

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістра

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
ПРОГНОЗУВАННЯ ПОПУЛЯРНОСТІ ЖИТЛА ДЛЯ ОРЕНДИ
RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS FOR
PREDICTING POPULARITY OF APARTMENTS FOR RENT

Виконав: студент(ка) 2 курсу, групи КНТ-211м

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

Кодак І.В.

(прізвище та ініціали)

Керівник Льовкін В.М.

(прізвище та ініціали)

Рецензент Коцур М.І.

(прізвище та ініціали)

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

Інститут, факультет ФКНТ
Кафедра програмних засобів
Ступінь вищої освіти магістр
Спеціальність 122 Комп'ютерні науки
(код і найменування)

Освітня програма (спеціалізація) Системи штучного інтелекту
(назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

Завідувач кафедри ПЗ, д.т.н, проф.
С.О. Субботін
“ ” 2022 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Кодака Івана Васильовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів прогнозування популярності житла для оренди
Research and Software Implementation of Methods for Predicting Popularity of Apartments for Rent

керівник проєкту (роботи) Льовкін Валерій Миколайович, к.т.н., доцент,
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від “4” листопада 2022 року № 363

2. Строк подання студентом проєкту (роботи) грудень 2022 року

3. Вихідні дані до проєкту (роботи) рекомендована література, технічне завдання

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження. 2. Матеріали і методи. 3. Проєктування програмного забезпечення. 4. Основні рішення щодо реалізації компонентів системи. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень) Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-5 Основна частина	Льовкін В.М., доцент		
Нормоконтролер	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання “1” вересня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз та дослідження предметної області	2-5 тижні	Розділи 1, 2
3	Розробка архітектури програми	6 тиждень	Розділ 3
4	Розробка програми	7-9 тижні	Розділ 4
5	Тестування та експериментальне дослідження програми	10-11 тижні	Розділ 5
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	12 тиждень	Додатки
7	Захист роботи	13 тиждень	

Студент(ка)

(підпис) Кодак І.В.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Льовкін В.М.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
98 с., 26 рис., 3 табл., 3 дод., 18 джерел.

ОРЕНДА ЖИТЛА, ОЦІНЮВАННЯ ЖИТЛА, ПОШУК ЖИТЛА,
ЗАДАЧА ПРОГНОЗУВАННЯ, ВЕБДОДАТОК.

Об'єктом виконаного дослідження є процес орендування житла від власників, а предметом – методи прогнозування популярності житла для оренди.

Мета дослідження полягала в розробці програмного забезпечення орендування житла від власників з наданням можливості прогнозування його популярності для підтримки наступних керуючих рішень власника.

Матеріали, методи та технічні засоби: мова Python, бібліотека машинного навчання scikit-learn, система MySQL, фреймворк Django.

Результати. Основними отриманими в роботі результатами є запропонований метод прогнозування популярності житла для оренди, його програмна реалізація і загалом програмне забезпечення, яке підтримує роботу над орендою житла від власників як шляхом різнобічного керування замовленнями з проживання, так і шляхом підтримки зручного отримання інформації та надання інформації про житло орендарям. Реалізовано підтримку роботи помічників, що дозволило наблизити процес підготовки житла і заселення до реально виконуваного процесу в цій предметній області.

Висновки. Запропонований метод прогнозування популярності житла підтвердив вищу точність прогнозування кількості оцінювань житла порівняно з іншими проаналізованими методами під час експериментального дослідження на реальній вибірці даних.

Галузь використання – короткострокова оренда житла, що надається безпосередньо власниками.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 98 pages, 26 figures, 3 tables, 3 appendixes, 18 sources.

APARTMENT RENT, APARTMENT EVALUATION, APARTMENT SEARCH, FORECASTING PROBLEM, WEB APP.

The object of the performed research is the process of renting apartments from owners, and the subject is methods of forecasting the popularity of apartments for rent.

The purpose of the study was to develop software for renting apartments from owners with the ability to predict its popularity to support subsequent management decisions of an owner.

Materials, methods and technical means: Python language, scikit-learn machine learning library, MySQL system, Django framework.

The results. The main results obtained in the work are the proposed method for predicting the popularity of apartments for rent, its software implementation and, in general, software supporting work on renting apartments from owners both by versatile management of apartment orders and by supporting convenient obtaining and providing information about apartments. Work of assistants is supported. It made it possible to bring the process of apartment preparation and occupancy closer to the actual process in this subject area.

Conclusions. The proposed method for predicting the popularity of apartments confirmed a higher accuracy of prediction of the number of apartment evaluations compared to the other analyzed methods during an experimental study on a real data sample.

The field of use is short-term apartment rental provided directly by the owners.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань дослідження	11
1.1 Особливості орендування житла в сучасному світі	11
1.2 Аналіз можливостей програмного забезпечення орендування житла ..	12
1.3 Висновки за розділом 1	16
2 Матеріали і методи.....	18
2.1 Задача прогнозування популярності житла для оренди	18
2.2 Алгоритм Random Forest	19
2.3 Метод прогнозування популярності житла для оренди.....	21
2.4 Висновки за розділом 2	23
3 Проектування програмного забезпечення	24
3.1 Обґрунтування вибору засобів розробки.....	24
3.2 Визначення функціональних вимог до програми.....	24
3.4 Архітектура програмної системи	28
3.4 Висновки за розділом 3	29
4 Основні рішення щодо реалізації компонентів системи.....	31
4.1 Структура роботи з даними.....	31
4.2 Класи вебдодатку	42
4.3 Висновки за розділом 4	50
5 Експлуатація, тестування та експериментальне дослідження програми	51
5.1 Призначення програми	51
5.2 Умови виконання програми	51
5.3 Виконання програми.....	51
5.4 Експериментальне дослідження	55
5.5 Висновки за розділом 5	58
Висновки	59
Перелік джерел посилання	60

Додаток А Технічне завдання	62
Додаток Б Текст програми	67
Додаток В Слайди презентації.....	90

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

RF – Random Forest;

БД – база даних.

ВСТУП

Оренда житла є дуже конкурентною сферою. Проте в обсягах пропозицій і потреб дана сфера не є статичною: це демонструється, наприклад, протягом поточного року в Україні, коли велика кількість людей вимушено переселилась і багато з цих людей виявились на ринку житла з необхідністю його орендувати, або коли місто чи країна розвивають туристичну діяльність, привертають нових туристів, з'являються нові пропозиції для оренди, включаючи нові готелі, новий збудований житловий фонд, частина якого виставляється як пропозиція для оренди. У такому випадку виявляється потреба з'єднати між собою власників житла з орендарями. Коли вони знаходяться в різних містах, то програмне забезпечення стає основним варіантом їхньої взаємодії. Саме тому використання інформаційних технологій в цій сфері дуже активне і важливе.

Однак, виставлення власного житла в оренду через програмну систему або пошук такого житла орендарем є важливими завданням, але виникає необхідність і у додаткових інструментах. Виставляючи своє житло в оренду, власники прагнуть привернути до нього увагу. В програмних системах, через які відбувається бронювання проживання, основним виразником популярності житла є кількість оцінювань або відгуків, які були залишені орендарями про житло. З іншого боку саме це диктує впевненість наступних орендарів у своєму виборі. Тому кожен власник прагне розширювати кількість відгуків про своє житло, отримувати нові оцінювання і відгуки, щоб продемонструвати таким чином поточний стан житла. Саме тому створення інструменту прогнозування популярності житла для оренди є важливим не тільки саме по собі, але і шляхом впровадження його в програмну систему, яка використовується для бронювання житла, щоб власник міг, не переходячи в інші програмні засоби, одразу отримати прогноз і зрозуміти, як буде розвиватися в формальному представленні його житло на даному вебсайті найближчим часом.

У підсумку розробка програмного забезпечення орендування житла від власників з наданням можливості прогнозування його популярності для підтримки наступних керуючих рішень власника є метою цієї роботи.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Особливості орендування житла в сучасному світі

2022 рік відзначається тим, що орендна плата за квартири зокрема в США стрімко зростає майже в 400 містах, а середня ціна за 1-кімнатну квартиру з червня 2021 року зросла більш ніж на 25 %, а типова вартість квартири з двома спальнями за цей період зросла на 26,5%, причинами цього процесу стали гостра нестача доступного житла, зростання цін на житло та стрімка інфляція [1]. Через це відбувається міграція з дорогих центрів міст – таких як Сан-Франциско, Нью-Йорк і округ Колумбія – до більш доступних передмість, додатково це обумовлено тим, що багато офісних працівників у високооплачуваних галузях, таких як техніка та фінанси, протягом останнього часу перейшли на більш гнучку віддалену роботу [2]. Це також створює виклик для монетарної політики: передбачається, що Федеральна резервна система США реагує на дані в режимі реального часу, щоб визначити, чи варто продовжувати підвищувати процентні ставки, щоб стримати попит, але значною частиною зростання базової інфляції є інфляція орендної плати, яка, ймовірно, пройшла свій пік, при цьому чим більше Федеральна резервна система підвищує ставки, тим більше вона перешкоджає житловому будівництву, що не тільки зменшує загальне зростання, але й позбавляє ринок нових будинків, у довгостроковій перспективі згорнуте будівництво означає менше будинків, що означає вищу орендну плату [3]. Аналогічна ситуація спостерігається і в Європейському Союзі [4], де вона ще й посилена переселенням українців через війну.

Тож загалом поточну ситуацію на ринку оренди житла можна характеризувати як достатньо розігріту. Вартість житла збільшується, це призводить до конкуренції між пропозиціями, щонайменше тимчасової, поки клієнти намагаються адаптуватися до ситуації і знайти нові рішення, і в такому випадку популярність житла є важливим результуючим показником на будь-якій платформі для власника цього житла. Адже в підсумку

популярність він може конвертувати в збільшення власного прибутку, у відповідне позиціонування житла, відсутність простоїв.

З іншого боку зараз є перші ознаки стабілізації ситуації на ринку. А за стабілізацією поступово слідує падіння зазвичай. Падіння вартості призводить до того, що більша кількість людей готова придбати житло, а не орендувати. Відповідно конкуренція серед пропозицій в такий момент ще збільшується.

Необхідність в інструментах, які б дозволили власнику розуміти, які можливості зі зміни вартості для його житла існують, підвищується в ситуації високої конкуренції. Але варто вказати і на те, що такі інструменти важливі і не тільки в критичній ситуації: вони загалом важливі для розуміння того, як позиціонування житла на ринку короткострокової оренди житла розвивається з часом. Це в свою чергу призводить до того, що власник розуміє не тільки загальні тренди зміни вартості житла на ринку, але і тренди розвитку власного оголошення. Адже спочатку, коли оголошення є непопулярним, власнику доводиться зменшувати вартість, а потім, зі зростанням популярності, він може встановлювати вищі значення, при цьому очікуючи потрібної йому заповненості житла, яке він здає в оренду.

1.2 Аналіз можливостей програмного забезпечення орендування житла

Достатньо велика кількість програмних платформ, які представлені щонайменше вебсайтами, використовується в світі для оренди житла. Тому за результатами аналізу в цьому підрозділі представлено не всі проаналізовані рішення, а тільки деякі виділені з них та загальні висновки.

Одним з найвідоміших вебсайтів для оренди житла в даний момент є Airbnb (рис. 1.1) [5]. Історично ця платформа багато в чому популяризувала індивідуальний туризм за рахунок можливості проживання в орендованих квартирах або навіть кімнатах в квартирах власників, що є початковою ідеологічною відмінністю її від booking.com [6], яка є ще однією з

найвідоміших платформ оренди житла. Однак, відрізняється більшою популярністю готелів на платформі, хоча на обох платформах загалом можна зараз знайти і готельні номери, і апартаменти.

Фотографії

Зручності

Відгуки

Розташування

\$72 ніч

★ 5,0 · 3 відгуки

Забронювати

★ 5,0 · 3 відгуки

Чистота	_____ 5,0	Точність	_____ 5,0
Спілкування	_____ 5,0	Розташування	_____ 5,0
Прибуття	_____ 5,0	Ціна/якість	_____ 5,0



Jasem

жовтень 2022

Дивовижне місце, приїду знову.

господар теж дивовижний, дуже швидко відповідає та...

[Показати більше >](#)



Aker

жовтень 2022

Місце чудове, з хорошим відпочинком. У будинку було все необхідне. Чудовий вид, затишний інтер'єр у будинку. Усе продумано до дрібниць, там було дуже м'яко. Wi-Fi...

[Показати більше >](#)



Kateryna

вересень 2022

це найкраще місце у світі! Наше свято було дивовижним! Гела дуже гостинна та корисна господиня! допомогла нам з усіма нашими запитаннями та бажаннями. Будинок...

[Показати більше >](#)

Рисунок 1.1 – Вебсайт Airbnb

Airbnb, як будь-яке подібне рішення, дозволяє виконувати пошук житла, використовуючи достатньо велику кількість його характеристик, замовляти житло для проживання, спілкуватися з власниками, отримувати інформацію про власні бронювання. Однак всі ці функції надаються в стандартному інтерфейсі для орендаря. Для власника ж функції дещо відрізняються.

Власник житла може отримувати інформацію про стан бронювання (прибуття сьогодні, прибуття завтра, прибуття через зазначену кількість днів, гість повинен надати документ підтвердження особистості, очікування відгуку гостя, очікування сплати, скасовано, виїзд сьогодні, зміни на розгляді, гість заселився, пропозиція прострочена, пропозиція відправлена, неможливо, запит скасований, запит прострочений, залишити відгук про гостя, спеціальна пропозиція відправлена, поїздка змінена Airbnb, зміна

поїздки відключена, зміна поїздки запитана, поїздка скасована, заявка), підтверджувати або скасовувати запити на бронювання, створювати власні оголошення, визначати розцінки на оренду, визначати вільні дати та контролювати велику кількість параметрів оголошення і відповідного бронювання (мінімальну, максимальну кількість ночей оренди).

Для Airbnb важливим показником популярності з одного боку та довіри, з іншого боку, є відгуки про житло. Це дозволяє зрозуміти, наскільки адекватним є господар, наскільки вірною є подана в описах та на фотографіях інформація. Відповідно більша кількість відгуків дозволяє зрозуміти, як люди, які вже завершили проживання в цьому житлі, оцінюють його, а окрім того розуміти, як це змінюється з часом. Однак для власника житла не надається ніяких особливих інструментів стосовно цього, вже не говорячи про механізми прогнозування.

Серед особливих інших платформ виділяється Plum Guide (рис. 1.2) [7], яка особлива тим, що має набагато менше пропозицій, проте всі вони контролюються командою платформи, яка чітко оцінює кожне житло за 150 критеріями, в підсумку залишаючи тільки 5 % серед всіх варіантів житла, які оформлюють заявки [8]. З одного боку це зменшує кількість пропозицій, але з іншого кожна пропозиція є якісною.

Ще одним особливим додатком є Sonder, який виставляє для оренди тільки ті пропозиції, якими вони самі володіють [9]. Це достатньо велика платформа, яка охоплює цілих 27 міст по всьому світу. Однак варто вказати, що останні дві платформи, на відміну від Airbnb та Booking.com можуть мати в підсумку абсолютно інший підхід до питань якості і популярності. Адже тільки Airbnb та Booking.com дозволяють створити повноцінні ринкові відносини між власниками і орендарями, не формуючи повністю ринок. Тобто наприклад, Airbnb підтримує орендарів, дозволяє знайти інший варіант проживання, якщо раптово власник не заселив орендаря, проте початкові пропозиції виходять від власників, а в подальшому через інформацію про такі проблеми у власників, через інформацію з оцінювань можна зрозуміти,

що використовувати послуги відповідного власника житла є небезпечним або неякісним. Інші ж платформи впливають на початковий набір пропозицій, чітко його формуючи або обмежуючи, тому в них питання популярності розглядається з іншого ракурсу. Однак у всіх цих платформах відсутні функціональні можливості, які б дозволили власнику прогнозувати майбутній стан, в якому буде перебувати його житло.

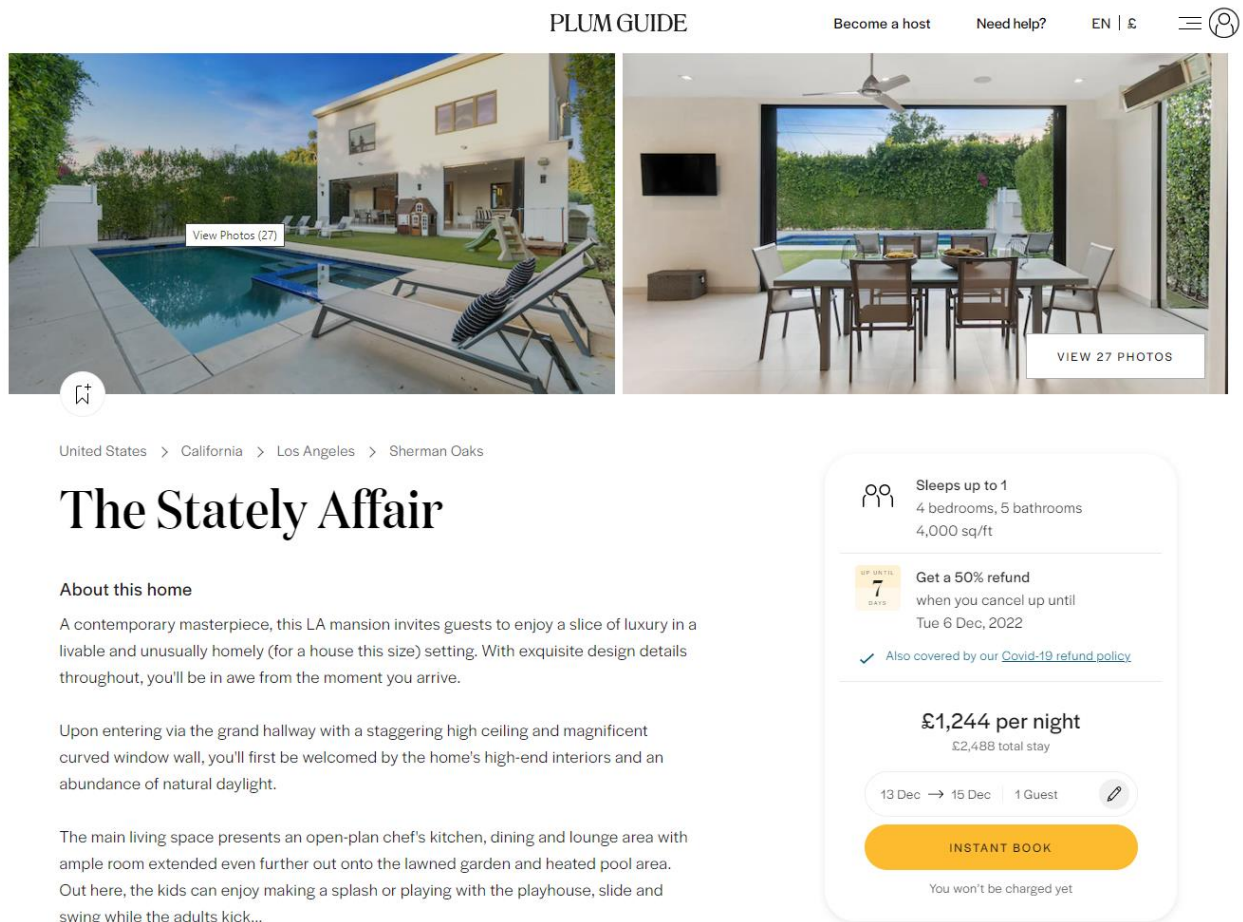


Рисунок 1.2 – Вебсайт Plum Guide

У підсумку було виконано порівняння вивчених аналогів з програмою, яка буде розроблятися в подальшому в роботі, а результати представлено в табл. 1.1.

Таблиця 1.1 – Порівняння програмних аналогів

Характеристика	Airbnb	Plum Guide	Дипломна розробка
Принцип формування пропозицій	Вільно власниками	Жорсткий відбір командою	Вільно власниками
Тип житла	Повне житло, частина	Повне житло	Повне житло, частина
Пошук житла	Так	Так	Так
Керування проживаннями	Так	Так	Так
Оцінювання орендарів	Так	Ні	Так
Оцінювання житла орендарями	Так	Ні	Так
Прогнозування популярності житла	Ні	Ні	Так

1.3 Висновки за розділом 1

Мета дослідження, розробка програмного забезпечення орендування житла від власників з наданням можливості прогнозування його популярності для підтримки наступних керуючих рішень власника, повинна виконуватися на основі завдань:

- проаналізувати програмні засоби орендування житла від власників, які на даний момент вже існують;
- виконати дослідження методів прогнозування популярності житла для оренди;
- виконати проектування програмного забезпечення орендування житла від власників;

– реалізувати програмне забезпечення орендування житла від власників.

