Міністерство освіти і науки України
Криворізький національний університет
Кафедра моделювання та програмного забезпечення

КВАЛІФІКАЦІЙНА РОБОТА

На здобуття ступеня вищої освіти бакалавра

зі спеціальності 121 – Інженерія програмного забезпечення

На тему: Розробка програмного забезпечення агрегації та аналізу оголошень з продажу нерухомості

Засвідчую, що в цій кваліфікаційній роботі немає запозичень із праць інших авторів без відповідних посилань.

Студент гр. ІПЗ-21-1

_____/Д. В. Власов /

Керівник
кваліфікаційної роботи _____ / А. М. Стрюк /

Завідувач кафедри _____ / А. М. Стрюк /

Кривий Ріг
2025

Криворізький національний університет

Факультет: Інформаційних технологій
Кафедра: Моделювання на програмного забезпечення
Ступінь вищої освіти: бакалавр
Спеціальність: 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ:
Зав. кафедри
_____ А.М. Стрюк
«___»_____20__ р.

ЗАВДАННЯ
на кваліфікаційну роботу

Студенту групи ІПЗ-21-1 Власову Денису Вячеславовичу

1. Тема: Розробка програмного забезпечення агрегації та аналізу оголошень з продажу нерухомості
   Затверджено наказом по КНУ № 199 від 10.04.2025
2. Термін подання студентом закінченої роботи «20» червня 2025р.
3. Вихідні дані по роботі: мови програмування Python та JavaScript, мова розмітки HTML, каскадна таблиця стилів CSS, фреймворк Django, реляційна база даних SQLite, локальний сервер Django.
4. Зміст пояснювальної записки (перелік питань, що їх треба розробити): Проаналізувати особливості сучасних агрегаторів нерухомості, обрати архітектуру програмного забезпечення, обґрунтувати вибір технологічного стеку, розробити механізми збору даних із зовнішніх ресурсів, реалізувати функціонал пошуку та фільтрації і аналітики, створити адміністративну панель для керування контентом, проілюструвати інтерфейс веб-сайту.
5. Перелік ілюстративного матеріалу: Модель проектування фреймворку Django, Deployment діаграма проекту, ER-діаграма бази даних, Структура директорій проекту, Кнопки оновлення даних у панелі адміністратора, Інтерфейс пошуку, Робота фільтрів оголошень, Таблиця оголошення у панелі адміністратора, Деякі фільтри таблиці оголошень, Повідомлення про успішність функцій парсингу даних та оновлення курсу валют, Форма для додавання нового оголошення, Оголошення на головній сторінці, Сторінка детального перегляду оголошення, Модальне вікно «Поділитися», Сторінка авторизації, Сторінка реєстрації, Профіль користувача, Сторінка обраного, Аналітична сторінка оголошень, Активна та неактивна кнопка додавання оголошення у обране.

Календарний план:

№	Найменування етапів кваліфікаційної роботи	Термін виконання етапів роботи
1	Визначення теми кваліфікаційної роботи	28.11.2024
2	Вивчення літератури за темою	09.01.2025 - 23.01.2025
3	Планування напрямку рішення завдання, робота над архітектурою ПЗ	01.02.2025 - 07.03.2025
4	Програмна реалізація дипломного проекту	08.03.2025 - 15.05.2025
5	Затвердження результатів програмної реалізації проекту з керівником дипломної роботи	15.05.2025
6	Написання текстової частини роботи	15.05.2025 - 02.06.2025

Дата видачі завдання: «28» листопада 2024 р.
Студент _____ Д.В. Власов
Керівник роботи _____ А.М. Стрюк

РЕФЕРАТ

АГРЕГАТОР НЕРУХОМОСТІ, DJANGO, ПАРСИНГ ДАНИХ, АНАЛІТИКА, ВЕБ-РОЗРОБКА, ІНТЕРАКТИВНИЙ ІНТЕРФЕЙС, ФІЛЬТРАЦІЯ, API, ОНЛАЙН-СЕРВІС, ОБРАНІ ОГОЛОШЕННЯ, СИСТЕМА КЕРУВАННЯ КОНТЕНТОМ.

Пояснювальна записка: 55 с., 1 табл., 20 рис., 3 дод., 7 джерел.

У роботі описано процес створення веб-агрегатора оголошень про продаж квартир у Кривому Розі на основі фреймворку Django. Розроблена система забезпечує збір, зберігання та зручне відображення інформації про нерухомість із різних джерел для кінцевого користувача. Основна мета полягала у створенні сервісу, який спрощує пошук, фільтрацію та аналіз квартир.

Для реалізації поставлених завдань використано Python, Django, HTML, CSS та JavaScript, а зберігання даних здійснюється у реляційній базі SQLite. Система дозволяє отримувати актуальні курси валют із офіційного API, підтримує роботу з обраними оголошеннями, динамічний пошук і аналітику.

Результатом роботи став стабільний веб-сервіс із простим адмініструванням, надійним захистом даних і можливістю масштабування. Сфера застосування — ринок нерухомості, де система допомагає агентствам, ріелторам і покупцям ефективно аналізувати пропозиції квартир.

ABSTRACT

REAL ESTATE AGGREGATOR, DJANGO, DATA PARSING, ANALYTICS, WEB DEVELOPMENT, INTERACTIVE INTERFACE, FILTERING, API, ONLINE SERVICE, FAVORITES, CONTENT MANAGEMENT SYSTEM.

Explanatory note: 55 pages, 1 table, 20 figures, 3 appendices, 7 sources.

The paper describes the process of developing a web aggregator for apartment listings in Kryvyi Rih using the Django framework. The developed system enables the collection, storage, and user-friendly presentation of real estate information from various sources for end users. The main goal was to create a service that simplifies the search, filtering, and analysis of apartment offers.

To achieve these objectives, Python, Django, HTML, CSS, and JavaScript were used, while data storage is handled by the SQLite relational database. The system can retrieve current exchange rates from the official API, supports working with favorite listings, and provides dynamic search and analytics capabilities.

As a result, a stable web service with straightforward administration, reliable data protection, and scalability options has been implemented. The system is intended for the real estate market, assisting agencies, realtors, and buyers in efficiently analyzing apartment offers.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

Слово / словосполучення	Слово / словосполучення Скорочення
(Application Programming Interface) інтерфейс прикладного програмування.	API
(Object-Relational Mapping) технологія відображення даних реляційної бази у вигляді об’єктів мови програмування.	ORM
(Create, Read, Update, Delete) основні операції роботи з даними у базі: створення, читання, оновлення, видалення.	CRUD
(Structured Query Language) мова структурованих запитів до баз даних.	SQL
(Cross-Site Request Forgery) тип атаки, пов’язаний із підробкою міжсайтових запитів.	CSRF
Мова розмітки гіпертексту.	HTML
Раскадна таблиця стилів.	CSS
Технологія асинхронного обміну даними між клієнтом і сервером без перезавантаження сторінки.	AJAX
(Model-Template-View) архітектурна модель проектування додатків у Django.	MTV

ВСТУП

У сучасних реаліях ринок нерухомості переживає стрімкі трансформації, пов’язані з активним впровадженням цифрових технологій. Пошук і купівля житла вже давно перемістилися в онлайн-простір: більшість потенційних покупців відмовилися від традиційних способів, коли потрібно було переглядати оголошення на дошках чи в газетах, і тепер користуються лише інтернет-сервісами. Серед багатьох інформаційних платформ усе більшої популярності набувають спеціалізовані агрегатори, які дають змогу оперативно отримувати найактуальніші пропозиції з різних джерел, зекономити час і зробити процес вибору житла набагато простішим та зручнішим. На ринку Криворіжжя, пошук житла ускладнюється тим, що інформація розміщена на багатьох різних онлайн-ресурсах і потребує додаткових зусиль для зручної навігації та порівняння пропозицій.

Актуальність цієї роботи обумовлена потребою створення інноваційного інструменту для систематизації та оперативного оновлення бази квартир, доступних для продажу у Кривому Розі. Реалізація агрегатора, що поєднує дані з різних джерел, дає змогу автоматизувати моніторинг нових пропозицій, мінімізувати кількість дублікатів та зробити ринок більш прозорим для кінцевих користувачів.

Основною метою цієї роботи є розробка і впровадження веб-сервісу, який забезпечує збирання та обробку оголошень про продаж квартир у Кривому Розі, дозволяє зручно здійснювати пошук, застосовувати фільтри, аналізувати ціни й зберігати цікаві пропозиції для подальшого перегляду. У ході виконання роботи було проаналізовано основні потреби користувачів і існуючі інструменти на ринку нерухомості, спроєктовано архітектуру агрегатора та структуру бази даних, реалізовано механізми автоматичного збору і оновлення оголошень, створено інтерфейс для пошуку, фільтрації та

аналітики квартир, а також впроваджено систему збереження обраних оголошень і обліку користувачів.

Результати цієї роботи можуть знайти застосування у сфері інформаційних онлайн-платформ, що спеціалізуються на наданні користувачам доступу до актуальної та структурованої бази оголошень про продаж житлової нерухомості. Особливо корисним такий сервіс є для локальних ринків, де важливо отримувати саме ті пропозиції, які відповідають географічним та ціновим очікуванням мешканців окремих міст чи регіонів. Розроблений агрегатор здатен забезпечити ефективний інструмент як для приватних осіб, що шукають квартиру для себе чи своєї родини, так і для агентств нерухомості, які працюють із великими обсягами інформації та зацікавлені у швидкому аналізі ринку. Окрім того, платформа може бути інтегрована у комплексні системи для моніторингу та аналітики ринку житлової нерухомості, що дозволяє використовувати її можливості для побудови статистичних звітів, прогнозування тенденцій або створення додаткових сервісів для професіоналів галузі. Таким чином, результати розробки мають універсальний характер і можуть використовуватися як у повсякденному пошуку житла, так і для розширеного аналізу локальних ринків на різних рівнях.

1 АНАЛІТИЧНЕ ОБГРУНТУВАННЯ ТА ПОСТАНОВКА ЗАДАЧІ

1.1 Аналіз проблеми й вибір напряму розробки

Ринок житлової нерухомості у Кривому Розі дедалі більше залежить від цифрових технологій. Шукати квартиру зараз звичніше в інтернеті, а не через знайомих чи газети — майже всі вже користуються спеціальними сайтами та онлайн-платформами. Однак для більшості це означає, що доводиться відкривати одразу кілька різних сайтів, щоразу спочатку шукати потрібні варіанти, переглядати оголошення вручну й порівнювати ціни самостійно. Єдиного зручного сервісу, де можна було б побачити всі актуальні пропозиції разом, досі не вистачає, через це пошук житла стає довгим і часто виснажливим.

Користувачі витрачають багато часу на перегляд схожих пропозицій, витрачають час на порівняння цін у різних районах міста. Все це призводить до перевантаження інформацією і не завжди допомагає знайти дійсно релевантний варіант житла.

Вивчення актуальної ситуації на ринку показало, що є попит на універсальний онлайн-сервіс, який дозволяє в одному місці отримувати структуровану інформацію з різних джерел, ефективно працювати з нею та швидко знаходити потрібні варіанти за заданими критеріями.

1.2 Огляд сучасних теоретичних і практичних рішень

Щоб добре розуміти, які функції мали б бути розроблені у проекті, слід звернутися до аналогічних сайтів агрегаторів нерухомості, та проаналізувати їхні сильні та слабкі сторони. Серед найвідоміших прикладів можна виділити такі ресурси, як Flatfy та Dom.ria (поєднує агрегаторську частину та власну базу оголошень). Ці сервіси вже стали звичними для багатьох людей, які хоча б раз шукали житло в інтернеті.

Flatfy також є одним із лідерів серед агрегаторів оголошень про продаж квартир в Україні. Його головна перевага — дуже велика база, що охоплює різні регіони, включаючи не тільки обласні центри, а й менші міста. Flatfy інтегрує дані з великої кількості джерел, а також дозволяє шукати як первинне, так і вторинне житло. Сервіс має зручний фільтр за багатьма параметрами, є можливість додавати пропозиції у обране, а також налаштовувати підписки на нові об’єкти за критеріями. Водночас серед недоліків платформи можна відзначити досить велику кількість неактуальних або застарілих оголошень. Часто користувачі зустрічають об’єкти, які вже давно зняті з продажу, а інформація по продавцях не завжди достатньо відкрита. Інтерфейс Flatfy не можна назвати простим через те, що він містить багато різних елементів.

Dom.ria — один із найбільш відомих українських порталів для пошуку нерухомості, який поєднує оголошення від агентств, забудовників і приватних осіб. Платформа пропонує зручний пошук із фільтрами за основними параметрами: ціна, район, тип нерухомості, кількість кімнат, площа тощо. Сервіс активно перевіряє фото житла, впроваджує спеціальні позначки для перевірених оголошень, а також надає можливість перегляду об’єкта на карті.

Водночас зустрічаються рекламні блоки, які можуть ускладнювати навігацію по сайту. Інтерфейс Dom.ria достатньо багатофункціональний, але іноді перевантажений додатковими опціями, що ускладнює перше знайомство із сервісом.

Порівняння з Dom.ria та Flatfy дозволяє оцінити, чого саме не вистачає користувачам на існуючих платформах, а які функції є дійсно корисними й повинні бути реалізовані в новому продукті. При створенні власного агрегатора для Кривого Рогу особливий акцент було зроблено на локальному сегменті — уся база даних та аналітика спрямовані виключно на пропозиції цього міста. Це дозволяє надати більш точну та актуальну інформацію, уникнути перевантаження зайвими об’єктами та сконцентруватися на потребах реальних мешканців. Було враховано позитивний досвід конкурентів

— простота інтерфейсу, наявність карти, фільтрів і аналітики — разом із урахуванням недоліків, наприклад, повільного оновлення або складності для новачків.

Окрему нішу серед онлайн-ресурсів займає розроблений агрегатор, орієнтований саме на ринок Кривого Рогу. На відміну від всеукраїнських платформ, цей сервіс цілеспрямовано зосереджується на локальному сегменті, що дозволяє забезпечити більшу точність даних та швидше оновлення пропозицій. Усі інструменти і фільтри налаштовані відповідно до особливостей місцевого ринку та звичних сценаріїв пошуку житла у місті. Простий і зрозумілий інтерфейс допомагає зручно налаштовувати фільтри за районами, вулицями, площею, кількістю кімнат і ціною, що суттєво економить час користувачів. Модуль аналітики дозволяє отримувати статистику по вартості житла у різних районах міста чи по заданих параметрах.

На сайті доступна функція створення обраного списку квартир, щоб зберігати цікаві пропозиції та повертатися до них у будь-який зручний момент. Окремо було реалізовано механізм автоматичного перерахунку ціни в гривнях за актуальним курсом, що важливо для багатьох користувачів, які звикли орієнтуватися у вартості житла у різних валютах. Значна увага приділена простоті для всіх — інтерфейс інтуїтивний, сторінки швидко завантажуються. Для користувачів із досвідом і для тих, хто шукає житло вперше, агрегатор дозволяє отримати повну інформацію щодо об’єкта: опис, фото, контакти продавця та посилання на джерело.

