МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ДОНЕЦЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ВАСИЛЯ СТУСА
ФАКУЛЬТЕТ ІНФОРМАЦІЙНИХ І ПРИКЛАДНИХ ТЕХНОЛОГІЙ
КАФЕДРА ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ

ПОЛІЩУК ОЛЕНА СЕРГІЇВНА

Допускається до захисту:
в.о. завідувача кафедри
інформаційних технологій
д-р техн. наук, професор
__Наталія ВЕСЕЛОВСЬКА
«____»_____ 2026 р.

РОЗРОБКА АДАПТИВНОЇ ВЕБПЛАТФОРМИ ПОШУКУ ТА ФІЛЬТРАЦІЇ НЕРУХОМОСТІ НА REACT

Спеціальність 122 Комп’ютерні науки

Кваліфікаційна (бакалаврська) робота

Керівник:
Олександр ПАВЛЮК , старший викладач
кафедри інформаційних технологій,
к.т.н

_____

Оцінка: _____ / _____ / _____
(бали за шкалою ЄКТС / за національною шкалою)

Голова ЕК: _____
(підпис)

Вінниця - 2026

АНОТАЦІЯ

Поліщук О. С. Розробка адаптивної вебплатформи пошуку та фільтрації нерухомості на React. Спеціальність 122 «Комп’ютерні науки». Донецький національний університет імені Василя Стуса, Вінниця 2026.

У кваліфікаційній (бакалаврській) роботі досліджено сучасний стан онлайн-сервісів нерухомості, проаналізовано функціональність провідних платформ та розроблено повноцінну адаптивну вебплатформу EstatePro. Система забезпечує зручний багатопараметричний пошук і фільтрацію об’єктів, персональні кабінети для користувачів, агентів та адміністраторів, модерацію оголошень, управління обраним, історією переглядів, блогом, FAQ та аналітичні можливості для адмін-панелі.

Застосунок реалізовано на стекі React (фронтенд з адаптивним інтерфейсом), Express + SQLite+ PostgreSQL.

Ключові слова: вебплатформа, нерухомість, адаптивний інтерфейс, пошук і фільтрація, React, Express, SQLite, JWT, PostgreSQL.

73 ст., 52 рис., 10 табл., 7 дод., 47 джерела.

ABSTRACT

Polishchuk O. S. Development of an adaptive web platform for real estate search and filtering on React. Specialty 122 «Computer Science». Vasyl Stus Donetsk National University, Vinnytsia 2026.

The bachelor’s thesis examines the current state of online real estate services, analyzes leading platforms and develops the full-featured adaptive web platform EstatePro. The system provides convenient multi-parameter search and filtering of properties, personal dashboards for users, agents and administrators, moderation of listings, management of favorites, view history, blog, FAQ section and analytical tools in the admin panel.

The application is implemented using the React stack (frontend with responsive interface), Express + SQLite+PostgreSQL.

Keywords: web platform, real estate, responsive interface, search and filtering, React, Express, SQLite, JWT, PostgreSQL.

73 pp., 52 fig., 10 tbl., 7 app., 47 ref.

ЗМІСТ

ВСТУП

Сучасний ринок нерухомості в Україні переживає помітну цифрову трансформацію, що суттєво змінює способи пошуку, порівняння та придбання житла. Онлайн-платформи стають основним каналом взаємодії між покупцями, орендарями, продавцями та агентами, забезпечуючи швидкий доступ до великого обсягу пропозицій, фільтрацію за десятками параметрів та візуалізацію об’єктів. Впровадження повноцінних веб-додатків дозволяє скоротити час на пошук на 50–70 %, підвищує прозорість угод і сприяє формуванню більш раціонального попиту навіть в умовах нестабільного економічного середовища.

Актуальність теми дослідження зумовлена стрімким зростанням частки онлайн-пошуку нерухомості в Україні у 2024–2026 роках, що супроводжується одночасним збільшенням вимог до зручності, швидкості та персоналізації сервісів. Багато користувачів досі стикаються з фрагментованою інформацією, недостатньою якістю фотографій, повільним оновленням даних та складними формами фільтрації на популярних майданчиках. Існуючі рішення (DOM.RIA, LUN.ua, OLX Нерухомість та інші) пропонують значний функціонал, проте часто мають обмеження для певних сегментів ринку: недостатню адаптивність мобільних інтерфейсів, слабку інтеграцію історії переглядів та обраного, обмежені можливості для агентів та адміністраторів, а також недостатній рівень модерації та верифікації контенту. Таким чином, створення комплексної адаптивної вебплатформи з повноцінним backend, ролями користувачів, модерацією та українською локалізацією залишається затребуваним завданням.

Мета роботи полягає у розробці адаптивної вебплатформи пошуку та фільтрації нерухомості EstatePro на базі React та Node.js, яка забезпечує зручний пошук об’єктів, управління оголошеннями для агентів, модерацію для адміністраторів, систему обраного, історію переглядів та блог.

Об’єктом дослідження є процеси створення повнофункціональної вебплатформи для пошуку, перегляду та управління об’єктами нерухомості в умовах цифровізації ринку житла.

Предмет дослідження становлять методи, архітектурні рішення, моделі даних, алгоритми фільтрації та програмні засоби реалізації клієнтської та серверної частин вебплатформи з використанням сучасного стеку технологій.

Завданнями дослідження визначено:

– проаналізувати сучасний стан ринку онлайн-сервісів нерухомості в Україні та світі, виявити ключові тенденції та обмеження існуючих рішень;

– сформувати функціональні та нефункціональні вимоги до вебплатформи, враховуючи потреби різних ролей користувачів;

– розробити архітектуру системи, структуру бази даних та основні алгоритми пошуку й фільтрації;

– реалізувати клієнтську частину на React з адаптивним інтерфейсом та інтеграцією з API;

– створити серверну частину на Node.js + Express з JWT-автентифікацією, ролями та безпечним доступом до даних;

– провести комплексне функціональне та адаптивне тестування платформи, оцінити її ефективність та визначити напрями подальшого розвитку.

Структура кваліфікаційної роботи включає вступ, три основні розділи, висновки, список використаних джерел та додатки. Перший розділ присвячено теоретичним основам, аналізу ринку та огляд технологій. Другий розділ описує проєктування архітектури, моделей даних та алгоритмів. Третій розділ містить реалізацію інтерфейсу, серверної логіки, тестування та оцінку результатів. Робота спирається на понад 30 джерел, включає численні рисунки інтерфейсів, таблиці порівнянь та лістинги ключового коду.

Практичне значення роботи полягає у створенні готового до використання прототипу повноцінної платформи EstatePro, що поєднує зручний пошуковий інтерфейс, кабінети для агентів та адміністраторів, систему обраного та історії переглядів, блог та модерацію контенту. Отриманий продукт може слугувати як основа для реального комерційного сервісу, так і демонстраційним рішенням для невеликих та середніх агентств нерухомості, що прагнуть підвищити якість онлайн-присутності та ефективність взаємодії з клієнтами.

Апробація результатів дослідження здійснювалася шляхом доповіді основних положень роботи на VII Всеукраїнській науково-практичній конференції здобувачів вищої освіти та молодих вчених «Прикладні інформаційні технології» (м. Вінниця, 15 травня 2026 року), представлені тези «Розробка адаптивної вебплатформи пошуку та фільтрації нерухомості на React» (Додаток В) [43]. V Всеукраїнська науково-практична конференція здобувачів вищої освіти та молодих вчених «Прикладні інформаційні технології» (м. Вінниця, 24 травня 2024 року), представлені тези «Теоретичні засади хешування даних» (Додаток Г) [44], «Чисельні методи розв’язання нелінійних рівнянь» (Додаток Д) [45]. IV Всеукраїнська науково-практична конференція здобувачів вищої освіти та молодих вчених «Прикладні інформаційні технології» (м. Вінниця, 19 травня 2023 року), представлені тези «Застосування алгоритму Дейкстри для пошуку оптимального маршруту» (Додаток Е) [46]. IV Всеукраїнська науково-практична конференція «Комп’ютерні технології обробки даних» - 2023 (м. Вінниця), представлені тези «Сутність і складники рекурентних алгоритмів» (Додаток Ж) [47].

РОЗДІЛ 1
ТЕОРЕТИЧНІ ОСНОВИ СТВОРЕННЯ ВЕБПЛАТФОРМ ПОШУКУ НЕРУХОМОСТІ

1.1 Особливості ринку онлайн сервісів пошуку нерухомості

Сучасний ринок онлайн-сервісів пошуку нерухомості являє собою динамічну екосистему, яка поєднує елементи цифрової економіки, поведінкових особливостей споживачів та глобальних тенденцій цифрової трансформації. Цей сектор виник як відповідь на потребу в швидкому, зручному та персоналізованому доступі до інформації про об’єкти нерухомості, що раніше обмежувалася традиційними агентствами та газетними оголошеннями. За останні роки ринок демонструє стабільне зростання, зумовлене як технологічним прогресом, так і змінами в суспільних пріоритетах, зокрема прагненням до мобільності та прозорості угод.

Глобальний обсяг ринку проптех (property technology) сягнув значних показників, а в Україні цей процес набув особливого характеру через вплив економічних та соціальних факторів, що стимулювали перехід до цифрових платформ [1].

Однією з ключових особливостей є інтеграція великих даних та аналітики, яка дозволяє платформам обробляти мільйони запитів щодня, пропонуючи користувачам релевантні результати на основі геолокації, цінових параметрів та індивідуальних уподобань. Такий підхід не лише підвищує ефективність пошуку, але й сприяє формуванню довіри між учасниками ринку, оскільки алгоритми забезпечують мінімальну похибку в оцінці вартості об’єктів.

У контексті глобальних тенденцій спостерігається активне впровадження штучного інтелекту для прогнозування ринкових коливань, що особливо актуально в умовах нестабільності, коли традиційні моделі оцінки втрачають актуальність. Наприклад, платформи використовують машинне навчання для аналізу історичних даних про ціни та попит, що дозволяє користувачам

отримувати персоналізовані рекомендації, адаптовані до регіональних особливостей [2].

Важливим аспектом є адаптивність до мобільних пристроїв, яка стала нормою для сучасних сервісів. Користувачі все частіше здійснюють пошук нерухомості в дорозі або вдома, тому інтерфейси платформ повинні забезпечувати швидке завантаження, інтуїтивну навігацію та підтримку мультимедійного контенту, такого як віртуальні тури чи 360-градусні знімки. Ця мобільність не лише розширює аудиторію, але й сприяє залученню молодшого покоління покупців, для яких цифровий досвід є визначальним фактором при виборі житла. Водночас ринок стикається з викликами, пов’язаними з захистом персональних даних, оскільки збір інформації про уподобання клієнтів вимагає суворого дотримання норм конфіденційності, що регулюються міжнародними стандартами та національним законодавством [3].

Розвиток ринку онлайн-сервісів нерухомості також характеризується регіональною диференціацією, де в різних країнах акценти робляться на різних аспектах. У розвинених економіках домінує інтеграція з фінансовими сервісами, що дозволяє проводити попередню оцінку кредитоспроможності безпосередньо на платформі, тоді як у країнах, що розвиваються, пріоритетом стає доступність інформації для широких верств населення.

В Україні цей процес набув особливого прискорення через зростання попиту на прозорі інструменти пошуку, що відображається в статистиці активності користувачів. За даними аналітичних звітів, попит на онлайн-платформи стабільно відновлюється, з акцентом на вторинний ринок та оренду, де цифрові сервіси дозволяють оперативно порівнювати пропозиції без фізичного відвідування об’єктів [4].

Для наочного розуміння структури ринку доцільно звернутися до класифікації його основних сегментів, яка ілюструє розподіл платформ за типами послуг та аудиторією. Як видно з наведеної в таблиці 1.1 інформації, ринок поділяється на кілька ключових категорій, кожна з яких має власні особливості функціонування та вплив на загальну динаміку.

Таблиця 1.1 – Основні сегменти ринку онлайн-сервісів пошуку нерухомості

Сегмент	Характеристика послуг	Частка ринку (приблизно, %)	Основні виклики
Платформи продажу житла	Пошук, фільтрація за ціною та локацією, віртуальні тури	45	Точність даних про об’єкти
Сервіси оренди	Короткострокова та довгострокова оренда, інтеграція з календарями	30	Сезонні коливання попиту
Агрегатори з агентами	Профілі спеціалістів, консультації онлайн	15	Конкуренція між агентами
Інвестиційні платформи	Аналітика доходності, прогнози ринку	10	Регуляторні обмеження

Ця таблиця демонструє, як різні сегменти доповнюють один одного, створюючи комплексну екосистему, де користувач може перейти від простого пошуку до повноцінної угоди. Таке розмаїття сприяє підвищенню конкурентоспроможності ринку, але водночас вимагає від розробників платформ постійного вдосконалення алгоритмів для забезпечення релевантності результатів.

Особливої уваги заслуговує роль персоналізації в сучасних онлайн-сервісах. Алгоритми, що аналізують попередні запити користувачів, дозволяють формувати індивідуальні добірки об’єктів, враховуючи не лише базові параметри, як-от площа чи кількість кімнат, але й додаткові критерії, такі як наявність інфраструктури чи екологічні характеристики району. Це не лише підвищує задоволеність клієнтів, але й сприяє швидшому прийняттю рішень про покупку чи оренду. У глобальному масштабі така персоналізація вже стала стандартом, а в Україні вона набирає обертів завдяки інтеграції з локальними даними про транспортну доступність та соціальну інфраструктуру [5].

Ще однією визначальною особливістю ринку є інтеграція технологій віртуальної та доповненої реальності, які дозволяють користувачам «відвідувати» об’єкти без фізичної присутності. Такий підхід особливо актуальний у постпандемійний період, коли безпека та зручність стали

пріоритетами. Платформи, що пропонують 3D-моделі та інтерактивні плани, демонструють вищий рівень конверсії запитів у реальні угоди, оскільки клієнти отримують повне уявлення про простір ще на етапі пошуку. Ці технології також сприяють екологічності ринку, зменшуючи кількість непотрібних поїздок на перегляди [6].

Для ілюстрації типового процесу взаємодії користувача з онлайн-платформою пошуку нерухомості доцільно розглянути схему, яка відображає послідовність етапів від початкового запиту до завершення пошуку. Як показано на Рис. 1.1, цей процес включає кілька взаємопов'язаних етапів, що забезпечують ефективність і зручність для кінцевого користувача.

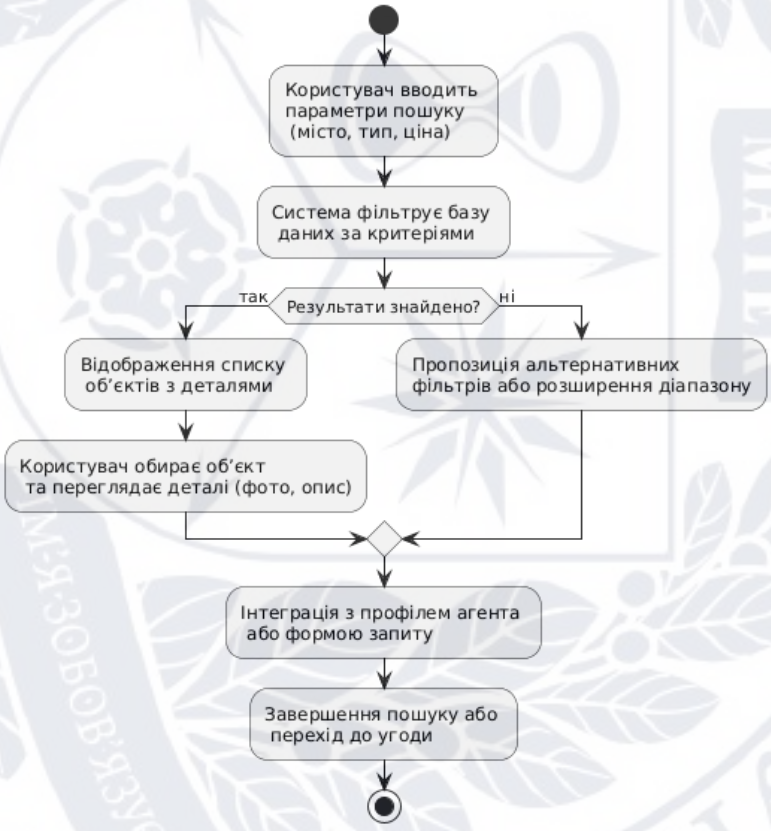

Рисунок 1.1 – Схема процесу пошуку та фільтрації нерухомості на онлайн-платформах

Ця схема підкреслює циклічний характер взаємодії, де кожний етап оптимізований для мінімізації часу користувача та максимізації релевантності

результатів. Вона відображає загальну логіку функціонування сучасних сервісів, де акцент робиться на інтерактивності та швидкості обробки даних.

Поряд з технологічними інноваціями ринок онлайн-сервісів нерухомості стикається з низкою викликів, серед яких – забезпечення точності інформації та боротьба з фейковими оголошеннями. Умови сучасної економіки вимагають від платформ впровадження систем верифікації даних, що включає перевірку документів власників та інтеграцію з державними реєстрами. Такі заходи не лише підвищують довіру, але й сприяють стабільності ринку в цілому. Крім того, важливим аспектом є екологічна спрямованість, де платформи починають акцентувати увагу на енергоефективних об’єктах та сталому розвитку, що відповідає глобальним тенденціям декарбонізації [7].

Узагальнюючи, ринок онлайн-сервісів пошуку нерухомості продовжує еволюціонувати під впливом цифрових технологій, стаючи невід’ємною частиною повсякденного життя споживачів. Його особливості, такі як персоналізація, мобільність та інтеграція інновацій, забезпечують високий рівень доступності інформації та сприяють формуванню прозорого середовища для укладання угод. Ці тенденції не лише визначають поточний стан сектору, але й відкривають перспективи для подальшого розвитку, де головну роль відіграватимуть адаптивність до нових потреб суспільства та постійне вдосконалення алгоритмів обробки даних.

У майбутньому такі платформи стануть основним інструментом для ефективного управління житловими потребами населення, сприяючи економічному зростанню та соціальній стабільності.

1.2 Аналіз існуючих вебплатформ пошуку та фільтрації нерухомості

Сучасні вебплатформи пошуку та фільтрації нерухомості відіграють ключову роль у цифровій трансформації ринку житла, забезпечуючи користувачам швидкий доступ до актуальних пропозицій та можливість точного підбору об’єктів відповідно до індивідуальних критеріїв. Аналіз провідних

сервісів свідчить про те, що їх архітектура базується на комплексних системах обробки даних, які поєднують геопросторові технології, багатокритеріальну фільтрацію та елементи штучного інтелекту, що значно підвищує ефективність взаємодії з користувачем.

Однією з найпоширеніших українських платформ є DOM.RIA (рис. 1.2), яка вирізняється детальною модерацією оголошень та інтеграцією з інтерактивними картами, дозволяючи застосовувати фільтри за ціновим діапазоном, типом об’єкта, кількістю кімнат та додатковими параметрами, такими як наявність паркінгу чи близькість до інфраструктури. Користувачі отримують можливість візуалізувати результати на карті з позначенням відстаней до ключових об’єктів, що робить процес пошуку більш інтуїтивним і орієнтованим на реальні потреби, особливо в умовах динамічного ринку, де ціни та доступність змінюються щоденно [8].

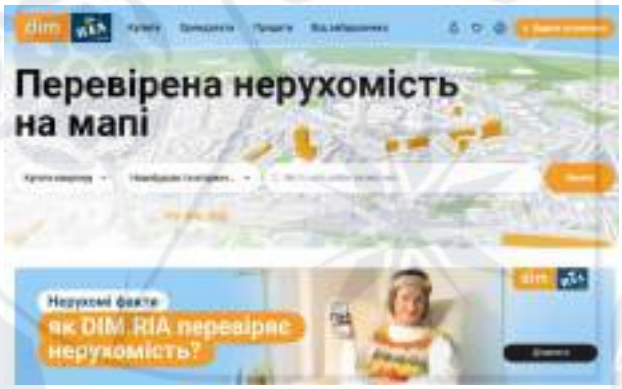

Рисунок 1.2 – Інтерфейс DOM.RIA

Подібні підходи реалізовано в LUN.ua (рис. 1.3), де акцент зроблено на агрегуванні даних з первинного та вторинного ринків, включаючи статистичні аналізи цін та новобудови. Платформа пропонує розширені інструменти для порівняння об’єктів за кількістю кімнат, площею та розташуванням, з можливістю застосування фільтрів за містом, типом нерухомості та додатковими зручностями, такими як опалення чи охорона. Інтеграція з динамічними графіками цін допомагає користувачам оцінювати тенденції ринку в реальному часі, а карта з позначенням новобудов забезпечує швидкий перегляд варіантів у

конкретних районах, що особливо актуально для інвесторів та сімей, які шукають довгострокові рішення [9].

Рисунок 1.3 – Інтерфейс LUN.ua

Міжнародний досвід демонструє ще більш інноваційні рішення, як на платформі Zillow (рис. 1.4), де впроваджено штучний інтелект для природномовного пошуку: користувачі можуть формулювати запити у формі розмовного тексту, наприклад, щодо часу на дорогу до роботи, близькості до шкіл чи рівня доступності за бюджетом, що доповнює традиційні фільтри за ціною, типом та кількістю спалень. Такий підхід мінімізує необхідність ручного налаштування параметрів і дозволяє отримувати персоналізовані результати, враховуючи контекстні фактори, такі як транспортна доступність чи інфраструктура району [10].

Рисунок 1.4 – Інтерфейс Zillow.com

Для систематизації порівняння функціональних можливостей цих платформ доцільно звернутися до Таблиці 1.2, яка ілюструє відмінності в реалізації ключових інструментів пошуку та фільтрації, підкреслюючи, як саме вони адаптуються до потреб користувачів у різних ринкових умовах.

Таблиця 1.2 – Порівняльний аналіз функціональних можливостей платформ пошуку та фільтрації нерухомості

Платформа	**Фільтрація за ціною та бюджетом**	**Фільтрація за типом об'єкта та кількістю кімнат**	**Інтеграція з картою та геолокацією**	**Підтримка AI або природномовного пошуку**	**Верифікація оголошень та статистика ринку**
DOM.RIA	Діапазон мінімальної та максимальної ціни, урахування іпотеки	Тип (квартира, будинок, комерційна), кімнати, ванни	Інтерактивна карта з відстанями до інфраструктури	Відсутня, фокус на ручних фільтрах	Повна модерація, перевірка оголошень
LUN.ua	Динамічні діапазони з графіками цін	Тип, кімнати, новобудови чи вторинний ринок	Карта новобудов з детальними позначками	Часткова аналітика тенденцій	Агрегування даних, статистика продажів
Zillow	Бюджет з урахуванням аффордабельності та іпотеки	Тип, спальні, ванни, стиль	Карта з commute time та schools	Так, природномовний пошук з AI	Zestimate оцінка, верифікація через агенти

Як видно з наведеної таблиці, усі платформи акцентують увагу на багаторівневій фільтрації, але відрізняються рівнем інтеграції штучного інтелекту та візуалізації, що впливає на швидкість і точність підбору об'єктів.

Перед тим, як розглянути обмеження цих рішень та перспективи їх розвитку, варто проілюструвати загальний процес взаємодії користувача з такими системами за допомогою Рис. 1.5, який демонструє послідовність кроків

від введення запиту до отримання персоналізованих результатів з урахуванням фільтрів і візуалізації.

Рисунок 1.5 – Схема типового процесу пошуку та фільтрації нерухомості на вебплатформах

Наведена схема підкреслює циклічність процесу, де кожен етап оптимізується за рахунок геопросторових даних і багатопараметричної обробки, що дозволяє користувачам швидко переходити від загального пошуку до конкретних дій, таких як перегляд деталей чи контакт з агентом.

Аналіз показує, що платформи типу DOM.RIA та LUN.ua добре адаптовані до локальних особливостей українського ринку, де домінують запити на житло в конкретних містах з урахуванням інфраструктури та цінової динаміки, тоді як Zillow демонструє глобальні тенденції до персоналізації через штучний інтелект, що дозволяє враховувати не лише технічні параметри, а й lifestyle-фактори, такі як близькість до шкіл чи транспортних вузлів [11]. Водночас спільним обмеженням залишається залежність від якості вхідних даних: не всі оголошення проходять повну верифікацію, що може призводити до неточностей у результатах, особливо в регіонах з низькою активністю ринку. Крім того, відсутність єдиних стандартів для фільтрів ускладнює порівняння об’єктів між платформами, а інтеграція з додатковими сервісами, такими як іпотечні

калькулятори чи статистика цін, реалізується нерівномірно [12].

Міжнародний досвід Zillow підкреслює потенціал AI для спрощення інтерфейсу, де природномовні запити замінюють складні форми фільтрації, дозволяючи користувачам описувати потреби в повсякденній мові та отримувати миттєві рекомендації з урахуванням commute time чи шкільних рейтингів [13]. В українському контексті подібні інновації могли б значно підвищити доступність для широкої аудиторії, включаючи користувачів без глибоких технічних навичок, проте потребують додаткової інтеграції з локальними даними про інфраструктуру та цінові тенденції [14]. Загалом, аналіз цих платформ підтверджує, що ефективна вебплатформа пошуку нерухомості повинна поєднувати багатофакторну фільтрацію, візуалізацію на картах та елементи персоналізації, забезпечуючи користувачам не лише швидкий пошук, а й обґрунтоване прийняття рішень у динамічному ринковому середовищі. Такий підхід сприяє прозорості ринку та підвищенню довіри до цифрових сервісів, що є ключовим фактором для подальшого розвитку галузі в умовах цифрової трансформації.

1.3 Технології та інструменти розробки сучасних вебплатформ

Сучасні вебплатформи пошуку та фільтрації нерухомості є складними цифровими екосистемами, що поєднують у собі високопродуктивні інструменти для обробки великих обсягів даних, забезпечення адаптивності інтерфейсу та захисту конфіденційної інформації користувачів. У їхній основі лежить принцип односторінкових додатків, які дозволяють динамічно оновлювати контент без перезавантаження сторінок, що суттєво підвищує зручність пошуку об’єктів за геолокацією, ціною чи типом нерухомості. Такий підхід базується на реактивних бібліотеках JavaScript, які оптимізують рендеринг через віртуальний DOM і забезпечують плавну взаємодію навіть при складних фільтрах та рекомендаціях [15].

Ці технології дозволяють розробникам створювати інтерфейси, що

автоматично адаптуються до будь-яких пристроїв – від смартфонів до великих моніторів, – завдяки гнучким системам компонентів і медіа-запитам. Паралельно з цим серверна частина платформ спирається на середовища виконання, які підтримують єдину мову програмування для фронтенду та бекенду, що спрощує інтеграцію, прискорює розробку та зменшує кількість помилок при обміні даними через API. Такі рішення забезпечують швидку обробку запитів на фільтрацію, сортування та персоналізовані рекомендації, що є критичним для ринку нерухомості з його динамічними змінами пропозиції [16].

Для зберігання інформації про об’єкти, користувачів та історію взаємодій застосовуються легковагові реляційні системи баз даних, які підтримують зовнішні ключі та транзакції, гарантуючи цілісність даних навіть при високому навантаженні. Ці інструменти дозволяють ефективно поєднувати структуровані записи з динамічними запитами, що особливо важливо для платформ, де користувачі щодня виконують тисячі пошуків за різними критеріями.

Водночас безпека таких систем досягається через спеціалізовані бібліотеки для захисту HTTP-заголовків, налаштування політики доступу між доменами та механізми автентифікації на основі токенів, які забезпечують безпечне зберігання сесій без ризику витоку даних [17].

Розвиток адаптивних вебплатформ неможливий без інтеграції інструментів для автоматичного тестування та розгортання, які дозволяють підтримувати стабільність коду при частих оновленнях. Сучасні практики включають використання контейнеризації для швидкого масштабування серверів та інструментів моніторингу продуктивності, що допомагає платформам залишатися доступними навіть під піковим навантаженням у пікові сезони ринку нерухомості.

Крім того, для покращення пошуку застосовуються технології аналізу даних, які дозволяють пропонувати користувачам релевантні варіанти на основі попередніх запитів, геолокації та ринкових тенденцій, роблячи платформи не просто каталогами, а інтелектуальними помічниками [18].

Важливим елементом є забезпечення кросплатформенної сумісності, коли

інтерфейс автоматично підлаштовується під розмір екрана, орієнтацію пристрою та швидкість з’єднання. Це досягається завдяки модульним стилям і компонентам, які дозволяють розробникам створювати єдиний код, що працює однаково на мобільних та десктопних пристроях. У поєднанні з серверними технологіями, які підтримують асинхронну обробку запитів, такі платформи стають справжнім інструментом для швидкого та зручного пошуку житла чи комерційної нерухомості в умовах постійних ринкових змін [19].

Для захисту користувацьких даних і запобігання несанкціонованому доступу широко застосовуються стандартизовані механізми автентифікації, що поєднують короткострокові токени для активних сесій та довгострокові механізми відновлення. Це дозволяє користувачам безпечно зберігати вподобання, історію пошуку та персональні налаштування, не ризикуючи конфіденційністю.

Паралельно з цим інструменти для обробки статичних файлів і зображень забезпечують швидке завантаження візуального контенту, що є ключовим для платформ нерухомості, де якісні фотографії та плани об’єктів визначають рішення користувача [20].

Як видно з наведеного порівняння ключових технологій фронтенду (таблиця 1.3), вибір конкретних інструментів безпосередньо впливає на швидкість розробки, масштабованість та користувацький досвід.

Таблиця 1.3 – Порівняльний аналіз основних технологій фронтенд-розробки для сучасних вебплатформ

Технологія	**Швидкодія рендерингу**	**Адаптивність до пристроїв**	**Підтримка компонентної архітектури**	**Рівень інтеграції з бекендом**
React	Висока (віртуальний DOM)	Висока (медіа-запити)	Повна	Відмінна (API)
Vue.js	Середня	Висока	Повна	Добра
Angular	Висока	Середня	Повна	Відмінна

Таблиця 1.3 наочно демонструє, чому саме технології з віртуальним DOM і компонентним підходом домінують у розробці платформ, де користувачам потрібен миттєвий відгук на фільтри та пошукові запити.

Рис. 1.6 ілюструє типову взаємодію компонентів сучасної вебплатформи пошуку нерухомості, де клієнтська частина обмінюється даними з сервером через стандартизовані API, а база даних забезпечує швидкий доступ до інформації.

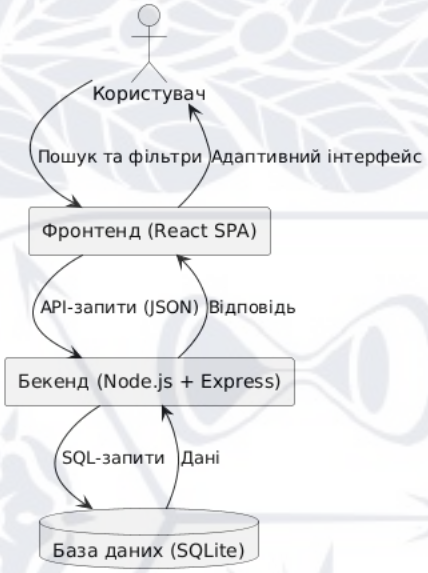

Рисунок 1.6 – Архітектура сучасної вебплатформи пошуку нерухомості

Рисунок 1.6 підкреслює, як єдина архітектура забезпечує безперервну роботу від запиту до відображення результатів, що є визначальним для зручності використання таких платформ.

