Хмельницький національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

Веб-застосунок для автоматизації пошуку об'єктів нерухомості
Назва теми

Рівень вищої освіти Перший(бакалаврський)

Галузь знань 12 «Інформаційні технології»

Спеціальність 121«Інженерія програмного забезпечення»

Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

Шифр КвРІПЗ.200125.01.09.ПЗ

Виконав студент ІІІ курсу група ІПЗс-20-1 _____ М.І. Шевчук
Підпис Ініціали, прізвище

Керівник канд. пед. наук, доцент _____ О. Г. Онишко
Науковий ступінь, звання Підпис Ініціали, прізвище

Нормоконтролер доцент кафедри ІПЗ _____ І.В. Гурман
Підпис Ініціали, прізвище

До захисту допускаю:
Завідувач кафедри інженерії програмного забезпечення _____ Л. П. Бедратюк
Підпис Ініціали, прізвище

6 червня 2023 р.

Хмельницький 2023

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

Факультет Інформаційних технологій
Кафедра Інженерії програмного забезпечення
Рівень вищої освіти Перший (бакалаврський)
Галузь знань 12 «Інформаційні технології»
Спеціальність 121 «Інженерія програмного забезпечення»
Освітня програма Освітньо-професійна програма «Інженерія програмного забезпечення»

ЗАТВЕРДЖУЮ
Завідувач кафедри
Л. П. Бедратюк
05 02 2023 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Шевчуку Миколі Ігоровичу
Прізвище, ім'я, по батькові студента

1. Тема проекту (роботи) Веб-застосунок для автоматизації пошуку об'єктів нерухомості

Керівник проекту (роботи) Онишко Оксана Григорівна, канд. техн. наук, доцент
Прізвище, ім'я, по батькові, науковий ступінь, вчене звання

Затверджена наказом ректора університету від 05.02.2023 р. № 11
2. Строк подання студентом проекту (роботи) на кафедру 05.06.2023 р.
3. Вихідні дані до проекту (роботи) Матеріали переддипломної практики
4. Зміст пояснювальної записки (перелік питань, які потрібно розробити) Дослідження предметної області та постановка задачі, проектування програмного забезпечення, програмна реалізація, тестування програмного забезпечення
5. Перелік графічного матеріалу (із зазначенням обов'язкових креслень) Три креслення: діаграма класів, діаграма послідовності, діаграма взаємодії

6. Консультанти розділів кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Нормоконтроль	Гурман І. В., доцент кафедри ІПЗ	6.06.2023	6.06.2023
Антиплагіат	Гурман І. В., доцент кафедри ІПЗ	5.06.2023	5.06.2023

7. Дата видачі завдання « 05 » лютого 2023р.

КАЛЕНДАРНИЙ ПЛАН

Назва етапів (розділів) Кваліфікаційної роботи	Строк виконання етапів проекту (роботи)	Примітка
1 Ознайомлення з тематикою проектування кваліфікаційної роботи, визначення та узгодження індивідуальних тем	01.12 – 30.12.2022	
2 Дослідження предметної області, в якій планується використання програмного засобу (ПЗ), визначення задач та вимог, розробка технічного завдання	02.01 – 31.01.2023	
3 Проектування програмного забезпечення	01.02 – 28.02.2023	
4 Програмна реалізація	01.03 – 10.04.2023	
5 Тестування програмного забезпечення	11.04 – 30.04.2023	
6 Написання вступу, загальних висновків, оформлення джерел посилання та додатків. Оформлення пояснювальної записки ДП згідно вимог стандартів	01.05 – 25.05.2023	
7 Попередній захист ДП	Травень 2023 (згідно графіка)	
8 Перевірка ДП на плагіат, нормконтроль, отримання відгуків та рецензій. Брошурування (зшиття) пояснювальної записки	26.05 – 30.05.2023	
9 Підготовка до захисту та захист ДП	з 01.06.2023	

Студент _____ М.І. Шевчук
Підпис Ініціали, прізвище

Керівник проекту (роботи) _____ О.Г. Оніщко
Підпис Ініціали, прізвище

АНОТАЦІЯ

Тема кваліфікаційної роботи: Веб-застосунок для автоматизації пошуку об'єктів нерухомості.

Автор роботи: Шевчук Микола Ігорович.

Керівник роботи: Онишко Оксана Григорівна.

Пояснювальна записка: 107 с.,32 рис.,4 табл., 4дод., 40 джерел.

Графічна частина: 23 слайдів.

TYPESCRIPT, REST, POSTGRESQL, NESTJS, NEXTJS, PRISMA.

Метою проєкту є розробка веб-застосунку, який забезпечує забезпечує швидкий та надійний пошук нерухомості.

У процесі кваліфікаційної роботи було проведено аналіз предметної області, виявлено причини частого виникнення проблем при пошуку нерухомості, проаналізовано існуючі альтернативні програмні продукти, а також порівняно архітектурні рішення та патерни для розробки веб-застосунків. Крім того, були визначені модулі системи та здійснена їх програмна реалізація.

Для розробки веб-застосунку був використаний фрейморк NextJS використовуючи мову TypeScript, бекенд NestJS фреймворк, та база даних PostgreSQL.

В результаті було реалізовано веб-застосунок для автоматизації пошуку нерухомості.

16.05.2025
Дата

Підпис

ВІДОМІСТЬ ДОКУМЕНТІВ

№ рядка	Формат	Позначення документа	Найменування документа	К-сть аркушів	№ екз.	Примітка
			Текстові документи			
1	А4	КвРІПЗ.200125.01.09.ПЗ	Пояснювальна записка	72		
2	А4		Завдання на кваліфікаційну роботу	1		
3	А4		Анотація	1		
			Графічні документи			
4	А3	КвРІПЗ.200125.01.09. Е8	Схема бази даних	1		
5	А3	КвРІПЗ.200125.01.09. Е8	Структура сервера	1		
6	А3	КвРІПЗ.200125.01.09.Е8	Діаграма використання	1		

КвРІПЗ.200125.01.09.ВД

Змн.	Арк.	№ докум.	Підпис	Дата
Виконав		Шевчук М.І.		6.06
Керівник		Онишко О.Г.		6.06
Н. Контр..		Гурман І.В.		6.06
Зав. Каф.		Бедратюк Л.П.		6.06

Веб-застосунок для автоматизації пошуку об'єктів нерухомості
Відомість документів

Літ.	Арк.	Аркушів
	1	1

ХНУ, ІПЗс-20-1

ЗМІСТ

ВСТУП

На сьогоднішній день найти хорошу нерухомість це не просто. Пошук займає багато часу, а агенства з нерухомості неможуть запропонувати саме те, чого ви бажаєте, або і зовсім намагаються вас підштовхнути до пропозиції яка вам недовподоби.

Люди, які шукають житло, можуть бути у відчаї та стають легкою мішенню для шахраїв. Одна з найпоширеніших афер на сайтах нерухомості – вимагання застави без огляду квартири. Часто шахрай терміново пояснює, що квартиру вже переглядали інші, і єдиний спосіб убезпечити її зараз — це передати заставу. Оскільки отримати хорошу квартиру часто дуже важко, особливо у великих містах, деякі люди можуть потрапити на це й надіслати гроші. Шахрай забирає заставу, а потенційний орендар не чує від нього жодного слова. Швидше за все, квартири ніколи не існувало або вона не належала шахраю.

Деякі рекламодавці навмисно вказуватимуть будинки, які вже продані справжніми ріелторами, щоб отримати комісію за рекомендацію. В інших випадках ви побачите, як люди публікують навіть нереальні об’єкти, щоб зібрати контактну інформацію потенційних орендарів/покупців, аби потім продати агентствам нерухомості як потенційних клієнтів.

Більш сіра версія цієї практики полягає в тому, що агентства публікують першокласну нерухомість за чудовою ціною, яка завжди продається перед тим, як потенційний орендар/покупець зв’яжеться з нею. Таким чином агентство збирає інформацію про зацікавлених осіб, і вони можуть представити подібні об’єкти за трохи вищою ціною.

Деякі проблеми, з якими стикається категорія нерухомості, є більш серйозними, ніж інші, але вони мають одну спільну рису. Усі вони негативно впливають на досвід користувача.

Тому було прийнято створити сервіс який дозвилить публікувати оголошення про нерухомість, а також автоматично підбирати для клієнта об’єкти з вказаними ним параметрами.

Формулювання задачі наукового дослідження. Задачею наукового дослідження є створення сервісу для автоматичного знаходження нерухомості.

Мета даної роботи полягає в розробці сервісу який автоматично знаходить нерухомість для клієнтів.

Об'єктом даного дослідження є інтелектуальні інформаційні системи та засоби пошуку об’єктів нерухомості.

Предметом дослідження є моделі вдосконалення пошуку об’єктів нерухомості.

Результатом є працюючий веб-застосунок для знаходження нерухомості.

Для досягнення поставленої мети були сформовані завдання на кваліфікаційну роботу:

– проаналізувати специфіку та особливості предметної області ринку нерухомості;

– визначити та довести актуальність даної теми;

– провести аналіз наявного програмного забезпечення, що відноситься до даної предметної області, з метою виявлення його переваг та недоліків;

– скласти список функціональних та нефункціональних вимог до програмного забезпечення;

– провести аналіз інструментів та методів розробки ПЗ;

– розробити оригінальний, цікавий та зручний інтерфейс користувача;

– провести різні види тестування програмного забезпечення.

1 ДОСЛІДЖЕННЯ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Змістовний аналіз предметної області, її структурних та функціональних особливостей

Під час аналізу предметної області було виявлено, що на даний момент буде актуальним створення веб-застосунку, який дозволить користувачам шукати об’єкти нерухомості за певними критеріями. Тому було вирішено створити такий веб-застосунок, а також реалізувати при можливості інтерграцію із відомими сервісами аби аналізувати їхні пропозиції і якщо вони відповідають параметрам клієнта то відіслати їх клієнту. Отже потрібно зрозуміти що таке нерухомість, дізнатися про її види і чому вона так важлива.

Під поняттям нерухомості розглядається майно юридичної або фізичної особи, яка не піддається переміщенню в просторі без нанесення певного збитку. Всі інші види майна визначаються як рухоме майно.

До нерухомості відносяться житлові та нежитлові приміщення, споруди, земельні ділянки та інше. Об'єкт нерухомості, міцно прикріплений фундаментом до відведеного під нього земельної ділянки, називають капітальним об'єктом нерухомості. В іншому випадку нерухомість вважається некапітальної.

Об'єкти нерухомості можуть використовуватися як з метою проживання і реєстрації за адресою, так і з метою отримання пасивного доходу шляхом ведення бізнесу. Перший вид нерухомості називається житловою нерухомістю, у другому випадку нерухомість нежитлова, також є інші види нерухомості.

Земля є основою для всіх видів нерухомості. Земля зазвичай відноситься до незабудованої власності та вільних земель. Забудовники купують землю та об’єднують її з іншими об’єктами (так звані збірки) і переплановують її, щоб збільшити щільність і вартість власності.

Житлова нерухомість складається з житла для окремих осіб, сімей або груп людей. Це найпоширеніший тип нерухомості та клас активів, з яким знайомі більшість людей. Серед житлових будинків є односімейні будинки, квартири, таунхауси та інші типи житлових приміщень.

Комерційна власність стосується землі та будівель, які використовуються підприємствами для здійснення своєї діяльності. Приклади включають торгові центри, окремі магазини, офісні будівлі, автостоянки, медичні центри та готелі.

Промислова нерухомість стосується землі та будівель, які використовуються промисловими підприємствами для таких видів діяльності, як заводи, механічне виробництво, дослідження та розробки, будівництво, транспортування, логістика та складування.

Це була коротко та необхідна інформація для розуміння з чим саме буде працювати готовий веб-застосунок. Очікується що веб-застосунок матиме весь необхідний функціонал для швидкого пошуку нерухомості та задовільнятиме всі потреби користувача.

1.2 Аналіз наявного програмно-технічного забезпечення предметної області

Наступним кроком дослідження предметної області являється аналіз наявного програмно-технічного забезпечення, тому розпочнемо з розгляду існуючих рішень на основі найбільш розповсюджених технологій.

Існує багато відомих сервісів в яких розміщено багато оголошень. Ці сервіси перевірені часом та людьми і мають великий попит. Найбільш відомі: ЛУН, Dom Ria, Zillow.com.

ЛУН позиціонує себе як найбільший сайт з пошуку квартир в Україні. По факту даний сервіс являється агрегатором, який збирає інформацію з багатьох українських сервісів та подає у потрібному клієнтові форматі. Головною

перевагою ЛУН є генерація даних способом збору інформації з інших сервісів, так як у такому випадку відпадає необхідність у створені форм для зберіганні інформації введеної користувачами. Також до переваг ЛУН можна добавити велику кількість фільтрів для пошуку необхідного об’єкту нерухомості – серед таких фільтрів є і карта. Також даний сервіс має стрічку новин, що допомагає користувачам бачити завжди актуальні події. Слід відзначити, що ЛУН підсумовує статистику по цінам на нерухомість кожного дня, що ще раз підкреслює про гарну підтримку та користь використовуваних алгоритмів. Серед недоліків я б відзначив вузька направленість сервісу та відсутня універсальність у даних. ЛУН спеціалізується лише на житловому виді нерухомості, а значить не має можливості слідкувати за комерційною, промисловою та земельною. Цілком можливо, що при бажані дана компанія могла б розширити сферу впливу, але через те, що універсальність у моделі даних відсутня потрібно буде перебудувати усю систему.

Основні переваги ЛУН:

– генерація даних за допомогою зовнішніх сервісів;

– кількість фільтрів для даних;

– актуальність та підтримка;

– алгоритми статистики;

Основні недоліки ЛУН:

– відсутня можливість створення власних об’яв;

– погана універсальність даних;

– вузька направленіст;

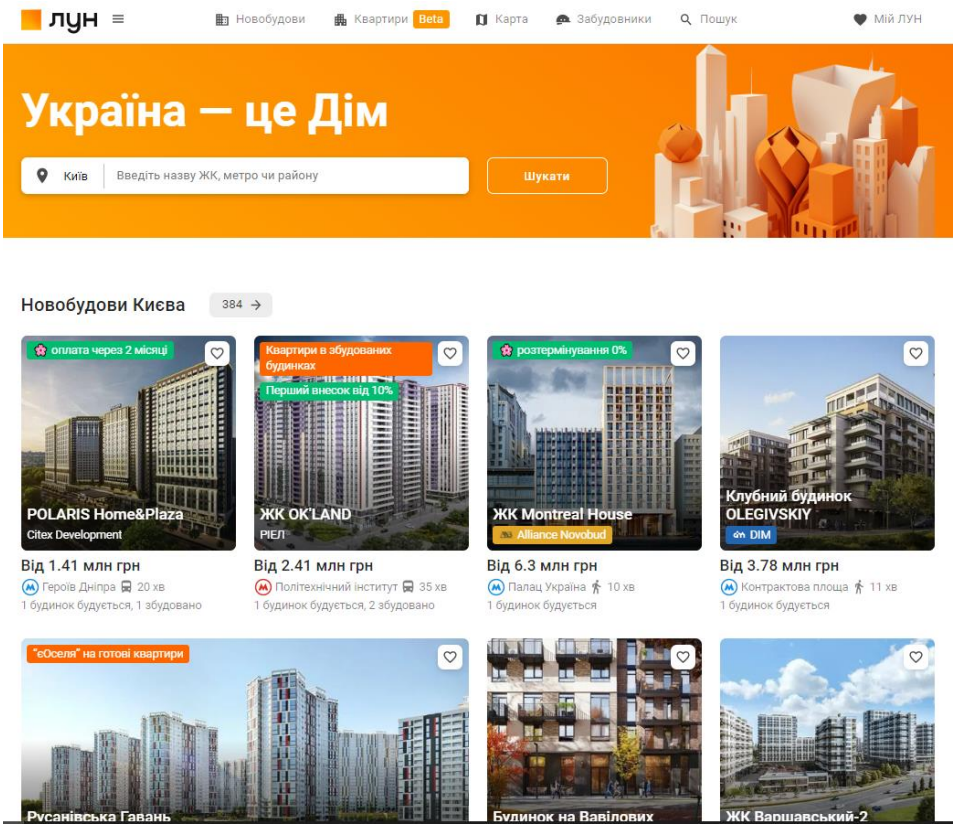


Рисунок 1.1 – інтерфейс сайту “Лун” - https://lun.ua/

Dom Ria – яскравий приклад сервісів по збору інформації шляхом введення її користувачами. Даний сервіс має вузьку направленість, а отже збирає дані лише про житлову нерухомість. Серед переваг слід відзначити можливість викликати інспектора по нерухомості, який може допомогти вам оцінити ваше житло і виставити ціну, яка відповідає ринку. Це означає, що даний сервіс відійшов від простого збереження та розміщення даних до бізнес моделі, яка надає послуги зв’язані з сферою нерухомості. Серед недоліків слід відзначити маленьку кількість фільтрів для пошуку, що робить функцію знаходження потрібної власності більш важкою. Також я б відзначив погану модель даних – кількість рис, якими описують нерухомість являється надзвичайно малою і по факту нормальний опис дають лише фотографії, розміщені користувачами.

Основні переваги Dom Ria:

– генерація даних користувачами;

– бізнес модель компанії.

Основні недоліки Dom Ria:

– погана модель даних;

– маленька кількість фільтрів для пошуку;

– вузька направленість

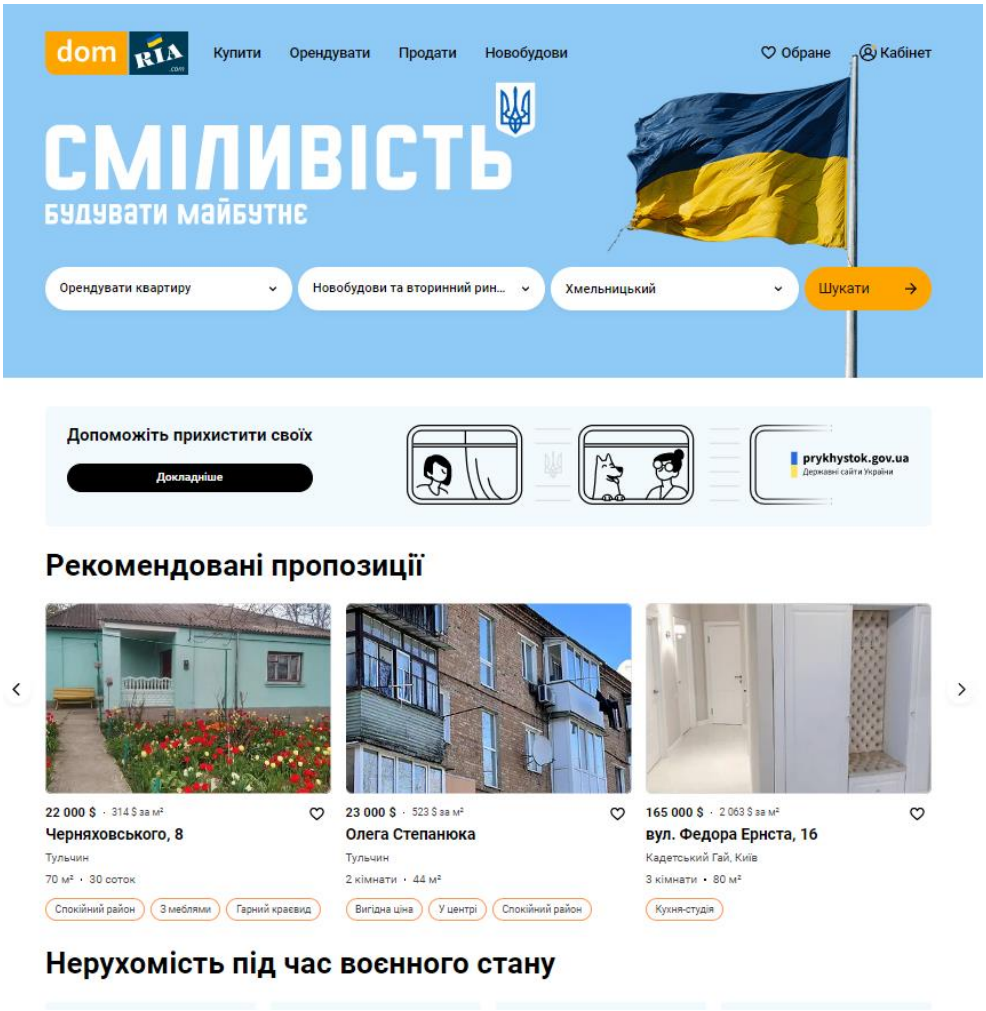


Рисунок 1.2 – інтерфейс сайту “Dom Ria” - https://dom.ria.com/uk/

Zillow - це провідний іноземний сервіс пошуку нерухомості, присвячений наданню споживачам даних та знань про місце де знаходиться будинок, та з'єднанні їх з кращими місцевими професіоналами, які можуть допомогти наприклад у його оздоблені, садівництві чи ремонті. Zillow служить повним життєвим циклом володіння та проживання в будинку: купівля, продаж, оренда, фінансування, реконструкція та інше. Він починається з живої бази даних

Zillow, що містить понад 110 мільйонів американських будинків - включаючи будинки на продаж, будинки для оренди та будинки, які зараз не продаються, а також оціночні цінності для дому, оцінки вартості оренди та іншу інформацію, пов'язану з домом. Це яскравий приклад програмного забезпечення світового рівня. Серед недоліків слід відзначити все ту ж вузьку направленість – даний сервіс спеціалізується лише на житловій нерухомості.

Основні переваги Zillow:

– використання стандарту RESO;

– велика кількість фільтрів для пошуку;

– можливість пошуку робітників;

– можливість створення об'яв власноруч.

Основні недоліки Zillow:

– вузька направленість з даними про нерухомість.

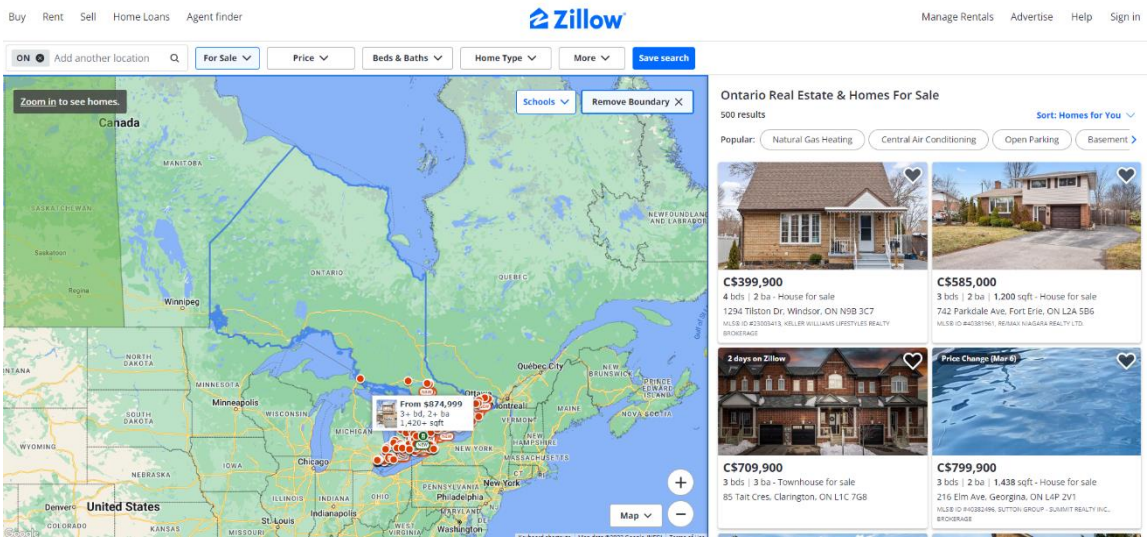


Рисунок 1.3 – інтерфейс сайту "Zillow" - https://www.zillow.com/

Отже, всі представленні веб-застосунки мають велику кількість клієнтів і є найкращими у своєму напрямку, але жоден не є ідеальним рішенням. Також аналізування існуючих аналогів та їх особливостей дозволяє визначити основні характеристики, функціональні та нефункціональні вимоги розроблюваного застосунку, а також оцінити його відповідність та актуальність.

1.3 Визначення вимог до програмного продукту

Після аналізу предметної області та аналогів на ринку потрібно визначитись з вимогами до веб-додатку, та описати їх. Найкращий спосіб для цього – використання мови UML. UML (англ. Unified Modeling Language) – уніфікована мова об’єктно-орієнтованого моделювання, може затсосовуватись на усіх етапах життєвого циклу розробки додатків. Для цього використовуються різні діаграми які надають можливість представити систему у такому вигляді, щоб її можна було легко перевести в програмний код. Щоб визначити вимоги потрібно створити діаграмму варіантів використання.

Діаграмма варіантів використання – суть діаграмми полягає в наступному: система яка проектується подається у вигляді множини сущностей або акторів, які взаємодію з системою за допомоги так названих варіантів використання. При цьому актором називається будь яка сущність, що взаємодіє з системою зовні, а варіанти використання описують сервісів, яка система надає аткорові.

Для нашої системи ми складаємо описи акторів (користувачів системи) та перераховуємо всі можливі дії (варіанти використання), які вони можуть виконувати.

У таблиці 1.1 наведені актори розроблюваного програмного застосунку.

Таблиця 1.1 – Опис акторів розроблюваного програмного застосунку

Актор	Короткий опис
Користувач	Може переглядати об’єкти нерухомості, використовувати фільтри для більш детального та швидкого пошуку. зареєструватися аби отримати більше можливостей, розміщувати власні оголошен, керувати ними в особистому кабінеті.

Кінець таблиці 1.1

Адміністратор	Проводить модерацію оголошень, може переглядати зареєстрованих користувачів. Примусово видалити невідповідні оголошення, або і зовсім певного користувача.

В таблиці 1.2 наведені способи використання розроблюваного веб-застосунку.

Таблиця 1.2 – Опис способів використання розроблюваного веб-застосунку.

Актор	Короткий опис
Користувач	Може переглядати існуючі оголошення користуючись фільтрами та пошуком, може провести реєстрацію для цього вводить номер телефону або електрону пошту, та вводить придуманий ним пароль. Після реєстрації має можливість створити оголошення. А також керувати оголошенням в особистому кабінеті.
Адміністратор	Може створювати нові категорії нерухомості, добавляти валюти в яких вказана ціна. Видалити з системи підозрілих користувачів. Виконувати модерацію оголошень, добавляти нові типи будинків для квартир, сторювати унікальні особливості для нерухомості, що додає змогу ліпше описати нерухомість.

Визначивши акторів системи та варіанти використання, було побудовано діаграму варіантів використання (рисунок 1.4).