Для того, щоб чіткіше зрозуміти, наскільки розроблений агрегатор відповідає потребам сучасних користувачів та як він виглядає на фоні вже існуючих популярних платформ, нижче наведено порівняльну таблицю з основними характеристиками й функціями кожного з сервісів: Flatfy, Dom.ria та локального агрегатора для Кривого Рогу. Розуміння особливостей аналогічних рішень допоможе у створенні власного веб-додатку.

Таблиця 1.2.1 – Порівняльна таблиця ключових характеристик платформ

№	Критерій порівнювання	Flatfy	Dom.ria	Розроблений агрегатор
1	Наявність детальних описів квартир	Часткова	Повна	Повна
2	Перерахунок цін за актуальним курсом	Відсутнє	Реалізовано	Реалізовано
3	Зрозумілість інтерфейсу	Нижче середньої	Середня	Висока
4	Простота фільтрації та пошуку	Висока	Середня	Висока
5	Можливість збереження оголошень	Реалізовано	Реалізовано	Реалізовано
6	Аналітика цін на нерухомість	Ні	Ні	Так
7	Відсутність рекламних банерів	Ні	Ні	Так
8	Контакти продавця	Ні	Так	Так
9	Вірогідність дублікатів оголошень	Низька	Низька	Низька

Отже, завдяки врахуванню сильних сторін даних платформ і подоланню їхніх недоліків, запропонований сервіс дозволяє закрити основні потреби користувачів та суттєво спрощує процес пошуку житла в місті.

1.3 Визначення вимог до системи та формування завдань

Основна вимога до майбутньої системи агрегації нерухомості полягала у тому, щоб сервіс давав змогу знаходити актуальні пропозиції з різних джерел у зручному для людини вигляді. Інтерфейс пошуку та фільтрації повинен бути зрозумілим для всіх. Крім цього, було передбачено можливість зберігати обрані оголошення, щоб користувач міг повернутися до них у будь-який зручний момент без потреби шукати їх серед інших. Окремий блок вимог

стосувався роботи з цінами, в якому, по задумці, користувачі мають бачити ціну оголошення у доларах США та автоматичний перерахунок вартості у гривнях.. Це означає, що варто було вирішити питання з отриманням актуального курсу долара США через API Національного Банку України.

Усі дії на сайті — від додавання оголошення до обраного до перегляду детальної інформації — мають відбуватися без затримок і без потреби постійного оновлення сторінки. Для аналітики за рівнем привабливості оголошення необхідно надати прості інструменти для аналізу цін на квартири по районах, вулицях, кількості кімнат та інших параметрах.

Безпека персональних даних і захист акаунтів також були визначені як ключові пріоритети. Усі особисті дані мають зберігатися у захищеному вигляді, а реєстрація і вхід — відбуватися через сучасні протоколи автентифікації. Було передбачено можливість гнучкого адміністрування сервісу з боку власників ресурсу, зокрема керування оголошеннями, користувачами й базою даних.

На основі сформульованих вимог було визначено головні завдання для системи. Необхідно реалізувати парсер, який автоматично збирає оголошення з основних сайтів нерухомості; розробити зручний пошуковий інтерфейс із гнучкими фільтрами; впровадити механізми збереження обраних квартир; забезпечити актуальність валютних курсів і коректний перерахунок цін; створити модуль аналітики, що дозволяє легко переглядати середню вартість житла за різними параметрами; впровадити багаторівневий захист даних і просту систему адміністрування. У результаті цього підходу стає можливим створення повноцінного інструменту для комфортного й ефективного пошуку квартир у Кривому Розі, який охоплює всі типові сценарії взаємодії користувача з ринком нерухомості.

2 ПРОЕКТУВАННЯ ТА АРХІТЕКТУРА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Архітектурні принципи й загальна схема проекту

Під час розробки агрегатору нерухомості для Кривого Рогу варто було створити таку архітектуру, яка б у майбутньому дозволяла проекту ефективно і легко масштабуватися. Далі було обрано класичну багаторівневу структуру з чітким розмежуванням функціональних компонентів. Основна ідея полягає в розподілі відповідальності між серверною логікою, шаром даних, клієнтською частиною та модулями інтеграції із зовнішніми ресурсами.

Проект побудовано на основі фреймворку Django, який використовується для розробки веб-додатків. Він дозволяє ефективно організовувати як бекенд-частину, так і роботу зі статичними файлами та підключення зовнішніх API. Архітектура самого фреймворка Django — це модель проектування MTV (див. рис. 2.1).

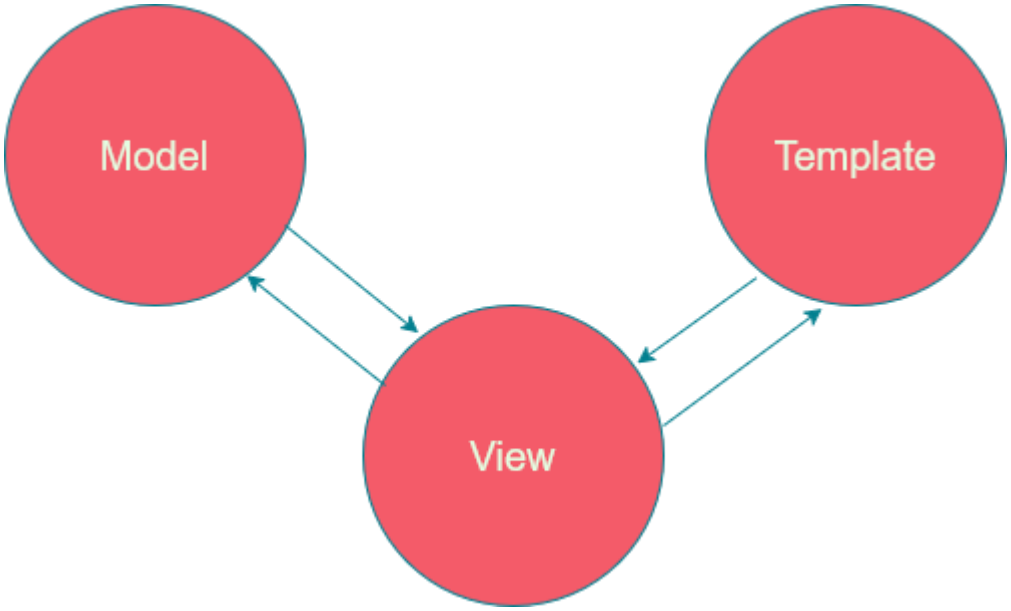

Рисунок 2.1 – Модель проектування фреймворку Django

Model — це рівень моделей, що відповідає за структуру та зберігання даних у базі. Усі сутності, що використовуються в системі, описуються як окремі моделі. Кожна з моделей визначає поля, типи даних і зв’язки між сутностями. За допомогою моделей реалізуються всі CRUD операції із записами у базі даних: створення, читання, оновлення та видалення. Взаємодія із базою у фреймворкі Django відбувається через спеціальний механізм, який дає можливість працювати із даними за допомогою об’єктів Python, а не SQL-запитів — Django ORM (Object-Relational Mapping).

Template — це шар шаблонів, який відповідає за формування інтерфейсу користувача. HTML-сторінки проєкту створюються на основі шаблонів із динамічним підключенням даних, отриманих із моделей. Templates дозволяють виводити списки оголошень, деталізовану інформацію, дані профілю, аналітичні таблиці, а також реагують на фільтри, пошукові запити чи інші дії користувача. Всі шаблони легко змінювати й адаптувати під потреби різних сторінок і пристроїв.

View — це подання, які містять логіку обробки вхідних HTTP-запитів, взаємодіє з моделями, формує дані для передачі у шаблони. Саме тут визначається, які дані необхідно отримати із бази, як їх відфільтрувати, яким чином обробити дії користувача (наприклад, додавання в обране, виконання пошуку, отримання деталей оголошення). View не займається безпосередньо формуванням HTML — він лише передає підготовлені дані у шаблони для відображення.

Модульний підхід архітектури дає змогу зручно інтегрувати нові функціональні блоки, підключати додаткові зовнішні сервіси або змінювати спосіб зберігання даних. Додатково було створено ізольовану логіку для роботи із зовнішніми API (для отримання актуального курсу долара США).

Нижче на рисунку 2.2 представлена Deployment діаграма, яка чітко окреслює взаємодію між основними компонентами: веб-клієнтом, серверною логікою, базою даних, парсерами і зовнішніми API.

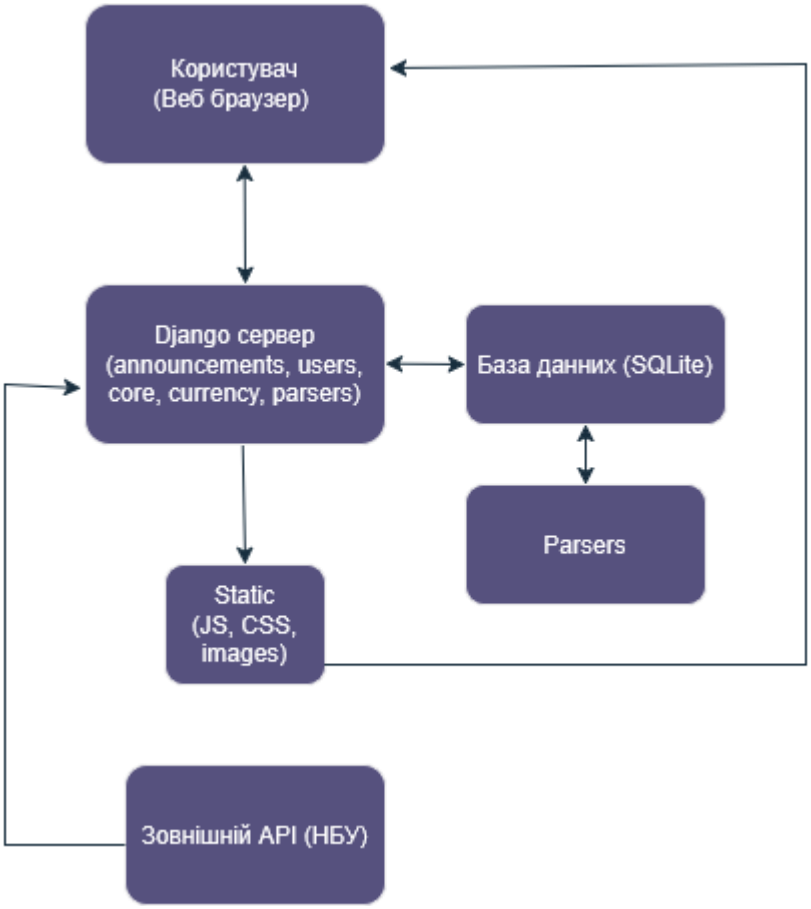

Рисунок 2.2 – Deployment діаграма проекту

Такий підхід забезпечує зрозумілість структури, спрощує тестування й дозволяє ефективно виявляти та локалізувати потенційні проблеми на кожному етапі роботи системи.

2.2 Структура основних компонентів та їх взаємодія

Побудова агрегатора нерухомості для Кривого Рогу ґрунтувалася на чіткому структуруванні функціональних компонентів із урахуванням гнучкості та зручності майбутньої підтримки. Основу системи складає серверна частина, організована на базі Django, яка об’єднує декілька незалежних додатків (аплікацій), кожна з яких відповідає за певний аспект роботи платформи. До основних компонентів було віднесено модулі оголошень, користувачів, роботи

з обраним, аналітики, збору й оновлення даних, а також допоміжні аплікації для керування курсом валюти й організації інформаційних сторінок.

Модуль оголошень відповідає за зберігання, фільтрацію, обробку й аналітику об’єктів нерухомості. Усі операції з оголошеннями, починаючи від додавання нових записів у базу даних і закінчуючи пошуком за різними критеріями, здійснюються саме через цей компонент. Модуль роботи з користувачами забезпечує автентифікацію, реєстрацію, ведення профілю, а також дозволяє користувачу взаємодіяти з іншими частинами системи, наприклад, додавати оголошення в обране чи переглядати особисту аналітику.

Суттєве значення має підсистема збору даних, реалізована у вигляді окремого модуля парсерів. Саме ця частина відповідає за автоматичне отримання нових оголошень із зовнішніх джерел і збереження їх у локальній базі. Взаємодія з парсерами здійснюється переважно через бекенд, що забезпечує централізовану обробку інформації та виключає дублювання. Додатково впроваджено спеціалізований модуль для роботи із валютними курсами, що дає змогу коректно перераховувати вартість квартир у різних валютах та підтримувати актуальність даних.

Статичні ресурси, такі як файли JavaScript, CSS і зображення, винесено в окремий компонент, що оптимізує їхню доставку й підвищує швидкість завантаження сторінок. Уся взаємодія з користувачем відбувається через інтерфейс, побудований на основі шаблонів Django, які отримують динамічні дані через відповідні view-функції. Саме через ці функції здійснюється логічний зв’язок між моделями, шаблонами та статичними файлами.

База даних виконує роль централізованого сховища всієї інформації. Вона зберігає користувачів, оголошення, обране й актуальний курсу долара США. При необхідності структура бази може доповнюватися додатковими таблицями, наприклад, для статистики переглядів чи зберігання архівних оголошень.

2.3 Вибір програмних засобів і технологій

Також потрібно було визначитися із переліком програмних засобів для початку розробки сайту. Необхідний стек технологій мав включати у себе інструменти для парсингу даних, організації зберігання інформації, побудови серверної логіки, формування сучасного веб-інтерфейсу та автоматизації ключових процесів.

Основу всієї системи становить мова програмування Python, що забезпечила гнучкість, простоту розробки й потужні можливості для автоматизації та обробки даних. Саме Python став ключовим інструментом для реалізації як серверної, так і аналітичної логіки. Всі автоматизовані процеси, парсинг інформації, обробка статистики й робота з базою даних виконуються переважно засобами цієї мови, що суттєво спростило розробку та подальшу підтримку проєкту.

Фреймворк Django був обраний через те, що він, якраз, виходить з мови програмування Python. Основна його задача в проекті — створення архітектури сайту та організація чіткого поділу проекту на незалежні модулі. Інструмент ORM цього фреймворку дозволяє зручно працювати з даними та забезпечувати автентифікацію користувачів і обробку складних форм. Як сховище інформації використовується реляційна база даних SQLite, яка легко підключається до Django, дозволяє централізовано керувати структурами даних і швидко мігрувати на інші системи за потреби.

Особливе місце в проекті займає автоматизація збору даних, реалізована через бібліотеки Python — Selenium і BeautifulSoup. Перша дозволяє імітувати дії користувача у браузері для обходу захисту від ботів, а друга — ефективно витягувати й обробляти потрібний контент зі сторінок.

