean,
    error: Error | null,
    filters: IFilters,
}

export type TUseFetchInitialPostsProps = {
    page: number;
    countOfItemsOnPage: number;
}
export type TUseInfiniteScrollForPosts = {
    page: number;
    countOfItemsOnPage: number;
    loading: boolean;
}

export type TPostCardProps = {
    post: IPost;
}

export interface IFilters {
    clientCategories?: Array<string>,
    city?: string,
    region?: string,
    countOfPlaces?: number,
    periodOfTime?: string,
    type?: string,
}
```

Рисунок 3.34 – types.ts для модуля з постами

Задання фільтрів виконується за допомогою модального вікна, яке викликається натисканням відповідної кнопки на домашній сторінці. Після натиснення submit-кнопки форми, в state встановлюються значення фільтрів. А оскільки вище розглянуті хуки для отримання постів мали в пропсах

значення фільтрів, то при їх зміні, відбувається ререндер і відповідні хуки викликаються знову з використанням вже нових фільтрів.

Сторінка окремого поста лише відображає дані розташовані за допомогою MUI Grid.

3.4 Тестування готової веб-системи

Тестування системи здійснено вручну, шляхом проходження всіх основних сценарій користування додатком та із залучення всіх можливостей розробленої системи.

3.4.1 Тестування серверу

Apollo GraphQL надає зручну можливість для тестування запитів та мутацій, а саме GraphQL Playground. Саме через нього і відбулося тестування. Цей інструмент автоматично генерує документацію та відображає схему в зручному вигляді (Рис.3.35, 3.36).

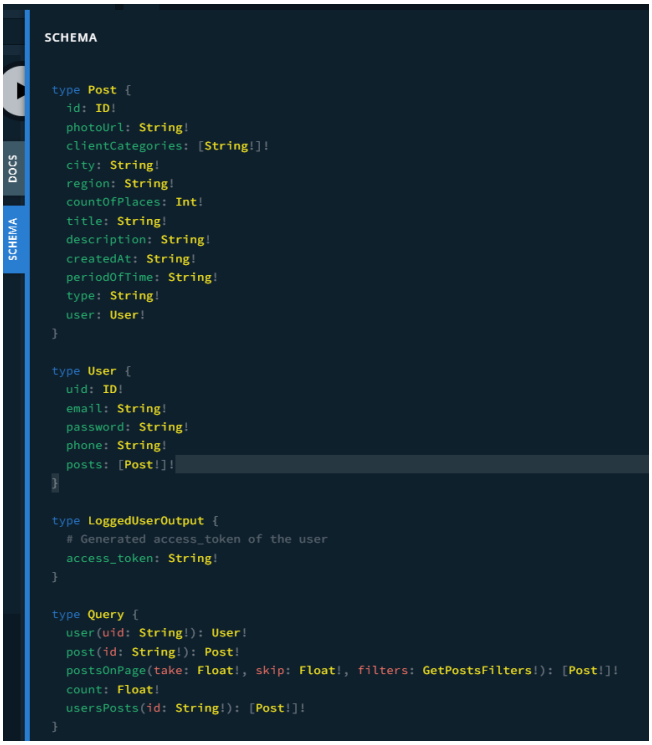

Рисунок 3.35 – Схема (Schema) згенерована GraphQL Playgrounds

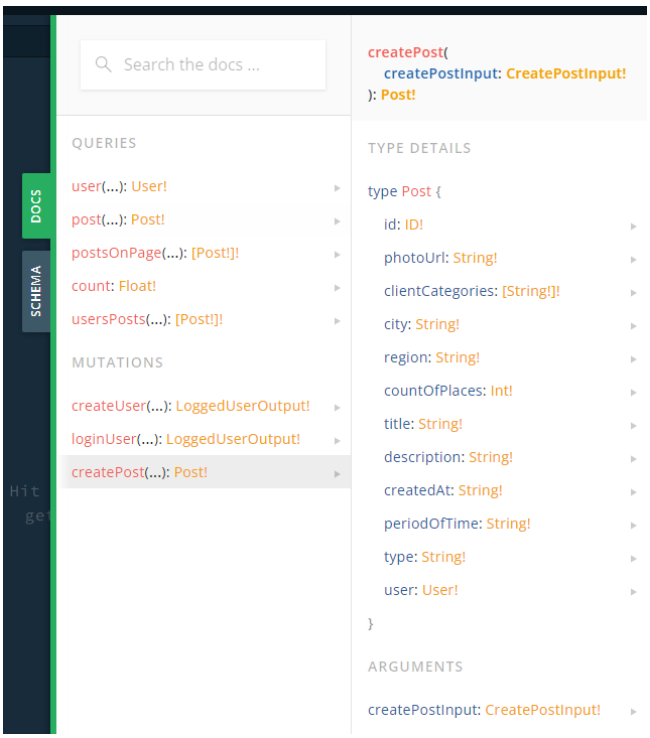

Рисунок 3.36 – Документація згенерована GraphQL Playgrounds

Спочатку було протестовано модуль, який відповідає за користувачів. Мутація на створення користувача (виконується під час реєстрації). Запит виконався успішно і сервер повернув jtw-токен для новоствореного користувача (Рис.3.37).

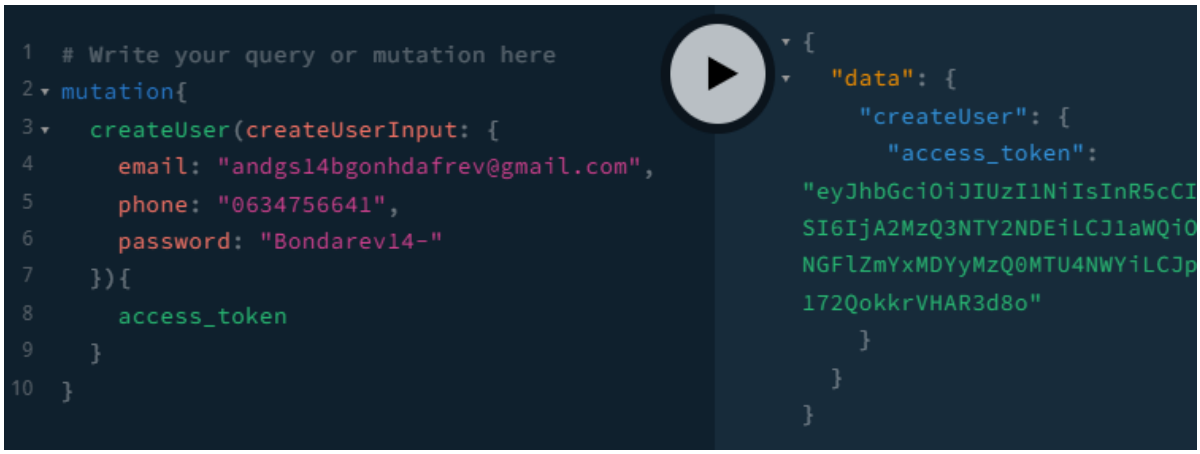

Рисунок 3.37 – Запит на мутацію для створення користувача і відповідь повернена сервером

Успішно перевірено мутацію для авторизації користувача (Рис.3.38).

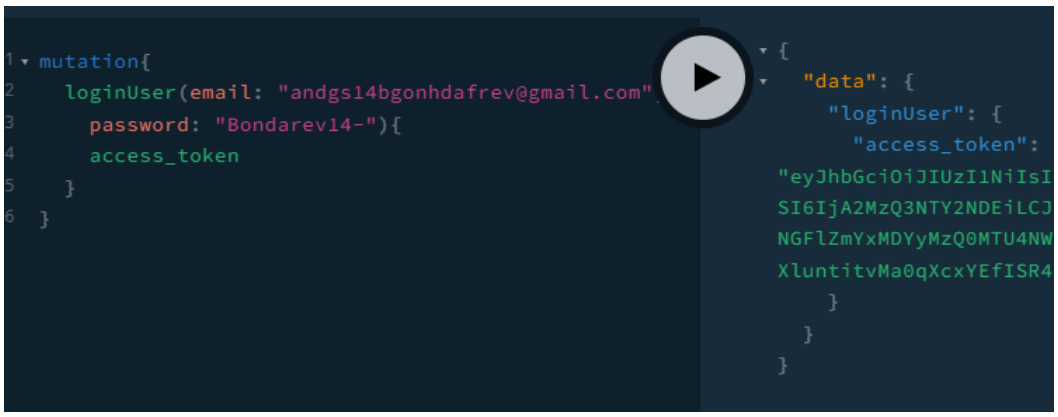

Рисунок 3.38 – Запит на мутацію для логіну користувача і відповідь повернена сервером

Для мутації на створення посту необхідно також задати хедер “Authorization” та передати в нього наступне значення “Bearer access_token”. Для даної мутації було використано додаток Altair для того, щоб була можливість завантажити зображення (Рис.3.39, 3.40).

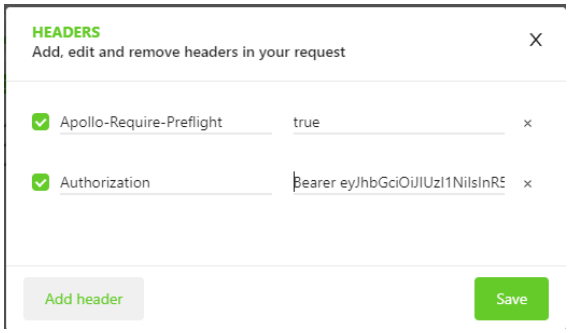

Рисунок 3.39 – Встановлення хедерів для запиту

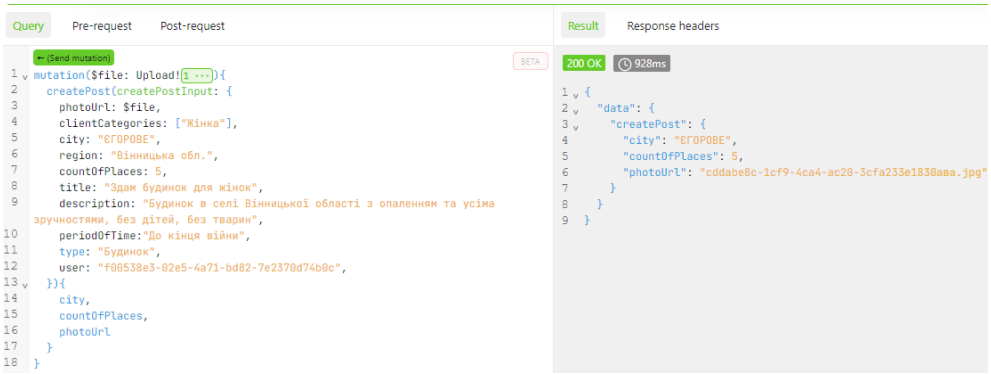

Рисунок 3.40 – Виконання запиту для мутації створення посту і відповідь сервера

Результат перевірки запиту на отримання всіх постів користувача (Рис.3.41).

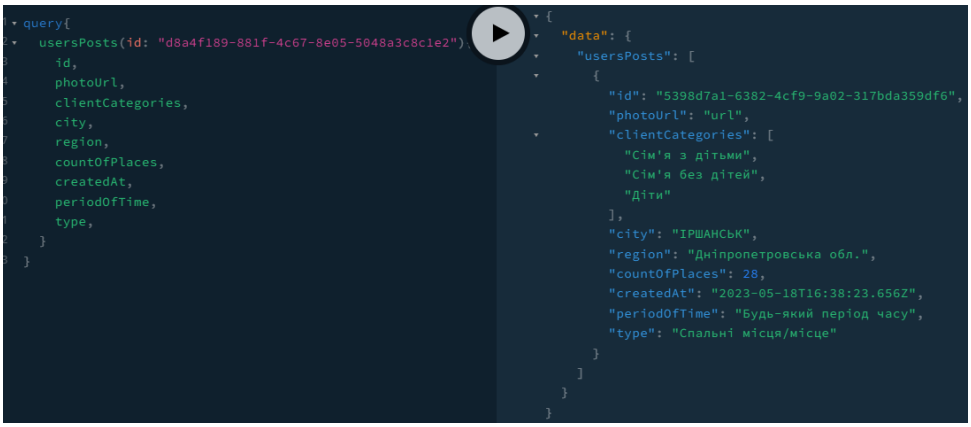

Рисунок 3.41 – Виконання запиту для отримання постів користувача і відповідь сервера

Результат перевірки запиту на отримання постів на сторінці з заданими фільтрами (Рис.3.42):

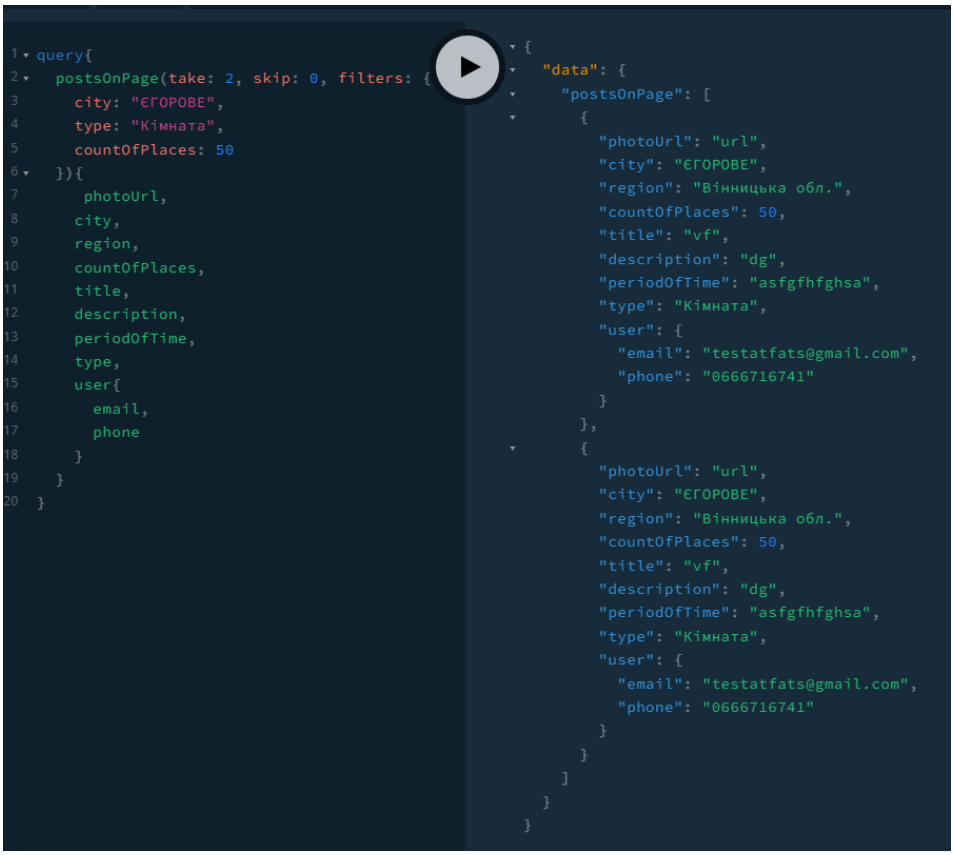

Рисунок 3.42 – Виконання запиту для отримання постів на сторінці із заданими фільтрами і відповідь сервера

3.4.2 Тестування клієнтської частини

Тестування клієнтської частини починалось із домашньої сторінки (Рис.3.43).

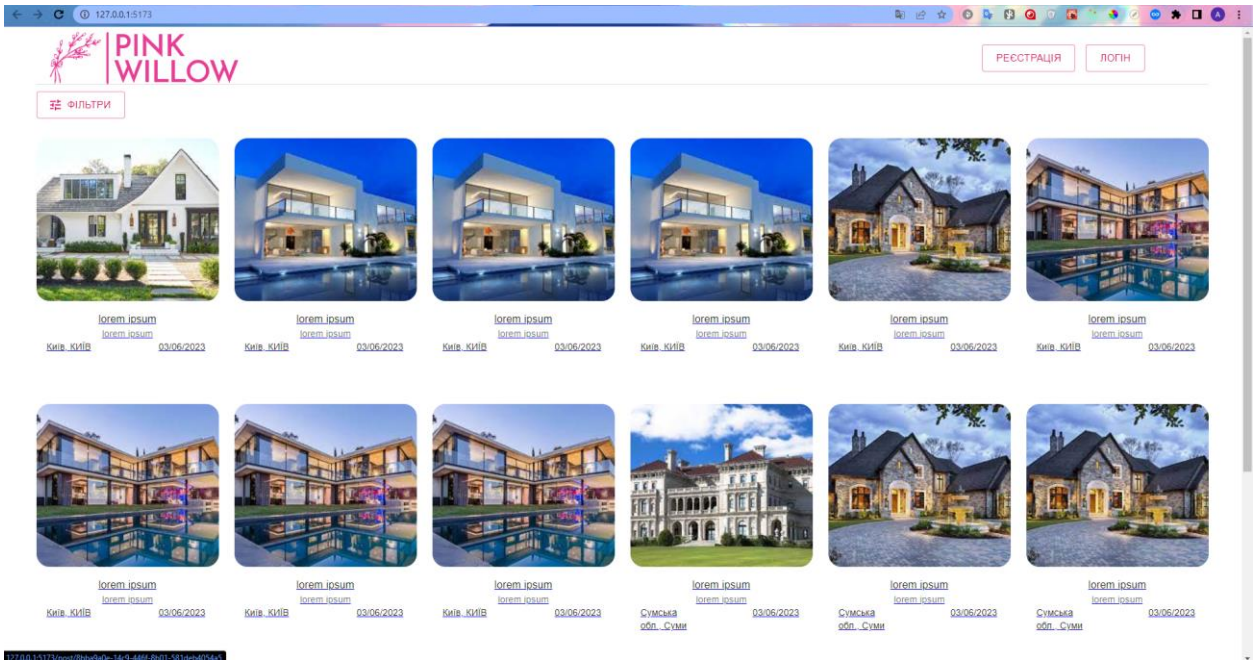

Рисунок 3.43 – Домашня сторінка

Кнопки реєстрації та логіну знаходяться на хедері і при натисненні на них відкриваються модальні вікна з відповідними формами (Рис.3.44, 3.45).

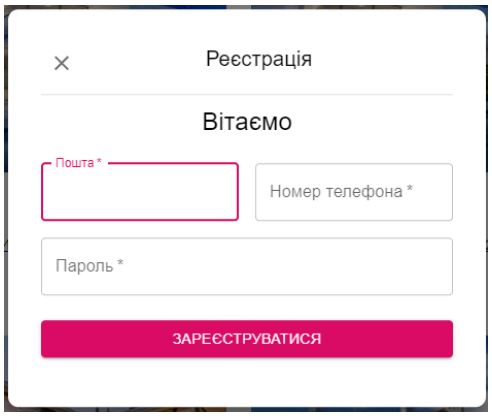

Рисунок 3.44 – Форма реєстрації

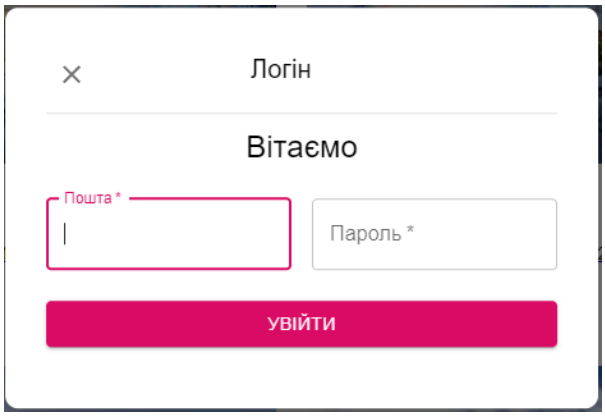

Рисунок 3.45 – Форма авторизації

На головній сторінці знаходиться кнопка Фільтри, натискання на яку відкриває модальне вікно в якому можна задати необхідні фільтри (Рис.3.46).

Для тестування було обрано наступні фільтри: Категорії клієнтів: Чоловіки, Період Часу: До кінця війни, Кількість місць від 2, Тип: будинок, в Сумській області, місто не задане (Рис.3.47). Також окремо було відфільтровано всі пропозиції в Києві (Рис.3.48).

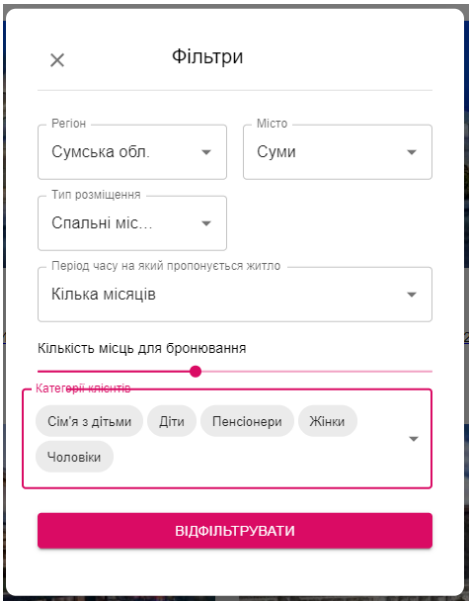

Рисунок 3.46 – Форма для задання фільтрів

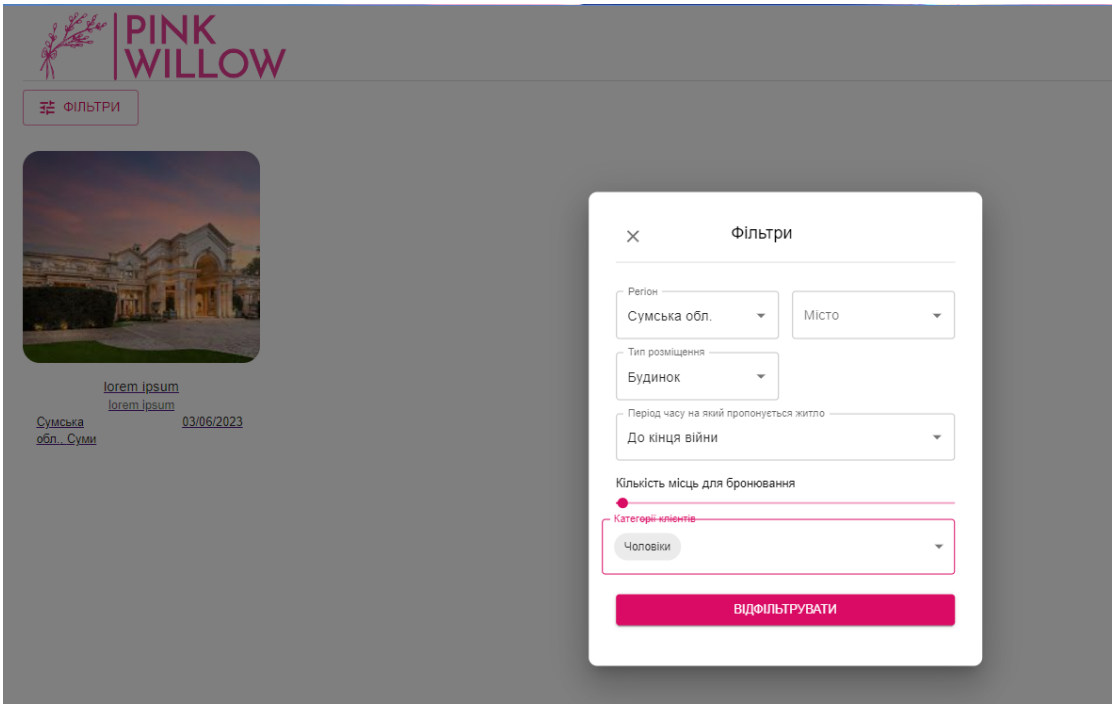

