

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проекту (роботи)

магістра

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ

(назва теми)

ПРОГНОЗУВАННЯ ВАРТОСТІ ОРЕНДИ КІМНАТИ ДЛЯ
ПРОЖИВАННЯ. RESEARCH AND SOFTWARE IMPLEMENTATION OF
METHODS FOR FORECASTING THE ROOM RENTAL PRICE

Виконав(ла): студент(ка) 2 курсу, групи КНТ-222м

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Системи штучного інтелекту

НУРІЄВ В.В.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ГОЛУБ Т.В.

(ПРИЗВИЩЕ та ініціали)

2023_____

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

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.
Сергій СУББОТІН
« _____ » _____ 2023 року

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

НУРІЄВА Віталія Валентиновича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів прогнозування вартості оренди кімнати для проживання. Research and Software Implementation of Methods for Forecasting the Room Rental Price

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

затверджені наказом закладу вищої освіти від « 14 » листопада 2023 року № 437

2. Строк подання студентом проєкту (роботи) 01 грудня 2023 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 програмного забезпечення. 5. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількості слайдів, плакатів)
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		
Нормоконтроль	КАМІНСЬКА Ж.К., асистент		

7. Дата видачі завдання « 04 » вересня 2023 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Срок виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз предметної області	2 тиждень	Розділ 1
3	Дослідження проблеми	3-5 тижні	Розділ 2
4	Розробка архітектури програми	6 тиждень	Розділ 3
5	Розробка програми	7-8 тижні	Розділ 4
6	Тестування та експериментальне дослідження програми	9-10 тижні	Розділ 5
7	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	11 тиждень	Додатки
8	Захист роботи	12 тиждень	

Студент(ка)

(підпис)

Віталій НУРІЄВ

(Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Валерій ЛЬОВКІН

(Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
109 с., 27 рис., 6 табл., 3 дод., 13 джерел.

ОРЕНДА, КІМНАТА, ОРЕНДАР, ОРЕНДОДАВЕЦЬ, ЗАДАЧА
ПРОГНОЗУВАННЯ, МАШИННЕ НАВЧАННЯ.

Об'єкт роботи – процес оренди кімнати для проживання. Предмет роботи – методи прогнозування вартості оренди кімнати для проживання. Мета роботи – розробити програмне забезпечення оренди кімнат для проживання з задоволенням вимог орендодавців до отримання індивідуальної інформації про орендарів.

Матеріали, методи та технічні засоби: засоби веброзробки, включаючи мови Python, HTML, CSS, фреймворк Django, засоби машинного навчання на основі Python-пакетів, інструмент для роботи з базою даних MySQL.

Результати. Програма в підсумку забезпечує необхідні засоби для пошуку орендарями кімнат для оренди, а орендодавцями орендарів, враховуючи зокрема індивідуальні характеристики, щоб забезпечити створення комунікації щодо подальшої оренди, відбір кандидатів для оренди власної кімнати, надаючи при цьому можливість прогнозувати вартість оренди на основі індивідуальних характеристик орендаря.

Висновки. Наукова новизна роботи полягає в створеному методі прогнозування вартості оренди кімнати для проживання, який базується на застосуванні ансамблів дерев рішень на основі принципів бустингу і який дозволяє врахувати індивідуальні характеристики орендаря, додаючи їх до ознак створеної моделі прогнозування, що визначають загальні характеристики житла і оренди.

Галузь використання. Оренда кімнат у квартирах або будинках.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 109 pages, 27 figures, 6 tables, 3 appendixes, 13 sources.

RENT, ROOM, TENANT, LANDLORD, FORECASTING PROBLEM, MACHINE LEARNING.

The object of the work is the process of renting a room for living. The subject of the work is methods of forecasting the cost of renting a room for living. The purpose of the work is to develop software for renting rooms for living in order to meet the requirements of landlords for obtaining individual information about tenants.

Materials, methods and technical means: web development tools, including Python, HTML, CSS languages, Django framework, machine learning tools based on Python packages, MySQL database tool.

The results. The program ultimately provides the necessary tools for tenants to search for rooms for rent, and for landlords to search for tenants, taking into account individual characteristics in particular, to ensure the creation of communication for further rent, the selection of candidates for renting their own room, while providing the opportunity to forecast the cost of rent based on the individual characteristics of the tenant.

Conclusions. The scientific novelty of the work consists in the created method of forecasting the cost of renting a room for living, which is based on the application of ensembles of decision trees based on the principles of boosting and which allows taking into account the individual characteristics of the tenant, adding them to the features of the created forecasting model, which determine the general characteristics of the housing and rent.

Field of use. Renting rooms in apartments or houses.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань дослідження	11
1.1 Огляд програмного забезпечення для оренди кімнат для проживання.	11
1.2 Порівняння аналогів	15
1.3 Висновки за розділом 1	16
2 Матеріали і методи.....	18
2.1 Задача прогнозування вартості оренди кімнати для проживання	18
2.2 Метод прогнозування вартості оренди кімнати для проживання.....	19
2.3 Висновки за розділом 2	22
3 Проєктування програмного забезпечення	24
3.1 Вибір рішень для веброзробки.....	24
3.2 Моделювання програмного забезпечення на основі функціональних вимог	25
3.3 Проєктування структури бази даних.....	29
3.4 Висновки за розділом 3	34
4 Основні рішення щодо реалізації програмного забезпечення	35
4.1 Порядок роботи програмного забезпечення	35
4.2 Рішення з реалізації вебзастосунку	36
4.3 Висновки за розділом 4	58
5 Експлуатація, тестування та експериментальне дослідження програми	59
5.1 Опис програми.....	59
5.2 Експлуатація програми.....	59
5.3 Експериментальне дослідження	64
5.4 Висновки за розділом 5	70
Висновки	71
Перелік джерел посилання	72
Додаток А Технічне завдання	74

Додаток Б Текст програми	80
Додаток В Слайди презентації.....	102

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

MVT – Model-View-Template;

XGBoost – eXtreme Gradient Boosting;

БД – база даних.

ВСТУП

Оренда кімнат для проживання є дуже особливою галуззю всередині широковідомої оренди житла. Це визначається тим, що власник житла зазвичай проживає разом з орендарем в одному житлі, а тому має певний ряд вимог, які мають бути забезпечені орендарем, щоб орендодавець довірив йому право орендувати кімнату. Велика частина з цих вимог є логічною і обумовленою тим, що має бути забезпечений необхідний рівень безпеки, довіри, має бути встановлена певна комунікація, а це потребує можливо навіть наявності спільних захоплень, поглядів. Тому програма, яка буде підтримувати ці процеси, вона має надати можливість отримувати необхідну для цього інформацію.

Такі особливості комунікації визначають також, що і вартість оренди залежить від фактично індивідуальних характеристик орендаря. Тому розуміння вартості оренди за тих індивідуальних характеристик, які має орендар, є важливим, адже у такому випадку далеко не кожен користувач зможе орендувати кожен доступну і підходящу для нього кімнату. Тобто врахування індивідуальних характеристик орендаря важливо як під час визначення комунікації в програмі, так і під час прогнозування вартості оренди, щоб підібрати потім варіант за адекватну ціну.

З іншого боку для власника житла прогнозування вартості оренди його кімнати за встановлення різних індивідуальних обмежень для орендарів дозволяє зрозуміти, як він може вплинути на ту суму, яку отримає за оренду в підсумку.

Мету роботи було визначено як розробку програмного забезпечення оренди кімнат для проживання з задоволенням вимог орендодавців до отримання індивідуальної інформації про орендарів, передбачаючи для цього виконання завдань:

- проаналізувати аналоги програми оренди кімнат для проживання;

- розробити метод прогнозування вартості оренди кімнати для проживання;
- виконати проєктування програмного забезпечення оренди кімнат для проживання;
- реалізувати програмне забезпечення оренди кімнат для проживання.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Огляд програмного забезпечення для оренди кімнат для проживання

Установлення особливостей оренди кімнат для проживання порівняно з орендою цілих апартаментів визначило подальший аналіз предметної області через необхідність перевірки наявності необхідних інструментів для роботи саме з кімнатами у складі будинків у існуючих програмах.

Першим розглянуто основний аналог виконуваної розробки – програму SpareRoom (рис. 1.1) [1].

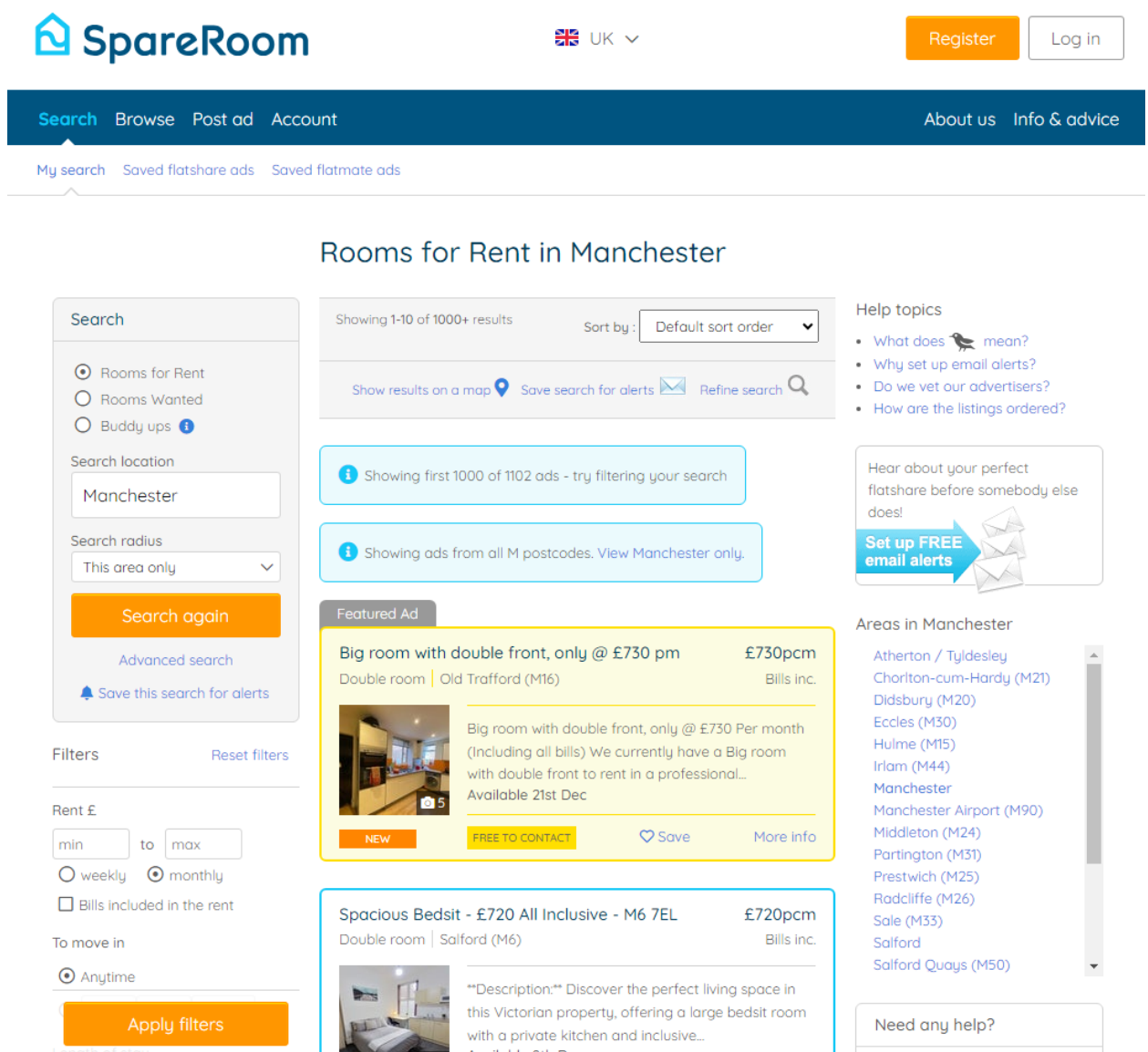


Рисунок 1.1 – Сторінка пошуку кімнат для оренди програми SpareRoom

Пошук у SpareRoom виконується окремо за кімнатами для оренди, оголошеннями самих орендарів, які шукають кімнати, та за орендарями, які шукають для себе співорендарів. Тобто і орендодавці можуть шукати орендарів, і орендарі орендодавців, при цьому всі оголошення стосуються саме кімнат або загалом певного варіанту сумісного проживання. Пошук включає велику кількість фільтрів, а розширений варіант стандартно вміщується в декілька екранів. Однак індивідуальні захоплення тим не менш при пошуку не враховуються.

Сторінка кімнати (рис. 1.2), що надається в оренду, включає дані всієї квартири або будинку, параметри кімнати, вимоги до орендаря.

The screenshot shows the SpareRoom website interface. At the top, there's a navigation bar with 'Search', 'Browse', 'Post ad', and 'Account'. Below this, a search bar shows 'Luxe City centre room 2 flatmates'. The main content area displays a listing for 'Flat share Manchester City Centre M1'. The listing includes a large photo of the flat, a price of '£625 pcm (double)', and a 'View on a map' button. To the right of the photo, there's a 'Contact the advertiser' section with a profile picture of 'Tara' and a 'Message' button. Below the photo, there's a table with details about the flatshare, including availability, extra costs, and amenities.

Availability	
Available	01 Feb 2024
Minimum term	12 months
Maximum term	None

Extra cost	
Deposit	£629.00
Bills included?	Some

Amenities	
Furnishings	Furnished
Disabled access	
Living room	shared
Broadband included	Yes

Gorgeous fully furnished 3 bed flat next to Aldi and Puregym, 5 mins from Piccadilly. Flat includes 24 hour security and

Рисунок 1.2 – Сторінка кімнати, що надається в оренду, в SpareRoom

Будь-який користувач може розмістити оголошення, яке стосується надання кімнати для оренди, цілого житла для розділення між співмешканцями, визначення оголошення про власний пошук кімнати.

Реагування на оголошення відбувається через відправлення повідомлення орендарем орендодавцю або орендодавцем орендарю, в залежності від того, яке саме оголошення демонструється.

Другий розглянутим вебзастосунком є iBilik (рис. 1.3) [2]. Це сервіс, який працює в Малайзії, надаючи можливість виконувати оренду кімнат та окремо орендувати житло для проживання на короткий строк.

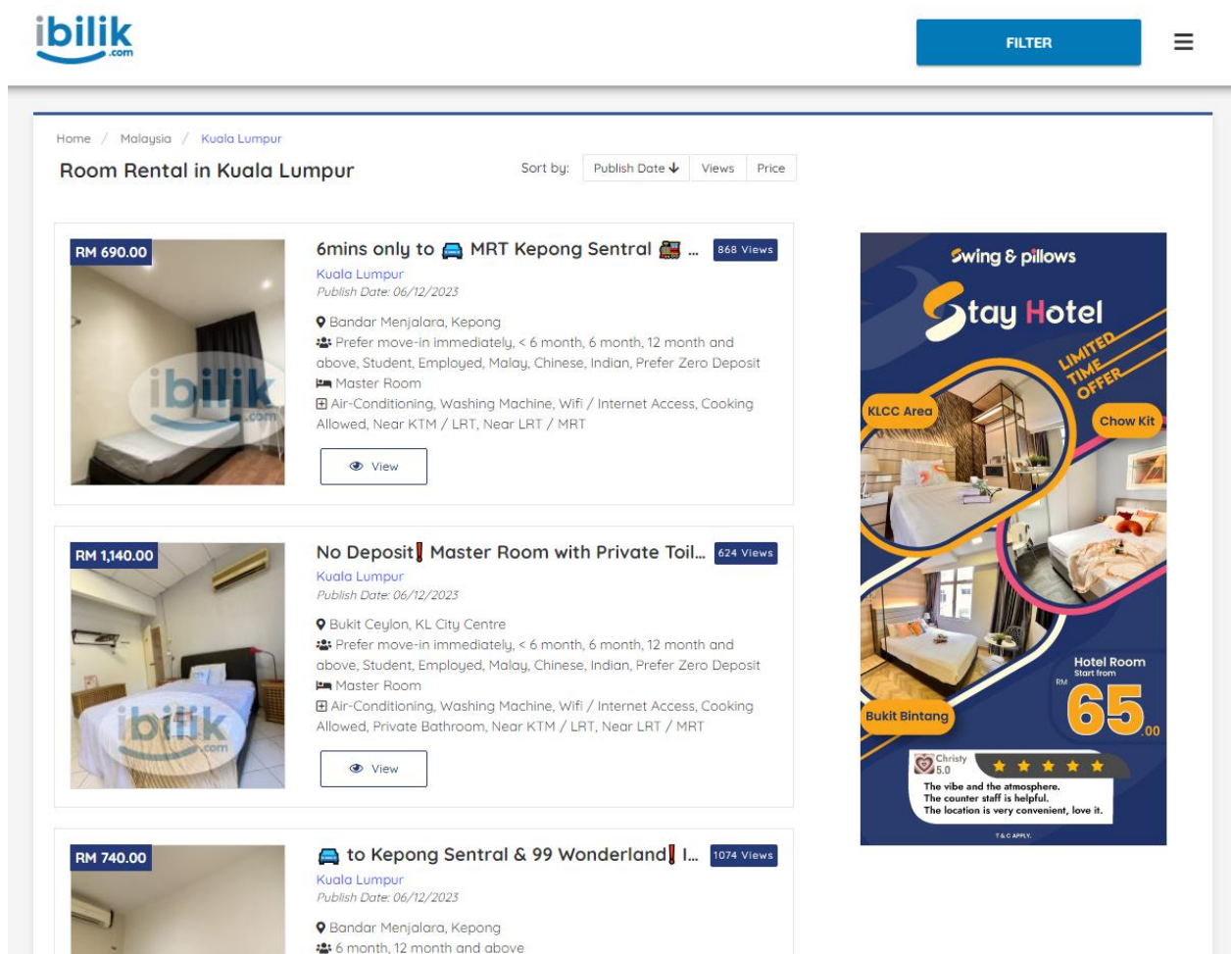


Рисунок 1.3 – Сторінка пошуку кімнат для оренди програми iBilik

За господарями надаються його контактні дані, включаючи номер телефону, тобто передбачаються різні варіанти подальшого контактування, а оплата оренди через вебсайт не передбачається. При пошуку можна

використати багато фільтрів, зокрема фільтри для визначення типу кімнати, вартості, індивідуальних характеристик (стать, національність, раса, професійна діяльність, строк оренди тощо), які однак не включають індивідуальні захоплення.

Одним з найбільших і найвідоміших вебсайтів для пошуку житла в світі є Airbnb (рис. 1.4) [3].

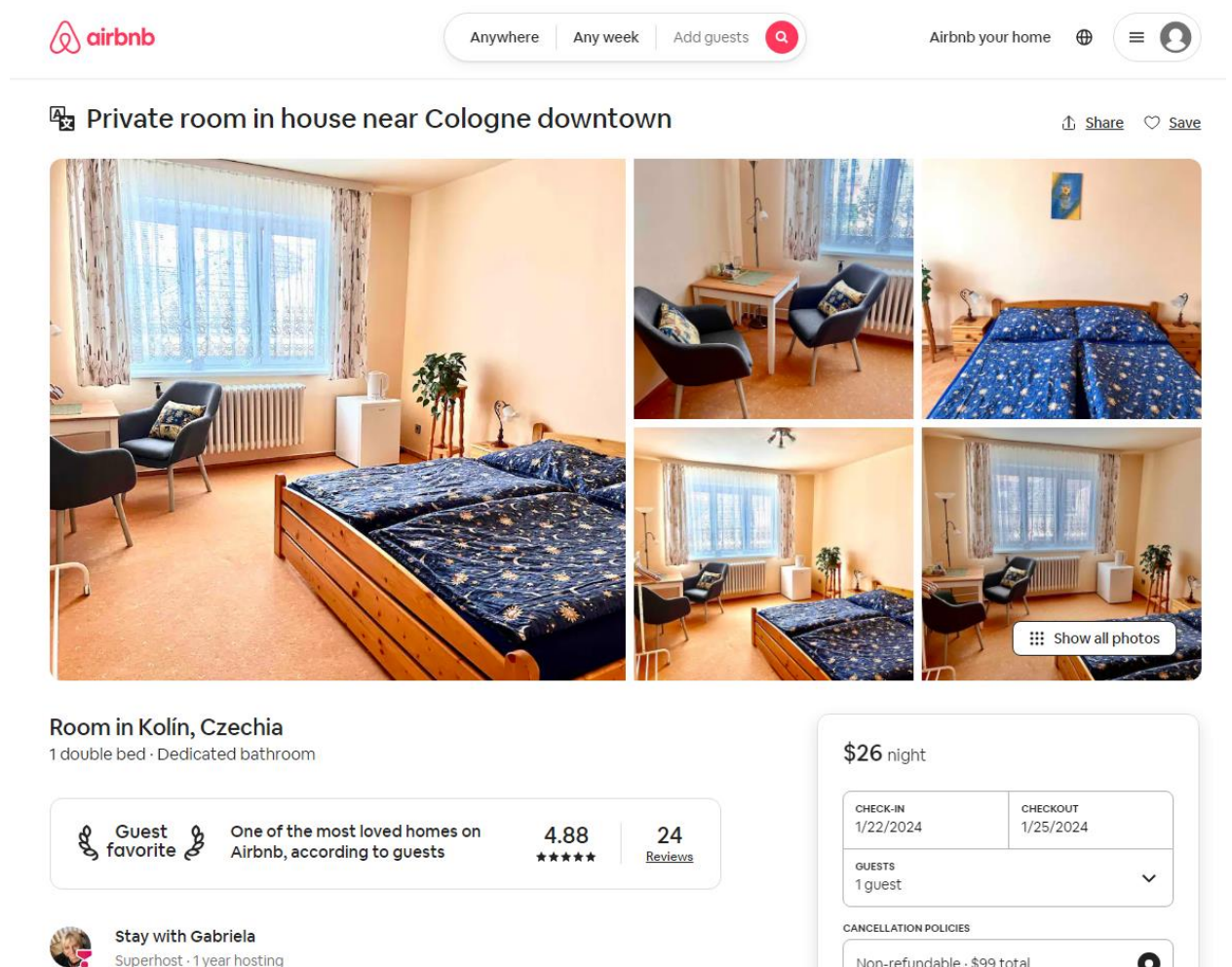


Рисунок 1.4 – Сторінка кімнати, що надається в оренду, в Airbnb

Airbnb направлений на оренду на короткий строк, але дозволяє серед інших типів житла виконувати пошук і оренду кімнат. Відмінністю даного вебзастосунку є те, що він дозволяє виконувати оренду через програму. Проте для оренди кімнат не надається окремих інструментів. Тобто не можна визначити окремі вимоги щодо орендарів, не можна і виконати пошук за

такими, тобто індивідуальними характеристиками або захопленнями. Немає також і можливості створити оголошення людям, які шукають кімнати, як і у попереднього рішення.

1.2 Порівняння аналогів

Зважаючи на виконаний аналіз, було реалізовано порівняння програм для оренди кімнат для проживання, включаючи до нього і програму RoomPersonaler, яка розробляється в роботі (табл. 1.1).

Основними відмінностями програми RoomPersonaler, яка розробляється, є:

- визначення індивідуальних захоплень і пошук за ними, що дозволяє більше дізнатися одразу про орендаря, не витрачаючи час на подальше персональне спілкування та знаходячи спільні риси, щоб бути більш задоволеним у підсумку від співмешканців у своєму житті;
- порівняння орендарів, що дозволяє приймати остаточне рішення про вибір заявки на оренду кімнати у програмі;
- прогнозування вартості оренди, що окрім всього має також враховувати персональні характеристики, щоб чіткіше враховувати контекст, властивий для оренди кімнат;
- створення оголошень про пошук кімнат, що присутнє також і в SpareRoom і дозволяє одразу привернути орендарю увагу до себе тих орендодавців, які не хочуть витрачати додатковий час на роботу, привертати увагу до своєї власності в програмі.

