

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук та технологій
(повне найменування факультету)

Кафедра комп'ютерних систем та мереж
(повне найменування кафедри)

Пояснювальна записка
до дипломного проекту (роботи)
бакалавра
(ступінь вищої освіти (освітній ступінь))

на тему **РОЗРОБКА КОМП'ЮТЕРНОЇ ПЛАТФОРМИ
ОНЛАЙН ОГОЛОШЕНЬ**

Виконав: студент 4 курсу, групи **КНТ-520**
спеціальності _____

123 «Комп'ютерна інженерія»
(код і найменування спеціальності)

Освітня програма (спеціалізація) _____
«Комп'ютерна інженерія»

РИБКІН Д.М.
(ПРИЗВИЩЕ та ініціали)

Керівник **СКРУПСЬКИЙ С.Ю.**
(ПРИЗВИЩЕ та ініціали)

Рецензент **КОЗІНА Г.Л.**
(ПРИЗВИЩЕ та ініціали)

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

Факультет Комп'ютерних наук і технологій
Кафедра «Комп'ютерні системи та мережі»
Ступінь вищої освіти бакалаврський
Спеціальність 123 Комп'ютерна інженерія
(код і найменування)
Освітня програма (спеціалізація) Комп'ютерна інженерія
(назва освітньої програми (спеціалізації))

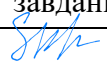
ЗАТВЕРДЖУЮ
Завідувач кафедри КУДЕРМЕТОВ Р.К.
“ ” 2024 року

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

РИБКІНА Дениса Максимовича
(ПРИЗВИЩЕ, ім'я, по батькові)

- Тема проекту (роботи) Розробка комп'ютерної платформи онлайн оголошень
керівник проекту (роботи) к. т. н., доцент, СКРУПСЬКИЙ Степан Юрійович
(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)
затверджені наказом вищого навчального закладу від “01” квітня 2024 року № 105
- Строк подання студентом проекту (роботи) 24 травня 2024 року
- Вихідні дані до проекту (роботи) наявні методи та логіка опрацювання,
розміщення онлайн оголошень купівлі-продаж
- Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
 - Визначення технологій розробки
 - Моделювання комп'ютерної системи;
 - Розробка системи.
- Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
 - Відношення таблиць БД;
 - Діаграма послідовності роботи з системою;
 - Інтерфейс вебсайту (сторінки).

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-3	СКРУПСЬКИЙ С. Ю., к.т.н., доцент		
нормоконтроль	ПОЛЬСЬКА О. В., ст. викл.		

7. Дата видачі завдання 25.03.2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Аналіз сучасних технологій для імплементації системи	12.04.2024 р.	
2	Розробка структури системи	21.04.2024 р.	
3	Розробка алгоритму роботи системи	23.04.2024 р.	
4	Розробка бази даних системи	29.04.2024 р.	
5	Реалізація серверної частини	01.05.2024 р.	
6	Реалізація клієнтської частини	14.05.2024 р.	
7	Оформлення результату у ПЗ	20.05.2024 р.	

Студент (ка)


(підпис)Денис РИБКІН
(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)


(підпис)Степан СКРУПСЬКИЙ
(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

ПЗ: 64 стор., 11 рис., 5 табл., 1 додаток, 17 джерел.

ВЕБСАЙТ, ОНЛАЙН ГОЛОШЕННЯ, СЕРВІС, ПЛАТФОРМА, NEST.JS, POSTGRESQL, NGINX, TYPEORM

Мета роботи полягає у розробці платформи з можливістю розміщення онлайн оголошень.

Система складається з серверної і клієнтської частини.

Клієнтська частина – інтерфейс користувача, завдяки якому можна взаємодіяти з системою. Функцією клієнтської частини також полягає у відправці запитів на сервер з метою отримання і відображення результатів, що були видані сервером.

Серверна частина приймає клієнтські запити, опрацьовує їх, здійснює запити до бази даних та надсилає відповіді клієнту. Серверна частина реалізована за допомогою технології Node.JS і сучасним фреймворком Nest.JS.

Клієнт-серверна архітектура широко використовується в різноманітних мережевих технологіях для доступу до різних мережевих сервісів.

ЗМІСТ

Вступ.....	7
1 Вибір технологій для реалізації системи	8
1.1 Вибір архітектури програмного забезпечення	8
1.2 Вибір технологій	10
1.3 Вибір технології для серверної частини	12
1.3.1 PHP.....	12
1.3.2 Python.....	13
1.3.3 Node.JS.....	15
1.4 Вибір СКБД.....	17
1.4.1 MongoDB.....	18
1.4.2 MySQL.....	19
1.4.3 PostgreSQL	20
1.4.4 Redis	22
1.5 Вибір прокси-серверу	23
1.6 Висновки з вибору технологій.....	24
2 Моделювання комп'ютерної системи	24
2.1 Загальні вимоги до комп'ютерної системи	24
2.2 Функціональні можливості користувачів	25
2.3 Функціонал системи.....	28
2.4 Опис таблиць БД	28
2.5 Діаграма послідовностей	33
3 Розробка системи	36
3.1 Файлова структура проекту.....	36
3.2 Ініціалізація розробки	38
3.2.1 Конфігурація бази даних	40
3.3 Імплементация модуля авторизації	43
3.4 Реалізація модуля файлів.....	49

3.5 Користувачський інтерфейс	52
Висновки	56
Перелік джерел посилання	57
Додаток А Основні коди програмних модулів.....	59

ВСТУП

В сучасному світі, де технології невпинно розвиваються, питання створення комп'ютерних систем для ведення власного бізнесу є актуальним. Ринок купівлі та продажу товарів та послуг не є винятком, і з кожним днем він стає все більш динамічним та комплексним. У такому контексті розробка комп'ютерної платформи онлайн оголошень купівлі-продажу стає стратегічно важливим кроком для задоволення потреб ринку та його учасників.

Проте, розвиток таких платформ неминуче стикається з викликами. Як зробити процес пошуку та розміщення оголошень максимально зручним та ефективним? Як забезпечити безпеку та конфіденційність користувачів платформи? Як забезпечити гарантованість виконання угод? За допомогою сучасних технологій можна створити комп'ютерну систему, яка буде посередником між продавцем і покупцем. Аналогічно OLX, продавці будуть реєструватися в системі, розміщувати свої оголошення з продажу товарів або послуг, і здійснювати угоди з клієнтами.

У цьому проекті буде отримано комп'ютерну платформу онлайн оголошень, яка буде не лише відповідати сучасним вимогам, але й виходити за їх межі, пропонуючи новаторські рішення та сервіси, що забезпечать користувачам найвищий рівень задоволення та комфорту.

Проект включає в себе огляд існуючих рішень у даній сфері, визначення основних вимог до системи, розробку її архітектури та інтерфейсу, а також тестування створеного прототипу. Ця робота має на меті створення ефективного інструменту, що враховує сучасні тенденції у веб-розробці та інтерактивному дизайні, для забезпечення користувачам комфортного та продуктивного досвіду використання.

1 ВИБІР ТЕХНОЛОГІЙ ДЛЯ РЕАЛІЗАЦІЇ СИСТЕМИ

1.1 Вибір архітектури програмного забезпечення

Під час розробки платформи, вибір правильної архітектури програмного забезпечення має величезне значення для успішності проекту. Архітектура програмного забезпечення визначає структуру, взаємозв'язок та розподіл функцій між компонентами системи, що визначається потребами бізнесу та технічними вимогами проекту.

Серед основних принципів архітектури програмного забезпечення варто зазначити модульність, чистоту інтерфейсів, розширюваність, переносність і високу зручність та ефективність. Правильний вибір архітектурного підходу дозволяє легко розширювати та модифікувати систему, забезпечуючи високу якість та ефективність розробки.

Для побудови веб-проекту буде застосовуватися клієнт-серверна архітектура. Клієнт-серверна архітектура - це модель взаємодії між комп'ютерами у мережі, де один комп'ютер (клієнт) надсилає запити до іншого комп'ютера (сервера) з метою отримання даних. Клієнт-серверна архітектура набула своєї популярності завдяки динамічному розвитку мережі Інтернет та зосередження значної частини ресурсів в базах даних на серверах, які обслуговують запити від клієнтів [1].

Клієнт-серверна архітектура включає наступні компоненти:

- сервери, які надають інформацію або інші послуги програмам, що звертаються до них;
- клієнти, які використовують сервіси, надані серверами;
- мережу, яка забезпечує взаємодію між клієнтами та серверами.

Клієнт – це комп'ютер або пристрій, який ініціює запити до сервера. Клієнтська частина системи може бути реалізована як веб-браузер, мобільний додаток, десктопний застосунок або будь-який інший інтерфейс, здатний взаємодіяти з сервером. Клієнтська частина відповідає за відображення інформації для користувача та обробку його введених даних перед відправленням їх на сервер.

Сервер – це комп'ютер або це програма або пристрій, який надає послуги або ресурси, що запитуються клієнтами. Серверний компонент приймає запити від клієнтів, обробляє їх і відправляє відповіді назад клієнтам. Він також відповідає за зберігання та керування даними, обробку бізнес-логіки і виконання інших завдань, необхідних для забезпечення функціональності системи.

Для взаємодії з системою користувач може скористатися звичайним веб-браузером, мобільним додатком або програмою на комп'ютері. Під час взаємодії із користувачем додаток або браузер генерує запит, який надсилається на сервер. На сервері проводиться обробка, яка може включати виклик інших серверних модулів або запит до бази даних. Після цього сервер відправляє відповідь на клієнт.

Клієнт-серверна архітектура має наступні переваги:

- розділення обов'язків. Ця архітектура дозволяє чітко розділити функціональні обов'язки між клієнтом і сервером, що полегшує розробку, тестування та підтримку системи;
- масштабованість. Клієнт-серверна архітектура дозволяє масштабувати систему шляхом додавання нових клієнтів або серверів за необхідності;
- централізоване керування даними. Сервер забезпечує централізоване керування даними, що дозволяє забезпечити їхню консистентність та безпеку;
- зменшення навантаження на клієнтську сторону. Це дозволяє підтримувати легкі та ефективні клієнтські додатки, що забезпечує кращу продуктивність та зручність для користувачів.

Серед недоліків які варто враховувати при розробці системи є наступні:

- одинична точка відмови. Якщо сервер виявиться недоступним, це може спричинити відмову всієї системи;
- мережева залежність. Робота системи залежить від надійності мережевого з'єднання між клієнтами та серверами;
- складність синхронізації даних. Виникають складності при забезпеченні консистентності даних між клієнтами та сервером, особливо в умовах розподіленості та паралельної обробки;

- віддалений доступ до ресурсів. Деякі операції можуть бути повільними через потребу в мережевому з'єднанні для доступу до сервера.

1.2 Вибір технологій

Оскільки клієнтська частина являти собою вебсайт, то його розробки передбачається використання таких мов HTML, CSS, JavaScript для клієнтської (frontend) частини і будь якої з мов програмування за допомогою якого можна підняти серверну (backend) частину. Оскільки серверна частина не передбачає наявність користувацького інтерфейсу, вона може складатися як з одної, так і декількох мов програмування в залежності від складності системи.

HTML (HyperText Markup Language) - це стандартна мова розмітки для створення веб-сторінок і веб-додатків. Вона використовується для структурування та відображення контенту в Інтернеті. HTML визначає структуру сторінки за допомогою різних елементів та тегів, які вказують браузеру, як відображати текст, зображення, відео, посилання та інші елементи на сторінці. Використання HTML є основною складовою для створення веб-сторінок та додатків, і вона часто поєднується з CSS (Cascading Style Sheets) для стилізації вигляду сторінок та з JavaScript для додавання інтерактивності.

Елементи HTML складаються з тегів, які визначають їхню структуру та вміст. Теги містять параметри, які дозволяють налаштовувати вигляд або змінювати вміст елементів. Наприклад, змінювати розмір або колір тексту. Однак для таких змін краще використовувати CSS.

Для полегшення розробки на мові HTML можуть використовуватися препроцесори. Препроцесор для HTML - це інструмент або мова, яка допомагає зручніше створювати та управляти HTML-кодом. Він дозволяє використовувати розширені функції та синтаксис, які покращують продуктивність і роблять розробку веб-сторінок більш ефективною. Препроцесори HTML можуть включати

функції, такі як вбудовані шаблони, змінні, імпорт файлів та багато інших. Після обробки препроцесором HTML-код перетворюється в стандартний HTML, який може бути інтерпретований браузером. Деякі з найпопулярніших препроцесорів для HTML включають Pug (раніше відомий як Jade), Haml, Slim та інші.

CSS (Cascading Style Sheets) - це мова стилів, яка використовується для оформлення та стилізації веб-документів, написаних мовою HTML або XHTML. CSS визначає вигляд елементів веб-сторінки, таких як кольори, шрифти, розміри, відступи, розташування та інші аспекти візуального оформлення. Використання CSS дозволяє відокремлювати структуру сторінки від її вигляду, що сприяє покращенню маневреності та підтримки веб-додатків. CSS можна використовувати як внутрішньо, так і зовнішньо, а також в режимі вбудованої таблиці стилів.

CSS є ключовою мовою для розробки веб-сайтів і відповідає стандартам W3C. Починаючи з CSS1 та CSS2.1, розробка стилів включала випуск різних частин специфікації. Однак починаючи з CSS 2.1, специфікація стала значно об'ємнішою, що вимагало зміни підходів до випуску рекомендацій.

CSS також має свої препроцесори (наприклад, SCSS), який надає розширений синтаксис, змінні, вкладеність, міксини, умовні конструкції і імпорт, що полегшує роботу з CSS та забезпечує кращу підтримку та обслуговування проектів.

JavaScript - це мова програмування, яка використовується для створення інтерактивних веб-сайтів та веб-додатків. Вона є однією з трьох основних технологій веб-розробки, разом з HTML (HyperText Markup Language) і CSS (Cascading Style Sheets). JavaScript дозволяє динамічно маніпулювати вмістом веб-сторінок, реагувати на події користувача, створювати анімацію, перевіряти введені дані, виконувати асинхронні запити до сервера та багато іншого. Вона є мовою з відкритим вихідним кодом, що активно розвивається та підтримується спільнотою програмістів [2]. JavaScript виконується в середовищі веб-браузера, але також може використовуватися для розробки серверних додатків з використанням платформ, таких як Node.js.