Клієнтська частина створена із застосуванням класичних веб-технологій: HTML, CSS і JavaScript. Динамічне наповнення сторінок організовано через Django-шаблони, а чистий JavaScript використовується для інтерактивних елементів — модальних вікон, фільтрів та кнопок додавання в обране.

Окремий акцент зроблено на аналітичних можливостях: усі розрахунки, статистика та формування зрізів виконуються засобами Python у тісній інтеграції з Django, що дозволяє швидко обробляти значні обсяги даних і формувати наочні звіти для користувача.

2.4 Опис структури даних та бази даних

Під час розробки системи особлива увага приділялася створенню логічної та надійної структури даних, яка б забезпечувала зручність зберігання й ефективну обробку інформації про оголошення, користувачів, обране та допоміжні параметри. Основу зберігання становить реляційна база даних, для якої використано SQLite — просте рішення, що забезпечує швидке розгортання та не потребує складної конфігурації. Усі структури даних формуються на рівні моделей Django, що дозволяє централізовано визначати основні сутності й підтримувати цілісність інформації.

Ключовою таблицею виступає модель «Announcement» (Оголошення), у якій зберігаються всі базові характеристики нерухомості: назва, ціна у валюті й у гривні, кількість кімнат, площа, поверх, район, вулиця, опис, контактні дані продавця, посилання на джерело й фотографії. Модель побудована з урахуванням потреб для фільтрації, сортування та аналітики, що дає змогу легко виконувати вибірки за будь-якими критеріями.

Модель користувача, містить усю необхідну інформацію для ідентифікації та автентифікації: логін, e-mail, ім’я, прізвище, статус. Усі зв’язки між оголошеннями та користувачами реалізовано через таблицю «Favorite» (Обране), що дає змогу фіксувати індивідуальні взаємодії користувачів із об’єктами нерухомості. Для забезпечення коректного відображення цін створено окрему модель для зберігання актуального курсу долара США. Це дозволяє коректно здійснювати перерахунок цін у національній валюті навіть при частих змінах курсу.

Важливою особливістю є чітке розмежування ролей і функцій між таблицями. Наприклад, таблиця оголошень не містить персональних даних користувачів, а зберігає лише необхідну інформацію для відображення й аналізу. З іншого боку, система зберігає мінімальні дані про користувача, що відповідає сучасним вимогам безпеки й конфіденційності. Усі дані, пов'язані з авторизацією та автентифікацією, обробляються стандартними засобами Django, які підтримують надійний захист паролів і контроль доступу.

Завдяки структурі бази даних, що передбачає можливість додавання нових зв'язків, в подальшому можливо розширювати функціонал. Вже на етапі проектування закладено потенціал для впровадження додаткових параметрів, зберігання історії змін, фіксації дій користувачів або інтеграції з іншими сервісами. Внаслідок цього розроблена модель даних дозволяє легко впроваджувати нові сценарії використання й адаптуватися до змін у бізнес-логіці.

Взаємозв'язки між головними сутностями й структура бази даних детально відображені на ER-діаграмі на рисунку 2.3.

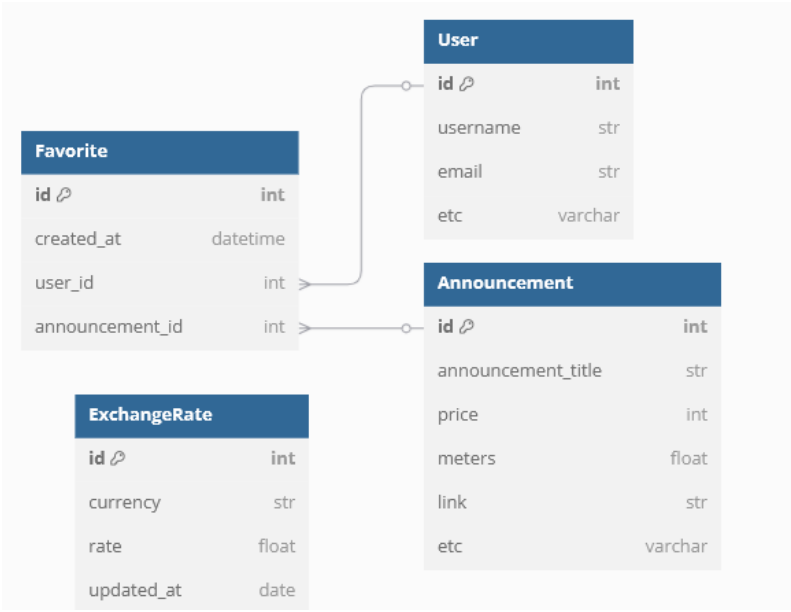

Рисунок 2.3 – ER-діаграма бази даних

Уся взаємодія між таблицями реалізована за допомогою зовнішніх ключів, що забезпечує не лише логічну цілісність даних, а й дає змогу будувати складні багаторівневі запити. Завдяки цьому система здатна швидко отримувати потрібну інформацію, формувати аналітичні вибірки, вести облік взаємодій користувачів із об’єктами нерухомості та підтримувати актуальність курсу валют для кожного оголошення.

2.5 Вимоги до надійності, безпеки й масштабування

Надійність забезпечується за рахунок використання стійкої модульної структури, коли кожен додаток відповідає лише за свою частину функціоналу, а всі дані проходять перевірку на коректність ще на рівні моделей. Для цього у моделях передбачено суворе визначення типів полів, використання обов’язкових атрибутів, а також реалізацію обмежень у вигляді унікальних значень та зовнішніх ключів, що гарантує збереження цілісності інформації під час створення, редагування чи видалення записів. Це дає змогу уникати помилок при роботі з даними й полегшує процес відновлення інформації у разі непередбачених збоїв.

Безпека користувацьких даних підтримується стандартними механізмами фреймворку Django, які включають систему аутентифікації та авторизації, захист від CSRF-атак за допомогою вбудованого middleware, а також автоматичну обробку запитів із перевіркою на SQL-ін’єкції. Доступ до адміністративної панелі обмежується лише для уповноважених осіб, причому всі критичні операції супроводжуються системою повідомлень, яка попереджає про зміни чи можливі помилки.

Питання масштабування системи також закладене у базову структуру. Використання окремих додатків для різних функціональних блоків дозволяє легко відстежувати, доповнювати чи змінювати будь-яку частину проекту. При необхідності система може бути розширена шляхом додавання нових додатків або інтеграції зовнішніх сервісів. Перехід з SQLite на більш

потужну систему керування базами даних здійснюється без зміни бізнес-логіки, оскільки вся взаємодія з даними організована через абстракції ORM.

Окремо варто згадати про моніторинг та обробку можливих помилок. Логування важливих подій дозволяє вчасно виявляти проблеми, швидко знаходити причини збоїв та приймати рішення щодо їхнього усунення. Для контролю коректності роботи основних функцій сайту та швидкого реагування на критичні ситуації впроваджено перевірки на рівні представлень, перевірку валідності форм, а також відображення системних повідомлень для адміністратора через messages framework. Це підвищує прозорість усіх операцій та гарантує, що користувач і адміністратор завжди отримують зрозумілий зворотній зв’язок.

Додатковий рівень захисту реалізовано під час роботи із персональними сторінками користувачів. Доступ до профілю, списку обраного та інших особистих розділів обмежується лише для автентифікованих відвідувачів. У разі спроби перейти на ці сторінки без входу у систему, відвідувач автоматично перенаправляється на сторінку авторизації. Завдяки цьому вдається гарантувати, що персональна інформація недоступна стороннім особам, а будь-які дії у профілі чи збережених даних може виконувати лише власник акаунта.

Гнучкість масштабування підтверджується можливістю розширення існуючих модулів без втручання у базову архітектуру, додавання нових джерел даних або функціоналу для аналізу й обробки інформації. Завдяки цьому проєкт залишається стабільним, надійним та здатним до адаптації під майбутні потреби користувачів і адміністраторів.

3 РЕАЛІЗАЦІЯ, ОСОБЛИВОСТІ ФУНКЦІОНУВАННЯ ТА ЗАСТОСУВАННЯ

3.1 Реалізація основних модулів і логіки роботи

Реалізація агрегатора нерухомості базується на чіткій структурі з окремими функціональними модулями, кожен із яких відповідає за визначений блок задач та забезпечує розширюваність і масштабованість системи. Проект побудовано із дотриманням принципів модульності та розмежування відповідальності, що дало змогу спростити підтримку, й полегшити впровадження нових можливостей у майбутньому (див. рис. 3.1).

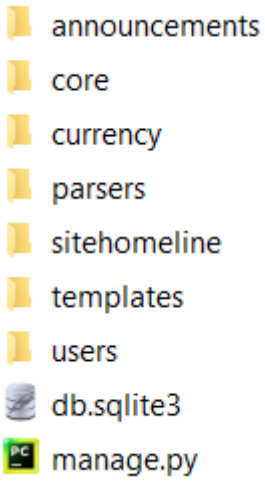

Рисунок 3.1 – Структура директорій проекту

У структурі проєкту було виділено декілька основних додатків (аплікацій), кожен із яких виконує свою унікальну роль. Наприклад:

- announcements — управління оголошеннями про продаж нерухомості;
- users — реєстрація, автентифікація та управління профілем користувачів;
- parsers — модуль збору інформації з зовнішніх ресурсів;

- currency — робота з курсами валют і перерахунок цін у гривню;
- core — реалізація інформаційних сторінок, політик, поширених питань;
- sitehomeline — головна конфігураційна аплікація, яка містить базові налаштування, основні файли маршрутизації (urls.py), параметри налаштування всього проекту, а також слугує “ядром” для взаємодії всіх інших додатків;
- templates — окрема директорія для зберігання спільних шаблонів, зокрема головного шаблону сайту, у якому міститься шапка (header) та футер (footer), що підключаються до всіх сторінок проекту. Це дозволяє централізовано керувати виглядом головних елементів інтерфейсу та забезпечує єдиний стиль для всіх піддодатків.

Завдяки такій організації логіка системи залишилась зрозумілою й прозорою: кожен додаток містить власні моделі, представлення (views), шаблони (templates), окремі скрипти й файли для налаштувань маршрутів (urls.py). У разі потреби до структури додатку можуть додаватися й інші функціональні файли (такі як utils.py, services.py**)** для допоміжних функцій, або для налаштування адміністративного інтерфейсу (admin.p.) Це дозволяє зберігати гнучкість — додавати або змінювати функціонал без впливу на інші частини системи.

На рівні моделей кожного додатку реалізовано структуровані класи, які описують основні сутності проєкту: оголошення, користувачів, обране, курси валют тощо. Кожна модель має власний набір атрибутів, визначених типів і зв’язків. Для гарантування логічної цілісності інформації між моделями встановлено зв’язки через зовнішні ключі.

View-функції та класи відповідають за формування логіки обробки HTTP-запитів, отримання та передачу даних, генерацію відповідей для шаблонів. У більшості додатків застосовано поділ на функціональні групи view, кожна з яких обслуговує окремий сегмент системи: вивід списку

оголошень, деталізацію, формування аналітичних розрахунків, додавання чи видалення з обраного тощо.

Важливою частиною організації є файл конфігурації settings.py, у якому централізовано задаються основні параметри проєкту: підключення додатків, шляхи до шаблонів, налаштування бази даних, обробка статичних файлів та медіафайлів.

Наприклад, ключові фрагменти файлу settings.py в конфігураційній аплікації sitehomeline для цього проєкту мають такий вигляд:

```
INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'announcements',
    'core',
    'currency',
    'parsers',
    'users',
    'django_extensions',
]
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}
TEMPLATES = [
    {
        'BACKEND':
'django.template.backends.django.DjangoTemplates',
        'DIRS': [
            BASE_DIR / 'templates',
        ],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
'django.template.context_processors.request',
'django.contrib.auth.context_processors.auth',
'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]
STATIC_URL = 'static/'
```

```
LOGIN_REDIRECT_URL = 'users:profile'
LOGOUT_REDIRECT_URL = 'home'
LOGIN_URL = 'users:login'
```

Цей фрагмент налаштувань визначає основні параметри роботи Django-проєкту: у списку INSTALLED_APPS перелічено всі активні додатки (як стандартні, так і власні), секція DATABASES задає використання SQLite як основної бази даних, а блок TEMPLATES містить параметри підключення шаблонів і потрібних контекстних процесорів. STATIC_URL визначає адресу для статичних файлів, а параметри LOGIN_REDIRECT_URL, LOGOUT_REDIRECT_URL та LOGIN_URL задають маршрути для перенаправлення користувачів після входу, виходу або при спробі доступу до захищених сторінок, що забезпечує коректну навігацію й зручність використання сайту.

У проекті, побудованому на Django, є файл головної маршрутизації (urls.py), де визначаються шляхи доступу до всіх основних модулів. Це дозволяє централізовано керувати навігацією сайту та забезпечує зручність масштабування у разі додавання нових додатків.

Наприклад, головний urls.py містить такі налаштування:

```
 urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('announcements.urls')),
    path('users/', include('users.urls', namespace="users")),
    path('core/', include('core.urls')),
]
```

Така структура забезпечує зрозумілий розподіл маршрутів між адміністративною панеллю, основними оголошеннями, профілями користувачів та інформаційними сторінками.

Для захисту від несанкціонованого доступу та забезпечення коректної обробки даних використовуються декоратори, middleware та внутрішній механізм Django — messages framework.

Між додатками організовано передачу даних через контекст шаблонів, виклики функцій, імпорт класів або API-запити. Для збору даних використовуються спеціалізовані сервіси на Python.

Особливу увагу було приділено розмежуванню логіки для клієнтської та адміністративної частин. Для роботи адміністратора організовано окремий рівень доступу через Django admin. Розроблені кастомізовані фільтри, що дозволяють швидко управляти даними в межах проєкту.

Окремо організовано роботу зі статичними файлами й медіаконтентом. Для завантаження зображень, їхньої оптимізації та безпечного зберігання використовується окремий каталог media, а статичні ресурси згруповано для ефективної роботи фронтенду. Центральний шаблон, розташований у директорії templates, містить загальну структуру сторінки, шапку та футер, що дозволяє уніфікувати вигляд усіх сторінок і суттєво спрощує підтримку дизайну.

Важливими складовими структури проекту є файли db.sqlite3 та manage.py.

db.sqlite3 — це база даних проекту у форматі SQLite, що зберігає всю основну інформацію про користувачів, оголошення, обране, курси валют та інші сутності. Її вибір обумовлено простотою використання, відсутністю складної конфігурації та ідеальною сумісністю з Django у проектах невеликого та середнього масштабу.

manage.py — це основний керуючий скрипт Django, що дозволяє виконувати основні операції з проектом: запуск локального сервера, застосування міграцій, створення адміністратора, управління базою даних, виконання тестів тощо. Саме цей файл забезпечує гнучке адміністрування проекту на всіх етапах його життєвого циклу.