Airbnb на відміну від всіх інших аналогів дозволяє виконувати сплату за замовлення оренди, але це швидше є особливістю, продиктованою тим, що програма орієнтована в першу чергу на оренду на короткий строк, адже при довгостроковій оренді зазвичай надаються додаткові дані про орендаря, укладається договір, а до того ж передбачається подальша постійна плата, тобто участь програми вже не потрібна.

Таблиця 1.1 – Порівняння програмного забезпечення для оренди кімнат для проживання

Критерій	SpareRoom	iBilik	Airbnb	RoomPersonaler
Оренда кімнат	Так	Так	Так	Так
Строк оренди кімнат	Від місяця	Короткий і від місяця	Короткий, можливо більший	Від місяця
Створення оголошень про пошук кімнат	Так	Ні	Ні	Так
Сплата за оренду кімнати	Ні	Ні	Так	Ні
Фільтри за індивідуальними характеристиками орендаря	Так	Так	Ні	Так
Визначення індивідуальних захоплень і пошук за ними	Ні	Ні	Ні	Так
Порівняння орендарів	Ні	Ні	Ні	Так
Прогнозування вартості оренди	Ні	Ні	Ні	Так

1.3 Висновки за розділом 1

Проблему оренди кімнат для проживання розглянуто: виявлено особливості оренди кімнат серед інших варіантів проживання, виділено програми, які враховують особливості оренди кімнат і надають такі

можливості, до яких віднесено SpareRoom і iBilik, порівняно ці аналоги з запланованими для реалізації ідеями і можливостями програми RoomPersonaler, що в даній роботі і розробляється, що дозволило виконати перше завдання, визначену на роботу.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Задача прогнозування вартості оренди кімнати для проживання

Задача прогнозування вартості оренди кімнати для проживання полягає в тому, щоб для кімнати, для якої створено оголошення про оренду, виконати прогнозування вартості оренди цієї кімнати за 1 місяць. Кожна кімната при цьому характеризується різними ознаками, частина з яких може відноситися до самої кімнати, частина до помешкання, в якому знаходиться ця кімната, та частина до індивідуальних характеристик орендарів, які визначаються оголошенням.

Дана задача прогнозування вартості оренди кімнати для проживання належить до задач, відомих як задачі регресії. У такій задачі є відомі дані, які встановлюють залежність між вихідною ознакою, якою в даному випадку є вартість оренди кімнати за 1 місяць, та вхідними ознаками, які мають бути встановлені під час розв'язання задачі. На основі цих відомих даних потрібно створити вибірку, а на основі вибірки виявити залежність між вартістю оренди кімнати за 1 місяць та значеннями характеристик помешкання та орендаря. Виявлена залежність як раз і є розв'язанням задачі прогнозування вартості оренди кімнати, яка розглядається в цій роботі.

Оскільки за своєю суттю задача прогнозування вартості оренди кімнати для проживання є задачею регресії, то її розв'язання може відбуватися тими інструментами, які застосовуються для розв'язання такого класу задач:

- дерева рішень, але не у варіанті придатному для класифікації, а у варіанті для виконання регресії;
- ансамблі дерев рішень, які створюються на основі методу eXtreme Gradient Boosting (XGBoost);
- лінійна регресія, яка є базовим і по суті найпростішим варіантом побудови моделей регресії.

XGBoost створює ансамбль дерев рішень, призначених для виконання регресії, який складається з певної (заданої) кількості адитивних функцій, при цьому задається функція втрат, яка вимірює різницю між справжньою міткою, тобто значення, яке необхідно спрогнозувати, та прогнозом ансамблю, відповідно XGBoost прагне знайти такий ансамбль, який мінімізує втрати на навчальному наборі даних, а також не є надто складним, з цією метою метод визначає регуляризовану функцію витрат як цільову функцію моделі (складається з двох частин: загальна втрата моделі на навчальному наборі даних та штраф за складність базових дерев) [4]-[5].

Оскільки цільова функція, яка використовується під час роботи методу XGBoost, включає інші функції як параметри (функції дерева регресії), її не можна оптимізувати за допомогою традиційних методів оптимізації, таких як градієнтний спуск, тому модель навчається жадібним поетапним способом, додаючи одне нове дерево за раз, а оскільки знайти найкраще дерево рішень для довільної функції втрат обчислювально неможливо, оскільки це вимагатиме перерахувати всі можливі дерева та вибрати найкраще, то замість цього використовується підхід ітераційної оптимізації: на кожній ітерації бустингу обирається дерево, яке на крок наближає загальний результат до мінімальних витрат, XGBoost використовує при цьому ідею, подібну до методу Ньютона-Рафсона, щоб працювати з використанням наближення другого порядку (цей метод зазвичай швидше збігається до мінімуму, коли друга похідна цільової функції відома і її легко обчислити, а в XGBoost зазвичай використовуються такі функції втрат, як наприклад, квадратична функція) [5].

2.2 Метод прогнозування вартості оренди кімнати для проживання

Метод прогнозування вартості оренди кімнати для проживання дозволяє розв'язати відповідну задачу, описану в підрозділі 2.1, враховуючи

індивідуальні характеристики орендаря разом з характеристиками житла і оренди. Тому для створення відповідної моделі має бути початково сформовано вибірку даних, яка визначає характеристики оголошень щодо оренди саме окремих кімнат, а не цілісних будинків, квартир або інших таких типів помешкань. Ця вибірка повинна включати індивідуальні характеристики орендаря, які встановлені в оголошенні, разом з характеристиками житла і оренди. Вибірка має характеризувати оголошення на певний момент. Тобто недоцільним є збір даних за різний час оренди кімнат, інакше дані можуть бути спотворені. Однак якщо оголошення зібрані за декілька днів, то такий випадок є доцільним, зважаючи на те, що кімнати здаються і орендуються на тривалий період часу (від 1 місяця). При зборі даних одразу потрібно аналізувати спосіб визначення вартості оренди, адже можливо, що вартість визначається за місяць, що є стандартним, але можуть бути варіанти і за тиждень, квартал, півріччя або рік. У такому випадку вартість оренди має бути приведена до місячної вартості:

- якщо вартість встановлена на тиждень, то доцільно відповідне значення помножити на 4,5;
- якщо вартість встановлена на квартал, то доцільно відповідне значення поділити на 3;
- якщо вартість встановлена на півріччя, то доцільно відповідне значення поділити на 6;
- якщо вартість встановлена на рік, то доцільно відповідне значення поділити на 12.

Усі екземпляри вибірки мають представляти різні оголошення: усі екземпляри, які стосуються вартості оренди тієї самої кімнати в різні моменти часу мають бути виключені окремо.

До вибірки можуть включатися випадки оренди кімнат, які стосуються різних міст у країні, адже однією з ознак у моделі має бути встановлено поштовий індекс, який визначає локацію, де розташовано помешкання.

У підсумку вибірка має бути сформована таким чином, щоб містити наступні вхідні ознаки:

- мінімальний строк, на який можна орендувати кімнату за цим оголошенням;
- максимальний строк, на який можна орендувати кімнату за цим оголошенням;
- мінімальний вік, з якого можна орендувати кімнату за цим оголошенням;
- максимальний вік, до якого можна орендувати кімнату за цим оголошенням;
- тип помешкання, в якому знаходиться кімната, яку пропонується орендувати в оголошенні: кімната в будинку, кімната в квартирі, спільна власність;
- індекс, за яким розташовано квартиру, тобто помешкання, де ця квартира знаходиться;
- зайнятість можливого орендаря кімнати: не важлива, студент, працівник;
- необхідність мати рекомендацію від колишнього орендодавця для оренди кімнати за цим оголошенням;
- стать, яку може мати орендар цієї кімнати: чоловіки або жінки, перевага (тільки) чоловікам, перевага (тільки) жінкам;
- можливість паркувати автомобіль біля квартири;
- наявність гаража для паркування автомобіля біля квартири;
- наявність саду або тераси на території помешкання, яку може використовувати орендар квартири;
- наявність балкону в помешканні, який може використовувати орендар квартири;
- наявність вітальні на території помешкання, яку може використовувати орендар квартири;

- наявність інтернету на території помешкання, яку може використовувати орендар квартири;
- можливість орендувати кімнату парі або сім'ї;
- можливість курити в кімнаті;
- можливість орендувати кімнату з наявними домашніми тваринами;
- вартість місяця оренди кімнати.

На наступному етапі відбувається опрацювання даних з вибірки, створеної на першому етапі. Усі значення мають бути перетворені до цілочисельних, тобто це або значення 0 чи 1, або ширша кількість можливих значень. Для вартості місяця оренди кімнати рекомендовано виконати нормування.

Далі має бути створено модель прогнозування вартості оренди кімнати для проживання, побудована за допомогою методу XGBoost на основі принципів бустингу з визначенням необхідної кількості дерев та максимальної глибини дерева, яку рекомендовано обирати в 2-3 рази меншою за кількість вхідних ознак.

Наступним етапом має бути виконано навчання моделі на основі вибірки, обробленої на другому етапі, де вихідною ознакою є вартість місяця оренди кімнати, а вхідними ознаками – всі інші ознаки вибірки даних.

Останній етап реалізується для нових оголошень, для яких має бути визначено значення всіх ознак окрім вихідної (вартості місяця оренди кімнати), яка має бути обчислена на основі моделі прогнозування вартості оренди кімнати для проживання, яку було навчено.

2.3 Висновки за розділом 2

Описано задачу прогнозування вартості оренди кімнати для проживання і розроблений метод, що дозволяє розв'язати цю задачу. Наукова новизна роботи полягає в створеному методі прогнозування вартості оренди кімнати для проживання, який базується на застосуванні ансамблів дерев

рішень на основі принципів бустингу і який дозволяє врахувати індивідуальні характеристики орендаря, додаючи їх до ознак створеної моделі прогнозування, що визначають загальні характеристики житла і оренди.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір рішень для веброзробки

Вибір рішень для веброзробки охоплює вибір цілого ряду засобів, фактично стеку технологій, до якого було включено базові засоби, до яких належать мови HTML, CSS, а в якості основної мови веброзробки використано Python, враховуючи можливість цією мовою створювати серверні рішення, реалізацію нею всіх сучасних парадигм програмування, простоту інтерпретації коду та наявність великої кількості спеціалізованих і поширених пакетів для реалізації машинного навчання. Проте веброзробка на основі Python може виконуватися на основі різних вебфреймворків, які визначають те, на скільки швидко вдасться створити результуюче рішення, яке буде відповідати всім вимогам. Вибір виконувався з Django [6]-[7], який має більше вбудованих засобів, включаючи одразу готові механізми роботи з даними, засоби безпеки, і Falcon [8]-[9] (табл. 3.1). Тож у результаті вибір зроблений на користь Django.

Таблиця 3.1 – Порівняння вебфреймворків Falcon і Django для реалізації програми

Критерій	Falcon	Django
Мова програмування	Python	Python
Створення і підключення моделі прогнозування вартості оренди кімнати для проживання	Так	Так
Базові можливості роботи з даними для адміністратора	Немає	Інтегровані
Створення вебсторінок за допомогою рушія шаблонів	Немає	Інтегровано
Підтримка засобів безпеки	Потребує встановлення зовнішніх засобів	Інтегровано, широка

3.2 Моделювання програмного забезпечення на основі функціональних вимог

Кожен користувач, який розпочав роботу з програмою початково знаходиться в ролі орендаря, при цьому не всі сценарії роботи орендаря такому користувачу доступні в повній мірі, адже для виконання певних дій може бути потрібна авторизація. Стандартно після реєстрації користувач працює в ролі орендаря.

Якщо користувач висловлює бажання внести дані свого житла, створити оголошення про оренду кімнати, то йому надаються для використання додаткові права орендодавця. Тоді він додатково до наявних сценаріїв отримує сценарії орендодавця.

Сценарії орендаря кімнати:

- пошук кімнати для подальшої оренди;
- пошук орендаря, який бажає орендувати кімнату;
- створення запиту на оренду кімнати, для якого є альтернативний сценарій скасування запиту на оренду кімнати, що відбувається у випадку наявного власного створеного запиту на оренду кімнати;
- перегляд запиту на оренду кімнати;
- перегляд переліку створених запитів на оренду кімнати;
- підтвердження запиту за оголошенням про пошук кімнати, для якого є альтернативний сценарій скасування запиту за оголошенням про пошук кімнати для оренди, що відбувається у випадку наявного власного створеного запиту за оголошенням про пошук кімнати;
- перегляд запиту за оголошенням про пошук кімнати;
- перегляд переліку адресованих запитів за оголошенням про пошук кімнати;
- перегляд даних житла, що включає сценарій перегляду всіх кімнат, які доступні для оренди за житлом;
- оформлення відгуку за результатами оренди кімнати про кімнату;

- перегляд відгуків за кімнатою;
- перегляд профільної сторінки орендодавця;
- перегляд профільної сторінки орендаря;
- створення оголошення про пошук кімнати для оренди, для якого є альтернативний сценарій редагування оголошення про пошук кімнати, що відбувається у випадку, коли є власноруч створене оголошення про пошук кімнати для оренди, та альтернативний сценарій зняття з розгляду оголошення про пошук кімнати, що відбувається у випадку, коли є власноруч створене оголошення про пошук кімнати для оренди і воно є активним;

- перегляд оголошення про пошук кімнати;
- перегляд переліку всіх власних оголошень про пошук кімнати;
- перегляд оголошення про оренду кімнати;
- відправлення короткого повідомлення;
- отримання всіх коротких повідомлень, відправлених користувачем;
- отримання комунікацій користувача;
- створення закладки на кімнаті для оренди, для якого є альтернативний сценарій вилучення закладки на кімнаті для оренди, що відбувається у випадку, коли є власноруч створена закладка на кімнаті для оренди;

- перегляд закладок кімнат для оренди;
- авторизація в програмі, для якого є альтернативний сценарій реєстрація (рис. 3.1).

Сценарії орендодавця на додачу до сценаріїв орендаря:

- створення запиту за оголошенням про пошук кімнати для оренди, створеним орендодавцем;
- оформлення відгуку за результатами оренди кімнати про орендаря;
- створення оголошення про оренду кімнати, для якого є альтернативні сценарії редагування оголошення про оренду кімнати, що використовується за наявності раніше створеного власного оголошення і тільки саме до нього, та зняття з розгляду оголошення про оренду кімнати,

що використовується за наявності раніше створеного власного оголошення, тільки саме до нього і тільки у випадку якщо це оголошення активне.

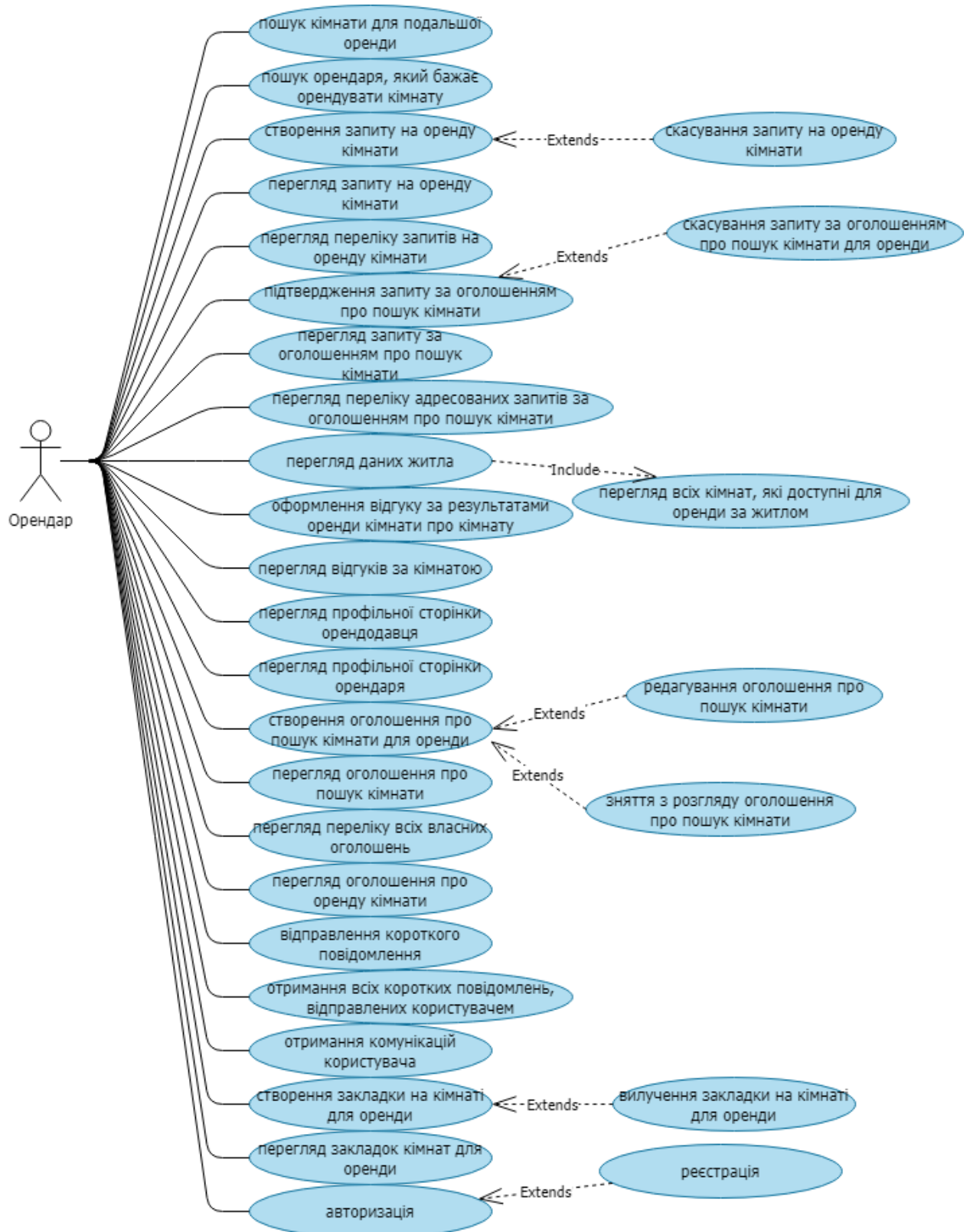


Рисунок 3.1 – Діаграма прецедентів орендаря кімнати

Подальші сценарії орендодавця:

- внесення даних житла, в якому орендуються кімнати, для якого є альтернативні сценарії редагування даних житла, що використовується за наявності збереженого запису про власне житло, та вилучення даних житла, що використовується за наявності збереженого запису про власне житло та за відсутності доданих кімнат;
- порівняння кандидатів на оренду кімнати;
- надсилання запиту на уточнення щодо орендаря орендодавцю, для якого є альтернативний сценарій скасування запиту на уточнення щодо орендаря орендодавцю, що відбувається у випадку, коли є власноруч створений запит на уточнення щодо орендаря орендодавцю і відповідь на цей запит ще не було отримано;
- створення відповіді на уточнення щодо орендаря орендодавцю;
- перегляд отриманих запитів на уточнення щодо орендаря;
- перегляд створених запитів на уточнення щодо орендаря;
- створення закладки на оголошенні про пошук кімнати для подальшої оренди, для якого є альтернативний сценарій вилучення закладки на оголошенні про пошук кімнати для подальшої оренди, що відбувається у випадку, коли є власноруч створена закладка на оголошенні про пошук кімнати для подальшої оренди;
- перегляд закладок оголошень про пошук кімнати для подальшої оренди;
- підтвердження запиту на оренду кімнати;
- переведення запиту на оренду кімнати до кандидатського стану;
- перегляд переліку адресованих запитів на оренду кімнати;
- перегляд переліку створених запитів за оголошенням про пошук кімнати;
- перегляд всіх кімнат, які доступні для оренди за житлом;
- перегляд переліку всіх власних оголошень про оренду кімнати;
- перегляд всіх власних записів про житло (рис. 3.2).

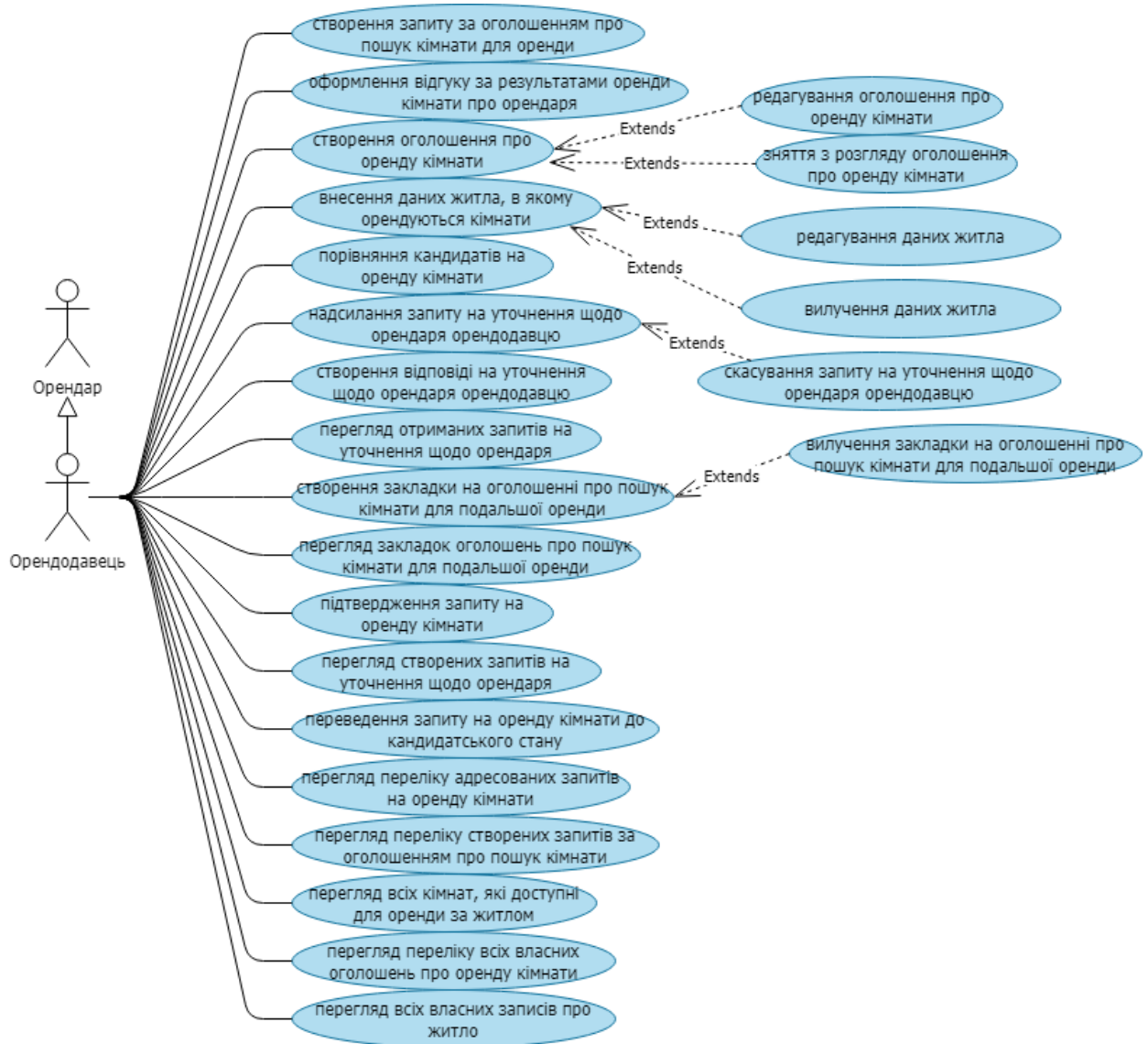


Рисунок 3.2 – Діаграма прецедентів орендодавця

3.3 Проектування структури бази даних

База даних (БД) враховує всі основні сутності предметної області оренди кімнат для проживання та сутностей, які відповідають забезпечуваним програмою функціям. Реалізацію роботи через створення описаної структури БД і підтримку роботи БД виконано на основі MySQL.

