

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)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ОРЕНДИ

(назва теми)

ЗАМІСЬКИХ БУДИНКІВ

COUNTRY HOUSE RENTAL SOFTWARE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-129

Спеціальності 121 Інженерія програмного

(код і найменування спеціальності)

забезпечення

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

КУЧЕР С.О.

(ПРІЗВИЩЕ та ініціали)

Керівник ГОЛУБ Т.В.

(ПРІЗВИЩЕ та ініціали)

Рецензент ПОЛЯКОВ М.О.

(ПРІЗВИЩЕ та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет ФКНТ

Кафедра програмних засобів

Ступінь вищої освіти бакалавр

Спеціальність 121 Інженерія програмного забезпечення

(код і найменування)

Освітня програма (спеціалізація) Інженерія програмного забезпечення

(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)

КУЧЕРА Станіслава Олександровича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення оренди замських будинків
Country House Rental Software

керівник проєкту (роботи) к.т.н., доцент ГОЛУБ Тетяна Василівна

(науковий ступінь, вчене звання, ПРИЗВИЩЕ, ім'я, по батькові)

затверджені наказом закладу вищої освіти від «04» квітня 2023 року № 87

2. Строк подання студентом проєкту (роботи) 6 червня 2023 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз предметної області та постановка завдань дослідження.
2. Проєктування програми. 3. Розробка програми. 4. Експлуатація, тестування та експериментальне дослідження програми.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, кількість слайдів, плакатів)

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРИЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ГОЛУБ Т.В., доцент		
Нормоконтроль	БСЛОВА А.В., асистент		

7. Дата видачі завдання « 6 » квітня 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	Розробка архітектури програми	3 тиждень	Розділ 2
4	Розробка програми	4-5 тижні	Розділ 3
5	Тестування та експериментальне дослідження програми	6 тиждень	Розділ 4
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	7 тиждень	Додатки
7	Захист роботи	8 тиждень	

Студент(ка)

(підпис)

Станіслав КУЧЕР

(Ім'я ПРИЗВИЩЕ)

Керівник проєкту (роботи)

(підпис)

Тетяна ГОЛУБ

(Ім'я ПРИЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 92 с., 27 рис., 2 табл., 3 дод., 10 джерел.

ЗАМІСЬКИЙ БУДИНОК, ОRENDA БУДИНКУ, БРОНЮВАННЯ, ПРОГРАМНА СИСТЕМА, PYTHON.

Об'єкт роботи – процес оренди заміських будинків.

Предмет роботи – програмне забезпечення оренди заміських будинків.

Мета роботи – розробити програмне забезпечення оренди заміських будинків.

Матеріали, методи та технічні засоби: мова програмування python, мови для побудови вебсторінок HTML, CSS, javascript та база даних MySQL.

Результати. Створено детальну структуру бази даних для врахування особливостей оренди заміських будинків, визначено і розподілено за ролями функціональні вимоги до програми, реалізовано програмну систему як вебсистему з описом отриманих результатів.

Висновки. Програмну систему, яку розроблено в підсумку, можна застосовувати для бронювання оренди заміських будинків і підтримки процесів, пов'язаних з бронюваннями. Програма дозволяє безпосередньо самому власнику заміського будинку або агенту визначити параметри оренди будинку, включаючи дати, коли він доступний для оренди, різну вартість у різні періоди, керувати бронюваннями звичайних користувачів, які фактично є орендарями, керувати переглядами будинків. Користувачі можуть бронювати оренду житла, перегляд будинків, коли вони бажають завчасно ознайомитися з будинком та шукати заміські будинки, які відповідають їх вимогам.

Галузь використання. Оренда заміських будинків, маєтків, віл.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 92 pages, 27 figures, 2 tables, 3 appendixes, 10 sources.

COUNTRY HOUSE, HOUSE FOR RENT, BOOKING, SOFTWARE SYSTEM, PYTHON.

The object of the work is the process of renting country houses.

The subject of the work is country house rental software.

The purpose of the work is to develop country house rental software.

Materials, methods and technical means: Python programming language, languages HTML, CSS, JavaScript for creating webpages and MySQL database.

The results. A detailed structure of the database was created to take into account the peculiarities of renting country houses, the functional requirements for the program were defined and distributed by roles, the program system was implemented as a web system with a description of the obtained results.

Conclusions. The resulting software system can be used to book country houses and support booking processes. The program allows directly the owner of the country house or the agent to define the parameters of renting the house, including the dates when it is available for rent, different prices in different periods, manage the bookings of regular users who are actually renters, manage house viewings. Users can book home rentals, view homes when they want to view a home ahead of time, and search for country houses that meet their requirements.

Field of use. Rent of country houses, estates, villas.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області та постановка завдань дослідження.....	9
1.1 Існуючі додатки для оренди заміських будинків	9
1.2 Порівняння аналогів	13
1.3 Висновки за розділом 1	14
2 Проєктування програми.....	15
2.1 Вибір технологій веброзробки.....	15
2.2 Вимоги до програми	16
2.3 Структура організації даних	20
2.4 Висновки за розділом 2	29
3 Розробка програми	30
3.1 Порядок функціонування програми.....	30
3.2 Реалізація функцій програмної системи.....	31
3.3 Висновки за розділом 3	41
4 Експлуатація, тестування та експериментальне дослідження програми	42
4.1 Призначення програми	42
4.2 Умови виконання програми	42
4.3 Принципи експлуатації програми	42
4.4 Висновки за розділом 4	48
Висновки	49
Перелік джерел посилання	50
Додаток А Технічне завдання	51
Додаток Б Текст програми	56
Додаток В Слайди презентації.....	86

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ПЗ – програмне забезпечення.

ВСТУП

Сфера оренди житла є активною сферою економіки майже в будь-якій країні. Кількість житла зазвичай є достатньо великою, але кількість запитів відповідає цій кількості, в багатьох містах або місцях створюючи нестачу пропозиції. З виникненням будь-яких економічних проблем, що зараз переживає вчергове світова економіка, орендарі втрачають можливість купувати житло, відповідно більше людей змушені залишатися в секторі оренди, тобто кількість кандидатів накопичується, а пропозиція так швидко не росте, а під час економічних проблем зупиняється, тобто загалом це призводить до зростання попиту [1]. У підсумку процентні ставки збільшуються, орендодавці страждають від цього, тому це додатково спричиняє зменшенню пропозицій, підвищення орендної плати [1]-[2].

Національна асоціація ріелторів у США виявила, що середній вік покупців заміських будинків на даний момент впав до 43 років порівняно з 61 роком, що було більше ніж десять років тому в 2003 році, коли ці дані вперше почали збирати [2]. Це загалом вказує на інтерес нових поколінь до проживання в заміських будинках, частиною чого є оренда таких будинків.