Рисунок 3.47 – Фільтрація за всіма параметрами окрім міста

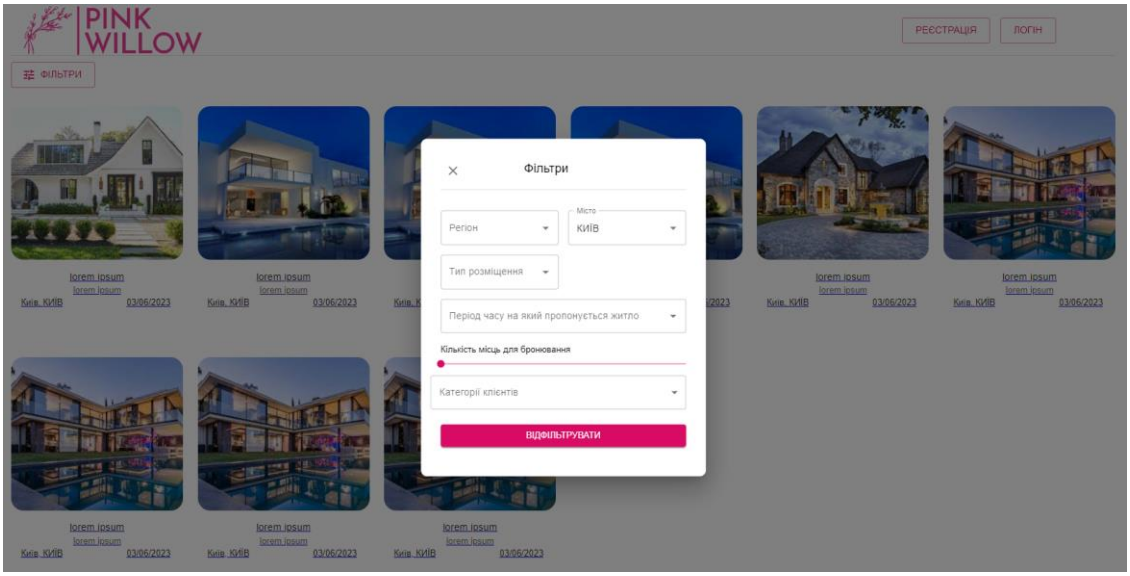

Рисунок 3.48 – Фільтрація по місту Київ

Шляхом натискання на картку з постом на головній сторінці або сторінці профілю відбувається переадресація на сторінку поста з повною інформацією (Рис.3.49).

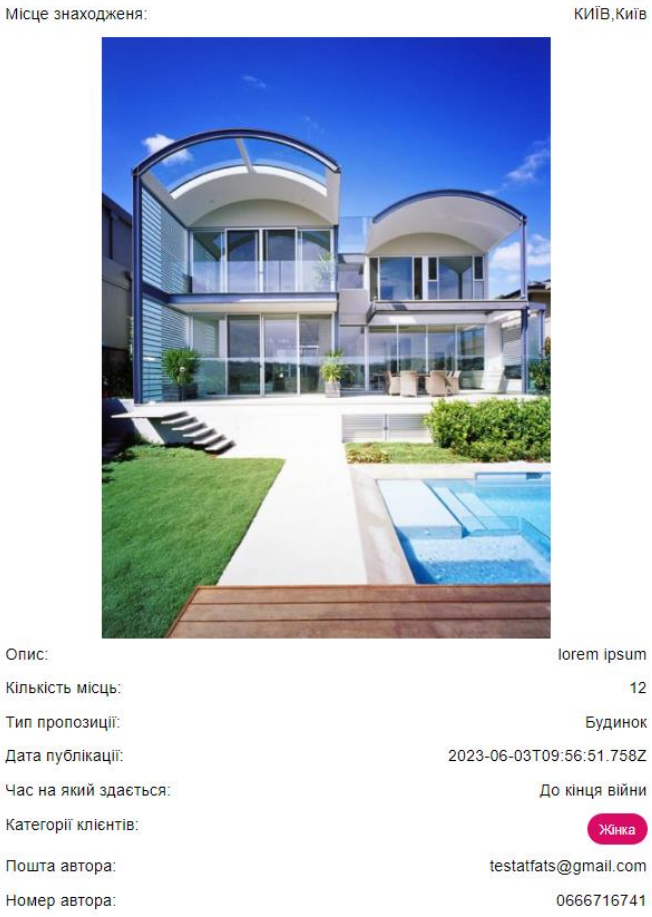

Рисунок 3.49 – Сторінка поста

Потім було перевірено форму створення, яка відкривається при натисканні кнопки на хедері, яка доступна, якщо користувач авторизувався (Рис 3.50, 3.51).

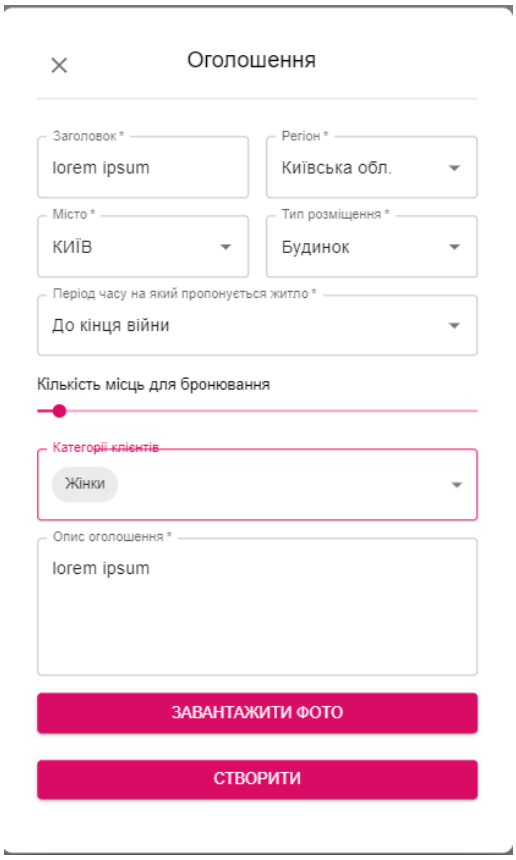

Рисунок 3.50 – Форма створення поста

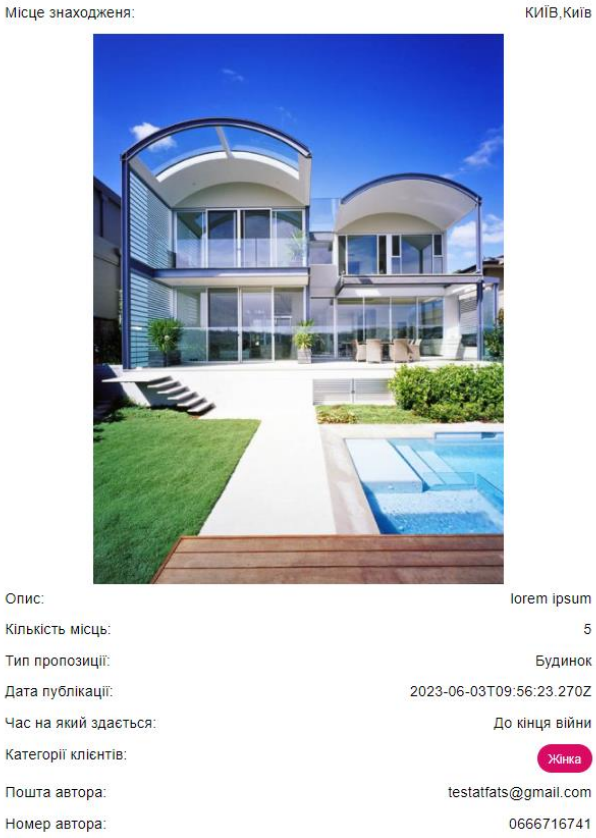

Рисунок 3.51 – Створений пост

Далі було перевірено вигляд сторінки профіля користувача, який відкривається при натисканні на значок аватара справа згори на хедері (Рис.3.52).

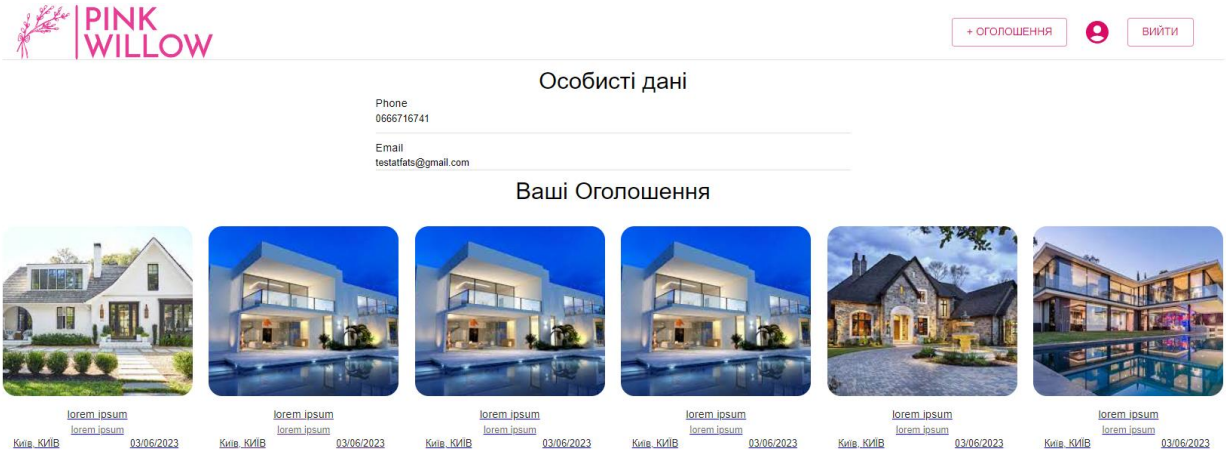

Рисунок 3.52 – Профіль користувача

ВИСНОВКИ

У ході виконання випускної кваліфікаційної роботи обгрунтовано актуальність теми у зв’язку із війною розв’язаною РФ. Було сформовано постановку задачі. Відповідно до неї було проаналізовано додатки-аналоги, було обрано технології для серверної та клієнтської частин системи. Було проведено літературний огляд обраних технологій.

Потім було спроектовано базу даних, визначено основні сутності та варіанти використання системи, створено діаграму використань.

Було створено та описано виконання та процес, методи, підходи, паттерни, ключові моменти в розробці додатку.

Було розроблено зручний і зрозумілий інтерфейс з простим, але найважливішим функціоналом. Було проведене ручне тестування шляхом виконання основних дій потенційного користувача.

Даний додаток допоможе бажаючим протягнути руку допомоги знайти тих хто цього потребує і навпаки допоможе постраждалим від РФ знайти безкоштовно тимчасове житло.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Wikipedia: Javascript: веб-сайт. URL: https://uk.wikipedia.org/wiki/JavaScript (дата звернення 01.05.2023)

2. Чому JavaScript — перспективна мова програмування? Поради початківцям: веб-сайт. URL: https://dou.ua/forums/topic/35184/ (дата звернення 31.05.2023)

3. React Documentation: веб-сайт. Url: https://uk.legacy.reactjs.org/ (дата звернення 01.05.2023)

4. ТОП навичок React JS розробника: веб-сайт. Url: https://nt.ua/blog/top-react-js-developer-skills (дата звернення 31.05.2023)

5. Understanding Redux: A tutorial with examples: веб-сайт. Url: https://blog.logrocket.com/understanding-redux-tutorial-examples/ (дата звернення 01.05.2023)

6. What is redux: веб-сайт. Url: https://www.freecodecamp.org/news/what-is-redux-store-actions-reducers-explained/ (дата звернення 01.05.2023)

7. Getting Started with Redux Toolkit: веб-сайт. Url: https://redux-toolkit.js.org/introduction/getting-started (дата звернення 04.05.2023)

8. Material UI - Overview: веб-сайт. Url: https://mui.com/material-ui/getting-started/overview/ (дата звернення 04.05.2023)

9. Brander: веб-сайт. Url: https://brander.ua/technologies/nodejs (дата звернення 04.05.2023)

10. Wikipedia:Node.js: веб-сайт. Url: https://uk.wikipedia.org/wiki/Node.js (дата звернення 04.05.2023)

11. Wikipedia: GraphQL: веб-сайт. Url: https://uk.wikipedia.org/wiki/GraphQL (дата звернення 05.05.2023)

12. Seoblog: веб-сайт. Url: https://seoblog.org.ua/5581/ (дата звернення 04.05.2023)

13. NestJS Documentation: веб-сайт. Url: https://docs.nestjs.com/ (дата звернення 07.05.2023)

14. Introduction to JSON Web Tokens: веб-сайт. Url: https://jwt.io/introduction (дата звернення 07.05.2023)

15. Як побудувати JWT у Go: веб-сайт. Url: https://senior.ua/articles/yak-pobuduvati-jwt-u-go (дата звернення 07.05.2023)

16. MongoDB Atlas Documentation: веб-сайт. Url: https://www.mongodb.com/docs/atlas/ (дата звернення 09.05.2023)

17. Wikipedia: MongoDB: веб-сайт. Url: https://uk.wikipedia.org/wiki/MongoDB (дата звернення 09.05.2023)

18. Wikipedia: Діаграма прецедентів: веб-сайт. Url: https://uk.wikipedia.org/wiki/%D0%94%D1%96%D0%B0%D0%B3%D1%80%D0%B0%D0%BC%D0%B0_%D0%BF%D1%80%D0%B5%D1%86%D0%B5%D0%B4%D0%B5%D0%BD%D1%82%D1%96%D0%B2 (дата звернення 11.05.2023)

19. NestJS Documentation: Providers: веб-сайт. Url: https://docs.nestjs.com/providers (дата звернення 12.05.2023)

ДОДАТОК А

А.1 Серверна частина

Main.ts