1.3 Вибір технології для серверної частини

Серверна частина складається з одної або декількох мов програмування, а також однієї або декількох баз даних, що використовуються для зберігання та управління даними на серверній стороні. Для проекту необхідно розглянути деякі з найпопулярніших мов програмування на стороні сервера.

1.3.1 PHP

PHP (Hypertext Preprocessor) - це мова програмування загального призначення, яка використовується переважно для розробки веб-додатків та веб-сайтів. Вона широко використовується для створення динамічних веб-сторінок, обробки форм, взаємодії з базами даних та багато іншого [3].

PHP вбудовується безпосередньо у HTML-код, що дозволяє використовувати його для створення шаблонів веб-сторінок. Ця мова є відкритою і безкоштовною, і її можна використовувати на більшості веб-серверів, що підтримують PHP-інтерпретатор.

PHP має велику спільноту користувачів та розробників, а також широкий набір фреймворків і бібліотек, які полегшують розробку веб-додатків. Вона підтримується на всіх основних операційних системах, таких як Windows, macOS та Linux, що робить її популярним вибором для розробки веб-проектів різної складності.

Як вже зазначалося, PHP найчастіше використовується для веб-сайтів та веб-додатків, і його використання в інших проєктах або застосунках не так часто помічається. Однак PHP вважається однією з найкращих мов програмування для серверної частини, оскільки вона має вбудовану підтримку для них. Крім того, ця мова програмування досить гнучка у використанні, що дозволяє використовувати її у всіх веб-проєктах, незалежно від їхньої складності чи обсягу.

Переваги PHP:

- простота вивчення. Синтаксис PHP легкий для розуміння, що робить його доступним для початківців і швидким для вивчення;

- широке поширення. PHP підтримується більшістю веб-хостингових провайдерів, що робить його доступним для використання на більшості веб-серверів;
- багато ресурсів. Існує велика кількість документації, туторіалів та відкритих бібліотек, які полегшують розробку веб-додатків на PHP;
- швидка розробка. Завдяки великій кількості готових функцій і фреймворків, PHP дозволяє розробляти веб-додатки швидше і ефективніше.

Недоліки PHP:

- змішаний код. Можливість вбудовувати PHP безпосередньо в HTML може призводити до змішаного коду, що робить його складним для розуміння і підтримки великих проектів;
- безпека. Історично PHP мала проблеми з безпекою, особливо коли використовувалися старі версії або коли розробники не слідували кращим практикам безпеки;
- швидкодія. У порівнянні з іншими мовами програмування, такими як Node.js або Python, PHP може бути менш ефективним з точки зору швидкодії;
- менш модульна структура. Натомість використання об'єктно-орієнтованого програмування (ООП) у PHP не так розповсюджене, як у деяких інших мовах, що може ускладнювати підтримку великих проектів.

Незважаючи на популярність даної мови програмування. Синтаксис її не є зручним у використанні і погано читається. До того ж він поребує додаткових знань і навичок у використанні, адже він є не тільки мовою програмування, а ще і шаблонізатором, що робить його використання не зовсім зручним.

1.3.2 Python

Python - це високорівнева, інтерпретована, мультипарадигмальна мова програмування загального призначення. Вона була розроблена Гвідо ван Россумом і вперше випущена в 1991 році. Синтаксис Python спрощений максимально, що дозволяє швидко освоїти мову за короткий час. Проте це може викликати дискомфорт у розробників, які переходять з інших мов програмування типу C,

оскільки в Python відсутні фігурні дужки, а вкладеність визначається відступами [4].

Ядро Python має дуже зручну структуру, а широкий асортимент вбудованих бібліотек дозволяє використовувати різноманітні корисні функції і можливості. Серед основних сфер використання можна виділити веб-розробку, data science (машинне навчання, аналіз даних та візуалізація) та автоматизацію процесів.

Проте Python не рідко використовується у якості сервера для створення веб-застосунків та API, де кожен HTTP запит обробляється в окремому потоці, який протягом свого життя обробляє безліч запитів, що відображається у наявності пулу потоків.

До переваг Python відносять:

- простота вивчення та розробки. Python має простий та зрозумілий синтаксис, що робить його досить легким для вивчення і використання, особливо для початківців у програмуванні;
- широке співпраця. Python має велику спільноту розробників, яка активно розвиває різноманітні бібліотеки та фреймворки, що дозволяє легко знаходити рішення для різних завдань;
- багатофункціональність. Python підтримує багато різних бібліотек та інструментів для різних завдань, що робить його дуже гнучким для використання в різних сферах;
- велика стандартна бібліотека. Python має велику стандартну бібліотеку, яка включає в себе різноманітні модулі для роботи з мережами, базами даних, обробки даних та багато іншого.

Серед недоліки виділяють:

- швидкодія. У порівнянні з іншими мовами, такими як C++ або Go, Python може бути менш ефективним з точки зору швидкодії, особливо при роботі з великими обсягами даних або великими обчислювальними завданнями;
- потребує великої кількості ресурсів. Python використовує більше пам'яті та процесорного часу порівняно з іншими мовами програмування, що може призвести до високого споживання ресурсів на сервері;

- нестабільність деяких бібліотек. Оскільки Python має велику кількість різноманітних бібліотек та фреймворків, не всі вони можуть бути настільки стабільними або підтримуваними, як інші мови програмування;
- повільна швидкість. Відсутність відповідного досвіду та складність розуміння коду через його синтаксис роблять Python менш вигідним варіантом для створення потрібного продукту.

1.3.3 Node.JS

Node.js - це відкрите середовище виконання JavaScript, яке дозволяє виконувати JavaScript-код на стороні сервера. В основі Node.js лежить двигун V8, який є використовуваним Google Chrome для виконання JavaScript-коду у веб-браузерах.

Основні особливості Node.js включають:

- асинхронний та подієвий ввід/вивід. Node.js працює асинхронно, що означає, що він може обробляти багато запитів одночасно без блокування виконання інших операцій. Це забезпечує високу продуктивність та масштабованість додатків;
- платформонезалежність. Node.js може працювати на різних операційних системах, таких як Windows, macOS та Linux;
- модульність. Node.js має велику кількість модулів, які можна легко встановлювати та використовувати за допомогою системи керування пакетами npm (Node Package Manager);
- розробка веб-додатків. Node.js дозволяє створювати швидкі та масштабовані веб-додатки, включаючи веб-сервери, API та інші застосунки;
- велика спільнота. Node.js має велику та активну спільноту розробників, яка надає підтримку, допомогу та безліч корисних ресурсів для вивчення та розвитку.

Node.js базується на движку V8, що використовується в браузері Google Chrome для трансляції JavaScript у машинний код [6]. Основна мета Node.js полягає в тому, щоб доповнити можливості JavaScript і дозволити використовувати його як серверну мову програмування.

У браузерях JavaScript не має API для взаємодії з пристроями вводу/виводу та операційною системою через питання безпеки. Node.js розв'язує цю проблему, дозволяючи викликати API, написаний на C++ (за допомогою бібліотеки libuv), для взаємодії з системними ресурсами.

Як вже зазначалось, Node.js використовує систему орієнтації на події з неблокуючою моделлю вводу/виводу, замість традиційної моделі паралелізму на основі потоків. Блокуючий ввід/вивід (Blocking I/O) - це підхід до обробки операцій вводу/виводу, коли програма зупиняє своє виконання, очікуючи завершення операції вводу/виводу, перед тим як продовжити виконання інших інструкцій. В Node.js, такі операції можуть бути, наприклад, читання з файлу або отримання даних з бази даних.

У блокуючого вводу/виводу, коли виконується операція, програма "блокується" і не може виконувати інші задачі, поки операція не буде завершена. Це може призводити до того, що програма стає менш масштабовною і продуктивною, особливо при великому навантаженні або великій кількості одночасних запитів.

У Node.JS "Event Loop" слідкує за тим, щоб асинхронні операції були завершені, і коли вони завершуються, він виконує колбеки або події, пов'язані з цими операціями. Це дозволяє Node.js продовжувати виконання інших задач, поки чекається завершення асинхронних операцій, забезпечуючи високу продуктивність та ефективність виконання програм.

Event Loop в Node.js працює в циклі, де він постійно перевіряє чергу подій і виконує обробку цих подій у порядку, в якому вони надходять. Це дозволяє Node.js обробляти багато подій одночасно та ефективно керувати великими навантаженнями [7].

Додатково, Node.JS має безліч фреймворків, серед яких є NestJS, що зараз набуває популярність і є прогресивним для створення масштабованих та ефективних серверних застосунків [8][9]. Він базується на TypeScript і використовує об'єктно-орієнтований стиль програмування, а також принципи SOLID. NestJS пропонує розробникам уніфіковану архітектуру та набір

інструментів для створення серверних додатків, включаючи контролери, провайдери, модулі, middleware та багато іншого. Він також інтегрується з різноманітними бібліотеками та інструментами, що допомагає розробникам швидше та ефективніше створювати сучасні веб-додатки.

Отже, для розробки системи можна вибрати платформу Node, оскільки недоліки, що вона має, є незначними для поточної задачі. Головними перевагами, які спонукали до вибору саме цієї платформи, є швидкість розробки та можливість використання однієї мови як на клієнтській, так і на серверній стороні.

1.4 Вибір СКБД

Система керування базами даних (СКБД) відіграє ключову роль у сучасній інформаційній технології. База даних (БД) представляє собою організовану структуру для зберігання, модифікації та обробки взаємозалежної інформації, яка може мати різні моделі даних, такі як реляційна, документ-орієнтована, ієрархічна та інші. Стосовно СКБД - це система керування базами даних, яка забезпечує зберігання, доступ, і обробку даних. Вона допомагає організувати великі обсяги інформації, забезпечуючи швидкий доступ до неї та забезпечуючи надійність та цілісність даних. Такі системи використовуються у різних галузях, включаючи бізнес, науку, та громадський сектор, для зберігання та обробки великих обсягів інформації [10]. Бази даних поділяються на реляційні та нереляційні.

Реляційні бази даних використовують реляційну модель, де дані зберігаються у вигляді таблиць зі стовпцями різного типу, кожен рядок таблиці має унікальний ключ і складається з атрибутів. Для роботи з реляційними базами даних використовується структурована мова запитів, така як SQL (Structured Query Language). Також вони зазвичай надають високий рівень транзакційної безпеки, що дозволяє виконувати операції з даними в умовах стабільності та надійності. До найпопулярніших СКБД відносять: MySQL, PostgreSQL, Oracle Database, тощо.

У нереляційних СКБД (NoSQL) дані зберігаються без жорсткої структури таблиць, зазвичай у вигляді ключ-значення, документів, колекцій тощо. Ці бази даних зазвичай дозволяють зберігати дані різного типу без строгої схеми, що робить їх дуже гнучкими для зберігання й обробки неструктурованих даних. Нереляційні СКБД можуть бути дуже ефективними в операціях з великим обсягом даних та високими навантаженнями, оскільки вони оптимізовані для розподіленого зберігання та обробки даних. Приклади: MongoDB, Cassandra, Redis.

Обираючи між реляційними та нереляційними СКБД, важливо враховувати особливості проекту, його вимоги до швидкодії, гнучкості та конкретних типів даних. Реляційні бази даних зазвичай краще підходять для проектів з визначеною структурою даних та потребою в транзакційній безпеці, тоді як нереляційні СКБД можуть бути більш ефективними для роботи з великими обсягами неструктурованих даних та вимогами до високої швидкодії обробки [11].

1.4.1 MongoDB

MongoDB - це документ-орієнтована система управління базами даних (NoSQL), яка спроектована для зберігання, організації та операцій з великими обсягами структурованих та неструктурованих даних у вигляді документів у форматі JSON-подібних об'єктів. Однією з ключових особливостей MongoDB є гнучкість схеми даних, оскільки вона не потребує фіксованих структур для записів, що дозволяє зберігати дані різної структури у одній колекції. MongoDB також підтримує широкий спектр можливостей, включаючи реплікацію, шардування, індексацію, агрегаційні операції та інше. Вона часто використовується у веб-розробці, аналітиці даних, IoT-проектах та інших областях, де необхідно працювати з великими обсягами даних та потрібна гнучкість у схемі [12].

Відмінністю MongoDB від реляційних БД є те, що MongoDB зберігає не рядки у таблиць, а документи у колекціях. Документи можуть зберігати складну за структурою інформацію, проте пошук таких даних є зазвичай обмеженим.

Для створення запитів до БД використовуються як консольна оболонка, так і додатки з графічним інтерфейсом, наприклад, MongoDB Compass.

MongoDB має наступні переваги:

- гнучкість схеми даних. MongoDB використовує документ-орієнтований підхід, що дозволяє зберігати дані різної структури у одній колекції. Це робить базу даних більш гнучкою у порівнянні з традиційними реляційними базами даних;
- швидкодія. MongoDB часто виявляється дуже швидким у виконанні операцій з даними, особливо коли вони орієнтовані на шукання та зчитування;
- простота. Не потрібно визначати структуру документів заздалегідь, що означає, що нові поля можна додавати до документів без зайвих складнощів.

Основними недоліками виступають:

- не підходить для багатьох сценаріїв. Хоча MongoDB є потужним інструментом, він не завжди підходить для всіх сценаріїв використання, зокрема для додатків, які вимагають складних транзакцій або зв'язків між даними;
- вимоги до ресурсів. Розміри даних у MongoDB зазвичай більше, ніж у реляційних БД і можуть вимагати значних обчислювальних та пам'яткових ресурсів, особливо при великих обсягах даних та високому навантаженні.

Роблячи висновки, MongoDB є непоганим вибором у створенні простих систем, де потрібна простота і швидкодія. Проте він не підходить у проектуванні системи, яка передбачає велику кількість запитів і зберігання структурованих даних.

1.4.2 MySQL

MySQL – це одна з найпопулярніших СКБД, яка використовується для зберігання, управління та доступу до даних у веб-додатках, корпоративних системах та інших програмних застосунках. Вона базується на реляційній моделі даних, де дані організовані у вигляді таблиць, а доступ до них здійснюється за допомогою структурованої мови запитів SQL. СКБД має велику спільноту користувачів та розробників і підтримується на різних операційних системах, таких як Windows, Linux, macOS та інші. MySQL має різноманітні типи зберігання даних, такі як InnoDB, MyISAM, MEMORY та інші, що дозволяє вибрати оптимальний тип для конкретної задачі [13].