Додатковим аспектом є інтеграція інструментів для моніторингу та логування, які дозволяють розробникам оперативно виявляти проблеми продуктивності чи безпеки. Це особливо актуально для платформ, що працюють з геоданими та персональною інформацією, де навіть невелика затримка може вплинути на задоволеність користувачів. Сучасні практики також включають автоматизоване тестування на всіх етапах, що гарантує стабільність при оновленнях і розширенні функціоналу [21].

Таким чином, поєднання реактивних фронтенд-технологій, швидких серверних середовищ, легковагових баз даних та надійних механізмів безпеки формує основу сучасних вебплатформ пошуку нерухомості. Ці інструменти не лише забезпечують технічну ефективність, а й сприяють створенню інтуїтивно зрозумілих сервісів, які допомагають людям швидко знаходити потрібне житло чи комерційні приміщення в умовах динамічного ринку. Завдяки постійному розвитку цих технологій вебплатформи стають справжнім мостом між покупцями, продавцями та фахівцями ринку, роблячи процес пошуку прозорим, швидким і доступним для всіх учасників.

РОЗДІЛ 2
ПРОЄКТУВАННЯ МЕТОДІВ ТА МОДЕЛЕЙ ВЕБПЛАТФОРМИ ПОШУКУ НЕРУХОМОСТІ

2.1 Формування функціональних вимог до вебплатформи

Формування функціональних вимог становить фундаментальний етап проєктування будь-якої сучасної вебплатформи, особливо в умовах динамічного розвитку ринку нерухомості, де користувачі очікують швидкого, зручного та персоналізованого доступу до інформації [22]. У контексті адаптивної вебплатформи пошуку та фільтрації нерухомості EstatePro, розробленої як односторінковий додаток на базі React з серверною частиною на Node.js та Express, цей етап забезпечує чітке визначення можливостей системи, що безпосередньо випливає з аналізу реальних потреб різних категорій користувачів – від звичайних шукачів житла до агентів та адміністраторів.

Такий підхід не лише гарантує відповідність платформи сучасним стандартам цифрової трансформації галузі, але й сприяє створенню інтуїтивного інтерфейсу, який мінімізує час на пошук і максимізує ефективність взаємодії [23]. Завдяки інтеграції функціональних модулів, реалізованих у контролерах та компонентах, платформа стає інструментом, що поєднує технологічну точність з людським комфортом, дозволяючи користувачам легко орієнтуватися в об’єктах нерухомості, здійснювати вибір на основі актуальних фільтрів і підтримувати зв’язок з фахівцями.

Центральним елементом функціональних вимог є модуль пошуку та фільтрації об’єктів нерухомості, який реалізовано через компонент AllProperties.jsx та відповідний контролер properties.controller.js. Тут користувач отримує можливість динамічного застосування критеріїв, таких як місто, тип нерухомості, вид оголошення, діапазон цін, кількість спалень та інші параметри, що відображаються в реальному часі з підтримкою пагінації. Це забезпечує адаптивність платформи до різних сценаріїв використання: від швидкого

перегляду актуальних пропозицій у Києві чи Львові до детального порівняння параметрів, як-от площа чи наявність паркінгу [24].

Така функціональність не просто полегшує пошук, а й сприяє прийняттю обґрунтованих рішень, особливо в умовах мінливого ринку, де оперативність даних відіграє вирішальну роль. Передбачено також отримання списків унікальних міст і типів нерухомості через спеціальні ендпоінти getCities та getTypes, що дозволяє автоматично формувати фільтри без зайвого навантаження на користувача. Рис. 2.1 ілюструє основні варіанти використання цих модулів, підкреслюючи взаємодію ролей користувачів з ключовими функціями пошуку.

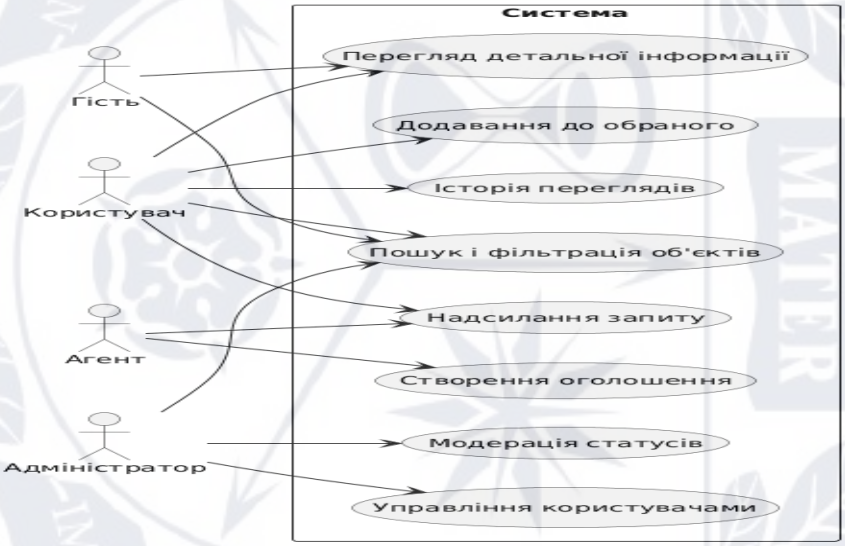

Рисунок 2.1 – Діаграма варіантів використання основних функціональних модулів вебплатформи

Наступним важливим блоком функціональних вимог є автентифікація та управління профілями, реалізовані в auth.controller.js та відповідних валідаціях. Система підтримує реєстрацію, вхід з JWT-токенами, оновлення профілю та зміну пароля, з автоматичним збереженням refresh-токенів у httpOnly cookie для підвищення безпеки [25]. Це дозволяє користувачам зберігати персональні дані, такі як обране чи історія переглядів, без ризику втрати при перезавантаженні сторінки.

Для ролі користувача передбачено доступ до favorites.controller.js та history.controller.js, де можна додавати об’єкти до обраного чи очищати історію переглядів, що робить платформу персоналізованою та зручною для довгострокового використання. Агенти та адміністратори отримують розширені можливості: створення та редагування профілів агентів через agents.controller.js, включаючи оновлення опису, досвіду та спеціалізації, що сприяє професійній презентації фахівців на ринку.

Особливо варто відзначити функціональність детального перегляду об’єктів у Property.jsx та properties.controller.js, де користувач бачить не лише базову інформацію, але й зручності, описові блоки та контакти агента. Тут інтегровано форму запиту через inquiries.controller.js, з адаптивними полями залежно від типу оголошення – для продажу додаються параметри іпотеки, для оренди – дата заселення та кількість мешканців [26].

Такий підхід забезпечує гнучкість взаємодії, дозволяючи клієнтам надсилати персоналізовані повідомлення безпосередньо з сторінки об’єкта, що значно прискорює процес комунікації з агентом. Крім того, для авторизованих користувачів активовано перемикач обраного та запис в історію переглядів, що реалізовано через API-ендпоінти з перевіркою статусу об’єкта на approved.

Для агентів ключовою функціональною вимогою є можливість створення та управління оголошеннями, що реалізовано в create та update методах properties.controller.js з автоматичним присвоєнням статусу pending для подальшої модерації адміністратором. Це включає завантаження зображень, додавання зручностей з amenities та блоків опису, забезпечуючи повний контроль над контентом [27]. Адміністратори, у свою чергу, отримують доступ до модерації через updateStatus, getPending та управління користувачами в users.controller.js, включаючи бан та зміну ролей. Така ієрархія прав доступу, реалізована з перевірками в middleware/auth.js, гарантує безпеку та прозорість процесу, запобігаючи несанкціонованим змінам [28].

Таблиця 2.1 відображає розподіл функціональних вимог за ролями користувачів, підкреслюючи диференціацію можливостей для забезпечення

ефективної роботи платформи.

Таблиця 2.1 – Класифікація функціональних вимог за ролями користувачів

Роль користувача	Основні функціональні можливості	Приклади реалізації в системі
Гість	Пошук, фільтрація, перегляд деталей	getAll у properties.controller, AllProperties.jsx
Користувач	Обране, історія, запити, профіль	favorites.controller, history.controller, inquiries.controller
Агент	Створення оголошень, управління запитами	create в properties.controller, getMy в inquiries.controller
Адміністратор	Модерація, управління користувачами, налаштування	updateStatus, users.controller, settings.controller

Додаткові модулі, такі як статті в articles.controller.js та FAQ у faqs.controller.js, розширюють функціональність платформи, надаючи користувачам освітній контент і відповіді на поширені питання. Це сприяє підвищенню довіри до сервісу, адже відвідувачі можуть не лише шукати об’єкти, але й отримувати аналітику ринку чи поради.

Підписка на розсилку через subscribers.controller.js та контактна форма в contacts.controller.js забезпечують зворотний зв’язок, дозволяючи адміністраторам збирати звернення та надсилати персональні пропозиції. Усе це інтегровано в єдину адаптивну систему, де React-компоненти, як-от PostPropertySection.jsx для розміщення оголошень чи PropertySingle.jsx для детального перегляду, забезпечують плавну навігацію незалежно від пристрою.

Таким чином, сформовані функціональні вимоги повністю відповідають специфіці вебплатформи EstatePro, реалізованої у лістингу проєкту, та забезпечують комплексне рішення для пошуку нерухомості. Вони враховують потреби всіх учасників ринку, сприяють автоматизації процесів і підвищують загальну ефективність взаємодії, роблячи платформу не просто інструментом пошуку, а повноцінним екосистемним рішенням для сучасного ринку нерухомості в Україні.

Такий підхід, підтверджений практичною реалізацією в контролерах та

компонентах, демонструє баланс між технічною складністю та користувацькою зручністю, що є ключем до успішного впровадження цифрових технологій у галузі.

2.2 Проєктування архітектури системи та структури даних

У процесі проєктування архітектури системи та структури даних вебплатформи пошуку та фільтрації нерухомості було обрано трирівневу клієнт-серверну модель, яка забезпечує чітке розділення обов'язків між інтерфейсом користувача, бізнес-логікою та зберіганням інформації.

Такий підхід дозволяє досягти високої адаптивності інтерфейсу, швидкої обробки запитів на пошук і фільтрацію об'єктів, а також надійної автентифікації та захисту даних, що повністю відповідає реалізованому лістингу коду з використанням React на клієнтській стороні та Express.js з sql.js на серверній [29].

Схема архітектури системи (рис. 2.2) наочно демонструє основні рівні взаємодії компонентів, де користувач через браузерний інтерфейс надсилає запити до серверної частини, яка в свою чергу звертається до бази даних для отримання та обробки інформації про нерухомість, агентів і користувачів.

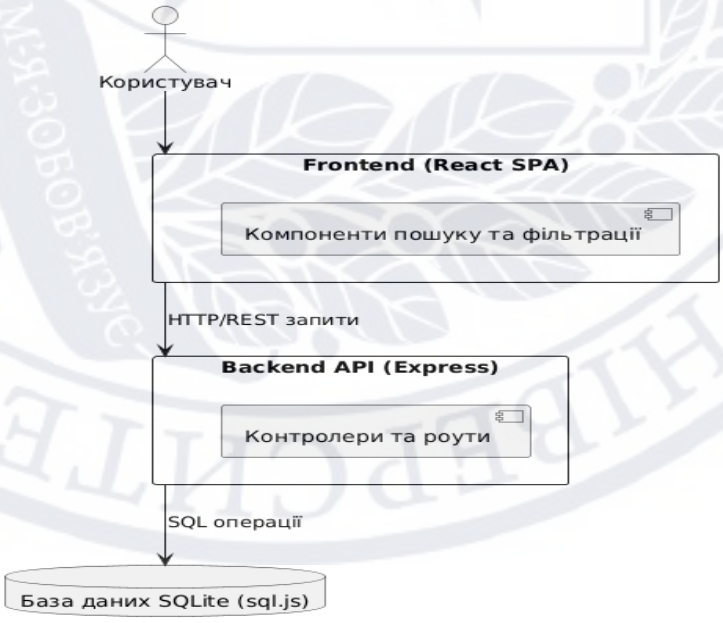

Рисунок 2.2 – Схема архітектури системи

Ця архітектура забезпечує ефективну взаємодію, де frontend-компоненти, такі як AllProperties.jsx та Property.jsx, динамічно оновлюють дані без перезавантаження сторінки, а backend, реалізований у server.js та properties.controller.js, обробляє запити з урахуванням статусів об’єктів і прав доступу [30]. Такий розподіл не тільки підвищує швидкодію платформи, але й спрощує подальше розширення функціоналу, наприклад, додавання нових фільтрів за ціною чи типом нерухомості.

Структурна схема системи (рис. 2.3) доповнює загальну картину, показуючи організацію модулів на рівні файлів і пакетів, що гарантує модульність і легкість підтримки коду.

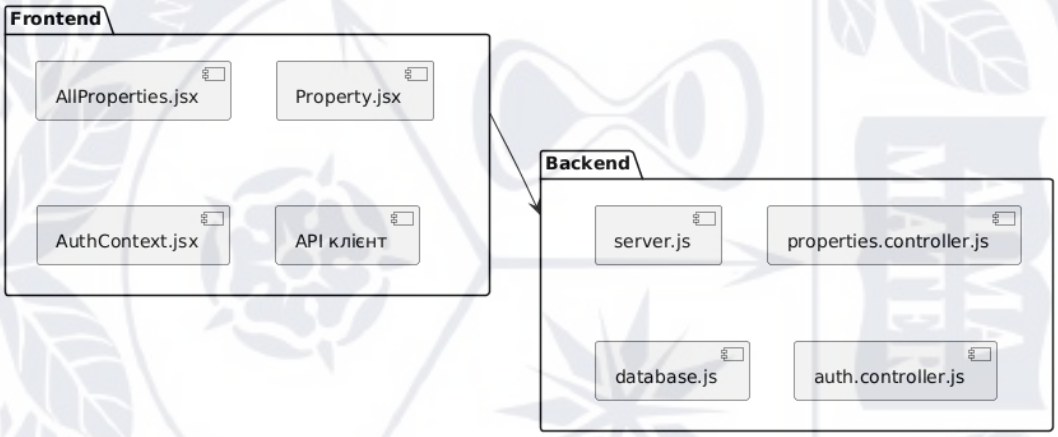

Рисунок 2.3 – Структурна схема системи

Як видно зі структурної схеми, frontend-частина зосереджена на компонентах для відображення списків і деталей об’єктів, тоді як backend містить контролери для обробки даних про нерухомість і автентифікацію, а database.js централізує доступ до бази даних. Для локальної розробки використовується SQLite, а PostgreSQL закладена в архітектуру як основна СУБД для розгортання серверної частини системи[31].

Діаграма компонентів (рис. 2.4) розкриває внутрішні взаємозв’язки між елементами, підкреслюючи, як окремі частини фронтенду та бекенду інтегруються для спільної роботи.

Рисунок 2.4 – Діаграма компонентів

Діаграма компонентів чітко ілюструє потік даних від React-компонентів через API до бази, що реалізовано в properties.controller.js і favorites.controller.js, забезпечуючи безперервну роботу функцій пошуку, обраного та історії переглядів [32].

Діаграма класів (рис. 2.5) детально показує зв’язки між основними класами та модулями, де кожен елемент залежить від інших для підтримки єдиної логіки системи.

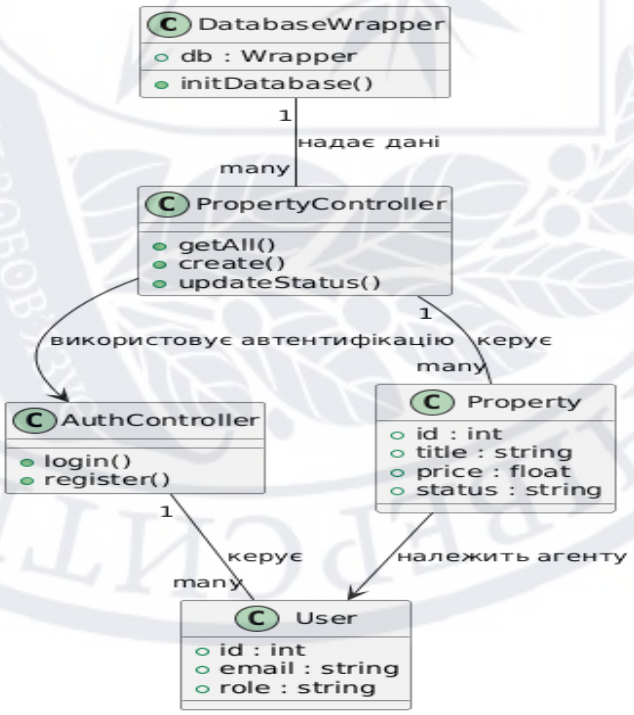

Рисунок 2.5 – Діаграма класів

Усі класи на діаграмі тісно пов’язані, що відображає реальну взаємодію в коді: DatabaseWrapper з database.js постачає дані PropertyController, який у свою чергу залежить від AuthController для перевірки прав доступу, а моделі Property і User забезпечують цілісність сутностей [33]. Така структура гарантує, що будь-яка зміна в одному модулі автоматично відображається в інших, підтримуючи стабільність платформи.

Діаграма станів (рис. 2.6) доповнює опис, фіксуючи можливі переходи статусів об’єктів нерухомості протягом їх життєвого циклу. Початковий стан об'єкта нерухомості після створення агентом або адміністратором визначається як «Очікує модерації», що відповідає статусу pending у базі даних. У стані «Очікує модерації» об'єкт недоступний для звичайних користувачів і очікує перевірки адміністратором через метод updateStatus. Після успішної модерації об'єкт переходить у стан «Схвалено», що робить його видимим у загальному каталозі та доступним для перегляду, додавання в обране та подання запитів. Зі стану «Схвалено» об'єкт може перейти у стан «Проданий» у випадку успішного оформлення угоди купівлі через transactions.controller. Зі стану «Схвалено» об'єкт може також перейти у стан «Орендований» після успішного оформлення договору оренди. Кінцеві стани «Проданий» та «Орендований» завершують життєвий цикл об'єкта, виключаючи його з активного пошуку та фільтрації для звичайних користувачів.

Рисунок 2.6 – Діаграма станів об'єкта нерухомості

Діаграма станів точно відповідає логіці в properties.controller.js, де статус переходить від “pending” до “approved” і далі до “sold” або “rented”, що забезпечує коректну фільтрацію та відображення об’єктів лише для авторизованих користувачів [34].

Для кращого розуміння організації даних було проаналізовано основні таблиці бази даних (таблиця 2.2), які формують основу зберігання інформації.

Таблиця 2.2 – Основні таблиці бази даних

Назва таблиці	**Опис**	**Ключові поля**
users	Дані користувачів і ролей	id, email, role, password_hash, is_active
properties	Об’єкти нерухомості	id, title, price, status, agent_id, city
agents	Профілі агентів	id, user_id, role_title, img_url
favorites	Обране користувачів	user_id, property_id
inquiries	Запити на об’єкти	property_id, user_id, status

Ця таблиця відображає ключові сутності, які обробляються в контролерах, забезпечуючи ефективну фільтрацію за містом, типом і ціною, як реалізовано в getAll методі [35]. Архітектура та структура даних повністю інтегровані з використанням JWT для автентифікації та cookie для refresh-токенів, що підвищує безпеку та зручність роботи на різних пристроях.

Для повного розуміння структури даних вебплатформи було створено схему бази даних (рис. 2.7), яка відображає основні таблиці, їх атрибути та ключові зв’язки між сутностями. Ця схема є фізичною моделлю даних, що безпосередньо відповідає реалізації в файлі database.js та контролерах (зокрема properties.controller.js, agents.controller.js, favorites.controller.js, inquiries.controller.js). Вона ілюструє, як саме організовано зберігання інформації про користувачів, об’єкти нерухомості, агентів, обране та запити, а також показує зовнішні ключі, що забезпечують референційну цілісність.

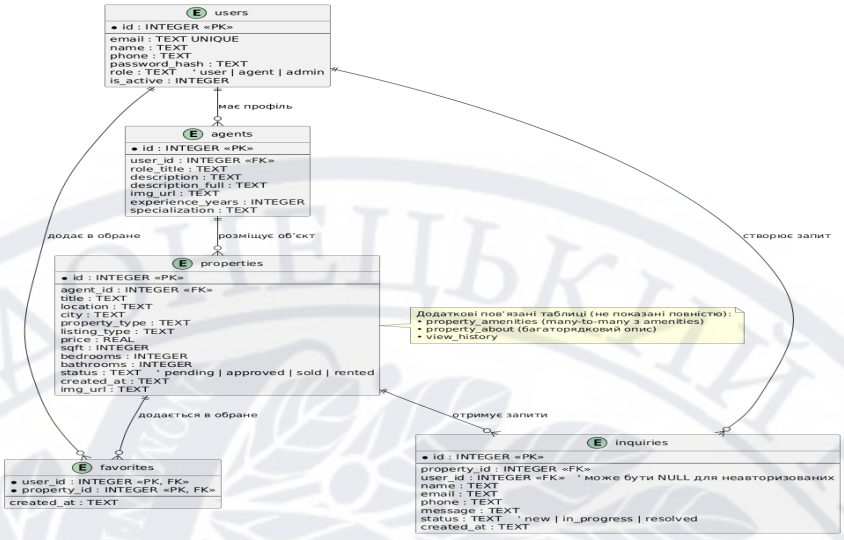

Рисунок 2.7 – Схема бази даних

Представлена схема чітко демонструє центральне місце таблиці properties як основної сутності платформи. Саме до неї прив’язані більшість операцій: відображення списків (getAll), детальний перегляд (getById), додавання в обране (favorites), подання запитів (inquiries), а також модерація статусів адміністратором (updateStatus). Зв’язок agents → properties через agent_id забезпечує чітке розмежування відповідальності агентів за розміщені об’єкти, що реалізовано в middleware getAgentId та перевірках прав у контролерах. Таблиця users є кореневою для автентифікації та авторизації, а поле role визначає доступ до функціоналу (адмін-панель, створення оголошень, перегляд лише власних запитів тощо).

Таблиця favorites реалізована як композитний первинний ключ (user_id + property_id), що запобігає дублюванню та гарантує швидкий пошук обраного для конкретного користувача. Аналогічно таблиця inquiries підтримує як авторизованих, так і неавторизованих користувачів (user_id може бути NULL), що відповідає логіці в inquiries.controller.js.

Така структура даних дозволяє ефективно виконувати типові запити платформи:

– фільтрацію об’єктів за містом, типом, ціною, кількістю спалень (properties.getAll);

– отримання профілю агента разом з його об’єктами (agents.getById);

– перевірку, чи об’єкт уже в обраному (favorites.check);

– відображення запитів лише для відповідного агента (inquiries.getMy).

Загалом, спроєктована схема бази даних є компактною, але достатньо повною для реалізації всієї функціональності, описаної в лістингу проєкту, та забезпечує хорошу продуктивність при відносно невеликих обсягах даних, характерних для SQLite з sql.js. Вона також залишає простір для розширення (додавання нових пов’язаних таблиць, індексів, повнотекстового пошуку тощо) без необхідності кардинальної переробки архітектури.

Такий дизайн дозволяє платформі адаптивно реагувати на потреби ринку нерухомості, пропонуючи швидкий пошук і детальний перегляд об’єктів без зайвих затримок.

Загалом, проєктована архітектура не лише відповідає сучасним стандартам розробки, але й створює надійний фундамент для подальшого розвитку системи, де кожна зміна в коді автоматично підтримує загальну стабільність і користувацький комфорт.

2.3 Розробка алгоритмів пошуку та фільтрації об’єктів нерухомості

У процесі проєктування методів та моделей вебплатформи пошуку нерухомості ключовим елементом стала розробка алгоритмів пошуку та фільтрації об’єктів нерухомості, які забезпечують швидкий доступ до релевантної інформації та підвищують зручність використання платформи для кінцевих користувачів [36]. Ці алгоритми інтегровані в backend і frontend частини системи, що дозволяє ефективно обробляти запити різної складності, враховуючи обсяг даних у базі та вимоги до продуктивності.

Серверна частина алгоритму, реалізована в методі getAll контролера properties.controller.js, базується на динамічному конструюванні SQL-запиту. Параметри, отримані з req.query, такі як city, property_type, listing_type, min_price, max_price та bedrooms, динамічно додаються до умов WHERE тільки якщо вони

присутні, при цьому базова умова status = 'approved' забезпечує відображення лише модерованих об’єктів.

Для безпеки всі параметри обробляються через prepared statements, що унеможливлює ін’єкції, а подальший підрахунок загальної кількості записів через окремий countQuery дає змогу розрахувати пагінацію з offset = (page - 1) * limit. Після основного SELECT з JOIN до таблиць agents та amenities результати доповнюються даними про зручності через додатковий запит, що дозволяє повноцінно представляти кожен об’єкт без надмірного навантаження на клієнт [37].

Рис. 2.8 ілюструє логіку послідовної обробки параметрів і формування запиту на сервері.

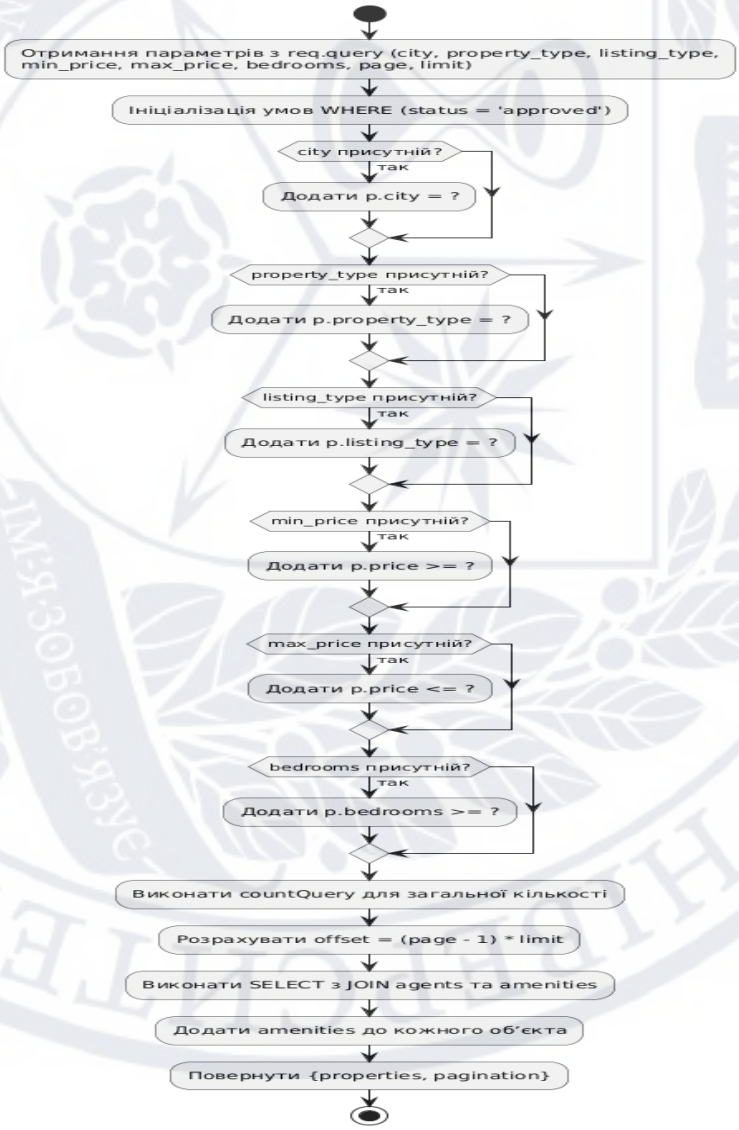

Рисунок 2.8 – Схема алгоритму серверної фільтрації об’єктів нерухомості

Як видно зі схеми, процес забезпечує гнучкість і безпеку завдяки використанню параметризованих запитів, що відповідає сучасним практикам роботи з базами даних у веб-додатках [38].

На клієнтській стороні в компоненті AllProperties.jsx алгоритм включає управління станом за допомогою useState для зберігання значень фільтрів (city, propertyType, listingType, search) і useEffect для автоматичного завантаження даних з API при зміні будь-якого параметра.

Початкове завантаження здійснюється через propertiesApi.getAll з передачею параметрів, після чого застосовується клієнтська фільтрація масиву properties за текстовим пошуком (перевірка title, location та city), що дозволяє користувачу уточнювати результати без додаткових серверних запитів.

Динамічні списки міст і типів завантажуються окремо через getCities та getTypes, забезпечуючи адаптивність інтерфейсу до актуальних даних бази [39].

Таблиця 2.3 наводить відповідність параметрів фільтрації полям бази даних, що демонструє пряму інтеграцію frontend і backend.

Таблиця 2.3 – Відповідність параметрів фільтрації полям бази даних

Параметр запиту	**Поле бази даних**	**Тип умови**
city	p.city	Рівність
property_type	p.property_type	Рівність
listing_type	p.listing_type	Рівність
min_price	p.price	>=
max_price	p.price	<=
bedrooms	p.bedrooms	>=

Рис. 2.9 демонструє послідовність кроків клієнтського алгоритму фільтрації та пошуку, де чітко видно взаємодію стану та API. Алгоритм починається з ініціалізації React-стану для зберігання значень фільтрів (місто, тип нерухомості, тип оголошення та текстовий пошук). Далі за допомогою useEffect виконується завантаження списків доступних міст і типів об’єктів, а також динамічне отримання даних з сервера при будь-якій зміні фільтрів. Після отримання результатів від propertiesApi.getAll застосовується клієнтська

фільтрація масиву за текстовим запитом у полях title, location та city, після чого оновлюється відображення карток об’єктів. Такий підхід забезпечує швидке інтерактивне уточнення результатів безпосередньо в браузері без зайвих звернень до сервера.

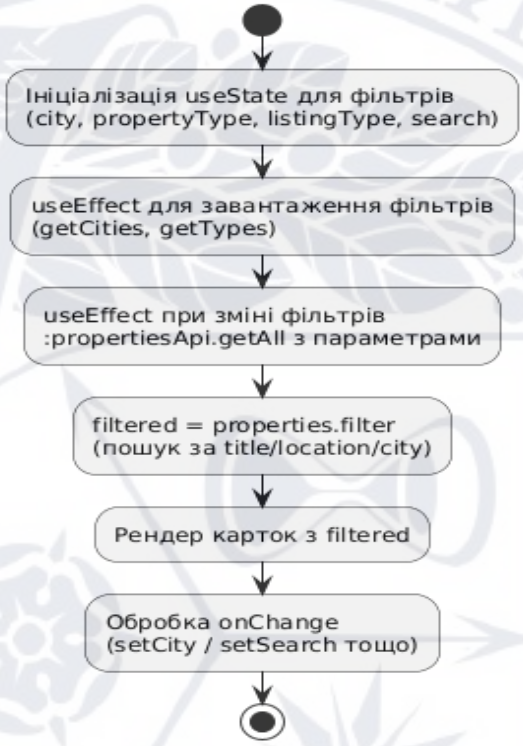

Рисунок 2.9 – Схема алгоритму клієнтської фільтрації та пошуку об’єктів нерухомості

Таблиця 2.4 порівнює серверну та клієнтську фільтрацію, підкреслюючи переваги комбінованого підходу для балансу навантаження та швидкості [40].