```
import { NestFactory } from '@nestjs/core';
import { AppModule } from './app.module';
import { ValidationPipe } from '@nestjs/common';
import * as graphqlUploadExpress from 'graphql-
upload/graphqlUploadExpress.js';
async function bootstrap() {
  const app = await NestFactory.create(AppModule);
  app.useGlobalPipes(new ValidationPipe());
  app.use(graphqlUploadExpress({ maxFileSize: 1000000, maxFiles: 10 }));
  app.enableCors();
  await app.listen(4000);
}
bootstrap();
```

app.module.ts

```
import { Module } from '@nestjs/common';
import { GraphQLModule } from '@nestjs/graphql';
import { ApolloDriver, ApolloDriverConfig } from '@nestjs/apollo';
import { UserModule } from './user/user.module';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './user/user.entity';
import { PostModule } from './post/post.module';
import { Post } from './post/post.entity';

@Module({
  imports: [
    TypeOrmModule.forRoot({
      type: 'mongodb',
      url:
'mongodb+srv://Bondarev14:Bondarev14@bachelorcluster.pqyr14m.mongodb.net/?ret
ryWrites=true&w=majority',
      synchronize: true,
      useUnifiedTopology: true,
      entities: [User, Post],
      useNewUrlParser: true,
      logging: true,
    }),
    GraphQLModule.forRoot<ApolloDriverConfig>({
      driver: ApolloDriver,
      autoSchemaFile: true,
    }),
    UserModule,
    PostModule,
  ],
})
export class AppModule {}
```

autModule

auth.module.ts

```
import { forwardRef, Module } from '@nestjs/common';
import { JwtModule } from '@nestjs/jwt';
import { ConfigService } from '@nestjs/config';
import { UserModule } from 'src/user/user.module';
import { AuthService } from './auth.service';
import { ConfigModule } from 'src/config.module';
import { JwtStrategy } from './jwt.strategy';
```

```
@Module({
  imports: [
    ConfigModule,
    forwardRef(() => UserModule),
    JwtModule.registerAsync({
      useFactory: (configService: ConfigService) => ({
        secret: configService.get<string>('JWT_SECRET'),
        signOptions: { expiresIn: '24h' },
      }),
      inject: [ConfigService],
    }),
    forwardRef(() => UserModule),
  ],
  providers: [AuthService, JwtStrategy],
  exports: [AuthService],
})
export class AuthModule {}
```

auth.service

```
import { forwardRef, Inject, Injectable } from '@nestjs/common';
import { JwtService } from '@nestjs/jwt';
import * as bcrypt from 'bcrypt';
import { User } from 'src/user/user.entity';
import { UserService } from 'src/user/user.service';
@Injectable()
export class AuthService {
  constructor(
    @Inject(forwardRef(() => UserService))
    private userService: UserService,
    private jwtTokenService: JwtService,
  ) {}
  async validateUser(email: string, password: string): Promise<any> {
    const user = await this.userService.getUserByEmail(email);
    if (user) {
      if (await bcrypt.compare(password, user.password)) {
        delete user.password;
        return user;
      }
    }
    return null;
  }

  async generateUserCredentials(user: User) {
    const payload = {
      email: user.email,
      phone: user.phone,
      uid: user.uid,
      sub: user._id,
    };

    return {
      access_token: this.jwtTokenService.sign(payload),
    };
  }
}
```

Auth.type.ts

```
import { Field, ObjectType } from '@nestjs/graphql';

@ObjectType()
export class LoggedUserOutput {
  @Field(() => String, { description: 'Generated access_token of the user' })
  access_token: string;
}
```

Jwt.strategy.ts

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

@Injectable()
export class JwtStrategy extends PassportStrategy(Strategy, 'jwt') {
  constructor(private readonly configService: ConfigService) {
    super({
      jwtFromRequest: ExtractJwt.fromAuthHeaderAsBearerToken(),
      ignoreExpiration: false,
      secretOrKey: configService.get<string>('JWT_SECRET'),
    });
  }

  async validate(payload: any) {
    return { payload, userId: payload.sub };
  }
}
```