З іншого боку оренда заміських будинків визначає необхідність взаємодії між людьми, які володіють такими будинками або представляють таких людей, тобто є агентами, та орендарями. Така необхідність має забезпечуватися використанням інформаційних технологій, при чому таких, щоб можна було забезпечити віддалену взаємодію. Тому при виконанні роботи таку необхідність потрібно забезпечити за допомогою відповідного програмного забезпечення (ПЗ).

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Існуючі додатки для оренди заміських будинків

Оренда заміських будинків є актуальною сферою зокрема в період, коли набирає активність зелений туризм, постають питання, пов'язані з проблемами довкілля. Відповідно частина людей бажає відпочивати або проживати не у великих містах, а за містом. Заміські будинки мають достатньо інший формат порівняно зі звичайними квартирами в містах, тому потрібно проаналізувати зокрема і те, як їх дані представляються в існуючих програмах.

Велика частина додатків у цій сфері пов'язана з підтримкою пропозицій конкретної компанії. Таким, наприклад, є CountryHouseCompany [3] (рис. 1.1).

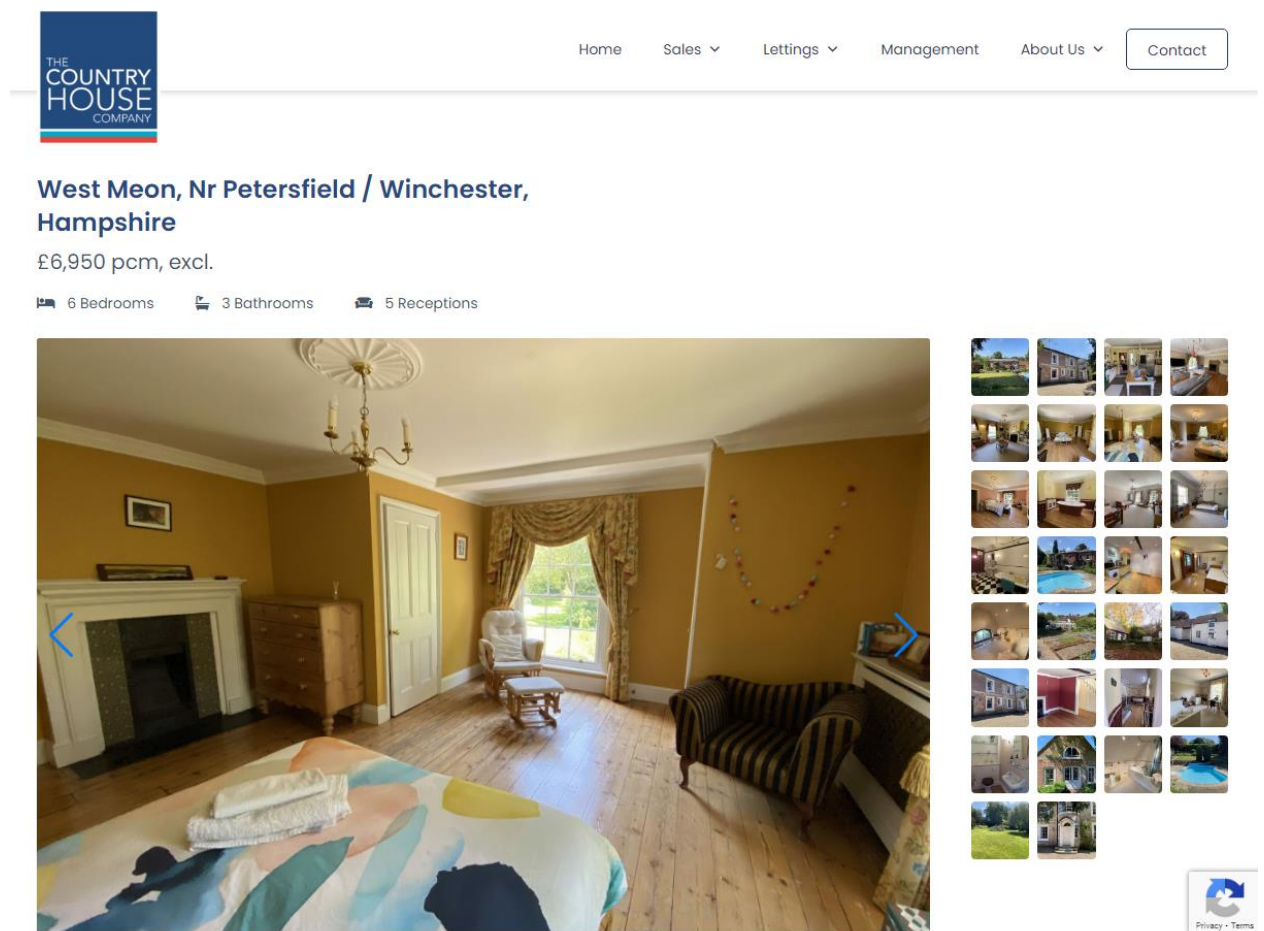


Рисунок 1.1 – Вебдодаток CountryHouseCompany

Вебдодаток CountryHouseCompany надає інформацію про будинки, які можна придбати або орендувати, вказуючи за кожним будинком основні його параметри, фотографії, вартість місяця оренди. Проблемаю цього вебдодатку і подібних є те, що загалом вони не мають можливості бронювання житла, тому що зазвичай потрібно зв'язуватися з відповідним агентством по телефону або подібним чином. Також користувачі не мають можливості залишати відгуки про житло. Тобто це вебсайт, на якому можна отримати загальну інформацію про заміські будинки з обмеженого кола, що керуються конкретним агентством, але він не має додаткових засобів.

CottageGems [4] (рис. 1.2) дозволяє виконувати оренду заміських будинків, але вже не від окремого агентства, тому має можливість пошуку, включаючи визначення місця, де розташовано будинок, дату початку та кількість ночей, кількість людей, які будуть проживати в заміському будинку.

The screenshot displays the CottageGems website interface. At the top, there is a navigation bar with the 'COTTAGE GEMS' logo and links for 'Locations', 'Inspiration', 'Occasions', 'Last Minute', 'Offers', and 'Destination Guides'. Below this, a breadcrumb trail reads 'Home page » England » Essex » Colchester » Broman's Barn'. The main section features a large image of a cozy living room with a pink sofa and a fireplace. To the right of the image is a 'Booking & Enquiries' section with a calendar for June 2023, showing 2 nights left. Below the calendar, there are fields for 'Arrival date', 'Nights' (set to 1), and 'Guests' (set to 1 guest). A 'Book on Owners Website' button is prominently displayed. At the bottom, a description reads: 'Cottage with exposed beams, This is a very creative conversion of an old Barn; very cosy and relaxing, the sun streaming in through the many windows and a wonderful log wood burner to enjoy in the evenings. Ground floor bedroom and'.

