

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

ПОЯСНЮВАЛЬНА ЗАПИСКА

до кваліфікаційної роботи освітнього ступеня «бакалавр»
за спеціальністю 121 «Інженерія програмного забезпечення»
(освітня програма «Інженерія програмного забезпечення»)
на тему:

Вебдодаток «Електронна дошка оголошень»

Виконав студент групи ІПЗк-23-1
Андрощук Дмитро Сергійович

Керівник роботи:
ЄФРЕМОВ Юрій Миколайович

Рецензент:
Годлевський Юрій Олександрович

Житомир – 2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДЕРЖАВНИЙ УНІВЕРСИТЕТ «ЖИТОМИРСЬКА ПОЛІТЕХНІКА»
ФАКУЛЬТЕТ ІНФОРМАЦІЙНО-КОМП'ЮТЕРНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНЖЕНЕРІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

«ЗАТВЕРДЖУЮ»
Зав. кафедри інженерії
програмного забезпечення
Вакалюк Тетяна
«14» лютого 2025 р.

ЗАВДАННЯ
на кваліфікаційну роботу

Здобувач вищої освіти: **АНДРОЩУК Дмитро Сергійович**

Керівник роботи: **ЄФРЕМОВ Юрій Миколайович**

Тема роботи: **«Електронна дошка оголошень»**,

затверджена Наказом закладу вищої освіти від **«14» лютого 2025 р., №61/с**

Вихідні дані для роботи: розробка веб-додатку для розміщення, перегляду та управління оголошеннями, що включає створення оголошень із мультимедійним контентом, фільтрацію, чати та авторизацію, а також системи адміністрування для керування користувачами, оголошеннями та запитами на преміум-статус; методи та засоби, які використовуються для створення веб-додатку з використанням React, Node.js, Express та SQLite.

Консультанти з бакалаврської кваліфікаційної роботи із зазначенням розділів, що їх стосуються:

Розділ	Консультант	Завдання видав	Завдання прийняв
1	Єфремов Ю. М.	27.02.2025р.	27.02.2025р.
2	Єфремов Ю. М.	27.02.2025р.	27.02.2025р.
3	Єфремов Ю. М.	27.02.2025р.	27.02.2025р.

РЕФЕРАТ

Випускна кваліфікаційна робота бакалавра, складається з веб-додатку «Електронна дошка оголошень» для розміщення, пошуку та управління оголошеннями і пояснювальної записки. Пояснювальна записка містить 70 сторінок, 35 ілюстрацій та 2 таблиці.

Метою роботи є розробка зручної платформи для створення, перегляду та адміністрування оголошень із підтримкою фільтрації, чатів і преміум-статусу.

У роботі визначено завдання розробки, проаналізовано аналоги системи, описано сценарії роботи додатку, його структуру, алгоритми взаємодії модулів і об'єктну модель. Наведено результати тестування системи.

КЛЮЧОВІ СЛОВА: ВЕБ-ДОДАТОК, ДОШКА ОГЛОШЕНЬ, NODE.JS, REACT, SQLITE.

					ІПЗ.КР.Б-121-25-ПЗ						
Змн.	Арк.	№ докум.	Підпис	Дата	Вебдодаток «Електронна дошка оголошень»				Літ.	Арк.	Аркушів
Розроб.		Андрощук Д. А.									
Керівник		Єфремов Ю.М.								3	
									Житомирська політехника, група ІПЗк-23-1		
Н. контр.											
Зав. каф.		Вакалюк Т. А.									

ABSTRACT

The qualification work consists of the "Electronic Bulletin Board" web application for posting, searching, and managing advertisements, along with an explanatory note. The explanatory note includes 70 pages, 35 illustrations, and 2 tables.

The aim of the work is to develop a user-friendly platform for creating, viewing, and administering advertisements with support for filtering, chats, and premium status features.

The work outlines the development tasks, analyzes system analogs, describes the application's workflows, structure, module interaction algorithms, and object model. The results of system testing are also presented.

KEYWORDS: WEB APPLICATION, BULLETIN BOARD, NODE.JS, REACT, SQLITE.

					ИПЗ.КР.Б-121-25-ПЗ	Апр.
						4

Зміст

РЕФЕРАТ	3
ABSTRACT	4
Перелік умовних скорочень	6
ВСТУП	7
РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-ДОДАТКУ КНИЖКОВОГО МАГАЗИНУ	11
1.1 Постановка завдання.....	11
1.2 Аналіз аналогів програмного продукту	13
1.3 Вибір архітектури веб-додатку	18
1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення.....	21
Висновки до розділу 1.....	23
РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ.....	25
2.1. Визначення варіантів використання та структури системи.....	25
2.2. Розробка структури бази даних.....	29
2.3. Реалізація функціоналу веб-додатку	33
ВИСНОВКИ ДО РОЗДІЛУ 2	36
РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ	38
3.1. Порядок встановлення та налаштування системи.....	38
3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з системою.....	39
3.3. Тестування вебдодатку	50
ВИСНОВКИ ДО РОЗДІЛУ 3	55
ВИСНОВКИ	57
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	59
ДОДАТКИ	62
Додатки А	63

Перелік умовних скорочень

API – прикладний програмний інтерфейс

БД – база даних

ПЗ – програмне забезпечення

ВСТУП

Актуальність теми. Сьогодні інформаційні технології стрімко змінюють те, як ми шукаємо та обмінюємося інформацією. Це особливо помітно в таких сферах, як розміщення оголошень, де все більше користувачів переходять на онлайн-платформи. Людям потрібен швидкий і зручний доступ до актуальних пропозицій, і саме тому створення веб-додатку для електронної дошки оголошень є дуже своєчасним. Така система дозволяє спростити пошук, додавання та керування оголошеннями, а також налагодити ефективну взаємодію між користувачами. Актуальність цього проєкту зумовлена не лише технічними чи бізнес-факторами, а й тим, що подібні платформи реально допомагають спільнотам, малому бізнесу та просто людям, які хочуть щось продати або знайти.

Сучасний розвиток інформаційних технологій кардинально змінює підходи до організації обміну інформацією, зокрема у сфері розміщення та пошуку оголошень. Зростання популярності онлайн-платформ, зміна поведінки користувачів та потреба в швидкому доступі до актуальних пропозицій роблять створення веб-додатків для електронних дощок оголошень надзвичайно актуальним. Розробка такої системи відповідає сучасним викликам, адже вона забезпечує зручний доступ до оголошень, оптимізує процеси їх створення, управління та пошуку, а також сприяє ефективній взаємодії між користувачами. Актуальність цього завдання зумовлена не лише економічними факторами, такими як підвищення конкурентоспроможності платформ, а й соціальною значущістю, адже електронні дошки оголошень сприяють швидкому обміну інформацією, що є важливим для розвитку місцевих спільнот, бізнесу та індивідуальних ініціатив.

Метою розробки даного програмного додатку є створення зручної, функціональної та безпечної платформи для розміщення, пошуку та управління оголошеннями, яка відповідає потребам як звичайних користувачів, так і адміністраторів системи. З огляду на актуальність теми, мета полягає в

забезпеченні інтуїтивно зрозумілого інтерфейсу для створення оголошень, підтримці категоризації, фільтрації, додавання мультимедійного контенту, а також реалізації системи авторизації та управління преміум-статусом користувачів.

Поставлена мета обумовлює необхідність виконання наступних завдань:

- 1) Провести аналіз існуючих електронних дощок оголошень з метою виявлення їхніх функціональних можливостей та підходів до реалізації.
- 2) Сформулювати вимоги до веб-застосунку, що враховують зручність використання, безпеку збереження даних та можливість масштабування системи.
- 3) Спроектувати структуру бази даних для ефективного зберігання та обробки інформації про оголошення, користувачів та інші супутні дані.
- 4) Провести аналіз існуючих електронних дощок оголошень з метою виявлення їхніх функціональних можливостей та підходів до реалізації.
- 5) Сформулювати вимоги до веб-застосунку, що враховують зручність використання, безпеку збереження даних та можливість масштабування системи.

Об'єктом дослідження є обмін інформацією в інтернеті за допомогою онлайн-дощок оголошень.

Для реалізації веб-додатку використано сучасні технології та методи розробки програмного забезпечення. Клієнтська частина побудована на основі бібліотеки React.js, яка забезпечує створення динамічного та інтерактивного інтерфейсу з високою швидкістю та модульною структурою. На серверній стороні використано технологію Node.js у поєднанні з фреймворком Express.js для обробки HTTP-запитів та управління API. Для зберігання даних обрано SQLite – легку та ефективну систему управління базами даних, яка ідеально підходить для проєктів середнього масштабу. Безпека забезпечується за допомогою JWT-токенів для авторизації та bcrypt для хешування паролів. Для обробки зображень використано бібліотеку Multer, що дозволяє завантажувати

та зберігати файли з урахуванням обмежень формату та розміру. Моделювання системи здійснювалося за допомогою об'єктно-орієнтованого аналізу, зокрема через створення ER-діаграм для структури бази даних та діаграм компонентів для опису архітектури. Асинхронне програмування використано для оптимізації взаємодії між клієнтом і сервером, а методи тестування – для забезпечення стабільності системи.

Предмет дослідження – веб-додаток для розміщення й пошуку оголошень, створений з використанням сучасних технологій. Апробація розробленого програмного додатку проводилася на засіданнях кафедри інформаційних технологій та групових зборах у 2025 році. Під час цих заходів було представлено основні функціональні можливості системи, продемонстровано її інтерфейс та проведено обговорення з викладачами та студентами. Отримані відгуки та зауваження сприяли вдосконаленню системи, зокрема покращенню зручності інтерфейсу та оптимізації роботи з базою даних. Планується подальша апробація шляхом демонстрації системи на внутрішніх семінарах кафедри, що дозволить зібрати додаткові пропозиції для її вдосконалення перед можливим впровадженням.

Структура пояснювальної записки складається з трьох основних розділів, висновків, списку використаних джерел та додатків, загальним обсягом 70 сторінок, із використанням 17 джерел та 1 додаток. У першому розділі "Аналіз задачі та особливостей використання інформаційних технологій" (17 сторінок) описано предметну область електронних дощок оголошень, проведено аналіз вимог, здійснено огляд аналогів, сформульовано завдання розробки та обґрунтовано вибір технологій. Другий розділ "Проектування веб-додатку" (21 сторінок) присвячено визначенню варіантів використання, розробці структури файлів, бази даних та реалізації функціоналу, включаючи опис API-ендпоінтів. Третій розділ "Інтерфейс та порядок роботи з веб-додатком" (19 сторінок) охоплює інструкції з встановлення, опис інтерфейсу, порядок роботи користувачів та результати тестування. У висновках (2 сторінки)

підведено підсумки роботи, а в додатках представлено лістинг коду, діаграми та приклади тестових даних.

Таким чином, розробка веб-додатку для електронної дошки оголошень є важливим внеском у вдосконалення процесів обміну інформацією в цифровому середовищі. Завдяки використанню сучасних технологій, таких як React.js, Node.js та SQLite, а також ретельному проєктуванню системи вдалося створити функціональний продукт, який відповідає потребам користувачів та має потенціал для подальшого масштабування та вдосконалення.

РОЗДІЛ 1. АНАЛІЗ ЗАДАЧІ ТА ОСОБЛИВОСТЕЙ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ СТВОРЕННЯ ВЕБ-ДОДАТКУ ЕЛЕКТРОННОЇ ДОШКИ ОГОЛОШЕНЬ

1.1 Постановка завдання

Головною метою розробки веб-додатку електронної дошки оголошень є створення зручної та функціональної платформи для розміщення, перегляду, пошуку та керування оголошеннями [1]. Додаток призначений для широкого кола користувачів, включаючи приватних осіб, які бажають продавати чи купувати товари або послуги, а також менеджерів, які забезпечують модерацію та адміністрування платформи.: клієнтів і менеджерів [2].

У рамках реалізації проєкту необхідно розробити веб-додаток, який дозволить користувачам створювати оголошення, переглядати їх за категоріями, фільтрувати за ціною чи іншими параметрами, а менеджерам — керувати користувачами, оголошеннями та запитами на преміум-статус. Основними етапами реалізації цього завдання є:

Під час створення серверної частини додатку я почав із проєктування бази даних. Треба було передбачити зберігання всього необхідного: користувачів, оголошень, категорій, фотографій, чатів, повідомлень і запитів на преміум-доступ. Для цього я розробив кілька таблиць. У таблиці користувачів зберігаються логін, пароль, email, роль (звичайний користувач чи адміністратор), а також статус преміум і дата реєстрації. Категорії мають просту структуру — назва ідентифікатора та самої категорії, наприклад «Нерухомість» чи «Транспорт». В оголошеннях зберігається заголовок, опис, ціна, контакти, категорія, хто створив, а також дати створення й оновлення, і навіть кольорова палітра для оформлення. Окремо зберігаються фото, які прив'язані до конкретного оголошення. Для чатів і повідомлень зроблено свої таблиці, де фіксується, хто з ким спілкується, тип чату (наприклад, преміум-запит), статус, текст повідомлень і час відправлення.

Базу даних я реалізував на SQLite, бо це зручно для проєктів середнього масштабу. Усі таблиці пов'язані між собою, наприклад, оголошення мають зовнішні ключі на користувачів і категорії. Коли структура бази була готова, я взявся за написання API. Зробив окремі маршрути для роботи з користувачами — це реєстрація, авторизація, оновлення профілю, видалення облікового запису та керування преміумом. Для оголошень реалізував створення, редагування, видалення, пошук за ключовими словами, фільтрацію за категоріями та сортування — наприклад, за ціною або датою. Фото можна завантажувати й видаляти, вони автоматично прив'язуються до відповідного оголошення. Також зроблено функціонал чатів — користувачі можуть створювати чати та обмінюватися повідомленнями. Категорії теж мають свій ендпоінт, з якого можна отримати весь список.

Усі маршрути чітко структуровані. Для авторизації та реєстрації є `/api/auth/register` і `/api/auth/login`. Оголошення обробляються через `/api/announcements` та подібні адреси, які включають ID або фото. Категорії, чати, повідомлення, користувачі — кожен має свої зрозумілі шляхи. Для преміум-запитів теж виділений окремий маршрут.

Наприкінці я протестував серверну частину. Перевіряв, чи все правильно працює — чи створюються нові записи, чи можна їх змінювати та видаляти. Також спеціально викликав помилки — наприклад, коли токен неправильний або коли завантажувється забагато фото. Крім того, перевіряв безпеку — чи справно працює авторизація через JWT і чи не можна отримати доступ до чужих даних.

Паралельно я займався створенням адміністративної частини додатку. Спочатку я продумав її дизайн він мав бути функціональним і зручним для менеджерів які керують користувачами оголошеннями та запитами. Я зверстав навігаційну панель яка дозволяє швидко переходити до керування користувачами чатами та перегляду профілю. Сторінка керування користувачами відображає таблицю з інформацією про кожного користувача включно з айді ім'ям електронною адресою роллю в системі та преміумстатусом також є кнопки щоб видалити користувача або змінити його статус. Сторінка

чатів показує всі запити від користувачів і дозволяє менеджеру на них відповідати. У профілі менеджера відображається його основна інформація. Щодо логіки роботи адміністративної частини я створив необхідні серверні маршрути для керування користувачами а також для роботи з чатами і повідомленнями. Я реалізував компоненти які відповідають за сторінки адміністратора наприклад сторінку з таблицею користувачів де можна змінити статус або видалити людину сторінку з чатами а також окрему сторінку чату де є можливість відповідати користувачу і надати чи скасувати преміум. Після цього я протестував права доступу щоб упевнитися що лише менеджери можуть бачити ці сторінки і перевіряв чи правильно оновлюються дані після редагування або видалення користувача.

Додаток адаптований для різних пристроїв завдяки компонентному підходу React та адаптивному дизайну, визначеному в styles.css. Безпека забезпечується JWT-авторизацією, захистом ролей та валідацією даних. Для тестування створено скрипт seed.js, який заповнює базу даних тестовими даними (користувачі, оголошення, фотографії). Апробація додатку проводилась на засіданнях кафедри та групових зборах, де було продемонстровано основний функціонал: створення оголошень, фільтрацію, чати та адміністративні функції. Публікацій результатів не було, але отримані відгуки підтвердили відповідність додатку поставленим вимогам. Окремо відзначили зручність адміністративного інтерфейсу і швидку роботу додатку, навіть при великій кількості тестових даних.

1.2 Аналіз аналогів програмного продукту

Розробка веб-додатку електронної дошки оголошень вимагає детального аналізу предметної області, включаючи особливості електронної комерції, потреби користувачів і сучасні технологічні рішення. Проведений аналіз аналогів дозволяє оцінити сильні та слабкі сторони існуючих платформ, виявити актуальні тенденції та визначити унікальні характеристики розробленого додатку. Для порівняння розглянуто чотири популярні платформи, що надають

функціонал для розміщення та перегляду оголошень, з метою оцінки їх ефективності та конкурентних переваг розробленої системи.

OLX (рис. 1.1) є однією з найбільших міжнародних платформ для розміщення оголошень, включаючи Україну [3]. Система підтримує широкий спектр категорій, фільтри за ціною, місцезнаходженням і ключовими словами, а також чат для спілкування між користувачами. Сильні сторони: великий вибір оголошень, зручна система фільтрації, інтеграція з мобільними додатками. Слабкі сторони: перевантажений інтерфейс для нових користувачів, обмежені можливості для менеджерів (наприклад, адміністрування користувачів). Розроблена система вирізняється простішим інтерфейсом, локалізацією для українського ринку та наявністю менеджерських функцій, таких як управління користувачами та преміум-запитами.

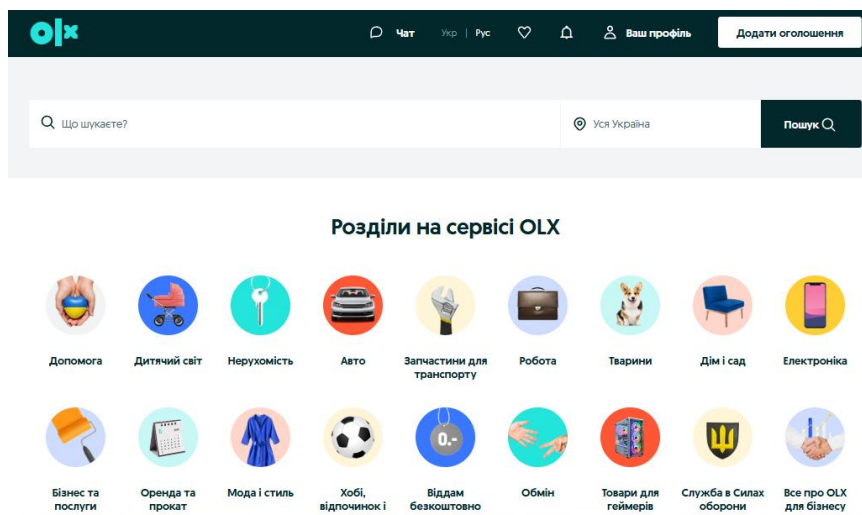


Рис. 1.1 – Головна сторінка OLX [3]

Prom.ua (рис. 1.2) – українська платформа, що поєднує функції маркетплейсу та дошки оголошень [4]. Вона пропонує категорії товарів і послуг, фільтри за ціною та рейтингом продавців, а також можливість створення інтернет-магазинів. Сильні сторони: локалізація для українського ринку, підтримка малого бізнесу, інтеграція з платіжними системами. Слабкі сторони: обмежена функціональність для приватних оголошень, складність адміністрування для менеджерів. Розроблена система перевершує Prom.ua

завдяки підтримці чатів для спілкування, можливості вибору кольорової палітри для оголошень (colorPalette) і менеджерським інструментам (наприклад, управління преміум-статусами).

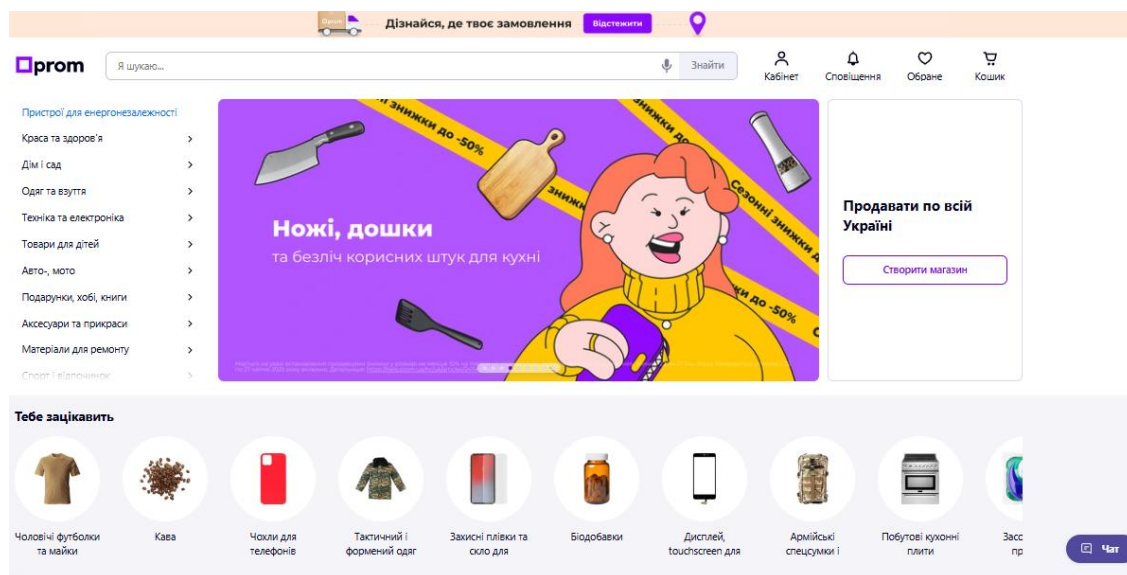


Рис. 1.2 – Головна сторінка Prom.ua [4]

eBay Classifieds (рис. 1.3) – міжнародна платформа для місцевих оголошень, що працює в кількох країнах [5]. Вона підтримує базовий пошук за категоріями, фільтри за ціною та розташуванням, а також просту систему розміщення оголошень. Сильні сторони: глобальний охоп, простий інтерфейс, підтримка кількох мов. Слабкі сторони: відсутність локалізації для України (інтерфейс англійською, ціни в доларах), обмежені інструменти для менеджерів. Розроблена система виграє за рахунок україномовного інтерфейсу, цін у гривнях і розширених адміністративних можливостей, таких як управління користувачами та чатами. Також варто зазначити, що наша система краще адаптована до потреб українських користувачів як з точки зору дизайну, так і з урахуванням місцевих особливостей поведінки на ринку онлайн-оголошень. Крім того, розширений функціонал для адміністраторів забезпечує більш гнучке управління сервісом і оперативне реагування на запити або порушення з боку користувачів.

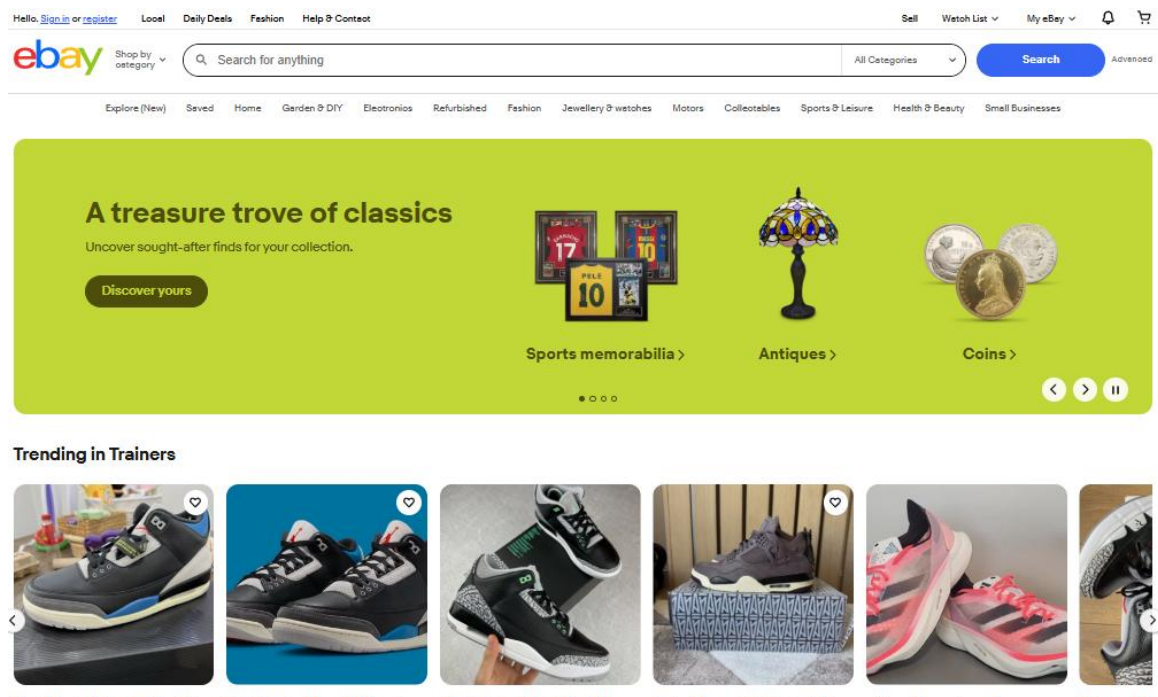


Рис. 1.3 – Головна сторінка eBay Classifieds [5]

Gumtree (рис. 1.4) – популярна платформа для місцевих оголошень, що працює переважно у Великобританії та Австралії [6]. Вона пропонує зручний пошук за категоріями, фільтри за ціною та розташуванням, а також інтеграцію з соціальними мережами. Сильні сторони: простий і зрозумілий інтерфейс, підтримка місцевих спільнот. Слабкі сторони: відсутність локалізації для України, обмежені можливості для менеджерів і відсутність функцій чату. Розроблена система перевершує Gumtree завдяки локалізації, інтеграції чатів (ChatsPage) і підтримці преміум-статусів, що підвищує привабливість оголошень.

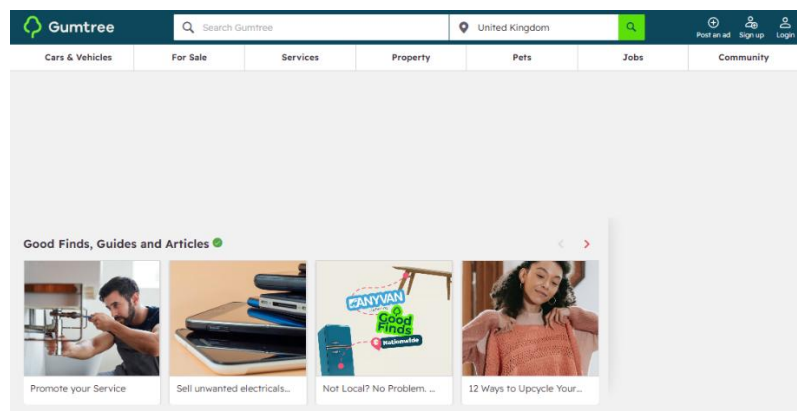


Рис. 1.4 – Головна сторінка Gumtree [6]

Аналіз аналогів показав, що сучасні дошки оголошень повинні поєднувати зручність для користувачів, ефективну обробку даних і гнучкість архітектури. Розроблена система використовує React для створення інтерактивного фронтенду, SQLite для зберігання даних і Express для серверної логіки, що забезпечує швидкодію та масштабованість. Порівняння з аналогами виявило конкурентні переваги розробленої системи, такі як локалізація, підтримка чатів, менеджерські функції та можливість додавання кольорових палітр до оголошень. Водночас система поступається глобальним платформам за масштабом і кількістю користувачів. Подальший розвиток може включати інтеграцію push-повідомлень і розширення фільтрів для пошуку, що підвищить її конкурентоспроможність.

Отже, порівняння за ключовими критеріями наведене в таблиці 1.1.

Таблиця 1.1.

Порівняння розробленої системи з аналогами

Система	Реєстрація	Кабінет користувача	Фільтри пошуку	Інструменти менеджера	Чат із менеджером	Локалізація (UA)
OLX	Швидка	Так	Так	Ні	Так	Так
Prom.ua	Швидка	Так	Так	Обмежено	Ні	Так
eBay Classifieds	Швидка	Так	Так	Ні	Ні	Ні
Gumtree	Швидка	Так	Так	Ні	Ні	Ні
Розроблена система	Швидка	Так	Так	Так	Так	Так

Після аналізу аналогів можна зробити висновок, що кожна платформа має свої переваги та недоліки. OLX вирізняється великим охоптом і зручними фільтрами, але поступається в менеджерських функціях. Prom.ua добре адаптований до українського ринку, але має обмеження для приватних оголошень. eBay Classifieds і Gumtree орієнтовані на міжнародні ринки, що ускладнює їх використання в Україні через відсутність локалізації. Розроблена система поєднує зручність для клієнтів (фільтри, чати, кольорові палітри) і менеджерів (управління користувачами, преміум-статусами), а також враховує

локальні особливості, що робить її конкурентоспроможною на українському ринку.

З урахуванням отриманих даних виникає потреба в розробці веб-додатку з акцентом на локалізацію та спеціалізовану функціональність. Особливостями розробленої системи є: лаконічний і зрозумілий дизайн, швидка реєстрація (auth/register), особистий кабінет із можливістю управління оголошеннями (MyAnnouncementsPage), розширені фільтри пошуку (announcements?sort, search, category), чат із менеджером (ChatsPage), менеджерські функції (AdminUsersPage, premium-request) і локалізація для українського ринку (україномовний інтерфейс, ціни в гривнях). Ці характеристики забезпечують баланс між простотою використання для клієнтів і ефективністю адміністрування, що є ключовою перевагою системи.

1.3 Вибір архітектури веб-додатку

Для створення веб-додатку електронної дошки оголошень можливо застосувати кілька архітектурних шаблонів програмного забезпечення, таких як:

- шаблон слоїв;
- MVC (Model-View-Controller);
- сервісно-орієнтована архітектура;
- шаблон клієнт-сервер;
- мікросервіси.

Для розробки інформаційної системи електронної дошки оголошень оптимальними є шаблони клієнт-сервер та MVC [7]. Однак, з огляду на специфіку проєкту, найкращим вибором є архітектурний шаблон клієнт-сервер (рис. 1.5).

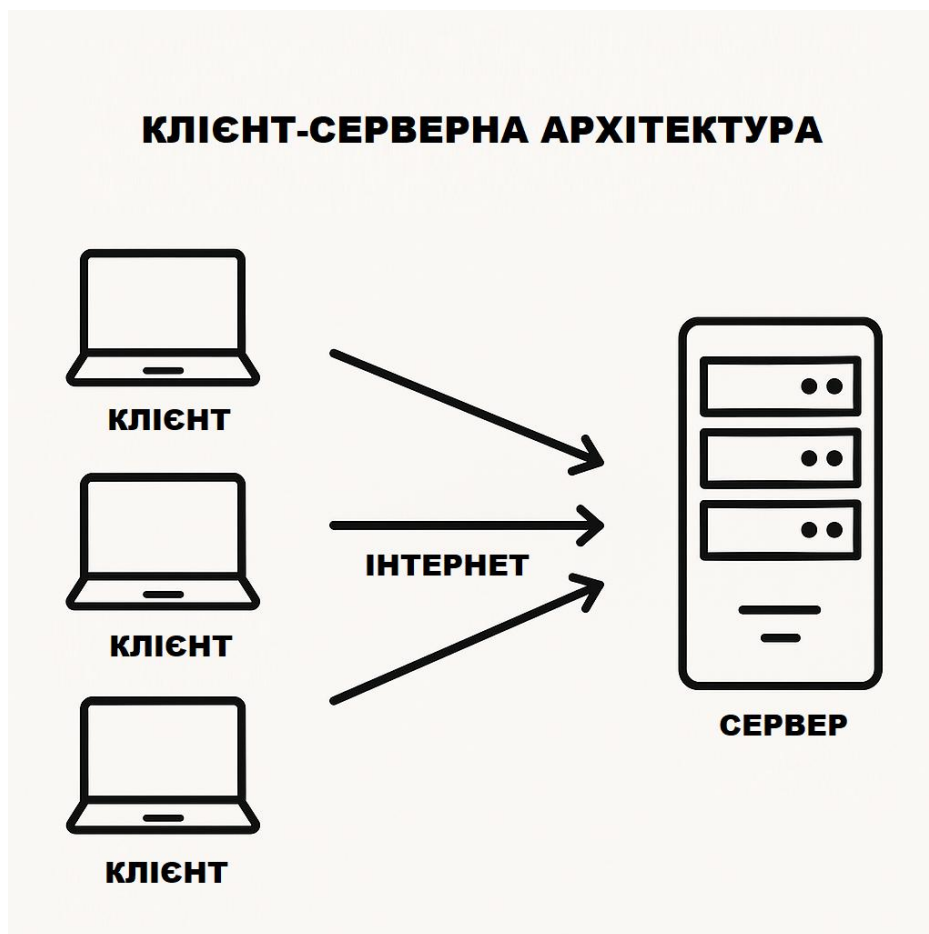


Рис. 1.5 – Схема шаблону клієнт-сервер

Шаблон клієнт-сервер є широко поширеним у розробці веб-додатків завдяки своїй простоті, гнучкості та ефективності [8]. Він базується на чіткому розподілі функцій між двома основними компонентами.