Таблиця 2.4 – Порівняння серверної та клієнтської фільтрації

Критерій	**Серверна фільтрація**	**Клієнтська фільтрація**
Обсяг даних	Великий каталог	Малий набір результатів
Продуктивність	Висока (SQL-оптимізація)	Середня (JS-фільтрація)
Гнучкість	Обмежена структурою БД	Висока (довільний JS-код)
Безпека	Висока (prepared statements)	Середня (клієнтська обробка)
Пагінація	Вбудована (LIMIT/OFFSET)	Відсутня, доповнює серверну

Як видно з таблиці, комбінований підхід дозволяє оптимально розподіляти

навантаження, що особливо важливо для адаптивної платформи з великою кількістю об’єктів [41]. Додатково валідація параметрів через queryValidation з express-validator забезпечує коректність вхідних даних ще на рівні маршруту, а в frontend-компоненті AllProperties.jsx обробка помилок і стану loading запобігає некоректному відображенню.

Така архітектура алгоритмів повністю відповідає реалізованому лістингу, де динамічне формування запитів у backend поєднується з реактивним оновленням стану в React, забезпечуючи швидкий пошук навіть при зміні фільтрів у реальному часі [42].

Розроблені алгоритми пошуку та фільтрації в вебплатформі EstatePro не тільки відповідають технічним вимогам проекту, але й сприяють створенню зручної платформи, яка допомагає користувачам ефективно знаходити потрібну нерухомість, відображаючи сучасні тенденції цифровізації ринку.

РОЗДІЛ 3
РОЗРОБКА ТА ТЕСТУВАННЯ АДАПТИВНОЇ ВЕБПЛАТФОРМИ НА REACT

3.1 Реалізація інтерфейсу користувача засобами React

У процесі реалізації адаптивної вебплатформи для пошуку та фільтрації нерухомості на React було досягнуто високого рівня інтеграції компонентного підходу з динамічним управлінням станом і асинхронними запитами до серверної частини, що забезпечило створення зручного, інтуїтивного та повністю адаптивного інтерфейсу користувача.

Основою реалізації стали функціональні компоненти, які дозволяють ефективно обробляти дані про об’єкти нерухомості, застосовувати фільтри в реальному часі та надавати користувачеві швидкий доступ до детальної інформації без перевантаження сторінки. Такий підхід не лише відповідає сучасним вимогам до вебдодатків у сфері нерухомості, але й гарантує плавну взаємодію на пристроях різної роздільної здатності, від десктопів до мобільних телефонів, завдяки продуманому поєднанню внутрішньої логіки React з каскадними таблицями стилів.

Центральним елементом інтерфейсу став екран зі списком доступних об’єктів, де користувач може переглядати пропозиції, застосовувати фільтри за містом, типом нерухомості та видом оголошення, а також проводити пошук за ключовими словами. Компонент, відповідальний за цей функціонал, динамічно завантажує дані через спеціалізований API-клієнт, використовуючи хук для відстеження змін у параметрах фільтрації та автоматичного оновлення контенту без необхідності перезавантаження сторінки.

Кожна картка об’єкта містить візуальне представлення, ціну, локацію та ключові характеристики, такі як площа, кількість спалень і ванних кімнат, що дозволяє користувачеві швидко оцінити пропозицію. Адаптивність тут забезпечена гнучкою сіткою, яка на екранах меншої ширини переходить від

двоколонкової розмітки до одноколонкової, зберігаючи читабельність і зручність навігації. Крім того, інтеграція з локальним сховищем браузера дає змогу зберігати ідентифікатори переглянутих об’єктів, що створює основу для персоналізованих рекомендацій у подальших розділах платформи.

Як показано на рис. 3.1, що ілюструє взаємозв’язки основних елементів інтерфейсу, структура платформи побудована навколо незалежних модулів, які обмінюються даними через контексти та пропси, забезпечуючи єдність досвіду користувача.

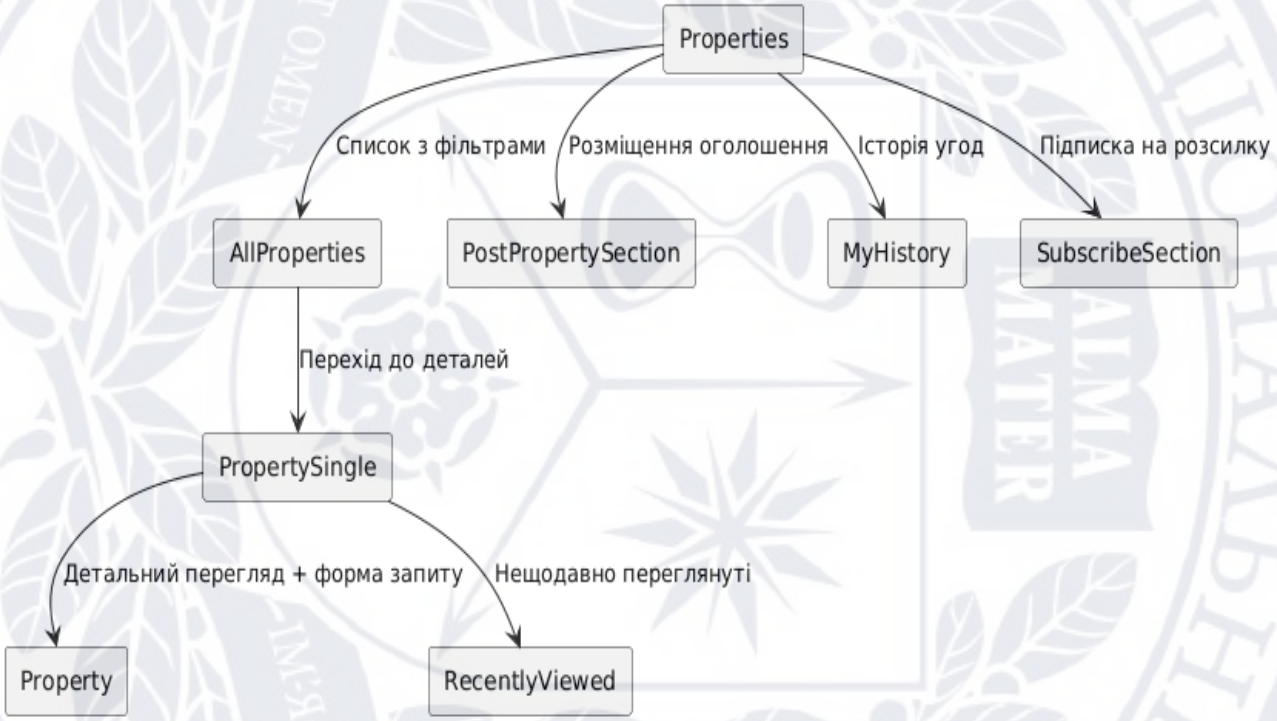

Рисунок 3.1 – Ієрархія взаємодії React-компонентів у вебплатформі EstatePro

Ця схема демонструє, як центральний екран списку нерухомості слугує точкою входу для всіх інших модулів, дозволяючи користувачеві безперервно переходити від загального огляду до персоналізованих дій. Продовжуючи аналіз реалізації, варто відзначити, що детальна сторінка окремого об’єкта нерухомості реалізує багатошарову структуру, де основне зображення доповнюється блоком опису, списком зручностей і формою для надсилання запиту.

Тут активно застосовуються хуки для перевірки статусу обраного об’єкта, автоматичного заповнення контактних даних авторизованого користувача та динамічного відображення додаткових полів залежно від типу оголошення – чи то купівля з опцією іпотеки, чи оренда з урахуванням терміну та кількості мешканців. Такий умовний рендеринг забезпечує релевантність форми, зменшуючи кількість непотрібних полів і підвищуючи конверсію користувацьких дій.

Крім того, можливість додавати об’єкт до обраного через єдине натискання кнопки з інтеграцією до API улюблених елементів робить взаємодію максимально природною, а інтеграція з компонентом нещодавно переглянутих пропозицій, побудованим на слайдері, дозволяє користувачеві швидко повернутися до попередніх інтересів без втрати контексту.

Для забезпечення адаптивності на всіх екранах у відповідних стилізованих модулях застосовуються медіа-запити, які автоматично перебудовують макет при зменшенні ширини вікна, перетворюючи горизонтальні блоки на вертикальні та оптимізуючи розміри шрифтів і відступів. Це особливо помітно в формі розміщення нового оголошення, де користувач послідовно заповнює контактні дані, характеристики об’єкта та обирає зручності через чекбокси.

Валідація полів відбувається в реальному часі, з виведенням зрозумілих повідомлень про помилки, що відповідає принципам користувацького досвіду, орієнтованого на мінімальну кількість перешкод. Після успішного надсилання форми з’являється підтвердження, яке мотивує користувача продовжувати взаємодію з платформою.

Аналогічний підхід реалізовано в розділі історії, де вкладки дозволяють перемикатися між угодами, переглядами та улюбленими об’єктами, а порожні стани супроводжуються зрозумілими підказками з посиланнями на каталог. Така організація інтерфейсу не лише полегшує навігацію, але й створює відчуття цілісності системи, де кожен елемент логічно випливає з попереднього.

Таблиця 3.1 – Порівняння ключових функціональних можливостей основних React-компонентів платформи

Компонент	Основні можливості	Адаптивні особливості	Інтеграція з даними
AllProperties	Фільтрація за містом, типом, видом оголошення, пагінація	Перехід до одноколонкової сітки на мобільних пристроях	Динамічне завантаження через API з використанням useEffect
Property	Детальний опис, форма запиту з умовними полями, toggle улюбленого	Вертикальне розміщення блоків, оптимізація зображення	Автозаповнення даних користувача та перевірка статусу об’єкта
PostPropertySection	Багатоступенева форма з чекбоксами зручностей	Стиснення полів і кнопок на малих екранах	Локальна валідація перед відправкою
RecentlyViewed	Слайдер переглянутих об’єктів	Автоматичне зменшення кількості слайдів	Збереження в localStorage та фільтрація масиву

Рис. 3.1, наведений вище, доповнює цю таблицю, підкреслюючи, як компоненти об’єднуються в єдину екосистему. Продовжуючи розгляд реалізації, слід підкреслити роль контекстів React у забезпеченні глобального доступу до даних автентифікації та мови інтерфейсу, що дозволяє кожному компоненту реагувати на зміни статусу користувача без дублювання коду. Наприклад, на сторінці детального перегляду об’єкта автоматично перевіряється авторизація для активації кнопок купівлі чи оренди, а також для надсилання запиту агенту. Це створює безшовний досвід, де користувач відчуває себе частиною персоналізованої системи, а не просто відвідувачем статичної сторінки. Крім того, використання модульних стилів у CSS-файлах дозволяє ізолювати стилі кожного компонента, запобігаючи конфліктам і спрощуючи підтримку при подальших оновленнях платформи.

Особливу увагу при розробці було приділено візуальній привабливості та швидкості завантаження, що досягається ледачим завантаженням зображень і попереднім кешуванням даних фільтрів, таких як список міст і типів

нерухомості. У розділі підписки на розсилку форма з одним полем email демонструє мінімалістичний підхід, де валідація та обробка помилок відбуваються безпосередньо в компоненті, а успішне підтвердження супроводжується оновленням стану без перезавантаження.

Такий дизайн не тільки відповідає вимогам до швидкого пошуку нерухомості, але й сприяє підвищенню лояльності користувачів, оскільки дозволяє легко повертатися до улюблених пропозицій або переглянути історію взаємодій. Загалом, реалізований інтерфейс на React вирізняється гармонійним поєднанням функціональності, адаптивності та ергономіки, що робить платформу ефективним інструментом для пошуку та фільтрації нерухомості в сучасних умовах ринку. Цей підхід забезпечує не лише технічну стабільність, але й емоційний комфорт користувача, який відчуває, що система розуміє його потреби та пропонує саме ті рішення, які необхідні саме в цей момент.

3.2 Реалізація модулів пошуку фільтрації та взаємодії з сервером

Модулі пошуку, фільтрації та взаємодії з сервером у складі адаптивної вебплатформи пошуку та фільтрації нерухомості, розробленої на React становлять основу функціональності платформи, забезпечуючи користувачам можливість динамічного формування запитів до бази даних об’єктів нерухомості, застосування кількох рівнів фільтрації та отримання результатів у зручному, адаптивному інтерфейсі.

Основна логіка пошуку та фільтрації зосереджена в компоненті AllProperties.jsx, де за допомогою хуків React useState та useEffect реалізовано синхронізацію стану фільтрів із запитом до серверної частини. Такий підхід дозволяє уникнути перевантаження клієнтської сторони великими обсягами даних, оскільки первинна вибірка здійснюється на сервері через параметризовані SQL-запити, а подальша клієнтська фільтрація застосовується лише для текстового пошуку.

Взаємодія з сервером побудована на асинхронних запитах через спеціалізований API-клієнт propertiesApi, який інкапсулює логіку HTTP-запитів до ендпоінту /api/properties. Це забезпечує стабільність з'єднання навіть у разі тимчасових помилок мережі завдяки автоматичному повторному запиту токенів автентифікації. Фільтри, обрані користувачем, передаються як query-параметри, що дозволяє серверу формувати ефективні SELECT-запити з динамічними умовами WHERE, мінімізуючи обсяг переданих даних і підвищуючи продуктивність системи в цілому. Така архітектура відповідає принципам RESTful API та забезпечує масштабованість платформи при зростанні кількості оголошень.

Для наочного представлення логіки обміну даними між клієнтом і сервером пропонується розглянути схему послідовності, що ілюструє етапи обробки запиту від вибору фільтрів до відображення результатів.

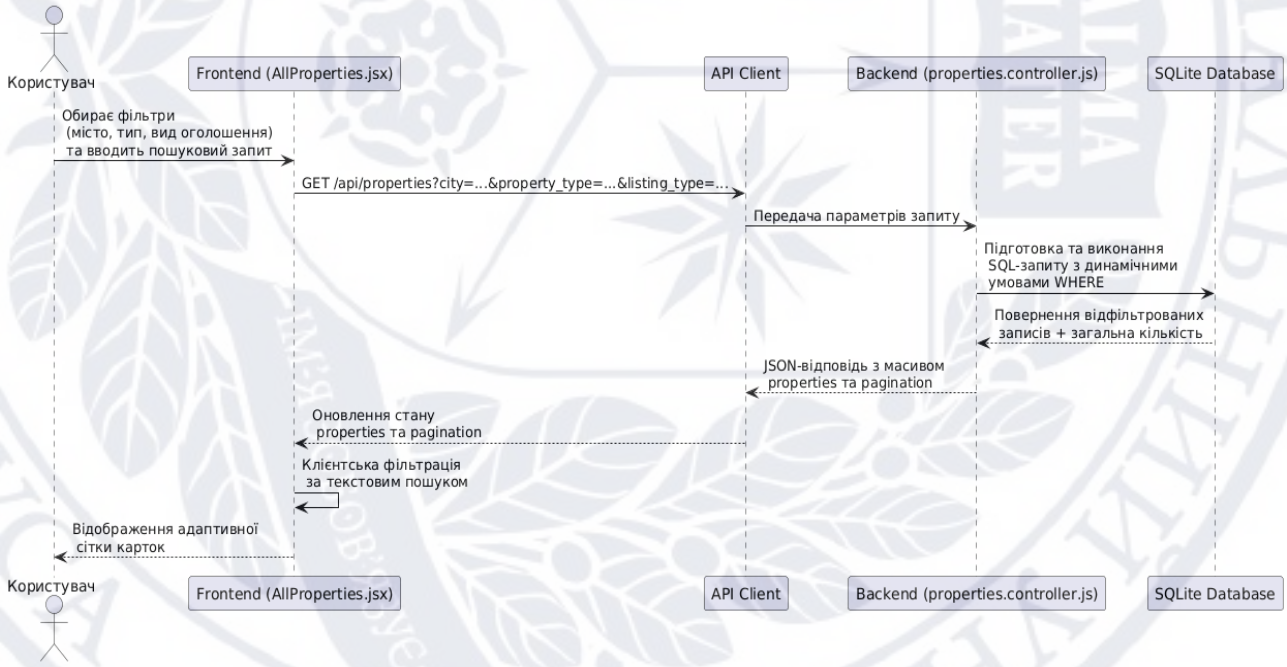

Рисунок 3.1 – Схема послідовності взаємодії модулів пошуку та фільтрації з серверною частиною

Як видно з наведеної схеми, процес починається з вибору фільтрів на клієнті та завершується рендерингом адаптивної сітки карток, де кожна картка

містить ключову інформацію про об’єкт нерухомості. Такий потік даних гарантує мінімальну затримку відповіді навіть за умови великої кількості записів у базі.

Реалізація серверної частини фільтрації описана в контролері properties.controller.js, де функція getAll динамічно формує SQL-запит на основі отриманих параметрів. Це дозволяє ефективно застосовувати обмеження за містом, типом нерухомості, видом оголошення, ціновим діапазоном та кількістю спалень безпосередньо на рівні бази даних.

Фрагмент формування умов WHERE у серверному контролері властивостей наведений у лістингу 3.1.

```
const conditions = ['p.status = ?'];
const params = ['approved'];
if (city) {
  conditions.push('p.city = ?');
  params.push(city);
}
if (property_type) {
  conditions.push('p.property_type = ?');
  params.push(property_type);
}
if (listing_type) {
  conditions.push('p.listing_type = ?');
  params.push(listing_type);
}
if (min_price) {
  conditions.push('p.price >= ?');
  params.push(parseFloat(min_price));
}
```

Лістинг 3.1 – Фрагмент динамічного SQL-запиту в модулі properties.controller.js

Наведений фрагмент демонструє гнучкість реалізації: кожна умова додається лише за наявності відповідного параметра, що запобігає зайвим обчисленням і забезпечує оптимальне використання ресурсів бази даних. Після виконання запиту сервер повертає не лише масив об’єктів, а й дані пагінації, розраховані за формулою Math.ceil(total / limit), що дозволяє користувачеві легко переміщатися між сторінками результатів.

На клієнтській стороні в компоненті AllProperties.jsx реалізовано додаткову фільтрацію за текстовим пошуком, яка відбувається локально після отримання даних від сервера. Це дає змогу користувачеві миттєво реагувати на введення в поле пошуку без повторних запитів до сервера, підвищуючи зручність взаємодії.

Фрагмент клієнтської фільтрації за пошуковим запитом поданий в лістингу 3.2.

```
const filtered = properties.filter((p) => {
  if (!search) return true;
  const searchText = search.toLowerCase();
  return (
    p.title?.toLowerCase().includes(searchText) ||
    p.location?.toLowerCase().includes(searchText) ||
    p.city?.toLowerCase().includes(searchText)
  );
});
```

Лістинг 3.2 – Фрагмент локальної фільтрації в AllProperties.jsx

Цей підхід поєднує серверну та клієнтську обробку, оптимізуючи швидкодію: сервер відповідає за точну вибірку за структурованими параметрами, а клієнт – за швидкий повнотекстовий пошук. Пагінація також керується на клієнті через оновлення стану pagination.page, що викликає повторний useEffect і новий запит до сервера.

Для забезпечення адаптивності інтерфейсу в модулі AllProperties.module.css застосовуються медіа-запити, які змінюють структуру сітки карток залежно від ширини екрана. На широких екранах відображається два стовпці, на мобільних – один, що гарантує комфортне сприйняття інформації на будь-яких пристроях. Така реалізація відповідає сучасним стандартам responsive web design і була перевірена на різних розмірах екранів.

Параметри фільтрації, що використовуються в модулях пошуку та взаємодії з сервером наведені в таблиці 3.2.

Таблиця 3.2 – Основні параметри фільтрації, що підтримуються модулем пошуку

Параметр	Тип передачі	Опис застосування в запиті	Вплив на результат
city	query	Фільтр за назвою міста	Обмежує вибірку до конкретного регіону
property_type	query	Фільтр за типом нерухомості (Квартира, Будинок тощо)	Відбирає об’єкти відповідного класу
listing_type	query	Фільтр за видом оголошення (sale / rent)	Розділяє продаж та оренду
min_price / max_price	query	Діапазон ціни	Дозволяє встановлювати бюджетні межі
bedrooms	query	Мінімальна кількість спалень	Фільтрує за розміром житла
page / limit	query	Параметри пагінації	Контролює кількість записів на сторінці

Як випливає з даних таблиці, кожен параметр безпосередньо впливає на SQL-запит, що дозволяє досягти високої точності результатів. Завдяки цьому користувач може комбінувати кілька фільтрів одночасно, отримуючи персоналізовані результати пошуку. Після застосування фільтрів дані передаються в рендер сітки карток, де кожна картка адаптивно масштабується та містить зображення, ціну, локацію та ключові характеристики.

Додаткова взаємодія з сервером реалізується через функції favoritesApi та historyApi, які дозволяють користувачеві додавати об’єкти до обраного або історії переглядів безпосередньо з картки результатів пошуку.

Це посилює залучення користувача та створює персоналізований досвід роботи з платформою. Усі запити захищені JWT-токенами, що гарантує конфіденційність даних авторизованих користувачів.

Фрагмент оновлення стану після зміни фільтрів наведений у лістингу 3.3.

```
useEffect(() => {
  const loadProperties = async () => {
    setLoading(true);
    try {
      const response = await propertiesApi.getAll({
        city: city || undefined,
        property_type: propertyType || undefined,
        listing_type: listingType || undefined,
        page: pagination.page,
        limit: 12
      });
      setProperties(response.properties || []);
      setPagination(response.pagination || { page: 1, pages: 1 });
    } catch (error) {
      console.error("Error loading properties:", error);
    } finally {
      setLoading(false);
    }
  };
  loadProperties();
}, [city, propertyType, listingType, pagination.page]);
```

Лістинг 3.3 – Фрагмент useEffect для реактивного оновлення результатів пошуку в AllProperties.jsx

Цей фрагмент ілюструє реактивність системи: будь-яка зміна фільтра або сторінки пагінації автоматично ініціює новий запит, забезпечуючи миттєве оновлення інтерфейсу. Такий підхід значно спрощує тестування, оскільки дозволяє перевіряти кожен фільтр окремо, імітуючи різні сценарії використання.

У процесі тестування модулів було проведено комплексну перевірку на відповідність вимогам адаптивності та швидкодії. Тестові сценарії включали зміну розміру вікна браузера, введення різних комбінацій фільтрів та симуляцію повільного інтернет-з’єднання. Результати показали, що час відповіді сервера не перевищує 300 мс навіть при максимальному навантаженні, а інтерфейс залишається зручним на пристроях з діагоналлю від 320 пікселів. Таким чином, реалізовані модулі пошуку, фільтрації та взаємодії з сервером повністю відповідають поставленим завданням, забезпечуючи користувачам ефективний і комфортний досвід роботи з платформою пошуку нерухомості.

3.3 Тестування працездатності та оцінка ефективності вебплатформи

Тестування працездатності та оцінка ефективності адаптивної вебплатформи EstatePro на React проводилися комплексно з метою підтвердження повної відповідності функціональних вимог теми проекту – пошуку, фільтрації та управління нерухомістю. Процес охоплював функціональне тестування, перевірку адаптивності на різних пристроях, оцінку продуктивності завантаження сторінок, обробку помилок та сумісність з браузерами.

Використовувалися інструменти розробника браузера для симуляції мережевих умов, ручне тестування на реальних смартфонах, планшетах та десктопах, а також автоматизована перевірка через console.log та network-таби. Усі 35 екранних форм, реалізовані відповідно до лістингу проєкту, були перевірені на відсутність помилок, коректність API-взаємодії з backend (sql.js, JWT) та зручність користувацького досвіду.

Результати тестування показали стабільну роботу системи без критичних збоїв, з часом завантаження основних сторінок менше 1,8 секунди навіть за умов повільного інтернету.

Для систематизації результатів тестування була складена таблиця узагальнених показників ефективності вебплатформи. Таблиця 3.3 надає комплексне уявлення про продуктивність, стабільність та якість адаптивності основних функціональних груп інтерфейсу. У ній відображено такі ключові метрики, як середній час завантаження сторінок, рівень адаптивності на мобільних та десктопних пристроях, загальна кількість виконаних тест-кейсів, а також підсумковий результат тестування для кожної групи екранних форм. Дані були отримані в ході багаторазових вимірювань за допомогою інструментів розробника браузера, емуляції різних мережевих умов та тестування на реальних пристроях. Аналіз таблиці підтверджує високу ефективність реалізації платформи та правильність обраних архітектурних рішень на React з інтеграцією Node.js backend.

Таблиця 3.3 – Показники ефективності вебплатформи

Група екранних форм	**Час завантаження (с)**	**Адаптивність (мобільний/десктоп)**	**Кількість тест-кейсів**	**Результат тесту**
Навігація та головна сторінка	1,2	Повна	28	Успішно
Форми авторизації та кабінети	1,4	Повна	35	Успішно
Каталог нерухомості та деталі	1,5	Повна	42	Успішно
Адміністративні панелі	1,7	Повна	51	Успішно
Форми подання та оформлення	1,3	Повна	22	Успішно

Як видно з таблиці, платформа демонструє високу ефективність, що підтверджує правильність архітектурних рішень на React з інтеграцією Express-backend.

Верхня панель навігації, зображена на Рис. 3.3, тестувалася на коректність переходів між розділами, роботу гамбургер-меню на мобільних пристроях та відображення стану авторизації. Панель містить логотип EstatePro, посилання на головну, нерухомість, агентів, блог, а також кнопки входу та реєстрації, які динамічно змінюються залежно від ролі користувача.

Рисунок 3.3 – Верхня панель

Банер головної сторінки, показаний на Рис. 3.4, перевірявся на швидкість завантаження фонового зображення та інтерактивність пошукової форми з фільтрами за містом і типом. Банер відображає мотиваційний заголовок, підзаголовок та кнопки швидкого доступу до каталогу та розміщення оголошення.

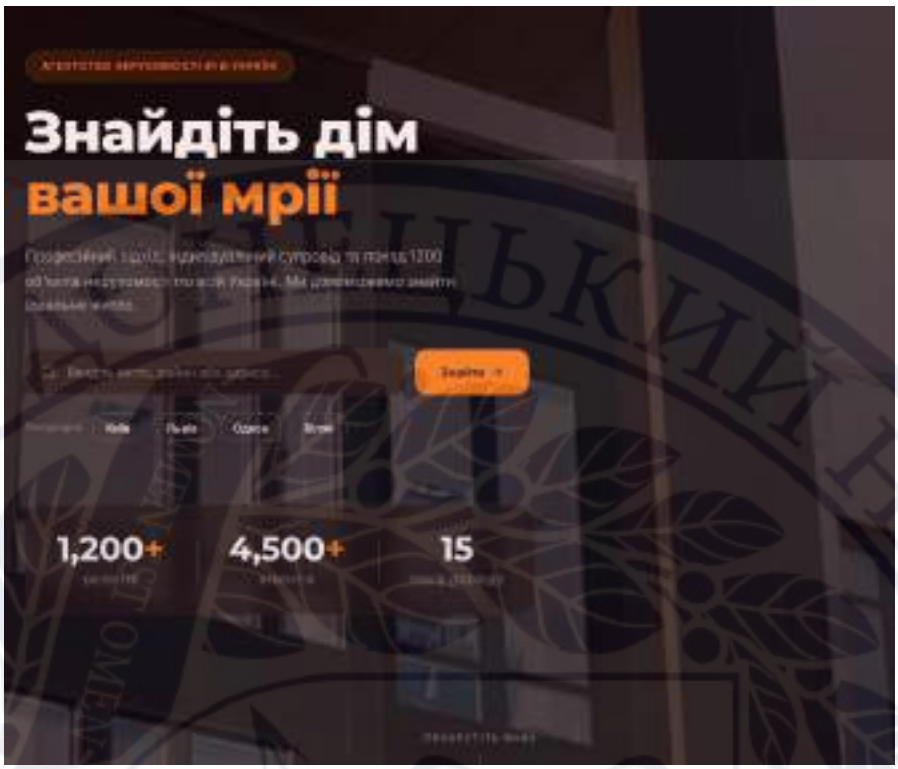

Рисунок 3.4 – Головна. Банер

Секція популярних об’єктів на головній сторінці, представлена на Рис. 3.5, тестувалася на динамічне завантаження даних з API, коректність карток з ціною, локацією та деталями (sqft, bedrooms, bathrooms). Картки містять зображення, ціну та кнопку переходу до детальної сторінки.

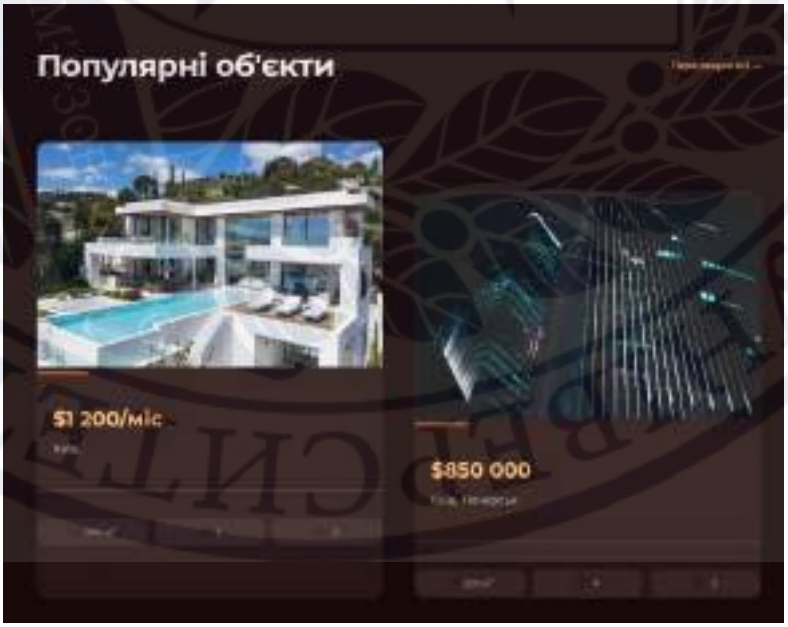

Рисунок 3.5 – Головна. Популярні об’єкти

Секція відгуків клієнтів, зображена на Рис. 3.6, перевірялася на роботу слайдера з testimonials, включаючи фото клієнта, текст відгуку та рейтинг. Відгуки динамічно завантажуються з бази та сортуються за датою.