Рисунок 1.2 – Вебдодаток CottageGems

CottageGems працює за принципом агрегатора, тому вебсайт не має можливості бронювання оренди, а тільки дозволяє переходити на окремий сайт будинку. За кожним будинком надається інформація про нього, включаючи кількість проживаючих, ліжок, ванн, можливість проживання з тваринами тощо. Вебсайт дозволяє перевіряти доступність заміського будинку на потрібні дати, вартість проживання, залишати відгуки, що також відрізняє його від попереднього вебдодатку.

HolidayHomes [5] (рис. 1.3) дозволяє виконувати повноцінний пошук, включаючи і кількість ванн, і кількість спалень, і наявність інтернету, тобто кількість параметрів є розширеною.

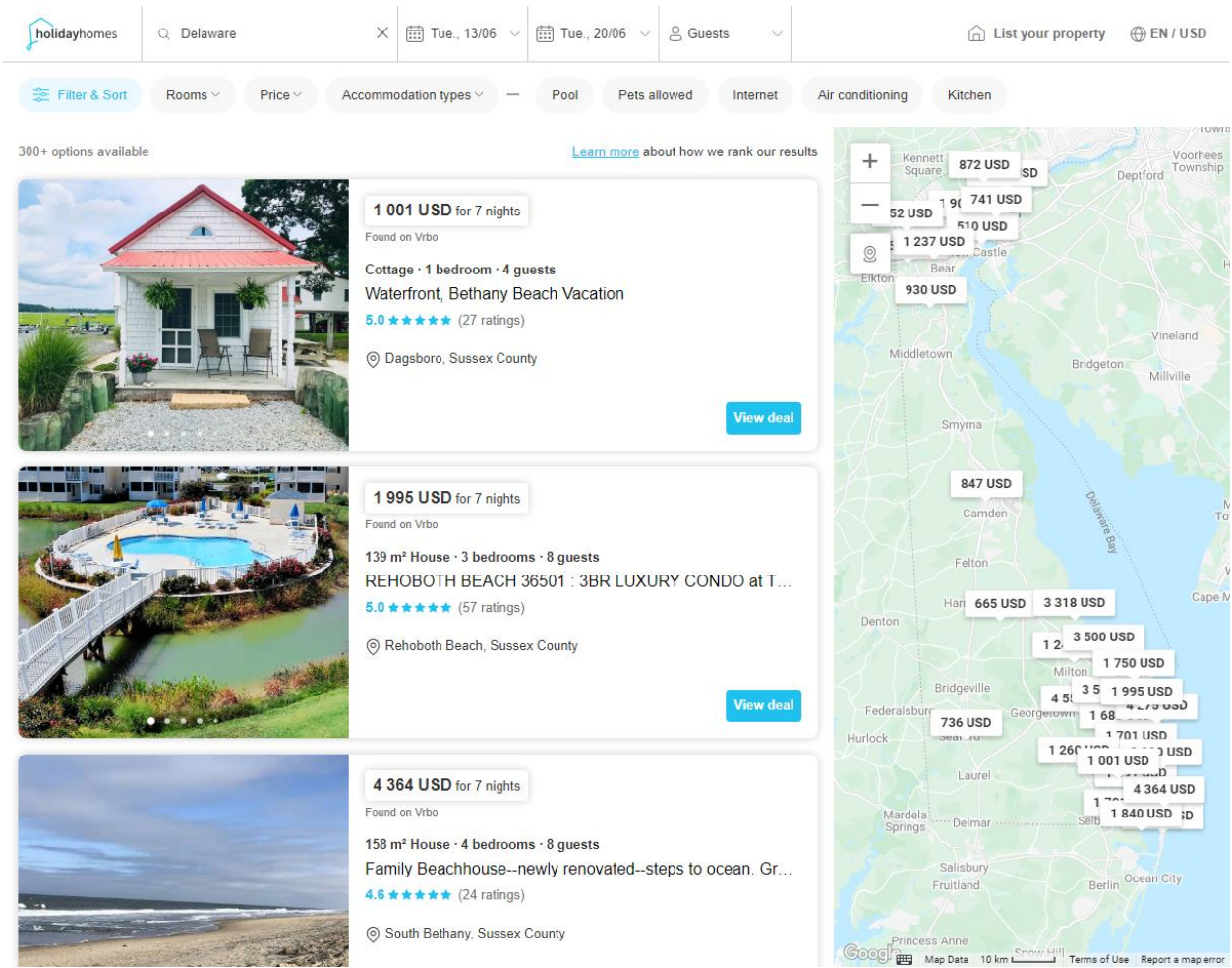


Рисунок 1.3 – Вебдодаток HolidayHomes

На вебсайті можна виставляти власний заміський будинок для оренди, відповідно надається можливість забронювати проживання прямо через цей вебдодаток. Також дозволяється залишати і переглядати відгуки.

В Україні таких спеціалізованих рішень небагато. В основному таку інформацію можна отримати або через вебсайти, які загалом надають дані про будь-яке житло, або через вебсайти, побудовані за принципом дошки оголошень, тобто ніяку специфіку заміських будинків вони не враховують. Прикладом такого вебсайт є Hotelmix [6] (рис. 1.4).