Сутностями БД, які визначають базову деталізацію, є:

– країна Country – сутність, яка розпочинає ієрархію з визначення місця розташування тієї будівлі, в якій знаходиться кімната, яку може орендувати або орендує користувач-орендар;

– локація Location (місто, селище), яке знаходиться в країні і в якій може знаходитися квартира або будинок, де здається квартира, що є другим рівнем ієрархії;

– район Locationregion локації, де знаходиться будинок або квартира, де здається квартира, що є третім рівнем ієрархії і який задається безпосередньо для квартири або будинку, а відповідно і для кімнати;

– захоплення людини Interest, що визначає основні захоплення, які може деталізувати користувач програми стосовно себе (наприклад, музика, футбол, для яких тоді далі має бути або може бути вказано, які музичні колективи слухає цей користувач, за які футбольні команди вболіває користувач);

– критерій порівняння Criterion, який може використовуватися для запитів про орендарів до орендодавців та для аналізу кандидатів (наприклад, комунікабельність, пунктуальність);

– тип самого житла (квартири або будинку), в якому здається кімната, Propertytype;

– тип самої кімнати Roomtype;

– професія, що визначає рід занять людини, Occupation;

– мова, яка використовується людиною для спілкування, Language.

Як зрозуміло з опису сценаріїв, з даних уже описаних сутностей, користувачі в програмі є або орендарями, або орендодавцями. Відповідно в залежності від того, яку роль виконує користувач, а точніше, якщо користувач пропонує свої кімнати для оренди, то йому доступний розширений набір функцій, а у протилежному випадку він є лише орендарем. Оскільки кожен такий користувач має обліковий запис у програмі, який є відповідно обліковим записом Django, то інформацію про користувача на базі його облікового запису в програмі виділено сутністю Useraccount.

Стосовно самої предметної області базовими є наступні сутності:

– житло Sharedproperty як цілісна одиниця, яка фактично визначає певний контейнер, до якого як окремі компоненти, тобто складові входять

окремі кімнати, які як раз і надаються в оренду через програму, тобто дана сутність необхідна, щоб визначити такий контейнер для орендодавця, яким фактично є будинок або квартира;

- оголошення про оренду кімнати в житлі Roomadvertisement, що як раз і є окремим компонентом, який виділено окремо від житла для того, щоб орендодавець міг розмістити у програмі декілька оголошень про оренду власних кімнат у житлі, вводячи інформацію, яка стосується житла загалом, тільки один раз;

- оголошення про пошук кімнати орендарем Tenantadvertisement, що є сутністю, яка дозволяє орендарю заявити про власне бажання орендувати кімнату.

Зважаючи на останні описані сутності, сутність, яка визначає фотографії у програмі, була розділена на логічні 3 окремі сутності, що включають:

- фотографію користувача (розширення профілю користувача як орендодавця, так і орендаря) Userphoto;
- фотографію житла Sharedpropertyphoto;
- фотографію кімнати для оренди Roomphoto.

Оскільки кімната є тільки складовою житла, а в певному житлі можуть проживати різні люди, то важливо не тільки орендарю представити інформацію про себе, але і надати інформацію про те, хто проживає вже в цій квартирі або будинку. До такої інформації належить:

- визначенням професій людей, які вже проживають в певному житлі Sharedpropertyoccupation, що фактично є таблицею, яка з'єднує таблиці Sharedproperty і Occupation;

- визначення мов, які знають і використовують або можуть використовувати люди, які проживають вже у цьому житлі, – таблиця Sharedpropertycommunication, що фактично є таблицею, яка з'єднує таблиці Sharedproperty і Language.

Аналогічно для оголошення про пошук кімнати орендарем визначаються райони міста, в яких готовий орендувати кімнату орендар, через таблицю Tenantadvertisementregion, яка реалізує зв'язок між таблицями Tenantadvertisement і Locationregion.

Будь-який користувач, тобто і орендар, і орендодавець, можуть визначити для себе власні захоплення, щоб інша сторона могла орієнтуватися на цю інформацію. Відповідна інформація виділена в таблицю Userinterest, що встановлює зв'язок між таблицями Useraccount і Interest. Для кожного захоплення може бути встановлено декілька конкретних виражень цих захоплень користувачів, тобто фактично значень захоплень. Ці значення представлені таблицею Userinterestitem. Це вже можуть бути як раз саме музичні колективи, якщо захопленням користувача є музика.

Мови, якими володіє користувач, представлені в таблиці Userlanguage, що встановлює зв'язок між таблицями Useraccount і Language.

Закладки на оголошення можуть встановлюватися обома сторонами, тому було виділено окремі таблиці:

- закладки до оголошень про оренду кімнат Roomadvertisementsave;
- закладки до оголошень про пошук кімнати орендарем Tenantadvertisementsave.

Ініціативою щодо визначення орендаря, якому буде надано кімнату для проживання, є запит, який зважаючи на два типи оголошень, може бути представлений таблицями БД:

- запит, створений орендодавцем і адресований орендарю, який шукає кімнату для оренди, Landlordrequest;
- запит, створений орендарем і адресований орендодавцю за його оголошенням про оренду кімнати, Tenantrequest.

При порівнянні декількох орендарів, які подали запити на оренду кімнати, орендодавцями використовуються критерії. Ці критерії використовуються зі стандартного набору базово, але при цьому стосуються конкретного орендаря, адже не всі критерії можуть бути задані для кожного

орендаря. Тобто якщо немає інформації про певну його характеристику (за певним критерієм), то значення має бути просто пропущено. Критерії, визначені орендодавцем для орендаря, знаходяться в таблиці `Candidatecriterion`. Окрім стандартних критеріїв можуть бути задані і додаткові, сформульовані самим орендодавцем. Тобто тоді орендодавець не вибирає наявний критерій, а формулює його самостійно. Додаткові критерії, визначені орендодавцем для орендаря, знаходяться в таблиці `Additcandidatecriterion`.

Орендодавці можуть створювати свої порівняння, приймати рішення не тільки на основі даних, наданих самим орендарем, або даних, представлених у відгуках, але і звертатися до орендодавців, у яких він раніше знімав кімнату, за уточненням. Уточнення можуть створюватися від одного орендодавця до іншого і стосуватися орендаря, який раніше проживав у орендодавця-адресата цього уточнення. Ці уточнення знаходяться в таблиці `Landlordadvice`. Уточнення складаються з конкретних критеріїв, які обирає до них орендодавець, який створює це уточнення. Стандартні критерії, визначені орендодавцем в уточненні, знаходяться в таблиці `Landlordadvicecriterion`. Додаткові критерії, визначені орендодавцем в уточненні, знаходяться в таблиці `Additlandlordadvicecriterion`.

Після підтвердження запиту орендодавцем орендар може залишити відгук про кімнату, яку він винаймає, зберігаючи ці дані в таблиці БД `Roomreview`, а орендодавець може залишити відгук про орендаря, який винаймає у нього кімнату, зберігаючи ці дані в таблиці `Tenantreview`.

Орендар і орендодавець можуть обмінюватися короткими повідомленнями, які зберігаються в таблиці `Shortmessage`.

Тож розглянуті сутності та відповідні таблиці дозволили загалом визначити структуру БД (рис. 3.3), яка була програмно в подальшому реалізована через створення відповідних класів моделей, що є способом, який використовується в Django для виконання цих задач.

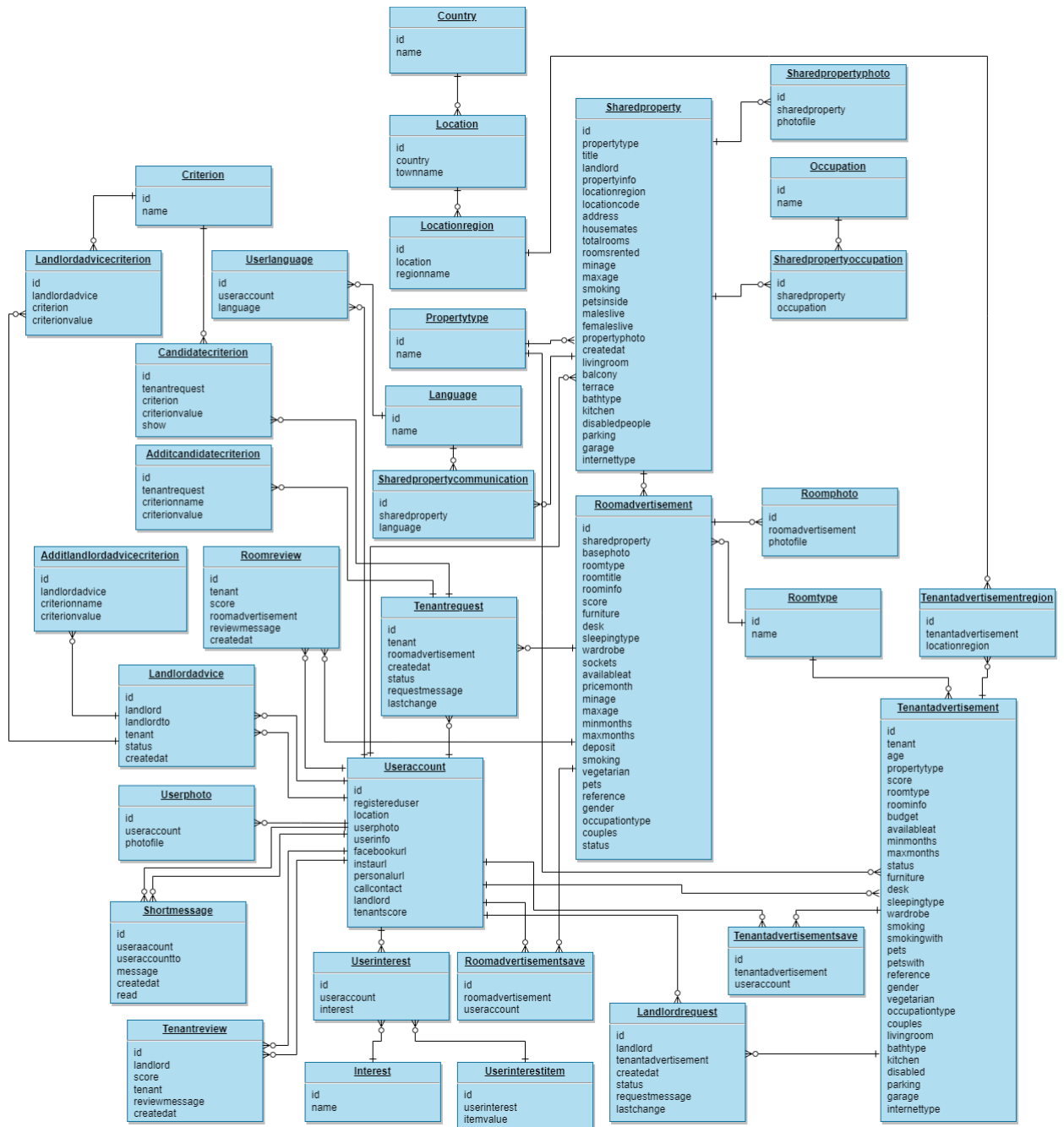


Рисунок 3.3 – Логічна схема БД

3.4 Висновки за розділом 3

Проектування виконано через вибір мов HTML, CSS, Python, фреймворку Django, системи для роботи з БД MySQL для проектування і реалізації програми, через визначення функціональних вимог, створення діаграм прецедентів, визначення структури БД.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Порядок роботи програмного забезпечення

Програма для оренди кімнат для проживання має порядок роботи, який визначається доступом користувач до тих чи інших розділів функцій, але для повноцінного виконання частини з цих функцій або для розширеного виконання інших функцій необхідно спочатку авторизуватися (рис. 4.1) або зареєструватися, якщо користувач не має облікового запису.

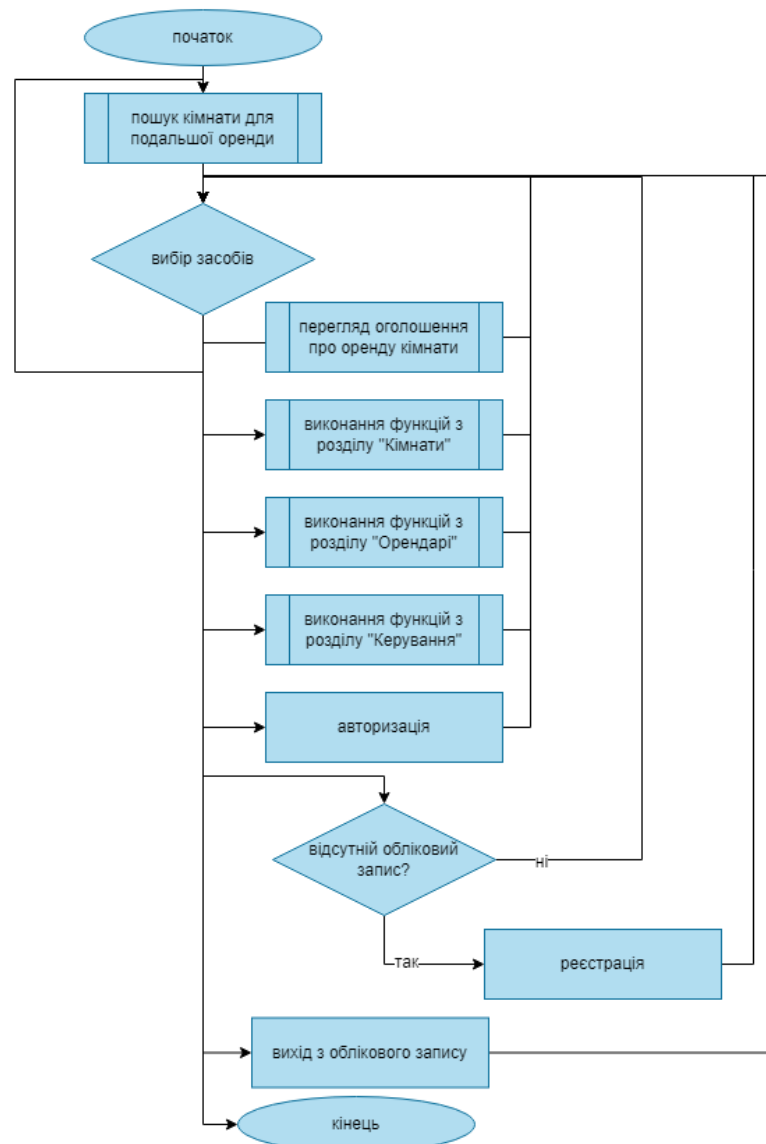


Рисунок 4.1 – Функціональна схема програми для оренди кімнат для проживання

Початком роботи є виконання пошуку кімнати для подальшої оренди. Також на функціональність впливає, чи користувач вже є орендодавцем. Для того щоб стати орендодавцем, потрібно з меню перейти до створення оголошення про оренду кімнати та внести необхідні дані спочатку про житло. В цей момент надається доступ для орендодавця. Відповідно з меню можна обрати з відповідних розділів потрібні функції.

4.2 Рішення з реалізації вебзастосунку

Створення Django-застосунків виконується в чіткій відповідності з архітектурним шаблоном Model-View-Template (MVT) [10], тому основними складовими результуючого проєкту в підсумку є моделі, які були створені відповідними класами, представлення, що є функціями та передбачається підключення шаблонів сторінок, створених за допомогою вільного CSS-шаблону [11].

Реалізація моделі для роботи над даними країни забезпечена через клас Country, а оскільки інформація про країну є початковою в ієрархії, тобто включає тільки назву країни, то серед полів виокремлено ідентифікатор id, який автоматично створюється, та поле name з назвою країни, що реалізовано типом CharField. Так само, тобто з ідентифікатором і назвою (поля id і name), влаштовані класи моделей:

- захоплення людини Interest;
- критерію порівняння Criterion, який може використовуватися для запитів про орендарів до орендодавців та для аналізу кандидатів,
- типу самого житла (квартири або будинку), в якому здається кімната, Propertytype;
- професії, що визначає рід занять людини, Occupation;
- мови, яка використовується людиною для спілкування, Language;
- типу кімнати Roomtype.

Подібно виконано реалізацію моделі для роботи над даними локацій (міст, селищ) через клас `Location`, що окрім ідентифікатора включає зовнішній ключ `country` з моделі `Country`, для чого використано тип `ForeignKey`, та поле `townname` з назвою міста, яке в цій країні знаходиться, що реалізовано типом `CharField`. Кожна наступна модель містить серед полів зокрема і ідентифікатор, але оскільки він створюється автоматично, то далі визначено всі поля для кожної моделі окрім ідентифікатора.

При пошуку кімнат саме житло визначається для користувача зазвичай районом, в якому це житло знаходиться, реалізація моделі для роботи над даними району забезпечена через клас `Locationregion` з полями:

- `location` – локація, в якій знаходиться цей район (зазвичай місто, в якому знаходиться цей район), це зовнішній ключ з моделі `Location`, для чого використано тип `ForeignKey`;

- `regionname` – назва району, який як раз і представляється цією моделлю, що реалізовано типом `CharField`.

Реалізація моделі для роботи над інформацією про користувача на базі його облікового запису в програмі забезпечена через клас `Useraccount` з полями:

- `registereduser` – обліковий запис, який використовується для роботи користувача (як орендодавця, так і орендаря), це зовнішній ключ з моделі `User`, яка за замовчуванням створюється в Django через підпакет `django.contrib.auth`, для чого використано тип `ForeignKey`;

- `location` – локація, в якій проживає користувач (орендодавець або орендар), це зовнішній ключ з моделі `Location`, для чого використано тип `ForeignKey`;

- `userphoto` – основна фотографія профілю користувача, що реалізовано типом `ImageField`;

- `userinfo` – інформація, яка описує користувача як людину, відповідно може використовуватися на додаток до інформації, представленій в самому

оголошенні про орендаря (або в профілі орендодавця), що реалізовано типом CharField;

- facebookurl – адреса персональної сторінки користувача в мережі Facebook, що реалізовано типом URLField;

- instaurl – адреса персональної сторінки користувача в мережі Instagram, що реалізовано типом URLField;

- personalurl – адреса персональної вебсторінки або вебсайту користувача, що реалізовано типом URLField;

- callcontact – номер телефону для з'єднання з користувачем як орендодавцем або орендарем, що реалізовано типом CharField;

- landlord – мітка, яка вказує на те, чи є цей користувач орендодавцем, що реалізовано типом BooleanField: мітка необхідна для того, щоб демонструвати користувачу відповідні функції в програмі, які визначають роботу орендодавця, що необхідно тоді, коли орендодавець висловив бажання створити і створив оголошення про оренду кімнати;

- tenantscore – оцінка, яку в середньому отримав цей користувач як орендар, що реалізовано типом FloatField: оцінка є у кожного користувача, оскільки у нього одразу після реєстрації доступні можливості орендування кімнат, при цьому користувач може їх не використовувати взагалі або до певного моменту і використовувати програму тільки для роботи з оголошеннями про власне житло, тобто кімнати які цей користувач виставляє для можливої оренди.

Реалізація моделі для роботи над фотографіями користувача (розширення профілю користувача як орендодавця, так і орендаря) забезпечена через клас Userphoto з полями:

- useraccount – користувач, до якого належить ця фотографія, це зовнішній ключ з моделі Useraccount, для чого використано тип ForeignKey;

- photofile – фотографія користувача (як орендодавця, так і орендаря), що реалізовано типом ImageField.

Реалізація моделі для роботи над самим цілим житлом, у якому здаються окремі кімнати або кімната, тобто над квартирою, будинком, забезпечена через клас `Sharedproperty` з полями:

- `propertytype` – тип самого житла, в якому далі визначаються кімнати для оренди, це зовнішній ключ з моделі `Propertytype`, для чого використано тип `ForeignKey`;

- `title` – назва, якою охарактеризовано житло, що реалізовано типом `CharField`;

- `landlord` – орендодавець, який володіє цим житлом і який виставляє інформацію про нього у вебзастосунку, це зовнішній ключ з моделі `Useraccount`, для чого використано тип `ForeignKey`;

- `propertyinfo` – інформація, яка описує житло, що реалізовано типом `CharField`;

- `locationregion` – район в певній локації, в якому знаходиться це житло, це зовнішній ключ з моделі `Locationregion`, для чого використано тип `ForeignKey`;

- `locationcode` – поштовий індекс розташування цього житла в районі локації (необхідно зазначати, оскільки один район може охоплювати декілька поштових індексів, а індекс чіткіше визначає розташування житла), що реалізовано типом `CharField`;

- `address` – адреса, за якою знаходиться житло, що реалізовано типом `CharField`;

- `housemates` – кількість людей, які проживають у цьому житлі, тобто співмешканців, що реалізовано типом `PositiveSmallIntegerField`;

- `totalrooms` – кількість кімнат у цьому житлі загалом, включаючи ті, що орендуються іншими людьми та ті, які використовуються власником, що реалізовано типом `PositiveSmallIntegerField`;

- `roomsrented` – кількість кімнат у цьому житлі, що орендуються іншими людьми, що реалізовано типом `PositiveSmallIntegerField`;

- minage – мінімальний вік людей, які проживають вже у цьому житлі, що реалізовано типом PositiveIntegerField;
- maxage – максимальний вік людей, які проживають вже у цьому житлі, що реалізовано типом PositiveIntegerField;
- smoking – мітка про те, чи палять люди, які проживають вже у цьому житлі, що реалізовано типом BooleanField;
- petsinside – мітка про те, чи проживають домашні тварини вже у цьому житлі, що реалізовано типом BooleanField;
- maleslive – кількість чоловіків, які вже проживають у цьому житлі, що реалізовано типом PositiveIntegerField;
- femaleslive – кількість жінок, які вже проживають у цьому житлі, що реалізовано типом PositiveIntegerField;
- propertyphoto – фотографія житла (основна), що реалізована типом ImageField;
- createdat – дата внесення інформації про житло в саму програму, що реалізована типом DateField;
- livingroom – мітка про те, чи є вітальня, яку можна використовувати зокрема і після оренди кімнати, в цьому житлі, що реалізовано типом BooleanField;
- balcony – мітка про те, чи є балкон, який можна використовувати зокрема і після оренди кімнати, в цьому житлі, що реалізовано типом BooleanField;
- terrace – мітка про те, чи є тераса або певний зовнішній простір, який можна використовувати зокрема і після оренди кімнати, в цьому житлі, що реалізовано типом BooleanField;
- bathtype – тип ванної кімнати в цьому житлі, при чому саме такої, яку може використовувати орендар під час проживання, що реалізовано типом PositiveIntegerField;

- kitchen – мітка про те, чи є кухня, яку можна використовувати зокрема і після оренди кімнати, в цьому житлі, що реалізовано типом BooleanField;

- disabledpeople – мітка про те, чи є все необхідне в житлі для використання і доступу до нього людям з обмеженими можливостями, що реалізовано типом BooleanField;

- parking – мітка про те, чи є поруч з житлом доступний для використання для власного автомобіля певний варіант паркування автомобіля, що реалізовано типом BooleanField;

- garage – мітка про те, чи є доступний для використання для власного автомобіля орендаря гараж, що реалізовано типом BooleanField;

- internettype – тип доступного для використання підключення до інтернету, що реалізовано типом PositiveSmallIntegerField.

Реалізація моделі для роботи над фотографіями житла забезпечена через клас Sharedpropertyphoto з полями:

- sharedproperty – житло, яке представляє певна фотографія, це зовнішній ключ з моделі Sharedproperty, для чого використано тип ForeignKey;

- photofile – фотографія житла (додаткова), що реалізована типом ImageField.

Реалізація моделі для роботи над визначенням професій людей, які вже проживають в певному житлі, забезпечена через клас Sharedpropertyoccupation з полями:

- sharedproperty – житло, в якому вже проживає людина, яка працює за цією професією, це зовнішній ключ з моделі Sharedproperty, для чого використано тип ForeignKey;

- occupation – професія, за якою працює людина, яка вже проживає у цьому житлі, це зовнішній ключ з моделі Occupation, для чого використано тип ForeignKey.

Реалізація моделі для роботи над визначенням мов, які знають і використовують або можуть використовувати люди, які проживають вже у цьому житлі, забезпечена через клас `Sharedpropertycommunication` з полями:

- `sharedproperty` – житло, в якому вже проживає людина, яка знає цю мову, це зовнішній ключ з моделі `Sharedproperty`, для чого використано тип `ForeignKey`;

- `language` – мова, якою вміє розмовляти і використовує або може використовувати людина, яка вже проживає в цьому житлі, це зовнішній ключ з моделі `Language`, для чого використано тип `ForeignKey`.