Клієнт – це програмний компонент, який забезпечує взаємодію з користувачем [9]. У даному проєкті клієнт реалізований за допомогою бібліотеки React, яка відображає веб-інтерфейс у браузері. Основна роль клієнта полягає у відображенні даних (оголошень, категорій, чатів), зборі введеної інформації (наприклад, даних для створення оголошення чи авторизації) та надсиланні запитів до сервера. Клієнт отримує відповіді від сервера у форматі JSON і візуалізує їх для користувача, забезпечуючи інтерактивний та зручний інтерфейс.

Сервер – центральний компонент, реалізований за допомогою Node.js та фреймворку Express [10]. Сервер обробляє запити від клієнта, виконує бізнес-

логіку (наприклад, створення оголошень, обробку фотографій, авторизацію через JWT), взаємодіє з базою даних SQLite та повертає відповідні дані клієнту. Також сервер забезпечує безпечне зберігання даних, таких як хешовані паролі, та керує завантаженням файлів (зображень оголошень) за допомогою middleware Multer.

Шаблон клієнт-сервер ідеально підходить для веб-додатку «Електронна дошка оголошень» з кількох причин. По-перше, розподіл обов'язків полягає в тому, що клієнтська частина, створена на основі React, відповідає за відображення інтерфейсу користувача та локальну обробку даних, таких як вибір кольорової палітри або попередній перегляд фотографій. Серверна частина, реалізована на платформі Express, займається обробкою запитів, управлінням базою даних і забезпеченням бізнес-логіки системи. Це значно спрощує розробку, тестування та масштабування проєкту, адже дозволяє незалежно вдосконалювати клієнтську та серверну частини. По-друге, безпека забезпечується тим, що всі чутливі операції, зокрема авторизація, управління користувачами та обробка запитів на преміум-статус, виконуються на сервері. Наприклад, паролі хешуються за допомогою бібліотеки bcrypt, а доступ до даних контролюється через JWT-токени та спеціальні middleware для авторизації. Контактна інформація оголошень приховується від неавторизованих користувачів, що підвищує рівень захисту даних. По-третє, продуктивність клієнтської частини підтримується завдяки React, який ефективно оновлює інтерфейс через віртуальний DOM, знижуючи навантаження на браузер. Серверна частина оптимізує обробку запитів, застосовуючи базу даних SQLite, яка підходить для проєктів такого масштабу, та швидкий фреймворк Express. Крім того, використання пагінації та фільтрації оголошень зменшує обсяг даних, що передаються між клієнтом і сервером, покращуючи швидкість відгуку системи. Отже, архітектурний шаблон клієнт-сервер є оптимальним вибором для реалізації веб-додатку «Електронна дошка оголошень», адже він забезпечує чітке розділення функцій, безпечну обробку даних і можливість подальшого

масштабування, що відповідає потребам проєкту і гарантує його ефективну розробку та підтримку в майбутньому.

Спросить ChatGPT

1.4. Обґрунтування вибору інструментальних засобів та вимоги до апаратного забезпечення

При розробці веб-додатку електронної дошки оголошень ключовим завданням є вибір інструментальних засобів, які забезпечують швидкість розробки, надійність системи та можливість масштабування. Враховуючи специфіку проєкту, необхідно було обрати технології, що підтримують динамічну обробку даних, зручне керування користувацькими оголошеннями та інтеграцію з базами даних для зберігання структурованої інформації.

Для серверної частини обрано платформу Node.js, яка використовує JavaScript для створення високопродуктивних мережових додатків [11]. Node.js вирізняється асинхронною моделлю обробки запитів, що забезпечує швидку реакцію на численні запити користувачів, наприклад, під час перегляду оголошень чи завантаження зображень. Завдяки широкій екосистемі модулів npm, Node.js дозволяє легко інтегрувати додаткові бібліотеки, такі як обробка авторизації чи завантаження файлів, що є важливим для функціоналу дошки оголошень. Крім того, використання JavaScript як для серверної, так і для клієнтської частини спрощує розробку, оскільки команда може працювати в єдиній мові програмування.

На сервері використано фреймворк Express, який є стандартом для Node.js завдяки своїм численным перевагам. По-перше, він відзначається простотою та гнучкістю, оскільки пропонує мінімалістичний підхід до створення API, що дозволяє швидко налаштовувати маршрути для обробки запитів, таких як створення оголошень або авторизація користувачів. По-друге, Express підтримує модульність завдяки можливості підключення різних middleware, що включають обробку CORS, парсинг JSON та роботу з файлами за допомогою бібліотеки multer. По-третє, ефективна маршрутизація дозволяє чітко структурувати API-

ендпоінти для взаємодії з оголошеннями, фотографіями, чатами та користувачами. Нарешті, активна спільнота користувачів і розробників забезпечує великий доступ до документації та плагінів, що значно спрощує розв’язання проблем і впровадження нових функцій у проєкт.

Для бази даних обрано SQLite через її легкість, компактність і достатню функціональність для проєкту середнього масштабу, такого як дошка оголошень [13]. SQLite не вимагає окремого серверного процесу, що спрощує розгортання додатку на локальних чи хмарних серверах. Реляційна модель SQLite ідеально підходить для структурованих даних, таких як таблиці користувачів, оголошень, категорій, фотографій і чатів. Бібліотека `sqlite3` забезпечує зручну інтеграцію з `Node.js`, дозволяючи виконувати запити асинхронно через модуль `sqlite`. Хоча SQLite менш масштабована порівняно з PostgreSQL чи MongoDB, її продуктивність достатня для початкового етапу проєкту з можливістю переходу на більш потужну базу даних у майбутньому.

На клієнтській стороні використано `React.js`, який є оптимальним вибором для створення інтерактивних інтерфейсів. Основні переваги цього фреймворку полягають у компонентному підході, що дозволяє створювати модульні елементи, такі як картки оголошень, форми редагування або чати, що значно спрощує підтримку і розширення коду. Висока продуктивність досягається завдяки віртуальному DOM, який забезпечує швидке оновлення інтерфейсу при зміні даних, що особливо важливо для динамічних списків оголошень чи чатів у реальному часі. Крім того, `React` відзначається гнучкістю, що дає змогу легко інтегруватися з бібліотеками для маршрутизації, управління станом або HTTP-запитів, наприклад, `fetch`, який використовується для взаємодії з API у проєкті. Популярність `React` пояснюється великою кількістю навчальних матеріалів і готових компонентів, що значно прискорює процес розробки, особливо для проєктів із обмеженим часом. Порівняно з `Angular`, який є більш складним і вимагає глибшого вивчення, або `Vue.js`, що має меншу екосистему, `React` забезпечує баланс між простотою, продуктивністю та доступністю ресурсів для команди з обмеженим складом. Використання бібліотеки `htm` для створення JSX-

подібного синтаксису без необхідності транспіляції через Babel додатково спрощує налаштування проєкту, зменшуючи залежності.

Для обробки зображень використано модуль `multer`, який забезпечує зручне завантаження файлів на сервер, а також бібліотеку `axios` у скрипті `seed.js` для завантаження тестових зображень із зовнішніх джерел [15]. Для авторизації та безпеки використано `jsonwebtoken` (JWT) для створення токенів і `bcrypt` для хешування паролів, що забезпечує захист даних користувачів.

Щодо апаратного забезпечення, веб-додаток не потребує значних ресурсів. Для серверної частини достатньо сервера з 2-4 ГБ оперативної пам'яті, двоядерним процесором і SSD-диском на 20-50 ГБ для зберігання бази даних і зображень. `SQLite` працює ефективно навіть на мінімальних конфігураціях, а `Node.js` із `Express` оптимізовано для низького споживання ресурсів. Для розробки та тестування вистачить сучасного ПК чи ноутбука з 8 ГБ ОЗУ, процесором `Intel Core i3` або еквівалентним і стабільним інтернет-з'єднанням. У майбутньому, при зростанні кількості користувачів, систему можна масштабувати на хмарні платформи, такі як `AWS`, `Google Cloud` чи `Render`, із можливим переходом на `PostgreSQL` або `MongoDB` для обробки великих обсягів даних.

Таким чином, вибір `Node.js` із `Express`, `SQLite` та `React` обґрунтований їх ефективністю, простотою інтеграції та відповідністю вимогам проєкту електронної дошки оголошень. Ці інструменти забезпечують швидку розробку, надійну роботу системи та можливість подальшого масштабування.

Висновки до розділу 1

Розділ 1 присвячений аналізу задачі та особливостей використання інформаційних технологій для створення веб-додатку електронної дошки оголошень. У ході дослідження було сформульовано завдання розробки зручної платформи для розміщення, перегляду та керування оголошеннями, яка підтримує клієнтські та менеджерські функції. Проведений аналіз аналогів (`OLX`, `Prom.ua`, `eBay Classifieds`, `Gumtree`) виявив сильні сторони (зручні фільтри, широкий охват) та слабкі (обмежені менеджерські функції, брак локалізації), що

дозволило визначити конкурентні переваги розробленої системи: локалізацію, чати, кольорові палітри та розширені адміністративні можливості. Обґрунтовано вибір архітектури клієнт-сервер, яка забезпечує чіткий розподіл функцій, безпеку (JWT, bcrypt) та продуктивність (React, SQLite). Вибір інструментів (Node.js, Express, React, SQLite) зумовлений їх ефективністю, простотою інтеграції та підтримкою масштабування. Вимоги до апаратного забезпечення мінімальні (2-4 ГБ ОЗУ, SSD 20-50 ГБ), що робить додаток доступним для розгортання. Отримані результати створюють міцну основу для реалізації проєкту, забезпечуючи баланс між функціональністю, безпекою та зручністю використання.

РОЗДІЛ 2. ПРОЄКТУВАННЯ ВЕБ-ДОДАТКУ

2.1. Визачення варіантів використання та структури системи

Веб-додаток «Дошка оголошень» розроблено для забезпечення зручного створення, перегляду та управління оголошеннями для користувачів, а також адміністрування системи менеджерами. Система орієнтована на два типи користувачів: клієнтів, які можуть розміщувати та переглядати оголошення, і менеджерів, які мають адміністративні функції для управління користувачами та їх запитам. Додаток підтримує інтерактивний інтерфейс, безпечну автентифікацію та обробку даних, включаючи завантаження зображень і чати для комунікації.

Вимоги користувачів до основних функцій системи охоплюють як клієнтську, так і адміністративну частини. Усі користувачі клієнтської частини можуть реєструватися і проходити автентифікацію, переглядати список оголошень, здійснювати пошук за ключовими словами, категоріями та ціною, а також користуватися функціями фільтрації та сортування за ціною, датою і категорією. Вони також мають змогу переглядати деталі оголошення, включаючи фотографії та контактну інформацію, якщо пройшли автентифікацію. Після автентифікації користувачі отримують додаткові можливості, зокрема створення, редагування і видалення власних оголошень, завантаження до них фотографій, надсилання запиту на преміум-статус, перегляд власних оголошень, участь у чатах із менеджерами та вихід із системи. Менеджери, які мають доступ до адміністративної частини, можуть переглядати список усіх користувачів, видаляти користувачів, надавати або скасовувати преміум-статус, переглядати та відповідати на повідомлення у чатах, а також видаляти оголошення і виходити з системи.

До функціональних вимог належить реалізація REST API для роботи з базою даних користувачів, оголошень, фотографій, категорій та чатів. Передбачена можливість пошуку оголошень за ключовими словами, категоріями та ціною. Система має підтримувати завантаження до восьми фотографій на одне

оголошення з обмеженням розміру кожного файлу до п'яти мегабайт. Звичайні користувачі можуть створити до п'яти оголошень, а преміум-користувачі — необмежену кількість. Також передбачена система чатів для комунікації між клієнтами та менеджерами і нотифікації про успішність дій, наприклад, після створення оголошення чи надсилання запиту на преміум-статус.

Нефункціональні вимоги охоплюють кілька аспектів. Щодо сприйняття користувачами, час навчання роботі з додатком для клієнтів не повинен перевищувати десяти хвилин, а для менеджерів — двадцяти. Час відповіді системи на стандартні запити, такі як перегляд або пошук оголошень, має бути не більше двох секунд, а на складні дії, як-от створення оголошення чи завантаження фото — до п'яти секунд. Інтерфейс повинен бути інтуїтивно зрозумілим, мати логічну навігацію та підтримувати українську мову. У плані продуктивності система має справно функціонувати при одночасній роботі кількох десятків користувачів без помітних затримок. Вимоги до сумісності передбачають коректну роботу додатка в усіх сучасних браузерях, зокрема Chrome, Firefox та Safari. Надійність реалізується завдяки використанню стабільної бази даних SQLite з каскадним видаленням пов'язаних записів, наприклад, автоматичним видаленням фотографій при видаленні оголошення.

Аналіз вимог дозволяє виділити ключові сутності системи: користувачі (клієнти та менеджери), оголошення, категорії, фотографії, чати та повідомлення. Розглянемо, як користувачі взаємодіють із системою.

Варіанти використання для клієнтської частини зображені на рис. 2.1. Клієнт може зареєструватися або увійти в систему. Йому доступний перегляд списку оголошень із можливістю пошуку та фільтрації за категоріями, ціною чи ключовими словами. Автентифікований клієнт може створювати, редагувати та видаляти власні оголошення, додавати до них фотографії, обирати кольорову палітру для оформлення, а також подавати запит на преміум-статус. У чатах клієнт може комунікувати з менеджерами щодо статусу запиту. Преміум-користувачі мають переваги: необмежену кількість оголошень і пріоритетне відображення їх оголошень у списку.

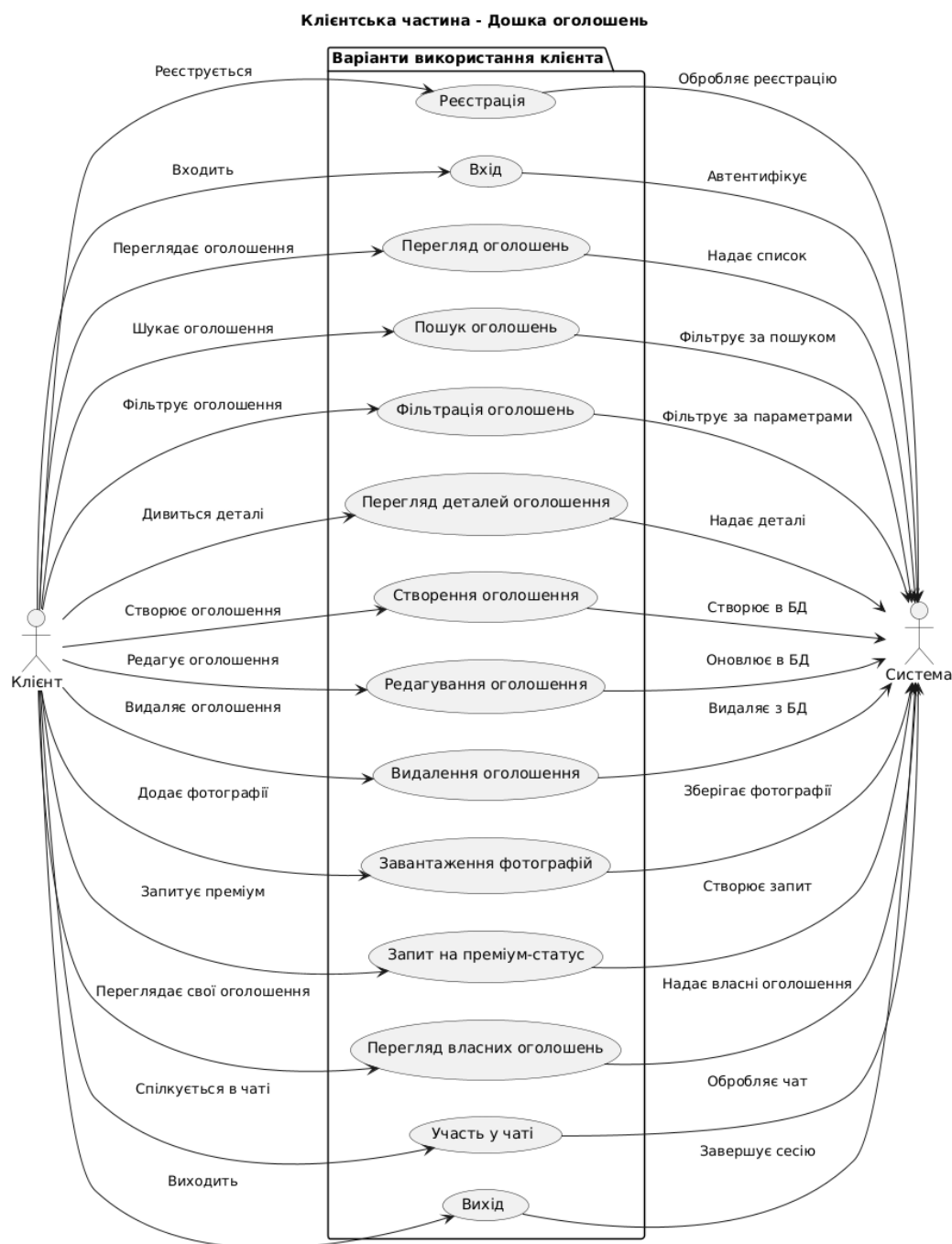


Рис. 2.1 – Діаграма варіантів використання для клієнтської частини

Варіанти використання для адміністративної частини показані на рис. 2.2. Менеджер після автентифікації отримує доступ до управління користувачами: перегляд списку, видалення користувачів, надання або скасування преміум-статусу. Менеджер також може переглядати та відповідати на запити в чатах, зокрема схвалювати чи відхиляти запити на преміум-статус. Крім того, менеджери можуть видаляти будь-які оголошення, якщо це необхідно.

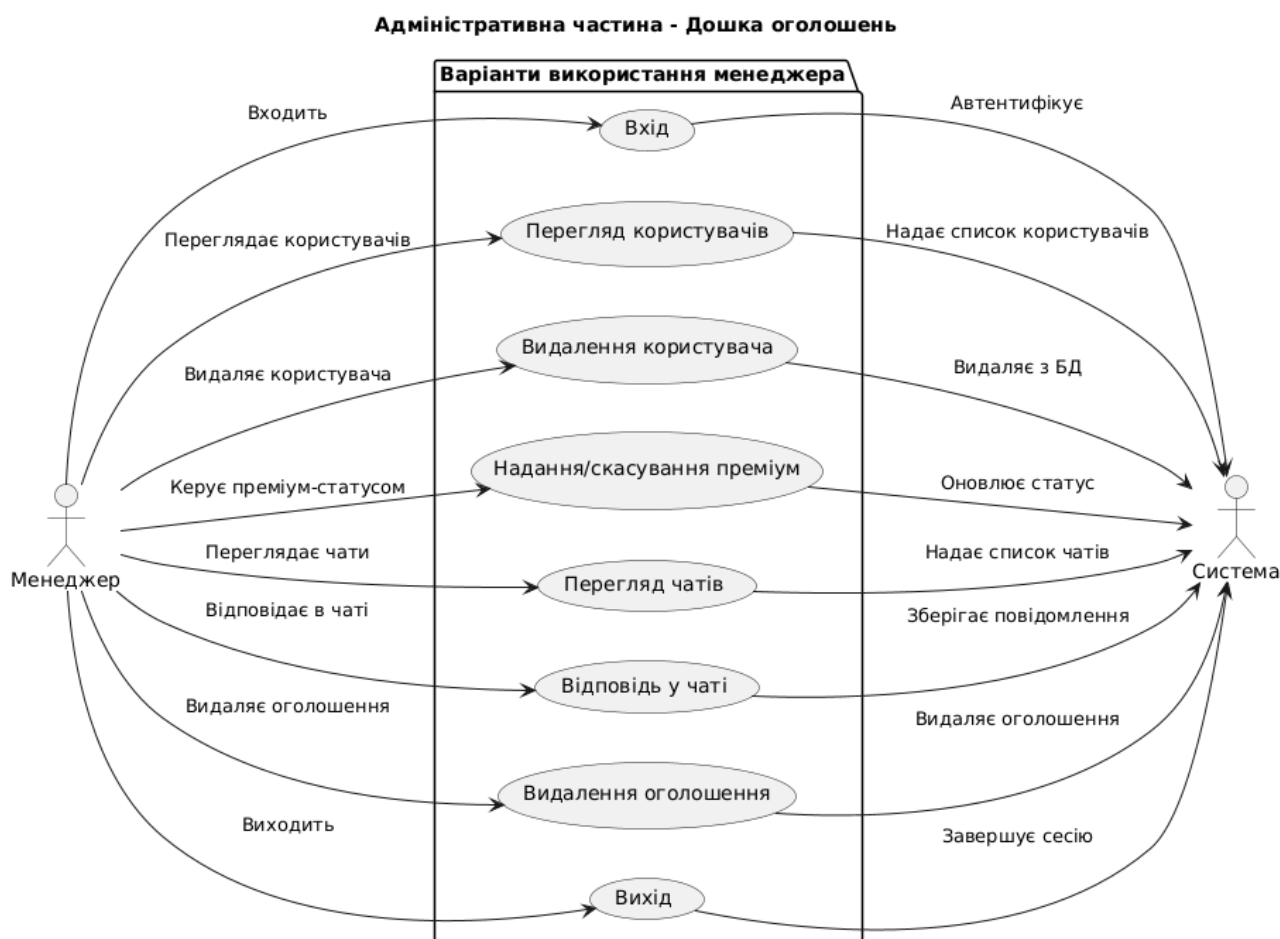


рис. 2.2 – Діаграма варіантів використання для адміністративної частини

Архітектура системи базується на клієнт-серверному підході. Фронтенд, реалізований за допомогою React, забезпечує інтерактивний інтерфейс із компонентами для створення оголошень, управління чатами та перегляду даних. Бекенд, побудований на Node.js із Express, обробляє API-запити та взаємодіє з базою даних SQLite. Для автентифікації використовується JWT (JSON Web Tokens), а для безпечного зберігання паролів — bcrypt [16, 17]. Файли зображень зберігаються на сервері в папці public/uploads, а їх метадані — у базі даних.

Таким чином, веб-додаток «Дошка оголошень» забезпечує повноцінний цикл взаємодії: від створення та перегляду оголошень для клієнтів до управління користувачами та їх запитам для менеджерів, відповідаючи функціональним і нефункціональним вимогам.

2.2. Розробка структури бази даних

У процесі створення веб-додатку електронної дошки оголошень ключовим етапом є розробка структури бази даних, яка забезпечує ефективне зберігання, організацію та доступ до даних, необхідних для функціонування системи. Для реалізації проекту було обрано реляційну базу даних SQLite, яка є легкою, вбудованою та ідеально підходить для веб-додатків середнього масштабу. SQLite підтримує реляційну модель даних, що дозволяє створювати зв'язки між таблицями через зовнішні ключі, забезпечуючи цілісність даних і логічну організацію. Структура бази даних була розроблена з урахуванням основних потреб системи, таких як управління користувачами, оголошеннями, фотографіями, категоріями, чатами та повідомленнями.

База даних містить шість основних таблиць: users, categories, announcements, photos, chats і messages (рис. 2.3). Кожна таблиця відповідає за зберігання певного типу інформації та пов'язана з іншими таблицями через зовнішні ключі, що забезпечує цілісність даних і функціональність системи.

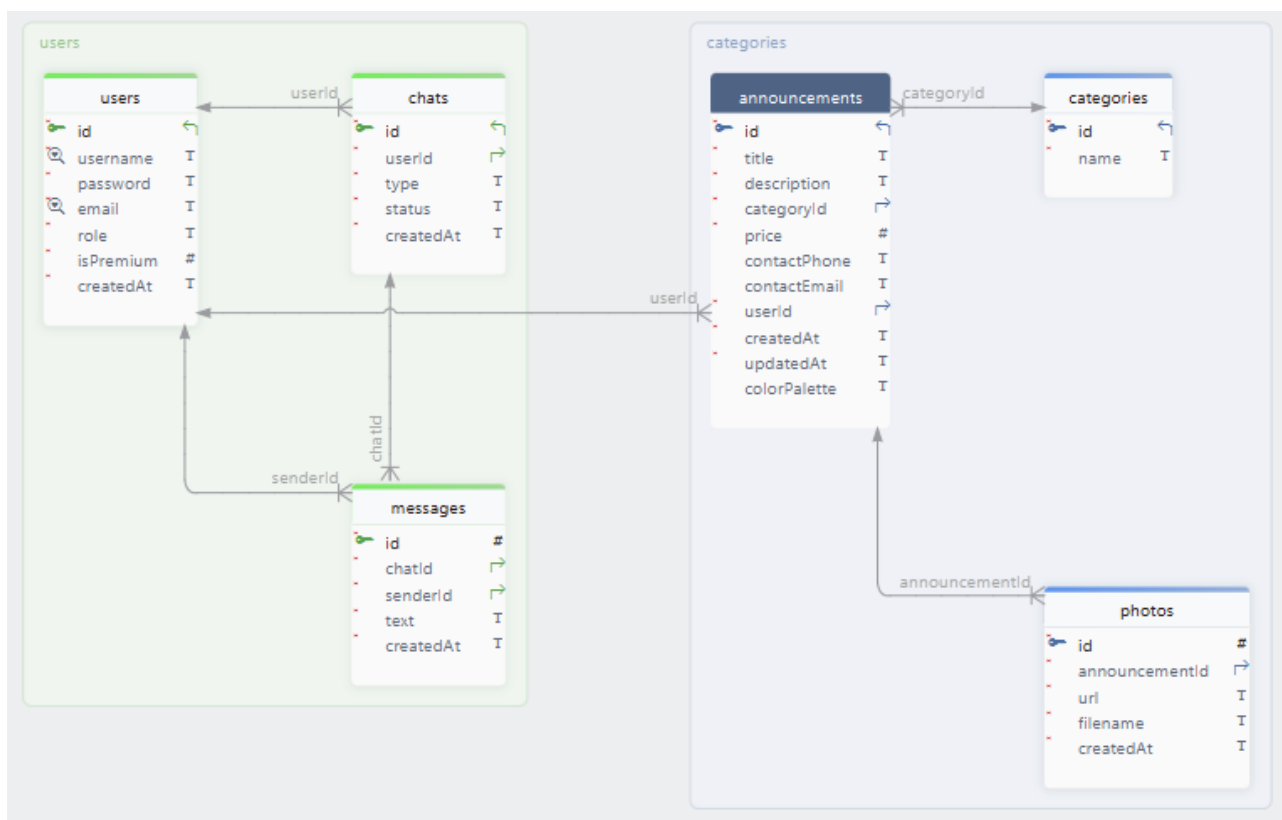


Рис. 2.3 – Структура бази даних

Таблиця users призначена для зберігання даних про зареєстрованих користувачів, включаючи клієнтів і менеджерів. Поля таблиці:

- id – цілочисельне поле, первинний ключ з автоінкрементом, унікальний ідентифікатор користувача;
- username – текстове поле, унікальний логін користувача, обов'язкове для заповнення;
- password – текстове поле, яке зберігає хешований пароль користувача для безпечного входу;
- email – текстове поле, унікальна електронна пошта користувача, обов'язкове для заповнення;
- role – текстове поле, яке визначає роль користувача (client для клієнтів або manager для менеджерів);
- isPremium – цілочисельне поле (0 або 1), яке вказує, чи має користувач преміум-статус;
- createdAt – текстове поле, яке зберігає дату і час створення облікового запису у форматі ISO.

Таблиця categories містить інформацію про категорії оголошень, які використовуються для їх класифікації. Поля таблиці:

- id – цілочисельне поле, первинний ключ з автоінкрементом, унікальний ідентифікатор категорії;
- name – текстове поле, яке містить назву категорії (наприклад, "Нерухомість", "Транспорт"), обов'язкове для заповнення.

Таблиця announcements відповідає за зберігання даних про оголошення, створені користувачами. Поля таблиці:

Таблиця 2.1

Поле	Тип даних	Обов'язкове	Опис
id	INTEGER	Так	Унікальний ідентифікатор оголошення, первинний ключ з автоінкрементом

title	TEXT	Так	Заголовок оголошення
description	TEXT	Так	Детальний опис оголошення
categoryId	INTEGER	Так	Зовнішній ключ, посилання на id у таблиці categories
price	REAL	Так	Ціна товару або послуги в гривнях
contactPhone	TEXT	Ні	Контактний номер телефону
contactEmail	TEXT	Ні	Контактна електронна пошта
userId	INTEGER	Так	Зовнішній ключ, посилання на id у таблиці users
createdAt	TEXT	Так	Дата і час створення у форматі ISO
updatedAt	TEXT	Так	Дата і час останнього оновлення
colorPalette	TEXT (JSON)	Ні	JSON-об'єкт з кольоровою палітрою для оформлення

Таблиця photos зберігає інформацію про фотографії, які додаються до оголошень. Поля таблиці:

Таблиця 2.2

Поле	Тип даних	Обов'язкове	Опис
id	INTEGER	Так	Унікальний ідентифікатор фотографії, первинний ключ з автоінкрементом
announcementId	INTEGER	Так	Зовнішній ключ, посилання на id у таблиці announcements
url	TEXT	Так	Шлях до файлу фотографії на сервері
filename	TEXT	Так	Ім'я файлу фотографії
createdAt	TEXT	Так	Дата і час завантаження у форматі ISO

Таблиця chats відповідає за зберігання інформації про чати між користувачами та менеджерами, зокрема запити на преміум-статус. Поля таблиці:

Таблиця 2.3

Поле	Тип даних	Обов'язкове	Опис
id	INTEGER	Так	Унікальний ідентифікатор чату, первинний ключ з автоінкрементом
userId	INTEGER	Так	Зовнішній ключ, посилання на id у таблиці users
type	TEXT	Так	Тип чату, наприклад premium-request для запитів на преміум-статус
status	TEXT	Так	Статус чату, наприклад pending або responded
createdAt	TEXT	Так	Дата і час створення чату у форматі ISO

Реляційні зв'язки між таблицями забезпечують цілісність даних і дозволяють ефективно отримувати пов'язану інформацію. Наприклад, таблиця announcements пов'язана з таблицею users через поле userId та з таблицею categories через поле categoryId. Таблиця photos пов'язана з announcements через поле announcementId з каскадним видаленням, що означає, що фотографії автоматично видаляються при видаленні відповідного оголошення. Таблиця chats пов'язана з users через поле userId, а таблиця messages пов'язана з chats через поле chatId і з users через поле senderId. Така структура зв'язків дозволяє підтримувати цілісність даних і спрощує роботу з пов'язаною інформацією в базі даних. SQLite була обрана для цього проєкту через її простоту, відсутність необхідності в окремому сервері баз даних і достатню продуктивність для потреб електронної дошки оголошень. Структура бази даних дозволяє ефективно керувати оголошеннями, користувачами, фотографіями та комунікацією, забезпечуючи гнучкість і масштабованість системи.

2.3. Реалізація функціоналу веб-додатку

Процес реалізації функціоналу веб-додатку електронної дошки оголошень розпочинається з розробки серверної частини, яка забезпечує обробку запитів, взаємодію з базою даних та передачу даних на клієнтську сторону. Веб-додаток побудовано з використанням серверної логіки на Node.js з фреймворком Express.js та клієнтської частини на React, що забезпечує інтерактивний інтерфейс для користувачів.

Серверна частина реалізована у файлі `server.js`. Першим кроком є створення структури бази даних із використанням SQLite через бібліотеку `sqlite3`. У файлі визначено ініціалізацію таблиць, таких як `users`, `categories`, `announcements`, `photos`, `chats` і `messages`. Наприклад, схема таблиці `announcements` включає поля для зберігання інформації про оголошення: `id`, `title`, `description`, `categoryId`, `price`, `contactPhone`, `contactEmail`, `userId`, `createdAt`, `updatedAt` і `colorPalette`. Поля `userId` і `categoryId` пов'язані з таблицями `users` і `categories` через зовнішні ключі, що забезпечує цілісність даних.

Кожен стовпець таблиці визначає тип даних, а зв'язки між таблицями дозволяють ефективно отримувати пов'язані дані, наприклад, інформацію про користувача, який створив оголошення, або категорію оголошення. Для заповнення бази даних початковими тестовими даними використовується файл `seed.js`, який створює користувачів, оголошення, категорії та завантажує зображення з URL.