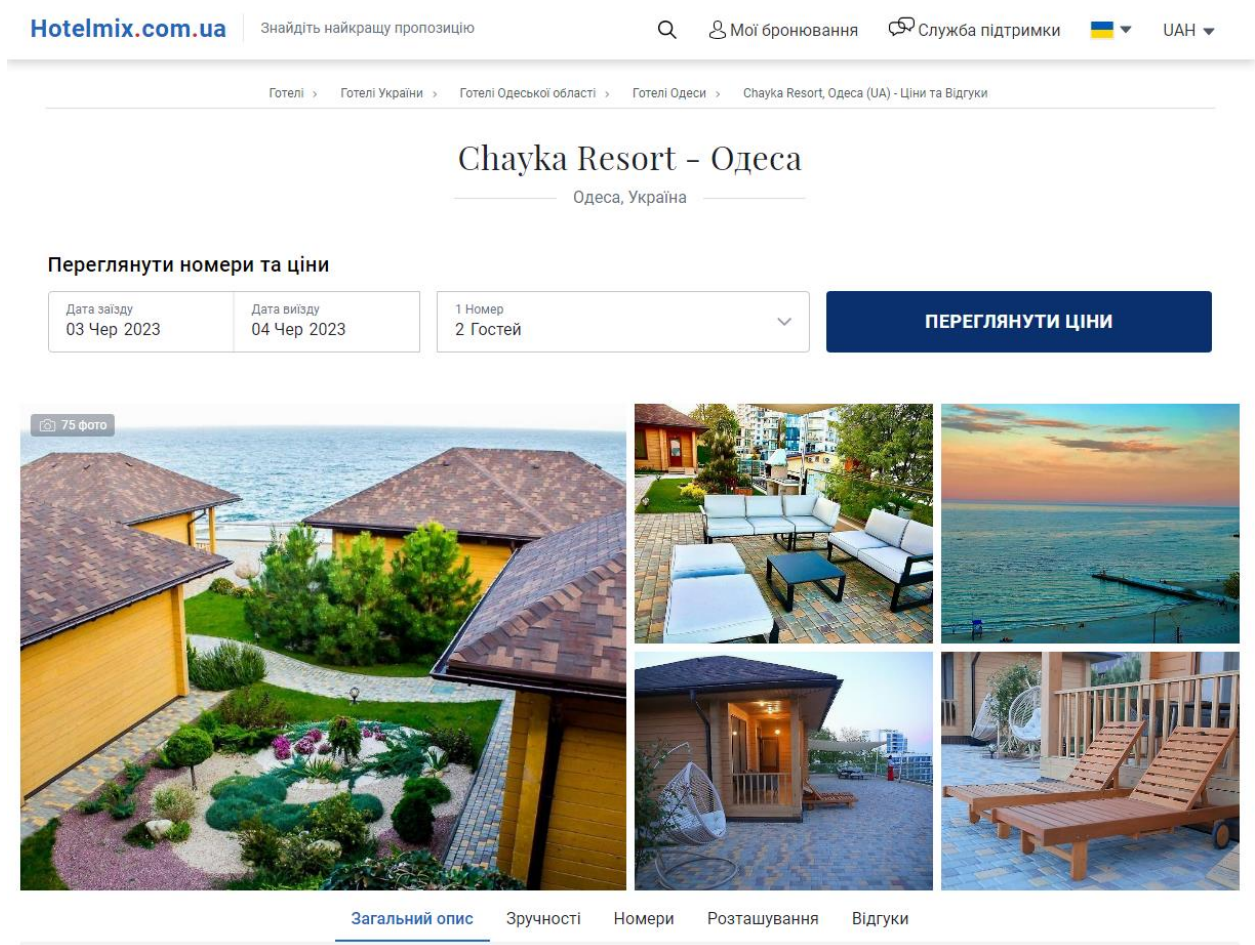


Рисунок 1.4 – Вебдодаток Hotelmix

Hotelmix містить одразу і готелі, і заміські будинки у різних країнах світу. Фактично це вебсайт, який займається підтримкою туризму. Тому розглядати його як потенційний аналог не можна.

1.2 Порівняння аналогів

Аналіз вказав на наявність різних категорій ПЗ, тому до фінального порівняння було виділено представників цих категорій (табл. 1.1).

Таблиця 1.1 – Порівняння аналогів

Показник	CountryHouse- Company	Cottage- Gems	Holiday- Homes	Дипломна розробка
Пошук серед наявних заміських будинків	—	+	+	+
Бронювання оренди	—	—	+	+
Виставлення власних пропозицій через додаток	—	—	+	+
Відгуки	—	+	+	+
Підтримка огляду будинків	—	—	—	+
Визначення деталей приміщень зі структуризацією	—	—	—	+

Дипломна розробка враховує основні можливості аналогів, при цьому надаючи також можливість підтримки огляду будинків, тобто можна запитати у власника або агента, зокрема коли це оренда на тривалий період, можливість оглянути будинок. Також передбачено визначення деталей окремих приміщень зі структуризацією, тобто можна надати за будинком можливість переглянути дані окремої кімнати, визначаючи, що саме розміщено в конкретній кімнаті або приміщенні на території, як саме ця кімната або приміщення виглядають.

1.3 Висновки за розділом 1

Серед аналогів існують програми, які дозволяють виконувати тільки перегляд оголошень про оренду заміських будинків, як належать конкретному агентству, програми, які агрегують оголошення, та програми, де можна виставляти різні пропозиції на самому вебсайті. До останньої категорії належить розробка цієї роботи.

Загальний перелік завдань, що необхідно виконати в роботі:

- виконати аналіз програм, які застосовуються для підтримки оренди заміських будинків;
- визначення технологій розробки і вимог до програми;
- визначення структури організації даних;
- програмна реалізація.

2 ПРОЄКТУВАННЯ ПРОГРАМИ

2.1 Вибір технологій веброзробки

Оскільки створюється у роботі єдина платформа, яка в подальшому може бути розширена додатковими додатками, логічним виглядає створення базового рішення для використання його клієнтами у вигляді вебдодатку. Але перспектива створення єдиної платформи вимагає кросплатформності рішень, що створюються. Однією з основних мов програмування для веброзробки є javascript, а її рішенням для створення платформ (backend) є NodeJS. Python [7]-[8] є мовою загального призначення, що дозволяє створювати вебдодатки, але при цьому надає і можливості роботи з іншими областями, що потенційно дозволяє потім створену програмну платформу розширювати.

Порівняння цих технологій (табл. 2.1) вказало на переваги NodeJS щодо швидкості роботи та певну перевагу python щодо простоти читання і написання коду, простоти відслідковування помилок, що фактично вказує на те, що саме розробку виконати на python є зручнішим, відповідно саме python став основною мовою розробки.