Основні особливості MySQL включають високу швидкодію, надійність, масштабованість та можливість розширення за допомогою різноманітних

додаткових модулів та розширень. Вона часто використовується для створення веб-сайтів, онлайн-магазинів, блогів, форумів, систем управління контентом та інших веб-додатків.

Отже, серед переваг можна виділити:

- відкритий код. MySQL є вільним програмним забезпеченням, що означає, що ви можете використовувати, модифікувати і розповсюджувати його безкоштовно;
- швидкодія. MySQL володіє високою швидкістю, що дозволяє ефективно обробляти великі обсяги даних та високі навантаження;
- надійність. MySQL відомий своєю стабільністю і надійністю. Вона має добре розроблену архітектуру, яка дозволяє запобігати втраті даних та забезпечувати високу доступність;
- підтримка комунітетом. Є велика активна спільнота користувачів та розробників MySQL, яка забезпечує широкий спектр документації, підтримки та додаткових ресурсів.

З недоліків зазначають:

- відсутність деяких функцій. Деякі додаткові функції, як є і у аналогічних СКБД, таких як PostgreSQL, відсутні у MySQL, що можуть вимагати додаткових витрат часу та ресурсів для налаштування;
- проблеми з безпекою. Як і в будь-якій іншій СКБД, безпека даних є важливою проблемою для MySQL. Недбале конфігурування та управління може призвести до потенційних проблем з безпекою.

Незважаючи на переваги, MySQL є проста реляційна база даних, яку рекомендується використовувати в простих і нескладних продуктах. Для розробки системи вона не підходить через обмежений функціонал і складність конфігурації.

1.4.3 PostgreSQL

PostgreSQL - це потужна об'єктно-реляційна система управління базами даних (СКБД), яка відкрита для використання та розвитку. Вона відома своєю надійністю, розширюваністю та можливостями у роботі з даними. Багато великих компаній та проєктів використовують PostgreSQL для зберігання та обробки своїх

даних через його високу продуктивність та гнучкість. Для використання необхідно підняти і налаштувати сервер PostgreSQL, а для виконання запитів можна скористатися стандартним консольним інтерфейсом або будь-яким найпопулярнішим додатком pgAdmin, або будь-яким іншим графічним інтерфейсом [14].

Основні переваги PostgreSQL:

- розширений функціонал. PostgreSQL підтримує багато розширених функцій, таких як геоданих, JSON-даних, текстовий пошук, які дозволяють зберігати та обробляти різноманітні типи даних [15];
- надійність. PostgreSQL відомий своєю стабільністю та надійністю. Він має вбудовані механізми, які забезпечують цілісність даних та відновлення після відмов;
- спільнота. Є активна та дружня спільнота користувачів та розробників PostgreSQL, що робить доступними різноманітні ресурси для підтримки та вдосконалення;
- відкритий код. Це дозволяє користувачам перевіряти код, робити внески у розвиток та адаптувати PostgreSQL до своїх потреб.

Недоліки PostgreSQL:

- високі вимоги до ресурсів. PostgreSQL може бути більш вимогливим до ресурсів порівняно з деякими іншими системами управління базами даних, особливо при великих обсягах даних;
- менша популярність. Хоча PostgreSQL має велику та віддану спільноту, він все ще менш популярний порівняно з деякими іншими СКБД, такими як MySQL або Microsoft SQL Server;
- швидкість. У деяких випадках швидкодія PostgreSQL може бути менш ефективною порівняно з іншими СКБД, хоча це залежить від конкретного апаратного забезпечення та оптимізації.

Беручи до уваги всі переваги та недоліки, PostgreSQL становить оптимальний вибір у якості реляційної бази даних для розробки продукту за рахунок

різноманіття функціоналу і простоту у використанні. Крім того, PostgreSQL легко масштабується, що може бути важливим при високому навантаженні на систему.

1.4.4 Redis

Redis - це динамічна система кешування та база даних ключ-значення з відкритим вихідним кодом, яка використовується для зберігання даних у форматі ключ-значення у швидкодіючій пам'яті. Вона підтримує різні типи даних, такі як рядки, хеші, списки, набори та сортовані набори, і надає широкі можливості для операцій з цими даними, такі як додавання, оновлення, видалення та пошук. Redis також відомий своєю високою продуктивністю та масштабованістю, що робить його популярним в багатьох сценаріях, таких як кешування, сесійне зберігання, реалізацію черг повідомлень та розподілених блокувань [16].

Переваги Redis такі:

- швидкість. Redis зберігає дані у пам'яті, що робить його надзвичайно швидким для доступу до даних, особливо для операцій читання та запису;
- простота використання. Redis має простий і легкий у використанні інтерфейс, що дозволяє швидко виконувати операції з даними;
- можливості. Можна вирішувати багато задач, зокрема кешування, брокер повідомлень, тощо.

До недоліки можна віднести:

- втрата даних при відмові. У випадку відмови сервера Redis, дані, які не були збережені на диск, можуть бути втрачені;
- відсутність складних операцій. Redis має обмежені можливості для виконання складних операцій, таких як додавання даних або виконання складних запитів;
- відсутність підтримки на деяких ОС. Redis не має офіційної підтримки для Windows систем.

Redis гарно підходить у якості реалізації механізмів, пов'язаних із тимчасового зберігання простих даних у форматі ключ-значення, таких як кешування даних запитів, ідемпотентності та інше. Використання Redis буде визначатися завданнями, які передбачає розроблення системи.

1.5 Вибір прокси-серверу

NGINX - це веб-сервер та проксі-сервер з відкритим вихідним кодом, який використовується для обробки HTTP-запитів та виконання різних функцій веб-сервера. Він відомий своєю високою продуктивністю, ефективністю та масштабованістю, що робить його популярним використанням у великих веб-проектах та облачних середовищах [17].

Основні функції NGINX включають обробку статичного та динамічного контенту, здійснення зворотного проксі-сервера, обробку навантаження, балансування навантаження та кешування веб-сторінок.

Він може виконувати роль веб-сервера для обслуговування веб-сайтів, а також функціонувати як проксі-сервер для передачі запитів на додаткові сервери, які обробляють більш важкі завдання, такі як додаткові додатки або бази даних.

NGINX має один головний і кілька робочих процесів. Традиційні веб-сервери використовують окремий потік для кожного запиту, у той час як NGINX обробляє декілька запитів в рамках одного робочого процесу, який розділяється на менші блоки, називані робочими сполуками.

Ці блоки відповідають за обробку потоків запитів, які надсилаються до робочого процесу через робочі з'єднання. Завершальним етапом є надання результатів цих запитів основним процесом.

Завдяки невеликій кількості повноважних потоків обробки, які розподіляються (зазвичай по одному на ядро), досягається ефективне використання пам'яті та ресурсів процесора, оскільки уникнуті затрати на перемикання контексту між потоками.

Кожне з'єднання може обробляти багато подібних запитів, що робить NGINX ідеальним вибором для високовідвідуваних веб-сайтів.

1.6 Висновки з вибору технологій

Отже, для створення комп'ютерної платформи онлайн були вибрані наступні серверні технології: Node.js (NestJS), PostgreSQL, Redis. Node.js (NestJS) забезпечує зручність у використанні, так як за допомогою нього можна використовувати JavaScript на сервері, що виключає необхідність вивчення інших мов програмування. Разом із Node.JS буде використовуватися фреймворк NestJS, який поєднує елементи ООП (об'єктно-орієнтоване програмування), ФП (функціональне програмування), ФПП (функціональне реактивне програмування), а також дає можливість реалізовувати проекти з сучасними архітектурними підходами. Це додає зручність і масштабільність у реалізації проектів.

Для створення користувацького інтерфейсу обрано мови: HTML(Pug), CSS(SCSS), JavaScript. Препроцесори полегшують роботу у написанні HTML/CSS коду, дозволяють використовувати змінні, вкладеність, міксіни, спрощують створення складних стилів і зменшують кількість помилок. Взаємодія з цими інструментами, допоможе гнучкіше й ефективніше розробляти веб-додатки.

У якості веб-серверу вибрано NGINX завдяки швидкості обробки запитів, зручності у використанні s може виконувати балансування навантаження між кількома серверами.

2 МОДЕЛЮВАННЯ КОМП'ЮТЕРНОЇ СИСТЕМИ

2.1 Загальні вимоги до комп'ютерної системи

Мета реалізації системи полягає у наданні зручної платформи для розміщення онлайн реклами з для купівлі або продажу товарів. Система надає послуги з безпеки, налагодженням комунікації між покупцями і продавцями. Отже, система повинна реалізувати наступний функціонал:

- інтуїтивно зрозумілий інтерфейс користувача;
- система реєстрації, автентифікації і авторизації з використанням JWT access і refresh токенів;
- CRUD операції над оголошеннями та іншими сутностями;
- можливість редагувати профіль аккаунтів, додавати додаткову інформацію;
- зручний спосіб пошуку онлайн оголошень за різними фільтрами і з курсовою пагінацією;
- функціонал з можливістю ставити відгуки на користувача і відображати цю інформацію.
- захист персональних даних користувачів є пріоритетом; система повинна використовувати сучасні методи шифрування та автентифікації для захисту інформації;
- система повинна бути масштабована, тобто новий функціонал повинен додаватись без труднощів;
- можливість деяким користувачам, які мають на це повноваження, видаляти небажанні оголошення;
- окрема сторінка для адміністраторів, які можуть додавати нові категорії, розміщувати новини, маніпулювати іншими даними, або блокувати доступ до деяких користувачів.

2.2 Функціональні можливості користувачів

В розроблюваній системі є кілька ролей, які визначають ти чи інші окремі функціональні можливості і перевірятимуться під час запитів до серверу. У системі передбачено 3 типи користувачів:

- гість (неzareєстрований користувач);
- zareєстрований користувач;

- модератор;
- адміністратор.

Кожен з цих типів користувачів має свої повноваження та список функцій, якими вони можуть користуватися. Гість є незареєстрованим користувачем з найменшим обсягом повноважень, як показано на рисунку 2.1.

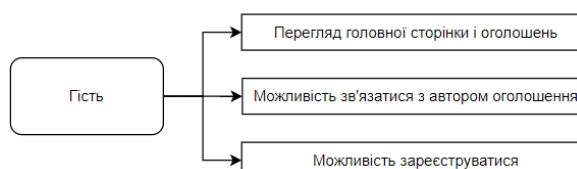


Рисунок 2.1 – Функціонал з роллю “Гість”

Дозволений функціонал наведено на рисунку 2.2.



Рисунок 2.2 – Функціонал з роллю “Користувач”

Модератор має трішки більше повноважень, ніж звичайний користувач. Зокрема, він може видаляти небажані оголошення, якщо такі підлягають під критерії небажаних згідно з законодавством країни, міжнародним правом або політикою компанії. Функціонал модератора зображений на рисунку 2.3.



Рисунок 2.3 – Функціонал з роллю “Модератор”

Адміністратор володіє набагато більшими повноваженнями порівняно з іншими ролями і має можливість виконувати практично всі дії в системі. Детальний перелік повноважень адміністратора наведено на рисунку 2.4.

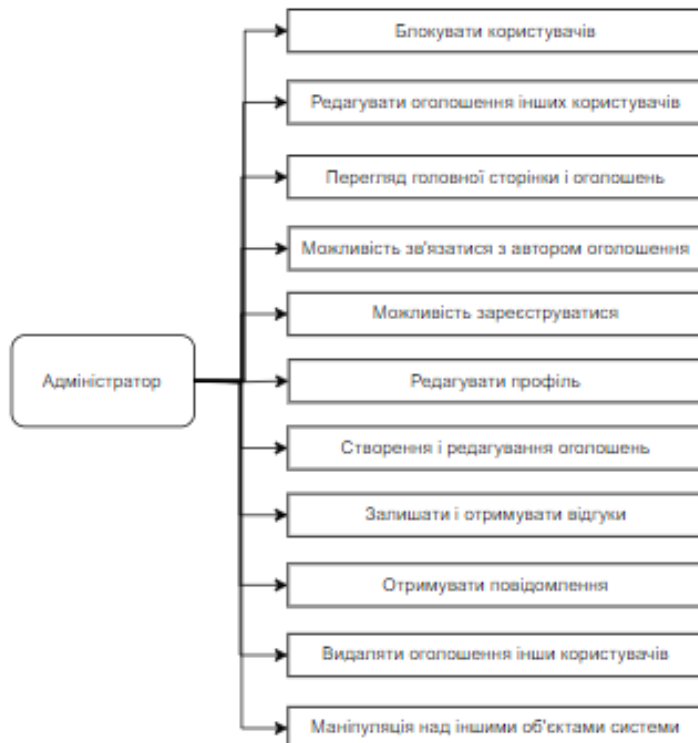


Рисунок 2.4 – Функціонал з роллю “Адміністратор”

2.3 Функціонал системи

Для будь якого типу користувачів система повинна надавати спектр можливостей, серед яких можна виділити:

- пошук оголошень: користувачі повинні мати можливість шукати оголошення за різними критеріями (назва, категорія, ціна, тощо);
- створення і редагування оголошень: система повинна надавати окрему сторінку для маніпуляцій над оголошенням: поле з назвою, описом, можливістю завантаження фотографій, тощо;
- реєстрація, авторизація та автентифікація: користувачі повинні мати можливість реєструватися та входити до системи для доступу до додаткових функцій;
- додавання оголошень: авторизовані користувачі повинні мати можливість додавати оголошення і керувати ними;
- система оцінювання: користувачі повинні мати можливість залишати відгуки на інших користувачів і переглядати відгуки на себе.

2.4 Опис таблиць БД

Для збереження інформації була обрана реляційна база даних PostgreSQL, в якій потрібно чітко сформулювати структуру бази. Як вже зазначалося, основними компонентами реляційних баз даних є таблиці, які використовуються для організації та зберігання даних. Визначивши основні функціональні вимоги та дані, що будуть зберігатися в базі даних, були ідентифіковані сутності проектованої системи.

Сутність представляє собою сукупність об'єктів, відомих як екземпляри. Кожен екземпляр сутності має набір характеристик, які в логічній моделі бази

даних називаються атрибутами сутності. Перший етап процесу проектування бази даних називається концептуальним проектуванням, і він включає створення концептуальної моделі даних предметної області. Ця модель створюється на основі функціональних вимог користувачів.

