

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

бакалавр

(ступінь вищої освіти)

на тему РОЗРОБКА ВЕБЗАСТОСУНКУ ДЛЯ ПРОДАЖУ
НЕРУХОМОСТІ

DEVELOPMENT OF A WEB APPLICATION FOR REAL
ESTATE SALES

Виконав: студент(ка) 4 курсу, групи КНТ-220сп
Спеціальності 122 Комп'ютерні науки
(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

ГАНОШЕНКО П.О.

(ПРІЗВИЩЕ та ініціали)

Керівник ЗАЙКО Т.А.

(ПРІЗВИЩЕ та ініціали)

Рецензент КОЦУР М.І.

(ПРІЗВИЩЕ та ініціали)

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

Факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 122 Комп'ютерні науки
 (код і найменування)
 Освітня програма (спеціалізація) Комп'ютерні науки
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
Сергій СУББОТІН
 “ ” 2023 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

ГАНОШЕНКА Пилипа Олександровича
 (ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Розробка вебзастосунку для продажу нерухомості.
Development of a Web Application for Real Estate Sales.

керівник проєкту (роботи) к.т.н., доцент, ЗАЙКО Тетяна Анатоліївна,
 (науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від “4” квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 23 травня 2023 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Проєктування архітектури вебзастосунку. 3. Реалізація вебзастосунку.
4. Експлуатація і тестування вебзастосунку.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЗАЙКО Т.А., доцент		
Нормоконтроль	ЛИПОВЕЦЬ М.В., асистент		

7. Дата видачі завдання « 04 » квітня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

(підпис) Пилип ГАНОШЕНКО
(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис) Тетяна ЗАЙКО
(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 102 с., 2 табл., 46 рис., 4 дод., 17 джерел.

ВЕБСАЙТ, ПРОДАЖ НЕРУХОМОСТІ, WEB, INTERNET, NODEJS, EXPRESS, GRAPHQL, REACT, POSTGRESQL, SQL.

Об'єкт дослідження – програмні засоби з продажу нерухомості.

Предмет дослідження – вебзастосунки з продажу нерухомості.

Мета роботи – розробка вебзастосунку для розміщення оголошень з продажу нерухомості.

Матеріали, методи та технічні засоби: мова програмування JavaScript, фреймворки React та Express, система управління базами даних PostgreSQL.

Результати. Створено вебзастосунок для розміщення оголошень з продажу нерухомості. У процесі створення був визначений перелік вимог для забезпечення високого рівня досвіду користувача, забезпечення безпеки чутливих даних користувачів. На основі визначених вимог був обраний набір програмних засобів для реалізації та архітектурних рішень для можливості підтримки та доповнення додатку.

Висновки. Розроблено вебзастосунок для розміщення оголошень з продажу нерухомості, створенні систему аутентифікації та авторизації облікових записів користувачів з системою ролей, з урахуванням безпеки чутливих даних і можливістю подальшої підтримки і розширення функціоналу.

Галузь використання. Розроблений застосунок може використовуватися інтернет комерції в галузі продажу нерухомості.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 102 pages, 2 tables, 46 figures, 4 appendixes, 17 sources.

WEBSITE, REAL ESTATE SALE, WEB, INTERNET, NODEJS, EXPRESS, GRAPHQL, REACT, POSTGRESQL, SQL.

Object of study is real estate sales software.

Subject of study are web applications for real estate sales

The purpose of the work is development of a web application for posting real estate ads.

Materials, methods, and tools: programming language JavaScript, frameworks React and Express, database management system PostgreSQL.

Results. A web application has been created for posting real estate ads. In the process of creation, a list of requirements was defined to ensure a high level of user experience, to ensure the security of sensitive user data. Based on the defined requirements, a set of software tools was selected for implementation and architectural solutions for the possibility of supporting and supplementing the application.

Conclusions. A web application has been developed for posting ads for real estate sales, creating a system of authentication and authorization of user accounts with a role system, taking into account the security of sensitive data and the possibility of further support and expansion of functionality.

The field of use. The developed application can be used by Internet commerce in the field of real estate sales.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	8
Вступ.....	9
1 Аналіз предметної області.....	11
1.1 Вебзастосунки з продажу нерухомості.....	11
1.2 Огляд аналогів.....	12
1.3 Висновки за розділом 1.....	16
2 Проєктування архітектури вебзастосунку.....	16
2.1 Вибір БД та СУБД.....	17
2.2 Проєктування моделі даних.....	21
2.3 Програмні засоби для створення API-серверу.....	23
2.4 Проєктування серверу аутентифікації.....	25
2.5 Проєктування клієнту та вибір програмних засобів.....	28
2.6 Висновки за розділом 2.....	30
3 Реалізація вебзастосунку.....	32
3.1 Ініціалізація проєкту та реалізація API-серверу.....	32
3.2 Реалізація серверу аутентифікації.....	36
3.3 Реалізація клієнтської частини.....	38
3.4 Висновки за розділом 3.....	47
4 Експлуатація і тестування вебзастосунку.....	48
4.1 Опис програми.....	48
4.2 Експлуатація вебзастосунку.....	48
4.3 Тестування вебзастосунку.....	54
4.4 Висновки за розділом 4.....	55
Висновки.....	56
Перелік джерел посилань.....	57

Додаток А Технічне завдання..... 59

Додаток Б Опис програми..... 64

Додаток В Текст програми..... 67

Додаток Д Слайди презентації..... 94

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

API – Application Programming Interface;

CLI – Command-Line Interface;

CORS – Cross-Origin Resource Sharing;

HTTP – Hypertext Transfer Protocol;

SPI – Single Page Application;

URL – Uniform Resource Locator;

БД – база даних;

ПЗ – програмне забезпечення;

СУБД – система управління базами даних.

ВСТУП

У сфері продажу нерухомості Інтернет відіграє важливу роль, оскільки дозволяє продавцям досягти більшої аудиторії та прискорити процес продажу. Деякі з переваг використання інтернету для продажу нерухомості включають.

Розміщення оголошень на сайтах нерухомості та використання соціальних мереж для просування продажу дозволяє залучити більше потенційних покупців. Використання інтернету для продажу нерухомості може прискорити процес продажу, оскільки покупці можуть переглядати фотографії та опис об'єкта нерухомості в будь-який час, не виходячи з дому.

Інтернет дозволяє продавцю керувати продажем нерухомості за допомогою онлайн-інструментів, таких як електронна пошта та месенджери. Це зручно для продавців, які для зустрічей із потенційними покупцями.

Розміщення оголошень на сайтах нерухомості є дешевшим, ніж традиційна реклама та дозволяє швидко обмінюватися інформацією між продавцями та потенційними покупцями. Це дозволяє швидко вирішувати питання та прискорювати процес продажу.

В цілому, використання інтернету для продажу нерухомості є ефективним та зручним способом залучення потенційних покупців та прискорення процесу продажу.

Одним з основних аспектів у додатку для продажу нерухомості, є забезпечення безпеки оскільки люди розміщують у таких додатках конфіденційну інформацію, таку як контактні дані, фотографії та описи своїх нерухомих об'єктів, а також фінансові дані. Необхідно забезпечити механізм аутентифікації користувачів, щоб запобігти несанкціонованому доступу до програми. Це може бути досягнуто шляхом використання паролів, пін-кодів, відбитків пальців або інших методів аутентифікації. В цілому, забезпечення безпеки у додатку для продажу нерухомості дуже важливо, щоб захистити особисту інформацію користувачів та запобігти потенційним загрозам безпеці.

Використання сучасних технологій та методологій створення ПЗ є важливим для створення додатків які можуть забезпечити безпеку користувачів та можливість підтримки та додавання нового функціоналу.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Вебзастосунки з продажу нерухомості

Сфера продажу нерухомості через інтернет сьогодні є дуже важливим напрямком, який продовжує зростати. Інтернет надає великі можливості для маркетингу нерухомості та зв'язку з клієнтами. Використання інтернету є дуже важливим фактором і допомагає продавцям та агентам нерухомості досягти більшої аудиторії та збільшити ефективність продажів.

Соціальні мережі є одним із способів залучення нових клієнтів. За допомогою соціальних мереж, таких як Facebook, Instagram або LinkedIn, можна просувати нерухомість, ділитися інформацією та спілкуватися з клієнтами.

Але незважаючи на те, що вони є корисним інструментом для інтернет-продажів нерухомості, вони також мають свої недоліки. Ви не можете гарантувати, що ваша цільова аудиторія буде присутня саме на цих платформах. Продаючи нерухомість ви орієнтуєтеся на клієнтів старшого віку. Тому представленість вашої цільової аудиторії може бути меншою на соціальних мережах, орієнтованих на молодших користувачів.

Соціальні мережі можуть мати обмежену функціональність для продажу нерухомості. Наприклад, вам може знадобитися показати більш детальну інформацію про нерухомість, ніж це можливо на соціальних мережах. Також вони переповнені інформацією, що може призвести до того, що ваше оголошення чи реклама загубиться серед інших постів.

Також ви стаєте залежним від алгоритмів – багато соціальних мереж використовують алгоритми, які можуть визначати, які пости будуть відображатися в стрічці користувача. Це може призвести до того, що ваша реклама не досягатиме бажаної аудиторії.

Альтернативою виступають спеціальні платформи розроблені саме для продажу нерухомості. Вони можуть надавати більші можливості для продажу нерухомості, ніж соціальні мережі.

На сайтах з продажу нерухомості можна розміщувати докладнішу інформацію про нерухомість, таку як фотографії, відео, планування, описи, технічні характеристики, розташування, а також контактні дані продавця.

Також на сайтах продажу нерухомості є можливість налаштувати більш точні критерії пошуку для пошуку конкретного типу нерухомості, наприклад, вибрати певний район, тип об'єкта нерухомості, кількість кімнат та інші фільтри.

На сайтах з продажу нерухомості покупці можуть отримати більш детальну інформацію про нерухомість та продавця, що може підвищити рівень довіри до продавця.

1.2 Огляд аналогів

Для визначення основних критеріїв для створення вебзастосунку з продажу нерухомості потрібно оглянути наявні аналоги з метою виділення особливостей та плюсів і мінусів кожного з них.

OLX є одним із найпопулярніших сайтів в Україні для оголошень про продаж та оренду нерухомості [1]. Як можна побачити зайшовши на сайт OLX, він не є спеціалізованим сайтом з продажу об'єктів нерухомості, а спеціалізується на продажу різних речей. З цього витікає проблема обмеженої функціональності необхідної для продажу нерухомості, оскільки критерії пошуку є більш загальними.

Хоча сайт має велику кількість фільтрів для пошуку необхідного оголошення, цей список не є повним (рис. 1.1). На сайті представлена велика кількість оголошень, які можуть бути низької якості, беззмістовні або мати нечіткі фотографії, що ускладнює пошук якісної нерухомості. Також відсутня верифікація продавців.

Серед плюсів можна виділити безкоштовність, що дозволяє розміщувати оголошення про продаж нерухомості безкоштовно, що може бути дуже вигідним для продавців. Також сайт надає простий та зрозумілий

інтерфейс для розміщення оголошень, що робить процес продажу нерухомості на сайті дуже зручним.

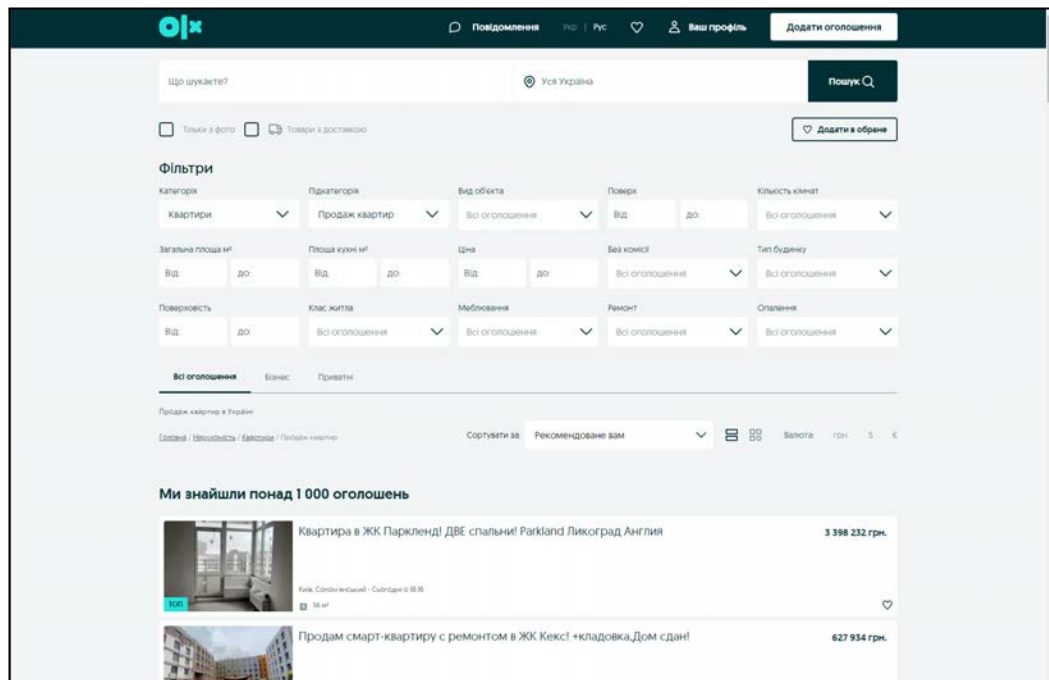


Рисунок 1.1 – Сайт з розміщення оголошень OLX [2]

Dom Ria є одним із найбільш популярних сайтів для продажу нерухомості в Україні, де представлений широкий вибір нерухомості у різних містах України (рис. 1.2) [1].

Сайт надає зручний пошук, який забезпечується великою кількістю фільтрів, що дозволяє швидко знаходити потрібну нерухомість. На сайті представлена докладна інформація про кожен об'єкт нерухомості, що дозволяє покупцю отримати повне уявлення про нерухомість.

З мінусів можна виділити те, що розміщення оголошень на сайті платне, що може бути незручним для продавців. Сайт є більш орієнтованим на розміщення оголошень від професійних ріелторів.

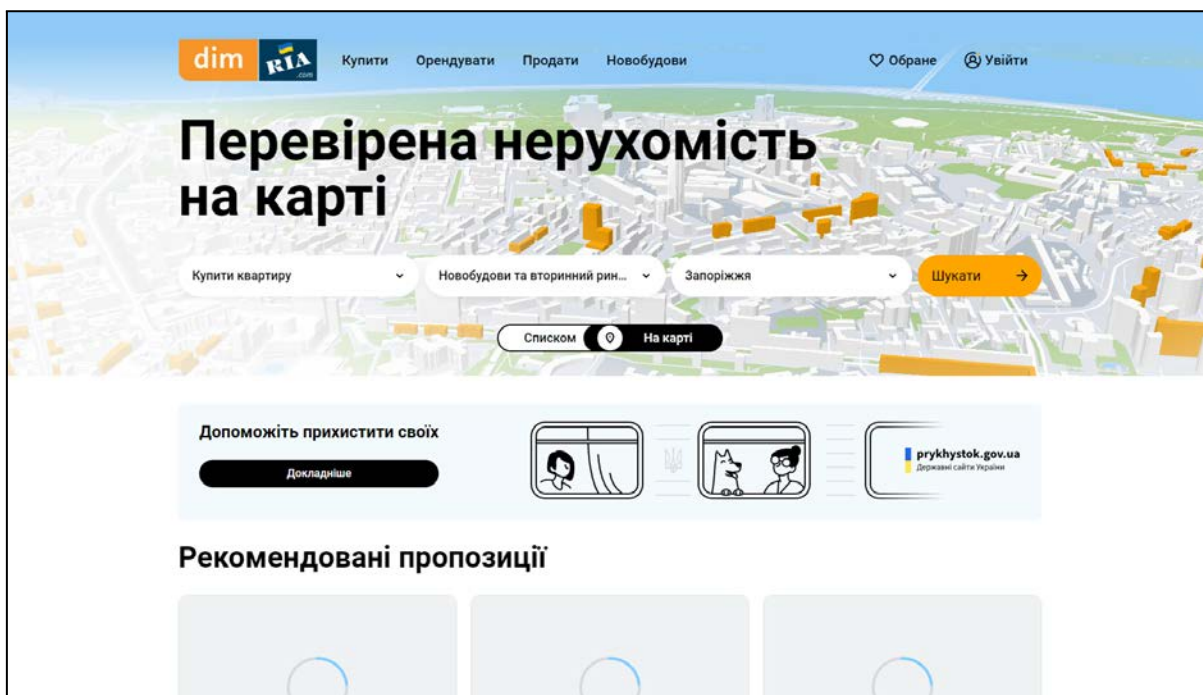


Рисунок 1.2 – Сайт з продажу нерухомості Dom RIA [3]

При порівнянні наведених вище сайтів можна виділити ряд відмінностей. Dom RIA має більш професійний та зручний інтерфейс, який забезпечує кращу взаємодію з користувачами. Однак, OLX має більш простий та легкий у використанні інтерфейс, що може бути більш привабливим для нових користувачів. Dom RIA надає більш детальну та точну інформацію про кожен об'єкт нерухомості, включаючи фотографії, планування, описи тощо. OLX може бути не таким точним і детальним в описі об'єктів нерухомості.