Таблиця 2.1 – Порівняння технологій веброзробки

Показник	python	NodeJS
Тип технології	Мова програмування	Програмна платформа
Кросплатформність	+	+
Простота читання і написання коду	++	+—
Проста відслідковування помилок	++	+—
Швидкість виконання	+—	+
Документування	++	++

2.2 Вимоги до програми

Увесь перелік вимог до функцій, які мають виконуватися в програмі, включає:

- пошук заміських будинків для оренди;
- бронювання оренди заміського будинку;
- встановлення строків можливої оренди заміського будинку;
- оновлення строків можливої оренди заміського будинку;
- скасування вказаного строку можливої оренди заміського будинку;
- отримання переліку заміських будинків у керуванні;
- оновлення інформації про заміський будинок;
- визначення окремих приміщень заміського будинку;
- оновлення і вилучення інформації про окремі приміщення заміського будинку;
- сплата бронювання;
- отримання переліку всіх бронювань оренди заміського будинку;
- отримання переліку скасованих бронювань оренди заміського будинку;
- отримання переліку підтверджених бронювань оренди заміського будинку;
- отримання переліку поточних бронювань оренди заміського будинку;
- отримання переліку завершених бронювань оренди заміського будинку;
- підтвердження бронювання за заявкою, за початком оренди, за завершенням оренди, скасування бронювання;
- запитування огляду будинку;
- підтвердження чи скасування огляду заміського будинку;
- створення облікових запитів агентів (орендодавців);
- авторизація;
- отримання послідовності оренд і оглядів;

- створення спілкування з агентом (орендодавцем);
- надсилання повідомлення в спілкуванні;
- реєстрація користувачів (орендарів);
- отримання переліку всіх запитів на огляд замиського будинку;
- отримання переліку скасованих запитів на огляд замиського будинку;
- отримання переліку підтверджених запитів на огляд замиського будинку;
- оцінювання досвіду оренди замиського будинку;
- визначення пакету страхування;
- оновлення пакету страхування;
- отримання переліку наявних пакетів страхування;
- відображення всіх даних замиського будинку;
- відображення досвіду агента;
- відображення досвіду користувача;
- отримання переліку всіх власноруч створених запитів на огляд;
- отримання переліку скасованих власноруч створених запитів на огляд;
- отримання переліку підтверджених власноруч створених запитів на огляд;
- отримання переліку всіх бронювань користувача;
- отримання переліку скасованих бронювань користувача;
- отримання переліку підтверджених бронювань користувача;
- отримання переліку завершених бронювань користувача;
- отримання переліку проблемних бронювань;
- отримання переліку агентів;
- отримання переліку найкращих замиських будинків за регіоном, за населеним пунктом, за країною.

Усі ці вимоги були далі розділені за сценаріями у відповідності з ролями користувачів. Першою роллю є неавторизований користувач (рис. 2.1). Звичайний орендодавець названий користувачем (рис. 2.2).