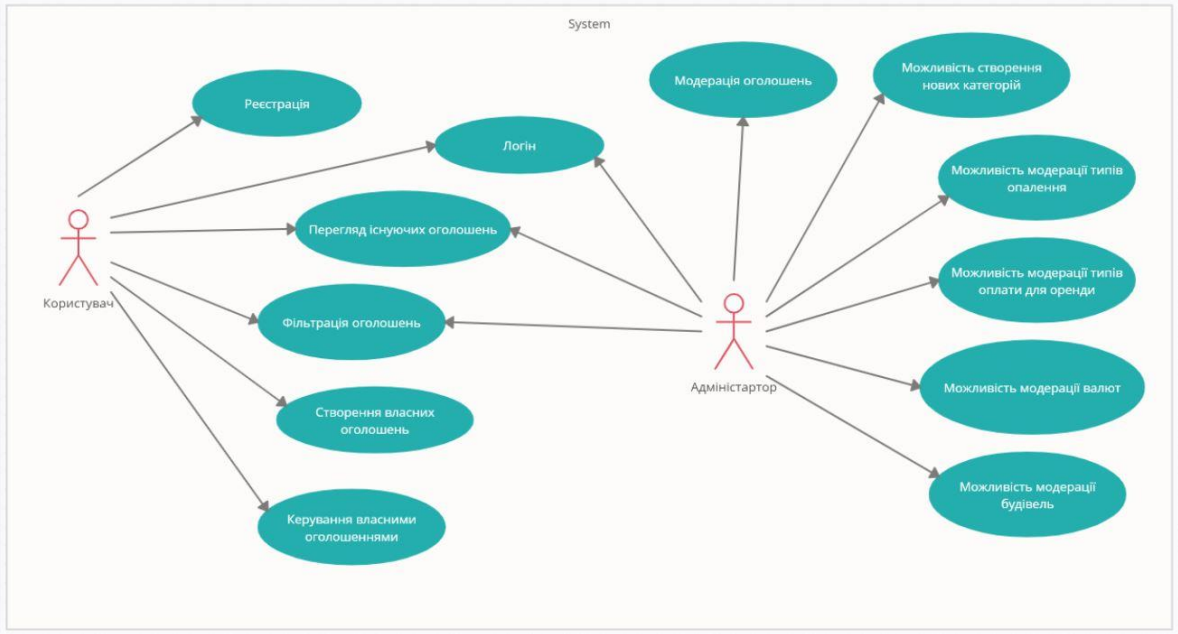


Рисунок 1.4 – Діаграма варіантів використання

1.4 Постановка задачі

Після проведення аналізу вимог до програмного продукту було підготовлено технічне завдання, яке можна знайти у додатку А. Це технічне завдання буде використовуватись як основа для подальшої розробки веб-застосунку, що має на меті автоматизувати пошук нерухомості.

При розробці кваліфікаційної роботи " Веб-застосунок для автоматизації пошуку об’єктів нерухомості:

– Необхідно створити інтуїтивно зрозумілий та привабливий користувацький інтерфейс, який забезпечить зручне використання веб-застосунку..

– Розробити систему веб-застосунку для швидкого та зручного пошуку нерухомості.

– Забезпечити можливість для адміністраторів додавання нових категорій нерухомості, створення нових унікальних особливостей нерухомості, додавання

типів будинків для квартир, також можливість додавання нових валют для опису ціни.

− Розробити зручну систему для можливості створення персональних оголошенн з продажу та оренди нерухомості.

Висновки до розділу

Був проведений аналіз та дослідження предметної області, виявлені проблеми, а також визначені цілі і завдання даної роботи. Була обґрунтована необхідність створення програмного продукту для автоматизації пошуку нерухомості. Також були сформульовані основні завдання, які ми ставимо перед собою у процесі розробки. Виконання цих завдань має на меті поліпшення якості та ефективності роботи веб-застосунку, забезпечення зручної та ефективної взаємодії користувачів та покращення надання послуг у даній сфері.

2 ПРОЕКТУВАННЯ ВЕБ-ЗАСТОСУНКА

2.1 Аналіз та вибір архітектури веб-застосунка

Більшість веб-сайтів будуються за допомогою комбінації клієнтського та серверного програмування, бази даних та фронтенд-інструментів. Клієнт-серверна архітектура є однією з найпоширеніших архітектур для розробки веб-додатків. У цій архітектурі функціональність додатку розподілена між двома основними компонентами: клієнтом і сервером.

Клієнт-серверна архітектура має кілька переваг, які сприяють розробці та ефективній роботі веб-додатків.

Розділення обов'язків: Клієнт і сервер виконують різні функції та мають розділені обов'язки. Клієнтська частина відповідає за відображення інтерфейсу користувача та взаємодію з ним, тоді як серверна частина обробляє бізнес-логіку, зберігає та обробляє дані. Це сприяє чистоті коду, полегшує розподіл роботи між розробниками та дозволяє ефективно масштабувати систему.

Масштабованість: Клієнт-серверна архітектура дозволяє масштабувати окремі компоненти системи незалежно один від одного. Можна масштабувати сервери, щоб впоратися з великим навантаженням або забезпечити високу доступність, а клієнти можуть бути запущені на різних пристроях з різними характеристиками.

Покращена продуктивність: Клієнтські додатки можуть зберігати локальні копії даних, що дозволяє зменшити час затримки при взаємодії з сервером. Крім того, серверна частина може використовувати різні оптимізації, такі як кешування та оптимізація запитів до бази даних, що покращує продуктивність додатка.

Зручність розробки: Клієнт-серверна архітектура дозволяє розділити розробку між фронтенд-і бекенд-розробниками. Це дозволяє спеціалізуватися на конкретних технологіях та областях, полегшує розробку та підтримку коду.

Більша безпека: За допомогою клієнт-серверної архітектури можна забезпечити більшу безпеку.

Хоча клієнт-серверна архітектура має багато переваг, вона також має деякі недоліки, про які варто знати.

Залежність від сервера: Клієнтські додатки вимагають постійного підключення до сервера для отримання даних та виконання операцій. Це означає, що якщо сервер недоступний або мережа має проблеми, то користувачам може бути обмежено або неможливо взаємодіяти з додатком.

Надмірна навантаженість сервера: Якщо сервер отримує велику кількість запитів від клієнтів, це може призвести до надмірної навантаженості та зменшення продуктивності. Потрібні додаткові ресурси для масштабування сервера та забезпечення його ефективної роботи.

Комплексність розробки: Клієнт-серверна архітектура вимагає розробки та підтримки двох різних компонентів - клієнта і сервера. Це може збільшити складність розробки, особливо якщо існують вимоги до синхронізації даних між клієнтом та сервером.

Проблеми з безпекою: Клієнтські додатки можуть бути менш безпечними, оскільки клієнт може бути підвержений злому або неправильному використанню. Належні заходи безпеки повинні бути реалізовані як на клієнтській, так і на серверній сторонах.

Синхронізація даних: У деяких випадках, коли клієнти працюють з одними й тими ж даними на сервері, можуть виникати проблеми з конфліктами та несинхронізованістю даних. Розробникам потрібно ретельно вирішувати ці проблеми та забезпечувати правильну синхронізацію даних.

Загалом, хоча клієнт-серверна архітектура має багато переваг, вона також має свої недоліки, які потрібно ураховувати при розробці веб-додатків.

Основні складові клієнт-серверної архітектури включають:

Клієнт: Це програмне забезпечення або пристрій, який виконується на боці користувача і надає інтерфейс для взаємодії зі сервером. Клієнт може бути

веб-браузером, мобільним додатком, настільним додатком або іншою формою програмного забезпечення.

Сервер: Це програмне забезпечення або апаратна платформа, яка забезпечує обробку запитів від клієнтів і надає доступ до ресурсів та послуг. Сервер може бути веб-сервером, базою даних, файловим сервером або іншою системою, яка забезпечує потрібні функції для веб-додатків.

Комунікаційний протокол: Це набір правил і форматів, які визначають спосіб взаємодії між клієнтом і сервером. Найпоширенішими протоколами є HTTP (Hypertext Transfer Protocol) і WebSocket, але також можуть використовуватись інші протоколи, такі як FTP (File Transfer Protocol), SMTP (Simple Mail Transfer Protocol) і т.д.

API (Application Programming Interface): Це набір інтерфейсів і методів, які дозволяють клієнту взаємодіяти з сервером і отримувати доступ до його функціональності. API визначає, які запити можуть бути зроблені клієнтом, які дані можуть бути передані і які відповіді отримані від сервера.

База даних: Це система для зберігання і організації даних, які використовуються сервером. База даних може бути реляційною, нереляційною або гібридною, залежно від потреб проєкту.

Серверні програми: Це програми або модулі, які виконуються на сервері і забезпечують обробку запитів, розрахунки, доступ до бази даних і інші функції, необхідні для роботи додатків.

Ці складові разом створюють клієнт-серверну архітектуру, де клієнт і сервер взаємодіють між собою через комунікаційний протокол і API. Клієнт надсилає запити до сервера, а сервер обробляє ці запити, забезпечує доступ до ресурсів і повертає відповіді клієнту.

Я вибрав клієнт-серверну архітектуру для свого проєкту з наступних причин. Клієнтська і серверна частини мають чітко визначені обов'язки. Це дозволяє краще організувати код і забезпечити легшу розробку і супровід системи, дозволяє масштабувати серверну інфраструктуру незалежно від

клієнтських додатків. Це дозволяє забезпечити потрібну продуктивність навіть при зростанні обсягів даних і навантаження, надає можливість додавати нові функціональні можливості до серверної частини без впливу на клієнтські додатки. Це спрощує розширення функціоналу і впровадження нових функцій в систему В цілому, клієнт-серверна архітектура відповідає потребам мого проєкту, надаючи розділення обов'язків, масштабованість, безпеку і можливість розширення.

2.2 Опис структури даних та моделі бази даних

Більшість сучасних веб-застосунків можуть зберігати свої дані в різних місцях, залежно від характеру і потреб проєкту. Основні місця для зберігання даних веб-застосунків це включають бази даних. Це один з найпоширеніших способів зберігання даних. Бази даних, такі як MySQL, PostgreSQL, MongoDB, дозволяють організовувати, зберігати і отримувати доступ до структурованих даних. Вони забезпечують ефективну організацію і пошук даних за допомогою мов запитів, таких як SQL.

Використовують бази даних з багатьох важливих причин і деякі з них ми перерахуємо, щоб зрозуміти важливість БД.

Зберігання структурованих даних: Бази даних дозволяють організувати та зберігати структуровані дані, такі як інформація про користувачів, продукти, замовлення, повідомлення тощо. Бази даних забезпечують стандартизовану структуру для зберігання даних, що дозволяє легко організовувати та керувати ними.

Ефективний доступ до даних, дозволяють ефективно виконувати запити для отримання, оновлення та видалення даних. Запити можуть бути складними, включати умови, сортування, об'єднання таблиць тощо. Бази даних

використовують оптимізовані алгоритми для швидкого доступу до даних та оптимального виконання запитів.

Забезпечення цілісності даних, забезпечують механізми для збереження цілісності даних. Вони дозволяють встановлювати правила, обмеження та валідацію даних, що допомагає уникати некоректних або неповних записів. Наприклад, база даних може вимагати унікальності значень у певних полях або перевіряти правильність формату даних.

Масштабованість, дозволяють масштабувати веб-сайт і обробку даних. Вони можуть працювати з великим обсягом даних і обробляти багатокористувацькі запити одночасно. Деякі бази даних підтримують реплікацію і шардування, що дозволяє розподілити навантаження і забезпечити високу доступність даних.

Забезпечення безпеки даних, надають механізми для забезпечення безпеки даних, такі як автентифікація, авторизація, шифрування тощо. Вони дозволяють обмежувати доступ до даних лише авторизованим користувачам і захищати конфіденційні дані. Загалом, використання баз даних допомагає веб-сайтам ефективно керувати та обробляти дані, забезпечувати безпеку та цілісність даних, а також масштабуватися для відповіді на зростаючі потреби користувачів.

Існує багато різних типів баз даних, які можна використовувати в залежності від потреб і вимог проєкту, найчасто використовуванні типи це реляційні бази даних (RDBMS) та нереляційні (NoSQL) бази даних.

Реляційні бази даних (RDBMS) є одним з найпоширеніших типів баз даних. Вони засновані на реляційній моделі даних і використовують таблиці для зберігання даних зі структурованими зв'язками між ними.

Основні переваги реляційних баз даних:

Структурованість: Дані зберігаються у вигляді таблиць з рядками і стовпцями, що дозволяє встановлювати чіткі залежності між даними і забезпечує консистентність.

Гнучкість: Реляційні бази даних підтримують мову SQL, яка надає потужні можливості для роботи з даними, включаючи запити, вставку, оновлення і видалення даних.

Зв'язки між даними: дозволяють встановлювати зв'язки між таблицями за допомогою ключів, що дозволяє ефективно зв'язувати дані з різних таблиць і отримувати комплексні результати запитів.

Цілісність даних: підтримуються механізми для забезпечення цілісності даних, таких як обмеження цілісності, унікальність значень, перевірки тощо. Це дозволяє підтримувати консистентність даних і запобігати введенню некоректних даних.

Масштабованість: можуть бути масштабовані для обробки великих обсягів даних і високої навантаженості. Вони підтримують індексування, оптимізацію запитів та розподілені системи для підвищення продуктивності.

Незважаючи на свої переваги, реляційні бази даних також мають певні обмеження, такі як складність масштабування горизонтально і певні обмеження в роботі з невструктурованими даними. Проте, вони залишаються потужним і надійним інструментом для багатьох застосувань, зокрема для систем управління даними підприємств та веб-додатків.

Окрім реляційних баз даних, існують інші типи баз даних, які використовуються для різних цілей. NoSQL бази даних (Not Only SQL) - це група баз даних, які не використовують традиційну реляційну модель з таблицями, рядками та стовпцями. Вони розроблені для роботи з великими обсягами невструктурованих даних, де важлива гнучкість, швидкодія та масштабованість. MongoDB є однією з популярних документ-орієнтованих NoSQL баз даних. Вона має свої переваги та недоліки.

Переваги MongoDB:

Гнучкість схеми: MongoDB дозволяє зберігати документи з різними структурами, що дозволяє легко змінювати схему даних без перебудови всієї бази даних.

Швидкодія: MongoDB використовує індекси для швидкого доступу до даних. Вона також підтримує горизонтальне масштабування, що дозволяє розподіляти навантаження на декілька серверів.

Підтримка геоданих, має вбудовану підтримку для роботи з геоданими, що дозволяє ефективно обробляти просторові запити та аналізувати географічні дані.

Має простий та зрозумілий синтаксис запитів, що спрощує розробку додатків з її використанням.

Недоліки MongoDB:

Може вимагати значних обсягів оперативної пам'яті для оптимальної продуктивності. Це може бути обмеженням у випадку, коли пам'ять обмежена.

Відсутність транзакцій у певних версіях MongoDB не було повноцінної підтримки транзакцій. Хоча в новіших версіях було додано деякі можливості транзакцій, вони не настільки розширені та потужні, як у традиційних реляційних базах даних.

Вимоги до ресурсів, може вимагати значних обсягів дискового простору для зберігання даних та індексів. Це варто враховувати при плануванні потреб в обсягу дискового простору.

Складність реплікації та кластеризації, налаштування та управління реплікацією та кластеризацією MongoDB можуть бути складними та вимагати досвіду адміністратора баз даних.

Загалом, MongoDB є потужною NoSQL базою даних з великим набором функцій та переваг. Проте, важливо враховувати особливості та обмеження цієї бази даних при виборі її для конкретного проєкту.

Після аналізування переваг і недоліків різних типів баз даних, було прийнято рішення розробляти дипломний проєкт з використанням реляційної бази даних PostgreSQL, адже ця база даних підтримує широкий спектр функцій і можливостей, включаючи вбудовану підтримку геоданих, повнотекстовий

пошук, високорівневі операції з даними, транзакційну безпеку і багато іншого. Це робить PostgreSQL потужним і гнучким інструментом для різних проєктів.

Враховуючи раніше розроблену діаграму варіантів використання (див. рисунок 1.11), було встановлено, які дані необхідно зберігати в базі даних. Зокрема, це включає дані про користувача, нерухомість, категорії, типи операцій та фотографії. Таким чином було створено таблиці бази даних «units», «features_house», «features_apartment», «units_rentings», «units_sale», «users», «types_action», «images», «categories».

units – таблиця яка містить дані про всі наявні об’єкти нерухомості. Має наступні поля:

– unit_id – унікальний ідентифікатор одиниці;

– owner_id – ідентифікатор власника одиниці, пов'язаний з таблицею "users";

– category_id – ідентифікатор категорії, пов'язаний з таблицею "categories";

– type_action_id – ідентифікатор типу дії, пов'язаний з таблицею "types_action";

– type_ofer_id – ідентифікатор типу пропозиції, пов'язаний з таблицею "types_ofer";

– address_detail_id – ідентифікатор деталей адреси, пов'язаний з таблицею "address_details";

– feature_house_id – ідентифікатор особливостей будинку, пов'язаний з таблицею "features_house";

– feature_apartment_id – ідентифікатор особливостей квартири, пов'язаний з таблицею "features_apartment";

– feature_sale_id – ідентифікатор особливостей продажу одиниці, пов'язаний з таблицею "units_sale";

– feature_renting_id – ідентифікатор особливостей оренди одиниці, пов'язаний з таблицею "units_rentings";

– title – заголовок одиниці;

– description – опис одиниці;

– total_area – загальна площа одиниці;

– activated – прапорець, що вказує, чи є одиниця активованою;

– createdAt – дата і час створення запису;

– updatedAt – дата і час останнього оновлення запису;

– images – масив зображень, пов'язаних з цією одиницею;

features_house – дана таблиця описує особливості будинку. Має наступні поля:

– features_id – унікальний ідентифікатор особливостей будинку;

– type_heating_id – ідентифікатор типу опалення, пов'язаний з таблицею "types_heating";

– type_wall_id – ідентифікатор типу стін, пов'язаний з таблицею "types_wall;

– dwelling_area – площа проживання в будинку;

– kitchen_area – площа кухні в будинку;

– number_floors – кількість поверхів у будинку (за замовчуванням 1);

– number_rooms – кількість кімнат у будинку;

– plot_size – площа ділянки, на якій розташований будинок;

– unit_land_measurement – одиниця виміру землі (за замовчуванням "sotka");

– year_construction – рік будівництва будинку (необов'язкове поле);

– unit – одиниця, до якої належать ці особливості будинку, пов'язана з таблицею "units";

features_apartment – дана таблиця описує особливості квартири. Має наступні поля:

– features_id – унікальний ідентифікатор особливостей квартири;

– type_heating_id – ідентифікатор типу опалення, пов'язаний з таблицею "types_heating";

– type_wall_id – ідентифікатор типу стін, пов'язаний з таблицею "types_wall";

– type_building_id – ідентифікатор типу будівлі (квартирного комплексу), пов'язаний з таблицею "types_building_apartment";

– year_construction_building – рік будівництва будівлі (квартирного комплексу) (необов'язкове поле);

– number_floors_building – кількість поверхів у будівлі (квартирного комплексу);

– floor – поверх, на якому розташована квартира;

– dwelling_area – площа проживання в квартирі;

– kitchen_area – площа кухні в квартирі;

– number_floors – кількість поверхів у квартирі (за замовчуванням 1);

– number_rooms – кількість кімнат у квартирі;

– unit – одиниця, до якої належать ці особливості квартири, пов'язана з таблицею "units";

units_rentings – дана таблиця описує особливості оренди. Має наступні поля:

– unit_renting_id – унікальний ідентифікатор оренди одиниці;

– type_paid_id – ідентифікатор типу сплати (оплати оренди), пов'язаний з таблицею "types_paid_rent";

– type_rent_id – ідентифікатор типу оренди, пов'язаний з таблицею "types_rent";

– type_utility_id – ідентифікатор типу комунальних послуг, пов'язаний з таблицею "types_utility";

– currency_id – ідентифікатор валюти, пов'язаний з таблицею "units_currency";

– price – ціна оренди;

– is_trading – прапорець, що вказує, чи доступна одиниця для оренди (за замовчуванням true);

– createdAt – дата і час створення запису (за замовчуванням поточний час);

– updatedAt – дата і час останнього оновлення запису;

units_sale – дана таблиця описує особливості продажу. Має наступні поля:

– unit_sale_id – унікальний ідентифікатор продажу одиниці;

– currency_id – ідентифікатор валюти, пов'язаний з таблицею "units_currency";

– price – ціна продажу;

– is_trading – прапорець, що вказує, чи доступна одиниця для продажу (за замовчуванням true);

– createdAt – дата і час створення запису (за замовчуванням поточний час);

– updatedAt – дата і час останнього оновлення запису;

types_action – дана таблиця містить типи операції (оренда, продаж). Має наступні поля:

– type_action_id – унікальний ідентифікатор типу дії;

– type_action_name – назва типу дії, яка є унікальною;

– activated – прапорець, що вказує, чи активований цей тип дії (за замовчуванням false);

– createdAt – дата і час створення запису (за замовчуванням поточний час);

– updatedAt – дата і час останнього оновлення запису;

categories – дана таблиця містить всі категорії. Має наступні поля:

– category_id – унікальний ідентифікатор категорії;

– category_name – назва категорії, яка є унікальною;

– activated – прапорець, що вказує, чи активована ця категорія (за замовчуванням false);

– createdAt – дата і час створення запису (за замовчуванням поточний час);

– updatedAt – дата і час останнього оновлення запису;

users – дана таблиця зберігає всіх користувачів. Має наступні поля:

– user_id – унікальний ідентифікатор користувача;

– first_name – ім'я користувача (за замовчуванням "not specified");

– last_name – прізвище користувача (за замовчуванням "not specified");

– phone – номер телефону користувача;

– email – електронна адреса користувача (унікальна);

– role – роль користувача (за замовчуванням "customer");

– password_hash – хеш паролю користувача;

– createdAt – дата і час створення запису (за замовчуванням поточний час);

– updatedAt – дата і час останнього оновлення запису;

images – дана таблиця зберігає зображення. Має наступні поля:

– image_id – унікальний ідентифікатор зображення;

– name – назва зображення (максимальна довжина 75 символів);

– url – URL-адреса зображення (унікальна);

– unit – зв'язок з моделлю "units" через поле "unit_id", що вказує на одиницю, пов'язану з цим зображенням;

– unit_id – ідентифікатор одиниці, пов'язаної з зображенням;

– createdAt – дата і час створення запису (за замовчуванням поточний час);

– updatedAt – дата і час останнього оновлення запису;

Таким чином, після створення логічної моделі бази даних, ми можемо приступити до проектування фізичної моделі, використовуючи її як основу (структура бази даних рисунок 2.1).

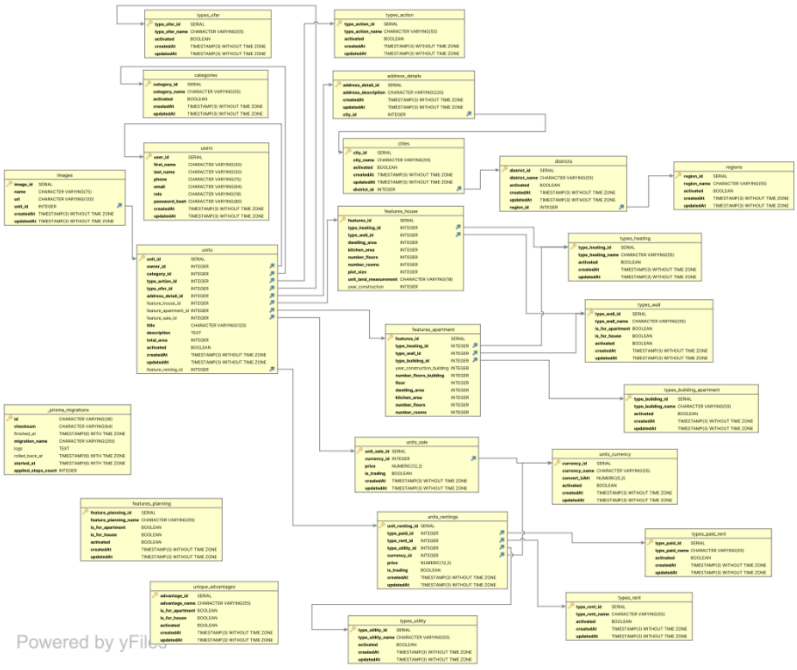


Рисунок 2.1 – Структура бази даних

В результаті, отримано значну кількість полів, які є необхідними для належної функціональності розроблюваного веб-сайту.

Варто відзначити, що деякі колекції мають повторювані поля, такі як UserId. Це здійснено з метою зв'язку з відповідними колекціями, що стосуються користувача та інформації про будинки.

Отже, була розроблена логічна модель бази даних, і згідно з нею можна перейти до проектування фізичної моделі.

2.3 Проектування серверної частини веб-застосунка

У сучасному світі розробка програмного забезпечення стає все більш складною та багатогранною задачею. Однією з ключових складових будь-якого сучасного додатку є серверна частина, яка відповідає за обробку запитів, взаємодію з базою даних та надання необхідної функціональності клієнтській частині.

Проектування серверної частини додатку є важливим етапом в розробці програмного забезпечення. Цей процес включає визначення архітектури, вибір технологій, проектування бази даних, створення API та бізнес-логіки.

Основні кроки проектування серверної частини додатку включають наступне:

Визначення вимог – ретельне вивчення вимог до додатку, включаючи функціональність, продуктивність, масштабованість та безпеку.

Архітектура – вибір підходящої архітектури для додатку. Це може бути клієнт-серверна архітектура, мікросервісна архітектура, шарова архітектура тощо. Розробка логічної та фізичної моделей бази даних.

Вибір технологій – вибір мови програмування, фреймворків, серверних платформ та інструментів, які відповідають вимогам проєкту. Враховуються фактори, такі як продуктивність, масштабованість, спільнота розробників та інші.

Проектування API – визначення і проектування API, яке взаємодіє з клієнтською частиною додатку. Враховуються розширюваність, зручність використання та документація API.

Розробка бізнес-логіки – реалізація логіки додатку, яка включає обробку запитів, валідацію даних, авторизацію, обробку помилок та інші бізнес-правила.

Безпека – забезпечення безпеки додатку, включаючи захист від атак, аутентифікацію, авторизацію та шифрування даних.

Тестування – розробка та виконання тестів для перевірки працездатності та якості серверної частини додатку.

Масштабування – розробка механізмів для масштабування додатку, якщо потрібно, для підтримки зростаючої навантаженості та обсягів даних.

Проектування серверної частини додатку є складним процесом, який вимагає ретельного аналізу та планування. Від правильного проектування залежить успіх та продуктивність додатку під час його експлуатації.

Для взаємодії з серверною частиною та керування запитами передбачається використання REST API - інтерфейсу програмування застосунків, який надає можливість створювати та керувати запитами до сервера. REST API (Representational State Transfer Application Programming Interface) - це архітектурний стиль та підхід до створення веб-сервісів, який базується на принципах REST. REST API використовує протокол HTTP для передачі даних та дозволяє взаємодіяти з веб-додатками та сервісами за допомогою стандартних HTTP-методів, таких як GET, POST, PUT і DELETE.

Найпоширеніші підходи до створення серверної частини додатку включають монолітну та мікросервісну архітектуру, ми розберемо кожну з них аби зрозуміти що нам підійде найкраще.

Монолітна архітектура: В цьому підході весь додаток розглядається як єдиний монолітний блок. Весь функціонал додатку реалізується у рамках одного великого програмного модуля. Цей підхід простий у розробці і розгортанні, але може стати обмеженням у разі масштабування та підтримки розширення функціоналу. Структура монолітної архітектури представлено на рисунку 2.2.

Монолітна архітектура має наступні переваги:

– Простота розробки: У монолітній архітектурі весь функціонал додатку знаходиться в єдиному програмному модулі, що спрощує розробку і зменшує складність.

– Простота розгортання: Розгортання монолітного додатку вимагає менше зусиль, оскільки потрібно розгорнути лише один програмний модуль.

– Простота тестування: У монолітній архітектурі тестування може бути менш складним, оскільки всі компоненти додатку взаємодіють у межах одного модуля.

Проте, монолітна архітектура також має свої недоліки:

– Складність масштабування: Монолітні додатки складніше масштабувати горизонтально, оскільки весь функціонал розміщений в одному програмному модулі.

– Залежність компонентів: Зміни або помилки в одному компоненті можуть вплинути на решту додатку, оскільки вони спільно використовують одну базу коду та ресурси.

– Обмежені можливості технологічного вибору: У монолітній архітектурі весь додаток використовує одну технологічну стек, що може обмежувати вибір оптимальних інструментів для кожного компоненту.

– Складність розробки великих проєктів: У великих монолітних проєктах розробка, розгортання і розширення можуть стати складними завданнями через велику кількість коду і функціоналу.