Dom RIA надає професійні послуги для агентств нерухомості, включаючи можливість створення віртуальних турів, виділених сторінок та персоналізованих послуг. OLX, з іншого боку, більше підходить для приватних осіб, і не надає таких професійних послуг.

Dom RIA пропонує більш детальну інформацію про кожен об'єкт нерухомості, таку як кількість кімнат, площа, наявність балкона та інші деталі, що робить сайт більш функціональним для покупців та продавців нерухомості, та надає можливість більш детально з

авдати фільтрацію списку об'єктів нерухомості (рис. 1.3). OLX орієнтований більше на приватних осіб, і інформація про нерухомість може бути обмежена.

Рисунок 1.3 – Пошукові фільтри Dom.Ria [3]

Підсумовуючи все наведене вище можна виділити ряд переваг спеціалізованих сайтів з продажі нерухомості через загальних платформ з можливістю публікації оголошень:

- більша цільова аудиторія: люди, які відвідують спеціалізовані сайти нерухомості, зазвичай вже зацікавлені в купівлі або продажу нерухомості. Це означає, що реклама на таких сайтах буде націлена саме на тих людей, які шукають нерухомість, а не випадкових відвідувачів;
- більш зручний пошук і фільтрація: на спеціалізованих сайтах нерухомості є більш зручний і точний пошук, який дозволяє користувачам налаштувати фільтри і знайти саме ті об'єкти, які їм потрібні;
- більш точні та докладні дані: спеціалізовані сайти з нерухомості зазвичай надають докладнішу інформацію про об'єкти, таку як розміри

кімнат, наявність балкона або паркування, вартість комунальних послуг тощо. Це допомагає покупцям зробити більш поінформований вибір;

– велика довіра: люди, які шукають нерухомість, часто вважають за краще використовувати перевірені та надійні джерела інформації. Спеціалізовані сайти з нерухомості можуть надавати більш високий рівень довіри, оскільки вони мають експертизу в галузі нерухомості та надають більш точну та докладну інформацію.

1.3 Висновки за розділом 1

Проаналізовані різні програмні застосунки з продажу нерухомості. Зокрема розглянуті спеціалізовані застосунки, загальні сайти з оголошеннями та соціальні мережі.

На основі виявлених особливостей виділений ряд переваг спеціалізованих сайтів в порівнянні з іншими платформами. Основною з цих переваг є більше деталізований функціонал для опису нерухомості, що дозволяє покупцям швидко знаходити задовільні оголошення, а продавцям та ріелторам більш детально описати пропозицію.

Визначений основний функціонал який потрібно реалізовувати в застосунку, під критерії якого будуть визначені технології для розробки.

2 ПРОЄКТУВАННЯ АРХІТЕКТУРИ ВЕБЗАСТОСУНКУ

Для забезпечення підтримки і масштабування вебзастосунків повинен мати мікросервісну архітектуру, яка також може забезпечити повторне використання сервісів.

Вебзастосунок буде складатися з наступних частин:

- API-сервер, який реалізує бізнес-логіку додатку та база даних, яку він використовує;
- сервер аутентифікації, який використовується клієнтом для отримання токена за яким API-сервер може аутентифікувати та авторизовувати користувачів;
- клієнт, який використовує сервер аутентифікації для аутентифікації користувачів та API-сервер для отримання даних і маніпуляції з ними.

2.1 Вибір БД та СУБД

Визначимося з основними сутностями з якими взаємодіє вебзастосунок з продажу нерухомості. В основному він працює з двома основними сутностями: користувач і об'єкт нерухомості.

Сутність користувача повинна повинна утримувати в собі набір полів необхідний для аутентифікації та авторизації користувача, а також набір полів контактних даних для надання можливості іншим користувачам отримати їх, для можливості подальшого зв'язку.

Сутність об'єкта нерухомості має поля що містять інформацію щодо щодо опису нерухомості, такі як площа, опис, кількість кімнат тощо, та користувача, що додав до БД екземпляр цієї сутності.

Визначившись з основними сутностями можливо виділити ряд критеріїв, яких потрібно дотримуватись при виборі БД:

- у наведених вище сутностей жорстка структура даних, тобто

однакові екземпляри сутностей мають однаковий набір полів, сутності не змінюються з плином часу;

- велика гнучкість масштабування не потрібна, оскільки вебзастосунок не повинен працювати з великим та безперервним потоком даних, тобто для обробки великої кількості запитів від користувачів достатньо горизонтального масштабування, шляхом додавання об'єму оперативної пам'яті, кількості дисків тощо;

- використання комплексних операції агрегації запитів до БД, що можуть забезпечити взаємодію між сутностями та створюють можливість створення великої кількості представлень самої сутності для використання в різних частинах програми, а також дають можливість створення агрегаційних запитів для проведення аналітики;

- забезпечення великої швидкості розробки та розробницького досвіду, оскільки швидкість розробки є одним з основних пріоритетів бізнесу, а розробницький досвід покращує якість коду;

- наявність екосистему і підтримки, що надає розробнику широкий вибір інструментів та готових рішень для маніпуляцій з даними.

Визначившись з основним набором критеріїв, порівняємо бази даних двох основних моделей: реляційну та нереляційну.

Реляційні бази даних надають сильно структуровану схему даних з певними правилами та зв'язками між таблицями. Це особливо корисно, коли дані мають чітко визначені стосунки та вимагають суворого контролю цілісності. Вони можуть добре масштабуватися вертикально, а також горизонтально. Однак, деякі системи реляційних баз даних мають обмеження на горизонтальне масштабування і можуть мати проблеми продуктивності при дуже великих навантаженнях. Реляційні бази даних мають потужну мову структурованих запитів SQL, яка дозволяє виконати складні операції об'єднання, агрегації, фільтрації та сортування даних. Вони добре підходять для програм, де потрібно проводити складний аналіз даних або формувати звіти [4].

Нереляційні бази даних (також відомі як NoSQL) пропонують гнучку модель даних, де немає суворої схеми. Дані зберігаються у вигляді колекцій, документів, графів або широких стовпців, що дозволяє гнучкіше моделювати неструктуровані або напівструктуровані дані. Нереляційні бази даних, такі як бази даних ключ-значення, документні бази даних і графові бази даних, зазвичай мають гарну горизонтальну масштабованість. Нереляційні бази даних зазвичай мають прості інтерфейси запитів, які завжди підтримують складні операції агрегації і об'єднання даних [5]-[6].

Для наглядного порівняння у табл.2.1 наведена відповідність кожного виду БД визначеним раніше критеріям.

Таблиця 2.1 – Порівняння реляційних та нереляційних баз даних

Критерій	Реляційні БД	Нереляційні БД
Сильно структурована схема даних	Так	Ні
Гнучкість масштабування бази даних	Вертикальне та горизонтальне масштабування	Горизонтальне масштабування
Комплексні операції агрегації та гнучкість запитів	Так	Прості запити
Велика швидкість розробки	Ні	Так
Розвинена екосистема та підтримка	Так	Менш розвинені

З таблиці видно, що реляційні бази даних більш підходять для реалізації поставлених задач.

Визначившись з типом, визначаємо ряд критеріїв для порівняння наявних систем управління реляційними базами даних:

- ліцензування, вартість та наявність безкоштовної ком'юніті версії дуже важливі фактори для створення тестових або некомерційних проєктів;

- продуктивність та швидкість роботи;
- підтримка шифрування даних та керування доступом, що забезпечують безпеку чутливих даних користувачів;
- оптимізація та планування запитів;
- розширюваність та наявність рішень для додаткових можливостей, підтримки комплексних типів даних тощо.

За визначеними критеріями у табл.2.2 порівняно ряд широко використовуваних у веброзробці СУБД: MySQL, PostgreSQL, MS SQL Server та Oracle Database [7].

Таблиця 2.2 – Порівняння СУБД

Критерій	MySQL	PostgreSQL	MS SQL Server	Oracle Database
Наявність безкоштовної версії	Так	Так	Ні	Ні
Швидкість роботи	Хороша	Висока	Висока	Дуже висока
Шифрування даних та керування доступом	Обмежено	Так	Так	Так
Оптимізація та планування запитів	Обмежено	Так	Так	Так
Розширюваність та підтримка комплексних типів даних	Обмежено	Так	Так	Так

З наведеної таблиці порівнянь зазначених раніше СУБД можемо зробити висновок, що найкращим вибором є PostgreSQL. Вона надає задовільний рівень швидкість роботи і продуктивність, оптимізацію, безпеку даних і при цьому є безкоштовною.

2.2 Проектування моделі даних

У вебзастосунку з продажу нерухомості маємо дві основні сутності: користувач і об'єкт нерухомості, при цьому користувач може бути автором об'єкта нерухомості. Автор у об'єкта нерухомості тільки один. Маємо залежність користувача до об'єкта нерухомості, як один до багатьох. Кожній сутності буде відповідна таблиця.

Визначимо основні поля сутності користувача та їх властивості у форматі: назва – тип даних – unique (якщо поле повинне бути унікальним) – pk (primary key – первинний ключ) – nullable (якщо поле може бути порожнім):

- id – uuid – unique – pk;
- email – text;
- name – text;
- password – text;
- activated – boolean;
- phone – text;
- lastOnline – timestamp with timezone – nullable.

Поле activated відображає чи активований обліковий запис. Поле lastOnline відображає дату останньої аутентифікації користувача.

Визначимо основні поля сутності об'єкта нерухомості та їх властивості у форматі: назва – тип даних – unique (якщо поле повинне бути унікальним) – pk (primary key – первинний ключ) – nullable (якщо поле може бути порожнім):

- id – uuid – unique – pk;
- author_id – uuid;
- title – text;
- description – text;
- images – text[];
- address – text;

- priceUAH – numeric;
- createdAt – timestamp with timezone;
- type – text;
- rooms – integer;
- livingAreaM2 – numeric;
- kitchenAreaM2 – numeric.

Поле `author_id` є зовнішнім ключем з'єднаним з відповідним записом сутності користувача. Поле `images` – масив посилань на зображення. Поле `priceUAH` – ціна в грн. Поле `type` – тип нерухомості, який містити значення `flat` або `house`. Поле `livingAreaM2` – житлова площа у кв. м. Поле `kitchenAreaM2` – площа кухні у кв. м.

Таким чином маємо модель даних зі зв'язком між сутностями об'єкта нерухомості (`estate`) і користувача (`user`) (рис. 2.1).

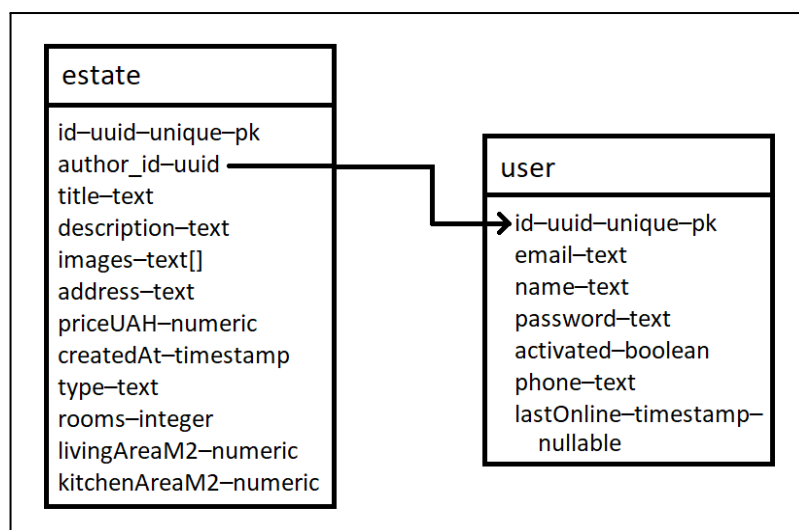


Рисунок 2.1 – Зв'язок між користувачем і об'єктом нерухомості

Також для реалізації функціоналу додавання об'єкта нерухомості до списку обраних нерухомостей користувача введемо додаткову сутність та таблицю обрані (`favorites`). Така таблиця зветься `bridge table`. Вона забезпечує нам залежність багато до багатьох [8].

Визначимо основні поля сутності обраних нерухомостей та їх властивості у форматі: назва – тип даних – unique (якщо поле повинне бути унікальним) – pk (primary key – первинний ключ) – nullable (якщо поле може бути порожнім):

- id – uuid – unique – pk;
- user_id – uuid;
- estate_id – uuid.

Поле user_id є зовнішнім ключем до відповідного запису користувача. Поле estate_id – зовнішній ключ до запису об'єкта нерухомості.

Разом маємо наступну модель даних та зв'язок між сутностями користувача (user), об'єкта нерухомості (estate) і сутності доданого до обраних запису користувача запису об'єкта нерухомості (favorites) (рис. 2.2).

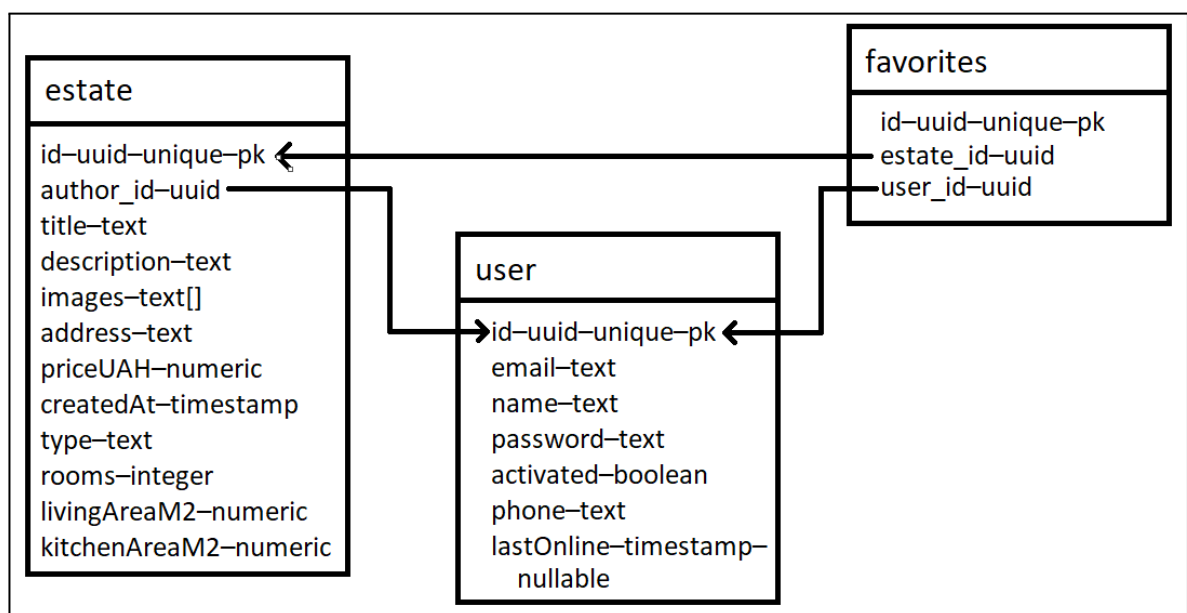


Рисунок 2.2 – Зв'язок між користувачем, об'єктом нерухомості і сутністю нерухомості обраною користувачем

2.3 Програмні засоби для створення API-серверу

Для реалізації взаємодії між клієнтською та серверною частинами застосунку зазвичай використовують два найпопулярніші варіанта: створення

REST API (Representational State Transfer) або використання мови запитів GraphQL.

Виділимо ряд відмінностей REST API і GraphQL:

- при використанні REST API клієнт надсилає запити на певні кінцеві точки, які називаються ендпоінти, та отримують набір даних, визначених сервером. Кожен кінцевий шлях має фіксовану структуру та повертає зумовлені дані. GraphQL клієнт надсилає один гнучкий запит, вказуючи лише ті дані, які їм потрібні. GraphQL дозволяє клієнтам точно визначити структуру та глибину даних, які вони хочуть отримати, та сервер повертає лише запитані дані;

- REST API клієнт отримує фіксований набір даних, включаючи всі поля, визначені на сервері. Це може призвести до надмірної передачі даних, оскільки клієнт можуть не потребувати всі поля визначені сервером. GraphQL дозволяє клієнтам отримувати лише необхідні дані, вказуючи конкретні поля. Це зменшує обсяг даних, що передаються;

- REST API клієнт може потребувати набір даних з різних ендпоінтів, що збільшує кількість запитів до серверу. GraphQL дозволяє отримати повний набір даних одним запитом;

- при зміні REST API потрібне додавання нових ендпоінтів або зміна існуючих, що може становити складнощі для клієнтів, які залежать від конкретних ендпоінтів. GraphQL використовує явне визначення схеми, яка описує типи даних, операції та зв'язки. Це дозволяє гнучко еволюціонувати API, додавати або видаляти поля без необхідності зміни клієнтського коду.