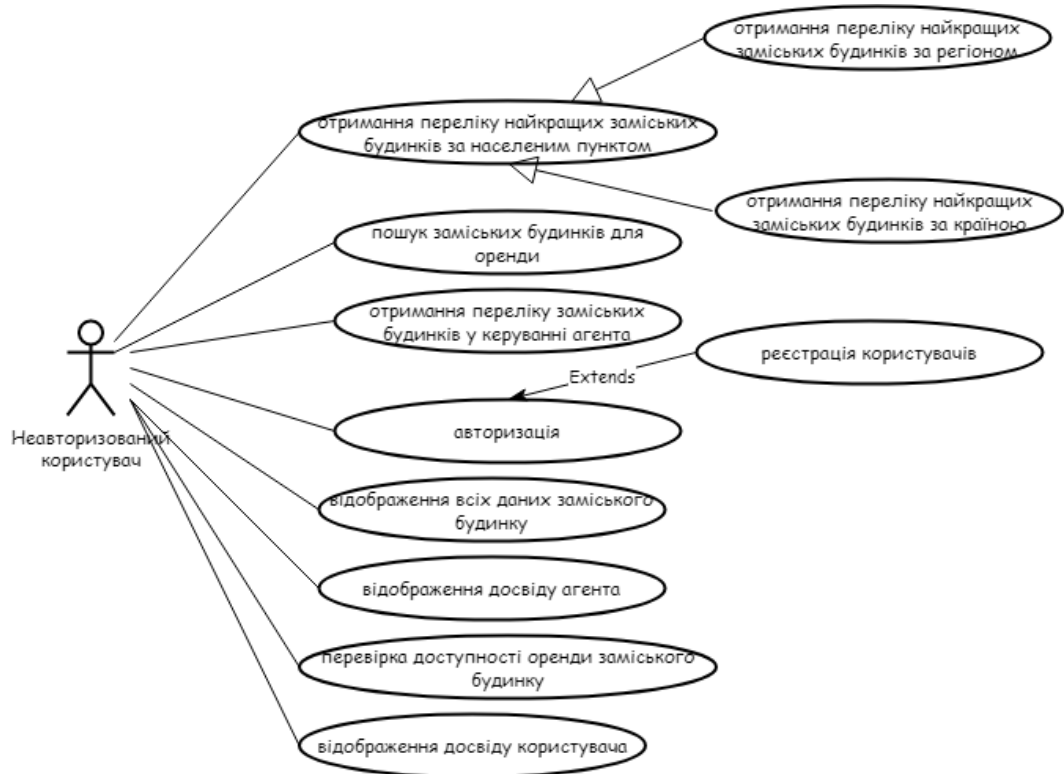


Рисунок 2.1 – Діаграма прецедентів неавторизованого користувача

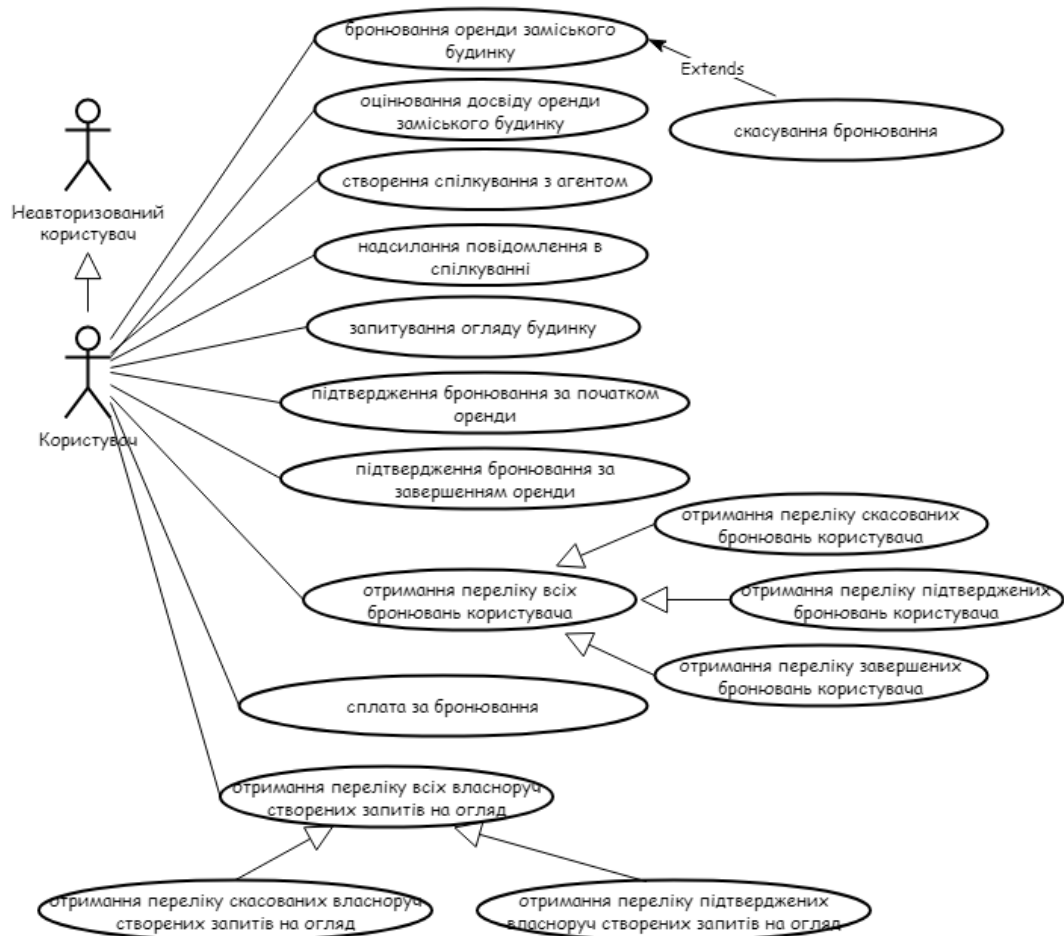


Рисунок 2.2 – Діаграма прецедентів користувача

Виставленням замських будинків займається агент (рис. 2.3).



Рисунок 2.3 – Діаграма прецедентів агента

У програмі представлений менеджер (рис. 2.4), який керує певними даними загалом.

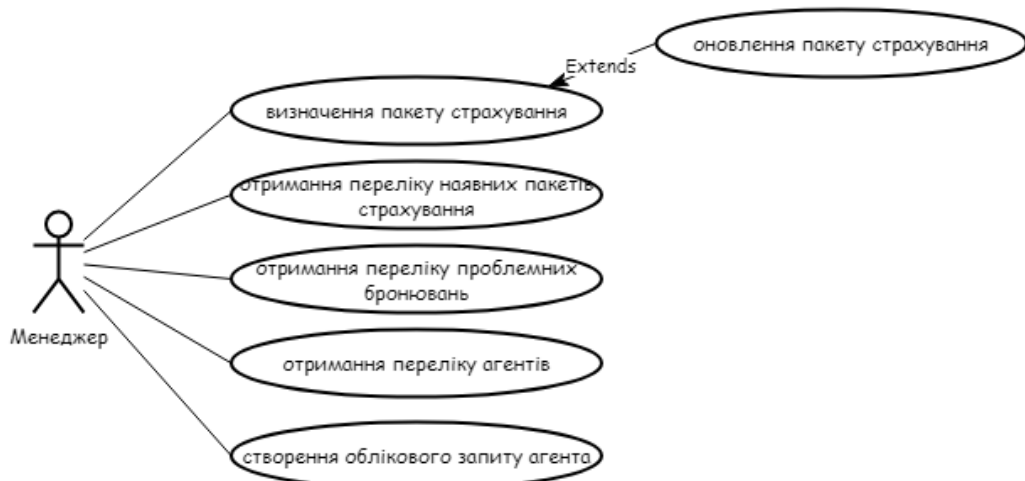


Рисунок 2.4 – Діаграма прецедентів менеджера

2.3 Структура організації даних

Структуру організації даних було визначено за допомогою бази даних (БД) [9] у підтримці з системою MySQL.

Спершу було вирішено питання організації структури адрес, в межах яких виконується пошук заміських будинків. Звичайно, що конкретна адреса є ключовою точкою зокрема організації пошуку, але оскільки це заміські будинки, то пошук часто виконується в межах певних районів (областей). Це може бути дещо різна структура в межах різних країн, але суть залишається. Наприклад, в Україні країна поділена на області: при пошуку часто звертається увага саме на те, в якій області розташовано цей будинок. Даний принцип міг би бути дещо інакшим: наприклад, в країні існують певні історичні регіони (Полісся, Карпати та інші), але фактично на практиці його застосовувати важко, адже люди в Україні рідко орієнтуються в межах цих регіонів, тому в підсумку базовою структурою обрано область. В інших країнах це може бути це може бути певний район, округ або, наприклад, графство, тобто історичний район, якщо людям так зручно орієнтуватися при визначенні сусідства. Тому фактично кожна країна може бути поділена на такі регіони, а конкретний розподіл вже залежить від країни і визначається в підсумку в БД.

Відповідно для такої структуризації адрес спочатку потрібно було визначити перелік країн, які в БД знаходяться в таблиці Country, яка до свого складу включає сам номер країни в полі id, назву її в полі title.

Наступним у структурі є певний регіон, під яким, як було описано, може конкретно для країни матися на увазі район, область, графство або будь-яка інша структура. Перелік регіонів у всіх країнах знаходиться в БД в таблиці Region, яка до свого складу включає сам номер регіону id, номер країни, до якої цей регіон належить, – поле country, яке зв'язує цю таблицю з таблицею Country, назву країни в полі title.

Відповідно найнижчою ланкою саме структуризації є конкретний населений пункт, який використовується як основа адреси. Оскільки будинки

заміські, то найчастіше це село, але знову ж таки може бути використаний і інший тип населеного пункту зокрема в інших країнах: в Україні розділення відбувається на міста, селища і села, тому саме село використано в назві сутності Village, але фактично можуть зберігатися і інші типи базових для відповідної країни населених пунктів. Таблиця Village до свого складу включає сам номер населеного пункту (села) id, номер регіону, до якого цей населений пункт належить, – поле region, яке зв’язує цю таблицю з таблицею Region, назву населеного пункту в полі title.