Реалізація моделі для роботи над оголошеннями про оренду кімнати в житлі забезпечена через клас `Roomadvertisement` з полями:

- `sharedproperty` – житло, в якому знаходиться і частиною якого є ця кімната, які пропонується для оренди, це зовнішній ключ з моделі `Sharedproperty`, для чого використано тип `ForeignKey`;

- `basephoto` – фотографія конкретної кімнати в житлі (основна), а не житла загалом, що реалізована типом `ImageField`;

- `roomtype` – тип кімнати, до якого належить ця кімната, це зовнішній ключ з моделі `Roomtype`, для чого використано тип `ForeignKey`;

- `roomtitle` – назва, якою охарактеризовано кімнату і яка використовується для ідентифікації кімнати серед всіх кімнат житла, що реалізовано типом `CharField`;

- `roominfo` – інформація, яка описує кімнату та принципи її оренди, що реалізовано типом `CharField`;

- `score` – оцінка, яку в середньому було виставлено кімнаті після її оренди конкретними орендарями, що реалізовано типом `FloatField`;

- `furniture` – мітка про те, чи є меблі в цій кімнаті (інакше потрібно мати власні меблі при оренді), що реалізовано типом `BooleanField`;

- `desk` – мітка про те, чи є в цій кімнаті певна робоча поверхня або стіл для роботи за ним, що реалізовано типом `BooleanField`;

- `sleepingtype` – тип спального місця, яке є в цій кімнаті, що реалізовано типом `PositiveSmallIntegerField`;
- `wardrobe` – мітка про те, чи є шафа в цій кімнаті для зберігання речей, що реалізовано типом `BooleanField`;
- `sockets` – кількість розеток у кімнаті, що реалізовано типом `PositiveSmallIntegerField`;
- `availableat` – дата, з якої ця кімната доступна для оренди, що реалізовано типом `DateField`;
- `pricemonth` – сума, яку необхідно сплатити за оренду цієї кімнати за 1 місяць, що реалізовано типом `PositiveIntegerField`;
- `minage` – мінімальний вік, за якого орендаря готовий прийняти в своєму житлі орендодавець, що реалізовано типом `PositiveSmallIntegerField`;
- `maxage` – максимальний вік, за якого орендаря готовий прийняти в своєму житлі орендодавець, що реалізовано типом `PositiveSmallIntegerField`;
- `minmonths` – мінімальна кількість місяців, на яку можна орендувати цю кімнату, що реалізовано типом `PositiveSmallIntegerField`;
- `maxmonths` – максимальна кількість місяців, на яку можна орендувати цю кімнату, що реалізовано типом `PositiveSmallIntegerField`;
- `deposit` – депозит, який необхідно залишити в заставу, якщо орендувати цю кімнату, що реалізовано типом `PositiveIntegerField`;
- `smoking` – мітка про те, чи дозволено палити в кімнаті, що реалізовано типом `BooleanField`;
- `pets` – мітка про те, чи дозволено проживати в кімнаті з власними домашніми тваринами, що реалізовано типом `BooleanField`;
- `reference` – необхідність наявності рекомендації від попередніх власників житла, яке орендувала людина, для оренди кімнати, що реалізовано типом `BooleanField`;
- `gender` – стать орендаря, який може орендувати цю кімнату, що реалізовано типом `PositiveSmallIntegerField` (будь-який, тільки чоловіки, тільки жінки);

- `occupationtype` – тип професійної зайнятості, а якого можна орендувати цю кімнату, що реалізовано типом `PositiveSmallIntegerField` (студенти, люди, які працюють);

- `vegetarian` – мітка про те, що кімнату можуть орендувати тільки вегетаріанці або будь-хто, що реалізовано типом `BooleanField`;

- `couples` – мітка про те, що кімнату можуть орендувати або не можуть орендувати пари, що реалізовано типом `BooleanField`;

- `status` – стан оголошення про оренду кімнати, що реалізовано типом `BooleanField`.

Реалізація моделі для роботи над фотографіями кімнат для оренди забезпечена через клас `Roomphoto` з полями:

- `roomadvertisement` – оголошення про оренду кімнати в житлі, до якого як додаткова відноситься ця фотографія кімнати, це зовнішній ключ з моделі `Roomadvertisement`, для чого використано тип `ForeignKey`;

- `photofile` – фотографія кімнати (додаткова), що реалізована типом `ImageField`.

Реалізація моделі для роботи над закладками до оголошень про оренду кімнат забезпечена через клас `Roomadvertisementsave` з полями:

- `roomadvertisement` – оголошення про оренду кімнати в житлі, до якого створена ця закладка, це зовнішній ключ з моделі `Roomadvertisement`, для чого використано тип `ForeignKey`;

- `useraccount` – користувач, який створив цю закладку до оголошення про оренду кімнати, це зовнішній ключ з моделі `Useraccount`, для чого використано тип `ForeignKey`.

Реалізація моделі для роботи над даними оголошення про пошук кімнати орендарем забезпечена через клас `Tenantadvertisement` з полями:

- `tenant` – орендар (користувацький запис), який шукає кімнату для оренди, це зовнішній ключ з моделі `Useraccount`, для чого використано тип `ForeignKey`;

- age – вік орендаря, який шукає кімнату для оренди, що реалізовано типом `PositiveSmallIntegerField`;
- propertytype – тип житла, в якому готовий орендувати кімнату цей орендар, це зовнішній ключ з моделі `Propertytype`, для чого використано тип `ForeignKey`;
- roomtype – тип кімнати, яку готовий орендувати цей орендар, це зовнішній ключ з моделі `Propertytype`, для чого використано тип `ForeignKey`;
- roominfo – опис себе, своїх вимог, побажань орендарем, що реалізовано типом `CharField`;
- budget – максимальний бюджет, який може виділити людина для оренди кімнати, що реалізовано типом `PositiveIntegerField`;
- availableat – дата, з якої цей орендар готовий орендувати кімнату, що реалізовано типом `DateField`;
- minmonths – мінімальна кількість місяців, на яку може орендувати кімнату цей орендар, що реалізовано типом `PositiveSmallIntegerField`;
- maxmonths – максимальна кількість місяців, на яку може орендувати кімнату цей орендар, що реалізовано типом `PositiveSmallIntegerField`;
- furniture – мітка про те, чи потрібні орендарю обов’язково меблі в кімнаті, чи він має власні або готовий забезпечити, що реалізовано типом `BooleanField`;
- desk – мітка про те, чи потрібна орендарю в цій кімнаті певна робоча поверхня або стіл для роботи за ним, що реалізовано типом `BooleanField`;
- sleepingtype – тип спального місця, яке в кімнаті потрібно орендарю, що реалізовано типом `PositiveSmallIntegerField` (найменший рівень, відповідно потрібен тип спального місця не менший ніж зазначений);
- wardrobe – мітка про те, чи потрібна шафа в кімнаті для зберігання речей орендарю, що реалізовано типом `BooleanField`;
- smoking – мітка про те, чи палить сам цей орендар, що реалізовано типом `BooleanField`;

- smokingwith – мітка про те, чи готовий проживати цей орендар в житлі, де живуть люди, які палять, що реалізовано типом BooleanField;
- pets – мітка про те, чи має цей орендар власних домашніх тварин, що реалізовано типом BooleanField;
- petswith – мітка про те, чи готовий орендар проживати в житлі, де живуть домашні тварини, що реалізовано типом BooleanField;
- reference – наявність окремої (документальної) рекомендації від попередніх власників житла, яке орендував цей орендар, що реалізовано типом BooleanField;
- gender – стать орендаря, що реалізовано типом BooleanField;
- occupationtype – тип професійної зайнятості орендаря, що реалізовано типом PositiveSmallIntegerField;
- vegetarian – мітка про те, чи є цей орендар вегетаріанцем, чи ні, що реалізовано типом BooleanField;
- couples – мітка про те, чи є цей орендар індивідуальним орендарем, чи він проживає в парі або сім'єю, що реалізовано типом BooleanField;
- livingroom – мітка про те, чи потребує орендар наявності у житлі доступної для нього вітальні, що реалізовано типом BooleanField;
- bathtype – тип ванної кімнати, до якої потребує доступ орендар під час оренди житла, що реалізовано типом PositiveSmallIntegerField;
- kitchen – мітка про те, чи потребує орендар наявності у житлі доступної для нього кухні, що реалізовано типом BooleanField;
- disabled – мітка про те, чи потребує цей орендар доступу для людей з обмеженими можливостями, що реалізовано типом BooleanField;
- parking – мітка про те, чи потребує цей орендар для використання для власного автомобіля певний варіант паркування автомобіля, що реалізовано типом BooleanField;
- garage – мітка про те, чи потребує цей орендар для використання для власного автомобіля гараж, що реалізовано типом BooleanField;

– `internettype` – тип необхідного для використання підключення до інтернету, що реалізовано типом `PositiveSmallIntegerField`.

Реалізація моделі для роботи над визначенням районів міста, в яких готовий орендувати кімнату орендар, забезпечена через клас `Tenantadvertisementregion` з полями:

– `tenantadvertisement` – оголошення про пошук кімнати орендарем, це зовнішній ключ з моделі `Tenantadvertisement`, для чого використано тип `ForeignKey`;

– `locationregion` – район міста, в якому готовий орендувати кімнату і проживати орендар, це зовнішній ключ з моделі `Locationregion`, для чого використано тип `ForeignKey`;

– `status` – стан оголошення про пошук кімнати для оренди, що реалізовано типом `BooleanField`.

Реалізація моделі для роботи над закладками до оголошень про пошук кімнати орендарем забезпечена через клас `Tenantadvertisementsave` з полями:

– `tenantadvertisement` – оголошення про пошук кімнати орендарем, до якого створена ця закладка, це зовнішній ключ з моделі `Tenantadvertisement`, для чого використано тип `ForeignKey`;

– `useraccount` – користувач, який створив цю закладку до оголошення про пошук кімнати орендарем, це зовнішній ключ з моделі `Useraccount`, для чого використано тип `ForeignKey`.

Реалізація моделі для роботи над захопленнями, які має користувач (орендар або орендодавець), забезпечена через клас `Userinterest` з полями:

– `useraccount` – користувач, який має це захоплення, це зовнішній ключ з моделі `Useraccount`, для чого використано тип `ForeignKey`;

– `interest` – захоплення, яке має цей користувач, це зовнішній ключ з моделі `Interest`, для чого використано тип `ForeignKey`.

Реалізація моделі для роботи над конкретними вираженнями цих захоплень користувачів забезпечена через клас `Userinterestitem` з полями:

- `userinterest` – захоплення, яке має користувач, це зовнішній ключ з моделі `Userinterest`, для чого використано тип `ForeignKey`;

- `itemvalue` – вираження (конкретне значення), яке представляє це захоплення користувача (наприклад, назва групи, яку слухає користувач, якщо його захопленням є музика), що реалізовано типом `CharField`.

Реалізація моделі для роботи над мовами, якими володіє користувач, забезпечена через клас `Userlanguage` з полями:

- `useraccount` – користувач, який володіє цією мовою, це зовнішній ключ з моделі `Useraccount`, для чого використано тип `ForeignKey`;

- `language` – мова, якою володіє користувач, це зовнішній ключ з моделі `Language`, для чого використано тип `ForeignKey`.

Реалізація моделі для роботи над запитом, створеним орендодавцем і адресованим орендарю, який шукає кімнату для оренди, забезпечена через клас `Landlordrequest` з полями:

- `landlord` – орендодавець, який створив цей запит, це зовнішній ключ з моделі `Useraccount`, для чого використано тип `ForeignKey`;

- `tenantadvertisement` – оголошення про пошук кімнати орендарем, до якого адресовано цей запит, створений орендодавцем, це зовнішній ключ з моделі `Tenantadvertisement`, для чого використано тип `ForeignKey`;

- `createdat` – дата створення орендодавцем запиту, адресованого орендарю, що реалізовано типом `DateTimeField`;

- `status` – стан запиту, створеного орендодавцем і адресованого орендарю, що реалізовано типом `PositiveSmallIntegerField`;

- `requestmessage` – коротке повідомлення при створенні запиту, що реалізовано типом `CharField`;

- `lastchange` – час останньої зміни стану запиту, створеного орендодавцем і адресованого орендарю, що реалізовано типом `DateTimeField`.

Реалізація моделі для роботи над запитом, створеним орендарем і адресованим орендодавцю за його оголошенням про оренду кімнати, забезпечена через клас `Tenantrequest` з полями:

- `tenant` – орендар, який створив цей запит, адресований орендодавцю за його оголошенням про оренду кімнати, це зовнішній ключ з моделі `Useraccount`, для чого використано тип `ForeignKey`;
- `roomadvertisement` – оголошення про оренду кімнати, до якого створено цей запит, це зовнішній ключ з моделі `Roomadvertisement`, для чого використано тип `ForeignKey`;
- `createdat` – дата створення орендарем запиту, адресованого орендодавцю, що реалізовано типом `DateTimeField`;
- `status` – стан запиту, створеного орендарем і адресованого орендодавцю, що реалізовано типом `PositiveSmallIntegerField`;
- `requestmessage` – коротке повідомлення при створенні запиту, що реалізовано типом `CharField`;
- `lastchange` – час останньої зміни стану запиту, створеного орендодавцем і адресованого орендарю, що реалізовано типом `DateTimeField`.

Реалізація моделі для роботи над критеріями, визначеними орендодавцем для орендаря, забезпечена через клас `Candidatecriterion` з полями:

- `tenantrequest` – запит, створений орендарем і адресований орендодавцю за його оголошенням про оренду кімнати, для якого критерій встановлено, це зовнішній ключ з моделі `Tenantrequest`, для чого використано тип `ForeignKey`;
- `criterion` – критерій, який встановлено для цього орендаря орендодавцем, це зовнішній ключ з моделі `Criterion`, для чого використано тип `ForeignKey`;
- `criterionvalue` – значення критерію, яке встановлено для даного орендаря орендодавцем, що реалізовано типом `CharField`;

– show – показувати цей критерій для орендаря чи ні (критерії можуть бути відновлені разом зі значеннями у випадку, якщо орендар був вилучений з порівняння, а потім зі сторінки запитів був доданий), реалізовано типом BooleanField.

Реалізація моделі для роботи над додатковими критеріями, визначеними орендодавцем для орендаря, забезпечена через клас Additcandidatecriterion з полями:

– tenantrequest – запит, створений орендарем і адресований орендодавцю за його оголошенням про оренду кімнати, для якого цей додатковий критерій встановлено, це зовнішній ключ з моделі Tenantrequest, для чого використано тип ForeignKey;

– criterionname – назва додаткового критерію, який встановлено для цього орендаря орендодавцем, що реалізовано типом CharField;

– criterionvalue – значення додаткового критерію, яке встановлено для даного орендаря орендодавцем, що реалізовано типом CharField.

Реалізація моделі для роботи над уточненнями, які можуть створюватися від одного орендодавця до іншого і стосуватися орендаря, який раніше проживав у орендодавця-адресата цього уточнення, забезпечена через клас Landlordadvice з полями:

– landlord – орендодавець, який створив це уточнення, це зовнішній ключ з моделі Useraccount, для чого використано тип ForeignKey;

– landlordto – орендодавець, до якого адресовано дане уточнення, це зовнішній ключ з моделі Useraccount, для чого використано тип ForeignKey;

– tenant – орендар, відносно якого дане уточнення було створено, це зовнішній ключ з моделі Useraccount, для чого використано тип ForeignKey;

– status – стан уточнення від одного орендодавця до іншого, яке стосується орендаря, який раніше проживав у орендодавця-адресата цього уточнення;

– createdat – час створення уточнення, що реалізовано типом CharField.

Реалізація моделі для роботи над критеріями, визначеними орендодавцем в уточненні, забезпечена через клас `Landlordadvicecriterion` з полями:

- `landlordadvice` – уточнення, яке створено від одного орендодавця до іншого, стосується орендаря, який раніше проживав у орендодавця-адресата цього уточнення, та для якого було визначено цей критерій, це зовнішній ключ з моделі `Landlordadvice`, для чого використано тип `ForeignKey`;
- `criterion` – критерій, який був визначений в уточненні, це зовнішній ключ з моделі `Criterion`, для чого використано тип `ForeignKey`;
- `criterionvalue` – значення за цим критерієм, яке було встановлено орендодавцем як відповідь на уточнення, що реалізовано типом `CharField`.

Реалізація моделі для роботи над додатковими критеріями, визначеними орендодавцем в уточненні, забезпечена через клас `Additlandlordadvicecriterion` з полями:

- `landlordadvice` – уточнення, яке створено від одного орендодавця до іншого, стосується орендаря, який раніше проживав у орендодавця-адресата цього уточнення, та для якого було визначено цей додатковий критерій, це зовнішній ключ з моделі `Landlordadvice`, для чого використано тип `ForeignKey`;
- `criterionname` – сам додатковий критерій (запитання або назва), який був визначений в уточненні, що реалізовано типом `CharField`;
- `criterionvalue` – значення за цим додатковим критерієм, яке було встановлено орендодавцем як відповідь на уточнення, що реалізовано типом `CharField`.

Реалізація моделі для роботи над відгуком орендаря про кімнату, яку він винаймав, забезпечена через клас `Roomreview` з полями:

- `tenant` – орендар, який створив цей відгук, це зовнішній ключ з моделі `Useraccount`, для чого використано тип `ForeignKey`;

- score – оцінка, на яку загалом можна оцінити оренду цієї кімнати за особистим досвідом орендаря, що реалізовано типом PositiveSmallIntegerField;

- roomadvertisement – оголошення про оренду кімнати, до якого створено цей запит, це зовнішній ключ з моделі Roomadvertisement, для чого використано тип ForeignKey;

- reviewmessage – враження від оренди цієї кімнати орендарем, представлене текстом на основі поля CharField;

- createdat – час створення відгуку, що реалізовано типом CharField.

Реалізація моделі для роботи над відгуком орендодавця про орендаря, який винаймав у нього кімнату, забезпечена через клас Tenantreview з полями:

- landlord – орендодавець, який створив цей відгук, це зовнішній ключ з моделі Useraccount, для чого використано тип ForeignKey;

- score – оцінка, на яку загалом можна оцінити орендаря, який винаймав у орендодавця кімнату, що реалізовано типом PositiveSmallIntegerField;

- tenant – орендар, про якого створив відгук цей орендодавець, це зовнішній ключ з моделі Useraccount, для чого використано тип ForeignKey;

- reviewmessage – враження від орендаря, який винаймав у цього орендодавця кімнату, представлене текстом на основі поля CharField;

- createdat – час створення відгуку, що реалізовано типом CharField.

Реалізація моделі для роботи над короткими повідомленнями між орендарями і орендодавцями забезпечена через клас Shortmessage з полями:

- useraccount – користувач, який створив коротке повідомлення, це зовнішній ключ з моделі Useraccount, для чого використано тип ForeignKey;

- useraccountto – користувач, якому направлено це коротке повідомлення, це зовнішній ключ з моделі Useraccount, для чого використано тип ForeignKey;

- message – текст короткого повідомлення, що реалізовано типом CharField;
- createdat – час створення короткого повідомлення, що реалізовано типом CharField;
- read – мітка про те, чи прочитане це коротке повідомлення, чи ні, що реалізовано типом BooleanField.

До функцій представлень за MVT належать зокрема наступні.

Пошук кімнати для подальшої оренди створено представленням на основі функції `roomadvertisement_find`, параметр якої `request` містить всі дані, необхідні для роботи, зокрема `request.user` – запис авторизованого користувача, якщо користувач авторизувався, `request.GET` – дані запиту, серед яких як зі словника виділяються параметри пошуку, зокрема включаючи такі: `roomtype` – тип кімнати, `scoreabove` – мінімальна оцінка, виставлена кімнаті, `furniture` – необхідність наявності меблів (значення по визначає обов’язкову відсутність меблів, інше значення – наявність, а відсутність такого параметра у запиті – те, що будь-який варіант підходить), `desk` – наявність поверхні або столу для роботи, `sleepingtype` – тип спального місця, `wardrobe` – наявність шафи, `sockets` – мінімальна кількість розеток, `availabled` – день, з якого потрібна оренда кімнати (`availablem` – місяць, `availabley` – рік), `roomprice` – максимальна вартість оренди, `tenantage` – вік орендаря, `rentmonths` – кількість місяців, на яку має бути орендовано кімнату, `deposit` – вимога до відсутності депозиту, `roomsmoking` – потреба в тому, щоб курити в кімнаті, `roompets` – наявність у орендаря домашніх тварин, `roomreference` – наявність або відсутність вимоги до наявності рекомендації від попереднього орендодавця у орендаря, `roomgender` – вимоги щодо статі орендаря у програмі (всі, тільки чоловіки, тільки жінки, чоловіча стать орендаря, жіноча стать орендаря), `roomossupration` – зайнятість орендаря, `roomcouples` – потреба в оренді кімнати парою людей, `proptype` – тип помешкання, в якому знаходиться кімната, `housemates` – максимальна кількість людей у помешканні, поруч з якими готовий жити орендар,

housematesabove – мінімальна така кількість, proproomsfrom – мінімальна кількість кімнат у помешканні, proproomsto – максимальна кількість кімнат у помешканні, rentedroomsfrom – мінімальна кількість кімнат, яка вже орендується у помешканні, rentedroomsto – максимальна кількість кімнат, яка вже орендується в помешканні, propageto – максимальний вік людей, які вже живуть у помешканні, propagefrom – мінімальний вік людей, які вже живуть у помешканні, propsmoking – прийнятність того, що в помешканні живуть люди, які палять, proppets – прийнятність наявності домашніх тварин у помешканні, propbalcony – вимога до наявності балкону в помешканні, propterrace – вимога до наявності саду або тераси в помешканні, propkitchen – вимога до наявності кухні в помешканні, propdis – вимога до надання доступу до приміщення людей з інвалідністю, proppark – вимога до можливості паркувати автомобіль поруч, propgarage – наявність гаража біля приміщення, propinternets – перелік варіантів доступу до інтернету, які потрібні і прийнятні, propbaths – перелік прийнятних типів ванної кімнати, propregion – район, в якому має знаходитись приміщення, в якій знаходиться кімната, а proplocation – відповідно населений пункт, propcount – країна, propcode – поштовий індекс, proplanguages – перелік мов, однією з яких має володіти хтось з тих, хто проживає у цьому помешканні, propsscups – вимоги до професій тих, хто проживає у помешканні, adorder – критерій для сортування результатів. Отримані результати також розширюються за рахунок додавання інформації до кожного оголошення щодо того, чи зберіг авторизований орендар, якщо він авторизувався, це оголошення, тобто створив собі закладку. Окрім того перевіряється початково встановлення прапорця adset щодо того, щоб встановити значення більшої частини з цих параметрів з останнього оголошення про пошук кімнати для оренди, яке створив авторизований орендар.

Пошук орендаря, який бажає орендувати кімнату, створено представленням на основі функції tenantadvertisement_find, що організована у

відповідності з попереднім пошуком, але при цьому перелік параметрів відповідає наявній інформації у оголошеннях про пошук кімнат для оренди.

Створення запиту на оренду кімнати реалізовано представленням на основі функції `new_tenantrequest`, номер оголошення про оренду кімнати передається через параметр `roomadvertisement`, а інші дані встановлюються через форму, що забезпечено класом `TenantrequestForm`.

Скасування запиту на оренду кімнати створено представленням на основі функції `cancel_tenantrequest`, номер якого передається через параметр `tenantrequest`, а у функції виконується перевірка того, що запит був поданий цим авторизованим користувачем як орендарем або що запит був адресований щодо кімнати, яка виставлена для оренди авторизованим зараз орендодавцем.

Перегляд переліку створених запитів на оренду кімнати створено представленням на основі функції `created_tenantrequest`, а дані про авторизованого орендаря, яким і мають бути створені запити, витягаються зі стандартного параметра `request`, який присутній у всіх представленнях.