Логіка взаємодії з базою даних реалізована через Django ORM. це дало змогу мінімізувати використання “сирих” SQL-запитів, централізувати керування міграціями, гарантувати цілісність і надійність збереження даних.

Усі сутності мають власні міграції, що полегшує зміни у схемі бази під час розробки та супроводу проекту.

Для виявлення й обробки помилок реалізовано систему логування, що дозволяє відслідковувати критичні події, фіксувати помилки під час виконання основних функцій і своєчасно діагностувати причини збоїв. Повідомлення для адміністратора формуються через стандартний механізм Django messages, що гарантує наочність і зрозумілість у разі виникнення виняткових ситуацій.

Весь код проєкту організовано так, щоб можна було масштабувати, доповнювати і переносити логіку окремих блоків із мінімальними змінами у загальній структурі. Це дозволяє у майбутньому безболісно додавати нові функціональні додатки, інтегрувати сторонні API або перейти на іншу СУБД без втрати працездатності вже реалізованої системи.

3.2 Механізми збору та оновлення даних

Для розробки агрегатора нерухомості було створено гнучкі механізми збору нових оголошень і підтримки актуальності даних про курси валют. Особливістю реалізованого рішення є те, що всі процеси запуску парсингу зовнішніх ресурсів та отримання курсу валют повністю підконтрольні адміністратору: відповідні дії здійснюються через окремі кнопки в адміністративному інтерфейсі Django (див. рис. 3.2).

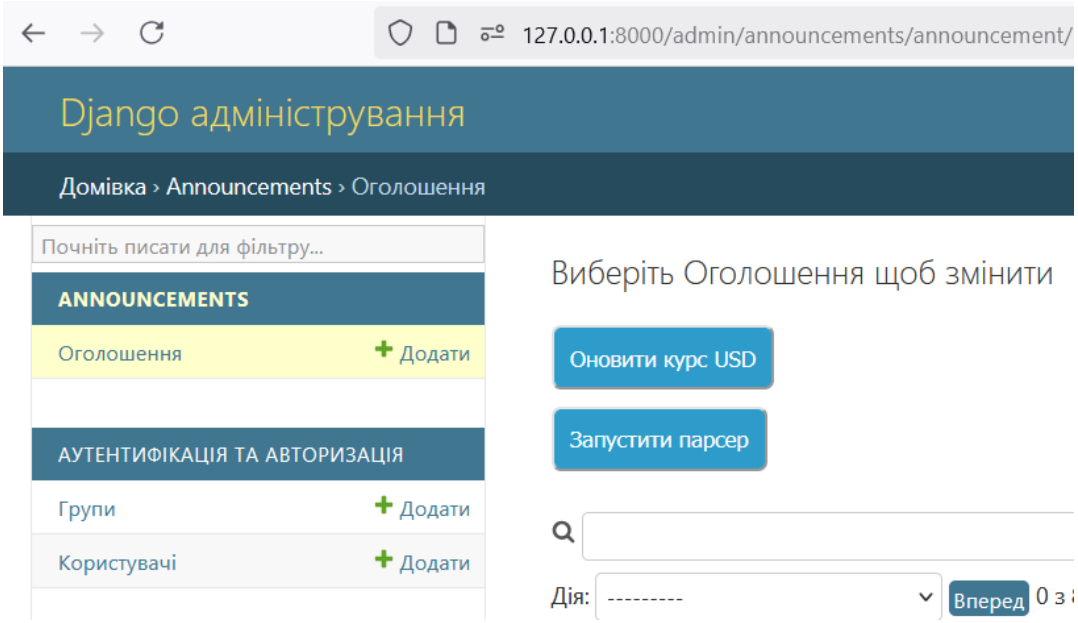

Рисунок 3.2 – Кнопки оновлення даних у панелі адміністратора

Такий підхід забезпечує керованість, зручність і додатковий рівень безпеки при взаємодії із зовнішніми джерелами.

3.2.1 Парсинг сайтів-джерел для отримання оголошень

Для організації процесу збору оголошень у системі створено цілу підсистему в окремому додатку parsers. Її структура передбачає розділення коду на кілька спеціалізованих файлів, кожен з яких виконує свою роль у загальному процесі:

admin.py містить кастомну дію для запуску парсингу прямо з адмін-панелі. Окрім запуску, цей файл відповідає за формування повідомлень адміністратору про результат роботи. Код функції, яка спрацьовує при натисканні кнопки запуску парсингу адміністратором, наведено нижче:

```
from parsers.utils import run_all_parsers
def run_parser(self, request):
    try:
        run_all_parsers()
        messages.success(request, f"Парсинг виконано
успішно!")
    except Exception as e:
        messages.error(request, f"Сталася помилка під час
```

```
виконання парсера: {str(e)}")
    return HttpResponseRedirect("../")
```

В цьому файлі імпортується та викликається функція з файлу utils.py, яка в свою чергу викликає функції парсингу сайтів, додає результат с кожного парсеру до бази даних та видаляє неактуальні оголошення. У проекті парсинг оголошень відбувається лише з сайту rieltor.ua, але в майбутньому, за необхідності, в систему можуть додаватись й інші файли, для збору даних оголошень з інших сайтів. Код наведено нижче:

```
from announcements.models import Announcement
def run_all_parsers():
    from parsers.rieltor_parser import run_parser_rieltor
    # інші парсери
    actual_links = set()
    # Виклик парсерів
    links_rieltor = run_parser_rieltor()
    # виклик інших парсерів
    actual_links.update(links_rieltor)
    # Видалення всіх неактуальних оголошень
Announcement.objects.exclude(link__in=actual_links).delete()
    return actual_links
```

Далі викликаються функції парсерерів. Код функції run_parser_rieltor з файлу rieltor_parser.py представлено у додатку А.

В кожному парсері має викликатись функція save_announcement з файлу services.py для перевірки кожного оголошення на унікальність. Якщо посилання на оригінальне оголошення вже є у базі – оголошення не додається повторно. Код функціїї run_parser_rieltor наведено нижче:

```
def save_announcement(apartments_info):
    if
Announcement.objects.filter(link=apartments_info['link']).ex
ists():
        return False
    try:
        an = Announcement(
announcement_title=apartments_info['announcement_title'],
            district=apartments_info['district'],
            street=apartments_info['street'],
            price=apartments_info['price'],
            rooms=apartments_info['rooms'],
            meters=apartments_info['meters'],
            content=apartments_info['content'],
            images=apartments_info['images'],
            floor=apartments_info['floor'],
            seller_name=apartments_info['seller_name'],
```

```
                phone=apartments_info['phone'],
                link=apartments_info['link'],)
            an.save()
            return True
        except Exception:
            return False
```

Ця система дозволяє організувати запуск парсингу з «одного місця» та обробляти результати у стандартному форматі для всієї системи, скільки б окремих парсерів там не було б.

3.2.2 Оновлення курсу валют через API

Для кінцевого користувача важливо бачити ціни в гривнях, що потребує постійної актуалізації даних про курс валют. У структурі проєкту для цього виділено окремий модуль currency, який об’єднує всі компоненти, пов’язані зі зберіганням, оновленням і використанням курсу іноземних валют.

Оновлення курсу долара відбувається у зручному та керованому режимі. У панелі адміністратора передбачено спеціальну кнопку, яка дозволяє ініціювати процес отримання актуального курсу через офіційний API Національного банку України. Після натискання цієї кнопки система запускає відповідний скрипт, що виконує HTTP-запит до API, обробляє відповідь та вилучає значення курсу долара. Для реалізації цієї логіки у модулі currency створено функцію update_usd_rate (у файлі utils.py), яка займається зверненням до стороннього ресурсу, парсингом отриманих даних і поверненням значення курсу у потрібному форматі. Функція update_usd_rate:

```
def update_usd_rate():
    """Оновлює курс долара США (USD) з офіційного API НБУта
зберігає в модель ExchangeRate"""

    url =
"https://bank.gov.ua/NBUStatService/v1/statdirectory/exchang
e?valcode=USD&json"
    try:
        # Запит до API НБУ
        response = requests.get(url, timeout=10)
        response.raise_for_status()
        data = response.json()
        usd_rate = data[0]['rate']
```

```
            # Оновлення або створення запису курсу USD
            ExchangeRate.objects.update_or_create(
                currency='USD',
                defaults={'rate': usd_rate}
            )
            print(f"Курс USD оновлено: {usd_rate} грн")
        except Exception as e:
            print(f"Помилка при отриманні курсу: {e}")
```

Далі значення нового курсу одразу зберігається у локальній базі даних за допомогою методів моделі Currency (models.py). Для збереження використовується метод update_or_create, що дозволяє або створити новий запис, або оновити вже існуючий для обраної валюти. Це гарантує унікальність даних і забезпечує коректність подальших розрахунків для оголошень. Код моделі ExchangeRate наведено нижче:

```
class ExchangeRate(models.Model):
    currency = models.CharField(max_length=10,
default='USD')
    rate = models.FloatField()
    updated_at = models.DateField(auto_now=True)
```

Весь цей процес інтегровано у адміністративний інтерфейс через кастомну дію, додану у файл admin.py. Вона відповідає не лише за запуск функції оновлення, а й за інформування адміністратора про результат виконання — через стандартний механізм повідомлень messages framework. У разі успішного виконання адміністратор отримує підтвердження із зазначенням нового курсу, у разі виникнення помилки — отримує зрозуміле попередження. Такий підхід підвищує прозорість взаємодії із зовнішніми ресурсами, дозволяє своєчасно відстежувати зміни та швидко реагувати на збої у роботі API. Функція run_update_usd з файлу admin.py, яка запускає оновлення курсу долара США:

```
def run_update_usd(self, request):
    try:
        update_usd_rate()
        messages.success(request, f"Курс USD оновлено")
    except Exception as e:
        messages.error(request, f"Помилка при отриманні
курсу: {str(e)}")
    return HttpResponseRedirect("../")
```

Система оновлення курсу валют у проекті побудована таким чином, що адміністратор має повний контроль над цим процесом: дані отримуються

безпосередньо з офіційного API, зберігаються у структурованій моделі та одразу використовуються для актуалізації цін у всіх оголошеннях. Завдяки цьому система залишається стабільною, а всі зміни легко відстежуються і підтверджуються повідомленнями у адміністративному інтерфейсі.

3.3 Функціонал пошуку та фільтрації

На сайті агрегатора для реалізації функціоналу пошуку та фільтрації було розроблено цілу низку технічних та інтерфейсних рішень, які дають змогу швидко знаходити об’єкти нерухомості відповідно до заданих параметрів.

Пошук організовано через спеціальну форму із полями для введення діапазону ціни, вибору кількості кімнат, а також району або вулиці. Для цих критеріїв, а також для фільтрації по площі, налаштовано відповідні фільтри на сервері. Всі параметри передаються через GET-запит, обробляються у представленнях Django, і в результаті формується вибірка оголошень, яка повністю відповідає вимогам користувача. Система дозволяє об’єднувати декілька критеріїв у межах одного запиту. Нижче на рисунках 3.3 та 3.4 зображені інтерфейс пошуку та робота фільтрів сайту відповідно.

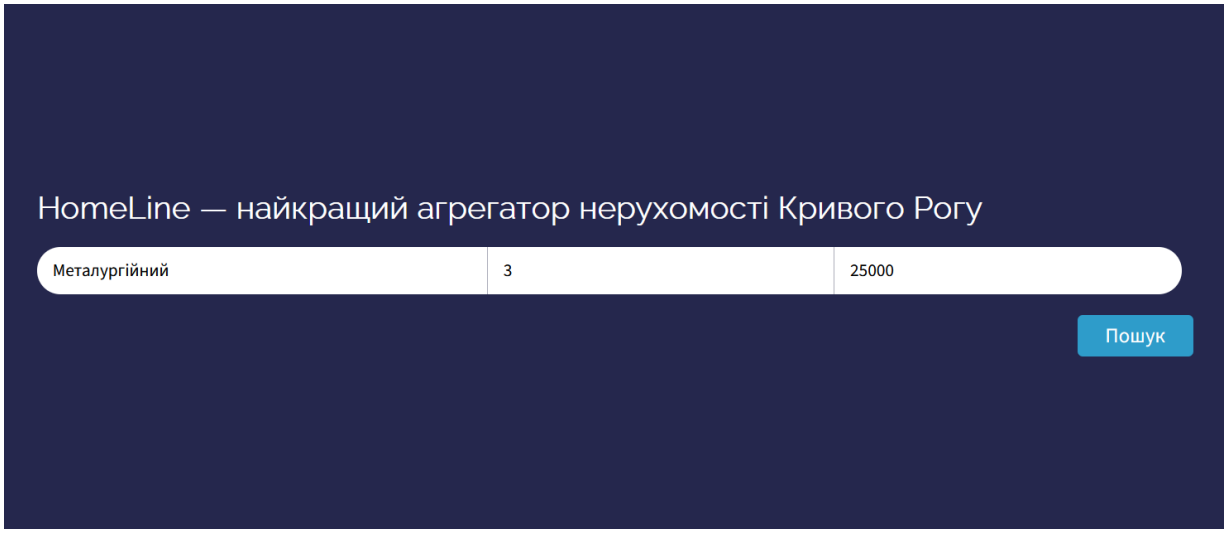

Рисунок 3.3 – Інтерфейс пошуку

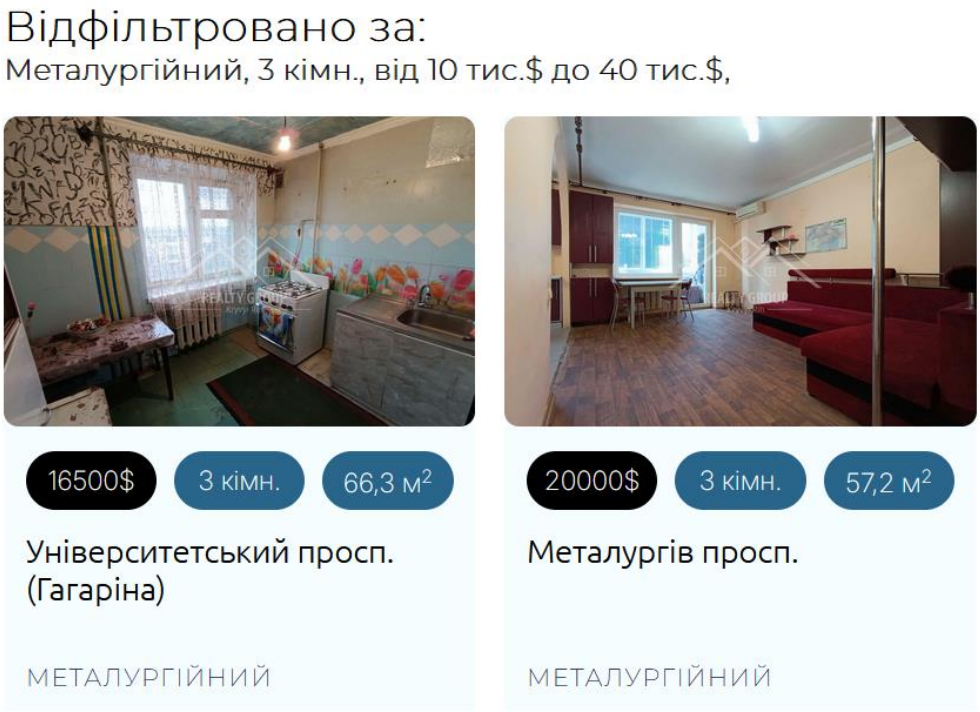

Рисунок 3.4 – Робота фільтрів оголошень

На серверному рівні було реалізовано функцію обробки пошукових параметрів, яка приймає дані із форми, проводить їх валідацію та формує відповідний запит до бази даних через Django ORM. Усі значення аналізуються окремо, і для кожного критерію формується окремий фільтр, який застосовується до початкової вибірки оголошень. Такий підхід дозволяє формувати надзвичайно гнучкі запити. Приклад коду функції-представлення, яка виконує пошук і фільтрацію, наведено нижче:

```
def apartments(request):
    an = Announcement.objects.all()
    # Пошук по району або вулиці
    if location:
        an = an.filter(Q(district__icontains=location) |
Q(street__icontains=location))

    # Фільтрація за ціною (діапазон)
    if price_range:
        if price_range == 'up_to_10000':
            an = an.filter(price__lte=10000)
        elif price_range == 'from10000_to40000':
            an = an.filter(price__gt=10000,
```

```
price__lte=40000)
        elif price_range == 'from40000_to100000':
            an = an.filter(price__gt=40000,
price__lte=100000)
        elif price_range == 'over_100000':
            an = an.filter(price__gt=100000)

    # Фільтрація за кількістю кімнат
    if rooms and rooms.isdigit():
        an = an.filter(rooms=int(rooms))

    # Фільтрація за квадратними метрами
    if meters_range:
        if meters_range == 'up_to25met':
            an = an.filter(meters__lte=25)
        elif meters_range == 'from25met_to40met':
            an = an.filter(meters__gt=25, meters__lte=40)
        elif meters_range == 'from40met_to75met':
            an = an.filter(meters__gt=40, meters__lte=75)
        elif meters_range == 'over_75met':
            an = an.filter(meters__gt=75)
```