Іншою важливою сутністю в предметній області звичайно є заміський будинок, адже за його відсутності всі система немає ніякого сенсу. Заміські будинки знаходяться в таблиці House, яка до свого складу включає сам номер заміського будинку id, номер орендодавця, який або володіє будинком або є агентом, який у програмі надає цей будинок для оренди, – поле landlord, яке зв’язує цю таблицю з таблицею Rentuser, назву, що буде використовуватися як назва оголошення, – поле advertisementtitle, деталізацію опису цього заміського будинку particulars, номер типу страхування, за яким мають забезпечуватися страхування (наприклад, може залежати від розміру будинку або його вартості), – поле insurancetype, кількість людей, які одночасно можуть проживати в цьому заміському будинку, – поле people, кількість місць для одночасних обідів – diningplaces (фактично може визначатися кількістю місць у їдальні, що важливо зокрема для організації спільних заходів: конференцій, весіль та інших), оцінка, яку всередньому виставляли орендарі цьому заміському будинку, – поле opinion, час дня, з якого можна виконати заїзд у цей заміський будинок при заселенні, – поле checkin (дуже важливе, оскільки нерідко власники або агенти, які віддають будинок в оренду, знаходяться в іншому місті, тому для зустрічі потрібно приїхати у відповідне село, тобто цей процес стає ще складнішим ніж зустріч під час оренди квартири), час дня, до якого можна виїхати з цього заміського будинку в останній день, – поле checkout, мінімальний вік, з якого приймаються орендарі в цьому будинку, – age, дозвіл на проживання з домашніми тваринами – поле petsallowed, дозвіл

на проживання з дітьми – `childrenallowed`, дозвіл на паління всередині будинку – `smokingallowed`, дозвіл на проведення вечірок або інших масштабних заходів у цьому заміському будинку або на його території – `partiesallowed`, наявність бездротового підключення до інтернету на території заміського будинку – `wifiaccess`, наявність високошвидкісного інтернету загалом (фізичного підключення через кабелі) – `placeaccess`, номер населеного пункту (села), де знаходиться цей заміський будинок, – `location`, яке зв'язує цю таблицю з таблицею `Village`, дата створення цього запису про заміський будинок – `saved`, необхідність перерви в один день для прибирання будинку після завершення оренди – `pause`, мінімальна тривалість бронювання в днях – `period`.

Описана структура таблиці `House` визначає основні характеристики заміського будинку, але далеко не всі можливі. В залежності від того, на скільки людина, яка володіє цим будинком, або агент, який представляє її інтереси, детально вносить інформацію, вона може вирішити додати ще набір деталей. Цей набір деталей може, наприклад, включати наявність басейну, тенісного корту на території, мангалу та іншого: для великих віл це може бути великий обсяг окремих властивостей, які у інших заміських будинків взагалі можуть бути відсутні. Тобто зрозуміло, що далеко не в кожному такому будинку є басейн, тому визначати його як поле таблиці `House` недоцільно, адже тоді велика частина даних буде використовуватися даремно. Тому такі властивості було вирішено виділити в окрему таблицю `Feature`, де спершу визначається, які взагалі в програмі такі властивості можуть бути, охоплюючи номер властивості `id` та її назву `title`. Фактично це може бути басейн, тенісний корт та інше.

Ці властивості визначаються не просто для існування в програмі, а для того щоб потім людина, яка володіє заміським будинком, або агент, який представляє її інтереси, міг визначити, яка властивість якому конкретно будинку відповідає. Збереження цих даних виконується на основ таблиці БД `HouseFeature`, яка до свого складу включає сам номер співвіднесення

властивості з замиським будинком – поле `id`, номер замиського будинку, до якого ця властивість належить, – поле `house`, яке зв'язує цю таблицю з таблицею `House`, властивість, яка представлена в цьому замиському будинку, – поле `feature`, яке зв'язує цю таблицю з таблицею `Feature`, кількість, якою характеризується ця властивість, – поле `quantity`.

Важливими для представлення замиських будинків є фотографії цих будинків. Вони прив'язуються до конкретного замиського будинку, а інформація про такі співвідношення знаходиться в таблиці `Housephoto`, яка до свого складу включає сам номер фотографії замиського будинку `id`, номер самого замиського будинку, до якого відноситься ця фотографія, – поле `house`, яке зв'язує цю таблицю з таблицею `House`, фотографія, яку збережено для цього замиського будинку, – поле `photo`, деталізацію опису того, що саме демонструє ця фотографія замиського будинку, – поле `particulars`.

Замиські будинки відрізняються від стандартних квартир тим, що вони можуть представляти собою достатньо великі приміщення, які можуть використовуватися, наприклад, для проживання декількох сімей, відповідно важливим стає чітке розуміння влаштованості такого замиського будинку всередині. Потрібно визначити, які саме кімнати присутні, які можливо приміщення існують. Наприклад, варто розуміти, що з себе представляє кожна ванна кімната, яка присутня в будинку, якою є кухня, якими є інші кімнати. Але не тільки кімнати входять до складу замиського будинку. Це може бути і гараж, і певне додаткове приміщення. Всі вони відносяться до конкретного замиського будинку та зберігаються в таблиці `Houseobject`, яка до свого складу включає сам номер `id`, номер замиського будинку, до якого належить це приміщення (об'єкт), – поле `house`, яке зв'язує цю таблицю з таблицею `House`, назву об'єкту – поле `objecttitle`, деталізацію опису цього конкретно об'єкту або приміщення (це вже не опис всього замиського будинку, а саме цієї конкретно, наприклад, кімнати) – поле `particulars`, порядковий номер, за яким буде відображатися це приміщення у складі даних всього замиського будинку, – поле `position`, номер типу об'єкта (ванна кімната, кухня, вітальня, їдальня,

підвальне приміщення, спальна кімната, гараж та таке інше) – поле `objecttype`, яке зв'язує цю таблицю з таблицею `Objecttype`, кількість людей, які можуть знаходитися в цьому приміщенні, – поле `people`.

Можливі типи об'єктів (приміщень), які використовуються для визначення складових заміських будинків, знаходяться в таблиці `Objecttype`, що за стандартною структурою включає номер типу `id` та його назву `title`.

Властивості, які були визначені для заміського будинку, можуть бути співвіднесені з конкретним приміщенням. Наприклад, наявність джакузі може бути визначена для конкретної ванни, в якій вказано, що саме в ній знаходиться це джакузі. Саме співвідношення між властивостями заміського будинку та об'єктами цього будинку визначено таблицею `HouseobjectFeature`, яка до свого складу включає номер співвідношення між властивістю заміського будинку та об'єктом цього будинку – поле `id`, властивість заміського будинку, яка визначається у співвідношенні, – поле `housefeature`, яке зв'язує цю таблицю з таблицею `HouseFeature`, об'єкт (приміщення) заміського будинку, до якого відноситься ця властивість, – поле `houseobject`, яке зв'язує цю таблицю з таблицею `Houseobject`.

Фотографії для представлення саме конкретного приміщення заміського будинку, а не всього будинку загалом, знаходяться в таблиці `Houseobjectphoto`, що до свого складу включає номер фотографії приміщення заміського будинку `id`, приміщення заміського будинку – поле `houseobject`, яке зв'язує цю таблицю з таблицею `Houseobject`, саму фотографію приміщення заміського будинку – поле `photo`, опис деталей, що показані на цій фотографії, – поле `particulars`.