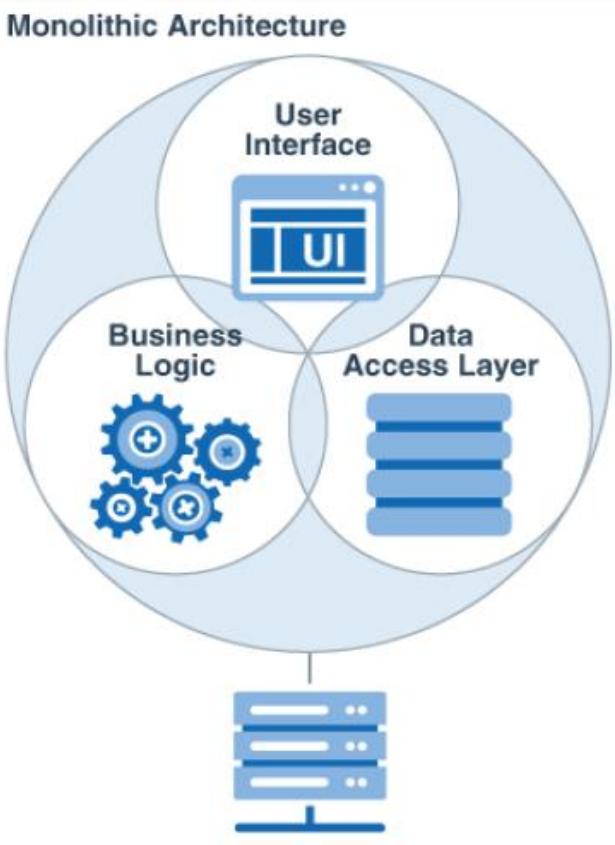


Рисунок 2.2 - Монолітна архітектура - https://www.atlassian.com/

Мікросервісна архітектура: У цьому підході функціональність додатку розбивається на невеликі, самостійні мікросервіси, які можуть бути розгорнуті

та масштабовані окремо. Кожен мікросервіс виконує конкретні завдання і спілкується з іншими мікросервісами через API. Цей підхід дозволяє гнучко масштабувати окремі компоненти додатку, але також вимагає складнішої архітектури та управління. Структура мікросервісної архітектури показана на рисунку 2.3.

Переваги мікросервісної архітектури:

– Модульність і розділення відповідальностей: Система розбивається на невеликі, самостійні сервіси, що полегшує розробку, модифікацію і розширення системи.

– Гнучкість і масштабованість: Кожен мікросервіс може бути масштабований окремо, що дозволяє ефективно використовувати ресурси і забезпечувати високу продуктивність.

– Незалежна розгортка і розвиток: Кожен сервіс може бути розгорнутий, оновлений і масштабований незалежно від інших, що полегшує розвиток і підтримку системи.

Недоліки мікросервісної архітектури:

– Складність управління: З кожним мікросервісом пов'язані додаткові виклики і управління, що може призвести до збільшення складності системи.

– Комунікація і мережеві запити: Мікросервіси взаємодіють між собою через мережу, що може вплинути на продуктивність і збільшити затримки.

– Вимоги до моніторингу і управління: З ростом кількості мікросервісів необхідно використовувати ефективні інструменти для моніторингу, управління та відстеження проблем.

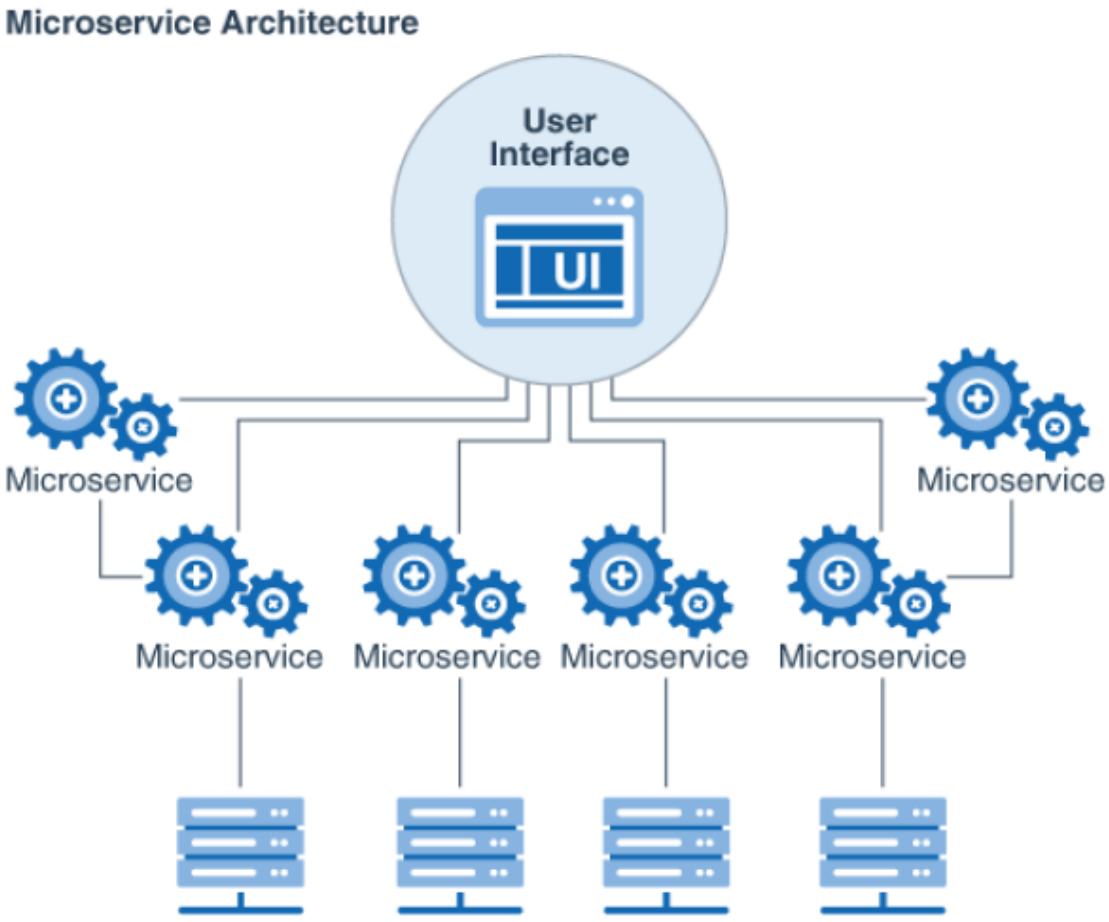


Рисунок 2.3 - Мікросервісна архітектура - https://www.atlassian.com/

Таким чином, можна зробити висновок, що монолітну архітектуру доцільно використовувати для розробки простих та невеликих програмних систем, які не вимагають значних ресурсів. З іншого боку, мікросервісна архітектура підходить для більш складних систем з великою кількістю незалежних модулів. У зв'язку з цим, для даного дипломного проєкту було обрано монолітну архітектуру, враховуючи його середню складність та обмежений час на розробку. Модулі даної системи будуть міцно пов'язані між собою, тому розбиття їх на окремі сервіси не є доцільним з точки зору затрат на розгортання та тестування.

2.4 Проектування інтерфейсу користувача

Для того, щоб створити дизайн проєкту було вирішено використати метод прототипування. Прототипування є важливою частиною процесу розробки користувацького інтерфейсу і дозволяє створювати і перевіряти концепції та функціональність інтерфейсу перед його фінальною реалізацією. Ось декілька ключових аспектів прототипування:

– Цілі прототипування: прототипи створюються для дослідження та валідації різних аспектів інтерфейсу, таких як навігація, розташування елементів, взаємодія з користувачем та функціональність. Мета полягає у забезпеченні ефективної та зручної взаємодії з користувачем.

– Типи прототипів: існують різні типи прототипів, від низькорівневих схематичних набросків до високорівневих інтерактивних прототипів. Вибір типу прототипу залежить від конкретних потреб проєкту та етапу розробки.

– Інструменти для прототипування: існує багато інструментів для створення прототипів, включаючи спеціалізовані програми для дизайну інтерфейсів, онлайн-інструменти, фреймворки та бібліотеки. Ці інструменти дозволяють створювати прототипи з різним рівнем деталізації та інтерактивності.

– Тестування та збір відгуків: прототипи використовуються для тестування та отримання відгуків від користувачів. Це дозволяє здійснити коригування та внести зміни до прототипу з метою покращення його властивостей та відповідності потребам користувачів.

Спочатку необхідно спроектувати головну сторінку веб-застонку. Головна сторінка повинна містити секцію фільтрів, сортування, та відображення результатів отже розглянемо кожну з них.

Розберемо секцію фільтрів на рисунку 2.4.

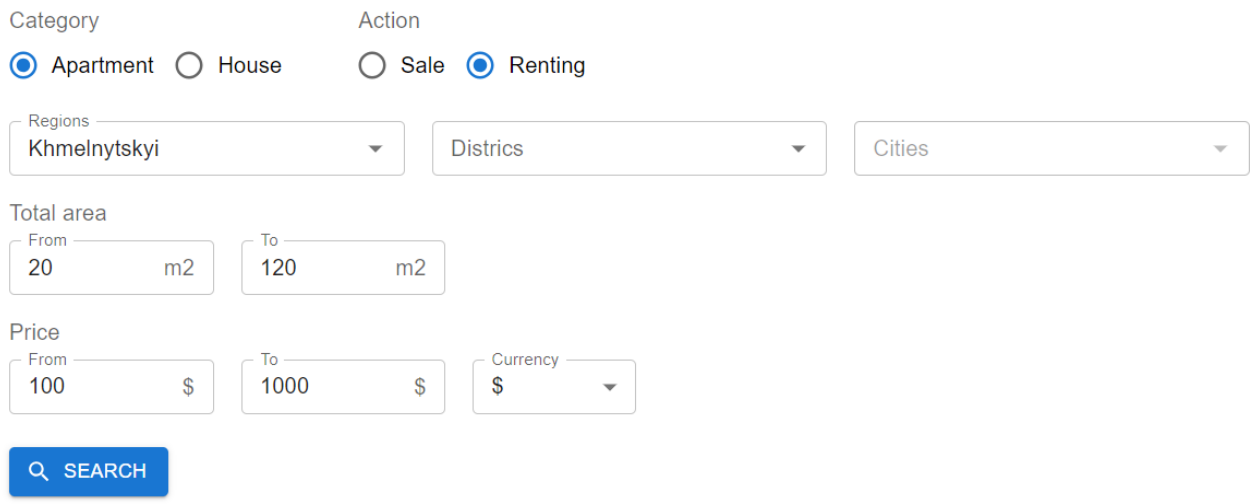


Рисунок 2.4 – Секція фільтрів

В даній секції ми спостерігаємо що користувач матиме можливість вибрати необхідну йому категорію, місце знаходження нерухомості, загальну площу та ціну. Тепер розберемо секцію фільтрів на рисунку 2.5.

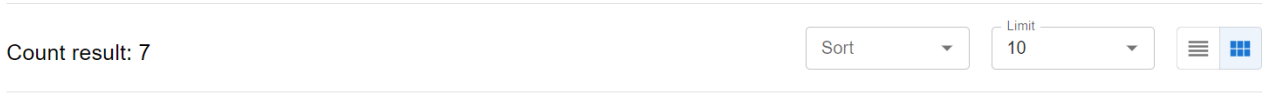


Рисунок 2.5 – Секція сортування

В секції сортування міститься інформація про загальну кількість найдених об’єктів нерухомості. Поля в яких можна вибрати варіант сортування та ліміт для пагінації а також дизайн відображення виведених елементів.

Також потрібно розробити сторінку для створення оголошення. Сторінка матиме різні поля в залежності між вибором категорії. Для початку розглянемо загальний вигляд сторінки на рисунку 2.6.

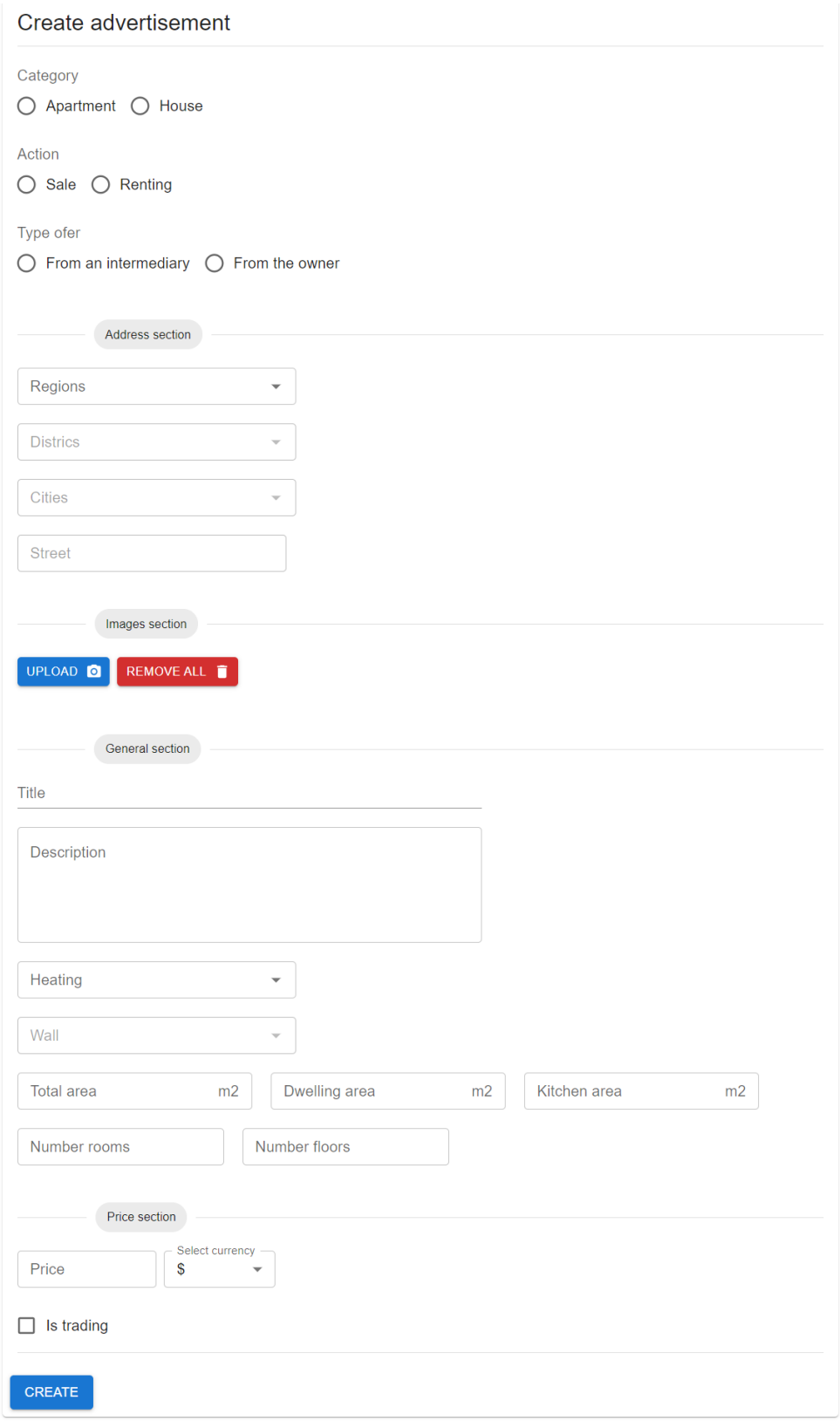


Рисунок 2.6 – Загальний вигляд сторінки створення оглошення

При виборі категорії квартири добавляється спеціальна секція з полями які описують дані що стосуються виключно квартир це можна спостерігати на рисунку 2.7, аналогічна ситуація при виборі категорії будинки це можна спостерігати на рисунку 2.8.

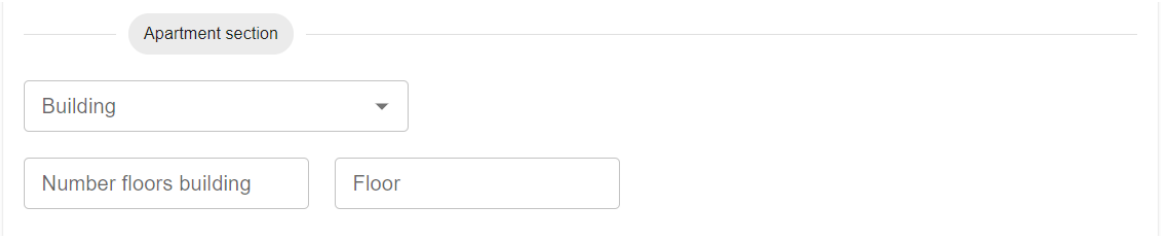


Рисунок 2.7 – Додаткові поля для квартир

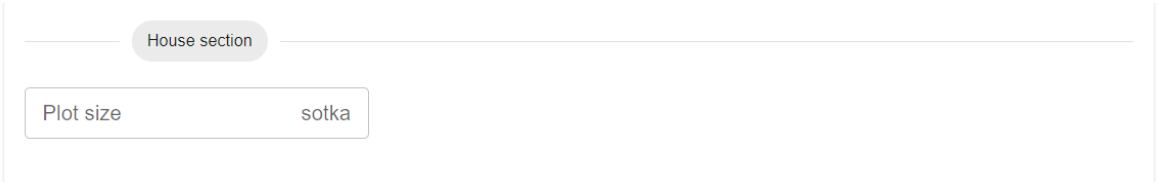


Рисунок 2.8 – Додаткові поля для будинків

Відповідно при виборі оренди додаються додаткові поля. На рисунку 2.9 маємо можливість спостерігати на додаткові поля для оренди.

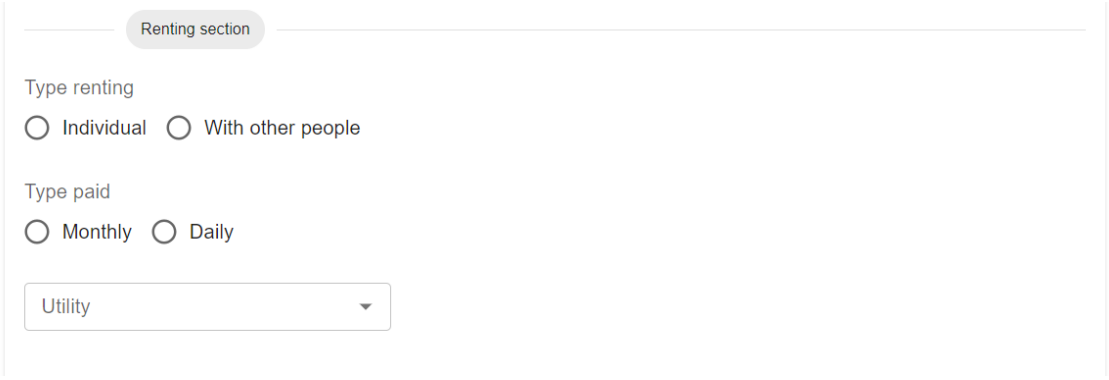


Рисунок 2.9 – Додаткові поля для оренди

2.5 Аналіз та вибір технологій і методів реалізації веб-застосунка

Сучасна розробка вимагає сучасних та актуальних технологій, але також потрібно розуміти що не всі сучасні технології потрібно використовувати тільки тому що про них багато хто говорить, адже недолік сучасних технології полягає в тому, що вони погано протестовані на реальних проєктах.

Враховуючи потреби веб-застосунку та аналізуючи технології було пряйнято рішення вибрати наступні інструменти. Мова програмування TypeScript, повноціний фрейвморк для розробки інтерфейсу користувача NextJS, для розробки серверної частини платформа NodeJS та найбільш прихильний для багатьох розробників фреймворк NestJS. База даних PostgreSQL та ORM Prisma. Також при розробці буде використовуватись HTMl, CSS, та різні бібліотеки. Отже розберемо всі переваги та недоліки кожної з технологій.

TS (TypeScript) - це мова програмування, яка є розширенням JavaScript з додатковими функціями, такими як статичне типізування, класи, інтерфейси та модулі.

Переваги TypeScript:

– Статична типізація: TypeScript дозволяє визначати типи даних змінних, параметрів функцій та повернутих значень, що полегшує виявлення та усунення помилок ще на етапі розробки. Це сприяє збільшенню безпеки та надійності коду.

– Розширена функціональність: TypeScript надає додаткові можливості, такі як класи, інтерфейси, декоратори та модулі, що дозволяє розробникам писати більш структурований та організований код. Це сприяє полегшенню розробки, підтримці та розширенню проєктів.

– Підтримка інструментів: TypeScript має широку підтримку серед редакторів коду та інтегрованих середовищ розробки. Багато редакторів, таких як Visual Studio Code, WebStorm та інші, надають розширену функціональність

для TypeScript, включаючи підказки, автодоповнення, виявлення помилок та рефакторинг.

– Поступове впровадження: TypeScript дозволяє поступово впроваджувати типізацію у вже існуючих JavaScript-проєктах. Розробники можуть додавати типи поступово, що дозволяє забезпечити більшу надійність та підтримку коду без необхідності переписування його повністю.

Недоліки TypeScript:

– Навчання та підготовка: Використання TypeScript вимагає додаткового часу та зусиль для оволодіння синтаксисом, концепціями та особливостями мови. Розробники, які не мають досвіду з TypeScript, можуть потребувати додаткового часу на навчання та адаптацію.

– Зайвий оверхед: Використання статичної типізації та інших функцій TypeScript може призвести до додаткового обсягу коду та збільшення розміру вихідного файлу. Це може вплинути на продуктивність додатку, особливо при великих проєктах.

– Залежність від інтерфейсу: TypeScript покладається на визначення типів інтерфейсів, що може стати проблемою, якщо структура даних змінюється часто або вимагає великої кількості інтерфейсів. Це може призвести до збільшення зусиль для оновлення та підтримки типізації.

Хоча TypeScript має свої недоліки, загальною думкою є те, що переваги, які він надає, переважають недоліки, особливо при розробці великих та складних проєктів. Вибір між TypeScript і JavaScript залежить від конкретних потреб та вимог проєкту.

Next.js - це реактивний фреймворк для розробки веб-додатків на базі React. Він надає покращену підтримку для серверного рендерингу, статичного генерування, маршрутизації та багатьох інших функцій, що полегшують розробку масштабованих та продуктивних додатків. Next.js також підтримує модель розширень, що дозволяє додавати функціональність через плагіни та

middleware. Його переваги включають простоту використання, ефективність та здатність працювати як з серверним, так і з клієнтським рендерингом.

Переваги Next.js:

– Серверний рендеринг: Next.js надає потужну підтримку для серверного рендерингу, що дозволяє веб-сторінкам відображатися швидко і мати коректну індексацію пошуковими системами.

– Статичне генерування: Next.js дозволяє генерувати статичні HTML-сторінки під час збірки, що полегшує кешування та покращує продуктивність.

– Хороша маршрутизація: Next.js надає простий та потужний механізм маршрутизації, що дозволяє легко налаштовувати шляхи до веб-сторінок.

– Розширення та плагіни: Next.js підтримує модель розширень, що дозволяє додавати функціональність через плагіни та middleware.

Недоліки Next.js:

– Комплексність: За деякими сценаріями, особливо при великому обсязі проєкту, Next.js може бути трохи складним у використанні та налаштуванні.

– Вивчення кривої навчання: Хоча Next.js має хорошу документацію, вивчення нового фреймворку може зайняти певний час та зусилля для розробників, які раніше не працювали з ним.

– Залежність від React: Next.js базується на React, тому знання React є передумовою для розробки з використанням Next.js. Це може бути обмеженням для розробників, які віддають перевагу іншим фреймворкам або бібліотекам.

NestJS - це прогресивний фреймворк для розробки серверних додатків на Node.js.

Переваги NestJS:

– Модульність: NestJS пропонує модульну структуру, що дозволяє організовувати код у логічні модулі та повторно використовувати його. Це спрощує розробку та підтримку додатків.

– TypeScript: NestJS побудований на основі TypeScript, що дозволяє використовувати сильну типізацію, забезпечує підказки під час розробки та поліпшує статичний аналіз коду.

– Декоратори: NestJS використовує декоратори для опису різних аспектів додатку, що спрощує розробку та забезпечує чистоту коду.

– Широкий екосистема: NestJS інтегрується з різними бібліотеками та фреймворками, такими як Express, TypeORM, GraphQL тощо. Це надає багато готових рішень для розробки різноманітних додатків.

– Підтримка реактивного програмування: NestJS надає підтримку для розробки реактивних додатків, що дозволяє обмінюватися реальними часовими даними з клієнтом.

Недоліки NestJS:

– Навчання: NestJS може мати деякий крутий початковий поріг навчання, особливо для розробників без досвіду з TypeScript та модульною архітектурою.

– Ресурсоємність: Завдяки додатковому шару абстракції та використанню TypeScript, NestJS може бути трохи більш ресурсоємким порівняно з іншими фреймворками.

– Залежність від екосистеми: Хоча широка екосистема NestJS є перевагою, це також означає, що деякі вибіркові компоненти або бібліотеки можуть бути менш популярними або менше підтримуватись.

– Перевантаження: Іноді NestJS може видаватись зайвою складністю для простих проєктів, які не потребують великого рівня абстракції та модульності.

PostgreSQL - це потужна та розширювана система управління базами даних (СУБД), яка використовує мову запитів SQL.

Переваги PostgreSQL:

– Надійність: PostgreSQL є дуже надійною системою управління базами даних з високою стабільністю та цілісністю даних.

– Розширюваність: PostgreSQL надає гнучкість та можливості для розширення бази даних шляхом використання розширень та модулів.

– Підтримка продвинутих функцій: PostgreSQL має потужні функції, такі як робота з геоданими, повнотекстовий пошук, робота з JSON та багато інших.

– Висока продуктивність: PostgreSQL забезпечує швидку та ефективну роботу з базою даних навіть при обробці великої кількості даних.

Недоліки PostgreSQL:

– Складність: PostgreSQL може бути складним для новачків, оскільки має велику кількість функцій та налаштувань.

– Вимоги до ресурсів: PostgreSQL вимагає достатньо великої кількості ресурсів, таких як пам'ять та процесор, для ефективної роботи.

– Обмежена підтримка географічної реплікації: PostgreSQL має обмежену підтримку географічної реплікації, що може бути проблемою для додатків, що вимагають географічного розподілення даних.

– Обмежена підтримка горизонтального масштабування: Порівняно з деякими іншими СУБД, PostgreSQL має обмежену підтримку горизонтального масштабування, що може бути проблемою для великих додатків з високими навантаженнями.

Висновки до розділу

У цьому розділі було здійснено проектування системи та вибрано оптимальний варіант її архітектури, враховуючи особливості предметної області. Згідно з вимогами специфікації, був розроблений користувацький інтерфейс. Також була спроектована структура бази даних. Була розроблена архітектура системи та визначені її компоненти.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ

3.1 Розробка бази даних

Для того щоб працювати з базами даних PostgreSQL потрібно встановити pgadmin4. Після цього ми маємо інструмент в якому можемо будувати нашу БД.