Усі значення параметрів, такі як 'from40met_to75met', 'up_to_10000' або 'over_100000', які використовуються для фільтрації оголошень за діапазонами ціни, площі та іншими характеристиками, формуються на стороні клієнта у JavaScript-файлі filter.js (код наведений у додатку Б). Саме цей скрипт відповідає за створення відповідних параметрів при зміні фільтрів у формі та передачу їх до серверної частини у вигляді GET-запиту. Після кожної події автоматично формується AJAX-запит на сервер із поточними параметрами фільтрації, а у відповідь повертається оновлений фрагмент HTML зі списком оголошень. Вміст блоку з результатами підміняється без повного перезавантаження сторінки, що дозволяє користувачам оперативно бачити актуальні результати відповідно до встановлених критеріїв.

Завдяки продуманому підходу до організації пошуку та фільтрації вдається досягти швидкої обробки запитів і отримувати результати за лічені секунди, навіть за великої кількості оголошень. Додатково, фільтри мають логічне групування, що дозволяє користувачам швидко зорієнтуватися серед великої кількості параметрів для пошуку.

3.4 Аналітичний блок проекту

Аналітичний блок проєкту відіграє роль інструмента для розрахунку та узагальнення статистичних показників, отриманих на основі актуальної бази даних оголошень. Основна мета полягає у тому, щоб надати користувачам доступ до об’єктивної інформації про середню вартість квартир у різних районах і на окремих вулицях міста, а також дозволити порівнювати ціни за кількістю кімнат. Було розроблено гнучку структуру, яка дозволяє будувати різноманітні аналітичні зрізи на основі агрегованих даних з усієї бази.

Сторінка аналітики містить інтерактивну форму для вибору критеріїв, таких як район, вулиця, або кількість кімнат. Після вибору параметрів система виконує запит до відповідної моделі та отримує дані для формування інформації з розрахованими показниками. Усі розрахунки виконуються на рівні бекенду засобами Python і Django ORM, що дозволяє отримувати середнє значення ціни. У коді views.py реалізовано функцію, яка обробляє параметри форми, проводить вибірку з бази даних, виконує підрахунки й передає результати у шаблон для відображення. Нижче представлено приклад коду, який підраховує середню вартість житла за квадратний метр по району та вулиці.

```
def analytics(request):
    # Базовий queryset з розрахунком ціни за м²
    base_qs =
Announcement.objects.annotate(price_per_m2=ExpressionWrapper
(F('price') / F('meters'),output_field=FloatField()))

    # Усі унікальні вулиці
    street_list =
Announcement.objects.order_by('street').values_list('street'
, flat=True).distinct()

    # Фільтр по вулиці
    current_street = request.GET.get('street', '').strip()
    if current_street:
        street_qs = base_qs.filter(street=current_street)
    else:
        street_qs = base_qs

    # Середня ціна за м² по районах
    by_district =
base_qs.values('district').annotate(avg_price=Avg('price_per
```

```
_m2')).order_by('district')

    # Середня ціна за м² по вулицях (з урахуванням фільтра)
    by_street =
street_qs.values('street').annotate(avg_price=Avg('price_per
_m2')).order_by('street')
```

Щоб організувати зручний і швидкий пошук потрібної вулиці серед великого списку, на сторінці аналітики реалізовано клієнтський скрипт, який дозволяє фільтрувати записи у таблиці без повторного звернення до сервера. При введенні назви у поле пошуку достатньо натиснути кнопку «Пошук», і у таблиці відображаються лише ті рядки, які відповідають заданому тексту. JavaScript код наведено нижче:

```
document.addEventListener('DOMContentLoaded', () => {
  // Фільтрація таблиці вулиць
  const streetInput =
document.getElementById('street_input');
  const streetTable =
document.getElementById('street_table');
  const streetBtn   = document.getElementById('street-
search-btn');
  if (streetInput && streetTable && streetBtn) {
    // Спочатку приховуємо всі строки
    streetTable.querySelectorAll('tbody tr').forEach(row =>
row.style.display = 'none');
    // Фільтрація по кнопці «Пошук»
    streetBtn.addEventListener('click', () => {
      const filterText =
streetInput.value.trim().toLowerCase();
      streetTable.querySelectorAll('tbody tr').forEach(row
=> {
        const name = row.cells[0].textContent.toLowerCase();
        row.style.display = filterText &&
name.includes(filterText) ? '' : 'none';
      });
    });
  }
```

Користувач може вивести не лише загальні показники, а й отримати вибірку по конкретній вулиці або за кількістю кімнат. Це дозволяє формувати зрозумілу картину цінової ситуації на ринку нерухомості в місті, а також порівнювати пропозиції у різних сегментах. Вся логіка розрахунків виконана на серверній стороні, що забезпечує точність і незалежність від зовнішніх факторів або втручання зі сторони клієнта.

3.5 Модуль оголошень

Модуль оголошень є центральною складовою функціональності сайту, оскільки саме через нього реалізується пошук та перегляд пропозицій квартир на продаж у Кривому Розі. Робота цього модуля розпочинається з головної сторінки, де користувач бачить список усіх наявних оголошень із бази даних. Вивід оголошень організовано так, щоб у першу чергу забезпечити швидкий доступ до найважливіших параметрів кожної квартири: ціни, площі, кількості кімнат, району та адреси.

Особливу увагу було приділено структурі картки оголошення: усі ключові характеристики згруповано разом. Для зручності користування до кожної картки оголошення додано фільтри у вигляді кнопок, які дозволяють уточнювати результати пошуку та швидко знаходити саме ті пропозиції, які відповідають заданим критеріям.

Всі оголошення виводяться у вигляді карток, які формуються динамічно за допомогою шаблонізатора Django. Код основного циклу рендерингу оголошень наведений у додатку В.

При переході на сторінку оголошення користувач отримує повну інформацію про вибрану пропозицію. Тут розміщуються фотографії, розширені характеристики, контактні дані продавця, детальний опис, а також інтерактивні елементи для взаємодії — наприклад, кнопка "Додати в обране" чи "Поділитися".

3.6 Організація роботи з обраними оголошеннями

Функція роботи з обраними оголошеннями реалізована для підвищення зручності користувача під час навігації та аналізу великої кількості пропозицій на сайті. Механізм “Обране” дає змогу швидко відмічати найбільш цікаві квартири, зберігати їх у власному профілі та оперативно повертатися до перегляду або порівняння у зручний момент. Вся логіка реалізована таким

чином, що відзначення чи видалення оголошення з обраного відбувається без затримок, а зміни одразу відображаються у інтерфейсі.

На рівні моделей системи створено окрему сутність, яка пов’язує користувача із обраними оголошеннями. Для цього використовується допоміжна модель, у якій зберігаються ID користувача та ID відповідного оголошення. Завдяки такій структурі забезпечується швидкий пошук, та робиться неможливим дублювання даних про вибрані об’єкти для одного користувача. Нижче наведено код моделі обраного:

```
class Favorite(models.Model):
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE,related_name='favorites')
    announcement = models.ForeignKey(Announcement,
on_delete=models.CASCADE, related_name='favorited_by')
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        # забороняємо користувачу додавати одне і те саме
оголошення в обране двічі
        unique_together = ('user', 'announcement')
```

У файлі views.py написано код, для додавання та видалення оголошення з обраного. Для інтерактивної взаємодії застосовано AJAX-запити, що дозволяє змінювати статус обраного без оновлення всієї сторінки. При натисканні на відповідну іконку або кнопку JavaScript-код ініціює запит до серверної функції, яка додає або видаляє відповідний запис у моделі Favorite. У разі успішної зміни стану елемент інтерфейсу одразу змінює свій вигляд, відображаючи актуальний статус. Ось приклад реалізації представлення для додавання у обране:

```
@login_required
def favorite_announcement(request):
    an_id = request.POST.get('an_id')
    announcement = get_object_or_404(Announcement,
id=an_id)
    fav, created =
Favorite.objects.get_or_create(user=request.user,
announcement=announcement)
    if not created:
        fav.delete()
        action = 'removed'
    else:
        action = 'added'
```

```
    return JsonResponse({'status': 'ok', 'action': action})
```

Аналогічно реалізовано функцію для видалення оголошення зі списку обраного. Уся ця логіка інтегрована у шаблони через динамічні елементи, які підсвічують або приховують відповідну кнопку залежно від статусу. Для підтримки зручного керування обраними оголошеннями у профілі користувача реалізовано окрему сторінку, на якій відображаються всі додані до обраного квартири з можливістю переходу до детального перегляду.

Інтерактивна робота з обраними реалізована через клієнтський JavaScript, що забезпечує відправлення AJAX-запиту та оновлення інтерфейсу без перезавантаження сторінки. Стисло ця логіка виглядає наступним чином:

```
document.addEventListener('DOMContentLoaded', () => {
const csrftoken = getCookie('csrftoken');
document.querySelectorAll('.fav-but').forEach(btn => {
btn.addEventListener('click', async () => {
// Формування POST-запиту із ідентифікатором оголошення
// та CSRF-токеном
// ...
// Отримуємо відповідь — додаємо або прибираємо з обраного
// Оновлюємо вигляд кнопки і статус
});
});
});
```

Обрані об’єкти зберігаються для кожного користувача навіть після виходу з облікового запису чи оновлення сторінки. Це забезпечується централізованим зберіганням даних у базі і унеможливлює втрату інформації.

3.7 Адміністративні можливості й роль користувача

На сайті було реалізовано для забезпечення гнучкого й надійного управління всіма даними, користувачами та функціональними модулями системи панель адміністратора. В основі лежить вбудований адміністративний інтерфейс Django admin, який дозволяє безпосередньо працювати з оголошеннями та користувачами, а також парсити данні та оновлювати курс долара США.

Адміністратор отримує доступ до захищеної панелі, де відображаються всі основні моделі проекту у вигляді інтерактивних таблиць. Звідси легко

додавати, редагувати та видаляти оголошення та переглядати публічні дані користувачів. Для підвищення зручності було реалізовано додаткові фільтри, поля для пошуку і сортування. Кастомізовані форми адміністративного інтерфейсу дозволяють швидко змінювати атрибути оголошення, оновлювати фотографії, працювати з описами й цінами без складних маніпуляцій. Нижче наведено приклад коду, який описує конфігурацію адміністративного інтерфейсу для моделі оголошень у файлі admin.py:

```
@admin.register(Announcement)
class AnnouncementAdmin(admin.ModelAdmin):
    list_display = ('announcement_title', 'district',
'price', 'price_uah', 'rooms', 'meters', 'floor',
'seller_name')
    list_filter = (PriceRangeFilter, 'rooms', 'floor',
'district', 'street')
    search_fields = ('announcement_title', 'district',
'street', 'seller_name', 'price')
    readonly_fields = ('price_uah',)
    fieldsets = (
        ('Основна інформація', {
            'fields': ('announcement_title', 'district',
'street', 'price', 'price_uah', 'rooms', 'meters', 'floor')
        }),
        ('Продавець та деталі', {
            'fields': ('seller_name', 'phone', 'content',
'link')
        }),
        ('Фото', {
            'fields': ('images',)
        }),
    )
```

Інтерфейс таблиці оголошення та частина інтерфейсу фільтрів таблиці наведені на рисунках 3.5 та 3.6 відповідно.

	НАЗВА	РАЙОН	ВАРТІСТЬ $	ВАРТІСТЬ ₴	КІМНАТИ	КВ. МЕТРИ	ПОВЕРХ	ІМ'Я ПРОДАВЦЯ
☐	бул Європейский, 6	Довгинцівський	15000	622891,5	1	34,0	1 / 9	Панченко Дмитро
☐	Едуарда Фукса вул. (Тухачевського), 87	Покровський	19500	809758,95	2	43,0	2 / 4	Панченко Дмитро
☐	Сахарова вул., 17	Довгинцівський	16500	685180,65	2	51,6	6 / 9	Панченко Дмитро
☐	Аненко, 4	Центрально-Міський	38000	1577991,8	3	62,0	5 / 5	Панченко Дмитро
☐	пр-т Гагаріна, 30	Металургійний	24000	996626,4	1	36,4	3 / 3	Панченко Дмитро
☐	Панаса Мирного вул., 7	Інгулецький	25000	1038152,5	2	55,0	3 / 3	Чаус Володимир
☐	Качалова вул., 8	Саксаганський	43000	1785622,3	3	62,3	2 / 4	Федоренко Денис

Рисунок 3.5 – Таблиця оголошення у панелі адміністратора

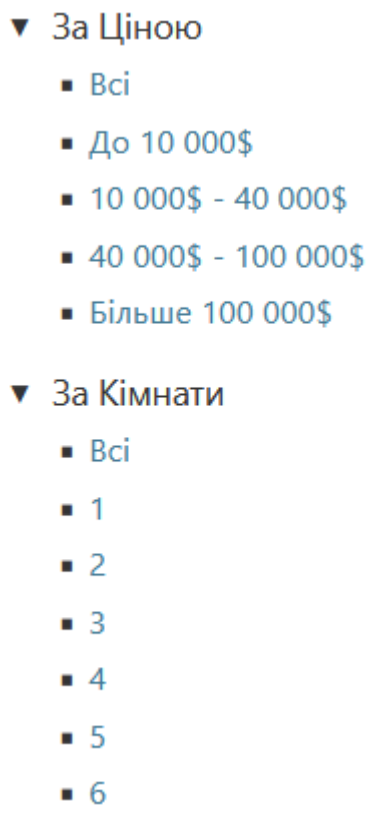

Рисунок 3.6 – Деякі фільтри таблиці оголошень

За необхідності адміністратор може вручну коригувати або видаляти оголошення з некоректними даними. Додатково у панелі адміністратора впроваджено функції для запуску парсингу нових оголошень із зовнішніх джерел та оновлення курсу долара за допомогою спеціальних кнопок. Після виконання цих дій система автоматично формує повідомлення про результат — підтвердження або попередження, що забезпечує прозорість і контроль над кожною операцією. На рисунку 3.7 зображено повідомлення, які виникають внаслідок запуску парсеру даних та оновлення курсу долара США.