Перший розділ є результатом аналізу проблеми орендування житла від власників, охоплюючи сучасний стан проблеми зі зростанням вартості оренди, сучасний стан програмних додатків, які підтримують даний процес, і їх особливостей, порівняння цих додатків.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Задача прогнозування популярності житла для оренди

Для формулювання задачі прогнозування популярності житла для оренди спершу потрібно визначити, яким чином оцінювати популярність житла. При дослідженні даного питання було враховано логіку, яку було визначено під час аналізу в розділі 1. Звичайно, що оцінити популярність можна за числом запитів на бронювання або за числом виконаних бронювань. Фактично, наприклад, останній показник можна визначити як частку часу, протягом якого житло було заброньовано, порівняно з усім доступним часом за цей період. Однак цей показник може бути корисним для платформ, на яких обмежено кількість пропозицій.

Для вільної платформи, яка розглядається в цій роботі, прикладом якої є Airbnb, цей показник не є адекватним вираженням популярності, адже звичайний орендар такої інформації не має про житло: в календарі він бачить недоступні дати на рівні з заброньованими датами. З іншого боку на вільній платформі джерелом уваги і сприйняття якості житла є оцінювання (відгуки). Відповідно з цього виходить логіка, що кількість оцінювань фактично корелює з популярністю житла в розумінні потенційного орендаря. Тому ґрунтуючись на цій логіці, адекватним є використання в якості показника популярності житла для оренди саме кількості оцінювань житла його орендарями, які вже завершили оренду. У підсумку це можуть бути як позитивні, так і негативні оцінювання, але всі вони разом говорять про популярність житла, а вже в розрізі позитивні або негативні визначають якість проживання.

Також було враховано логіку, яку представлено зокрема в роботі [10] і яка застосована в сервісі Tripadvisor [11], де заклади оцінюються за кількістю відгуків. Важливість даної ознаки проявляється і в динаміці, тобто важливим є не тільки накопичити певну кількість оцінювань і після чого житло стабільно вважатиметься популярним, важливо, щоб ця кількість

оновлювалась в подальшому. Якщо кількість оцінювань не буде змінюватись протягом певного періоду, наприклад, року, то це означає, що фактично популярність житла значно впала, житло стало непопулярним. Якщо житло отримує нові оцінки, то орендар розуміє його поточний стан, якщо ж нових оцінювань немає, то орендар розуміє, що дані застарілі. Тому свідченням популярності є кількість оцінювань на поточний момент, а не загалом на вебсайті.

Тож задача прогнозування популярності житла для оренди є задачею прогнозування кількості оцінювань на наступний період для даного житла за набором значень вхідних ознак. В подальшому в якості базового періоду використовується місяць, тобто прогнозування виконується на місяць, а кількість оцінювань обчислюється на кожен місяць.

2.2 Алгоритм Random Forest

Алгоритм Random Forest (RF) – це алгоритм машинного навчання, який вдало на практиці на даний момент використовується для розв’язання задач прогнозування, якою є задача, що розглядається в роботі, та задача класифікації.

Алгоритм RF додає додаткову випадковість моделі, вирощуючи дерева: замість пошуку найважливішої ознаки під час поділу вузла, алгоритм шукає найкращу ознаку серед випадкової підмножини ознак, це призводить до широкого розмаїття, що загалом призводить до кращої моделі [12]. Тобто фактично дерева спеціалізуються на різних ознаках, формуючи відповідну підмножину ознак, а загальне рішення формується на основі усереднення рішень всіх дерев, кожне з яких спеціалізується на підмножині ознак.

Перевагами алгоритму RF є:

- можливість роботи і з категоріальними, і в той же час з безперервними значеннями достатньо якісно для практичного застосування;

- можливість обробляти спостереження, в яких відсутні значення якихось ознак або в яких присутні викиди значень, автоматично;
- відзначається стабільністю: якщо в набір даних додається нове спостереження (нові дані), то на загальний алгоритм це не сильно вплине, оскільки дані, які було додано, вплинуть на одне дерево, але фактично на всі дерева разом не вплинуть (щонайменше це малоймовірно);
- порівняно несуттєво піддається впливу шуму [13].

Ці переваги є важливими для задачі прогнозування популярності житла для оренди, оскільки вибірка, з якою потрібно працювати, з одного боку, постійно розширюватиметься, а з іншого боку, в ній можуть зустрічатися і пропущені значення часом, і шум. Тому адекватним для розв'язання цієї задачі є використання алгоритму RF.

Основу роботи алгоритму RF складають два етапи (перший полягає в об'єднанні N дерев рішень із створенням випадкового лісу, а другий – у створенні прогнозів для кожного дерева, створеного на першому етапі), які можна розподілити на наступну послідовність кроків:

- вибрати M точок даних випадковим чином із навчальної вибірки даних;
- створити дерева рішень для вибраних точок даних (підмножин);
- визначити вихід для кожного дерева рішень та проаналізувати отримані результати;
- для розв'язання задачі прогнозування виконати усереднення отриманих результатів і визначити загальний вихід [14].

Основні відмінності алгоритму RF від звичайних дерев рішень наступні:

- перенавчання: традиційне для дерев рішень перенавчання не відбувається, оскільки випадкові ліси формуються з підмножин даних, а кінцевий результат базується на середньому значенні, це не дозволяє реалізуватись перенавчанню в підсумку;

- швидкість: враховуючи наявність цілого набору дерев, звичайно, що алгоритм RF відносно повільніший, ніж дерево рішень;
- процес: випадковий ліс збирає дані випадковим чином, формує дерево рішень і усереднює результати, він не покладається на жодні формули, як у деревах рішень [14]-[15].

2.3 Метод прогнозування популярності житла для оренди

Розглядаючи алгоритм RF, було виявлено, що він має проблеми для роботи з часовими рядами в певних ситуаціях [16]. Це ситуації, коли часовий ряд представляє зростаючий тренд. У такому випадку зустрічаються результати, які фактично не входять в інтервали, які були використані на навчальних даних. Тоді ліс приймає рішення узгоджено і в підсумку отримане значення не відповідає реальним результатам, адже вони виходять з відомого інтервалу. Цей підхід виключив використання, наприклад, кумулятивної оцінки, адже тоді б значення весь час збільшувалось і це не дозволило б використати алгоритм RF для цієї ситуації. Тому прогнозування відбувалось саме для конкретної кількості оцінювань на виході. Відповідно значення фактично в тих же самих інтервалах зустрічались і в навчальній вибірці, і в тестовій, а потім і в реальних ситуаціях, адже вибірка даних може бути використана достатньо велика в підсумку, зважаючи на наявність зокрема вибірок [17].

Зважаючи на описану постановку задачі прогнозування популярності житла для оренди, яка є задачею прогнозування, та логіку алгоритму Random Forest, було сформульовано відповідний метод для розв'язання цієї задачі.

На першому етапі роботи методу створюється вибірка даних, яка в подальшому використовується для навчання. У цій вибірці даних кожне спостереження повинно включати набір з наступних ознак:

- середня вартість доби проживання за відповідний місяць;

- тип житла (використовується розподіл на 3 типи: повне житло, приватна кімната, спільна кімната);
- район проживання;
- кількість оцінювань за кожен з попередніх 6 місяців (6 окремих значень);
- середня вартість доби проживання за всі попередні 6 місяців;
- кількість оцінювань за відповідний місяць (використовується як вихідна ознака, тобто саме це значення прогнозується).

На другому етапі відбувається створення і навчання моделі на основі алгоритму RF та сформованої вибірки даних.

На третьому етапі реалізується збір даних про житло, для якого потрібно виконати прогнозування. Фактично при цьому витягається перелік всього активного житла з програми. Активним вважається житло, для якого на наступний місяць встановлено хоча би один день, на яке може відбуватися або вже відбулося бронювання проживання. Ще однією умовою є те, що для житла має бути сформовано вже історію з 6 попередніми місяцями позиціонування на платформі. Для досягнення точних результатів прогнозування має виконуватися вперше тільки після 6 місяців після першого виставлення житла в оренду в програмі.

Для кожного житла визначається середня вартість доби проживання на майбутній місяць. Це і є місяць, для якого відбудеться прогнозування кількості оцінювань.

Після цього витягається інформація про кількість оцінювань за минулий, позаминулий і так далі за сумарно 6 попередніх місяців. Для періоду в ці 6 місяців обчислюється середня вартість доби проживання.

Встановлюються як постійні значення значення району, типу житла. Якщо для житла має бути змінено ці значення, то вважається, що це має бути нове оголошення, тобто відповідно воно розглядається як інше житло.

На четвертому етапі виконується прогнозування для заданих значень, що характеризують житло на попередньому етапі, кількості оцінювань на

наступний місяць для кожного варіанта житла. Це і є результатом роботи методу.

2.4 Висновки за розділом 2

Розділ 2 присвячено виконаному дослідженню проблеми прогнозування популярності житла для оренди. Для цього було визначено спосіб обчислення показника популярності. В основі цього способу лежить кількість оцінювань житла орендарями. Проаналізовано алгоритм RF.

Науковою новизною роботи є метод прогнозування популярності житла для оренди, який дозволяє прогнозувати кількість оцінювань житла за наступний місяць на основі характеристик житла і даних минулих місяців, використовуючи алгоритм RF, що дозволяє завчасно отримувати інформацію, яку можна використати для прийняття рішень про зміну параметрів оголошення про оренду житла.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Обґрунтування вибору засобів розробки

Першочергова множина мов програмування для вибору була сформована з мов, які дозволяють створювати вебдодатки, є актуальними мовами програмування, достатньо легко інтегруються для роботи зі сховищами даних. У подальшому серед них вибір був зведений до Ruby та Python (табл. 3.1), серед яких з певною перевагою була обрана мова Python.

Таблиця 3.1 – Порівняння мов для реалізації програмного забезпечення орендування житла від власників

Критерій	Мова Ruby	Мова Python
Складність навчання	Вища разом з фреймворком	Нижча
Простота відлагодження	Нижча	Висока (сприяє читабельності)
Зручність повторного використання коду на основі бібліотек	Має меншу кількість бібліотек	Відомий великою кількістю бібліотек
Основний фреймворк для створення вебдодатків	Ruby on Rails	Django
Зручність, налагодженість реалізації машинного навчання	Невисока	Найвища

3.2 Визначення функціональних вимог до програми

Виділені функціональні можливості програми містять такий список:

- виставлення власного житла для орендування;
- керування власним житлом, виставленим для орендування;

- розрахунок вартості оренди житла на період;
- пошук житла для оренди;
- відображення схожого житла;
- прогнозування популярності житла для оренди;
- відображення запланованих проживань орендаря;
- відображення історії проживань орендаря;
- відображення календаря заселення людей за житлом власника;
- відображення історії проживань за житлом власника;
- відображення доступного для підтримки житла;
- відображення проживань для підтримки;
- спілкування між орендарями і власниками житла або помічниками;
- відображення всіх оцінювань житла;
- відображення виявлених проблем за житлом;
- відображення оцінювань орендаря власниками житла;
- скасування проживання;
- оцінювання житла орендарем;
- оцінювання орендаря власником або помічником;
- оформлення запиту на проживання;
- керування запитами на проживання за житлом;
- відображення інформації про житло;
- відображення пропозиції про оренду житла;
- редагування профілів орендаря і власника;
- відображення профілю орендаря, власника;
- авторизація і реєстрація користувачів, у тому числі за допомогою облікових записів Google.

Використовуючи ці функціональні можливості, було промодельовано сценарії роботи власника житла з програмою (рис. 3.1).

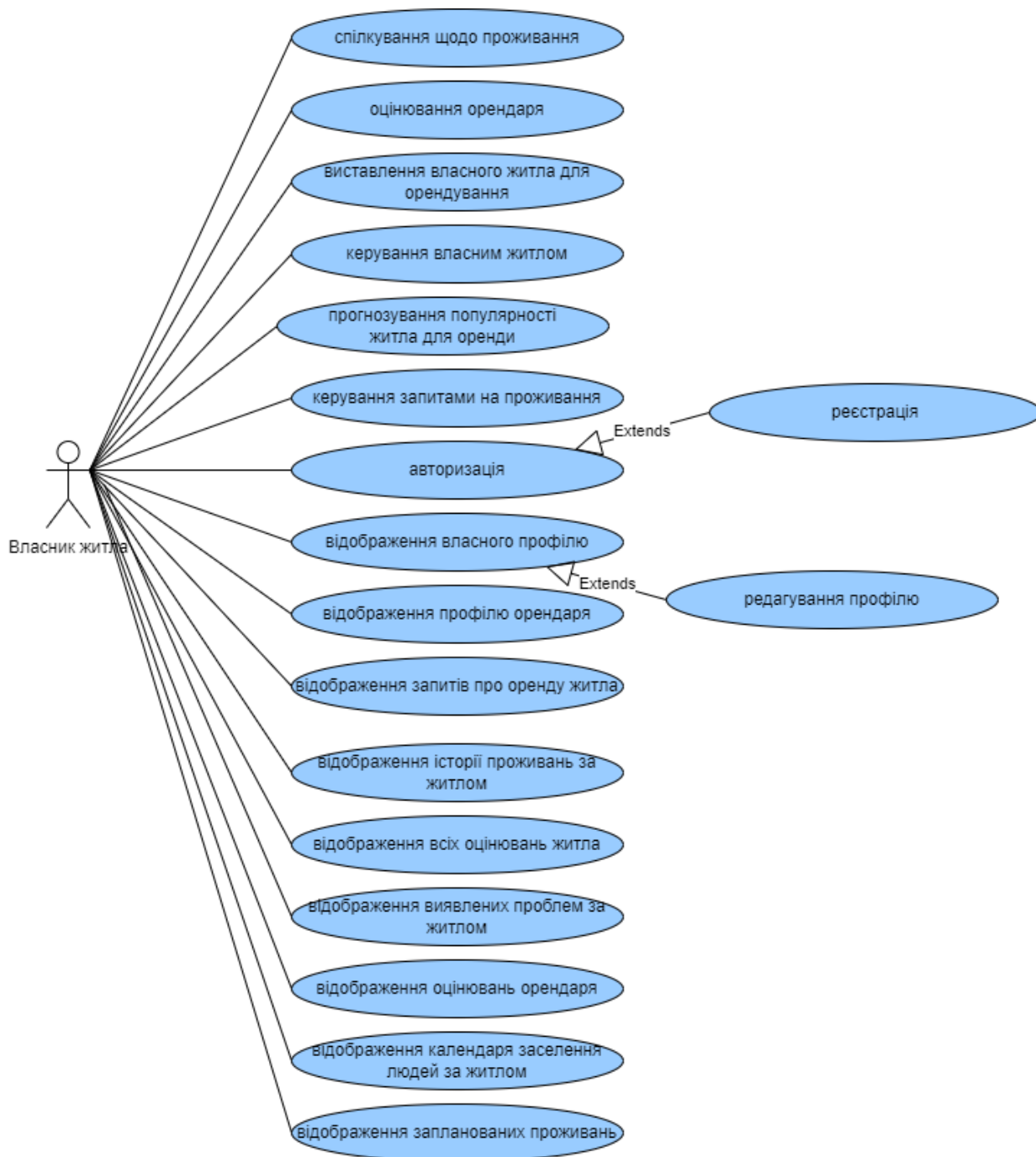


Рисунок 3.1 – Діаграма варіантів використання програми власником житла

Іншою стороною взаємодії є орендар (рис. 3.2). Допоміжну функцію для власника житла виконує актор помічник (рис. 3.3). Помічник відповідає за підтримку житла, яке належить власнику, і не має засобів керування.

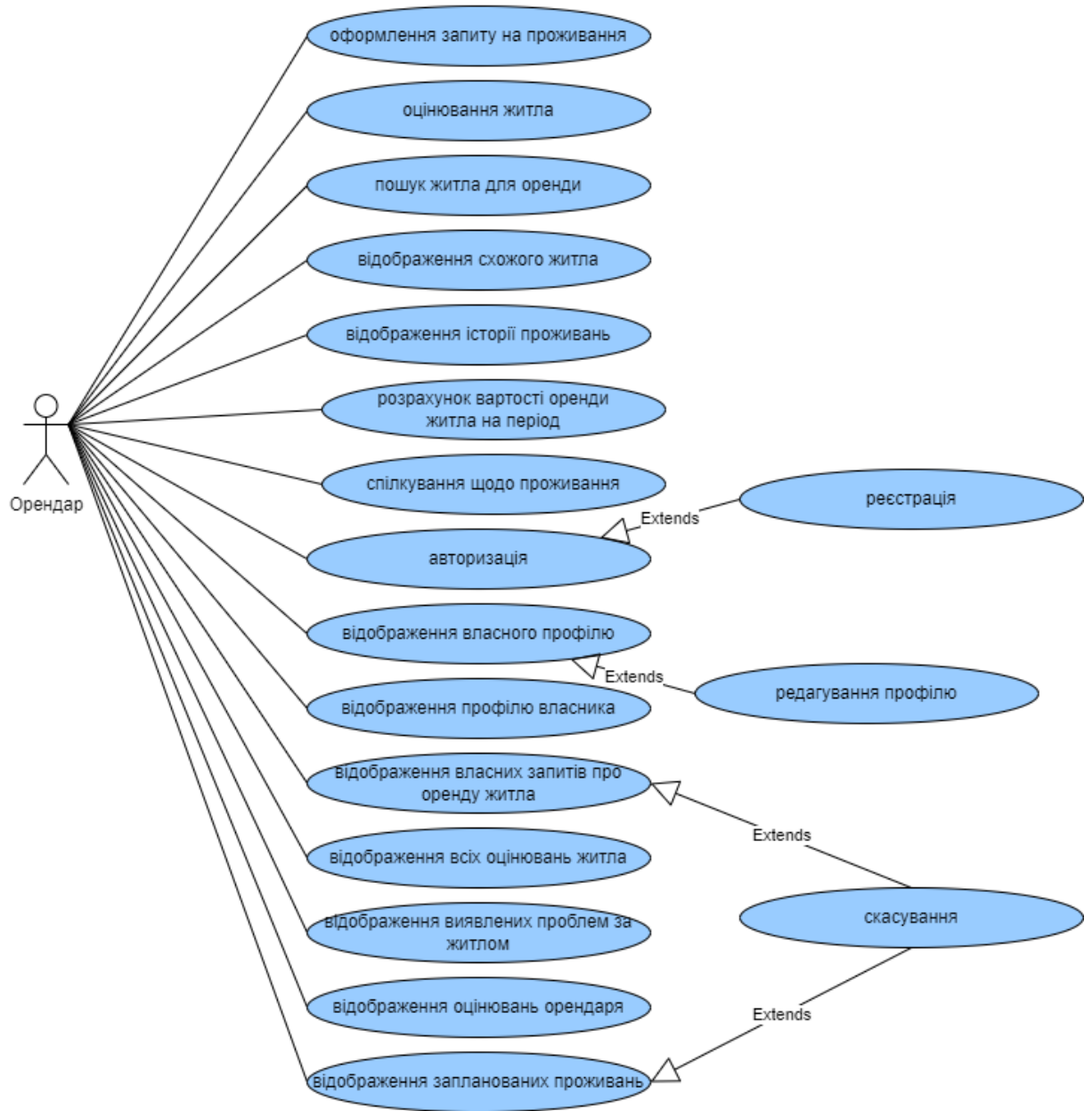


Рисунок 3.2 – Діаграма варіантів використання програми орендарем

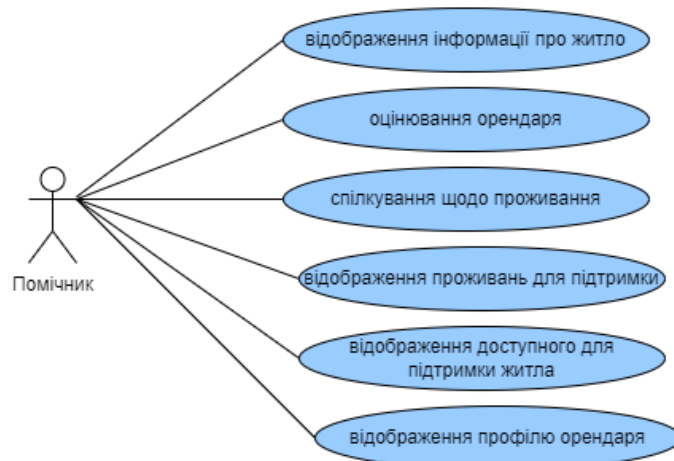


Рисунок 3.3 – Діаграма варіантів використання програми помічником

3.4 Архітектура програмної системи

Виходячи з описаної функціональності, яку повинна мати програма система, розробка якої виконується в роботі, була визначена її архітектура.