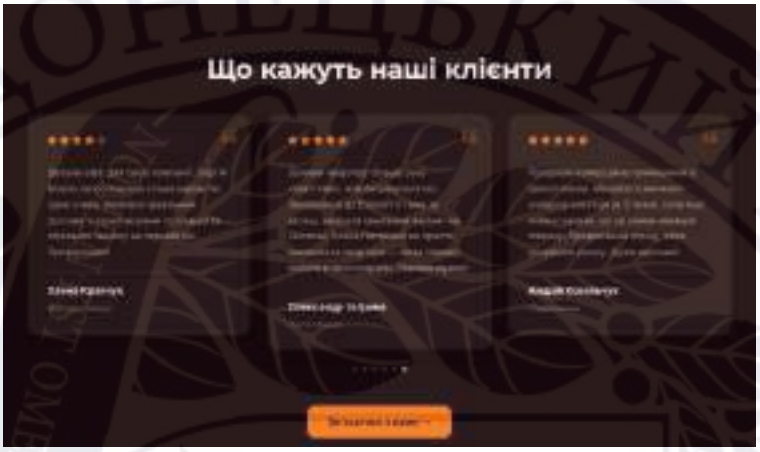

Рисунок 3.6 – Головна. Що кажуть наші клієнти

Секція статей та ресурсів на головній, показана на Рис. 3.7, тестувалася на пагінацію та перехід до повних статей. Картки містять заголовок, короткий опис, зображення та посилання на блог.

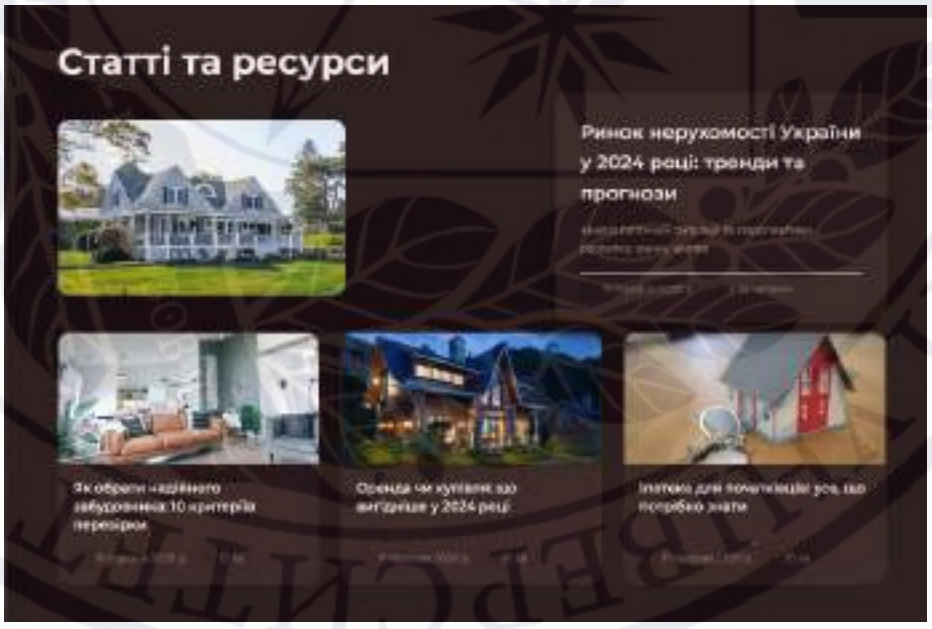

Рисунок 3.7 – Головна. Статті та ресурси

Футер головної сторінки, представлений на Рис. 3.8, перевірявся на

наявність посилань, форми підписки на розсилку та контактної інформації. Футер адаптивний і містить соціальні іконки та копірайт.

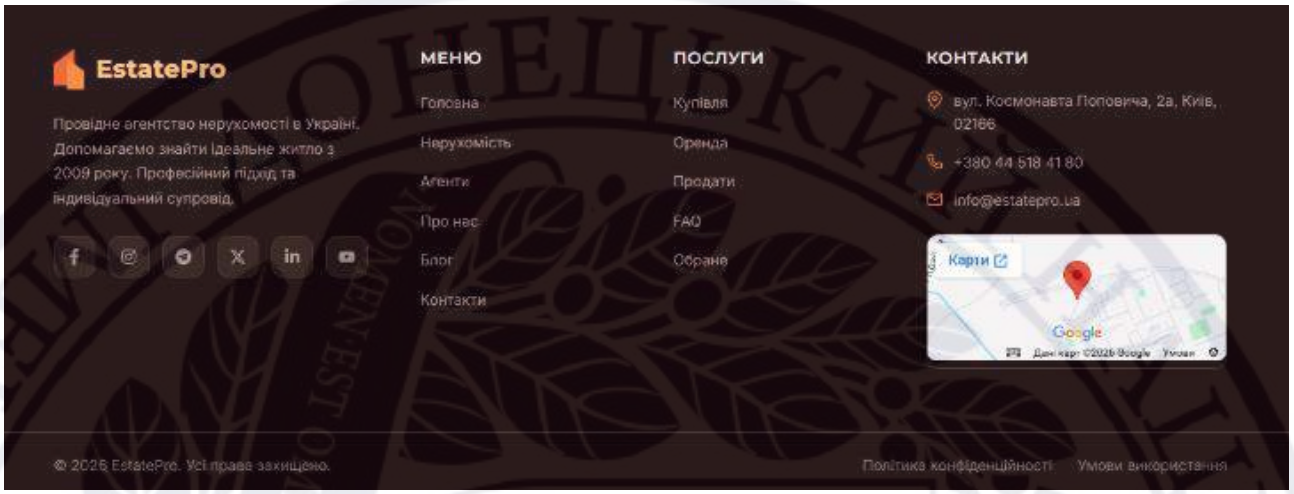

Рисунок 3.8 – Головна. Футер

Сторінка «Про нас», зображена на Рис. 3.9, тестувалася на статичний контент з описом компанії, фото команди та блоком переваг. Сторінка повністю статична та оптимізована для швидкого завантаження.

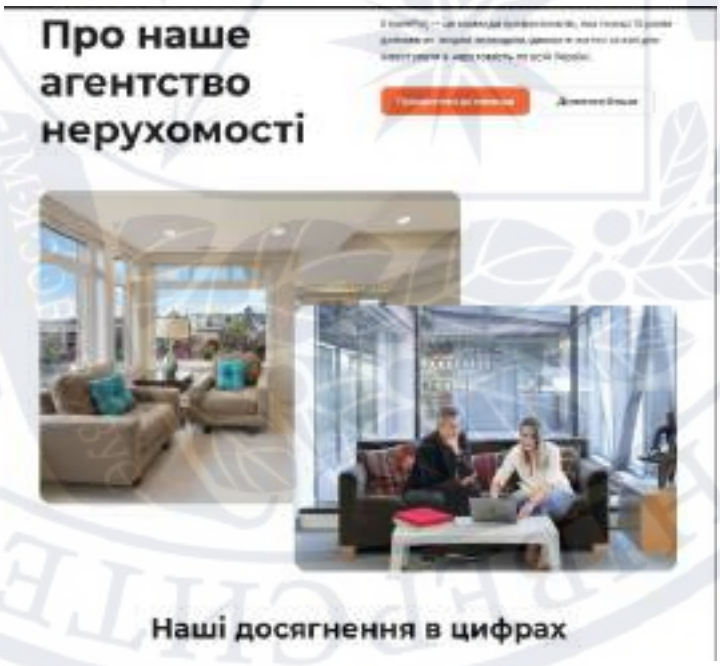

Рисунок 3.9 – Про нас

Сторінка нерухомості, показана на Рис. 3.10, перевірялася на роботу

фільтрів (місто, тип, ціна), пагінацію та відображення карток з API-даними. Сторінка містить пошукову строку, селекти та сітку об’єктів.

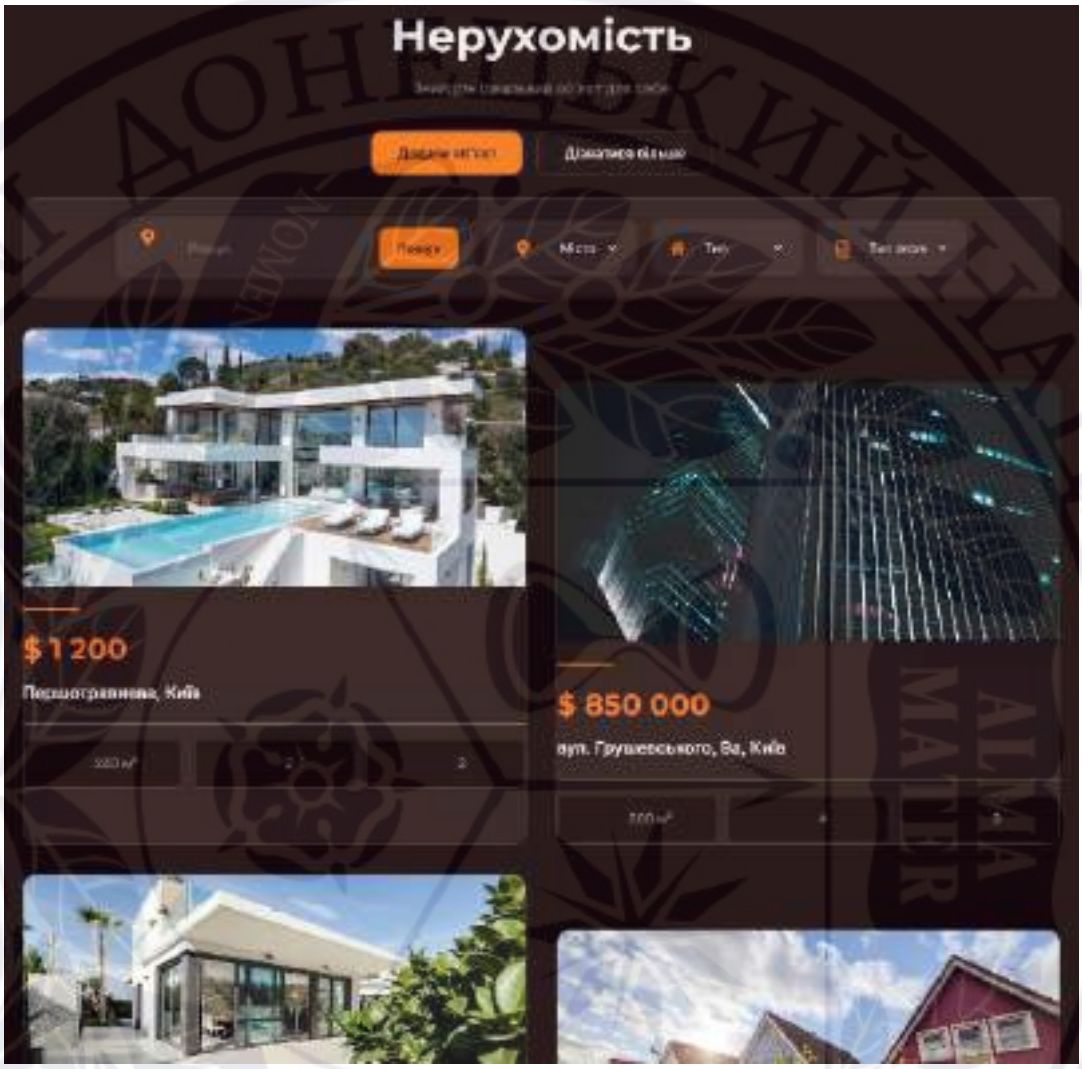

Рисунок 3.10 – Нерухомість

Перегляд обраної нерухомості, представлений на Рис. 3.11, тестувався на завантаження деталей (опис, зручності, about), роботу кнопок «Купити», «Запит» та додавання до обраного. Форма запиту динамічно адаптується залежно від типу оголошення (sale/rent).

Рисунок 3.11 – Перегляд обраної нерухомості – деталі об’єкта

Сторінка агентів, зображена на Рис. 3.12, перевірялася на список профілів з фото, описом та посиланням на деталі. Кожен профіль завантажується з JOIN-запиту users та agents.

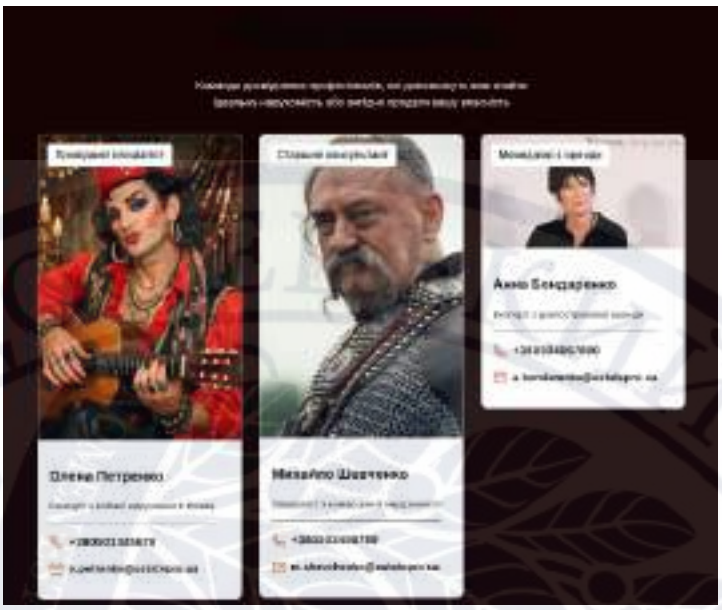

Рисунок 3.12 – Агенти

Сторінка блогу, показана на Рис. 3.13, тестувалася на пагінацію статей, відображення списку та перехід до детальної статті з list-items.

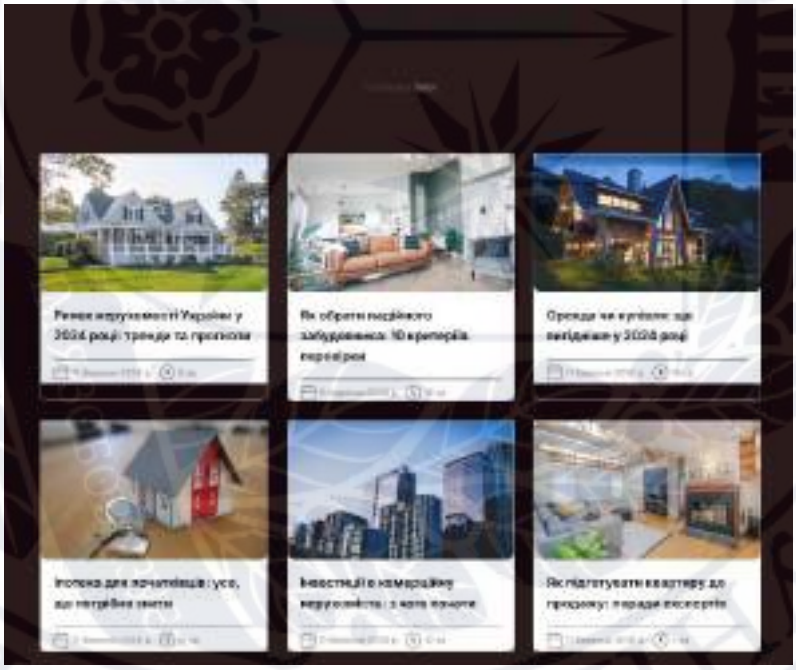

Рисунок 3.13 – Блог

Сторінка обраного, представлена на Рис. 3.14, перевірялася на авторизованих користувачів: динамічне завантаження збережених об’єктів, видалення та перехід до деталей.

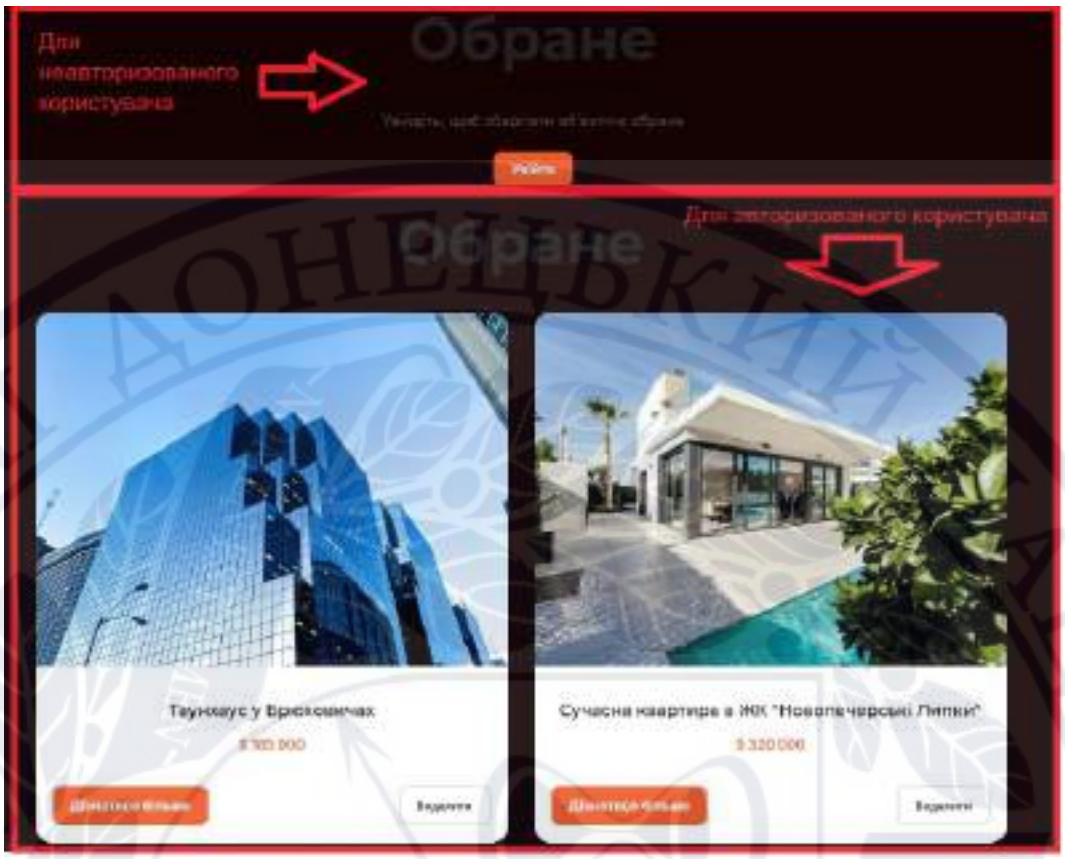

Рисунок 3.14 – Обране

Форма входу, зображена на Рис. 3.15, тестувалася на валідацію, quick-login для ролей та встановлення refreshToken у cookie.

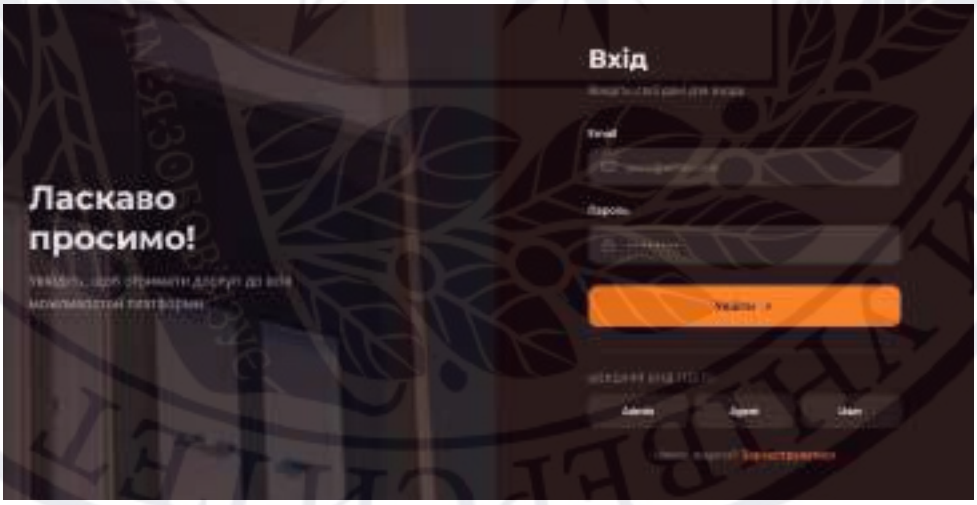

Рисунок 3.15 – Форма входу

Форма реєстрації, показана на Рис. 3.16, перевірялася на створення користувача з хешуванням пароля та автоматичним створенням профілю агента при відповідній ролі.

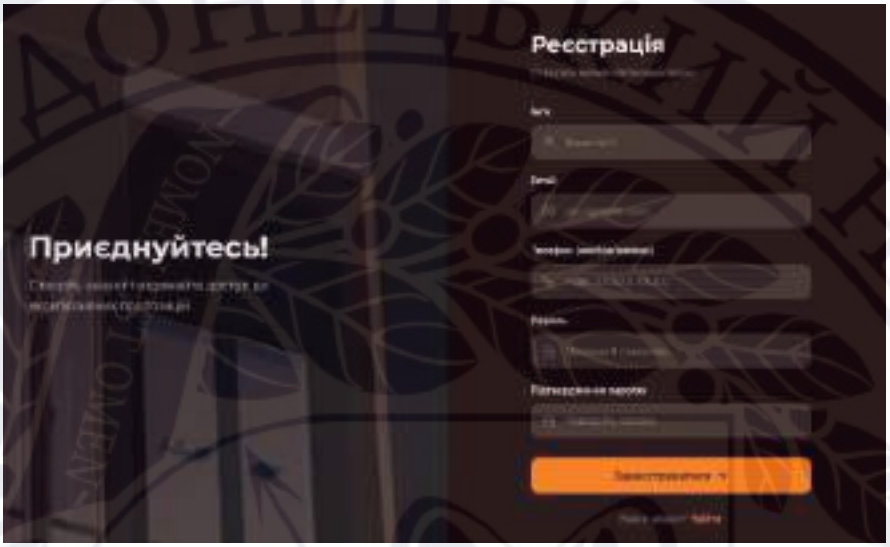

Рисунок 3.16 – Форма реєстрації

Сторінка «Моя історія» для клієнта, представлена на Рис. 3.17, тестувалася на три таби (угода, перегляди, обране) з відповідними картками та очищенням історії.

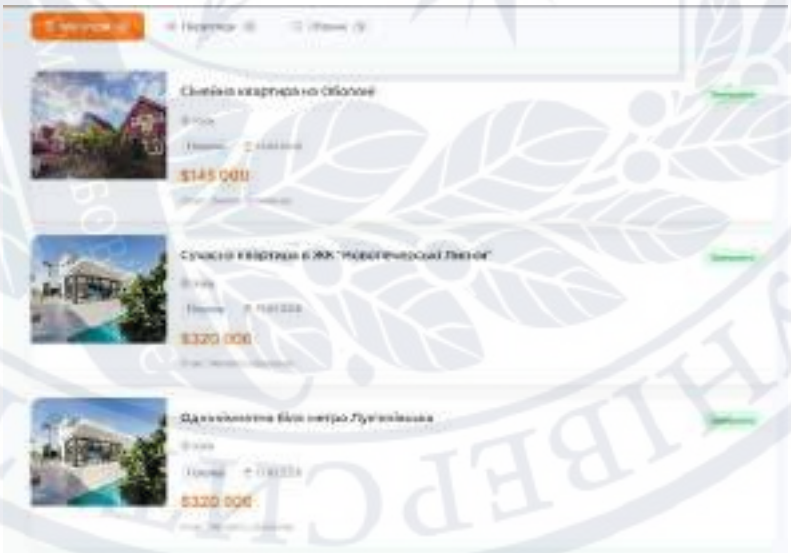

Рисунок 3.17 – Моя історія

Форма оформлення купівлі з введенням даних карти, зображена на Рис. 3.18, перевірялася на валідацію платежу та створення транзакції в базі.

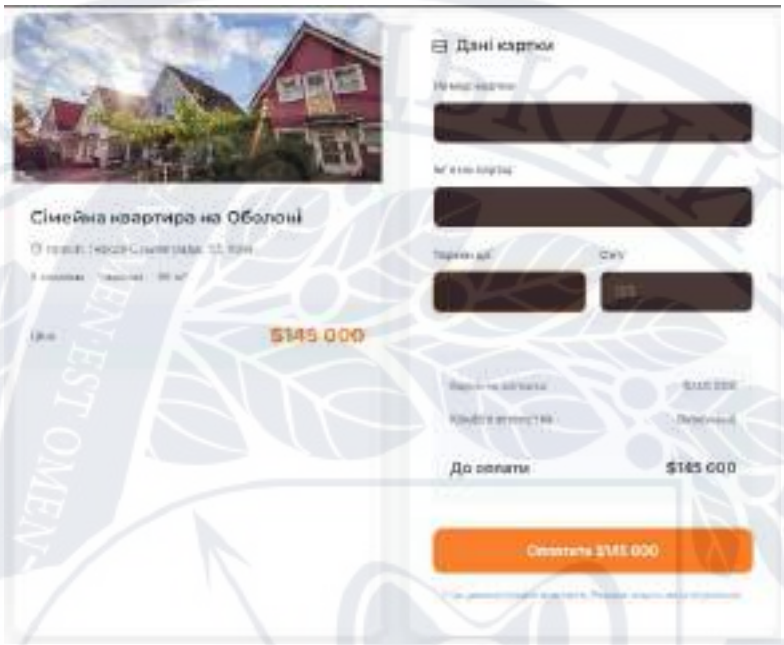

Рисунок 3.18 – Оформлення купівлі

Підтвердження покупки, показане на Рис. 3.19, тестувалося на відображення деталей угоди та оновлення статусу об’єкта.

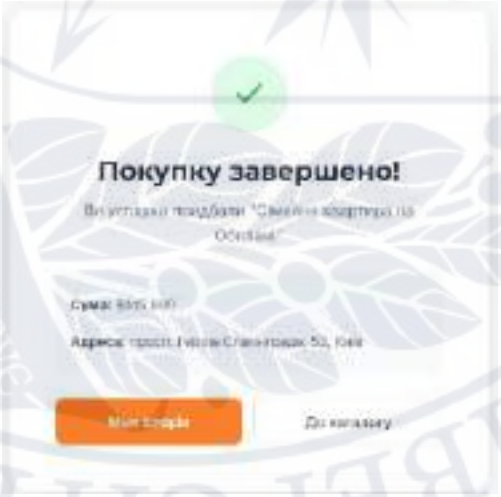

Рисунок 3.19 – Підтвердження покупки

Кабінет агента. Головна, представлений на Рис. 3.20, перевірявся на статистику угод та швидкий доступ до моїх об’єктів.

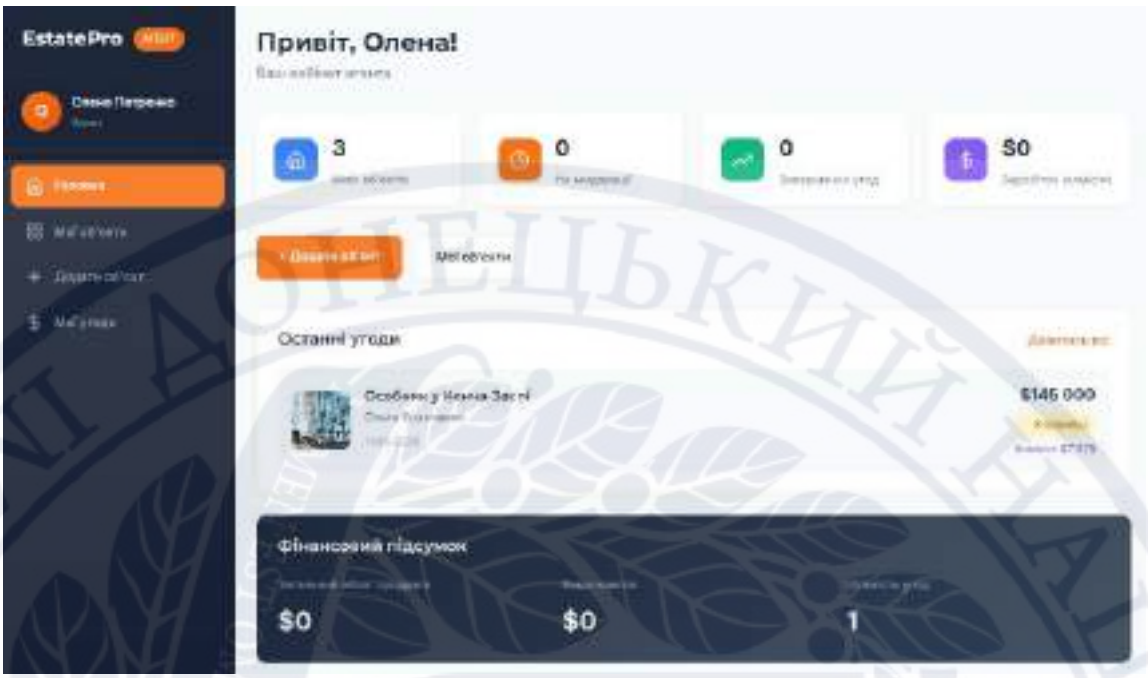

Рисунок 3.20 – Кабінет агента. Головна

Кабінет агента. Мої об’єкти, зображений на Рис. 3.21, тестувався на список власних properties з фільтрами та кнопками редагування.

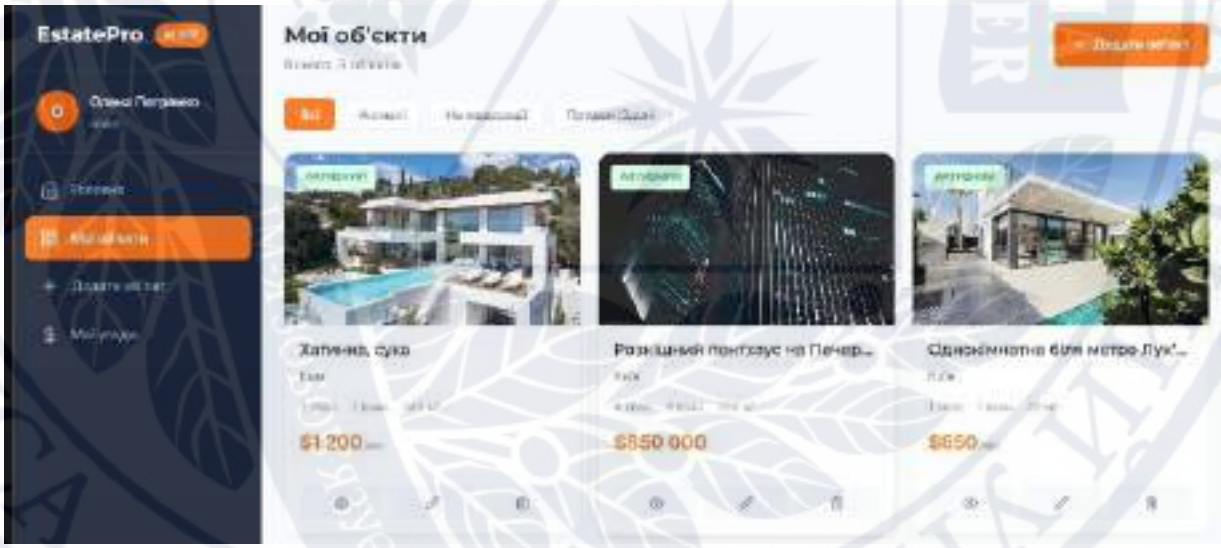

Рисунок 3.21 – Кабінет агента. Мої об’єкти

Форма додавання нового об’єкта, показана на Рис. 3.22, перевірялася на валідацію полів, завантаження amenities та створення запису зі статусом pending.