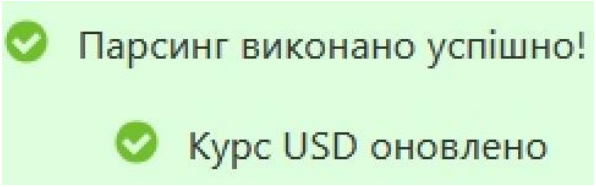

Рисунок 3.7 – Повідомлення про успішність функцій парсингу даних та оновлення курсу валют

Адміністратор має можливість додавати нові оголошення безпосередньо через панель керування. Для цього передбачено зручну форму, в якій можна швидко внести всі основні параметри: назву квартири, опис, ціну, район, адресу, контактні дані, посилання та додати фотографії. Такий підхід дозволяє оперативно поповнювати каталог актуальними пропозиціями — нове оголошення з'являється на сайті одразу після збереження, і користувачі можуть знайти його через пошук або фільтрацію. На рисунку 3.8 зображена форма для додавання нового оголошення.

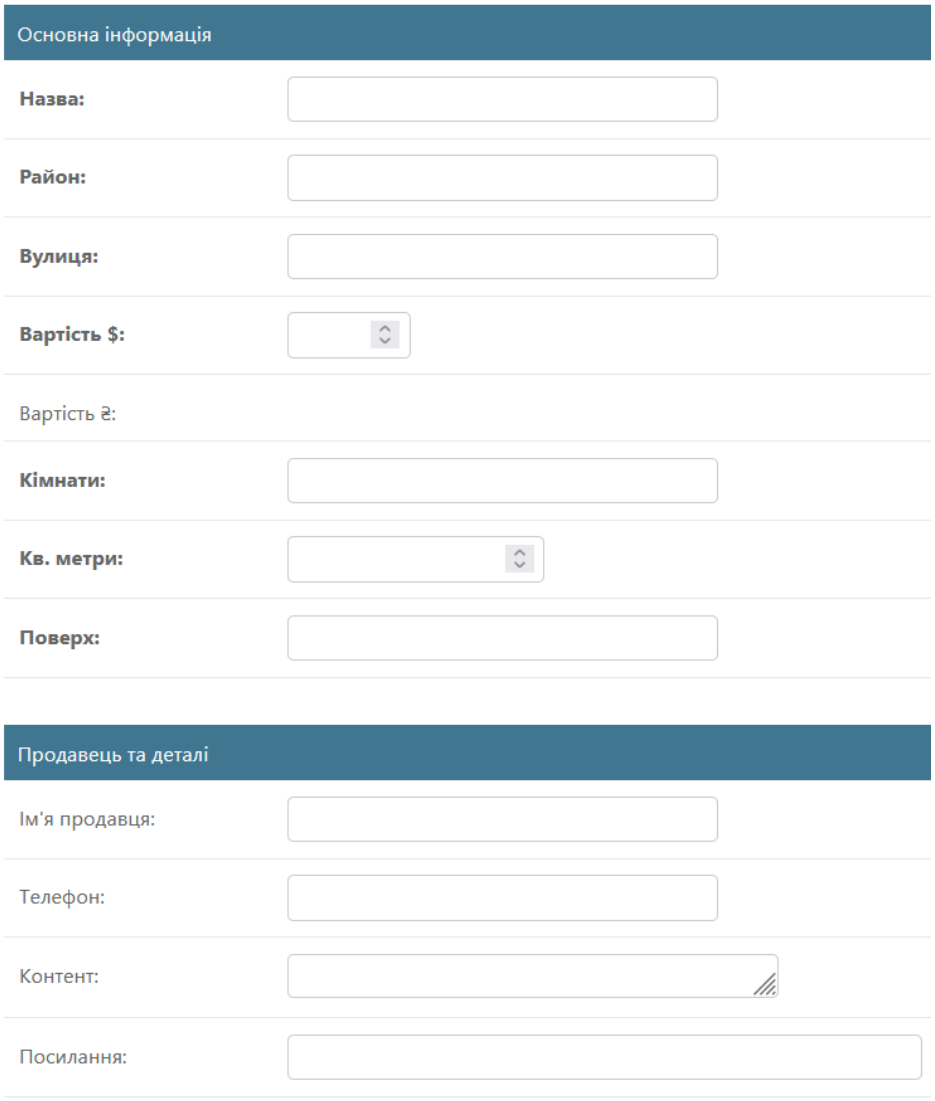

Рисунок 3.8 – Форма для додавання нового оголошення

Завдяки кастомізації панелі управління вдається розділяти права доступу між адміністраторами й звичайними користувачами. Для останніх реалізовано окремий профіль, у якому зосереджені лише персональні дані, можливість додавати оголошення у обране, переглядати аналітику та працювати з фільтрами. Адміністратор же бачить повний перелік об’єктів і керує ними у кілька кліків.

Повний цикл адміністрування забезпечується як класичними засобами Django admin, так і додатковими перевірками на рівні моделей, представлень та форм. Повідомлення про результат кожної дії відображаються через стандартний messages framework, що дозволяє одразу інформувати адміністратора або користувача про успішність чи проблеми під час роботи з даними.

3.8 Інтерфейс користувача

Головна сторінка сайту містить яскраву панель пошуку у верхній частині, що дає змогу обрати район, кількість кімнат та ціновий діапазон. Оголошення подаються у вигляді акуратних карток, кожна з яких містить фотографію квартири, ціну у гривнях, основні характеристики та короткий опис. На одній сторінці зібрано десятки пропозицій, і користувач може одразу побачити, які квартири відповідають його потребам. На рисунку 3.9 зображено деякі оголошення головної сторінки.

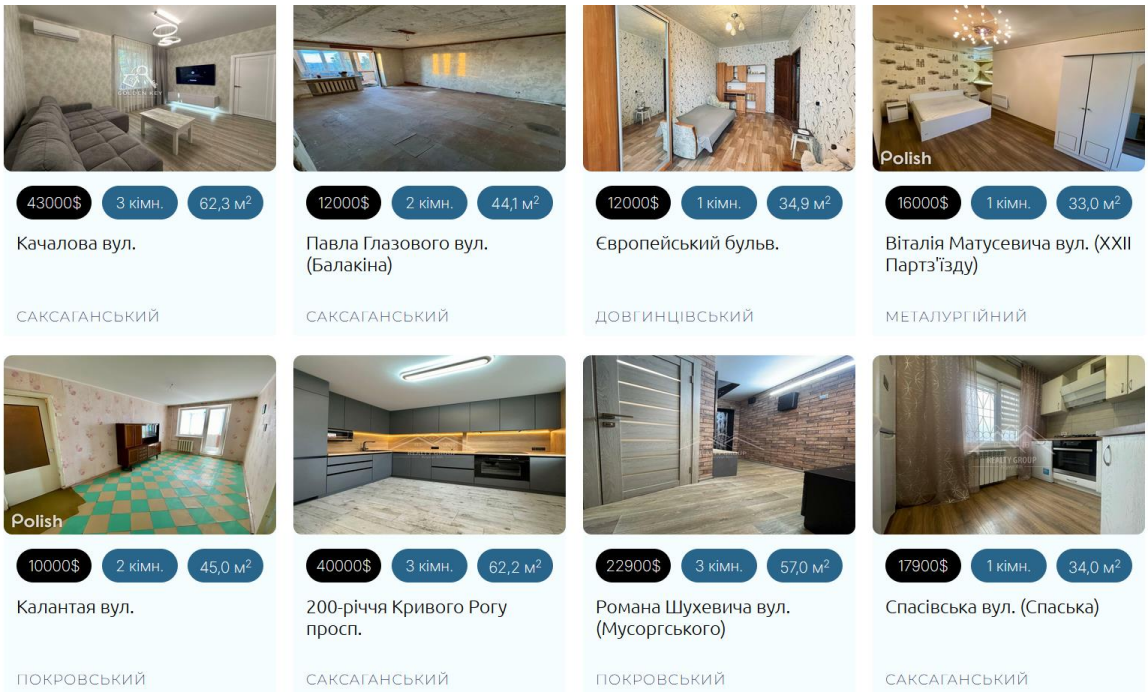

Рисунок 3.9 – Оголошення на головній сторінці

Для кожного оголошення реалізовано окрему сторінку з розширеним описом. Тут виведено не лише повний текст оголошення, а й докладні параметри — площа, поверх, контактна інформація продавця, фото-галерея об’єкта. На сторінці оголошення користувачеві доступна функція додавання квартири у обране: це дозволяє створювати персональний список пропозицій для подальшого порівняння або повернення. На рисунку 3.10 представлено сторінку детального перегляду оголошення.

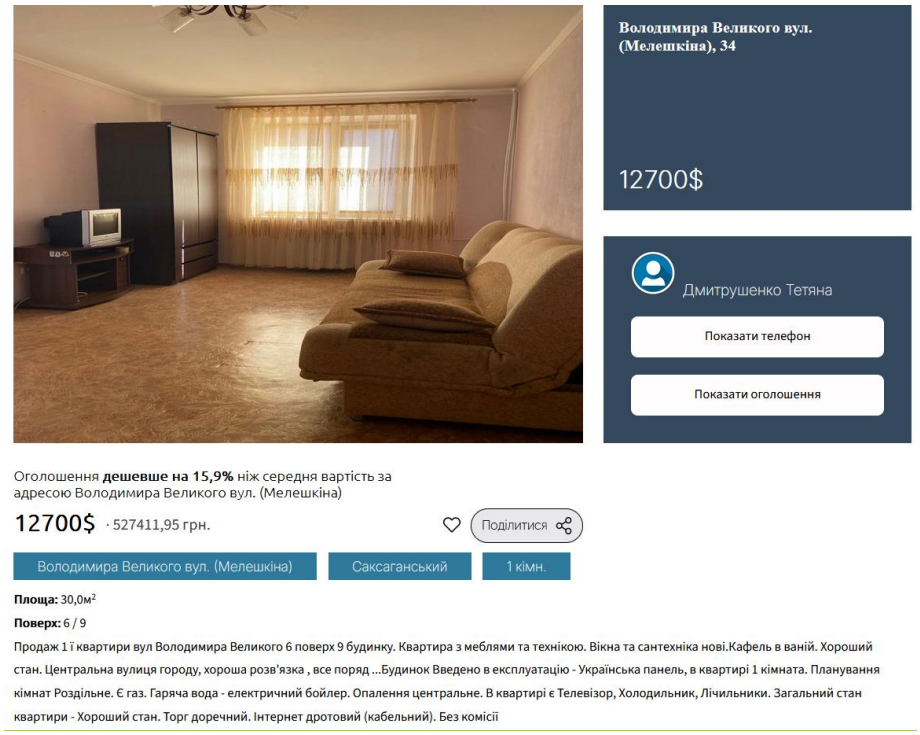

Рисунок 3.10 – Сторінка детального перегляду оголошення

Також на сторінці детального перегляду оголошення зроблено окрему кнопку «Поділитися», яка дозволяє швидко скопіювати посилання або відправити його через популярні соціальні мережі (див. рис. 3.11).

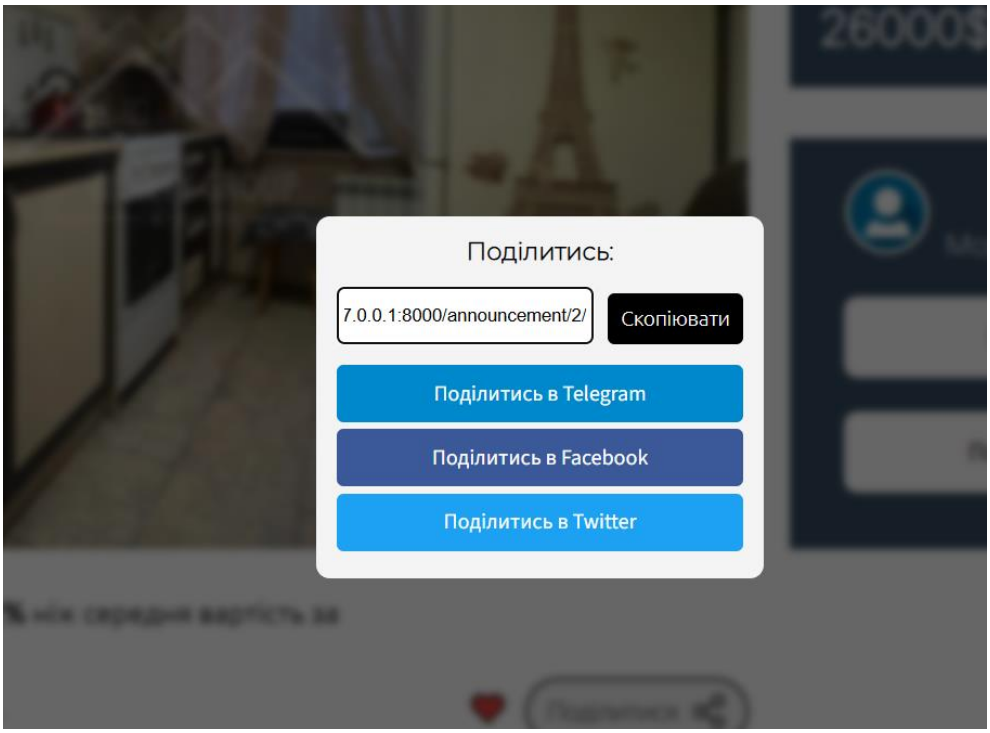

Рисунок 3.11 – Модальне вікно «Поділитися»

Перш ніж скористатися персональним кабінетом, користувач проходить процедуру реєстрації або авторизації. Форми створені максимально простими, поля розташовані логічно, передбачені підказки й зрозумілі повідомлення про помилки у разі некоректного заповнення. Сторінки входу й реєстрації виконані у мінімалістичному стилі — зайвих елементів немає, усе працює швидко і без перевантаження зайвою інформацією. На рисунках 3.12 та 3.13 зображені сторінки авторизації та реєстрації відповідно.