Серверна логіка побудована на Express.js. API-ендпоінти, такі як `/api/announcements` для отримання списку оголошень чи `/api/auth/register` для реєстрації користувачів, обробляють запити клієнта. Наприклад, створення оголошення включає перевірку ліміту для звичайних користувачів (не більше 5 оголошень) і збереження даних у базі (рис. 2.4).

```

app.post('/api/announcements', authMiddleware, roleMiddleware(['client']), async
(req, res) => {
  if (!req.user.isPremium) {
    const userAnnouncementsCount = await db.get(
      'SELECT COUNT(*) as count FROM announcements WHERE userId = ?',
      req.user.id
    );
    if (userAnnouncementsCount.count >= 5) {
      return res.status(403).json({ message: 'Ви досягли ліміту оголошень...' });
    }
  }
  const { title, description, categoryId, price, contactPhone, contactEmail,
    colorPalette } = req.body;
  const now = new Date().toISOString();
  const colorPaletteJson = colorPalette ? JSON.stringify(colorPalette) : null;
  const result = await db.run(
    'INSERT INTO announcements (...) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)',
    [title, description, parseInt(categoryId), parseFloat(price), contactPhone,
    contactEmail, req.user.id, now, now, colorPaletteJson]
  );
  res.status(201).json({ ...newAnnouncement, colorPalette: parsedColorPalette,
    photos: [] });
});

```

Рис. 2.4 – Контролер для створення оголошення

Для безпеки паролі користувачів хешуються за допомогою бібліотеки bcrypt перед збереженням у базі даних, а авторизація реалізована через JSON Web Tokens (JWT). Наприклад, при авторизації перевіряється валідність пароля (рис. 2.5).

```

app.post('/api/auth/login', async (req, res) => {
  const { username, password } = req.body;
  const user = await db.get('SELECT * FROM users WHERE username = ?', username);
  if (!user || !await bcrypt.compare(password, user.password)) {
    return res.status(401).json({ message: 'Невірний логін або пароль' });
  }
  const token = jwt.sign({ id: user.id, username: user.username, role: user.role,
    isPremium: user.isPremium === 1 }, JWT_SECRET, { expiresIn: '24h' });
  res.json({ message: 'Авторизація успішна', token, user: { id: user.id, username:
    user.username, email: user.email, role: user.role, isPremium: user.isPremium === 1 }
  });
});

```

Рис. 2.5 – Функція авторизації

Клієнтська частина реалізована у файлі app.js з використанням React. Основний компонент App керує маршрутизацією та станом авторизації через контекст AuthContext. Стан користувача зберігається в локальному сховищі браузера для збереження між сесіями (рис. 2.6).

```
const loadUser = () => {
  const storedUser = localStorage.getItem('user');
  const storedToken = localStorage.getItem('token');
  return storedUser && storedToken ? { user: JSON.parse(storedUser), token:
storedToken } : { user: null, token: null };
};
const [auth, setAuth] = React.useState(loadUser());
```

Рис. 2.6 – Завантаження даних із localStorage

Функціонал додатку включає авторизацію, створення та редагування оголошень, завантаження фотографій, запит на преміум-статус, чати з менеджерами та керування користувачами для менеджерів. Наприклад, компонент CreateAnnouncementPage дозволяє клієнтам створювати нові оголошення з вибором категорії, кольорової палітри та завантаженням зображень (рис. 2.7).

```
const handleSubmit = async (e) => {
  e.preventDefault();
  setLoading(true);
  try {
    const response = await fetch(`${API_URL}/announcements`, {
      method: 'POST',
      headers: { 'Content-Type': 'application/json', Authorization: `Bearer
${token}` },
      body: JSON.stringify({ ...formData, colorPalette })
    });
    const data = await response.json();
    if (!response.ok) throw new Error(data.message || 'Помилка створення
оголошення');
    alert('Оголошення успішно створено');
    onNavigate(null, 'announcement-details', { id: data.id });
  } catch (err) {
    setError(err.message);
  } finally {
    setLoading(false);
  }
};
```

Рис. 2.7 – Функція створення оголошення

Для менеджерів реалізована можливість керування користувачами через компонент AdminUsersPage, який дозволяє видаляти користувачів або змінювати їх преміум-статус

Пошук оголошень реалізований через фільтри на клієнтській стороні та серверні запити з параметрами, такими як категорія, ціна чи пошуковий запит.

Хук `useEffect` використовується для асинхронного завантаження даних, наприклад, списку оголошень чи чатів, при зміні залежностей, таких як токен авторизації чи параметри пошуку (рис. 2.8).

```
React.useEffect(() => {
  handleSearch();
}, [search, category, minPrice, maxPrice, sort, page, token]);
```

Рис. 2.8– Завантаження даних через `useEffect`

Кінцеві точки API, визначені в `server.js`, забезпечують основний функціонал додатку. Основні з них для оголошень наведено в таблиці 2.1.

Таблиця 2.4

Кінцеві точки для оголошень

Кінцева точка	HTTP-метод	Опис
/api/announcements	GET	Отримання списку оголошень
/api/announcements	POST	Створення нового оголошення
/api/announcements/:id	GET	Отримання оголошення за ID
/api/announcements/:id	PUT	Оновлення оголошення
/api/announcements/:id	DELETE	Видалення оголошення
/api/announcements/:id/photos	POST	Завантаження фотографій до оголошення

Таким чином, реалізація функціоналу веб-додатку електронної дошки оголошень охоплює серверну логіку з обробкою запитів, базою даних SQLite та клієнтську частину з інтерактивним інтерфейсом на React. Додаток забезпечує авторизацію, створення та керування оголошеннями, підтримку чатів і адміністрування користувачів, відповідаючи вимогам проєкту та надаючи зручний інструмент для розміщення та пошуку оголошень.

ВИСНОВКИ ДО РОЗДІЛУ 2

У другому розділі проведено проєктування веб-додатку «Дошка оголошень», визначено вимоги користувачів до функціоналу системи та розроблено діаграми варіантів використання для клієнтської й адміністративної

частин. На основі цих даних створено чітку структуру системи, що враховує потреби клієнтів і менеджерів.

Розроблено файлову структуру, яка розділяє серверну частину (Node.js, Express.js) та клієнтську (React), забезпечуючи модульність і зручність підтримки. Описано організацію модулів, зокрема server.js для API, seed.js для тестових даних та app.js для інтерфейсу.

Структуру бази даних реалізовано з використанням SQLite, визначено таблиці (users, categories, announcements, photos, chats, messages) із реляційними зв'язками для ефективного зберігання даних і забезпечення цілісності.

Функціонал веб-додатку успішно реалізовано: від авторизації й пошуку оголошень до створення, редагування оголошень, управління чатами та адміністрування користувачів. Описано алгоритми ключових функцій, таких як створення оголошень і обробка запитів на преміум-статус, із прикладами коду.

Таким чином, розділ охоплює повний цикл проектування, створюючи основу для ефективної роботи системи дошки оголошень.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З ВЕБ-ДОДАТКОМ

3.1. Порядок встановлення та налаштування системи

Розроблений веб-додаток "Електронна дошка оголошень" побудовано за архітектурою клієнт-сервер, що забезпечує відокремлення серверної логіки від клієнтської частини. Серверна частина, реалізована на Node.js з використанням Express, відповідає за обробку запитів, взаємодію з базою даних SQLite та надання API. Клієнтська частина, побудована на React, забезпечує інтерактивний інтерфейс користувача через веб-браузер. Для коректної роботи системи рекомендується використовувати сервер із такими мінімальними характеристиками:

- двоядерний процесор із частотою 2 ГГц або вище;
- 2 ГБ оперативної пам'яті;
- 1 ГБ вільного місця на диску для бази даних, файлів додатку та завантажених зображень.

Для встановлення та запуску серверної частини виконайте наступні кроки:

- Встановіть середовище Node.js (версія 14 або вище) на сервері.
- Склонуйте репозиторій проєкту або розпакуйте архів із кодом у вибрану директорію.
- У кореневій папці проєкту виконайте команду `npm install` у терміналі, щоб встановити всі залежності, зазначені у файлі `package.json`. Залежності включають Express, SQLite3, Multer, JWT, Bcrypt та інші.
- Запустіть сервер командою `node server.js`. Сервер буде доступний за адресою `http://localhost:3002` (порт можна змінити у коді, змінивши змінну `port` у файлі `server.js`).
- Для ініціалізації бази даних та створення необхідних таблиць сервер автоматично виконає скрипт із функції `initializeDatabase` у `server.js`. Щоб заповнити базу тестовими даними, виконайте команду `node seed.js`, яка створить користувачів, категорії, оголошення та фотографії.

Клієнтська частина розміщується у папці public і не потребує окремого сервера, оскільки статичні файли (HTML, CSS, JavaScript) обслуговуються сервером Express. Для роботи клієнтської частини необхідний сучасний веб-браузер, наприклад:

- Google Chrome (версія 90+);
- Mozilla Firefox (версія 85+);
- Microsoft Edge (версія 90+).

Перед запуском переконайтеся, що папка public/uploads існує для зберігання завантажених зображень. Якщо папка відсутня, сервер автоматично створить її під час першого завантаження зображення. Після виконання всіх кроків додаток готовий до роботи: відкрийте <http://localhost:3002> у браузері, щоб отримати доступ до головної сторінки.

3.2. Структура інтерфейсу. Інтерфейс та порядок роботи з системою

Веб-додаток "Електронна дошка оголошень" має інтуїтивно зрозумілий інтерфейс, який дозволяє користувачам переглядати, створювати, редагувати та видаляти оголошення, а також взаємодіяти з системою через чати та запити на преміум-статус. Неавтентифіковані користувачі можуть переглядати оголошення, але для доступу до повного функціоналу (створення оголошень, редагування, чати) потрібна автентифікація.

На кожній сторінці присутня верхня навігаційна панель (рис. 3.1, рис. 3.2), яка містить:

- логотип "Дошка оголошень", що перенаправляє на головну сторінку;
- посилання на основні розділи: "Оголошення", "Мої оголошення" (для автентифікованих користувачів), "Чати" та "Преміум-статус" (для клієнтів);
- для менеджерів додатково відображається пункт "Управління користувачами";
- панель автентифікації: кнопки "Вхід" і "Реєстрація" для неавтентифікованих користувачів або ім'я користувача та кнопка "Вийти" для автентифікованих.

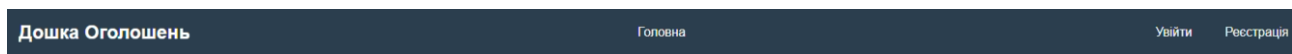


Рис. 3.1 – Навігаційна панель для неавтентифікованих користувачів



Рис. 3.2 – Навігаційна панель для автентифікованих користувачів

У нижній частині кожної сторінки розташований футер (рис. 3.3), який містить контактну інформацію, посилання на соціальні мережі та інформацію про авторські права. Футер є статичним і забезпечує зручний доступ до додаткової інформації.

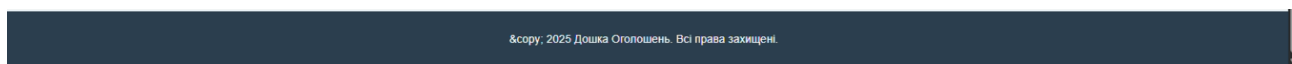


Рис. 3.3 – Футер сторінки

Головна сторінка, зображена на рисунку 3.4, відображає сітку оголошень із можливістю зручної фільтрації за категоріями, ціною, пошуковим запитом, а також сортуванням за ціною або датою. Фільтри розташовані у верхній частині сторінки, що робить їх легко доступними для користувача і дозволяє швидко налаштувати відображення оголошень відповідно до своїх потреб. Самі оголошення представлені у вигляді карток із зображенням, заголовком, ціною та датою публікації, що робить перегляд інформації максимально інформативним і зручним. Для неавтентифікованих користувачів контактні дані, такі як телефон і електронна пошта, приховані, що мотивує їх пройти автентифікацію для отримання повного доступу до інформації. Це також допомагає захистити приватність власників оголошень і зменшує ризик несанкціонованого використання контактних даних. Таким чином, головна сторінка поєднує в собі простоту використання та ефективність у пошуку необхідної інформації.

Дошка оголошень

Фільтри

Пошук:

Категорія:

Мінімальна ціна:

Максимальна ціна:

Сортування:

Пошук за назвою або опі

Всі категорії

Від

До

Спочатку новіші

Скинути фільтри

Новий iPhone 15 Pro

Категорія: Транспорт

Ціна: 32297 грн

Опубліковано: 09.04.2025

Продам новий iPhone 15 Pro 256GB, колір - сірий. На гарантії, повний комплект. (Оголошення №6)

Продам диван

Категорія: Нерухомість

Ціна: 23338 грн

Опубліковано: 04.04.2025

Продам новий iPhone 15 Pro 256GB, колір - сірий. На гарантії, повний комплект. (Оголошення №4)

Немає фото

Шукаю роботу водія

Категорія: Транспорт

Ціна: 10807 грн

Опубліковано: 27.03.2025

Надаю послуги з ремонту квартир: штукатурка, шпаклівка, фарбування, укладання плитки та ламінату

Рис. 3.4 – Головна сторінка

Для пошуку оголошень користувач вводить запит у спеціальне поле пошуку на головній сторінці та натискає кнопку "Пошук". Після цього система обробляє запит і відображає результати, які відповідають введеним критеріям, безпосередньо на тій же сторінці, як показано на рисунку 3.5. Для більш точного підбору оголошень користувач може скористатися фільтрами, які дозволяють обрати потрібну категорію, вказати діапазон цін або відсортувати результати за зростанням або спаданням ціни чи дати публікації. Якщо за заданими параметрами не вдається знайти жодного оголошення, система виводить повідомлення "Оголошень не знайдено", що інформує користувача про відсутність результатів. Такий підхід робить пошук зручним і допомагає швидко знаходити потрібну інформацію навіть при великій кількості оголошень. Крім того, інтерфейс пошуку розроблено таким чином, щоб користувач міг легко змінювати умови пошуку без необхідності повертатися на початкову сторінку.

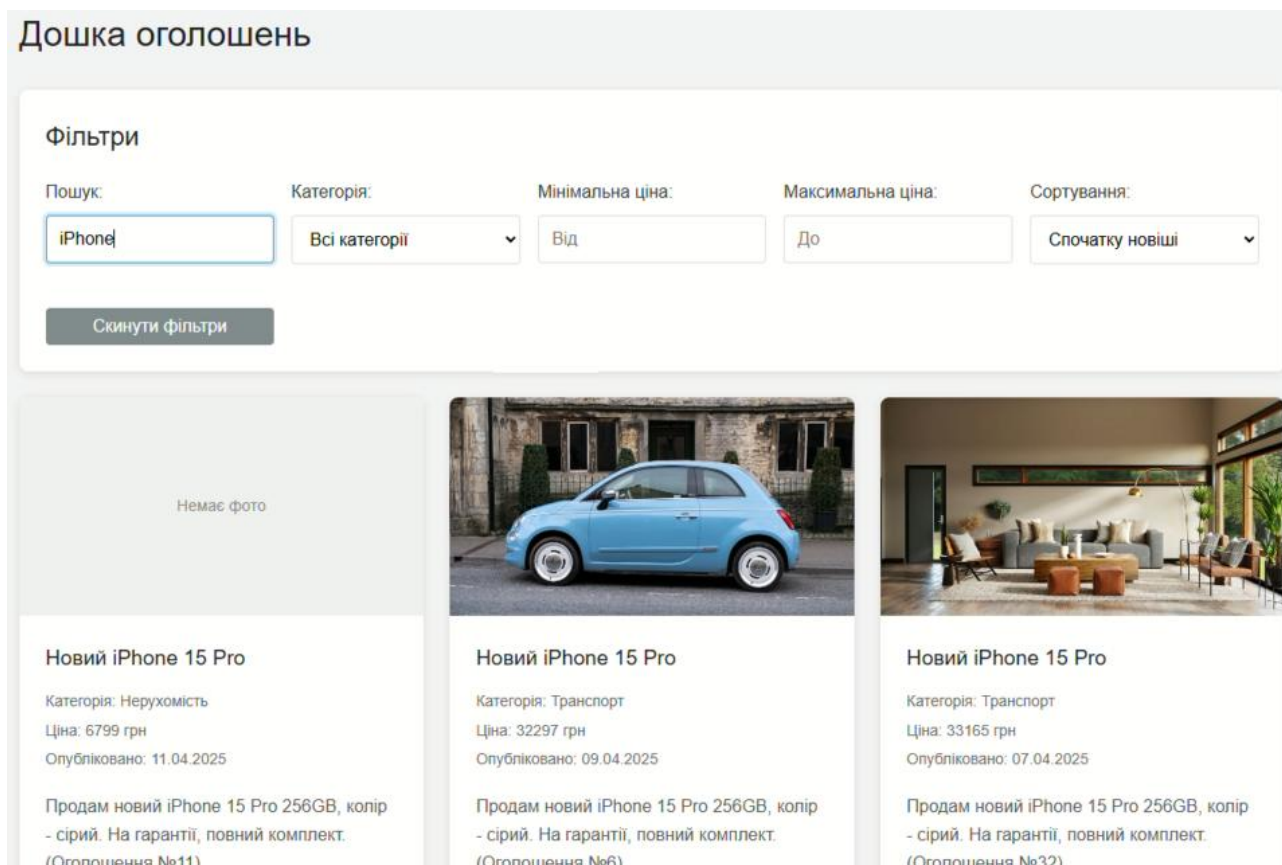


Рис. 3.5 – Сторінка результатів пошуку

При натисканні на картку оголошення відкривається сторінка з детальною інформацією, як показано на рисунку 3.6. Вона містить карусель із фотографіями, де можна переглянути до восьми зображень, що дозволяє краще ознайомитися з товаром або послугою. Також на сторінці відображаються заголовок, опис оголошення, категорія, ціна та контактні дані, які доступні лише автентифікованим користувачам. Якщо користувач обрав кольорову палітру, вона також відображається на цій сторінці, що додає додатковий рівень персоналізації. Крім того, для власника оголошення або менеджера доступні кнопки редагування та видалення, що дає змогу швидко вносити зміни або видаляти оголошення безпосередньо зі сторінки деталей. Такий дизайн забезпечує зручність користування і дозволяє ефективно керувати оголошеннями.

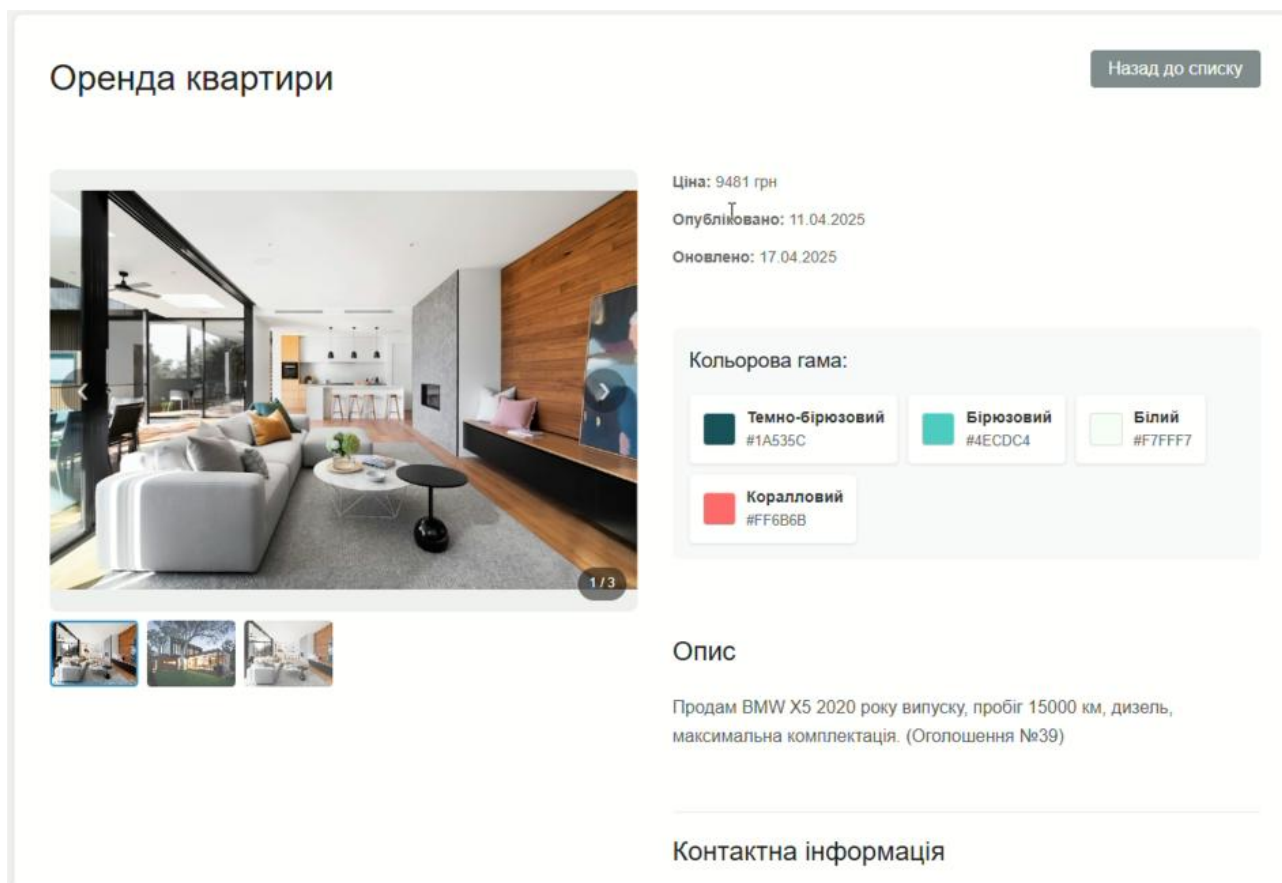


Рис. 3.6 – Сторінка деталей оголошення

Автентифіковані користувачі з роллю "client" можуть створити нове оголошення, натиснувши кнопку "Створити нове оголошення" на сторінці "Мої оголошення", як це показано на рисунку 3.7. Після цього відкривається форма створення оголошення, зображена на рисунку 3.8, яка включає необхідні поля для введення заголовка, опису, вибору категорії, вказання ціни, контактних даних, а також можливість обрати кольорову палітру. Крім того, користувач може завантажити до восьми фотографій, що допомагає зробити оголошення більш привабливим і інформативним. Важливо відзначити, що звичайні користувачі мають обмеження на кількість активних оголошень — не більше п'яти, тоді як преміум-користувачі не стикаються з таким лімітом і можуть розміщувати необмежену кількість оголошень. Такий підхід дозволяє підтримувати баланс між доступністю сервісу та мотивацією користувачів до переходу на преміум-версію.

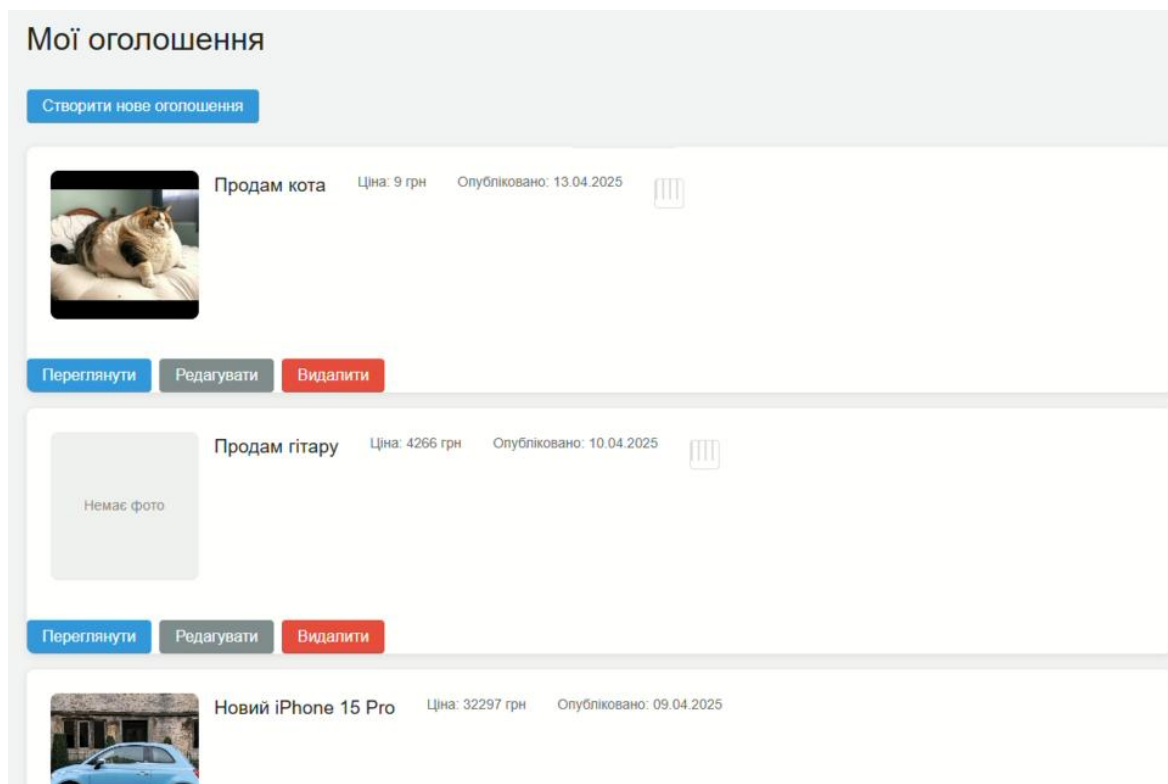


Рис. 3.7 – Сторінка моїх оголошень

Дошка Оголошень

Головна Додати оголошення Мії оголошення Повідомлення Отримати преміум user1 Вийти

Створення нового оголошення

1. Основна інформація 2. Фотографії

Заголовок:
Продам кота

Категорія:
Нерухомість

Опис:
Продам величезного котика, ~~яке~~ як мамонт

Ціна (грн):
9

Телефон:
232323232

Email:
voland@peklo.ua

Кольорова гама

Класична Морська Природня Яскрава

Скасувати Продовжити

Рис. 3.8 – Форма створення оголошення

Для редагування оголошення користувач переходить на сторінку редагування через кнопку "Редагувати" на сторінці деталей або "Мої оголошення" (рис. 3.9). Форма редагування аналогічна формі створення, але заповнена даними існуючого оголошення. Користувач може змінити текст, категорію, ціну, контакти, кольорову палітру або додати/видалити фотографії.

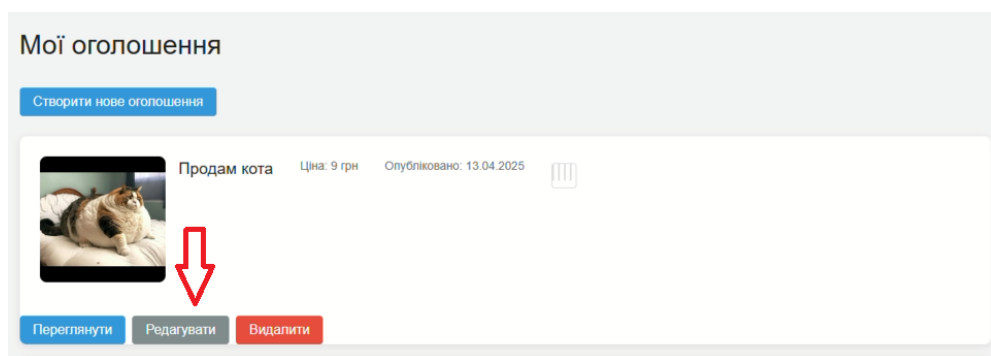


Рис. 3.9 – Кнопка для редагування оголошення

Користувач може видалити оголошення через кнопку "Видалити" (рис. 3.10) на сторінці деталей або "Мої оголошення". Після підтвердження система видаляє оголошення та всі пов'язані фотографії. Менеджери також можуть видаляти оголошення будь-якого користувача.

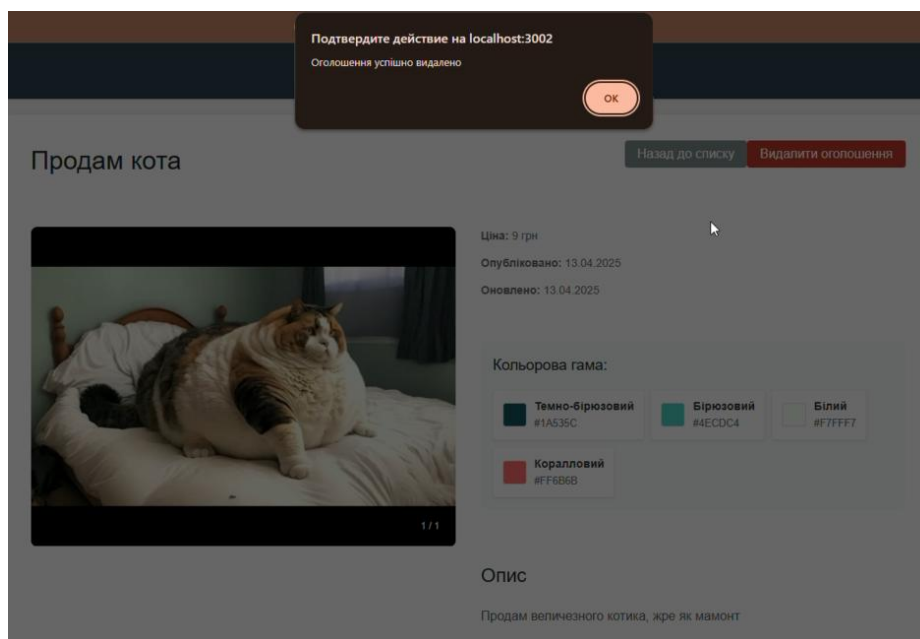


Рис. 3.10 – Видалення оголошення

Клієнти можуть подати запит на преміум-статус через сторінку "Преміум-статус" (рис. 3.11). Сторінка описує переваги (необмежена кількість оголошень, пріоритетне відображення, пріоритетна підтримка) і містить кнопку "Відправити запит". Запит створює чат із менеджером, де користувач може обговорити деталі. Менеджери обробляють запити через сторінку чатів.

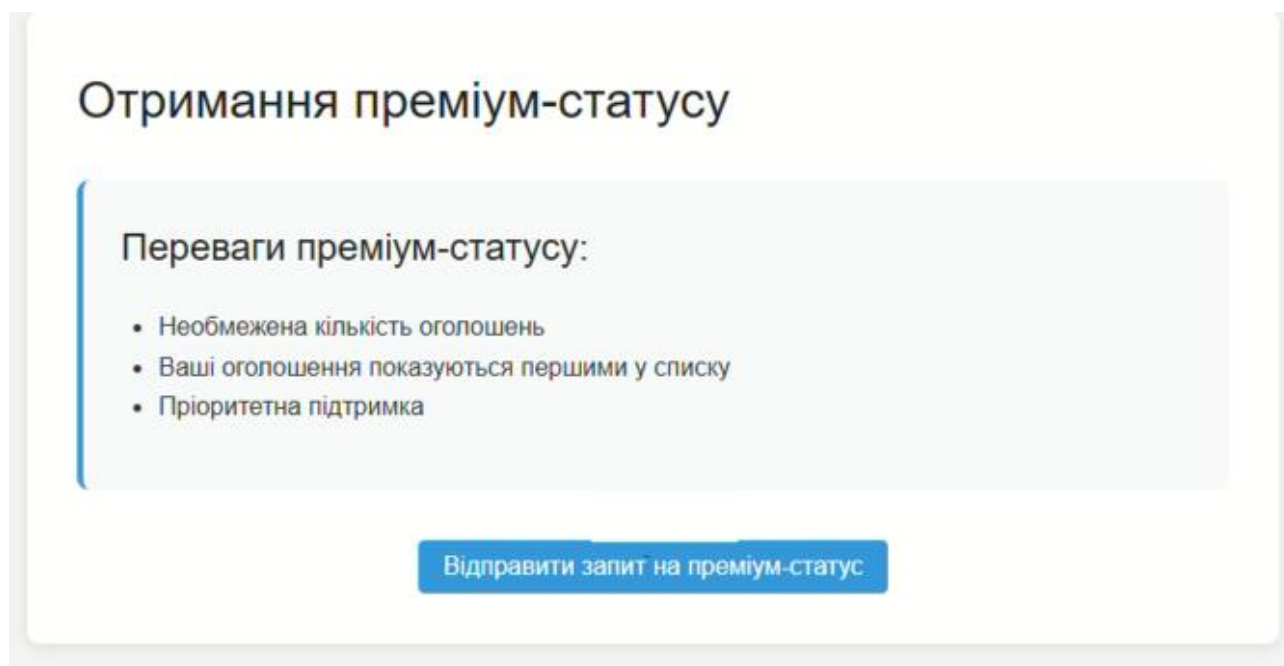


Рис. 3.11 – Сторінка запиту на преміум-статус

Сторінка "Чати" (рис. 3.12) відображає список активних чатів користувача (для клієнтів) або всіх чатів (для менеджерів). Кожен чат показує тип (запит на преміум або чат із менеджером), статус, дату створення та останнє повідомлення. Натиснувши на чат, користувач переходить на сторінку чату (рис. 3.13), де може надсилати повідомлення та переглядати історію листування. Менеджери можуть схвалювати або відхиляти запити на преміум-статус із цієї сторінки.