Архітектура побудована на тій лозіці, що в основі вебдодатку використовується фреймворк Django. Саме через нього фактично обробляються запити користувачів, які потрапляють спершу до вебсервера. Вебсервер комунікує напряду з Django-сервером, що належить до вебсистеми загалом. З вебсервером взаємодіють і орендарі, і власники житла, і їх помічники. Тобто будь-який користувач через свій клієнтський пристрій звертається до вебсервера, який взаємодіє з Django за отриманим запитом.

Django в процесі оброблення запитів, використовуючи всі власні складові на основі архітектури Model-View-Template, звертається за необхідності через моделі до бази даних (БД), що розташована на сервері БД. З сервером БД самостійно також взаємодіє підсистема керування оцінюваннями, де реалізується сам метод, а також його етапи зі збору необхідних даних з БД, а результати в підсумку записуються в БД, звідки їх бере Django за відповідного запиту власника житла.

Під час забезпечення авторизації користувачів через облікові записи Google Django використовує підсистему авторизації і через неї, використовуючи відповідний інтерфейс, звертається до Google-серверів, зокрема до серверу аутентифікації.

Платіжна підсистема обробляє запити на сплату за проживання. Реалізовано цей механізм має бути через платіжну систему PayPal, а тому відповідна платіжна підсистема взаємодіє з PayPal-шлюзом.

Створена архітектура (рис. 3.4) дозволяє в підсумку і організувати підтримку вебдодатку, і виконувати всі механізми роботи з даними.

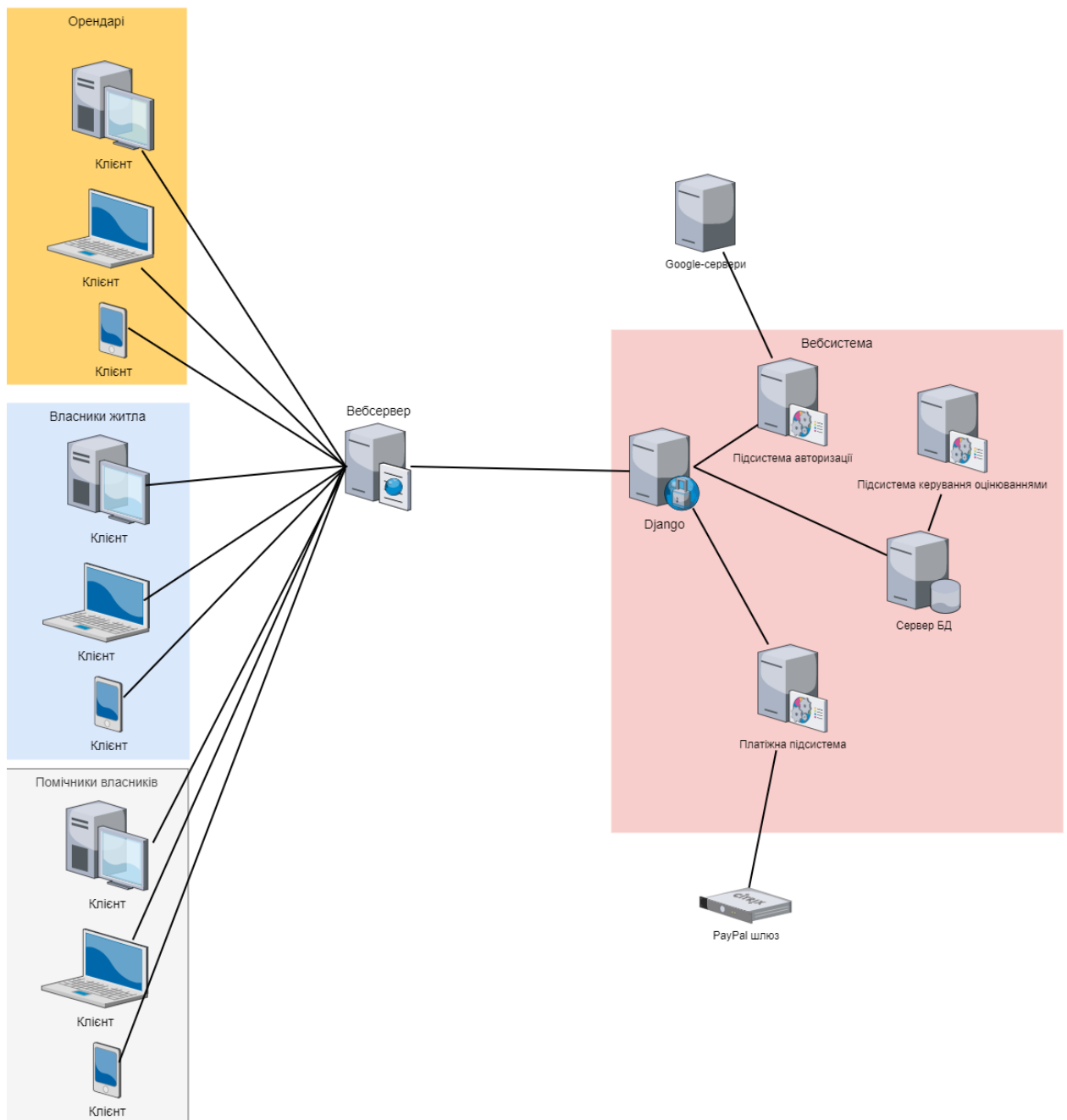


Рисунок 3.4 – Архітектура програмної системи

3.4 Висновки за розділом 3

Третій розділ став результатом проєктування програмного забезпечення. За цими результатами було остаточно визначено сценарії роботи всіх акторів з програмою, серед яких є як власники і орендарі, так і помічники власників, що дозволяє застосовувати програму для реальної роботи з нею, коли замість власника віддає ключі, інформує, контролює

орендаря, виконує підготовку до заселення інший користувач, який не може керувати замовленнями, але може отримати інформацію про те, з ким і коли він має контактувати, кого заселити, а також за необхідності спілкуватися з орендарем через програму, щоб узгодити потрібні питання. Програма має розроблятися через використання мови Python і фреймворку веброботи Django.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

4.1 Структура роботи з даними

БД є концептуально важливим сховищем, яке об'єднує результати роботи над замовленнями проживання житла, над самим житлом і пов'язаними з ним даними та результати прогнозування популярності житла, таким чином надаючи користувачу вебдодатку можливість працювати зі всіма даними через єдиний інтерфейс.

Окрім того структура БД визначає моделі для роботи з даними в Django.

Безпосередньо саме житло, яке здається в оренду, зберігається в таблиці `stylehomesoft_listing` з полями:

- `id` – ідентифікатор житла;
- `capture` – назва, за якою житло подається на своїй сторінці;
- `listingtype` – тип житла: котедж, пентхауз, номер в готелі, квартира, кімната в квартирі, гуртожиток, спільна кімната;
- `owner` – власник, який виставляє житло в оренду на сайті;
- `detailed` – деталізований опис житла;
- `rooms` – доступна кількість кімнат, якою може скористатися орендар;
- `maxguests` – максимальна кількість гостей, яка одночасно може орендувати житло;
- `maxdays` – максимальна кількість днів, на яку здається житло поспіль;
- `conditioner` – доступний або недоступний кондиціонер у житлі;
- `heat` – доступна або недоступна система обігріву житла;
- `washmachine` – доступна або недоступна пральна машина для використання орендарем;
- `access` – доступність житла для проживання осіб з інвалідністю;

- children – приймає або не приймає власник житла дітей (квартира може бути небезпечною для дітей);
- pets – приймає або не приймає власник житла орендарів з власними домашніми тваринами;
- smoking – можливість або заборона палити в житлі;
- silencetime – початок години тиші (нічний час);
- parties – можливість або неможливість використовувати житло для вечірок під час оренди;
- neighborhood – район, в якому знаходиться житло;
- address – повна адреса житла;
- xaddress – перша координата житла на мапі;
- yaddress – друга координата житла на мапі;
- sleepingplaces – кількість спальних місць, які сумарно можуть використовувати орендарі;
- toiletries – наявність або відсутність туалетних засобів для орендарів;
- dining – можливість приготування їжі власником для гостей або відсутність такої можливості;
- workplaces – кількість придатних робочих місць;
- style – стиль, який використовується в інтер'єрі житла;
- parkingplaces – кількість доступних місць для безкоштовного паркування автомобіля;
- payedparking – наявність платної парковки;
- nightcheckin – можливість нічного заселення в житло;
- nightcheckinprice – додаткова вартість нічного заселення;
- storey – поверх, на якому розташовано житло;
- stories – кількість поверхів у будівлі або у житлі, якщо це котедж;
- checkinfrom – час, з якого стандартно можна заїхати в житло в заброньований день;
- checkinto – час, до якого стандартно можна заїхати в житло в заброньований день;

- checkout – час, з якого стандартно потрібно виїхати з житла в останній день бронювання;
- kitchen – наявність кухні в житлі та можливість або неможливість на ній готувати;
- dishes – наявність посуду для приготування їжі на кухні;
- bath – варіант ванного вузла (відсутній, власна ванна, власний душ, власні ванна і душ, в блоці, на поверсі);
- dryer – наявність або відсутність фена для волосся;
- gasdetector – наявність або відсутність детектора чадного газу в житлі;
- smokedetector – наявність або відсутність детектора диму в житлі;
- petsinside – наявність або відсутність власних домашніх тварин у власника житла в спільних кімнатах;
- gym – наявність або відсутність доступу до спортивного залу (наприклад, зал наявний в будинку, в одній з квартир якого проживає людина);
- internet – варіант інтернет-доступу (відсутній, дротовий необмежений, дротовий обмежений, бездротовий необмежений, бездротовий обмежений, необмежений дротовий і бездротовий та обмежений дротовий і бездротовий);
- restzone – наявність зони відпочинку на вулиці;
- bikes – кількість велосипедів, які готовий надати власник у користування гостям;
- bedlinen – наявність або відсутність постільної білизни;
- position – середня оцінка, яка була виставлена на даний момент загалом житлу орендарями;
- lastposition – середня оцінка, яка була виставлена за останні 100 оцінювань житлу, якщо вони були створені протягом останніх півроку (якщо ні, то середня оцінка за оцінюваннями протягом півроку);
- reviewsnumber – кількість оцінювань житла;

- `promoed` – доступність промокодів для бронювання цього житла;
- `safety` – середня оцінка житла за рівнем безпеки проживання від орендарів;
- `accuracy` – середня оцінка житла за точністю власника (зокрема наскільки власник виконав умови прибуття за часом) від орендарів;
- `sleep` – середня оцінка житла за комфортом сну;
- `cleaness` – середня оцінка житла за чистотою.

Поле `owner` представляє зовнішній ключ з таблиці `stylehomesoft_owner` за зв'язком один до багатьох від зовнішньої таблиці: один користувач може бути власником різного житла, а для кожного житла повинен бути заданий один власник, який керує повністю всією інформацією про житло.

Поле `style` представляє зовнішній ключ з таблиці `stylehomesoft_style` за зв'язком один до багатьох від зовнішньої таблиці: один стиль може бути представлений в різному житлі, для чого він і був виділений серед даних, а кожне житло повинно характеризуватися одним стилем.

Поле `neighborhood` представляє зовнішній ключ з таблиці `stylehomesoft_neighborhood` за зв'язком один до багатьох від зовнішньої таблиці: один район може мати багато варіантів житла, а кожне житло повинно бути зафіксовано у одному конкретному районі.

Стили оформлення житла зберігаються в таблиці `stylehomesoft_style` з полями:

- `id` – ідентифікатор стилю;
- `capture` – назва стилю.

Міста зберігаються в таблиці `stylehomesoft_city` з полями:

- `id` – ідентифікатор міста, з житлом з якого працює програма;
- `capture` – назва міста;
- `country` – країна, в якій знаходиться місто.

Райони міст зберігаються в таблиці `stylehomesoft_neighborhood` з полями:

- `id` – ідентифікатор району міста;

- capture – назва району;
- city – місто, в якому знаходиться район;
- safety – рівень безпеки проживання;
- activity – рівень активності в районі міста.

Поле city представляє зовнішній ключ з таблиці stylehomesoft_city за зв'язком один до багатьох від зовнішньої таблиці: одне місто може мати декілька районів, а район відповідно належить конкретному місту.

Оцінювання житла зберігаються в таблиці stylehomesoft_listingopinion з полями:

- id – ідентифікатор оцінювання житла;
- listing – житло, оцінювання якого представлено;
- renter – орендар, який виконав оцінювання;
- position – загальний рівень (оцінка) проживання;
- opinion – текст, який висловлює оцінку житла;
- created – момент часу, коли було виконано оцінювання;
- safety – рівень (оцінка) безпеки проживання;
- accuracy – рівень (оцінка) точності власника;
- sleep – рівень (оцінка) житла за комфортом сну;
- cleanliness – рівень (оцінка) житла за чистотою.

Поле listing представляє зовнішній ключ з таблиці stylehomesoft_listing за зв'язком один до багатьох від зовнішньої таблиці: одне житло може багато разів бути оцінено орендарями (деякими і декілька разів у випадку повторного проживання), а кожне оцінювання стосується тільки одного конкретного житла.

Поле renter представляє зовнішній ключ з таблиці stylehomesoft_renter за зв'язком один до багатьох від зовнішньої таблиці: один орендар після власного проживання кожного такого разу може оцінити різне житло, а кожне оцінювання фіксується за конкретним орендарем, який проживав у цьому житлі.

Оцінювання орендарів зберігаються в таблиці stylehomesoft_renteropinion з полями:

- id – ідентифікатор оцінювання орендаря;
- renter – орендар, про якого цією оцінкою висловив свою думку власник житла або його помічник;
- owner – власник житла або його помічник, який цією оцінкою висловив свою думку про орендаря;
- position – число, яке відповідає оцінці;
- opinion – текстова оцінка;
- created – момент часу, коли було виконано оцінювання.

Поле renter представляє зовнішній ключ з таблиці stylehomesoft_renter за зв'язком один до багатьох від зовнішньої таблиці: один орендар може бути оцінений різними власниками житла, а кожне оцінювання стосується конкретного орендаря.

Поле owner представляє зовнішній ключ з таблиці stylehomesoft_owner за зв'язком один до багатьох від зовнішньої таблиці: один власник може оцінити різних орендарів, а кожне оцінювання фіксується за конкретним власником житла.

Дані власників житла та осіб, які їм допомагають, зберігаються в таблиці stylehomesoft_owner з полями:

- id – ідентифікатор власника житла або особи, яка виконує функції помічника;
- user – користувацький запис, яким користується цей власник житла;
- phone – номер телефону власника житла;
- first_name – ім'я власника житла;
- last_name – прізвище власника житла;
- detailed – текстова презентація власником житла себе;
- photo – фотографія власника житла;
- city – місто, в якому проживає власник житла.

Поле `user` представляє зовнішній ключ з таблиці `user` аутентифікації користувачів з системи Django за зв'язком один до одного: один власник може використовувати тільки один користувацький запис.

Поле `city` представляє зовнішній ключ з таблиці `stylehomesoft_city` за зв'язком один до багатьох від зовнішньої таблиці: в одному місті можуть проживати різні власники житла, проте для кожного власника має бути зафіксовано одне місто як основне місто його проживання.

Дані орендарів житла зберігаються в таблиці `stylehomesoft_renter` з полями:

- `id` – ідентифікатор орендаря житла;
- `user` – користувацький запис, яким користується цей орендар житла;
- `first_name` – ім'я орендаря житла;
- `last_name` – прізвище орендаря житла;
- `detailed` – текстова презентація орендарем житла;
- `photo` – фотографія орендаря житла;
- `passport` – дані паспорту орендаря житла;
- `position` – середня оцінка, виставлена власниками житла орендарю їх житла;
- `photo` – фотографія орендаря житла;
- `city` – місто, в якому проживає орендар житла.

Поле `user` представляє зовнішній ключ з таблиці `user` аутентифікації користувачів з системи Django за зв'язком один до одного: один орендар може використовувати тільки один користувацький запис.

Поле `city` представляє зовнішній ключ з таблиці `stylehomesoft_city` за зв'язком один до багатьох від зовнішньої таблиці: в одному місті можуть проживати різні орендарі житла, проте для кожного орендаря житла має бути зафіксовано одне місто як основне місто його проживання.

Дані про бронювання і після цього проживання за цими бронюваннями зберігаються в таблиці `stylehomesoft_rental` з полями:

- `id` – ідентифікатор проживання;

- renter – орендар, який проживає в житлі в певний момент;
- listing – житло, де проживає орендар;
- book_date – момент часу, коли було виконано бронювання цього проживання;
- checkin_date – момент часу, коли має розпочатися проживання;
- checkout_date – момент часу, коли проживання має бути завершено;
- price – сума, яку має бути сплачено за проживання;
- payed – сума, яку вже було сплачено за проживання;
- promod – застосування промокоду;
- state – стан проживання;
- children – кількість дітей, які будуть проживати;
- guestnumber – кількість гостей загалом, які будуть проживати.

Поле renter представляє зовнішній ключ з таблиці stylehomesoft_renter за зв'язком один до багатьох від зовнішньої таблиці: один орендар може мати різні проживання в програмі, а кожне проживання може бути створено тільки одним орендарем.

Поле listing представляє зовнішній ключ з таблиці stylehomesoft_listing за зв'язком один до багатьох від зовнішньої таблиці: в одному житлі можуть в різні періоди відбуватися проживання різних людей, а кожне проживання відбувається тільки в одному конкретному житлі.

Тарифи на проживання зберігаються в таблиці stylehomesoft_price з полями:

- id – ідентифікатор тарифу на проживання;
- listing – житло, на проживання в якому визначено тариф;
- start_act – дата, з якої тариф активується;
- end_act – дата, до якої активний тариф;
- days – мінімальна кількість днів, від якої починає активуватися тариф на проживання в цьому житлі;
- mindays – мінімальна кількість днів, на яку на ці дати можна орендувати житло;

- price – вартість доби проживання;
- refundpolicy – варіант повернення сплачених грошей за цим тарифом (не повертати, повертати повністю, повертати 50 %).

Поле listing представляє зовнішній ключ з таблиці stylehomesoft_listing за зв'язком один до багатьох від зовнішньої таблиці: щодо проживання в одному житлі може діяти багато різних тарифів у різні періоди часу, але кожен тариф визначається для конкретного житла.

Повідомлення між власником або його помічником та орендарем зберігаються в таблиці stylehomesoft_note з полями:

- id – ідентифікатор повідомлення між власником або його помічником та орендарем;
- owner – власник або його помічник (перша сторона повідомлення);
- renter – орендар (друга сторона повідомлення);
- created – момент часу, коли було написано повідомлення;
- notetext – текст повідомлення між власником або його помічником та орендарем;
- author – визначення того, яка зі сторін є автором повідомлення.

Поле renter представляє зовнішній ключ з таблиці stylehomesoft_renter за зв'язком один до багатьох від зовнішньої таблиці: один орендар може мати різну кількість повідомлень до різних (або одного) власника або помічника, а у кожному повідомленні має бути чітко вказано одного орендаря.

Поле owner представляє зовнішній ключ з таблиці stylehomesoft_owner за зв'язком один до багатьох від зовнішньої таблиці: один власник може мати різну кількість повідомлень до різних орендарів, а у кожному повідомленні має бути чітко вказано одного власника житла.

Промокоди, які можуть бути використані для зменшення вартості проживання, зберігається в таблиці stylehomesoft_promo з полями:

- id – ідентифікатор промокоду;
- promo_text – текст промокоду, який необхідно ввести для його застосування;

– discount – знижка від вартості проживання, яка надається за цього промокоду;

– active – активний промокод або вже деактивований.

Помічники, які допомагають власникам забезпечувати проживання орендарів, зберігаються в таблиці `stylehomesoft_helper` з полями:

– id – ідентифікатор запису;

– listing – житло, з забезпеченням якого допомагає помічник;

– helper – користувач-помічник;

– state – активність або неактивність даної участі в допомозі.