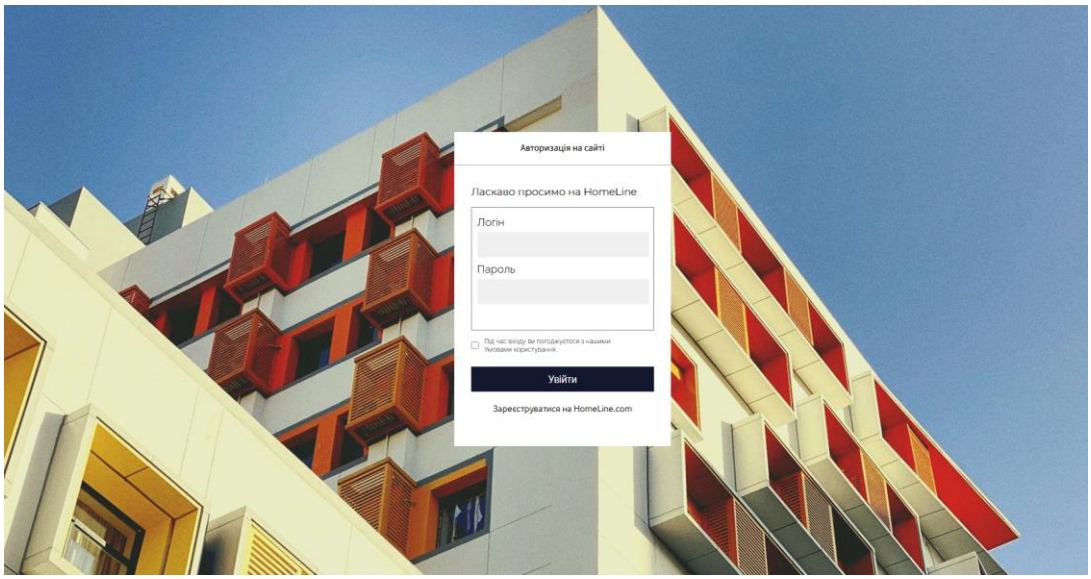

Рисунок 3.12 – Сторінка авторизації

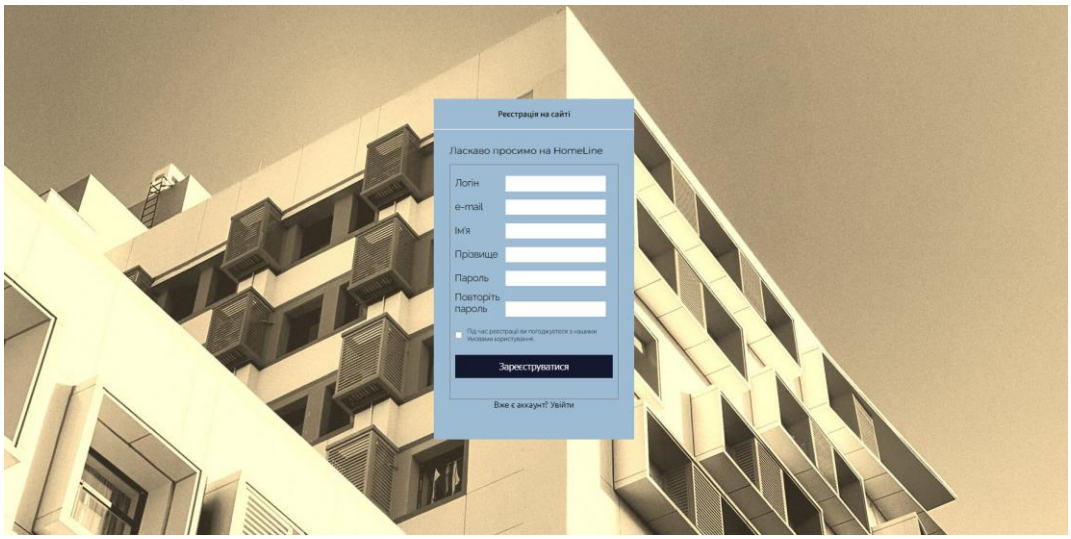

Рисунок 3.13 – Сторінка реєстрації

На рисунку 3.14 зображено персональний профіль користувача

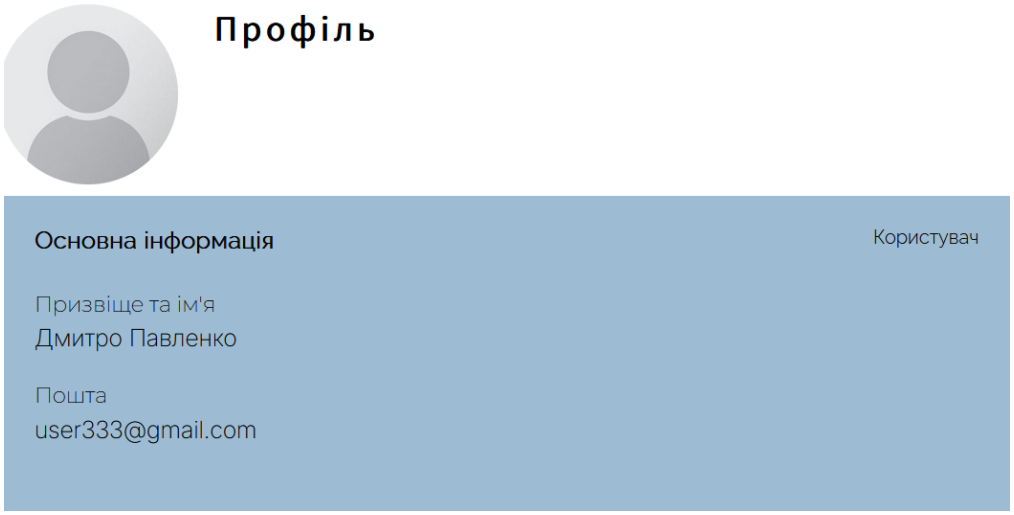

Рисунок 3.14 – Профіль користувача

Авторизований користувач може додавати оголошення у обране та переглядати їх на сторінці обраного. Тут можна швидко переглянути деталі вподобаних пропозицій, перейти на сторінку оголошення або видалити зайві позиції зі списку (див. рис. 3.15).

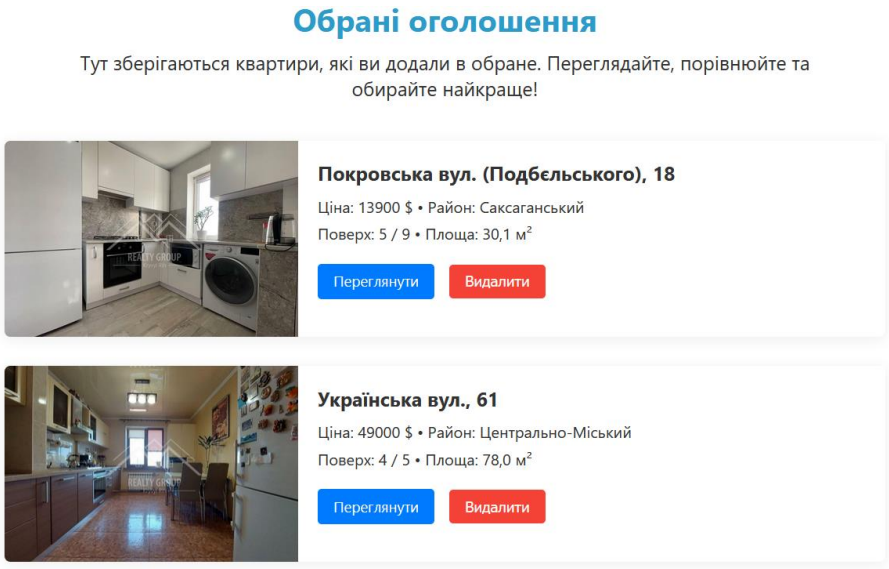

Рисунок 3.15 – Сторінка обраного

Розділ аналітики реалізовано з можливістю фільтрувати й сортувати дані за різними параметрами: район, вулиця та кількість кімнат. Користувач може оперативно порівнювати середню вартість квартир у різних частинах міста та визначати найбільш вигідні пропозиції ринку. На рисунку 3.8.8 зображено сторінку аналітики по квартирам.

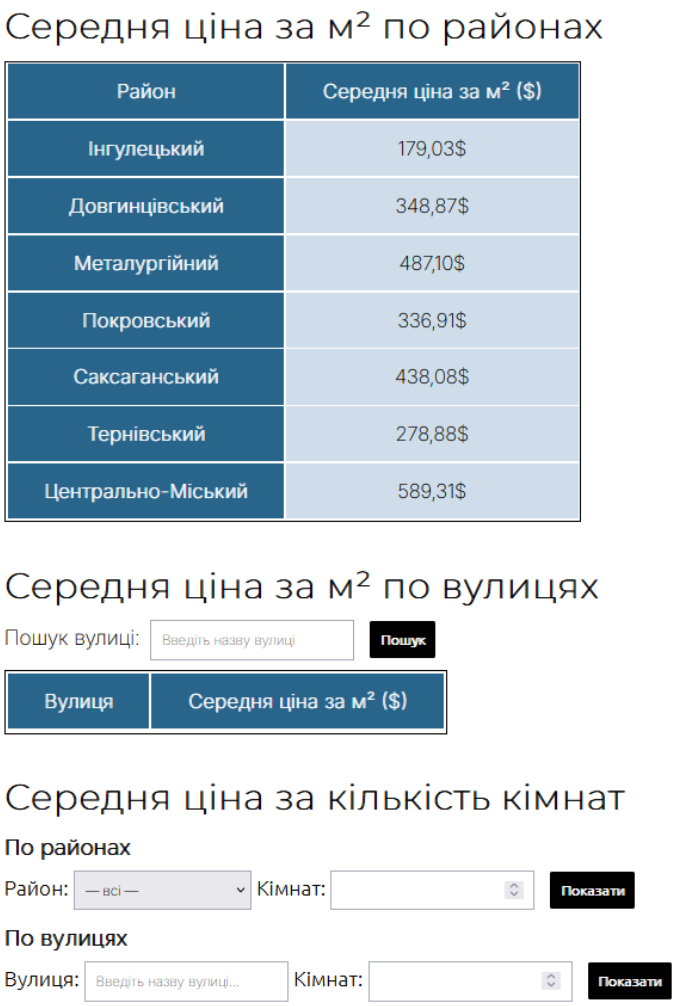

Рисунок 3.16 – Аналітична сторінка оголошень

Завдяки продуманому дизайну всі основні функції залишаються у прямому доступі — користувач не витрачає час на пошуки потрібної інформації. У результаті взаємодія з сайтом залишається комфортною та зрозумілою.

3.9 Особливості тестування

Тестування агрегатора нерухомості охопило низку практичних етапів, спрямованих на перевірку роботи основних модулів і гарантування стабільності у різних сценаріях використання. Було виконано ручне тестування всіх сторінок, що входять до функціоналу сайту. Перевірялися пошук, фільтрація, додавання й видалення оголошень із обраного, робота реєстрації та авторизації, а також коректна робота аналітичного блоку. Для кожної дії відстежувалась логіка взаємодії між клієнтською та серверною частинами, правильність обробки форм, швидкість відповіді інтерфейсу.

Паралельно із ручним тестуванням перевірялась стабільність роботи у різних браузерах: Google Chrome, Mozilla Firefox, Microsoft Edge. Особлива увага приділялася цілісності верстки, узгодженості роботи інтерактивних елементів, обробці помилок під час некоректного введення даних у форми чи відсутності обов’язкових параметрів. Усі основні функції, як-то перемикання сторінок, додавання обраного, відкриття модальних вікон, працювали однаково стабільно на кожній платформі.

На етапі тестування продуктивності застосовано інструмент Django Debug Toolbar. Основний акцент зроблено на моніторингу запитів до бази даних, у першу чергу — на перевірці дублювання SQL-запитів, що часто призводить до зайвого навантаження й сповільнення роботи сайту.

За результатами перевірки не виявлено жодного випадку дублювання чи нераціональних SQL-запитів: ORM Django забезпечив оптимальну взаємодію з базою, а структуру моделей побудовано коректно. Це дозволило досягти швидкої реакції сервера навіть під час вибірки великої кількості оголошень чи масової фільтрації за кількома критеріями.

У процесі тестування також з’ясувалася проблема із дублюванням оголошень при повторному запуску парсера даних. При невірно організованій перевірці у базі виникало кілька однакових записів, що могло ускладнити пошук чи статистику. Для вирішення додано перевірку на унікальність посилання на оголошення — тепер новий об’єкт додається лише за відсутності

такого запису у базі. Це дозволило значно підвищити якість та актуальність інформації у каталозі нерухомості.

Комплексний підхід до тестування допоміг переконатися у стабільній роботі сервісу. Усі основні сценарії використання були детально перевірені, а знайдені недоліки усунуто на ранньому етапі.

3.10 Аналіз результатів роботи системи

Результати роботи системи агрегатора нерухомості були детально проаналізовані з урахуванням поставлених завдань та вимог, що висувалися на етапі проектування. Було оцінено швидкість роботи функціональних модулів: система демонструє оперативну обробку запитів, коректне виконання пошуку та фільтрації, а також відображення великих вибірок оголошень без суттєвих затримок чи збоїв. Перевірка швидкості відгуку основних сторінок та інтерактивних елементів засвідчила достатній рівень оптимізації та коректність логіки, закладеної на рівні представлень і моделей. Додаткову увагу було приділено роботі аналітичного блоку — система впевнено формує статистичні показники, обчислює середню вартість житла за заданими критеріями й дозволяє користувачам отримувати узагальнені дані у зручному форматі.

Було проаналізовано, наскільки зручно користувачам взаємодіяти з різними розділами сайту: функція додавання у обране працює швидко, статуси елементів змінюються миттєво, а всі обрані об’єкти залишаються доступними навіть після виходу з профілю. Система дозволяє не лише зберігати цікаві пропозиції, а й повертатися до них у будь-який момент, що спрощує навігацію та порівняння оголошень. На рисунку 3.17 зображено кнопку до та після додавання оголошення в список обраних.

Рисунок 3.17 – Активна та неактивна кнопка додавання оголошення у обране

Оцінка роботи модулів збору та оновлення інформації засвідчила надійність реалізованих механізмів. Після запуску парсингу база поповнюється лише унікальними записами, дублювання або втрата даних не спостерігалися. Оновлення курсу валюти через API НБУ також виконується без затримок і помилок — відразу після отримання нового курсу ціни у відповідних оголошеннях ціна перераховується у гривню й відображається на сторінках. Дана логіка підтверджує ефективність поєднання моделей, представлень та клієнтських скриптів, які відповідають за цілісний цикл обробки та показу даних.