Визначивши відмінності REST API від GraphQL можемо визначити, який підхід більше підходить для створюваного застосунку. Хоча у даному випадку підтримка версіонування API для підтримки змін клієнтами без зміни коду нам не потрібна, оскільки клієнт буде лише один, гнучкість схеми даних може стати вирішальним фактором, оскільки прискорює швидкість розробки. А також оскільки у застосунку пуде декілька представлень однієї сутності, які будуть потребувати різний набір полів. Отже задовільним

вибором стане використання мови запитів GraphQL.

Для створення GraphQL-серверу використовуємо Hasura GraphQL Engine, який автоматично створить на основі визначених в базі даних таблиць та полів можливі запити на отримання, створення оновлення та видалення даних.

Також Hasura надає можливості керування доступом та безпекою даних. Розробники можуть визначити правила авторизації та аутентифікації на рівні бази даних, що забезпечує гранулярний контроль за доступом до даних.

2.4 Проєктування серверу аутентифікації

Для аутентифікації користувачів використаємо JWT (JSON Web Token), який на даний момент є стандартом в веброзробці.

JWT (JSON Web Token) – це компактний формат передачі інформації між сторонами як JSON-об'єкта. Він використовується для аутентифікації та авторизації у вебдодатках та API.

JWT складається з трьох основних компонентів:

- заголовок (Header): Заголовок містить інформацію про тип токена (JWT) і алгоритм шифрування, що використовується: HMAC, RSA тощо;
- корисне навантаження (Payload): Корисне навантаження містить твердження (claims), які являють собою твердження про користувача або інші метадані. Затвердження можуть включати інформацію, таку як ідентифікатор користувача, роль, термін дії токена та інші додаткові дані;
- підпис (Signature): Підпис створюється шляхом кодування заголовка, корисного навантаження та секретного ключа або приватного ключа, у разі використання асиметричного шифрування. Вона використовується для перевірки цілісності даних та аутентифікації відправника.

Коли користувач успішно аутентифікується на сервері, йому

генерується JWT, який може бути переданий клієнту. Клієнт може включити JWT в кожен наступний запит до сервера як аутентифікаційний токен, зазвичай в заголовку Authorization або в Cookie. Сервер перевіряє справжність і цілісність токена, розшифровує корисне навантаження і виконує необхідні перевірки на основі інформації, що міститься, такі як перевірка закінчення терміну дії, перевірка дозволів доступу та інші перевірки без необхідності звернення до бази даних [9].

Для забезпечення безпеки визначений термін дії токена повинен становити як можна менший термін. Оптимально використовувати 5-10 хвилин. Але в такому разі по закінченню терміна дії, користувач повинен буде заново входити в обліковий запис. Для того щоб вирішити цю проблему можна використовувати пару токенів: токен доступу (access token) і токен оновлення (refresh token) [10]-[11].

Коли користувач успішно завершив процес аутентифікації на сервері, йому видається пара токенів: токен доступу та токен оновлення. Токен доступу зазвичай має обмежений термін дії, після чого він стає недійсним. Токен оновлення, у свою чергу, має тривалий термін дії та використовується для отримання нового access token без необхідності повторної автентифікації користувача.

Коли термін дії токена доступу закінчується, клієнт відправляє токен оновлення на сервер. Сервер перевіряє валідність refresh token і, якщо він дійсний, генерує нову пару токенів: новий токен доступу та новий токен оновлення.

Виділимо дії користувача, обробку яких потрібно реалізувати:

- логін – отримання користувачем пари токенів, використовуючи облікові дані: електронна пошта та пароль. При отриманні сервером запиту на аутентифікацію, він валідує дані, отримує з бази даних запис користувача за електронною поштою, перевіряє чи співпадає хеш паролю зі значенням хешу з бази даних. Якщо все добре то відповідає зі статусом 200 та парою токенів. Токен доступу знаходиться в тілі відповіді, а токен оновлення

відправляється `http-cookie` зі шляхом (`path`), який дорівнює шляху ендпоінту оновлення. Сервер робить запис ключа оновлення в сховище ключей. При виникненні помилок відповідає статусом 400 та об'єктом помилки;

- реєстрація – створення запису користувача в базі даних. Сервер отримує облікові дані користувача: електронну пошту, пароль, ім'я, номер телефону тощо, валідує дані, перевіряє чи створено вже обліковий запис зі вказаною електронною поштою та створює запис в базі даних. Пароль при цьому хешується хеш функцією `bcrypt` для того щоб не зберігати паролі у відкритому вигляді. Поле `activated` має значення `false`, а також на вказану електронну пошту відправляється посилання на активацію облікового запису. Якщо все добре відповідає зі статусом 200. У разі помилки відповідає статусом 400 та об'єктом помилки;

- активація облікового запису відбувається за проходженням за посиланням відправлений на електронну пошту вказану при реєстрації. На сторінці активації відправляється запит до серверу з токеном вказаним в посиланні. Перевіряється валідність токена, отримується ідентифікатор користувача та оновлюється поле `activated` користувача з отриманим ідентифікатором на значення `true`. Сервер відповідає статусом 200 при активації та статусом 400 та об'єктом помилки при помилці;

- оновлення – отримання користувачем нової пари токенів, використовуючи токен оновлення. Користувач відправляє на сервер запит з токеном оновлення встановленим при логіні в `http-cookie`. Сервер отримує значення токена, валідує його та перевіряє чи є запис цього токена в сховищі токенів оновлення. Видаляє токен зі сховища та створює нову пару токенів. Новий токен додається у сховище. Повертає нову пару токенів користувачу зі статусом 200, інакше повертає об'єкт помилки за статусом 400;

- вихід з облікового запису виконується шляхом отримання та валідацією відправленого користувачем токена оновлення. Видалення відправленого користувачем токена оновлення зі сховища даних та видалення цього токена з `http-cookie`. Якщо все гаразд повертає статус 200, інакше –

статус 400 та об'єкт помилки. Також клієнт повинен видалити користувача та токен доступу зі свого сховища.

Отриманий від сервера аутентифікації токен користувач використовує при відправленні запитів до API-сервер, додаючи його в заголовок Authorization з префіксом Bearer. Якщо API-сервер відповідає статусом 401, то клієнт повинен виконати оновлення токенів.

Для створення серверу аутентифікації використаємо платформу Node.js. Виділимо ряд переваг платформи Node.js:

- Node.js працює на стороні сервера та використовує JavaScript як мову програмування, що дозволяє використовувати одні і ті самі реалізації на стороні як клієнта так і сервера;
- Node.js використовує механізм однопотокової обробки подій та неблокуючий введення/виведення, що забезпечує високу продуктивність та масштабованість серверних додатків. Це особливо корисно для серверів аутентифікації, які повинні обробляти велику кількість запитів на аутентифікацію та авторизацію;
- Node.js має велику екосистему модулів та пакетів, доступних через менеджер пакетів npm. Це дозволяє розробникам використовувати готові модулі для реалізації різних функцій аутентифікації, таких як генерація та перевірка JWT-токенів, хешування паролів, інтеграція із зовнішніми сервісами аутентифікації тощо.

Для обробки HTTP-запитів використаємо фреймворк Express. Вибір цього фреймворку обґрунтований в першу чергу його популярністю та розвиненістю екосистеми та гнучкість. Express пропонує широкий спектр можливостей, і він може бути легко розширений за допомогою різних плагінів та проміжних обробників (middleware).

2.5 Проектування клієнту та вибір програмних засобів

Клієнтська частина вебзастосунку повинна реалізовувати наступні взаємодії з сутностями користувача та об'єктом нерухомості, а також загальні можливості:

- можливість реєстрації користувача;
- можливість входу в обліковий запис користувача;
- можливість активації облікового запису користувача;
- відображення списку нерухомостей з можливостями фільтрації за типом, кількістю кімнат, площею, ціною тощо;
- відображення більш детальної інформації про об'єкт нерухомості;
- можливість створення нового об'єкта нерухомості з вказанням детальної інформації для користувача, який увійшов в обліковий запис;
- можливість редагування створеного об'єкта нерухомості користувачем, який створив її;
- можливість додати нерухомість до списку обраних або видалити з нього, якщо поточний користувач не її автор;
- відображення списку обраних нерухомостей з можливостями фільтрації за типом, кількістю кімнат, площею, ціною тощо, якщо користувач увійшов до облікового запису;
- відображення списку нерухомостей створеного поточним користувачем з можливостями фільтрації за типом, кількістю кімнат, площею, ціною тощо, якщо користувач увійшов до облікового запису;
- можливість зміну мови: українська або російська;
- можливість зміни теми додатку: світла або темна.

Для реалізації клієнтської частини вебзастосунку використаємо фреймворк React. Він дозволяє створювати ефективні та масштабовані вебдодатки, використовуючи компонентний підхід.

Виділимо його основні переваги:

- React ґрунтується на компонентній моделі, де інтерфейс користувача розбивається на незалежні компоненти. Це дозволяє розробляти програми модульно, спрощує підтримку та повторне використання коду;
- React використовує віртуальний DOM, який представляє весь інтерфейс користувача в пам'яті. Реакт оновлює лише змінені компоненти у віртуальному DOM та ефективно оновлює реальний DOM лише за необхідності. Це покращує продуктивність та швидкодія додатків;
- React використовує односпрямований потік даних. При такому підході дані у додатку рухаються від верхнього рівня до нижнього рівня компонентів. Це робить керування станом більш передбачуваним і спрощує налагодження та тестування програм;
- React має велику екосистему. Існує також безліч сторонніх бібліотек та інструментів, які розширюють функціональність React та допомагають у розробці програм.

Для реалізації організації і управління даними в додатку використаємо бібліотек React Query. React Query – це бібліотека для керування станом та обробки даних у React-додатках. Вона надає простий та ефективний спосіб роботи з даними, включаючи запити до сервера, кешування, оновлення даних та обробку стану завантаження.

React Query полегшує виконання запитів до сервера та обробку даних у вашому React-додатку. Він надає зручні хуки для виконання запитів, кешування даних та автоматичного оновлення даних під час зміни. React Query обробляє безліч типових сценаріїв роботи з даними, таких як відображення завантаження, обробка помилок та інвалідування кешу.

2.6 Висновки за розділом 2

У розділі була розглянута і проаналізована загальна структура вебзастосунку.

Була спроектована модель даних, структури та їх взаємодія. Висунуті вимоги до вибору бази даних та системи управління базами даних. Розглянуті та порівняні між собою найбільш популярні варіанти. Як найкращий варіант був обраний PostgreSQL, який більш задовольняє висунуті умови.

Були розглянуті варіанти реалізації API-серверу. Як найбільш задовільний за висунутими вимогами варіант був обран GraphQL. Для реалізації GraphQL серверу був обраний програмний засіб Hasura GraphQL Engine.

Був розглянут найбільш поширений варіант реалізації аутентифікації вебдодатків. На основі його був визначений повний сценарій його використання, на основі якого буде реалізований сервер аутентифікації. Для реалізації буде використана платформа Node.js та фреймворк Express.

Були розглянуті та детально описані. сценарії користувача клієнтської частини. Для реалізації була обрана бібліотека React.

3 РЕАЛІЗАЦІЯ ВЕБЗАСТОСУНКУ

3.1 Ініціалізація проєкту та реалізація API-серверу

Як було визначено у другому розділі застосунок буде складатися з трьох основних компонентів: API-серверу з використанням Hasura GraphQL Engine, серверу аутентифікації та клієнту. Під кожну частину створена директорія. Оскільки проєкт розробляється однією людиною і у додатку лише один клієнт, версіонування застосунку може бути загальним, отже ініціалізуємо Git у кореневій директорії (рис. 3.1).



Рисунок 3.1 – Структура кореневої директорії

Ініціалізуємо Hasura згідно документації. GraphQL-сервер запускається у docker-контейнері. За допомогою утиліти curl був завантажений docker-compose.yaml файл з необхідною на конфігурацією, який містить два образи (image): graphql-engine та postgres. Згідно документації змінна середовища пароля адміністратора була оновлена та додана змінна для визначення визначення ролі та алгоритму шифрування JWT, та секретний ключ до нього. Для збереження міграцій бази даних встановлений Hasura CLI за допомогою пакетного менеджера npm. Додано скрипт start до файлу package.json для запуску консолі. API-сервер запускається за допомогою docker-compose, консоль підключається скриптом npm run start [12].

Через консоль створена база даних з назвою default та схему під назвою public. Створена таблиця users, з визначеними при проєктуванні

полями (рис. 3.2).

CONFIGURE FIELDS

Columns

Edit
id - uuid, primary key, unique, default: gen_random_uuid()

Edit
email - text, unique

Edit
name - text

Edit
password - text

Edit
activated - boolean, default: false

Edit
phone - text, unique

Edit
lastOnline - timestamp with time zone, nullable

Add a new column

TABLE PROPERTIES

Primary Key

Edit
id - users_pkey

Foreign Keys

Add a foreign key

Рисунок 3.2 – Структура таблиці users

Для різних ролей користувачів встановлені різні дозволи (рис. 3.3).

Role	insert	select	update	delete	
admin	✓	✓	✓	✓	
user	✗	⌵	⌵	✗	☐
anonymous	✗	⌵	✗	✗	☐
Enter new role	✗	✗	✗	✗	

Рисунок 3.3 – Дозволи таблиці users

Для неавторизованого користувача (anonymous) встановлено дозволи на лише на отримання даних. Для авторизованого користувача (user) – на

отримання та оновлення даних, але лише для свого облікового запису, тобто запису таблиці ідентифікатор якого дорівнює значенню X-Hasura-User-Id в JWT, який він отримає від серверу аутентифікації (рис. 3.4).

Pre-update check ?

☐ Without any checks

☒ With same custom check as **select**

1 `{"id":{"_eq":"X-Hasura-User-Id"}}`

☐ With custom check: ?

Рисунок 3.4 – Дозволи на редагування користувачем лише свого облікового запису

Користувачу заборонено змінювати поля `activated`, `password`, `lastOnline` та `id`.

Створена таблиця `estate` для об'єктів нерухомості з визначеним при проєктуванні набором полів (рис. 3.5). Поле `author_id` встановлено як зовнішній ключ та зв'язано з записом з відповідним `id` у таблиці `users`. Додані дозволи на отримання даних для неавторизованих та авторизованих користувачів, але заборонений доступ до полів `password` та `activated`. Дозволене створення та оновлення записів для авторизованих користувачів, але заборонене оновлення полів `id`, `author_id` та `createdAt`.

Створено таблицю `favorites` з визначеними при проєктуванні полями. Поля `estate_id` та `user_id` встановлені як зовнішні ключі та зв'язані з відповідними полями в таблицях `estate` та `user` (рис. 3.6). Встановлені необхідні дозволи на отримання, створення і видалення.

Columns

Edit

id - uuid, primary key, unique, default: gen_random_uuid()

Edit

author_id - uuid

Edit

title - text

Edit

description - text

Edit

images - text[], default: '':text[]

Edit

address - text

Edit

priceUAH - numeric

Edit

createdAt - timestamp with time zone, default: now()

Edit

type - text

Edit

rooms - integer

Edit

livingAreaM2 - numeric

Edit

kitchenAreaM2 - numeric

Add a new column

TABLE PROPERTIES

Primary Key ⓘ

Edit

id - estate_pkey

Foreign Keys ⓘ

Edit

author_id → users . id - estate_author_fkey

Add a new foreign key

Рисунок 3.5 – Структура таблиці estate

Columns

Edit

id - uuid, primary key, unique, default: gen_random_uuid()

Edit

user_id - uuid

Edit

estate_id - uuid

Add a new column

TABLE PROPERTIES

Primary Key ⓘ

Edit

id - favorites_pkey

Foreign Keys ⓘ

Edit

estate_id → estate . id - favorites_estate_id_fkey

Edit

user_id → users . id - favorites_user_id_fkey

Add a new foreign key

Рисунок 3.6 – Структура таблиці favorites

3.2 Реалізація серверу аутентифікації

Як було зазначено при проєктуванні: для реалізації використано платформу Node.js та Express – фреймворк для роботи з HTTP [13]-[14]. Для типізації коду використовується мова програмування Typescript, що покращує досвід розробника та забезпечує самодокументованість коду.