Таблиця `stylehomesoft_helper` є по суті реалізацією зв'язку багато до багатьох між таблицями `stylehomesoft_listing` і `stylehomesoft_owner`: одне житло може мати багато помічників для його підтримки, визначених власником, а один помічник може бути задіяний в підтримці різного житла.

Поле `helper` є зовнішнім ключем з таблиці `stylehomesoft_owner`, а поле `listing` є зовнішнім ключем з таблиці `stylehomesoft_listing`.

Агрегована інформація про оцінювання житла орендарями зберігається в таблиці `OpinionStatistic` з полями:

– id – ідентифікатор статистичного запису;

– listing – житло, якого стосується агреговане оцінювання;

– month – місяць, за яким виконана агрегація оцінювань;

– number – кількість оцінювань за даний місяць цього житла;

– obtained – спосіб отримання інформації (розрахунком існуючих оцінювань або прогнозуванням).

Поле `listing` представляє зовнішній ключ з таблиці `stylehomesoft_listing` за зв'язком один до багатьох від зовнішньої таблиці: кожне житло може мати багато агрегованих записів про оцінювання за кожен місяць його додавання до програми, але кожен запис стосується одного конкретного житла.

Схема БД, що відповідає даним таблицям і зв'язкам між ними, подана на рис. 4.1.

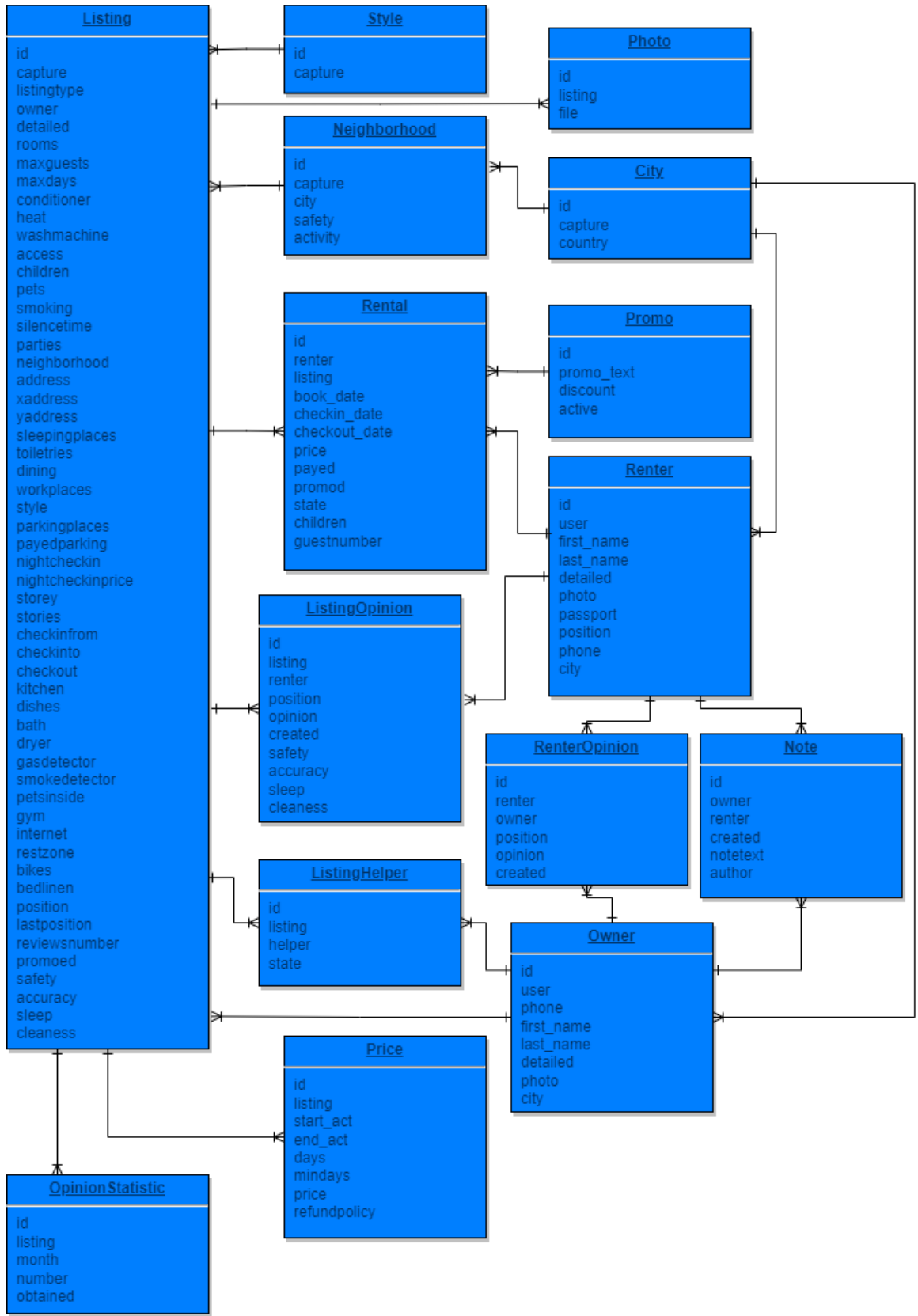


Рисунок 4.1 – Схема БД

Світлини житла зберігаються в таблиці Photo з полями:

- id – ідентифікатор світлини;

- listing – житло, яке зображено на світлині;
- file – дані файлу з фотографією.

Поле listing представляє зовнішній ключ з таблиці stylehomesoft_listing за зв'язком один до багатьох від зовнішньої таблиці: кожне житло може мати багато світлин, але кожна світлина представляє конкретне житло.

4.2 Класи вебдодатку

Не всі розроблені одиниці вебдодатку представлені саме класами, проте основна кількість одиниць, які реалізують функціональні можливості програми, реалізована саме за допомогою класів. Зважаючи на їх велику кількість, основна увага в даному підрозділі віддана тим, що презентують базові рішення для побудови основних функцій програми.

Безпосередньо для реалізації методу, запропонованого в розділі 2, було використано бібліотеку scikit-learn, а відповідний скрипт запускається на початку кожного місяця, агрегуючи дані для минулого місяця і виконуючи прогноз на наступний. Отримані результати заносяться в таблицю БД, а потім при відображенні власнику житла витягаються. Ці дані відображаються виключно власнику житла про його житло, а також доступні адміністратору системи через адміністративну панель. Інші користувачі не мають до них доступ. Для формування списку таких агрегованих записів з БД створено клас OwnerListingOpinionStatistics, що є реалізацією представлення на основі списку даних, яке визначається класом ListView. Для цього було перевизначено метод get_queryset, в якому формується безпосередньо сам список агрегованих записів. Для уникнення отримання даних іншими користувачами відбувається за переданою через параметр self і його атрибут request інформацією визначення користувача, який авторизувався в системі в даний момент, а за цим об'єктом user встановлюється відповідний об'єкт моделі Owner, і вже за ним і номером житла, переданим через параметр listing GET-форми, виконується запит на наявність такого житла. Якщо воно

дійсно належить даному власнику, то повертається відповідний об'єкт, а за ним запитується список агрегованих записів, впорядковуючи від останнього місяця. Якщо такого об'єкта не знайдено, тобто вказано номер житла, яке не асоційовано з авторизованим власником, то повертається тільки порожній перелік.

Одним з представлень вебдодатку, яке реалізує найрозгалуженішу логіку, є представлення `ListingSearch`, яке також є реалізацією класу `ListView` на основі моделі `Listing`. Дане представлення реалізує підтримку пошуку житла за заданими параметрами на задані дати для проживання. Метод `get_queryset` обробляє всі параметри, отримані через GET-запит.

Параметр `search_start` представляє дату початку проживання, яка відповідно передана як рядок, а тому за допомогою виклику функції `strptime` пакету `datetime` на основі шаблону «`%d.%m.%y`» перетворюється на дату класу `date`, за якою відбувається подальша перевірка. Аналогічно обробляється значення параметра `search_end`, що визначає дату закінчення проживання.

Також передаються через параметри GET-запиту наступні значення:

- `min_price` – мінімальна вартість за весь період проживання;
- `max_price` – максимальна вартість за весь період проживання;
- тип житла через відповідні параметри, до яких входять котедж (`lt1`), пентхауз (`lt2`), номер в готелі (`lt3`), квартира (`lt4`), кімната в квартирі (`lt5`), гуртожиток (`lt6`), спільна кімната (`lt6`): користувач може визначити декілька типів житла, відповідно відфільтровується будь-який з тих, що відповідають одному з заданих варіантів;
- `min_rooms` – мінімальна потрібна орендарю кількість кімнат;
- `max_rooms` – максимальна потрібна орендарю кількість кімнат: кількість кімнат у всіх відібраних варіантів житла має знаходитися в проміжку між двома останніми параметрами;
- `guests` – кількість гостей, яка буде проживати за запитом: відбувається перевірка, щоб максимальна кількість гостей, яка дозволена

власником у житлі, не була меншою за дану кількість з додаванням кількості дітей;

- conditioner – необхідний кондиціонер у житлі: за значення False залишаються варіанти житла і з кондиціонером, і без, а за значення True залишаються тільки варіанти з кондиціонером;

- heat – необхідна система обігріву в житлі: за значення False залишаються варіанти житла і з обігрівом, і без, а за значення True залишаються тільки варіанти з обігрівом;

- washmachine – необхідна пральна машина в житлі: за значення False залишаються варіанти житла і з пральною машиною, і без, а за значення True залишаються тільки варіанти з пральною машиною;

- access – необхідний варіант проживання з доступом для осіб з інвалідністю: за значення False залишаються варіанти житла і з доступом, і без, а за значення True залишаються тільки варіанти з забезпеченим доступом для проживання осіб з інвалідністю;

- children – кількість дітей, яка буде проживати: відбувається перевірка того, чи приймає житло дітей, якщо значення параметра більше 0, а якщо значення дорівнює 0, то залишаються і ті варіанти житла, які приймають дітей, і ті, які не приймають;

- pets – можливість проживання з власними домашніми тваринами: за значення False залишаються варіанти житла і з власними домашніми тваринами, і без, а за значення True залишаються тільки варіанти з можливістю проживання з власними домашніми тваринами;

- smoking – можливість палити в приміщенні: за значення False залишаються варіанти житла і з такою можливістю, і без, а за значення True залишаються тільки варіанти з можливістю палити в приміщенні;

- parties – можливість проводити вечірки: за значення False залишаються варіанти житла і з такою можливістю, і без, а за значення True залишаються тільки варіанти з можливістю проводити вечірки;

– neighborhood – номер району, в якому знаходиться житло: у випадку, якщо значення не дорівнює -1, то залишаються тільки варіанти житла з даного району міста;

– city – номер міста, в якому знаходиться житло: у випадку, якщо значення не дорівнює -1, то залишаються тільки варіанти житла з даного міста, але даний параметр розглядається тільки в тому випадку, якщо не задано параметр neighborhood;

– sleepingplaces – кількість спальних місць, які необхідні для проживання: залишаються тільки варіанти житла, де кількість спальних місць не менша ніж задана;

– toiletries – необхідний варіант проживання з уже наданими власником туалетними засобами для орендарів: за значення False залишаються варіанти житла і з цими засобами, і без, а за значення True залишаються тільки варіанти з наданими власником туалетними засобами;

– dining – необхідний варіант проживання з можливістю приготування їжі власником для гостей: за значення False залишаються варіанти житла і з цією можливістю, і без, а за значення True залишаються тільки варіанти з можливістю приготування їжі власником для гостей;

– workplaces – кількість придатних робочих місць: залишаються тільки ті варіанти житла, які містять кількість робочих місць не меншу ніж задана;

– style – номер стилю, який використовується в інтер'єрі житла: залишаються тільки варіанти житла, які оформлені в даному стилі;

– parkingplaces – кількість необхідних місць для безкоштовного паркування автомобіля: залишаються тільки варіанти житла, які мають не меншу кількість місць для безкоштовного паркування автомобіля ніж задана;

– payedparking – необхідність платної парковки: за значення False залишаються варіанти житла і з доступністю до платної парковки, і без, а за значення True залишаються тільки варіанти з доступністю до платної парковки;

– `nightcheckin` – необхідність нічного заселення в житло: за значення `False` залишаються варіанти житла і з можливістю нічного заселення в житло, і без, а за значення `True` залишаються тільки варіанти з можливістю нічного заселення в житло;

– `min_storey` – мінімальний поверх, на якому має бути розташовано житло;

– `max_storey` – максимальний поверх, на якому має бути розташовано житло: залишаються тільки ті варіанти житла, які розташовані на поверсі, номер якого знаходиться між попереднім і даним параметром;

– `not_last_storey` – вимога до того, щоб житло не знаходилося на останньому поверсі: якщо встановлено значення `True`, то відкидаються всі варіанти житла, у яких кількість поверхів дорівнює поверху, на якому розташовано житло, відповідно якщо встановлено значення `False`, то залишаються всі варіанти житла без перевірки даного правила;

– `max_stories` – максимальна кількість поверхів у будівлі, в якій розташовано житло: відкидаються всі варіанти житла, у яких кількість поверхів у будівлі більше ніж задана;

– `checkinfrom` – час, з якого готовий у перший день заселитися орендар: задається як рядок, перетворюється на об'єкт класу `time` з пакету `datetime` за допомогою функції `strptime` на основі шаблону «%H:%M»;

– `checkinto` – час, до якого готовий у перший день заселитися орендар: задається як рядок, перетворюється на об'єкт класу `time` з пакету `datetime` за допомогою функції `strptime` на основі шаблону «%H:%M», відповідно залишаються тільки ті варіанти житла, які відповідають хоч одному з 4 варіантів перетину інтервалів між найранішим і найпізнішим часом заїзду в житло та значеннями попереднього і поточного параметрів;

– `checkoutfrom` – час, з якого готовий у останній день виселитися орендар: задається як рядок, перетворюється на об'єкт класу `time` з пакету `datetime` за допомогою функції `strptime` на основі шаблону «%H:%M»;

– checkoutto – час, до якого готовий у останній день виселитися орендар: задається як рядок, перетворюється на об'єкт класу time з пакету datetime за допомогою функції strptime на основі шаблону «%H:%M»;

– kitchen – наявність кухні в житлі: за значення False залишаються варіанти житла і з кухнею, і без, а за значення True залишаються тільки варіанти житла з кухнею;

– dishes – наявність посуду для приготування їжі на кухні: за значення False залишаються варіанти житла і з посудом, і без, а за значення True залишаються тільки варіанти житла з посудом для приготування їжі на кухні;

– bath – варіант ванного вузла (відсутній, на поверсі, в блоці, власна ванна або душ, власна ванна, власний душ, власні ванна і душ): залишаються варіанти ванного вузла не менше заданого і рівні і встановленому (для трьох останніх варіантів);

– dryer – наявність фена для волосся: за значення False залишаються варіанти житла і з феном, і без, а за значення True залишаються тільки варіанти житла з феном;

– gasdetector – наявність детектора чадного газу в житлі: за значення False залишаються варіанти житла і з детектором, і без, а за значення True залишаються тільки варіанти житла з детектором чадного газу в житлі;

– smokedetector – наявність детектора диму в житлі: за значення False залишаються варіанти житла і з детектором, і без, а за значення True залишаються тільки варіанти житла з детектором диму в житлі;

– petsinside – наявність власних домашніх тварин у власника житла в спільних кімнатах: за значення False залишаються варіанти житла і з власними домашніми тваринами, і без, а за значення True залишаються тільки варіанти житла з домашніми тваринами власників;

– gym – наявність доступу до спортивного залу: за значення False залишаються варіанти житла і з доступом до спортивного залу, і без, а за значення True залишаються тільки варіанти житла з доступом до спортивного залу;

– internet – варіант інтернет-доступу: для кожного заданого значення інтернет-доступу визначаються допустимі варіанти інтернет-доступу в житлі, відповідно залишаються тільки ті варіанти житла, які надають один з таких варіантів доступу до інтернету;

– restzone – наявність зони відпочинку на вулиці: за значення False залишаються варіанти житла і з зоною відпочину, і без, а за значення True залишаються тільки варіанти житла з зоною відпочинку;

– bikes – кількість велосипедів, необхідна орендарю: залишаються тільки ті варіанти житла, в яких власники надають не меншу кількість велосипедів ніж задана;

– bedlinen – наявність постільної білизни: за значення False залишаються варіанти житла і з постільною білизною, і без, а за значення True залишаються тільки варіанти житла з наявністю постільної білизни;

– min_position – мінімальна середня оцінка, яка була виставлена на даний момент загалом житлу орендарями: залишаються тільки варіанти житла, середня оцінка яких не менша ніж задана;

– min_lastposition – мінімальна середня оцінка, яка була виставлена житлу орендарями останнім часом: залишаються тільки варіанти житла, середня остання оцінка яких не менша ніж задана;

– safety – мінімальна середня оцінка житла за рівнем безпеки проживання від орендарів: залишаються тільки варіанти житла, середня оцінка яких за цим показником не менша ніж задана;

– assurasy – мінімальна середня оцінка житла за точністю власника від орендарів: залишаються тільки варіанти житла, середня оцінка яких за цим показником не менша ніж задана;

– sleep – мінімальна середня оцінка житла за комфортом сну: залишаються тільки варіанти житла, середня оцінка яких за цим показником не менша ніж задана;

– cleanliness – мінімальна середня оцінка житла за чистотою: залишаються тільки варіанти житла, середня оцінка яких за цим показником не менша ніж задана.

Окрім попереднього прямого відбору за заданими значеннями параметрів після цього відбувається ще додатковий набір операцій.

Спочатку встановлюються всі варіанти вартості проживання на заданий період, тобто ті варіанти, які охоплюють хоча б частину періоду, за яким відбувається пошук. Також відбираються всі підтверджені проживання в даному житлі за цей же період. Якщо хоча б одне проживання належить даному періоду, то цей варіант житла відкидається з результуючого списку. Якщо ж таких проживань немає, то спочатку відбираються всі часові проміжки, в які змінюється вартість проживання в кожному житлі. Для кожного такого часового проміжку відбирається та вартість, яка відповідає найбільш підходящому варіанту вартості за кількістю гостей та за кількістю днів. З кожного такого цінового інтервалу складається загальна ціна. Якщо ціну хоча б на одному з проміжків не вдалося дізнатися, то такий варіант житла відкидається зі списку. Якщо ж для житла вдалося розрахувати повну вартість, то вона перевіряється на те, чи входить вона до заданого в параметрах проміжку вартості, який підходить орендарю.

Усі відібрані таким чином варіанти проживання повертаються у вигляді списку за результатами роботи методу.

Клас SimilarListings реалізує представлення відображення подібних варіантів житла на поточний. Поточний варіант житла визначається через параметри GET-запиту районом міста, стилем дизайну, типом житла, середньою оцінкою. Це саме ті параметри, за якими знаходиться подібне житло, тобто воно повинно бути розташоване в тому ж районі того ж міста, мати той же самий стиль дизайну, відповідати тому ж типу житла та мати приблизно рівну середню оцінку (в інтервалі 0,5 від оцінки поточного житла).

Клас `HelperListings` створює представлення відображення житла, з яким допомагає авторизований користувач (помічник). Для цього відбираються всі записи про підтримку житла авторизованим помічником (визначений як об'єкт класу `Owner`) моделі `ListingHelper`. Кожен такий варіант житла зберігається в загальному списку житла і повертається як результат роботи даного методу.

4.3 Висновки за розділом 4

Четвертий розділ став результатом реалізації програмного забезпечення. За цими результатами було описано структуру роботи з даними та логіку і структуру створених класів для підтримки основних функціональних можливостей програми.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Призначення програми

Програма призначена для того, щоб створити інструмент для бронювання житла від власників орендарями, виконуючи для цього пошук житла, його оцінювання. Для підтримки даного процесу програма забезпечує власників житла можливістю виставляти своє житло в оренду, керувати бронюваннями орендарів та прогнозуванням популярності житла в наступний період. Орендарів програма забезпечує можливістю шукати житло, бронювати його.

5.2 Умови виконання програми

Основні вимоги до технічних засобів формують вимоги до серверної частини. Сервер повинен забезпечувати і роботу вебдодатка, і роботу інтелектуальної частини, що реалізує запропонований метод. Це потребує відповідної обчислювальної потужності. Тому на сервері має бути забезпечено щонайменше процесор, який має не менше ніж 4 ядра, частоту від 2 ГГц, оперативна пам'ять має бути не меншою від 4 Гб, на жорсткому диску має бути доступно не менше ніж 10 Гб, адже передбачається можливість завантаження фотографій житла власниками на сервер.