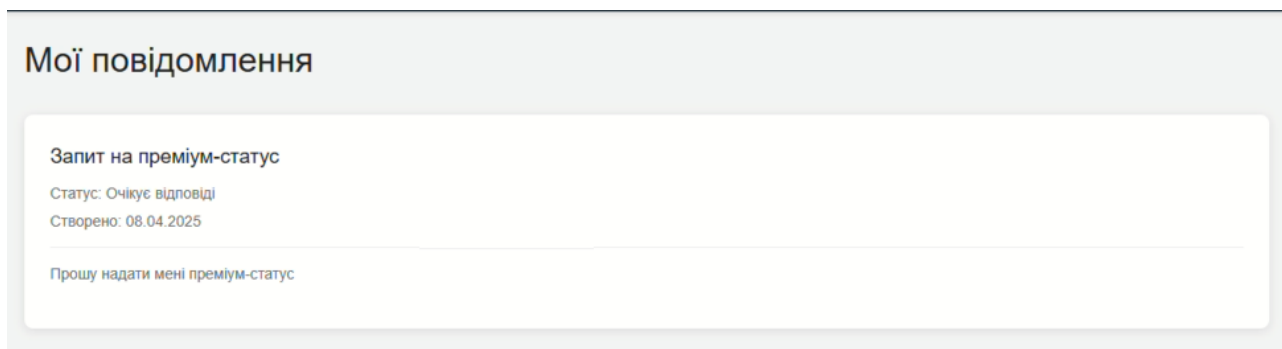


Рис. 3.12 – Сторінка чатів для користувача

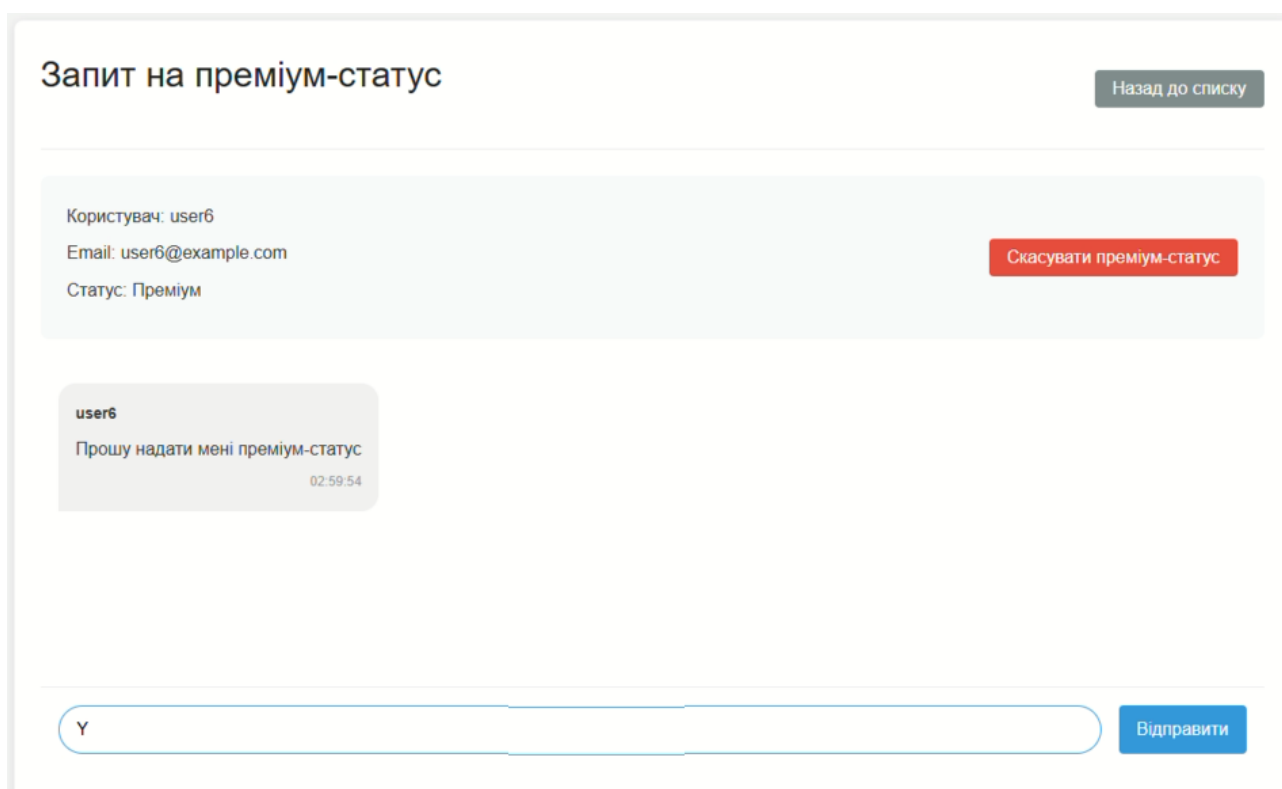


Рис. 3.13 – Сторінка чату для менеджерів

Менеджери мають доступ до сторінки "Управління користувачами" (рис. 3.14), де відображається таблиця з інформацією про всіх користувачів (ID, ім'я, email, роль, статус, дата реєстрації). Менеджери можуть видаляти користувачів (разом із їх оголошеннями) або змінювати преміум-статус через відповідні кнопки.

Управління користувачами						
ID	Ім'я користувача	Email	Роль	Статус	Дата реєстрації	Дії
1	manager1	manager1@example.com	Менеджер	Звичайний	02.04.2025	
2	manager2	manager2@example.com	Менеджер	Звичайний	06.04.2025	<button>Видалити</button>
3	user1	user1@example.com	Клієнт	Звичайний	17.03.2025	<button>Видалити</button> <button>Надати преміум</button>
4	user2	user2@example.com	Клієнт	Звичайний	01.04.2025	<button>Видалити</button> <button>Надати преміум</button>
5	user3	user3@example.com	Клієнт	Преміум	16.03.2025	<button>Видалити</button> <button>Скасувати преміум</button>
6	user4	user4@example.com	Клієнт	Звичайний	10.04.2025	<button>Видалити</button> <button>Надати преміум</button>
7	user5	user5@example.com	Клієнт	Звичайний	05.03.2025	<button>Видалити</button> <button>Надати преміум</button>
8	user6	user6@example.com	Клієнт	Звичайний	19.02.2025	<button>Видалити</button> <button>Надати преміум</button>
9	user7	user7@example.com	Клієнт	Звичайний	21.03.2025	<button>Видалити</button> <button>Надати преміум</button>
10	user8	user8@example.com	Клієнт	Звичайний	17.03.2025	<button>Видалити</button> <button>Надати преміум</button>
11	user9	user9@example.com	Клієнт	Преміум	10.03.2025	<button>Видалити</button> <button>Скасувати преміум</button>

Рис. 3.14 – Сторінка управління користувачами

Неавтентифіковані користувачі можуть натиснути "Вхід" або "Реєстрація" на навігаційній панелі. Сторінка входу (рис. 3.15) містить поля для імені користувача та пароля, а сторінка реєстрації (рис. 3.16) додатково включає email і вибір ролі (client або manager). Після успішної автентифікації користувач перенаправляється на головну сторінку.

Вхід в акаунт

Логін:

Пароль:

Увійти

[Ще не маєте акаунту? Зареєструватися](#)

Рис.3.15 – Сторінка входу

Реєстрація

Логін:

Email:

Пароль:

Підтвердження пароля:

Тип користувача:

Клієнт

Зареєструватися

[Вже маєте акаунт? Увійти](#)

Рис. 3.16 – Сторінка реєстрації

Користувач завершує сеанс, натиснувши кнопку "Вийти" на навігаційній панелі. Система видаляє токен автентифікації з локального сховища та повертає користувача на головну сторінку (рис. 3.17).

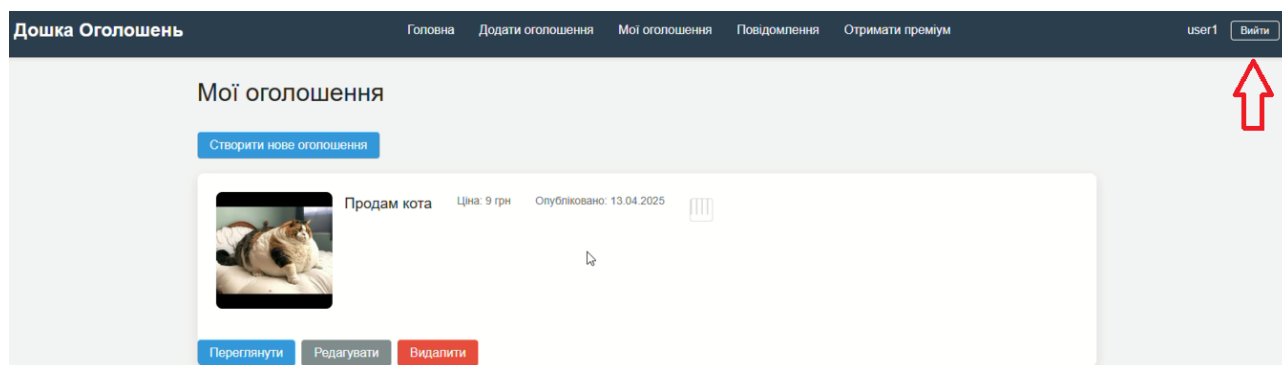


Рис. 3.17 – Вихід із системи

Усі дії супроводжуються сповіщеннями (рис. 3.18), які з'являються у вигляді повідомлень про успіх або помилку (наприклад, "Оголошення успішно

створено" або "Помилка: недопустима роль"). Форми мають валідацію на клієнтській стороні (обов'язкові поля, формат email, розмір файлів) та серверній стороні (перевірка унікальності імені користувача, розміру зображень).

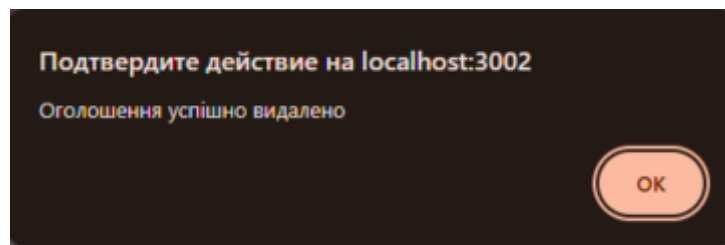


Рис. 3.18 – Сповіщення

Інтерфейс є адаптивним і коректно відображається на різних пристроях завдяки CSS-стилям у файлі styles.css. Навігація здійснюється через верхню панель, а основний контент розташований у центральній частині сторінки. Усі дії супроводжуються чіткими повідомленнями, що робить систему зручною для користувачів із різним рівнем досвіду.

3.3. Тестування вебдодатку

Тестування вебдодатку електронної дошки оголошень є критично важливим етапом її розробки та впровадження, що забезпечує надійність, безпеку та відповідність функціоналу заявленим вимогам. Реалізована система, представлена у коді проєкту (server.js, app.js, seed.js, index.html, styles.css), використовує технології React для клієнтської частини, Node.js з Express для серверної частини та SQLite як базу даних. Тестування охоплює як фронтенд, так і бекенд, а також їх взаємодію, включаючи перевірку API-ендпоінтів, компонентів інтерфейсу, безпеки та продуктивності. У цьому розділі розглядається процес тестування, типи тестів, інструменти, приклади сценаріїв та рекомендації для їх реалізації.

Модульне тестування зосереджене на перевірці окремих компонентів системи, таких як функції, методи та React-компоненти. На клієнтській частині необхідно протестувати компоненти, такі як AnnouncementCard,

EditAnnouncementPage, ChatPage, а також хуки, наприклад, useState та useEffect. На серверній частині тестуються API-ендпоінти, такі як /api/auth/register, /api/announcements, та логіка обробки запитів, наприклад, авторизація через JWT або хешування паролів за допомогою bcrypt.

Для тестування системи доцільно використовувати відповідні інструменти залежно від її частин. Для фронтенду рекомендується застосовувати Jest разом із React Testing Library для перевірки поведінки компонентів користувацького інтерфейсу. Це дозволяє переконатися що вміст компонентів правильно відображається а їхня інтеракція з користувачем працює належним чином. Наприклад тестується компонент AnnouncementCard перевіряючи чи правильно відображається заголовок ціна та перше фото оголошення а також чи викликається функція при натисканні кнопки "Переглянути". Для бекенду доцільно використовувати Jest у поєднанні з бібліотекою Supertest що дозволяє ефективно перевіряти відповіді API на HTTP-запити. Наприклад тестується ендпоінт авторизації /api/auth/login щоб переконатися що при введенні коректних даних система повертає статус 200 і дійсний JWT-токен а при неправильному паролі відповідає статусом 401. Рекомендується забезпечити принаймні 80 відсотків покриття коду модульними тестами що дозволить вчасно виявити помилки у критичних частинах додатку таких як створення оголошень авторизація користувачів та обробка зображень. Такий підхід гарантує стабільність системи під час її подальшого розвитку та масштабування.

Інтеграційне тестування перевіряє взаємодію між компонентами системи, зокрема між фронтеном і бекендом, а також між сервером і базою даних SQLite. Наприклад, необхідно переконатися, що створення оголошення через CreateAnnouncementPage коректно передає дані до сервера, записує їх у таблицю announcements і повертає відповідь із новим оголошенням.

Інструменти: Cypress або Playwright для енд-ту-енд тестування взаємодії клієнтської частини з сервером. Supertest для тестування API у зв'язці з SQLite.

Приклад: тестування сценарію створення оголошення:

— користувач авторизується через LoginPage;

- переходить на сторінку CreateAnnouncementPage;
- заповнює форму (заголовок, опис, ціна, категорія, контакти);
- завантажує фотографію через PhotoUploader;
- відправляє форму та перевіряє, що оголошення з'являється у списку на MyAnnouncementsPage.

Рекомендація: перевірити всі ключові сценарії користувацької взаємодії (реєстрація, створення/редагування оголошень, запит на преміум-статус, чати) та обробку помилок (наприклад, спроба завантаження файлу розміром більше 5 МБ).

Функціональне тестування оцінює відповідність системи специфікаціям, перевіряючи коректність виконання всіх заявлених функцій. Кожен модуль (авторизація, створення/редагування оголошень, чати, управління користувачами для менеджерів) тестується окремо.

Інструменти: ручне тестування за підготовленими сценаріями або автоматизоване тестування через Cypress чи Selenium.

Прикладом перевірки функціональності може бути тестування фільтрації оголошень на головній сторінці. Якщо ввести у поле пошуку слово "квартира", система має повернути оголошення, що належать до категорії "Нерухомість". Крім того, вибір сортування за ціною за зростанням відображає оголошення від найдешевшого до найдорожчого. Для менеджерів проводиться тестування управління користувачами на сторінці адміністрування, де перевіряють додавання або скасування преміум-статусу, а також видалення користувача з одночасною перевіркою каскадного видалення усіх пов'язаних з ним даних. Для якісного тестування рекомендується скласти чек-лист функціональних вимог і послідовно перевірити кожен пункт, приділяючи особливу увагу крайовим випадкам, наприклад, створенню оголошення без фотографій або спробі видалення чужого оголошення. Такий підхід допомагає виявити можливі помилки і забезпечити надійну роботу системи.

Система обробляє запити від багатьох користувачів, тому важливо оцінити її продуктивність під навантаженням. Оскільки SQLite є легкою базою даних,

вона може мати певні обмеження при великій кількості одночасних запитів, тому необхідно проводити ретельне тестування. Для цього можна використовувати інструменти, такі як JMeter або Artillery, які дозволяють симулювати одночасні запити до API. Наприклад, можна провести симуляцію 100 одночасних запитів до ендпоінта `/api/announcements`, щоб оцінити час відповіді і стабільність роботи сервера. Також важливо перевірити час завантаження головної сторінки при великій кількості оголошень, наприклад, коли в базі даних зберігається тисяча записів. Рекомендується забезпечити, щоб час відповіді не перевищував дві секунди при п'ятдесяти одночасних користувачах, а система могла витримувати до двохсот запитів за хвилину без збоїв. Окрім продуктивності, система працює з конфіденційними даними, такими як логіни, паролі та контактна інформація, тому потрібно перевірити її на наявність вразливостей, включаючи SQL-ін'єкції, XSS-атаки та несанкціонований доступ. Для цього використовують інструменти OWASP ZAP для автоматичного сканування і Burp Suite для ручного тестування. Наприклад, слід перевірити ендпоінт `/api/auth/login` на стійкість до SQL-ін'єкцій, вводячи у поле пароля вираз `' OR '1'='1`, а також протестувати поля форми створення оголошення на захист від XSS-атак, вводячи у поле заголовка скрипт `<script>alert('XSS')</script>`. Рекомендується переконатися, що паролі надійно хешуються за допомогою bcrypt, всі вхідні дані проходять валідацію на сервері, а токени JWT перевіряються перед виконанням запитів. Для захисту від XSS слід використовувати безпекові заголовки, наприклад Content-Security-Policy. Важливо також, щоб система коректно працювала в різних браузерях і на різних пристроях, враховуючи адаптивний дизайн, який реалізований за допомогою медіа-запитів у файлі styles.css для роздільних здатностей 576, 768 та 991 пікселів. Такий підхід забезпечує стабільність, безпеку і зручність користування додатком у різних умовах.

Інструменти: BrowserStack або ручне тестування на реальних пристроях.

Приклад: перевірка відображення HomePage та AnnouncementDetailsPage у браузерах Chrome, Firefox, Safari та на мобільних пристроях (iOS, Android) з різними розмірами екрану.

Рекомендація: забезпечити коректне відображення та функціональність на 90% цільових платформ, перевіряючи адаптивність форм, списків оголошень і каруселі фотографій.

Для тестування необхідно створити окреме середовище з копією бази даних, заповненою тестовими даними за допомогою файлу `seed.js`. Локальний сервер запускається на порту 3002, як зазначено у файлі `server.js`, щоб уникнути впливу на продуктивну систему. Тестові дані включають користувачів, серед яких є менеджери і клієнти, оголошення, категорії, фотографії та чати. Для кожного типу тестування розробляються відповідні тестові сценарії. Наприклад, у першому сценарії користувач проходить реєстрацію, авторизується, створює оголошення з фотографією, потім редагує його і зрештою видаляє. У другому сценарії менеджер переглядає список користувачів, надає преміум-статус одному з них і відповідає на запит у чаті. Третій сценарій передбачає перевірку поведінки системи при некоректних даних, наприклад, коли користувач намагається завантажити файл розміром більше п'яти мегабайт або вводить невалідну електронну пошту під час реєстрації. Виконання тестів розпочинається з модульних, які допомагають виявити базові помилки в окремих компонентах і функціях. Далі виконуються інтеграційні тести, що перевіряють взаємодію між фронтендом, бекендом і базою даних. Після цього проводяться функціональні тести, які підтверджують відповідність системи вимогам. Тести продуктивності і безпеки виконуються на завершальних етапах, коли основний функціонал вже стабілізовано. Всі виявлені дефекти фіксуються у системі відстеження помилок, наприклад у Jira, при цьому пріоритет виправлення залежить від критичності проблеми. Помилки, пов'язані з авторизацією, створенням оголошень або безпекою, мають вищий пріоритет у порівнянні з косметичними дефектами інтерфейсу, такими як зміщення елементів на малих екранах. Такий підхід дозволяє організувати ефективний процес тестування і своєчасно усувати найважливіші проблеми.

Рекомендації для підвищення ефективності

– Автоматизація передбачає використання інструментів, таких як Cypress або Playwright, для автоматизації повторюваних сценаріїв, зокрема авторизації, створення оголошень і відправки повідомлень у чатах. Тести інтегруються в CI/CD-пайплайн, наприклад, через GitHub Actions, для автоматичного запуску під час кожного оновлення коду.

– Моніторинг включає впровадження логування помилок на сервері за допомогою таких інструментів, як Winston, що дозволяє швидко виявляти проблеми у продакшені.

– Документація охоплює створення опису тестових сценаріїв і результатів, що спрощує процес подальшої підтримки системи.

Тестування електронної дошки оголошень має бути комплексним, охоплюючи модульні, інтеграційні, функціональні, продуктивності, безпеки та сумісності. Використання сучасних інструментів (Jest, Cypress, JMeter, OWASP ZAP) і ретельна підготовка тестових сценаріїв забезпечать високу якість продукту. Автоматизація ключових тестів і інтеграція з CI/CD дозволять підтримувати стабільність системи при оновленнях. Після завершення тестування система буде готова до впровадження, а виявлені недоліки можна усунути, підвищивши її надійність і зручність для користувачів.

ВИСНОВКИ ДО РОЗДІЛУ 3

Розділ, присвячений інтерфейсу та порядку роботи з веб-додатком "Електронна дошка оголошень", детально описує процес встановлення, налаштування, використання системи та її тестування для забезпечення якості.

Встановлення системи є простим завдяки чіткій клієнт-серверній архітектурі, де серверна частина на Node.js із SQLite та клієнтська на React ефективно взаємодіють, потребуючи мінімальних ресурсів і сучасного браузера.

Інтерфейс додатка вирізняється інтуїтивністю: навігаційна панель забезпечує швидкий доступ до ключових функцій (оголошення, чати, преміум-статус, управління користувачами), футер містить додаткову інформацію, а сторінки (головна, деталі оголошення, чати) логічно структуровані для зручності

клієнтів і менеджерів. Адаптивний дизайн і сповіщення підвищують комфорт користувача. Тестування охоплює модульні, інтеграційні, функціональні, продуктивності, безпеки та сумісності, використовуючи інструменти Jest, Cypress, JMeter і OWASP ZAP. Автоматизація тестів і CI/CD-інтеграція забезпечують стабільність при оновленнях. Загалом, додаток є зручним і надійним рішенням для розміщення оголошень, поєднуючи простоту розгортання, продуманий інтерфейс і якість, підтверджену тестуванням, що робить його готовим до впровадження з потенціалом для вдосконалення.

ВИСНОВКИ

Розробка веб-додатку «Електронна дошка оголошень», представлена в даній роботі, є важливим кроком у напрямі вдосконалення обміну інформацією в цифровому середовищі. Зростання популярності онлайн-платформ, потреба в швидкому доступі до актуальних пропозицій та необхідність зручної взаємодії між користувачами підкреслюють актуальність створення такого програмного забезпечення. Система спрямована на забезпечення зручності для клієнтів, які розміщують і переглядають оголошення, та ефективного адміністрування для менеджерів, сприяючи розвитку місцевих спільнот і бізнесу.

Метою роботи було створення функціональної платформи для розміщення, пошуку та управління оголошеннями, яка відповідає потребам користувачів і адміністраторів. У процесі реалізації використано сучасні технології: React.js для інтерактивного інтерфейсу, Node.js з Express.js для серверної логіки та SQLite як легку базу даних. Застосування об'єктно-орієнтованого аналізу, зокрема побудова ER-діаграм і діаграм компонентів, забезпечило чітку структуру системи. Асинхронне програмування, JWT-авторизація, bcrypt-хешування та методи тестування підвищили надійність і безпеку продукту.

Розроблена система охоплює ключові показники предметної області, включаючи створення оголошень із мультимедійним контентом, фільтрацію, чати, авторизацію та менеджерські функції. Адаптивний дизайн і локальне зберігання даних забезпечують зручність на різних пристроях, а модульна архітектура сприяє масштабованості. Проведений аналіз вимог підтвердив відповідність системи функціональним і нефункціональним характеристикам, хоча для підвищення безпеки потрібне вдосконалення, наприклад, захист від XSS-атак.

Апробація системи на засіданнях кафедри в 2025 році підтвердила її практичну цінність, отримавши позитивні відгуки. Порівняння з аналогами (OLX, Prom.ua, eBay Classifieds, Gumtree) виявило конкурентні переваги:

локалізацію, чати та менеджерські функції, хоча система поступається за масштабом.

Таким чином, створений веб-додаток є комплексним рішенням, яке оптимізує обмін інформацією, відповідаючи сучасним вимогам ринку. Гнучка архітектура відкриває можливості для вдосконалення, зокрема через інтеграцію push-повідомлень чи розширення фільтрів. Робота демонструє успішне застосування теоретичних знань і вносить внесок у розвиток цифрових платформ в Україні, сприяючи ефективній взаємодії користувачів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. РОЗРОБКА ДОЩОК ОГОЛОШЕНЬ [Електронний ресурс] / РОЗРОБКА ДОЩОК ОГОЛОШЕНЬ – Режим доступу до ресурсу: <https://webcase.com.ua/uk/sozдание-doski-obyavlenij/>
2. User interface це не просто прикраса продукту [Електронний ресурс] / User interface це не просто прикраса продукту – Режим доступу до ресурсу: <https://foxminded.ua/user-interface-tse/>
3. OLX [Електронний ресурс] / OLX – Режим доступу до ресурсу: <https://www.olx.ua/uk/>
4. Prom.ua [Електронний ресурс] / Prom.ua – Режим доступу до ресурсу: <https://prom.ua/ua/>
5. eBay [Електронний ресурс] / eBay – Режим доступу до ресурсу: <https://www.ebay.com/>
6. Gumtree [Електронний ресурс] / Gumtree – Режим доступу до ресурсу: <https://www.gumtree.com/>
7. Основні типи архітектури програмного забезпечення [Електронний ресурс] / Основні типи архітектури програмного забезпечення – Режим доступу до ресурсу: <https://www.artofba.com/uk/post/main-types-of-software-architecture>
8. Розробка серверно-клієнтського веб-додатку [Електронний ресурс] / Розробка серверно-клієнтського веб-додатку – Режим доступу до ресурсу: <https://promoter.net.ua/articles/rozrobka-serverno-klijentskogo-veb-dodatku.html>
9. Клієнт-серверна архітектура [Електронний ресурс] / Клієнт-серверна архітектура – Режим доступу до ресурсу: <https://training.qatestlab.com/blog/technical-articles/client-server-architecture/>
10. Що таке веб-сервер [Електронний ресурс] / Що таке веб-сервер – Режим доступу до ресурсу: <https://freehost.com.ua/ukr/faq/wiki/chto-takoe-veb-server/>
11. Що таке Node.js? Основи серверної розробки на JavaScript [Електронний ресурс] / Що таке Node.js? – Режим доступу до ресурсу:

<https://devzone.org.ua/post/shcho-take-nodejs-osnovy-servernoyi-rozrobky-na-javascript>

12. Створюємо наш перший API за допомогою Node.js та Express: Створюємо сервер [Електронний ресурс] / Створюємо наш перший API за допомогою Node.js та Express: Створюємо сервер – Режим доступу до ресурсу: <https://code.tutsplus.com/uk/code-your-first-api-with-nodejs-and-express-set-up-the-server--cms-31698t>
13. Що таке SQLite, для чого і як його використовувати [Електронний ресурс] / Що таке SQLite, для чого і як його використовувати – Режим доступу до ресурсу: <https://foxminded.ua/prohramuvannia-baz-danykh-na-sqlite/>
14. React [Електронний ресурс] / React – Режим доступу до ресурсу: <https://uk.legacy.reactjs.org/>
15. Multer [Електронний ресурс] / Multer – Режим доступу до ресурсу: <https://www.npmjs.com/package/multer>
16. JSON Web Token (JWT) [Електронний ресурс] / JSON Web Token (JWT) – Режим доступу до ресурсу: <https://www.vpnunlimited.com/ua/help/cybersecurity/json-web-token>
17. node.bcrypt.js [Електронний ресурс] / node.bcrypt.js – Режим доступу до ресурсу: <https://www.npmjs.com/package/bcrypt>
18. REST API: що це таке і як працює [Електронний ресурс] / REST API: що це таке і як працює – Режим доступу до ресурсу: <https://www.ibm.com/uk/cloud/learn/rest-apis>
19. Основи тестування програмного забезпечення [Електронний ресурс] / Основи тестування програмного забезпечення – Режим доступу до ресурсу: <https://testingdocs.com/testing-basics/>
20. JMeter: керівництво для початківців [Електронний ресурс] / JMeter: керівництво для початківців – Режим доступу до ресурсу: <https://jmeter.apache.org/usermanual/get-started.html>
21. Artillery: інструмент для тестування навантаження [Електронний ресурс] / Artillery – Режим доступу до ресурсу: <https://artillery.io/docs/>

- 22.OWASP ZAP: автоматичний сканер вразливостей [Електронний ресурс] / OWASP ZAP – Режим доступу до ресурсу: <https://www.zaproxy.org/>
- 23.Burp Suite: інструмент для тестування безпеки [Електронний ресурс] / Burp Suite – Режим доступу до ресурсу: <https://portswigger.net/burp>
- 24.Хешування паролів за допомогою bcrypt [Електронний ресурс] / Хешування паролів за допомогою bcrypt – Режим доступу до ресурсу: <https://www.npmjs.com/package/bcrypt>
- 25.JWT авторизація у веб-додатках [Електронний ресурс] / JWT авторизація у веб-додатках – Режим доступу до ресурсу: <https://jwt.io/introduction/>
- 26.Захист від XSS-атак у веб-додатках [Електронний ресурс] / Захист від XSS-атак у веб-додатках – Режим доступу до ресурсу: <https://owasp.org/www-community/attacks/xss/>
- 27.Content Security Policy (CSP) для підвищення безпеки [Електронний ресурс] / Content Security Policy (CSP) – Режим доступу до ресурсу: <https://developer.mozilla.org/uk/docs/Web/HTTP/CSP>
- 28.Адаптивний дизайн веб-сайтів [Електронний ресурс] / Адаптивний дизайн веб-сайтів – Режим доступу до ресурсу: <https://web.dev/responsive-web-design-basics/>
- 29.Медіа-запити CSS: як працюють [Електронний ресурс] / Медіа-запити CSS – Режим доступу до ресурсу: https://developer.mozilla.org/uk/docs/Web/CSS/Media_Queries/Using_media_queries
- 30.Валідація форм у React [Електронний ресурс] / Валідація форм у React – Режим доступу до ресурсу: <https://reactjs.org/docs/forms.html>

ДОДАТКИ

					ІІЗ.КР.Б – 121 – 25 – ІІЗ	Арк .
						62

```

server.js
const express = require('express');
const bodyParser = require('body-parser');
const cors = require('cors');
const jwt = require('jsonwebtoken');
const bcrypt = require('bcrypt');
const path = require('path');
const multer = require('multer');
const fs = require('fs');
const sqlite3 = require('sqlite3').verbose();
const { open } = require('sqlite');
const app = express();
const port = 3002;

// Секретний ключ для JWT
const JWT_SECRET = 'your_jwt_secret_key';

// Налаштування multer для завантаження зображень
const storage = multer.diskStorage({
  destination: function (req, file, cb) {
    const uploadDir = path.join(__dirname, 'public', 'uploads');
    // Перевіряємо і створюємо директорію, якщо вона не існує
    if (!fs.existsSync(uploadDir)) {
      fs.mkdirSync(uploadDir, { recursive: true });
    }
    cb(null, uploadDir);
  },
  filename: function (req, file, cb) {
    // Створюємо унікальну назву файлу
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    const ext = path.extname(file.originalname);
    cb(null, 'image-' + uniqueSuffix + ext);
  }
});

// Фільтр файлів - приймаємо тільки зображення
const fileFilter = (req, file, cb) => {
  if (file.mimetype.startsWith('image/')) {
    cb(null, true);
  } else {
    cb(new Error('Дозволені тільки файли зображень!'), false);
  }
};

// Налаштування multer
const upload = multer({
  storage: storage,
  fileFilter: fileFilter,
  limits: {
    fileSize: 5 * 1024 * 1024 // Максимальний розмір файлу 5MB
  }
});

// Налаштування middleware
app.use(cors());
app.use(bodyParser.json());

// Обслуговування статичних файлів з папки public
app.use(express.static(path.join(__dirname, 'public')));

// Ініціалізація бази даних
let db;
async function initializeDatabase() {
  db = await open({
    filename: 'database.sqlite',
    driver: sqlite3.Database
  });
  // Створення таблиць, якщо вони ще не існують
  await db.exec(`

```

```

CREATE TABLE IF NOT EXISTS users (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  username TEXT UNIQUE NOT NULL,
  password TEXT NOT NULL,
  email TEXT UNIQUE NOT NULL,
  role TEXT NOT NULL,
  isPremium INTEGER NOT NULL DEFAULT 0,
  createdAt TEXT NOT NULL
);
CREATE TABLE IF NOT EXISTS categories (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  name TEXT NOT NULL
);
CREATE TABLE IF NOT EXISTS announcements (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  title TEXT NOT NULL,
  description TEXT NOT NULL,
  categoryId INTEGER NOT NULL,
  price REAL NOT NULL,
  contactPhone TEXT,
  contactEmail TEXT,
  userId INTEGER NOT NULL,
  createdAt TEXT NOT NULL,
  updatedAt TEXT NOT NULL,
  colorPalette TEXT,
  FOREIGN KEY (userId) REFERENCES users(id),
  FOREIGN KEY (categoryId) REFERENCES categories(id)
);
CREATE TABLE IF NOT EXISTS photos (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  announcementId INTEGER NOT NULL,
  url TEXT NOT NULL,
  filename TEXT NOT NULL,
  createdAt TEXT NOT NULL,
  FOREIGN KEY (announcementId) REFERENCES announcements(id) ON DELETE
CASCADE
);
CREATE TABLE IF NOT EXISTS chats (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  userId INTEGER NOT NULL,
  type TEXT NOT NULL,
  status TEXT NOT NULL,
  createdAt TEXT NOT NULL,
  FOREIGN KEY (userId) REFERENCES users(id) ON DELETE CASCADE
);
CREATE TABLE IF NOT EXISTS messages (
  id INTEGER PRIMARY KEY AUTOINCREMENT,
  chatId INTEGER NOT NULL,
  senderId INTEGER NOT NULL,
  text TEXT NOT NULL,
  createdAt TEXT NOT NULL,
  FOREIGN KEY (chatId) REFERENCES chats(id) ON DELETE CASCADE,
  FOREIGN KEY (senderId) REFERENCES users(id)
);
`);
// Додаємо базові категорії, якщо таблиця порожня
const categoriesCount = await db.get('SELECT COUNT(*) as count FROM
categories');
if (categoriesCount.count === 0) {
const defaultCategories = [
  { name: 'Нерухомість' },
  { name: 'Транспорт' },
  { name: 'Робота' },
  { name: 'Електроніка' },