Для бази даних розроблювальної платформи визначені 6 сутностей:

- користувач;
- категорія;
- оголошення;
- файл. Збереження інформації про завантаженні зображення чи відео;
- оцінка. Оцінка користувача, відгук;
- refresh токен. Зберігання refresh токenu користувача у захищеному форматі, для оновлення access токenu, що має коротший термін дії.

Кожна сутність має первинний ключ, який однозначно характеризує кожен екземпляр сутності, а також додаткові поля, що містять інформацію про дату створення сутності, дату її останнього оновлення і дату видалення. У системі нічого реально не видаляється, а лише помічається як видалене. Це забезпечує більший контроль над даними, що також дозволяє можливість відновити випадково видалені сутності, наприклад, оголошення.

Інформація про таблиці бази даних представлена нижче у вигляді таблиць (табл. 2.1 – 2.5).

Таблиця 2.1 – Користувач

Поле	Тип даних	Значення за умовчанням	Обов'язковість	Примітки
id	uuid	uuid_generate_v4()	Так	РК
created_at	timestamp	timestamp(6) with time zone	Так	-
updated_at	timestamp	timestamp(6) with time zone	Так	-
deleted_at	timestamp	-	Hi	-

Продовження таблиці 2.1

email	varchar	-	Так	UQ
full_name	varchar	-	Так	-
phone	varchar	-	Hi	-
role	users_role_enum	USER	Hi	-
password	varchar	-	Так	-
avatar	varchar	-	Hi	-
refresh_token_id	uuid	-	Hi	FK

Тип users_role_enum має наступні значення: USER, MODERATOR, ADMIN.

Таблиця 2.2 – Оголошення

Поле	Тип даних	Значення за умовчанням	Обов'язковість	Примітки
id	uuid	uuid_generate_v4()	Так	PK
created_at	timestamp	timestamp(6) with time zone	Так	-
updated_at	timestamp	timestamp(6) with time zone	Так	-
deleted_at	timestamp	-	Hi	-
name	varchar	-	Так	-
description	varchar	-	Так	-
price	text	0	Hi	-
price_type	adverts_price_type_enum	FREE	Hi	-

Продовження таблиці 2.2

is_used	boolean	-	Ні	-
images	varchar[]	{}	Ні	-
category_id	uuid	-	Так	FK
user_id	uuid	0	Так	FK

Тип adverts_price_type_enum має наступні значення: 'FREE, BARTER, MONEY.

Таблиця 2.3 – Категорія

Поле	Тип даних	Значення за умовчанням	Обов'язковість	Примітки
id	uuid	uuid_generate_v4()	Так	PK
created_at	timestamp	timestamp(6) with time zone	Так	-
updated_at	timestamp	timestamp(6) with time zone	Так	-
deleted_at	timestamp	-	Ні	-
name	varchar	-	Так	-
subcategories	varchar[]	-	Так	-

Таблиця 2.4 – Refresh токен

Поле	Тип даних	Значення за умовчанням	Обов'язковість	Примітки
id	uuid	uuid_generate_v4()	Так	PK
created_at	timestamp	timestamp(6) with time zone	Так	-

Продовження таблиці 2.4

updated_at	timestamp	timestamp(6) with time zone	Так	-
deleted_at	timestamp	-	Ні	-
token	varchar	-	Так	-
expires_in	varchar	-	Так	-
payload	varchar	-	Так	-

Таблиця 2.5 – Файл

Поле	Тип даних	Значення за умовчанням	Обов'язковість	Примітки
id	uuid	uuid_generate_v4()	Так	PK
created_at	timestamp	timestamp(6) with time zone	Так	-
updated_at	timestamp	timestamp(6) with time zone	Так	-
deleted_at	timestamp	-	Ні	-
original_name	varchar	-	Так	-
path	varchar	-	Так	-
mimetype	varchar	-	Так	-
owner_id	uuid	-	Так	FK
filename	varchar	-	Так	-
size	varchar	-	Так	-

Для генерації унікальних ідентифікаторів використовується функція `uuid_generate_v4()`, що створює UUID (англ. *universally unique identifier*) для кожного нового запису, що дозволяє виділяти кожен запис як унікальний.

Відношення таблиць бази даних зображено на рисунку 2.5.

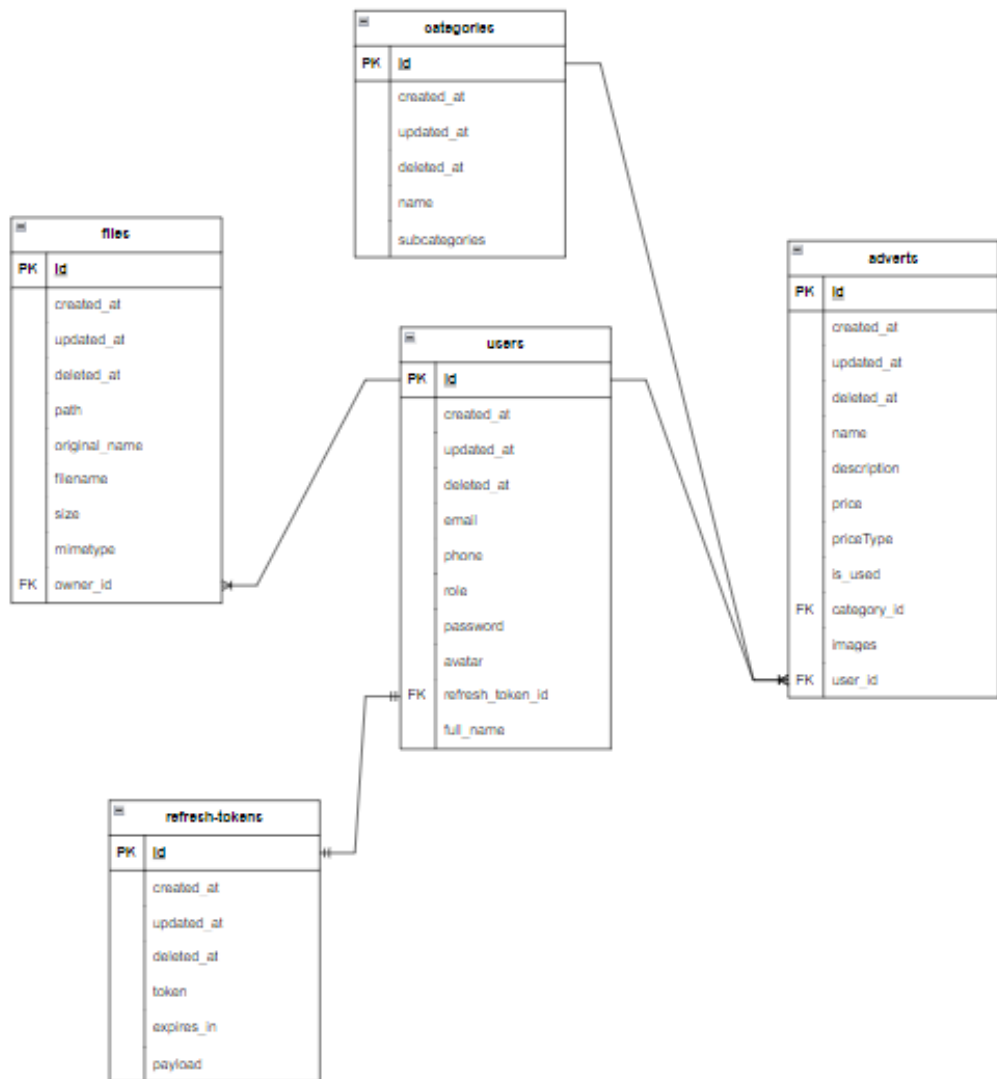


Рисунок 2.5 – Відношення таблиць бази даних

2.5 Діаграма послідовностей

Діаграма послідовностей (Sequence Diagram) показує порядок взаємодії між об'єктами системи в процесі виконання певних функцій.

До створеної вище діаграми використань створено діаграму послідовності у випадку автентифікації користувача на сайті. Дану діаграму зображено на рисунку 2.6.

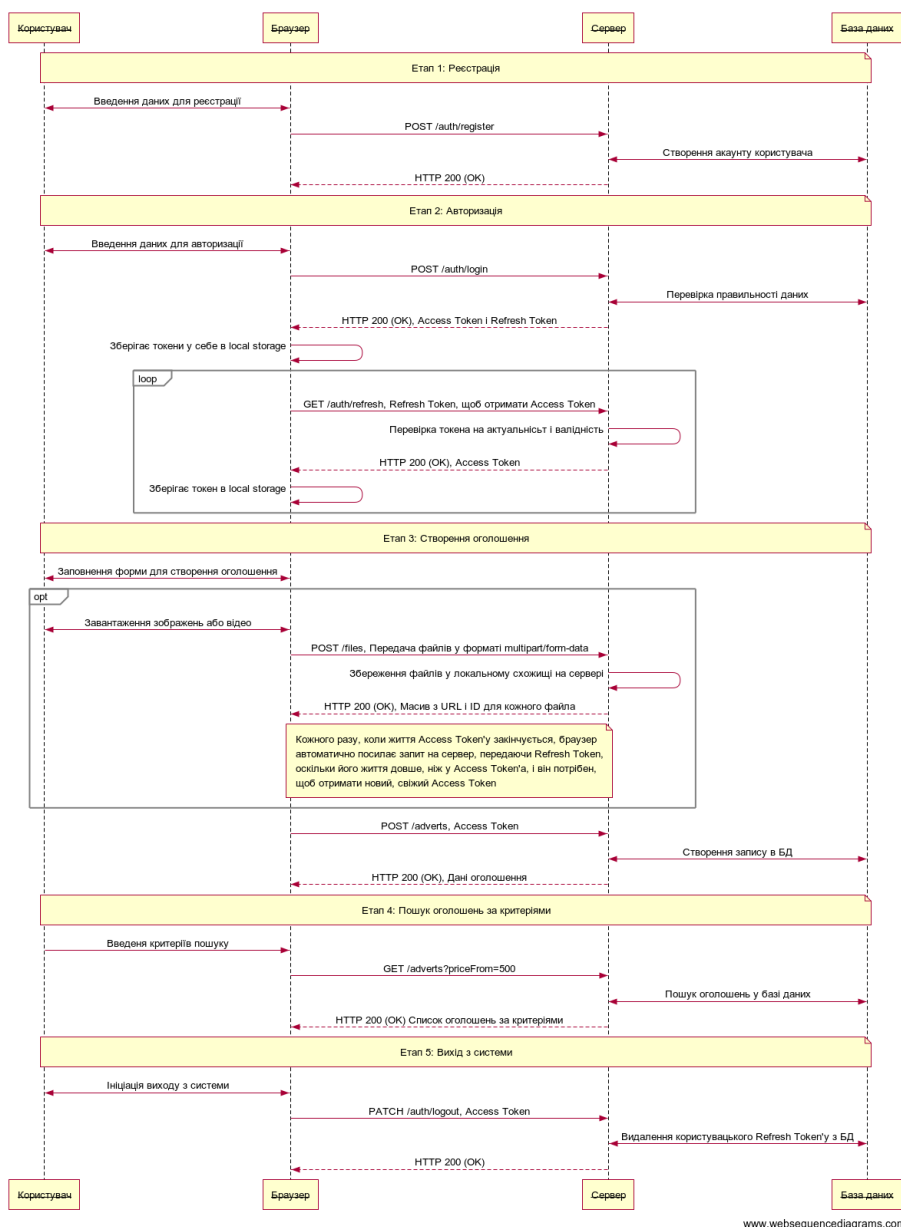


Рисунок 2.6 – UML діаграма послідовності користування системою

Процес включає в себе наступні базові етапи користування системою:

- реєстрація. Користувач вводить необхідні дані, браузер відправляє запит POST, щоб зареєструвати нового користувача;
- авторизація. Після успішної реєстрації, користувач повинен авторизуватись і ввести свої дані, щоб увійти в систему. Браузер надсилає запит POST, сервер обробляє ці дані і проводить свою внутрішню перевірку, після чого віддає два токени: Access Token і Refresh Token. Браузер зберігає ці токени. Кожного разу браузер новий Access Token, використовуючи Refresh Token, адже

життя Access Token'у коротке, тому для комфортного користування платформою, браузер автоматично повинен оновляти токен доступу;

- створення оголошень. Користувач заповнює форму, за бажанням може додати кілька файлів, для цього він завантажує їх, браузер передає серверу масив файлів, сервер зберігає їх і віддає посилання на них. Потім браузер надсилає POST запит передаючи дані для створення оголошення і сервер зберігає їх;

- пошук оголошень за критеріями;

- вихід з системи. Браузер посилає запит на сервер, той видаляє Refresh Token, після чого браузер видаляє в себе збережені токени, так що перебування в системі можливе тільки через етап авторизації.

Для пошуку і відображень списку оголошення використовується курсова пагінація, головно перевагою якої є стійкість до зміщення рядків. Наприклад, якщо запис видалено, наступний запис, який слідував би за ним, все одно відображається, оскільки запит обробляє курсор, а не конкретне зміщення. На відміну від стратегії offset-based pagination, якщо елемент із попередньої сторінки видаляється під час розбиття сторінок клієнтом, усі наступні результати будуть зміщені на одиницю вперед, що означає, що користувач може пропустити контент.

Цей процес забезпечує високий рівень безпеки та зручності для користувачів, дозволяючи їм швидко реєструватися, авторизуватись, створювати і шукати оголошення, а також безпечно виходити з системи, не турбуючись про збережені токени.

Отже, як наслідок маємо систему з базовими CRUD операціями над сутностями, автентифікацією з використанням пар JWT токенів, що, як вже зазначалось вище, є доволі популярним і надійним методом автентифікації користувачів, пошуковим механізмом із курсовою пагінацією, щоб уникнути проблем з розбиттям сторінок, а також механізмом завантаження файлів, їх фільтрація і обробка.

3 РОЗРОБКА СИСТЕМИ

Архітектура середовища розробки грає ключову роль, як і реалізація бази даних, розробка користувацького інтерфейсу. В якості основного фреймворка для розробки серверно ї частини обрано NestJS, який завдяки своїй модульній архітектурі і підтримки TypeScript робить його для створення ефективних, надійних та масштабованих серверних додатків.