Інтерфейс PgAdmin зображений на рисунку 3.1

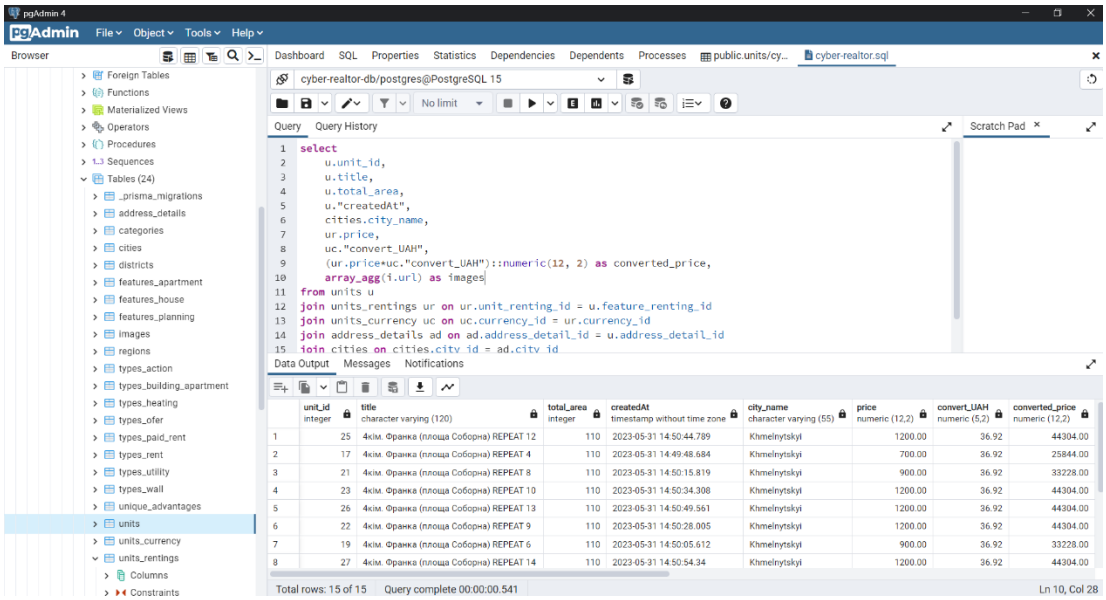


Рисунок 3.1 - Інтерфейс PgAdmin

Щоб взаємодіяти з базами даних PostgreSQL ми будемо використовувати ORM Prisma. Prisma відкриває новий рівень досвіду розробників під час роботи з базами даних завдяки своїй інтуїтивно зрозумілій моделі даних, автоматичним міграціям, безпеці типів і автозавершенню.

Щоб розпочати роботу, встановимо Prisma CLI як залежність від розробки. Prisma CLI дозволить запускати різні команди та взаємодіяти з проєктом.

Node.js дає змогу легко взаємодіяти з базами даних MongoDB. Для цього знадобиться драйвер MongoDB Node.js, щоб підключитися до бази даних і виконати запити. Його можна встановити його за допомогою наступної команди:

```
npm install -D prisma
```

Потрібно ініціалізувати Prisma у проєкті, виконавши:

```
npx prisma init
```

Це створить новий каталог prisma з файлом schema.prisma. Це основний конфігураційний файл, який містить схему бази даних. Ця команда також створює файл .env у проєкті.

У файлі .env ми повинні бачимо змінну середовища DATABASE_URL із фіктивним рядком підключення. Замінимо цей рядок підключення на рядок для нашого екземпляра PostgreSQL.

```
DATABASE_URL="postgresql://postgres:userpostgress@localhost:5432/cyber-realtor-db"
```

У файлі schema.prisma описуємо таблиці які будуть присутні у нашій БД. Описавши всі таблиці у схемі Prisma, запускаємо міграції для створення фактичних таблиць у базі даних. Щоб згенерувати та виконати першу міграцію, виконуємо таку команду в терміналі:

```
npx prisma migrate dev --name "init"
```

У програмі NestJS рекомендовано абстрагуватися від API клієнта Prisma. Для цього створимо новий сервіс, який міститиме Prisma Client. Ця служба під назвою PrismaService відповідатиме за створення екземпляра PrismaClient і підключення до бази даних.

Nest CLI дає простий спосіб генерувати модулі та служби безпосередньо з CLI. Виконаємо таку команду у терміналі:

```
npx nest generate module prisma
npx nest generate service prisma
```

Далі напишимо наступний код у файлі prisma.service.ts аби ми могли використовувати сервіс prisma:

```
import { INestApplication, Injectable } from '@nestjs/common';
import { PrismaClient } from '@prisma/client';

@Injectable()
export class PrismaService extends PrismaClient {
  async enableShutdownHooks(app: INestApplication) {
    this.$on('beforeExit', async () => {
      await app.close();
    });
  }
}
```

І ще необхідно написати наступний код у файлі prisma.module.ts:

```
import { Module } from '@nestjs/common';
import { PrismaService } from './prisma.service';

@Module({
  providers: [PrismaService],
  exports: [PrismaService],
})
export class PrismaModule {}
```

Після цього ми закінчили налаштування Prisma! Тепер ми приступимо до створення REST API.

3.2 Розробка програмних модулів

Проєкт містить дві складові: серверну частину та клієнтська частину, які розроблялися як окремі додатки тому ми будемо розглядати їх по черзі і розпочнемо з серверної частини, структура проєкту серверного додатку зображена на рисунку 3.2.

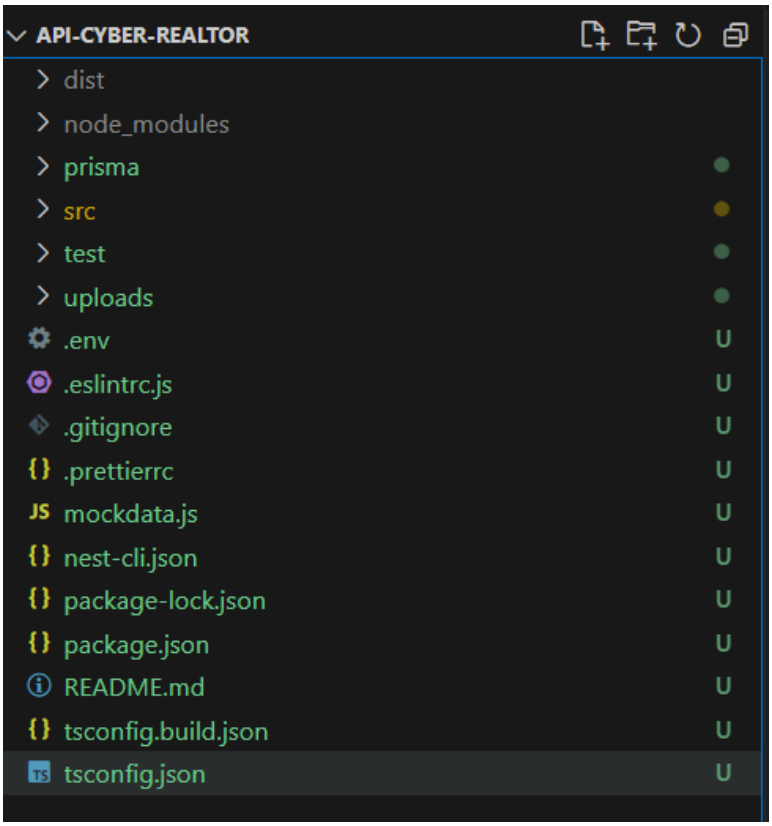


Рисунок 3.2 - Структура серверного додатку

Додаток містить чотири основних папок:

– prisma – містить файли для взаємодії з базою даних;

– src – містить всі модулі;

– test – містить всі файли тестів;

– uploads – містить папки з зображеннями;

Розглянемо папку prisma (рисунок 3.3)

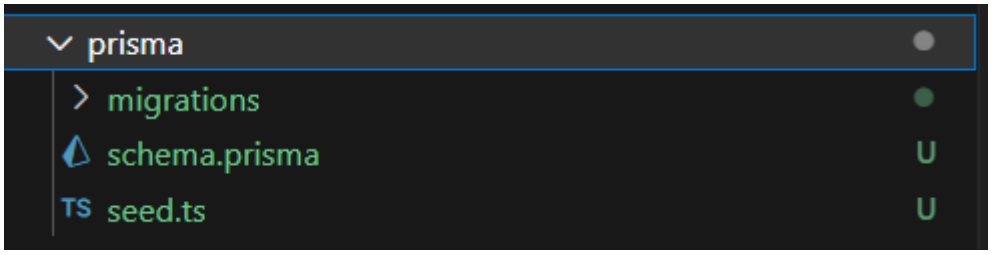


Рисунок 3.3 - Структура папки prisma

Вона складається з папки migrations де зберігаються всі міграції до бази даних, з файла schema.prisma де описується структура кожної таблиці (приклад опису структури таблиці зображено на рисунку 3.4) та файла seed.ts де знаходяться дані які потрібно записати у базу даних після міграцій.

```
model categories {
  category_id   Int      @id @default(autoincrement())
  category_name String   @unique @db.VarChar(55)
  activated     Boolean  @default(false)
  createdAt     DateTime @default(now())
  updatedAt     DateTime @updatedAt
  units         units[]
}
```

Рисунок 3.4 – Приклад структури таблиці

Тепер розглянемо папку src в якій містяться всі модулі необхідні для коректної роботи серверного додатку, коротко розглянемо кожний з них.

– categories – містить файли для роботи з категоріями;
– cities – містить файли для роботи з містами;
– districts – містить файли для роботи з районами;
– regions – містить файли для роботи з регіонами;
– units– містить файли для роботи з об’єктами нерухомості;
– images – містить файли для роботи з зображеннями;

Розглянемо структуру кожного модуля на прикладі categories (приклад дивитися на рисунку 3.5).

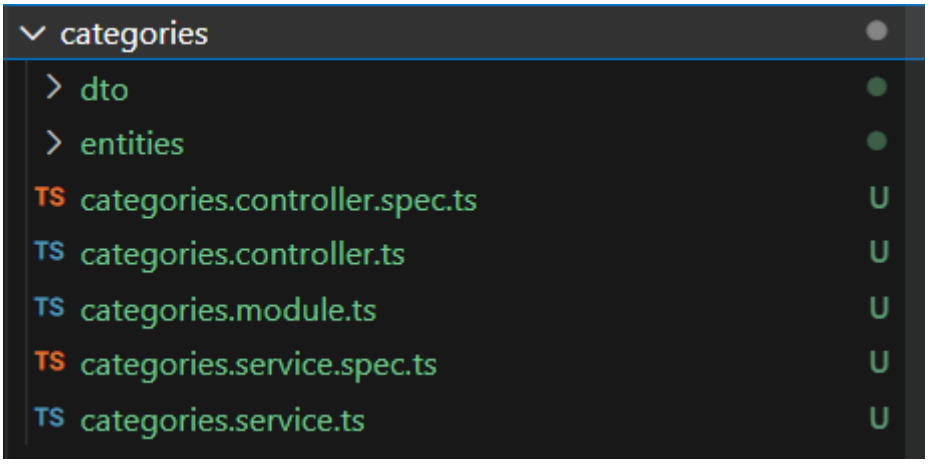


Рисунок 3.5 – Приклад структури модуля categories

Модуль складається з таких файлів як controller, module, service. Controller призначений для того аби приймати запити, service відповідає за бізнес логіку, module об’єднує всі ці файли за для подальшого їх користування. Найбільш цікавий та важлий це service який відповідає за бізнес логіку, переважно це саме той файл в якому виконується найбільше роботи.

Приклад сервісу:

```
@Injectable()
export class CategoriesService {
  constructor(private prisma: PrismaService) {}
  create(createCategoryDto: CreateCategoryDto) {
    return this.prisma.categories.create({ data: createCategoryDto
});
  }

  findAll() {
    return this.prisma.categories.findMany({
      where: { activated: true },
      select: {
        category_id: true,
        category_name: true,
      },
    });
  }

  findOne(id: number) {
    return this.prisma.categories.findFirst({ where: { category_id:
id } });
  }

  update(id: number, updateCategoryDto: UpdateCategoryDto) {
    return this.prisma.categories.update({
      where: { category_id: id },
```

```
      data: updateCategoryDto,
    });
  }

  remove(id: number) {
    return this.prisma.categories.delete({ where: { category_id: id
} });
  }
}
```

Розглянемо структуру папки test на рисунку 3.6.

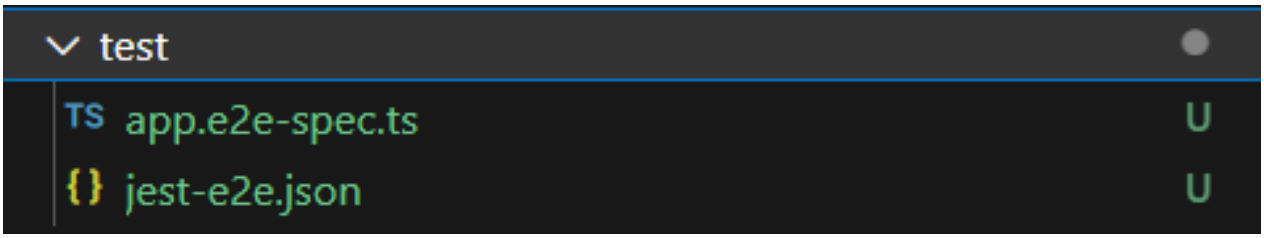


Рисунок 3.6 – Структура папки test

Тепер розглянемо папку uploads. В папці містяться інші папки, які призначені для зберігання зображень конкретного об’єкту нерухомості. Тобто кожна одиниця нерухомості має свою унікальну папку з своєми зображеннями. Розглянемо структуру папки uploads на рисунку 3.7 та вміст папки з зображеннями об’єкта нерухомості на рисунку 3.8.

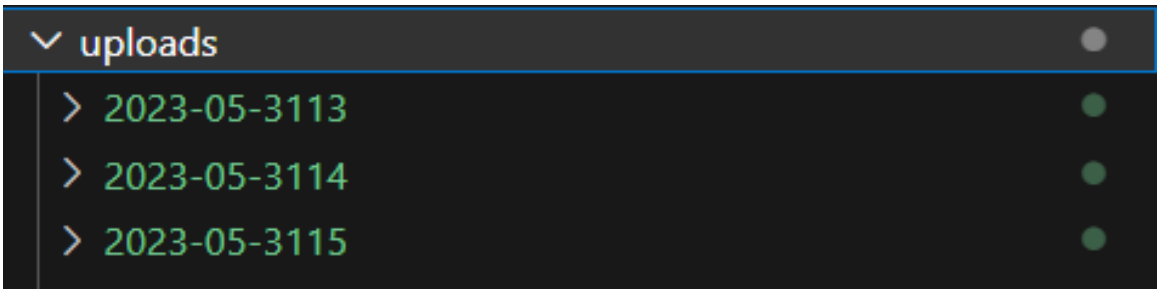


Рисунок 3.7 – Структура папки uploads

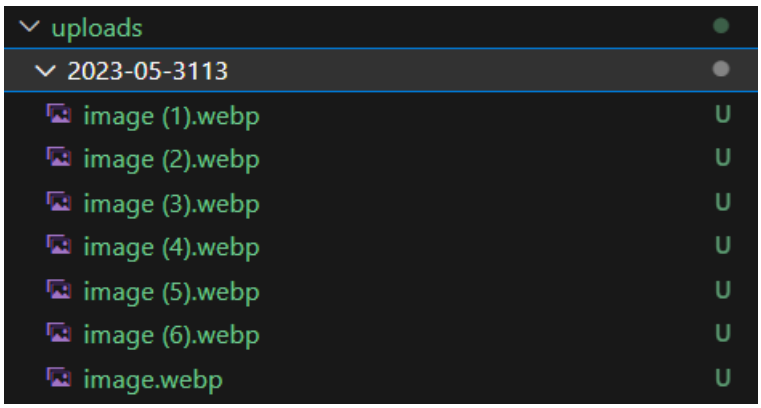


Рисунок 3.8 – Структура папки зображень об’єкта нерухомості

Також розглянемо структуру клієнтського додатку на рисунку 3.9.

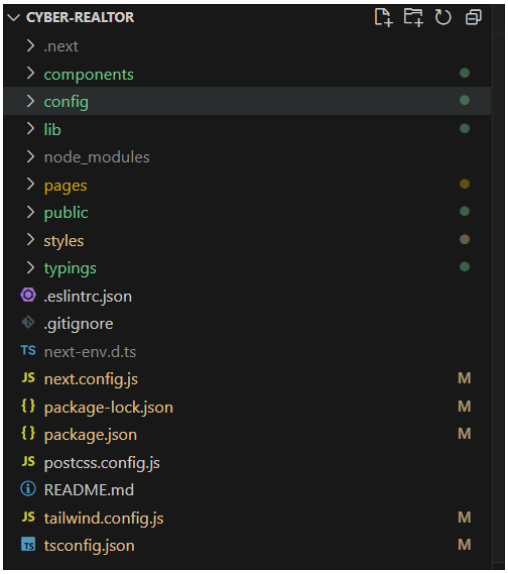


Рисунок 3.9 – Структура клієнтського додатку

Клієнтський додаток містить три основні папки pages, components, styles.

– pages – папка містить файли, які відповідають за сторінки додатку. Кожен файл у цій папці представляє окрему сторінку;

– components – тут зазвичай компоненти, які використовуються на різних сторінках додатку. Це можуть бути загальні компоненти, які можна повторно використовувати, або специфічні компоненти, які використовуються лише на певних сторінках;

– styles – у цій папці можна зберігати файли стилів, такі як CSS або SCSS. Це дозволяє організувати стилізацію компонентів та сторінок додатку;

Розглянемо структуру папки components на рисунку 3.10.

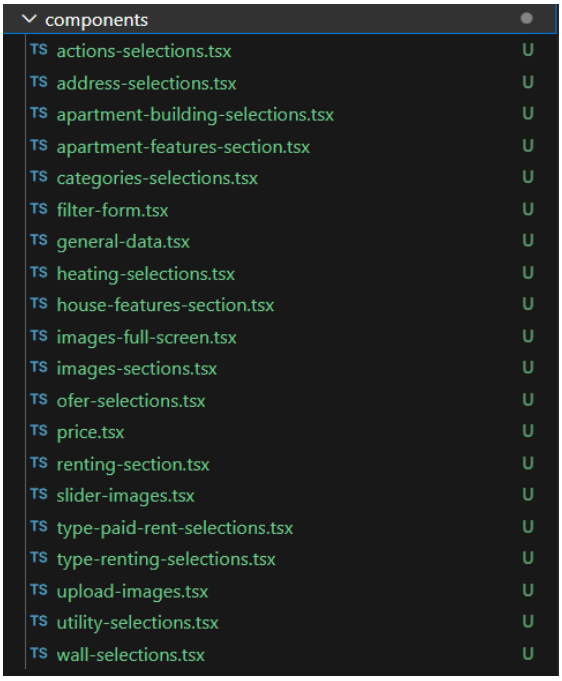


Рисунок 3.10 – Структура папки components

Тепер розглянемо код компонент categories-selections.tsx, щоб зрозуміти як будуються решта компонентів. Для початку нам потрібно отримати всі категорії з сервера після чого відобразити їх для користувача.

Код отримання всіх категорії:

```
async function getData() {
      try {
        const categoriesData = await
fetch("http://localhost:4000/categories");
        const categoriesParse = await categoriesData.json();

        // поміщаємо отриманні категорії в стейт
        setCategories(categoriesParse);

        // вказуємо на те що завантаження відбулось успішно
        setLoading(true);
      } catch (error) {
        // виводимо можливі помилки
        console.error(error);
```

```
    }
  }
```

Код відображення категорії:

```
return (
      <FormControl>
        <FormLabel id="radio-categories">Category</FormLabel>
        <RadioGroup
          aria-labelledby="radio-categories"
          name="radio-categories-group"
          row
        >
          {categoriesElement}
        </RadioGroup>
      </FormControl>
    );
```

3.3 Керівництво користувача

Коли користувач заходить на розроблений сайт, першою сторінкою, яка йому відкриється, буде головна сторінка веб-застосунку (рисунок 3.1). На цій сторінці розміщується секція фільтрів та об’єкти нерухомості. У секції фільтрів користувач має змогу вибрати категорію нерухомості а також для чого йому нерухомість, для ореди чи купівлі. Можна вибрати область, район, та місто де користувач хоче шукати нерухомість. За потреби користувач має можливість вказувати діапазон площі, та діапазон цін з можливістю вказування валюти.

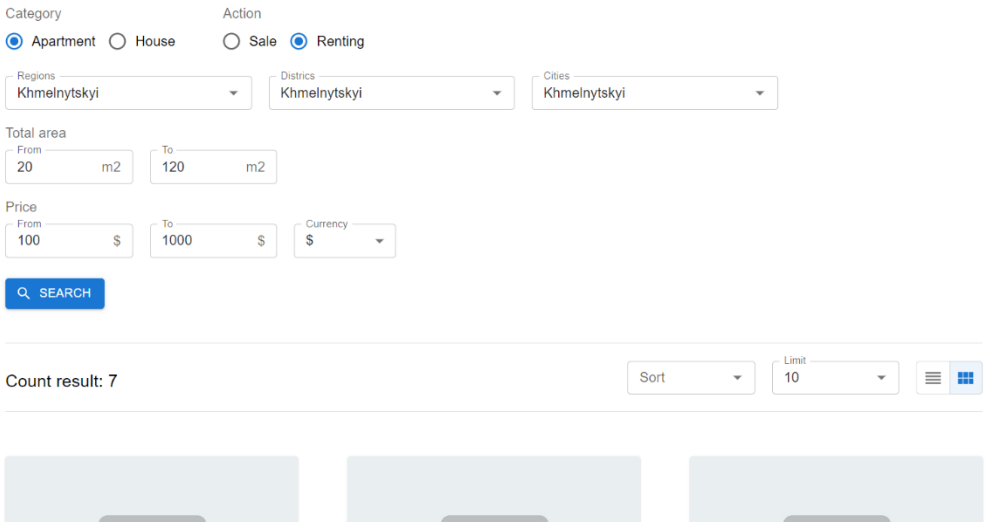


Рисунок 3.11 – Головна сторінка веб-застосунку

Після вибору нерухомості користувач має змогу переглянути фото нерухомості у двох режимах, звичайному (зображено на рисунку 3.2) та повноекраному (зображено на рисунку 3.3). Для того аби переглянути фото у повноекраному режимі користувачу доведеться два рази клацнути на зображення, після чого відкривається модальне вікно на всю площу сторінки, також при цьому режимі є можливість розглядати фото збільшуючи його, для цього потрібно два рази клацнути на фото та за допомогою миші оглядати бажанні частини (зображено на рисунку 3.4).

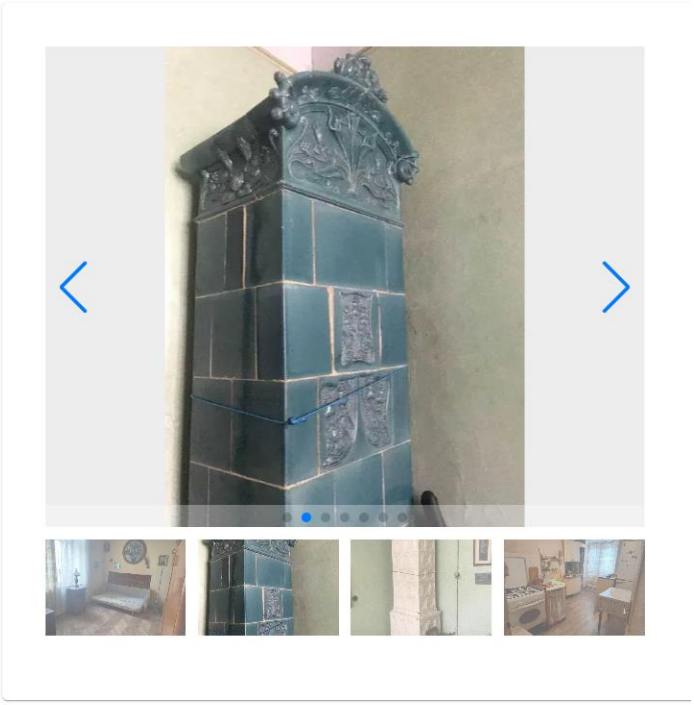


Рисунок 3.12 – Перегляд фото у звичайному режимі

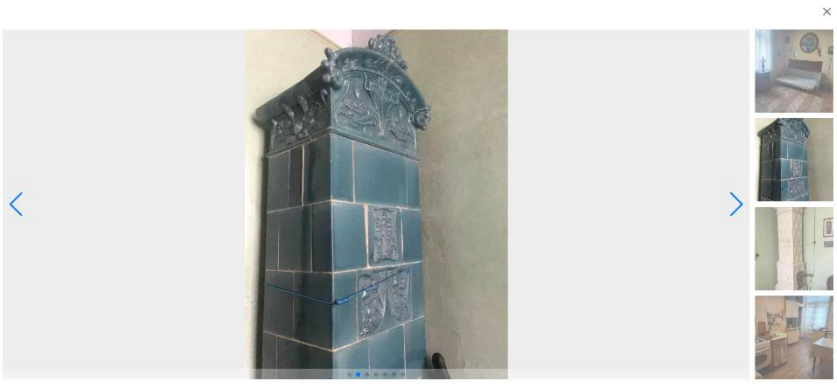


Рисунок 3.13 – Перегляд фото у повноекраному режимі

Рисунок 3.14 – Перегляд фото у повноекраному режимі

Для того аби зрозуміти як створюються оголошення створимо власне оголошення для оренди квартири. Для початку на сторінці створення оголошення виберемо категорію квартири, тип операції оренда. Вкажемо адресу і зазначемо що це пропозиція від власника, все це зображено на рисунку 3.5.

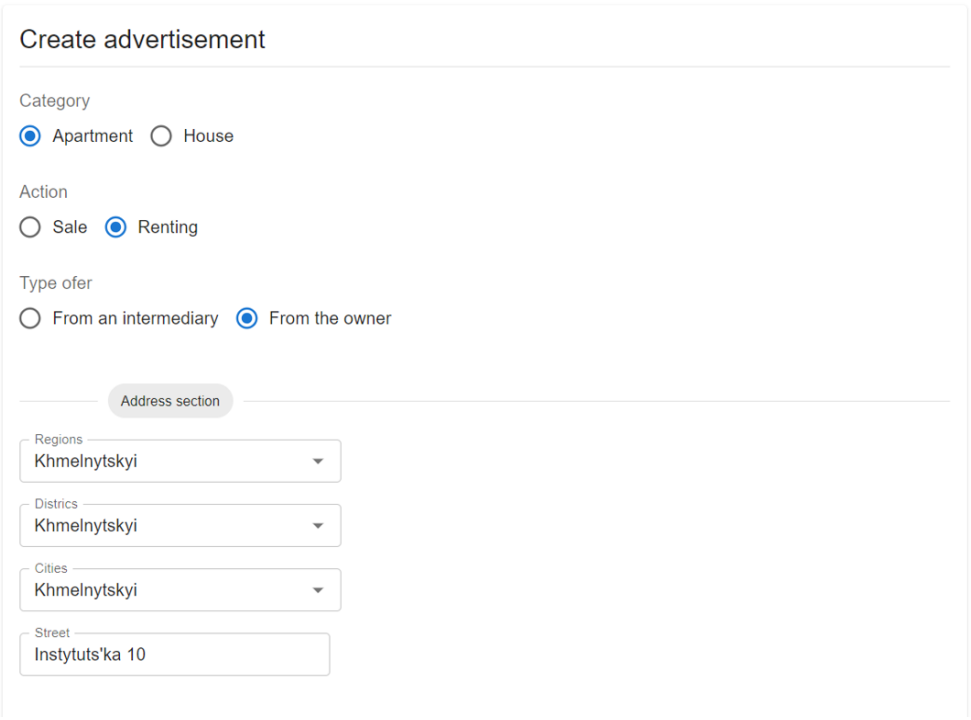


Рисунок 3.15 – Перший етап створення оголошення

Тепер завантажимо фотографій, для зручності можна просто перетягнути фото з папки, зображено на рисунку 3.6.