Перегляд запиту на оренду кімнати створено представленням на основі функції `look_tenantrequest`, номер якого передається через параметр `tenantrequest`, а у функції виконується перевірка того, що запит був поданий цим авторизованим користувачем як орендарем або що запит був адресований щодо кімнати, яка виставлена для оренди авторизованим зараз орендодавцем.

Підтвердження запиту за оголошенням про пошук кімнати створено представленням на основі функції `confirm_landlordrequest`, номер якого передається через параметр `landlordrequest`, а у функції виконується перевірка того, що запит був адресований щодо оголошення про пошук кімнати, яке створене авторизованим орендарем.

Скасування запиту за оголошенням про пошук кімнати для оренди створено представленням на основі функції `cancel_landlordrequest`, номер якого передається через параметр `landlordrequest`, а у функції виконується

перевірка того, що запит був адресований щодо оголошення про пошук кімнати, яке створене авторизованим орендарем, або створений цим авторизованим орендодавцем.

Перегляд запиту за оголошенням про пошук кімнати створено представленням на основі функції `look_landlordrequest`, номер якого передається через параметр `landlordrequest`, а у функції виконується перевірка того, що запит був поданий цим авторизованим користувачем як орендодавцем або що запит був адресований щодо оголошення про пошук кімнати, яке створено авторизованим зараз орендарем.

Перегляд переліку адресованих запитів за оголошенням про пошук кімнати створено представленням на основі функції `addressed_landlordrequest`, а дані про авторизованого орендаря, якому і мають бути адресовані запити, витягаються зі стандартного параметра `request`.

Перегляд даних житла, що включає сценарій перегляду всіх кімнат, які доступні для оренди за житлом створено представленням на основі функції `look_sharedproperty`, номер якого передається через параметр `sharedproperty`.

Оформлення відгуку за результатами оренди кімнати про кімнату створено представленням на основі функції `new_roomreview`, а номер кімнати визначається параметром `roomad`, а всі інші дані задаються формою `RoomreviewForm`.

Перегляд відгуків за кімнатою створено представленням на основі функції `look_roomreviews`, а номер кімнати визначається параметром `roomad`.

Перегляд профільної сторінки орендодавця створено представленням на основі функції `landlord_page`, а логін орендодавця передається через параметр `landlord`. Перегляд профільної сторінки орендаря створено представленням на основі функції `tenant_page`, а логін орендодавця передається через параметр `tenant`.

Створення оголошення про пошук кімнати для оренди реалізовано представленням на основі функції `new_tenantadvertisement`, а дані

користувача вибираються за авторизованим обліковим записом, при цьому в основі використовується клас форми `TenantadvertisementForm`.

Редагування оголошення про пошук кімнати створено представленням на основі функції `edit_tenantadvertisement`, де номер оголошення про пошук кімнати задається параметром `tenantadvertisement`, перевіряючи, що це оголошення створено користувачем, який авторизувався, як орендарем.

Зняття з розгляду оголошення про пошук кімнати створено представленням на основі функції `cancel_tenantadvertisement`, де номер оголошення про пошук кімнати задається параметром `tenantadvertisement`, перевіряючи, що це оголошення створено користувачем, який авторизувався, як орендарем.

Перегляд оголошення про пошук кімнати створено представленням на основі функції `look_tenantadvertisement`, де номер оголошення про пошук кімнати задається параметром `tenantadvertisement`.

Перегляд переліку всіх власних оголошень про пошук кімнати створено представленням на основі функції `my_tenantadvertisements`, а самі дані користувача обираються з даних авторизації з параметра `request`.

Перегляд оголошення про оренду кімнати створено представленням на основі функції `look_roomadvertisement`, де номер оголошення про оренду кімнати задається параметром `roomadvertisement`.

Отримання всіх коротких повідомлень, відправлених користувачем, організовано разом з відправленням короткого повідомлення представленням на основі функції `short_messages`, а логін користувача, з яким має відбуватися комунікація визначається параметром `useraccount_login`. У функції витягаються всі повідомлення зі спілкування авторизованого користувача з цим користувачем, що задано у параметрі, а також підтримується форма на основі класу `ShortmessageForm` для створення повідомлення.

Отримання комунікацій користувача створено представленням на основі функції `short_message_communications`, а вибір комунікацій

відбувається за користувачем, який авторизувався. Так формується перелік тих користувачів, з якими спілкувався користувач.

Створення закладки на кімнаті для оренди створено представленням на основі функції `new_roomadvertisementsave`, де номер оголошення про оренду кімнати задається параметром `roomadvertisement`. Вилучення закладки на кімнаті для оренди створено представленням на основі функції `delete_roomadvertisementsave`, де номер оголошення про оренду кімнати задається параметром `roomadvertisement`, перевіряючи авторство цієї закладки авторизованим користувачем.

Перегляд закладок кімнат для оренди створено представленням на основі функції `roomadvertisementsaves`, а користувач, для якого ці закладки демонструються, визначається даними авторизації з параметра `request`.

Реалізація моделі прогнозування вартості оренди кімнати для проживання виконана за допомогою класу `XGBRegressor` з пакету `xgboost` [12].

4.3 Висновки за розділом 4

Програмне забезпечення повністю реалізовано, для чого за шаблоном проєктування MVT у Django-проєкті було розроблено необхідні представлення, моделі, створено модель прогнозування вартості оренди кімнати.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис програми

Програма забезпечує необхідні засоби для пошуку орендарями кімнат для оренди, а орендодавцями орендарів, враховуючи зокрема індивідуальні характеристики, щоб забезпечити створення комунікації щодо подальшої оренди, відбір кандидатів для оренди власної кімнати. У підсумку це дозволяє людям, які бажають винайняти кімнату, знайти власного орендодавця, а людям, які готові здати в оренду кімнату у власній квартирі або будинку, знайти орендаря для цієї кімнати або кімнат.

5.2 Експлуатація програми

Виконання взаємодії з програмою інтуїтивно зрозуміле і відповідає функціональній схемі з розділу 4, розпочинаючи стандартно роботу з пошуку оголошень про оренду кімнати, встановлюючи наступні фільтри пошуку: тип кімнати, мінімальну оцінку, виставлену кімнаті, необхідність наявності меблів, наявність поверхні або столу для роботи, тип спального місця, наявність шафи, мінімальну кількість розеток, день, з якого потрібна оренда кімнати, максимальну вартість оренди, вік орендаря, кількість місяців, на яку має бути орендовано кімнату, вимогу до відсутності депозиту, потребу в тому, щоб курити в кімнаті, наявність у орендаря домашніх тварин, наявність або відсутність вимоги до наявності рекомендації від попереднього орендодавця у орендаря, вимоги щодо статі орендаря у програмі (всі, тільки чоловіки, тільки жінки, чоловіча стать орендаря, жіноча стать орендаря), зайнятість орендаря, потребу в оренді кімнати парою людей, тип помешкання, в якому знаходиться кімната, максимальну кількість людей у помешканні, поруч з якими готовий жити орендар, мінімальну таку кількість, мінімальну кількість кімнат у помешканні, максимальну кількість кімнат у

помешканні, мінімальну кількість кімнат, яка вже орендується у помешканні, максимальну кількість кімнат, яка вже орендується в помешканні, максимальний вік людей, які вже живуть у помешканні, мінімальний вік людей, які вже живуть у помешканні, прийнятність того, що в помешканні живуть люди, які палять, прийнятність наявності домашніх тварин у помешканні, вимогу до наявності балкону в помешканні, вимогу до наявності саду або тераси в помешканні, вимогу до наявності кухні в помешканні, вимогу до надання доступу до приміщення людей з інвалідністю, вимогу до можливості паркувати автомобіль поруч, наявність гаража біля приміщення, перелік варіантів доступу до інтернету, які потрібні і прийнятні, перелік прийнятних типів ванної кімнати, район, в якому має знаходитись приміщення, в якому знаходиться кімната, відповідно населений пункт, країна, поштовий індекс, перелік мов, однією з яких має володіти хтось з тих, хто проживає у цьому помешканні, вимоги до професій тих, хто проживає у помешканні, критерій для сортування результатів.

З результатів пошуку, натиснувши на назву або зображення кімнати, можна відкрити сторінку оголошення про оренду кімнати (рис. 5.1). Оголошення містить основну фотографію кімнати згори, а внизу сторінки є інші фотографії, віднесені до цієї кімнати, відгуки про кімнату з посиланням для відкриття всіх відгуків. Якщо навести курсор на фотографію кімнати, то з'явиться іконка для створення закладки. Під описом кімнати знаходиться посилання для переходу до створення запиту на оренду кімнати. Нижче знаходиться інформація зліва про помешкання, в якому розташовано кімнату, але включаючи тільки основну інформацію, при цьому натискаючи на назву помешкання, можна відкрити сторінку з даними помешкання, інформацією про власника. У іншій частині сторінки знаходиться інформація вже про саму кімнату.

Подібний пошук організований для пошуку оголошень від самих орендарів, а з нього можна перейти до оголошення про пошук кімнати для оренди (рис. 5.2).

RoomPersonaler

Кімнати ▾ Орендарі ▾ Керування ▾

Вийти



Кімната в будинку з грифонами

Міфічні грифони, що прикрашають будинок, роблять його містичним, але атмосфера кімнати зробить її для вас затишним місцем. В нас вдома тихо, можна спокійно працювати, насолоджуватись життям. Будемо раді цікавим людям та проводити іноді час разом.

Створити запит →

Квартира з грифонами

Тип житла

Квартира

Місто

Запоріжжя

Ванна кімната

Душ

5.0

Оцінка

Мінімальний вік

20

Максимальний вік

50

Мінімальна тривалість, місяців

3

Вартість

6000 ₴

Меблі є

Можна з домашніми тваринами

Шафа є

рекомендації не потрібні

Палити не можна

Парам не здається

Рисунок 5.1 – Оголошення про оренду кімнати

RoomPersonaler

Кімнати ▾ Орендарі ▾ Керування ▾

Вийти



Тарас Жеведь

Живу динамічно, тому вдома не дуже багато часу проводжу. Робота, зустрічі, активності - тож я не дуже буду вам заважати. А точніше зовсім не буду! Я шукаю дуже зручну кімнату в центрі міста. І буду радий вашим пропозиціям!

Створити запит →

5.0

Оцінка

Тип житла

Квартира

Тип кімнати

Спальня і кабінет

Мінімальна оцінка

4.5

Вік

22

Мінімальна тривалість, місяців

6

Максимальна тривалість, місяців

36

Меблі

необхідні

Стіл

необхідний

Вітальня

не обов'язково

Рисунок 5.2 – Оголошення про пошук кімнати для оренди

Оголошення про пошук кімнати для оренди також при наведенні курсором на фотографію орендаря у випадку, якщо працює з програмою орендодавець, містить іконку з закладкою.

Для того щоб дістатися сторінки пошуку оголошень від самих орендарів, потрібно використати меню. Для цього використовується розділ “Орендарі”. Він містить підрозділи “Пошук” для виконання пошуку, “Запити” для перегляду запитів, адресованих цьому орендарю, який авторизувався, “Закладки” для відтворення закладок по оголошенням від орендарів, але тільки для орендодавців, “Оголошення” для переходу до переліку власних оголошень, створених орендарем, який авторизувався, “Створити” для створення нового оголошення про оренду кімнати.

Розділ “Кімнати” містить підрозділи “Пошук” для пошуку серед оголошень про оренду кімнат, “Запити” для перегляду запитів (як своїх, так і адресованих – у такому випадку потрібно обрати кімнату свою спочатку) до оголошень про оренду кімнат, “Закладки” для відтворення закладок по оголошенням про оренду кімнат, “Оголошення” для переходу до переліку власних оголошень, створених орендодавцем, який авторизувався, “Створити” для створення нового оголошення про пошук кімнати для оренди.

Якщо користувач не володіє відповідними правами, тобто не є орендодавцем, то використати функцію, яка доступна орендодавцям, він не зможе. Однією з таких функцій є створення оголошення про оренду кімнат. Для цього спочатку орендодавець має внести дані власного житла, а тоді вже для нього натиснути на посилання додати кімнату. Тоді відкриється сторінка, яка дозволяє визначити параметри кімнати та виконати прогнозування вартості оренди кімнати (рис. 5.3). Для прогнозування частина параметрів береться з даних житла, а не кімнати. Після встановлення значень, натиснувши на кнопку “Обчислити вартість”, можна виконати прогнозування. Кнопка “Створити” дозволяє зберегти оголошення.

RoomPersonaler Кімнати ▾ Орендарі ▾ Керування ▾ [Вийти](#)

Додавання кімнати для оренди

Модернова квартира на Франка
Квартира, до якої належить кімната

Назва кімнати
Бірюзовий спокій

Тип кімнати
Спальня і кабінет

Меблі
Наявні

Стіл для роботи
Присутній

Шафа
Наявна

Кількість розеток
5

Зайнятість
Студент

Вимога до рекомендації
Відсутня

Вимога до статі орендаря
Переважно жінки

Обмеження куріння
Так

Можливість оренди паром
Відсутня

Мінімальний вік
18

Максимальний вік
25

Мінімальний строк оренди
3

Максимальний строк оренди
12

Можна заїхати з
11.12.2023

Тип спального місця
Ліжко

Тільки для вегетаріанців
Ні

Наявність домашніх тварин
Ні

Сума депозиту
30000

Вартість місяця оренди

Фотографії: [Додати файл](#)

Опис

Вартість за такий варіант може бути 29787 ₴

[Обчислити вартість](#) [Створити](#)

Рисунок 5.3 – Створення оголошення про оренду кімнати

Розділ “Керування” містить підрозділи “Запити” з запитами до орендарів, “Перевірка” з запитами до орендодавців, у яких раніше потрібний орендар орендував житло (створити можна зі сторінки з інформацією про орендодавця і його відгуком також), “Порівняння” для порівняння орендарів між собою, підрозділ “Житло” з внесеними записами про власне житло, підрозділ “Профіль” для перегляду і зміни власних даних, “Повідомлення” для перегляду всіх комунікацій користувача (відправити повідомлення можна зі сторінки з інформацією про користувача).

На сторінці порівняння (рис. 5.4) відображаються орендарі, які подали запити на кімнати власника, вилучити їх можна, натискаючи на іконку, яка спливає при наведенні курсора на фотографію орендаря. Звідси також можна одразу підтвердити або скасувати запит цього орендаря. За кожним орендарем можна задавати різні критерії, вводити відповідні значення. Якщо це текстове значення, то критерій відображається помаранчевим. Якщо числове (тобто текстом задане число), то воно має знаходитися в інтервалі

від 0 до 10, тоді значення до 6 включно показуються червоні, від 7 до 8 – помаранчевим, 9 і 10 – зеленим. Кожне значення можна змінити, для чого надається відповідне посилання біля критерію. Внизу розташовані поля для додавання нових критеріїв. Тоді вводиться назва критерію. Значення підказуються при цьому з переліку, що випадає, на основі стандартних критеріїв. У другому полі потрібно задати значення критерію. Коли ці значення вводяться, активується кнопка зберегти. Якщо вводиться нове ім'я критерію, то він зберігається як додатковий.

RoomPersonaler Кімнати ▾ Орендарі ▾ Керування ▾ [Вийти](#)

Порівняння кандидатів на оренду

Іван Перверзев

[підтвердити запит](#) [скасувати запит](#)

Комунікабельність	
змінити	10
Бюджет	
змінити	7
Професія	
змінити	організатор масових заходів
Захоплення	
змінити	8
Вимоги	
змінити	7

[Введіть критерій](#) [Введіть значення](#)

Тарас Жеведь

[підтвердити запит](#) [скасувати запит](#)

Комунікабельність	
змінити	8
Бюджет	
змінити	10
Професія	
змінити	менеджер
Захоплення	
змінити	9
Вимоги	
змінити	9

[Введіть критерій](#) [Введіть значення](#)

Мартін Петков

[підтвердити запит](#) [скасувати запит](#)

Комунікабельність	
змінити	9
Бюджет	
змінити	9
Професія	
змінити	фотограф
Захоплення	
змінити	9
Вимоги	
змінити	6

[Введіть критерій](#) [Введіть значення](#)

Рисунок 5.4 – Порівняння кандидатів для оренди кімнати

5.3 Експериментальне дослідження

Для перевірки методу прогнозування вартості оренди кімнати для проживання проведено експериментальне дослідження, в основі якого використано вибірку даних [13], що розміщена у відкритому доступі. Ця

вибірка включає дані, зібрані стосовно оренди кімнат у Великобританії (переважно у великих містах) і характеризуються 4133 записами. Ці записи було оброблено, вилучаючи з них ті, які стосуються оренди будинків, квартир. Вони були віднесені як до оренд всього будинку, квартири, адже це варіанти, коли вся квартира або будинок знімаються співмешканцями. Такі варіанти не розглядалися для уникнення викривлення. Було створено дерево рішень і на його основі визначено інформативність ознак кожної ознаки з вибірки (рис. 5.5).

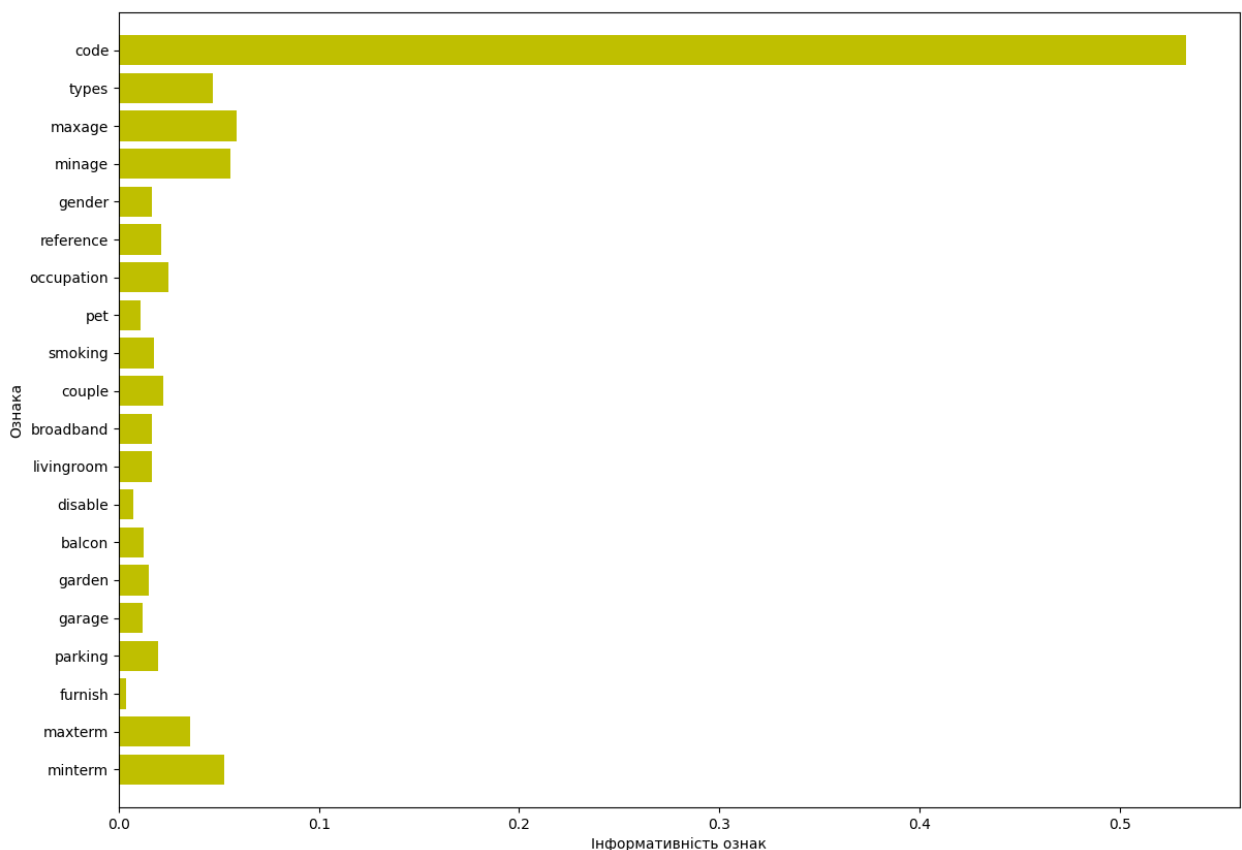


Рисунок 5.5 – Інформативність ознак вибірки даних при побудові моделі

Усі дані були оброблені і у випадку відсутності значень за деякими характеристиками було використано стандартне за цією характеристикою (наприклад, вважається, що рекомендація для оренди не потрібна, якщо значення пропущено).

Спочатку було проведено порівняння моделей на основі лінійної регресії, дерева рішень, XGBoost за використання всіх вхідних ознак у моделі (табл. 5.1).

Таблиця 5.1 – Результати дослідження за використання всіх вхідних ознак у моделі

Модель	Кількість дерев	Глибина дерева	Середньоквадратична похибка
Лінійна регресія	—	—	0,044624
Дерево рішень	1	20	0,04509
Дерево рішень	1	10	0,047664
Дерево рішень	1	8	0,046845
XGBoost	50	6	0,036499
XGBoost	50	8	0,035438
XGBoost	50	20	0,038259
XGBoost	100	6	0,035637
XGBoost	100	8	0,0355
XGBoost	100	20	0,038259
XGBoost	500	6	0,035576
XGBoost	500	8	0,035464
XGBoost	500	20	0,038259
XGBoost	1000	6	0,035576
XGBoost	1000	8	0,035464
XGBoost	1000	20	0,038259

Наступним було проведено порівняння моделей на основі лінійної регресії, дерева рішень, XGBoost за використання мінімального строку і віку, максимального строку і віку, типу помешкання та індексу в якості вхідних ознак моделі (табл. 5.2). Тобто це відібрані ознаки, які мають найбільшу інформативність.

Таблиця 5.2 – Результати дослідження за використання мінімального строку і віку, максимального строку і віку, типу помешкання та індексу в якості вхідних ознак моделі

Модель	Кількість дерев	Глибина дерева	Середньоквадратична похибка
Лінійна регресія	—	—	0,046565
Дерево рішень	1	20	0,043433
Дерево рішень	1	10	0,044338
Дерево рішень	1	8	0,043229
XGBoost	50	6	0,040053
XGBoost	50	8	0,038839
XGBoost	50	20	0,040493
XGBoost	100	6	0,040113
XGBoost	100	8	0,039158
XGBoost	100	20	0,040493
XGBoost	500	6	0,040538
XGBoost	500	8	0,039371
XGBoost	500	20	0,040493
XGBoost	1000	6	0,040538
XGBoost	1000	8	0,039371
XGBoost	1000	20	0,040493

Наступним було проведено порівняння моделей на основі лінійної регресії, дерева рішень, XGBoost за використання мінімального строку і віку, максимального строку і віку, типу помешкання, індексу, зайнятості, рекомендацій та статі в якості вхідних ознак моделі (табл. 5.3). У даному випадку до найбільш інформативних ознак додано ознаки з більшою інформативністю.

Таблиця 5.3 – Результати дослідження за використання мінімального строку і віку, максимального строку і віку, типу помешкання, індексу, зайнятості, рекомендацій та статі в якості вхідних ознак моделі

Модель	Кількість дерев	Глибина дерева	Середньоквадратична похибка
Лінійна регресія	—	—	0,046492
Дерево рішень	1	20	0,047303
Дерево рішень	1	10	0,043812
Дерево рішень	1	8	0,042998
XGBoost	50	6	0,038016
XGBoost	50	8	0,037867
XGBoost	50	20	0,039413
XGBoost	100	6	0,037379
XGBoost	100	8	0,037768
XGBoost	100	20	0,039413
XGBoost	500	6	0,037637
XGBoost	500	8	0,037851
XGBoost	500	20	0,039413
XGBoost	1000	6	0,037637
XGBoost	1000	8	0,037851
XGBoost	1000	20	0,039413

Останнім було проведено порівняння моделей за використання мінімального строку і віку, максимального строку і віку, типу помешкання, індексу, зайнятості, рекомендацій, статі, паркування, гаража, саду, балкону, вітальні, інтернету, пари, куріння, домашніх тварин в якості вхідних ознак моделі (табл. 5.4). У даному випадку до тільки інформативні ознаки з інформативністю, що ледь перевищує нульові значення, були відкинуті.