Post module

Post.module.ts

```
import { Module, forwardRef } from '@nestjs/common';
import { TypeOrmModule } from '@nestjs/typeorm';
import { Post } from './post.entity';
import { PostResolver } from './post.resolver';
import { PostService } from './post.service';
import { UserModule } from 'src/user/user.module';
@Module({
  imports: [TypeOrmModule.forFeature([Post]), forwardRef(() => UserModule)],
  providers: [PostResolver, PostService],
  exports: [PostService],
})
export class PostModule {}
```

post.entity.ts

```
import {
  Column,
  Entity,
  ManyToOne,
  ObjectIdColumn,
  OneToMany,
  PrimaryColumn,
} from 'typeorm';
import { User } from 'src/user/user.entity';

@Entity()
export class Post {
  @ObjectIdColumn()
  _id: string;

  @PrimaryColumn()
  id: string;

  @Column()
  photoUrl: string;

  @Column()
  clientCategories: string[];

  @Column()
  city: string;
```

```
  @Column()
  region: string;

  @Column()
  countOfPlaces: number;

  @Column()
  title: string;

  @Column()
  description: string;

  @Column()
  createdAt: string;

  @Column()
  periodOfTime: string;

  @Column()
  type: string;

  @Column()
  user: string;
}
```

post.input.ts

```
import { InputType, Field } from '@nestjs/graphql';
import { IsOptional, IsUrl } from 'class-validator';
import { Stream } from 'stream';
import * as GraphQLUpload from 'graphql-upload/GraphQLUpload.js';
export interface FileUpload {
  filename: string;
  mimetype: string;
  encoding: string;
  createReadStream: () => Stream;
}
@InputType()
export class CreatePostInput {
  @Field(() => GraphQLUpload, { nullable: true })
  @IsOptional()
  photoUrl?: Promise<FileUpload>;
  @Field((type) => [String])
  clientCategories: string[];
  @Field()
  city: string;
  @Field()
  region: string;
  @Field()
  countOfPlaces: number;
  @Field()
  title: string;
  @Field()
  description: string;
  @Field()
  periodOfTime: string;
  @Field()
  type: string;
  @Field()
  user: string;
}

@InputType()
export class GetPostsFilters {
```

```
  @Field((type) => [String], { nullable: true })
  clientCategories?: string[];
  @Field({ nullable: true })
  city?: string;
  @Field({ nullable: true })
  region?: string;
  @Field({ nullable: true })
  countOfPlaces?: number;
  @Field({ nullable: true })
  periodOfTime?: string;
  @Field({ nullable: true })
  type?: string;
}
```

post.resolver.ts

```
import {
  Args,
  Mutation,
  Parent,
  Query,
  ResolveField,
  Resolver,
} from '@nestjs/graphql';
import { PostType } from './post.type';
import { IPost, PostService } from './post.service';
import { CreatePostInput, GetPostsFilters } from './post.input';
import { Post } from './post.entity';
import { UserService } from 'src/user/user.service';
import { Inject, forwardRef, UseGuards, UseInterceptors } from
'@nestjs/common';
import { JwtAuthGuard } from 'src/auth/jwt-auth.guard';

export type postsWithCount = {
  data: [PostType];
  count: number;
};

@Resolver((of) => PostType)
export class PostResolver {
  constructor(
    private postService: PostService,
    @Inject(forwardRef(() => UserService))
    private userService: UserService,
  ) {}

  @Query((returns) => PostType)
  post(@Args('id') id: string) {
    return this.postService.getPost(id);
  }

  @Query((returns) => [PostType])
  postsOnPage(
    @Args('take') take: number,
    @Args('skip') skip: number,
    @Args('filters') filters: GetPostsFilters,
  ) {
    return this.postService.getPagePosts(take, skip, filters);
  }

  @Query((returns) => Number)
  count() {
    return this.postService.getPostsCount();
  }
```

```
  @Query((returns) => [PostType])
  usersPosts(@Args('id') id: string) {
    return this.postService.getUserPostsById(id);
  }

  @Mutation((returns) => PostType)
  createPost(@Args('createPostInput') createPostInput: CreatePostInput) {
    return this.postService.createPost(createPostInput);
  }

  @ResolveField()
  async user(@Parent() post: Post) {
    return this.userService.getUser(post.user);
  }
}
```

post.service.ts

```
import {
  HttpException,
  HttpStatus,
  Inject,
  Injectable,
  forwardRef,
} from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { Post } from './post.entity';
import {
  ArrayContains,
  FindManyOptions,
  FindOptionsWhere,
  MongoRepository,
  ObjectLiteral,
} from 'typeorm';
import { v4 as uuid } from 'uuid';
import { CreatePostInput, GetPostsFilters } from './post.input';
import { UserService } from 'src/user/user.service';
import { PostType } from './post.type';
import { MoreThanOrEqual, Like, MoreThan } from 'typeorm';
import { take, skip, async } from 'rxjs';
import { createWriteStream } from 'fs';
import { join } from 'path';
export interface IPost {
  id?: string;
  photoUrl?: string;
  clientCategories?: Array<string>;
  city?: string;
  region?: string;
  countOfPlaces?: number;
  title?: string;
  description?: string;
  createdAt?: string;
  periodOfTime?: string;
  type?: string;
  user?: string;
}
@Injectable()
export class PostService {
  constructor(
    @InjectRepository(Post) private postRepository: MongoRepository<Post>,
    @Inject(forwardRef(() => UserService))
    private userService: UserService,
  ) {}
```

```
  async createPost(createPostInput: CreatePostInput): Promise<Post> {
    const {
      photoUrl,
      clientCategories,
      city,
      region,
      countOfPlaces,
      title,
      description,
      periodOfTime,
      type,
      user,
    } = createPostInput;
    const id = uuid();
    let result;
    if (photoUrl !== null && photoUrl !== undefined) {
      const { createReadStream, filename } = await photoUrl;
      const name = `${id}${filename}`;
      const promise = new Promise<string>(async (resolve, reject) => {
        createReadStream()
          .pipe(createWriteStream(`D:/web/BachelorWork/Client/public/${name}`
))
          .on('finish', () => resolve(name))
          .on('error', () =>
            reject(
              new HttpException('Could not save image',
HttpStatus.BAD_REQUEST),
            ),
          );
      });
      result = await promise;
    }
    const post = this.postRepository.create({
      id,
      photoUrl: result || 'null',
      clientCategories,
      city,
      region,
      countOfPlaces,
      title,
      description,
      periodOfTime,
      type,
      createdAt: new Date().toISOString(),
      user,
    });
    await this.userService.assignPostToUser(user, post.id);
    return this.postRepository.save(post);
  }

  async getPost(id: string): Promise<Post> {
    return this.postRepository.findOneBy({
      id,
    });
  }

  async getAllPostsOfUser(postsIds: string[]): Promise<Post[]> {
    return this.postRepository.find({
      where: {
        id: { $in: postsIds },
      },
    });
  }
```

```
  async getPagePosts(take: number, skip: number, filters: GetPostsFilters) {
    const options: FindManyOptions<Post> = {
      skip,
      take,
      order: {},
      where: {},
    };
    if (filters.city) options.where['city'] = filters.city;
    if (filters.periodOfTime)
      options.where['periodOfTime'] = filters.periodOfTime;
    if (filters.region) options.where['region'] = filters.region;
    if (filters.countOfPlaces !== undefined)
      options.where['countOfPlaces'] = { $gte: filters.countOfPlaces };
    if (filters.type) options.where['type'] = filters.type;
    if (filters.clientCategories && filters.clientCategories.length !== 0) {
      options.where['clientCategories'] = { $all: filters.clientCategories };
    }
    const [result] = await this.postRepository.findAndCount(options);
    return result;
  }

  async getPostsCount() {
    return await this.postRepository.count();
  }

  async getUserPostsById(id: string): Promise<Post[]> {
    return this.postRepository.findBy({
      user: id,
    });
  }
}
```

post.type.ts

```
import { Field, ID, Int, ObjectType } from '@nestjs/graphql';
import { UserType } from 'src/user/user.type';

@ObjectType('Post')
export class PostType {
  @Field((type) => ID)
  id: string;
  @Field()
  photoUrl: string;
  @Field((type) => [String])
  clientCategories: string[];
  @Field()
  city: string;
  @Field()
  region: string;
  @Field((type) => Int)
  countOfPlaces: number;
  @Field()
  title: string;
  @Field()
  description: string;
  @Field()
  createdAt: string;
  @Field()
  periodOfTime: string;
  @Field()
  type: string;
  @Field((type) => UserType)
  user: string;
}
```

User module
User.entity.ts

```
import { Column, Entity, ObjectIdColumn, PrimaryColumn } from 'typeorm';

@Entity()
export class User {
  @ObjectIdColumn()
  _id: string;

  @PrimaryColumn()
  uid: string;

  @Column()
  email: string;

  @Column()
  password: string;

  @Column()
  phone: string;

  @Column()
  posts: string[];
}
```

User.input.ts

```
import { InputType, Field } from '@nestjs/graphql';
import { IsEmail, IsPhoneNumber, IsStrongPassword } from 'class-validator';

@InputType()
export class CreateUserInput {
  @IsEmail()
  @Field()
  email: string;
  @IsPhoneNumber('UA')
  @Field()
  phone: string;
  @IsStrongPassword()
  @Field()
  password: string;
}
```

User.module.ts

```
import { Module, forwardRef } from '@nestjs/common';
import { UserResolver } from './user.resolver';
import { UserService } from './user.service';
import { TypeOrmModule } from '@nestjs/typeorm';
import { User } from './user.entity';
import { PostModule } from 'src/post/post.module';
import { AuthModule } from 'src/auth/auth.module';
@Module({
  imports: [
    TypeOrmModule.forFeature([User]),
    forwardRef(() => PostModule),
    forwardRef(() => AuthModule),
  ],
  providers: [UserResolver, UserService],
  exports: [UserService],
})
export class UserModule {}
```

User.resolver.ts

```
import {
  Args,
  Mutation,
  Parent,
  Query,
  ResolveField,
  Resolver,
} from '@nestjs/graphql';
import { UserType } from './user.type';
import { UserService } from './user.service';
import { CreateUserInput } from './user.input';
import { PostService } from 'src/post/post.service';
import { Inject, forwardRef, UseGuards } from '@nestjs/common';
import { User } from './user.entity';
import { LoggedUserOutput } from 'src/auth/auth.type';

@Resolver((of) => UserType)
export class UserResolver {
  constructor(
    @Inject(forwardRef(() => PostService))
    private postService: PostService,
    private userService: UserService,
  ) {}
  @Query((returns) => UserType)
  user(@Args('uid') uid: string) {
    return this.userService.getUser(uid);
  }

  @Mutation((returns) => LoggedUserOutput)
  createUser(@Args('createUserInput') createUserInput: CreateUserInput) {
    return this.userService.createUser(createUserInput);
  }

  @Mutation(() => LoggedUserOutput)
  loginUser(@Args('email') email: string, @Args('password') password: string)
{
    return this.userService.loginUser(email, password);
  }

  @ResolveField()
  async posts(@Parent() user: User) {
    return this.postService.getAllPostsOfUser(user.posts);
  }
}
```

User.service.ts

```
import {
  HttpException,
  HttpStatus,
  Inject,
  Injectable,
  forwardRef,
} from '@nestjs/common';
import { InjectRepository } from '@nestjs/typeorm';
import { User } from './user.entity';
import { MongoRepository, Repository } from 'typeorm';
import { v4 as uuid } from 'uuid';
import { CreateUserInput } from './user.input';
import * as bcrypt from 'bcrypt';
import { AuthService } from 'src/auth/auth.service';
@Injectable()
export class UserService {
```

```
  constructor(
    @InjectRepository(User) private userRepository: MongoRepository<User>,
    @Inject(forwardRef(() => AuthService))
    private authService: AuthService,
  ) {}

  async createUser(createUserInput: CreateUserInput) {
    const { email, phone, password } = createUserInput;
    const isUserExists = await this.getUserByEmail(email);
    try {
      if (isUserExists) {
        throw new Error('CONFLICT, Email`s already used');
      } else {
        const saltOrRounds = 10;
        const hashPassword = await bcrypt.hash(password, saltOrRounds);
        const user = await this.userRepository.create({
          uid: uuid(),
          email,
          phone,
          password: hashPassword,
          posts: [],
        });
        await this.userRepository.save(user);
        return this.authService.generateUserCredentials(user);
      }
    } catch (err) {
      throw new HttpException(
        'CONFLICT, Email`s already used',
        HttpStatus.CONFLICT,
      );
    }
  }

  async loginUser(email: string, password: string) {
    const user = await this.authService.validateUser(email, password);
    if (!user) {
    } else {
      return this.authService.generateUserCredentials(user);
    }
  }

  async getUser(uid: string): Promise<User> {
    return this.userRepository.findOneBy({
      uid,
    });
  }

  async getUserByEmail(email: string): Promise<User> {
    return this.userRepository.findOneBy({
      email,
    });
  }

  async assignPostToUser(uid: string, postId: string): Promise<User> {
    const user = await this.userRepository.findOneBy({
      uid,
    });
    user.posts.push(postId);
    return this.userRepository.save(user);
  }
}
```

User.type.ts

```
import { Field, ID, ObjectType } from '@nestjs/graphql';
import { PostType } from 'src/post/post.type';

@ObjectType('User')
export class UserType {
  @Field((type) => ID)
  uid: string;
  @Field()
  email: string;
  @Field()
  password: string;
  @Field()
  phone: string;
  @Field((type) => [PostType])
  posts: string[];
}
```

Package.json

```
{
  "name": "BachelorWork",
  "version": "0.0.1",
  "description": "",
  "author": "",
  "private": true,
  "license": "UNLICENSED",
  "scripts": {
    "build": "nest build",
    "format": "prettier --write \"src/**/*.ts\" \"test/**/*.ts\"",
    "start": "nest start",
    "start:dev": "nest start --watch",
    "start:debug": "nest start --debug --watch",
    "start:prod": "node dist/main",
    "lint": "eslint \"{src,apps,libs,test}/**/*.ts\" --fix",
    "test": "jest",
    "test:watch": "jest --watch",
    "test:cov": "jest --coverage",
    "test:debug": "node --inspect-brk -r tsconfig-paths/register -r ts-
node/register node_modules/.bin/jest --runInBand",
    "test:e2e": "jest --config ./test/jest-e2e.json"
  },
  "dependencies": {
    "@apollo/server": "^4.6.0",
    "@nestjs/apollo": "^11.0.5",
    "@nestjs/common": "^9.0.0",
    "@nestjs/config": "^2.3.1",
    "@nestjs/core": "^9.0.0",
    "@nestjs/graphql": "^11.0.5",
    "@nestjs/jwt": "^10.0.3",
    "@nestjs/mongoose": "^9.2.2",
    "@nestjs/passport": "^9.0.3",
    "@nestjs/platform-express": "^9.0.0",
    "@nestjs/typeorm": "^9.0.1",
    "@types/bcrypt": "^5.0.0",
    "@types/mongodb": "^4.0.7",
    "@types/passport-jwt": "^3.0.8",
    "bcrypt": "^5.1.0",
    "class-transformer": "^0.5.1",
    "class-validator": "^0.14.0",
    "graphql": "^16.6.0",
    "graphql-tools": "^8.3.19",
    "graphql-upload": "^14.0.0",
    "mongodb": "^3.7.3",
    "mongoose": "^7.0.3",
    "passport": "^0.6.0",
```

```
    "passport-jwt": "^4.0.1",
    "reflect-metadata": "^0.1.13",
    "rxjs": "^7.2.0",
    "typeorm": "^0.3.14",
    "uuid": "^9.0.0"
  },
  "devDependencies": {
    "@nestjs/cli": "^9.0.0",
    "@nestjs/schematics": "^9.0.0",
    "@nestjs/testing": "^9.0.0",
    "@types/express": "^4.17.13",
    "@types/jest": "29.5.0",
    "@types/multer": "^1.4.7",
    "@types/node": "18.15.11",
    "@types/supertest": "^2.0.11",
    "@typescript-eslint/eslint-plugin": "^5.0.0",
    "@typescript-eslint/parser": "^5.0.0",
    "eslint": "^8.0.1",
    "eslint-config-prettier": "^8.3.0",
    "eslint-plugin-prettier": "^4.0.0",
    "jest": "29.5.0",
    "prettier": "^2.3.2",
    "source-map-support": "^0.5.20",
    "supertest": "^6.1.3",
    "ts-jest": "29.0.5",
    "ts-loader": "^9.2.3",
    "ts-node": "^10.0.0",
    "tsconfig-paths": "4.2.0",
    "typescript": "^4.7.4"
  },
  "jest": {
    "moduleFileExtensions": [
      "js",
      "json",
      "ts"
    ],
    "rootDir": "src",
    "testRegex": ".*\\.spec\\.ts$",
    "transform": {
      "^.+\\.(t|j)s$": "ts-jest"
    },
    "collectCoverageFrom": [
      "**/*.(t|j)s"
    ],
    "coverageDirectory": "../coverage",
    "testEnvironment": "node"
  }
}
```