```

```

    { name: 'Послуги' }
  ];
  for (const category of defaultCategories) {
    await db.run('INSERT INTO categories (name) VALUES (?)', category.name);
  }
  console.log('База даних ініціалізована');
}

// Middleware для перевірки авторизації користувача
const authMiddleware = async (req, res, next) => {
  const token = req.headers.authorization?.split(' ')[1];
  if (!token) {
    return res.status(401).json({ message: 'Необхідна авторизація' });
  }
  try {
    // Перевіряємо та декодуємо токен
    const decoded = jwt.verify(token, JWT_SECRET);
    // Отримуємо актуальні дані користувача з бази
    const user = await db.get('SELECT * FROM users WHERE id = ?',
    decoded.id);
    if (!user) {
      return res.status(401).json({ message: 'Користувача не знайдено' });
    }
    // Зберігаємо інформацію про користувача
    req.user = {
      id: user.id,
      username: user.username,
      role: user.role,
      isPremium: user.isPremium === 1
    };
    next();
  } catch (error) {
    return res.status(401).json({ message: 'Невірний токен' });
  }
};

// Middleware для перевірки ролі користувача
const roleMiddleware = (roles) => {
  return (req, res, next) => {
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ message: 'Доступ заборонено' });
    }
    next();
  };
};

// API МАРШРУТИ
// Реєстрація нового користувача
app.post('/api/auth/register', async (req, res) => {
  try {
    const { username, password, email, role } = req.body;
    // Перевірка наявності користувача з таким логіном або email
    const existingUser = await db.get('SELECT * FROM users WHERE username =
? OR email = ?', [username, email]);
    if (existingUser) {
      return res.status(400).json({ message: 'Користувач з таким логіном або
електронною поштою вже існує' });
    }
    // Перевірка допустимої ролі
    if (role !== 'client' && role !== 'manager') {
      return res.status(400).json({ message: 'Недопустима роль' });
    }
    // Хешування пароля для безпечного зберігання
    const hashedPassword = await bcrypt.hash(password, 10);
    const createdAt = new Date().toISOString();
    // Створення нового користувача

```

```

    const result = await db.run(
      'INSERT INTO users (username, password, email, role, isPremium,
createdAt) VALUES (?, ?, ?, ?, ?, ?)',
      [username, hashedPassword, email, role, 0, createdAt]
    );
    const newUserId = result.lastID;
    // Генерація JWT токена для авторизації
    const token = jwt.sign(
      { id: newUserId, username, role, isPremium: false },
      JWT_SECRET,
      { expiresIn: '24h' }
    );
    // Відправлення відповіді
    res.status(201).json({
      message: 'Користувач успішно зареєстрований',
      token,
      user: {
        id: newUserId,
        username,
        email,
        role,
        isPremium: false
      }
    });
  } catch (error) {
    console.error('Помилка реєстрації:', error);
    res.status(500).json({ message: 'Помилка сервера при реєстрації' });
  }
});

// Авторизація користувача
app.post('/api/auth/login', async (req, res) => {
  try {
    const { username, password } = req.body;
    // Пошук користувача за логіном
    const user = await db.get('SELECT * FROM users WHERE username = ?',
username);
    if (!user) {
      return res.status(401).json({ message: 'Невірний логін або пароль' });
    }
    // Перевірка пароля
    const isPasswordValid = await bcrypt.compare(password, user.password);
    if (!isPasswordValid) {
      return res.status(401).json({ message: 'Невірний логін або пароль' });
    }
    // Генерація JWT токена
    const token = jwt.sign(
      { id: user.id, username: user.username, role: user.role, isPremium:
user.isPremium === 1 },
      JWT_SECRET,
      { expiresIn: '24h' }
    );
    // Відправлення відповіді з токеном та інформацією про користувача
    res.json({
      message: 'Авторизація успішна',
      token,
      user: {
        id: user.id,
        username: user.username,
        email: user.email,
        role: user.role,
        isPremium: user.isPremium === 1
      }
    });
  } catch (error) {

```

```

        console.error('Помилка авторизації:', error);
        res.status(500).json({ message: 'Помилка сервера при авторизації' });
    }
});

// Отримання всіх категорій
app.get('/api/categories', async (req, res) => {
    try {
        const categories = await db.all('SELECT * FROM categories');
        res.json(categories);
    } catch (error) {
        console.error('Помилка отримання категорій:', error);
        res.status(500).json({ message: 'Помилка сервера при отриманні
категорій' });
    }
});

// Отримання оголошень з можливістю сортування, пошуку та фільтрації
app.get('/api/announcements', async (req, res) => {
    try {
        let query = `
        SELECT a.*
        FROM announcements a
        LEFT JOIN users u ON a.userId = u.id
        WHERE 1=1
        `;
        const params = [];
        // Фільтрація за користувачем
        if (req.query.userId) {
            query += ' AND a.userId = ?';
            params.push(parseInt(req.query.userId));
        }
        // Фільтрація за категорією
        if (req.query.category) {
            query += ' AND a.categoryId = ?';
            params.push(parseInt(req.query.category));
        }
        // Пошук за текстом (заголовки та опис)
        if (req.query.search) {
            query += ' AND (a.title LIKE ? OR a.description LIKE ?)';
            const searchTerm = `%${req.query.search}%`;
            params.push(searchTerm, searchTerm);
        }
        // Фільтрація за ціною (мінімальна ціна)
        if (req.query.minPrice) {
            query += ' AND a.price >= ?';
            params.push(parseFloat(req.query.minPrice));
        }
        // Фільтрація за ціною (максимальна ціна)
        if (req.query.maxPrice) {
            query += ' AND a.price <= ?';
            params.push(parseFloat(req.query.maxPrice));
        }
        // Сортування результатів
        if (req.query.sort) {
            switch (req.query.sort) {
                case 'priceAsc': // За зростанням ціни
                    query += ' ORDER BY a.price ASC';
                    break;
                case 'priceDesc': // За спаданням ціни
                    query += ' ORDER BY a.price DESC';
                    break;
                case 'dateAsc': // За зростанням дати (спочатку старіші)
                    query += ' ORDER BY a.createdAt ASC';
                    break;
                case 'dateDesc': // За спаданням дати (спочатку новіші)

```

```

    query += ' ORDER BY a.createdAt DESC';
    break;
  default:
    break;
  }
  } else {
    // За замовчуванням спочатку оголошення від преміум-користувачів, потім
    за датою
    query += ' ORDER BY u.isPremium DESC, a.createdAt DESC';
  }
  // Отримуємо всі оголошення, що відповідають фільтрам
  const filteredAnnouncements = await db.all(query, params);
  // Пагінація результатів
  const page = parseInt(req.query.page) || 1;
  const limit = parseInt(req.query.limit) || 10;
  const startIndex = (page - 1) * limit;
  const endIndex = page * limit;
  const paginatedAnnouncements = filteredAnnouncements.slice(startIndex,
endIndex);
  // Перевірка авторизації для приховування контактних даних
  const token = req.headers.authorization?.split(' ')[1];
  let isAuthenticated = false;
  if (token) {
    try {
      jwt.verify(token, JWT_SECRET); // Перевіряємо валідність токена
      isAuthenticated = true;
    } catch (error) {
      isAuthenticated = false;
    }
  }
  // Отримуємо фотографії для всіх оголошень на поточній сторінці
  const announcementIds = paginatedAnnouncements.map(a => a.id);
  let allPhotos = [];
  if (announcementIds.length > 0) {
    const placeholders = announcementIds.map(() => '?').join(',');
    allPhotos = await db.all(
      `SELECT * FROM photos WHERE announcementId IN (${placeholders})`,
      announcementIds
    );
  }
  // Формуємо результат з фотографіями та обмеженням даних для гостей
  const result = paginatedAnnouncements.map(announcement => {
    // Парсимо JSON поле colorPalette
    let colorPalette = [];
    try {
      if (announcement.colorPalette) {
        colorPalette = JSON.parse(announcement.colorPalette);
      }
    } catch (error) {
      console.error('Помилка парсингу colorPalette:', error);
    }
    const announcementData = {
      ...announcement,
      colorPalette,
      photos: allPhotos
        .filter(photo => photo.announcementId === announcement.id)
        .map(photo => ({
          id: photo.id,
          url: photo.url
        })))
    };
    if (!isAuthenticated) {
      delete announcementData.contactPhone;
      delete announcementData.contactEmail;
    }
  });

```

```

    }
    return announcementData;
  });
  // Відправлення результатів з метаданими про пагінацію
  res.json({
    total: filteredAnnouncements.length,
    page,
    totalPages: Math.ceil(filteredAnnouncements.length / limit),
    announcements: result
  });
} catch (error) {
  console.error('Помилка отримання оголошень:', error);
  res.status(500).json({ message: 'Помилка сервера при отриманні оголошень' });
}
});

// Отримання одного оголошення за ID
app.get('/api/announcements/:id', async (req, res) => {
  try {
    const id = parseInt(req.params.id);
    // Отримуємо дані оголошення
    const announcement = await db.get('SELECT * FROM announcements WHERE id = ?', id);
    if (!announcement) {
      return res.status(404).json({ message: 'Оголошення не знайдено' });
    }
    // Парсимо JSON поле colorPalette
    let colorPalette = [];
    try {
      if (announcement.colorPalette) {
        colorPalette = JSON.parse(announcement.colorPalette);
      }
    } catch (error) {
      console.error('Помилка парсингу colorPalette:', error);
    }
    // Отримуємо фотографії для оголошення
    const photos = await db.all('SELECT id, url FROM photos WHERE announcementId = ?', id);
    // Перевірка авторизації для приховування контактних даних
    const token = req.headers.authorization?.split(' ')[1];
    let isAuthenticated = false;
    if (token) {
      try {
        jwt.verify(token, JWT_SECRET); // Перевіряємо валідність токена
        isAuthenticated = true;
      } catch (error) {
        isAuthenticated = false;
      }
    }
    const result = {
      ...announcement,
      colorPalette,
      photos
    };
    if (!isAuthenticated) {
      delete result.contactPhone;
      delete result.contactEmail;
    }
    res.json(result);
  } catch (error) {
    console.error('Помилка отримання оголошення:', error);
    res.status(500).json({ message: 'Помилка сервера при отриманні оголошення' });
  }
}

```

```

    });
    // Створення нового оголошення (тільки для клієнтів)
    app.post('/api/announcements', authMiddleware,
    roleMiddleware(['client']), async (req, res) => {
        try {
            const { title, description, categoryId, price, contactPhone,
            contactEmail, colorPalette } = req.body;
            // Перевірка кількості оголошень для звичайного користувача
            if (!req.user.isPremium) {
                const userAnnouncementsCount = await db.get(
                'SELECT COUNT(*) as count FROM announcements WHERE userId = ?',
                req.user.id
                );
                if (userAnnouncementsCount.count >= 5) {
                    return res.status(403).json({
                    message: 'Ви досягли ліміту оголошень. Для розміщення більшої кількості
оголошень необхідно отримати преміум-статус'
                    });
                }
            }
            const now = new Date().toISOString();
            // Підготовка кольорової гами для зберігання
            const colorPaletteJson = colorPalette ? JSON.stringify(colorPalette) :
null;

            // Створення нового оголошення
            const result = await db.run(
            `INSERT INTO announcements (title, description, categoryId, price,
            contactPhone, contactEmail, userId, createdAt, updatedAt, colorPalette)
            VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)` ,
            [title, description, parseInt(categoryId), parseFloat(price),
            contactPhone, contactEmail, req.user.id, now, now, colorPaletteJson]
            );
            const newAnnouncementId = result.lastID;
            // Отримуємо створене оголошення
            const newAnnouncement = await db.get('SELECT * FROM announcements WHERE
id = ?', newAnnouncementId);
            // Парсимо JSON поле colorPalette для відповіді
            let parsedColorPalette = [];
            try {
                if (newAnnouncement.colorPalette) {
                    parsedColorPalette = JSON.parse(newAnnouncement.colorPalette);
                }
            } catch (error) {
                console.error('Помилка парсингу colorPalette:', error);
            }
            // Відправлення відповіді
            res.status(201).json({
            ...newAnnouncement,
            colorPalette: parsedColorPalette,
            photos: []
            });
        } catch (error) {
            console.error('Помилка створення оголошення:', error);
            res.status(500).json({ message: 'Помилка сервера при створенні
оголошення' });
        }
    });

    // Завантаження фото для оголошення
    app.post('/api/announcements/:id/photos', authMiddleware,
    upload.array('photos', 8), async (req, res) => {
        try {
            const id = parseInt(req.params.id);
            // Перевіряємо існування оголошення
            const announcement = await db.get('SELECT * FROM announcements WHERE id

```

```

= '?', id);
    if (!announcement) {
        return res.status(404).json({ message: 'Оголошення не знайдено' });
    }
    // Перевірка власника оголошення
    if (announcement.userId !== req.user.id && req.user.role !== 'manager')
{
    return res.status(403).json({ message: 'Ви не можете додавати фото до
чужого оголошення' });
}
    // Перевірка кількості фото
    const existingPhotosCount = await db.get(
        'SELECT COUNT(*) as count FROM photos WHERE announcementId = ?',
        id
    );
    const newPhotosCount = req.files.length;
    if (existingPhotosCount.count + newPhotosCount > 8) {
        return res.status(400).json({ message: 'Максимальна кількість фото для
оголошення - 8' });
    }
    // Створення записів для нових фото
    const uploadedPhotos = [];
    const now = new Date().toISOString();
    for (const file of req.files) {
        const url = `/uploads/${file.filename}`;
        const result = await db.run(
            'INSERT INTO photos (announcementId, url, filename, createdAt) VALUES
            (?, ?, ?, ?)',
            [id, url, file.filename, now]
        );
        uploadedPhotos.push({
            id: result.lastID,
            url
        });
    }
    // Оновлення часу редагування оголошення
    await db.run(
        'UPDATE announcements SET updatedAt = ? WHERE id = ?',
        [now, id]
    );
    res.status(201).json({
        message: 'Фото успішно завантажені',
        photos: uploadedPhotos
    });
} catch (error) {
    console.error('Помилка завантаження фото:', error);
    res.status(500).json({ message: 'Помилка сервера при завантаженні фото'
});
}
});
// Видалення фото з оголошення
app.delete('/api/photos/:id', authMiddleware, async (req, res) => {
    try {
        const photoId = parseInt(req.params.id);
        // Отримуємо інформацію про фото
        const photo = await db.get('SELECT * FROM photos WHERE id = ?',
photoId);
        if (!photo) {
            return res.status(404).json({ message: 'Фото не знайдено' });
        }
        // Отримуємо інформацію про оголошення
        const announcement = await db.get('SELECT * FROM announcements WHERE id
= ?', photo.announcementId);
        if (!announcement) {

```

```

    return res.status(404).json({ message: 'Оголошення не знайдено' });
  }
  // Перевірка прав доступу (власник оголошення або менеджер)
  if (req.user.role !== 'manager' && announcement.userId !== req.user.id)
  {
    return res.status(403).json({ message: 'Ви не можете видалити чуже фото'
});
  }
  // Видалення файлу з сервера
  const filePath = path.join(__dirname, 'public', photo.url);
  try {
    if (fs.existsSync(filePath)) {
      fs.unlinkSync(filePath);
    }
  } catch (err) {
    console.error('Помилка видалення файлу:', err);
  }
  // Видалення фото з бази даних
  await db.run('DELETE FROM photos WHERE id = ?', photoId);
  // Оновлення часу редагування оголошення
  await db.run(
    'UPDATE announcements SET updatedAt = ? WHERE id = ?',
    [new Date().toISOString(), announcement.id]
  );
  res.json({ message: 'Фото успішно видалено' });
} catch (error) {
  console.error('Помилка видалення фото:', error);
  res.status(500).json({ message: 'Помилка сервера при видаленні фото' });
}
});

// Оновлення існуючого оголошення (тільки для власника)
app.put('/api/announcements/:id', authMiddleware,
roleMiddleware(['client']), async (req, res) => {
  try {
    const id = parseInt(req.params.id);
    // Перевіряємо існування оголошення
    const announcement = await db.get('SELECT * FROM announcements WHERE id
= ?', id);
    if (!announcement) {
      return res.status(404).json({ message: 'Оголошення не знайдено' });
    }
    // Перевірка власника оголошення
    if (announcement.userId !== req.user.id) {
      return res.status(403).json({ message: 'Ви не можете редагувати чуже
оголошення' });
    }
    const { title, description, categoryId, price, contactPhone,
contactEmail, colorPalette } = req.body;
    // Підготовка кольорової гами для зберігання
    const colorPaletteJson = colorPalette ? JSON.stringify(colorPalette) :
null;

    // Оновлення даних оголошення
    await db.run(
      `UPDATE announcements SET
title = COALESCE(?, title),
description = COALESCE(?, description),
categoryId = COALESCE(?, categoryId),
price = COALESCE(?, price),
contactPhone = COALESCE(?, contactPhone),
contactEmail = COALESCE(?, contactEmail),
colorPalette = ?,
updatedAt = ?
WHERE id = ?`,
      [

```

```

    title,
    description,
    categoryId ? parseInt(categoryId) : null,
    price ? parseFloat(price) : null,
    contactPhone,
    contactEmail,
    colorPaletteJson,
    new Date().toISOString(),
    id
  ]
);
// Отримуємо оновлене оголошення
const updatedAnnouncement = await db.get('SELECT * FROM announcements
WHERE id = ?', id);
// Парсимо JSON поле colorPalette для відповіді
let parsedColorPalette = [];
try {
  if (updatedAnnouncement.colorPalette) {
    parsedColorPalette = JSON.parse(updatedAnnouncement.colorPalette);
  }
} catch (error) {
  console.error('Помилка парсингу colorPalette:', error);
}
// Отримуємо фотографії для оголошення
const photos = await db.all('SELECT id, url FROM photos WHERE
announcementId = ?', id);
// Відправлення оновленого оголошення
res.json({
  ...updatedAnnouncement,
  colorPalette: parsedColorPalette,
  photos
});
} catch (error) {
  console.error('Помилка оновлення оголошення:', error);
  res.status(500).json({ message: 'Помилка сервера при оновленні
оголошення' });
}
});
// Видалення оголошення (для власника або менеджера)
app.delete('/api/announcements/:id', authMiddleware, async (req, res) =>
{
  try {
    const id = parseInt(req.params.id);
    // Перевіряємо існування оголошення
    const announcement = await db.get('SELECT * FROM announcements WHERE id
= ?', id);
    if (!announcement) {
      return res.status(404).json({ message: 'Оголошення не знайдено' });
    }
    // Перевірка прав доступу (власник або менеджер)
    if (req.user.role !== 'manager' && announcement.userId !== req.user.id)
    {
      return res.status(403).json({ message: 'Ви не можете видалити чуже
оголошення' });
    }
    // Отримуємо всі фотографії оголошення для видалення файлів
    const photos = await db.all('SELECT * FROM photos WHERE announcementId =
?', id);
    // Видалення файлів фотографій
    for (const photo of photos) {
      const filePath = path.join(__dirname, 'public', photo.url);
      try {
        if (fs.existsSync(filePath)) {
          fs.unlinkSync(filePath);

```

```

    }
  } catch (err) {
    console.error('Помилка видалення файлу:', err);
  }
}
// Видалення оголошення (каскадно видаляться і фотографії)
await db.run('DELETE FROM announcements WHERE id = ?', id);
// Відправлення підтвердження
res.json({ message: 'Оголошення успішно видалено' });
} catch (error) {
  console.error('Помилка видалення оголошення:', error);
  res.status(500).json({ message: 'Помилка сервера при видаленні оголошення' });
}
});

// Запит на отримання преміум-статусу (для клієнта)
app.post('/api/premium-request', authMiddleware,
roleMiddleware(['client']), async (req, res) => {
  try {
    // Перевірка, чи користувач вже має преміум-статус
    if (req.user.isPremium) {
      return res.status(400).json({
        message: 'Ви вже маєте преміум-статус'
      });
    }
    // Перевірка, чи є вже активний запит від цього користувача
    const existingRequest = await db.get(
      "SELECT * FROM chats WHERE userId = ? AND type = 'premium-request' AND
status = 'pending'",
      req.user.id
    );
    if (existingRequest) {
      return res.status(400).json({
        message: 'У вас вже є активний запит на отримання преміум-статусу'
      });
    }
    const now = new Date().toISOString();
    // Створення нового чату для запиту на преміум
    const chatResult = await db.run(
      "INSERT INTO chats (userId, type, status, createdAt) VALUES (?,
'premium-request', 'pending', ?)",
      [req.user.id, now]
    );
    const newChatId = chatResult.lastID;
    // Створення першого повідомлення в чаті з запитом
    await db.run(
      "INSERT INTO messages (chatId, senderId, text, createdAt) VALUES (?, ?,
'Прошу надати мені преміум-статус', ?)",
      [newChatId, req.user.id, now]
    );
    // Відправлення відповіді
    res.status(201).json({
      message: 'Запит на преміум-статус відправлено',
      chatId: newChatId
    });
  } catch (error) {
    console.error('Помилка створення запиту на преміум:', error);
    res.status(500).json({ message: 'Помилка сервера при створенні запиту на преміум' });
  }
});

// Отримання списку чатів користувача
app.get('/api/chats', authMiddleware, async (req, res) => {
  try {

```

```

let query = 'SELECT * FROM chats';
const params = [];
// Для менеджера - всі чати, для клієнта - тільки його чати
if (req.user.role !== 'manager') {
  query += ' WHERE userId = ?';
  params.push(req.user.id);
}

query += ' ORDER BY createdAt DESC';
const chats = await db.all(query, params);
// Додавання останнього повідомлення до кожного чату
const chatsWithLastMessage = [];
for (const chat of chats) {
  const lastMessage = await db.get(
    'SELECT * FROM messages WHERE chatId = ? ORDER BY createdAt DESC LIMIT
1',
    chat.id
  );
  chatsWithLastMessage.push({
    ...chat,
    lastMessage
  });
}
// Відправлення списку чатів
res.json(chatsWithLastMessage);
} catch (error) {
  console.error('Помилка отримання чатів:', error);
  res.status(500).json({ message: 'Помилка сервера при отриманні чатів'
});
}
});

// Отримання повідомлень чату
app.get('/api/chats/:id/messages', authMiddleware, async (req, res) => {
  try {
    const chatId = parseInt(req.params.id);
    // Перевіряємо існування чату
    const chat = await db.get('SELECT * FROM chats WHERE id = ?', chatId);
    if (!chat) {
      return res.status(404).json({ message: 'Чат не знайдено' });
    }
    // Перевірка прав доступу (власник чату або менеджер)
    if (req.user.role !== 'manager' && chat.userId !== req.user.id) {
      return res.status(403).json({ message: 'Ви не маєте доступу до цього
чату' });
    }
    // Отримання всіх повідомлень чату та сортування за часом
    const messages = await db.all(
      'SELECT * FROM messages WHERE chatId = ? ORDER BY createdAt ASC',
      chatId
    );
    // Відправлення повідомлень
    res.json(messages);
  } catch (error) {
    console.error('Помилка отримання повідомлень:', error);
    res.status(500).json({ message: 'Помилка сервера при отриманні
повідомлень' });
  }
});

// Відправка повідомлення в чат
app.post('/api/chats/:id/messages', authMiddleware, async (req, res) =>
{
  try {
    const chatId = parseInt(req.params.id);
    const { text } = req.body;
    // Перевіряємо існування чату

```

```

const chat = await db.get('SELECT * FROM chats WHERE id = ?', chatId);
if (!chat) {
return res.status(404).json({ message: 'Чат не знайдено' });
}
// Перевірка прав доступу (власник чату або менеджер)
if (req.user.role !== 'manager' && chat.userId !== req.user.id) {
return res.status(403).json({ message: 'Ви не маєте доступу до цього чату' });
}
const now = new Date().toISOString();
// Створення нового повідомлення
const result = await db.run(
'INSERT INTO messages (chatId, senderId, text, createdAt) VALUES (?, ?, ?, ?)',
[chatId, req.user.id, text, now]
);
// Оновлення статусу чату, якщо потрібно
if (chat.status === 'pending' && req.user.role === 'manager') {
await db.run(
'UPDATE chats SET status = 'responded' WHERE id = ?',
chatId
);
}
// Отримання створеного повідомлення
const newMessage = await db.get('SELECT * FROM messages WHERE id = ?',
result.lastID);
// Відправлення створеного повідомлення
res.status(201).json(newMessage);
} catch (error) {
console.error('Помилка відправки повідомлення:', error);
res.status(500).json({ message: 'Помилка сервера при відправці повідомлення' });
}
});
// Управління преміум-статусом користувача (для менеджера)
app.post('/api/users/:id/premium', authMiddleware,
roleMiddleware(['manager']), async (req, res) => {
try {
const userId = parseInt(req.params.id);
const { isPremium } = req.body;
// Пошук користувача за ID
const user = await db.get('SELECT * FROM users WHERE id = ?', userId);
if (!user) {
return res.status(404).json({ message: 'Користувача не знайдено' });
}
// Оновлення статусу преміум
await db.run(
'UPDATE users SET isPremium = ? WHERE id = ?',
[isPremium ? 1 : 0, userId]
);
// Отримання оновленого користувача
const updatedUser = await db.get('SELECT * FROM users WHERE id = ?',
userId);
// Генерація нового токена для користувача
const token = jwt.sign(
{
id: updatedUser.id,
username: updatedUser.username,
role: updatedUser.role,
isPremium: updatedUser.isPremium === 1
},
JWT_SECRET,
{ expiresIn: '24h' }
);

```

```

// Відправлення відповіді з оновленими даними
res.json({
  message: isPremium ? 'Преміум-статус наданий' : 'Преміум-статус
скасований',
  user: {
    id: updatedUser.id,
    username: updatedUser.username,
    email: updatedUser.email,
    role: updatedUser.role,
    isPremium: updatedUser.isPremium === 1
  },
  token
});
} catch (error) {
  console.error('Помилка оновлення преміум-статусу:', error);
  res.status(500).json({ message: 'Помилка сервера при оновленні преміум-
статусу' });
}
});

// Видалення користувача (для менеджера)
app.delete('/api/users/:id', authMiddleware,
roleMiddleware(['manager']), async (req, res) => {
  try {
    const userId = parseInt(req.params.id);
    // Пошук користувача за ID
    const user = await db.get('SELECT * FROM users WHERE id = ?', userId);
    if (!user) {
      return res.status(404).json({ message: 'Користувача не знайдено' });
    }
    // Отримання всіх оголошень користувача для видалення фотографій
    const userAnnouncements = await db.all('SELECT id FROM announcements
WHERE userId = ?', userId);
    for (const announcement of userAnnouncements) {
      const photos = await db.all('SELECT url FROM photos WHERE announcementId
= ?', announcement.id);
      // Видалення файлів фотографій
      for (const photo of photos) {
        const filePath = path.join(__dirname, 'public', photo.url);
        try {
          if (fs.existsSync(filePath)) {
            fs.unlinkSync(filePath);
          }
        } catch (err) {
          console.error('Помилка видалення файлу:', err);
        }
      }
    }
    // Видалення користувача (каскадно видаляться всі пов'язані дані)
    await db.run('DELETE FROM users WHERE id = ?', userId);
    // Відправлення підтвердження
    res.json({ message: 'Користувача та всі його дані успішно видалено' });
  } catch (error) {
    console.error('Помилка видалення користувача:', error);
    res.status(500).json({ message: 'Помилка сервера при видаленні
користувача' });
  }
});

// Отримання списку користувачів (для менеджера)
app.get('/api/users', authMiddleware, roleMiddleware(['manager']), async
(req, res) => {
  try {
    // Отримання всіх користувачів
    const users = await db.all('SELECT id, username, email, role, isPremium,
createdAt FROM users');

```


```

// Перетворення булевих значень
const safeUsers = users.map(user => ({
  ...user,
  isPremium: user.isPremium === 1
}));
// Відправлення списку
res.json(safeUsers);
} catch (error) {
  console.error('Помилка отримання користувачів:', error);
  res.status(500).json({ message: 'Помилка сервера при отриманні
користувачів' });
}
});
// Для всіх інших маршрутів повертаємо index.html
app.get('/app/*', (req, res) => {
  res.sendFile(path.join(__dirname, 'public', 'index.html'));
});
// Обробка 404
app.use((req, res) => {
  if (req.path.startsWith('/api/')) {
    return res.status(404).json({ message: 'API-маршрут не знайдено' });
  }
  res.status(404).sendFile(path.join(__dirname, 'public', 'index.html'));
});
// Обробка помилок
app.use((err, req, res, next) => {
  console.error('Помилка сервера:', err);
  res.status(500).json({ message: 'Помилка сервера' });
});
// Запуск сервера
(async () => {
  try {
    await initializeDatabase();
    app.listen(port, () => {
      console.log(`Сервер запущено на порту ${port}`);
      console.log(`Відкрийте у браузері: http://localhost:${port}`);
    });
  } catch (err) {
    console.error('Помилка при запуску сервера:', err);
  }
})();

```

seed.js

```

// seed.js
const sqlite3 = require('sqlite3').verbose();
const { open } = require('sqlite');
const bcrypt = require('bcrypt');
const fs = require('fs');
const path = require('path');
const axios = require('axios');

// Шлях до бази даних
const DB_PATH = path.join(__dirname, 'database.sqlite');
// Шлях до папки для зображень
const UPLOADS_DIR = path.join(__dirname, 'public', 'uploads');
// Функція для генерації випадкового числа
function getRandomInt(min, max) {
  return Math.floor(Math.random() * (max - min + 1)) + min;
}
// Функція для генерації випадкової дати в межах останніх днів
function getRandomDate(daysBack = 30) {
  const today = new Date();
  const randomDaysAgo = getRandomInt(0, daysBack);

```

```

    const pastDate = new Date(today);
    pastDate.setDate(today.getDate() - randomDaysAgo);
    return pastDate.toISOString();
  }
  // Функція для вибору випадкового елемента з масиву
  function getRandomElement(array) {
    return array[Math.floor(Math.random() * array.length)];
  }
  // Кольорові палітри
  const colorPalettes = [
    [
      { hex: '#756867', name: 'Древесина' },
      { hex: '#D5D6D2', name: 'Гравій' },
      { hex: '#353C3F', name: 'Грифель' },
      { hex: '#FF8D3F', name: 'Оранжевий' }
    ],
    [
      { hex: '#1A535C', name: 'Темно-бірюзовий' },
      { hex: '#4ECDC4', name: 'Бірюзовий' },
      { hex: '#F7FFF7', name: 'Білий' },
      { hex: '#FF6B6B', name: 'Коралловий' }
    ],
    [
      { hex: '#8D8741', name: 'Оливковий' },
      { hex: '#659DBD', name: 'Голубий' },
      { hex: '#DAAD86', name: 'Пісочний' },
      { hex: '#FBEBC1', name: 'Кремовий' }
    ],
    [
      { hex: '#05386B', name: 'Темно-синій' },
      { hex: '#379683', name: 'Зелений' },
      { hex: '#5CDB95', name: 'Салатовий' },
      { hex: '#EDF5E1', name: 'Світло-зелений' }
    ]
  ];
  // Тестові дані для заповнення
  const titles = [
    'Затишна квартира в центрі', 'Новий iPhone 15 Pro', 'Продам автомобіль BMW X5',
    'Шукаю роботу програміста', 'Надаю послуги ремонту', 'Продам ноутбук Dell',
    'Оренда офісу в бізнес-центрі', 'Продаж земельної ділянки', 'Курси англійської мови',
    'Ремонт побутової техніки', 'Вакансія менеджера з продажу', 'Продам диван',
    'Оренда квартири', 'Продам телевізор Samsung', 'Шукаю роботу водія',
    'Пропоную послуги перекладача', 'Продам гітару', 'Шукаю няню для дитини',
    'Продаж ресторанного бізнесу', 'Здам в оренду приміщення'
  ];
  const descriptions = [
    'Продам затишну 2-кімнатну квартиру в центрі міста. Євроремонт, меблі, техніка. Поруч парк, магазини, школа.',
    'Продам новий iPhone 15 Pro 256GB, колір - сірий. На гарантії, повний комплект.',
    'Продам BMW X5 2020 року випуску, пробіг 15000 км, дизель, максимальна комплектація.',
    'Шукаю роботу програміста. Маю досвід роботи з JavaScript, React, Node.js. Розгляну всі пропозиції.',
    'Надаю послуги з ремонту квартир: штукатурка, шпаклівка, фарбування, укладання плитки та ламінату.',
    'Продам ноутбук Dell XPS 15, i7, 16GB RAM, 512GB SSD, GeForce GTX 1650. Гарний стан, повний комплект.',
    'Здам в оренду офіс 50 кв.м. в бізнес-центрі класу А. Сучасний ремонт,