3.1 Файлова структура проекту

Старт проекту на NestJS можна почати з команди “npm init”, щоб згенерувати файл package.json, який буде містити інформацію про всі сторонні модулі проекту, від яких залежить сам проект.

У корні проекту повинен бути створений файли конфігурацій проекту і директорія “src/”, де і буде знаходитись весь головний код розробки, який буде скомпільований у папку “dist/”.

Зокрема, знадобляться також наступні файли:

- README.md: файл з документацією проекту;
- tsconfig.json: файл конфігурації для мови typescript;
- .env: файл із змінними середовища проекту;
- .eslintrc.js: файл з правилами, як має виглядати код. Це потрібно для комфортної розробки, а також поліпшення читання коду іншими розробниками.

Повинні бути встановленні наступні залежності:

- @nestjs/common, @nestjs/config, @nestjs/core, @nestjs/platform-express, rxjs, express-rate-limit: для роботи nestjs;
- @nestjs/jwt, @nestjs/passport, passport-jwt, request-context: для механізму авторизації/автентфікації;

- class-transformer, class-validator: для валідації полів;
- @nestjs/typeorm, reflect-metadata, typeorm, pg, typeorm-naming-strategies: для бази даних;
- fs-capacitor, busboy, transliteration, multer: для механізму завантаження файлів;
- bcrypt, bignumber.js, busboy, dotenv, helmet, lodash: для інших потреб, які потребують розробку.

Директорія src:

- src/main.ts: Це файл entry point. Він містить код, що запускає весь проект;
- src/app.module.ts: Головний модуль проекту, що об'єднує всі дочірні модулі;
- src/config: Зберігає файли конфігурацій для інших модулів;
- src/common: Зберігає глобальні файли, наприклад: декоратори, сутності, інтерфейси, які можуть бути використані іншими модулями;
- src/migrations: Зберігає файли, що містять скрипти міграцій для БД;
- src/users: Модуль, що займається обробкою даних користувачів;
- src/auth: Модуль, що містить механізм для реєстрації, авторизації і автентифікації;
- src/files: Модуль, що займається обробкою файлів;
- src/adverts: Модуль, що займається обробкою оголошень;
- src/categories: Модуль, що займається обробкою категорій;
- src/votes: Модуль, що займається обробкою відгуків.

Кожен модуль імпортує в собі безліч залежних компонентів. Серед яких є:

- контролер: відповідає за обробку HTTP-запитів і повернення відповідей клієнту. Контролери є важливою частиною архітектури NestJS, оскільки вони дозволяють організувати маршрутизацію і логіку обробки запитів у зручний і зрозумілий спосіб;
- провайдери: це основний концепт, який відповідає за ін'єкцію залежностей. Провайдери використовуються для створення та управління

сервісами, репозиторіями, фабриками та іншими об'єктами, що можуть мати залежності, які потрібно автоматично впроваджувати в інші класи.

Модулі можуть також імпортувати інші модулі (провайдери яких стануть залежними для того модуля), а також експортувати свої провайдери для інших модулів.

3.2 Ініціалізація розробки

Приклад коду `main.ts`, що запускає проект зазначений у лістингу 3.1.

Лістинг 3.1 – `main.ts`, створення робочих процесів

```
import 'dotenv/config'

import { NestFactory } from '@nestjs/core'
import { ValidationPipe, Logger } from '@nestjs/common'
import { ConfigService } from '@nestjs/config'
import RateLimit from 'express-rate-limit'
import helmet from 'helmet'

import { AppModule } from 'src/app.module'
import { Environment } from '../config/env.config'

const logger = new Logger('Main')

const DEFAULT_HOST = 'localhost'
const DEFAULT_PORT = 3000

async function bootstrap(): Promise<void> {
  const app = await NestFactory.create(AppModule)

  app.enableCors()
  app.use(helmet())
  app.use(RateLimit({ windowMs: 15 * 60 * 1000, max: 200 }))

  app.useGlobalPipes(
    new ValidationPipe({
      transform: true,
    }),
  )

  const configService: ConfigService<Environment> =
    app.get(ConfigService)
```

```

const host = configService.get('HOST', DEFAULT_HOST)
const port = +configService.get('PORT', DEFAULT_PORT)

await app.listen(port, host, () =>
  logger.log(`Listening on ${host}:${port}`),
)
}

void bootstrap()

```

Як вже було зазначено вище, головний модуль проекту, що об'єднує всі дочірні модулі називається `app.module.ts`. Код зазначений в лістингу 3.2.

Лістинг 3.2 – `app.module.ts`, головний модуль проекту

```

import { MiddlewareConsumer, Module, NestModule } from
 '@nestjs/common'
import { ConfigModule } from '@nestjs/config'
import { TypeOrmModule } from '@nestjs/typeorm'
const requestContext = require('request-context')

import { envConfig } from './config/env.config'
import { typeOrmConfig } from './config/typeorm.config'
import { CategoryModule } from './categories/category.module'
import { UserModule } from './users/user.module'
import { AdvertModule } from './adverts/advert.module'
import { AuthModule } from './auth/auth.module'
import { FilesModule } from './files/files.module'

@Module({
  imports: [
    ConfigModule.forRoot(envConfig),
    TypeOrmModule.forRootAsync(typeOrmConfig),
    UserModule,
    CategoryModule,
    AdvertModule,
    FilesModule,
    AuthModule,
  ],
})
export class AppModule implements NestModule {
  configure(consumer: MiddlewareConsumer): MiddlewareConsumer |
  void {

    consumer.apply(requestContext.middleware('request')).forRoutes('*')
  }
}

```

Бібліотека request-context використовується для підключення проміжного програмного забезпечення для доступу до даних у контексті запиту. Необхідно «загорнути» обробку запитів у домен, щоб отримувати доступ до даних лише для поточного життєвого циклу запиту. Усі наступні функції виконуватимуться у створеному «простір імен».

3.2.1 Конфігурація бази даних

Для забезпечення ефективної роботи з базою даних, було обрано ORM TypeORM яка є найсучаснішою ORM для середовища Node.JS. Використання TypeORM має кілька вагомих переваг, особливо якщо ви працюєте з TypeScript або JavaScript і взаємодієте з реляційними базами даних. Наступні налаштування у лістингу 3.3 покращують процес розробки.

Лістинг 3.3 – Конфігурація для підключення до БД

```
import { ConfigModule, ConfigService } from '@nestjs/config'
import {
  TypeOrmModuleOptions,
  TypeOrmModuleAsyncOptions,
} from '@nestjs/typeorm'

import { Environment } from './env.config'
import { UserSubscriber } from
'../users/subscribers/user.subscriber'
import { SnakeNamingStrategy } from 'typeorm-naming-strategies'
import { NodeEnvironment } from '../common/enums/node-
environment.enum'
import { RefreshTokenSubscriber } from '../auth/subscribers/refresh-
token.subscriber'

const DEFAULT_PSQL_HOST = 'localhost'
const DEFAULT_PSQL_PORT = 5432

export const typeOrmConfig: TypeOrmModuleAsyncOptions = {
  imports: [ConfigModule],
  inject: [ConfigService],
  useFactory: (
    configService: ConfigService<Environment>,
  ): TypeOrmModuleOptions => ({
    type: 'postgres',
    entities: [`${__dirname}/../**/*.entity{.ts,.js}`],
    database: configService.get<string>('PSQL_DATABASE'),
    host: configService.get('PSQL_HOST') || DEFAULT_PSQL_HOST,
    port: +configService.get('PSQL_PORT') || DEFAULT_PSQL_PORT,
    username: configService.get('PSQL_USERNAME'),
```

```

    password: configService.get('PSQL_PASSWORD'),
    namingStrategy: new SnakeNamingStrategy(),
    synchronize:
        configService.get('NODE_ENV') ===
NodeEnvironment.DEVELOPMENT,
    logging: false,
    subscribers: [UserSubscriber, RefreshTokenSubscriber],
  )),
}

```

Наступні властивості конфігурації мають намір полегшати розробку:

- “namingStrategy”: усі сутності ORM, що мають імена властивостей не у snake_case, згідно з конвенцією іменування баз даних, то вони будуть автоматично створюватися у базі у snake_case;
- “synchronize”: дозволяє синхронізувати сутності ORM з сутностями у базі даних. Це означає, що всі властивості, що знає ORM, перейдуть до сутностей у базі даних. Це не є безпечним, якщо ви запускаєте проект не у режимі розробки;
- “subscribers”: використовується для підписки на події над сутностями ORM, до яких «прив’язаний» subscriber. Наприклад, замість того, щоб кожного разу хешувати пароль, можна створити підписку на подію збереження чи оновлення даних користувача, і коли подія відбувається, автоматично хешувати пароль, якщо він новий.

Приклад subscriber’у визначенню у лістингу 3.4.

Лістинг 3.4 – Subscriber для ORM моделі «UserEntity»

```

import {
    EntitySubscriberInterface,
    EventSubscriber,
    InsertEvent,
    UpdateEvent,
} from 'typeorm'
import { UserEntity } from '../entities/user.entity'

export const generateHash = (password: string): string =>
    bcrypt.hashSync(password, 10)

@EntitySubscriber()
export class UserSubscriber implements
EntitySubscriberInterface<UserEntity> {
    listenTo() {

```

```

        return UserEntity
    }

    beforeInsert(ev: InsertEvent<UserEntity>): void {
        if (ev.entity.password) {
            ev.entity.password = generateHash(ev.entity.password)
        }
    }

    beforeUpdate(ev: UpdateEvent<UserEntity>): void {
        if (ev.entity?.password !== ev.databaseEntity?.password) {
            ev.entity.password = generateHash(ev.entity.password)
        }
    }
}

```

Усі сутності ORM знаходяться у файлах з назвою *.entity.ts і можуть знаходитись у будь якій папці проекту. Ці сутності описують модель реальної таблиці в базі даних.

Одна з важливих тем є міграції. Міграції в TypeORM — це механізм для управління змінами в схемі бази даних. Вони дозволяють автоматизувати процес створення, зміни і видалення таблиць та їхніх колонок. Міграції допомагають забезпечити синхронізацію схеми бази даних з моделями вашого додатку, що особливо важливо при роботі в команді або при розгортанні на різних середовищах. Для роботи з міграціями використовується typeorm-cli пакет. Наступні скрипти у package.json полегшать роботу з міграціями, зазначені у лістингу 3.5.

Лістинг 3.5 – Скрипти package.json для роботи з міграціями

```

"typeorm:cli": "node --require ts-node/register --require tsconfig-
paths/register ./node_modules/typeorm/cli.js --dataSource
src/config/orm-cli.config.ts",
"migration:create": "npm run typeorm:cli -- migration:create",
"migration:generate": "npm run typeorm:cli -- migration:generate -p
true",
"migration:fake": "npm run typeorm:cli -- migration:run --fake",
"migration:revert": "npm run typeorm:cli -- migration:revert",
"migration:run": "npm run typeorm:cli -- migration:run -t=each"

```

Наприклад, для того, щоб сгенерувати скрипт міграції на основі моделей сутностей ORM, необхідно виконати наступну команду: `npm run`

migration:generate src/migrations/init.

3.3 Імплементація модуля авторизації

Модуль авторизації є найголовнішим модулем системи, оскільки саме він відповідає за безпеку. Забезпечення належного рівня безпеки є критичним для будь-якої системи, особливо в контексті захисту конфіденційної інформації та запобігання несанкціонованому доступу.

Основні завдання модуля авторизації включають ідентифікацію користувачів, управління доступом, захист даних, моніторинг та логування. Процес автентифікації дозволяє системі перевірити особу користувача за допомогою різних методів, таких як паролі, токени або біометричні дані. Важливо забезпечити надійний процес автентифікації, щоб запобігти доступу зловмисників до системи. Після успішної автентифікації модуль авторизації визначає, до яких ресурсів і дій користувач має доступ. У даній платформі для механізму авторизації і автентифікації використовується пара токенів JWT: Access Token і Refresh Token.

JWT (JSON Web Token) є стандартом для безпечного обміну інформацією між двома сторонами. У контексті авторизації, JWT часто використовується для створення токенів доступу (Access Token) та токенів оновлення (Refresh Token). Обидва типи токенів відіграють важливу роль у забезпеченні безпеки та зручності користувачів при взаємодії з системою.

Access Token — це короткоживучий токен, який використовується для автентифікації запитів до серверу. Він містить інформацію про користувача та його права доступу, що дозволяє серверу швидко перевірити легітимність запиту без необхідності постійно звертатися до бази даних.

Refresh Token — це довгоживучий токен, який використовується для отримання нового Access Token після того, як старий токен втратив свою дійсність. Це дозволяє користувачам залишатися авторизованими протягом тривалого часу

без необхідності повторної автентифікації.

Задля покращення і ефективності процесу розробки було прийнято рішення використати бібліотеки `@nestjs/jwt` і `@nestjs/passport`, які вже мають реалізований базовий функціонал. На зазначеному нижче лістингу 3.6 представлено код конфігурації модуля авторизації.

Лістинг 3.6 – Конфігурація модуля авторизації

```
import { forwardRef, Module } from '@nestjs/common'
import { PassportModule } from '@nestjs/passport'
import { TypeOrmModule } from '@nestjs/typeorm'
import { JwtModule } from '@nestjs/jwt'

import { AuthService } from '../services/auth.service'
import { UserModule } from '../users/user.module'
import { JwtStrategy } from '../strategies/jwt.strategy'
import { AuthController } from '../controllers/auth.controller'
import { RefreshTokenEntity } from '../entities/refresh-token.entity'

@Module({
  imports: [
    TypeOrmModule.forFeature([RefreshTokenEntity]),
    forwardRef(() => UserModule),
    PassportModule.register({ defaultStrategy: 'jwt' }),
    JwtModule,
  ],
  controllers: [AuthController],
  providers: [AuthService, JwtStrategy],
  exports: [AuthService],
})
export class AuthModule {}
```

Модуль імпортує контролер авторизації. В контроллері реалізовані методи реєстрації, авторизації, оновлення refresh токена і виходу із системи. Для імплементції авторизації за допомогою токенів необхідно спочатку перевірити наявність користувача в системі і перевірити правильність введеного паролю (лістинг 3.7). Оскільки, як вже було зазначено, зберігання паролю у початковому вигляді небезпечно, то у базі даних зберігається хеш паролю. Отже, задля перевірки правильності введеного паролю сервер повинен порівнювати не самі паролі, а їх хеши.