А.2 Клієнтська частина

Vite.config.ts

```
import { defineConfig } from 'vite'
import react from '@vitejs/plugin-react'
import tsconfigPaths from 'vite-tsconfig-paths'
import path from 'path'

// https://vitejs.dev/config/
export default defineConfig({
  resolve: {
    alias: {
      src: '/src',
    },
  },
  plugins: [react(), tsconfigPaths()],

})
```

Package.json

```
{
  "name": "client",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "tsc && vite build",
    "lint": "eslint src --ext ts,tsx --report-unused-disable-directives --
max-warnings 0",
    "preview": "vite preview"
  },
  "dependencies": {
    "@emotion/react": "^11.10.8",
    "@emotion/styled": "^11.10.8",
    "@mui/icons-material": "^5.11.16",
    "@mui/material": "^5.12.3",
    "@reduxjs/toolkit": "^1.9.5",
    "axios": "^1.3.6",
    "jwt-decode": "^3.1.2",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-redux": "^8.0.5",
    "react-router-dom": "^6.10.0",
    "vite-tsconfig-paths": "^4.2.0"
  },
  "devDependencies": {
    "@types/node": "^18.16.1",
    "@types/react": "^18.0.28",
    "@types/react-dom": "^18.0.11",
    "@typescript-eslint/eslint-plugin": "^5.57.1",
    "@typescript-eslint/parser": "^5.57.1",
    "@vitejs/plugin-react": "^4.0.0-beta.0",
    "eslint": "^8.38.0",
    "eslint-plugin-react-hooks": "^4.6.0",
    "eslint-plugin-react-refresh": "^0.3.4",
    "typescript": "^5.0.2",
    "vite": "^4.3.0"
  }
}
```

Tsconfig.json

```
{
  "compilerOptions": {
    "target": "ESNext",
    "lib": ["DOM", "DOM.Iterable", "ESNext"],
    "module": "ESNext",
    "skipLibCheck": true,
    "baseUrl": ".",
    "paths": {
      "src/*": ["src/*"]
    },
    /* Bundler mode */
    "moduleResolution": "bundler",
    "allowImportingTsExtensions": true,
    "resolveJsonModule": true,
    "isolatedModules": true,
    "noEmit": true,
    "jsx": "react-jsx",

    /* Linting */
    "strict": true,
    "noUnusedLocals": true,
    "noUnusedParameters": true,
    "noFallthroughCasesInSwitch": true
  },
  "include": ["src"],
  "references": [{ "path": "./tsconfig.node.json" }]
}
```

Index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <link rel="icon" type="image/svg+xml" href="/vite.svg" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>Pink Willow</title>
  </head>
  <body>
    <div id="root"></div>
    <script type="module" src="/src/main.tsx"></script>
  </body>
</html>
```

Main.tsx

```
import ReactDOM from 'react-dom/client'
import App from 'src/App.tsx'
import { Provider } from 'react-redux'
import { store } from 'src/redux/store.ts'
import createTheme from '@mui/material/styles/createTheme'
import ThemeProvider from '@mui/material/styles/ThemeProvider'

const theme = createTheme({
  palette: {
    primary: {
      main: '#DA0B64',
    },
    secondary: {
      main: '#0bda80',
    },
  },
```

```
});

ReactDOM.createRoot(document.getElementById('root') as HTMLElement).render(

    <Provider store={store}>
      <ThemeProvider theme={theme}>
        <App />
      </ThemeProvider>
    </Provider>
)
```

App.tsx

```
import HomePage from "src/pages/home-page";
import {
  createBrowserRouter,
  RouterProvider,
} from "react-router-dom";
import ProfilePage from "./pages/profile-page";
import { useAppDispatch } from "./redux/hooks";
import { useEffect } from "react";
import { setToken } from "./features/register/register-slice";
import { decodeToken } from "./features/profile/profile-slice";
import PostPage from "./pages/post-page";
const router = createBrowserRouter([
  {
    path: "/",
    element: <HomePage/>,
  },
  {
    path: "/profile",
    element: <ProfilePage/>,
  },
  {
    path: "post/:postId",
    element: <PostPage />,
  },
]);

const App = () => {
  const dispatch = useAppDispatch();
  useEffect(() => {
    const token = localStorage.getItem('token');
    if(token !== null){
      dispatch(setToken(token));
      dispatch(decodeToken(token));
    }
  },[])
  return (
    <>
      <RouterProvider router={router} />
    </>
  )
}

export default App
```

Types.ts

```
export interface IUser {
   email?: string,
   phone?: string,
   uid?: string,
   password?: string,
```

```
   posts?: IPost[],
}

export interface IPost {
   id?: string,
   photoUrl?: string | File,
   clientCategories?: Array<string>,
   city?: string,
   region?: string,
   countOfPlaces?: number,
   title?: string,
   description?: string,
   createdAt?: string,
   periodOfTime?: string,
   type?: string,
   user?: string,
}
```

Redux/hooks.ts

```
import { useDispatch, useSelector } from 'react-redux'
import type { TypedUseSelectorHook } from 'react-redux'
import type { RootState, AppDispatch } from 'src/redux/store'

export const useAppDispatch: () => AppDispatch = useDispatch
export const useAppSelector: TypedUseSelectorHook<RootState> = useSelector
```

Redux/store.ts

```
import { configureStore } from '@reduxjs/toolkit'
import authReducer from 'src/features/register/register-slice'
import postsReducer from 'src/features/posts/posts-slice'
import profileReducer from 'src/features/profile/profile-slice'
export const store = configureStore({
  reducer: {
    auth: authReducer,
    posts: postsReducer,
    profile: profileReducer,
  },
})

export type RootState = ReturnType<typeof store.getState>
export type AppDispatch = typeof store.dispatch
```

pages/home-page.tsx

```
import Box from "@mui/material/Box/Box";
import CardsGrid from "src/features/posts/cards-grid/cards-grid-component";
import Filters from "src/features/posts/filters/filters-component";
import Header from "src/features/ui/header/header-component"

const HomePage = () => {
   return (
      <Box sx={{
         maxWidth: '1800px',
         margin: '0 auto'
      }}>
         <Header/>
         <Filters/>
         <CardsGrid/>
      </Box>
   );
}

export default HomePage;
```

pages/post-page.tsx

```
import { Box, Chip, Grid, Typography } from "@mui/material";
import Header from "src/features/ui/header/header-component";
import image from 'src/assets/house.jpg'
import { useParams } from "react-router-dom";
import { useAppSelector } from "src/redux/hooks";

const PostPage = () => {
   const {postId} = useParams();
   const post = useAppSelector((state) => state.posts.posts.find((post) =>
post.id === postId));
   return <Box sx={{
      maxWidth: '1800px',
      margin: '0 auto'
   }}>
      <Header/>
      <Grid container direction={"column"} alignItems={'center'}>
         <Grid item sx={{
            margin: '25px auto'
         }}>
            <Typography variant='h5'>{post?.title}</Typography>
         </Grid>
         <Grid sx={{
            width: '640px',
            marginBottom: '10px'
         }} container item justifyContent={'space-between'}>
            <Grid item><Typography variant='body1'>Місце
знаходженя:</Typography></Grid>
            <Grid item><Typography
variant='body1'>{post?.city},{post?.region}</Typography></Grid>
         </Grid>
         <Grid>
            <img src={image || post?.photoUrl} alt="" />
         </Grid>
         <Grid sx={{
            width: '640px',
            marginBottom: '10px'
         }} container item justifyContent={'space-between'}>
            <Grid item><Typography variant='body1'>Опис:</Typography></Grid>
            <Grid item><Typography
variant='body1'>{post?.description}</Typography></Grid>
         </Grid>
         <Grid sx={{
            width: '640px',
            marginBottom: '10px'
         }} container item justifyContent={'space-between'}>
            <Grid item><Typography variant='body1'>Кількість
місць:</Typography></Grid>
            <Grid item><Typography
variant='body1'>{post?.countOfPlaces}</Typography></Grid>
         </Grid>
         <Grid sx={{
            width: '640px',
            marginBottom: '10px'
         }} container item justifyContent={'space-between'}>
            <Grid item><Typography variant='body1'>Тип
пропозиції:</Typography></Grid>
            <Grid item><Typography
variant='body1'>{post?.type}</Typography></Grid>
         </Grid>
         <Grid sx={{
            width: '640px',
            marginBottom: '10px'
         }} container item justifyContent={'space-between'}>
            <Grid item><Typography variant='body1'>Дата
```

```
публікації:</Typography></Grid>
            <Grid item><Typography
variant='body1'>{post?.createdAt}</Typography></Grid>
         </Grid>
         <Grid sx={{
            width: '640px',
            marginBottom: '10px'
         }} container item justifyContent={'space-between'}>
            <Grid item><Typography variant='body1'>Час на який
здається:</Typography></Grid>
            <Grid item><Typography
variant='body1'>{post?.periodOfTime}</Typography></Grid>
         </Grid>
         <Grid sx={{
            width: '640px',
            marginBottom: '10px'
         }} container item justifyContent={'space-between'}>
            <Grid item><Typography variant='body1'>Категорії
клієнтів:</Typography></Grid>
            <Grid item>{post?.clientCategories.map((item) => <Chip
label={item} color="primary"/>)}</Grid>
         </Grid>
         <Grid sx={{
            width: '640px',
            marginBottom: '10px'
         }} container item justifyContent={'space-between'}>
            <Grid item><Typography variant='body1'>Пошта
автора:</Typography></Grid>
            <Grid item><Typography variant='body1'>{post.user!== undefined &&
post?.user.email}</Typography></Grid>
         </Grid>
         <Grid sx={{
            width: '640px',
            marginBottom: '10px'
         }} container item justifyContent={'space-between'}>
            <Grid item><Typography variant='body1'>Номер
автора:</Typography></Grid>
            <Grid item><Typography variant='body1'>{post.user!== undefined &&
post?.user.phone}</Typography></Grid>
         </Grid>
      </Grid>
   </Box>;
}

export default PostPage;
```

pages/profile-page.tsx

```
import Box from "@mui/material/Box/Box";
import { useEffect } from "react";
import ProfilePosts from "src/features/profile/profile-posts-grid";
import { decodeToken, selectUser } from "src/features/profile/profile-slice";
import ProfileUserData from "src/features/profile/profile-user-data";
import { selectToken } from "src/features/register/register-slice";
import Header from "src/features/ui/header/header-component"
import { useAppDispatch, useAppSelector } from "src/redux/hooks";

const ProfilePage = () => {
   const token = useAppSelector(selectToken);
   const dispatch = useAppDispatch();
   useEffect(() => {
      dispatch(decodeToken(token));
   },[])

   const user = useAppSelector(selectUser);
```

```
   return (
      <Box sx={{
         maxWidth: '1800px',
         margin: '0 auto'
      }}>
         <Header/>
         <ProfileUserData user={user}/>
         <ProfilePosts/>
      </Box>
   );
}

export default ProfilePage;
```

features/ui/header-component.tsx

```
import { Box, Button, Divider, Grid, IconButton } from "@mui/material";
import LoginForm from "src/features/register/login-form/login-form";
import ModalWindow from "src/features/register/modal/modal-component";
import RegisterForm from "src/features/register/register-form/register-form";
import AccountCircleIcon from '@mui/icons-material/AccountCircle';
import logo from 'src/assets/pink-willow-low-resolution-logo-color-on-
transparent-background.png'
import ModalWindowForPostCreation from "src/features/posts/creation-
form/modal-component/modal-component";
import CreationForm from "src/features/posts/creation-form/form/creation-
form";
import { useAppDispatch, useAppSelector } from "src/redux/hooks";
import { useNavigate } from "react-router-dom";
import { logout, selectToken } from "src/features/register/register-slice";
const Header = () => {
   const token = useAppSelector(selectToken);
   const navigate = useNavigate();
   const dispatch = useAppDispatch();
   return  <header>
      <Box sx={{ flexGrow: 1, height: '80px', paddingLeft: '20px' }}>
         <Grid container>
            <Grid item xs={true}>
               <Box  sx={{
                  width: 300,
                  height: 80,
                  "&:hover": {
                     cursor: 'pointer'
                   }
               }}>
                  <img src={logo} onClick={() => navigate('/')} style={{
                     maxHeight: '100%',
                     maxWidth: '100%',
                  }}/>
               </Box>
            </Grid>
            {token ? (
               <Grid item xs={3} container spacing={2} alignItems="center"
justifyContent="center">
                  <Grid item>
                     <ModalWindowForPostCreation>
                        <CreationForm/>
                     </ModalWindowForPostCreation>
                  </Grid>
                  <Grid item>
                     <IconButton onClick={() => navigate('/profile')}>
                        <AccountCircleIcon color='primary' sx={{
                           fontSize: '40px'
                        }}/>
                     </IconButton>
```

```
                </Grid>
                <Grid item>
                   <Button variant='outlined' size="large" onClick={() =>
dispatch(logout())}>Вийти</Button>
                </Grid>
             </Grid>
          ) : (
             <Grid item xs={3} container spacing={2} alignItems="center"
justifyContent="center">
                <Grid item>
                   <ModalWindow buttonLabel="Реєстрація"
modalLabel="Реєстрація">
                      <RegisterForm/>
                   </ModalWindow>
                </Grid>
                <Grid item>
                   <ModalWindow buttonLabel="Логін" modalLabel="Логін">
                      <LoginForm/>
                   </ModalWindow>
                </Grid>
             </Grid>
          )}
       </Grid>
    </Box>
    <Divider sx={{ marginBottom:'10px',width:'100%'}}/>
  </header>
}

export default Header;
```

features/ui/loader-component.tsx

```
import Backdrop from "@mui/material/Backdrop/Backdrop"
import CircularProgress from
"@mui/material/CircularProgress/CircularProgress"

export const Loader = () => {
   return (
    <Backdrop
    sx={{ color: '#fff'}}
    open={true}
  >
    <CircularProgress color="inherit" />
  </Backdrop>
   )
}
```

Features/ui/card-component.tsx