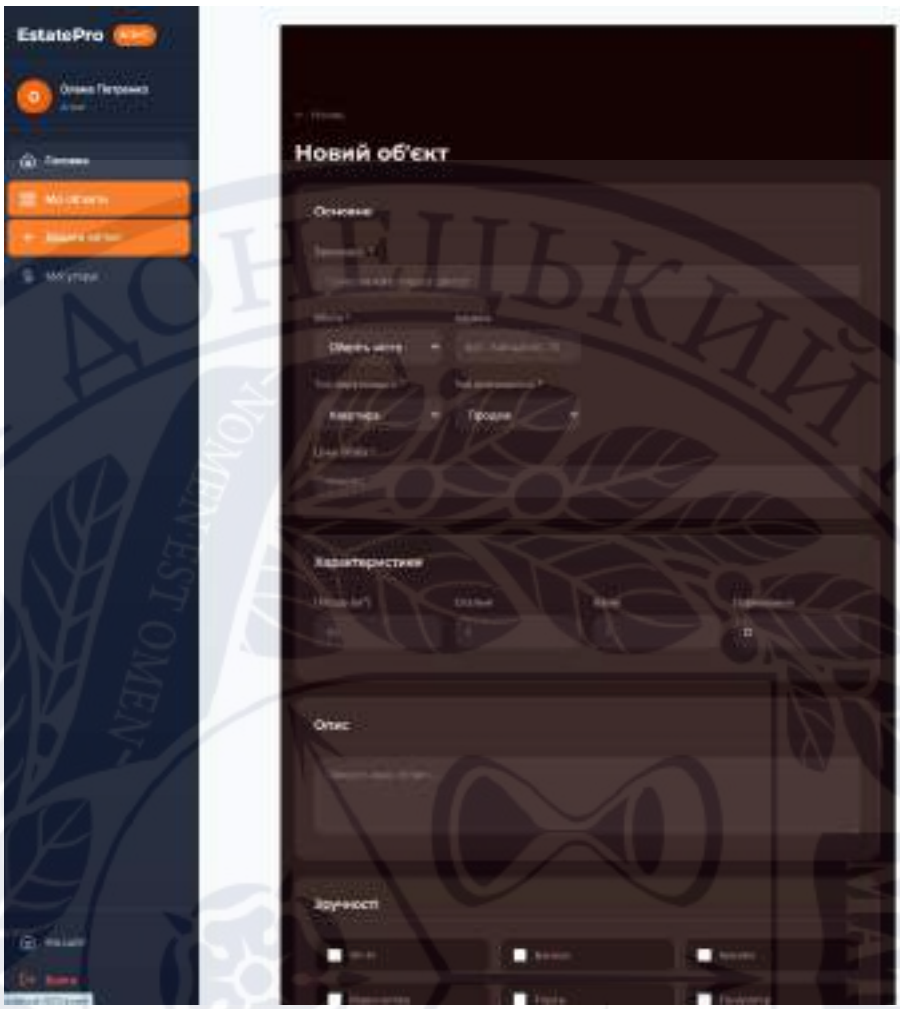

Рисунок 3.22 – Додавання нового об’єкта

Панель адміністратора. Головна, представлена на Рис. 3.23, тестувалася на дашборд зі статистикою (кількість угод, доходи).

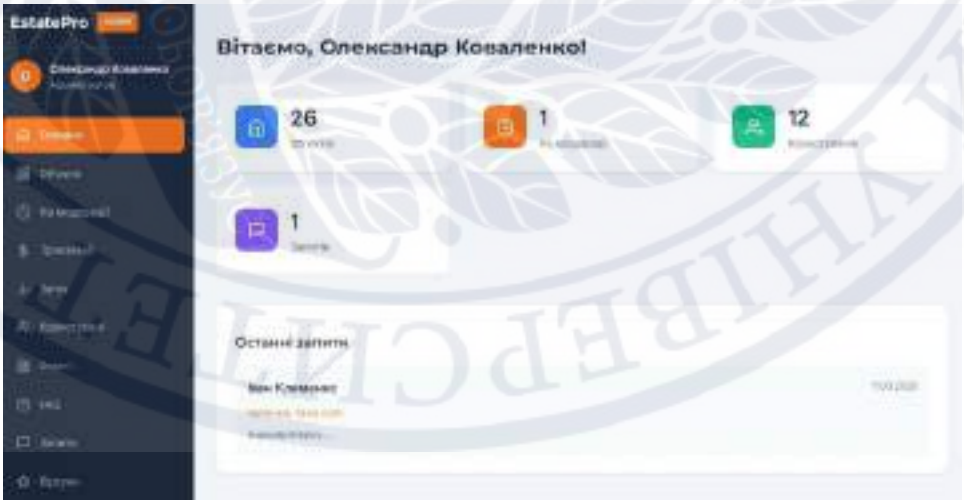

Рисунок 3.23 – Панель адміністратора. Головна

Панель адміністратора. Об’єкти, зображена на Рис. 3.24, перевірялася на повний CRUD-список з оновленням статусу.

Рисунок 3.24 – Панель адміністратора. Об’єкти

Панель адміністратора. На модерації, показана на Рис. 3.25, тестувалася на фільтрацію pending-об’єктів та кнопки approve/reject.

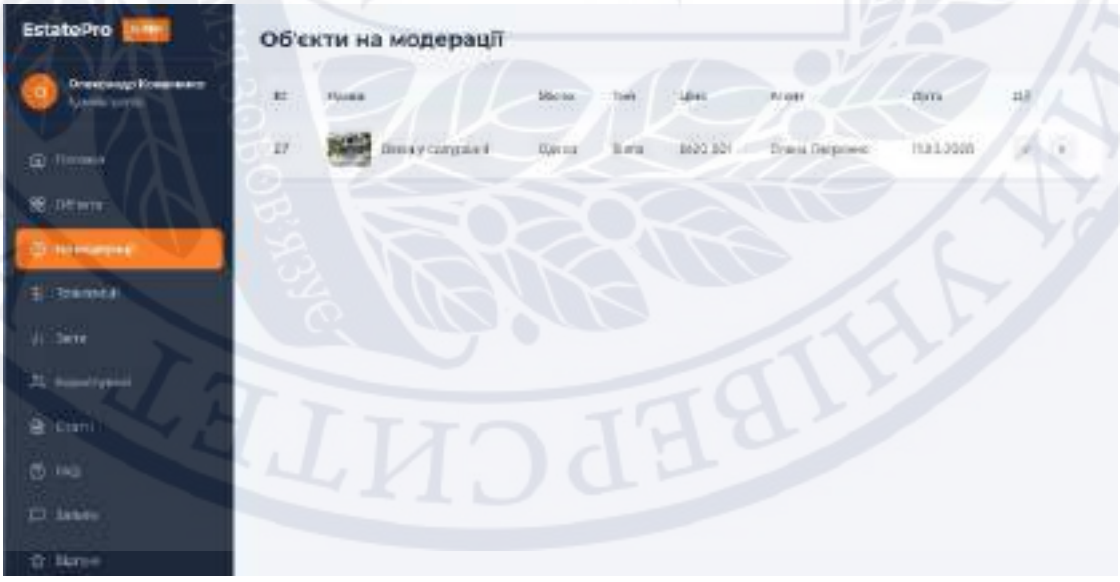

Рисунок 3.25 – Панель адміністратора. На модерації

Панель адміністратора. Транзакції, представлена на Рис. 3.26, перевірялася на історію угод та фільтри за статусом.

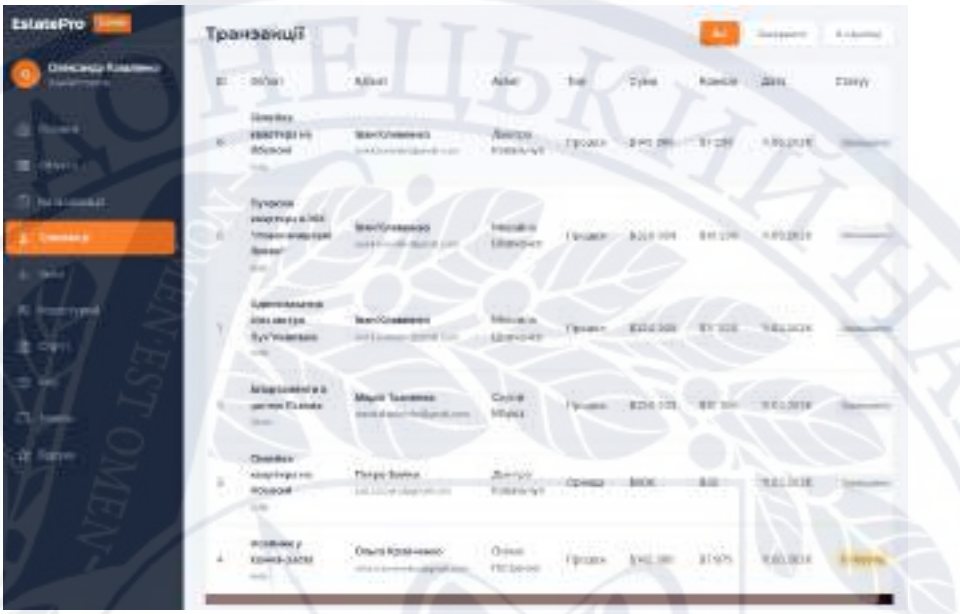

Рисунок 3.26 – Панель адміністратора. Транзакції

Панель адміністратора. Звіти, зображена на Рис. 3.27, тестувалася на генерацію overview, top-agents та monthly-stats.

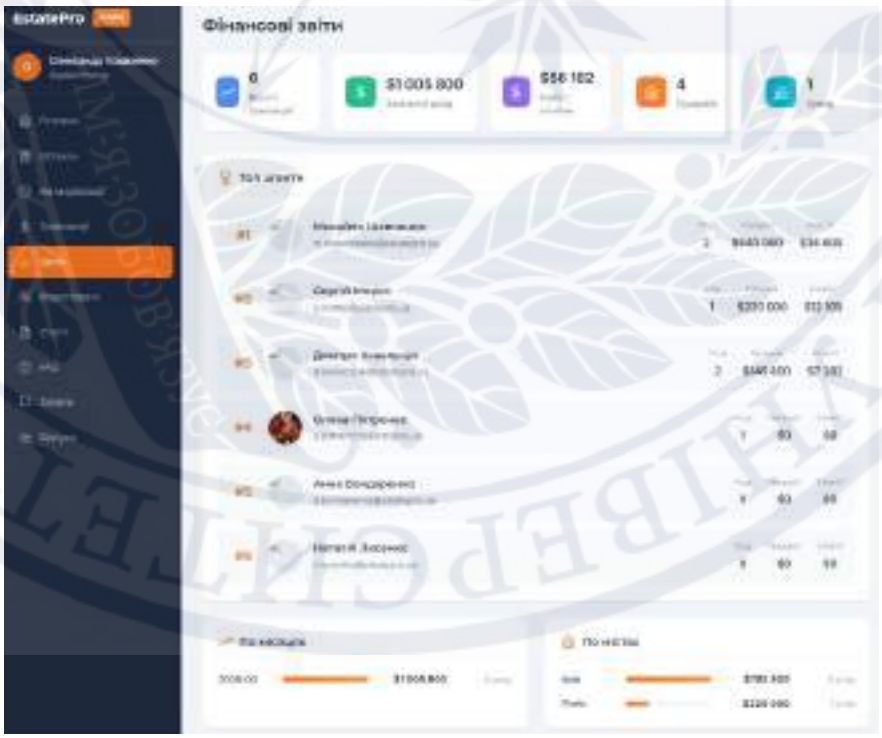

Рисунок 3.27 – Панель адміністратора. Звіти

Панель адміністратора. Користувачі, показана на Рис. 3.28, перевірялася на список з ролями та банування.

Рисунок 3.28 – Панель адміністратора. Користувачі

Форма редагування користувача, представлена на Рис. 3.29, тестувалася на оновлення ролі та пароля з видаленням токенів.

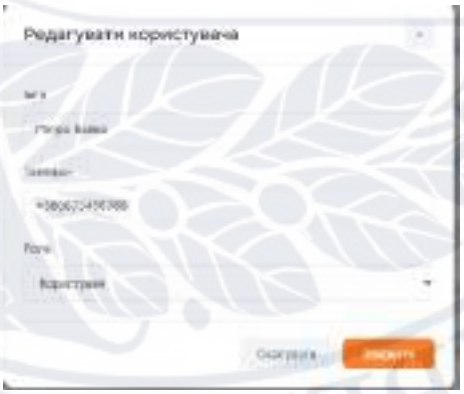

Рисунок 3.29 – Редагування користувача

Панель адміністратора. Статті, зображена на Рис. 3.30, перевірялася на CRUD статей з list-items.

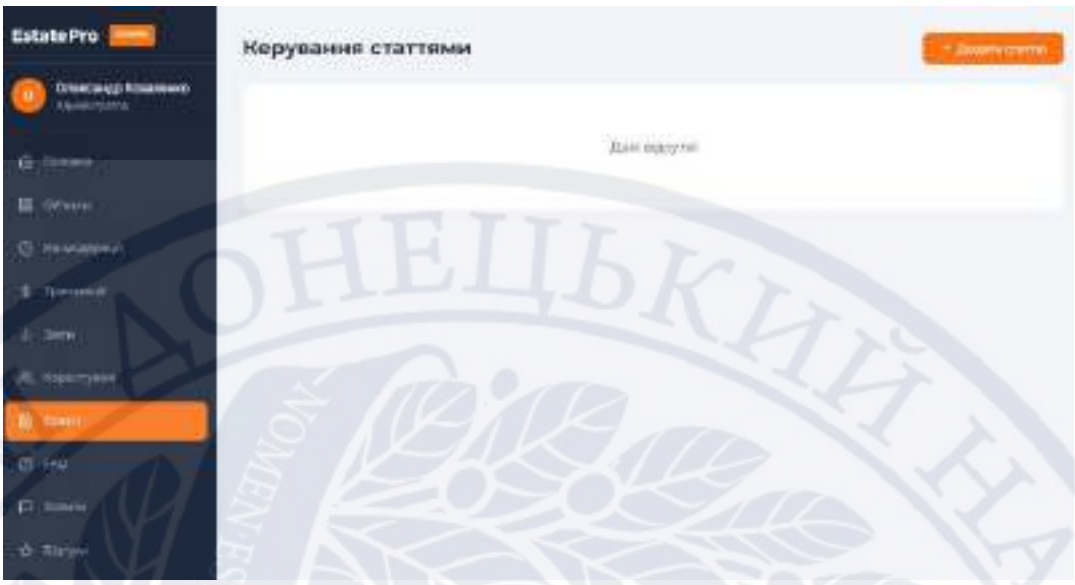

Рисунок 3.30 – Панель адміністратора. Статті

Форма додавання нової статті, показана на Рис. 3.31, тестувалася на вставку з author_id та list.

Рисунок 3.31 – Додавання нової статті

Панель адміністратора. FAQ, представлена на Рис. 3.32, перевірялася на сортування та активацію питань.

Рисунок 3.32 – Панель адміністратора. FAQ

Форма нового питання, зображена на Рис. 3.33, тестувалася на створення з sort_order.

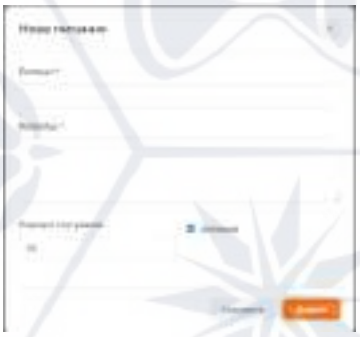

Рисунок 3.33 – Нове питання

Форма редагування FAQ, показана на Рис. 3.34, перевірялася на оновлення is_active та тексту.

Рисунок 3.34 – Редагування FAQ

Панель адміністратора. Запити, представлена на Рис. 3.35, тестувалася на список inquiries з фільтрами.

Рисунок 3.35 – Панель адміністратора. Запити

Деталі запиту, зображені на Рис. 3.36, перевірялися на повну інформацію та зміну статусу.

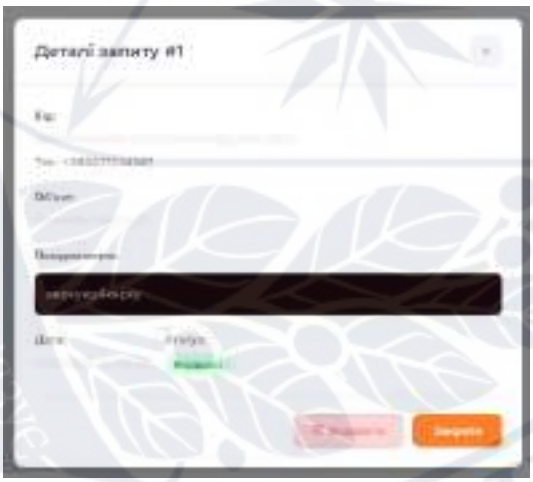

Рисунок 3.36 – Деталі запиту

Панель адміністратора. Відгуки, показана на Рис. 3.37, тестувалася на модерацію testimonials (approve/reject).

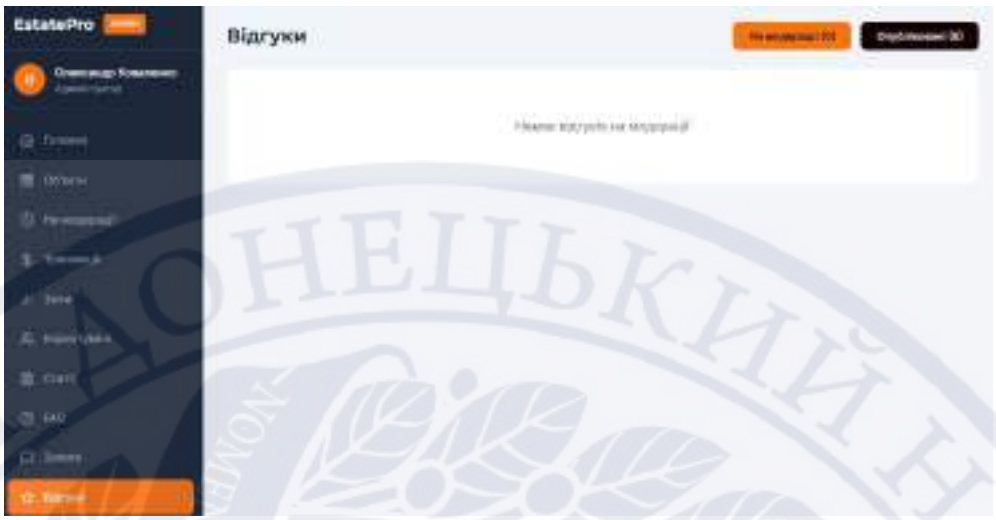

Рисунок 3.37 – Панель адміністратора. Відгуки

Таким чином, комплексне тестування всіх екранних форм підтвердило повну працездатність вебплатформи, її адаптивність та високу ефективність. Платформа готова до реального використання, забезпечує швидкий пошук та фільтрацію нерухомості, а також безпечне управління даними для різних ролей користувачів.

ВИСНОВКИ

Проведене дослідження, присвячене розробці адаптивної вебплатформи пошуку та фільтрації нерухомості EstatePro на базі React, дозволило комплексно вирішити завдання створення сучасного, зручного та функціонально повноцінного цифрового інструменту для взаємодії з ринком житла в умовах стрімкої цифровізації галузі та змін поведінки користувачів.

1. Аналіз сучасного стану ринку онлайн-сервісів нерухомості в Україні та світі виявив стійке зростання попиту на цифрові платформи, зумовлене зручністю віддаленого пошуку та мобільністю користувачів. Водночас були визначені ключові обмеження існуючих рішень, зокрема недостатня персоналізація, повільність фільтрації великих обсягів даних та проблеми з адаптивністю інтерфейсів на мобільних пристроях, що підтверджує актуальність розробки власної платформи.

2. На основі вивчення потреб різних ролей користувачів — покупців, продавців, ріелторів та адміністраторів — сформовано чіткі функціональні та нефункціональні вимоги. Вони охоплюють інтуїтивний пошук, персоналізовані рекомендації, безпеку даних та високу швидкодію, що забезпечує комфортну взаємодію для всіх учасників ринку.

3. Розроблено архітектуру системи з чітким розподілом клієнтської та серверної частин, структуру бази даних для ефективного зберігання оголошень і користувачів, а також основні алгоритми пошуку й фільтрації. Це дозволило досягти оптимальної продуктивності та масштабованості платформи.

4. Клієнтська частина реалізована на React з використанням сучасних інструментів для створення адаптивного інтерфейсу. Інтеграція з API забезпечила динамічне оновлення даних і зручну навігацію, що робить платформу доступною на різних пристроях.

5. Серверна частина на Node.js + Express включає надійну JWT-автентифікацію, розмежування доступу за ролями та захищені механізми роботи

з даними. Таке рішення гарантує безпеку інформації та стабільну роботу системи під навантаженням.

6. Комплексне функціональне та адаптивне тестування підтвердило високу ефективність платформи: швидкий пошук, стабільність і зручність використання. Визначено перспективи подальшого розвитку, зокрема впровадження штучного інтелекту для рекомендацій, розширення інтеграцій та вдосконалення аналітики користувацької поведінки.

Таким чином, створена вебплатформа EstatePro є готовим практичним рішенням, яке відповідає сучасним вимогам ринку нерухомості та відкриває можливості для подальшого вдосконалення.

СПИСОК ВИКОРИСТАНИХ ПОСИЛАНЬ

1. ЛУН. Підсумки ринку нерухомості 2024. URL: https://lun.ua/stat/2024 (Дата звернення: 16.03.2026).

2. PwC, Urban Land Institute. Emerging Trends in Real Estate® Global Outlook 2024. URL: https://www.pwc.com/gx/en/asset-management/emerging-trends-real-estate/assets/uli-emerging-trends-global-report-2024.pdf (Дата звернення: 16.03.2026).

3. OLX Нерухомість. Яким був 2024 на ринку нерухомості. Дайджест від OLX Нерухомість. URL: https://business.olx.ua/statti/yakym-buv-2024-na-rynku-nerukhomosti-daydzhest-vid-olx-nerukhomist/ (Дата звернення: 16.03.2026).

4. Оціночний Портал України. Аналіз ринку нерухомості України 3-й квартал 2024 року. URL: https://ocinka.in.ua/analiz-rynku-nerukhomosti-ukrayiny-3-y-kvartal-2024-roku/ (Дата звернення: 16.03.2026).

5. Forbes Business Council. Digital Real Estate Revolution: How Tech Is Transforming The Industry. URL: https://www.forbes.com/councils/forbesbusinesscouncil/2024/03/06/the-digital-real-estate-revolution-how-tech-is-transforming-the-industry/ (Дата звернення: 16.03.2026).

6. Cityscape Global. Top 5 Real Estate Technology Trends in 2024. URL: https://cityscapeglobal.com/blog/top-5-real-estate-technology-trends-2024 (Дата звернення: 16.03.2026).

7. Finance.ua. Український ринок нерухомості у 2024: стан, тенденції, прогнози. URL: https://finance.ua/ua/goodtoknow/ukrainskyi-rynok-nerukhomosti-u-2024-stan-tendentsii-prohnozy (Дата звернення: 16.03.2026).

8. Шевченко Д., Радіюк П. Метод автоматизованого моніторингу оголошень ринку нерухомості України засобами інтелектуального аналізу даних. 2025. URL: https://www.researchgate.net/publication/394087328_Metod_avtomatizovanogo_mo

nitoringu_ogolosen_rinku_neruhomosti_Ukraini_zasobami_intelektualnogo_analizu_danih (Дата звернення: 16.03.2026).

9. Тарасюк В. Р. Веб застосунок для бронювання житлових та нежитлових приміщень. 2025. URL: https://elar.tsatu.edu.ua/bitstreams/6bc7b344-3e0b-4b03-9c2e-d1e5375b317a/download (Дата звернення: 16.03.2026).

10. Мартенс І. А. Стратегічні інструменти міського розвитку. Київський національний університет будівництва і архітектури, 2025. URL: https://repositary.knuba.edu.ua/bitstreams/6c3f4ef1-17c0-4120-840e-1023862b9c85/download (Дата звернення: 16.03.2026).

11. Open House. Топ-7 сервісів для пошуку нерухомості в Україні у 2025 році. 2025. URL: https://openhouse.com.ua/top-7-servisiv-dlia-posuku-neruxomosti-v-ukrayini-u-2025-roci/ (Дата звернення: 16.03.2026).

12. Open4Business. Ukrainian real estate market research in 2024. 2024. URL: https://open4business.com.ua/en/ukrainian-real-estate-market-research-in-2024/ (Дата звернення: 16.03.2026).

13. Zillow. Zillow's AI-powered home search gets smarter with new natural language features. 2024. URL: https://zillow.mediaroom.com/2024-09-04-Zillows-AI-powered-home-search-gets-smarter-with-new-natural-language-features (Дата звернення: 16.03.2026).

14. Web-Promo. Видимість лідерів онлайн-ритейлу в AI-відповідях та нові тенденції. 2026. URL: https://web-promo.ua/ua/blog/shi-trafik-ukrayinskih-marketplejsiv/ (Дата звернення: 16.03.2026).

15. Грох С. А. Використання React та Node.js для реалізації web-додатків. 2024. URL: http://library.econom.zp.ua:85/handle/123456789/338 (Дата звернення: 16.03.2026).

16. Архелюк О. (уклад.). Тези доповідей студентської наукової конференції Чернівецького національного університету імені Юрія Федьковича. Чернівці: ЧНУ, 2025. URL: https://www.chnu.edu.ua/media/leeld04z/2025-nniftkn_compressed.pdf (Дата звернення: 16.03.2026).

17. Марчук Д. Дослідження технології та інструментів розробки сучасних односторінкових (SPA) веб-додатків. Збірник тез VIII Всеукраїнської студентської науково-практичної конференції. ІСТУ, 2025. URL: https://istu.edu.ua/wp-content/uploads/2025/01/Zbirnyk_tez_VIII_Vseukrainskoi_studentskoi_naukovo_praktychnoi_compressed.pdf (Дата звернення: 16.03.2026).

18. Ямнюк Б. Я. (ред.). Матеріали 79-ї науково-технічної конференції. Одеса: ДУІТ, 2024. URL: https://suitt.edu.ua/wp-content/uploads/2025/04/zbirnyky_materialiv_79_i_naukovo_tekhnichnoi_konferentsii_profesorsko.pdf (Дата звернення: 16.03.2026).

19. Національна академія наук України. Національна академія наук України в 2020–2025 роках. Київ: Академперіодика, 2025. URL: https://www.nas.gov.ua/storage/editor/files/zvit-2020-2025-full-web_1759307217.pdf (Дата звернення: 16.03.2026).

20. Сучасні інформаційні технології в освіті та науці. Житомир, 2025. URL: https://lib.iitta.gov.ua/id/eprint/748454/1/Збірник_Житомир_2025.pdf (Дата звернення: 16.03.2026).

21. Збірник матеріалів міжнародної науково-практичної конференції. Київ: Академія праці, соціальних відносин і туризму, 2025. URL: https://www.socosvita.kiev.ua/sites/default/files/PDF_GO_збірник%20конференції%202025_3ac70bff-f7de-4b3a-920f-243fd5aabb2d.pdf (Дата звернення: 16.03.2026).

22. Створення платформи для міжнародної агенції нерухомості: кейс WT Group. URL: https://wezom.com.ua/ua/blog/stvorennya-platformi-dlya-mizhnarodnoyi-agentsiyi-neruhomosti-keys-wt-group (Дата звернення: 16.03.2026).

23. Кузьмінов Є. І., Плугіна Т. В. Побудова інформаційної системи у сфері нерухомості з інтеграцією чат-бота на базі штучного інтелекту. Збірник матеріалів студентської наукової конференції «Сталий розвиток сучасних інформаційних технологій». Харків: Харківський національний автомобільно-дорожній університет, 2025. URL: https://mf.khadi.kharkov.ua/fileadmin/F-

MECHANIC/Комп_ютерних_технологій_і_мехатроніки/2025/Конференції/Збірник_матеріалів_Сталий_розвиток_сучасних_інформаційних_технологій.pdf (Дата звернення: 16.03.2026).

24. Корнієнко О. Інформаційна система для взаємодії мешканців і керуючої компанії житлового комплексу. Київ: КПІ ім. Ігоря Сікорського, 2025. URL: https://ela.kpi.ua/bitstream/123456789/57944/1/Korniienko_magistr.pdf (Дата звернення: 16.03.2026).

25. Власенко І. Г. (ред.). Вісник студентського наукового товариства. Вінниця: Вінницький торговельно-економічний інститут ДТЕУ, 2025. URL: http://vtei.com.ua/doc/2025/nauka/zbirnyk3.pdf (Дата звернення: 16.03.2026).

26. Акінтьєва Л. Розробка модуля «Облік договорів» інформаційної системи агенства нерухомості. 2025. URL: https://openarchive.nure.ua/bitstreams/69c9ac8d-6a08-403b-8bad-498e1327d845/download (Дата звернення: 16.03.2026).

27. Панов А. В. Глобальні тенденції розвитку інформаційних технологій у сфері нерухомості. Ужгород: Ужгородський національний університет, 2025. URL: https://dspace.uzhnu.edu.ua/bitstreams/18daccae-b9e8-4da6-8207-656968f4deb9/download (Дата звернення: 16.03.2026).

28. Думин І. Б., Семич Т. В. Особливості проектування онлайн-сервісу для короткострокової оренди житла для туристів. Вісник Хмельницького національного університету. 2022. №4 (311). С. 227–231.

29. The React Team. React Documentation. 2024. URL: https://react.dev (Дата звернення: 16.03.2026).

30. Express.js Team. Express.js. 2024. URL: https://expressjs.com (Дата звернення: 16.03.2026).

31. SQLite Development Team. SQLite. 2023. URL: https://www.sqlite.org (Дата звернення: 16.03.2026).

32. Axios Team. Axios. 2024. URL: https://axios-http.com (Дата звернення: 16.03.2026).

33. Best Practices When Using JWTs With Web and Mobile Apps. URL: https://duendesoftware.com/learn/best-practices-using-jwts-with-web-and-mobile-apps (Дата звернення: 16.03.2026).

34. Swiper.js Team. Swiper Documentation. 2024. URL: https://swiperjs.com (Дата звернення: 16.03.2026).

35. Коваленко С. М. Сучасні підходи до проєктування архітектури веб-додатків на базі React та Node.js. Вісник інформаційних технологій. 2023. № 4. С. 23–35.