```

```

меблі, кондиціонер.',
    'Продам земельну ділянку 15 соток під будівництво. Комунікації поруч,
хороший під\'їзд.',
    'Курси англійської мови для початківців та середнього рівня.
Індивідуальний підхід, гнучкий графік.',
    'Ремонт пральних машин, холодильників, мікрохвильовок та іншої побутової
техніки. Швидко та якісно.'
];
// Список URL стокових зображень для кожної категорії
const categoryImages = {
  1: [ // Нерухомість
    'https://images.unsplash.com/photo-1560518883-
ce09059eeffa?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1600607687939-
ce8a6c25118c?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1600585154340-
be6161a56a0c?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1583847268964-
b28dc8f51f92?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1568605114967-
8130f3a36994?w=800&h=600&q=80'
  ],
  2: [ // Транспорт
    'https://images.unsplash.com/photo-1494976388531-
d1058494cdd8?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1549317661-
bd32c8ce0db2?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1543465077-
db45d34b88a5?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1596979874675-
3274f1ebb6fb?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1618221195710-
dd6b41faaea6?w=800&h=600&q=80'
  ],
  3: [ // Робота
    'https://images.unsplash.com/photo-1521737711867-
e3b97375f902?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1497032628192-
86f99bcd76bc?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1454165804606-
c3d57bc86b40?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1486312338219-
ce68d2c6f44d?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1573164713988-
8665fc963095?w=800&h=600&q=80'
  ],
  4: [ // Електроніка
    'https://images.unsplash.com/photo-1496181133206-
80ce9b88a853?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1587614382346-
4ec70e388b28?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1504707748692-
419802cf939d?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1498049794561-
7780e7231661?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1525547719571-
a2d4ac8945e2?w=800&h=600&q=80'
  ],
  5: [ // Послуги
    'https://images.unsplash.com/photo-1521737604893-
d14cc237f11d?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1484863137850-
59afcfce05386?w=800&h=600&q=80',
    'https://images.unsplash.com/photo-1507679799987-

```

```

c73779587ccf?w=800&h=600&q=80',
  'https://images.unsplash.com/photo-1556740758-
90de374c12ad?w=800&h=600&q=80',
  'https://images.unsplash.com/photo-1517245386807-
bb43f82c33c4?w=800&h=600&q=80'
]
};

// Функція для завантаження зображення з URL
async function loadImage(url, filename) {
  try {
    // Перевіряємо чи існує директорія, якщо ні - створюємо
    if (!fs.existsSync(UPLOADS_DIR)) {
      fs.mkdirSync(UPLOADS_DIR, { recursive: true });
    }
    const filePath = path.join(UPLOADS_DIR, filename);
    // Якщо файл вже існує - пропускаємо завантаження
    if (fs.existsSync(filePath)) {
      return filename;
    }
    const response = await axios({
      method: 'GET',
      url: url,
      responseType: 'stream'
    });
    const writer = fs.createWriteStream(filePath);
    response.data.pipe(writer);
    return new Promise((resolve, reject) => {
      writer.on('finish', () => resolve(filename));
      writer.on('error', reject);
    });
  } catch (error) {
    console.error(`Помилка завантаження зображення ${url}:`, error.message);
    throw error;
  }
}

// Основна функція для заповнення бази даних
async function seedDatabase() {
  console.log('Починаємо заповнення бази даних тестовими даними...');
  try {
    // Відкриваємо з'єднання з базою даних
    const db = await open({
      filename: DB_PATH,
      driver: sqlite3.Database
    });
    // Перевіряємо, чи існує база даних і таблиці
    try {
      // Перевіряємо, чи таблиці вже створені
      await db.get('SELECT 1 FROM users LIMIT 1');
    } catch (error) {
      console.error('Помилка: База даних або таблиці не існують');
      console.log('Спочатку запусить сервер для створення структури бази
даних');
    }
    await db.close();
    return;
  }
  // Очищаємо всі таблиці перед заповненням
  await db.exec(`
DELETE FROM messages;
DELETE FROM chats;
DELETE FROM photos;
DELETE FROM announcements;
DELETE FROM users;
DELETE FROM sqlite_sequence;
-- Відновлюємо категорії

```

```

DELETE FROM categories;
INSERT INTO categories (name) VALUES
  ('Нерухомість'),
  ('Транспорт'),
  ('Робота'),
  ('Електроніка'),
  ('Послуги');
`);
console.log('Таблиці очищені, починаємо заповнення...');
// Хешування пароля для тестових користувачів
const passwordHash = await bcrypt.hash('password123', 10);
// 1. Створення менеджерів
await db.run(`
  INSERT INTO users (username, password, email, role, isPremium,
createdAt)
  VALUES (?, ?, ?, ?, ?, ?)
  `, [ 'manager1', passwordHash, 'manager1@example.com', 'manager', 0,
getRandomDate(60) ]);
await db.run(`
  INSERT INTO users (username, password, email, role, isPremium,
createdAt)
  VALUES (?, ?, ?, ?, ?, ?)
  `, [ 'manager2', passwordHash, 'manager2@example.com', 'manager', 0,
getRandomDate(60) ]);
console.log('Створено 2 менеджерів');
// 2. Створення звичайних користувачів
for (let i = 1; i <= 10; i++) {
  const isPremium = i % 3 === 0 ? 1 : 0; // Кожен третій користувач -
преміум
  await db.run(`
    INSERT INTO users (username, password, email, role, isPremium,
createdAt)
    VALUES (?, ?, ?, ?, ?, ?)
    `, [
      `user${i}`,
      passwordHash,
      `user${i}@example.com`,
      'client',
      isPremium,
      getRandomDate(60)
    ]);
  }
  console.log('Створено 10 звичайних користувачів');
  // Отримуємо всіх користувачів клієнтського типу
  const clientUsers = await db.all("SELECT id FROM users WHERE role =
'client'");
  // 3. Створення оголошень і завантаження фотографій
  console.log('Починаємо створення оголошень та завантаження
фотографій...');
  for (let i = 1; i <= 40; i++) {
    const userId = getRandomElement(clientUsers).id;
    const categoryId = getRandomInt(1, 5);
    const titleIndex = getRandomInt(0, titles.length - 1);
    const createdAt = getRandomDate(30);
    const updatedAt = new Date(new Date(createdAt).getTime() +
getRandomInt(1, 10) * 24 * 60 * 60 * 1000).toISOString();
    // Визначаємо, чи матиме оголошення кольорову гаму
    const hasColorPalette = Math.random() > 0.3; // 70% оголошень мають
кольорову гаму
    let colorPaletteJson = null;
    if (hasColorPalette) {
      colorPaletteJson = JSON.stringify(getRandomElement(colorPalettes));
    }
    // Створюємо оголошення

```

```

const result = await db.run(`
INSERT INTO announcements (
title, description, categoryId, price, contactPhone, contactEmail,
userId, createdAt, updatedAt, colorPalette
) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
`, [
titles[titleIndex],
`${descriptions[titleIndex % descriptions.length]} (Оголошення №${i})`,
categoryId,
getRandomInt(100, 50000),
`+380${getRandomInt(930000000, 999999999)}`,
`contact${i}@example.com`,
userId,
createdAt,
updatedAt,
colorPaletteJson
]);
const announcementId = result.lastID;
// Додаємо фотографії до оголошення (70% оголошень матимуть фото)
if (Math.random() > 0.3) {
const photosCount = getRandomInt(1, 5); // Від 1 до 5 фотографій
try {
// Вибираємо випадкові зображення для категорії
const categoryImageUrls = categoryImages[categoryId];
for (let j = 0; j < photosCount; j++) {
const imageUrl = getRandomElement(categoryImageUrls);
const imageExt = path.extname(new URL(imageUrl).pathname) || '.jpg';
const filename = `image-${announcementId}-${j + 1}-
${Date.now()}${imageExt}`;
// Завантажуємо зображення
const savedFilename = await downloadImage(imageUrl, filename);
const fileUrl = `/uploads/${savedFilename}`;
// Зберігаємо інформацію про фото в базі даних
await db.run(`
INSERT INTO photos (announcementId, url, filename, createdAt)
VALUES (?, ?, ?, ?)
`, [
announcementId,
fileUrl,
savedFilename,
createdAt
]);
}
} catch (error) {
console.error(`Помилка при завантаженні фото для оголошення
${announcementId}:`, error.message);
}
}
if (i % 10 === 0) {
console.log(`Створено ${i} з 40 оголошень...`);
}
console.log('Створено 40 оголошень з фотографіями та кольоровими
гамами');
// 4. Створення чатів і повідомлень
for (const user of clientUsers) {
// 50% користувачів матимуть запити на преміум
if (Math.random() > 0.5) {
const now = getRandomDate(20);
const status = getRandomElement(['pending', 'responded']);
// Створюємо чат
const chatResult = await db.run(`
INSERT INTO chats (userId, type, status, createdAt)
VALUES (?, ?, ?, ?)

```

```

    `, [
    user.id,
    'premium-request',
    status,
    now
    ]);
    const chatId = chatResult.lastID;
    // Додаємо початкове повідомлення
    const messageResult = await db.run(`
    INSERT INTO messages (chatId, senderId, text, createdAt)
    VALUES (?, ?, ?, ?)
    `, [
    chatId,
    user.id,
    'Прошу надати мені преміум-статус',
    now
    ]);
    // Якщо статус "responded", додаємо відповідь від менеджера
    if (status === 'responded') {
    const managerId = getRandomInt(1, 2);
    const isApproved = Math.random() > 0.3; // 70% запитів схвалено
    const responseDate = new Date(new Date(now).getTime() + getRandomInt(1,
24) * 60 * 60 * 1000).toISOString());
    await db.run(`
    INSERT INTO messages (chatId, senderId, text, createdAt)
    VALUES (?, ?, ?, ?)
    `, [
    chatId,
    managerId,
    isApproved
    ? 'Ваш запит на преміум-статус схвалено. Тепер ви маєте преміум-статус!'
    : 'Ваш запит на преміум-статус відхилено.',
    responseDate
    ]);
    // Якщо запит схвалено і користувач ще не має преміум-статусу, оновлюємо
його статус
    if (isApproved) {
    await db.run(`
    UPDATE users SET isPremium = 1 WHERE id = ? AND isPremium = 0
    `, [user.id]);
    }
    // Додаємо ще 1-3 повідомлення для листування
    const additionalMessagesCount = getRandomInt(1, 3);
    let lastMessageDate = new Date(responseDate);
    for (let i = 0; i < additionalMessagesCount; i++) {
    const sender = i % 2 === 0 ? user.id : managerId;

    const responseTexts = [
    'Дякую за відповідь!',
    'Коли зміни вступлять в силу?',
    'Можу я дізнатися більше про переваги преміум-статусу?',
    'Як довго буде діяти преміум-статус?',
    'Дякую за вашу допомогу!'
    ];
    const managerTexts = [
    'Зміни вже вступили в силу.',
    'Преміум-статус діє необмежений термін.',
    'Ви можете розміщувати необмежену кількість оголошень.',
    'Ваші оголошення завжди будуть на перших позиціях у пошуку.',
    'Звертайтеся, якщо виникнуть питання!'
    ];
    lastMessageDate = new Date(lastMessageDate.getTime() + getRandomInt(5,
60) * 60 * 1000);
    await db.run(`

```


```

INSERT INTO messages (chatId, senderId, text, createdAt)
VALUES (?, ?, ?, ?)
`, [
  chatId,
  sender,
  sender === user.id ? getRandomElement(responseTexts) :
getRandomElement(managerTexts),
  lastMessageDate.toISOString()
]);
}
}
}
}
console.log('Створено чати та повідомлення');
// Закриваємо з'єднання з базою даних
await db.close();
console.log('База даних успішно заповнена тестовими даними!');
} catch (error) {
  console.error('Помилка при заповненні бази даних:', error);
}
}

// Запускаємо функцію, якщо скрипт викликано напряму
if (require.main === module) {
  seedDatabase().then(() => {
    console.log('Скрипт завершено');
  }).catch(err => {
    console.error('Помилка виконання скрипту:', err);
  });
}
module.exports = { seedDatabase };

```

eslint.config.js

```

module.exports = [
  {
    rules: {
      // Add rules here.
    }
  }
];

```

app.js

```

// Створюємо HTM з бібліотекою React
const html = htm.bind(React.createElement);
// URL API
const API_URL = window.location.origin + '/api';
// Контекст для авторизації
const AuthContext = React.createContext({
  isAuthenticated: false,
  user: null,
  token: null,
  login: () => { },
  logout: () => { },
  updateUserData: () => { }
});
// Головний компонент застосунку
function App() {
  const [currentPage, setCurrentPage] = React.useState('home');

```

```

const [params, setParams] = React.useState({});
// Обробник кліка по посиланню
const handleNavigation = (e, page, pageParams = {}) => {
  if (e && e.preventDefault()) {
    e.preventDefault();
  }
  setCurrentPage(page);
  setParams(pageParams);
  window.scrollTo(0, 0);
};
return html`
<${AuthProvider}>
  <div className="app">
    <${Header} onNavigate=${handleNavigation} />
    <main className="main-content">
      <${Routes} currentPage=${currentPage} params=${params}
onNavigate=${handleNavigation} />
    </main>
    <${Footer} />
  </div>
</${AuthProvider}>
`;
}
// Провайдер автентифікації
function AuthProvider({ children }) {
  // Ініціалізуємо стан з локального сховища
  const [authState, setAuthState] = React.useState(() => {
    const token = localStorage.getItem('token');
    const user = JSON.parse(localStorage.getItem('user') || 'null');
    return {
      isAuthenticated: !!token,
      user,
      token
    };
  });
  // Функція для входу користувача
  const login = (token, user) => {
    localStorage.setItem('token', token);
    localStorage.setItem('user', JSON.stringify(user));
    setAuthState({
      isAuthenticated: true,
      user,
      token
    });
  };
  // Функція для виходу користувача
  const logout = () => {
    localStorage.removeItem('token');
    localStorage.removeItem('user');
    setAuthState({
      isAuthenticated: false,
      user: null,
      token: null
    });
  };
  // Функція для оновлення даних користувача
  const updateUserData = (userData) => {
    const updatedUser = { ...authState.user, ...userData };
    localStorage.setItem('user', JSON.stringify(updatedUser));
    setAuthState({
      ...authState,
      user: updatedUser
    });
  };
};

```

```

        return html`
<${AuthContext.Provider}
value=${{
    ...authState,
    login,
    logout,
    updateUserData
}}
>
${children}
</${AuthContext.Provider}>
`;
    }

    // Компонент маршрутизації
    function Routes({ currentPage, params, onNavigate }) {
        const { isAuthenticated, user } = React.useContext(AuthContext);
        // Рендеринг потрібної сторінки
        const renderPage = () => {
            switch (currentPage) {
                case 'home':
                    return html`<${HomePage} onNavigate=${onNavigate} />`;
                case 'login':
                    return html`<${LoginPage} onNavigate=${onNavigate} />`;
                case 'register':
                    return html`<${RegisterPage} onNavigate=${onNavigate} />`;
                case 'announcement-details':
                    return html`<${AnnouncementDetailsPage} onNavigate=${onNavigate}
params=${params} />`;
                case 'create-announcement':
                    return html`<${CreateAnnouncementPage} onNavigate=${onNavigate} />`;
                case 'edit-announcement':
                    return html`<${EditAnnouncementPage} onNavigate=${onNavigate}
params=${params} />`;
                case 'my-announcements':
                    return html`<${MyAnnouncementsPage} onNavigate=${onNavigate} />`;
                case 'chats':
                    return html`<${ChatsPage} onNavigate=${onNavigate} />`;
                case 'chat':
                    return html`<${ChatPage} onNavigate=${onNavigate} params=${params} />`;
                case 'admin-users':
                    return html`<${AdminUsersPage} onNavigate=${onNavigate} />`;
                case 'premium-request':
                    return html`<${PremiumRequestPage} onNavigate=${onNavigate} />`;
                default:
                    return html`<${HomePage} onNavigate=${onNavigate} />`;
            }
        };
        return html`
<div className="page-content">
${renderPage()}
</div>
`;
    }

    // Компонент хедера
    function Header({ onNavigate }) {
        const { isAuthenticated, user, logout } = React.useContext(AuthContext);
        return html`
<header className="header">
<div className="logo">
<a href="/" onClick=${(e) => onNavigate(e, 'home')}>Дошка Оголошень</a>
</div>
<nav className="navigation">
<ul>
<li key="home">

```

```

        <a href="/" onClick=${(e) => onNavigate(e, 'home')}>Головна</a>
    </li>
    ${isAuthenticated && user.role === 'client' ? [
        html`<li key="create-announcement">
            <a href="/create" onClick=${(e) => onNavigate(e, 'create-
announcement')}>>
                Додати оголошення
            </a>
        </li>`,
        html`<li key="my-announcements">
            <a href="/my-announcements" onClick=${(e) => onNavigate(e, 'my-
announcements')}>>
                Мої оголошення
            </a>
        </li>`,
        html`<li key="chats">
            <a href="/chats" onClick=${(e) => onNavigate(e, 'chats')}>>
                Повідомлення
            </a>
        </li>`,
        !user.isPremium ? html`<li key="premium">
            <a href="/premium" onClick=${(e) => onNavigate(e, 'premium-
request')}>>
                Отримати преміум
            </a>
        </li>` : null
    ] : null}
    ${isAuthenticated && user.role === 'manager' ? [
        html`<li key="admin-users">
            <a href="/admin/users" onClick=${(e) => onNavigate(e, 'admin-
users')}>>
                Користувачі
            </a>
        </li>`,
        html`<li key="manager-chats">
            <a href="/chats" onClick=${(e) => onNavigate(e, 'chats')}>>
                Запити
            </a>
        </li>`
    ] : null}
    </ul>
</nav>
<div className="auth-links">
    ${isAuthenticated ? html`
        <div className="user-panel">
            <span className="username">
                ${user.username} ${user.isPremium ? '★' : ''}
            </span>
            <button onClick=${logout} className="logout-btn">Вийти</button>
        </div>
        ` : [
            html`<a key="login" href="/login" onClick=${(e) => onNavigate(e,
'login')}>>Увійти</a>`,
            html`<a key="register" href="/register" onClick=${(e) => onNavigate(e,
'register')}>>Реєстрація</a>`
        ]}
    </div>
</header>
`;
}

// Компонент футера
function Footer() {
    return html`
        <footer className="footer">

```

```

    <div className="footer-content">
      <p>&copy; 2025 Дошка Оголошень. Всі права захищені.</p>
    </div>
  </footer>
`
};

// Компонент для відображення фотографій в карусельному форматі
function PhotoCarousel({ photos }) {
  const [currentPhoto, setCurrentPhoto] = React.useState(0);
  if (!photos || photos.length === 0) {
    return html`<div className="photo-placeholder">Немає фотографій</div>`;
  }
  const handlePrev = () => {
    setCurrentPhoto(prev => (prev === 0 ? photos.length - 1 : prev - 1));
  };
  const handleNext = () => {
    setCurrentPhoto(prev => (prev === photos.length - 1 ? 0 : prev + 1));
  };
  return html`
    <div className="photo-carousel">
      <div className="main-photo">
        <img src=${photos[currentPhoto].url} alt="Фото оголошення" />
      <div className="carousel-btn prev"
        onClick=${handlePrev}></div>
      <div className="carousel-btn next"
        onClick=${handleNext}></div>
      <div className="photo-counter">${currentPhoto + 1} /
      ${photos.length}</div>
      <div className="thumbnails">
        ${photos.map((photo, index) => html`
          <div
            key=${photo.id}
            className=${`thumbnail ${index === currentPhoto ? 'active' : ''}`}
            onClick=${() => setCurrentPhoto(index)}
          >
            <img src=${photo.url} alt="Мініатюра" />
          </div>
        `)}
      </div>
    </div>
  `;
}

// Компонент для відображення кольорової гами оголошення
function ColorDisplay({ colors }) {
  if (!colors || colors.length === 0) {
    return null;
  }
  return html`
    <div className="color-display">
      <h3>Кольорова гама:</h3>
      <div className="color-chips">
        ${colors.map(color => html`
          <div className="color-chip" key=${color.hex}>
            <div className="color-preview" style=${{
              backgroundColor: color.hex
            }}>
            <div className="color-info">
              <div className="color-name">${color.name}</div>
              <div className="color-hex">${color.hex}</div>
            </div>
          </div>
        `)}
      </div>
    </div>
  `;
}

```

```

        </div>
      </div>
    `))}
  </div>
</div>
`;
}

// Компонент для завантаження фотографій
function PhotoUploader({ photos = [], announcementId, onPhotosChange, token,
maxPhotos = 8 }) {
  const [uploading, setUploading] = React.useState(false);
  const [error, setError] = React.useState(null);
  const fileInputRef = React.useRef(null);
  // Обробник завантаження нових фотографій
  const handleUpload = async (e) => {
    const files = e.target.files;
    if (!files || files.length === 0) return;
    // Перевірка максимальної кількості фотографій
    if (photos.length + files.length > maxPhotos) {
      setError(`Максимальна кількість фотографій: ${maxPhotos}`);
      return;
    }
    setUploading(true);
    setError(null);
    const formData = new FormData();
    Array.from(files).forEach(file => {
      formData.append('photos', file);
    });
    try {
      const response = await
fetch(`${API_URL}/announcements/${announcementId}/photos`, {
method: 'POST',
headers: {
Authorization: `Bearer ${token}`
},
body: formData
});
      const data = await response.json();
      if (!response.ok) {
        throw new Error(data.message || 'Помилка завантаження фотографій');
      }
      onPhotosChange([...photos, ...data.photos]);
    } catch (err) {
      setError(err.message);
    } finally {
      setUploading(false);
      if (fileInputRef.current) {
        fileInputRef.current.value = '';
      }
    }
  };
  // Обробник видалення фотографії
  const handleDelete = async (photoId) => {
    if (!confirm('Ви впевнені, що хочете видалити цю фотографію?')) {
      return;
    }
    try {
      const response = await fetch(`${API_URL}/photos/${photoId}`, {
method: 'DELETE',
headers: {
Authorization: `Bearer ${token}`
},
});
    } catch (err) {
      setError(err.message);
    } finally {
      setUploading(false);
    }
  };
  if (!response.ok) {

```

```

        const errorData = await response.json();
        throw new Error(errorData.message || 'Помилка видалення
фотографії');
    }
    onPhotosChange(photos.filter(photo => photo.id !== photoId));
    } catch (err) {
    setError(err.message);
    }

    };
    return html`
        <div className="photo-uploader">
        <h3>Фотографії (${photos.length}/${maxPhotos})</h3>
        ${error ? html`<div className="error">${error}</div>` : null}
        <div className="photos-grid">
        ${photos.map(photo => html`
            <div className="photo-item" key=${photo.id}>
                <img src=${photo.url} alt="Фото оголошення" />
                <button
                    className="photo-delete-btn"
                    onClick=${() => handleDelete(photo.id)}
                    title="Видалити фото"
                >×</button>
            </div>
        `)}
        ${photos.length < maxPhotos ? html`
            <div className="photo-upload-placeholder" onClick=${() =>
fileInputRef.current.click()}>
                <span className="plus-icon">+</span>
                <span>Додати фото</span>
            </div>
        ` : null}
        </div>
        <input
            type="file"
            ref=${fileInputRef}
            style=${{{ display: 'none' }}}
            accept="image/*"
            multiple
            onChange=${handleUpload}
            disabled=${uploading}
        />
        ${uploading ? html`<div className="loading">Завантаження...</div>` : null}
        <div className="upload-hint">
        <p>Підтримуються формати: JPG, PNG, GIF</p>
        <p>Максимальний розмір файлу: 5 МБ</p>
        </div>
        </div>
        `;
    }

    // Компонент для вибору кольорової гами
    function ColorPaletteSelector({ onChange, selectedColors = [] }) {
        // Набір готових палітр
        const palettes = [
            {
                id: 1,
                name: 'Класична',
                colors: [
                    { hex: '#756867', name: 'Древесина' },
                    { hex: '#D5D6D2', name: 'Гравій' },
                    { hex: '#353C3F', name: 'Грифель' },
                    { hex: '#FF8D3F', name: 'Оранжевий' }
                ]
            },
        ],
    }

```

```

id: 2,
name: 'Морська',
colors: [
  { hex: '#1A535C', name: 'Темно-бірюзовий' },
  { hex: '#4ECDC4', name: 'Бірюзовий' },
  { hex: '#F7FFF7', name: 'Білий' },
  { hex: '#FF6B6B', name: 'Коралловий' }
],
{
id: 3,
name: 'Природня',
colors: [
  { hex: '#8D8741', name: 'Оливковий' },
  { hex: '#659DBD', name: 'Голубий' },
  { hex: '#DAAD86', name: 'Пісочний' },
  { hex: '#FBEED1', name: 'Кремовий' }
],
{
id: 4,
name: 'Яскрава',
colors: [
  { hex: '#05386B', name: 'Темно-синій' },
  { hex: '#379683', name: 'Зелений' },
  { hex: '#5CDB95', name: 'Салатовий' },
  { hex: '#EDF5E1', name: 'Світло-зелений' }
],
};
const [activePaletteId, setActivePaletteId] = React.useState(
  selectedColors.length > 0 ? findPaletteId(selectedColors) : null
);

// Функція для пошуку ID палітри на основі обраних кольорів
function findPaletteId(colors) {
  if (!colors || colors.length === 0) return null;
  // Порівнюємо кольори з палітрами
  for (const palette of palettes) {
    const paletteHexes = palette.colors.map(c => c.hex).sort();
    const selectedHexes = colors.map(c => c.hex).sort();
    if (paletteHexes.length === selectedHexes.length &&
      paletteHexes.every((hex, i) => hex === selectedHexes[i])) {
      return palette.id;
    }
  }
  return null;
}

// Обробник вибору палітри
const handlePaletteSelect = (palette) => {
  setActivePaletteId(palette.id);
  onChange(palette.colors);
};

return html`
  <div className="color-palette-selector">
    <h3>Кольорова гама</h3>
    <div className="palettes-grid">
      ${palettes.map(palette => html`
        <div
          key=${palette.id}
          className=${`palette-card ${activePaletteId === palette.id ? 'active' : ''}`}
          onClick=${() => handlePaletteSelect(palette)}
        >
          <div className="palette-title">${palette.name}</div>

```


```

        <div className="palette-colors">
          ${palette.colors.map(color => html`
            <div
              key=${color.hex}
              className="color-sample"
              style=${{{ backgroundColor: color.hex }}}
              title=${color.name}
            ></div>
          `)}
        </div>
      </div>
    </div>
  </div>
  ${selectedColors.length > 0 ? html`
    <div className="selected-colors">
      <h4>Обрані кольори:</h4>
      <div className="color-chips">
        ${selectedColors.map(color => html`
          <div className="color-chip" key=${color.hex}>
            <div className="color-preview" style=${{{
              backgroundColor: color.hex
            }}}></div>
            <div className="color-info">
              <div className="color-
name">${color.name}</div>
              <div className="color-
hex">${color.hex}</div>
            </div>
          </div>
        `)}
      </div>
    </div>
  ` : null}
</div>
`;
}

// Компонент головної сторінки
function HomePage({ onNavigate }) {
  const [announcements, setAnnouncements] = React.useState([]);
  const [categories, setCategories] = React.useState([]);
  const [loading, setLoading] = React.useState(true);
  const [error, setError] = React.useState(null);
  const [filters, setFilters] = React.useState({
    search: '',
    category: '',
    minPrice: '',
    maxPrice: '',
    sort: 'dateDesc',
    page: 1
  });

  const [totalPages, setTotalPages] = React.useState(1);
  const { token } = React.useContext(AuthContext);

  // Отримання категорій
  React.useEffect(() => {
    fetch(`${API_URL}/categories`)
      .then(response => {
        if (!response.ok) {
          throw new Error('Помилка завантаження категорій');
        }
        return response.json();
      })
      .then(data => {
        setCategories(data);
      })
      .catch(err => {

```

```

setError(err.message);
    });

    }, []);
    // Отримання оголошень
    React.useEffect(() => {
setLoading(true);
    const queryParams = new URLSearchParams();
    if (filters.search) queryParams.append('search', filters.search);
    if (filters.category) queryParams.append('category', filters.category);
    if (filters.minPrice) queryParams.append('minPrice', filters.minPrice);
    if (filters.maxPrice) queryParams.append('maxPrice', filters.maxPrice);
    if (filters.sort) queryParams.append('sort', filters.sort);
    if (filters.page) queryParams.append('page', filters.page);
    const headers = {};
    if (token) {
headers.Authorization = `Bearer ${token}`;
    }
    fetch(`${API_URL}/announcements?${queryParams.toString()}`, { headers })
    .then(response => {
    if (!response.ok) {
        throw new Error('Помилка завантаження оголошень');
    }
    return response.json();
})
    .then(data => {
        setAnnouncements(data.announcements);
        setTotalPages(data.totalPages);
        setLoading(false);
    })
    .catch(err => {
setError(err.message);
setLoading(false);
    });

    }, [filters, token]);
    // Обробник зміни фільтрів
    const handleFilterChange = (e) => {
    const { name, value } = e.target;
setFilters(prev => ({
    ...prev,
    [name]: value,
    page: 1 // Скидання сторінки при зміні фільтрів
}));
    };
    // Обробник зміни сторінки
    const handlePageChange = (page) => {
setFilters(prev => ({
    ...prev,
    page
}));
    window.scrollTo(0, 0);
    };
    // Обробник скидання фільтрів
    const handleResetFilters = () => {
setFilters({
    search: '',
    category: '',
    minPrice: '',
    maxPrice: '',
    sort: 'dateDesc',
    page: 1
});
    };
    // Рендеримо список оголошень
    const renderAnnouncements = () => {

```

```

    if (announcements.length === 0) {
      return html`<div className="no-announcements" key="no-
announcements">Оголошень не знайдено</div>`;
    }
    return announcements.map(announcement => html`
      <div className="announcement-card" key=${announcement.id}>
        <div className="announcement-photos">
          ${announcement.photos && announcement.photos.length > 0
            ? html`<img src=${announcement.photos[0].url}
alt="${announcement.title}" />`
            : html`<div className="no-photo">Немає фото</div>`
          }
        </div>
        <div className="announcement-content">
          <h3 className="announcement-title">
            ${announcement.title}
            ${announcement.isPremium ? html`<span className="premium-badge">★</span>` : ''}
          </h3>
          <div className="announcement-category">
            Категорія: ${categories.find(c => c.id === announcement.categoryId)?.name ||
            'Невідома'}
          </div>
          <div className="announcement-price">
            Ціна: ${announcement.price} грн
          </div>
          <div className="announcement-date">
            Опубліковано: ${new Date(announcement.createdAt).toLocaleDateString('uk-UA')}
          </div>
          <div className="announcement-description">
            ${announcement.description.length > 150
              ? `${announcement.description.substring(0, 150)}...`
              : announcement.description}
          </div>
          ${announcement.colorPalette && announcement.colorPalette.length > 0 ? html`
            <div className="color-preview">
              ${announcement.colorPalette.map(color => html`
                <div className="color-dot" key=${color.hex}
style=${{{ backgroundColor: color.hex }} title=${color.name}></div>
              `)}
            </div>
          ` : null}
          ${announcement.contactPhone ? html`
            <div className="announcement-contact">
              Телефон: ${announcement.contactPhone}
            </div>
          ` : ''}
          ${announcement.contactEmail ? html`
            <div className="announcement-contact">
              Email: ${announcement.contactEmail}
            </div>
          ` : ''}
          <button
            className="btn btn-primary"
            onClick=${(e) => onNavigate(e, 'announcement-details', { id: announcement.id })}
          >
            Детальніше
          </button>
        </div>
      </div>
    `);
  };
  // Рендеримо пагінацію
  const renderPagination = () => {
    if (totalPages <= 1) return null;