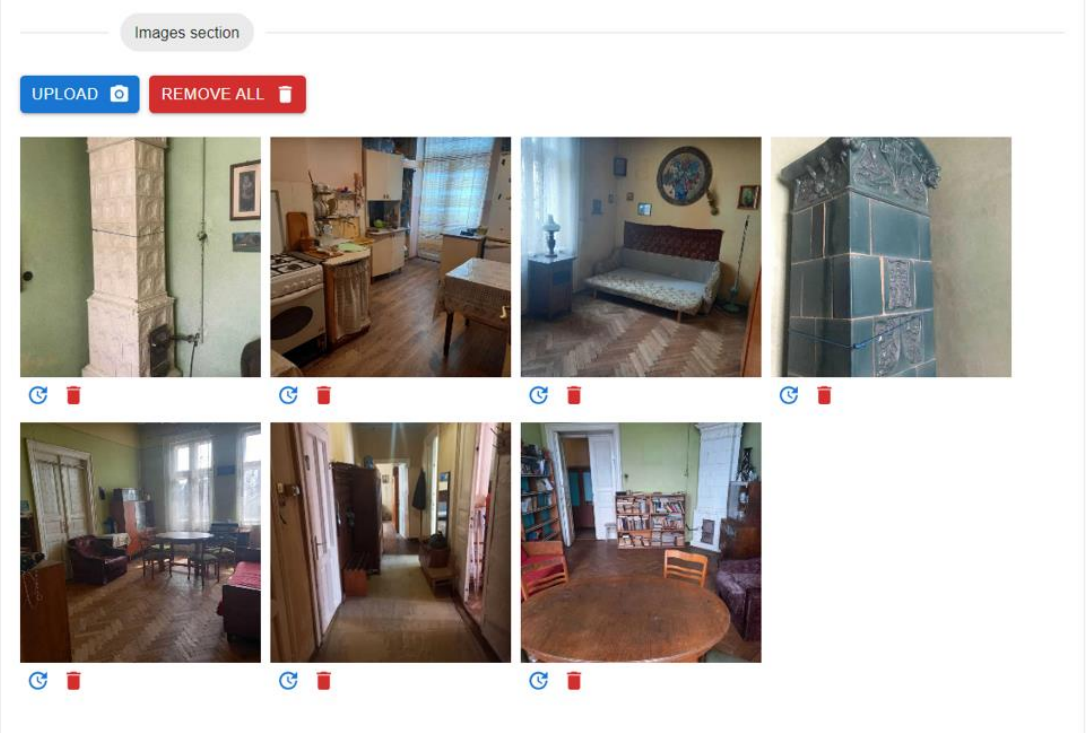


Рисунок 3.16 – Завантаження фото для оголошення

Наступним короком буде додавання загальної інформаці. Напишемо опис оголошення, вкажемо загальну, житлову та площу кухні, тип опалення, матеріал для стін, кількість кімнат та поверхів, все це зображено на рисунку 3.7.

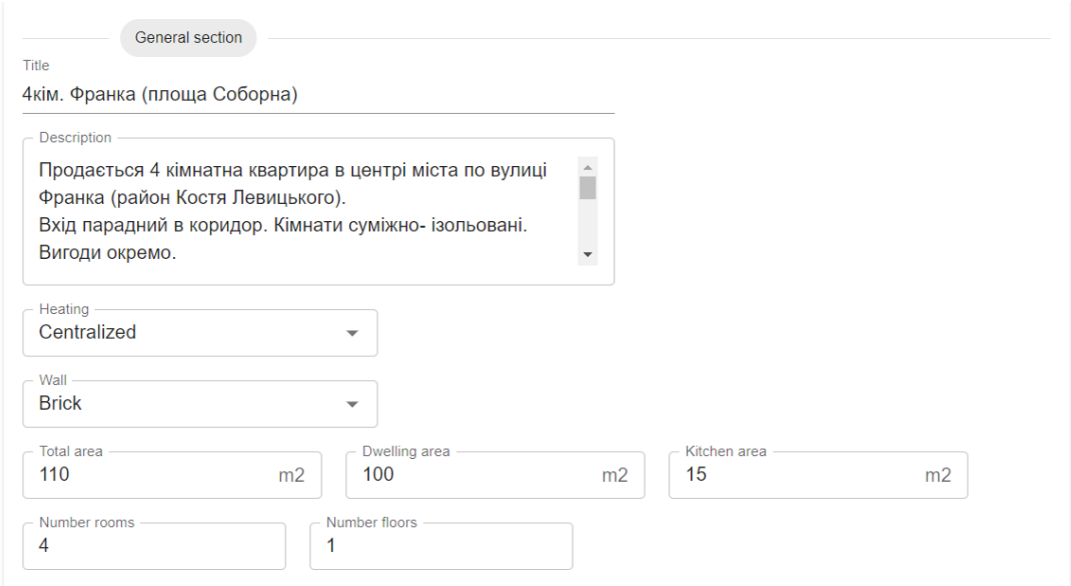


Рисунок 3.17 – Додавання загальної інформації

Тепер добавимо інформацію стосовно будинку де розташована квартира (зображено на рисунку 3.8), та інформацію про особливості оренди (зображено на рисунку 3.9).

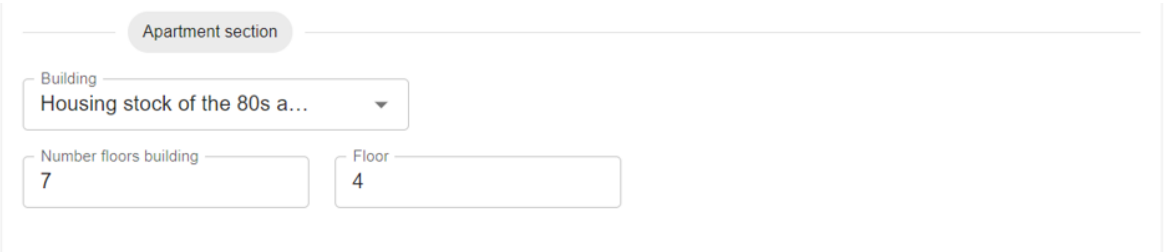


Рисунок 3.18 – Додаткова інформація про квартиру

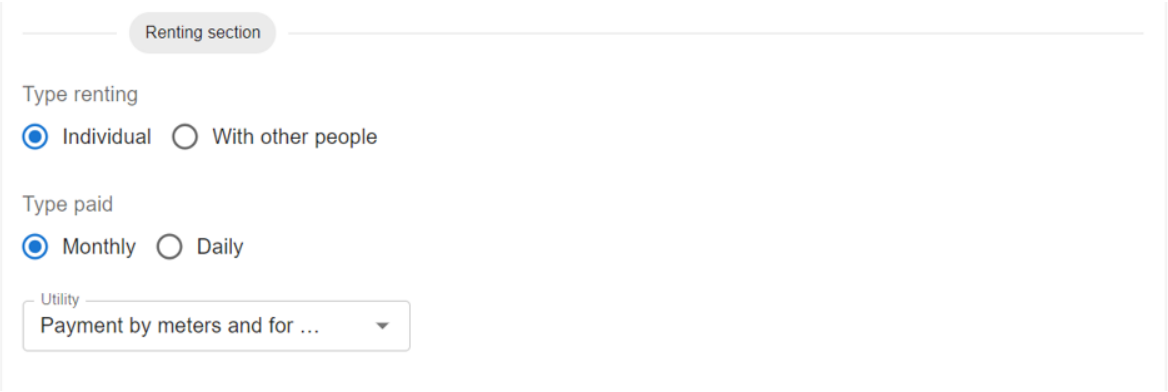


Рисунок 3.19 – Інформація про особливості оренди

Остання складова оголошення це вказання ціни (зображено на рисунку 3.10). Крім цифрового значення ціни потрібно вказати валюту, та вказати чи можливий торг.

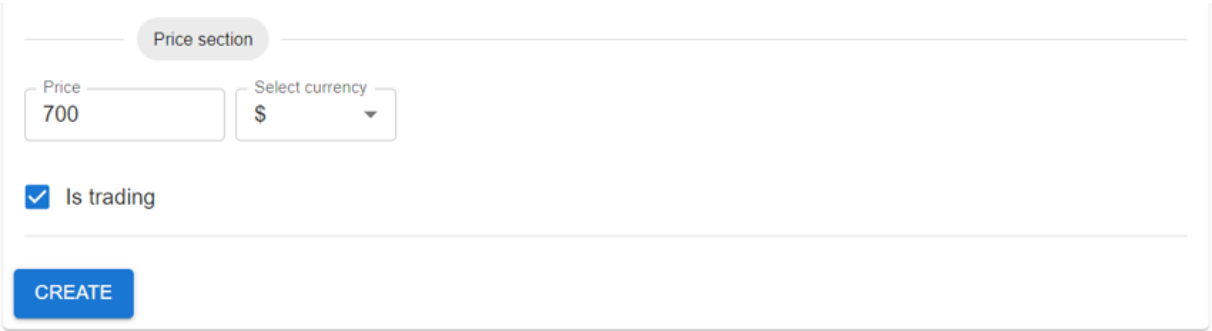


Рисунок 3.20 – Вказування ціни

3.4 Технічні характеристики веб-застосунку

Для безперебійної роботи платформи необхідно, щоб пристрій, на якому вона виконується, відповідав наступним мінімальним вимогам:

– 2 Гб внутрішньої пам’яті, з часом може знадобитися і більше;

– 1 Гб оперативної пам’яті, з часом може знадобитися і більше;

– 2-ядерний 2-ядерний процесор з тактовою частотою 2 ГГц або краще;

– доступ до мережі «Інтернет» зі швидкістю мінімум 10 Мбіт/с.

3.5 Розгортання та встановлення системи

Для того, щоб розгорнути даний проєкт, потрібно встановити на систему Node.js 14 для підняття серверу та PostgresSQL для підняття серверу бази даних. При запуску програми потрібно слідувати наступним інструкціям:

– в корневій папці проєкту визвати команду «npm install» для встановлення усіх модулів;

– використовуючи файл .env.example створити свій .env файл з всіма змінними навколишнього середовища запрошеними проєктом;

– в корневій папці проєкту визвати команду «npm run start» для запуску системи;

– в корневій папці проєкту визвати команду «npm run start:dev» для запуску системи в режимі розробки;

– в корневій папці проєкту визвати команду «npm run test» для запуску наявних тестів.

3.6 Тестування веб-застосунку

Проєкт програмного забезпечення має завершитися не лише розробкою, але й проведенням тестування. Ця процедура полягає в перевірці роботи програмного продукту та відповідності його вимогам, зазначеним у технічному завданні.

Тестування програмного додатку є важливою і необхідною частиною процесу розробки. Основна мета тестування полягає в перевірці функціональності, якості і надійності додатку перед його впровадженням. Тестування допомагає виявити помилки, дефекти і несправності в програмному додатку. Це дозволяє розробникам виправити їх до впровадження додатку в реальні умови, що зменшує ризик виникнення проблем після випуску. Дозволяє перевірити, чи відповідає програмний додаток вимогам і специфікаціям, які були поставлені перед ним. Це допомагає забезпечити, що додаток виконує очікувану функціональність і працює належним чином. Допомагає забезпечити якість програмного додатку, перевіряючи його працездатність, стабільність, продуктивність і безпеку. Це дозволяє забезпечити, що додаток буде надійним і задовольняти вимоги користувачів.

Тестування програмного забезпечення зазвичай можна розділити на дві основні категорії: функціональне тестування та нефункціональне тестування.

Функціональне тестування - це процес перевірки функціональності програмного забезпечення з метою переконатися, що воно виконує очікувані функції та відповідає вимогам. Під час функціонального тестування перевіряється правильність роботи окремих функцій, взаємодія з користувачем, обробка даних, логіка програми та інші аспекти, що стосуються функціональності програмного забезпечення. Тестування може включати сценарії, тест-кейси, ручне або автоматизоване виконання тестів з метою виявлення помилок, недоліків та невідповідностей вимогам.

Нефункціональне тестування - це процес перевірки якості програмного забезпечення, що не пов'язаний з його функціональністю. Воно спрямоване на оцінку характеристик програмного забезпечення, таких як продуктивність, надійність, безпека, сумісність, масштабованість та інші. Під час нефункціонального тестування перевіряються параметри, які визначаються вимогами до системи, стандартами та очікуваннями користувачів. Це можуть бути тести на навантаження, тести на безпеку, тести на швидкодію, тести на сумісність з різними платформами та інші види тестів, спрямовані на оцінку характеристик, які не відносяться безпосередньо до функціональності програмного забезпечення.

Нефункціональне тестування може включати наступні види тестів:

– Тестування продуктивності: перевірка швидкодії, завантаженості та масштабованості системи при різних навантаженнях.

– Тестування надійності: перевірка стійкості та відновлюваності системи після виникнення помилок, відмов та небажаних ситуацій.

– Тестування безпеки: перевірка системи на вразливості та виявлення потенційних проблем безпеки.

– Тестування сумісності: перевірка взаємодії програмного продукту з різними операційними системами, пристроями або іншими компонентами.

– Тестування доступності: перевірка, чи відповідає програмний продукт вимогам щодо доступності для користувачів з обмеженими можливостями.

– Тестування використання ресурсів: перевірка споживання пам'яті, процесорного часу та інших ресурсів системою під час її роботи.

– Тестування відмовостійкості: перевірка поведінки системи під час відмов окремих компонентів або невдалої роботи.

Здійснимо перевірку функціональності веб-додатка. Основною метою функціонального тестування є перевірка відповідності програми її специфікаціям. Тестові сценарії показано у таблиці 3.1.

Таблиця 3.1 - Тестовий випадок №1.

Тестовий випадок призначений для перевірки коректності програми для повного циклу роботи адміністратора.			
Steps (Кроки/дії)		Expected Results (Очікувані результати)	Actual results (Реальні результати):
1.	Відкрити веб-сайт	Повинна з’явитися головна сторінка.	З’явилася головна сторінка з об’єктами нерухомості.
2.	Натиснути кнопку «Login»	Повинна з’явитися сторінка логіну.	З’явилася сторінка з формою для логіну.
3.	Авторизація	Після введення даних та їх відправки повинен відбутись перехід на особистий кабінет адміністратора	Відбувається перехід на особистий кабінет адміністратора зі списком категорії та інших складових системи.
4.	Додавання категорії	Повинна з’явитися форма додавання категорії.	З’явилась форма додавання категоріх На основі введених даних, записується нова катеорія, також нові дані відображаються для адміністратора.
5.	Вихід з облікового запису	Повинне відбутися перенаправлення адміністратора на головну сторінку додатку.	Адміністратор вийшов з особистого кабінету і немає змоги редагувати дані веб-застосунку.

Таблиця 3.2 - Тестовий випадок №2.

Тестовий випадок призначений для перевірки коректності програми для повного циклу роботи звичайного користувача.			
Steps (Кроки/дії)		Expected Results (Очікувані результати)	Actual results (Реальні результати):
1.	Відкрити веб-сайт	Повинна з’явитися головна сторінка.	З’явилася головна сторінка з об’єктами нерухомості.
2.	Вибрати у фільтрі категорію квартири для оренди	Повинні з’явитися оголошення квартири які можна орендувати.	З’явилися оголошення квартир які можна орендувати.
3.	Вибрати у фільтрі категорію продаж квартир	Повинні з’явитися оголошення стосовно продажу квартир.	З’явилися оголошення стосовно продажу квартир.
4.	Перегляд оголошення	Після натискання на оголошення повинен здійснитися перехід на сторінку оголошення.	З’явилася сторінка оголошення
5.	Реєстрація.	Після введення даних та натиснення кнопки «Register» користувач повинен бути зареєстрований у системі.	Користувач успішно зареєструвався і його перенаправлено на сторінку логіну.

Кінець таблиці 3.2

Тестовий випадок призначений для перевірки коректності програми для повного циклу роботи звичайного користувача.			
Steps (Кроки/дії)		Expected Results (Очікувані результати)	Actual results (Реальні результати):
6.	Логін	Після введення даних і натиснення кнопки «Login» повинне відбутися перенаправлення на сторінку особистого кабінету.	Відбувається перехід на сторінку особистого кабінету користувача де він має змогу сворювати та контролювати свої оголошення.
7.	Вихід з облікового запису	Повинне відбутися перенаправлення користувача на головну сторінку додатку.	Користувач вийшов з особистого кабінету і немає змоги керувати своєми оголошеннями.

Модульне тестування – це вид функціонального тестування, при якому перевіряється на працездатність необхідних модулів додатку. Модуль – це найменша частина програмного забезпечення, яка може бути протестована. Для даного програмного забезпечення необхідно перевірити усі методи дій та дані які вони повертають.

Для модульного тестування використаємо бібліотеку Jest. Наступний фрагмент коду реалізує тестування запуску сервера, перевіряючи чи повертається відповідь на запит:

```
describe('AppController', () => {
  let appController: AppController;
```

```
  beforeEach(async () => {
    const app: TestingModule = await Test.createTestingModule({
      controllers: [AppController],
      providers: [AppService],
    }).compile();

    appController = app.get<AppController>(AppController);
  });

  describe('root', () => {
    it('should return "Hello World!"', () => {
      expect(appController.getHello()).toBe('Hello World!');
    });
  });
});
```

Результат виконання тесту представлено на рисунку 3.21:

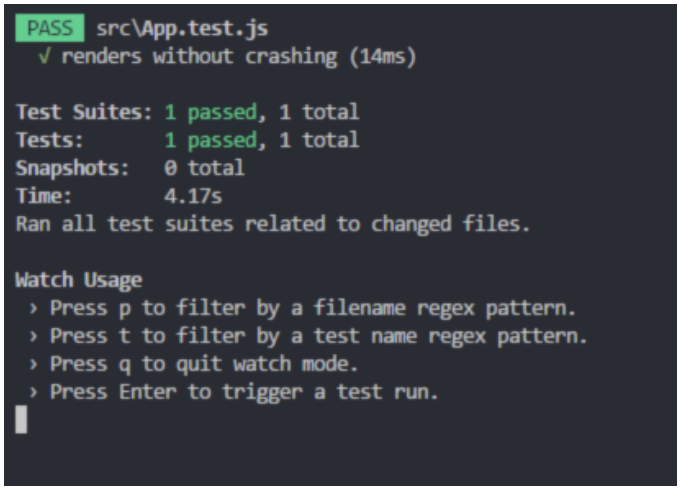


Рисунок 3.21 - Результат тестування сервера

Висновки до розділу

У даному розділі було описано процес створення бази даних і її підключення до веб-застосунку. Також була проведена декомпозиція системи на модулі та розглянуто детально кожен з них. Встановлені технічні характеристики для веб-застосунку, а також розглянутий алгоритм для інсталяції та експлуатації розробленого програмного продукту. Проведено тестування в

резулітаті якого можна підтвердити, що програмний продукт успішно пройшов тестування на 100% і повністю відповідає очікуваним вимогам, які були викладені у технічному завданні. Завершено розробку проєкту і він готовий до використання в реальних умовах.

ВИСНОВКИ

Під час виконання кваліфікаційної роботи було проведено дослідження предметної області, обґрунтовано актуальність розробки даного проєкту та виявлено потенційні проблеми, з якими зіштовхуються люди, які не використовують веб-застосунки у сфері нерухомості. Також були проаналізовані існуючі рішення у цій сфері. На основі аналізу схожих програмних засобів був складений список їх основних переваг та недоліків, а також визначено функціональні і нефункціональні вимоги до майбутнього програмного продукту. З урахуванням цих вимог було сформульовано технічне завдання та варіанти використання додатку.

Після цього було проведено проектування структури бази даних, включаючи визначення необхідних полів, таблиць та зв'язків між ними. Далі, з урахуванням проектованої архітектури програмного продукту та бази даних, було прийнято рішення щодо вибору технологій та інструментів для розробки системи, які найкраще відповідають даному типу архітектури. Конкретно, було вибрано мову програмування TypeScript, фреймворки NextJS та NestJS, а також реляційну базу даних PostgreSQL. Головним фактором при цьому виборі була швидкість роботи, яку надають ці обрані технології.

Далі, було проведено декомпозицію програмної системи на модулі та здійснено проектування цих модулів, включаючи визначення способів їх взаємодії. Крім того, на цьому етапі було розроблено інтерфейс для веб-додатку. За допомогою спроектованої архітектури та інтерфейсу було створено базу даних, а також серверну та клієнтську частини системи, а також механізми для їх взаємодії.

Останнім етапом було проведення тестування, під час якого були виявлені та усунуті будь-які несправності, які стосуються функціональності та зовнішнього вигляду. Для цього були виконані модульні та функціональні тести, а результати тестових випробувань були задокументовані у звіті.

Проведені тести підтвердили, що система виконує всі необхідні функції, зазначені у технічному завданні, та має зручний та мінімалістичний інтерфейс, необтяжений зайвими елементами.

Отже, поставлене завдання в рамках кваліфікаційної роботи було успішно виконано. Розроблена програмна система для замовлення турів та подорожей працює належним чином і повністю функціональна. Цей проєкт не лише дозволив отримати практичні навички, але й поглибити теоретичні знання у всіх аспектах життєвого циклу розробки програмного забезпечення.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Дипломний проєкт: методичні вказівки щодо його виконання для студентів спеціальності 121 «Інженерія програмного забезпечення» / Онишко О. Г., Л. П., Радельчук Г. І., Форкун Ю. В., Яшина О. М. Хмельницький : ХНУ, 2020. 77с .
2. Next.js – URL: https://nextjs.org/ . (дата звернення – 16.04.2023).
3. Next.js Routing – URL: https://nextjs.org/docs/app/building-your-application/routing/ . (дата звернення – 16.04.2023).
4. Next.js Rendering – URL: https://nextjs.org/docs/app/building-your-application/rendering/ . (дата звернення – 16.04.2023).
5. Next.js Styling – URL: https://nextjs.org/docs/app/building-your-application/styling/ . (дата звернення – 18.04.2023).
6. Next.js Optimizing – URL: https://nextjs.org/docs/app/building-your-application/optimizing/ . (дата звернення – 18.04.2023).
7. Next.js Data Fetching – URL: https://nextjs.org/docs/app/building-your-application/data-fetching/ . (дата звернення – 18.04.2023).
8. Next.js Configuring – URL: https://nextjs.org/docs/app/building-your-application/configuring/ . (дата звернення – 19.04.2023).
9. Nest.js – URL: https://docs.nestjs.com/. (дата звернення – 22.04.2023).
10. Nest.js Controllers – URL: https://docs.nestjs.com/controllers/ . (дата звернення – 22.04.2023).
11. Nest.js Providers – URL: https://docs.nestjs.com/providers/ . (дата звернення – 22.04.2023).
12. Nest.js Modules – URL: https://docs.nestjs.com/modules/ . (дата звернення – 25.04.2023).
13. Nest.js Middleware – URL: https://docs.nestjs.com/middleware/ . (дата звернення – 25.04.2023).

14. Nest.js Exception Filters – URL: https://docs.nestjs.com/exception-filters/ . (дата звернення – 22.04.2023).

15. Nest.js Pipes – URL: https://docs.nestjs.com/pipes/ . (дата звернення – 25.04.2023).

16. Nest.js Guards – URL: https://docs.nestjs.com/guards/ . (дата звернення – 02.05.2023).

17. Nest.js Interceptors – URL: https://docs.nestjs.com/interceptors/ . (дата звернення – 02.05.2023).

18. Nest.js Custom Decorators – URL: https://docs.nestjs.com/custom-decorators/ . (дата звернення – 02.05.2023).

19. Nest.js Configuration – URL: https://docs.nestjs.com/guards/techniques/configuration/ . (дата звернення – 03.05.2023).

20. Nest.js Database – URL: https://docs.nestjs.com/techniques/database/ . (дата звернення – 03.05.2023).

21. Nest.js Validation – URL: https://docs.nestjs.com/techniques/validation/ . (дата звернення – 03.05.2023).

22. Nest.js Cashing – URL: https://docs.nestjs.com/techniques/cashing/ . (дата звернення – 05.05.2023).

23. Postgresql Tutorial – URL: https://www.postgresqltutorial.com/ . (дата звернення – 05.05.2023).

24. Install PostgreSQL on Windows – URL: https://www.postgresqltutorial.com/postgresql-getting-started/install-postgresql/ . (дата звернення – 05.05.2023).

25. Connect To a PostgreSQL Database Server – URL: https://www.postgresqltutorial.com/postgresql-getting-started/connect-to-postgresql-database/ . (дата звернення – 05.05.2023).

26. PgAdmin4 – URL: https://www.pgadmin.org// . (дата звернення – 07.05.2023).

27. PgAdmin4 Docs – URL: https://www.pgadmin.org/docs/ . (дата звернення – 07.05.2023).

28. SQL Tutorial – URL: https://www.w3schools.com/sql/default.asp/ . (дата звернення – 10.05.2023).

29. The SQL SELECT Statement – URL: https://www.w3schools.com/sql/sql_select.asp/ . (дата звернення – 10.05.2023).

30. SQL WHERE Clause – URL: https://www.w3schools.com/sql/sql_where.asp/ . (дата звернення – 10.05.2023).

31. SQL Joins Clause – URL: https://www.w3schools.com/sql/sql_join.asp/ . (дата звернення – 10.05.2023).

32. SQL GROUP BY Statement – URL: https://www.w3schools.com/sql/sql_groupby.asp/ . (дата звернення – 11.05.2023).

33. SQL INSERT INTO SELECT Statement – URL: https://www.w3schools.com/sql/sql_insert_into_select.asp/ . (дата звернення – 11.05.2023).

34. Swiper– URL: https://swiperjs.com/ . (дата звернення – 12.05.2023).

35. Swiper get started – URL: https://swiperjs.com/get-started/ . (дата звернення – 12.05.2023).

36. Prisma ORM – URL: https://www.prisma.io/ . (дата звернення – 12.05.2023).

37. Prisma ORM Learn – URL: https://www.prisma.io/learn/ . (дата звернення – 13.05.2023).

38. Building a REST API with NestJS and Prisma – URL: https://www.prisma.io/blog/nestjs-prisma-rest-api-7D056s1BmOL0/ . (дата звернення – 14.05.2023).

39. Building a REST API with NestJS and Prisma: Input Validation & Transformation – URL: https://www.prisma.io/blog/nestjs-prisma-validation-7D056s1kOla1 . (дата звернення – 14.05.2023).

40. Building a REST API with NestJS and Prisma: Error Handling – URL: https://www.prisma.io/blog/nestjs-prisma-error-handling-7D056s1kOop2/ . (дата звернення – 15.05.2023).

ДОДАТОК А
(обов’язковий)

ТЕХНІЧНЕ ЗАВДАННЯ

Введення

Робота виконується в рамках проєкту розробки веб-застосунку для автоматизації пошуку об’єктів нерухомості. Технічне завдання розроблено у відповідності до стандарту ГОСТ 19.201–78.

1 Підстава для розробки

Підставою для розробки є «Завдання на кваліфікаційну роботу», затверджене завідувачем кафедри інженерії програмного забезпечення. Найменування розробки: Веб-застосунок для автоматизації пошуку об’єктів нерухомості.

2 Призначення розробки

2.1 Функціональне призначення

Функціональним призначенням застосунку є клієнтська частина для перегляду об’єктів нерухомості, та серверна частина з можливістю редагування вмісту сайту.

2.2 Експлуатаційне призначення

Програма повинна експлуатуватися на будь-яких пристроях, на яких є браузер. Кінцевим користувачем застосунку може виступати будь-яка особа.

3 Вимоги до програми

3.1 Вимоги до функціональних характеристик

Для звичайного користувача:

- переглядати об’єкти нерухомості;
- використовувати фільтри для пошуку;
- створення власних оголошень;
- можливість керувати власними оголошеннями;
- реєстрація та авторизація;

Для адміністратора:

- авторизація;
- створення нових категорій;

- створення та редагування будівель для оренди;
- можливість додавання та редагування даних про валюти;
- можливість додавання та редагуання даних про особливості нерухомості;
- можливість проводити модерацію оголошень;
- можливість блокувати користувача за шахрайство;

3.2 Вимоги до надійності

Веб-застосунок повинен забезпечувати такі вимоги до надійності:

- можливість самостійно відновлюватись у разі збою;
- можливість резервного копіювання бази даних.