Для збірки проєкту використовується Webpack. У створеному файлі конфігурації webpack.config.js вказані точки входу та директорія, в яку Webpack збирає проєкт. До конфігурації Webpack доданий плагін для транспіляції коду з мови Typescript в JavaScript. Створений файл конфігурації Typescript tsconfig.json [15].

Ініціалізований сервер Express запускається на порті, що визначений в змінній оточення. Встановлено проміжні обробники (middleware) для отримання тіла запиту, встановлення заголовків CORS та отримання cookie [16].

При проєктуванні архітектури серверу аутентифікації були визначені основні події які він повинен реалізовувати, а саме: логін, реєстрація, активація, оновлення токенів та вихід з облікового запису. Для кожної дії створений ендпоінт. Наприклад: ендпоінт реєстрації має шлях /api/auth/register. Аналогічні шляхи будуть мати інші обробники.

Всі обробники подій, пов'язані з аутентифікацією, знаходяться у директорії /modules/auth. Така структура спрощує можливість додавання нового функціоналу. Таким чином директорія modules містить бізнес логіку додачу.

Бізнес-логіка додатку використовує сторонні сервіси, такі як доступ до бази даних користувачів, сервіс електронної пошти тощо. Сервіси додатку знаходяться у директорії services.

Як було визначено при проєктуванні маємо наступний набір сервісів:

- users – сервіс взаємодії з базою даних користувачів, який буде використовуватись для додавання нових користувачів, оновлення статусу

activated, та отримання облікових даних користувача для перевірки при вході користувача в обліковий запис;

- refreshTokens – сервіс для зберігання токенів оновлення користувачів, виданих при вході в обліковий запис, який має функції додавання в сховище, пошук токена у сховищі та видалення токена зі сховища;

- emailService – сервіс відправлення електронних повідомлень з посиланням на активацію облікового запису користувача;

- tokens – сервіс, який використовується для підписання токенів та верифікації їх.

Таким чином отримуємо наступну структуру (рис. 3.7).

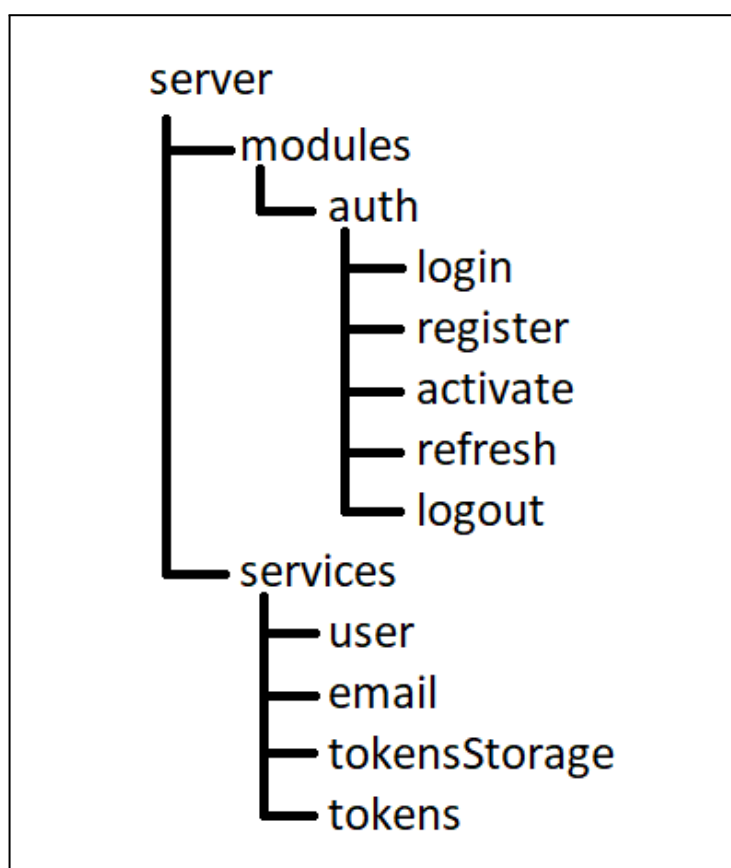


Рисунок 3.7 – Файлова структура серверу аутентифікації

Сервіс users використовує базу даних користувачів API-серверу. Він має повні права доступу бо використовує секретний ключ адміністратора

визначений API-сервером та доданий через змінні оточення.

Сервіс `refreshTokens` зберігає ключі у змінній. Основним мінусом цього підходу є неможливість горизонтального масштабування проєкту. Із плюсів можна виділити простоту імплементації та використання. При необхідності є можливість, завдяки визначеній архітектурі, легко замінити на Redis або аналоги без змін API, який надає сервіс.

Сервіс `tokensStorage` використовує бібліотеку `jsonwebtoken` для видання та верифікації JWT ключів. При виданні токена встановлюється час дії токена, який задається через змінні оточення. Ключі які використовуються для підписання та верифікації токенів також зберігаються у змінних оточення, що дозволяє змінити їх значення без редагування програмного коду. Реалізація має простий API та виконується синхронно.

Реалізуємо функціонал аутентифікації за алгоритмами наведеними при проєктуванні архітектури серверу аутентифікації.

3.3 Реалізація клієнтської частини

React-додаток ініціалізовано за допомогою утиліти `create-react-app` з темплейтом `Typescript`.

Для стилізації та використання готових компонентів і зміни теми додатку використовується бібліотека `Chakra UI`. Весь додаток обернено в провайдер стилей, який дозволяє компонентам використовувати значення встановлені в конфігурації провайдера. Глобальні стилі теж задані в об'єкті конфігурації. Провайдер дозволяє отримувати поточну тему: `light` або `dark`, за допомогою хуків, які надає бібліотека `Chakra UI`.

Для зміни теми реалізовано компонент `ThemeSwitchButton`, який рендерить компонент кнопки, при натисканні якої змінюється тема додатку (рис. 3.8). Компонент `ThemeSwitchButton` доданий до хедеру [17].

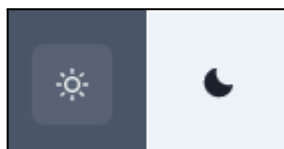


Рисунок 3.8 – Кнопка зміни теми (зліва – на світлу, справа – на темну)

Для зміни мови реалізована бібліотека translations. Вона надає провайдер, в який передається об'єкт, що містить переклади. Переклади зберігаються у файлі translations.json та імпортуються в JavaScript файл з використанням Webpack плагіну, який при збірці проєкту перетворює його на звичайний JavaScript об'єкт (рис. 3.9).

```
{
  "uk": {
    "buyFlat": "Купити квартиру",
    "buyHouse": "Купити будинок",
    "sell": "Продати",
    "login": "Увійти",
    "register": "Зареєструватися",
    "email": "Електронна пошта",
    ...
  },
  "ru": {
    "buyFlat": "Купить квартиру",
    "buyHouse": "Купить дом",
    "buyAll": "Все",
    "sell": "Продать",
    "login": "Войти",
    "register": "Зарегистрироваться",
    "email": "Электронная почта",
    ...
  }
}
```

Рисунок 3.9 – Структура translations.json

Файл перекладів являє собою JSON-об'єкт, що містить поля мов, які в свою чергу містять ключі перекладів.

При ініціалізації до провайдера передається значення `initialLanguage`, яке дорівнює `uk`, що визначає українську початковою мовою.

Для управління переключення мови додатку, реалізований хук `useTranslations`, який повертає об'єкт, що містить наступні поля та методи:

- `selectedLanguage` – містить значення поточної мови;
- функція `selectLanguage` – встановлює нове значення `selectedLanguage`;
- функція `t` – повертає переклад за переданим ключем.

Для зміни мови користувачем, реалізований компонент `LanguageSelect`. Він використовує функцію `selectLanguage` для зміни мови і `selectedLanguage` для відображення поточного значення.

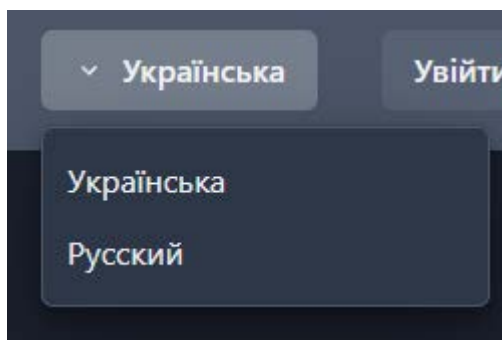


Рисунок 3.10 – Компонент вибору мови `LanguageSelect`

Для надання дозволу до певного функціоналу лише для аутентифікованих користувачів, реалізована бібліотека `auth`. Вона надає провайдер, яким обернений весь додаток, щоб отримувати в середині компонентів доступ до полів, які відображають стан аутентифікації, та методів, що дозволяють виконувати дії аутентифікації. Провайдер приймає API-клієнт, який повинен реалізовувати інтерфейс та використовується у методах аутентифікації для зв'язку з сервером аутентифікації.

Доступ до функціоналу, який надає провайдер, можна отримати у інших компонентах за допомогою хука `useAuth`. Список функціоналу:

- `userId` – ідентифікатор поточного користувача. Якщо користувач

не аутентифікований – дорівнює null;

- `authenticated` – дорівнює `true` коли значення `userId` відмінне від `null`;
- `registered` – метод, що приймає дані користувача, та відправляє запит на реєстрацію на сервер аутентифікації;
- `activate` – метод, що приймає токен на активацію та робить запит на активацію на сервер аутентифікації;
- `login` – метод, що приймає адресу електронної пошти і пароль, та відправляє запит на логін на сервер аутентифікації;
- `refresh` – метод, що відправляє запит до сервера аутентифікації на оновлення токенів;
- `logout` – метод, що відправляє запит на виходу з облікового запису поточного користувача;
- `loadingInitialRefresh` – стан, чи закінчився процес оновлення токенів коли користувач відкриває додаток;
- `notAuthenticatedAfterInitialRefresh` – стан, чи успішно оновлені токени при відкритті додатку.

Для того щоб автоматично аутентифікувати користувача коли він відкриває додаток, якщо він раніше увійшов до облікового запису, при ініціалізації додатку виконується метод `refresh`. При цьому стан `loadingInitialRefresh` дорівнює `true` до завершення запиту до серверу аутентифікації. Якщо користувач був успішно аутентифікований `notAuthenticatedAfterInitialRefresh` дорівнює `true` інакше `false`.

Ці додаткові стани дозволяють контролювати доступ користувача до інтерфейсу, який не повинен бути доступним до виконання оновлення токенів. Загалом у додатку недоступний для неаутентифікованого користувача функціонал розміщений на приватних сторінках, які обробляються компонентом `PrivateRoute`, який теж реалізує створена бібліотека. Але якщо є необхідність додати функціонал для аутентифікованих користувачів, що має бути доступним на публічних сторінках, можна

використовувати зазначені стани.

Згаданим компонентом `PrivateRoute` можна обгорнути компонент сторінки, яка повинна бути приватною. Якщо стан `loadingInitialRefresh` дорівнює `true`, то компонент рендерить індикатор завантаження замість сторінки, інакше, якщо `authenticated` дорівнює `true`, то компонент рендерить сторінку. В іншому випадку, він перенаправляє користувача за адресою сторінки реєстрації за допомогою компонента `NavigateWithRedirect`, який реалізований у бібліотеці `redirect`.

Основний функціонал створеної бібліотеки `redirect` пов'язаний в перенаправленням користувача після виконаної дії до сторінки, з якої він був перенаправлений спочатку. Для цього використовується бібліотека `react-router-dom`, яка в свою чергу базується на `History API` браузера та його об'єкт `state`, в який за ключем `from` вказується початкова адреса. Бібліотека надає компонент `RedirectLink`, який перенаправляє користувача на вказану адресу з можливістю повернення до сторінки з якої він переходить за допомогою використання зазначеного раніше компоненту `NavigateWithRedirect` або функції `redirect`, яку надає хук `useRedirect`.

Також `RedirectLink` приймає флаг `forward`, що означає передання наявного значення ключа `from` об'єкта `state`, замість поточної адреси.

З використання функціоналу створеної бібліотеки `auth` реалізовані сторінки та компоненти пов'язані з виконанням дії з аутентифікації користувача.

Компонент `AuthButton` який використовується в хедері, відображає кнопку Увійти та pop-over з формою логіну, та посиланням на сторінку реєстрації (рис. 3.11).

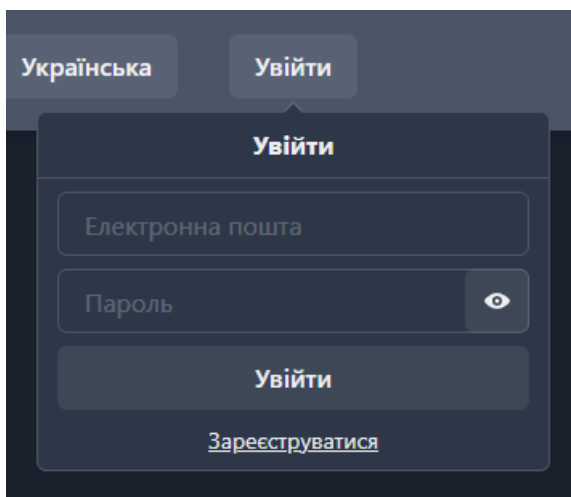


Рисунок 3.11 – Компонент AuthButton

Посилання Зареєструватися являє собою компонент `RedirectLink` зі створеної бібліотеки `redirect`, та перенаправляє користувача на сторінку реєстрації (рис. 3.12). Кнопка Назад використовує функцію `redirect` для повернення до сторінки з якої користувач перейшов до сторінки реєстрації (рис. 3.13).

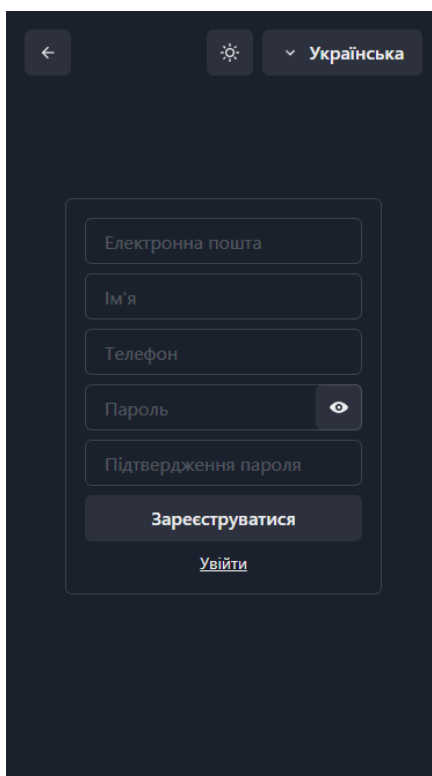


Рисунок 3.12 – Сторінка реєстрації (мобільна версія)

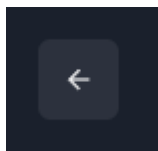


Рисунок 3.13 – Кнопка повернення до початкової сторінки

Для маршрутизації у вебзастосунку використовується бібліотека `react-router-dom`. У конфігу вказані маршрути до всіх сторінок. Приватні маршрути обгорнуті в компонент `PrivateRoute`. Сторінки пов'язані з аутентифікацією заборонені для аутентифікованих користувачів.

Для взаємодії з API-сервером створена бібліотека `api`, а також використовується бібліотека `react-query`. Бібліотека `api` експортує провайдер, який приймає API-клієнт, який реалізує визначений інтерфейс.

API-клієнт повинен реалізувати метод `request`, який приймає об'єкт з полями `query` та `role`, і опціональним полем `variables`. Поле `query` має бути GraphQL-запитом, `variables` – змінними до нього, `role` – роль (`user` або `anonymous`). Поле `role` яке дорівнює `user` використовується на приватних сторінках, до яких мають доступ лише аутентифіковані користувачі.

При реалізації інтерфейсу API-клієнту, використовується бібліотека `axios`, яка дозволяє додати перехоплювачі (`interceptors`) на запити та на відповіді. Для реалізації додавання токена доступу до заголовку `Authorization` з префіксом токена `Bearer` використовується перехоплювач запиту. Тобто коли аутентифікований користувач відкриває приватну сторінку і робиться запит до API-серверу, цей запит перехоплює `axios interceptor`, і до запиту додається заголовок `Authorization`.

Токен доступу має малий час життя, тож коли API-сервер відповідає, що токен доступу застарий, треба відправити запит на оновлення токена і у разі його успішного виконання повторити запит до API-сервера з новим токеном доступу. Для реалізації цього механізму використовується перехоплювач на помилку у відповіді.

Сторінка зі списком нерухомості використовує компонент EstateList, який робить запити до API-серверу (рис. 3.14).