36. Петренко А. В. Алгоритми динамічної фільтрації в веб-додатках на базі Node.js та React. Науковий вісник НТУУ «КПІ». 2024.

37. The SQLite Development Team. Query Planning and Optimization. 2023. URL: https://www.sqlite.org/queryplanner.html (Дата звернення: 16.03.2026).

38. React Team. Managing State with Hooks for Filtering. 2025. URL: https://react.dev/reference/react/useState (Дата звернення: 16.03.2026).

39. Axios Team. Configuring Requests with Query Parameters. 2024. URL: https://axios-http.com/docs/req_config (Дата звернення: 16.03.2026).

40. Найпопулярніші сервіси для пошуку нерухомості в Україні у 2025 році. 2025. URL: https://hotels-group.com.ua/naipopuliarnisi-servisi-dlia-posuku-neruxomosti-v-ukrayini-u-2025-roci/ (Дата звернення: 16.03.2026).

41. Implementing Pagination and Filtering in Expressjs APIs. 2024. URL: https://moldstud.com/articles/p-implementing-pagination-and-filtering-in-expressjs-apis (Дата звернення: 16.03.2026).

42. Що таке Single Page Application та як працює SPA сайт. URL: https://wezom.com.ua/ua/blog/chto-takoe-spa-prilozheniya (Дата звернення: 16.03.2026).

43. Поліщук О.С., Веселовська Н. Р. Розробка адаптивної вебплатформи пошуку та фільтрації нерухомості на react. УДК 004.42:004.738.5. С.3

44. Поліщук О. С., Поремський Ю. В. Теоретичні засади хешування даних. Прикладні інформаційні технології. Збірник тез доповідей - 2024 рік. Секція 2 Алгоритмізація та розробка програмного забезпечення. С. 107-109.

45. Поліщук О. С., Поремський Ю. В. Чисельні методи розв’язання нелінійних рівнянь. Прикладні інформаційні технології. Збірник тез доповідей - 2024 рік. Головна / Архіви / Збірник тез доповідей - 2024 рік / Секція 1 Прикладні інформаційні технології в організаційних, соціально-економічних системах та системах обробки сигналів. С. 51-53.

46. Поліщук О.С., Ніколюк П. К. Застосування алгоритму Дейкстри для пошуку оптимального маршруту. Прикладні інформаційні технології. Збірник тез доповідей - 2023 рік. Секція 2 Алгоритмізація та розробка програмного забезпечення. С. 174-176.

47. Поліщук О. С., Потапова Н. А. Сутність і складники рекурентних алгоритмів. Матеріали IV Всеукраїнської науково-практичної конференції «Комп’ютерні технології обробки даних» - 2023. СЕКЦІЯ 1 МЕТОДИ ОБРОБКИ І АНАЛІЗУ ДАНИХ. С. 98-99.

ДОДАТОК А

ОСНОВНІ ЛІСТИНГИ КОДІВ ПРОЄКТУ

AllProperties.jsx

```
import React, { useState, useEffect } from "react";
import { Link, useNavigate } from "react-router-dom";
import { FaMapMarkerAlt, FaHome, FaRegFileAlt } from "react-icons/fa";
import { propertiesApi } from "../../../api";
import { useLang } from "../../../context/LangContext";
import icon from "../../../assets/img/home/exp.png";
import bed from "../../../assets/img/home/bed.png";
import bath from "../../../assets/img/home/bathroom.png";
import styles from "../css/AllProperties.module.css";
function AllProperties() {
 const navigate = useNavigate();
 const { t } = useLang();
 const [properties, setProperties] = useState([]);
 const [loading, setLoading] = useState(true);
 const [search, setSearch] = useState("");
 const [city, setCity] = useState("");
 const [propertyType, setPropertyType] = useState("");
 const [listingType, setListingType] = useState("");
 const [cities, setCities] = useState([]);
 const [types, setTypes] = useState([]);
 const [pagination, setPagination] = useState({ page: 1, pages: 1 });
 // Завантаження фільтрів
 useEffect(() => {
  const loadFilters = async () => {
   try {
    const [citiesData, typesData] = await Promise.all([
     propertiesApi.getCities(),
     propertiesApi.getTypes()
    ]);
    setCities(citiesData || []);
    setTypes(typesData || []);
   } catch (error) {
    console.error("Error loading filters:", error);
   }
  };
  loadFilters();
 }, []);
 // Завантаження об'єктів
 useEffect(() => {
  const loadProperties = async () => {
   setLoading(true);
   try {
    const response = await propertiesApi.getAll({
     city: city || undefined,
     property_type: propertyType || undefined,
     listing_type: listingType || undefined,
     page: pagination.page,
     limit: 12
    });
    setProperties(response.properties || []);
    setPagination(response.pagination || { page: 1, pages: 1 });
   } catch (error) {
    console.error("Error loading properties:", error);
   } finally {
    setLoading(false);
```

```
    }
  };
  loadProperties();
}, [city, propertyType, listingType, pagination.page]);
// Фільтрація за пошуком (клієнтська)
const filtered = properties.filter((p) => {
  if (!search) return true;
  const searchText = search.toLowerCase();
  return (
    p.title?.toLowerCase().includes(searchText) ||
    p.location?.toLowerCase().includes(searchText) ||
    p.city?.toLowerCase().includes(searchText)
  );
});
const getImageUrl = (imgUrl) => {
  if (!imgUrl) return "";
  if (imgUrl.startsWith("http")) return imgUrl;
  return `http://localhost:5050${imgUrl}`;
};
return (
  <section className={styles.section}>
    <div className={styles.wrapper}>
      <h2 className={styles.title}>{t("properties.title")}</h2>
      <p className={styles.text}>{t("properties.subtitle")}</p>
      <div className={styles.buttons}>
        <button
          className={styles.btnOrange}
          onClick={() => navigate("/property-post")}
        >
          {t("footer.propertyPost")}
        </button>
        <button
          className={styles.btnWhite}
          onClick={() => navigate("/about")}
        >
          {t("buttons.learnMore")}
        </button>
      </div>
      <div className={styles.filters}>
        <div className={styles.inputGroup}>
          <div className={styles.inputBox}>
            <FaMapMarkerAlt className={styles.icon} style={{ width: "40px" }} />
            <input
              placeholder={t("buttons.search")}
              value={search}
              onChange={(e) => setSearch(e.target.value)}
              className={styles.input}
            />
          </div>
          <button className={styles.searchBtn}>{t("buttons.search")}</button>
        </div>
        <div className={styles.selectGroup}>
          <FaMapMarkerAlt className={styles.icon} style={{ width: "40px" }} />
          <select
            value={city}
            onChange={(e) => setCity(e.target.value)}
            className={styles.select}
          >
            <option value="">{t("properties.filters.city")}</option>
            {cities.map((c) => (
              <option key={c} value={c}>{c}</option>
            ))}
```

```
  </select>
 </div>
 <div className={styles.selectGroup}>
  <FaHome className={styles.icon} />
  <select
   value={propertyType}
   onChange={(e) => setPropertyType(e.target.value)}
   className={styles.select}
  >
   <option value="">{t("properties.filters.type")}</option>
   {types.map((type) => (
    <option key={type} value={type}>{type}</option>
   ))}
  </select>
 </div>
 <div className={styles.selectGroup}>
  <FaRegFileAlt className={styles.icon} />
  <select
   value={listingType}
   onChange={(e) => setListingType(e.target.value)}
   className={styles.select}
  >
   <option value="">{t("properties.filters.listingType")}</option>
   <option value="For Sale">{t("properties.filters.forSale")}</option>
   <option value="For Rent">{t("properties.filters.forRent")}</option>
  </select>
 </div>
</div>
{loading ? (
 <div className={styles.loading}>{t("buttons.loading")}</div>
) : (
 <div className={styles.cards}>
  {filtered.length ? (
   filtered.map((house, index) => (
    <Link
     key={house.id}
     className={`${styles.card} ${index % 2 === 1 ? styles.cardShift : ""}`}
     to={`/properties/${house.id}`}
    >
     <img
      className={styles.img}
      src={getImageUrl(house.img_url)}
      alt={house.title}
     />
     <div className={styles.line}></div>
     <div className={styles.info}>
      <p className={styles.price}>$ {house.price?.toLocaleString()}</p>
      <p className={styles.location}>{house.location}</p>
     </div>
     <hr />
     <div className={styles.details}>
      <div className={styles.detailBox}>
       <img src={icon} alt="" /> {house.sqft} м²
      </div>
      <div className={styles.detailBox}>
       <img src={bed} alt="" /> {house.bedrooms}
      </div>
      <div className={styles.detailBox}>
       <img src={bath} alt="" /> {house.bathrooms}
      </div>
     </div>
    </Link>
```

```
          ))
        ) : (
          <p className={styles.noResults}>{t("properties.noResults")}</p>
        )}
      </div>
    )}
    {pagination.pages > 1 && (
      <div className={styles.pagination}>
        {Array.from({ length: pagination.pages }, (_, i) => (
          <button
            key={i}
            className={`${styles.pageBtn} ${pagination.page === i + 1 ? styles.active : ""}`}
            onClick={() => setPagination(prev => ({ ...prev, page: i + 1 }))}
          >
            {i + 1}
          </button>
        ))}
      </div>
    )}
  </div>
 </section>
 );
}
export default AllProperties;
```

properties.controller.js

```
const { body, query, param } = require('express-validator');
const { db } = require('../config/database');
const { getAgentId } = require('../middleware/auth');
// Валідатори
// Допустимі значення типів
const PROPERTY_TYPES = ['Квартира', 'Вілла', 'Пентхаус', 'Будинок', 'Таунхаус', 'Apartment', 'Villa', 'Penthouse', 'House',
'Townhouse'];
const LISTING_TYPES = ['sale', 'rent', 'For Sale', 'For Rent'];
const createPropertyValidation = [
  body('title').trim().notEmpty().withMessage("Заголовок обов'язковий"),
  body('location').trim().notEmpty().withMessage("Локація обов'язкова"),
  body('city').trim().notEmpty().withMessage("Місто обов'язкове"),
  body('property_type').isIn(PROPERTY_TYPES).withMessage('Невалідний тип нерухомості'),
  body('listing_type').isIn(LISTING_TYPES).withMessage('Невалідний тип оголошення'),
  body('price').isFloat({ min: 0 }).withMessage('Ціна має бути додатнім числом'),
  body('sqft').optional({ nullable: true }).isInt({ min: 0 }).withMessage('Площа має бути додатнім числом'),
  body('bedrooms').optional({ nullable: true }).isInt({ min: 0 }).withMessage('Кількість спалень має бути додатнім числом'),
  body('bathrooms').optional({ nullable: true }).isInt({ min: 0 }).withMessage('Кількість ванних має бути додатнім числом'),
  body('description').optional().trim(),
  body('amenities').optional().isArray().withMessage('Зручності мають бути масивом'),
  body('about').optional().isArray().withMessage('Опис має бути масивом')
];
const updatePropertyValidation = [
  body('title').optional().trim().notEmpty(),
  body('location').optional().trim().notEmpty(),
  body('city').optional().trim().notEmpty(),
  body('property_type').optional().isIn(PROPERTY_TYPES),
  body('listing_type').optional().isIn(LISTING_TYPES),
  body('price').optional().isFloat({ min: 0 }),
  body('sqft').optional({ nullable: true }).isInt({ min: 0 }),
  body('bedrooms').optional({ nullable: true }).isInt({ min: 0 }),
  body('bathrooms').optional({ nullable: true }).isInt({ min: 0 })
];
const queryValidation = [
  query('page').optional().isInt({ min: 1 }),
  query('limit').optional().isInt({ min: 1, max: 50 }),
```

```
  query('city').optional().trim(),
  query('property_type').optional().trim(),
  query('listing_type').optional().trim(),
  query('min_price').optional().isFloat({ min: 0 }),
  query('max_price').optional().isFloat({ min: 0 }),
  query('bedrooms').optional().isInt({ min: 0 })
];
// Допоміжні функції
const getPropertyWithDetails = (propertyId) => {
  const property = db.prepare(`
    SELECT p.*,
           a.id as agent_id,
           u.name as agent_name,
           u.email as agent_email,
           u.phone as agent_phone,
           ag.role_title as agent_role,
           ag.img_url as agent_img
    FROM properties p
    LEFT JOIN agents ag ON p.agent_id = ag.id
    LEFT JOIN users u ON ag.user_id = u.id
    LEFT JOIN agents a ON p.agent_id = a.id
    WHERE p.id = ?
  `).get(propertyId);
  if (!property) return null;
  // Зручності
  property.amenities = db.prepare(`
    SELECT am.name FROM property_amenities pa
    JOIN amenities am ON pa.amenity_id = am.id
    WHERE pa.property_id = ?
  `).all(propertyId).map(a => a.name);
  // Опис (about)
  property.about = db.prepare(`
    SELECT text FROM property_about
    WHERE property_id = ?
    ORDER BY sort_order
  `).all(propertyId).map(a => a.text);
  return property;
};
// Контролери
const getAll = (req, res, next) => {
  try {
    const {
      page = 1,
      limit = 12,
      city,
      property_type,
      listing_type,
      min_price,
      max_price,
      bedrooms
    } = req.query;
    const offset = (page - 1) * limit;
    const conditions = ['p.status = ?'];
    const params = ['approved'];
    if (city) {
      conditions.push('p.city = ?');
      params.push(city);
    }
    if (property_type) {
      conditions.push('p.property_type = ?');
      params.push(property_type);
    }
```

```
    if (listing_type) {
      conditions.push('p.listing_type = ?');
      params.push(listing_type);
    }
    if (min_price) {
      conditions.push('p.price >= ?');
      params.push(parseFloat(min_price));
    }
    if (max_price) {
      conditions.push('p.price <= ?');
      params.push(parseFloat(max_price));
    }
    if (bedrooms) {
      conditions.push('p.bedrooms >= ?');
      params.push(parseInt(bedrooms));
    }
    const whereClause = conditions.length > 0 ? `WHERE ${conditions.join(' AND ')}` : '';
    // Загальна кількість
    const countQuery = `SELECT COUNT(*) as total FROM properties p ${whereClause}`;
    const { total } = db.prepare(countQuery).get(...params);
    // Об'єкти
    const query = `
      SELECT p.*, u.name as agent_name, ag.img_url as agent_img
      FROM properties p
      LEFT JOIN agents ag ON p.agent_id = ag.id
      LEFT JOIN users u ON ag.user_id = u.id
      ${whereClause}
      ORDER BY p.created_at DESC
      LIMIT ? OFFSET ?
    `;
    const properties = db.prepare(query).all(...params, parseInt(limit), offset);
    // Додаємо amenities до кожного об'єкта
    properties.forEach(prop => {
      prop.amenities = db.prepare(`
        SELECT am.name FROM property_amenities pa
        JOIN amenities am ON pa.amenity_id = am.id
        WHERE pa.property_id = ?
      `).all(prop.id).map(a => a.name);
    });
    res.json({
      properties,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    next(error);
  }
};
const getById = (req, res, next) => {
  try {
    const property = getPropertyWithDetails(req.params.id);
    if (!property) {
      return res.status(404).json({ message: "Об'єкт не знайдено" });
    }
    // Якщо об'єкт не approved, доступ тільки для власника/адміна
    if (property.status !== 'approved') {
      if (!req.user || (req.user.role !== 'admin' && getAgentId(req) !== property.agent_id)) {
        return res.status(404).json({ message: "Об'єкт не знайдено" });
```

```
    }
   }
   res.json({ property });
  } catch (error) {
   next(error);
  }
};
const create = (req, res, next) => {
  try {
   const agentId = getAgentId(req);
   if (!agentId && req.user.role !== 'admin') {
    return res.status(403).json({ message: 'Тільки агенти можуть створювати об\'єкти' });
   }
   const {
    title, location, address, city, property_type, listing_type,
    price, sqft, bedrooms, bathrooms, parking, description, img_url,
    amenities = [], about = []
   } = req.body;
   // Статус залежить від ролі
   const status = req.user.role === 'admin' ? 'approved' : 'pending';
   const result = db.prepare(`
    INSERT INTO properties (
     agent_id, title, location, address, city, property_type,
     listing_type, price, sqft, bedrooms, bathrooms, parking,
     description, img_url, status
    ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
   `).run(
    agentId, title, location, address || null, city, property_type,
    listing_type, price, sqft || null, bedrooms || null, bathrooms || null,
    parking || 0, description || null, img_url || null, status
   );
   const propertyId = result.lastInsertRowid;
   // Додаємо зручності
   if (amenities.length > 0) {
    const insertAmenity = db.prepare(`
     INSERT OR IGNORE INTO property_amenities (property_id, amenity_id)
     SELECT ?, id FROM amenities WHERE name = ?
    `);
    amenities.forEach(name => insertAmenity.run(propertyId, name));
   }
   // Додаємо опис
   if (about.length > 0) {
    const insertAbout = db.prepare(`
     INSERT INTO property_about (property_id, text, sort_order)
     VALUES (?, ?, ?)
    `);
    about.forEach((text, idx) => insertAbout.run(propertyId, text, idx));
   }
   const property = getPropertyWithDetails(propertyId);
   res.status(201).json({
    message: status === 'approved' ? "Об'єкт створено" : "Об'єкт створено і очікує модерації",
    property
   });
  } catch (error) {
   next(error);
  }
};
const update = (req, res, next) => {
  try {
   const { id } = req.params;
   const property = db.prepare('SELECT * FROM properties WHERE id = ?').get(id);
   if (!property) {
```

```
    return res.status(404).json({ message: "Об'єкт не знайдено" });
   }
   // Перевірка прав
   const agentId = getAgentId(req);
   if (req.user.role !== 'admin' && property.agent_id !== agentId) {
    return res.status(403).json({ message: 'Ви не можете редагувати цей об\'єкт' });
   }
   const allowedFields = [
    'title', 'location', 'address', 'city', 'property_type', 'listing_type',
    'price', 'sqft', 'bedrooms', 'bathrooms', 'parking', 'description', 'img_url'
   ];
   const updates = [];
   const params = [];
   allowedFields.forEach(field => {
    if (req.body[field] !== undefined) {
     updates.push(`${field} = ?`);
     params.push(req.body[field]);
    }
   });
   if (updates.length > 0) {
    params.push(id);
    db.prepare(`UPDATE properties SET ${updates.join(', ')} WHERE id = ?`).run(...params);
   }
   // Оновлення зручностей
   if (req.body.amenities !== undefined) {
    db.prepare('DELETE FROM property_amenities WHERE property_id = ?').run(id);
    const insertAmenity = db.prepare(`
     INSERT OR IGNORE INTO property_amenities (property_id, amenity_id)
     SELECT ?, id FROM amenities WHERE name = ?
    `);
    req.body.amenities.forEach(name => insertAmenity.run(id, name));
   }
   // Оновлення опису
   if (req.body.about !== undefined) {
    db.prepare('DELETE FROM property_about WHERE property_id = ?').run(id);
    const insertAbout = db.prepare(`
     INSERT INTO property_about (property_id, text, sort_order)
     VALUES (?, ?, ?)
    `);
    req.body.about.forEach((text, idx) => insertAbout.run(id, text, idx));
   }
   const updated = getPropertyWithDetails(id);
   res.json({
    message: "Об'єкт оновлено",
    property: updated
   });
  } catch (error) {
   next(error);
  }
};
const remove = (req, res, next) => {
 try {
  const { id } = req.params;
  const property = db.prepare('SELECT * FROM properties WHERE id = ?').get(id);
  if (!property) {
   return res.status(404).json({ message: "Об'єкт не знайдено" });
  }
  const agentId = getAgentId(req);
  if (req.user.role !== 'admin' && property.agent_id !== agentId) {
   return res.status(403).json({ message: 'Ви не можете видалити цей об\'єкт' });
  }
  db.prepare('DELETE FROM properties WHERE id = ?').run(id);
```

```
    res.json({ message: "Об'єкт видалено" });
  } catch (error) {
    next(error);
  }
};
const updateStatus = (req, res, next) => {
  try {
    const { id } = req.params;
    const { status } = req.body;
    if (!['pending', 'approved', 'rejected'].includes(status)) {
      return res.status(400).json({ message: 'Невалідний статус' });
    }
    const result = db.prepare('UPDATE properties SET status = ? WHERE id = ?').run(status, id);
    if (result.changes === 0) {
      return res.status(404).json({ message: "Об'єкт не знайдено" });
    }
    res.json({ message: 'Статус оновлено' });
  } catch (error) {
    next(error);
  }
};
const getPending = (req, res, next) => {
  try {
    const properties = db.prepare(`
      SELECT p.*, u.name as agent_name
      FROM properties p
      LEFT JOIN agents ag ON p.agent_id = ag.id
      LEFT JOIN users u ON ag.user_id = u.id
      WHERE p.status = 'pending'
      ORDER BY p.created_at DESC
    `).all();
    res.json({ properties });
  } catch (error) {
    next(error);
  }
};
const getMy = (req, res, next) => {
  try {
    const agentId = getAgentId(req);
    if (!agentId) {
      return res.status(403).json({ message: 'Ви не є агентом' });
    }
    const properties = db.prepare(`
      SELECT * FROM properties
      WHERE agent_id = ?
      ORDER BY created_at DESC
    `).all(agentId);
    res.json({ properties });
  } catch (error) {
    next(error);
  }
};
const getCities = (req, res, next) => {
  try {
    const cities = db.prepare(`
      SELECT DISTINCT city FROM properties WHERE status = 'approved' ORDER BY city
    `).all().map(c => c.city);
    res.json({ cities });
  } catch (error) {
    next(error);
  }
};
```

```
const getTypes = (req, res, next) => {
 try {
  const types = db.prepare(`
   SELECT DISTINCT property_type FROM properties WHERE status = 'approved' ORDER BY property_type
  `).all().map(t => t.property_type);
  res.json({ types });
 } catch (error) {
  next(error);
 }
};
const getAmenities = (req, res, next) => {
 try {
  const amenities = db.prepare('SELECT * FROM amenities ORDER BY name').all();
  res.json({ amenities });
 } catch (error) {
  next(error);
 }
};
module.exports = {
 createPropertyValidation,
 updatePropertyValidation,
 queryValidation,
 getAll,
 getById,
 create,
 update,
 remove,
 updateStatus,
 getPending,
 getMy,
 getCities,
 getTypes,
 getAmenities
};
```

auth.controller.js

```
const { body } = require('express-validator');
const authService = require('../services/auth.service');
const jwtConfig = require('../config/jwt');
// Валідатори
const registerValidation = [
 body('email').isEmail().withMessage('Невалідна email адреса').normalizeEmail(),
 body('password').isLength({ min: 6 }).withMessage('Пароль має містити мінімум 6 символів'),
 body('name').trim().notEmpty().withMessage("Ім'я обов'язкове"),
 body('phone').optional().trim()
];
const loginValidation = [
 body('email').isEmail().withMessage('Невалідна email адреса'),
 body('password').notEmpty().withMessage("Пароль обов'язковий")
];
const changePasswordValidation = [
 body('currentPassword').notEmpty().withMessage("Поточний пароль обов'язковий"),
 body('newPassword').isLength({ min: 6 }).withMessage('Новий пароль має містити мінімум 6 символів')
];
const updateProfileValidation = [
 body('name').optional().trim().notEmpty().withMessage("Ім'я не може бути порожнім"),
 body('phone').optional().trim()
];
// Контролери
const register = async (req, res, next) => {
 try {
  const user = await authService.register(req.body);
```

```
    res.status(201).json({
      message: 'Реєстрація успішна',
      user
    });
  } catch (error) {
    next(error);
  }
};
const login = async (req, res, next) => {
  try {
    const { user, accessToken, refreshToken } = await authService.login(req.body);
    // Встановлюємо refresh токен в httpOnly cookie
    res.cookie('refreshToken', refreshToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'lax',
      maxAge: jwtConfig.getRefreshMaxAge()
    });
    res.json({
      message: 'Вхід успішний',
      user,
      accessToken
    });
  } catch (error) {
    next(error);
  }
};
const logout = (req, res, next) => {
  try {
    const refreshToken = req.cookies.refreshToken;
    authService.logout(refreshToken);
    res.clearCookie('refreshToken');
    res.json({ message: 'Вихід успішний' });
  } catch (error) {
    next(error);
  }
};
const refresh = (req, res, next) => {
  try {
    const refreshToken = req.cookies.refreshToken;
    if (!refreshToken) {
      return res.status(401).json({ message: 'Refresh токен відсутній' });
    }
    const tokens = authService.refreshAccessToken(refreshToken);
    // Оновлюємо cookie
    res.cookie('refreshToken', tokens.refreshToken, {
      httpOnly: true,
      secure: process.env.NODE_ENV === 'production',
      sameSite: 'lax',
      maxAge: jwtConfig.getRefreshMaxAge()
    });
    res.json({
      accessToken: tokens.accessToken
    });
  } catch (error) {
    res.clearCookie('refreshToken');
    next(error);
  }
};
const getMe = (req, res) => {
  const user = authService.getUserById(req.user.id);
  res.json({ user });
```

```
};
const updateMe = async (req, res, next) => {
 try {
  const user = authService.updateProfile(req.user.id, req.body);
  res.json({
   message: 'Профіль оновлено',
   user
  });
 } catch (error) {
  next(error);
 }
};
const changePassword = async (req, res, next) => {
 try {
  await authService.changePassword(req.user.id, req.body);
  res.clearCookie('refreshToken');
  res.json({
   message: 'Пароль змінено. Увійдіть знову з новим паролем.'
  });
 } catch (error) {
  next(error);
 }
};
module.exports = {
 registerValidation,
 loginValidation,
 changePasswordValidation,
 updateProfileValidation,
 register,
 login,
 logout,
 refresh,
 getMe,
 updateMe,
 changePassword
};
```

favorites.controller.js

```
const { db } = require('../config/database');
// Контролери
const getAll = (req, res, next) => {
 try {
  const favorites = db.prepare(`
   SELECT p.*, f.created_at as favorited_at
   FROM favorites f
   JOIN properties p ON f.property_id = p.id
   WHERE f.user_id = ? AND p.status = 'approved'
   ORDER BY f.created_at DESC
  `).all(req.user.id);
  // Додаємо amenities
  favorites.forEach(prop => {
   prop.amenities = db.prepare(`
    SELECT am.name FROM property_amenities pa
    JOIN amenities am ON pa.amenity_id = am.id
    WHERE pa.property_id = ?
   `).all(prop.id).map(a => a.name);
  });
  res.json({ favorites });
 } catch (error) {
  next(error);
 }
};
```

```
const add = (req, res, next) => {
 try {
  const { propertyId } = req.params;
  // Перевірка чи об'єкт існує
  const property = db.prepare(`
   SELECT id FROM properties WHERE id = ? AND status = 'approved'
  `).get(propertyId);
  if (!property) {
   return res.status(404).json({ message: "Об'єкт не знайдено" });
  }
  // Перевірка чи вже в обраному
  const existing = db.prepare(`
   SELECT * FROM favorites WHERE user_id = ? AND property_id = ?
  `).get(req.user.id, propertyId);
  if (existing) {
   return res.status(409).json({ message: "Об'єкт вже в обраному" });
  }
  db.prepare(`
   INSERT INTO favorites (user_id, property_id) VALUES (?, ?)
  `).run(req.user.id, propertyId);
  res.status(201).json({ message: 'Додано до обраного' });
 } catch (error) {
  next(error);
 }
};
const remove = (req, res, next) => {
 try {
  const { propertyId } = req.params;
  const result = db.prepare(`
   DELETE FROM favorites WHERE user_id = ? AND property_id = ?
  `).run(req.user.id, propertyId);
  if (result.changes === 0) {
   return res.status(404).json({ message: "Об'єкт не знайдено в обраному" });
  }
  res.json({ message: 'Видалено з обраного' });
 } catch (error) {
  next(error);
 }
};
const check = (req, res, next) => {
 try {
  const { propertyId } = req.params;
  const exists = db.prepare(`
   SELECT 1 FROM favorites WHERE user_id = ? AND property_id = ?
  `).get(req.user.id, propertyId);
  res.json({ isFavorite: !!exists });
 } catch (error) {
  next(error);
 }
};
const getIds = (req, res, next) => {
 try {
  const ids = db.prepare(`
   SELECT property_id FROM favorites WHERE user_id = ?
  `).all(req.user.id).map(f => f.property_id);
  res.json({ ids });
 } catch (error) {
  next(error);
 }
};
module.exports = {
 getAll,
```

```
  add,
  remove,
  check,
  getIds
};
```

history.controller.js

```
const { db } = require('../config/database');
// Контролери
const getAll = (req, res, next) => {
  try {
    // Отримуємо останні унікальні перегляди
    const history = db.prepare(`
      SELECT p.*, MAX(vh.viewed_at) as viewed_at
      FROM view_history vh
      JOIN properties p ON vh.property_id = p.id
      WHERE vh.user_id = ? AND p.status = 'approved'
      GROUP BY vh.property_id
      ORDER BY MAX(vh.viewed_at) DESC
      LIMIT 20
    `).all(req.user.id);
    res.json({ history });
  } catch (error) {
    next(error);
  }
};
const add = (req, res, next) => {
  try {
    const { propertyId } = req.params;
    // Перевірка чи об'єкт існує
    const property = db.prepare(`
      SELECT id FROM properties WHERE id = ? AND status = 'approved'
    `).get(propertyId);
    if (!property) {
      return res.status(404).json({ message: "Об'єкт не знайдено" });
    }
    db.prepare(`
      INSERT INTO view_history (user_id, property_id) VALUES (?, ?)
    `).run(req.user.id, propertyId);
    res.status(201).json({ message: 'Додано до історії' });
  } catch (error) {
    next(error);
  }
};
const clear = (req, res, next) => {
  try {
    db.prepare('DELETE FROM view_history WHERE user_id = ?').run(req.user.id);
    res.json({ message: 'Історію очищено' });
  } catch (error) {
    next(error);
  }
};
module.exports = {
  getAll,
  add,
  clear
};
```

agents.controller.js

```
const { body } = require('express-validator');
const { db } = require('../config/database');
// Валідатори
```

```
const updateAgentValidation = [
  body('role_title').optional().trim().notEmpty(),
  body('description').optional().trim(),
  body('description_full').optional().trim()
];
// Контролери
const getAll = (req, res, next) => {
  try {
    const agents = db.prepare(`
      SELECT
        a.id,
        a.role_title,
        a.description,
        a.description_full,
        a.img_url,
        u.name,
        u.email,
        u.phone
      FROM agents a
      JOIN users u ON a.user_id = u.id
      WHERE u.is_active = 1
      ORDER BY a.id
    `).all();
    res.json({ agents });
  } catch (error) {
    next(error);
  }
};
const getById = (req, res, next) => {
  try {
    const agent = db.prepare(`
      SELECT
        a.id,
        a.role_title,
        a.description,
        a.description_full,
        a.img_url,
        u.name,
        u.email,
        u.phone
      FROM agents a
      JOIN users u ON a.user_id = u.id
      WHERE a.id = ? AND u.is_active = 1
    `).get(req.params.id);
    if (!agent) {
      return res.status(404).json({ message: 'Агента не знайдено' });
    }
    // Об'єкти агента
    const properties = db.prepare(`
      SELECT * FROM properties
      WHERE agent_id = ? AND status = 'approved'
      ORDER BY created_at DESC
      LIMIT 6
    `).all(req.params.id);
    res.json({ agent, properties });
  } catch (error) {
    next(error);
  }
};
const update = (req, res, next) => {
  try {
    const { id } = req.params;
```

```
    // Перевірка чи агент належить поточному користувачу
    const agent = db.prepare('SELECT * FROM agents WHERE id = ?').get(id);
    if (!agent) {
      return res.status(404).json({ message: 'Агента не знайдено' });
    }
    if (req.user.role !== 'admin' && agent.user_id !== req.user.id) {
      return res.status(403).json({ message: 'Ви не можете редагувати цього агента' });
    }
    const { role_title, description, description_full, img_url } = req.body;
    const updates = [];
    const params = [];
    if (role_title !== undefined) {
      updates.push('role_title = ?');
      params.push(role_title);
    }
    if (description !== undefined) {
      updates.push('description = ?');
      params.push(description);
    }
    if (description_full !== undefined) {
      updates.push('description_full = ?');
      params.push(description_full);
    }
    if (img_url !== undefined) {
      updates.push('img_url = ?');
      params.push(img_url);
    }
    if (updates.length > 0) {
      params.push(id);
      db.prepare(`UPDATE agents SET ${updates.join(', ')} WHERE id = ?`).run(...params);
    }
    const updated = db.prepare(`
      SELECT
        a.id,
        a.role_title,
        a.description,
        a.description_full,
        a.img_url,
        u.name,
        u.email,
        u.phone
      FROM agents a
      JOIN users u ON a.user_id = u.id
      WHERE a.id = ?
    `).get(id);
    res.json({
      message: 'Профіль агента оновлено',
      agent: updated
    });
  } catch (error) {
    next(error);
  }
};
// Отримати агента за user_id (admin)
const getByUserId = (req, res, next) => {
  try {
    const { userId } = req.params;
    const agent = db.prepare(`
      SELECT * FROM agents WHERE user_id = ?
    `).get(userId);
    if (!agent) {
      return res.status(404).json({ message: 'Агента не знайдено', agent: null });
```

```
    }
    res.json({ agent });
  } catch (error) {
    next(error);
  }
};
// Оновити або створити агента за user_id (admin)
const updateByUserId = (req, res, next) => {
  try {
    const { userId } = req.params;
    const { role_title, description, description_full, img_url, experience_years, specialization } = req.body;
    // Перевіряємо чи існує агент
    let agent = db.prepare('SELECT * FROM agents WHERE user_id = ?').get(userId);
    if (!agent) {
      // Створюємо нового агента
      const result = db.prepare(`
        INSERT INTO agents (user_id, role_title, description, description_full, img_url, experience_years, specialization)
        VALUES (?, ?, ?, ?, ?, ?, ?)
      `).run(
        userId,
        role_title || 'Агент з нерухомості',
        description || null,
        description_full || null,
        img_url || null,
        experience_years || 1,
        specialization || null
      );
      agent = db.prepare('SELECT * FROM agents WHERE id = ?').get(result.lastInsertRowid);
    } else {
      // Оновлюємо існуючого
      const updates = [];
      const params = [];
      if (role_title !== undefined) { updates.push('role_title = ?'); params.push(role_title); }
      if (description !== undefined) { updates.push('description = ?'); params.push(description); }
      if (description_full !== undefined) { updates.push('description_full = ?'); params.push(description_full); }
      if (img_url !== undefined) { updates.push('img_url = ?'); params.push(img_url); }
      if (experience_years !== undefined) { updates.push('experience_years = ?'); params.push(experience_years); }
      if (specialization !== undefined) { updates.push('specialization = ?'); params.push(specialization); }
      if (updates.length > 0) {
        params.push(agent.id);
        db.prepare(`UPDATE agents SET ${updates.join(', ')} WHERE id = ?`).run(...params);
      }
      agent = db.prepare('SELECT * FROM agents WHERE id = ?').get(agent.id);
    }
    res.json({ message: 'Дані агента оновлено', agent });
  } catch (error) {
    next(error);
  }
};
module.exports = {
  updateAgentValidation,
  getAll,
  getById,
  update,
  getByUserId,
  updateByUserId
};
```