3.3 Умови експлуатації та вимоги до технічних засобів

Веб-застосунок повинен працювати на всіх пристроях, які мають браузер та стабільний доступ до мережі «Інтернет»: смартфонах, планшетах та комп’ютерах.

Для роботи платформи без збоїв, потрібно, щоб пристрій, на якому вона запускається задовольняв наступні мінімальні вимоги:

- 2 Гб внутрішньої пам’яті;
- 1 Гб оперативної пам’яті;
- 2-ядерний процесор;
- доступ до мережі «Інтернет» зі швидкістю мінімум 10 Мбіт/с.

3.4 Вимоги до інформаційної та програмної сумісності

Для розробки веб-застосунку був використаний фронтенд TypeScript фреймворк NextJS, бекенд фреймворк NestJS, та база даних PostgreSQL.

3.5 Спеціальні вимоги

Програма повинна мати зручний та зрозумілий зовнішній інтерфейс користувача.

4 Вимоги до програмної документації

У момент здачі проєкту замовнику надається наступний набір документів:

- текст програми;

- опис програми;
- технічне завдання;
- керівництво користувача.

5 Стадії та етапи розробки

Стадії та етапи розробки веб-застосунку для автоматизації пошуку нерухомості.

Таблиця А.1 – Стадії та етапи розробки проєкту

Стадія розробки	Етапи робіт	Зміст робіт
1	2	3
Технічне завдання 02.01.23 – 31.01.23	Обґрунтування необхідності розробки програми	Коротка характеристика програмного забезпечення; підстава і призначення розробки; вимоги до програмної системи і документація; стадії і етапи розробки програми; порядок контролю і приймання
Ескізний проєкт 01.02.23 – 26.02.23	Розробка ескізного проєкту	Попередня розробка структури вхідних і вихідних даних; уточнення середовища програмування; розробка і опис загальної алгоритмічної структури
Технічний проєкт 29.02.23 – 19.03.23	Розробка технічного проєкту	Уточнення структури вхідних і вихідних даних; розробка докладного алгоритму; розробка структури програми
Робочий проєкт 20.03.23 – 15.04.23	Розробка програмного забезпечення	Реалізація програмного забезпечення; відлагодження; проведення попереднього тестування
Розробка програмної документації 16.04.23 – 22.04.23	Розробка документації до програмного забезпечення	Розробка необхідної документації, передбаченої технічним завданням
Тестування системи 23.04.23 – 30.04.23	Проведення тестування програмного забезпечення	Розробка методики тестування; проведення основних тестів; коректування програмного забезпечення
Впровадження	Підготовка і передача програми	Підготовка і розгортання програмного забезпечення
Розробка	Розробка	Розробка необхідної документації,

програмної документації 16.04.23 – 22.04.23	документації до програмного забезпечення	передбаченої технічним завданням

6 Порядок контролю та приймання

Контроль здійснюється кінцевими користувачами системи, підключеними на етапі тестування застосунку.

ДОДАТОК Б
(обов’язковий)

ДІАГРАМИ

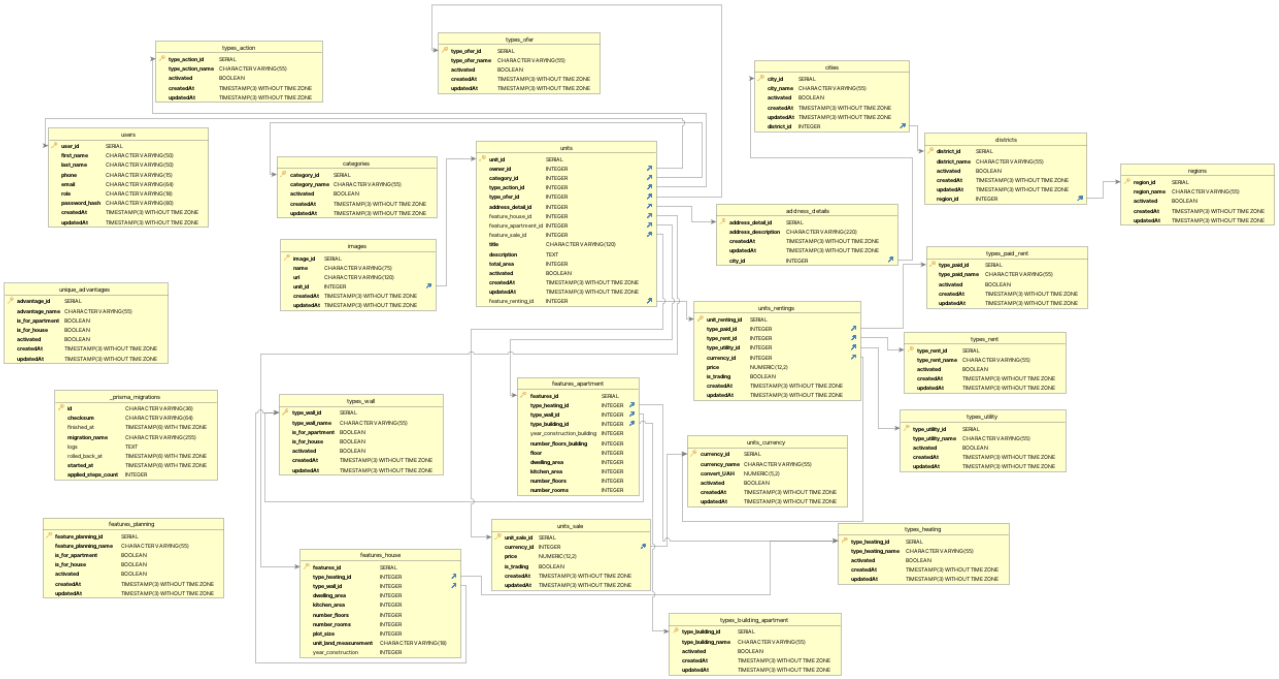


Рисунок Б.1 – Схема бази даних

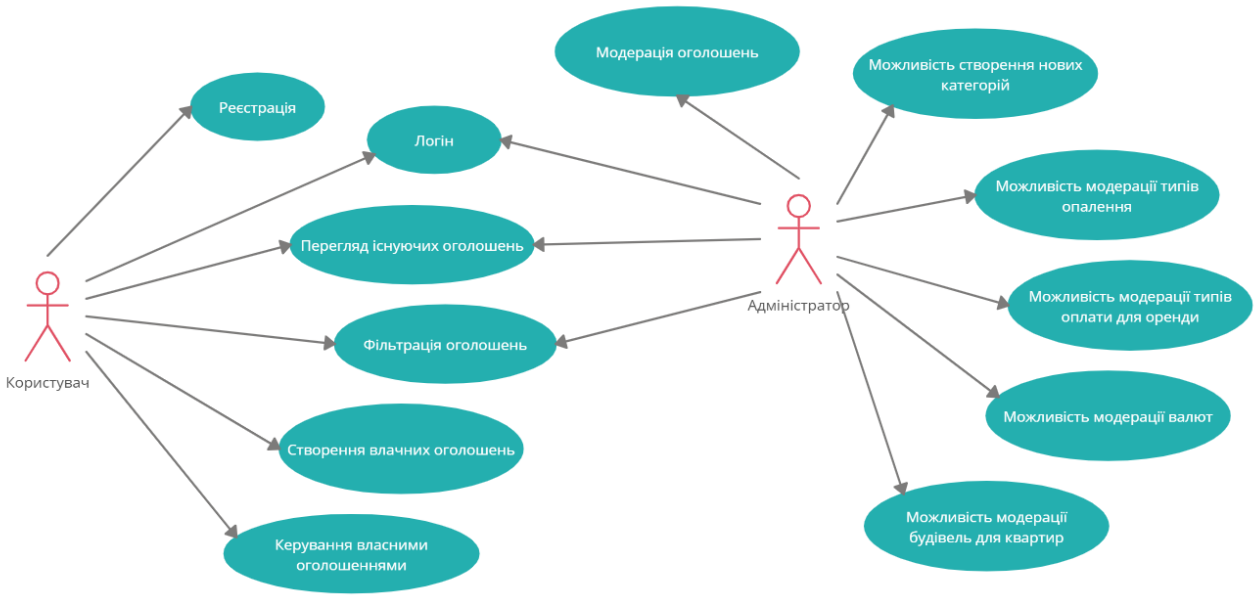


Рисунок Б.2 – Діаграма варіантів використання

ДОДАТОК В
(обов’язковий)

КОД (ЛІСТИНГ) ПРОГРАМИ

index.tsx

```
import {
  Autocomplete,
  Button,
  Card,
  CardActions,
  CardContent,
  CardMedia,
  Divider,
  Pagination,
  TextField,
  ToggleButton,
  ToggleButtonGroup,
  Typography,
} from "@mui/material";
import ReorderIcon from "@mui/icons-material/Reorder";
import ViewModuleIcon from "@mui/icons-material/ViewModule";
import FilterForm from "@/components/filter-form";
import { useCallback, useEffect, useState } from "react";

interface FilterDataState {
  category_id: number | null;
  action_id: number | null;
  region_id: number | null;
  district_id: number | null;
  city_id: number | null;
  total_area_from: number | null;
  total_area_to: number | null;
  price_from: number | null;
  price_to: number | null;
  currency_id: number | null;
}

export default function Home() {
  const initialFilter = {
    category_id: 1,
    action_id: 2,
    region_id: 1,
    district_id: null,
    city_id: null,
    total_area_from: 20,
    total_area_to: 120,
    price_from: 100,
    price_to: 1000,
    currency_id: 2,
  };

  const limitOptions = ["10", "20", "30"];
  const sortOptions = ["highest price", "lowest price"];

  const [filter, setFilter] = useState<FilterDataState>(initialFilter);

  function getFilterQueryString(filter: any): string {
    const copyFilter: FilterDataState = JSON.parse(JSON.stringify(filter));

    const filterFormatForServer: any = {
      category: copyFilter.category_id,
      action: copyFilter.action_id,
      region: copyFilter.region_id,
      district: copyFilter.district_id,
      city: copyFilter.city_id,
      totalAreaFrom: copyFilter.total_area_from,
      totalAreaTo: copyFilter.total_area_to,
      priceFrom: copyFilter.price_from,
      priceTo: copyFilter.price_to,
      currency: copyFilter.currency_id,
```

```
  };

  let query = "";

  const entries = Object.entries(Object.entries(filterFormatForServer));

  for (const [index, [key, value]] of entries) {
    if (!value) {
      delete filterFormatForServer[key];
    } else {
      if (Array.from(entries).length - 1 == +index) {
        query += `${key}=${value}`;
      } else {
        query += `${key}=${value}&`;
      }
    }
  }

  return query;
}

const changeFilter = useCallback((filterdata: any) => {
  setFilter(filterdata);
}, []);

const initialView = "table";
const [typeView, setVariantView] = useState<string>(initialView);

const [typeSort, setSort] = useState<string | null>(null);

const initialPage = 1;
const [page, setPage] = useState<number>(1);

const initialLimit = "10";
const [limit, setLimit] = useState<string>(initialLimit);

useEffect(() => {
  const filterQueryForServer = getFilterQueryString(filter);
  const url = `http://localhost:4000/units?${filterQueryForServer}&page=${page}&limit=${limit}`;
  const data = fetch(url);
}, [filter, limit]);

return (
  <main className="flex min-h-screen flex-col justify-between py-24 px-44">
    <div>
      <FilterForm onFilter={changeFilter}></FilterForm>
      <Divider className="mt-10" />
      <div className="mt-5 flex flex-wrap items-end justify-between">
        <Typography
          className="grow mr-5 mb-0"
          gutterBottom
          variant="h6"
          component="div"
        >
          Count result: 7
        </Typography>
        <div className="mr-5">
          <Autocomplete
            disablePortal
            size="small"
            id="sort-autocomplete"
            value={typeSort || null}
            options={sortOptions}
            sx={{ minWidth: 150, maxWidth: 150 }}
            renderInput={(params) => <TextField {...params} label="Sort" />}
            onChange={(event, value) => {
              setSort(value);
            }}
```

```
          />
        </div>
        <div className="mr-5">
          <Autocomplete
            disablePortal
            size="small"
            id="limit-autocomplete"
            value={limit}
            options={limitOptions}
            sx={{ minWidth: 150, maxWidth: 150 }}
            renderInput={(params) => <TextField {...params} label="Limit" />}
            onChange={(event, value) => {
              setLimit(value as string);
            }}
          />
        </div>
        <div>
          <ToggleViewAdvertisement
            initialValue={initialView}
            onToggle={(variantView) => {
              setVariantView(variantView);
            }}
          ></ToggleViewAdvertisement>
        </div>
      </div>
      <Divider className="mt-5" />
      <div className="mt-8 flex justify-between flex-wrap">
        <MediaCard></MediaCard>
        <MediaCard></MediaCard>
        <MediaCard></MediaCard>
        <MediaCard></MediaCard>
        <MediaCard></MediaCard>
        <MediaCard></MediaCard>
      </div>
      <Divider className="mt-10" />
      <div className="mt-5 flex justify-center">
        <Pagination count={10} color="primary" />
      </div>
    </div>
  </main>
 );
}

type tableOrList = "table" | "list";

type PropsToggle = {
 initialValue?: tableOrList;
 onToggle: (c: tableOrList) => void;
};

function ToggleViewAdvertisement({ initialValue, onToggle }: PropsToggle) {
 const [alignment, setAlignment] = useState<tableOrList>(
  initialValue || "table"
 );

 const handleChange = (
  event: React.MouseEvent<HTMLElement>,
  newView: tableOrList
 ) => {
  setAlignment(newView);
  onToggle(newView);
 };

 return (
  <ToggleButtonGroup
   size="small"
   color="primary"
   value={alignment}
```

```
      exclusive
      onChange={handleChange}
    >
      <ToggleButton value="list">
        <ReorderIcon></ReorderIcon>
      </ToggleButton>
      <ToggleButton value="table">
        <ViewModuleIcon></ViewModuleIcon>
      </ToggleButton>
    </ToggleButtonGroup>
  );
}

function MediaCard() {
  return (
    <Card className="mt-5" sx={{ maxWidth: 345 }}>
      <CardMedia
        sx={{ height: 240 }}
        image="/jk-placeholder-image.jpg"
        title="green iguana"
      />
      <CardContent>
        <Typography gutterBottom variant="h5" component="div">
          Lizard
        </Typography>
        <Typography variant="body2" color="text.secondary">
          Lizards are a widespread group of squamate reptiles, with over 6,000
          species, ranging across all continents except Antarctica
        </Typography>
      </CardContent>
      <CardActions>
        <Button size="small">Share</Button>
        <Button size="small">Learn More</Button>
      </CardActions>
    </Card>
  );
}
```

units.service.ts

```
import { Injectable } from '@nestjs/common';
import { CreateUnitDto } from './dto/create-unit.dto';
import { UpdateUnitDto } from './dto/update-unit.dto';
import { PrismaService } from 'src/prisma/prisma.service';
import { FilesService } from 'src/files/files.service';
import { ImagesService } from 'src/images/images.service';
import { Prisma } from '@prisma/client';

enum Categories {
  APARTMENT = 1,
  HOUSE = 2,
}

enum TypesActions {
  SALE = 1,
  RENTING = 2,
}

@Injectable()
export class UnitsService {
  constructor(
    private prisma: PrismaService,
    private readonly filesService: FilesService,
    private readonly imagesService: ImagesService,
  ) {}
```

```
async create(createUnitDto: any, files: any[]) {
  console.log('createUnitDto ', createUnitDto);

  const generalData = JSON.parse(createUnitDto.general_data);
  const unitData: any = {};

  let feature_apartment_id: number;

  if (generalData.category_id === Categories.APARTMENT) {
    const feature_apartment_payload = JSON.parse(
      createUnitDto.feature_apartment,
    );

    console.log('Feature Apartment ', feature_apartment_payload);

    const feature_apartment: any =
      await this.prisma.features_apartment.create({
        data: { ...feature_apartment_payload },
      });

    feature_apartment_id = feature_apartment.features_id;
    unitData.feature_apartment_id = feature_apartment_id;
  }

  console.log('feature_apartment_id ', feature_apartment_id);

  let feature_house_id: number;

  if (generalData.category_id === Categories.HOUSE) {
    const feature_house_payload = JSON.parse(createUnitDto.feature_house);

    console.log('Feature House ', feature_house_payload);

    const feature_house: any = await this.prisma.features_house.create({
      data: { ...feature_house_payload },
    });

    feature_house_id = feature_house.features_id;
    unitData.feature_house_id = feature_house_id;
  }

  console.log('feature_house_id ', feature_house_id);

  let unit_renting_id: number;

  if (generalData.type_action_id === TypesActions.RENTING) {
    const renting_detail_payload = JSON.parse(createUnitDto.renting_detail);

    console.log('Detail renting ', renting_detail_payload);

    const renting_detail = await this.prisma.units_rentings.create({
      data: { ...renting_detail_payload },
    });

    unit_renting_id = renting_detail.unit_renting_id;
    unitData.feature_renting_id = unit_renting_id;
  }

  console.log('unit_renting_id ', unit_renting_id);

  let unit_sale_id: number;

  if (generalData.type_action_id === TypesActions.SALE) {
    const sale_detail_payload = JSON.parse(createUnitDto.sale_detail);

    console.log('Detail renting ', sale_detail_payload);

    const renting_detail = await this.prisma.units_sale.create({
```

```
      data: { ...sale_detail_payload },
    });

    unit_sale_id = renting_detail.unit_sale_id;
    unitData.feature_sale_id = unit_sale_id;
  }

  console.log('unit_sale_id ', unit_sale_id);

  const address_detail_payload = JSON.parse(createUnitDto.address_detail);

  const address_detail = await this.prisma.address_details.create({
    data: { ...address_detail_payload },
  });

  const address_detail_id = address_detail.address_detail_id;

  console.log('address_detail_id ', address_detail_id);

  const unit: any = await this.prisma.units.create({
    data: {
      ...generalData,
      ...unitData,
      address_detail_id,
    },
  });

  let savedImages: any = await this.filesService.saveFiles(
    files,
    unit.unit_id,
  );

  console.log('unit.unit_id', unit.unit_id);

  savedImages = savedImages.map((image) => ({
    ...image,
    unit_id: unit.unit_id,
  }));

  await this.imagesService.create(savedImages);

  return unit;
}

async findAll(queryData: any) {
  const makePagination = (
    queryData: any,
  ): { limit: string; offset: string } => {
    const page = +queryData?.page || 1;
    const limit = +queryData?.limit || 10;

    return {
      limit: `LIMIT ${limit}`,
      offset: `OFFSET (${page} - 1) * ${limit}`,
    };
  };

  const makeFilter = async (
    queryData: any,
  ): Promise<{ where: string; join: string }> => {
    enum Action {
      SALE = 1,
      RENTING = 2,
    }

    let categoryAndActionPart = '';

    let addressPart = '';
```

```
let totalAreaPart = '';

let pricePart = '';

let joinDetailsPart = '';

if (queryData?.category && queryData?.action) {
  categoryAndActionPart = `u.category_id = ${queryData.category} AND u.type_action_id = ${queryData.action}`;
}

if (+queryData?.action == Action.RENTING) {
  joinDetailsPart =
    'join units_rentings ud ON ud.unit_renting_id = u.feature_renting_id';
}

if (+queryData?.action == Action.SALE) {
  joinDetailsPart =
    'join units_sale ud ON ud.unit_sale_id = u.feature_sale_id';
}

if (queryData?.region) {
  addressPart += `r.region_id = ${queryData.region}`;

  if (queryData?.district) {
    addressPart += ` AND d.district_id = ${queryData.district}`;

    if (queryData?.city) {
      addressPart += ` AND cities.city_id = ${queryData.city}`;
    }
  }
}

if (queryData?.totalAreaFrom && queryData?.totalAreaTo) {
  totalAreaPart += `u.total_area >= ${queryData?.totalAreaFrom} AND u.total_area <= ${queryData?.totalAreaTo}`;
} else if (queryData?.totalAreaFrom) {
  totalAreaPart += `u.total_area >= ${queryData?.totalAreaFrom}`;
} else {
  totalAreaPart += `u.total_area <= ${queryData?.totalAreaTo}`;
}

if (queryData?.currency) {
  const currency = await this.prisma.units_currency.findUnique({
    where: {
      currency_id: +queryData.currency,
    },
    select: {
      currency_id: true,
      currency_name: true,
      convert_UAH: true,
    },
  });

  const convert_uah = Number.parseFloat(currency.convert_UAH.toString());

  const converted_price = '(ud.price*uc."convert_UAH")::numeric(12, 2)';

  if (queryData?.priceFrom && queryData?.priceTo) {
    const priceFrom = Number.parseInt(queryData.priceFrom) * convert_uah;
    const priceTo = Number.parseInt(queryData.priceTo) * convert_uah;

    pricePart += `${converted_price} >= ${priceFrom} AND ${converted_price} <= ${priceTo}`;
  } else if (queryData?.priceFrom) {
    const priceFrom = Number.parseInt(queryData.priceFrom) * convert_uah;

    pricePart += `${converted_price} >= ${priceFrom}`;
  } else {
    const priceTo = Number.parseInt(queryData.priceTo) * convert_uah;
```

```
      pricePart += `${converted_price} <= ${priceTo}`;
    }
  }

  let where = 'WHERE ';

  if (categoryAndActionPart) {
    where += `${categoryAndActionPart}`;
  }

  if (addressPart) {
    if (where.length > 10) where += ' AND ';

    where += `${addressPart}`;
  }

  if (totalAreaPart) {
    if (where.length > 10) where += ' AND ';

    where += `${totalAreaPart}`;
  }

  if (pricePart) {
    if (where.length > 10) where += ' AND ';

    where += `${pricePart}`;
  }

  return {
    where: where,
    join: joinDetailsPart,
  };
};

const dataFilter = await makeFilter(queryData);
const dataPagination = makePagination(queryData);

const generatedFilterString = dataFilter.where;
const generatedJoinString = dataFilter.join;
const generatedLimitString = dataPagination.limit;
const generatedOffsetString = dataPagination.offset;

type MQProps = {
  isCount?: boolean;
};

function makeQuery({ isCount }: MQProps) {
  // SELECT COUNT(*) AS count

  const selectString = `
  u.unit_id,
  u.title,
  u.total_area,
  u."createdAt",
  cities.city_name,
  ud.price,
  uc."convert_UAH" AS converted_uah,
  (ud.price*uc."convert_UAH")::numeric(12, 2) AS converted_price,
  array_agg(i.url) AS images
  `;

  const countString = `COUNT(DISTINCT u.unit_id)::integer AS count`;

  const querySQL = `
  SELECT
  ${isCount ? countString : selectString}
  FROM units u
```

```
    ${generatedJoinString}
    JOIN units_currency uc ON uc.currency_id = ud.currency_id
    JOIN address_details ad ON ad.address_detail_id = u.address_detail_id
    JOIN cities ON cities.city_id = ad.city_id
    JOIN districts d ON d.district_id = cities.district_id
    JOIN regions r ON r.region_id = d.region_id
    JOIN images i ON i.unit_id = u.unit_id
    ${generatedFilterString}
    ${!isCount ? 'GROUP BY u.unit_id, ud.price, cities.city_name' : ''}${
      !isCount ? ', converted_uah' : ''
    }
    ${!isCount ? generatedLimitString : ''}
    ${!isCount ? generatedOffsetString : ''}
    `;

    return querySQL;
  }

  const querySQL = Prisma.sql([makeQuery({})]);
  const querySQLCount = Prisma.sql([makeQuery({ isCount: true })]);

  const units = await this.prisma.$queryRaw(querySQL);
  const count = await this.prisma.$queryRaw(querySQLCount);

  console.log(count[0].count);

  return {
    units: units,
    count: Number(count[0].count),
  };
}

findAllIDs() {
  return this.prisma.units.findMany({ select: { unit_id: true } });
}

async findOne(id: number) {
  const unit: any = await this.prisma.units.findUnique({
    where: {
      unit_id: id,
    },
    select: {
      title: true,
      description: true,
      total_area: true,
      owner: {
        select: {
          email: true,
        },
      },
      category: {
        select: {
          category_name: true,
        },
      },
      type_action: {
        select: {
          type_action_name: true,
        },
      },
      type_ofer: {
        select: {
          type_ofer_name: true,
        },
      },
      feature_house: {
        select: {
          type_heating: {
```

```
      select: { type_heating_name: true },
    },
    type_wall: {
      select: { type_wall_name: true },
    },
    number_floors: true,
    dwelling_area: true,
    kitchen_area: true,
    number_rooms: true,
    year_construction: true,
    unit_land_measurement: true,
    plot_size: true,
  },
},
feature_apartment: {
  select: {
    type_heating: {
      select: { type_heating_name: true },
    },
    type_wall: {
      select: { type_wall_name: true },
    },
    type_building: {
      select: { type_building_name: true },
    },
    number_floors_building: true,
    number_floors: true,
    floor: true,
    dwelling_area: true,
    kitchen_area: true,
    number_rooms: true,
    year_construction_building: true,
  },
},
feature_sale: {
  select: {
    currency: {
      select: {
        currency_name: true,
      },
    },
    price: true,
    is_trading: true,
  },
},
feature_renting: {
  select: {
    type_paid: {
      select: { type_paid_name: true },
    },
    type_rent: {
      select: { type_rent_name: true },
    },
    type_utility: {
      select: { type_utility_name: true },
    },
    currency: {
      select: {
        currency_name: true,
      },
    },
    price: true,
    is_trading: true,
  },
},
address_detail: {
  select: {
    address_description: true,
```

```
            city: {
              select: {
                city_name: true,
                district: {
                  select: {
                    district_name: true,
                    region: {
                      select: {
                        region_name: true,
                      },
                    },
                  },
                },
              },
            },
          },
        },
        images: {
          select: {
            url: true,
          },
        },
      },
    });

    console.log('unit ', unit);

    return unit;
    // return `This action returns a #${id} unit`;
  }

  async update(id: number, updateUnitDto: any) {
    const unit = await this.prisma.units.update({
      where: { unit_id: id },
      data: {
        feature_renting: {
          update: {
            price: 17000.0,
          },
        },
      },
    });

    return unit;
    // return `This action updates a #${id} unit`;
  }

  remove(id: number) {
    return `This action removes a #${id} unit`;
  }
}
```

ДОДАТОК Г
(обов’язковий)

ПРЕЗЕНТАЦІЙНІ МАТЕРІАЛИ

Хмельницький Національний університет
Факультет інформаційних технологій
Кафедра інженерії програмного забезпечення

Кваліфікаційна робота на тему: “Веб-застосунок для автоматизації пошуку об’єктів нерухомості”

Виконав студент Шевчук Микола Ігорович
Керівник: Онишко О. Г. канд. пед. наук, доцент

Актуальність теми

На сьогоднішній день найти хорошу нерухомість це не просто. Пошук займає багато часу, а агенства з нерухомості неможуть запропонувати саме те, чого ви бажаєте, або і зовсім намагаються вас підштовхнути до пропозиції яка вам недовподоби.

Люди, які шукають житло, можуть бути у відчаї та стають легкою мішенню для шахраїв. Одна з найпоширеніших афер на сайтах нерухомості - вимагання застави без огляду квартири. Часто шахрай терміново пояснює, що квартиру вже переглядали інші, і єдиний спосіб убезпечити її зараз — це передати заставу. Оскільки отримати хорошу квартиру часто дуже важко, особливо у великих містах, деякі люди можуть потрапити на це й надіслати гроші. Шахрай забирає заставу, а потенційний орендар не чує від нього жодного слова. Швидше за все, квартири ніколи не існувало або вона не належала шахраю.
В інших випадках ви побачите, як люди публікують навіть нереальні об’єкти, щоб зібрати контактну інформацію потенційних орендарів/покупців, аби потім цю інформацію продати агентствам нерухомості. Більш сіра версія цієї практики полягає в тому, що агентства публікують першокласну нерухомість за чудовою ціною, яка завжди продається перед тим, як потенційний орендар/покупець зв’яжеться з нею. Таким чином агентство збирає інформацію про зацікавлених осіб, і вони можуть представити подібні об’єкти за трохи вищою ціною.