```

```

        return html`
          <div className="pagination" key="pagination">
            ${Array.from({ length: totalPages }, (_, i) => i + 1).map(page => html`
              <button
                key=${page}
                className=${`pagination-btn ${filters.page === page ? 'active' : ''}`}
                onClick=${() => handlePageChange(page)}
              >
                ${page}
              </button>
            `)}
          </div>
        `;
      };
      // Рендеримо категорії у селекті
      const renderCategoryOptions = () => {
        return [
          html`<option value="" key="all-categories">Всі категорії</option>`,
          ...categories.map(category => html`
            <option value=${category.id} key=${category.id}>
              ${category.name}
            </option>
          `)
        ];
      };
      return html`
        <div className="home-page">
          <h1>Дошка оголошень</h1>
          <div className="filters">
            <h2>Фільтри</h2>
            <div className="filter-form">
              <div className="form-group">
                <label htmlFor="search">Пошук:</label>
                <input
                  type="text"
                  id="search"
                  name="search"
                  value=${filters.search}
                  onChange=${handleFilterChange}
                  placeholder="Пошук за назвою або описом"
                />
              </div>
              <div className="form-group">
                <label htmlFor="category">Категорія:</label>
                <select
                  id="category"
                  name="category"
                  value=${filters.category}
                  onChange=${handleFilterChange}
                >
                  ${renderCategoryOptions()}
                </select>
              </div>
              <div className="form-group">
                <label htmlFor="minPrice">Мінімальна ціна:</label>
                <input
                  type="number"
                  id="minPrice"
                  name="minPrice"
                  value=${filters.minPrice}
                  onChange=${handleFilterChange}
                  placeholder="Від"
                />
              </div>
            </div>
          </div>
        </div>
      `;
    };
  };

```


```

        <div className="form-group">
            <label htmlFor="maxPrice">Максимальна ціна:</label>
            <input
type="number"
id="maxPrice"
name="maxPrice"
value=${filters.maxPrice}
onChange=${handleFilterChange}
placeholder="До"
            />
        </div>
        <div className="form-group">
            <label htmlFor="sort">Сортування:</label>
            <select
id="sort"
name="sort"
value=${filters.sort}
onChange=${handleFilterChange}
            >
                <option value="dateDesc" key="sort-1">Спочатку
новіші</option>
                <option value="dateAsc" key="sort-2">Спочатку
старіші</option>
                <option value="priceAsc" key="sort-3">Спочатку
дешевші</option>
                <option value="priceDesc" key="sort-4">Спочатку
дорожчі</option>
            </select>
        </div>
        <button className="btn btn-secondary"
onClick=${handleResetFilters}>
Скинути фільтри
        </button>
    </div>
    ${loading
? html`<div className="loading" key="loading">Завантаження...</div>`
: error
? html`<div className="error" key="error">${error}</div>`
: html`
        <div className="announcements-list" key="announcements-
list">
            ${renderAnnouncements()}
        </div>
            ${renderPagination()}
        `
    }
    </div>
    `;
}

// Компонент сторінки авторизації
function LoginPage({ onNavigate }) {
    const [formData, setFormData] = React.useState({
        username: '',
        password: ''
    });

    const [error, setError] = React.useState(null);
    const [loading, setLoading] = React.useState(false);
    const { login } = React.useContext(AuthContext);

    // Обробник зміни полів форми
    const handleChange = (e) => {
        const { name, value } = e.target;
        setFormData(prev => ({
            ...prev,

```

```

        [name]: value
    }));
    };
    // Обробник відправки форми
    const handleSubmit = async (e) => {
        e.preventDefault();
        setLoading(true);
        setError(null);
        try {
            const response = await fetch(`${API_URL}/auth/login`, {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json'
                },
                body: JSON.stringify(formData)
            });
            const data = await response.json();
            if (!response.ok) {
                throw new Error(data.message || 'Помилка входу');
            }
            // Авторизуємо користувача
            login(data.token, data.user);
            onNavigate(null, 'home');
        } catch (err) {
            setError(err.message);
        } finally {
            setLoading(false);
        }
    };
    return html`
    <div className="auth-page">
    <h1>Вхід в акаунт</h1>
    ${error ? html`<div className="error" key="login-error">${error}</div>` : ''}
    <form className="auth-form" onSubmit=${handleSubmit}>
    <div className="form-group">
    <label htmlFor="username">Логін:</label>
    <input
    type="text"
    id="username"
    name="username"
    value=${formData.username}
    onChange=${handleChange}
    required
    />
    </div>
    <div className="form-group">
    <label htmlFor="password">Пароль:</label>
    <input
    type="password"
    id="password"
    name="password"
    value=${formData.password}
    onChange=${handleChange}
    required
    />
    </div>
    <button type="submit" className="btn btn-primary" disabled=${loading}>
    ${loading ? 'Вхід...' : 'Увійти'}
    </button>
    </form>
    <p className="auth-link">
    Ще не маєте акаунту? <a href="/register" onClick=${(e) => onNavigate(e,
    'register')} key="register-link">Зареєструватися</a>
    </p>
    
```

```

    </div>
    `;
  }
}

// Компонент сторінки реєстрації
function RegisterPage({ onNavigate }) {
  const [formData, setFormData] = React.useState({
    username: '',
    email: '',
    password: '',
    passwordConfirm: '',
    role: 'client'
  });

  const [error, setError] = React.useState(null);
  const [loading, setLoading] = React.useState(false);
  const { login } = React.useContext(AuthContext);

  // Обробник зміни полів форми
  const handleChange = (e) => {
    const { name, value } = e.target;
    setFormData(prev => ({
      ...prev,
      [name]: value
    }));
  };

  // Обробник відправки форми
  const handleSubmit = async (e) => {
    e.preventDefault();
    setLoading(true);
    setError(null);

    // Перевірка співпадіння паролів
    if (formData.password !== formData.passwordConfirm) {
      setError('Паролі не співпадають');
      setLoading(false);
      return;
    }

    try {
      const response = await fetch(`${API_URL}/auth/register`, {
        method: 'POST',
        headers: {
          'Content-Type': 'application/json'
        },
        body: JSON.stringify({
          username: formData.username,
          email: formData.email,
          password: formData.password,
          role: formData.role
        })
      });

      const data = await response.json();
      if (!response.ok) {
        throw new Error(data.message || 'Помилка реєстрації');
      }

      // Авторизуємо користувача після успішної реєстрації
      login(data.token, data.user);
      onNavigate(null, 'home');
    } catch (err) {
      setError(err.message);
    } finally {
      setLoading(false);
    }
  };

  // Варіанти для вибору ролі
  const roleOptions = [
    { value: 'client', label: 'Клієнт' },
    { value: 'manager', label: 'Менеджер' }
  ];
}

```

```

    ];
    return html`
    <div className="auth-page">
      <h1>Рєєстрація</h1>
      ${error ? html`<div className="error" key="register-error">${error}</div>` : ''}
      <form className="auth-form" onSubmit=${handleSubmit}>
        <div className="form-group">
          <label htmlFor="username">Логін:</label>
          <input
            type="text"
            id="username"
            name="username"
            value=${formData.username}
            onChange=${handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label htmlFor="email">Email:</label>
          <input
            type="email"
            id="email"
            name="email"
            value=${formData.email}
            onChange=${handleChange}
            required
          />
        </div>
        <div className="form-group">
          <label htmlFor="password">Пароль:</label>
          <input
            type="password"
            id="password"
            name="password"
            value=${formData.password}
            onChange=${handleChange}
            required
            minLength="6"
          />
        </div>
        <div className="form-group">
          <label htmlFor="passwordConfirm">Підтвердження пароля:</label>
          <input
            type="password"
            id="passwordConfirm"
            name="passwordConfirm"
            value=${formData.passwordConfirm}
            onChange=${handleChange}
            required
            minLength="6"
          />
        </div>
        <div className="form-group">
          <label htmlFor="role">Тип користувача:</label>
          <select
            id="role"
            name="role"
            value=${formData.role}
            onChange=${handleChange}
            required
          >
            ${roleOptions.map(option => html`
              <option key=${option.value}
                value=${option.value}>${option.label}</option>
            `)}
          </select>
        </div>
      </form>
    </div>
  `;
}

```


```

        `))}
      </select>
    </div>
    <button type="submit" className="btn btn-primary" disabled=${loading}>
${loading ? 'Реєстрація...' : 'Зареєструватися'}
    </button>
  </form>
  <p className="auth-link">
Вже маєте акаунт? <a href="/login" onClick=${(e) => onNavigate(e, 'login')}
key="login-link">Увійти</a>
  </p>
</div>
`;
}

// Компонент сторінки деталей оголошення
function AnnouncementDetailsPage({ onNavigate, params = {} }) {
  const [announcement, setAnnouncement] = React.useState(null);
  const [loading, setLoading] = React.useState(true);
  const [error, setError] = React.useState(null);
  const { isAuthenticated, user, token } = React.useContext(AuthContext);
  // Завантаження даних оголошення
  React.useEffect(() => {
    if (!params.id) {
      setError('ID оголошення не вказано');
      setLoading(false);
      return;
    }
    const headers = {};
    if (token) {
      headers.Authorization = `Bearer ${token}`;
    }
    fetch(`${API_URL}/announcements/${params.id}`, { headers })
      .then(response => {
        if (!response.ok) {
          throw new Error('Помилка завантаження оголошення');
        }
        return response.json();
      })
      .then(data => {
        setAnnouncement(data);
        setLoading(false);
      })
      .catch(err => {
        setError(err.message);
        setLoading(false);
      });
    }, [params.id, token]);
  // Обробник видалення оголошення
  const handleDelete = async () => {
    if (!confirm('Ви впевнені, що хочете видалити це оголошення?')) {
      return;
    }
    try {
      const response = await
fetch(`${API_URL}/announcements/${announcement.id}`, {
method: 'DELETE',
headers: {
Authorization: `Bearer ${token}`
}
});
    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || 'Помилка видалення
оголошення');
    }
  }
}

```


```

    }
    alert('Оголошення успішно видалено');
    onNavigate(null, 'home');
  } catch (err) {
    alert(`Помилка: ${err.message}`);
  }
};
if (loading) {
  return html`<div className="loading">Завантаження...</div>`;
}
if (error) {
  return html`<div className="error">${error}</div>`;
}
if (!announcement) {
  return html`<div className="error">Оголошення не знайдено</div>`;
}
return html`
  <div className="announcement-details">
    <div className="announcement-details-header">
<h1>${announcement.title}</h1>
      <div className="announcement-details-actions">
        <button className="btn btn-secondary" onClick=${(e) => onNavigate(e,
'home')}>
Назад до списку
      </button>
      ${isAuthenticated && (user.id === announcement.userId || user.role ===
'manager')} ? html`
        <button className="btn btn-danger" onClick=${handleDelete}>
Видалити оголошення
      </button>
      ` : ''}
      ${isAuthenticated && user.id === announcement.userId ? html`
        <button
          className="btn btn-primary"
          onClick=${(e) => onNavigate(e, 'edit-announcement', { id: announcement.id
})}
        >
Редагувати
      </button>
      ` : ''}
    </div>
  </div>
  <div className="announcement-details-content">
    <div className="announcement-details-gallery">
      ${announcement.photos && announcement.photos.length > 0
        ? html`<${PhotoCarousel} photos=${announcement.photos} />`
        : html`<div className="no-photos">Немає фотографій</div>`}
    </div>
    <div className="announcement-details-info">
      <div className="announcement-info-block">
        <div className="announcement-price">
          <strong>Ціна:</strong> ${announcement.price} грн
        </div>
        <div className="announcement-date">
          <strong>Опубліковано:</strong> ${new
Date(announcement.createdAt).toLocaleDateString('uk-UA')}
        </div>
        ${announcement.updatedAt !== announcement.createdAt ? html`
          <div className="announcement-date">
            <strong>Оновлено:</strong> ${new
Date(announcement.updatedAt).toLocaleDateString('uk-UA')}
          </div>
          ` : ''}
      </div>
    </div>
  </div>

```

```

        </div>
        ${announcement.colorPalette && announcement.colorPalette.length > 0 ? html`
            <${ColorDisplay} colors=${announcement.colorPalette} />
            ` : ''}
        <div className="announcement-description">
            <h2>Опис</h2>
        <p>${announcement.description}</p>
        </div>
        <div className="announcement-contacts">
        <h2>Контактна інформація</h2>
        ${announcement.contactPhone ? html`
            <div className="announcement-contact">
                <strong>Телефон:</strong> <a
href=${`tel:${announcement.contactPhone}`}>${announcement.contactPhone}</a>
            </div>
            ` : ''}
        ${announcement.contactEmail ? html`
            <div className="announcement-contact">
                <strong>Email:</strong> <a
href=${`mailto:${announcement.contactEmail}`}>${announcement.contactEmail}</a>
            </div>
            ` : ''}
        </div>
        </div>
        </div>
        </div>
        `;
    }

    // Компонент створення оголошення
    function CreateAnnouncementPage({ onNavigate }) {
        const [formData, setFormData] = React.useState({
            title: '',
            description: '',
            categoryId: '',
            price: '',
            contactPhone: '',
            contactEmail: ''
        });

        const [categories, setCategories] = React.useState([]);
        const [loading, setLoading] = React.useState(false);
        const [error, setError] = React.useState(null);
        const [step, setStep] = React.useState(1); // 1 - основна інформація, 2 -
        фото та кольори
        const [announcementId, setAnnouncementId] = React.useState(null);
        const [photos, setPhotos] = React.useState([]);
        const [colorPalette, setColorPalette] = React.useState([]);
        const { token } = React.useContext(AuthContext);

        // Отримання категорій
        React.useEffect(() => {
            fetch(`${API_URL}/categories`)
                .then(response => {
                    if (!response.ok) {
                        throw new Error('Помилка завантаження категорій');
                    }
                    return response.json();
                })
                .then(data => {
                    setCategories(data);
                    if (data.length > 0) {
                        setFormData(prev => ({
                            ...prev,
                            categoryId: data[0].id
                        }));
                    }
                })
        }, []);
    }

```

```

    })
    .catch(err => {
      setError(err.message);
    });
    }, []));
    // Обробник зміни полів форми
    const handleChange = (e) => {
      const { name, value } = e.target;
      setFormData(prev => ({
        ...prev,
        [name]: value
      }));
    };
    // Обробник вибору кольорової палітри
    const handleColorPaletteChange = (colors) => {
      setColorPalette(colors);
    };
    // Обробник зміни фотографій
    const handlePhotosChange = (newPhotos) => {
      setPhotos(newPhotos);
    };
    // Обробник відправки форми - перший крок
    const handleSubmitFirstStep = async (e) => {
      e.preventDefault();
      setLoading(true);
      setError(null);
      try {
        const response = await fetch(`${API_URL}/announcements`, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
            Authorization: `Bearer ${token}`
          },
          body: JSON.stringify({ ...formData, colorPalette })
        });
        const data = await response.json();
        if (!response.ok) {
          throw new Error(data.message || 'Помилка створення оголошення');
        }
        setAnnouncementId(data.id);
        setStep(2); // Переходимо до кроку завантаження фотографій
      } catch (err) {
        setError(err.message);
      } finally {
        setLoading(false);
      }
    };
    // Обробник завершення створення оголошення
    const handleFinish = () => {
      alert('Оголошення успішно створено');
      onNavigate(null, 'home');
    };
    // Обробник переходу назад до першого кроку
    const handleBackToStep1 = () => {
      setStep(1);
    };
    // Рендеримо перший крок - основна інформація
    const renderStep1 = () => {
      return html`
        <form className="announcement-form"
          onSubmit=${handleSubmitFirstStep}>
          <div className="form-group">
            <label htmlFor="title">Заголовок:</label>
            <input

```

```

type="text"
id="title"
name="title"
value=${formData.title}
onChange=${handleChange}
required

    />
  </div>
  <div className="form-group">
    <label htmlFor="categoryId">Категорія:</label>
    <select

id="categoryId"
name="categoryId"
value=${formData.categoryId}
onChange=${handleChange}
required

    >
    ${categories.map(category => html`
      <option key=${category.id} value=${category.id}>
        ${category.name}
      </option>
    `)}
    </select>
  </div>
  <div className="form-group">
    <label htmlFor="description">Опис:</label>
    <textarea

id="description"
name="description"
value=${formData.description}
onChange=${handleChange}
required
    rows="6"
    ></textarea>
  </div>
  <div className="form-group">
    <label htmlFor="price">Ціна (грн):</label>
    <input

type="number"
id="price"
name="price"
value=${formData.price}
onChange=${handleChange}
required
    min="0"
    />
  </div>
  <div className="form-group">
    <label htmlFor="contactPhone">Телефон:</label>
    <input

type="tel"
id="contactPhone"
name="contactPhone"
value=${formData.contactPhone}
onChange=${handleChange}
required

    />
  </div>
  <div className="form-group">
    <label htmlFor="contactEmail">Email:</label>
    <input

type="email"
id="contactEmail"
name="contactEmail"

```


```

value=${formData.contactEmail}
onChange=${handleChange}
required

        />
    </div>
<${ColorPaletteSelector} onChange=${handleColorPaletteChange}
selectedColors=${colorPalette} />
    <div className="form-actions">
        <button type="button" className="btn btn-secondary"
onClick=${() => onNavigate(null, 'home')}>
Скасувати
    </button>
        <button type="submit" className="btn btn-primary"
disabled=${loading}>
${loading ? 'Створення...' : 'Продовжити'}
    </button>
    </div>
</form>
`;
};
// Рендеримо другий крок - завантаження фотографій
const renderStep2 = () => {
    return html`
        <div className="announcement-photos-step">
<h2>Додавання фотографій</h2>
<p>Додайте фотографії до вашого оголошення (до 8 фотографій):</p>
<${PhotoUploader}
photos=${photos}
announcementId=${announcementId}
onPhotosChange=${handlePhotosChange}
token=${token}
maxPhotos=${8}
        />
        <div className="form-actions">
            <button type="button" className="btn btn-secondary"
onClick=${handleBackToStep1}>
Назад
            </button>
            <button type="button" className="btn btn-primary"
onClick=${handleFinish}>
Завершити створення
            </button>
        </div>
    </div>
    `;
    return html`
        <div className="create-announcement">
<h1>Створення нового оголошення</h1>
        <div className="steps-indicator">
            <div className=${`step ${step} === 1 ? 'active' : ''}`}>1. Основна
інформація</div>
            <div className=${`step ${step} === 2 ? 'active' : ''}`}>2.
Фотографії</div>
        </div>
        ${error ? html`<div className="error">${error}</div>` : ''}
        ${step === 1 ? renderStep1() : renderStep2()}
    </div>
    `;
}

// Компонент редагування оголошення
function EditAnnouncementPage({ onNavigate, params = {} }) {
    const [announcement, setAnnouncement] = React.useState(null);
    const [formData, setFormData] = React.useState({

```

```

        title: '',
        description: '',
        categoryId: '',
        price: '',
        contactPhone: '',
        contactEmail: ''
    });

    const [categories, setCategories] = React.useState([]);
    const [loading, setLoading] = React.useState(true);
    const [updating, setUpdating] = React.useState(false);
    const [error, setError] = React.useState(null);
    const [photos, setPhotos] = React.useState([]);
    const [colorPalette, setColorPalette] = React.useState([]);
    const { token } = React.useContext(AuthContext);
    // Завантаження даних оголошення та категорій
    React.useEffect(() => {
        if (!params.id) {
            setError('ID оголошення не вказано');
            setLoading(false);
            return;
        }
        // Отримуємо категорії
        fetch(`${API_URL}/categories`)
            .then(response => {
                if (!response.ok) {
                    throw new Error('Помилка завантаження категорій');
                }
                return response.json();
            })
            .then(categoriesData => {
                setCategories(categoriesData);
                // Отримуємо дані оголошення
                return fetch(`${API_URL}/announcements/${params.id}`, {
                    headers: {
                        Authorization: `Bearer ${token}`
                    }
                })
                    .then(response => {
                        if (!response.ok) {
                            throw new Error('Помилка завантаження оголошення');
                        }
                        return response.json();
                    })
                    .then(data => {
                        setAnnouncement(data);
                        setPhotos(data.photos || []);
                        setColorPalette(data.colorPalette || []);
                        // Заповнюємо форму даними оголошення
                        setFormData({
                            title: data.title,
                            description: data.description,
                            categoryId: data.categoryId,
                            price: data.price,
                            contactPhone: data.contactPhone,
                            contactEmail: data.contactEmail
                        });
                        setLoading(false);
                    })
                    .catch(err => {
                        setError(err.message);
                        setLoading(false);
                    });
            }, [params.id, token]);
    }, [params.id, token]);

```


```

        // Обробник зміни полів форми
        const handleChange = (e) => {
            const { name, value } = e.target;
            setFormData(prev => ({
                ...prev,
                [name]: value
            }));
        };
        // Обробник вибору кольорової палітри
        const handleColorPaletteChange = (colors) => {
            setColorPalette(colors);
        };
        // Обробник зміни фотографій
        const handlePhotosChange = (newPhotos) => {
            setPhotos(newPhotos);
        };
        // Обробник збереження змін
        const handleSubmit = async (e) => {
            e.preventDefault();
            setUpdating(true);
            setError(null);
            try {
                const response = await
                fetch(`${API_URL}/announcements/${params.id}`, {
                    method: 'PUT',
                    headers: {
                        'Content-Type': 'application/json',
                        Authorization: `Bearer ${token}`
                    },
                    body: JSON.stringify({
                        ...formData,
                        colorPalette
                    })
                });
            } catch (err) {
                const data = await response.json();
                if (!response.ok) {
                    throw new Error(data.message || 'Помилка оновлення оголошення');
                }
            }
            alert('Оголошення успішно оновлено');
            onNavigate(null, 'announcement-details', { id: params.id });
            setError(err.message);
            finally {
                setUpdating(false);
            }
        };
        if (loading) {
            return html`<div className="loading">Завантаження...</div>`;
        }
        if (error) {
            return html`<div className="error">${error}</div>`;
        }
        if (!announcement) {
            return html`<div className="error">Оголошення не знайдено</div>`;
        }
        return html`
        <div className="edit-announcement">
        <h1>Редагування оголошення</h1>
        ${error ? html`<div className="error">${error}</div>` : ''}
        <form className="announcement-form" onSubmit=${handleSubmit}>
            <div className="form-group">
                <label htmlFor="title">Заголовок:</label>
                <input
                    type="text"

```

```

id="title"
name="title"
value=${formData.title}
onChange=${handleChange}
required
    />
</div>
<div className="form-group">
    <label htmlFor="categoryId">Категорія:</label>
    <select
id="categoryId"
name="categoryId"
value=${formData.categoryId}
onChange=${handleChange}
required
    >
${categories.map(category => html`
    <option key=${category.id} value=${category.id}>
    ${category.name}
    </option>
`)}
    </select>
</div>
<div className="form-group">
    <label htmlFor="description">Опис:</label>
    <textarea
id="description"
name="description"
value=${formData.description}
onChange=${handleChange}
required
    rows="6"
    ></textarea>
</div>
<div className="form-group">
    <label htmlFor="price">Ціна (грн):</label>
    <input
type="number"
id="price"
name="price"
value=${formData.price}
onChange=${handleChange}
required
    min="0"
    />
</div>
<div className="form-group">
    <label htmlFor="contactPhone">Телефон:</label>
    <input
type="tel"
id="contactPhone"
name="contactPhone"
value=${formData.contactPhone}
onChange=${handleChange}
required
    />
</div>
<div className="form-group">
    <label htmlFor="contactEmail">Email:</label>
    <input
type="email"
id="contactEmail"
name="contactEmail"
value=${formData.contactEmail}

```


```

onChange=${handleChange}
required
    />
</div>
<${ColorPaletteSelector} onChange=${handleColorPaletteChange}
selectedColors=${colorPalette} />
<${PhotoUploader}
photos=${photos}
announcementId=${params.id}
onPhotosChange=${handlePhotosChange}
token=${token}
maxPhotos=${8}
    />
    <div className="form-actions">
        <button type="button" className="btn btn-secondary"
onClick=${() => onNavigate(null, 'announcement-details', { id: params.id })}>
Скасувати
        </button>
        <button type="submit" className="btn btn-primary"
disabled=${updating}>
${updating ? 'Збереження...' : 'Зберегти зміни'}
        </button>
    </div>
</form>
</div>
`;
}

// Компонент сторінки моїх оголошень
function MyAnnouncementsPage({ onNavigate }) {
    const [announcements, setAnnouncements] = React.useState([]);
    const [loading, setLoading] = React.useState(true);
    const [error, setError] = React.useState(null);
    const { token, user } = React.useContext(AuthContext);

    // Завантаження моїх оголошень
    React.useEffect(() => {
        if (!token) {
            setError('Необхідна авторизація');
            setLoading(false);
            return;
        }

        fetch(`${API_URL}/announcements?userId=${user.id}`, {
            headers: {
                Authorization: `Bearer ${token}`
            }
        })
        .then(response => {
            if (!response.ok) {
                throw new Error('Помилка завантаження оголошень');
            }
            return response.json();
        })
        .then(data => {
            setAnnouncements(data.announcements);
            setLoading(false);
        })
        .catch(err => {
            setError(err.message);
            setLoading(false);
        });

        }, [token, user.id]);

    // Обробник видалення оголошення
    const handleDelete = async (id) => {
        if (!confirm('Ви впевнені, що хочете видалити це оголошення?')) {
            return;
        }
    }
}

```


```

    }
    try {
      const response = await fetch(`${API_URL}/announcements/${id}`, {
method: 'DELETE',
headers: {
  Authorization: `Bearer ${token}`
}
});
    if (!response.ok) {
      const errorData = await response.json();
      throw new Error(errorData.message || 'Помилка видалення оголошення');
    }
    // Оновлюємо список без видаленого оголошення
    setAnnouncements(prev => prev.filter(a => a.id !== id));
    alert('Оголошення успішно видалено');
  } catch (err) {
    alert(`Помилка: ${err.message}`);
  }
};
if (loading) {
  return html`<div className="loading">Завантаження...</div>`;
}
if (error) {
  return html`<div className="error">${error}</div>`;
}
return html`
  <div className="my-announcements">
<h1>Мої оголошення</h1>
    <div className="actions">
      <button className="btn btn-primary" onClick=${(e) => onNavigate(e,
'create-announcement')}>
Створити нове оголошення
      </button>
    </div>
    ${announcements.length === 0 ? html`
      <div className="no-announcements">
        У вас ще немає оголошень
      </div>
    ` : html`
      <div className="announcements-list my-announcements-list">
        ${announcements.map(announcement => html`
          <div className="announcement-card" key=${announcement.id}>
            <div className="announcement-card-content">
              <div className="announcement-image">
                ${announcement.photos && announcement.photos.length > 0
? html`<img src=${announcement.photos[0].url}
alt="${announcement.title}" />`
: html`<div className="no-photo">Немає фото</div>`
                }
              </div>
              <div className="announcement-info">
                <h3 className="announcement-title">${announcement.title}</h3>
                <div className="announcement-price">
Ціна: ${announcement.price} грн
                </div>
                <div className="announcement-date">
Опубліковано: ${new
Date(announcement.createdAt).toLocaleDateString('uk-UA')}
                </div>
                ${announcement.colorPalette && announcement.colorPalette.length
> 0 ? html`
                  <div className="color-preview">
                    ${announcement.colorPalette.map(color => html`

```


```

        <div className="color-dot" key=${color.hex} style=${{
backgroundColor: color.hex }} title=${color.name}></div>
      `)}
    </div>
    ` : null}
  </div>
</div>
<div className="announcement-actions">
  <button
    className="btn btn-primary"
    onClick=${(e) => onNavigate(e, 'announcement-details', { id:
announcement.id })}
  >
    Переглянути
  </button>
  <button
    className="btn btn-secondary"
    onClick=${(e) => onNavigate(e, 'edit-announcement', { id:
announcement.id })}
  >
    Редагувати
  </button>
  <button
    className="btn btn-danger"
    onClick=${() => handleDelete(announcement.id)}
  >
    Видалити
  </button>
</div>
</div>
`)}
</div>
`
</div>
`
;
}

// Сторінка запиту на преміум
function PremiumRequestPage({ onNavigate }) {
  const [loading, setLoading] = React.useState(false);
  const [error, setError] = React.useState(null);
  const [success, setSuccess] = React.useState(false);
  const { token, user } = React.useContext(AuthContext);
  // Обробник запиту на преміум
  const handleRequest = async () => {
    setLoading(true);
    setError(null);
    setSuccess(false);
    try {
      const response = await fetch(`${API_URL}/premium-request`, {
method: 'POST',
headers: {
Authorization: `Bearer ${token}`
}
});
    } catch (err) {
      const data = await response.json();
      if (!response.ok) {
        throw new Error(data.message || 'Помилка відправки запиту');
      }
    }
    setSuccess(true);
  } catch (err) {
    setError(err.message);
  } finally {
    setLoading(false);
  }
}

```

```

    };
    // Якщо користувач вже має преміум-статус
    if (user.isPremium) {
        return html`
            <div className="premium-request">
                <h1>Преміум-статус</h1>
                <div className="success-message">
У вас вже є преміум-статус!
                </div>
                <div className="premium-benefits">
<h2>Ваші переваги:</h2>
                <ul>
<li>Необмежена кількість оголошень</li>
<li>Ваші оголошення показуються першими у списку</li>
<li>Пріоритетна підтримка</li>
                </ul>
            </div>
        </div>
        `;
    }
    return html`
        <div className="premium-request">
<h1>Отримання преміум-статусу</h1>
        ${error ? html`<div className="error">${error}</div>` : ''}
        ${success ? html`
            <div className="success-message">
                Запит на отримання преміум-статусу успішно відправлено. Очікуйте на
                відповідь від менеджера.
            </div>
            ` : ''}
            <div className="premium-benefits">
<h2>Переваги преміум-статусу:</h2>
            <ul>
<li>Необмежена кількість оголошень</li>
<li>Ваші оголошення показуються першими у списку</li>
<li>Пріоритетна підтримка</li>
            </ul>
        </div>
        ${!success ? html`
            <div className="request-actions">
                <button
                    className="btn btn-primary"
                    onClick=${handleRequest}
                    disabled=${loading}
                >
                    ${loading ? 'Відправка запиту...' : 'Відправити запит на преміум-статус'}
                </button>
            </div>
            ` : ''}
        </div>
        `;
    }
}

// Сторінка чатів
function ChatsPage({ onNavigate }) {
    const [chats, setChats] = React.useState([]);
    const [loading, setLoading] = React.useState(true);
    const [error, setError] = React.useState(null);
    const { token, user } = React.useContext(AuthContext);
    // Завантаження списку чатів
    React.useEffect(() => {
        if (!token) {
            setError('Необхідна авторизація');
            setLoading(false);
            return;
        }
    });
}

```

```

    }
    fetch(`${API_URL}/chats`, {
      headers: {
        Authorization: `Bearer ${token}`
      }
    })
      .then(response => {
        if (!response.ok) {
          throw new Error('Помилка завантаження чатів');
        }
        return response.json();
      })
      .then(data => {
        setChats(data);
        setLoading(false);
      })
      .catch(err => {
        setError(err.message);
        setLoading(false);
      });

    }, [token]);

    if (loading) {
      return html`<div className="loading">Завантаження...</div>`;
    }
    if (error) {
      return html`<div className="error">${error}</div>`;
    }
    return html`
    <div className="chats-page">
    <h1>${user.role === 'manager' ? 'Запити користувачів' : 'Мої повідомлення'}</h1>
    ${chats.length === 0 ? html`
      <div className="no-chats">
        ${user.role === 'manager' ? 'Немає активних запитів від
        користувачів' : 'У вас немає активних чатів'}
      </div>
    ` : html`
      <div className="chats-list">
        ${chats.map(chat => html`
          <div className="chat-card" key=${chat.id} onClick=${(e) =>
onNavigate(e, 'chat', { id: chat.id })}>
            <div className="chat-info">
              <h3 className="chat-type">
                ${chat.type === 'premium-request' ? 'Запит на преміум-статус' :
'Чат з менеджером'}
              </h3>
              <div className="chat-status">
                Статус: ${chat.status === 'pending' ? 'Очікує відповіді' :
'Відповідь отримано'}
              </div>
              <div className="chat-date">
                Створено: ${new Date(chat.createdAt).toLocaleDateString('uk-
UA')}}
              </div>
            </div>
          </div>
          ${chat.lastMessage ? html`
            <div className="chat-last-message">
              <p>${chat.lastMessage.text.length > 50
? `${chat.lastMessage.text.substring(0, 50)}...`
: chat.lastMessage.text}</p>
              <span className="message-time">
                ${new
Date(chat.lastMessage.createdAt).toLocaleTimeString('uk-UA')}}
              </span>
            </div>
          ` : ''}
        `)}
      </div>
    `}
  `;