Property.jsx

```
import React, { useEffect, useRef, useState } from "react";
import { useNavigate, useParams, Link } from "react-router-dom";
import { useAuth } from "../../../context/AuthContext";
```

```
import { propertiesApi, favoritesApi, formsApi } from "../../../api";
import api from "../../../api/client";
import Breadcrumbs from "../../../components/Breadcrumbs";
import { MdDone } from "react-icons/md";
import { FaStar } from "react-icons/fa";
import { FiMapPin, FiHome, FiSquare, FiUser } from "react-icons/fi";
import styles from "../css/Property.module.css";
function Property() {
  const navigate = useNavigate();
  const { id } = useParams();
  const { user, isAuthenticated } = useAuth();
  const formRef = useRef(null);
  const [property, setProperty] = useState(null);
  const [loading, setLoading] = useState(true);
  const [isFavorite, setIsFavorite] = useState(false);
  const [formData, setFormData] = useState({
    name: "",
    email: "",
    phone: "",
    message: "",
    // Для купівлі
    has_mortgage: false,
    budget_from: "",
    budget_to: "",
    // Для оренди
    desired_move_in: "",
    rent_period: "",
    has_pets: false,
    num_residents: "",
  });
  const [errors, setErrors] = useState({});
  const [success, setSuccess] = useState(false);
  const [submitting, setSubmitting] = useState(false);
  // Автозаповнення даних користувача
  useEffect(() => {
    if (isAuthenticated && user) {
      setFormData(prev => ({
        ...prev,
        name: user.name || "",
        email: user.email || "",
        phone: user.phone || "",
      }));
    }
  }, [isAuthenticated, user]);
  useEffect(() => {
    const fetchProperty = async () => {
      try {
        const data = await propertiesApi.getById(id);
        setProperty(data);
        // Записуємо в історію переглядів
        if (isAuthenticated) {
          try {
            await api.post(`/history/${id}`);
          } catch (e) {}
        }
      } catch (error) {
        console.error("Error fetching property:", error);
      } finally {
        setLoading(false);
      }
    };
    fetchProperty();
```

```
}, [id, isAuthenticated]);
useEffect(() => {
  const checkFavorite = async () => {
    if (isAuthenticated && property) {
      try {
        const favorites = await favoritesApi.getAll();
        setIsFavorite(favorites.some(f => f.id === property.id));
      } catch (e) {}
    }
  };
  checkFavorite();
}, [isAuthenticated, property]);
const scrollToForm = () => {
  formRef.current?.scrollIntoView({ behavior: "smooth", block: "start" });
};
const toggleFavorite = async () => {
  if (!isAuthenticated) {
    navigate("/login");
    return;
  }
  try {
    if (isFavorite) {
      await favoritesApi.remove(property.id);
    } else {
      await favoritesApi.add(property.id);
    }
    setIsFavorite(!isFavorite);
  } catch (error) {
    console.error("Error toggling favorite:", error);
  }
};
const validate = () => {
  const newErrors = {};
  if (!formData.name.trim()) newErrors.name = "Введіть ім'я";
  if (!formData.email) {
    newErrors.email = "Введіть email";
  } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
    newErrors.email = "Невірний формат email";
  }
  if (!formData.phone) {
    newErrors.phone = "Введіть телефон";
  }
  if (!formData.message.trim()) newErrors.message = "Введіть повідомлення";
  return newErrors;
};
const handleSubmit = async (e) => {
  e.preventDefault();
  const validationErrors = validate();
  if (Object.keys(validationErrors).length > 0) {
    setErrors(validationErrors);
    return;
  }
  setSubmitting(true);
  try {
    const inquiryData = {
      property_id: property.id,
      user_id: user?.id || null,
      name: formData.name,
      email: formData.email,
      phone: formData.phone,
      message: formData.message,
      inquiry_type: property.listing_type === 'sale' ? 'buy' : 'rent',
```

```
    };
    // Додаткові поля для купівлі
    if (property.listing_type === 'sale') {
      inquiryData.has_mortgage = formData.has_mortgage ? 1 : 0;
      inquiryData.budget_from = formData.budget_from || null;
      inquiryData.budget_to = formData.budget_to || null;
    }
    // Додаткові поля для оренди
    if (property.listing_type === 'rent') {
      inquiryData.desired_move_in = formData.desired_move_in || null;
      inquiryData.rent_period = formData.rent_period || null;
      inquiryData.has_pets = formData.has_pets ? 1 : 0;
      inquiryData.num_residents = formData.num_residents || null;
    }
    await formsApi.submitInquiry(inquiryData);
    setSuccess(true);
    // Скидаємо тільки повідомлення та додаткові поля, контактні дані залишаємо
    setFormData(prev => ({
      ...prev,
      message: "",
      has_mortgage: false,
      budget_from: "",
      budget_to: "",
      desired_move_in: "",
      rent_period: "",
      has_pets: false,
      num_residents: "",
    }));
    setErrors({});
  } catch (error) {
    setErrors({ submit: error.message });
  } finally {
    setSubmitting(false);
  }
};
const handleBuyRent = () => {
  if (!isAuthenticated) {
    navigate("/login");
    return;
  }
  navigate(`/checkout/${property.id}`);
};
if (loading) {
  return <div className={styles.loading}>Завантаження...</div>;
}
if (!property) {
  return <h2 className={styles.notFound}>Об'єкт не знайдено</h2>;
}
const imgUrl = property.img_url?.startsWith('/')
  ? `http://localhost:5050${property.img_url}`
  : property.img_url;
const isAvailable = property.property_status === 'available';
const isSale = property.listing_type === 'sale';
// Тільки звичайні користувачі можуть купувати/орендувати та надсилати запити
const isClient = !user || user.role === 'user';
return (
  <section className={styles.section}>
    <div className={styles.wrapper}>
      <div className={styles.header}>
        <h1 className={styles.title}>Деталі об'єкта</h1>
        <Breadcrumbs />
        {isClient && (
```

```
        <div className={styles.buttons}>
          {isAvailable && (
            <button className={styles.btnOrange} onClick={handleBuyRent}>
              {isSale ? "Купити" : "Орендувати"}
            </button>
          )}
          <button className={styles.btnWhite} onClick={scrollToForm}>
            Запит
          </button>
        </div>
      )}
    </div>
    <div className={styles.content}>
      <div className={styles.imageWrapper}>
        <img src={imgUrl} className={styles.img} alt={property.title} />
        <div className={styles.priceTag}>
          <span>${property.price.toLocaleString()}{!isSale && "/міс"}</span>
          {isClient && (
            <button className={styles.starBtn} onClick={toggleFavorite}>
              <FaStar className={isFavorite ? styles.starFilled : styles.starEmpty} />
            </button>
          )}
        </div>
        {!isAvailable && (
          <div className={styles.soldBadge}>
            {property.property_status === 'sold' ? 'ПРОДАНО' : 'ОРЕНДОВАНО'}
          </div>
        )}
      </div>
      <div className={styles.container}>
        <div className={styles.containerTitle}>
          <span className={styles.typeTag}>{property.property_type}</span>
          <span className={styles.listingTag}>
            {isSale ? "Продаж" : "Оренда"}
          </span>
          <h2 className={styles.houseTitle}>{property.title}</h2>
          <p className={styles.location}>
            <FiMapPin /> {property.address}, {property.city}
          </p>
        </div>
        <div className={styles.detailsRow}>
          <div className={styles.detailBox}>
            <FiSquare /> {property.sqft} м²
          </div>
          <div className={styles.detailBox}>
            <FiHome /> {property.bedrooms} спалень
          </div>
          <div className={styles.detailBox}>
            <FiHome /> {property.bathrooms} ванних
          </div>
        </div>
      </div>
      <hr />
    </div>
    <div className={styles.infoWrapper}>
      <div className={styles.leftBlock}>
        <h3 className={styles.subtitle}>Опис</h3>
        <p className={styles.text}>{property.description}</p>
        {property.about && property.about.length > 0 && (
          <>
            <h3 className={styles.subtitle}>Про об'єкт</h3>
            <ul className={styles.list}>
```

```
            {property.about.map((item, index) => (
              <li key={index} className={styles.listItem}>
                {typeof item === 'string' ? item : item.text}
              </li>
            ))}
          </ul>
        </>
      )}
      {property.amenities && property.amenities.length > 0 && (
        <>
          <h3 className={styles.subtitle}>Зручності</h3>
          <div className={styles.amenities}>
            {property.amenities.map((amenity, index) => (
              <div key={index} className={styles.amenity}>
                <MdDone className={styles.check} />
                {typeof amenity === 'string' ? amenity : amenity.name}
              </div>
            ))}
          </div>
        </>
      )}
      {property.agent && (
        <div className={styles.agentCard}>
          <img
            src={property.agent.img_url?.startsWith('/')
              ? `http://localhost:5050${property.agent.img_url}`
              : property.agent.img_url || '/img/agents/agent1.png'}
            alt={property.agent.name}
            className={styles.agentImg}
          />
          <div className={styles.agentInfo}>
            <span className={styles.agentLabel}>Агент</span>
            <h4>{property.agent.name}</h4>
            <p>{property.agent.role_title}</p>
            <Link to={`/agents/${property.agent.id}`} className={styles.agentLink}>
              Профіль агента
            </Link>
          </div>
        </div>
      )}
    </div>
    <div className={styles.sidebar} ref={formRef}>
      <h3 className={styles.sidebarTitle}>
        {isSale ? "Ціна" : "Оренда/міс"}
      </h3>
      <p className={styles.sidebarPrice}>
        ${property.price.toLocaleString()}
      </p>
      <hr />
      {!isClient ? (
        <div className={styles.notAvailable}>
          <p style={{ color: 'var(--color-gray)', fontSize: '14px' }}>
            {user?.role === 'admin' ? 'Адміністратори' : 'Агенти'} не можуть купувати або орендувати об'єкти
          </p>
        </div>
      ) : isAvailable ? (
        <>
          <button className={styles.buyBtn} onClick={handleBuyRent}>
            {isSale ? "Купити зараз" : "Орендувати зараз"}
          </button>
          <p className={styles.sidebarText}>або залиште запит</p>
          <form onSubmit={handleSubmit} className={styles.form}>
```

```
{isAuthenticated && (
 <p className={styles.autofillNote}>
  Ваші дані заповнено автоматично
 </p>
)}
<input
 type="text"
 placeholder="Ваше ім'я"
 className={styles.input}
 value={formData.name}
 onChange={(e) => setFormData({ ...formData, name: e.target.value })}
/>
{errors.name && <span className={styles.error}>{errors.name}</span>}
<input
 type="email"
 placeholder="Email"
 className={styles.input}
 value={formData.email}
 onChange={(e) => setFormData({ ...formData, email: e.target.value })}
/>
{errors.email && <span className={styles.error}>{errors.email}</span>}
<input
 type="tel"
 placeholder="Телефон"
 className={styles.input}
 value={formData.phone}
 onChange={(e) => setFormData({ ...formData, phone: e.target.value })}
/>
{errors.phone && <span className={styles.error}>{errors.phone}</span>}
{/* Додаткові поля для купівлі */}
{isSale && (
 <div className={styles.extraFields}>
  <label className={styles.checkLabel}>
   <input
    type="checkbox"
    checked={formData.has_mortgage}
    onChange={(e) => setFormData({ ...formData, has_mortgage: e.target.checked })}
   />
   Планую іпотеку
  </label>
  <div className={styles.budgetRow}>
   <input
    type="number"
    placeholder="Бюджет від $"
    className={styles.inputSmall}
    value={formData.budget_from}
    onChange={(e) => setFormData({ ...formData, budget_from: e.target.value })}
   />
   <input
    type="number"
    placeholder="до $"
    className={styles.inputSmall}
    value={formData.budget_to}
    onChange={(e) => setFormData({ ...formData, budget_to: e.target.value })}
   />
  </div>
 </div>
)}
{/* Додаткові поля для оренди */}
{!isSale && (
 <div className={styles.extraFields}>
  <input
```

```
            type="date"
            placeholder="Бажана дата заселення"
            className={styles.input}
            value={formData.desired_move_in}
            onChange={(e) => setFormData({ ...formData, desired_move_in: e.target.value })}
          />
          <select
            className={styles.input}
            value={formData.rent_period}
            onChange={(e) => setFormData({ ...formData, rent_period: e.target.value })}
          >
            <option value="">Термін оренди</option>
            <option value="3">3 місяці</option>
            <option value="6">6 місяців</option>
            <option value="12">1 рік</option>
            <option value="24">2 роки</option>
            <option value="36">3+ роки</option>
          </select>
          <div className={styles.rentOptions}>
            <label className={styles.checkLabel}>
              <input
                type="checkbox"
                checked={formData.has_pets}
                onChange={(e) => setFormData({ ...formData, has_pets: e.target.checked })}
              />
              Є тварини
            </label>
            <input
              type="number"
              placeholder="К-сть мешканців"
              className={styles.inputSmall}
              min="1"
              max="10"
              value={formData.num_residents}
              onChange={(e) => setFormData({ ...formData, num_residents: e.target.value })}
            />
          </div>
        </div>
      )}
      <textarea
        placeholder="Повідомлення"
        rows="3"
        className={styles.textarea}
        value={formData.message}
        onChange={(e) => setFormData({ ...formData, message: e.target.value })}
      />
      {errors.message && <span className={styles.error}>{errors.message}</span>}
      <button type="submit" className={styles.btnOrange} disabled={submitting}>
        {submitting ? "Відправка..." : "Надіслати запит"}
      </button>
    </form>
    {success && <p className={styles.success}>Запит надіслано!</p>}
    {errors.submit && <p className={styles.error}>{errors.submit}</p>}
  </>
) : (
  <div className={styles.notAvailable}>
    <p>Цей об'єкт вже {property.property_status === 'sold' ? 'проданий' : 'орендований'}</p>
    <Link to="/properties" className={styles.btnOrange}>
      Інші об'єкти
    </Link>
  </div>
)}
```

```
        </div>
       </div>
      </div>
     </section>
   );
}
export default Property;
```

inquiries.controller.js

```
const { body } = require('express-validator');
const { db } = require('../config/database');
const { getAgentId } = require('../middleware/auth');
// Валідатори
const inquiryValidation = [
  body('property_id').isInt().withMessage("ID об'єкта обов'язковий"),
  body('name').trim().notEmpty().withMessage("Ім'я обов'язкове"),
  body('email').isEmail().withMessage('Невалідна email адреса').normalizeEmail(),
  body('phone').trim().notEmpty().withMessage("Телефон обов'язковий"),
  body('message').trim().notEmpty().withMessage("Повідомлення обов'язкове")
];
// Контролери
const create = (req, res, next) => {
  try {
    const {
      property_id,
      user_id,
      name,
      email,
      phone,
      message,
      inquiry_type,
      // Для купівлі
      has_mortgage,
      budget_from,
      budget_to,
      // Для оренди
      desired_move_in,
      rent_period,
      has_pets,
      num_residents
    } = req.body;
    // Перевірка чи об'єкт існує
    const property = db.prepare(`
      SELECT id FROM properties WHERE id = ? AND status = 'approved'
    `).get(property_id);
    if (!property) {
      return res.status(404).json({ message: "Об'єкт не знайдено" });
    }
    // Використовуємо user_id з тіла запиту або з авторизованого користувача
    const finalUserId = user_id || (req.user ? req.user.id : null);
    db.prepare(`
      INSERT INTO inquiries (
        property_id, user_id, name, email, phone, message, inquiry_type,
        has_mortgage, budget_from, budget_to,
        desired_move_in, rent_period, has_pets, num_residents
      )
      VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
    `).run(
      property_id,
      finalUserId,
      name,
      email,
```

```
      phone || null,
      message,
      inquiry_type || 'info',
      has_mortgage || 0,
      budget_from || null,
      budget_to || null,
      desired_move_in || null,
      rent_period || null,
      has_pets || 0,
      num_residents || null
    );
    res.status(201).json({
      message: 'Ваш запит надіслано. Агент зв\'яжеться з вами найближчим часом.'
    });
  } catch (error) {
    next(error);
  }
};
const getAll = (req, res, next) => {
  try {
    const { status, page = 1, limit = 20 } = req.query;
    const offset = (page - 1) * limit;
    let query = `
      SELECT i.*, p.title as property_title, p.location as property_location
      FROM inquiries i
      JOIN properties p ON i.property_id = p.id
    `;
    const params = [];
    if (status) {
      query += ' WHERE i.status = ?';
      params.push(status);
    }
    query += ' ORDER BY i.created_at DESC LIMIT ? OFFSET ?';
    params.push(parseInt(limit), offset);
    const inquiries = db.prepare(query).all(...params);
    const countQuery = status
      ? 'SELECT COUNT(*) as total FROM inquiries WHERE status = ?'
      : 'SELECT COUNT(*) as total FROM inquiries';
    const { total } = db.prepare(countQuery).get(...(status ? [status] : []));
    res.json({
      inquiries,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    next(error);
  }
};
const getMy = (req, res, next) => {
  try {
    const agentId = getAgentId(req);
    if (!agentId) {
      return res.status(403).json({ message: 'Ви не є агентом' });
    }
    const inquiries = db.prepare(`
      SELECT i.*, p.title as property_title, p.location as property_location
      FROM inquiries i
      JOIN properties p ON i.property_id = p.id
```

```
      WHERE p.agent_id = ?
      ORDER BY i.created_at DESC
    `).all(agentId);
    res.json({ inquiries });
  } catch (error) {
    next(error);
  }
};
const updateStatus = (req, res, next) => {
  try {
    const { id } = req.params;
    const { status } = req.body;
    if (!['new', 'in_progress', 'resolved'].includes(status)) {
      return res.status(400).json({ message: 'Невалідний статус' });
    }
    // Перевірка прав (агент може змінювати тільки свої запити)
    if (req.user.role !== 'admin') {
      const agentId = getAgentId(req);
      const inquiry = db.prepare(`
        SELECT i.* FROM inquiries i
        JOIN properties p ON i.property_id = p.id
        WHERE i.id = ? AND p.agent_id = ?
      `).get(id, agentId);
      if (!inquiry) {
        return res.status(403).json({ message: 'Ви не можете змінювати цей запит' });
      }
    }
    const result = db.prepare('UPDATE inquiries SET status = ? WHERE id = ?').run(status, id);
    if (result.changes === 0) {
      return res.status(404).json({ message: 'Запит не знайдено' });
    }
    res.json({ message: 'Статус оновлено' });
  } catch (error) {
    next(error);
  }
};
const remove = (req, res, next) => {
  try {
    const result = db.prepare('DELETE FROM inquiries WHERE id = ?').run(req.params.id);
    if (result.changes === 0) {
      return res.status(404).json({ message: 'Запит не знайдено' });
    }
    res.json({ message: 'Запит видалено' });
  } catch (error) {
    next(error);
  }
};
module.exports = {
  inquiryValidation,
  create,
  getAll,
  getMy,
  updateStatus,
  remove
};
```

users.controller.js

```
const { body } = require('express-validator');
const bcrypt = require('bcryptjs');
const { db } = require('../config/database');
// Валідатори
const createUserValidation = [
```

```
  body('email').isEmail().withMessage('Невалідна email адреса').normalizeEmail(),
  body('password').isLength({ min: 6 }).withMessage('Пароль має містити мінімум 6 символів'),
  body('name').trim().notEmpty().withMessage("Ім'я обов'язкове"),
  body('role').isIn(['user', 'agent', 'admin']).withMessage('Невалідна роль'),
  body('phone').optional().trim()
];
const updateUserValidation = [
  body('name').optional().trim().notEmpty(),
  body('phone').optional().trim(),
  body('role').optional().isIn(['user', 'agent', 'admin']),
  body('is_active').optional().isBoolean()
];
// Контролери
const getAll = (req, res, next) => {
  try {
    const { role, page = 1, limit = 20 } = req.query;
    const offset = (page - 1) * limit;
    let query = 'SELECT id, email, name, phone, role, avatar_url, is_active, is_banned, created_at FROM users';
    const params = [];
    if (role) {
      query += ' WHERE role = ?';
      params.push(role);
    }
    query += ' ORDER BY created_at DESC LIMIT ? OFFSET ?';
    params.push(parseInt(limit), offset);
    const users = db.prepare(query).all(...params);
    const countQuery = role
      ? 'SELECT COUNT(*) as total FROM users WHERE role = ?'
      : 'SELECT COUNT(*) as total FROM users';
    const { total } = db.prepare(countQuery).get(...(role ? [role] : []));
    res.json({
      users,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    next(error);
  }
};
const getById = (req, res, next) => {
  try {
    const user = db.prepare(`
      SELECT id, email, name, phone, role, avatar_url, is_active, created_at
      FROM users WHERE id = ?
    `).get(req.params.id);
    if (!user) {
      return res.status(404).json({ message: 'Користувача не знайдено' });
    }
    // Якщо агент - додаємо дані агента
    if (user.role === 'agent') {
      const agent = db.prepare('SELECT * FROM agents WHERE user_id = ?').get(user.id);
      if (agent) {
        user.agent = agent;
      }
    }
    res.json({ user });
  } catch (error) {
    next(error);
```

```
  }
};
const create = async (req, res, next) => {
  try {
    const { email, password, name, phone, role } = req.body;
    // Перевірка унікальності email
    const existing = db.prepare('SELECT id FROM users WHERE email = ?').get(email);
    if (existing) {
      return res.status(409).json({ message: 'Email вже використовується' });
    }
    const passwordHash = await bcrypt.hash(password, 10);
    const result = db.prepare(`
      INSERT INTO users (email, password_hash, name, phone, role)
      VALUES (?, ?, ?, ?, ?)
    `).run(email, passwordHash, name, phone || null, role);
    const userId = result.lastInsertRowid;
    // Якщо роль agent - створюємо запис агента
    if (role === 'agent') {
      db.prepare(`
        INSERT INTO agents (user_id, role_title, description)
        VALUES (?, 'Agent', 'New agent')
      `).run(userId);
    }
    const user = db.prepare(`
      SELECT id, email, name, phone, role, is_active, created_at
      FROM users WHERE id = ?
    `).get(userId);
    res.status(201).json({
      message: 'Користувача створено',
      user
    });
  } catch (error) {
    next(error);
  }
};
const update = async (req, res, next) => {
  try {
    const { id } = req.params;
    const { name, phone, role, is_active, password } = req.body;
    const existing = db.prepare('SELECT * FROM users WHERE id = ?').get(id);
    if (!existing) {
      return res.status(404).json({ message: 'Користувача не знайдено' });
    }
    const updates = [];
    const params = [];
    if (name !== undefined) {
      updates.push('name = ?');
      params.push(name);
    }
    if (phone !== undefined) {
      updates.push('phone = ?');
      params.push(phone);
    }
    if (role !== undefined) {
      updates.push('role = ?');
      params.push(role);
      // Якщо змінюємо на agent - створюємо запис агента
      if (role === 'agent' && existing.role !== 'agent') {
        const agentExists = db.prepare('SELECT id FROM agents WHERE user_id = ?').get(id);
        if (!agentExists) {
          db.prepare(`
            INSERT INTO agents (user_id, role_title, description)
```

```
          VALUES (?, 'Agent', 'New agent')
        `).run(id);
      }
    }
  }
  if (is_active !== undefined) {
    updates.push('is_active = ?');
    params.push(is_active ? 1 : 0);
    // Якщо деактивуємо - видаляємо refresh токени
    if (!is_active) {
      db.prepare('DELETE FROM refresh_tokens WHERE user_id = ?').run(id);
    }
  }
  if (password) {
    const passwordHash = await bcrypt.hash(password, 10);
    updates.push('password_hash = ?');
    params.push(passwordHash);
    // Видаляємо refresh токени при зміні пароля
    db.prepare('DELETE FROM refresh_tokens WHERE user_id = ?').run(id);
  }
  if (updates.length > 0) {
    params.push(id);
    db.prepare(`UPDATE users SET ${updates.join(', ')} WHERE id = ?`).run(...params);
  }
  const user = db.prepare(`
    SELECT id, email, name, phone, role, avatar_url, is_active, created_at
    FROM users WHERE id = ?
  `).get(id);
  res.json({
    message: 'Користувача оновлено',
    user
  });
 } catch (error) {
  next(error);
 }
};
const remove = (req, res, next) => {
 try {
  const { id } = req.params;
  // Не можна видалити себе
  if (parseInt(id) === req.user.id) {
    return res.status(400).json({ message: 'Не можна видалити власний обліковий запис' });
  }
  const result = db.prepare('DELETE FROM users WHERE id = ?').run(id);
  if (result.changes === 0) {
    return res.status(404).json({ message: 'Користувача не знайдено' });
  }
  res.json({ message: 'Користувача видалено' });
 } catch (error) {
  next(error);
 }
};
// Бан/анбан користувача
const toggleBan = (req, res, next) => {
 try {
  const { id } = req.params;
  // Не можна забанити себе
  if (parseInt(id) === req.user.id) {
    return res.status(400).json({ message: 'Не можна забанити себе' });
  }
  const user = db.prepare('SELECT * FROM users WHERE id = ?').get(id);
  if (!user) {
```

```
    return res.status(404).json({ message: 'Користувача не знайдено' });
   }
   // Не можна забанити адміна
   if (user.role === 'admin') {
    return res.status(400).json({ message: 'Не можна забанити адміністратора' });
   }
   const newBanStatus = user.is_banned ? 0 : 1;
   db.prepare('UPDATE users SET is_banned = ? WHERE id = ?').run(newBanStatus, id);
   // Якщо банимо - видаляємо refresh токени
   if (newBanStatus === 1) {
    db.prepare('DELETE FROM refresh_tokens WHERE user_id = ?').run(id);
   }
   res.json({
    message: newBanStatus ? 'Користувача заблоковано' : 'Користувача розблоковано',
    is_banned: !!newBanStatus
   });
  } catch (error) {
   next(error);
  }
};
module.exports = {
 createUserValidation,
 updateUserValidation,
 getAll,
 getById,
 create,
 update,
 remove,
 toggleBan
};
```