Деякі проблеми, з якими стикається категорія нерухомості, є більш серйозними, ніж інші, але вони мають одну спільну рису. Усі вони негативно впливають на досвід користувача.
Тому було прийнято створити сервіс який дозволить публікувати оголошення про нерухомість, а також автоматично підбирати для клієнта об’єкти з вказаними ним параметрами.

Мета та завдання проекту

Мета проекту - розробка веб-застосунку, який забезпечує покращення знаходження обєктів нерухомості, та який забезпечує користувачам розмізщувати власні оголошення.

Для досягнення поставленої мети необхідно виконати наступні завдання:

- дослідити предметну область та виявити потреби потенційних користувачів веб-застосунку;
- провести аналіз існуючих рішень;
- розробити технічне завдання;
- розробити архітектуру веб-застосунку та бази даних;
- обрати технології для розробки;
- розробити зручний та привітний інтерфейс для користувачів;
- розробити веб-застосунок;
- протестувати готовий веб-застосунок.

Порівняння наявного ПЗ

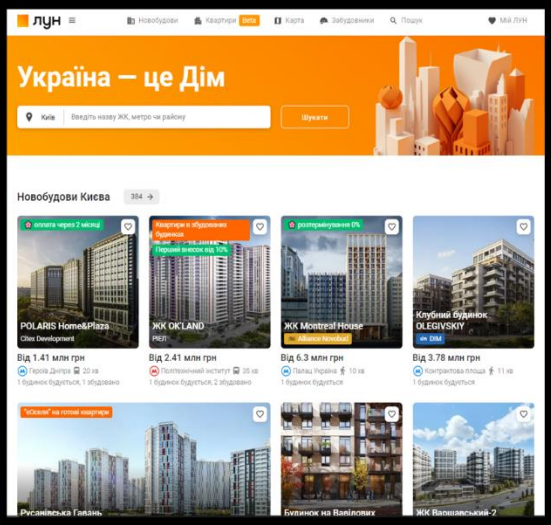
Існує багато відомих сервісів в яких розміщено багато оголошень. Ці сервіси перевірені часом та людьми і мають великий попит. Найбільш відомі: ЛУН, Dom Ria, Zillow.com.

ЛУН позиціонує себе як найбільший сайт з пошуку квартир в Україні. По факту даний сервіс являється агрегатором, який збирає інформацію з багатьох українських сервісів та подає у потрібному клієнтові форматі.

Dom Ria – яскравий приклад сервісів по збору інформації шляхом введення її користувачами. Даний сервіс має вузьку направленість, а отже збирає дані лише про житлову нерухомість. Серед переваг слід відзначити можливість викликати інспектора по нерухомості, який може допомогти вам оцінити ваше житло і виставити ціну, яка відповідає ринку.

Zillow - це провідний іноземний сервіс пошуку нерухомості, присвячений наданню споживачам даних та знань про місце де знаходиться будинок, та з'єднанні їх з кращими місцевими професіоналами, які можуть допомогти наприклад у його оздоблені, садівництві чи ремонті.

Порівняння наявного ПЗ



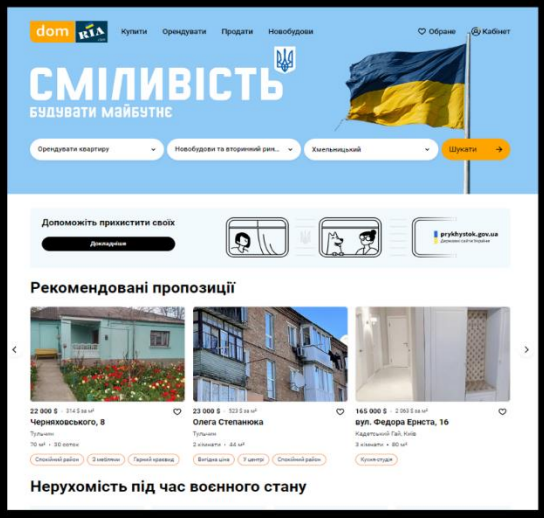
Основні переваги ЛУН:

- генерація даних за допомогою зовнішніх сервісів;
- кількість фільтрів для даних;
- актуальність та підтримка;
- алгоритми статистики.

Основні недоліки ЛУН:

- відсутня можливість створення власних об’яв;
- погана універсальність даних;
- вузька направленість.

Порівняння наявного ПЗ



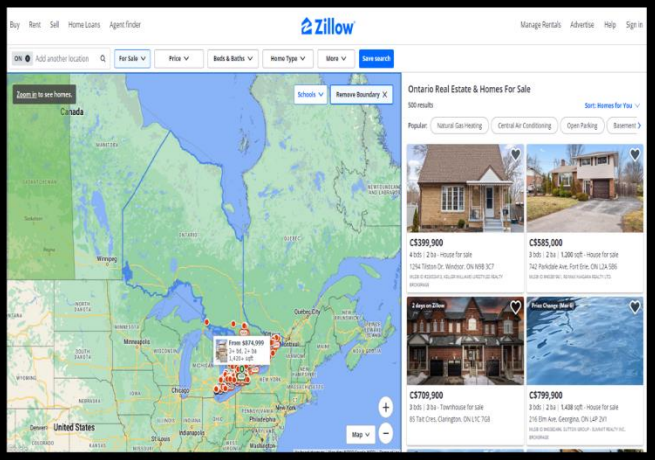
Основні переваги Dom Ria:

- генерація даних користувачами;
- бізнес модель компанії.

Основні недоліки Dom Ria:

- погана модель даних;
- маленька кількість фільтрів для пошуку;
- вузька направленість

Порівняння наявного ПЗ



Основні переваги Zillow:

- використання стандарту RESO;
- велика кількість фільтрів для пошуку;
- можливість пошуку робітників;
- можливість створення об’яв власноруч.

Основні недоліки Zillow:

- вузька направленість з даними про нерухомість.

Функціональні вимоги

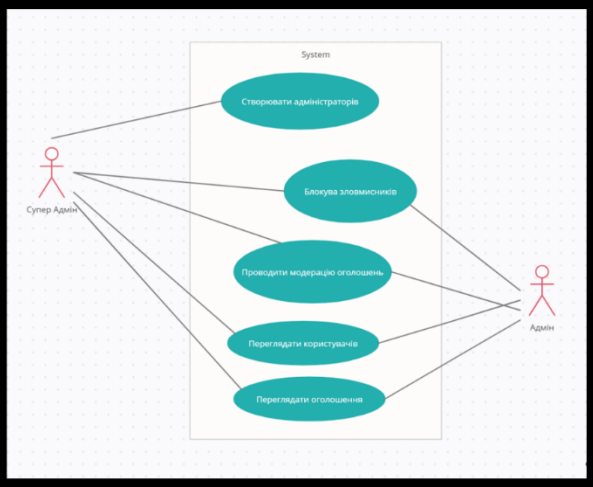
Для звичайного користувача:

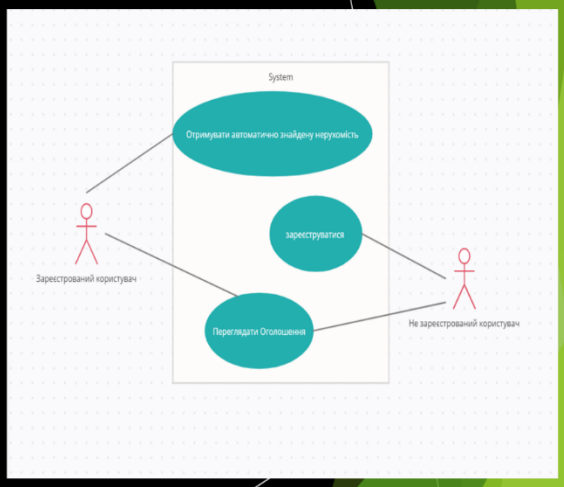
- Перегляд оголошень
- Авторизація
- Створенення власного оголошення
- Контролювати власні оглошення

Для адміністратора:

- Авторизація
- Створення ролей адміністраторів з відповідними доступами до редагування вмісту сайту
- Проводити модерацію оглошення
- Створювати або редагувати категорії
- Створювати або редагувати типи стін
- Створювати або редагувати типи опалення
- Створювати або редагувати типи операцій
- Створювати або редагувати типи оренди
- Створювати або редагувати типи платні
- Створювати або редагувати типи комунальних
- Створювати або редагувати типи будинків для квартир
- Блокувати або видаляти користувачів

Діаграма варіантів використання





Використані технології

Архітектура рівня доступа до даних

Клієнт серверна архітектура

Архітектура рівня представлення

MVC - Model View Controller

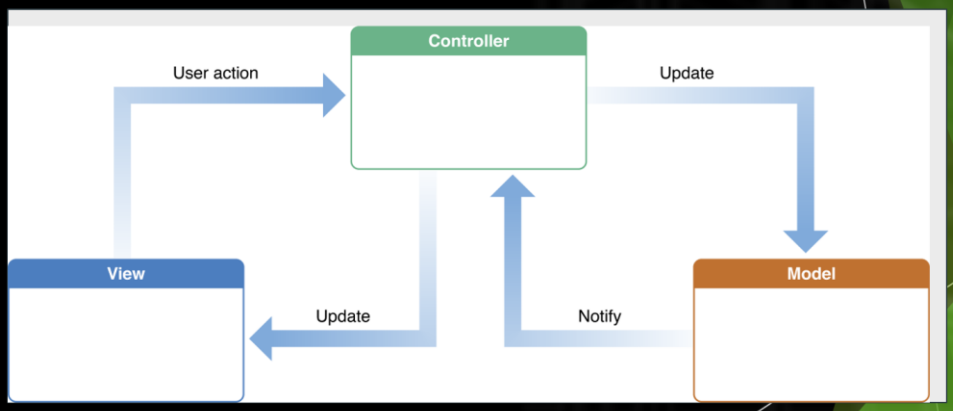
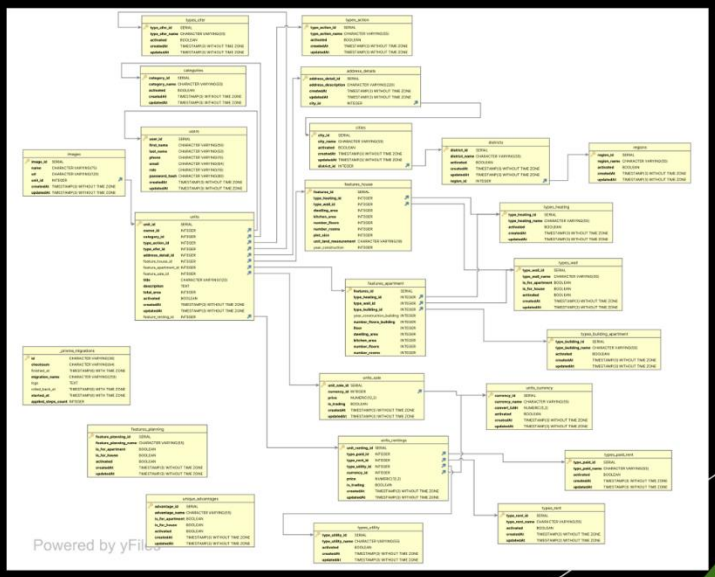
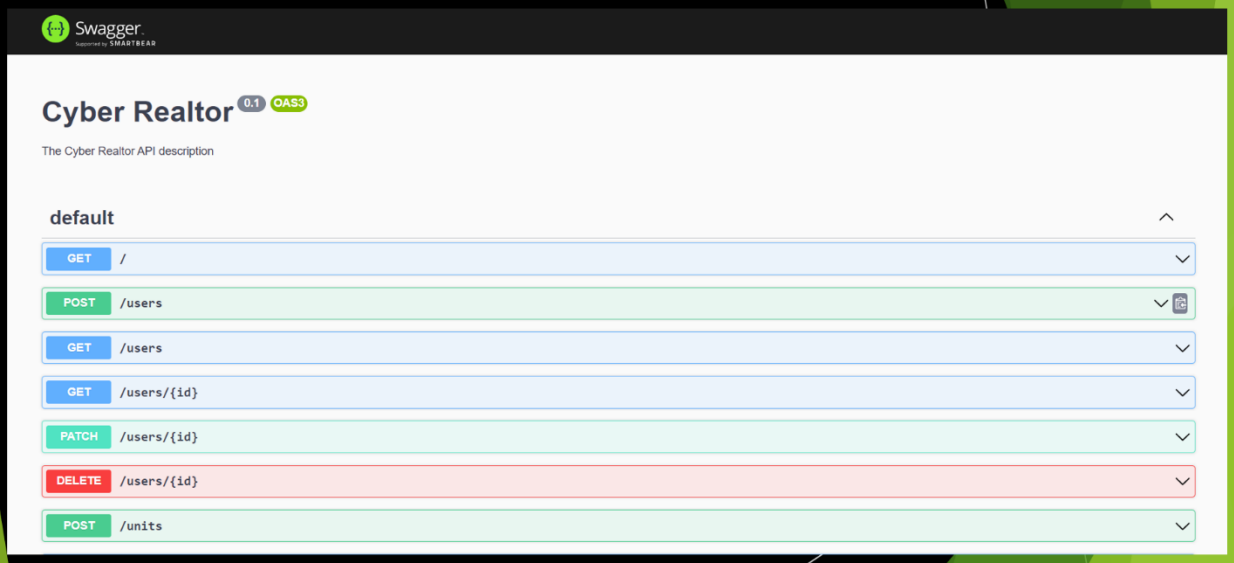


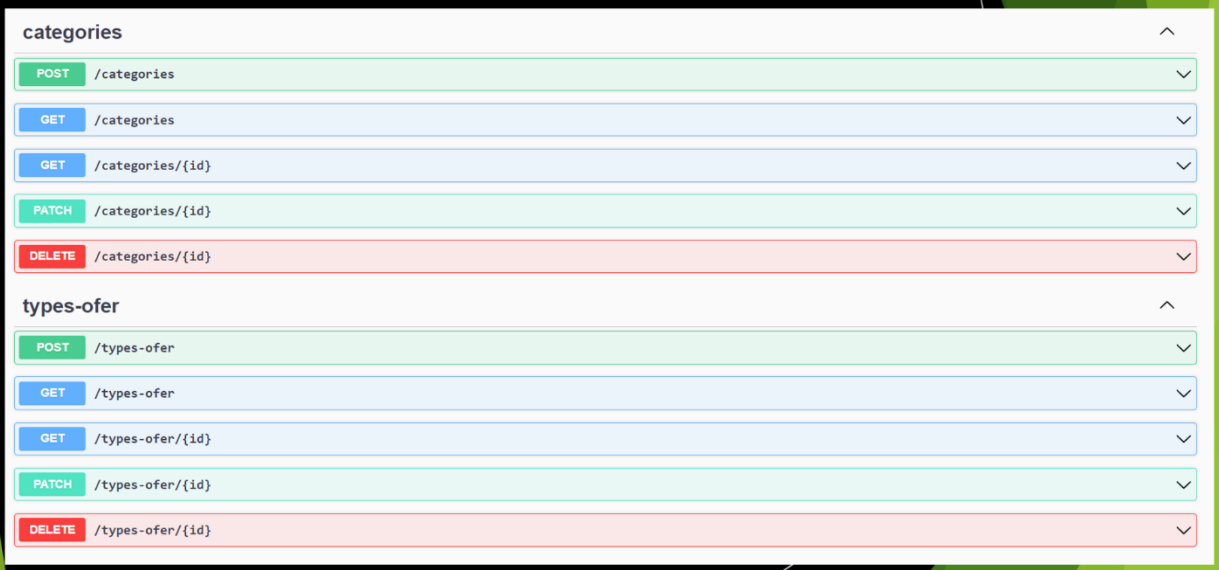
Схема бази даних



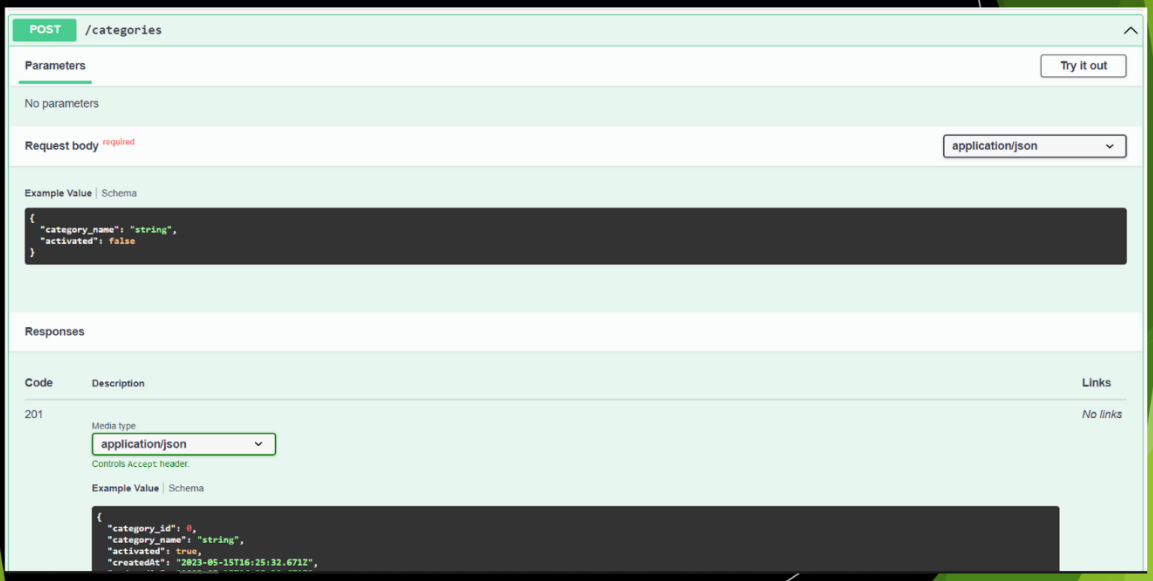
Реалізація проекту. Розробка REST API



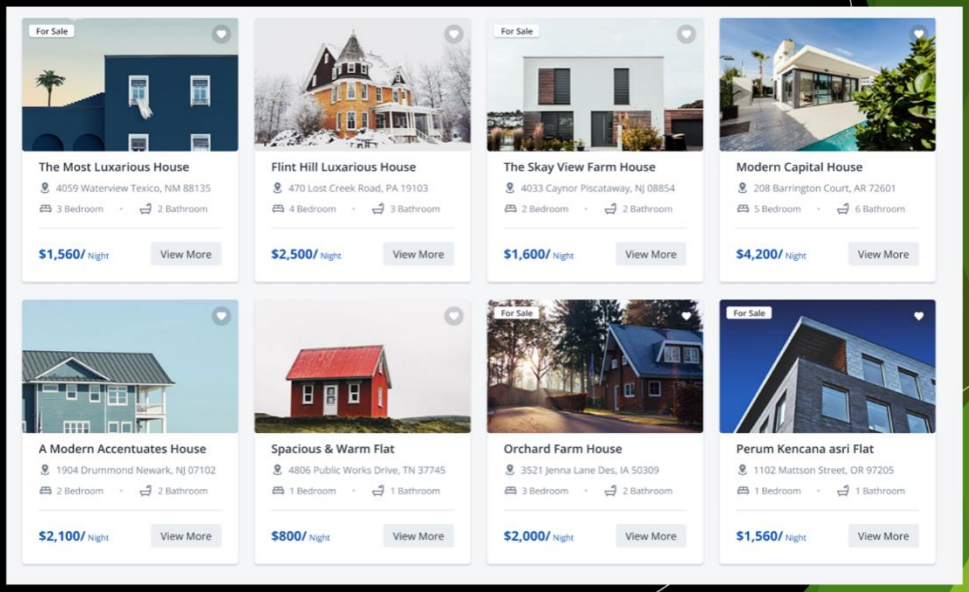
Реалізація проекту. Розробка REST API



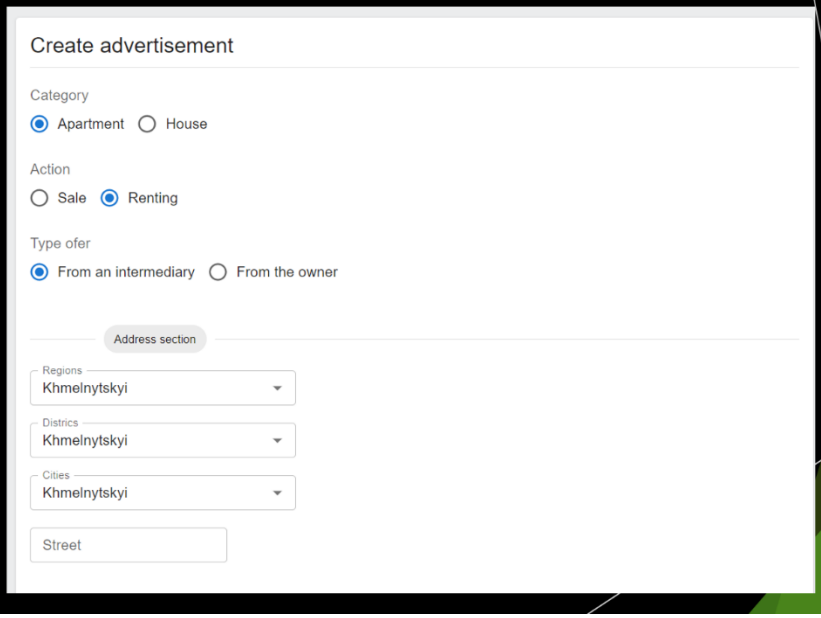
Реалізація проекту. Розробка REST API



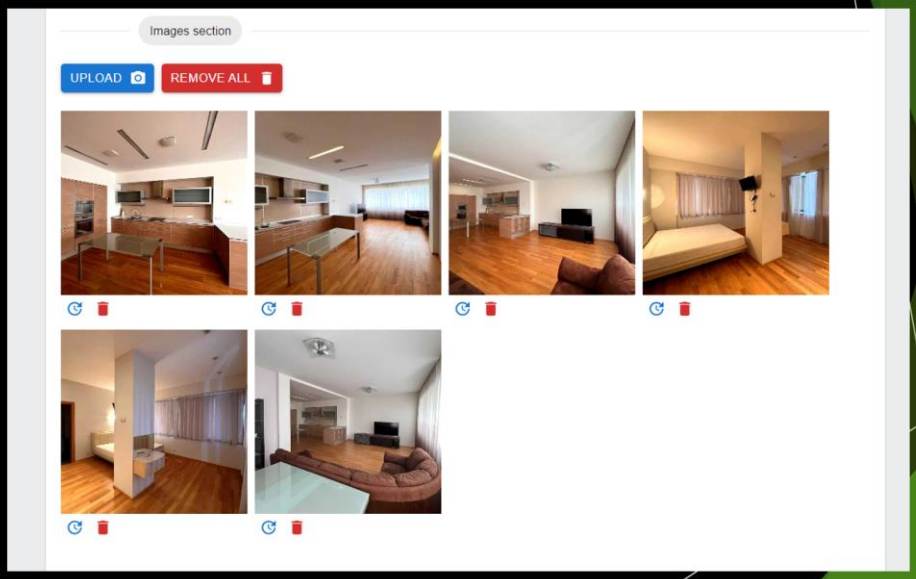
Реалізація проекту. Головна сторінка



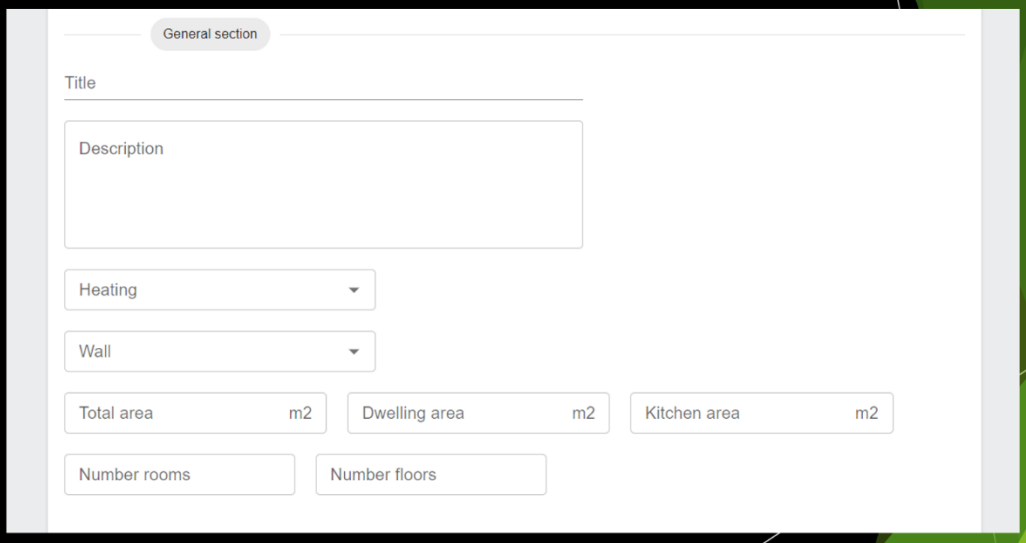
Реалізація проекту. Створення оголошення



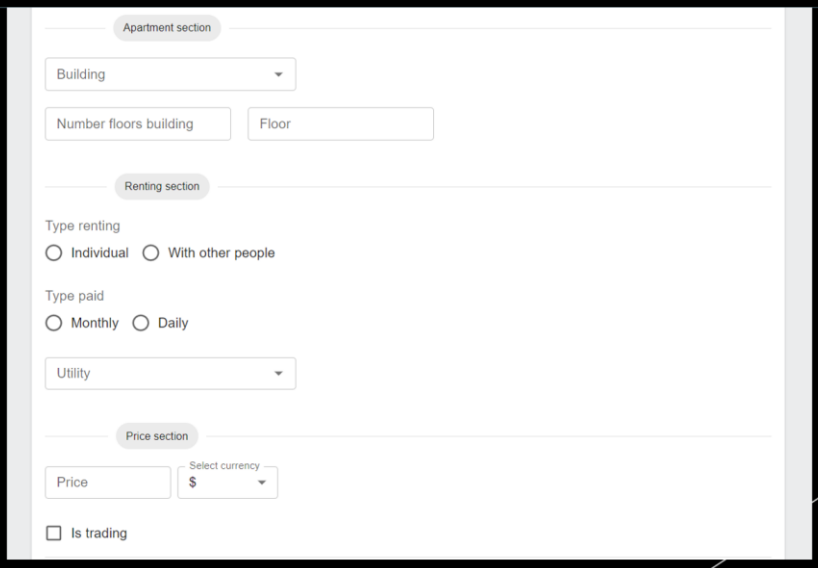
Реалізація проекту. Створення оголошення



Реалізація проекту. Створення оголошення



Реалізація проекту. Створення оголошення

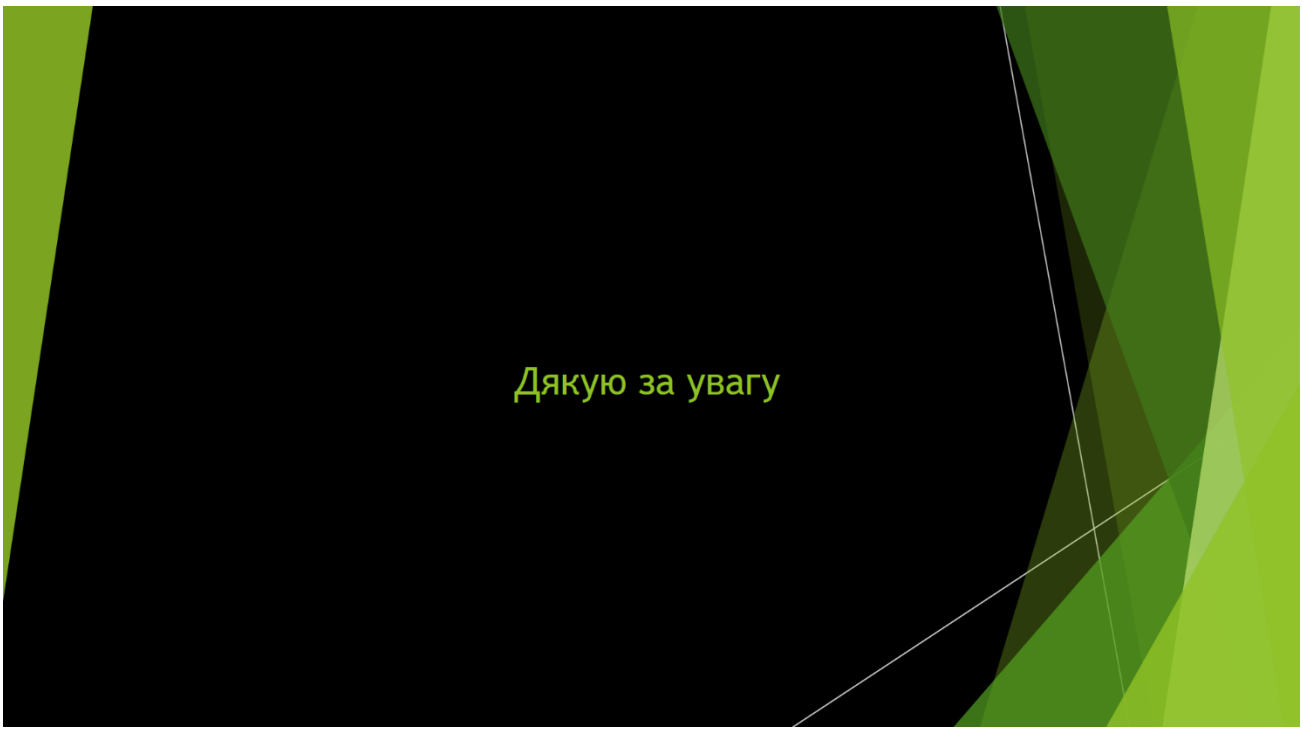


Висновки

Спроектовано інтерфейс, логіку та архітектуру веб-застосунку, базу даних і метод роботи API.