```
import { Card, CardMedia, CardContent, Typography, Grid } from
"@mui/material";
import { FC } from "react";
import { TPostCardProps } from "../../posts/types";
import { Link } from "react-router-dom";
const PostCard: FC<TPostCardProps> = (props) => {
  const {post} = props;
  return   <Link to={`/post/${post.id}`}>
    <Card sx={{ maxWidth: 345, border: "none", boxShadow: "none", marginTop:
'30px'  }}>
      <CardMedia
        component="img"
        height="250"
        image={`/${post.photoUrl}`}
        alt="Image"
        sx={{
          borderRadius: '20px'
```

```
        }}
      />
      <CardContent>
        <Grid container alignItems={'center'} justifyContent={'center'}
direction={'column'}>
          <Grid item>
            <Typography variant="body1">
              {post.title}
            </Typography>
          </Grid>
          <Grid item>
            <Typography variant="body2" color="text.secondary">
              {post.description}
            </Typography>
          </Grid>
          <Grid item container spacing={12} direction={'row'}>
            <Grid item xs={6}>
              <Typography variant="body2">
                {post.region}, {post.city}
              </Typography>
            </Grid>
            <Grid item xs={6}>
              <Typography variant="body2">
                {post.createdAt!== undefined && new
Date(post.createdAt).toLocaleDateString("en-GB")}
              </Typography>
            </Grid>
          </Grid>
        </Grid>


      </CardContent>
    </Card>
  </Link>;
}

export default PostCard;
```

features/register/login-form.tsx

```
import { useState } from "react";
import Button from '@mui/material/Button';
import TextField from '@mui/material/TextField';
import Grid from '@mui/material/Grid';
import {Box} from '@mui/material';
import { useAppDispatch } from "src/redux/hooks";
import { login } from "src/features/register/register-slice";
const LoginForm = () => {
   const [formData,setFormData] = useState({email: '', password: ''});
   const dispatch = useAppDispatch();
   const onSubmit = (e: React.FormEvent<HTMLFormElement>) => {
      e.preventDefault();
      dispatch(login(formData));
   }
   return (
      <Box component="form" noValidate onSubmit={onSubmit} sx={{ mt: 3 }}>
         <Grid container spacing={2}>
            <Grid item xs={12} sm={6}>
               <TextField
                  name="email"
                  required
                  fullWidth
                  id="email"
                  label="Пошта"
                  autoFocus
```

```
                  value={formData.email}
                  onChange={(e) => setFormData({...formData, email:
e.target.value})}
               />
            </Grid>
            <Grid item xs={12} sm={6}>
               <TextField
                  required
                  fullWidth
                  name="password"
                  label="Пароль"
                  type="password"
                  id="password"
                  value={formData.password}
                  onChange={(e) => setFormData({...formData, password:
e.target.value})}
               />
            </Grid>
         </Grid>
         <Button
            type="submit"
            fullWidth
            variant="contained"
            sx={{ mt: 3, mb: 2 }}
         >
            Увійти
         </Button>
      </Box>
   )
}
export default LoginForm;
```

features/register/modal-component.tsx

```
import { Box, Button, Container, Divider, Grid, IconButton, Modal, Typography
} from "@mui/material";
import { FC, useState } from "react";
import CloseIcon from '@mui/icons-material/Close';
import { useAppSelector } from "src/redux/hooks";
import { selectRegisterSuccess } from "src/features/register/register-slice";

type ModalProps = {
   buttonLabel: string,
   modalLabel: string,
   children: React.ReactElement
}
const ModalWindow: FC<ModalProps> = (props) => {
   const [open, setOpen] = useState(false);
   const handleOpen = () => setOpen(true);
   const handleClose = () => setOpen(false);
   const success = useAppSelector(selectRegisterSuccess);
   return (
      <div>
         <Button size="large" onClick={handleOpen}
variant='outlined'>{props.buttonLabel}</Button>
         <Modal
            open={open}
            onClose={handleClose}
            aria-labelledby="modal-modal-title"
            aria-describedby="modal-modal-description"
         >
            <Container component="main" maxWidth="xs">
               <Box
                  sx={{
                     marginTop: 8,
```

```
                        display: 'flex',
                        flexDirection: 'column',
                        alignItems: 'center',
                        position: 'absolute',
                        top: '40%',
                        left: '50%',
                        transform: 'translate(-50%, -50%)',
                        width: 400,
                        bgcolor: 'background.paper',
                        boxShadow: 24,
                        p: 4,
                        borderRadius:'10px'
                     }}
                  >
                     <Grid container spacing={20}>
                        <Grid item xs={2} sm={1}>
                        <IconButton onClick={handleClose}>
                           <CloseIcon/>
                        </IconButton>
                        </Grid>
                        <Grid item xs={12} sm={6}>
                           <Typography component="h1" variant="h6">
                              {props.modalLabel}
                           </Typography>
                        </Grid>
                     </Grid>
                     <Divider sx={{
                        marginTop: '10px',
                        marginBottom:'10px',
                        width:'100%'
                        }}
                     />
                     <Typography component="h1" variant="h5">
                        Вітаємо
                     </Typography>
                     {props.children}
                  </Box>
               </Container>
            </Modal>
         </div>
      );
}

export default ModalWindow;
```

features/register/register-form.tsx

```
import { useState } from "react";
import { useAppDispatch, useAppSelector } from "src/redux/hooks";
import { register } from "src/features/register/register-slice";
import Button from '@mui/material/Button';
import TextField from '@mui/material/TextField';
import Grid from '@mui/material/Grid';
import {Box} from '@mui/material';
const RegisterForm = () => {
   const [formData,setFormData] = useState({email: '', phone: '', password:
''});
   const dispatch = useAppDispatch();
   const onSubmit = (e: React.FormEvent<HTMLFormElement>) => {
      e.preventDefault();
      dispatch(register(formData));
   }
   return (
      <Box component="form" noValidate onSubmit={onSubmit} sx={{ mt: 3 }}>
         <Grid container spacing={2}>
```

```
            <Grid item xs={12} sm={6}>
               <TextField
                  name="email"
                  required
                  fullWidth
                  id="email"
                  label="Пошта"
                  autoFocus
                  value={formData.email}
                  onChange={(e) => setFormData({...formData, email:
e.target.value})}
                  />
            </Grid>
            <Grid item xs={12} sm={6}>
               <TextField
                  required
                  fullWidth
                  id="phone"
                  label="Номер телефона"
                  name="phone"
                  value={formData.phone}
                  onChange={(e) => setFormData({...formData, phone:
e.target.value})}
                  />
            </Grid>
            <Grid item xs={12}>
               <TextField
                  required
                  fullWidth
                  name="password"
                  label="Пароль"
                  type="password"
                  id="password"
                  value={formData.password}
                  onChange={(e) => setFormData({...formData, password:
e.target.value})}
                  />
            </Grid>
         </Grid>
         <Button
         type="submit"
         fullWidth
         variant="contained"
         sx={{ mt: 3, mb: 2 }}
         >
         Зареєструватися
         </Button>
      </Box>
   )
}
export default RegisterForm;
```

features/register/register-component.tsx

```
import LoginForm from "src/features/register/login-form/login-form";
import ModalWindow from "src/features/register/modal/modal-component"
import RegisterForm from "src/features/register/register-form/register-form"

const RegisterComponent = () => {
   return <>
      <ModalWindow buttonLabel="Sign up" modalLabel="Sign up">
         <RegisterForm/>
      </ModalWindow>
      <ModalWindow buttonLabel="Sign in" modalLabel="Sign in">
         <LoginForm/>
```

```
      </ModalWindow>
    </>
}

export default RegisterComponent;
```

features/register/types.ts

```
export interface IAuthState {
   loading: boolean,
   error: Error | null,
   token: string | null,
   success: boolean,
 }
```

features/register/register-slice.ts

```
import { createAsyncThunk, createSlice,  } from '@reduxjs/toolkit'
import { IAuthState } from 'src/features/register/types'
import { RootState } from 'src/redux/store'
import axios from 'axios';
import { IUser } from 'src/types';

export const register = createAsyncThunk(
  "auth/register",
  async ( userData: IUser ) => {
    const response = await axios.post("http://localhost:4000/graphql", {
      query: `
        mutation createUser($email: String!, $phone: String!, $password:
String!){
          createUser(createUserInput: {
            email: $email,
            phone: $phone,
            password: $password
          }){
            access_token
          }
      }
      `,
      variables: {
        email: userData.email,
        phone: userData.phone,
        password: userData.password
      }
    });
    return response.data.data.createUser;
  }
);

export const login = createAsyncThunk(
  "auth/login",
  async ( userData: IUser ) => {
    const response = await axios.post("http://localhost:4000/graphql", {
      query: `
        mutation loginUser($email: String!, $password: String!){
          loginUser(email: $email, password: $password){
            access_token
          }
      }
      `,
      variables: {
        email: userData.email,
        password: userData.password
      }
```

```
    });
    return response.data.data.loginUser;
  }
);

const initialState: IAuthState = {
   loading: false,
   token: null,
   error: null,
   success: false,
}

export const authSlice = createSlice({
  name: 'auth',
  initialState,
  reducers: {
    toggleLoading: (state) => {
      state.loading = !state.loading;
    },
    setToken: (state, action) => {
      state.token = action.payload;
    },
    logout: (state) => {
      state.token = null;
      localStorage.clear();
    }
  },
  extraReducers: (builder) => {
    builder.addCase(register.pending, (state) => {
      state.loading = true;
    })
    builder.addCase(register.fulfilled, (state, action) => {
      state.token = action.payload.access_token;
      if(state.token !== null){
        window.localStorage.setItem('token', state.token)
      }
      state.loading = false;
      state.success = true;
    })
    builder.addCase(login.pending, (state) => {
      state.loading = true;
    })
    builder.addCase(login.fulfilled, (state, action) => {
      state.token = action.payload.access_token;
      if(state.token !== null){
        window.localStorage.setItem('token', state.token)
      }
      state.loading = false;
      state.success = true;

    })
  },
})

export const { toggleLoading, setToken, logout } = authSlice.actions

export const selectRegisterLoading = (state: RootState) => state.auth.loading
export const selectRegisterSuccess = (state: RootState) => state.auth.success
export const selectToken = (state: RootState) => state.auth.token

export default authSlice.reducer
```

features/profile/types.ts

```
import { IUser } from "src/types";

export interface IProfileState {
   loading: boolean,
   error: Error | null,
   user: IUser,
   success: boolean,
}
```

features/profile/profile-slice.ts

```
import { createAsyncThunk, createSlice } from '@reduxjs/toolkit'
import { RootState } from 'src/redux/store'
import axios from 'axios';
import { IUser } from 'src/types';
import { IProfileState } from './types';
import jwt_decode from 'jwt-decode';



 export const getUserPosts = createAsyncThunk(
   "profile/getUserPosts",
   async ( uid: string ) => {
     const response = await axios.post("http://localhost:4000/graphql", {
       query: `
         query usersPosts($id: String!){
            usersPosts(id: $id){
               id,
               photoUrl,
               clientCategories,
               city,
               region,
               countOfPlaces,
               title,
               description,
               createdAt,
               periodOfTime,
               type,
            }
         }
       `,
       variables: {
         id: uid,
       },
     });
     return response.data.data.usersPosts;
   }
 );


const initialState: IProfileState = {
   loading: false,
   user: {
      uid: '',
      email: '',
      phone: '',
      posts: [],
   },
   error: null,
   success: false,
}

export const profileSlice = createSlice({
  name: 'profile',
  initialState,
```

```
  reducers: {
    toggleLoading: (state) => {
      state.loading = !state.loading;
    },
    decodeToken: (state, action) => {
      const token = action.payload;
      if(token !== null){
         const userData: IUser = jwt_decode(token)
         state.user.email = userData.email;
         state.user.phone = userData.phone;
         state.user.uid = userData.uid;
      }
    }
  },
  extraReducers: (builder) => {
      builder.addCase(getUserPosts.pending, (state) => {
         state.loading = true;
      })
      builder.addCase(getUserPosts.fulfilled, (state, action) => {
         state.user.posts = action.payload;
      })
  }
})

export const { toggleLoading, decodeToken } = profileSlice.actions

export const selectProfileLoading = (state: RootState) =>
state.profile.loading
export const selectProfileSuccess = (state: RootState) =>
state.profile.success
export const selectUser = (state: RootState) => state.profile.user

export default profileSlice.reducer
```

features/profile/profile-user-data.tsx

```
import { Divider, Grid, Typography } from "@mui/material";
import { FC } from "react";
import { IUser } from "src/types";
type ProfileUserDataProps = {
   user: IUser
}
const ProfileUserData: FC<ProfileUserDataProps> = ({user}) => {
   return <Grid container justifyContent={'center'} sx={{
      maxWidth:'700px',
      margin: '0 auto'
   }}>
      <Typography variant="h4">Особисті дані</Typography>
      <Grid xs={12} item>
         <Typography variant="body1">
            Phone
         </Typography>
         <Typography variant="body2">
            {user.phone}
         </Typography>
         <Divider sx={{ marginBottom:'10px',width:'100%', marginTop:
'10px'}}/>
      </Grid>
      <Grid xs={12} item>
         <Typography variant="body1">
            Email
         </Typography>
         <Typography variant="body2">
            {user.email}
         </Typography>
```

```
         <Divider sx={{ marginBottom:'10px',width:'100%'}}/>
      </Grid>
   </Grid>
}

export default ProfileUserData;
```

features/profile/profile-posts-grid.tsx

```
import Grid from "@mui/material/Grid/Grid";
import PostCard from "../ui/card/card-component";
import { useAppDispatch, useAppSelector } from "src/redux/hooks";
import { useEffect } from "react";
import { getUserPosts, selectUser } from "./profile-slice";
import Typography from "@mui/material/Typography";

const ProfilePosts = () => {
   const dispatch = useAppDispatch();
   const user = useAppSelector(selectUser);
   useEffect(() => {
      if(user.uid !== undefined)
         dispatch(getUserPosts(user.uid))
   },[user.uid])
   return (
      <Grid container justifyContent={'center'} sx={{
         margin: '0 auto',