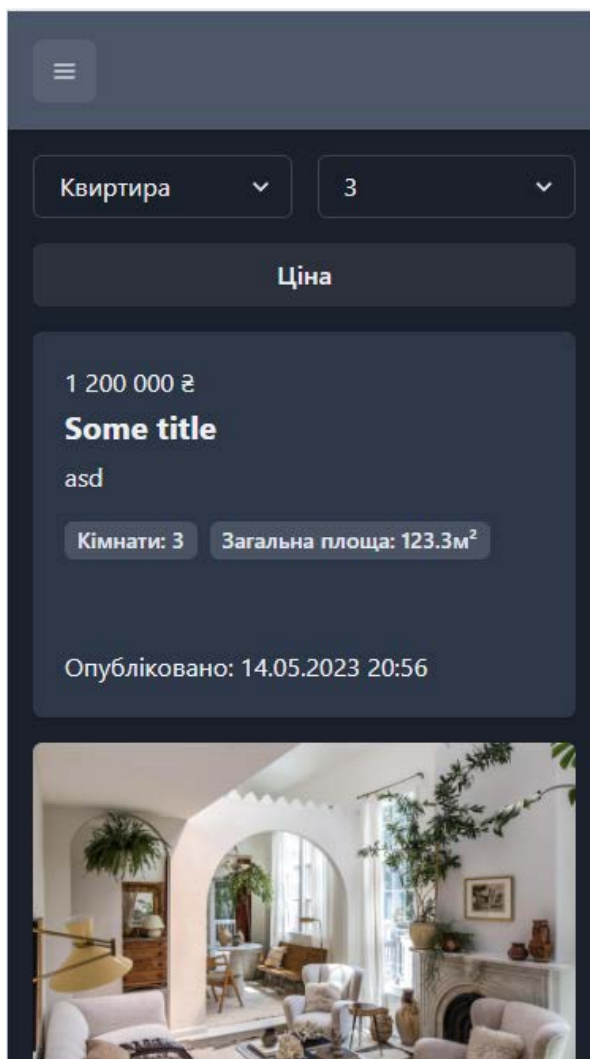


Рисунок 3.14 – Сторінка списку нерухомості

Створений хук `useFilter`, за допомогою якого управляється кожний окремий фільтр. Він використовує `useState` для зберігання значення, а також синхронізує значення з параметром пошуку. Тобто коли фільтр з назвою `type` дорівнює `flat`, то до адресу додається параметр пошуку `type=flat`. А якщо користувач пройде за адресою яка має параметр пошуку `type=flat`, то значення додається до фільтру при ініціалізації компоненту.

Цей механізм дозволяє користувачу відправили комусь URL і той хто зайдет на сторінку буде мати так само налаштовані фільтри, або ж користувач

може знайти сторінку з настроєними фільтрами в історії браузера.

Фільтри які мають два значення, використовуються два фільтри. Наприклад фільтр ціни має мінімальне та максимальне значення, тому він має використовувати два фільтри `priceFrom` та `priceTo` (рис. 3.15).

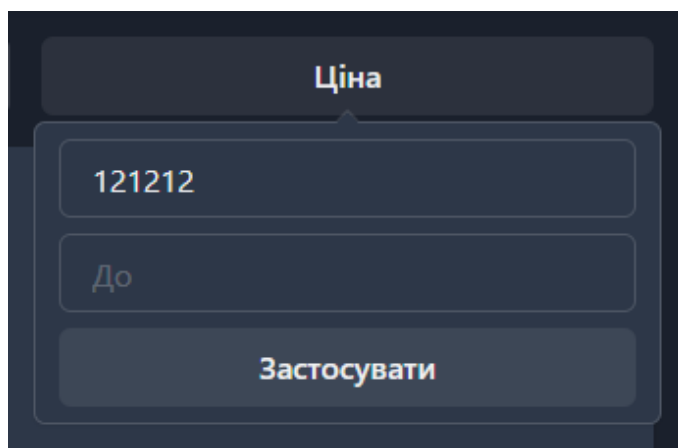


Рисунок 3.15 – Фільтр ціни

На сторінці з детальним описом об’єкту нерухомості використовується бібліотека `swiper` для виводу зображень нерухомості. Також імплементована функція додавання об’єкту нерухомості до обраних. Цю функцію реалізує компонент `FavoritesButton`. Ця функція доступна лише аутентифікованим користувачам, тому для визначення чи потрібно відображати компонент використовується хук `useAuth`, а саме значення `authenticated`.

Якщо поточний користувач є автором оголошення на сторінці інформації якого він знаходиться, то замість компоненту `FavoritesButton` відображається посилання на сторінку редагування інформації цього оголошення.

Форма редагування (рис. 3.16) та форма створення нового оголошення використовують один і той самий компонент `Form`, але по різному обробляють відправку форми.

Рисунок 3.16 – Форма редагування

Після відправки форми користувач переадресовується на сторінку інформації об'єкта нерухомості.

Аутентифіковані користувачі мають доступ до сторінки з обраними оголошеннями та сторінки з оголошеннями створеними самим користувачем, а також то сторінки зі списком обраних оголошень.

3.4 Висновки за розділом 3

За визначеними у другому розділі вимогами та спроектованою архітектурою, реалізований вебзастосунок. Реалізовані три основні частини: API-сервер, сервер аутентифікації та клієнт.

Описані основні моменти пов'язані з ініціалізацією кожної частини. Обґрунтований вибір версіонування додатку. Описана реалізація внутрішньої архітектури кожної частини, а також опис функціоналу та деталей реалізації основних модулів кожної частини.

4 ЕКСПЛУАТАЦІЯ І ТЕСТУВАННЯ ВЕБЗАСТОСУНКУ

4.1 Опис програми

Спроектований та реалізований вебзастосунок дозволяє користувачам переглядати створені іншими користувачами оголошення з продажу об'єктів нерухомості. Спеціалізовані додатки для продажу нерухомості пропонують низку переваг та принесуть користь як покупцям, так і продавцям.

Неаутентифікований користувач має можливість переглянути список існуючих оголошень. Додані фільтри дозволяють користувачу знайти об'єкт нерухомості, який задовольняє його вимогам. Для користувачів доступний перегляд детальної інформації про нерухомість та надається контактна інформація продавця.

Для отримання додаткового функціоналу, користувач може зареєструватися. Аутентифікований користувач має доступ до створення власних оголошень з продажу нерухомості, можливість додати оголошення, яке йому сподобалось, до обраних та переглянути списки обраних оголошень та оголошень, створених їм самим.

Для використання створеного вебзастосунку, користувач повинен мати лише браузер з увімкненою підтримкою JavaScript та підключення до мережі Інтернет. Адаптивний інтерфейс підходить як для використання на комп'ютері, так і для мобільних пристроїв.

4.2 Експлуатація вебзастосунку

Створений вебзастосунок може використовуватися користувачем за допомогою браузера. Користувач, використовуючи телефон або комп'ютер, вводить у адресну строку браузера адресу, за якої розташований додаток.

Користувач потрапляє на головну сторінку, з якої він може потрапити до інших сторінок з функціоналом, який користувач шукав.

У компоненті хедер користувач може обрати мову та тему

вебзастосунку (рис. 4.1).

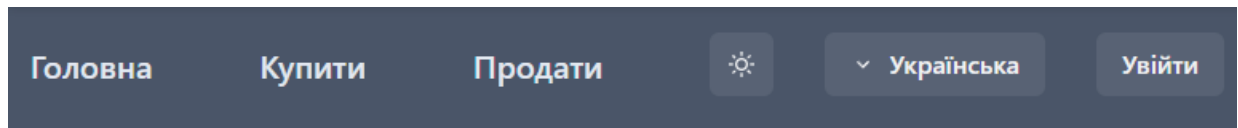


Рисунок 4.1 – Компонент херд

Натиснувши на кнопку купити, користувач може обрати у випадаючому меню тип об'єкту нерухомості, який йому потрібен (рис. 4.2).

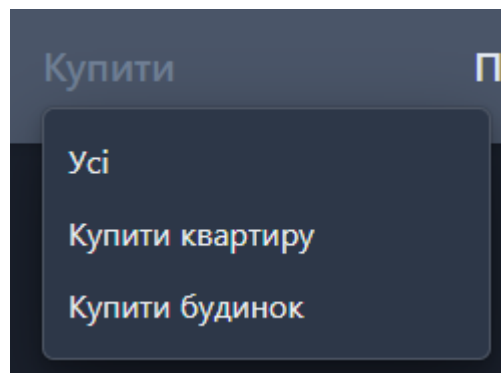


Рисунок 4.2 –Випадаюче меню

Обравши потрібний тип нерухомості, користувач попадає на сторінку зі списком об'єктів нерухомості (рис. 4.3). Одразу буде встановлене значення фільтру типу нерухомості, який обрав користувач у випадаючому меню.

Також користувачу доступні інші фільтри, змінюючи які він отримує оновлений список нерухомості, елементи якого відповідають встановленим значенням фільтрів.

Обравши об'єкт нерухомості, опис якого задовольняє користувача, він може перейти до сторінки з більш детальною інформацією (рис. 4.4).

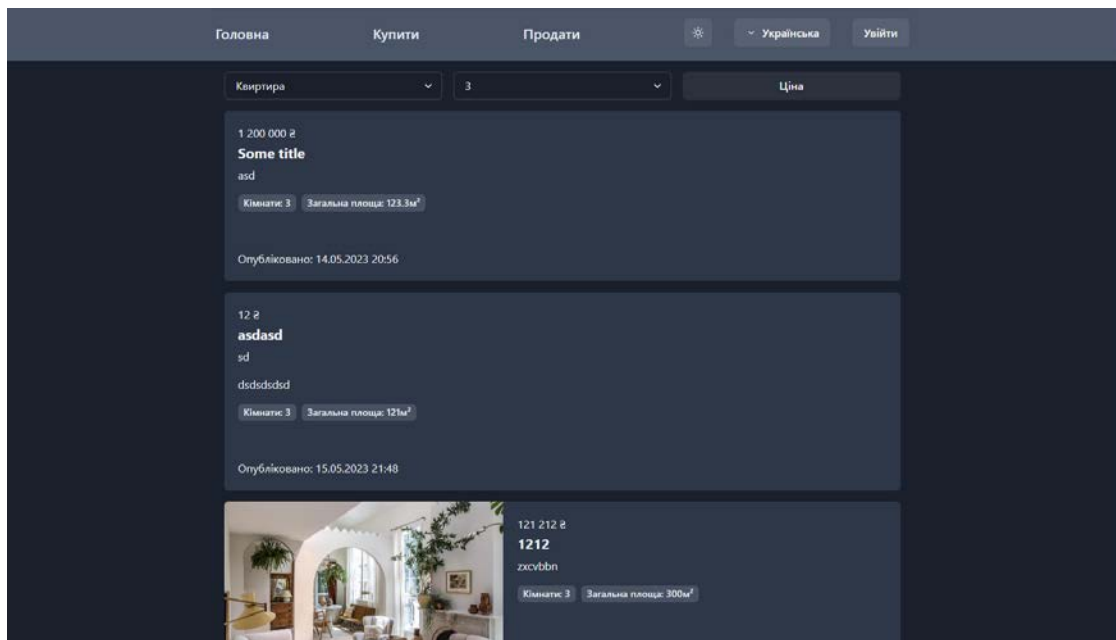


Рисунок 4.3 – Сторінка зі списком нерухомості

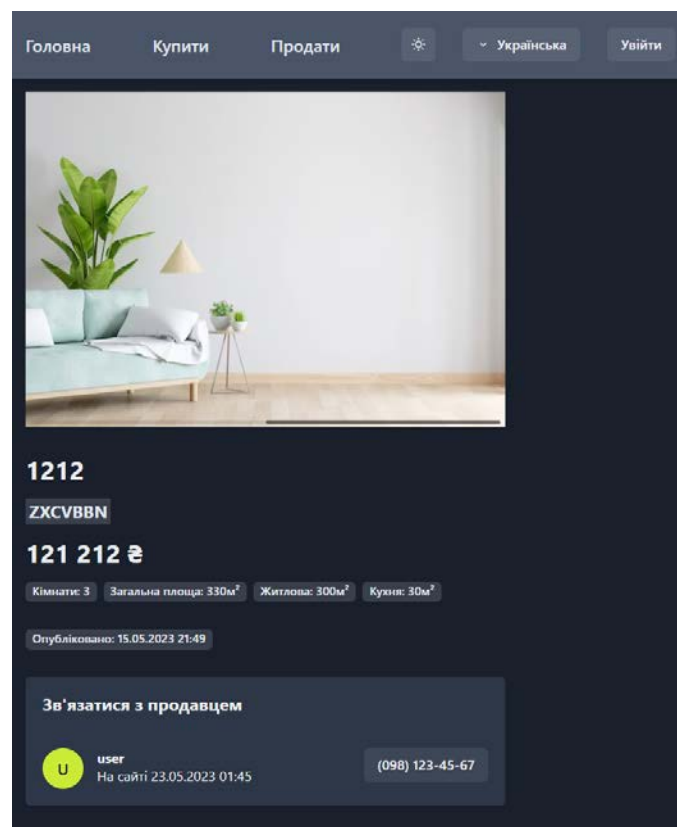


Рисунок 4.4 – Сторінка з інформацією про обраний об'єкт нерухомості

У разі якщо об'єкт нерухомості заінтригував користувача, він може зв'язатися з продавцем інформація про якого вказана нижче на сторінці.

Якщо користувач хоче використати додатковий функціонал, він повинен увійти до облікового запису. Для цього він повинен натиснути на кнопку Увійти, після чого він може використати форму для введення облікових даних для входу в обліковий запис, або натиснути на посилання Зареєструватися (рис. 4.5).

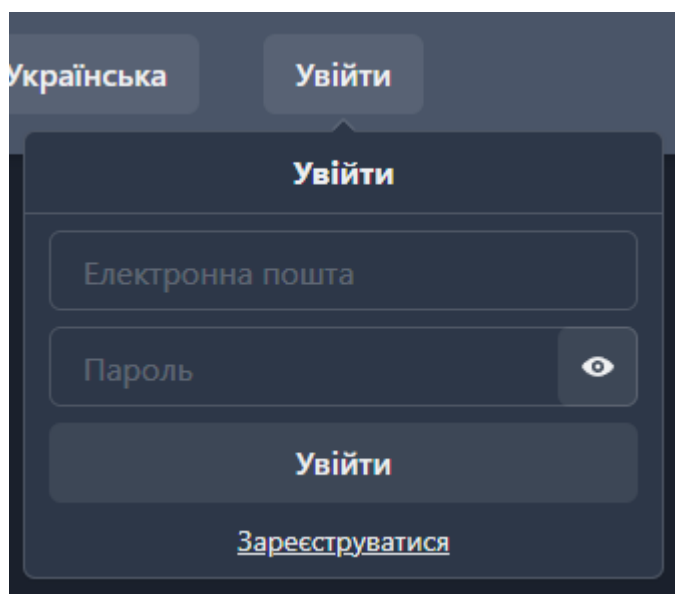
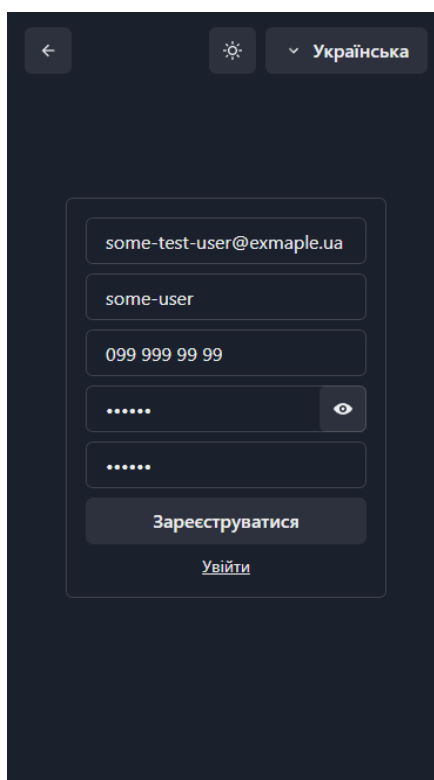
The image shows a login interface with a dark blue background. At the top, there are two buttons: 'Українська' (Ukrainian) and 'Увійти' (Login). Below them is a modal window titled 'Увійти'. Inside this window, there are two input fields: 'Електронна пошта' (Email) and 'Пароль' (Password). The password field has a toggle icon (an eye) to the right of it. Below the input fields is a large 'Увійти' button. At the bottom of the modal, there is a link that says 'Зареєструватися' (Register).

Рисунок 4.5 – Форма входу в обліковий запис

Якщо користувач натискає на посилання Зареєструватися, він буде переадресований на сторінку реєстрації (рис. 4.6). Після введення даних нового облікового запису, користувач отримує повідомлення (рис. 4.7), про надсилання листа на вказану адресу електронної пошти.

Пройшовши за посиланням, вказаним в листі, що відправлений на вказану при реєстрації електронну пошту, він перейде на сторінку активації облікового запису. Після успішної активації, користувач може перейти на головну сторінку та увійти в активований обліковий запис.