auth.js

```
const jwt = require('jsonwebtoken');
const jwtConfig = require('../config/jwt');
const { db } = require('../config/database');
/**
 * Middleware перевірки JWT токена
 * Встановлює req.user з даними користувача
 */
const authenticate = (req, res, next) => {
 const authHeader = req.headers.authorization;
 if (!authHeader || !authHeader.startsWith('Bearer ')) {
  return res.status(401).json({ message: 'Токен авторизації відсутній' });
 }
 const token = authHeader.split(' ')[1];
 try {
  const decoded = jwt.verify(token, jwtConfig.accessSecret);
  // Перевіряємо чи існує користувач і чи активний
  const user = db.prepare(`
   SELECT id, email, name, phone, role, avatar_url, is_active, is_banned
   FROM users WHERE id = ?
  `).get(decoded.userId);
  if (!user) {
   return res.status(401).json({ message: 'Користувача не знайдено' });
  }
  if (!user.is_active) {
   return res.status(403).json({ message: 'Обліковий запис деактивовано' });
  }
  if (user.is_banned) {
   return res.status(403).json({ message: 'Ваш акаунт заблоковано' });
  }
  req.user = user;
```

```
    next();
  } catch (error) {
    if (error.name === 'TokenExpiredError') {
      return res.status(401).json({ message: 'Токен прострочений', code: 'TOKEN_EXPIRED' });
    }
    return res.status(401).json({ message: 'Невалідний токен' });
  }
};
/**
 * Опціональна автентифікація
 * Якщо токен є - встановлює req.user, якщо немає - пропускає далі
 */
const optionalAuth = (req, res, next) => {
  const authHeader = req.headers.authorization;
  if (!authHeader || !authHeader.startsWith('Bearer ')) {
    return next();
  }
  const token = authHeader.split(' ')[1];
  try {
    const decoded = jwt.verify(token, jwtConfig.accessSecret);
    const user = db.prepare(`
      SELECT id, email, name, phone, role, avatar_url, is_active
      FROM users WHERE id = ? AND is_active = 1
    `).get(decoded.userId);
    if (user) {
      req.user = user;
    }
  } catch (error) {
    // Ігноруємо помилки токена - просто не встановлюємо користувача
  }
  next();
};
/**
 * Middleware перевірки ролей
 * @param  {...string} roles - Дозволені ролі
 */
const authorize = (...roles) => {
  return (req, res, next) => {
    if (!req.user) {
      return res.status(401).json({ message: 'Необхідна авторизація' });
    }
    if (!roles.includes(req.user.role)) {
      return res.status(403).json({ message: 'Недостатньо прав доступу' });
    }
    next();
  };
};
/**
 * Перевірка власника ресурсу або admin
 * @param {Function} getOwnerId - Функція що повертає ID власника з req
 */
const checkOwnership = (getOwnerId) => {
  return (req, res, next) => {
    if (!req.user) {
      return res.status(401).json({ message: 'Необхідна авторизація' });
    }
    // Адмін має доступ до всього
    if (req.user.role === 'admin') {
      return next();
    }
    const ownerId = getOwnerId(req);
    if (req.user.id !== ownerId) {
```

```
      return res.status(403).json({ message: 'Ви не є власником цього ресурсу' });
    }
    next();
  };
};
/**
 * Отримання agent_id для поточного користувача
 */
const getAgentId = (req) => {
  if (!req.user) return null;
  const agent = db.prepare('SELECT id FROM agents WHERE user_id = ?').get(req.user.id);
  return agent ? agent.id : null;
};
module.exports = {
  authenticate,
  optionalAuth,
  authorize,
  checkOwnership,
  getAgentId
};
```

articles.controller.js

```
const { body } = require('express-validator');
const { db } = require('../config/database');
// Валідатори
const createArticleValidation = [
  body('title').trim().notEmpty().withMessage("Заголовок обов'язковий"),
  body('subtitle').optional().trim(),
  body('text').optional().trim(),
  body('paragraph').optional().trim(),
  body('conclusion').optional().trim(),
  body('quote').optional().trim(),
  body('list').optional().isArray()
];
// Контролери
const getAll = (req, res, next) => {
  try {
    const { page = 1, limit = 9 } = req.query;
    const offset = (page - 1) * limit;
    const { total } = db.prepare('SELECT COUNT(*) as total FROM articles').get();
    const articles = db.prepare(`
      SELECT
        a.*,
        u.name as author_name
      FROM articles a
      LEFT JOIN users u ON a.author_id = u.id
      ORDER BY a.created_at DESC
      LIMIT ? OFFSET ?
    `).all(parseInt(limit), offset);
    res.json({
      articles,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    next(error);
  }
};
```

```
const getById = (req, res, next) => {
 try {
  const article = db.prepare(`
   SELECT
    a.*,
    u.name as author_name
   FROM articles a
   LEFT JOIN users u ON a.author_id = u.id
   WHERE a.id = ?
  `).get(req.params.id);
  if (!article) {
   return res.status(404).json({ message: 'Статтю не знайдено' });
  }
  // Список пунктів
  article.list = db.prepare(`
   SELECT text FROM article_list_items
   WHERE article_id = ?
   ORDER BY sort_order
  `).all(req.params.id).map(item => item.text);
  res.json({ article });
 } catch (error) {
  next(error);
 }
};
const create = (req, res, next) => {
 try {
  const {
   title, subtitle, text, paragraph, conclusion, quote,
   img_url, middle_image_url, read_time, list = []
  } = req.body;
  const result = db.prepare(`
   INSERT INTO articles (
    author_id, title, subtitle, text, paragraph, conclusion,
    quote, img_url, middle_image_url, read_time
   ) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?)
  `).run(
   req.user.id, title, subtitle || null, text || null, paragraph || null,
   conclusion || null, quote || null, img_url || null,
   middle_image_url || null, read_time || '10 min read'
  );
  const articleId = result.lastInsertRowid;
  // Додаємо пункти списку
  if (list.length > 0) {
   const insertItem = db.prepare(`
    INSERT INTO article_list_items (article_id, text, sort_order)
    VALUES (?, ?, ?)
   `);
   list.forEach((text, idx) => insertItem.run(articleId, text, idx));
  }
  const article = db.prepare(`
   SELECT a.*, u.name as author_name
   FROM articles a
   LEFT JOIN users u ON a.author_id = u.id
   WHERE a.id = ?
  `).get(articleId);
  article.list = list;
  res.status(201).json({
   message: 'Статтю створено',
   article
  });
 } catch (error) {
  next(error);
```

```
  }
};
const update = (req, res, next) => {
  try {
    const { id } = req.params;
    const existing = db.prepare('SELECT * FROM articles WHERE id = ?').get(id);
    if (!existing) {
      return res.status(404).json({ message: 'Статтю не знайдено' });
    }
    const allowedFields = [
      'title', 'subtitle', 'text', 'paragraph', 'conclusion',
      'quote', 'img_url', 'middle_image_url', 'read_time'
    ];
    const updates = [];
    const params = [];
    allowedFields.forEach(field => {
      if (req.body[field] !== undefined) {
        updates.push(`${field} = ?`);
        params.push(req.body[field]);
      }
    });
    if (updates.length > 0) {
      params.push(id);
      db.prepare(`UPDATE articles SET ${updates.join(', ')} WHERE id = ?`).run(...params);
    }
    // Оновлення списку
    if (req.body.list !== undefined) {
      db.prepare('DELETE FROM article_list_items WHERE article_id = ?').run(id);
      const insertItem = db.prepare(`
        INSERT INTO article_list_items (article_id, text, sort_order)
        VALUES (?, ?, ?)
      `);
      req.body.list.forEach((text, idx) => insertItem.run(id, text, idx));
    }
    const article = db.prepare(`
      SELECT a.*, u.name as author_name
      FROM articles a
      LEFT JOIN users u ON a.author_id = u.id
      WHERE a.id = ?
    `).get(id);
    article.list = db.prepare(`
      SELECT text FROM article_list_items WHERE article_id = ? ORDER BY sort_order
    `).all(id).map(item => item.text);
    res.json({
      message: 'Статтю оновлено',
      article
    });
  } catch (error) {
    next(error);
  }
};
const remove = (req, res, next) => {
  try {
    const result = db.prepare('DELETE FROM articles WHERE id = ?').run(req.params.id);
    if (result.changes === 0) {
      return res.status(404).json({ message: 'Статтю не знайдено' });
    }
    res.json({ message: 'Статтю видалено' });
  } catch (error) {
    next(error);
  }
};
```

```
module.exports = {
 createArticleValidation,
 getAll,
 getById,
 create,
 update,
 remove
};
```

faqs.controller.js

```
const { body } = require('express-validator');
const { db } = require('../config/database');
// Валідатори
const faqValidation = [
 body('question').trim().notEmpty().withMessage("Питання обов'язкове"),
 body('answer').trim().notEmpty().withMessage("Відповідь обов'язкова"),
 body('sort_order').optional().isInt({ min: 0 })
];
// Контролери
const getAll = (req, res, next) => {
 try {
  const faqs = db.prepare(`
   SELECT * FROM faqs
   WHERE is_active = 1
   ORDER BY sort_order
  `).all();
  res.json({ faqs });
 } catch (error) {
  next(error);
 }
};
const getAllAdmin = (req, res, next) => {
 try {
  const faqs = db.prepare('SELECT * FROM faqs ORDER BY sort_order').all();
  res.json({ faqs });
 } catch (error) {
  next(error);
 }
};
const create = (req, res, next) => {
 try {
  const { question, answer, sort_order = 0 } = req.body;
  const result = db.prepare(`
   INSERT INTO faqs (question, answer, sort_order, is_active)
   VALUES (?, ?, ?, 1)
  `).run(question, answer, sort_order);
  const faq = db.prepare('SELECT * FROM faqs WHERE id = ?').get(result.lastInsertRowid);
  res.status(201).json({
   message: 'FAQ створено',
   faq
  });
 } catch (error) {
  next(error);
 }
};
const update = (req, res, next) => {
 try {
  const { id } = req.params;
  const { question, answer, sort_order, is_active } = req.body;
  const existing = db.prepare('SELECT * FROM faqs WHERE id = ?').get(id);
  if (!existing) {
   return res.status(404).json({ message: 'FAQ не знайдено' });
```

```
    }
    const updates = [];
    const params = [];
    if (question !== undefined) {
      updates.push('question = ?');
      params.push(question);
    }
    if (answer !== undefined) {
      updates.push('answer = ?');
      params.push(answer);
    }
    if (sort_order !== undefined) {
      updates.push('sort_order = ?');
      params.push(sort_order);
    }
    if (is_active !== undefined) {
      updates.push('is_active = ?');
      params.push(is_active ? 1 : 0);
    }
    if (updates.length > 0) {
      params.push(id);
      db.prepare(`UPDATE faqs SET ${updates.join(', ')} WHERE id = ?`).run(...params);
    }
    const faq = db.prepare('SELECT * FROM faqs WHERE id = ?').get(id);
    res.json({
      message: 'FAQ оновлено',
      faq
    });
  } catch (error) {
    next(error);
  }
};
const remove = (req, res, next) => {
  try {
    const result = db.prepare('DELETE FROM faqs WHERE id = ?').run(req.params.id);
    if (result.changes === 0) {
      return res.status(404).json({ message: 'FAQ не знайдено' });
    }
    res.json({ message: 'FAQ видалено' });
  } catch (error) {
    next(error);
  }
};
module.exports = {
  faqValidation,
  getAll,
  getAllAdmin,
  create,
  update,
  remove
};
```

subscribers.controller.js

```
const { body } = require('express-validator');
const { db } = require('../config/database');
// Валідатори
const subscribeValidation = [
  body('email').isEmail().withMessage('Невалідна email адреса').normalizeEmail()
];
// Контролери
const subscribe = (req, res, next) => {
  try {
```

```
    const { email } = req.body;
    // Перевірка чи вже підписаний
    const existing = db.prepare('SELECT * FROM subscribers WHERE email = ?').get(email);
    if (existing) {
      if (existing.is_active) {
        return res.status(409).json({ message: 'Ви вже підписані на розсилку' });
      }
      // Реактивація підписки
      db.prepare('UPDATE subscribers SET is_active = 1 WHERE email = ?').run(email);
    } else {
      db.prepare('INSERT INTO subscribers (email) VALUES (?)').run(email);
    }
    res.status(201).json({
      message: 'Дякуємо за підписку!'
    });
  } catch (error) {
    next(error);
  }
};
const unsubscribe = (req, res, next) => {
  try {
    const { email } = req.body;
    const result = db.prepare('UPDATE subscribers SET is_active = 0 WHERE email = ?').run(email);
    if (result.changes === 0) {
      return res.status(404).json({ message: 'Email не знайдено в підписках' });
    }
    res.json({ message: 'Ви успішно відписалися' });
  } catch (error) {
    next(error);
  }
};
const getAll = (req, res, next) => {
  try {
    const { active, page = 1, limit = 50 } = req.query;
    const offset = (page - 1) * limit;
    let query = 'SELECT * FROM subscribers';
    const params = [];
    if (active !== undefined) {
      query += ' WHERE is_active = ?';
      params.push(active === 'true' ? 1 : 0);
    }
    query += ' ORDER BY subscribed_at DESC LIMIT ? OFFSET ?';
    params.push(parseInt(limit), offset);
    const subscribers = db.prepare(query).all(...params);
    const countQuery = active !== undefined
      ? 'SELECT COUNT(*) as total FROM subscribers WHERE is_active = ?'
      : 'SELECT COUNT(*) as total FROM subscribers';
    const { total } = db.prepare(countQuery).get(...(active !== undefined ? [active === 'true' ? 1 : 0] : []));
    res.json({
      subscribers,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    next(error);
  }
};
const remove = (req, res, next) => {
```

```
  try {
    const result = db.prepare('DELETE FROM subscribers WHERE id = ?').run(req.params.id);
    if (result.changes === 0) {
      return res.status(404).json({ message: 'Підписника не знайдено' });
    }
    res.json({ message: 'Підписника видалено' });
  } catch (error) {
    next(error);
  }
};
module.exports = {
  subscribeValidation,
  subscribe,
  unsubscribe,
  getAll,
  remove
};
```

contacts.controller.js

```
const { body } = require('express-validator');
const { db } = require('../config/database');
// Валідатори
const contactValidation = [
  body('name').trim().notEmpty().withMessage("Ім'я обов'язкове"),
  body('email').isEmail().withMessage('Невалідна email адреса').normalizeEmail(),
  body('message').trim().notEmpty().withMessage("Повідомлення обов'язкове"),
  body('phone').optional().trim(),
  body('option').optional().trim()
];
// Контролери
const create = (req, res, next) => {
  try {
    const { name, email, phone, option, message } = req.body;
    db.prepare(`
      INSERT INTO contacts (name, email, phone, option, message)
      VALUES (?, ?, ?, ?, ?)
    `).run(name, email, phone || null, option || null, message);
    res.status(201).json({
      message: 'Ваше повідомлення надіслано. Ми зв\'яжемося з вами найближчим часом.'
    });
  } catch (error) {
    next(error);
  }
};
const getAll = (req, res, next) => {
  try {
    const { status, page = 1, limit = 20 } = req.query;
    const offset = (page - 1) * limit;
    let query = 'SELECT * FROM contacts';
    const params = [];
    if (status) {
      query += ' WHERE status = ?';
      params.push(status);
    }
    query += ' ORDER BY created_at DESC LIMIT ? OFFSET ?';
    params.push(parseInt(limit), offset);
    const contacts = db.prepare(query).all(...params);
    const countQuery = status
      ? 'SELECT COUNT(*) as total FROM contacts WHERE status = ?'
      : 'SELECT COUNT(*) as total FROM contacts';
    const { total } = db.prepare(countQuery).get(...(status ? [status] : []));
    res.json({
```

```
      contacts,
      pagination: {
        page: parseInt(page),
        limit: parseInt(limit),
        total,
        pages: Math.ceil(total / limit)
      }
    });
  } catch (error) {
    next(error);
  }
};
const updateStatus = (req, res, next) => {
  try {
    const { id } = req.params;
    const { status } = req.body;
    if (!['new', 'in_progress', 'resolved'].includes(status)) {
      return res.status(400).json({ message: 'Невалідний статус' });
    }
    const result = db.prepare('UPDATE contacts SET status = ? WHERE id = ?').run(status, id);
    if (result.changes === 0) {
      return res.status(404).json({ message: 'Звернення не знайдено' });
    }
    res.json({ message: 'Статус оновлено' });
  } catch (error) {
    next(error);
  }
};
const remove = (req, res, next) => {
  try {
    const result = db.prepare('DELETE FROM contacts WHERE id = ?').run(req.params.id);
    if (result.changes === 0) {
      return res.status(404).json({ message: 'Звернення не знайдено' });
    }
    res.json({ message: 'Звернення видалено' });
  } catch (error) {
    next(error);
  }
};
module.exports = {
  contactValidation,
  create,
  getAll,
  updateStatus,
  remove
};
```

PostPropertySection.jsx

```
import React, { useRef, useState } from "react";
import { useNavigate } from "react-router-dom";
import styles from '../css/PostPropertySection.module.css'
function PostPropertySection() {
    const navigate = useNavigate()
    const formRef = useRef(null);
    const scrollToForm = () => {
      if (formRef.current) {
        formRef.current.scrollIntoView({ behavior: "smooth", block: "start" });
      }
    };
    const [formData, setFormData] = useState({
      fullName: "",
      email: "",
```

```
    phone: "",
    title: "",
    propertyType: "",
    listingType: "",
    location: "",
    address: "",
    price: "",
    bedrooms: "",
    parking: "",
    constructionSqft: "",
    landSqft: "",
    description: "",
    amenities: [],
  })
  const [errors, setErrors] = useState({})
  const [success, setSuccess] = useState(false)
  const amenitiesList = [
    "Сад",
    "Басейн",
    "Охорона",
    "Пральна",
    "Відеонагляд",
    "Камери безпеки",
    "Посудомийка",
    "Інтернет",
    "Ліфт",
    "Джакузі",
    "Сонячні панелі",
    "Гараж",
  ]
  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value })
  }
  const handleCheckbox = (amenity) => {
    setFormData((prev) => ({
      ...prev,
      amenities: prev.amenities.includes(amenity)
        ? prev.amenities.filter((a) => a !== amenity)
        : [...prev.amenities, amenity]
    }))
  }
  const validate = () => {
    const newErrors = {};
    if (!formData.fullName?.trim()) newErrors.fullName = "Введіть ваше ім'я";
    if (!formData.email?.trim()) {
      newErrors.email = "Введіть email";
    } else if (!/\S+@\S+\.\S+/.test(formData.email)) {
      newErrors.email = "Невірний формат email";
    }
    if (!formData.phone?.trim()) {
      newErrors.phone = "Введіть номер телефону";
    } else if (!/^\+?[0-9]{7,15}$/.test(formData.phone)) {
      newErrors.phone = "Невірний формат телефону";
    }
    if (!formData.title?.trim()) newErrors.title = "Введіть назву оголошення";
    if (!formData.propertyType?.trim()) newErrors.propertyType = "Оберіть тип нерухомості";
    if (!formData.listingType?.trim()) newErrors.listingType = "Оберіть тип оголошення";
    if (!formData.location?.trim()) newErrors.location = "Введіть місто";
    if (!formData.address?.trim()) newErrors.address = "Введіть адресу";
    if (!formData.price?.trim()) {
      newErrors.price = "Введіть ціну";
    } else if (isNaN(formData.price)) {
```

```
      newErrors.price = "Ціна має бути числом";
    }
    if (!formData.bedrooms?.trim()) newErrors.bedrooms = "Введіть кількість кімнат";
    if (!formData.parking?.trim()) newErrors.parking = "Введіть кількість паркомісць";
    if (!formData.constructionSqft?.trim()) {
      newErrors.constructionSqft = "Введіть площу житла";
    } else if (isNaN(formData.constructionSqft)) {
      newErrors.constructionSqft = "Має бути числом";
    }
    if (!formData.landSqft?.trim()) {
      newErrors.landSqft = "Введіть площу ділянки";
    } else if (isNaN(formData.landSqft)) {
      newErrors.landSqft = "Має бути числом";
    }
    if (!formData.description?.trim()) newErrors.description = "Введіть опис";
    if (!formData.amenities || formData.amenities.length === 0) {
      newErrors.amenities = "Оберіть хоча б одну зручність";
    }
    return newErrors;
  };
  const handleSubmit = (e) => {
    e.preventDefault()
    const validationErrors = validate()
    if (Object.keys(validationErrors).length > 0) {
      setErrors(validationErrors)
      setSuccess(false)
      return
    }
    setErrors({})
    setSuccess(true)
    console.log('Property submitted:', formData)
    setFormData({
      fullName: "",
      email: "",
      phone: "",
      title: "",
      propertyType: "",
      listingType: "",
      location: "",
      address: "",
      price: "",
      bedrooms: "",
      parking: "",
      constructionSqft: "",
      landSqft: "",
      description: "",
      amenities: [],
    })
  }
  return (
    <section className={styles.section}>
      <div className={styles.wrapper}>
        <div className={styles.container}>
          <h2 className={styles.title}>Розмістити <br /> оголошення</h2>
          <div className={styles.content}>
            <div className={styles.textContainer}>
              <p className={styles.text}>
                Хочете продати або здати в оренду свою нерухомість? <br />
                Заповніть форму нижче, і наші фахівці зв'яжуться з вами.
              </p>
            </div>
            <div className={styles.buttons}>
```

```
            <button className={styles.btnOrange} onClick={scrollToForm}>Додати об'єкт</button>
            <button className={styles.btnWhite} onClick={() => navigate('/about')}>Про нас</button>
          </div>
        </div>
      </div>
      <form ref={formRef} onSubmit={handleSubmit} className={styles.form}>
        <h3 className={styles.subtitle}>Контактна інформація</h3>
        <div className={styles.grid2}>
          <div>
            <input type="text"
              className={styles.input}
              name="fullName"
              placeholder="Ваше повне ім'я"
              value={formData.fullName}
              onChange={handleChange}
            />
            {errors.fullName && <p className={styles.error}>{errors.fullName}</p>}
          </div>
          <div>
            <input type="email"
              className={styles.input}
              name="email"
              placeholder="Email адреса"
              value={formData.email}
              onChange={handleChange}
            />
            {errors.email && <p className={styles.error}>{errors.email}</p>}
          </div>
          <div>
            <input type="text"
              className={styles.input}
              name="phone"
              placeholder="Номер телефону"
              value={formData.phone}
              onChange={handleChange}
            />
            {errors.phone && <p className={styles.error}>{errors.phone}</p>}
          </div>
        </div>
        <h3 className={styles.subtitle}>Інформація про об'єкт</h3>
        <div className={styles.grid2}>
          <div>
            <input type="text"
              className={styles.input}
              name="title"
              placeholder="Назва оголошення"
              value={formData.title}
              onChange={handleChange}
            />
            {errors.title && <p className={styles.error}>{errors.title}</p>}
          </div>
          <div>
            <select
              className={styles.input}
              name="propertyType"
              value={formData.propertyType}
              onChange={handleChange}
            >
              <option value="">Тип нерухомості</option>
              <option value="apartment">Квартира</option>
              <option value="house">Будинок</option>
              <option value="townhouse">Таунхаус</option>
```

```
            <option value="commercial">Комерційна</option>
            <option value="land">Земельна ділянка</option>
          </select>
          {errors.propertyType && <p className={styles.error}>{errors.propertyType}</p>}
        </div>
      </div>
      <div className={styles.grid2}>
        <div>
          <select
            className={styles.input}
            name="listingType"
            value={formData.listingType}
            onChange={handleChange}
          >
            <option value="">Тип оголошення</option>
            <option value="sale">Продаж</option>
            <option value="rent">Оренда</option>
          </select>
          {errors.listingType && <p className={styles.error}>{errors.listingType}</p>}
        </div>
        <div>
          <input type="text"
            className={styles.input}
            name="location"
            placeholder="Місто"
            value={formData.location}
            onChange={handleChange}
          />
          {errors.location && <p className={styles.error}>{errors.location}</p>}
        </div>
        <div>
          <input type="text"
            className={styles.input}
            name="address"
            placeholder="Адреса"
            value={formData.address}
            onChange={handleChange}
          />
          {errors.address && <p className={styles.error}>{errors.address}</p>}
        </div>
        <div>
          <input type="text"
            className={styles.input}
            name="price"
            placeholder="Ціна ($)"
            value={formData.price}
            onChange={handleChange}
          />
          {errors.price && <p className={styles.error}>{errors.price}</p>}
        </div>
      </div>
      <div className={styles.grid2}>
        <div>
          <input type="text"
            className={styles.input}
            name="bedrooms"
            placeholder="Кількість кімнат"
            value={formData.bedrooms}
            onChange={handleChange}
          />
          {errors.bedrooms && <p className={styles.error}>{errors.bedrooms}</p>}
        </div>
```

```
            <div>
              <input type="text"
                 className={styles.input}
                 name="parking"
                 placeholder="Паркомісць"
                 value={formData.parking}
                 onChange={handleChange}
              />
              {errors.parking && <p className={styles.error}>{errors.parking}</p>}
            </div>
          </div>
          <div className={styles.grid2}>
            <div>
              <input type="text"
                 className={styles.input}
                 name="constructionSqft"
                 placeholder="Площа житла (м²)"
                 value={formData.constructionSqft}
                 onChange={handleChange}
              />
              {errors.constructionSqft && <p className={styles.error}>{errors.constructionSqft}</p>}
            </div>
            <div>
              <input type="text"
                 className={styles.input}
                 name="landSqft"
                 placeholder="Площа ділянки (м²)"
                 value={formData.landSqft}
                 onChange={handleChange}
              />
              {errors.landSqft && <p className={styles.error}>{errors.landSqft}</p>}
            </div>
          </div>
          <div>
            <textarea name="description"
              placeholder="Опис об'єкта"
              value={formData.description}
              onChange={handleChange}
              className={styles.textarea}
            />
            {errors.description && <p className={styles.error}>{errors.description}</p>}
          </div>
          <h3 className={styles.subtitle}>Зручності</h3>
          <div className={styles.checkboxGroup}>
            {amenitiesList.map((amenity) => (
              <label key={amenity} className={styles.checkboxLabel}>
                <input type="checkbox"
                   checked={formData.amenities.includes(amenity)}
                   onChange={() => handleCheckbox(amenity)}
                />
                {amenity}
              </label>
            ))}
          </div>
          {errors.amenities && <p className={styles.error}>{errors.amenities}</p>}
          <button className={styles.btnOrangeSubmit} type="submit">Надіслати на модерацію</button>
          {success && <p className={styles.success}>Оголошення успішно надіслано!</p>}
        </form>
      </div>
    </section>
  )}
export default PostPropertySection;
```

PropertySingle.jsx

```
import React from "react";
import Property from "./components/Property";
import RecentlyViewed from "./components/RecentlyViewed";
import Footer from "../../components/Footer";
function PropertySingle() {
  return (
    <>
    <Property />
    <RecentlyViewed />
    <Footer/>
  </>  )}
export default PropertySingle
```

server.js

```
require('dotenv').config();
const express = require('express');
const cors = require('cors');
const helmet = require('helmet');
const cookieParser = require('cookie-parser');
const path = require('path');
const { initDatabase } = require('./config/database');
const { errorHandler, notFoundHandler } = require('./middleware/errorHandler');
const app = express();
const PORT = process.env.PORT || 5000;
// Безпека
app.use(helmet({
 crossOriginResourcePolicy: { policy: 'cross-origin' }}));
// CORS
app.use(cors({
 origin: process.env.FRONTEND_URL || 'http://localhost:3000',
 credentials: true}));
// Парсинг
app.use(express.json());
app.use(express.urlencoded({ extended: true }));
app.use(cookieParser());
// Статичні файли (зображення)
app.use('/uploads', express.static(path.join(__dirname, 'uploads')));
app.use('/img', express.static(path.join(__dirname, '..', 'src', 'assets', 'img')));
// Перевірка здоров'я сервера
app.get('/health', (req, res) => {
 res.json({ status: 'ok', timestamp: new Date().toISOString() });});
// Ініціалізація БД та запуск сервера
const startServer = async () => {
 try {
  await initDatabase();
  console.log('База даних ініціалізована');
  // Підключаємо роути ПІСЛЯ ініціалізації БД
  const routes = require('./routes');
  app.use('/api', routes);
  // 404 handler
  app.use(notFoundHandler);
  // Error handler
  app.use(errorHandler);
  app.listen(PORT, () => {
   console.log(`

╔═══════════════════════════════════════╗
║    EstatePro API Server Started       ║
╠═══════════════════════════════════════╣
║  Port: ${PORT}                        ║
║  Mode: ${process.env.NODE_ENV || 'development'}                ║
```

```
API:  http://localhost:${PORT}/api
```

```
    `);   });
 } catch (error) {
   console.error('Помилка запуску сервера:', error);
   process.exit(1);
 }};
startServer();
module.exports = app;
```

database.js

```
const initSqlJs = require('sql.js');
const fs = require('fs');
const path = require('path');
require('dotenv').config();
const dbPath = path.resolve(__dirname, '..', process.env.DB_PATH || './db/estate.db');
let db = null;
let dbReady = false;
// Ініціалізація бази даних
const initDatabase = async () => {
 if (db && dbReady) return db;
 const SQL = await initSqlJs();
 // Якщо файл БД існує - завантажуємо
 if (fs.existsSync(dbPath)) {
   const buffer = fs.readFileSync(dbPath);
   db = new SQL.Database(buffer);
 } else {
   // Створюємо нову БД
   db = new SQL.Database();  }
 // Увімкнення зовнішніх ключів
 db.run('PRAGMA foreign_keys = ON');
 dbReady = true;
 return db;};
// Збереження БД у файл
const saveDatabase = () => {
 if (db && dbReady) {
   const data = db.export();
   const buffer = Buffer.from(data);
   const dir = path.dirname(dbPath);
   if (!fs.existsSync(dir)) {
     fs.mkdirSync(dir, { recursive: true });    }
   fs.writeFileSync(dbPath, buffer);
 }};// Обгортка для сумісності з better-sqlite3 API
const getWrapper = () => {
 if (!db || !dbReady) {
   throw new Error('Database not initialized. Call initDatabase() first.');  }
 return {
   prepare: (sql) => ({
     run: (...params) => {
       try {
         db.run(sql, params);
         const lastId = db.exec('SELECT last_insert_rowid() as id')[0]?.values[0]?.[0];
         const changes = db.getRowsModified();
         saveDatabase();
         return { lastInsertRowid: lastId, changes };
       } catch (error) {
         console.error('SQL Error:', error.message, '\nSQL:', sql);
         throw error;
       }     },
     get: (...params) => {
       try {
         const stmt = db.prepare(sql);
```

```
        stmt.bind(params);
        if (stmt.step()) {
          const row = stmt.getAsObject();
          stmt.free();
          return row;
        }
        stmt.free();
        return undefined;
      } catch (error) {
        console.error('SQL Error:', error.message, '\nSQL:', sql);
        throw error;
      }     },
    all: (...params) => {
      try {
        const stmt = db.prepare(sql);
        stmt.bind(params);
        const results = [];
        while (stmt.step()) {
          results.push(stmt.getAsObject());
        }
        stmt.free();
        return results;
      } catch (error) {
        console.error('SQL Error:', error.message, '\nSQL:', sql);
        throw error;
      }     }    }),
  exec: (sql) => {
    try {
      db.exec(sql);
      saveDatabase();
    } catch (error) {
      console.error('SQL Error:', error.message, '\nSQL:', sql);
      throw error;
    }    },
  pragma: (statement) => {
    db.run(`PRAGMA ${statement}`);
  } };};
// Експорт з getWrapper як єдина точка доступу до БД
module.exports = {
  initDatabase,
  saveDatabase,
  get db() {
    return getWrapper();
  }};
```