      }}>
         <Typography variant='h4'>Ваші Оголошення</Typography>
         <Grid container spacing={3}>
            {user.posts !== undefined && user.posts.map((item) => <Grid item
xs={2} key={item.id}>
               <PostCard post={item}/>
            </Grid>)}
         </Grid>
      </Grid>
   );
}
export default ProfilePosts;
```

features/posts/cards-grid-component.tsx

```
import Grid from "@mui/material/Grid/Grid";
import PostCard from "src/features/ui/card/card-component";
import { useAppSelector } from "src/redux/hooks";
import { selectCountOfItemsOnPage, selectPostsLoading, selectPage,
selectPosts } from "../posts-slice";
import { useFetchInitialPosts, useInfiniteScrollForPosts } from "../hooks";
const CardsGrid = () => {

   const countOfItemsOnPage = useAppSelector(selectCountOfItemsOnPage);
   const page = useAppSelector(selectPage);
   const posts = useAppSelector(selectPosts);
   const loading = useAppSelector(selectPostsLoading);
   useFetchInitialPosts({page, countOfItemsOnPage});
   useInfiniteScrollForPosts({page, countOfItemsOnPage, loading});


   return (
      <Grid container spacing={3}>
         {posts.map((item) => <Grid item xs={2} key={item.id}>
            <PostCard post={item}/>
         </Grid>)}
      </Grid>
   );
}
```

```
export default CardsGrid;
```

features/posts/filters-component.tsx

```
import ModalWindowForFilters from "./modal-component"

const Filters = () => {
   return <ModalWindowForFilters/>;
}

export default Filters;
```

features/posts/modal-component.tsx

```
import { Autocomplete, Box, Button, Chip, Container, Divider, FormControl,
Grid, IconButton, InputLabel, MenuItem, Modal, OutlinedInput, Select,
SelectChangeEvent, Slider, TextField, Typography } from "@mui/material";
import { useState } from "react";
import CloseIcon from '@mui/icons-material/Close';
import TuneIcon from '@mui/icons-material/Tune';
import data from 'src/features/posts/creation-form/form/CitiesAndVillages -
14 March.json'
import { IPost } from "src/types";
import { useAppDispatch } from "src/redux/hooks";
import { setFilters, setPage } from "../posts-slice";

const ModalWindowForFilters = () => {
   const [filters,setFiltersData] = useState<IPost>({countOfPlaces: 0,
clientCategories: []});
   const [open, setOpen] = useState(false);
   const [inputValueRegion, setInputValueRegion] = useState('');
   const [inputValueCity, setInputValueCity] = useState('');
   const [inputValueType, setInputValueType] = useState('');
   const [inputValueTime, setInputValueTime] = useState('');
   const handleOpen = () => setOpen(true);
   const handleClose = () => setOpen(false);
   const typeList = ['Будинок', "Квартира", "Кімната", "Спальні місця/місце"]
   const periodOfTimeOptions = ['Будь-який період часу', "До кінця війни",
"Кілька днів", "Кілька тижнів", "Кілька місяців", "Півроку", "Більше ніж
рік"]
   const cities = data.filter((item) => item.object_category !==
'Село').map((item) => item.object_name).sort();
   const regionsList = ["АР Крим","Вінницька обл.", "Волинська обл.",
"Дніпропетровська обл.", "Донецька обл.", "Житомирська обл.", "Закарпатська
обл.", "Запорізка обл.", "Івано-Франківська обл.", "Кіровоградська обл.",
"Київська обл.", "Луганська обл.", "Львівська обл.", "Миколаївська обл.",
"Одеська обл.", "Полтавська обл.", "Рівненська обл.", "Сумська обл.",
"Тернопільська обл.", "Харківська обл.", "Херсонська обл.", "Хмельницька
обл.", "Черкаська обл.", "Чернівецька обл.", "Чернігівська обл."]
   const uniqueCities = [...new Set(cities)];
   const dispatch = useAppDispatch();
   const onSubmit = (e: React.FormEvent<HTMLFormElement>) => {
      e.preventDefault();
      dispatch(setPage(1));
      dispatch(setFilters(filters));
   }

   const handleSliderChange = (event: Event, newValue: number | number[]) =>
{
      setFiltersData({...filters, countOfPlaces:newValue as number});
   };


   const names = ['Чоловіки',"Жінки", "Діти", "Сім'я з дітьми", "Сім'я без
дітей", "Пенсіонери", "Інваліди", "Особи з домашніми тваринами"]
```

```
   const handleChange = (event: SelectChangeEvent<typeof
filters.clientCategories>) => {
      const {
         target: { value },
      } = event;
      setFiltersData(
         {...filters, clientCategories: typeof value === 'string' ?
value.split(',') : value,}
      );
   };

   return (
      <div>
         <Button startIcon={<TuneIcon/>} size="large" onClick={handleOpen}
variant='outlined'>Фільтри</Button>
         <Modal
            open={open}
            onClose={handleClose}
            aria-labelledby="modal-modal-title"
            aria-describedby="modal-modal-description"
         >
            <Container component="main" maxWidth="xs">
               <Box
                  sx={{
                     marginTop: 8,
                     display: 'flex',
                     flexDirection: 'column',
                     alignItems: 'center',
                     position: 'absolute',
                     top: '40%',
                     left: '50%',
                     transform: 'translate(-50%, -50%)',
                     width: 400,
                     bgcolor: 'background.paper',
                     boxShadow: 24,
                     p: 4,
                     borderRadius:'10px'
                  }}
               >
                  <Grid container spacing={17}>
                     <Grid item xs={2} sm={1}>
                     <IconButton onClick={handleClose}>
                        <CloseIcon/>
                     </IconButton>
                     </Grid>
                     <Grid item xs={12} sm={6}>
                        <Typography component="h1" variant="h6">
                           Фільтри
                        </Typography>
                     </Grid>
                  </Grid>
                  <Divider sx={{
                     marginTop: '10px',
                     marginBottom:'10px',
                     width:'100%'
                     }}
                  />
                  <Box component="form"  onSubmit={onSubmit} sx={{ mt: 3 }}>
                     <Grid container spacing={2}>
                        <Grid item xs={12} sm={6}>
                           <Autocomplete
                              id="region"
                              options={regionsList}
```

```
                                renderInput={(params) => <TextField {...params}
label="Регіон" />}
                                inputValue={inputValueRegion}
                                onInputChange={(event, newInputValue) => {
                                   setInputValueRegion(newInputValue);
                                }}
                                value={filters.region}
                                onChange={(event, newValue: string) => {
                                   setFiltersData({...filters, region:
newValue});
                                }}
                             />
                          </Grid>
                          <Grid item xs={12} sm={6}>
                             <Autocomplete
                                id="city"
                                options={uniqueCities}
                                renderInput={(params) => <TextField {...params}
label="Місто" />}
                                inputValue={inputValueCity}
                                onInputChange={(event, newInputValue) => {
                                   setInputValueCity(newInputValue);
                                }}
                                value={filters.city}
                                onChange={(event, newValue: string) => {
                                   setFiltersData({...filters, city:
newValue});
                                }}
                             />
                          </Grid>
                          <Grid item xs={12} sm={6}>
                             <Autocomplete
                                id="type"
                                options={typeList}
                                renderInput={(params) => <TextField {...params}
label="Тип розміщення" />}
                                inputValue={inputValueType}
                                onInputChange={(event, newInputValue) => {
                                   setInputValueType(newInputValue);
                                }}
                                value={filters.type}
                                onChange={(event, newValue: string) => {
                                   setFiltersData({...filters, type:
newValue});
                                }}
                             />
                          </Grid>
                          <Grid item xs={12} sm={12}>
                             <Autocomplete
                                id="periodOfTime"
                                options={periodOfTimeOptions}
                                renderInput={(params) => <TextField {...params}
label="Період часу на який пропонується житло" />}
                                inputValue={inputValueTime}
                                onInputChange={(event, newInputValue) => {
                                   setInputValueTime(newInputValue);
                                }}
                                value={filters.periodOfTime}
                                onChange={(event, newValue: string) => {
                                   setFiltersData({...filters, periodOfTime:
newValue});
                                }}
                             />
                          </Grid>
```

```
                        <Grid item xs={12} sm={12}>
                           <Typography variant="body2">Кількість місць для
бронювання</Typography>
                           <Slider
                              size="small"
                              value={filters.countOfPlaces}
                              onChange={handleSliderChange}
                              aria-label="Small"
                              valueLabelDisplay="auto"
                           />
                        </Grid>
                        <FormControl sx={{ width: '100%' }}>
                           <InputLabel id="demo-multiple-chip-
label">Категорії клієнтів</InputLabel>
                           <Select
                              sx={{ width: '100%' }}
                              labelId="demo-multiple-chip-label"
                              id="demo-multiple-chip"
                              multiple
                              value={filters.clientCategories}
                              onChange={handleChange}
                              input={<OutlinedInput id="select-multiple-chip"
label="Chip" />}
                              renderValue={(selected) => (
                                 <Box sx={{ display: 'flex', flexWrap:
'wrap', gap: 0.5 }}>
                                 {selected.map((value) => (
                                    <Chip key={value} label={value} />
                                 ))}
                                 </Box>
                              )}
                              MenuProps={{
                                 PaperProps: {
                                 style: {
                                    maxHeight: 48 * 4.5 + 8,
                                    width: 250,
                                 },
                                 },
                              }}
                           >
                              {names.map((name) => (
                                 <MenuItem
                                 key={name}
                                 value={name}
                                 >
                                 {name}
                                 </MenuItem>
                              ))}
                           </Select>
                        </FormControl>
                     </Grid>
                     <Button
                        type="submit"
                        fullWidth
                        variant="contained"
                        sx={{ mt: 3, mb: 2 }}
                     >
                        Відфільтрувати
                     </Button>
                  </Box>
               </Box>
            </Container>
         </Modal>
      </div>
```

```
   );
}

export default ModalWindowForFilters;
```

features/posts/hooks.ts

```
import { useAppDispatch, useAppSelector } from "src/redux/hooks";
import { TUseFetchInitialPostsProps, TUseInfiniteScrollForPosts } from
"./types";
import { useEffect } from "react";
import { getPosts, selectPostsFilters } from "./posts-slice";

export const useFetchInitialPosts = (props: TUseFetchInitialPostsProps) => {
   const filters = useAppSelector(selectPostsFilters);
   const {page, countOfItemsOnPage} = props;
   const dispatch = useAppDispatch();
   useEffect(() => {
      dispatch(getPosts({take: countOfItemsOnPage, skip: (page - 1) *
countOfItemsOnPage, filters}));
   },[filters])
}

export const useInfiniteScrollForPosts = (props: TUseInfiniteScrollForPosts)
=> {
   const {page, countOfItemsOnPage, loading} = props;
   const filters = useAppSelector(selectPostsFilters);
   const dispatch = useAppDispatch();
   const handleScroll = () => {
      if (window.innerHeight + document.documentElement.scrollTop !==
document.documentElement.offsetHeight || loading) {
        return;
      }
      dispatch(getPosts({take: countOfItemsOnPage, skip: (page - 1) *
countOfItemsOnPage, filters}));
    };

    useEffect(() => {
      window.addEventListener('scroll', handleScroll);
      return () => window.removeEventListener('scroll', handleScroll);
    }, [loading, filters]);
}
```

features/posts/posts-slice.ts

```
import { PayloadAction, createAsyncThunk, createSlice } from
'@reduxjs/toolkit'
import { RootState } from 'src/redux/store'
import axios from 'axios';
import { IFilters, IPostsState } from './types';
import { IPost } from 'src/types';

const initialState: IPostsState = {
  page: 1,
  countOfItemsOnPage: 18,
  posts: [],
  loading: false,
  error: null,
  success: false,
  filters: {
    countOfPlaces: 0,
  }
}

type getPostsProps = {
  take: number,
```

```
  skip: number,
  filters: IFilters
}
export const getPosts = createAsyncThunk(
  "posts/getPosts",
  async ( data: getPostsProps ) => {
    try {
      const response = await axios.post("http://localhost:4000/graphql", {
      query: `
        query postsOnPage($take: Float!, $skip: Float!, $filters:
GetPostsFilters!){
          postsOnPage(take: $take, skip: $skip, filters: $filters){
            id,
            photoUrl,
            clientCategories,
            city,
            region,
            countOfPlaces,
            title,
            description,
            createdAt,
            periodOfTime,
            type,
            user{
              email,
              phone
            }
          }
      }
      `,
      variables: {
        take: data.take,
        skip: data.skip,
        filters: data.filters,
      },
    });
    return response.data.data.postsOnPage;
    } catch (error) {
    }

  }
);

type createPostProps = {
  data: IPost,
  token: string | null,
}
export const createPost = createAsyncThunk(
  "posts/createPost",
  async ( props: createPostProps ) => {
    try {
      const response = await axios.post("http://localhost:4000/graphql", {
      query: `
      mutation createPost($clientCategories: [String!]!, $city: String!,
$region: String!, $countOfPlaces: Float!, $title: String!, $description:
String!, $periodOfTime: String!, $type: String!, $user: String!){
        createPost(createPostInput: {
          clientCategories: $clientCategories,
          city: $city,
          region: $region,
          countOfPlaces: $countOfPlaces,
          title: $title,
          description: $description,
          periodOfTime: $periodOfTime,
```

```
          type: $type,
          user: $user,
        }){
          city,
          countOfPlaces
        }
      }
      `,
      variables: {
        clientCategories: props.data.clientCategories,
        city: props.data.city,
        region: props.data.region,
        countOfPlaces: props.data.countOfPlaces,
        title: props.data.title,
        description: props.data.description,
        periodOfTime: props.data.periodOfTime,
        type: props.data.type,
        user: props.data.user,
      }
    },
    );
    return response.data.data.createPost;

    } catch (error) {
    }

  }
);

export const postsSlice = createSlice({
  name: 'posts',
  initialState,
  reducers: {
    toggleLoading: (state) => {
      state.loading = !state.loading;
    },
    setPage: (state, action) => {
      state.page = action.payload;
    },
    setFilters: (state, action) => {
      state.filters = action.payload;
      state.posts = [];
    },
  },
  extraReducers: (builder) => {
    builder.addCase(getPosts.fulfilled, (state, action) => {
      state.posts = [...state.posts, ...action.payload];
      state.loading = false;
      state.success = true;
      state.page++;
    })
    builder.addCase(getPosts.pending, (state) => {
      state.loading = true;
    })
    builder.addCase(createPost.pending, (state) => {
      state.loading = true;
    })
  }
})

export const { toggleLoading, setPage, setFilters} = postsSlice.actions

export const selectPostsLoading = (state: RootState) => state.posts.loading
export const selectPosts = (state: RootState) => state.posts.posts
```