На завершення варто відзначити роботу адміністративної частини: панель адміністратора зручна для додавання, коригування або видалення оголошень, а також для моніторингу поточного стану каталогу та запуску службових процесів. Форма для створення нового оголошення дозволяє швидко вносити інформацію та одразу бачити результат на сайті. Повідомлення про успішність чи помилки виводяться через стандартний механізм, що гарантує зворотний зв’язок і прозорість у керуванні контентом.

У підсумку, результати аналізу роботи системи свідчать про досягнення заданих цілей, стабільність функціонування та зручність використання для обох категорій — і для адміністраторів, і для звичайних відвідувачів. Усі основні функції працюють узгоджено, а побудована структура дозволяє швидко масштабувати чи доповнювати проект за потреби.

3.11 Можливості розвитку та масштабування

Вибір Django як основного фреймворку дозволяє поступово розширювати функціонал, додаючи нові додатки й сервіси без необхідності докорінної перебудови системи. Модульна організація проекту дає змогу кожному компоненту виконувати власну роль, незалежно обслуговуючи пошук, фільтрацію, аналітику, обробку оголошень, роботу із профілями користувачів чи зовнішніми API. Завдяки такій побудові новий функціонал — наприклад, додавання каталогу іншої нерухомості, впровадження повідомлень або інтеграція з платіжними сервісами — може бути реалізований як окремий модуль, що мінімізує ризики для стабільності основного коду.

Розширення аналітичного розділу відкриває можливість для глибшої обробки даних, додавання нових метрик та зрізів, візуалізації на основі діаграм чи графіків. Також можлива інтеграція нових джерел даних для більш повного охоплення ринку нерухомості. Для цього достатньо реалізувати додатковий парсер у відповідному додатку та налаштувати алгоритми злиття даних, що дозволить швидко отримувати нові оголошення без ручного оновлення бази.

Система побудована так, щоб була можливість міграції на більш продуктивні інструменти із зростанням навантаження. Наприклад, для роботи з великими масивами оголошень база даних SQLite може бути замінена на PostgreSQL чи MySQL без необхідності змінювати бізнес-логіку у представленнях чи шаблонах. Django ORM забезпечує універсальний підхід до керування даними, тому зміна сховища відбувається переважно на рівні налаштувань проєкту.

Додатково проект готовий до впровадження нових інструментів адміністрування — наприклад, експорту аналітики у різні формати, гнучкого керування правами доступу, підключення модулів для взаємодії з картографічними сервісами або онлайн-платежами. Весь спектр змін може реалізовуватись поступово, з урахуванням потреб користувачів, ринку або нових технологічних можливостей.

Окрім технічного розширення, проект дозволяє адаптувати й саму бізнес-логіку під нові сценарії використання — наприклад, запровадження персональних рекомендацій для користувачів або створення окремих кабінетів для ріелторів із власними інструментами аналітики та управління пропозиціями.

Стратегія розвитку та масштабування проекту передбачає збереження єдиного стилю роботи інтерфейсу, незалежно від кількості розширень чи обсягу функціоналу. Гнучкість налаштувань і модульність дозволяють швидко впроваджувати інновації, підтримувати актуальність сервісу та адаптувати агрегатор до змін ринку нерухомості без ризику втратити контроль над системою.

ВИСНОВКИ

У ході виконання даної кваліфікаційної роботи було повністю розроблено та впроваджено функціональний веб-агрегатор оголошень з продажу квартир у Кривому Розі, орієнтований на потреби реальних користувачів місцевого ринку нерухомості. Аналіз ринку та існуючих рішень дозволив визначити найбільш ефективний шлях розробки: акцент зроблено на зручності пошуку, простоті інтерфейсу, актуальності даних та аналітичних можливостях.

Було доведено, що для сучасного користувача важливим є отримання структурованої, достовірної та швидко оновлюваної інформації з різних джерел. Врахування недоліків існуючих платформ — перевантаженість зайвими елементами, складність у навігації — стало основою для формування вимог до власного проекту. В результаті розроблено агрегатор, який фокусується лише на одному місті, дозволяючи надати більш точний і корисний сервіс мешканцям Кривого Рогу.

Архітектура проєкту побудована на фреймворку Django, що забезпечило гнучкий модульний поділ системи, ефективне керування базою даних, безпечну автентифікацію та масштабованість. Організація коду у вигляді окремих додатків для оголошень, користувачів, парсерів, валют та аналітики спростила подальший розвиток проєкту, дозволила легко додавати або змінювати функціональні блоки без впливу на інші частини системи.

Основні технічні завдання були вирішені комплексно. Створено ефективну підсистему збору й оновлення оголошень за допомогою автоматизованого парсингу та періодичної очистки бази від неактуальних записів. Запроваджено механізм динамічного перерахунку цін у гривнях, що особливо актуально для користувачів у періоди коливання валютного курсу. Реалізовано розширений функціонал для пошуку та фільтрації, який дозволяє легко знаходити необхідні квартири за різними параметрами.

Важливою особливістю стало впровадження блоку аналітики, який надає статистичну інформацію про ринок житла у Кривому Розі — середню вартість квадратного метра, порівняння цін за районами, вулицями чи кількістю кімнат. Такий підхід не лише економить час користувача, але й допомагає приймати зважені рішення на основі актуальних даних.

Функція роботи з обраними оголошеннями організована так, щоб забезпечити максимальний комфорт при збереженні та порівнянні цікавих пропозицій. Інтерактивні елементи інтерфейсу, використання AJAX-запитів та простий дизайн сприяли покращенню користувацького досвіду.

Вся система протестована вручну у популярних браузерах; виконано перевірку на дублювання SQL-запитів, що засвідчило коректність реалізації взаємодії з базою даних. Особливу увагу приділено захисту персональних даних, обмеженню доступу до особистих сторінок і розмежуванню ролей у системі. Додаткові перевірки та інформування про результат дій реалізовано через механізми Django messages.

Розроблений агрегатор залишається відкритим до масштабування. Передбачено можливість підключення нових джерел для збору даних, розширення аналітичних модулів, інтеграції сторонніх API або перенесення на потужнішу СУБД у разі збільшення навантаження. Така архітектура забезпечує стабільну роботу та дозволяє швидко адаптувати проєкт до нових викликів ринку.

Створена система повністю відповідає поставленим на початку роботи вимогам: вона об'єднує дані з різних джерел, дозволяє швидко та зручно шукати й аналізувати оголошення, гарантує безпечне збереження особистої інформації користувачів і надає розвинуті інструменти для адміністратора. Проект демонструє ефективність застосування сучасних технологій Python і Django для вирішення задач у сфері нерухомості та має всі передумови для подальшого розвитку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Flatfy [Електронний ресурс]. – Режим доступу : https://flatfy.ua/uk/.

2. Dom.ria [Електронний ресурс]. – Режим доступу : https://dom.ria.com/uk/.

3. Django Software Foundation. Django documentation [Електронний ресурс]. – Режим доступу : https://docs.djangoproject.com/en/5.2/.

4. BeautifulSoup documentation [Електронний ресурс]. – Режим доступу : https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

5. Selenium documentation [Електронний ресурс]. – Режим доступу : https://selenium-python.readthedocs.io//.

6. Національний банк України. API валютних курсів. [Електронний ресурс]. – Режим доступу : https://bank.gov.ua/ua/open-data/api-dev.

7. Rieltor.ua [Електронний ресурс]. – Режим доступу : https://rieltor.ua/.

ДОДАТОК А

Повний код функції run_parser_rieltor.

```
def run_parser_rieltor():
    """
    Парсинг та збереження квартир з rieltor.ua у базу даних
    """
    options = Options()
    options.add_argument("user-agent=Mozilla/5.0 (Windows NT
10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/120.0.0.0 Safari/537.36")
    options.add_argument("--disable-blink-
features=AutomationControlled")  # приховати автоматизацію

    driver = webdriver.Chrome(options=options)
    driver.get('https://rieltor.ua/krivoj-rog/flats-
sale/?page=1#9.26/47.9195/33.3864')
    html = driver.page_source
    soup = BeautifulSoup(html, 'html.parser')
    all_announcements_count = int(soup.find('span',
attrs={'data-listing-count': True}).text.strip().split()[0])
# визначаємо кількість оголошень
    all_pages_count = (all_announcements_count // 20) + 1 if
all_announcements_count % 20 != 0 else
all_announcements_count // 20 # визначаємо кількість
сторінок

    added_count = 0

    for page in range(1, all_pages_count+1):
        print(f"Парсинг сторінки {page}...")

        driver.get(f'https://rieltor.ua/krivoj-rog/flats-
sale/?page={page}#9.26/47.9195/33.3864')
        html = driver.page_source
        soup = BeautifulSoup(html, 'html.parser')
        current_page = soup.find_all(class_='catalog-card-
media')
        if not current_page:
            print("Більше немає сторінок з оголошеннями")
            break

        for announcement in range(len(current_page)):
            href = current_page[announcement].get('href')
            driver.get(href)
            time.sleep(1.5)
            html2 = driver.page_source
```

```
            main_soup = BeautifulSoup(html2, 'html.parser')
            apartments_info = {}

            try:
                count_class_name =
len(main_soup.find_all(class_='address-link'))
                if count_class_name == 6:
                    apartments_info['announcement_title'] =
main_soup.find_all(class_='address-link')[0].text +
main_soup.find_all(class_='address-link')[0].next_sibling
                    apartments_info['street'] =
main_soup.find_all(class_='address-link')[0].text
                else:
                    title =
main_soup.find_all(class_='address-
link')[2].next_sibling.split(',')
                    apartments_info['announcement_title'] =
title[1] + ', ' + title[2]
                    street =
main_soup.find_all(class_='address-
link')[2].next_sibling.split(',')[1]
                    apartments_info['street'] = street

                apartments_info['district'] =
main_soup.find_all(class_='address-link')[-1].text.strip('
р-н')
                apartments_info['price'] =
int(main_soup.find(class_='offer-view-price-
title').text.replace('$', '').replace(' ', ''))
                apartments_info['rooms'] =
main_soup.find(class_='offer-view-details-
column').find('a').text.split(' ')[0]
                apartments_info['meters'] =
float(main_soup.find(class_='offer-view-details-
column').find_all(class_='offer-view-details-
row')[1].text.strip().split(' / ')[0])
                apartments_info['content'] =
main_soup.find_all(class_='offer-view-section-text')[0].text
+ main_soup.find_all(class_='offer-view-section-
text')[1].text
                apartments_info['images'] = [img.get('src')
for img in main_soup.find_all(class_='offer-photo-
gallery__image')]
                floor = main_soup.find(class_='offer-view-
details-column').find_all(class_='offer-view-details-
row')[2].text.strip()
                apartments_info['floor'] =
f'{floor.split()[1]} / {floor.split()[3]}'
                apartments_info['seller_name'] =
main_soup.find(class_='offer-view-rieltor-
name').text.strip()
                apartments_info['phone'] =
```

```
main_soup.find(class_='offer-view-rieltor-
phones').text.strip()
                apartments_info['link'] = href
            except Exception as e:
                print(f"Помилка збору даних оголошення
({href}): {e}")
                continue

            added = save_announcement(apartments_info)
            if not added:
                print(f"Оголошення {apartments_info['link']}
вже є у базі — пропускаємо.")
            else:
                print("Нове оголошення:",
apartments_info['link'])
                added_count += 1

    driver.quit()
    print(f"Додано нових оголошень: {added_count}")
    return added_count
```

ДОДАТОК Б

Повний код файлу filter.js.

```
function filterByPrice(price) {
    price = parseFloat(price);

    let range;
    if (price <= 10000) {
        range = 'up_to_10000';
    } else if (price <= 40000) {
        range = 'from10000_to40000';
    } else if (price <= 100000) {
        range = 'from40000_to100000';
    } else {
        range = 'over_100000';
    }

    window.location.href = `?price_range=${range}`;
}

function filterByMeters(meters) {
    meters = parseFloat(meters);

    let range;
    if (meters <= 25) {
        range = 'up_to25met';
    } else if (meters <= 40) {
        range = 'from25met_to40met';
    } else if (meters <= 75) {
        range = 'from40met_to75met';
    } else {
        range = 'over_75met';
    }

    window.location.href = `?meters_range=${range}`;
}

document.addEventListener('DOMContentLoaded', function () {
    const form = document.querySelector('.search-form');
    if (!form) return;

    form.addEventListener('submit', function (event) {
        const priceInput =
form.querySelector('input[name="price"]');
        const priceValue = parseFloat(priceInput.value);

        if (!priceInput.value || isNaN(priceValue)) {
```

```
            return;
        }

        let range = '';
        if (priceValue <= 10000) {
            range = 'up_to_10000';
        } else if (priceValue <= 40000) {
            range = 'from10000_to40000';
        } else if (priceValue <= 100000) {
            range = 'from40000_to100000';
        } else {
            range = 'over_100000';
        }

        priceInput.remove();

        const hiddenInput = document.createElement('input');
        hiddenInput.type = 'hidden';
        hiddenInput.name = 'price_range';
        hiddenInput.value = range;
        form.appendChild(hiddenInput);
    });
});
```

ДОДАТОК В

Повний код рендерингу оголошень.

```
<div class="announcement_menu">

     {% for row in homeline_rows %}

          <div class="announcements">

               {% for an in row %}

                    <article>

                         <div class="apartment_an">

                              <div class="apartment_img">

                                   <a href="{% url
'announcement' an.id %}" class="an-wrap">

                                        <img src="{{
an.images.0 }}" alt="Оголошення" height="211"
loading="lazy">

                                   </a>

                              </div>

                              <div class="apartment_text">

                                   <div class="tags">

                                        <ul
class="tags_list">

                                             <li>


          <button class="price" onclick="filterByPrice({{
an.price }})" aria-label="Ціна у валюті">


               {{ an.price }}$


          </button>
```

```
    </li>
                                                <li>
                                                    <a
href="?rooms={{ an.rooms }}">

    <button class="features" aria-label="Кількість
кімнат">{{ an.rooms }} кімн.</button>

                                                    </a>
                                                </li>
                                                <li>
                                                    <button
class="features" onclick="filterByMeters({{ an.meters }})"
aria-label="Квадратні метри">{{ an.meters }}
м<sup><small>2</small></sup></button>
                                                </li>
                                            </ul>
                                        </div>
                                        <div class="info-info">
                                            <a href="#"
class="an-wrap">
                                                <h3
class="street">{{ an.street }}</h3>
                                            </a>
                                            <div class="place-
filter">
                                                <button
class="district" onclick="alert('Кнопка нажата!')">{{
an.district |upper }}</button>
                                            </div>
                                        </div>
                                    </div>
```

```
                            </div>

                        </article>

                    {% endfor %}

                </div>

            {% endfor %}
</div>
```