5.3 Виконання програми

Запускається програма відповідно до стандартної процедури запуску Django-додатку на сервері, також має бути заплановано запуск кожного місяця скрипта підтримки роботи методу `my_method.py`.

Клієнтські пристрої звертаються до сервера через єдину адресу. За цією адресою відтворюється варіант, в якому працює орендар, але з

заблокованими сторінками, які потребують будь-якого збереження даних, тобто виконується перенаправлення на сторінку авторизації. З цієї сторінки можна перейти до реєстрації. Під час реєстрації за замовчуванням реалізується реєстрація орендаря, але з цієї сторінки можна перейти до реєстрації власника або помічника. Можна або ввести нові облікові дані, або використати обліковий запис Google. Після авторизації відтворюється основна сторінка для відповідної ролі користувача. Дизайн вебдодатку створений на основі вільного шаблону [18].

Орендар (рис. 5.1) виконує пошук через однойменний пункт загального меню. Під час пошуку потрібно визначити значення всіх параметрів пошуку, які були описані в розділі 4.

The screenshot displays the APARENT web application interface. At the top, there is a navigation bar with the APARENT logo and links for 'Пошук' (Search), 'Проживання' (Living), 'Спілкування' (Communication), 'Профіль' (Profile), and 'Вийти' (Logout). Below the navigation bar is a large image of a modern living room with a TV, plants, and a window. Below the image, there is a section for selecting dates and the number of guests. The date range is from 06/12/2022 to 07/12/2022, with 2 guests and 0 children. A blue button labeled 'Зарезервувати' (Reserve) is present. The price is shown as 55 \$. To the right, the section is titled 'ЖИТЛО' (Accommodation) and 'Happy Apartments'. A description follows: 'У нас затишно і тепло як вдома. Ми робимо особливий акцент на чистоті та зручності в інтер'єрі. Багато корисних дрібничок допоможуть зробити ваш відпочинок незабутнім. Розташування в центрі, зовсім поруч з фонтаном Треві допоможе залишатись поруч з практично усіма історичними місцями Риму.' Below the description are four circular icons representing amenities: 'Ванна в номері' (Bathroom in room), 'Кухня доступна' (Kitchen available), 'Кондиціонер доступний' (Air conditioner available), and 'Обігрів доступний' (Heating available).

Рисунок 5.1 – Вебсторінка вебдодатку під час роботи орендаря

За кожним результатом можна перейти на сторінку цього житла (рис. 5.1), де одразу доступна форма бронювання, яка дозволяє створити запит до власника, вказуючи дати, кількість гостей та окремо від цієї кількості кількість дітей. Автоматично розраховується вартість такого проживання. Використовується єдине числення в американських доларах.

На сторінці виводяться всі характеристики житла. Нижче також розташовано інформацію про господаря з кнопкою, яка дозволяє написати йому повідомлення. Ще нижче знаходяться останні оцінювання (можна перейти до повної сторінки оцінювань орендарями). Біля кожного оцінювання відображається інформація про орендаря, натиснувши на ім'я, можна перейти до його профілю. Внизу сторінки також доступні подібні варіанти проживання.

Після натискання на кнопку Зарезервувати на окремій сторінці при підтвердженні бронювання можна ввести промокод, який може знизити вартість проживання. Після підтвердження на наступній сторінці потрібно ввести платіжні дані. Якщо власник скасовує бронювання, то гроші повертаються за нього за тими параметрами повернення, які були вказані в оголошенні.

Керувати власним профілем можна з однойменного розділу основного меню. Спілкування через однойменний пункт меню відображає список всіх власників, яким орендар писав повідомлення, звідки можливо перейти до повного переліку повідомлень за цим власником.

Пункт меню Проживання відображає історію бронювань, запити (бронювання орендарів, які ще не були підтверджені власниками) та майбутні проживання (бронювання, підтверджені власником). Підпункти цього меню мають однойменні назви. Для майбутніх проживань доступна кнопка Скасувати (як і для запитів).

Помічник власника (рис. 5.2) має доступ до проживань через однойменний пункт основного меню. На цій сторінці є список з варіантами житла, доступними для підтримки цьому користувачу. Можна обрати або всі,

або конкретне житло зі списку. За кожним проживанням вказується інформація про орендаря, строки проживання, номер телефону орендаря, адреса житла, фотографія орендаря та кнопка Написати для спілкування, якщо необхідно домовитись про заїзд орендаря.

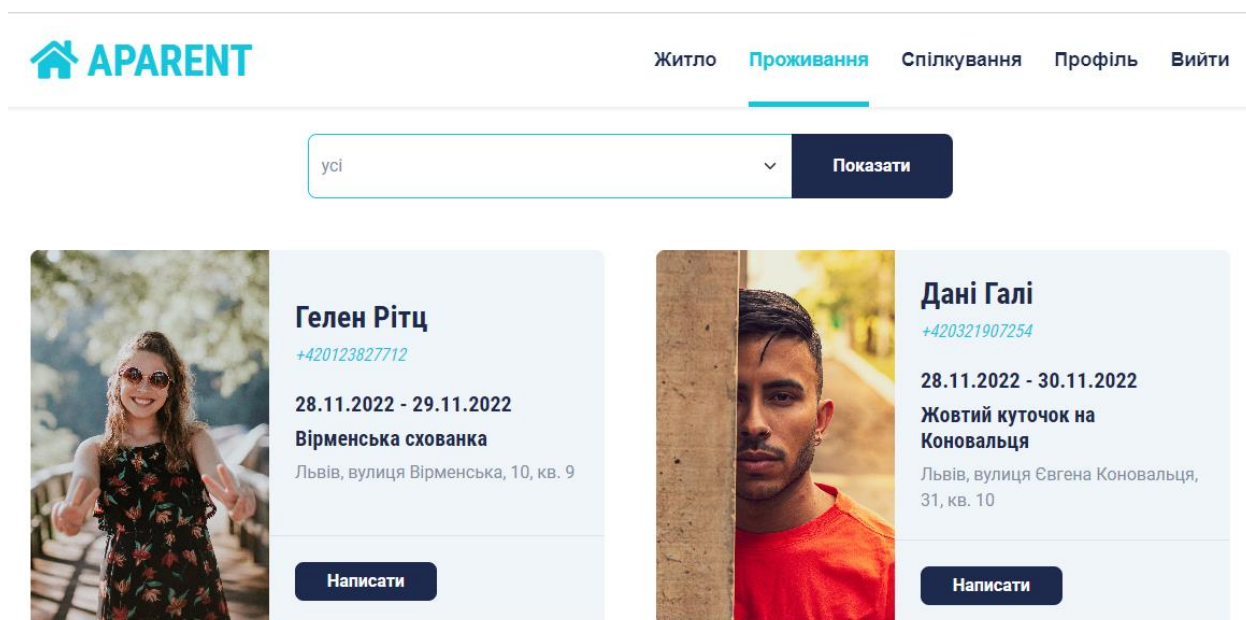


Рисунок 5.2 – Вебсторінка вебдодатку під час роботи помічника власника

Власник житла має різні керуючі інструменти, які не доступні помічнику. Через пункт меню Житло власник може переглядати, керувати власним житлом. Зі сторінки зі списком житла біля кожного житла за допомогою кнопки Популярність відбувається перехід на сторінку перегляду рівня популярності (рис. 5.3). На ній можна обрати потрібне житло серед тільки тих варіантів, у яких є власником авторизований користувач, тоді відображається графік з популярністю за поточний рік. Це кількість оцінювань, яка була накопичена за рік на кожен відповідний місяць.

За пунктом меню Проживання власник може перейти до заселень (власник може також обирати дату з календаря), запитів (може скасовувати бронювання) та історії (бронювання, які вже відбулись).

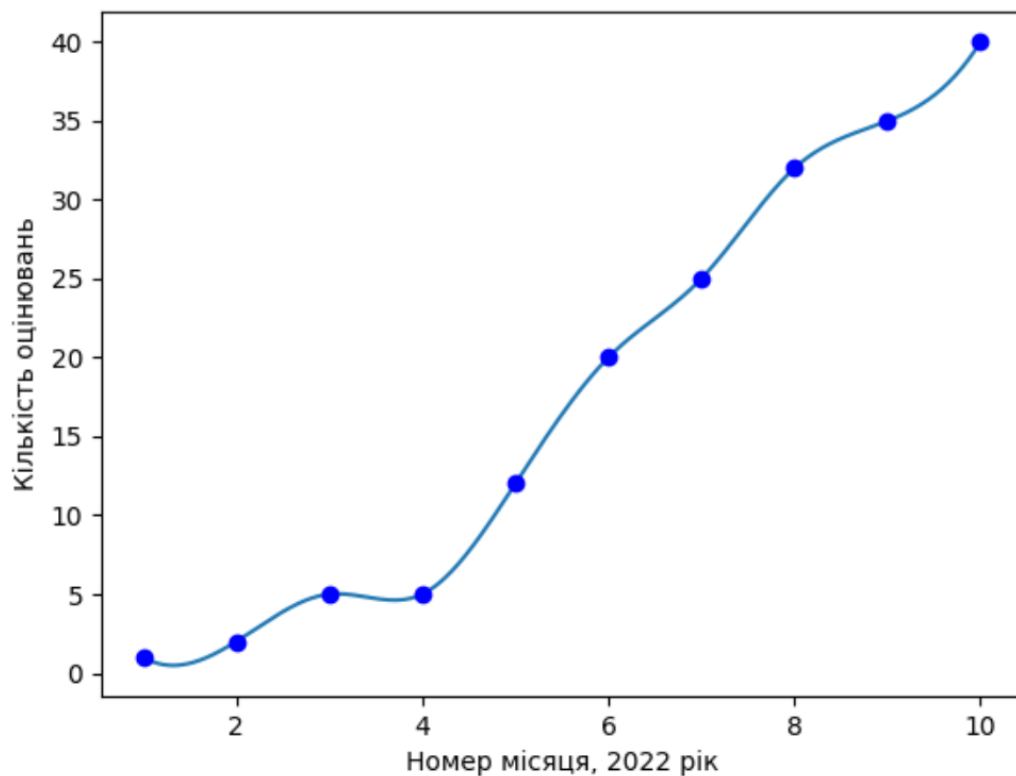


Рисунок 5.3 – Вебсторінка вебдодатку під час роботи власника житла

5.4 Експериментальне дослідження

Для перевірки ефективності застосування методу, який був представлений в розділі 2, для прогнозування популярності житла для оренди була використана вибірка даних [16], яка складається з декількох файлів і містить інформацію про оренду житла в місті Копенгаген (Данія) на платформі Airbnb.

Оскільки дані про оренду містяться в декількох файлах, то для створення вибірки, на якій відбувалось навчання і експерименти з тестовими даними, спочатку було реалізовано злиття цих файлів у єдину вибірку. Перед цим було виконано оброблення вибірки. Для цього було вилучено ті оголошення, які протягом останніх 6 місяців взагалі не були виставлені на платформі.

Спочатку було виділено всі оголошення, які доступні в файлах listings,

потім для кожного оголошення було виділено набір записів, кожен з яких представляє кількість оцінювань за 7 місяців (6 використовуються на вхід, а 7-ий місяць на виході). Відповідно за наявності даних за 12 місяців формувався запис за перші 7 місяців, потім за місяці, починаючи з другого і до восьмого, і т.д. Кожен запис було сформовано на основі наступних показників:

- тип житла: містить в файлі з оголошеннями;
- район проживання: містить в файлі з оголошеннями;
- кількість оцінювань за кожен з попередніх 6 місяців (6 окремих значень): в файлі з відгуками містяться всі оцінювання, для цього вони агрегуються і для кожного місяця обчислюється кількість оцінювань сумарно за місяць;
- середня вартість доби проживання за відповідний місяць: встановлене середнє значення на наступний місяць (сьомий місяць);
- середня вартість доби проживання за всі попередні 6 місяців: обчислюється безпосередньо на основі даних двох кварталів усередненням;
- кількість оцінювань за відповідний місяць (використовується як вихідна ознака, тобто саме це значення прогнозується): також обчислюється на основі даних про кількість оцінювань з файлу оцінювань.

На рис. 5.4 представлено графік залежності кількості оцінювань житла від місяця 2022 року. Графік сформований для 10 найпопулярніших оголошень на платформі Airbnb. Виділена кількість оцінювань за перші 8 місяців року. Крива, що відображена на графіку, визначає кумулятивне значення кількості оцінювань для демонстрації темпів збільшення цих значень у найпопулярніших варіантів житла.

Перевірка ефективності (табл. 5.1) використання методу прогнозування популярності житла для оренди виконувалась в порівнянні з іншими методами, здатними вирішувати задачі прогнозування, до яких було віднесено метод опорних векторів, який був задіяний на основі регресійної моделі, та лінійна регресія.

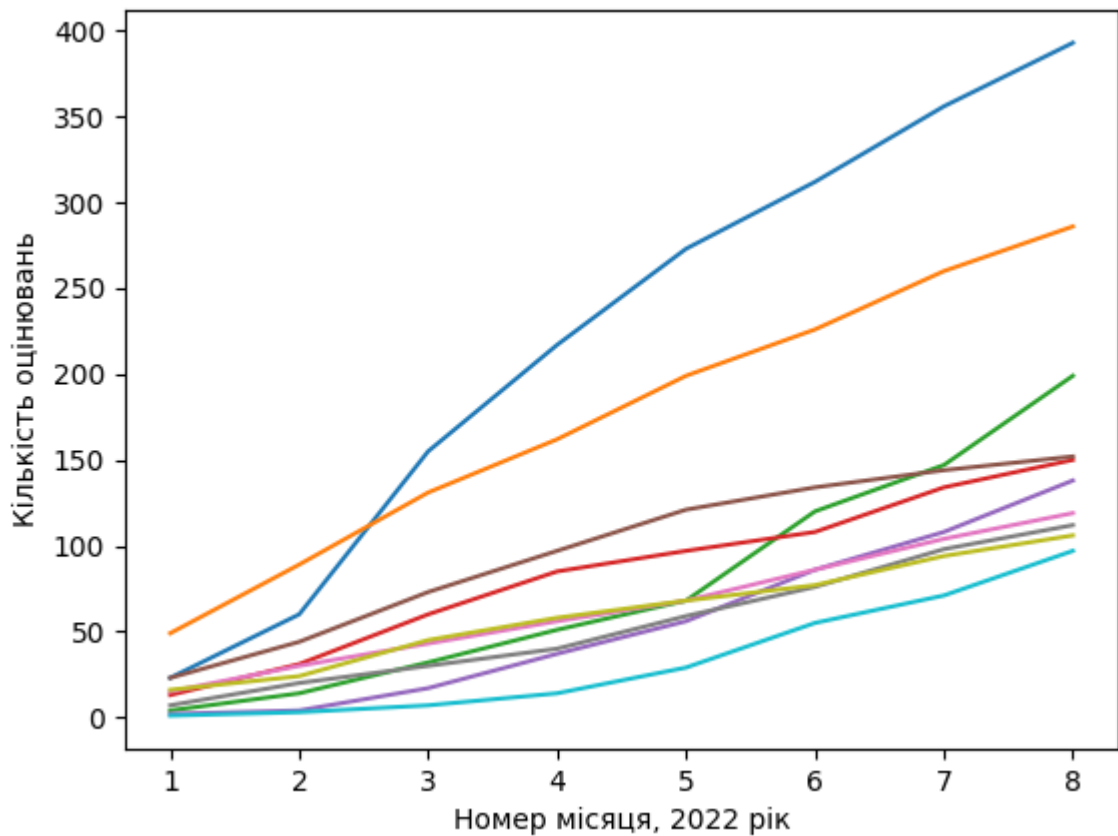


Рисунок 5.4 – Графік залежності кількості оцінювань житла від місяця 2022 року

Таблиця 5.1 – Результати експериментального дослідження

Модель або метод	R^2
Метод опорних векторів (регресійна модель)	0,71
Лінійна регресія	0,59
Метод прогнозування популярності житла для оренди	0,84

Для оцінки підсумкової ефективності (точності прогнозування) використовувався показник R^2 . Близькість його значення до 1 характеризує кращі значення, тобто такі, які повністю (за значення 1) відповідають очікуваним. Метод прогнозування популярності житла для оренди дозволив отримати значення показника R^2 в 0,84, що є найкращим результатом і

достатньо суттєво перевищує відповідне значення для лінійної регресії та методу опорних векторів.

5.5 Висновки за розділом 5

П'ятий розділ став результатом практичного застосування програми в межах даної проблеми: виконано перевірку роботи програми і описано процедури роботи з нею користувачів, виконано експериментальне дослідження, яке дозволило перевірити ефективність використання методу прогнозування популярності житла для оренди і результати якого підтвердили кращі прогнози саме даного методу порівняно з методом опорних векторів і лінійною регресією.

ВИСНОВКИ

Орендування житла від власників є відомою сферою, розвитку якої зокрема сприяли інформаційні технології. Проте на даний момент слід констатувати, що таке програмне забезпечення має дуже обмежену функціональність стосовно аналітичної підтримки власників житла. В сучасних таких програмах відсутні засоби прогнозування, аналізу накопиченої інформації про проживання. Тому дана робота була направлена на усунення цих недоліків шляхом надання інструменту прогнозування популярності житла для оренди. Оцінювати популярність було вирішено на основі кількості оцінювань житла, які були виконані орендарями щомісяця.

Науковою новизною роботи є метод прогнозування популярності житла для оренди, який дозволяє прогнозувати кількість оцінювань житла за наступний місяць на основі характеристик житла і даних минулих місяців, використовуючи алгоритм RF, що дозволяє завчасно отримувати інформацію, яку можна використати для прийняття рішень про зміну параметрів оголошення про оренду житла.

Практична цінність роботи полягає в розробленому програмному забезпеченні, яке може використовуватись у вигляді координаційного центру для власників житла, їх помічників та орендарів. Помічникам надається інформація, необхідна для розуміння розкладу заселення в житло, де помічник підтримує ці процеси. Власникам житла надається можливість традиційно виставляти житло в оренду, керувати бронюваннями, але також отримувати візуальне представлення інформації про популярність власного житла, включаючи прогнозування такої популярності на наступний місяць. Орендар же має коло засобів для пошуку, бронювання житла, отримання інформації про проживання, які вже завершилися, які обробляються та які можуть розпочнутись в певний момент у майбутньому.

Усе це в підсумку дозволяє стверджувати, що мета і завдання роботи досягнуто в повній мірі.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Apartment rents are shooting up in hundreds of cities across the U.S. Here's why. – CBS News [Electronic resource]. – Access mode : <https://www.cbsnews.com/news/rent-apartments-rising-housing-market-us/>.
2. Bhattarai, A. Rents are rising everywhere. See how much prices are up in your area [Electronic resource] / A. Bhattarai, C. Alcantara, A. V. Dam. – Access mode : <https://www.washingtonpost.com/business/interactive/2022/rising-rent-prices/>.
3. Thompson, D. Why the Rent Inflation Is So Damn High [Electronic resource] / D. Thompson. – Access mode : <https://www.theatlantic.com/ideas/archive/2022/08/rent-inflation-housing-demand-prices/671179/>.
4. Carbonaro, G. Europe's cost of living crisis [Electronic resource] : This is how much more rent you are paying in your country / G. Carbonaro. – Access mode : <https://www.euronews.com/next/2022/10/10/europes-cost-of-living-crisis-this-is-how-much-more-rent-you-are-paying-in-your-country>.
5. Оренда замських будинків та кондо – Airbnb [Електрон. ресурс]. – Режим доступу : <https://www.airbnb.com.ua/>.
6. Booking.com Офіційний сайт. Найкращі готелі, авіаквитки, авто напрокат і помешкання [Електрон. ресурс]. – Режим доступу : <https://www.booking.com>.
7. The World's Most Inspiring and Creative Vacation Rentals Plum Guide [Electronic resource]. – Access mode : <https://www.plumguide.com/>.
8. Norah, L. Airbnb Alternatives [Electronic resource] : the Best Sites Like Airbnb in 2022 / L. Norah. – Access mode : <https://www.findingtheuniverse.com/alternatives-to-airbnb/>.
9. Sonder. A better way to stay. [Electronic resource]. – Access mode : <https://www.sonder.com/>.
10. How To Improve Your Hotel's Popularity Ranking On TripAdvisor [Electronic resource]. – Access mode : <https://www.guesttouch.com/blog/how->

your-hotels-popularity-ranking-on-tripadvisor-can-be-improved.