Таблиця 5.4 – Результати дослідження за використання мінімального строку і віку, максимального строку і віку, типу помешкання, індексу, зайнятості, рекомендацій, статі, паркування, гаража, саду, балкону, вітальні, інтернету, пари, куріння, домашніх тварин в якості вхідних ознак моделі

Модель	Кількість дерев	Глибина дерева	Середньоквадратична похибка
Лінійна регресія	—	—	0,044644
Дерево рішень	1	20	0,048673
Дерево рішень	1	10	0,046737
Дерево рішень	1	8	0,04418
XGBoost	50	6	0,035562
XGBoost	50	8	0,035624
XGBoost	50	20	0,037634
XGBoost	100	6	0,035042
XGBoost	100	8	0,035248
XGBoost	100	20	0,037634
XGBoost	500	6	0,034695
XGBoost	500	8	0,035248
XGBoost	500	20	0,037634
XGBoost	1000	6	0,034695
XGBoost	1000	8	0,035248
XGBoost	1000	20	0,037634

Найкращий результат для лінійної регресії було отримано за використання всіх вхідних ознак у моделі. Відповідно найкраще значення середньоквадратичної похибки за використання одного дерева рішення було отримано за використання мінімального строку і віку, максимального строку і віку, типу помешкання, індексу, зайнятості, рекомендацій та статі в якості вхідних ознак моделі (табл. 5.3). Для методу XGBoost найкраще значення отримано за використання мінімального строку і віку, максимального строку

і віку, типу помешкання, індексу, зайнятості, рекомендацій, статі, паркування, гаража, саду, балкону, вітальні, інтернету, пари, куріння, домашніх тварин в якості вхідних ознак моделі (табл. 5.4). Це варіант, який отримано за запропонованим методом. Отримане значення середньоквадратичної похибки за запропонованим методом на 22,25 % краще ніж значення, отримане за лінійною регресією, і на 19,31 % краще ніж значення, отримане за допомогою використання окремого дерева рішень.

5.4 Висновки за розділом 5

Усі сценарії використання програми було реалізовано, а експериментальне дослідження проведено на основі вибірки, що включає дані, зібрані стосовно оренди кімнат у Великобританії (переважно у великих містах). Отримане значення середньоквадратичної похибки за запропонованим методом виявилось на 22,25 % краще ніж значення, отримане за лінійною регресією, і на 19,31 % краще ніж значення, отримане за допомогою використання окремого дерева рішень.

ВИСНОВКИ

Виконання даної дипломної кваліфікаційної роботи призвело до розробки програмного забезпечення оренди кімнат для проживання, що в свою чергу дозволило задовольнити вимоги орендодавців до отримання індивідуальної інформації про орендарів.

Оренда кімнат в квартирі призводить до спільного проживання, а відповідно вимагає встановлення певних обмежень, вимог до орендаря, для чого необхідно визначити індивідуальні характеристики орендарів. У роботі виділено програми, які враховують особливості оренди кімнат і надають такі можливості, до яких віднесено SpareRoom і iBilik. На відміну від них виконана розробка дозволяє визначати індивідуальні захоплення, пошук за ними, порівнювати потенційних орендарів для прийняття рішення про вибір орендаря, прогнозувати вартість оренди, створювати оголошення про пошук кімнат самими користувачами як потенційними орендарями.

При реалізації програми використано засоби веброзробки, включаючи мови Python, HTML, CSS, фреймворк Django, засоби машинного навчання на основі Python-пакетів, інструмент для роботи з базою даних MySQL.

Програма в підсумку забезпечує необхідні засоби для пошуку орендарями кімнат для оренди, а орендодавцями орендарів, враховуючи зокрема індивідуальні характеристики, щоб забезпечити створення комунікації щодо подальшої оренди, відбір кандидатів для оренди власної кімнати, надаючи при цьому можливість прогнозувати вартість оренди на основі індивідуальних характеристик орендаря.

Наукова новизна роботи полягає в створеному методі прогнозування вартості оренди кімнати для проживання, який базується на застосуванні ансамблів дерев рішень на основі принципів бустингу і який дозволяє врахувати індивідуальні характеристики орендаря, додаючи їх до ознак створеної моделі прогнозування, що визначають загальні характеристики житла і оренди.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. SpareRoom for flatshare, house share, flat share & rooms for rent [Electronic resource]. – Access mode : <https://www.spareroom.co.uk/>.
2. iBilik – Find Room For Rent – Homestay For Rent in Malaysia [Electronic resource]. – Access mode : <https://www.ibilik.my/>.
3. Vacation Homes & Condo Rentals – Airbnb [Electronic resource]. – Access mode : <https://www.airbnb.com/>.
4. XGBoost – GeeksforGeeks [Electronic resource]. – Access mode : <https://www.geeksforgeeks.org/xgboost/>.
5. Yehoshua, R. XGBoost [Electronic resource] : The Definitive Guide (Part 1) / R. Yehoshua. – Access mode : <https://towardsdatascience.com/xgboost-the-definitive-guide-part-1-cc24d2dcd87a>.
6. What Is Django and What Is Django Used for? [Electronic resource]. – Access mode : <https://www.stxnnext.com/blog/what-is-django/>.
7. What is Django Python? Everything You Need to Know in 2022. Simplilearn [Electronic resource]. – Access mode : <https://www.simplilearn.com/tutorials/django-tutorial/what-is-django-python>.
8. Falcon – Full Stack Python [Electronic resource]. – Access mode : <https://www.fullstackpython.com/falcon.html>.
9. The Falcon Web Framework – Falcon 3.1.3 documentation [Electronic resource]. – Access mode : <https://falcon.readthedocs.io/en/stable/>.
10. All You Need to Know About Django MVT Architecture – TechVidvan [Electronic resource]. – Access mode : <https://techvidvan.com/tutorials/djangos-mvt-architecture/>.
11. Kider – Free Responsive HTML5 Bootstrap 5 Educational Website Template [Electronic resource]. – Access mode : <https://themewagon.com/themes/free-responsive-html5-bootstrap-5-preschool-website-template-kider/>.
12. Python API Reference — xgboost 2.0.2 documentation [Electronic

resource]. – Access mode : https://xgboost.readthedocs.io/en/stable/python/python_api.html.

13. Uk Room Prices [Electronic resource]. – Access mode : <https://www.kaggle.com/datasets/kwewqaweae/uk-room-prices>.

ДОДАТОК А
Технічне завдання

Вступ

Оренда кімнат для проживання є дуже особливою галуззю всередині широковідомої оренди житла. Це визначається тим, що власник житла зазвичай проживає разом з орендарем в одному житлі, а тому має певний ряд вимог, які мають бути забезпечені орендарем, щоб орендодавець довірив йому право орендувати кімнату. Велика частина з цих вимог є логічною і обумовленою тим, що має бути забезпечений необхідний рівень безпеки, довіри, має бути встановлена певна комунікація, а це потребує можливо навіть наявності спільних захоплень, поглядів. Тому програма, яка буде підтримувати ці процеси, вона має надати можливість отримувати необхідну для цього інформацію. Такі особливості комунікації визначають також, що і вартість оренди залежить від фактично індивідуальних характеристик орендаря. Тому розуміння вартості оренди за тих індивідуальних характеристик, які має орендар, є важливим, адже у такому випадку далеко не кожен користувач зможе орендувати кожную доступну і підходящу для нього кімнату. Тобто врахування індивідуальних характеристик орендаря важливо як під час визначення комунікації в програмі, так і під час прогнозування вартості оренди, щоб підібрати потім варіант за адекватну ціну.

A.1 Підстави для розробки

Наказ № 437 від 14 листопада 2023 р. за Національним університетом «Запорізька політехніка» з зазначенням теми роботи «Дослідження та програмна реалізація методів прогнозування вартості оренди кімнати для проживання» та завдання на роботу є підставами для розробки.

А.2 Призначення розробки

Програма має забезпечувати необхідні засоби для пошуку орендарями кімнат для оренди, а орендодавцями орендарів, враховуючи зокрема індивідуальні характеристики, щоб забезпечити створення комунікації щодо подальшої оренди, відбір кандидатів для оренди власної кімнати. У підсумку це має дозволити людям, які бажають винайняти кімнату, знайти власного орендодавця, а людям, які готові здати в оренду кімнату у власній квартирі або будинку, знайти орендаря для цієї кімнати або кімнат.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Забезпечення функціональних характеристик повинно вкладатися в наступний набір функцій:

- пошук кімнати для подальшої оренди;
- пошук орендаря, який бажає орендувати кімнату;
- створення запиту на оренду кімнати;
- створення запиту за оголошенням про пошук кімнати для оренди, створеним орендодавцем;
- керування запитом на оренду кімнати, в тому числі його скасування з обох сторін, підтвердження, переведення до кандидатського стану, перегляд самого запиту та переліку таких запитів адресованих або створених;
- керування запитом за оголошенням про пошук кімнати для оренди, створеним орендодавцем, в тому числі його скасування з обох сторін, підтвердження, перегляд самого запиту та переліку таких запитів адресованих або створених;
- перегляд всіх кімнат, які доступні для оренди за житлом;
- оформлення відгуку за результатами оренди кімнати про орендаря або про кімнату;

- перегляд відгуків за кімнатою;
- перегляд профільної сторінки орендодавця;
- перегляд профільної сторінки орендаря;
- створення оголошення про пошук кімнати для оренди;
- керування створеними оголошеннями про пошук кімнати для оренди, включаючи перегляд такого оголошення, редагування або зняття з розгляду, перегляд переліку всіх власних оголошень;
- створення оголошення про оренду кімнати;
- керування створеними оголошеннями про оренду кімнати, включаючи перегляд такого оголошення, редагування або зняття з розгляду, перегляд переліку всіх власних оголошень;
- внесення даних житла, в якому оренднуються кімнати;
- керування даними житла, в якому оренднуються кімнати, включаючи перегляд всіх збережених записів орендодавцем, редагування, вилучення за відсутності доданих кімнат, перегляд даних житла;
- порівняння кандидатів на оренду кімнати;
- відправлення короткого повідомлення від орендодавця орендарю або від орендаря орендодавцю;
- отримання всіх коротких повідомлень, відправлених між орендарем та орендодавцем;
- отримання комунікацій користувача;
- надсилання запиту на уточнення щодо орендаря орендодавцю;
- скасування запиту на уточнення щодо орендаря орендодавцю;
- створення відповіді на уточнення щодо орендаря орендодавцю;
- перегляд отриманих запитів на уточнення щодо орендаря;
- перегляд створених запитів на уточнення щодо орендаря;
- створення закладки на оголошенні про пошук кімнати для подальшої оренди;
- вилучення закладки на оголошенні про пошук кімнати для подальшої оренди;

- перегляд закладок оголошень про пошук кімнати для подальшої оренди;
- створення закладки на кімнаті для оренди;
- вилучення закладки на кімнаті для оренди;
- перегляд закладок кімнат для оренди;
- авторизація в програмі або реєстрація.

A.3.2 Умови експлуатації

Програмне забезпечення в підсумку має підтримуватись під час експлуатації через:

- організацію хостингу для роботи з сервером;
- розміщення на сервері вебзастосунку з базовим налаштуванням Django;
- наявність інтерпретатора python 3.

Або у складі цього серверу, або як окремий сервер БД має бути використано і налаштовано систему MySQL.

A.3.3 Вимоги до складу та параметрів технічних засобів

Вимоги до організації технічних засобів сервера включають мінімально процесор Dual Intel Xeon, який має частоту 3,2 ГГц, 4 Гб доступної оперативної пам'яті, 10 Гб доступної пам'яті на жорсткому диску сервера.

A.4 Порядок контролю та приймання

Строки виконання дипломної кваліфікаційної роботи магістра визначені в завданні на роботу безпосередньо, включаючи строк подання роботи, календарний план виконання роботи з зазначеними етапами,

результатами та строками.

Захист роботи відбувається за оголошеним завчасно графіком захисту дипломних кваліфікаційних робіт і відповідно з графіком роботи екзаменаційної комісії.

ДОДАТОК Б
Текст програми

```

from .models import Roomadvertisement
from .models import Roomtype
from .models import Propertytype
from .models import Locationregion
from .models import Location
from .models import Country
from .models import Language
from .models import Sharedpropertyoccupation
from .models import Sharedpropertycommunication
from .models import Occupation
from .models import Roomadvertisementsave
from .models import Tenantadvertisement
from .models import Useraccount
from .models import Tenantadvertisementregion
from .models import Tenantadvertisementsave
from .models import Userlanguage
from .models import Interest
from .models import Userinterest
from .models import Userinterestitem
from django.db.models import Q

def roomadvertisement_find(request):
    found_roomadvertisement_data = {}
    found_roomadvertisement_data['advertisements'] = []
    found_roomadvertisement_data['countries'] =
Country.objects.all()
    found_roomadvertisement_data['locations'] =
Location.objects.all()
    found_roomadvertisement_data['regions'] =
Locationregion.objects.all()
    found_roomadvertisement_data['propertytypes'] =
Propertytype.objects.all()
    found_roomadvertisement_data['occupations'] =
Occupation.objects.all()
    found_roomadvertisement_data['languages'] =
Language.objects.all()
    found_roomadvertisement_data['roomtypes'] =
Roomtype.objects.all()
    check_get = request.method == 'GET'
    start_form = True
    roomadvertisements =
Roomadvertisement.objects.filter(status=True)
    roomadvertisements =
roomadvertisements.select_related('roomtype')
    roomadvertisements =
roomadvertisements.select_related('sharedproperty')
    roomadvertisements =
roomadvertisements.select_related('sharedproperty__propertytype')

```

```

        roomadvertisements =
roomadvertisements.select_related('landlord')
        roomadvertisements =
roomadvertisements.select_related('sharedproperty__locationregion')
        roomadvertisements =
roomadvertisements.select_related('sharedproperty__locationregion__loc
ation')
        roomadvertisements =
roomadvertisements.select_related('sharedproperty__locationregion__loc
ation__country')
        tenant_properties = False
        if check_get and 'adset' in request.GET and request.user:
            tenant =
Useraccount.objects.get(registereduser=request.user)
            tenant_ad =
Tenantadvertisement.objects.filter(tenant=tenant)
            tenant_ad = tenant_ad.select_related('roomtype')
            tenant_ad = tenant_ad.select_related('tenant')
            tenant_ad = tenant_ad.select_related('propertytype')
            if tenant_ad and tenant_ad.count() > 0:
                tenant_ad = tenant_ad.values()[0]
                tenant_properties = True
                start_form = False
            if check_get and 'roomtype' in request.GET or
tenant_properties:
                if tenant_properties:
                    find_room_type = tenant_ad.roomtype
                else:
                    find_room_type = self.request.GET.get('roomtype')
                    find_room_type = int(find_room_type)
                    find_room_type = Roomtype.objects.get(id =
find_room_type)
                roomadvertisements =
roomadvertisements.filter(roomtype=find_room_type)
                start_form = False
                if check_get and 'scoreabove' in request.GET:
                    room_score_above = self.request.GET.get('scoreabove')
                    room_score_above = float(room_score_above)
                    roomadvertisements = roomadvertisements.filter
(score__gte=room_score_above)
                start_form = False
                if check_get and 'furniture' in request.GET and 'furniture'
!= 'no' or (tenant_properties and tenant_ad.furniture):
                    roomadvertisements = roomadvertisements.filter
(furniture=True)
                start_form = False
                elif check_get and 'furniture' in request.GET and
'furniture' == 'no':

```

```

        roomadvertisements = roomadvertisements.filter
(furniture=False)
        start_form = False
        if check_get and 'desk' in request.GET and 'desk' != 'no':
            roomadvertisements = roomadvertisements.filter
(desk=True)
            start_form = False
            elif check_get and 'desk' in request.GET and 'desk' == 'no'
or (tenant_properties and tenant_ad.desk):
                roomadvertisements = roomadvertisements.filter
(desk=False)
                start_form = False
                if check_get and 'sleepingtype' in request.GET or
tenant_properties:
                    if tenant_properties:
                        find_sleep_type = tenant_ad.sleepingtype
                    else:
                        find_sleep_type =
self.request.GET.get('sleepingtype')
                        find_sleep_type = int(find_sleep_type)
                        roomadvertisements = roomadvertisements.filter
(sleepingtype__gte=find_sleep_type)
                        start_form = False
                        if check_get and 'wardrobe' in request.GET and 'wardrobe' !=
'no' or (tenant_properties and tenant_ad.wardrobe):
                            roomadvertisements = roomadvertisements.filter
(wardrobe=True)
                            start_form = False
                            elif check_get and 'wardrobe' in request.GET and 'wardrobe'
== 'no':
                                roomadvertisements = roomadvertisements.filter
(wardrobe=False)
                                start_form = False
                                if check_get and 'sockets' in request.GET:
                                    find_sockets = self.request.GET.get('sockets')
                                    find_sockets = int(find_sockets)
                                    roomadvertisements = roomadvertisements.filter
(sockets__gte=find_sockets)
                                    start_form = False
                                    if check_get and 'availabled' in request.GET or
tenant_properties:
                                        if tenant_properties:
                                            find_available_at = tenant_ad.availableat
                                        else:
                                            available_at_day =
self.request.GET.get('availabled')
                                            available_at_day = int(available_at_day)
                                            available_at_month =
self.request.GET.get('availablem')

```

```

        available_at_month = int(available_at_month)
        available_at_year =
self.request.GET.get('availableley')
        available_at_year = int(available_at_year)
        import datetime
        find_available_at = datetime.date(available_at_year,
available_at_month, available_at_day)
        roomadvertisements =
roomadvertisements.filter(availableat__lte = find_available_at)
        start_form = False
        if check_get and 'roomprice' in request.GET or
tenant_properties:
            if tenant_properties:
                find_room_price = tenant_ad.budget
            else:
                find_room_price = self.request.GET.get('roomprice')
                find_room_price = int(find_room_price)
            roomadvertisements =
roomadvertisements.filter(pricemonth__lte=find_room_price)
            start_form = False
            if check_get and 'tenantage' in request.GET or
tenant_properties:
                if tenant_properties:
                    find_tenant_age = tenant_ad.age
                else:
                    find_tenant_age = self.request.GET.get('tenantage')
                    find_tenant_age = int(find_tenant_age)
                roomadvertisements =
roomadvertisements.filter(minage__lte=find_tenant_age,
maxage__gte=find_tenant_age)
                start_form = False
                if check_get and 'rentmonths' in request.GET:
                    find_room_months = self.request.GET.get('rentmonths')
                    find_room_months = int(find_room_months)
                    roomadvertisements =
roomadvertisements.filter(minmonths__lte=find_room_months,
maxmonths__gte=find_room_months)
                    start_form = False
                    if tenant_properties and tenant_ad.minmonths:
                        roomadvertisements =
roomadvertisements.filter(minmonths__lte = tenant_ad.minmonths,
maxmonths__gte = tenant_ad.minmonths)
                    if tenant_properties and tenant_ad.maxmonths:
                        roomadvertisements =
roomadvertisements.filter(minmonths__lte = tenant_ad.maxmonths)
                    if check_get and 'deposit' in request.GET:
                        roomadvertisements =
roomadvertisements.filter(deposit__gt=0)
                        start_form = False

```

```

        if check_get and 'roomsmoking' in request.GET and
'roomsmoking' != 'no':
            roomadvertisements = roomadvertisements.filter
(smoking=True)
            start_form = False
        elif check_get and 'roomsmoking' in request.GET and
'roomsmoking' == 'no':
            roomadvertisements = roomadvertisements.filter
(smoking=False)
            start_form = False
        if tenant_properties and tenant_ad.smoking:
            roomadvertisements = roomadvertisements.exclude
(smoking=False)
        if check_get and 'roomvegetarian' in request.GET and
'roomvegetarian' != 'no':
            roomadvertisements = roomadvertisements.filter
(vegetarian=True)
            start_form = False
        elif check_get and 'roomvegetarian' in request.GET and
'roomvegetarian' == 'no':
            roomadvertisements = roomadvertisements.filter
(vegetarian=False)
            start_form = False
        if tenant_properties and not tenant_ad.vegetarian:
            roomadvertisements = roomadvertisements.exclude
(vegetarian=True)
        if check_get and 'roompets' in request.GET and 'roompets' !=
'no':
            roomadvertisements = roomadvertisements.filter
(pets=True)
            start_form = False
        elif check_get and 'roompets' in request.GET and 'roompets'
== 'no':
            roomadvertisements = roomadvertisements.filter
(pets=False)
            start_form = False
        if tenant_properties and tenant_ad.pets:
            roomadvertisements = roomadvertisements.exclude
(pets=False)
        if check_get and 'roomreference' in request.GET and
'roomreference' != 'no':
            roomadvertisements = roomadvertisements.filter
(reference=True)
            start_form = False
        elif check_get and 'roomreference' in request.GET and
'roomreference' == 'no':
            roomadvertisements = roomadvertisements.filter
(reference=False)
            start_form = False

```

```

        if tenant_properties and not tenant_ad.reference:
            roomadvertisements = roomadvertisements.exclude
(reference=True)
        if check_get and 'roomgender' in request.GET:
            find_room_gender = self.request.GET.get('roomgender')
            find_room_gender = int(find_room_gender)
            if find_room_gender <= 2:
                roomadvertisements =
roomadvertisements.filter(gender=find_room_gender)
            elif find_room_gender == 3:
                roomadvertisements =
roomadvertisements.filter(Q(gender=0) | Q(gender=1))
            elif find_room_gender == 4:
                roomadvertisements =
roomadvertisements.filter(Q(gender=0) | Q(gender=2))
            start_form = False
        if tenant_properties and not tenant_ad.gender == 1:
            roomadvertisements = roomadvertisements.exclude
(gender=2)
        elif tenant_properties and not tenant_ad.gender == 2:
            roomadvertisements = roomadvertisements.exclude
(gender=1)
        if check_get and 'roomoccupation' in request.GET or
tenant_properties:
            if tenant_properties:
                find_room_occupation = tenant_ad.occupationtype
                roomadvertisements =
roomadvertisements.filter(occupationtype__lte=find_room_occupation)
            else:
                find_room_occupation =
self.request.GET.get('roomoccupation')
                find_room_occupation = int(find_room_occupation)
                roomadvertisements =
roomadvertisements.filter(occupationtype__gte=find_room_occupation)
            start_form = False
        if check_get and 'roomcouples' in request.GET and
'roomcouples' != 'no':
            roomadvertisements = roomadvertisements.filter
(couples=True)
            start_form = False
        elif check_get and 'roomcouples' in request.GET and
'roomcouples' == 'no':
            roomadvertisements = roomadvertisements.filter
(couples=False)
            start_form = False
        if tenant_properties and tenant_ad.couples:
            roomadvertisements = roomadvertisements.exclude
(couples=False)

```

```

        if check_get and 'proptype' in request.GET or
tenant_properties:
            if tenant_properties:
                find_property_type = tenant_ad.propertytype
            else:
                find_property_type =
self.request.GET.get('proptype')
                find_property_type = int(find_property_type)
                find_property_type = Propertytype.objects.get(id =
find_property_type)
                roomadvertisements =
roomadvertisements.filter(propertytype=find_property_type)
                start_form = False
            if check_get and 'housemates' in request.GET:
                find_housemates = self.request.GET.get('housemates')
                find_housemates = int(find_housemates)
                roomadvertisements =
roomadvertisements.filter(sharedproperty__housemates__lte=find_housema
tes)

                start_form = False
            if check_get and 'housematesabove' in request.GET:
                roomadvertisements =
roomadvertisements.filter(sharedproperty__housemates__gte=2)
                start_form = False
            if check_get and 'proproomsfrom' in request.GET:
                find_prop_rooms = self.request.GET.get('proproomsfrom')
                find_prop_rooms = int(find_prop_rooms)
                roomadvertisements =
roomadvertisements.filter(sharedproperty__totalrooms__gte=find_prop_ro
oms)

                start_form = False
            if check_get and 'proproomsto' in request.GET:
                find_prop_rooms = self.request.GET.get('proproomsto')
                find_prop_rooms = int(find_prop_rooms)
                roomadvertisements =
roomadvertisements.filter(sharedproperty__totalrooms__lte=find_prop_ro
oms)

                start_form = False
            if check_get and 'rentedroomsfrom' in request.GET:
                find_prop_rooms =
self.request.GET.get('rentedroomsfrom')
                find_prop_rooms = int(find_prop_rooms)
                roomadvertisements =
roomadvertisements.filter(sharedproperty__roomsrented__gte=find_prop_r
ooms)

                start_form = False
            if check_get and 'rentedroomsto' in request.GET:
                find_prop_rooms = self.request.GET.get('rentedroomsto')
                find_prop_rooms = int(find_prop_rooms)