The image shows a mobile application interface for registration. At the top, there is a navigation bar with a back arrow, a settings icon, and a language dropdown menu set to 'Українська'. The main content area contains a registration form with the following fields: an email field with the text 'some-test-user@exmaple.ua', a username field with 'some-user', a phone number field with '099 999 99 99', a password field with six dots and a toggle icon, and a confirmation password field with six dots. Below the fields are two buttons: 'Зареєструватися' (Register) and a link 'Увійти' (Login).

Рисунок 4.6 – Сторінка з формою реєстрації

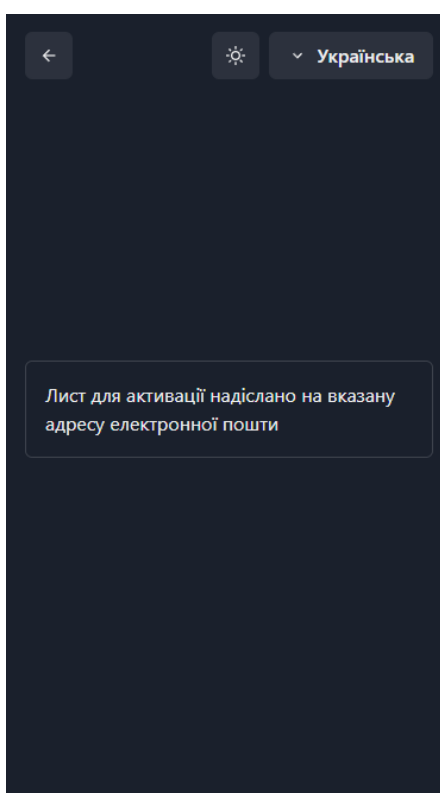


Рисунок 4.7 – Повідомлення про відправлення листа з посиланням на активацію облікового запису

Увійшовши до облікового запису користувач отримує доступ до форми створення нового оголошення (рис. 4.8).

The screenshot shows a web interface for creating a new listing. At the top, there is a navigation bar with links: 'Головна', 'Купити', and 'Продати'. To the right of the navigation bar are icons for settings, language selection (set to 'Українська'), and a user profile ('user'). Below the navigation bar, the form includes a 'Тип' dropdown menu, a 'Заголовок' text field, an 'Адреса' text field, a 'Ціна €' text field, a 'Кімнати' dropdown menu, a 'Житлова площа м²' text field, and a 'Площа кухні м²' text field. There is also a large 'Опис' text area and a 'temporary image select field' with a plus icon. At the bottom left of the form is a 'Зберегти' button.

Рисунок 4.8 – Форма створення нового оголошення

Після успішного створення оголошення, користувач буде переадресований то сторінки з інформацією про об'єкт нерухомості.

При відкритті користувачем сторінки з інформацією про об'єкт нерухомості, автором якого він є сам, він не побачить інформацію про продавця, але на сторінці буде кнопка переходу до редагування (рис. 4.9). Також аутентифікованому користувачу доступний лист оголошень, автором яких він є.

The screenshot shows a real estate listing page. At the top, there is a navigation bar with links: 'Головна', 'Купити', and 'Продати'. To the right of the navigation bar are icons for settings, language selection (set to 'Українська'), and a user profile ('user'). Below the navigation bar, the listing details are displayed: 'sdsd' (property name), 'SDSDSD' (property ID), '121 212 €' (price), 'Кімнати: 2' (rooms), 'Загальна площа: 122м²' (total area), 'Житлова: 111м²' (living area), and 'Кухня: 11м²' (kitchen). Below these details is the number '1212' and the publication date 'Опубліковано: 14.05.2023 21:22'. At the top right of the listing details is a 'Редагувати' button with a pencil icon.

Рисунок 4.9 – Сторінка з інформацією про об'єкт нерухомості автором якої є поточний користувач

4.3 Тестування вебзастосунку

Для тестування правильності реалізації вебзастосунку використаний метод мануального тестування, тобто проходження всіх сценаріїв користувача вручну, без використання автоматичних тестів.

Таким чином була протестована правильність виконання наступних сценаріїв:

- перегляд списку оголошень неаутентифікованим користувачем;
- перегляд детальної інформації обраного об'єкту нерухомості;
- реєстрація користувача;
- активація облікового запису;
- заборона увійти до неактивованого облікового запису;
- вхід до облікового запису користувача;
- створення нового оголошення аутентифікованим користувачем;
- редагування оголошення аутентифікованим користувачем, що є автором оголошення;
- перегляд аутентифікованим користувачем списку оголошень автором яких є користувач;
- додавання оголошення до списку обраних аутентифікованим користувачем;
- видалення оголошення зі списку обраних аутентифікованим користувачем;
- вихід з облікового запису.

Також була перевірена адаптивність дизайну шляхом перевірки відображення елементів керування вебзастосунку при різній роздільній здатності дисплея.

4.4 Висновки за розділом 4

Були описані сценарії користувача та основні елементи вебінтерфейсу вебзастосунку. Проведено тестування створеного вебзастосунку шляхом мануального тестування. Перевірена адаптивність дизайну вебзастосунку.

ВИСНОВКИ

У ході виконання був виконаний аналіз предметної області. Порівнюючи аналоги виділені плюси і мінуси різних додатків, які мають можливість публікації оголошень з продажу нерухомості. Загалом був виділений ряд пунктів, чому спеціалізовані додатки для продажу нерухомості кращі ніж зайти з публікації оголошень загального призначення та соціальні мережі.

Були висунуті критерії до створюваного додатку, на основі яких була спроектована архітектура вебзастосунку. Була розроблена схема даних та взаємодія між ними, на основі чого був обґрунтований вибір бази даних та системи управління базами даних. Було обрані програмні засоби для реалізації API-серверу, для взаємодії з даними.

Була розроблена архітектура серверу аутентифікації. Визначено та обґрунтовано вибір способу аутентифікації користувачів. Описаний функціонал, який треба імплементувати у сервері аутентифікації. Визначений та обґрунтований вибір програмних засобів, що використовувались при реалізації сервера аутентифікації.

Розроблена клієнтська частина вебзастосунку. Розроблена архітектура клієнту. Визначений набір сторінок та доступ до них. Імплементована взаємодія клієнту з сервером аутентифікації та API-сервером. Спроектований адаптивний дизайн.

Реалізовані всі частини визначені спроектованою архітектурою вебзастосунку. Описані деталі реалізації основних частин та рішення прийняті у процесі реалізації.

Створений програмний продукт протестовано за виділеними сценаріями користувача. Перевірена адаптивність на дисплеях з різною роздільною здатністю.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАНЬ

1. Топ 40 кращих сайтів з нерухомості в Україні [Електрон. ресурс]. – Режим доступу: <https://avrida.com/blog/top-40-krashchyyh-saytiv-z-neruhomosti>.
2. OLX Нерухомість [Електрон. ресурс]. – Режим доступу: <https://www.olx.ua/d/uk/nedvizhimost/>.
3. Dom Ria [Електрон. ресурс]. – Режим доступу: <https://dom.ria.com/uk/>.
4. Relational vs. Non-Relational Database: Pros & Cons [Electronic resource]. – Access mode: <https://aloe.co/blog/relational-vs-non-relational-database-pros-cons>.
5. Relational Vs Non Relational Database (Key Differences) [Electronic resource]. – Access mode: <https://databasetown.com/relational-vs-non-relational-database>.
6. Comparing Database Management Systems: MySQL, PostgreSQL, MSSQL Server, MongoDB, Elasticsearch, and others [Electronic resource]. – Access mode: <https://www.altexsoft.com/blog/business/comparing-database-management-systems-mysql-postgresql-mssql-server-mongodb-elasticsearch-and-others>.
7. Top-4 popular DBSM comparison [Electronic resource]. – Access mode: <https://bytescout.com/blog/2014/11/popular-dbms-comparison.html>.
8. Six-Step Relational Database Design™: A step by step approach to relational database design and development Second Edition : підручник. – 2nd ed. – [S. l.] : CreateSpace Independent Publishing Platform, 2013. – 232 p.
9. Introduction to JSON Web Tokens [Electronic resource]. – Access mode: <https://jwt.io/introduction>.
10. What Are Refresh Tokens and How to Use Them Securely [Electronic resource]. – Access mode: <https://auth0.com/blog/refresh-tokens-what-are-they-and-when-to-use-them>.

11. How Refresh Tokens Work: A Complete Guide for Beginners [Electronic resource]. – Access mode: <https://passage.1password.com/post/how-refresh-tokens-work-a-complete-guide-for-beginners>.
12. Hasura GraphQL Engine Documentation [Electronic resource]. – Access mode: <https://hasura.io/docs/latest/index/>.
13. Node.js HTTP Documentation [Electronic resource]. – Access mode: <https://nodejs.org/api/http.html>.
14. Express Docs [Electronic resource]. – Access mode: <https://expressjs.com/>.
15. Webpack Typescript plugin [Electronic resource]. – Access mode: <https://webpack.js.org/guides/typescript/>.
16. Express cors middleware [Electronic resource]. – Access mode: <https://expressjs.com/en/resources/middleware/cors.html>.
17. Color Mode – Chakra UI [Electronic resource]. – Access mode: <https://chakra-ui.com/docs/styled-system/color-mode>.

ДОДАТОК А
Технічне завдання

Вступ

Вебзастосунок з продажу нерухомості використовується користувачами для публікації оголошень з продажу об'єктів нерухомості і перегляду оголошень створених іншими користувачами. Фільтрування допомагає знайти оголошення, що може задовольнити користувача за рядом критеріїв.

A.1 Підстава для розробки

Підставою для розробки є завдання на дипломну кваліфікаційну роботу на тему «Розробка вебзастосунку для продажу нерухомості», затверджене наказом Національного університету «Запорізька політехніка» №87 від 04 квітня 2023р.

A.2 Призначення розробки

Вебдодаток дає змогу користувачам розміщувати оголошення з продажу нерухомості на спеціалізованій для цього платформі. Застосунок надає ряд можливостей для пошуку оголошень, що задовольняють критеріям, які визначає користувач застосунку. Вебзастосунок реалізує систему аутентифікації користувачів, для надання останнім додаткового функціоналу.

A.3 Основні вимоги до програми, що розробляється

A.3.1 Вимоги до функціональних характеристик

Вебзастосунок реалізує наступний набір функціоналу:

- перегляд списку оголошень об'єктів нерухомості для всіх користувачів вебзастосунку;

- фільтрація списку об'єктів нерухомості за набором властивостей одиниці нерухомості;
- перегляд більш детальної інформації вибраного об'єкта нерухомості, включаючи контактну інформацію продавця;
- можливість зміни мову інтерфейсу за бажанням користувача;
- можливість зміни теми інтерфейсу за бажанням користувача;
- система аутентифікації користувачів, з використанням двох токенів;
- надання доступу аутентифікованим користувачам до створення оголошень з продажу об'єктів нерухомості;
- надання доступу аутентифікованим користувачам до можливості додавання оголошення об'єкту нерухомості до списку обраних, та перегляд списку обраних оголошень.

Система аутентифікації реалізує наступний функціонал:

- реєстрація нових користувачів, з необхідністю підтвердження електронної пошти;
- активація облікових записів користувачів шляхом підтвердження електронної пошти;
- вхід в обліковий запис користувача;
- оновлення токенів за допомогою токена оновлення;
- вихід з облікового запису користувача.

А.3.2 Вимоги до надійності

Вебзастосунок повинен бути призначений для того, щоб при підвищенні навантаження мати можливість горизонтального масштабування. Система аутентифікації повинна використовувати токени доступу з малим часом життя.

A.3.3 Умови експлуатації

Вебзастосунок повинен валідувати дані, які вводить користувач, оскільки вони можуть відхилятися від заданих умов. Також клієнтська частина повинна обробляти помилки, що приходять у відповіді від API-сервера.

A.3.4 Вимоги до складу та параметрів технічних засобів

Користувач для взаємодії з клієнтською веббраузер з увімкненим параметром виконання JavaScript-коду.

Вебсервер має підтримувати програмне забезпечення для роботи з контейнерами Docker.

A.3.5 Вимоги до інтерфейсу програми

Вебзастосунок має браузерний графічний інтерфейс, повинен мати адаптивний дизайн для підтримки пристроями з різною роздільною здатністю дисплея.

A.4 Стадії та етапи розробки

Розробка вебзастосунку має наступні стадії:

- дослідження предметної області;
- ознайомлення з аналогами, виділення основних концепцій;
- визначення функціональних вимог;
- розбиття на функціональні частини та їх визначення;
- визначення архітектури;
- визначення та обґрунтування програмних засобів для реалізації;

- реалізація застосунку дотримуючись визначеної архітектури та використовуючи визначені програмні засоби;
- тестування програмного забезпечення.

ДОДАТОК Б
Опис програми

Б.1 Загальні відомості

Вебзастосунок з продажу нерухомості використовується користувачами для публікації оголошень з продажу об'єктів нерухомості і перегляду оголошень створених іншими користувачами. Фільтрування допомагає знайти оголошення, що може задовольнити користувача за рядом критеріїв.

Б.2 Функціональне призначення

Вебзастосунок з продажу нерухомості повинен надавати користувачам можливість перегляду списку оголошень з продажу об'єктів нерухомості, з системою фільтрації для пошуку оголошень за критерії, що встановлює користувач.

Користувач може перейти на сторінку з більш детальною інформацією про об'єкт нерухомості. Також на сторінці користувач може отримати контактну інформацію продавця.

Аутентифіковані користувачі мають можливість виставляти свої оголошення з продажу нерухомості. Аутентифікація реалізована з використанням JWT та токена оновлення та малим строком дії токена доступу, що надає достатній рівень захищеності облікових записів користувача.

Також аутентифікований користувач має можливість додавання оголошення до списку обраних та перегляду цього списку, для швидкого пошуку оголошень, які сподобалися користувачеві.

Б.3 Опис логічної структури

Вебзастосунок з продажу нерухомості складається з трьох логічних модулів: API-серверу, серверу аутентифікації та клієнту, який з ними взаємодіє.

API-сервер реалізує роботу з базою даних. Сервер аутентифікації використовується аутентифікації користувачів на визначення дозволів до взаємодії з даними.

Б.4 Використані технічні засоби

API-сервер використовує Hasura GraphQL Engine для автоматичного створення GraphQL-запитів, які клієнт може використовувати, базуючись на структурі бази даних.

Сервер аутентифікації використовує платформу Node.js та фреймворк Express для створення HTTP-серверу.

API-сервер та сервер аутентифікації запуснені як контейнери у Docker.

Клієнтська частина являє собою браузерний інтерфейс, що потребує включену підтримку JavaScript.

Б.5 Виклик і завантаження

Оскільки застосунок є Docker-контейнером, для запуску необхідний встановлений Docker. Для взаємодії з інтерфейсом додатку, користувачу потрібно в браузері зайти на встановлену в конфігурації додатку адресу.