11. Tripadvisor [Electronic resource] : Over a billion reviews & contributions for Hotels, Attractions, Restaurants, and more. – Access mode : <https://www.tripadvisor.com/>.

12. Donges, N. Random Forest Classifier [Electronic resource] : A Complete Guide to How It Works in Machine Learning / N. Donges. – Access mode : <https://builtin.com/data-science/random-forest-algorithm>.

13. How Is Random Forest Used For Classification and Regression Problems? [Electronic resource]. – Access mode : <https://h2o.ai/wiki/random-forest/>.

14. Random Forest Algorithm – How It Works & Why It's So Effective [Electronic resource]. – Access mode : <https://www.turing.com/kb/random-forest-algorithm>.

15. Mbaabu, O. Introduction to Random Forest in Machine Learning [Electronic resource] / O. Mbaabu. – Access mode : <https://www.section.io/engineering-education/introduction-to-random-forest-in-machine-learning/#:~:text=A%20random%20forest%20is%20a%20machine%20learning%20technique%20that's%20used,provide%20solutions%20to%20complex%20problems>.

16. Arora, A. Why Random Forests can't predict trends and how to overcome this problem? [Electronic resource] / A. Arora. – Access mode : <https://medium.datadriveninvestor.com/why-wont-time-series-data-and-random-forests-work-very-well-together-3c9f7b271631>.

17. Inside Airbnb [Electronic resource] : Get the Data. – Access mode : <http://insideairbnb.com/get-the-data/>.

18. MEDINOVA Free Website Template. Free CSS Templates. Free CSS [Electronic resource]. – Access mode : <https://www.free-css.com/free-css-templates/page283/medinova>.

ДОДАТОК А
Технічне завдання

Вступ

Орендування житла завжди є достатньо активним напрямком діяльності, а зараз вимушено цей напрямок в Україні отримує додатковий розвиток. Виставляючи своє житло в оренду, власники прагнуть привернути до нього увагу. В програмних системах, через які відбувається бронювання проживання, основним виразником популярності житла є кількість оцінювань або відгуків, які були залишені орендарями про житло. Саме це диктує впевненість наступних орендарів у своєму виборі. Тому кожен власник прагне розширювати кількість відгуків про своє житло, отримувати нові оцінювання і відгуки, щоб продемонструвати таким чином поточний стан житла. Саме тому створення інструменту прогнозування популярності житла для оренду є важливим не тільки саме по собі, але і впровадження його в програмну систему, яка використовується для бронювання житла, щоб власник міг, не переходячи в інші програмні засоби, одразу отримати прогноз і зрозуміти, як буде розвиватися в формальному представленні його житло на даному вебсайті найближчим часом.

A.1 Підстави для розробки

Розробка дипломної кваліфікаційної роботи магістра виконується на основі завдання за темою «Дослідження та програмна реалізація методів прогнозування популярності житла для оренди», затвердженою наказом № 363 від 4 листопада 2022 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Програма розробляється для того, щоб створити інструмент для бронювання житла від власників орендарями, виконуючи для цього пошук

житла, його оцінювання. Для підтримки даного процесу програма повинна забезпечувати власників житла можливістю виставляти своє житло в оренду, керувати бронюваннями орендарів та прогнозуванням популярності житла в наступний період. Орендарів програма повинна забезпечувати можливістю шукати житло, бронювати його.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Програма повинна забезпечити користувачів наданням таких функціональних можливостей:

- виставлення власного житла для орендування;
- керування власним житлом, виставленим для орендування;
- встановлення вартості оренди житла на період;
- пошук житла для оренди;
- відображення схожого житла;
- прогнозування популярності житла для оренди;
- відображення запланованих проживань орендаря;
- відображення історії проживань орендаря;
- відображення календаря заселення людей за житлом власника;
- відображення історії проживань за житлом власника;
- відображення доступного для підтримки житла;
- відображення проживань для підтримки;
- скасування проживання;
- спілкування між орендарями і власниками житла або помічниками;
- відображення всіх оцінювань житла;
- відображення виявлених проблем за житлом;
- відображення оцінювань орендаря власниками житла;
- оцінювання житла орендарем;
- оцінювання орендаря власником або помічником;

- оформлення запиту на проживання;
- керування запитами на проживання за житлом;
- відображення інформації про житло;
- відображення пропозиції про оренду житла;
- редагування профілів орендаря і власника;
- відображення профілю орендаря, власника;
- авторизація і реєстрація користувачів, у тому числі за допомогою облікових записів Google.

А.3.2 Вимоги до складу та параметрів технічних засобів

Основні вимоги до технічних засобів формують вимоги до серверної частини. Сервер повинен забезпечувати і роботу вебдодатка, і роботу інтелектуальної частини, що реалізує запропонований метод. Це потребує відповідної обчислювальної потужності. Тому на сервері має бути забезпечено щонайменше процесор, який має не менше ніж 4 ядра, частоту від 2 ГГц, оперативна пам'ять має бути не меншою від 4 Гб, на жорсткому диску має бути доступно не менше ніж 10 Гб, адже передбачається можливість завантаження фотографій житла власниками на сервер.

А.3.3 Умови експлуатації

Експлуатація програми передбачає розгортання вебсервера, в ролі якого для вебдодатка може використовуватися uWSGI, Apache зокрема. Для керування БД має застосовуватися MySQL.

А.3.4 Вимоги до програмної документації

Основна програмна документація включає безпосередньо це технічне завдання, текст програми, що є додатками до пояснювальної записки.

А.4 Порядок контролю та приймання

Порядок контролю та приймання роботи визначається календарним планом, який входить до складу завдання на дипломну кваліфікаційну роботу магістра.

ДОДАТОК Б
Текст програми

Б.1 Текст файла helpviews.py

```

from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import ListView
from kodakapp.models import Owner
import datetime
from datetime import date

class HelperListings(LoginRequiredMixin, ListView):
    model = Listing
    template_name = 'kodakapp/helper_listings.html'

    def get_queryset(self, *args, **kwargs):
        user_acc = self.request.user
        helper = Owner.objects.get(user = user_acc)
        lhs = ListingHelper.objects.filter(helper = helper,
state = True)
        listings = []
        for l in lhs:
            if l.listing not in listings:
                listings.append(l.listing)
        return listings

class HelperRentals(LoginRequiredMixin, ListView):
    model = Rental
    template_name = 'kodakapp/helper_rentals.html'

    def get_queryset(self, *args, **kwargs):
        rentals = super().get_queryset(*args, **kwargs)
        user_acc = self.request.user
        helper = Owner.objects.get(user = user_acc)
        lhs = ListingHelper.objects.filter(helper = helper,
state = True)
        listings = []
        for l in lhs:
            if l.listing not in listings:
                listings.append(l.listing)
        moment = datetime.datetime.now()
        moment_day = date(moment.year, moment.month,
moment.day)
        rentals = rentals.filter(listing__in = listings,
checkin_date__gte = moment_day)
        rentals = rentals.order_by('checkin_date')
        return rentals

```

```

class HelperListingRentals(LoginRequiredMixin, ListView):
    model = Rental
    template_name = 'kodakapp/helper_listing_rentals.html'

    def get_queryset(self, *args, **kwargs):
        rentals = super().get_queryset(*args, **kwargs)
        user_acc = self.request.user
        helper = Owner.objects.get(user = user_acc)
        listing = int(self.request.GET.get('listing', '-1'))

        if listing:
            lh = ListingHelper.objects.filter(helper =
helper, state = True, listing = listing)
            if lh:
                moment = datetime.datetime.now()
                moment_day = date(moment.year,
moment.month, moment.day)
                rentals = rentals.filter(listing = listing,
checkin_date__gte = moment_day)
                rentals = rentals.order_by('checkin_date')
                return rentals
            else:
                return []

class HelperOwnerListingRentals(LoginRequiredMixin,
ListView):
    model = Rental
    template_name =
'kodakapp/helper_ownerlisting_rentals.html'

    def get_queryset(self, *args, **kwargs):
        rentals = super().get_queryset(*args, **kwargs)
        user_acc = self.request.user
        helper = Owner.objects.get(user = user_acc)
        owner = int(self.request.GET.get('owner', '-1'))
        owner = Owner.objects.get(id = owner)
        if owner:
            lhs = ListingHelper.objects.filter(helper =
helper, state = True, listing__owner = owner)
            if lhs:
                listings = []
                for l in lhs:
                    if l.listing not in listings:
                        listings.append(l.listing)
                moment = datetime.datetime.now()

```

```

        moment_day = date(moment.year,
moment.month, moment.day)
        rentals = rentals.filter(listing__in =
listings, checkin_date__gte = moment_day)
        rentals = rentals.order_by('checkin_date')
        return rentals
    else:
        return []
else:
    return []

```

Б.2 Текст файла listingviews.py

```

from django.contrib.auth.mixins import LoginRequiredMixin
from django.shortcuts import render, redirect,
get_object_or_404
from django.urls import reverse_lazy
from django.views.generic import DetailView, ListView
from kodakapp.forms import ListingFrm, ListingsSearchFrm
from kodakapp.models import Listing, ListingOpinion, Price,
Owner, Style, Neighborhood, City, Rental
from django.db.models import Q, F
import datetime
from datetime import date, timedelta, datetime

class ListingObserve(DetailView):
    model = Listing
    template_name = 'kodakapp/listing.html'

    def get(self, request, pk):
        apartment =
Listing.objects.prefetch_related('helpers').get(id = pk)
        opinions = ListingOpinion.objects.filter(listing =
apartment).order_by('-created')[0:5]
        moment = datetime.datetime.now()
        moment_day = date(moment.year, moment.month,
moment.day)
        prices = Price.objects.filter(listing = apartment,
start_act__gte = moment).order_by('start_act')
        rentals = Rental.objects.filter(listing =
apartment, checkin_date__gte = moment, state = 1)
        dates = []
        for price in prices:
            start_day = price.start_act
            end_day = price.end_act

```

```

        while start_day <= end_day:
            start_day = start_day + timedelta(days = 1)
            rental = rentals.filter(checkin_date__lte =
start_day, checkout_date__gte = start_day)
            if not rental:
                date.append(start_day)
        context = {'listing': apartment,
                    'opinions': opinions,
                    'free_dates': dates}
        return render(request, self.template_name, context)

class ListingProposition(DetailView):
    model = Listing
    template_name = 'kodakapp/listing.html'

    def get(self, request, pk):
        apartment =
Listing.objects.prefetch_related('helpers').get(id = pk)
        opinions = ListingOpinion.objects.filter(listing =
apartment).order_by('-created')[0:5]
        guests = int(self.kwargs['guests'])
        moment = datetime.datetime.now()
        moment_day = date(moment.year, moment.month,
moment.day)

        start_day = int(self.kwargs['start_day'])
        start_month = int(self.kwargs['start_month'])
        start_year = int(self.kwargs['start_year'])
        end_day = int(self.kwargs['end_day'])
        end_month = int(self.kwargs['end_month'])
        end_year = int(self.kwargs['end_year'])
        start_date = date(start_year, start_month,
start_day)

        end_date = date(end_year, end_month, end_day)
        diff = start_date - end_date
        if apartment.maxdays < diff.days:
            possible = False
            prices = Price.objects.filter(listing = apartment,
start_act__gte = moment, guests__gte =
guests).order_by('start_act')
            rentals = Rental.objects.filter(listing =
apartment, checkin_date__gte = moment, state = 1)
            dates = []
            for price in prices:
                start_day = price.start_act
                end_day = price.end_act
                while start_day <= end_day:

```

```

        start_day = start_day + timedelta(days = 1)
        rental = rentals.filter(checkin_date__lte =
start_day, checkout_date__gte = start_day)
        if not rental:
            date.append(start_day)
            query1 = Q(start_act__lte = start_date,
end_act__gte = end_date)
            query2 = Q(start_act__gte = start_date,
end_act__gte = start_date, end_act__lte = end_date)
            query3 = Q(start_act__lte = start_date,
end_act__gte = start_date, end_act__lte = end_date)
            query4 = Q(start_act__gte = start_date,
end_act__lte = end_date)
            prices = prices.filter(query1 | query2 | query3 |
query4).order_by('start_act').distinct()
            price_dates = []
            for price in prices:
                start_day = price.start_act
                end_day = price.end_act
                if start_day not in price_dates:
                    price_dates.append(start_day)
                if end_day not in price_dates:
                    price_dates.append(end_day)
            days = []
            price = 0
            possible = True
            for i in range(1, len(price_dates)):
                if price_dates[i-1] < start_date:
                    temp1 = start_date
                else:
                    temp1 = price_dates[i-1]
                if price_dates[i] > end_date:
                    temp2 = end_date
                else:
                    temp2 = price_dates[i-1]
                calc_days = temp2 - temp1
                curr_prices = prices.filter(start_act =
price_dates[i-1], end_act = price_dates[i], days__lte =
calc_days).order_by('-days')
                guests_value = 1000
                to_do = curr_prices
                for p in to_do.values():
                    if p.guests < guests_value:
                        guests_value = p.guests
                curr_prices = curr_prices.filter(guests =
guests_value)

```

```

curr_prices =
curr_prices.order_by('-days')
curr_price =
curr.prices.values()[0]
curr_price = curr.prices.values()[0]
if curr_price:
    if curr_price.mindays <= calc_days:
        price = price + curr_price.price *
calc_days
    else:
        possible = False
else:
    possible = False
context = {'listing': apartment,
           'opinions': opinions,
           'free_dates': dates,
           'price': price,
           'possible': possible}
return render(request, self.template_name, context)

class ListingAdd(LoginRequiredMixin, View):
    template_name = 'kodakapp/add_listing.html'

    def get(self, request, pk = None):
        listingfrm = ListingFrm()
        cities = City.objects.all()
        neighborhoods = Neighborhood.objects.all()
        styles = Style.objects.all()
        frm_context = {'new_form': listingfrm,
                       'cities': cities,
                       'neighborhoods': neighborhoods,
                       'styles': styles }
        return render(request, self.template_name,
frm_context)

    def post(self, request, pk = None):
        listingfrm = ListingFrm(request.POST, request.FILES
or None)

        if not listingfrm.is_valid():
            cities = City.objects.all()
            neighborhoods = Neighborhood.objects.all()
            styles = Style.objects.all()
            frm_context = {'new_form': listingfrm,
                           'cities': cities,
                           'neighborhoods': neighborhoods,
                           'styles': styles }

```

```

        return render(request, self.template_name,
frm_context)
        listing = listingfrm.save(commit = False)
        logged_user = self.request.user
        owner = Owner.objects.get(user = logged_user)
        listing.owner = owner
        listing.save()
        listingfrm.save_m2m()
        return
    redirect(reverse_lazy('kodakapp:ownerlistings'))

class ListingEdit(LoginRequiredMixin, View):
    template_name = 'kodakapp/add_listing.html'

    def get(self, request, pk):
        apartment = get_object_or_404(Listing, id = pk)
        cities = City.objects.all()
        neighborhoods = Neighborhood.objects.all()
        styles = Style.objects.all()
        listingfrm = ListingFrm (instance = apartment)
        frm_context = {'new_form': listingfrm,
                        'cities': cities,
                        'neighborhoods': neighborhoods,
                        'styles': styles }
        return render(request, self.template_name,
frm_context)

    def post(self, request, pk = None):
        apartment = get_object_or_404(Listing, id = pk)
        listingfrm = ListingFrm (request.POST,
request.FILES or None, instance = apartment)
        if not listingfrm.is_valid():
            cities = City.objects.all()
            neighborhoods = Neighborhood.objects.all()
            styles = Style.objects.all()
            frm_context = {'new_form': listingfrm
                            'cities': cities,
                            'neighborhoods': neighborhoods,
                            'styles': styles }
            return render(request, self.template_name,
frm_context)

        listingfrm = listingfrm.save(commit = False)
        listingfrm.save()
        listingfrm.save_m2m()
        return reverse_lazy('kodakapp:ownerlistings')

```

```

class ListingSearch(ListView):
    model = Listing
    template_name = 'kodakapp/listings_find.html'

    def get_queryset(self, *args, **kwargs):
        listings = super().get_queryset(*args, **kwargs)
        listings_result = []
        search_start =
str(self.request.GET.get('search_start', ''))
        search_start = datetime.strptime(search_start,
'%d.%m.%y')
        search_end = str(self.request.GET.get('search_end',
''))
        search_end = datetime.strptime(search_end,
'%d.%m.%y')
        min_price = int(self.request.GET.get('min_price',
'0'))
        max_price = int(self.request.GET.get('max_price',
'-1'))
        listing_types = []
        listing_type_1 = int(self.request.GET.get('lt1',
'0'))
        listing_types.append(1)
        listing_type_2 = int(self.request.GET.get('lt2',
'0'))
        listing_types.append(2)
        listing_type_3 = int(self.request.GET.get('lt3',
'0'))
        listing_types.append(3)
        listing_type_4 = int(self.request.GET.get('lt4',
'0'))
        listing_types.append(4)
        listing_type_5 = int(self.request.GET.get('lt5',
'0'))
        listing_types.append(5)
        listing_type_6 = int(self.request.GET.get('lt6',
'0'))
        listing_types.append(6)
        listing_type_7 = int(self.request.GET.get('lt7',
'0'))
        listing_types.append(7)
        listings = listings.filter(listingtype__in =
listing_types)
        min_rooms = int(self.request.GET.get('min_rooms',
'1'))

```

```

        max_rooms = int(self.request.GET.get('max_rooms',
'1000'))
        listings = listings.filter(rooms__gte = min_rooms,
rooms__lte = max_rooms)
        guests = int(self.request.GET.get('guests', '1'))
        listings = listings.filter(maxguests__gte = guests)
        conditioner =
bool(self.request.GET.get('conditioner', '0'))
        if conditioner:
            listings = listings.filter(conditioner = True)
        heat = bool self.request.GET.get('heat', '0'))
        if heat:
            listings = listings.filter(heat = True)
        washmachine =
bool(self.request.GET.get('washmachine', '0'))
        if washmachine:
            listings = listings.filter(washmachine = True)
        access = bool(self.request.GET.get('access', '0'))
        if access:
            listings = listings.filter(access = True)
        children = int(self.request.GET.get('children',
'0'))
        if children:
            listings = listings.filter(children = True)
        pets = bool(self.request.GET.get('pets', '0'))
        if pets:
            listings = listings.filter(pets = True)
        smoking = bool(self.request.GET.get('smoking',
'0'))
        if smoking:
            listings = listings.filter(smoking = True)
        parties = bool(self.request.GET.get('parties',
'0'))
        if parties:
            listings = listings.filter(parties = True)
        city = int(self.request.GET.get('city', '-1'))
        neighborhood =
int(self.request.GET.get('neighborhood', '-1'))
        if neighborhood:
            nei = Neighborhood.objects.get(id =
neighborhood)
            listings = listings.filter(neighborhood = nei)
        else:
            cit = City.objects.get(id = city)
            listings = listings.filter(neighborhood__city =
cit)

```

```

        sleepingplaces =
int(self.request.GET.get('sleepingplaces', '1'))
        listings = listings.filter(sleepingplaces__gte =
sleepingplaces)
        toiletries =
bool(self.request.GET.get('toiletries', '0'))
        if toiletries:
            listings = listings.filter(toiletries = True)
        dining = bool(self.request.GET.get('dining', '0'))
        if dining:
            listings = listings.filter(dining = True)
        workplaces = int(self.request.GET.get('workplaces',
'0'))
        listings = listings.filter(workplaces__gte =
workplaces)
        style = int(self.request.GET.get('style', '-1'))
        sty = Style.objects.get(id = style)
        listings = listings.filter(style = sty)
        parkingplaces =
int(self.request.GET.get('parkingplaces', '0'))
        listings = listings.filter(parkingplaces__gte =
parkingplaces)
        payedparking =
bool(self.request.GET.get('payedparking', '0'))
        if payedparking:
            listings = listings.filter(payedparking = True)
        nightcheckin =
bool(self.request.GET.get('nightcheckin', '0'))
        if nightcheckin:
            listings = listings.filter(nightcheckin = True)
        min_storey = int(self.request.GET.get('min_storey',
'-1'))
        listings = listings.filter(storey__gte =
min_storey)
        max_storey = int(self.request.GET.get('max_storey',
'-1'))
        listings = listings.filter(storey__lte =
max_storey)
        not_last_storey =
bool(self.request.GET.get('not_last_storey', '0'))
        listings = listings.filter(storey__le =
F('stories'))
        max_stories =
int(self.request.GET.get('max_stories', '-1'))
        listings = listings.filter(stories__lte =
max_stories)