```

```

        roomadvertisements =
roomadvertisements.filter(sharedproperty__roomsrented__lte=find_prop_r
ooms)
        start_form = False
        if check_get and 'propageto' in request.GET:
            find_prop_age = self.request.GET.get('propageto')
            find_prop_age = int(find_prop_age)
            roomadvertisements =
roomadvertisements.filter(sharedproperty__maxage__lte=find_prop_age)
            start_form = False
            if check_get and 'propagefrom' in request.GET:
                find_prop_age = self.request.GET.get('propagefrom')
                find_prop_age = int(find_prop_age)
                roomadvertisements =
roomadvertisements.filter(sharedproperty__minage__lte=find_prop_age)
                start_form = False
                if check_get and 'propsmoking' in request.GET and
'propsmoking' != 'no':
                    roomadvertisements = roomadvertisements.filter
(sharedproperty__smoking=True)
                    start_form = False
                    elif check_get and 'propsmoking' in request.GET and
'propsmoking' == 'no':
                        roomadvertisements = roomadvertisements.filter
(sharedproperty__smoking=False)
                        start_form = False
                        if tenant_properties and not tenant_ad.smokingwith:
                            roomadvertisements = roomadvertisements.exclude
(sharedproperty__smoking=True)
                            if check_get and 'proppets' in request.GET and 'proppets' !=
'no':
                                roomadvertisements = roomadvertisements.filter
(sharedproperty__petsinside=True)
                                start_form = False
                                elif check_get and 'proppets' in request.GET and 'proppets'
== 'no':
                                    roomadvertisements = roomadvertisements.filter
(sharedproperty__petsinside=False)
                                    start_form = False
                                    if tenant_properties and not tenant_ad.petswith:
                                        roomadvertisements = roomadvertisements.exclude
(sharedproperty__petsinside=True)
                                        if check_get and 'propmale' in request.GET and 'propmale' !=
'no':
                                            roomadvertisements = roomadvertisements.filter
(sharedproperty__maleslive__gt=0)
                                            start_form = False
                                            elif check_get and 'propmale' in request.GET and 'propmale'
== 'no':

```

```

        roomadvertisements = roomadvertisements.filter
(sharedproperty__maleslive=0)
        if check_get and 'propfemale' in request.GET and
'propfemale' != 'no':
            roomadvertisements = roomadvertisements.filter
(sharedproperty__femaleslive__gt=0)
            start_form = False
        elif check_get and 'propfemale' in request.GET and
'propfemale' == 'no':
            roomadvertisements = roomadvertisements.filter
(sharedproperty__femaleslive=0)
            if check_get and 'propbalcony' in request.GET and
'propbalcony' != 'no':
                roomadvertisements = roomadvertisements.filter
(balcony=True)
                start_form = False
            elif check_get and 'propbalcony' in request.GET and
'propbalcony' == 'no':
                roomadvertisements = roomadvertisements.filter
(balcony=False)
                start_form = False
            if check_get and 'propterrace' in request.GET and
'propterrace' != 'no':
                roomadvertisements = roomadvertisements.filter
(terrace=True)
                start_form = False
            elif check_get and 'propterrace' in request.GET and
'propterrace' == 'no':
                roomadvertisements = roomadvertisements.filter
(terrace=False)
                start_form = False
            if check_get and 'propkitchen' in request.GET and
'propkitchen' != 'no' or (tenant_properties and tenant_ad.kitchen):
                roomadvertisements = roomadvertisements.filter
(kitchen=True)
                start_form = False
            elif check_get and 'propkitchen' in request.GET and
'propkitchen' == 'no':
                roomadvertisements = roomadvertisements.filter
(kitchen=False)
                start_form = False
            if check_get and 'propdis' in request.GET and 'propdis'!=
'no' or (tenant_properties and tenant_ad.disabled):
                roomadvertisements = roomadvertisements.filter
(disabledpeople=True)
                start_form = False
            elif check_get and 'propdis' in request.GET and 'propdis' ==
'no':

```

```

        roomadvertisements = roomadvertisements.filter
(disabledpeople=False)
        start_form = False
        if check_get and 'proppark' in request.GET and 'proppark' !=
'no' or (tenant_properties and tenant_ad.parking):
            roomadvertisements = roomadvertisements.filter
(parking=True)
            start_form = False
            elif check_get and 'proppark' in request.GET and 'proppark'
== 'no':
                roomadvertisements = roomadvertisements.filter
(parking=False)
                start_form = False
                if check_get and 'propgarage' in request.GET and
'propgarage' != 'no' or (tenant_properties and tenant_ad.garage):
                    roomadvertisements = roomadvertisements.filter
(garage=True)
                    start_form = False
                    elif check_get and 'propgarage' in request.GET and
'propgarage' == 'no':
                        roomadvertisements = roomadvertisements.filter
(garage=False)
                        start_form = False
                        if check_get and 'propinternets' in request.GET:
                            find_propinternets =
self.request.GET.get('propinternets')
                            find_propinternets = find_propinternets.split('-')
                            find_propinternet_ = []
                            for pi in find_propinternets:
                                find_propinternet_.append(int(pi))
                            roomadvertisements =
roomadvertisements.filter(sharedproperty__internettype__in=find_propin
ternet_)
                            start_form = False
                            if tenant_properties and tenant_ad.internettype:
                                roomadvertisements =
roomadvertisements.filter(sharedproperty__internettype__in=tenant_ad.i
nternettype)
                                if check_get and 'propbaths' in request.GET:
                                    find_bath_types = self.request.GET.get('propbaths')
                                    find_bath_types = find_bath_types.split('-')
                                    find_bath_types_ = []
                                    for bt in find_bath_types:
                                        find_bath_types_.append(int(bt))
                                    roomadvertisements =
roomadvertisements.filter(sharedproperty__bathtype__in=find_bath_types
_)
                                    start_form = False
                                    if check_get and 'propregion' in request.GET:

```

```

        find_prop_region = self.request.GET.get('propregion')
        find_prop_region = int(find_prop_region)
        find_prop_region = Locationregion.objects.get(id =
find_prop_region)
        roomadvertisements =
roomadvertisements.filter(sharedproperty__locationregion =
find_prop_region)
        start_form = False
        if tenant_properties:
            tenant_regions =
Tenantadvertisementregion.objects.filter(tenantadvertisement =
tenant_ad).select_related('locationregion').values_list('locationregio
n')
            if tenant_regions:
                roomadvertisements =
roomadvertisements.filter(sharedproperty__locationregion__in =
tenant_regions)
                if check_get and 'proplocation' in request.GET:
                    find_prop_location =
self.request.GET.get('proplocation')
                    find_prop_location = int(find_prop_location)
                    find_prop_location = Location.objects.get(id =
find_prop_location)
                    roomadvertisements =
roomadvertisements.filter(sharedproperty__locationregion__location =
find_prop_location)
                    start_form = False
                if check_get and 'propcount' in request.GET:
                    find_prop_country = self.request.GET.get('propcount')
                    find_prop_country = int(find_prop_country)
                    find_prop_country = Country.objects.get(id =
find_prop_country)
                    roomadvertisements =
roomadvertisements.filter(sharedproperty__locationregion__location__co
untry = find_prop_country)
                    start_form = False
                if check_get and 'propcode' in request.GET:
                    find_prop_code = self.request.GET.get('propcode')
                    roomadvertisements =
roomadvertisements.filter(sharedproperty__locationcode=find_prop_code)
                    start_form = False
                find_prop_languages = []
                if check_get and 'proplanguages' in request.GET:
                    find_prop_languages =
self.request.GET.get('proplanguages')
                    find_prop_languages = find_prop_languages.split('-')
                    find_prop_languages_ = []
                    for lang in find_prop_languages:
                        prop_lang_id = int(lang)

```

```

        prop_lang = Language.objects.get(id = prop_lang_id)
        find_prop_languages.append(prop_lang)
    start_form = False
    find_prop_occupations = []
    if check_get and 'propoccups' in request.GET:
        find_prop_occupations =
self.request.GET.get('propoccups')
        find_prop_occupations = find_prop_occupations.split('-')
        find_prop_occupations_ = []
        for occup in find_prop_occupations:
            prop_occup_id = int(occup)
            prop_occup = Occupation.objects.get(id=prop_occup_id)
            find_prop_occupations.append(prop_occup)
        start_form = False
    if not (check_get and 'adorder' in request.GET):
        ad_order = 0
    else:
        ad_order = self.request.GET.get('adorder')
        ad_order = int(ad_order)
        if ad_order == 0:
            roomadvertisements = roomadvertisements.order_by('-
score')
        if ad_order == 1:
            roomadvertisements =
roomadvertisements.order_by('pricemonth')
        elif ad_order == 2:
            roomadvertisements = roomadvertisements.order_by('-
pricemonth')
        elif ad_order == 3:
            roomadvertisements =
roomadvertisements.order_by('availableat')
        elif ad_order == 4:
            roomadvertisements = roomadvertisements.order_by('-
availableat')
        roomadvertisements_after_check = []
        for room in roomadvertisements:
            select_room = True
            if check_get and 'proplanguages' in request.GET:
                prop_communcations =
Sharedpropertycommunication.objects.filter(sharedproperty =
room.sharedproperty, language__in = find_prop_languages)
                if not prop_communcations or
prop_communcations.count() < 1:
                    select_room = False
            if check_get and 'propoccups' in request.GET:
                prop_occupations =
Sharedpropertyoccupation.objects.filter(sharedproperty =
room.sharedproperty, occupation__in=find_prop_occupations)

```

```

        if not prop_occupations or prop_occupations.count()
< 1:
            select_room = False
            if select_room:
                if request.user:
                    tenant =
Useraccount.objects.filter(registereduser=request.user).values()
                    roomsaves =
Roomadvertisementsave.objects.filter(roomadvertisement=room,
useraccount=tenant[0])
                    if roomsaves and roomsaves.count() > 0:
                        room.saved = True
                        roomadvertisements_after_check.append(room)
            if not start_form:
                found_roomadvertisement_data['advertisements'] =
roomadvertisements_after_check
                return render(request, "roompersonaler/find_room_ad.html",
found_roomadvertisement_data)

    def tenantadvertisement_find(request):
        found_tenantadvertisement_data = {}
        found_tenantadvertisement_data['advertisements'] = []
        found_tenantadvertisement_data['countries'] =
Country.objects.all()
        found_tenantadvertisement_data['locations'] =
Location.objects.all()
        found_tenantadvertisement_data['regions'] =
Locationregion.objects.all()
        found_roomadvertisement_data['propertytypes'] =
Propertytype.objects.all()
        found_roomadvertisement_data['languages'] =
Language.objects.all()
        found_roomadvertisement_data['roomtypes'] =
Roomtype.objects.all()
        check_get = request.method == 'GET'
        start_form = True
        tenantadvertisements =
Tenantadvertisement.objects.filter(status=True)
        tenantadvertisements =
tenantadvertisements.select_related('roomtype')
        tenantadvertisements =
tenantadvertisements.select_related('propertytype')
        roomadvertisements =
roomadvertisements.select_related('tenant')
        if check_get and 'roomtype' in request.GET:
            find_room_type = self.request.GET.get('roomtype')
            find_room_type = int(find_room_type)
            find_room_type = Roomtype.objects.get(id =
find_room_type)

```

```

        tenantadvertisements =
tenantadvertisements.filter(roomtype=find_room_type)
        start_form = False
        if check_get and 'scoreabove' in request.GET:
            tenant_score_above = self.request.GET.get('scoreabove')
            tenant_score_above = float(room_score_above)
            tenantadvertisements = tenantadvertisements.filter
(score__gte=tenant_score_above)
            start_form = False
            if check_get and 'tenfurniture' in request.GET and
'tenfurniture' != 'no':
                tenantadvertisements = tenantadvertisements.filter
(furniture=True)
                start_form = False
            elif check_get and 'tenfurniture' in request.GET and
'tenfurniture' == 'no':
                tenantadvertisements = tenantadvertisements.filter
(furniture=False)
                start_form = False
            if check_get and 'tendesk' in request.GET and 'tendesk'!=
'no':
                tenantadvertisements = tenantadvertisements.filter
(desk=True)
                start_form = False
            elif check_get and 'tendesk' in request.GET and 'tendesk' ==
'no':
                tenantadvertisements = tenantadvertisements.filter
(desk=False)
                start_form = False
            if check_get and 'tensleepingtype' in request.GET:
                find_sleep_type =
self.request.GET.get('tensleepingtype')
                find_sleep_type = int(find_sleep_type)
                tenantadvertisements = tenantadvertisements.filter
(sleepingtype__lte=find_sleep_type)
                start_form = False
            if check_get and 'tenwardrobe' in request.GET and
'tenwardrobe' != 'no':
                tenantadvertisements = tenantadvertisements.filter
(wardrobe=True)
                start_form = False
            elif check_get and 'tenwardrobe' in request.GET and
'tenwardrobe' == 'no':
                tenantadvertisements = tenantadvertisements.filter
(wardrobe=False)
                start_form = False
            if check_get and 'tensmoking' in request.GET and
'tensmoking' != 'no':

```



```

        if check_get and 'tengender' in request.GET and 'tengender'
== 1:
            tenantadvertisements = tenantadvertisements.filter
(gender=True)
            start_form = False
        elif check_get and 'tengender' in request.GET and
'tengender' == 2:
            tenantadvertisements = tenantadvertisements.filter
(gender=False)
            start_form = False
        if check_get and 'tenveget' in request.GET and 'tenveget' !=
'no':
            tenantadvertisements = tenantadvertisements.filter
(vegetarian=True)
            start_form = False
        elif check_get and 'tenveget' in request.GET and 'tenveget'
== 'no':
            tenantadvertisements = tenantadvertisements.filter
(vegetarian=False)
            if check_get and 'tenoccup' in request.GET:
                tenant_occupationtype = self.request.GET.get('tenoccup')
                tenant_occupationtype = int(tenant_occupationtype)
                tenantadvertisements = tenantadvertisements.filter
(occupationtype__gte=tenant_occupationtype)
            if check_get and 'tencouple' in request.GET and 'tencouple'
!= 'no':
                tenantadvertisements = tenantadvertisements.filter
(couples=True)
                start_form = False
            elif check_get and 'tencouple' in request.GET and
'tencouple' == 'no':
                tenantadvertisements = tenantadvertisements.filter
(couples=False)
            if check_get and 'tenlivingroom' in request.GET and
'tenlivingroom' != 'no':
                tenantadvertisements = tenantadvertisements.filter
(livingroom=True)
                start_form = False
            elif check_get and 'tenlivingroom' in request.GET and
'tenlivingroom' == 'no':
                tenantadvertisements = tenantadvertisements.filter
(livingroom=False)
            if check_get and 'tenantbaths' in request.GET:
                find_bath_types = self.request.GET.get('tenantbaths')
                find_bath_types = find_bath_types.split('-')
                find_bath_types_ = []
                for bt in find_bath_types:
                    find_bath_types_.append(int(bt))

```

```

        tenantadvertisements =
tenantadvertisements.filter(bathtype__in= find_bath_types)
        if check_get and 'teninternets' in request.GET:
            find_propinternets =
self.request.GET.get('teninternets')
            find_propinternets = find_propinternets.split('-')
            find_propinternet_ = []
            for pi in find_propinternets:
                find_propinternet_.append(int(pi))
            tenantadvertisements =
tenantadvertisements.filter(internettype__in=find_propinternet_)
            start_form = False
            if check_get and 'tenkitchen' in request.GET and
'tenkitchen' != 'no':
                tenantadvertisements = tenantadvertisements.filter
(kitchen=True)
                start_form = False
            elif check_get and 'tenkitchen' in request.GET and
'tenkitchen' == 'no':
                tenantadvertisements = tenantadvertisements.filter
(kitchen=False)
                if check_get and 'tendisabled' in request.GET and
'tendisabled' != 'no':
                    tenantadvertisements = tenantadvertisements.filter
(disabled=True)
                    start_form = False
                elif check_get and 'tendisabled' in request.GET and
'tendisabled' == 'no':
                    tenantadvertisements = tenantadvertisements.filter
(disabled=False)
                    if check_get and 'tenparking' in request.GET and
'tenparking' != 'no':
                        tenantadvertisements = tenantadvertisements.filter
(parking=True)
                        start_form = False
                    elif check_get and 'tenparking' in request.GET and
'tenparking' == 'no':
                        tenantadvertisements = tenantadvertisements.filter
(parking=False)
                        if check_get and 'tengarage' in request.GET and 'tengarage'
!= 'no':
                            tenantadvertisements = tenantadvertisements.filter
(garage=True)
                            start_form = False
                        elif check_get and 'tengarage' in request.GET and
'tengarage' == 'no':
                            tenantadvertisements = tenantadvertisements.filter
(garage=False)
                            if check_get and 'availabled' in request.GET:

```

```

        available_at_day = self.request.GET.get('availabled')
        available_at_day = int(available_at_day)
        available_at_month = self.request.GET.get('availablem')
        available_at_month = int(available_at_month)
        available_at_year = self.request.GET.get('availabley')
        available_at_year = int(available_at_year)
        import datetime
        find_available_at = datetime.date(available_at_year,
available_at_month, available_at_day)
        tenantadvertisements =
tenantadvertisements.filter(availableat__lte = find_available_at)
        start_form = False
        if check_get and 'fromrentmonths' in request.GET:
            find_room_months =
self.request.GET.get('fromrentmonths')
            find_room_months = int(find_room_months)
            tenantadvertisements =
tenantadvertisements.filter(minmonths__gte=find_room_months)
            start_form = False
            if check_get and 'torentmonths' in request.GET:
                find_room_months = self.request.GET.get('torentmonths')
                find_room_months = int(find_room_months)
                tenantadvertisements =
tenantadvertisements.filter(minmonths__lte=find_room_months)
                start_form = False
                if check_get and 'tenbudget' in request.GET:
                    find_ten_budget = self.request.GET.get('tenbudget')
                    find_ten_budget = int(find_ten_budget)
                    tenantadvertisements =
tenantadvertisements.filter(budget__gte=find_ten_budget)
                    start_form = False
                    if check_get and 'proptype' in request.GET:
                        find_property_type = self.request.GET.get('proptype')
                        find_property_type = int(find_property_type)
                        find_property_type = Propertytype.objects.get(id =
find_property_type)
                        tenantadvertisements =
tenantadvertisements.filter(propertytype=find_property_type)
                        start_form = False
                        if check_get and 'fromage' in request.GET:
                            find_tenant_age = self.request.GET.get('fromage')
                            find_tenant_age = int(find_tenant_age)
                            tenantadvertisements =
tenantadvertisements.filter(age__gte=find_tenant_age)
                            start_form = False
                            if check_get and 'toage' in request.GET:
                                find_tenant_age = self.request.GET.get('toage')
                                find_tenant_age = int(find_tenant_age)

```

```

        tenantadvertisements =
tenantadvertisements.filter(age__lte=find_tenant_age)
        start_form = False
        if check_get and 'tenregion' in request.GET:
            find_tenant_region = self.request.GET.get('tenregion')
            find_tenant_region = int(find_tenant_region)
            find_tenant_region = Locationregion.objects.get(id =
find_tenant_region)
            start_form = False
            if check_get and 'tenlocation' in request.GET:
                find_tenant_location =
self.request.GET.get('tenlocation')
                find_tenant_location = int(find_tenant_location)
                find_tenant_location = Location.objects.get(id =
find_tenant_location)
                start_form = False
                if check_get and 'tencount' in request.GET:
                    find_tenant_country = self.request.GET.get('tencount')
                    find_tenant_country = int(find_tenant_country)
                    find_tenant_country = Country.objects.get(id =
find_tenant_country)
                    if check_get and 'teninterests' in request.GET:
                        find_tenant_interests =
self.request.GET.get('teninterests')
                        find_tenant_interests = find_tenant_interests.split('--
')

                        find_tenant_interests_ = []
                        for ti in find_tenant_interests:
                            find_tenant_interests_values = ti.split('-')
                            interest_obj = find_tenant_interests_values[0]
                            interest_obj = int(interest_obj)
                            interest_value = find_tenant_interests_values[1]
                            find_tenant_interests_.append([interest_obj,
interest_value])
                        start_form = False
                        if not (check_get and 'adorder' in request.GET):
                            ad_order = 0
                        else:
                            ad_order = self.request.GET.get('adorder')
                            ad_order = int(ad_order)
                            if ad_order == 0:
                                tenantadvertisements =
tenantadvertisements.order_by('-score')
                            elif ad_order == 1:
                                tenantadvertisements =
tenantadvertisements.order_by('age')
                            elif ad_order == 2:
                                tenantadvertisements =
tenantadvertisements.order_by('-budget')

```

```

        elif ad_order == 3:
            tenantadvertisements =
tenantadvertisements.order_by('availableat')
        elif ad_order == 4:
            tenantadvertisements =
tenantadvertisements.order_by('-availableat')
            tenantadvertisements_after_check = []
            for tenantad in tenantadvertisements:
                select_tenant = True
                if check_get and 'teninterests' in request.GET:
                    interest_found = False
                    for ten_interest in find_tenant_interests_:
                        if ten_interest[1]:
                            user_interests =
Userinterestitem.objects.filter(userinterest__useraccount=tenantad.ten
ant, userinterest__interest=ten_interest[0],
itemvalue=ten_interest[1])
                        else:
                            user_interests =
Userinterestitem.objects.filter(userinterest__useraccount=tenantad.ten
ant, userinterest__interest=ten_interest[0])
                            if user_interests and user_interests.count() >=
0:
                                interest_found = True
                                if not interest_found:
                                    select_tenant = False
                                if check_get and 'tenlang' in request.GET:
                                    find_tenant_language =
self.request.GET.get('tenlang')
                                    find_tenant_language = int(find_tenant_language)
                                    find_tenant_language = Language.objects.get(id =
find_tenant_language)

                                    tenant_languages =
Userlanguage.objects.filter(useraccount = tenantad.tenant, language =
find_tenant_language)
                                    if not tenant_languages or tenant_languages.count()
< 1:
                                        select_tenant = False
                                        if check_get and 'tenregion' in request.GET:
                                            tenant_regions =
Tenantadvertisementregion.objects.filter(locationregion=find_tenant_re
gion, tenantadvertisement=tenantad)
                                            if not tenant_regions or tenant_regions.count() ==
0:
                                                select_tenant = False
                                                if check_get and 'tenlocation' in request.GET:

```

```

        tenant_locations =
Tenantadvertisementregion.objects.filter(locationregion__location=find
_tenant_location, tenantadvertisement=tenantad)
        if not tenant_locations or tenant_locations.count()
== 0:
            select_tenant = False
            if check_get and 'tencountry' in request.GET:
                tenant_countries =
Tenantadvertisementregion.objects.filter(locationregion__location__cou
ntry=find_tenant_country, tenantadvertisement=tenantad)
                if not tenant_countries or tenant_countries.count()
== 0:
                    select_tenant = False
                    if select_tenant:
                        if request.user:
                            landlord =
Useraccount.objects.filter(registereduser=request.user).values()
                            if landlord.values()[0].landlord:
                                tensaves =
Tenantadvertisementsave.objects.filter(tenantadvertisement=tenantad,
useraccount= landlord.values()[0])
                                if tensaves and tensaves.count() > 0:
                                    tenantad.saved = True
                                    tenantadvertisements_after_check.append(tenantad)
                    if not start_form:
                        found_tenantadvertisement_data['advertisements'] =
tenantadvertisements_after_check
                        return render(request, "roompersonaler/find_tenant_ad.html",
found_tenantadvertisement_data)