ДОДАТОК В
Текст програми

```

import { Request, Response } from "express";
import bcrypt from "bcryptjs";
import { config } from "../../config";
import {
  getUserByEmail,
  getUserByPhone,
  insertUser,
} from "../../services/users";
import { sendActivationToken } from "../../services/emailService";
import { getActivationToken } from "../../services/tokens";

export const registerHandler = async (req: Request, res: Response) => {
  try {
    const { email, name, password, phone } = req.body;
    if (
      !email ||
      typeof email !== "string" ||
      !name ||
      typeof name !== "string" ||
      !password ||
      typeof password !== "string" ||
      !phone ||
      typeof phone !== "string"
    ) {
      return res.status(400).json({
        status: "Error",
        message: "INVALID_REGISTER_DATA",
      });
    }
    const transformedEmail = email.toLowerCase();
    const userByEmail = await getUserByEmail(transformedEmail);
    if (userByEmail) {
      return res.status(400).json({
        status: "Error",
        message: "EMAIL_ALREADY_REGISTERED",
      });
    }
    const userByPhone = await getUserByPhone(phone);
    if (userByPhone) {
      return res.status(400).json({

```

```

        status: "Error",
        message: "PHONE_ALREADY_REGISTERED",
    });
}
const hashedPassword = bcrypt.hashSync(
    password,
    config.passwordEncryptionRounds
);
const { id } = await insertUser(
    transformedEmail,
    hashedPassword,
    name,
    phone
);
const activationToken = getActivationToken(id);
await sendActivationToken(activationToken);
return res.status(200).json({
    status: "OK",
});
} catch (err) {
    res.status(500).json({
        status: "Error",
        message: "SOMETHING_WENT_WRONG",
        details: JSON.stringify(err),
    });
}
};

import { Request, Response } from "express";
import { getUserById, activateUser } from "../services/users";
import { sendActivationToken } from "../services/emailService";
import {
    decodeToken,
    getActivationToken,
    verifyActivationToken,
} from "../services/tokens";

export const activationHandler = async (req: Request, res: Response) => {
    try {

```

```

const { activationToken } = req.body;
let payload;
try {
  payload = verifyActivationToken(activationToken);
} catch (err: any) {
  if (err.message === "jwt expired") {
    const { id } = decodeToken(activationToken);
    if (!id) {
      return res.status(400).json({
        status: "Error",
        message: "INVALID_ACTIVATION_TOKEN",
      });
    }
    const newToken = getActivationToken(id);
    await sendActivationToken(newToken);
    return res.status(400).json({
      status: "Error",
      message: "ACTIVATION_TOKEN_EXPIRED",
    });
  }
  return res.status(400).json({
    status: "Error",
    message: "INVALID_ACTIVATION_TOKEN",
  });
}
const { id } = payload;
if (!id || typeof id !== "string") {
  return res.status(400).json({
    status: "Error",
    message: "INVALID_ACTIVATION_TOKEN",
  });
}
const user = await getUserById(id);
if (!user) {
  return res.status(400).json({
    status: "Error",
    message: "INVALID_ACTIVATION_TOKEN",
  });
}
if (user.activated) {
  return res.status(400).json({

```

```

        status: "Error",
        message: "USER_ALREADY_ACTIVATED",
    });
}
await activateUser(id);
return res.status(200).json({
    status: "OK",
});
} catch (err) {
    res.status(500).json({
        status: "Error",
        message: "SOMETHING_WENT_WRONG",
        details: JSON.stringify(err),
    });
}
};

```

```

import { Request, Response } from "express";
import bcrypt from "bcryptjs";
import { config } from "../../config";
import { getUserByEmail, updateLastOnline } from "../../services/users";
import * as RefreshTokens from "../../services/refreshTokens";
import { getAccessToken, getRefreshToken } from "../../services/tokens";

```

```

export const loginHandler = async (req: Request, res: Response) => {
    try {
        const { email, password } = req.body;
        if (
            !email ||
            typeof email !== "string" ||
            !password ||
            typeof password !== "string"
        ) {
            return res.status(400).json({
                status: "Error",
                message: "INVALID_CREDENTIALS",
            });
        }
        const user = await getUserByEmail(email);
    }
};

```

```

    if (!user) {
        return res.status(400).json({
            status: "Error",
            message: "INVALID_CREDENTIALS",
        });
    }
    if (!user.activated) {
        return res.status(400).json({
            status: "Error",
            message: "NOT_ACTIVATED",
        });
    }
    if (!bcrypt.compareSync(password, user.password)) {
        return res.status(400).json({
            status: "Error",
            message: "INVALID_CREDENTIALS",
        });
    }
    await updateLastOnline(user.id);
    const refreshToken = getRefreshToken(user.id);
    await RefreshTokens.add(refreshToken, user.id);
    return res
        .status(200)
        .cookie("refresh", refreshToken, {
            path: "/api/auth",
        })
        .json({
            status: "OK",
            token: getAccessToken(user.id),
        });
} catch (err) {
    res.status(500).json({
        status: "Error",
        message: "SOMETHING_WENT_WRONG",
        details: JSON.stringify(err),
    });
}
};

```

```
import { Request, Response } from "express";
```

```
import * as RefreshTokens from "../../services/refreshTokens";

export const logoutHandler = async (req: Request, res: Response) => {
  try {
    const { refresh } = req.cookies;
    if (!(await RefreshTokens.find(refresh))) {
      return res.status(400).json({
        status: "Error",
        message: "MISSING_TOKEN",
      });
    }
    await RefreshTokens.remove(refresh);
    res.status(200).clearCookie("refresh", { path: "/api/auth" }).json({
      status: "OK",
    });
  } catch (err) {
    res.status(500).json({
      status: "Error",
      message: "SOMETHING_WENT_WRONG",
      details: JSON.stringify(err),
    });
  }
};
```

```
import { Request, Response } from "express";
import { config } from "../../config";
import * as RefreshTokens from "../../services/refreshTokens";
import { getAccessToken, getRefreshToken } from "../../services/tokens";
import { updateLastOnline } from "../../services/users";
```

```
export const refreshHandler = async (req: Request, res: Response) => {
  try {
    const { refresh } = req.cookies;
    if (!refresh) {
      return res.status(400).json({
        status: "Error",
        message: "INVALID_TOKEN",
      });
    }
  }
};
```

```

    }
    const entry = await RefreshTokens.find(refresh);
    if (!entry) {
      return res.status(400).json({
        status: "Error",
        message: "INVALID_TOKEN",
      });
    }
    await RefreshTokens.remove(refresh);
    const { userId } = entry;
    await updateLastOnline(userId);
    const refreshToken = getRefreshToken(userId);
    await RefreshTokens.add(refreshToken, userId);
    return res
      .status(200)
      .cookie("refresh", refreshToken, {
        path: "/api/auth",
      })
      .json({
        status: "OK",
        token: getAccessToken(userId),
      });
  } catch (err) {
    res.status(500).json({
      status: "Error",
      message: "SOMETHING_WENT_WRONG",
      details: JSON.stringify(err),
    });
  }
};

import { GraphQLClient, gql } from "graphql-request";

const      hasuraUrl      =      process.env.HASURA_URL      ??
"http://localhost:8080/v1/graphql";
const hasuraAdminSecret = process.env.HASURA_ADMIN_SECRET ?? "";

const client = new GraphQLClient(hasuraUrl, {

```

```

    headers: { "x-hasura-admin-secret": hasuraAdminSecret },
  });

```

```

interface User {
  id: string;
  email: string;
  password: string;
  name: string;
  phone: string;
  activated: boolean;
  lastOnline: string | null;
}

```

```

const getUserByIdQuery = gql`
  query getUserById($id: uuid!) {
    users_by_pk(id: $id) {
      id
      email
      password
      name
      phone
      activated
      lastOnline
    }
  }
`;

```

```

export const getUserById = async (id: string) => {
  const { users_by_pk } = await client.request<{ users_by_pk: User }>(
    getUserByIdQuery,
    {
      id,
    }
  );
  return users_by_pk;
};

```

```

const activateUserQuery = gql`
  mutation activateUser($id: uuid!) {
    update_users_by_pk(pk_columns: { id: $id }, _set: { activated: true })
  {
    id
    email
    password
    name
    phone
    activated
    lastOnline
  }
}
`;

```

```

export const activateUser = async (id: string) => {
  const { users_by_pk } = await client.request<{ users_by_pk: User }>(
    activateUserQuery,
    {
      id,
    }
  );
  return users_by_pk;
};

```

```

const getUserByEmailQuery = gql`
  query getUserByEmail($email: String) {
    users(where: { email: { _eq: $email } }) {
      id
      email
      password
      name
      phone
      activated
      lastOnline
    }
  }
`;

```

```
export const getUserByEmail = async (email: string) => {
  const { users } = await client.request<{ users: User[] }>(
    getUserByEmailQuery,
    {
      email,
    }
  );
  return users[0];
};
```

```
const getUserByPhoneQuery = gql`
  query getUserByPhone($phone: String) {
    users(where: { phone: { _eq: $phone } }) {
      id
      email
      password
      name
      phone
      activated
      lastOnline
    }
  }
`;
```

```
export const getUserByPhone = async (phone: string) => {
  const { users } = await client.request<{ users: User[] }>(
    getUserByPhoneQuery,
    {
      phone,
    }
  );
  return users[0];
};
```

```
const insertUserQuery = gql`
  mutation insertUser(
    $email: String
```

```

    $name: String
    $password: String
    $phone: String
  ) {
    insert_users_one(
      object: { email: $email, name: $name, password: $password, phone:
$phone }
    ) {
      id
      email
      password
      name
      phone
      activated
      lastOnline
    }
  }
}
`;

```

```

export const insertUser = async (
  email: string,
  password: string,
  name: string,
  phone: string
) => {
  const { insert_users_one } = await client.request<{ insert_users_one:
User }>(
    insertUserQuery,
    {
      email,
      password,
      name,
      phone,
    }
  );
  return insert_users_one;
};

```

```
const updateLastOnlineQuery = gql`
```

```

mutation ($id: uuid!, $time: timestampz) {
  update_users_by_pk(pk_columns: { id: $id }, _set: { lastOnline: $time
})) {
    id
    email
    password
    name
    phone
    activated
    lastOnline
  }
}
`
;

```

```

export const updateLastOnline = async (id: string) => {
  const { update_users_by_pk } = await client.request<{
    update_users_by_pk: User;
  }>(updateLastOnlineQuery, {
    id,
    time: new Date().toISOString(),
  });
  return update_users_by_pk;
};

```

```

interface RefreshTokenEntry {
  token: string;
  userId: string;
}

```

```

const refreshTokens: RefreshTokenEntry[] = [];

```

```

const add = async (token: string, userId: string) => {
  refreshTokens.push({ token, userId });
};

```

```

const find = async (token: string) => {
  return refreshTokens.find((t) => t.token === token);
};

```

```

const remove = async (token: string) => {

```

```

    refreshTokens.filter((t) => t.token !== token);
  };

export { add, find, remove };

import {
  FC,
  PropsWithChildren,
  createContext,
  useContext,
  useState,
  useCallback,
  useEffect,
  useRef,
} from "react";
import { getIdFromJwt } from "../getIdFromJwt";
import { NavigateWithRedirect } from "../redirect";

export type AuthToken = string;

export type ActivationToken = string;

export interface Credentials {
  email: string;
  password: string;
}

export type RegisterData = Credentials & { name: string; phone: string };

interface Auth {
  userId: null | string;
  authenticated: boolean;
  register(registerData: RegisterData): Promise<void>;
  activate(activationToken: ActivationToken): Promise<void>;
  login(credentials: Credentials): Promise<void>;
  refresh(): Promise<void>;
  logout(): Promise<void>;
  loadingInitialRefresh: boolean;
  notAuthenticatedAfterInitialRefresh: boolean;
}

```

```

const context = createContext({} as Auth);

export interface Client {
  register(registerData: RegisterData): Promise<void>;
  activate(activationToken: ActivationToken): Promise<void>;
  login(credentials: Credentials): Promise<AuthToken>;
  refresh(): Promise<AuthToken>;
  logout(): Promise<void>;
}

interface Props {
  client: Client;
}

interface TokenService {
  token: AuthToken | null;
  get(): AuthToken | null;
  set(token: AuthToken | null): void;
  clear(): void;
}

export const tokenService: TokenService = {
  token: null,
  get() {
    return this.token;
  },
  set(token) {
    this.token = token;
  },
  clear() {
    this.token = null;
  },
};

export const AuthProvider: FC<PropsWithChildren<Props>> = ({
  children,
  client,
}) => {
  const [userId, setUserId] = useState<string | null>(null);
  const [loadingInitialRefresh, setInitialRefreshLoading] =
    useState(true);

```

```

const refreshRequestRef = useRef<null |
ReturnType<Auth["refresh"]>>(null);

const authenticated = !!userId;
const notAuthenticatedAfterInitialRefresh =
  !authenticated && !loadingInitialRefresh;

const register = useCallback(
  async (registerData: RegisterData) => {
    try {
      await client.register(registerData);
    } catch (err) {
      throw err;
    }
  },
  [client]
);

const activate = useCallback(
  async (activationToken: ActivationToken) => {
    try {
      await client.activate(activationToken);
    } catch (err) {
      throw err;
    }
  },
  [client]
);

const login = useCallback(
  async (credentials: Credentials) => {
    try {
      const token = await client.login(credentials);
      tokenService.set(token);
      setUserId(getIdFromJwt(token));
    } catch (err) {
      throw err;
    }
  },
  [client]
);

```

```

const refresh = useCallback(async () => {
  try {
    const token = await client.refresh();
    tokenService.set(token);
    setUserId(getIdFromJwt(token));
  } catch (err) {
    throw err;
  }
}, [client]);

const logout = useCallback(async () => {
  try {
    await client.logout();
    tokenService.clear();
    setUserId(null);
  } catch (err) {}
}, [client]);

useEffect(() => {
  const initiallyRefresh = async () => {
    try {
      if (!refreshRequestRef.current) {
        refreshRequestRef.current = refresh();
      }
      await refreshRequestRef.current;
      setInitialRefreshLoading(false);
    } catch {}
  };
  initiallyRefresh();
}, [refresh]);

return (
  <context.Provider
    value={{
      userId,
      authenticated,
      register,
      activate,
      login,
      refresh,
      logout,

```

```

        loadingInitialRefresh,
        notAuthenticatedAfterInitialRefresh,
    }}
    >
    {children}
  </context.Provider>
);
};

export const useAuth = () => useContext(context);

export const PrivateRoute: FC<PropsWithChildren> = ({ children }) => {
  const { loadingInitialRefresh, authenticated } = useAuth();
  if (loadingInitialRefresh) {
    return null;
  }

  if (authenticated) {
    return <>{children}</>;
  }

  return <NavigateWithRedirect to="/login" />;
};

import { FC, PropsWithChildren, useRef, useMemo } from "react";
import {
  AuthProvider as AuthProviderLib,
  AuthToken,
  Client,
} from "../../../../../shared/auth";
import axios, { AxiosError } from "axios";
import { useToast } from "@chakra-ui/react";

interface ResponseSchema {
  status: "OK";
}

interface ErrorSchema {
  status: "Error";
  message: string;
  details?: string;
}
```

```

}

type ResponseData<T = {}> = ResponseSchema & T;

interface TokenResponse {
  token: AuthToken;
}

const messagesToShow = [
  "INVALID_CREDENTIALS",
  "NOT_ACTIVATED",
  "EMAIL_ALREADY_REGISTERED",
  "PHONE_ALREADY_REGISTERED",
  "INVALID_ACTIVATION_TOKEN",
  "ACTIVATION_TOKEN_EXPIRED",
  "USER_ALREADY_ACTIVATED",
  "INVALID_TOKEN",
];

export const AuthProvider: FC<PropsWithChildren> = ({ children }) => {
  const toast = useToast();
  const instanceRef = useRef(
    axios.create({
      baseURL: process.env.REACT_APP_AUTH_URL,
      withCredentials: true,
    })
  );

  const interceptorRef = useRef<null | number>(null);

  const client: Client = useMemo(() => {
    return {
      async register(registerData) {
        await instanceRef.current.post<ResponseData>("/register",
registerData);
      },
      async activate(activationToken) {
        await instanceRef.current.post<ResponseData>("/activate", {
          activationToken,
        });
      },
      async login(credentials) {

```

```

        const { data } = await instanceRef.current.post<
            ResponseData<TokenResponse>
        >("/login", credentials);
        return data.token;
    },
    async refresh() {
        const { data } = await instanceRef.current.post<
            ResponseData<TokenResponse>
        >("/refresh");
        return data.token;
    },
    async logout() {
        await instanceRef.current.post<ResponseData>("/logout");
    },
};
}, []);

if (interceptorRef.current !== null) {

instanceRef.current.interceptors.response.eject(interceptorRef.current);
}

interceptorRef.current = instanceRef.current.interceptors.response.use(
    (res) => res,
    ({ response }: AxiosError<ErrorSchema>) => {
        const error = response!.data;
        if (!messagesToShow.includes(error.message)) {
            toast({
                title: error.message,
                description: error.details,
                status: "error",
            });
        }
        throw error;
    }
);

return <AuthProviderLib client={client}>{children}</AuthProviderLib>;
};

import {

```

```

    createContext,
    FC,
    PropsWithChildren,
    useCallback,
    useContext,
  } from "react";

export type Query = string;

export type Variables = object;

export type Role = "anonymous" | "user";

interface Context {
  request<T>({
    query,
    variables,
    role = "anonymous",
  }): {
    query: Query;
    variables?: Variables;
    role?: Role;
  }): Promise<T>;
}

const context = createContext({} as Context);

export interface Client {
  request<T>({
    query,
    variables,
    role = "anonymous",
  }): {
    query: Query;
    variables?: Variables;
    role?: Role;
  }): Promise<T>;
}

interface Props {
  client: Client;

```

```

}

export const ApiProvider: FC<PropsWithChildren<Props>> = ({
  client,
  children,
}) => {
  const request = useCallback(
    async <T,>({
      query,
      variables,
      role,
    }): {
      query: Query;
      variables?: Variables;
      role?: Role;
    }): Promise<T> => {
      return await client.request<T>({ query, variables, role });
    },
    [client]
  );

  return <context.Provider value={{ request }}>{children}</context.Provider>;
};

export const useApi = () => useContext(context);

import { PropsWithChildren, FC, useRef, useMemo } from "react";
import { QueryClientProvider, QueryClient } from "@tanstack/react-query";
import { ReactQueryDevtools } from "@tanstack/react-query-devtools";
import axios, { AxiosRequestConfig } from "axios";
import { tokenService, useAuth } from "../../shared/auth";
import {
  ApiProvider as ApiProviderLib,
  Client,
  Query,
  Variables,
  Role,
} from "../../shared/api";

const queryClient = new QueryClient({});