```

```

        checkinfrom =
str(self.request.GET.get('checkinfrom', '12:00'))
        checkinfrom = datetime.strptime(checkinfrom,
'%H:%M').time()
        checkinto = str(self.request.GET.get('checkinto',
'20:00'))
        checkinto = datetime.strptime(checkinto,
'%H:%M').time()
        query1 = Q(checkinfrom_date__lte = checkinfrom,
checkinto_date__gte = checkinto)
        query2 = Q(checkinfrom_date__gte = checkoutfrom,
checkinto_date__gte = checkoutfrom, checkinto_date__lte =
checkinto)
        query3 = Q(checkinfrom_date__lte = checkoutfrom,
checkinto_date__gte = checkoutfrom, checkinto_date__lte =
checkinto)
        query4 = Q(checkinfrom_date__gte = checkoutfrom,
checkinto_date__lte = checkinto)
        listings = listings.filter(query1 | query2 | query3
| query4)
        checkoutfrom =
str(self.request.GET.get('checkoutfrom', '06:00'))
        checkoutfrom = datetime.strptime(checkoutfrom,
'%H:%M').time()
        checkoutto = str(self.request.GET.get('checkoutto',
'12:00'))
        checkoutto = datetime.strptime(checkoutto,
'%H:%M').time()
        listings = listings.filter(checkout__lte =
checkoutto)
        kitchen = bool(self.request.GET.get('kitchen',
'0'))
        if kitchen:
            listings = listings.filter(kitchen = True)
        dishes = bool(self.request.GET.get('dishes', '0'))
        if dishes:
            listings = listings.filter(dishes = True)
        bath = int(self.request.GET.get('bath', '0'))
        if bath <= 4:
            listings = listings.filter(bath__gte = bath)
        elif bath == 5:
            listings = listings.filter(bath = 4)
        elif bath == 6:
            listings = listings.filter(bath = 5)
        elif bath == 7:
            listings = listings.filter(bath__gte = 4)

```

```

        dryer = bool(self.request.GET.get('dryer', '0'))
        if dryer:
            listings = listings.filter(dryer = True)
        gasdetector =
bool(self.request.GET.get('gasdetector', '0'))
        if gasdetector:
            listings = listings.filter(gasdetector = True)
        smokedetector =
bool(self.request.GET.get('smokedetector', '0'))
        if smokedetector:
            listings = listings.filter(smokedetector =
True)
        petsinside =
bool(self.request.GET.get('petsinside', '0'))
        if petsinside:
            listings = listings.filter(petsinside = True)
        gym = bool(self.request.GET.get('gym', '0'))
        if gym:
            listings = listings.filter(gym = True)
        internet = int(self.request.GET.get('internet',
'0'))
        if internet == 1:
            listings = listings.filter(internet__in = [1,
2, 5, 6])
        elif internet == 2:
            listings = listings.filter(internet__in = [2,
6])
        elif internet == 3:
            listings = listings.filter(internet__in = [3,
4, 5, 6])
        elif internet == 4:
            listings = listings.filter(internet__in = [4,
6])
        elif internet == 5:
            listings = listings.filter(internet__in = [5,
6])
        elif internet == 6:
            listings = listings.filter(internet = 6)
        restzone = bool(self.request.GET.get('restzone',
'0'))
        if restzone:
            listings = listings.filter(restzone = True)
        bikes = int(self.request.GET.get('bikes', '0'))
        listings = listings.filter(bikes__gte = bikes)
        bedlinen = bool(self.request.GET.get('bedlinen',
'0'))

```

```

        if bedlinen:
            listings = listings.filter(bedlinen = True)
        min_position =
int(self.request.GET.get('min_position', '0'))
        listings = listings.filter(position__gte =
min_position)
        min_lastposition =
int(self.request.GET.get('min_lastposition', '0'))
        listings = listings.filter(lastposition__gte =
min_lastposition)
        safety = int(self.request.GET.get('safety', '0'))
        listings = listings.filter(safety__gte = safety)
        accuracy = int(self.request.GET.get('accuracy',
'0'))
        listings = listings.filter(accuracy__gte =
accuracy)
        sleep = int(self.request.GET.get('sleep', '0'))
        listings = listings.filter(sleep__gte = sleep)
        cleanliness = int(self.request.GET.get('cleaness',
'0'))
        listings = listings.filter(cleaness__gte =
cleaness)
        listings = listings.order_by('-position')
        listings = listings.values()
        for listing in listings:
            if listing.maxguests < guests + children:
                possible = False
            else:
                prices = Price.objects.filter(listing =
listing, guests__gte = guests)
                query1 = Q(start_act__lte = search_start,
end_act__gte = search_end)
                query2 = Q(start_act__gte = search_start,
end_act__gte = search_start, end_act__lte = search_end)
                query3 = Q(start_act__lte = search_start,
end_act__gte = search_start, end_act__lte = search_end)
                query4 = Q(start_act__gte = search_start,
end_act__lte = search_end)
                prices = prices.filter(query1 | query2 |
query3 | query4).order_by('start_act').distinct()
                rentals = Rental.objects.filter(listing =
listing, state = 1)
                query1 = Q(checkin_date__lte =
search_start, checkout_date__gte = search_end)

```

```

        query2 = Q(checkin_date__gte =
search_start, checkout_date__gte = search_start,
checkout_date__lte = search_end)
        query3 = Q(checkin_date__lte =
search_start, checkout_date__gte = search_start,
checkout_date__lte = search_end)
        query4 = Q(checkin_date__gte =
search_start, checkout_date__lte = search_end)
        rentals = rentals.filter(query1 | query2 |
query3 | query4).distinct()
        if not rentals:
            price_dates = []
            for price in prices:
                start_day = price.start_act
                end_day = price.end_act
                if start_day not in price_dates:
                    price_dates.append(start_day)
                if end_day not in price_dates:
                    price_dates.append(end_day)
            price = 0
            possible = True
            for i in range(1, len(price_dates)):
                if price_dates[i-1] < search_start:
                    temp1 = search_start
                else:
                    temp1 = price_dates[i-1]
                if price_dates[i] > search_end:
                    temp2 = search_end
                else:
                    temp2 = price_dates[i-1]
                calc_days = temp2 - temp1
                curr_prices =
prices.filter(start_act = price_dates[i-1], end_act =
price_dates[i], days__lte = calc_days)
                guests_value = 1000
                for p in curr_prices:
                    if p.guests < guests_value:
                        guests_value = p.guests
                curr_prices =
curr_prices.filter(guests = guests_value)
                curr_prices =
curr_prices.order_by('-days')
                curr_price =
curr.prices.values()[0]
                if curr_price.mindays <= calc_days:

```

```

        price = price +
curr_price.price * calc_days
        else:
            possible = False
            if possible and price >= min_price and
price <= max_price:
                listing.price = price
                listings_result.append(listing)
            return listings_result

    def get_context_data(self, **kwargs):
        search_parameters = super(ListingSearch,
self).get_context_data(**kwargs)
        search_parameters ['parameters_form'] =
ListingsSearchFrm (initial={
            'search_start':
self.request.GET.get('search_start', ''),
            'search_end':
self.request.GET.get('search_end', ''),
            'min_price': self.request.GET.get('min_price',
'0'),
            'max_price': self.request.GET.get('max_price',
'-1'),
            'listing_types':
self.request.GET.get('listing_types', '0'),
            'min_rooms': self.request.GET.get('min_rooms',
'1'),
            'max_rooms': self.request.GET.get('max_rooms',
'1000'),
            'guests': self.request.GET.get('guests', '1'),
            'conditioner':
self.request.GET.get('conditioner', '0'),
            'heat': self.request.GET.get('heat', '0'),
            'washmachine':
self.request.GET.get('washmachine', '0'),
            'access': self.request.GET.get('access', '0'),
            'children': self.request.GET.get('children',
'0'),
            'pets': self.request.GET.get('pets', '0'),
            'smoking': self.request.GET.get('smoking',
'0'),
            'parties': self.request.GET.get('parties',
'0'),
            'city': self.request.GET.get('city', '-1'),
            'neighborhood':
self.request.GET.get('neighborhood', '-1'),

```

```

        'sleepingplaces':
self.request.GET.get('sleepingplaces', '1'),
        'toiletries':
self.request.GET.get('toiletries', '0'),
        'dining': self.request.GET.get('dining', '0'),
        'workplaces':
self.request.GET.get('workplaces', '0'),
        'style': self.request.GET.get('style', '-1'),
        'parkingplaces':
self.request.GET.get('parkingplaces', '0'),
        'payedparking':
self.request.GET.get('payedparking', '0'),
        'nightcheckin':
self.request.GET.get('nightcheckin', '0'),
        'min_storey':
self.request.GET.get('min_storey', '-1'),
        'max_storey':
self.request.GET.get('max_storey', '-1'),
        'not_last_storey':
self.request.GET.get('not_last_storey', '0'),
        'max_stories':
self.request.GET.get('max_stories', '-1'),
        'checkinfrom':
self.request.GET.get('checkinfrom', '12:00'),
        'checkinto': self.request.GET.get('checkinto',
'20:00'),
        'checkoutfrom':
self.request.GET.get('checkoutfrom', '06:00'),
        'checkoutto':
self.request.GET.get('checkoutfrom', '12:00'),
        'kitchen': self.request.GET.get('kitchen',
'0'),
        'dishes': self.request.GET.get('dishes', '0'),
        'bath': self.request.GET.get('bath', '0'),
        'dryer': self.request.GET.get('dryer', '0'),
        'gasdetector':
self.request.GET.get('gasdetector', '0'),
        'smokedetector':
self.request.GET.get('smokedetector', '0'),
        'petsinside':
self.request.GET.get('petsinside', '0'),
        'gym': self.request.GET.get('gym', '0'),
        'internet': self.request.GET.get('internet',
'0'),
        'restzone': self.request.GET.get('restzone',
'0'),

```

```

        'bikes': self.request.GET.get('bikes', '0'),
        'bedlinen': self.request.GET.get('bedlinen',
'0'),

        'lt7': self.request.GET.get('lt7', '0'),
        'lt6': self.request.GET.get('lt6', '0'),
        'lt5': self.request.GET.get('lt5', '0'),
        'lt4': self.request.GET.get('lt4', '0'),
        'lt3': self.request.GET.get('lt3', '0'),
        'lt2': self.request.GET.get('lt2', '0'),
        'lt1': self.request.GET.get('lt1', '0'),
        'min_position':
self.request.GET.get('min_position', '0'),
        'min_lastposition':
self.request.GET.get('min_lastposition', '0'),
        'safety': self.request.GET.get('safety', '0'),
        'accuracy': self.request.GET.get('accuracy',
'0'),

        'sleep': self.request.GET.get('sleep', '0'),
        'cleanness': self.request.GET.get('cleanness',
'0')

    })
    return search_parameters

```

```

class SimilarListings(ListView):
    model = Listing
    template_name = 'kodakapp/listings_similar.html'

    def get_queryset(self, *args, **kwargs):
        listings = super().get_queryset(*args, **kwargs)
        neighborhood =
int(self.request.GET.get('neighborhood', '-1'))
        style = int(self.request.GET.get('style', '-1'))
        list_type = int(self.request.GET.get('list_type',
'0'))

        position = float(self.request.GET.get('position',
'0'))

        min_position = position - 0.5
        max_position = position + 0.5
        listings = listings.filter(position__gte =
min_position, position__lte = max_position)
        if style:
            listings = listings.filter(style = style)
        if neighborhood:
            listings = listings.filter(neighborhood =
neighborhood)
        if list_type:

```

```

        listings = listings.filter(listingtype =
list_type)
        listings.order_by('-position')
        return listings.values()

    def get_context_data(self, **kwargs):
        similar_parameters = super(SimilarListings,
self).get_context_data(**kwargs)
        similar_parameters ['parameters_form'] =
ListingsSearchFrm (initial={
            'neighborhood':
self.request.GET.get('neighborhood', '-1'),
            'style': self.request.GET.get('style', '-1'),
            'list_type': self.request.GET.get('list_type',
'0'),
            'position': self.request.GET.get('position',
'0')
        })
        return similar_parameters

class ListingOpinions(ListView):
    model = ListingOpinion
    template_name = 'kodakapp/listing_opinions.html'

    def get_queryset(self, *args, **kwargs):
        opinions = super().get_queryset(*args, **kwargs)
        listing = int(self.request.GET.get('listing', '-1'))

        listing = Listing.objects.filter(id = listing,
owner = owner)
        if listing:
            opinions = opinions.filter(listing = listing)
            opinions = opinions.order_by('-created')
            return opinions
        else:
            return []

```

Б.3 Текст файла ownerviews.py

```

from django.contrib.auth.mixins import LoginRequiredMixin
from django.views.generic import ListView
from kodakapp.models import Owner, Listing, Price
from django.shortcuts import render, get_object_or_404
from kodakapp.forms import PriceFrm
from django.urls import reverse_lazy

```

```

import datetime
from datetime import date

class OwnerListings(LoginRequiredMixin, ListView):
    model = Listing
    template_name = 'kodakapp/owner_listings.html'

    def get_queryset(self, *args, **kwargs):
        user_acc = self.request.user
        owner = Owner.objects.get(user = user_acc)
        ls = Listing.objects.filter(owner = owner)
        listings = []
        moment = datetime.datetime.now()
        moment_day = date(moment.year, moment.month,
moment.day)
        for l in ls:
            prices = Price.objects.filter(listing = l,
end_act__gte = moment_day)
            if prices:
                listings.append(l)
        return listings

class OwnerOldListings(LoginRequiredMixin, ListView):
    model = Listing
    template_name = 'kodakapp/owner_old_listings.html'

    def get_queryset(self, *args, **kwargs):
        user_acc = self.request.user
        owner = Owner.objects.get(user = user_acc)
        ls = Listing.objects.filter(owner = owner)
        listings = []
        moment = datetime.datetime.now()
        moment_day = date(moment.year, moment.month,
moment.day)
        for l in ls:
            prices = Price.objects.filter(listing = l,
end_act__gte = moment_day)
            if not prices:
                listings.append(l)
        return listings

class OwnerRentals(LoginRequiredMixin, ListView):
    model = Rental
    template_name = 'kodakapp/owner_rentals.html'

    def get_queryset(self, *args, **kwargs):

```

```

        rentals = super().get_queryset(*args, **kwargs)
        user_acc = self.request.user
        owner = Owner.objects.get(user = user_acc)
        listings = Listing.objects.filter(owner = owner)
        if listings:
            rentals = rentals.filter(listing__in =
listings, checkin_date__gte = moment_day)
            rentals = rentals.order_by('checkin_date')
            return rentals
        else:
            return []

```

```

class OwnerListingRentals(LoginRequiredMixin, ListView):
    model = Rental
    template_name = 'kodakapp/owner_listing_rentals.html'

    def get_queryset(self, *args, **kwargs):
        rentals = super().get_queryset(*args, **kwargs)
        user_acc = self.request.user
        owner = Owner.objects.get(user = user_acc)
        listing = int(self.request.GET.get('listing', '-
1'))

        listing = Listing.objects.filter(id = listing,
owner = owner)
        if listing:
            rentals = rentals.filter(listing = listing,
checkin_date__gte = moment_day)
            rentals = rentals.order_by('checkin_date')
            return rentals
        else:
            return []

```

```

class OwnerListingPrices(LoginRequiredMixin, ListView):
    model = Price
    template_name = 'kodakapp/owner_listing_prices.html'

    def get_queryset(self, *args, **kwargs):
        prices = super().get_queryset(*args, **kwargs)
        user_acc = self.request.user
        owner = Owner.objects.get(user = user_acc)
        listing = int(self.request.GET.get('listing', '-
1'))

        listing = Listing.objects.filter(id = listing,
owner = owner)
        if listing:

```

```

        prices = prices.filter(listing = listing,
end_act__gte = moment_day)
        prices = prices.order_by('start_act')
        return prices
    else:
        return []

class OwnerListingOldPrices(LoginRequiredMixin, ListView):
    model = Price
    template_name = 'kodakapp/owner_listing_prices.html'

    def get_queryset(self, *args, **kwargs):
        prices = super().get_queryset(*args, **kwargs)
        user_acc = self.request.user
        owner = Owner.objects.get(user = user_acc)
        listing = int(self.request.GET.get('listing', '-
1'))

        listing = Listing.objects.filter(id = listing,
owner = owner)
        if listing:
            prices = prices.filter(listing = listing,
end_act__le = moment_day)
            prices = prices.order_by('-start_act')
            return prices
        else:
            return []

class PriceAdd(LoginRequiredMixin, View):
    template_name = 'kodakapp/add_price.html'

    def get(self, request, pk = None):
        pricefrm = PriceFrm()
        frm_context = {'new_form': pricefrm}
        return render(request, self.template_name,
frm_context)

    def post(self, request, pk = None):
        pricefrm = PriceFrm (request.POST)
        if not pricefrm.is_valid():
            frm_context = {'new_form': pricefrm}
            return render(request, self.template_name,
frm_context)

        price = pricefrm.save(commit = False)
        price.save()
        return
        redirect(reverse_lazy('kodakapp:listingprices'))

```

```

class ListingEdit(LoginRequiredMixin, View):
    template_name = 'kodakapp/edit_listing.html'

    def get(self, request, pk):
        apartment = get_object_or_404(Listing, id = pk)
        pricefrm = PriceFrm (instance = apartment)
        frm_context = {'new_form': listingfrm}
        return render(request, self.template_name,
frm_context)

    def post(self, request, pk = None):
        apartment = get_object_or_404(Listing, id = pk)
        pricefrm = PriceFrm (request.POST, None, instance =
apartment)

        if not pricefrm.is_valid():
            frm_context = {'new_form': pricefrm }
            return render(request, self.template_name,
frm_context)

        listingfrm = listingfrm.save(commit = False)
        listingfrm.save()
        return reverse_lazy('kodakapp:ownerlistings')

class OwnerListingOpinionStatistics(ListView):
    model = OpinionStatistic
    template_name =
'kodakapp/listing_opinionstatistics.html'

    def get_queryset(self, *args, **kwargs):
        opinions = super().get_queryset(*args, **kwargs)
        user = self.request.user
        owner = Owner.objects.get(user = user)
        listing = int(self.request.GET.get('listing', '-
1'))

        listing = Listing.objects.filter(id = listing,
owner = owner)
        if listing:
            opinions = opinions.filter(listing = listing)
            opinions = opinions.order_by('-month')
            return opinions
        else:
            return []