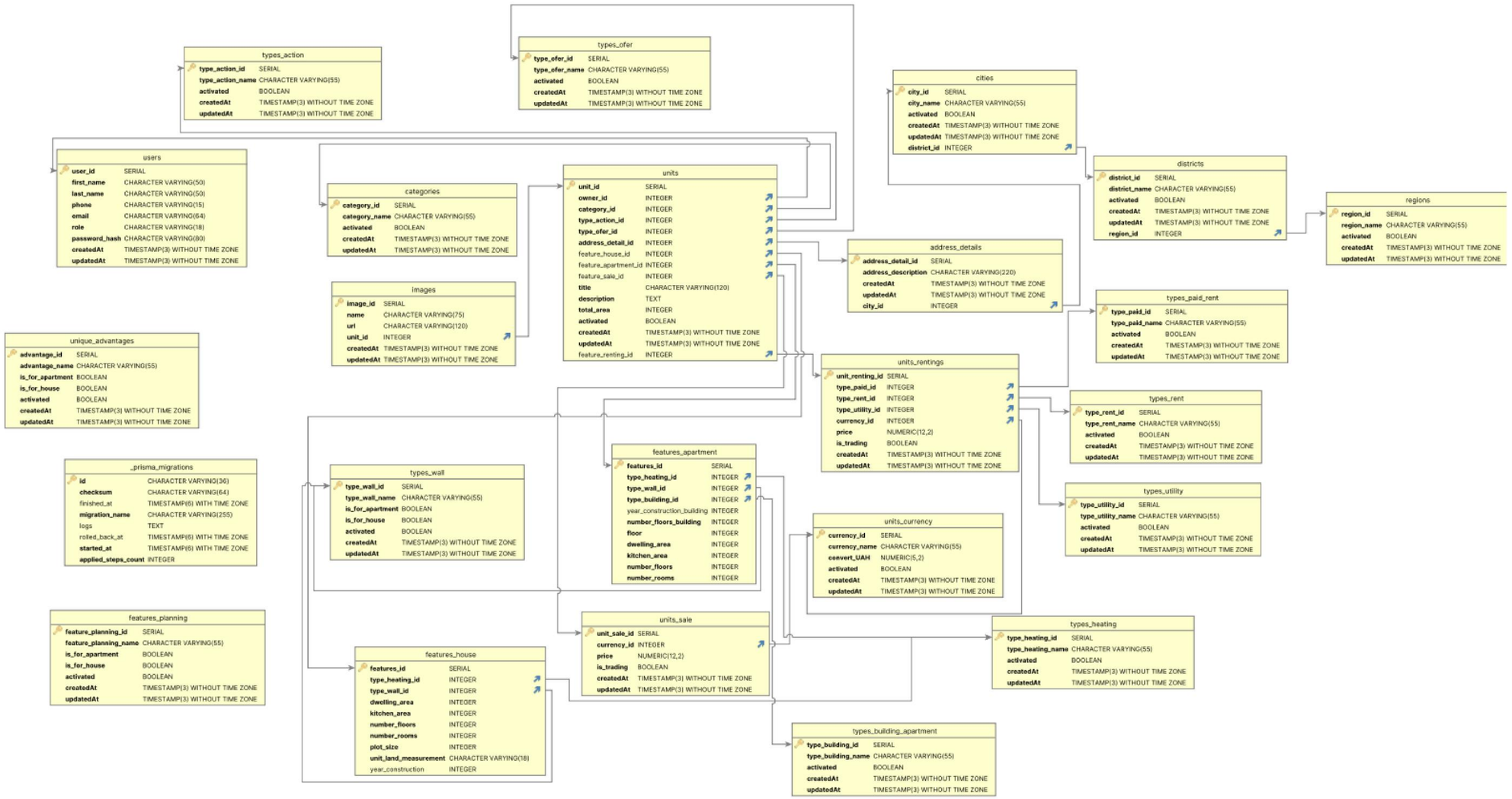
Створений веб-сайт відповідає поставленому завданню та усім визначеним вимогам.

Програмний продукт успішно сконструйований та готовий до використання. Судячи з цього, можна зробити висновок, що даний програмний продукт є цілком працездатним та справним.


Дякую за увагу

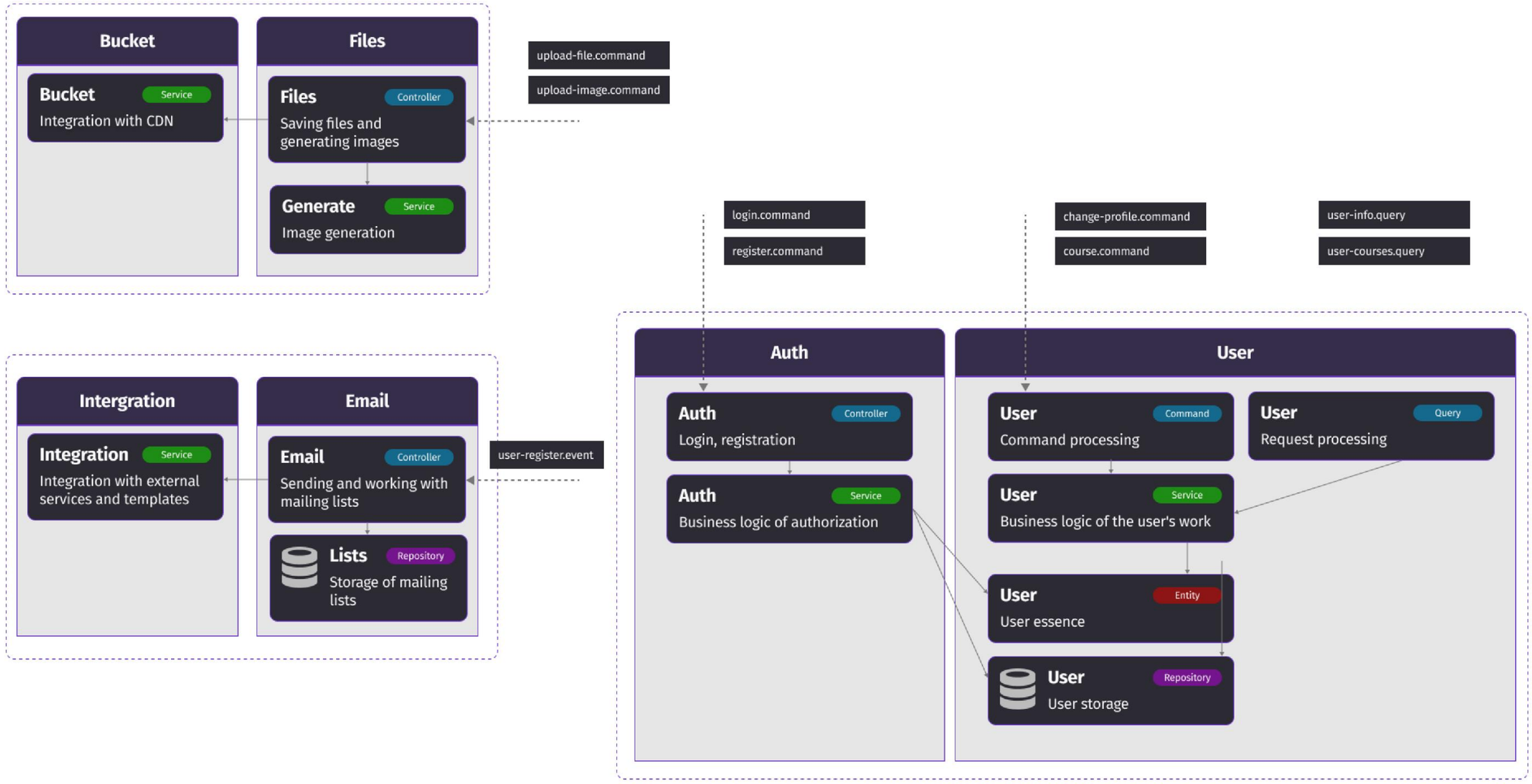
ГРАФІЧНА ЧАСТИНА

Структура бази даних



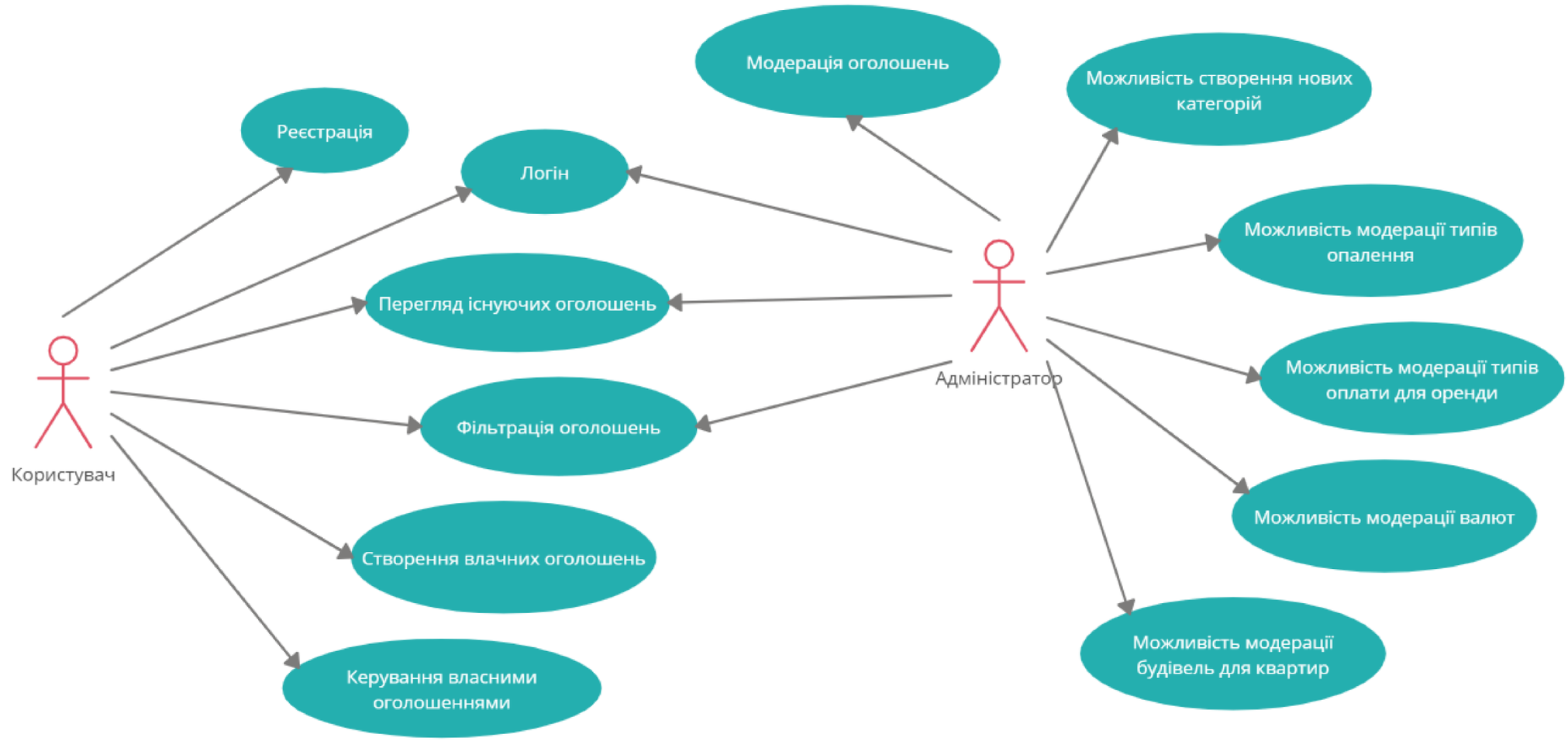
					КвРІП3.200123.01.09.Е8			
Зм.	Арк.	№докум.	Підпис	Дата	Веб-застосунок для автоматизації пошуку об'єктів нерухомості діаграма варіантів використання	Літера	Маса	Масштаб
Розробив		Шевчук М.І.		22.05.23				
Керівник		Онишко О.Г.		22.05.23		Аркуш 1	Аркушів 3	
Консульт.								
Н.Контр.		Гурман І.В.		22.05.23		ХНУ, ІПЗс-20-1		
Зав.каф.		Бедратюк Л.П.		22.05.23				

Структура серверного додатку



					КвРІП3.200123.01.09.Е8	
Зм.	Арк.	№докум.	Підпис	Дата	Веб-застосунок для автоматизації пошуку об'єктів нерухомості діаграма варіантів використання	Літера / Маса / Масштаб
Розробив		Шевчук М.І.		22.05.23		
Керівник		Онишко О.Г.		22.05.23		Аркуш 2 / Аркушів 3
Консульт.						
Н.Контр.		Гурман І.В.		22.05.23		ХНУ, ІП3с-20-1
Зав.каф.		Бедратюк Л.П.		22.05.23		

Діаграма варіантів використання



					КвРІП3.200123.01.09.Е8	
Зм.	Арк.	№докум.	Підпис	Дата	Веб-застосунок для автоматизації пошуку об'єктів нерухомості діаграма варіантів використання	Літера / Маса / Масштаб
Розробив		Шевчук М.І.		22.05.23		
Керівник		Онишко О.Г.		22.05.23		Аркуш 3 / Аркушів 3
Консульт.						
Н.Контр.		Гурман І.В.		22.05.23		ХНУ, ІП3с-20-1
Зав.каф.		Бедратюк Л.П.		22.05.23		

Завідувачу кафедри
інженерії програмного забезпечення
проф. Бедратюку Л. П.
студента групи ІПЗс-20-1
Шевчук М І
Прізвище, ініціали

ЗАЯВА

Прошу закріпити за мною тему кваліфікаційної роботи освітнього ступеня «бакалавр» за спеціальністю 121 «Інженерія програмного забезпечення»: Веб-застосунок для автоматизації пошуку об'єктів нерухомості

(керівник роботи – Онищко Оксана Григорівна)
Прізвище, ім'я, по батькові

05.02.2023
Дата

Підпис студента

Завідувачу кафедри інженерії програмного забезпечення проф. Бедратюку Л. П.

здобувача вищої освіти
Шевчука М. І.
Прізвище, ініціали

факультет ІТ, 3 курс, група ІПЗс-20-1

ЗАЯВА

З правилами чинного Положення «Про систему забезпечення академічної доброчесності в Хмельницькому національному університеті» від 01.07.2022, згідно з яким виявлення плагіату є підставою для відмови в допуску кваліфікаційної роботи до захисту і застосування заходів дисциплінарної та академічної відповідальності, ознайомлений. Про використання програмно-технічних засобів для перевірки кваліфікаційних робіт здобувачів вищої освіти на плагіат оповіщений та надаю свою згоду на обробку й збереження університетом моєї роботи в інституційному репозитарії університету.

Також надаю університету право на передачу моєї роботи для обробки та збереження в базах даних програмно-технічних засобів (Unicheck та/або Anti-Plagiarism) і використання роботи для виявлення академічного плагіату в інших роботах, які перевіряються програмно-технічними засобами та користувачами, що мають доступ до цих програмно-технічних засобів, виключно в обмежених цілях для виявлення плагіату в текстах робіт.

Робота для перевірки університетом надається в друкованому та електронному варіанті. Електронна версія моєї роботи збігається (ідентична) з друкованою.

05.06.2023
дата

підпис

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

ДЕКЛАРАЦІЯ УЧАСНИКА ОСВІТНЬОГО ПРОЦЕСУ

щодо дотримання академічної доброчесності

Цією декларацією я, Шевчук Микола Ігорович,

студент III курсу спеціальності 121 – Інженерія програмного забезпечення, група ІПЗс-20-1

здобувач вищої освіти (шифр та назва спец-ті, курс, академічна група)

підтверджую, що ознайомився (-лась) з Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті та Кодексом академічної доброчесності і **зобов'язуюсь** дотримуватися їх вимог під час освітнього процесу, проведення наукової діяльності, виконання організаційно-адміністративних функцій тощо.

Усвідомлюю, що у разі порушення мною принципів академічної доброчесності нестиму відповідальність перед академічною спільнотою ХНУ згідно з нормами, визначеними Положенням про систему забезпечення академічної доброчесності у Хмельницькому національному університеті, законодавства України.

05 лютого 2023 р.

Підпис

Mon Jun 05 07:33:55 EEST 2023, Хіврич Володимир Русланович, Хмельницький національний університет, ХНУ

Anti-Plagiarism v-15.257

Максимальне співпадіння з одним документом 11.0%

Словники перевірки: en_US, ru_RU, ua_UA. **Помилок в документах: 15%**

	Документ		Сумарний збіг по Базі Даних	
ID: 114662 Назва: БКР Веб-застосунок для автоматизації пошуку об'єктів нерухомості Додано в БД: 2023-06-05 Автора: Шевчук М.І. Керівники: Онишко О.Г. к.п.н. доц. Консультанти: Опоненти:	Символи	Лексеми	Символи	Лексеми
	76309	723	12962 (17%)	137 (19%)

Джерело плагіату

ID	Опис	Наявність плагіату в документі	
		Символи	Лексеми
111954	Назва: Звіт з переддипломної практики бакалавр на тему: Веб застосунок для автоматизації пошуку нерухомості Додано в БД: 2023-03-09 Автора: Шечук М.І. Керівники: Онишко О.Г. Консультанти: Опоненти:	8575 (11.0%)	75 (10.0%)

UNICHECK

Ім'я користувача:
Кафедра ІПЗ

ID перевірки:
1015421160

Дата перевірки:
05.06.2023 08:33:43 EEST

Тип перевірки:
Doc vs Internet + Library

Дата звіту:
05.06.2023 08:34:16 EEST

ID користувача:
100005589

Назва документа: **Шевчук_ІПЗс_20_1_Кваліфікаційна_робота (1)**

Кількість сторінок: 70 Кількість слів: 10918 Кількість символів: 88088 Розмір файлу: 6.93 MB ID файлу: 1015083686

14.2%
Схожість

Найбільша схожість: **6.27%** з джерелом з Бібліотеки (ID файлу: **1011233803**)

8.96% Джерела з Інтернету 546 ..... Сторінка 72

11.5% Джерела з Бібліотеки 128 ..... Сторінка 75

0% Цитат

Вилучення цитат вимкнене

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

РІШЕННЯ ЕКСПЕРНОЇ КОМІСІЇ
КАФЕДРИ ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ
ПРО ДОПУСК КВАЛІФІКАЦІЙНОЇ РОБОТИ ДО ЗАХИСТУ

Підтверджуємо ознайомлення з результатами звіту/звітів подібності щодо роботи, продукованими програмно-технічним засобом (ами) перевірки текстів на плагіат:

Назва: «Веб-застосунок для автоматизації пошуку об'єктів нерухомості»

Автор: Шевчук Микола Ігорович
Спеціальність: 121 – Інженерія програмного забезпечення
Освітня програма: Освітньо-професійна програма «Інженерія програмного забезпечення»
Науковий керівник: Онишко Оксана Григорівна, кандидат пед. наук, доцент
Після аналізу звіту подібності зроблено такий висновок:

№	Висновок	Позначка про відповідність
1	Запозичення, виявлені в роботі, є законними і не є плагіатом. Робота приймається до захисту.	**відповідає**
2	Виявлені запозичення не є плагіатом, розміщені в розділах, які не описують безпосередньо авторське дослідження, але кількість цитат перевищує обсяг, виправданий поставленою метою роботи. Робота приймається до захисту, але має бути відкоригована. Відкоригований варіант має бути поданий на кафедру за 2 дні до захисту, разом із заявою щодо самостійності виконання письмової роботи та ідентичності друкованої й електронної версії роботи	
3	Виявлені запозичення не є плагіатом, але частково розміщені в розділах, які описують безпосередньо авторське дослідження, а кількість цитат перевищує обсяг, виправданий поставленою метою роботи. В зв'язку з цим мета роботи та поставлені завдання не були досягнені. Робота може бути допущена до захисту (наступного року) після того, як буде відкоригована та допрацьована і успішно пройде повторну перевірку на академічний плагіат.	
4	Робота містить навмисні текстові спотворення, передбачувані спроби укриття запозичень або інші прояви академічного плагіату. Робота містить фабрикацію або фальсифікацію даних. Робота не допускається до захисту.	
5	Інше:	

Підтвердження:

Запозичення, виявлені в роботі, є законними і не є плагіатом, оскільки:

1) у тексті кваліфікаційної роботи системами перевірки на плагіат виявлено схожість з деякими документами в частині загальновживаних обов'язкових словосполучень у стандартних бланках (титулка, відомість документів), у структурі змісту, назвах розділів/підрозділів тощо, у назвах публікацій у переліку джерел посилання;

2) в якості запозичень системою було зафіксовано деякі послідовності вихідного коду і посиланання на бібліотеки, які є стандартними мовними конструкціями програмування та не можуть розглядатися як об'єкт авторських прав і, відповідно, їх порушення;

3) усі запозичення є фрагментарними або мають належним чином оформленні посилання;

4) виявлені модифікації тексту не впливають на відсоток схожості.

Сумарний обсяг всіх запозичень, визначений системою виявлення збігів ідентичності/ схожості, складає 14,2% і адресується до 546 джерел, що, з урахуванням наведених обгрунтувань, відповідає характеру теми і свідчить на користь кваліфікаційної роботи.

Дата 6.06.23

Завідувач кафедри _____ Леонід БЕДРАТЮК

Гарант освітньої програми _____ Леонід БЕДРАТЮК

Керівник кваліфікаційної роботи _____ Оксана ОНИШКО

ХМЕЛЬНИЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ

РЕЦЕНЗІЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ
освітнього ступеня «Бакалавр»

Дипломник Шевчук Микола Ігорович

Тема Веб-застосунок для автоматизації пошуку об'єктів нерухомості

Спеціальність 121 – Інженерія програмного забезпечення

Обсяг кваліфікаційної роботи:

Кількість листів креслень _____; кількість сторінок записки _____

1.Короткий зміст пояснювальної записки та прийнятих рішень У кваліфікаційній роботі досліджено і проаналізовано предметну область, визначено усі функціональні та нефункціональні вимоги. Був проведений аналіз існуючих програм на ринку, розглянуто їх переваги і недоліки, та доведено актуальність розробки нового програмного забезпечення. Розглянуто інструменти для реалізації спроектованих рішень, в результаті чого було вибрано два основних фреймворки NextJS та NestJS. NextJS завдяки своїй універсальності, допоміг швидко та якісно зробити клієнтську частину. NestJS є рішенням для серверного API, за домогою якого було створено безпечний та надійний серверний застосунок. В результаті використання цих інструментів і їх архітектурних практик був створений веб-застосунок. Також було проведено тестування програми, за результатами якого доведено, що розроблене програмне забезпечення працює коректно та готове до експуатації

2. Висновок про відповідність роботи поставленому завданню Кваліфікаційна робота виконана відповідно до поставленого завдання та з дотриманням всіх вимог.

3. Характеристика виконання кожного розділу роботи, ступінь використання останніх досягнень науки і техніки та передових методів роботи У вступі доведено актуальність теми, визначено мету та завдання дипломного проектування. У першому розділі проведено аналіз предметної області, розглянуто існуючі рішення та визначені функціональні і нефункціональні вимоги до розроблюваного програмного забезпечення. У другому розділі проведено аналіз сучасних архітектур, розглянуто їх переваги і недоліки та визначено, що система буде відповідати монолітній архітектурі та моделі клієнт-сервер. У третьому розділі підготовлено всі залежності для написання коду та виконано практичну розробку програмних модулів і описано їх особливості, в результаті чого створено програмний продукт. Також у цьому розділі виконано модульне тестування системи та проведено його у відповідності до функціональних вимог, в результаті було підтверджено коректну роботу програми.

4. Позитивні сторони роботи Тематика кваліфікаційної роботи є актуальною, оскільки на сьогодні в Україні веб застосунки для пошуку нерухомості не є достатньо розвинутими та не мають достатньої кількості функціональних можливостей. Також було застосовано новітні технології для побудови програмного продукту та актуальні архітектурні рішеня.

5. Негативні сторони роботи У роботі відсутній красивий дизайн, потрібно було б використати більше кольорів, а також можна було б зробити виявлення місця знаходження користувача для того аби знаходити для нього нерухомість поблизу.

6. Оцінка графічного оформлення та пояснювальної записки Графічне оформлення виконано відповідно до теми кваліфікаційної роботи та подано у вигляді діаграм і рисунків. Пояснювальна записка оформлена згідно вимог чинних стандартів.

7. Відгук про кваліфікаційну роботу в цілому Кваліфікаційна робота заслуговує позитивної оцінки. Матеріал пояснювальної записки структурований, послідовний, чіткий та простий, що дозволяє чітко зрозуміти викладений матеріал у рамках тематики проектування. Графічний матеріал дає можливість наочно побачити деталі проектування системи.

8. Інші зауваження

9. Оцінка кваліфікаційної роботи Кваліфікаційна робота виконана у повному обсязі, відповідає поставленій задачі та заслуговує на оцінку «добре».

РЕЦЕНЗЕНТ Кисіль Тетяна Миколаївна, кандидат фіз-мат наук, доцент кафедри комп`ютерної інженерії та інформаційних систем

"06" червня 2023 р. _____
(підпис)