```

```

const refreshRetry = new Set<AxiosRequestConfig>();

export const ApiProvider: FC<PropsWithChildren> = ({ children }) => {
  const { refresh } = useAuth();
  const instanceRef = useRef(
    axios.create({
      baseURL: process.env.REACT_APP_API_URL,
    })
  );
  const requestInterceptorRef = useRef<null | number>(null);
  const responseInterceptorRef = useRef<null | number>(null);
  const refreshRequest = useRef<null | Promise<void>>(null);

  const apiClient: Client = useMemo(() => {
    return {
      async request<T>({
        query,
        variables,
        role = "anonymous",
      }: {
        query: Query;
        variables?: Variables;
        role?: Role;
      }) {
        const { data } = await instanceRef.current.post<T>(
          "",
          {
            query,
            variables,
          },
          {
            headers: {
              "x-hasura-role": role,
            },
          }
        );
        return data;
      },
    };
  }, []);

```

```

if (requestInterceptorRef.current !== null) {
  instanceRef.current.interceptors.request.eject(
    requestInterceptorRef.current
  );
}

requestInterceptorRef.current =
instanceRef.current.interceptors.request.use(
  (config) => {
    const token = tokenService.get();
    if (!!token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  }
);

if (responseInterceptorRef.current !== null) {
  instanceRef.current.interceptors.response.eject(
    responseInterceptorRef.current
  );
}

responseInterceptorRef.current =
instanceRef.current.interceptors.response.use(
  async (res) => {
    const code = res.data.errors?.[0].extensions.code;
    if (res.data.data !== undefined) {
      return res.data;
    }
    if (code !== "invalid-jwt" || refreshRetry.has(res.config)) {
      refreshRetry.delete(res.config);
      throw res;
    }
    if (!refreshRequest.current) {
      refreshRequest.current = refresh();
    }
    await refreshRequest.current;
    refreshRequest.current = null;
    refreshRetry.add(res.config);
  }
);

```

```

        return instanceRef.current(res.config);
    },
    (err) => err
);

return (
    <QueryClientProvider client={queryClient}>
        <ApiProviderLib client={apiClient}>
            <ReactQueryDevtools /> {children}
        </ApiProviderLib>
    </QueryClientProvider>
);

};

import { FC, useRef } from "react";
import {
    Button,
    Flex,
    Popover,
    PopoverArrow,
    PopoverBody,
    PopoverContent,
    PopoverHeader,
    PopoverTrigger,
    Text,
    Menu,
    MenuButton,
    MenuList,
    MenuItem,
    Divider,
} from "@chakra-ui/react";
import { useTranslation } from "../../../../../shared/translations";
import { useAuth } from "../../../../../shared/auth";
import { LoginForm } from "../LoginForm";
import { RedirectLink } from "../../../../../shared/redirect";
import { Link } from "react-router-dom";
import { useViewer } from "../../../../../entities/viewer";

interface Props {
    withPopUp?: boolean;
}

```

```

export const AuthButton: FC<Props> = ({ withPopUp }) => {
  const { authenticated, logout } = useAuth();
  const { t } = useTranslation();
  const { data: user } = useViewer();

  const ref = useRef<HTMLFormElement>(null);
  const restoreFocus = () => ref.current?.focus();

  if (authenticated) {
    return (
      <Menu>
        <MenuButton as={Button}>{user?.name}</MenuButton>
        <MenuList>
          <MenuItem as={Link} to="/estate/my">
            {t("myAds")}
          </MenuItem>
          <MenuItem as={Link} to="/estate/favorites">
            {t("favorites")}
          </MenuItem>

          <MenuItem as={Link} to="/settings">
            {t("settings")}
          </MenuItem>
          <Divider />
          <MenuItem onClick={logout}>{t("logout")}</MenuItem>
        </MenuList>
      </Menu>
    );
  }

  if (withPopUp) {
    return (
      <Popover>
        <PopoverTrigger>
          <Button>{t("login")}</Button>
        </PopoverTrigger>
        <PopoverContent ref={ref}>
          <PopoverArrow />
          <PopoverHeader fontWeight="700" textAlign="center">
            {t("login")}
          </PopoverHeader>
        </PopoverContent>
      </Popover>
    );
  }
}

```

```

        </PopoverHeader>
        <PopoverBody as={Flex} direction="column" gap="2">
            <LoginForm onSubmit={restoreFocus} />
            <Text
                textDecor="underline"
                fontSize="14"
                as={RedirectLink}
                to="/register"
                alignSelf="center"
            >
                {t("register")}
            </Text>
        </PopoverBody>
    </PopoverContent>
</Popover>
);
}

return (
    <Button as={RedirectLink} to="/login">
        {t("login")}
    </Button>
);
};

```

ДОДАТОК Д
Слайди презентації

Національний університет "Запорізька політехніка"
Дипломна кваліфікаційна робота

Розробка вебзастосунку для продажу нерухомості
Development of a Web Application for Real Estate Sales

Виконав
студент групи КНТ-220сп
Керівник
доцент, к.т.н.

Пилип ГАНОШЕНКО
Тетяна ЗАЙКО

Рисунок Д.1 – Слайд №1

Об'єкт дослідження – процес пошуку покупцями об'єктів нерухомості для покупки.

Предмет дослідження – програмне забезпечення для публікації оголошень з продажу об'єктів нерухомості.

Мета роботи – створення вебзастосунку для публікації оголошень з продажу об'єктів нерухомості, що надає користувачам можливість пошуку за вказаними параметрами.

Рисунок Д.2 – Слайд №2

Завдання

- оглянути наявні аналоги;
- визначити вимоги до програмного продукту;
- спроектувати архітектуру вебзастосунку;
- визначити критерії до програмних засобів;
- визначити програмні засоби для реалізації, що відповідають поставленим критеріям;
- реалізувати вебзастосунок.

3

Рисунок Д.3 – Слайд №3

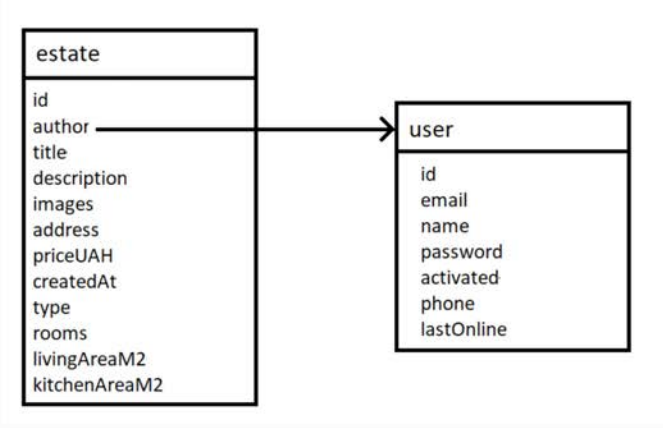
Огляд аналогів

	Dom Ria	OLX
Спеціалізований сайт	Так	Ні
Наявність розширеної фільтрації для пошуку за необхідними критеріями	Так	Ні
Можливість надання більш детальної інформації про об'єкт нерухомості	Так	Ні
Зв'язок з продавцями	Так	Так
Безкоштовність	Ні	Так

4

Рисунок Д.4 – Слайд №4

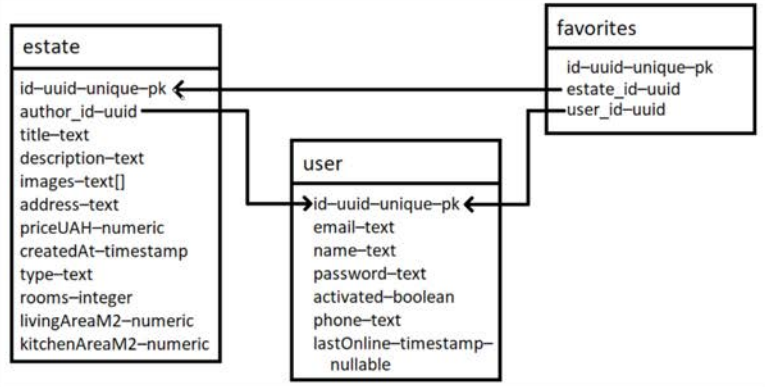
Сутності



5

Рисунок Д.5 – Слайд №5

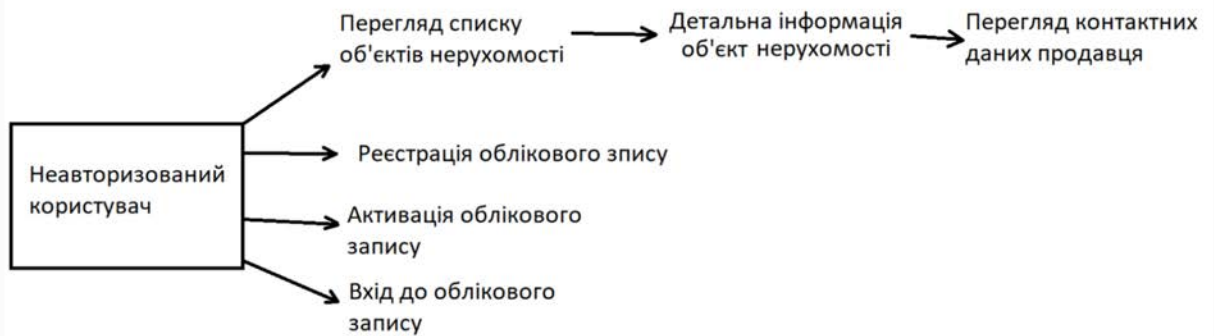
Модель даних



6

Рисунок Д.6 – Слайд №6

Функціонал доступний неавторизованому користувачу



7

Рисунок Д.7 – Слайд №7

Функціонал доступний авторизованому користувачу



8

Рисунок Д.8 – Слайд №8

Авторизація та аутентифікація



Рисунок Д.9 – Слайд №9

Архітектура

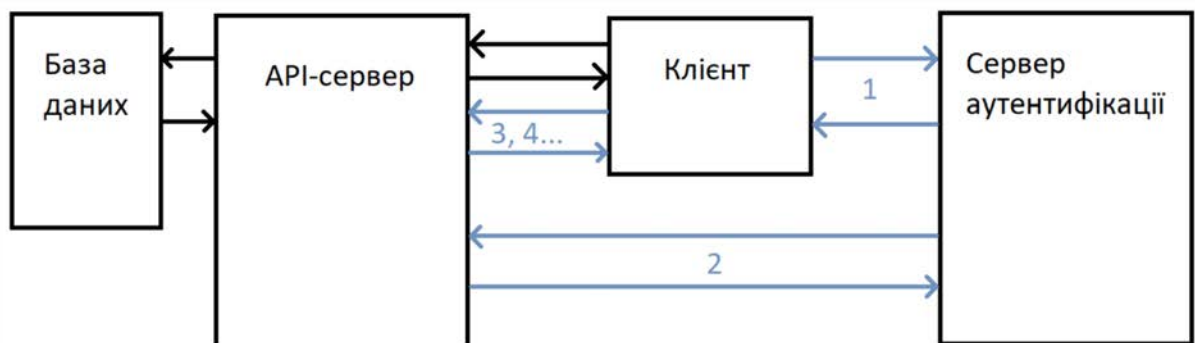


Рисунок Д.10 – Слайд №10

Вибір взаємодії клієнту з сервером

	GraphQL	REST
Кількість ендпоінтів	1	Багато
Гнучкість запитів	Так	Ні
Надмірність даних	Ні	Так
Швидка еволюція API	Так	Ні
Необхідність надсилання додаткових запитів	Ні	Так
Складність реалізації на сервері	Так	Ні

11

Рисунок Д.11 – Слайд №11

Вибір мови програмування

	JavaScript	PHP
Асинхронність та масштабованість	Так	Ні
Використання для клієнту і серверу	Так	Ні
Наявність менеджера пакетів	Так	Так
Продуктивність	Висока	Середня
Масштабованість	Так	Ні
Краще підходить для створення мікросервісів	Так	Ні

12

Рисунок Д.12 – Слайд №12

Інші технології

Фронтенд фреймворк – React з бібліотекою React Query

Система збірки – Webpack

Контейнеризація – Docker

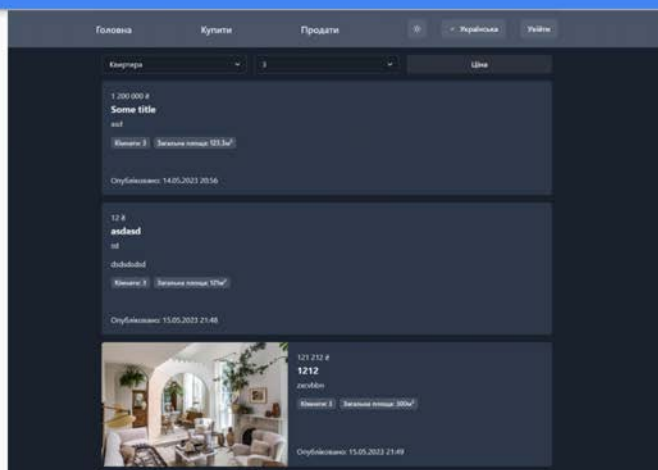
База даних – PostgreSQL

API-сервер – Hasura GraphQL Server

13

Рисунок Д.13 – Слайд №13

Огляд реалізованого застосунку



14

Рисунок Д.14 – Слайд №14

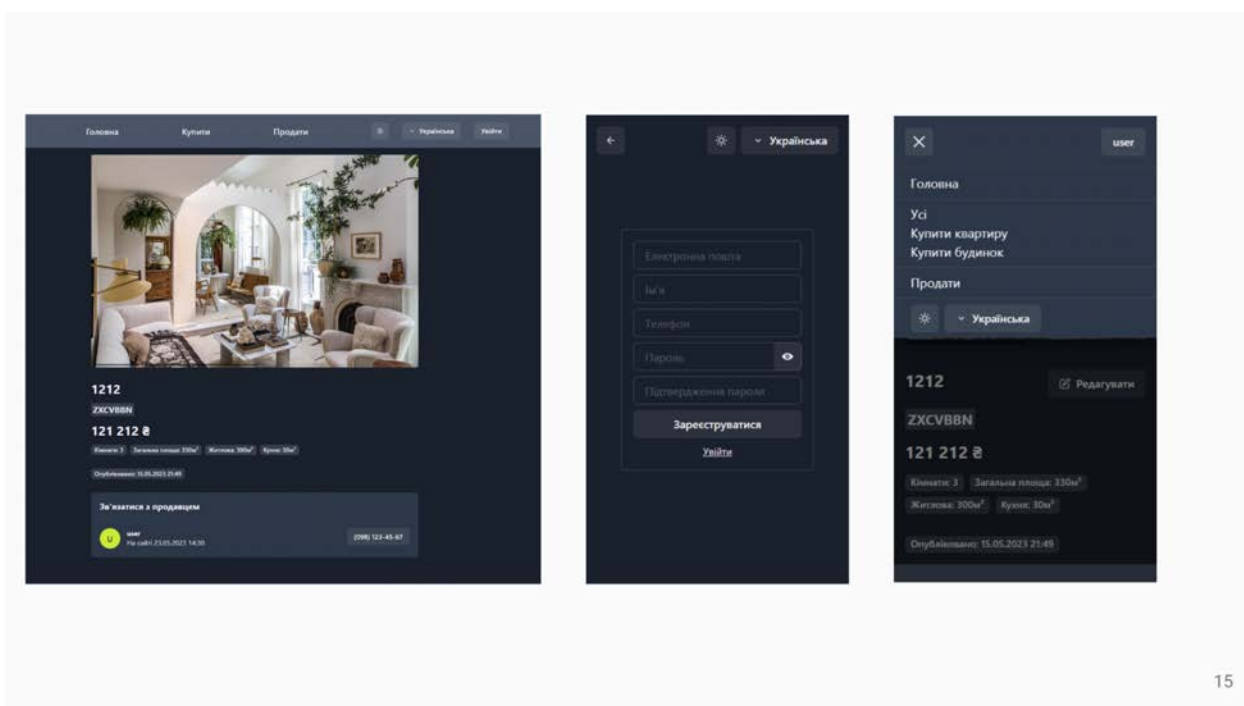


Рисунок Д.15 – Слайд №15

Висновки

У ході виконання дипломної кваліфікаційної роботи було розроблено сайт з продажу нерухомості.

Було сформульоване технічне завдання.

Була спроектована модель даних та архітектура програми.

Був визначений ряд функціоналу та на основі нього висунутий ряд вимог до програмний засобів.

Був реалізований вебзастосунок, що відповідає визначеним вимогам.

Рисунок Д.16 – Слайд №16