```

```

        ` : ''}
      </div>
    `)}
  </div>
`}
</div>
`;
}

// Сторінка чату
function ChatPage({ onNavigate, params = {} }) {
  const [chat, setChat] = React.useState(null);
  const [messages, setMessages] = React.useState([]);
  const [loading, setLoading] = React.useState(true);
  const [error, setError] = React.useState(null);
  const [newMessage, setNewMessage] = React.useState('');
  const [sendingMessage, setSendingMessage] = React.useState(false);
  const [users, setUsers] = React.useState({});
  const { token, user } = React.useContext(AuthContext);
  const messagesEndRef = React.useRef(null);

  // Отримання інформації про чат та повідомлення
  React.useEffect(() => {
    if (!token || !params.id) {
      setError('Необхідна авторизація або ID чату');
      setLoading(false);
      return;
    }

    // Отримуємо повідомлення чату
    fetch(`${API_URL}/chats/${params.id}/messages`, {
      headers: {
        Authorization: `Bearer ${token}`
      }
    })
      .then(response => {
        if (!response.ok) {
          throw new Error('Помилка завантаження повідомлень');
        }
        return response.json();
      })
      .then(data => {
        setMessages(data);
        return fetch(`${API_URL}/users`, {
          headers: {
            Authorization: `Bearer ${token}`
          }
        });
      })
      .then(response => {
        if (!response.ok) {
          throw new Error('Помилка завантаження користувачів');
        }
        return response.json();
      })
      .then(usersData => {
        // Створюємо об'єкт з id користувача в якості ключа
        const usersMap = {};
        usersData.forEach(user => {
          usersMap[user.id] = user;
        });
        setUsers(usersMap);
        setLoading(false);
      })
      .catch(err => {
        setError(err.message);
        setLoading(false);
      });
  }, [token, params.id]);
}

```

```

    });
    }, [token, params.id]);
    // Прокрутка вниз при отриманні нових повідомлень
    React.useEffect(() => {
      if (messagesEndRef.current) {
        messagesEndRef.current.scrollToView({ behavior: 'smooth' });
      }
    }, [messages]);
    // Обробник відправки повідомлення
    const handleSendMessage = async (e) => {
      e.preventDefault();
      if (!newMessage.trim()) {
        return;
      }
      setSendingMessage(true);
      try {
        const response = await
        fetch(`${API_URL}/chats/${params.id}/messages`, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
            Authorization: `Bearer ${token}`
          },
          body: JSON.stringify({ text: newMessage })
        });
        const data = await response.json();
        if (!response.ok) {
          throw new Error(data.message || 'Помилка відправки повідомлення');
        }
        setMessages(prev => [...prev, data]);
        setNewMessage('');
      } catch (err) {
        alert(`Помилка: ${err.message}`);
      } finally {
        setSendingMessage(false);
      }
    };
    // Обробник зміни статусу преміум для користувача
    const handlePremiumChange = async (userId, isPremium) => {
      try {
        const response = await fetch(`${API_URL}/users/${userId}/premium`, {
          method: 'POST',
          headers: {
            'Content-Type': 'application/json',
            Authorization: `Bearer ${token}`
          },
          body: JSON.stringify({ isPremium })
        });
        const data = await response.json();
        if (!response.ok) {
          throw new Error(data.message || 'Помилка зміни статусу преміум');
        }
        // Оновлюємо статус користувача в масиві користувачів
        setUsers(prev => ({
          ...prev,
          [userId]: {
            ...prev[userId],
            isPremium
          }
        }));
      }
    };
    // Відправляємо повідомлення про зміну статусу
    const response2 = await fetch(`${API_URL}/chats/${params.id}/messages`,
    {
      method: 'POST',

```

```

headers: {
  'Content-Type': 'application/json',
  Authorization: `Bearer ${token}`
},
body: JSON.stringify({
  text: isPremium
    ? 'Ваш запит на преміум-статус схвалено. Тепер ви маєте преміум-статус!'
    : 'Ваш запит на преміум-статус відхилено.'
}))
});
const messageData = await response2.json();
if (!response2.ok) {
  throw new Error(messageData.message || 'Помилка відправки повідомлення');
}
setMessages(prev => [...prev, messageData]);
} catch (err) {
  alert(`Помилка: ${err.message}`);
}
};
if (loading) {
  return html`<div className="loading">Завантаження...</div>`;
}
if (error) {
  return html`<div className="error">${error}</div>`;
}
// Отримуємо ID клієнта (для преміум запитів)
const clientId = messages.length > 0 ? messages[0].senderId : null;
const clientUser = clientId ? users[clientId] : null;
const isPremiumRequest = messages.length > 0 && messages[0].text ===
'Прошу надати мені преміум-статус';
return html`
  <div className="chat-page">
    <div className="chat-header">
      <h1>
        ${isPremiumRequest ? 'Запит на преміум-статус' : 'Чат з менеджером'}
      </h1>
      <button className="btn btn-secondary" onClick=${(e) => onNavigate(e,
        'chats')}>
        Назад до списку
      </button>
    </div>
    ${user.role === 'manager' && clientUser && isPremiumRequest ? html`
      <div className="premium-request-actions">
        <div className="client-info">
          <p>Користувач: ${clientUser.username}</p>
          <p>Email: ${clientUser.email}</p>
          <p>Статус: ${clientUser.isPremium ? 'Преміум' : 'Звичайний'}</p>
        </div>
        <div className="premium-actions">
          ${!clientUser.isPremium ? html`
            <button
              className="btn btn-primary"
              onClick=${() => handlePremiumChange(clientUser.id, true)}
            >
              Надати преміум-статус
            </button>
          ` : html`
            <button
              className="btn btn-danger"
              onClick=${() => handlePremiumChange(clientUser.id, false)}
            >
              Скасувати преміум-статус
            </button>
          `}
        </div>
      </div>
    `}
  </div>
`

```


```

        </button>
      `}
    </div>
  </div>
  ` : ''}
  <div className="chat-messages">
    ${messages.map(message => {
      const isCurrentUser = message.senderId === user.id;
      const sender = users[message.senderId] || { username: 'Невідомий користувач' };
      return html`
        <div
          key=${message.id}
          className=${`message ${isCurrentUser ? 'current-user' : 'other-user'}`}
        >
          <div className="message-bubble">
            <div className="message-sender">
              ${sender.username}
            </div>
            <div className="message-text">${message.text}</div>
            <div className="message-time">
              ${new Date(message.createdAt).toLocaleTimeString('uk-UA')}
            </div>
          </div>
        </div>
      `;
    })}
    <div ref=${messagesEndRef} />
  </div>
  <form className="chat-form" onSubmit=${handleSendMessage}>
    <input
      type="text"
      value=${newMessage}
      onChange=${(e) => setNewMessage(e.target.value)}
      placeholder="Введіть повідомлення..."
      disabled=${sendingMessage}
    />
    <button
      type="submit"
      className="btn btn-primary"
      disabled=${sendingMessage || !newMessage.trim()}
    >
      ${sendingMessage ? 'Відправка...' : 'Відправити'}
    </button>
  </form>
</div>
`;
}

// Сторінка управління користувачами (для менеджера)
function AdminUsersPage({ onNavigate }) {
  const [users, setUsers] = React.useState([]);
  const [loading, setLoading] = React.useState(true);
  const [error, setError] = React.useState(null);
  const { token, user } = React.useContext(AuthContext);
  // Завантаження списку користувачів
  React.useEffect(() => {
    if (!token || user.role !== 'manager') {
      setError('Доступ заборонено');
      setLoading(false);
      return;
    }
  }, [token, user]);

  fetch(`${API_URL}/users`, {
    headers: {
      Authorization: `Bearer ${token}`
    }
  })

```

```

    }
  })
  .then(response => {
    if (!response.ok) {
      throw new Error('Помилка завантаження користувачів');
    }
    return response.json();
  })
  .then(data => {
    setUsers(data);
    setLoading(false);
  })
  .catch(err => {
    setError(err.message);
    setLoading(false);
  });

  }, [token, user.role]);

  // Обробник видалення користувача
  const handleDeleteUser = async (userId) => {
    if (!confirm('Ви впевнені, що хочете видалити цього користувача? Всі його оголошення також будуть видалені.')) {
      return;
    }
    try {
      const response = await fetch(`${API_URL}/users/${userId}`, {
method: 'DELETE',
headers: {
  Authorization: `Bearer ${token}`
}
});
      if (!response.ok) {
        const errorData = await response.json();
        throw new Error(errorData.message || 'Помилка видалення користувача');
      }
    } catch (err) {
      alert(`Помилка: ${err.message}`);
    }
  };

  // Оновлюємо список користувачів без видаленого
  setUsers(prev => prev.filter(u => u.id !== userId));
  alert('Користувача успішно видалено');

  // Обробник зміни преміум-статусу
  const handleTogglePremium = async (userId, isPremium) => {
    try {
      const response = await fetch(`${API_URL}/users/${userId}/premium`, {
method: 'POST',
headers: {
  'Content-Type': 'application/json',
  Authorization: `Bearer ${token}`
},
body: JSON.stringify({ isPremium })
});
      const data = await response.json();
      if (!response.ok) {
        throw new Error(data.message || 'Помилка зміни статусу преміум');
      }
    } catch (err) {
      alert(`Помилка: ${err.message}`);
    }
  };

  // Оновлюємо список користувачів із зміненим статусом
  setUsers(prev => prev.map(u =>
    u.id === userId ? { ...u, isPremium } : u
  ));
  alert(isPremium
    ? 'Преміум-статус успішно надано'
    : 'Преміум-статус успішно скасовано'
  );

```

```

);
    } catch (err) {
alert(`Помилка: ${err.message}`);
    }
};
if (loading) {
return html`<div className="loading">Завантаження...</div>`;
}
if (error) {
return html`<div className="error">${error}</div>`;
}
return html`
<div className="admin-users-page">
<h1>Управління користувачами</h1>
<div className="users-list">
<table className="users-table">
<thead>
<tr>
<th>ID</th>
<th>Ім'я користувача</th>
<th>Email</th>
<th>Роль</th>
<th>Статус</th>
<th>Дата реєстрації</th>
<th>Дії</th>
</tr>
</thead>
<tbody>
${users.map(u => html`
<tr key=${u.id}>
<td>${u.id}</td>
<td>${u.username}</td>
<td>${u.email}</td>
<td>${u.role === 'client' ? 'Клієнт' : 'Менеджер'}</td>
<td>${u.isPremium ? 'Преміум' : 'Звичайний'}</td>
<td>${new Date(u.createdAt).toLocaleDateString('uk-UA')}</td>
<td className="user-actions">
${u.id !== user.id ? html`
<button
className="btn btn-sm btn-danger"
onClick=${() => handleDeleteUser(u.id)}
>
Видалити
</button>
${u.role === 'client' ? html`
<button
className=${`${'btn btn-sm' } ${u.isPremium ? 'btn-warning' :
'btn-success'}`}
onClick=${() => handleTogglePremium(u.id, !u.isPremium)}
>
${u.isPremium ? 'Скасувати преміум' : 'Надати преміум'}
</button>
` : ''}
` : ''}
</td>
</tr>
`)}
</tbody>
</table>
</div>
</div>
`;
}
}

```

// Рендеринг застосунку в DOM

```
ReactDOM.render(React.createElement(App),
document.getElementById('root'));
```

index.html

```
<!DOCTYPE html>
<html lang="uk">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Дошка оголошень</title>
  <link rel="stylesheet" href="styles.css">
  <!-- Спочатку завантажуюмо всі бібліотеки -->
  <script src="https://unpkg.com/react@17/umd/react.development.js"></script>
  <script src="https://unpkg.com/react-dom@17/umd/react-
dom.development.js"></script>
  <script src="https://unpkg.com/htm@3.1.1/dist/htm.js"></script>
</head>
<body>
  <div id="root"></div>
  <!-- Завантажуємо скрипт застосунку після всіх бібліотек -->
  <script src="app.js"></script>
</body>
</html>
```

styles.css

```
/* Базові стилі */
* {
  box-sizing: border-box;
  margin: 0;
  padding: 0;
}
html {
  font-size: 16px;
}
body {
  font-family: 'Roboto', Arial, sans-serif;
  line-height: 1.6;
  color: #333;
  background-color: #f4f4f4;
}
a {
  text-decoration: none;
  color: #3498db;
}
a:hover {
  text-decoration: underline;
}
h1, h2, h3, h4, h5, h6 {
  margin-bottom: 1rem;
  color: #222;
  font-weight: 500;
}
h1 {
  font-size: 2rem;
  margin-bottom: 1.5rem;
}
h2 {
  font-size: 1.5rem;
  margin-bottom: 1.2rem;
```

```

}
ul, ol {
    margin-left: 1.5rem;
    margin-bottom: 1rem;
}
p {
    margin-bottom: 1rem;
}
/* Структура застосунку */
.app {
    display: flex;
    flex-direction: column;
    min-height: 100vh;
}
.main-content {
    flex: 1;
    padding: 20px;
    max-width: 1200px;
    margin: 0 auto;
    width: 100%;
}
/* Хедер */
.header {
    background-color: #2c3e50;
    color: white;
    padding: 1rem;
    display: flex;
    justify-content: space-between;
    align-items: center;
    box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
}
.logo a {
    color: white;
    text-decoration: none;
    font-size: 1.5rem;
    font-weight: bold;
}
    .navigation ul {
display: flex;
list-style: none;
margin: 0;
    }
        .navigation li {
margin-right: 1rem;
        }
            .navigation a {
color: white;
text-decoration: none;
padding: 0.5rem;
border-radius: 4px;
transition: background-color 0.2s;
            }
                .navigation a:hover {
background-color: rgba(255, 255, 255, 0.1);
text-decoration: none;
                }
    .auth-links {
display: flex;
gap: 1rem;
align-items: center;
    }
        .auth-links a {
color: white;
text-decoration: none;

```


```

padding: 0.4rem 0.8rem;
border-radius: 4px;
transition: background-color 0.2s;
}

.auth-links a:hover {
background-color: rgba(255, 255, 255, 0.1);
text-decoration: none;
}

.user-panel {
display: flex;
align-items: center;
gap: 1rem;
}

.username {
color: white;
font-weight: 500;
}

.logout-btn {
background: none;
border: 1px solid white;
color: white;
padding: 0.25rem 0.75rem;
border-radius: 4px;
cursor: pointer;
transition: all 0.2s;
}

.logout-btn:hover {
background-color: rgba(255, 255, 255, 0.1);
}

/* Футер */
.footer {
background-color: #2c3e50;
color: white;
padding: 1.5rem;
text-align: center;
margin-top: 2rem;
}

.footer-content p {
margin: 0;
font-size: 0.9rem;
}

/* Кнопки */
.btn {
display: inline-block;
padding: 0.5rem 1rem;
background-color: #3498db;
color: white;
border: none;
border-radius: 4px;
cursor: pointer;
font-size: 1rem;
text-decoration: none;
transition: background-color 0.2s, transform 0.1s;
}

.btn:hover {
background-color: #2980b9;
text-decoration: none;
}

.btn:active {
transform: translateY(1px);
}

.btn:disabled {
background-color: #ccc;
cursor: not-allowed;
}

```


```

}
.btn-primary {
    background-color: #3498db;
}
.btn-secondary {
    background-color: #7f8c8d;
}
.btn-danger {
    background-color: #e74c3c;
}
    .btn-danger:hover {
        background-color: #c0392b;
    }
.btn-warning {
    background-color: #f39c12;
}
    .btn-warning:hover {
        background-color: #d35400;
    }
.btn-success {
    background-color: #2ecc71;
}
    .btn-success:hover {
        background-color: #27ae60;
    }
.btn-sm {
    padding: 0.25rem 0.5rem;
    font-size: 0.875rem;
}
/* ФОРМИ */
.form-group {
    margin-bottom: 1.5rem;
}
    .form-group label {
display: block;
margin-bottom: 0.5rem;
font-weight: 500;
    }
        .form-group input,
        .form-group select,
        .form-group textarea {
width: 100%;
padding: 0.75rem;
border: 1px solid #ddd;
border-radius: 4px;
font-size: 1rem;
font-family: inherit;
transition: border-color 0.2s;
        }
            .form-group input:focus,
            .form-group select:focus,
            .form-group textarea:focus {
border-color: #3498db;
outline: none;
box-shadow: 0 0 0 3px rgba(52, 152, 219, 0.2);
            }
.form-actions {
    display: flex;
    gap: 1rem;
    margin-top: 1.5rem;
    flex-wrap: wrap;
}
/* Сторінки авторизації */
.auth-page {

```

```

    max-width: 500px;
    margin: 2rem auto;
    padding: 2rem;
    background-color: white;
    border-radius: 8px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}
.auth-form {
    margin-top: 1rem;
}
.auth-link {
    margin-top: 1.5rem;
    text-align: center;
}
/* ФІЛЬТРИ */
.filters {
    background-color: white;
    padding: 1.5rem;
    border-radius: 8px;
    margin-bottom: 1.5rem;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}
.filter-form {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
    gap: 1rem;
}
/* СПИСОК ОГЛОШЕНЬ */
.announcements-list {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(300px, 1fr));
    gap: 1.5rem;
    margin-top: 1.5rem;
}
.announcement-card {
    background-color: white;
    border-radius: 8px;
    box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    overflow: hidden;
    transition: transform 0.2s, box-shadow 0.2s;
    display: flex;
    flex-direction: column;
}
.announcement-card:hover {
    transform: translateY(-5px);
    box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
}
.announcement-photos {
    height: 200px;
    overflow: hidden;
    position: relative;
}
.announcement-photos img {
    width: 100%;
    height: 100%;
    object-fit: cover;
    transition: transform 0.3s;
}
.announcement-card:hover .announcement-photos img {
    transform: scale(1.05);
}
.no-photo {
    display: flex;
    align-items: center;
}

```


```

        justify-content: center;
        height: 100%;
        background-color: #f0f0f0;
        color: #888;
        font-size: 0.9rem;
    }
    .announcement-content {
        padding: 1.5rem;
        display: flex;
        flex-direction: column;
        flex-grow: 1;
    }
    .announcement-title {
        font-size: 1.2rem;
        margin-bottom: 0.8rem;
        display: flex;
        justify-content: space-between;
        align-items: flex-start;
    }
    .premium-badge {
        color: gold;
        font-size: 1.25rem;
        margin-left: 0.5rem;
    }
    .announcement-category,
        .announcement-price,
        .announcement-date {
        font-size: 0.875rem;
        color: #666;
        margin-bottom: 0.25rem;
    }
    .announcement-description {
        margin: 1rem 0;
        flex-grow: 1;
        color: #555;
    }
    .color-preview {
        display: flex;
        margin: 0.5rem 0;
    }
    .color-dot {
        width: 20px;
        height: 20px;
        border-radius: 50%;
        border: 1px solid #ddd;
        display: inline-block;
        margin-right: 5px;
    }
    .announcement-contact {
        margin-bottom: 0.5rem;
        font-size: 0.875rem;
    }
    .announcement-actions {
        display: flex;
        gap: 0.5rem;
        margin-top: 1rem;
        flex-wrap: wrap;
    }
    /* Детальна сторінка оголошення */
    .announcement-details {
        background-color: white;
        padding: 2rem;
        border-radius: 8px;
        box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
    }

```


```

}
.announcement-details-header {
    display: flex;
    justify-content: space-between;
    align-items: flex-start;
    margin-bottom: 2rem;
    flex-wrap: wrap;
    gap: 1rem;
}
.announcement-details-content {
    display: grid;
    grid-template-columns: 1fr 1fr;
    gap: 2rem;
}
.announcement-details-gallery {
    margin-bottom: 1.5rem;
}
.announcement-details-info {
    display: flex;
    flex-direction: column;
    gap: 1.5rem;
}
.announcement-info-block {
    display: flex;
    flex-direction: column;
    gap: 0.5rem;
}
.announcement-info {
    display: flex;
    flex-wrap: wrap;
    gap: 2rem;
    margin-bottom: 1rem;
}
.announcement-contacts {
    margin-top: auto;
    padding-top: 1rem;
    border-top: 1px solid #eee;
}
/* Фото карусель */
.photo-carousel {
    max-width: 100%;
    margin-bottom: 2rem;
}
.main-photo {
    position: relative;
    border-radius: 8px;
    overflow: hidden;
    margin-bottom: 0.5rem;
    height: 400px;
    background-color: #f0f0f0;
}
    .main-photo img {
width: 100%;
height: 100%;
object-fit: contain;
display: block;
    }
        .carousel-btn {
position: absolute;
top: 50%;
transform: translateY(-50%);
background-color: rgba(0, 0, 0, 0.5);
color: white;
border: none;

```

```

width: 40px;
height: 40px;
border-radius: 50%;
font-size: 20px;
display: flex;
align-items: center;
justify-content: center;
cursor: pointer;
opacity: 0.7;
transition: opacity 0.2s;
}
.carousel-btn:hover {
opacity: 1;
}
.carousel-btn.prev {
left: 10px;
}
.carousel-btn.next {
right: 10px;
}
.photo-counter {
position: absolute;
bottom: 10px;
right: 10px;
background-color: rgba(0, 0, 0, 0.5);
color: white;
padding: 5px 10px;
border-radius: 15px;
font-size: 12px;
}
.thumbnails {
display: flex;
gap: 8px;
overflow-x: auto;
padding-bottom: 5px;
}
.thumbnail {
width: 80px;
height: 60px;
border-radius: 4px;
overflow: hidden;
cursor: pointer;
opacity: 0.7;
transition: opacity 0.2s;
}
.thumbnail.active {
opacity: 1;
border: 2px solid #3498db;
}
.thumbnail img {
width: 100%;
height: 100%;
object-fit: cover;
}
.photo-placeholder {
display: flex;
align-items: center;
justify-content: center;
height: 300px;
background-color: #f0f0f0;
color: #888;
border-radius: 8px;
}
/* Завантажувач фото */

```

```

.photo-uploader {
    margin: 2rem 0;
}
.photos-grid {
    display: grid;
    grid-template-columns: repeat(auto-fill, minmax(150px, 1fr));
    gap: 10px;
    margin: 1rem 0;
}
.photo-item {
    position: relative;
    height: 150px;
    border-radius: 6px;
    overflow: hidden;
    border: 1px solid #ddd;
}
    .photo-item img {
width: 100%;
height: 100%;
object-fit: cover;
    }
        .photo-delete-btn {
position: absolute;
top: 5px;
right: 5px;
width: 24px;
height: 24px;
border-radius: 50%;
background-color: rgba(231, 76, 60, 0.8);
color: white;
border: none;
font-size: 16px;
display: flex;
align-items: center;
justify-content: center;
cursor: pointer;
        }
.photo-upload-placeholder {
    height: 150px;
    border: 2px dashed #ccc;
    border-radius: 6px;
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    cursor: pointer;
    color: #888;
}
    .photo-upload-placeholder:hover {
border-color: #3498db;
color: #3498db;
    }
.plus-icon {
    font-size: 24px;
    margin-bottom: 5px;
}
.upload-hint {
    font-size: 0.8rem;
    color: #777;
    margin-top: 0.5rem;
}
/* Палітра кольорів */
.color-palette-selector {
    margin: 2rem 0;
}

```


```

}
.palettes-grid {
  display: grid;
  grid-template-columns: repeat(auto-fill, minmax(200px, 1fr));
  gap: 15px;
  margin: 1rem 0;
}
.palette-card {
  border: 1px solid #ddd;
  border-radius: 8px;
  padding: 10px;
  cursor: pointer;
  transition: transform 0.2s, box-shadow 0.2s;
}
.palette-card:hover, .palette-card.active {
  transform: translateY(-2px);
  box-shadow: 0 4px 8px rgba(0, 0, 0, 0.1);
}
.palette-card.active {
  border-color: #3498db;
}
.palette-title {
  margin-bottom: 10px;
  font-weight: bold;
}
.palette-colors {
  display: flex;
  gap: 5px;
}
.color-sample {
  width: 30px;
  height: 30px;
  border-radius: 50%;
  border: 1px solid #ddd;
}
.selected-colors {
  margin-top: 1.5rem;
  padding: 15px;
  background-color: #f8f9fa;
  border-radius: 8px;
}
.color-chips {
  display: flex;
  flex-wrap: wrap;
  gap: 10px;
  margin-top: 10px;
}
.color-chip {
  display: flex;
  align-items: center;
  gap: 10px;
  padding: 8px 12px;
  background-color: white;
  border-radius: 4px;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.1);
}
.color-preview {
  width: 30px;
  height: 30px;
  border-radius: 4px;
  border: 1px solid #ddd;
}
.color-info {
  font-size: 0.9rem;

```


```

}
.color-name {
    font-weight: bold;
}
.color-hex {
    font-size: 0.8rem;
    color: #666;
}
.color-display {
    margin: 1.5rem 0;
    padding: 15px;
    background-color: #f8f9fa;
    border-radius: 8px;
}
/* Сторінка моїх оголошень */
.my-announcements-list {
    display: flex;
    flex-direction: column;
    gap: 1rem;
}
    .my-announcements-list .announcement-card {
        display: flex;
        flex-direction: column;
    }
.announcement-card-content {
    display: flex;
    gap: 1rem;
    padding: 1.5rem;
}
.announcement-image {
    width: 150px;
    height: 150px;
    overflow: hidden;
    border-radius: 8px;
    flex-shrink: 0;
}
    .announcement-image img {
width: 100%;
height: 100%;
object-fit: cover;
    }
        /* Сторінка преміум-запиту */
        .premium-request {
            background-color: white;
            padding: 2rem;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
            max-width: 800px;
            margin: 0 auto;
        }
.premium-benefits {
    background-color: #f8f9fa;
    padding: 1.5rem;
    border-radius: 8px;
    margin: 1.5rem 0;
    border-left: 4px solid #3498db;
}
    .premium-benefits ul {
margin-left: 1.5rem;
    }
        .request-actions {
            margin-top: 2rem;
            text-align: center;
        }
}

```


```

/* Сторінка чатів */
.chats-list {
  display: flex;
  flex-direction: column;
  gap: 1rem;
  margin-top: 1rem;
}
.chat-card {
  background-color: white;
  padding: 1.5rem;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  cursor: pointer;
  transition: transform 0.2s;
}
.chat-card:hover {
  transform: translateY(-2px);
}
.chat-info {
  margin-bottom: 0.8rem;
}
.chat-type {
  margin-bottom: 0.5rem;
}
.chat-status, .chat-date {
  font-size: 0.875rem;
  color: #666;
  margin-bottom: 0.25rem;
}
.chat-last-message {
  font-size: 0.875rem;
  color: #666;
  display: flex;
  justify-content: space-between;
  align-items: flex-end;
  border-top: 1px solid #eee;
  padding-top: 0.8rem;
  margin-top: 0.8rem;
}
/* Сторінка чату */
.chat-page {
  display: flex;
  flex-direction: column;
  height: 80vh;
  background-color: white;
  border-radius: 8px;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
  padding: 1.5rem;
}
.chat-header {
  display: flex;
  justify-content: space-between;
  align-items: center;
  margin-bottom: 1.5rem;
  padding-bottom: 1rem;
  border-bottom: 1px solid #eee;
}
.premium-request-actions {
  display: flex;
  justify-content: space-between;
  align-items: center;
  background-color: #f8f9fa;
  padding: 1.5rem;
  border-radius: 8px;
}

```


```

        margin-bottom: 1.5rem;
        flex-wrap: wrap;
        gap: 1rem;
    }
    .client-info p {
        margin-bottom: 0.5rem;
    }
        .chat-messages {
            flex-grow: 1;
            overflow-y: auto;
            padding: 1rem;
            display: flex;
            flex-direction: column;
            gap: 1rem;
            margin-bottom: 1rem;
        }
    .message {
        display: flex;
        margin-bottom: 1rem;
    }
        .message.current-user {
            justify-content: flex-end;
        }
    .message-bubble {
        max-width: 70%;
        padding: 1rem;
        border-radius: 1rem;
        position: relative;
    }
    .current-user .message-bubble {
        background-color: #3498db;
        color: white;
        border-bottom-right-radius: 0;
    }
    .other-user .message-bubble {
        background-color: #f1f1f1;
        color: #333;
        border-bottom-left-radius: 0;
    }
    .message-sender {
        font-weight: bold;
        margin-bottom: 0.5rem;
        font-size: 0.875rem;
    }
    .current-user .message-sender {
        text-align: right;
    }
    .message-time {
        font-size: 0.75rem;
        text-align: right;
        margin-top: 0.5rem;
        color: rgba(255, 255, 255, 0.8);
    }
    .other-user .message-time {
        color: #999;
    }
    .chat-form {
        display: flex;
        gap: 1rem;
        padding: 1rem;
        border-top: 1px solid #eee;
    }
        .chat-form input {
            flex-grow: 1;

```


```
padding: 0.75rem 1rem;
border: 1px solid #ddd;
border-radius: 2rem;
font-size: 1rem;
}

.chat-form input:focus {
  outline: none;
  border-color: #3498db;
}

/* Адмін-панель користувачів */
.users-table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 1rem;
  background-color: white;
  border-radius: 8px;
  overflow: hidden;
  box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
}

.users-table th,
.users-table td {
  padding: 1rem;
  border: 1px solid #ddd;
  text-align: left;
}

.users-table th {
  background-color: #f2f2f2;
  font-weight: bold;
}

.users-table tr:nth-child(even) {
  background-color: #f9f9f9;
}

.users-table tr:hover {
  background-color: #f0f0f0;
}

.user-actions {
  display: flex;
  gap: 0.5rem;
  flex-wrap: wrap;
}

/* Кроки створення оголошення */
.steps-indicator {
  display: flex;
  margin-bottom: 2rem;
  border-bottom: 1px solid #ddd;
}

.step {
  padding: 1rem 1.5rem;
  margin-right: 1rem;
  opacity: 0.6;
  position: relative;
}

.step.active {
  opacity: 1;
  font-weight: bold;
  color: #3498db;
}

.step.active:after {
  content: ' ';
  position: absolute;
  bottom: -1px;
  left: 0;
  width: 100%;
  height: 3px;
}
```

```

        background-color: #3498db;
    }
    /* Загальні стилі для станів */
    .loading,
        .error,
        .success-message,
        .no-announcements,
        .no-chats {
        text-align: center;
        padding: 2rem;
        background-color: white;
        border-radius: 8px;
        box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
        margin: 1rem 0;
    }
    .loading {
        color: #3498db;
        display: flex;
        flex-direction: column;
        align-items: center;
        justify-content: center;
    }

    .loading:before {
        content: '';
        width: 30px;
        height: 30px;
        border: 3px solid #f3f3f3;
        border-top: 3px solid #3498db;
        border-radius: 50%;
        animation: spin 1s linear infinite;
        margin-bottom: 1rem;
    }
    @keyframes spin {
        0% {
            transform: rotate(0deg);
        }
        100% {
            transform: rotate(360deg);
        }
    }
    .error {
        color: #e74c3c;
        border-left: 4px solid #e74c3c;
    }
    .success-message {
        color: #2ecc71;
        border-left: 4px solid #2ecc71;
    }
    /* Адаптивний дизайн */
    @media (max-width: 991px) {
        .announcement-details-content {
            grid-template-columns: 1fr;
        }
    }
    @media (max-width: 768px) {
        .header {
            flex-direction: column;
            gap: 1rem;
        }
        .navigation ul {
            flex-wrap: wrap;
            justify-content: center;
        }
        .auth-links {

```


```

        justify-content: center;
    }
    .filter-form {
        grid-template-columns: 1fr;
    }
    .announcements-list {
        grid-template-columns: 1fr;
    }
    .announcement-card-content {
        flex-direction: column;
    }
    .announcement-image {
        width: 100%;
        height: 200px;
    }
    .premium-request-actions {
        flex-direction: column;
        align-items: flex-start;
    }
    .main-photo {
        height: 300px;
    }
}
@media (max-width: 576px) {
    .header {
        padding: 0.8rem;
    }
    .main-content {
        padding: 10px;
    }
    .auth-page,
    .announcement-details,
    .premium-request,
    .chat-page {
        padding: 1.2rem;
    }
    .form-actions {
        flex-direction: column;
    }
    .form-actions button {
        width: 100%;
    }
    .users-table {
        font-size: 0.85rem;
    }
    .users-table th,
    .users-table td {
        padding: 0.6rem;
    }
}
/* Утиліти */
.text-center {
    text-align: center;
}
.mt-1 {
    margin-top: 0.5rem;
}
.mt-2 {
    margin-top: 1rem;
}
.mt-3 {
    margin-top: 1.5rem;
}
.mt-4 {

```


```

margin-top: 2rem;
}

.mb-1 {
margin-bottom: 0.5rem;
}

.mb-2 {
margin-bottom: 1rem;
}

.mb-3 {
margin-bottom: 1.5rem;
}

.mb-4 {
margin-bottom: 2rem;
}

.p-1 {
padding: 0.5rem;
}

.p-2 {
padding: 1rem;
}

.p-3 {
padding: 1.5rem;
}

.p-4 {
padding: 2rem;
}

.d-flex {
display: flex;
}

.justify-content-between {
justify-content: space-between;
}

.align-items-center {
align-items: center;
}

.flex-wrap {
flex-wrap: wrap;
}

.flex-column {
flex-direction: column;
}

.gap-1 {
gap: 0.5rem;
}

.gap-2 {
gap: 1rem;
}

```

CHANGELOG.md

This file explains how Visual Studio created the project.
The following steps were used to generate **this** project:

- Create project file (`AdDesk.esproj`).
- Create `launch.json` to enable debugging.
- Install npm packages: `npm init && npm i --save-dev eslint`.
 - Create `app.js`.
 - Update `package.json` entry point.
 - Create `eslint.config.js` to enable linting.
- Add project to solution.
 - Write **this** file.