```
export const selectPostById = (state: RootState, id: string) =>
state.posts.posts.find((post) => post.id === id)
export const selectPage = (state: RootState) => state.posts.page
export const selectCountOfItemsOnPage = (state: RootState) =>
state.posts.countOfItemsOnPage
export const selectPostsSuccess = (state: RootState) => state.posts.success
export const selectPostsFilters = (state: RootState) => state.posts.filters

export default postsSlice.reducer
```

features/posts/types.ts

```
import { IPost } from "src/types";

export interface IPostsState {
   posts: IPost[],
   page: number,
   countOfItemsOnPage: number,
   loading: boolean,
   success: boolean,
   error: Error | null,
   filters: IFilters,
}

export type TUseFetchInitialPostsProps = {
   page: number;
   countOfItemsOnPage: number;
}
export type TUseInfiniteScrollForPosts = {
   page: number;
   countOfItemsOnPage: number;
   loading: boolean;
}

export type TPostCardProps = {
   post: IPost;
}

export interface IFilters {
   clientCategories?: Array<string>,
   city?: string,
   region?: string,
   countOfPlaces?: number,
   periodOfTime?: string,
   type?: string,
}
```

Features/posts/creation-form/modal-component.tsx

```
import { Box, Button, Container, Divider, Grid, IconButton, Modal, Typography
} from "@mui/material";
import { FC, useState } from "react";
import CloseIcon from '@mui/icons-material/Close';

type ModalProps = {
   children: React.ReactElement
}
const ModalWindowForPostCreation: FC<ModalProps> = (props) => {
   const [open, setOpen] = useState(false);
   const handleOpen = () => setOpen(true);
   const handleClose = () => setOpen(false);
   return (
      <div>
         <Button size="large" onClick={handleOpen} variant='outlined'>+
Оголошення</Button>
         <Modal
```

```
            open={open}
            onClose={handleClose}
            aria-labelledby="modal-modal-title"
            aria-describedby="modal-modal-description"
         >
            <Container component="main" maxWidth="xs">
               <Box
                  sx={{
                     marginTop: 8,
                     display: 'flex',
                     flexDirection: 'column',
                     alignItems: 'center',
                     position: 'absolute',
                     top: '40%',
                     left: '50%',
                     transform: 'translate(-50%, -50%)',
                     width: 400,
                     bgcolor: 'background.paper',
                     boxShadow: 24,
                     p: 4,
                     borderRadius:'10px'
                  }}
               >
                  <Grid container spacing={17}>
                     <Grid item xs={2} sm={1}>
                     <IconButton onClick={handleClose}>
                        <CloseIcon/>
                     </IconButton>
                     </Grid>
                     <Grid item xs={12} sm={6}>
                        <Typography component="h1" variant="h6">
                           Оголошення
                        </Typography>
                     </Grid>
                  </Grid>
                  <Divider sx={{
                     marginTop: '10px',
                     marginBottom:'10px',
                     width:'100%'
                     }}
                  />
                  {props.children}
               </Box>
            </Container>
         </Modal>
      </div>
   );
}

export default ModalWindowForPostCreation;
```

Features/posts/creation-form/form/city-autocomplete.tsx

```
import Autocomplete from "@mui/material/Autocomplete/Autocomplete";
import TextField from "@mui/material/TextField/TextField";
import { FC, useState } from "react";
import data from 'src/features/posts/creation-form/form/CitiesAndVillages -
14 March.json'
import { IPost } from "src/types";
type CityAutocompleteProps = {
   formData: IPost;
   setFormData: React.Dispatch<React.SetStateAction<IPost>>;
}
const CityAutocomplete: FC<CityAutocompleteProps> = ({formData, setFormData})
=> {
```

```
   const [inputValue, setInputValue] = useState('');
   const cities = data.filter((item) => item.object_category !==
'Село').map((item) => item.object_name).sort();
   const uniqueCities = [...new Set(cities)];
   return <Autocomplete
      id="city"
      options={uniqueCities}
      renderInput={(params) => <TextField {...params} required label="Місто"
/>}
      inputValue={inputValue}
      onInputChange={(event, newInputValue) => {
         setInputValue(newInputValue);
      }}
      value={formData.city}
      onChange={(event, newValue: string) => {
         setFormData({...formData, city: newValue});
       }}
   />;
}

export default CityAutocomplete;
```

Features/posts/creation-form/form/period-of-time-autocomplete.tsx

```
import Autocomplete from "@mui/material/Autocomplete/Autocomplete";
import TextField from "@mui/material/TextField/TextField";import { FC,
useState } from "react";
import { IPost } from "src/types";
type PeriodOfTimeAutocompleteProps = {
   formData: IPost;
   setFormData: React.Dispatch<React.SetStateAction<IPost>>;
}
const PeriodOfTimeAutocomplete: FC<PeriodOfTimeAutocompleteProps> =
({formData, setFormData}) => {
   const [inputValue, setInputValue] = useState('');
   const periodOfTimeOptions = ['Будь-який період часу', "До кінця війни",
"Кілька днів", "Кілька тижнів", "Кілька місяців", "Півроку", "Більше ніж
рік"]
   return <Autocomplete
      id="periodOfTime"
      options={periodOfTimeOptions}
      renderInput={(params) => <TextField {...params} required label="Період
часу на який пропонується житло" />}
      inputValue={inputValue}
      onInputChange={(event, newInputValue) => {
         setInputValue(newInputValue);
      }}
      value={formData.periodOfTime}
      onChange={(event, newValue: string) => {
         setFormData({...formData, periodOfTime: newValue});
       }}
   />;
}

export default PeriodOfTimeAutocomplete;
```

Features/posts/creation-form/form/region-autocomplete.tsx

```
import Autocomplete from "@mui/material/Autocomplete/Autocomplete";
import TextField from "@mui/material/TextField/TextField";import { FC,
useState } from "react";
import { IPost } from "src/types";
type RegionAutocompleteProps = {
   formData: IPost;
   setFormData: React.Dispatch<React.SetStateAction<IPost>>;
}
```

```
const RegionAutocomplete: FC<RegionAutocompleteProps> = ({formData,
setFormData}) => {
   const [inputValue, setInputValue] = useState('');
   const regionsList = ["АР Крим","Вінницька обл.", "Волинська обл.",
"Дніпропетровська обл.", "Донецька обл.", "Житомирська обл.", "Закарпатська
обл.", "Запорізка обл.", "Івано-Франківська обл.", "Кіровоградська обл.",
"Київська обл.", "Луганська обл.", "Львівська обл.", "Миколаївська обл.",
"Одеська обл.", "Полтавська обл.", "Рівненська обл.", "Сумська обл.",
"Тернопільська обл.", "Харківська обл.", "Херсонська обл.", "Хмельницька
обл.", "Черкаська обл.", "Чернівецька обл.", "Чернігівська обл."]
   return <Autocomplete
      id="region"
      options={regionsList}
      renderInput={(params) => <TextField {...params} required label="Регіон"
/>}
      inputValue={inputValue}
      onInputChange={(event, newInputValue) => {
         setInputValue(newInputValue);
      }}
      value={formData.region}
      onChange={(event, newValue: string) => {
         setFormData({...formData, region: newValue});
       }}
   />;
}

export default RegionAutocomplete;
```

Features/posts/creation-form/form/type-autocomplete.tsx

```
import Autocomplete from "@mui/material/Autocomplete/Autocomplete";
import TextField from "@mui/material/TextField/TextField";import { FC,
useState } from "react";
import { IPost } from "src/types";
type TypeAutocompleteProps = {
   formData: IPost;
   setFormData: React.Dispatch<React.SetStateAction<IPost>>;
}
const TypeAutocomplete: FC<TypeAutocompleteProps> = ({formData, setFormData})
=> {
   const [inputValue, setInputValue] = useState('');
   const typeList = ['Будинок', "Квартира", "Кімната", "Спальні місця/місце"]
   return <Autocomplete
      id="type"
      options={typeList}
      renderInput={(params) => <TextField {...params} required label="Тип
розміщення" />}
      inputValue={inputValue}
      onInputChange={(event, newInputValue) => {
         setInputValue(newInputValue);
      }}
      value={formData.type}
      onChange={(event, newValue: string) => {
         setFormData({...formData, type: newValue});
       }}
   />;
}

export default TypeAutocomplete;
```

Features/posts/creation-form/form/select-client-category.tsx

```
import { FormControl, InputLabel, Select, OutlinedInput, Box, Chip, MenuItem,
SelectChangeEvent } from "@mui/material";
import { FC } from "react";
import { IPost } from "src/types";
```

```
type SelectClientCategoryProps = {
   formData: IPost;
   setFormData: React.Dispatch<React.SetStateAction<IPost>>;
}
const SelectClientCategory: FC<SelectClientCategoryProps> =
({formData,setFormData}) => {
   const names = ['Чоловіки',"Жінки", "Діти", "Сім'я з дітьми", "Сім'я без
дітей", "Пенсіонери", "Інваліди", "Особи з домашніми тваринами"]

   const handleChange = (event: SelectChangeEvent<typeof
formData.clientCategories>) => {
      const {
        target: { value },
      } = event;
      setFormData(
        {...formData, clientCategories: typeof value === 'string' ?
value.split(',') : value,}
      );
    };
   return <FormControl sx={{ width: '100%' }}>
      <InputLabel id="demo-multiple-chip-label">Категорії
клієнтів</InputLabel>
      <Select
         sx={{ width: '100%' }}
         labelId="demo-multiple-chip-label"
         id="demo-multiple-chip"
         required
         multiple
         value={formData.clientCategories}
         onChange={handleChange}
         input={<OutlinedInput id="select-multiple-chip" label="Chip" />}
         renderValue={(selected) => (
            <Box sx={{ display: 'flex', flexWrap: 'wrap', gap: 0.5 }}>
            {selected.map((value) => (
               <Chip key={value} label={value} />
            ))}
            </Box>
         )}
         MenuProps={{
            PaperProps: {
              style: {
                maxHeight: 48 * 4.5 + 8,
                width: 250,
              },
            },
          }}
      >
         {names.map((name) => (
            <MenuItem
            key={name}
            value={name}
            >
            {name}
            </MenuItem>
         ))}
      </Select>
   </FormControl>

}

export default SelectClientCategory;
```

Features/posts/creation-form/form/creation-form.tsx

```
import { ChangeEvent, useState } from "react";
```

```
import Button from '@mui/material/Button';
import TextField from '@mui/material/TextField';
import Grid from '@mui/material/Grid';
import { useAppDispatch, useAppSelector } from "src/redux/hooks";
import CityAutocomplete from "./city-autocomplete";
import RegionAutocomplete from "./region-autocomplete";
import {Box, Slider, Typography} from '@mui/material';
import { IPost } from "src/types";
import SelectClientCategory from "./select-client-category";
import TypeAutocomplete from "./type-autocomplete";
import PeriodOfTimeAutocomplete from "./period-of-time-autocomplete";
import { createPost } from "../../posts-slice";
import { selectToken } from "src/features/register/register-slice";
import { selectUser } from "src/features/profile/profile-slice";
const CreationForm = () => {
   const [formData,setFormData] = useState<IPost>({title: '', description:
'', city: '',  region: '', countOfPlaces: 0, clientCategories: [],type: '',
periodOfTime: '', photoUrl: 'url'});
   const dispatch = useAppDispatch();
   const token = useAppSelector(selectToken);
   const user = useAppSelector(selectUser)
   const onSubmit = (e: React.FormEvent<HTMLFormElement>) => {
      e.preventDefault();
      formData.user = user.uid;
      dispatch(createPost({token, data: formData}));
   }

   const handleSliderChange = (event: Event, newValue: number | number[]) =>
{
      setFormData({...formData, countOfPlaces:newValue as number});
    };

   const handleFileChange = (e: ChangeEvent<HTMLInputElement>) => {
      if (e.target.files) {
         setFormData({...formData, photoUrl: e.target.files[0]});
      }
   };

   return (
      <Box component="form"  onSubmit={onSubmit} sx={{ mt: 3 }}>
         <Grid container spacing={2}>
            <Grid item xs={12} sm={6}>
               <TextField
                  name="title"
                  required
                  fullWidth
                  id="title"
                  label="Заголовок"
                  autoFocus
                  value={formData.title}
                  onChange={(e) => setFormData({...formData, title:
e.target.value})}
                  />
            </Grid>
            <Grid item xs={12} sm={6}>
               <RegionAutocomplete formData={formData}
setFormData={setFormData}/>
            </Grid>
            <Grid item xs={12} sm={6}>
               <CityAutocomplete formData={formData}
setFormData={setFormData}/>
            </Grid>
            <Grid item xs={12} sm={6}>
               <TypeAutocomplete formData={formData}
```

```
setFormData={setFormData}/>
            </Grid>
            <Grid item xs={12} sm={12}>
               <PeriodOfTimeAutocomplete formData={formData}
setFormData={setFormData}/>
            </Grid>
            <Grid item xs={12} sm={12}>
               <Typography variant="body2">Кількість місць для
бронювання</Typography>
               <Slider
                  size="small"
                  value={formData.countOfPlaces}
                  onChange={handleSliderChange}
                  aria-label="Small"
                  valueLabelDisplay="auto"
               />
            </Grid>
            <Grid item xs={12} sm={12}>
               <SelectClientCategory formData={formData}
setFormData={setFormData}/>
            </Grid>
            <Grid item xs={12} sm={12}>
               <TextField
                  required
                  fullWidth
                  multiline
                  rows={4}
                  name="description"
                  label="Опис оголошення"
                  id="description"
                  value={formData.description}
                  onChange={(e) => setFormData({...formData, description:
e.target.value})}
               />
            </Grid>
            <Grid item xs={12} sm={12}>
               <Button
                  variant="contained"
                  component="label"
                  fullWidth
                  >
                  Завантажити фото
                  <input
                     type="file"
                     hidden
                     onChange={handleFileChange}
                  />
               </Button>
            </Grid>


         </Grid>
         <Button
            type="submit"
            fullWidth
            variant="contained"
            sx={{ mt: 3, mb: 2 }}
         >
            Створити
         </Button>
      </Box>
   )
}
export default CreationForm;
```