Лістинг 3.7 – Перевірка введеного паролю користувача порівняння хешей

```

async validateUser(dto: ValidateUserDto): Promise<UserEntity> {
  const { email, password } = dto
  const user = await this._userService.getUserByEmail(email)

  const isPasswordValid = await validateHash(password,
user.password)

  if (!isPasswordValid) {
    throw new BadRequestException('Password is incorrect')
  }

  return user
}

```

Далі створюються пара токенів Access Token і Refresh Token в яких зберігається ідентифікатор користувача в системі і його роль. Ці дані будуть використовуватися для ідентифікації користувача в системі, а також для RBAC (Role-Based Access Control) механізму, щоб обмежити користувацький доступ до деяких ресурсів відповідно ролі самого користувача. Код генерації пари токенів зазначений у лістингу 3.8.

Лістинг 3.8 – Генерація пари токенів

```

async createTokens(dto: CreateTokenDto): Promise<TokenPayloadDto> {
  const { userId, role } = dto

  const [accessToken, refreshToken] = await Promise.all([
    this._generateAccessToken({ userId, role }),
    this._generateRefreshToken({ userId, role }),
  ])

  return {
    accessToken,
    refreshToken,
  }
}

```

Потім метод контролеру повертає дані користувача (виключаючи пароль), ці дані перехоплює interceptor і зберігає їх у контексті поточного потоку запиту. В NestJS, інтерсептори (interceptors) - це спеціальні класи, які дозволяють перехоплювати та змінювати вхідні запити або вихідні відповіді. Інтерсептори можуть бути використані для виконання різних завдань, таких як логування,

трансформація даних, обробка помилок та багато іншого. Вони працюють подібно до middleware, але з більшою гнучкістю та інтеграцією в контексті конкретних обробників запитів. Приклад роботи інтерцептору зазначений у лістингу 3.9

Лістинг 3.9 – Інтерцептор авторизації

```
import {
  CallHandler,
  ExecutionContext,
  Injectable,
  NestInterceptor,
} from '@nestjs/common'
const requestContext = require('request-context')

import { Observable } from 'rxjs'
import { UserEntity } from '../../users/entities/user.entity'

@Injectable()
export class AuthInterceptor implements NestInterceptor {
  intercept(context: ExecutionContext, next: CallHandler):
  Observable<any> {
    const request = context.switchToHttp().getRequest()
    const user = <UserEntity>request.user

    requestContext.set('request.user_key', user)

    return next.handle()
  }
}
```

У подальшому ми зможемо отримувати об'єкт даних користувача безпосередньо з даних самого запиту. Важливо зазначити, що клієнтська сторона нічого не знає і не передає даних користувача через мережу, коли надсилає будь який інший запит. Бібліотека request-context зберігає дані користувача лише на серверній частині запиту.

Як вже було зазначено, Access Token – короткоживучий токен, отже він потребує постійного оновлення. Автентифікація проходить за допомогою бібліотеки passport, яка перевіряє дійсність і актуальність токена доступу, а потім наявність користувача в системі. Якщо якщо не існує, або токен вже не живий – викидується помилка автентифікації. Клієнт-браузер постійно передає Refresh Token (через умовний проміжок часу), щоб запросити новий токен доступу. Наступний код з лістингу 3.10 демонструє роботу оновлення токена.

Лістинг 3.10 – Оновлення Access Token'у

```

async refreshAccessToken(dto: RefreshAccessTokenDto):
Promise<string> {
    const { refreshToken } = dto

    const decoded = this._jwtService.decode(refreshToken) as {
        userId: string
        role: RoleEnum
    }

    if (!decoded) {
        throw new BadRequestException('Refresh token is corrupted')
    }

    const { userId, role } = decoded

    const savedRefreshToken = await
    this._findRefreshTokenByUserId(userId)

    if (!savedRefreshToken) {
        throw new NotFoundException('Refresh token is unknown to
server')
    }

    // tokens are saved in sha256 hash so we need to compare hashes
    const isTokenMatch =
        createSHA256Hash(refreshToken) === savedRefreshToken.token

    if (!isTokenMatch) {
        throw new BadRequestException('Refresh token mismatch')
    }

    try {
        await this._jwtService.verifyAsync(
            refreshToken,
            this._getRefreshTokenOptions(),
        )
    } catch (e) {
        throw new UnauthorizedException(
            'Refresh token is invalid or expired',
        )
    }

    return this._generateAccessToken({
        userId,
        role,
    })
}

```

Приклад роботи модуля продемонстровано на рисунках 3.1-3.5.

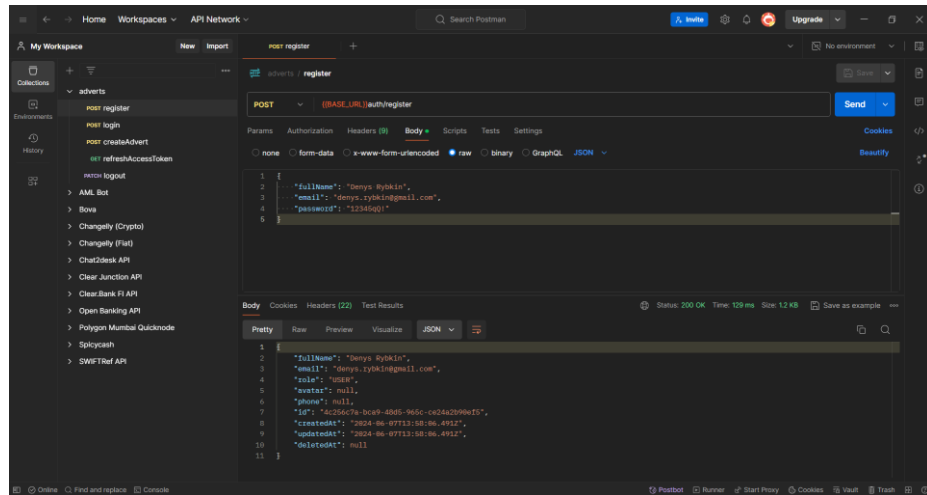


Рисунок 3.1 – Реєстрація користувача

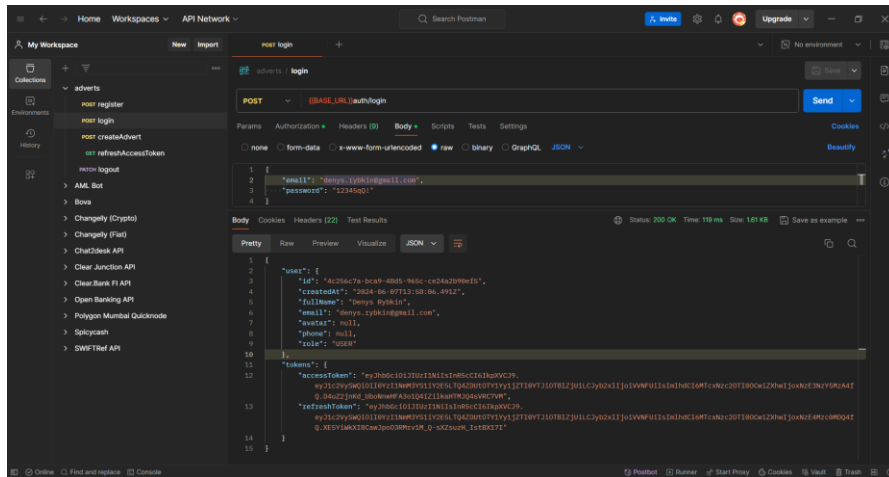


Рисунок 3.2 – Авторизація користувача

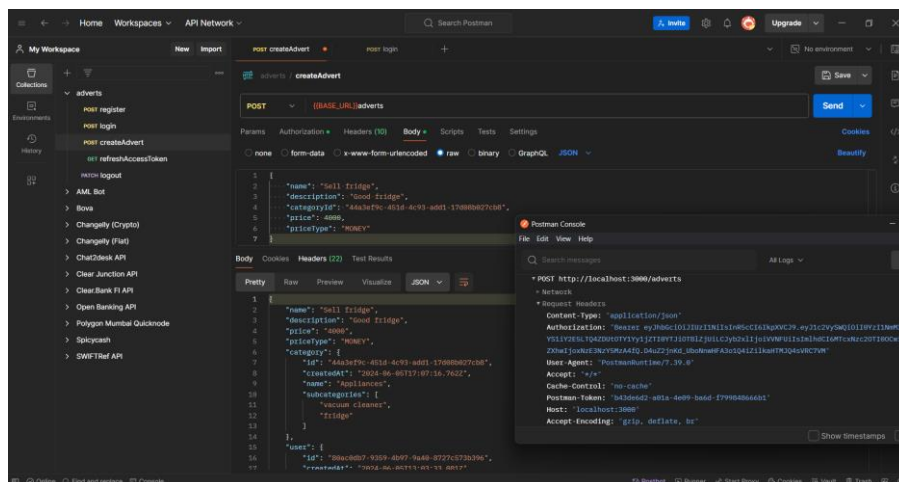


Рисунок 3.3 – Створення оголошення

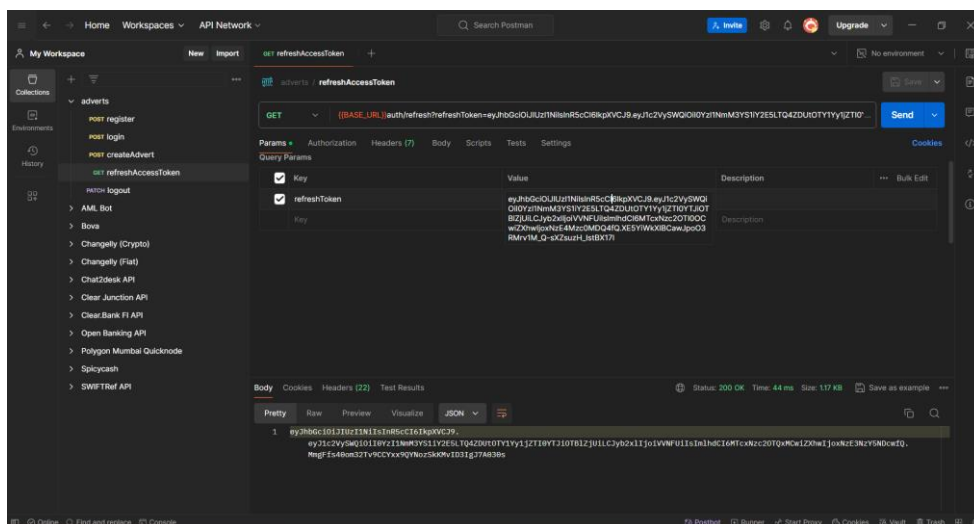


Рисунок 3.4 – Оновлення токена доступу

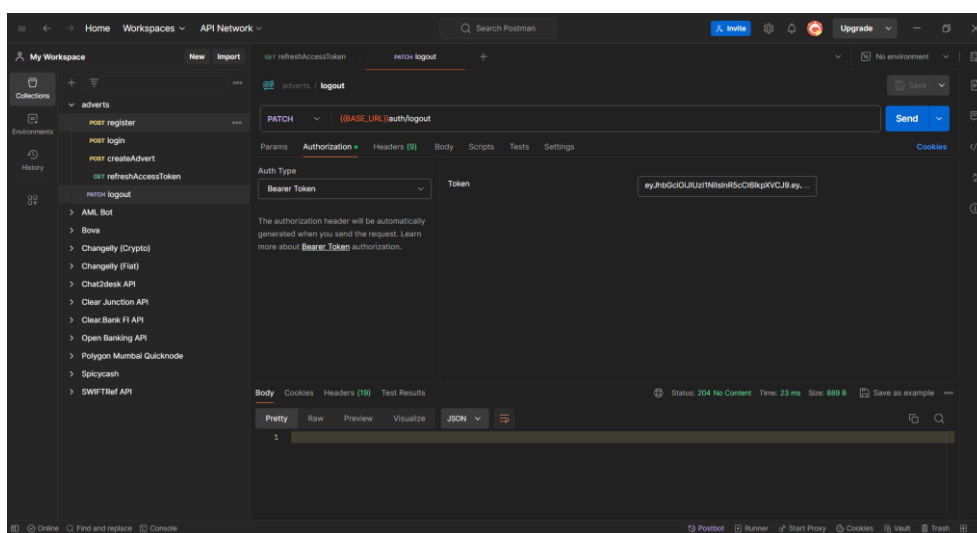


Рисунок 3.5 – Вихід із системи

3.4 Реалізація модуля файлів

Коли ви відправляєтесь файл на сервер, це відбувається у форматі `multipart/form-data`. Цей формат дозволяє передавати різні типи даних, такі як тексти, файли тощо, в одному запиті HTTP. Коли сервер отримує такий запит, він розбирає його, виділяючи файли та інші дані.

У NestJS процес завантаження файлів зазвичай виглядає наступним чином:

- створення контролера або модуля для обробки завантаження. Ви створюєте контролер або модуль, який містить метод(и), які будуть обробляти завантаження файлів. У цих методах ви використовуєте спеціальні декоратори та інтерцептори для обробки файлів;

- використання multer для Обробки Запитів. Ви використовуєте бібліотеку multer (або інші аналогічні бібліотеки) для обробки запитів типу multipart/form-data. multer розбирає запит, виділяючи файли та інші дані, та передає їх далі у вашу програму для обробки;

- визначення методів для обробки файлів. Ви визначаєте методи у вашому контролері або сервісі, які будуть викликані під час завантаження файлів. Ці методи отримують інформацію про завантажені файли (наприклад, шляхи до файлів, їх імена тощо) та можуть здійснювати подальшу обробку цих файлів (наприклад, збереження на сервері, обробка даних тощо).

Після обробки файлів ви повертаєте відповідь клієнту, яка може містити, наприклад, підтвердження успішного завантаження файлів або іншу інформацію, що стосується завантаження.

Це загальний опис процесу, завантаження і отримання файлів наведено у прикладі у лістингу 3.11.