```

ДОДАТОК В
Слайди презентації

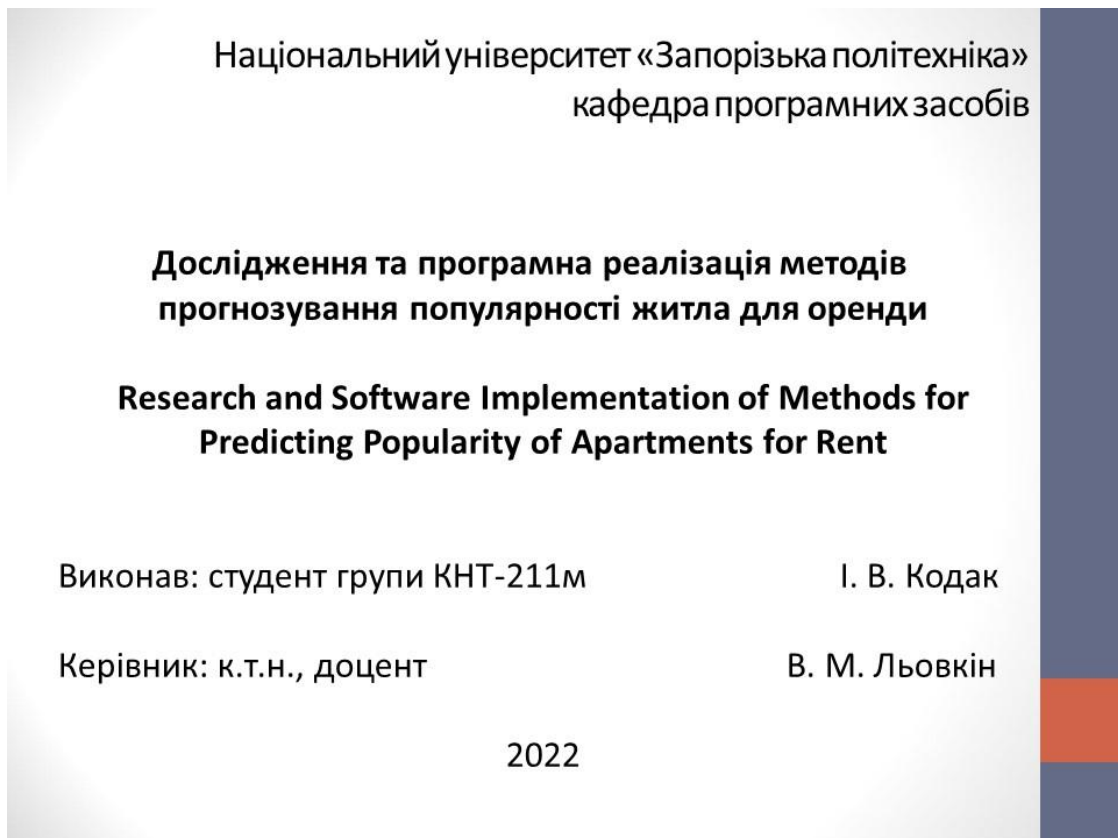


Рисунок В.1 – Титульний слайд

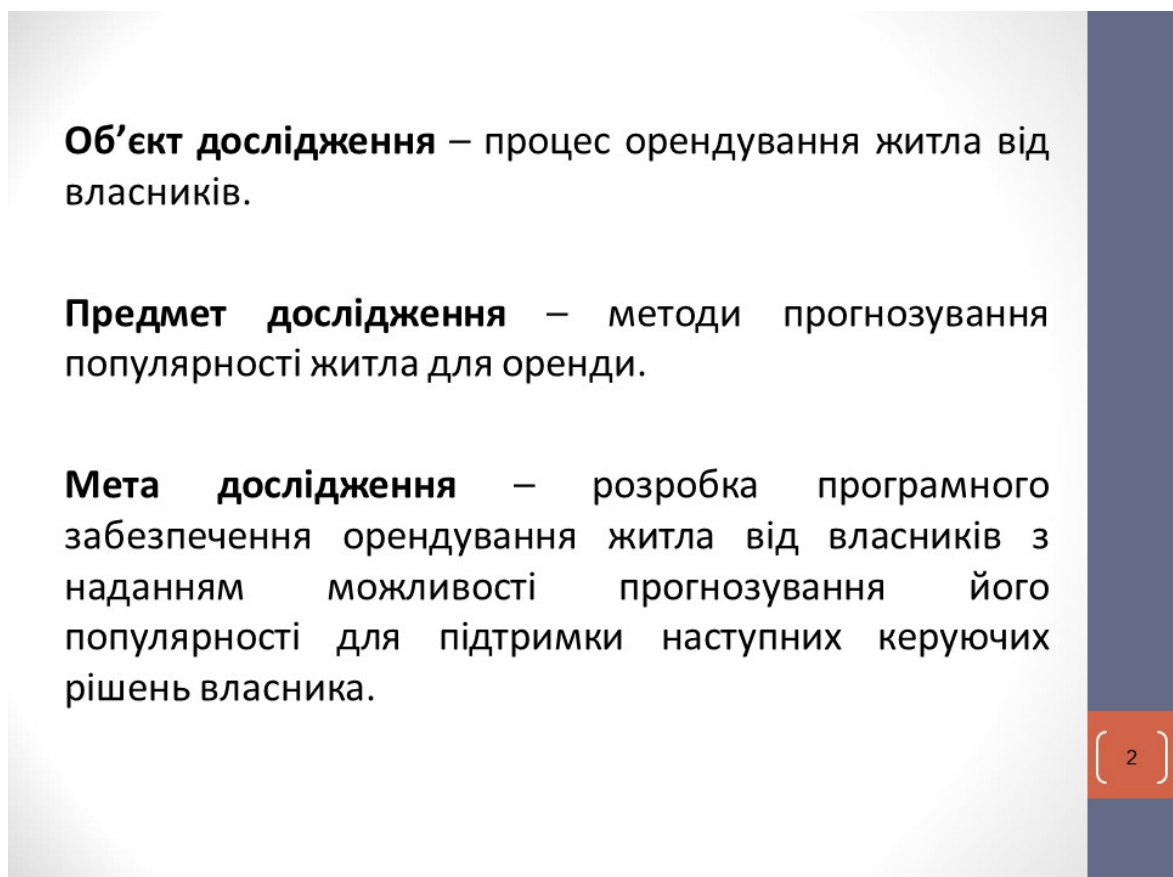


Рисунок В.2 – Об'єкт, предмет, мета дослідження

Завдання роботи

- проаналізувати програмні засоби орендування житла від власників;
- виконати дослідження методів прогнозування популярності житла для оренди;
- виконати проектування програмного забезпечення орендування житла від власників;
- реалізувати програмне забезпечення орендування житла від власників.

(3)

Рисунок В.3 – Завдання роботи

Порівняння програмних аналогів

Характеристика	Airbnb	Plum Guide	Дипломна розробка
Принцип формування пропозицій	Вільно власниками	Жорсткий відбір командою	Вільно власниками
Тип житла	Повне житло, частина	Повне житло	Повне житло, частина
Пошук житла	Так	Так	Так
Керування проживаннями	Так	Так	Так
Оцінювання орендарів	Так	Ні	Так
Оцінювання житла орендарями	Так	Ні	Так
Прогнозування популярності житла	Ні	Ні	Так

(4)

Рисунок В.4 – Порівняння програмних аналогів

Метод прогнозування популярності житла для оренди

На першому етапі роботи методу створюється вибірка даних, яка в подальшому використовується для навчання. Для цього реалізується процедура злиття і передобробки даних.

На другому етапі відбувається створення і навчання моделі на основі алгоритму Random Forest та сформованої вибірки даних.

На третьому етапі реалізується збір даних про житло, для якого потрібно виконати прогнозування. Залишається тільки активне протягом щонайменше півроку житло.

Для кожного житла визначається середня вартість доби проживання на майбутній місяць. Після цього витягається інформація про кількість оцінювань за попередні місяці за півроку. Для цього періоду також обчислюється середня вартість доби проживання. Встановлюються як постійні значення значення району, типу житла.

На четвертому етапі виконується прогнозування для заданих значень, що характеризують житло на попередньому етапі, кількості оцінювань на наступний місяць для кожного варіанта житла.

(5)

Рисунок В.5 – Метод прогнозування популярності житла для оренди

Порівняння мов для реалізації програмного забезпечення орендування житла від власників

Критерій	Мова Ruby	Мова Python
Складність навчання	Вища разом з фреймворком	Нижча
Простота відлагодження	Нижча	Висока (сприяє читабельності)
Зручність повторного використання коду на основі бібліотек	Має меншу кількість бібліотек	Відомий великою кількістю бібліотек
Основний фреймворк для створення вебдодатків	Ruby on Rails	Django
Зручність, налагодженість реалізації машинного навчання	Невисока	Найвища

(6)

Рисунок В.6 – Порівняння мов для реалізації програмного забезпечення орендування житла від власників

Діаграми варіантів використання програми

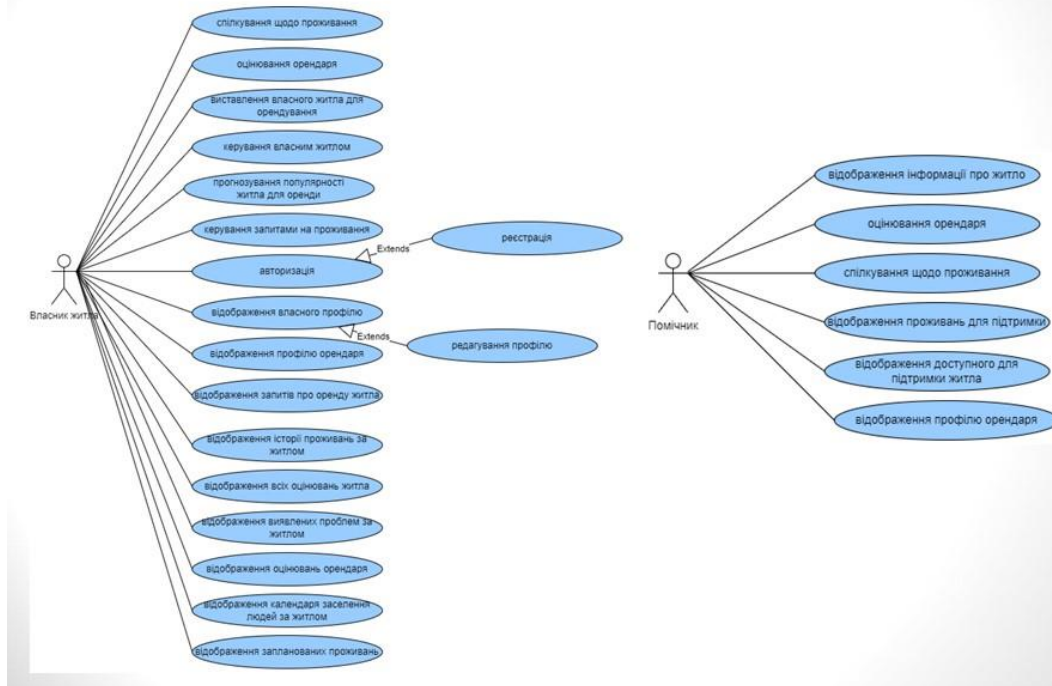


Рисунок В.7 – Діаграми варіантів використання програми

Діаграми варіантів використання програми

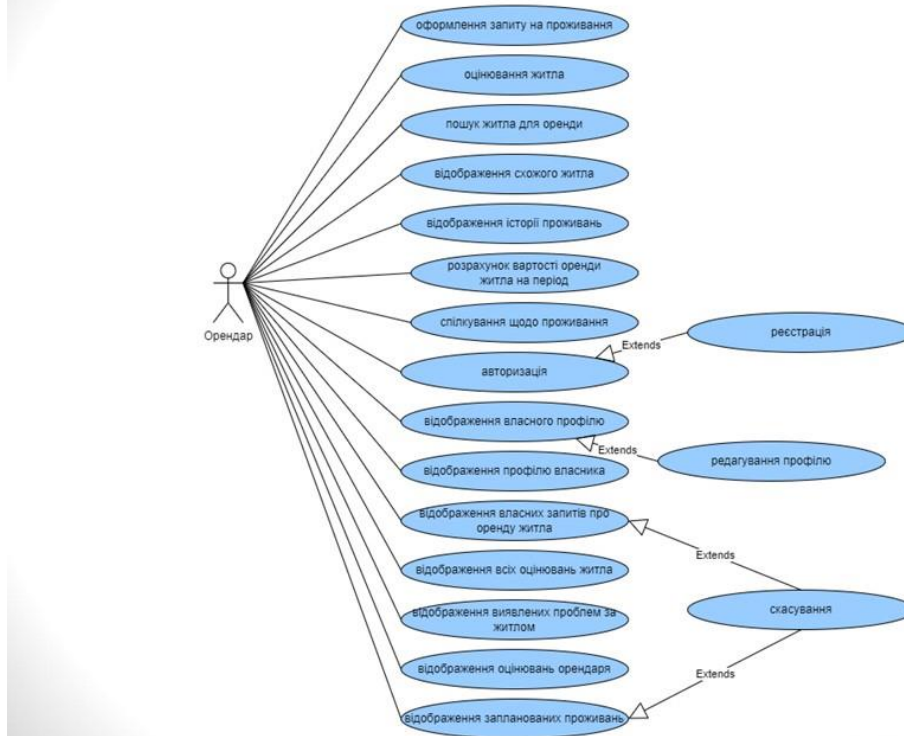


Рисунок В.8 – Діаграми варіантів використання програми

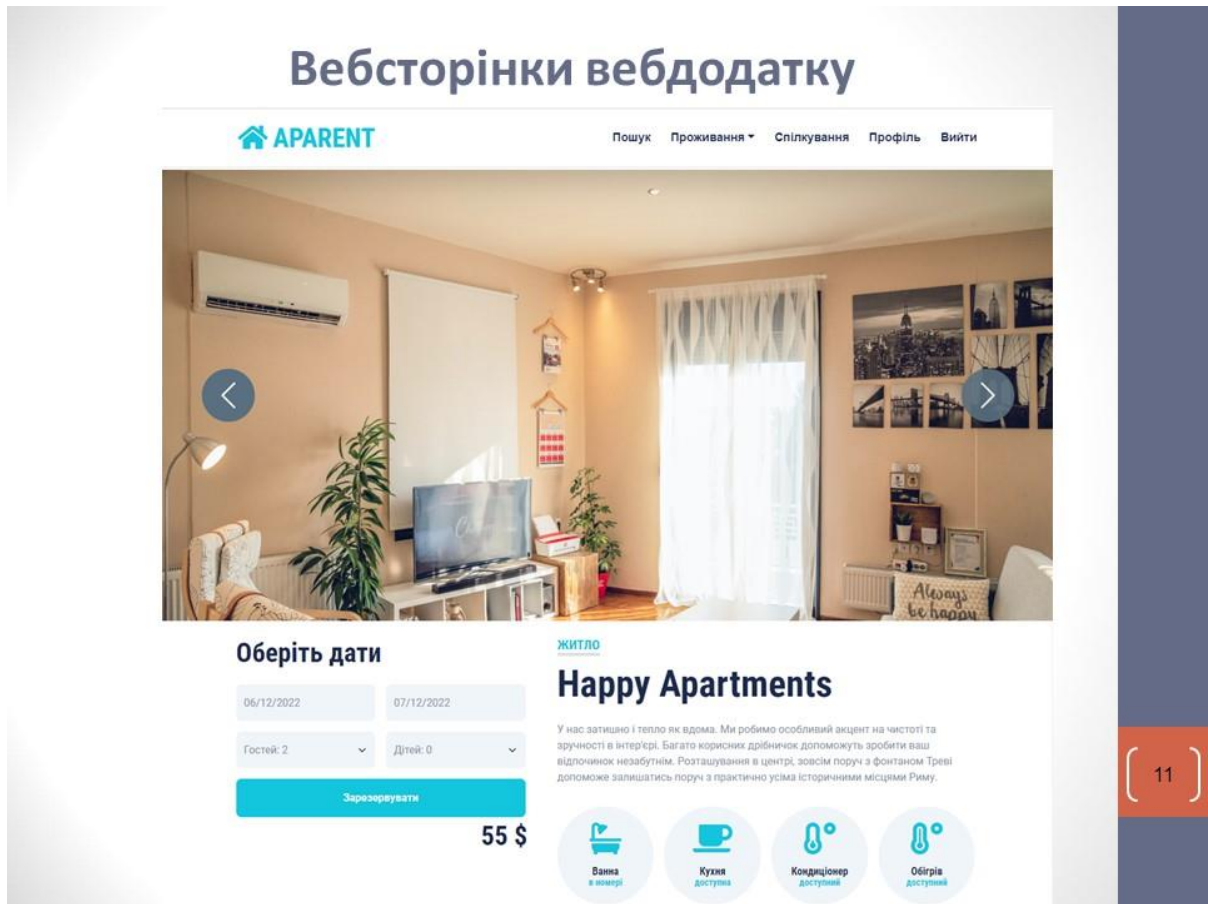


Рисунок В.11 – Вебсторінки вебдодатку

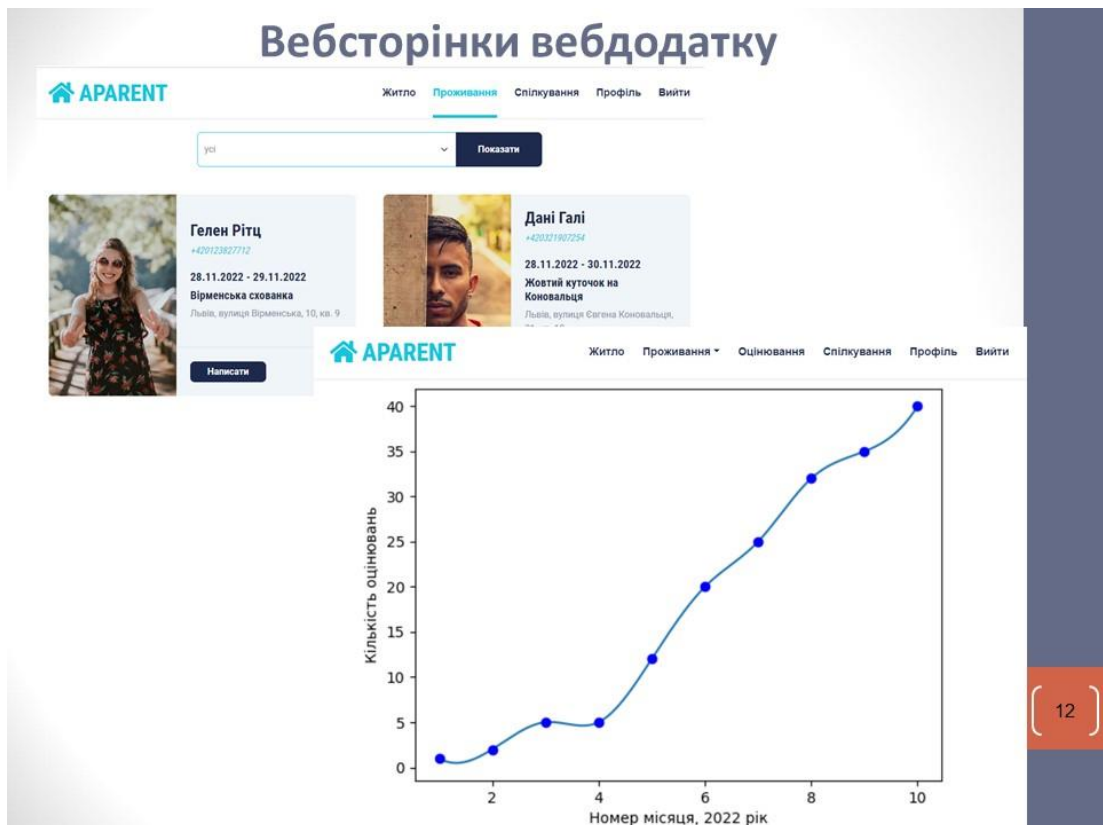
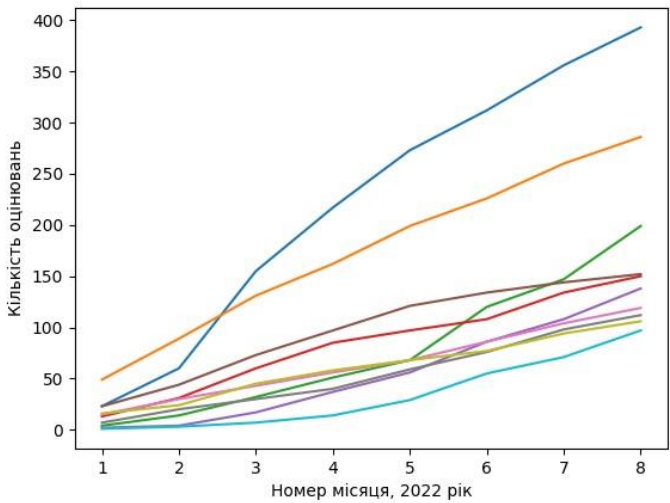


Рисунок В.12 – Вебсторінки вебдодатку

Вибірка даних



(13)

Рисунок В.13 – Вибірка даних

Експериментальне дослідження

Модель або метод	R^2
Метод опорних векторів (регресійна модель)	0,71
Лінійна регресія	0,59
Метод прогнозування популярності житла для оренди	0,84

(14)

Рисунок В.14 – Експериментальне дослідження

Висновки

Науковою новизною роботи є метод прогнозування популярності житла для оренди, який дозволяє прогнозувати кількість оцінювань житла за наступний місяць на основі характеристик житла і даних минулих місяців, використовуючи алгоритм Random Forest, що дозволяє завчасно отримувати інформацію, яку можна використати для прийняття рішень про зміну параметрів оголошення про оренду житла.

Практична цінність роботи полягає в розробленому програмному забезпеченні, яке може використовуватись у вигляді координаційного центру для власників житла, їх помічників та орендарів. Помічникам надається інформація, необхідна для розуміння розкладу заселення в житло, де помічник підтримує ці процеси. Власникам житла надається можливість традиційно виставляти житло в оренду, керувати бронюваннями, але також отримувати візуальне представлення інформації про популярність власного житла, включаючи прогнозування такої популярності на наступний місяць. Орендар же має коло засобів для пошуку, бронювання житла, отримання інформації про проживання, які вже завершилися, які обробляються та які можуть розпочнутись в певний момент у майбутньому.

Рисунок В.15 – Висновки