```

ДОДАТОК В
Слайди презентації

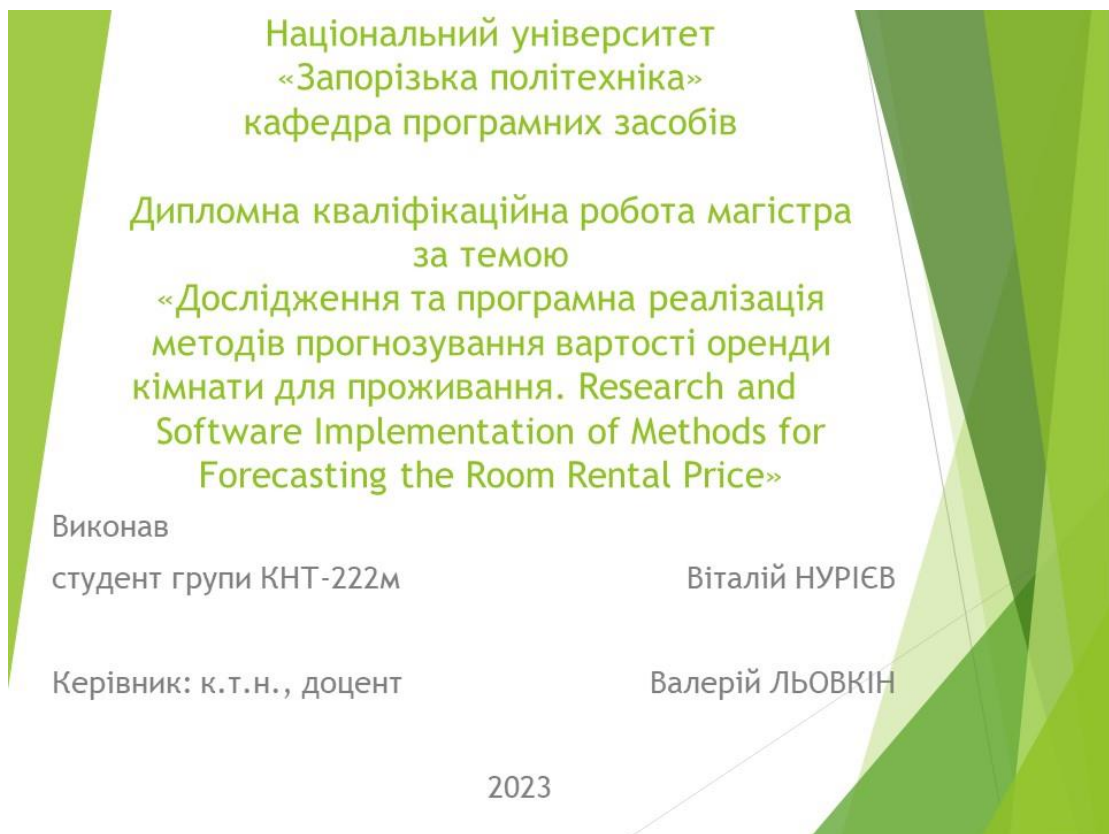


Рисунок В.1 – Титульний слайд

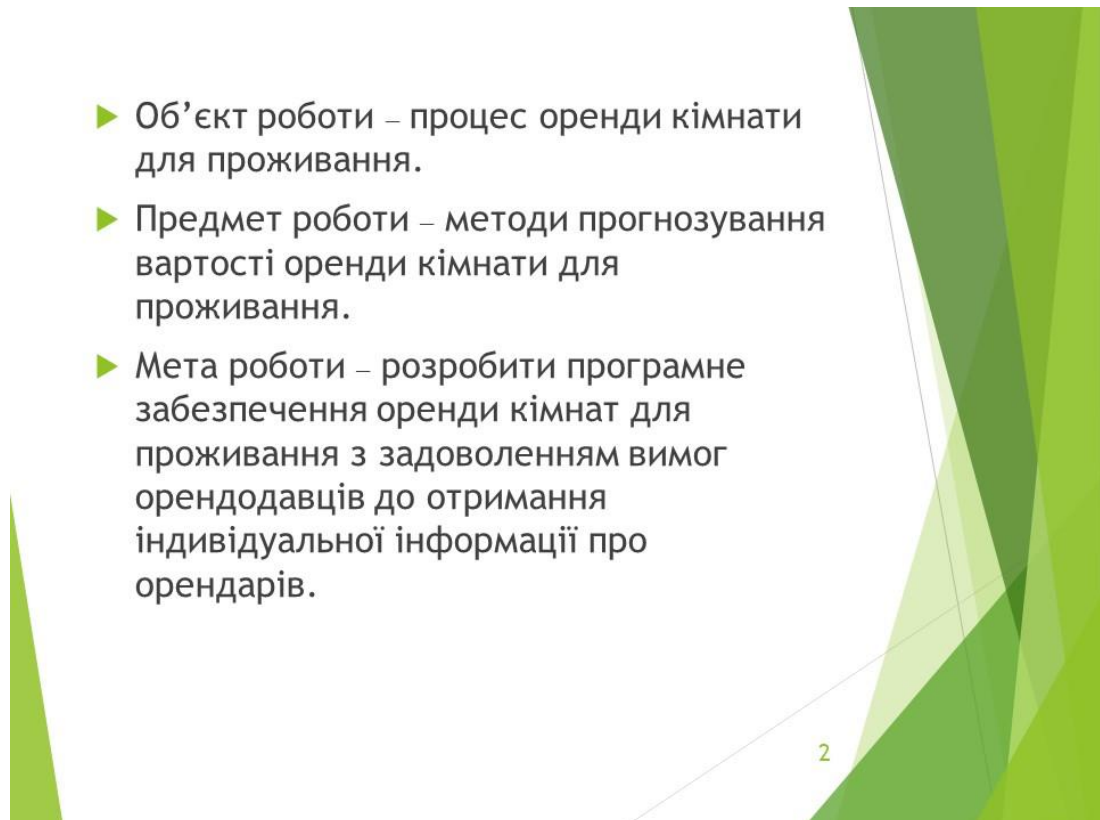


Рисунок В.2 – Об'єкт, предмет та мета роботи

Завдання роботи

- ▶ проаналізувати аналоги програми оренди кімнат для проживання;
- ▶ розробити метод прогнозування вартості оренди кімнати для проживання;
- ▶ виконати проєктування програмного забезпечення оренди кімнат для проживання;
- ▶ реалізувати програмне забезпечення оренди кімнат для проживання.

Рисунок В.3 – Завдання роботи

Порівняння програмного забезпечення для оренди кімнат для проживання

Критерій	SpareRoom	iBilik	Airbnb	RoomPersonaler
Оренда кімнат	Так	Так	Так	Так
Строк оренди кімнат	Від місяця	Короткий і від місяця	Короткий, можливо більший	Від місяця
Створення оголошень про пошук кімнат	Так	Ні	Ні	Так
Сплата за оренду кімнати	Ні	Ні	Так	Ні
Фільтри за індивідуальними характеристиками орендаря	Так	Так	Ні	Так
Визначення індивідуальних захоплень і пошук за ними	Ні	Ні	Ні	Так
Порівняння орендарів	Ні	Ні	Ні	Так
Прогнозування вартості оренди	Ні	Ні	Ні	Так

Рисунок В.4 – Порівняння програмного забезпечення для оренди кімнат для проживання

Метод прогнозування вартості оренди кімнати для проживання

- ▶ має бути початково сформовано вибірку даних, яка визначає характеристики оголошень щодо оренди саме окремих кімнат, включаючи індивідуальні характеристики орендаря, які встановлені в оголошенні, разом з характеристиками житла і оренди;
- ▶ опрацювання даних з вибірки, перетворюючи їх зокрема до цілочисельних;
- ▶ створення моделі прогнозування вартості оренди кімнати для проживання, побудованої на основі принципів бустингу з визначенням необхідної кількості дерев та максимальної глибини дерева;
- ▶ навчання моделі на основі вибірки, де вихідною ознакою є вартість місяця оренди кімнати, а вхідними ознаками - всі інші ознаки вибірки даних;
- ▶ визначення для нових оголошень значення всіх ознак окрім вихідної, яка має бути обчислена на основі моделі прогнозування вартості оренди кімнати для проживання, яку було навчено.

5

Рисунок В.5 – Метод прогнозування вартості оренди кімнати для проживання

Порівняння вебфреймворків Falcon і Django для реалізації програми

Критерій	Falcon	Django
Мова програмування	Python	Python
Створення і підключення моделі прогнозування вартості оренди кімнати для проживання	Так	Так
Базові можливості роботи з даними для адміністратора	Немає	Інтегровані
Створення вебсторінок за допомогою рушія шаблонів	Немає	Інтегровано
Підтримка засобів безпеки	Потребує встановлення зовнішніх засобів	Інтегровано, широка

6

Рисунок В.6 – Порівняння вебфреймворків Falcon і Django для реалізації програми

Діаграми прецедентів орендаря кімнати і орендодавця

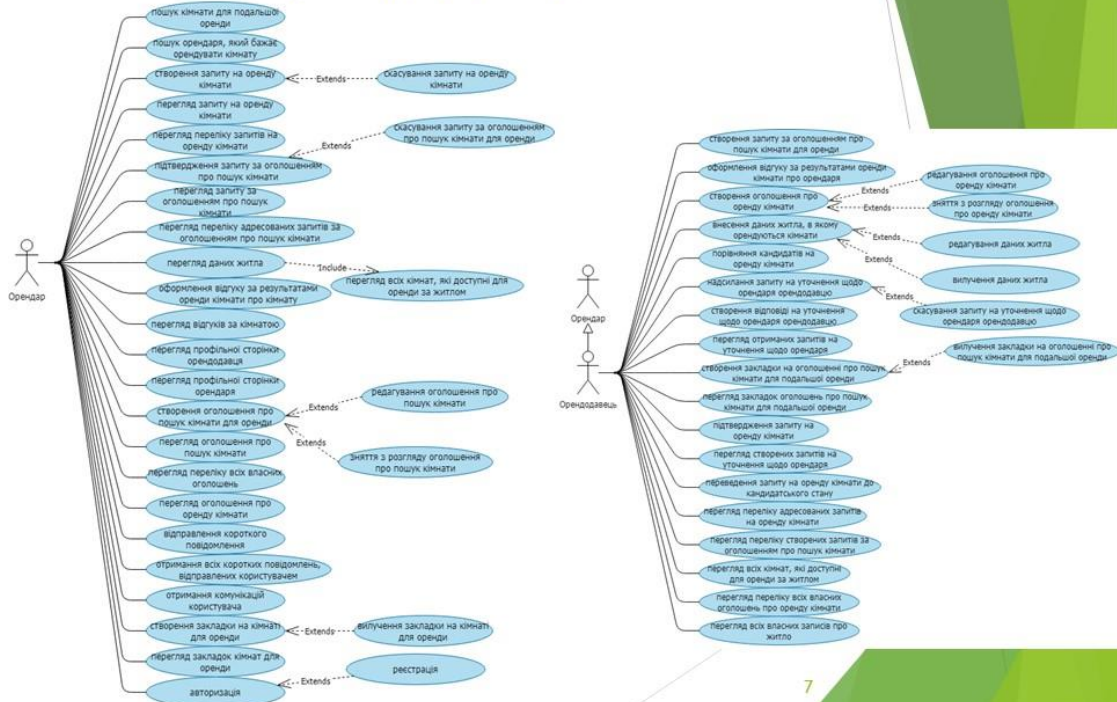


Рисунок В.7 – Діаграми прецедентів орендаря кімнати і орендодавця

Логічна схема бази даних

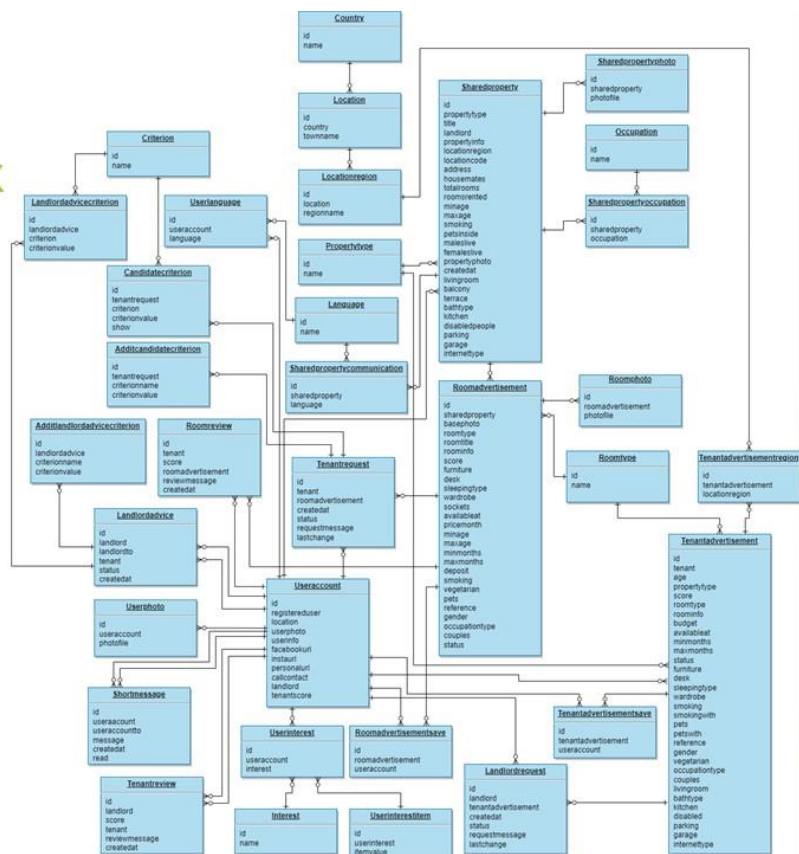


Рисунок В.8 – Схема реляційної бази даних

Функціональна схема програми для оренди кімнат для проживання

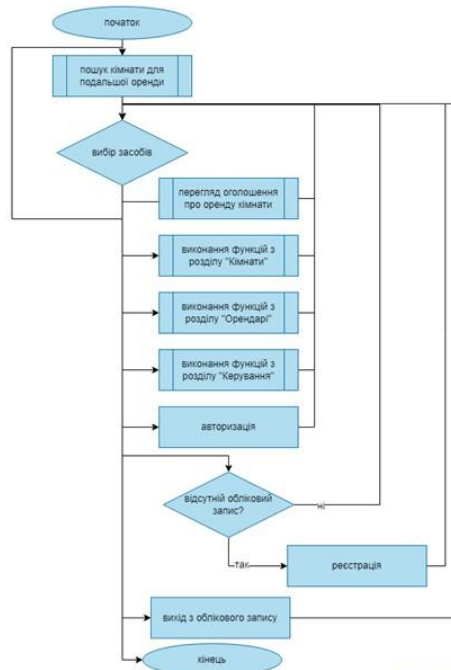


Рисунок В.9 – Функціональна схема програми для оренди кімнат для проживання

Вебсторінки програми для оренди кімнат

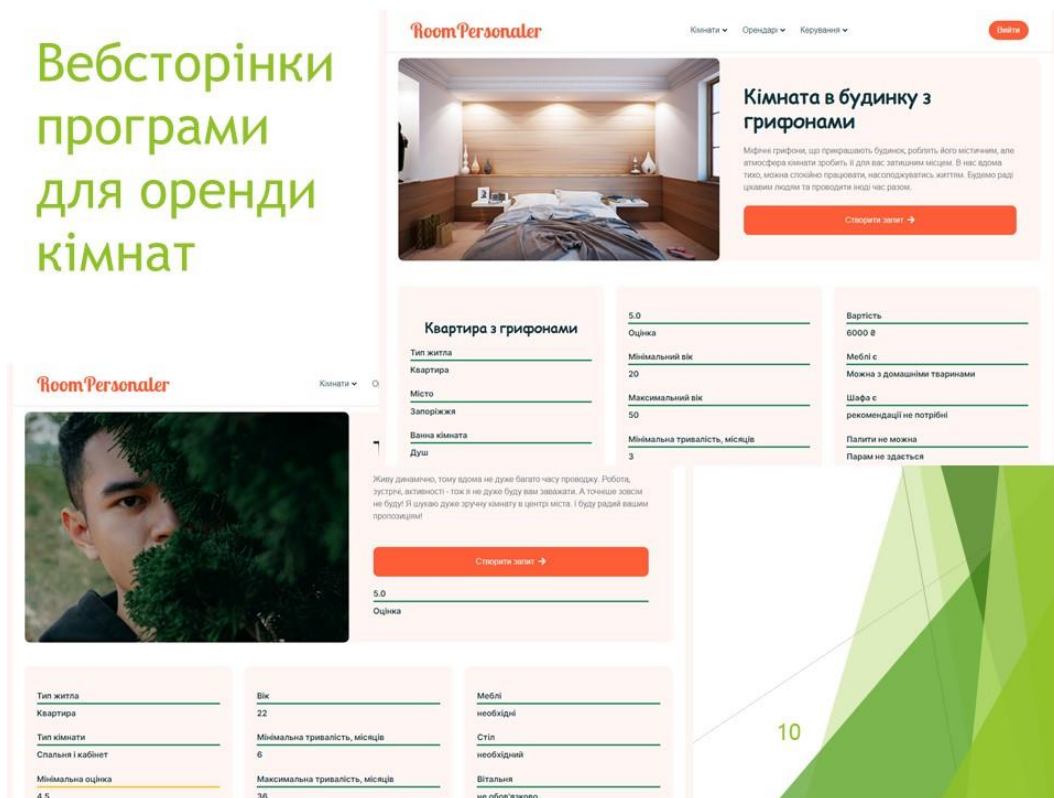


Рисунок В.10 – Вебсторінки програми для оренди кімнат

Вебсторінки програми для оренди кімнат

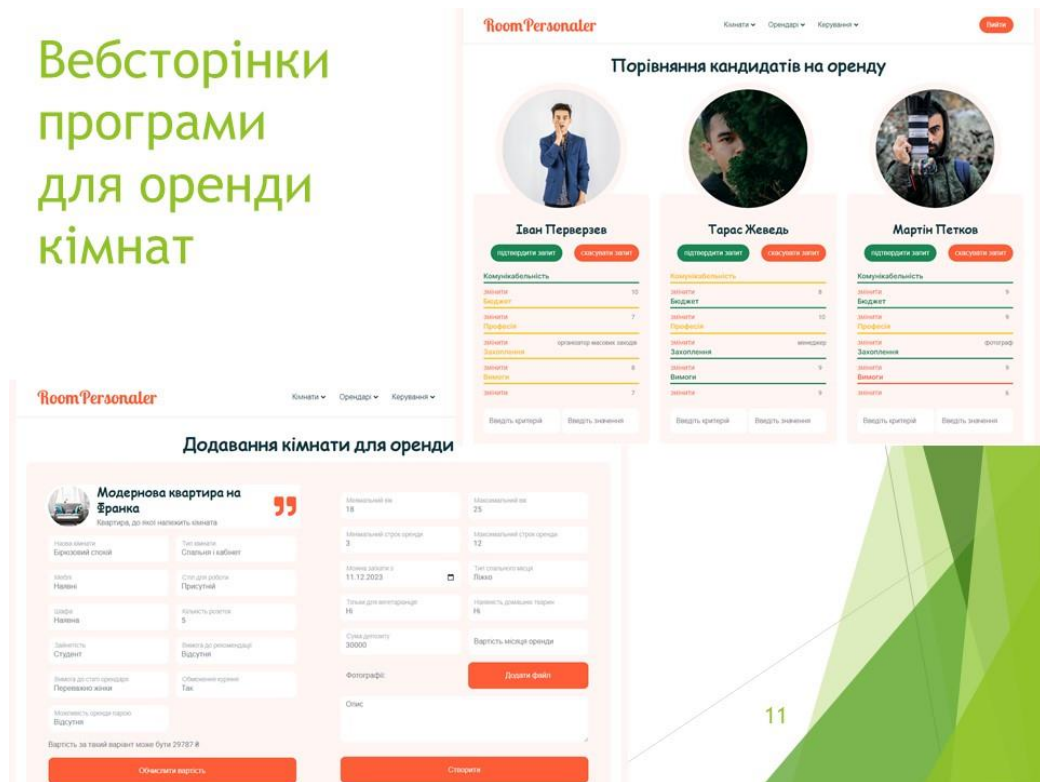


Рисунок В.11 – Вебсторінки програми для оренди кімнат

Експериментальне дослідження проблеми прогнозування вартості оренди кімнати для проживання

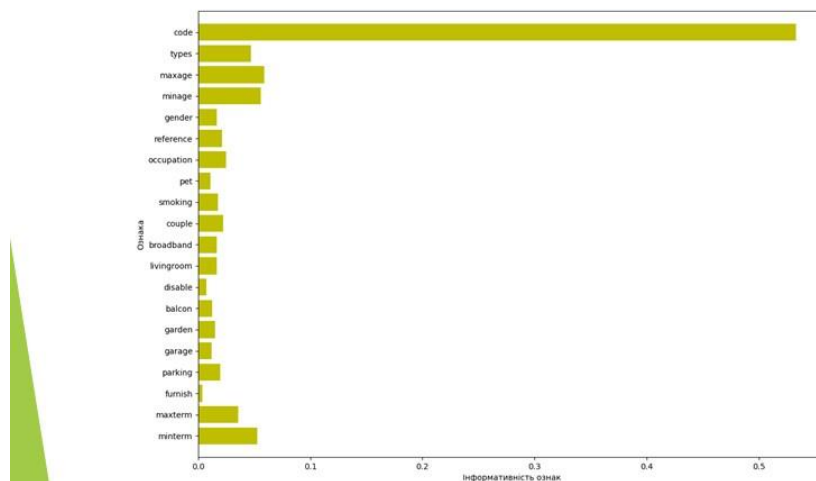


Рисунок В.12 – Експериментальне дослідження проблеми прогнозування вартості оренди кімнати для проживання

Експериментальне дослідження проблеми прогнозування вартості оренди кімнати для проживання

Модель	Кількість дерев	Глибина дерева	Середньоквадратична похибка
Лінійна регресія	-	-	0,044644
Дерево рішень	1	20	0,048673
Дерево рішень	1	10	0,046737
Дерево рішень	1	8	0,04418
XGBoost	50	6	0,035562
XGBoost	50	8	0,035624
XGBoost	50	20	0,037634
XGBoost	100	6	0,035042
XGBoost	100	8	0,035248
XGBoost	100	20	0,037634
XGBoost	500	6	0,034695
XGBoost	500	8	0,035248
XGBoost	500	20	0,037634
XGBoost	1000	6	0,034695
XGBoost	1000	8	0,035248
XGBoost	1000	20	0,037634

13

Рисунок В.13 – Експериментальне дослідження проблеми прогнозування вартості оренди кімнати для проживання

Висновки за роботою

- ▶ При реалізації програми використано засоби веброзробки, включаючи мови Python, HTML, CSS, фреймворк Django, засоби машинного навчання на основі Python-пакетів, інструмент для роботи з базою даних MySQL.
- ▶ Програма в підсумку забезпечує необхідні засоби для пошуку орендарями кімнат для оренди, а орендодавцями орендарів, враховуючи зокрема індивідуальні характеристики, щоб забезпечити створення комунікації щодо подальшої оренди, відбір кандидатів для оренди власної кімнати, надаючи при цьому можливість прогнозувати вартість оренди на основі індивідуальних характеристик орендаря.
- ▶ Наукова новизна роботи полягає в створеному методі прогнозування вартості оренди кімнати для проживання, який базується на застосуванні ансамблів дерев рішень на основі принципів бустингу і який дозволяє врахувати індивідуальні характеристики орендаря, додаючи їх до ознак створеної моделі прогнозування, що визначають загальні характеристики житла і оренди.

14

Рисунок В.14 – Висновки за роботою