Лістинг 3.11 – Контролер файлів

```
@Controller('files')
export class FileController {
  @Get('/:id')
  async getFile(
    @Param('id') id: string,
    @Res() res: Response,
  ): Promise<Response<unknown>> {
    const [file, fileStream] =
      await this._filesService.getFileWithReadStream(id)

    res.writeHead(HttpStatus.OK, {
```

```

        'Content-Type': 'application/octet-stream', // i.e.
        "download"

        'Content-Disposition': `attachment;
filename=${file.originalName}`,
    })

    return fileStream.pipe(res)
}

@Post('upload')
@UseGuards(AuthGuard, RolesGuard)
@Roles(RoleEnum.MODERATOR, RoleEnum.USER, RoleEnum.ADMIN)
@UseInterceptors(
    FilesInterceptor('file-upload', 10, {
        storage: diskStorage({
            destination:
configService.get('DOWNLOAD_FILE_FOLDER'), // Folder to save
uploaded files
            filename: (req, file, callback) => {
                const uniqueSuffix = Date.now() + '-' +
Math.round(Math.random() * 1e9)
                const ext = extname(file.originalname)
                const filename = `${file.fieldname}-
${uniqueSuffix}${ext}`
                callback(null, filename)
            },
        }),
    )
)
async uploadFiles(
    @UploadedFiles() files: Array<Express.Multer.File>,
    @AuthUser() user: UserEntity
) {
    const savedFiles = await this._filesService.createFiles(
        files,
        user.id,
    )

    return savedFiles.map(file => ({ ...file, url: file.url }))
}
}

```

Оскільки файли можуть бути різного розміру і розширення. Тож для цього передбачена валідація файлів на розмір і типи файлів, яким дозволено завантажуватись. Приклади роботи завантаження і скачування файлів показано на рисунках 3.6-3.7.

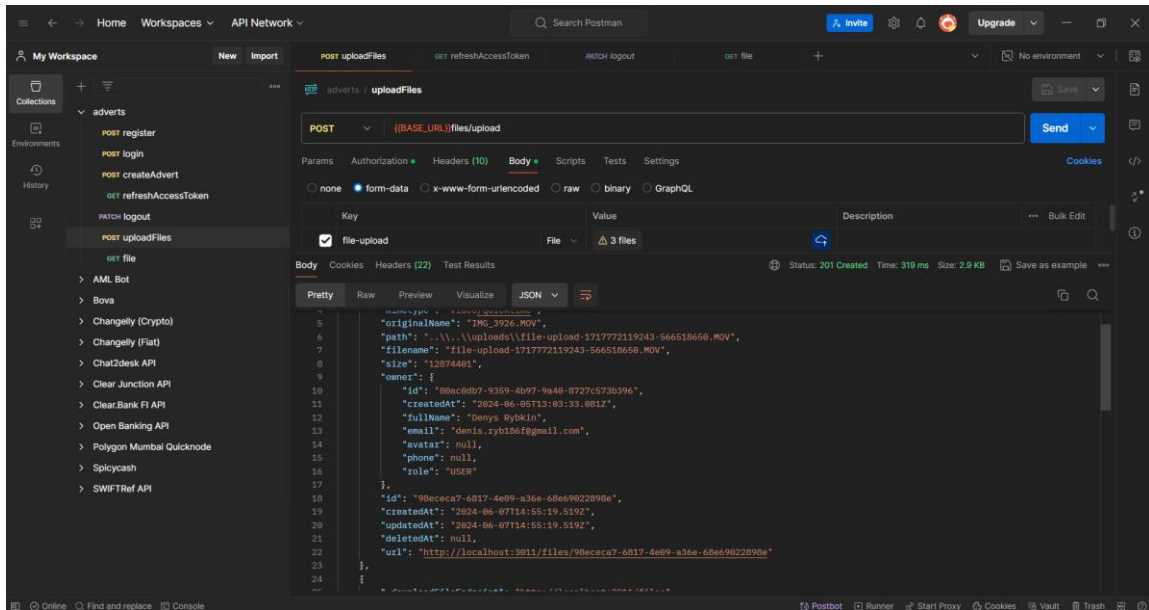


Рисунок 3.6 - Завантаження файлів фото/відео

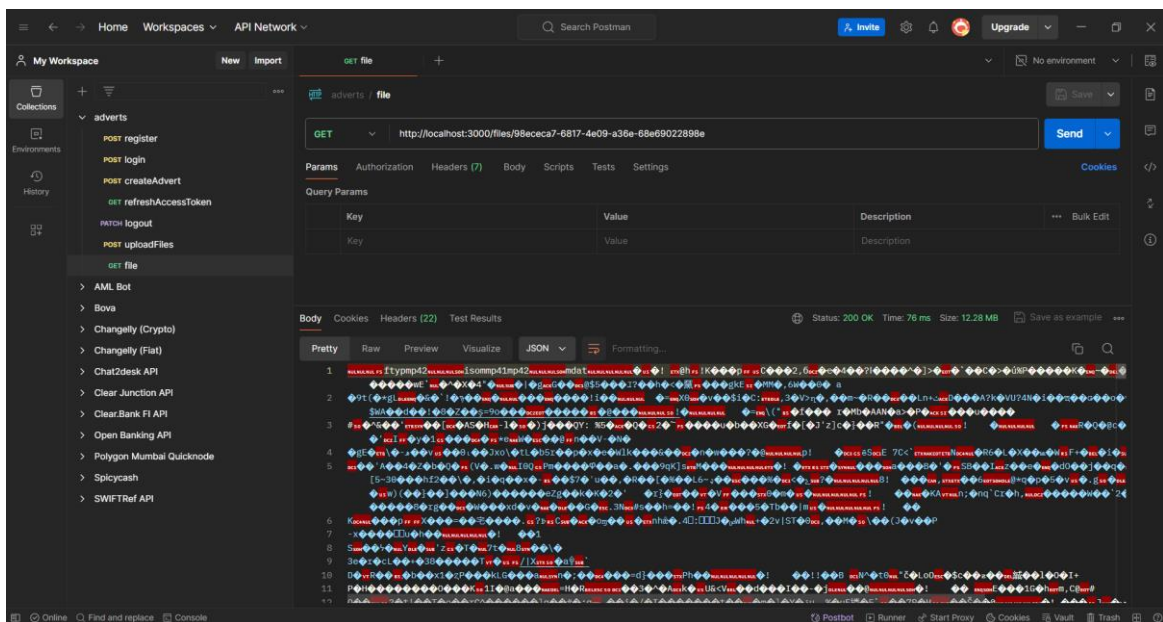


Рисунок 3.7 – Отримання файлу

3.5 Користувачький інтерфейс

Процес авторизації зі сторони користувачього інтерфейсу зображено на рисунках 3.8-3.9.

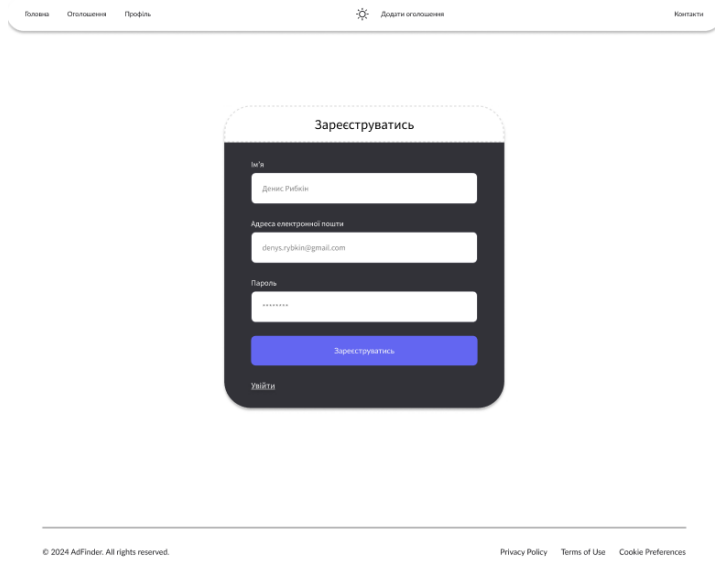


Рисунок 3.8 – Реєстрація користувача

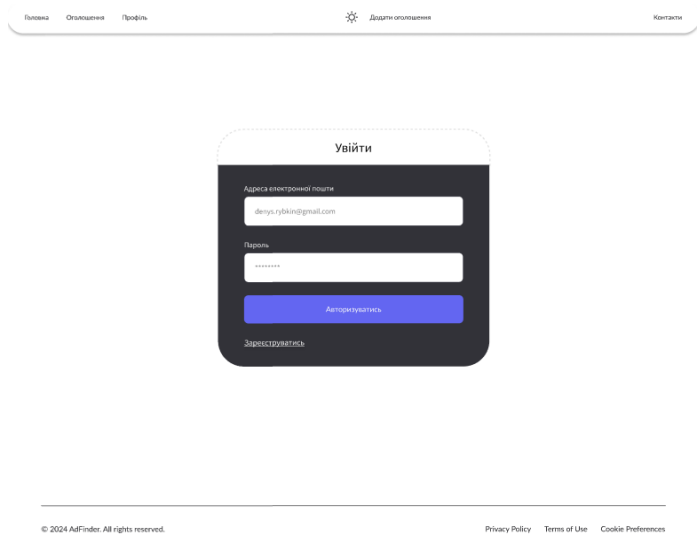


Рисунок 3.9 – Авторизація користувача

Після успішної реєстрації користувач попадає знову на головну сторінку і браузерний скрипт зберігає токени в local storage, надсилаючи кожного разу запит на оновлення токenu доступу. Щоб створити оголошення, користувач потрібен натиснути на вкладку «Додати оголошення» у header секції. Далі він попаде на сторінку створення оголошення, де заповнює всі необхідні поля. Приклад зображено на рисунку 3.10.

ГоловнаОголошенняПрофіль

Додати оголошення

Контакти

Створити оголошення

Вкажіть назву*

Продаю Xiaomi POCO F3 6/128GB

Опис*

Продам свій Xiaomi POCO F3 з 6 ГБ оперативної пам'яті та 128 ГБ пам'яті. Цей потужний смартфон у відмінному стані забезпечує безперебійну роботу користувача завдяки високопродуктивному процесору Snapdragon 870. POCO F3 має приголомшливий 6,67-дюймовий AMOLED-дисплей із частотою оновлення 120 Гц, що робить його ідеальним для ігор, потокового передавання та щоденного використання. Він також може похвалитися універсальною потрійною камерою для зйомки високоякісних фотографій і відео.

Характеристики:

- 6GB RAM
- 128GB Storage
- Snapdragon 870 Processor
- 6.67" AMOLED Display with 120Hz refresh rate
- Triple Camera System

Фото

Натисніть на фото, щоб завантажити зображення

Додати фото

Ціна

Обмін

Безкоштовно

1550

Опублікувати

© 2024 AdFinder. All rights reserved.

Privacy PolicyTerms of UseCookie Preferences

Рисунок 3.10 – Сторінка створення оголошення

Після створення оголошення, користувач зможе знайти його на сторінці пошуку оголошень, задавши необхідні критерії пошуку. Приклад зображено на рисунку 3.11.

ГоловнаОголошенняПрофіль

Додати оголошення

Контакти

Xiaomi POCO

Категорія

Ціна

Сортувати

Знайдено 1 оголошень

Продаю Xiaomi POCO F3 6/128GB

1550 грн

1

Рисунок 3.11 – Пошук оголошень

Для того, щоб переглянути детальну інформацію з описом про саме оголошення, необхідно зі списку натиснути на назву оголошення, що собою посиленням. Приклад сторінки з інформацією про оголошення зображено на рисунку 3.12.

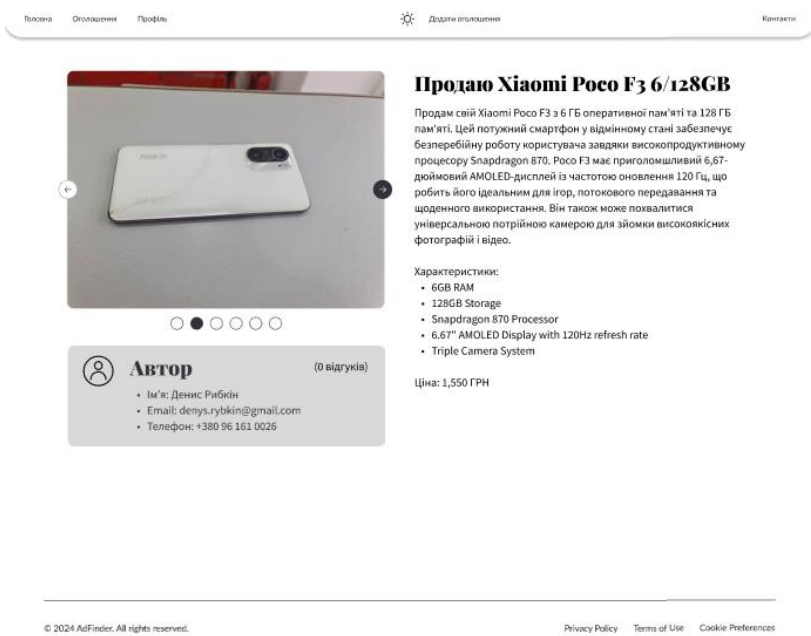


Рисунок 3.12 – Інформація про створене оголошення

ВИСНОВКИ

Унаслідок було спроектовано комп'ютерну платформу для розміщення онлайн оголошень купівлі-продаж. Платформа дозволяє спростити фізичним особам підприємцям ведення власного бізнесу і через інтернет продавати товари, зокрема навіть ті, що вже можуть не продаватися у звичайних магазинах. Ще одною особливістю системи є те, що покупці можуть з легкістю знайти будь який потрібний товар по задовільній ціні, вказуючи необхідні для себе критерії пошуку. Також важливою перевагою є орієнтація на якість послуг, оскільки платформа має інтуїтивно зрозумілий інтерфейс і службу підтримки.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Клієнт-серверна архітектура. URL: <https://www.theknowledgeacademy.com/blog/client-server-architecture/> (дата звернення: 25.04.2024).
2. Coding for Web Design 101: How HTML, CSS, and JavaScript Work. URL: <https://blog.hubspot.com/marketing/web-design-html-css-javascript> дата звернення: 25.04.2024);
3. Що таке PHP. Мануал. URL: <https://www.php.net/manual/en/intro-what-is.php> (дата звернення: 25.04.2024).
4. What is Python? Executive Summary. URL: <https://www.python.org/doc/essays/blurbs/> (дата звернення: 25.04.2024);
5. About Node.js. URL: <https://nodejs.org/en/about> (дата звернення: 25.04.2024).
6. The V8 JavaScript Engine. URL: <https://nodejs.org/en/learn/getting-started/the-v8-javascript-engine> (дата звернення: 25.04.2024).
7. The Node.js Event Loop. URL: <https://nodejs.org/en/learn/asynchronous-work/event-loop-timers-and-nexttick> (дата звернення: 25.04.2024).
8. Що таке фреймворк? Пояснюємо простими словами. URL: <https://brainlab.com.ua/uk/blog-uk/shho-take-frejmwork-poyasnyuyemo-prostymy-slovamy> (дата звернення: 25.04.2024).
9. Introduction to Nest.js — A Framework for Node.js. URL: <https://medium.com/google-developer-indonesia/introduction-to-nest-js-a-framework-for-node-js-1d5d3f60afef> (дата звернення: 25.04.2024).
10. Що таке база даних? URL: <https://www.oracle.com/ua/database/what-is-database/> (дата звернення: 25.04.2024).
11. Порівняльна характеристика реляційних та NoSQL баз даних. URL: <https://ir.lib.vntu.edu.ua/bitstream/handle/123456789/29461/9907.pdf> (дата звернення: 25.04.2024).

12. What Is MongoDB? URL: <https://www.techtarget.com/searchdatamanagement/definition/MongoDB> (дата звернення: 25.04.2024).
13. What is MySQL? URL: <https://www.oracle.com/mysql/what-is-mysql/> (дата звернення: 25.04.2024).
14. What is PostgreSQL? URL: <https://aws.amazon.com/rds/postgresql/what-is-postgresql> (дата звернення: 25.04.2024).
15. JSONB запити в PostgreSQL. URL <https://habr.com/ru/articles/254425/> (дата звернення: 25.04.2024).
16. What is Redis? URL: <https://www.ibm.com/topics/redis> (дата звернення: 25.04.2024).
17. What is NGINX? URL: <https://www.nginx.com/resources/glossary/nginx/> (дата звернення: 25.04.2024).

ДОДАТОК А

ОСНОВНІ КОДИ ПРОГРАМНИХ МОДУЛІВ

Лістинг А.1 – Код пошуку оголошень за критеріями

```

async searchAdverts(
  dto: SearchAdvertsDto,
): Promise<Pagination<AdvertEntity>> {
  const queryBuilder =
    this._advertRepository.createQueryBuilder('adverts')

  queryBuilder
    .leftJoinAndSelect(`${queryBuilder.alias}.category`, 'category')
    .leftJoinAndSelect(`${queryBuilder.alias}.user`, 'user')
    .where('TRUE')

  if (dto.ids) {
    queryBuilder.andWhereInIds(dto.ids)
  }

  if (dto.categoriesIds?.length) {
    queryBuilder.andWhere('category.id IN (:...categoriesIds)', {
      categoriesIds: dto.categoriesIds,
    })
  }

  if (typeof dto.withImages === 'boolean') {
    if (dto.withImages) {
      queryBuilder.andWhere(`${queryBuilder.alias}.images != {}`)
    } else {
      queryBuilder.andWhere(`${queryBuilder.alias}.images = {}`)
    }
  }

  if (dto.name) {
    queryBuilder.andWhere(
      `LOWER(${queryBuilder.alias}.name) LIKE
      LOWER(%${dto.name}%)`,
    )
  }

  if (typeof dto.isUsed === 'boolean') {
    queryBuilder.andWhere(`${queryBuilder.alias}.isUsed = :isUsed`,
    {
      isUsed: dto.isUsed,
    })
  }

  if (dto.priceTypes) {

```

```

    queryBuilder.andWhere(
      `${queryBuilder.alias}.priceType IN (:...priceTypes)`,
      { priceTypes: dto.priceTypes },
    )
  }

  if (dto.priceTypes) {
    queryBuilder.andWhere(
      `${queryBuilder.alias}.priceType IN (:...priceTypes)`,
      { priceTypes: dto.priceTypes },
    )
  }

  if (dto.price && Object.keys(dto.price).length) {
    const { from, to } = dto.price

    if (from) {
      queryBuilder.andWhere(
        `${queryBuilder.alias}.price >= :priceFrom`,
        { priceFrom: from.toString(10) },
      )
    }

    if (to) {
      queryBuilder.andWhere(
        `${queryBuilder.alias}.price <= :priceTo`,
        { priceTo: to.toString(10) },
      )
    }
  }

  const { limit = DEFAULT_LIMIT, sortBy = 'NEW' } = dto.pagination || {}

  let cursor = dto.pagination?.cursor

  if (cursor) {
    queryBuilder.where(`${queryBuilder.alias}.id > :cursor`, {
      cursor })
  }

  queryBuilder.take(limit)

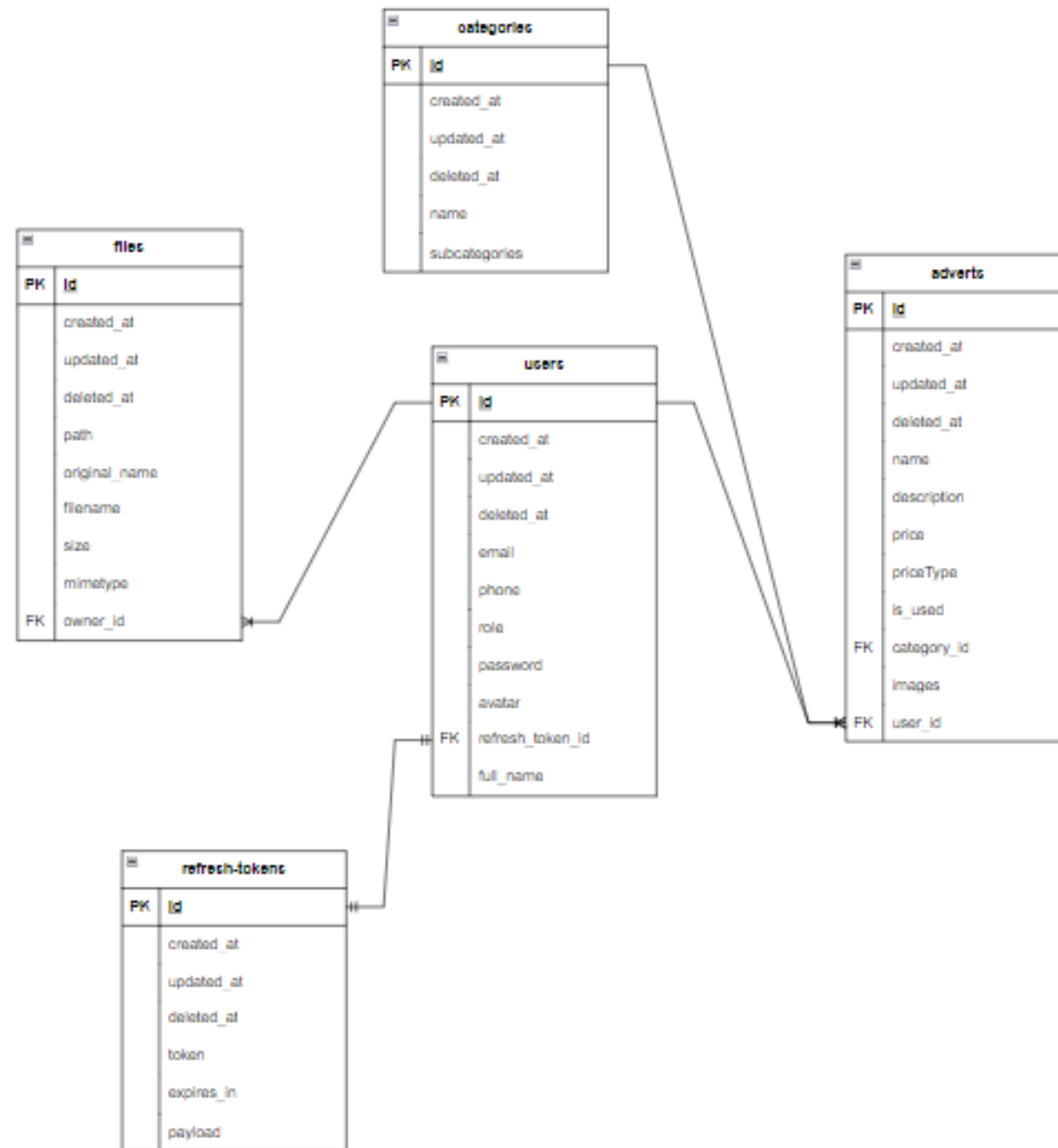
  if (sortBy === SortByEnum.NEW)
    queryBuilder.orderBy(`${queryBuilder.alias}.createdAt`, 'DESC')
  else if (sortBy === SortByEnum.OLD)
    queryBuilder.orderBy(`${queryBuilder.alias}.createdAt`, 'ASC')
  else if (sortBy === SortByEnum.CHEAP)
    queryBuilder.orderBy(`${queryBuilder.alias}.price`, 'ASC')
  else if (sortBy === SortByEnum.EXPENSIVE)
    queryBuilder.orderBy(`${queryBuilder.alias}.price`, 'DESC')

  const [adverts, count] = await queryBuilder.getManyAndCount()

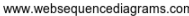
```

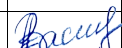
```
cursor = adverts.length === limit ? adverts[adverts.length - 1].id :
null

return {
  nodes: adverts,
  count,
  cursor,
}
```



					13.02070849.00042 ПЛ1							
					Розробка комп'ютерної платформи онлайн оголошень			Лист.	Маса	Масшт.		
Зам.	Лист.	№ докум.	Підп.	Дата								
Розроб.		Рибкін Д.М.										
Перев.		Скрупський С.Ю.										
Т.контр.												
					Використання таблиць БД			Лист 1		Листів 1		
Н.контр.		Польська О. В.						НУ «Запорізька політехніка» КНТ-520				
Затв.		Кудерметов Р.К.										



					13.02070849.00042 ПЛ2										
					Розробка комп'ютерної платформи онлайн оголошень					Лист.		Маса	Масшт.		
Зам.	Лист.	№ докум.	Підп.	Дата											
Розроб.		Рибкін Д.М.													
Перев.		Скрупський С.Ю.													
Т.контр.															
Н.контр.		Польська О. В.			Діаграма послідовності роботи з системою					НУ «Запорізька політехніка» КНТ-520					
Затв.		Кудерметов Р.К.													

ГоловнаОголошенняПрофіль

Додати оголошення

Контакти

Зареєструватись

Ім'я

Ваше ім'я

Адреса електронної пошти

Ваш e-mail

Пароль

Ваш пароль

Зареєструватись

Увійти

Контакти

Ви зможете зв'язатись з нами

Email

contact@adfinder.com

Телефон

+380 (63) 312-0098

Адреса

вул. Водограйна, буд. 12, Київ, Україна

ГоловнаОголошенняПрофіль

Додати оголошення

Контакти

Ім'я

Ваше ім'я

Адреса електронної пошти

Ваш e-mail

Повідомлення

Коротко опишіть ваш запит

Відправити

ГоловнаОголошенняПрофіль

Додати оголошення

Контакти

Продаю Xiaomi POCO F3 6/128GB

Продам свій Xiaomi POCO F3 з 6 ГБ оперативної пам'яті та 128 ГБ пам'яті. Цей потужний смартфон у відмінному стані забезпечує безперебійну роботу користувача завдяки високопродуктивному процесору Snapdragon 870. POCO F3 має приголомшливий 6,67-дюймовий AMOLED-дисплей із частотою оновлення 120 Гц, що робить його ідеальним для ігор, потокового передавання та щоденного використання. Він також може похвалитися універсальною потрійною камерою для зйомки високоякісних фотографій і відео.

Характеристики:

- 6GB RAM
- 128GB Storage
- Snapdragon 870 Processor
- 6.67" AMOLED Display with 120Hz refresh rate
- Triple Camera System

Ціна: 1,550 ₴

Автор

Ім'я: Денис Рибкін

Email: denys.rybkin@gmail.com

Телефон: +380 96 161 0026

(0 відгуків)

ГоловнаОголошенняПрофіль

Додати оголошення

Контакти

Пошук...

Категорія

Ціна

Сортувати

Знайдено 37 оголошень

Laptop Asus F512IDTOPi Ryzen 3 3200U8GB Ram SSD NVME 325...

109\$

WARRANTY Asus TUF (F417H) (i5-8300H/ RAM 8GB GTX 1050TI)

472\$

Asus Rog Strix G713RM TOPiRyzen 913060Laptop. 360Hz15.6GB...

2025\$

Laptop Asus Lenovo from PolandLaptop Asus Lenovo from Poland

27\$

Powerful Laptop Asus! Used

86\$

Gaming Laptop Asus (i7-8750h NVIDIA 1050ti gaming. 240ssd.1tb hdd...

329\$

Selling Laptop Asus UX501V 15.6 inchSelling Laptop Asus UX501V...

208\$

Laptop Asus F512IDTOPi Ryzen 3 3200U8GB Ram SSD NVME 256...

621\$

12...

Створити оголошення

Введіть назву*

Телевізор, MacBook, і т.д.

Опис*

Детальний опис оголошення

Відео

Налаштувати

Додати фото

Ціна

Обсяг

Категорія

грн

Опублікувати

ГоловнаОголошенняПрофіль

Додати оголошення

Контакти

Знайди своє оголошення

Пошук...

ГоловнаОголошенняПрофіль

Додати оголошення

Контакти

Увійти

Адреса електронної пошти

Ваш e-mail

Пароль

Ваш пароль

Авторизуватись

Зареєструватись

ГоловнаОголошенняПрофіль

Додати оголошення

Контакти

13.02070849.00042 ПЛЗ

Розробка комп'ютерної платформи онлайн оголошень

Інтерфейс вебсайту (сторінки)

Лист.	Маса	Масшт.
Лист 1		Листів 1

НУ «Запорізька політехніка» КНТ-520

Зам.	Лист.	№ докум.	Підп.	Дата	
	Розроб.	Рибкін Д.М.	Рибкін		
	Перев.	Скрупський С.Ю.	Скрупський		
	Т.контр.				
	Н.контр.	Польська О. В.	Польська		
	Затв.	Кудерметов Р.К.	Кудерметов		