Самі оренди заміських будинків у програмі визначаються їх бронюваннями. У подальшому за бронюванням людина, яка володіє будинком, або агент підтверджують це бронювання, а потім обидві сторони вказують, що вони заїхали в будинок і відповідно здали в оренду будинок, в кінці – відповідно протилежно: що одні виїхали, а інші отримали зданий будинок. Усе це входить до поняття бронювання і визначено таблицею `Order`,

яка до свого складу включає номер бронювання оренди заміського будинку – поле `id`, номер самого заміського будинку, бронювання якого виконується, – поле `house`, яке зв'язує цю таблицю з таблицею `House`, користувача, який забронював оренду цього заміського будинку, – поле `rentuser`, яке зв'язує цю таблицю з таблицею `Rentuser`, коментар до бронювання зі сторони орендаря, – поле `comment`, дату початку самої оренди заміського будинку – `date1` та дату, коли заміський будинок має бути повернений з оренди, – поле `date2`, підтвердження певного етапу цього бронювання – поле `confirmed` (бронювання спочатку непідтверджене, потім може бути підтвердженим попередньо, тобто оренду ще не розпочато, або скасованим, потім бронювання може розпочатись і бути підтвердженим окремо зі сторони орендодавця або зі сторони орендаря, потім теж саме підтвердження повторюється за завершенням оренди), вартість бронювання – поле `price`, страхування, яке застосовано до цього бронювання, – поле `insurance`, яке зв'язує цю таблицю з таблицею `Insurance`, дата бронювання – поле `saved`.

При пошуку заміських будинків потрібно розуміти, коли вони є вільними. Це визначається з двох боків. По-перше, сам орендодавець, тобто людина, яка володіє будинком, або агент визначає, коли він взагалі готовий здавати в оренду цей будинок. По-друге, за кожного бронювання, коли воно виявляється підтвердженим орендодавцем, тобто коли за ним можна здійснювати оренду, потрібно зменшувати цей проміжок часу. Це дозволить потім функціонально не перевіряти кожен раз весь масив бронювань, щоб зрозуміти, коли цей будинок можна забронювати, а коли ні, а одразу мати підготовлені дані про те, що цей проміжок дат недоступний. Обидві ці сторони проблеми в підсумку визначаються таблицею `Rentable`, яка до свого складу включає сам номер доступного проміжку дат для оренди заміського будинку – поле `id`, номер заміського будинку, до якого ця інформація відноситься, – поле `house`, початок доступного проміжку дат – `date1`, закінчення проміжку дат – `date2`.

Користувачі програми усі об'єднані в одну сутність. Під час реєстрації при цьому має бути зазначено, яку роль під конкретно цим обліковим записом збирається виконувати користувач. Тобто одним і тим самим обліковим записом не можна скористатися і для того, щоб бронювати оренду заміського будинку, і для того щоб пропонувати свій заміський будинок для оренди. Дані користувачів зберігаються в таблиці Rentuser, яка до свого складу включає номер користувача id, логін – поле login, пароль – поле password, фотографію – photo, оцінку, яку всередньому виставляли цьому користувачу, – поле opinion, ім'я повне користувача – поле name, деталі про користувача – поле particulars, роль користувача (орендар або орендодавець) – поле role.

Оскільки оренда заміських будинків пов'язана з певними ризиками, зокрема тими, що майно може бути зіпсовано, то в програмі передбачено наявність варіантів страховок, які можуть бути оформлені для захисту інтересів під час оренди. Страхувальники, тобто самі страхові компанії, зберігаються в таблиці Insurer, яка до свого складу включає номер страхувальника id, назву компанії – поле company, адресу, за якою розташована компанія – address та вебсайт – поле website.

Самі пакети страхування, які надаються страхувальниками, зберігаються в таблиці Insurance, яка до свого складу включає номер пакету страхування id, номер страхувальника – поле insurer, яке зв'язує цю таблицю з таблицею Insurer, назву пакету – поле title, деталі пакету у вигляді опису – поле particulars, вартість пакету – поле price, тип страхувального об'єкта, до якого застосовується цей пакет, – поле insurancetype, кількість днів, на яку діє цей пакет, – поле quantity.

Оцінювання користувачами, які задіяні в оренді заміського будинку, застосовуються за кожним бронюванням. Тобто орендар може оцінити заміський будинок з конкретного бронювання, орендодавець – орендаря. Оцінювання зберігаються в таблиці HouseOpinion, яка до свого складу включає номер оцінювання id, номер бронювання, згідно з яким реалізується це оцінювання, – поле order, визначення факту, що це оцінювання виконано

орендодавцем, тобто тоді воно про того орендаря, який оформив бронювання, або що це оцінювання виконано орендарем, тобто тоді воно про той замський будинок, який орендувався, – поле `rentler` (2 можливих значення), дату оцінювання – поле `saved`, оцінку – поле `number`, враження щодо оренди – поле `opinion`.

Якщо орендар зацікавився певним замським будинком, то у нього може виникнути бажання уточнити певне питання. Для цього він може написати повідомлення користувачу, який в межах програми володіє цим будинком, а фактично є або власником цього замського будинку, або агентом, який представляє інтереси власника. Усі повідомлення направляються між певними двома сторонами, відповідно є деяке спілкування, в межах якого надсилаються повідомлення. При спробі відправити перше повідомлення таке спілкування має створюватися. Спілкування зберігаються в таблиці `Talk`, яка до свого складу включає номер спілкування `id`, номер орендодавця, до якого був спочатку адресований запит у повідомленні, – поле `landlord`, яке зв'язує цю таблицю з таблицею `Rentuser`, номер орендаря, який створив запит спочатку у повідомленні, – поле `rentler`, яке зв'язує цю таблицю з таблицею `Rentuser`, дату створення спілкування – поле `saved`.

Самі повідомлення належать до конкретних спілкувань і зберігаються в таблиці `Message`, яка до свого складу включає номер повідомлення `id`, номер спілкування, до якого відноситься повідомлення, – поле `talk`, визначення авторства повідомлення – поле `creator` (2 можливих значення), час відправлення – поле `saved`, текст повідомлення – поле `messagetext`.

Для конкретного замського будинку може передбачатися, що вартість в різний період часу є різною. Так може бути, наприклад, на свята або в різний період часу в гірській місцевості або за інших подібних ситуацій. Умови щодо вартості оренди замського будинку знаходяться в таблиці `Prisepperiod`, яка до свого складу включає номер цієї умови – поле `id`, номер замського будинку, для якого визначено цю умову, – поле `house`, яке зв'язує цю таблицю з таблицею `House`, вартість оренди однієї доби за такої умови – поле `price`, дату

початку оренди, з якої діє ця умова, – поле rentstart, дату завершення оренди, до якої діє ця умова, – поле rentend (рис. 2.5).

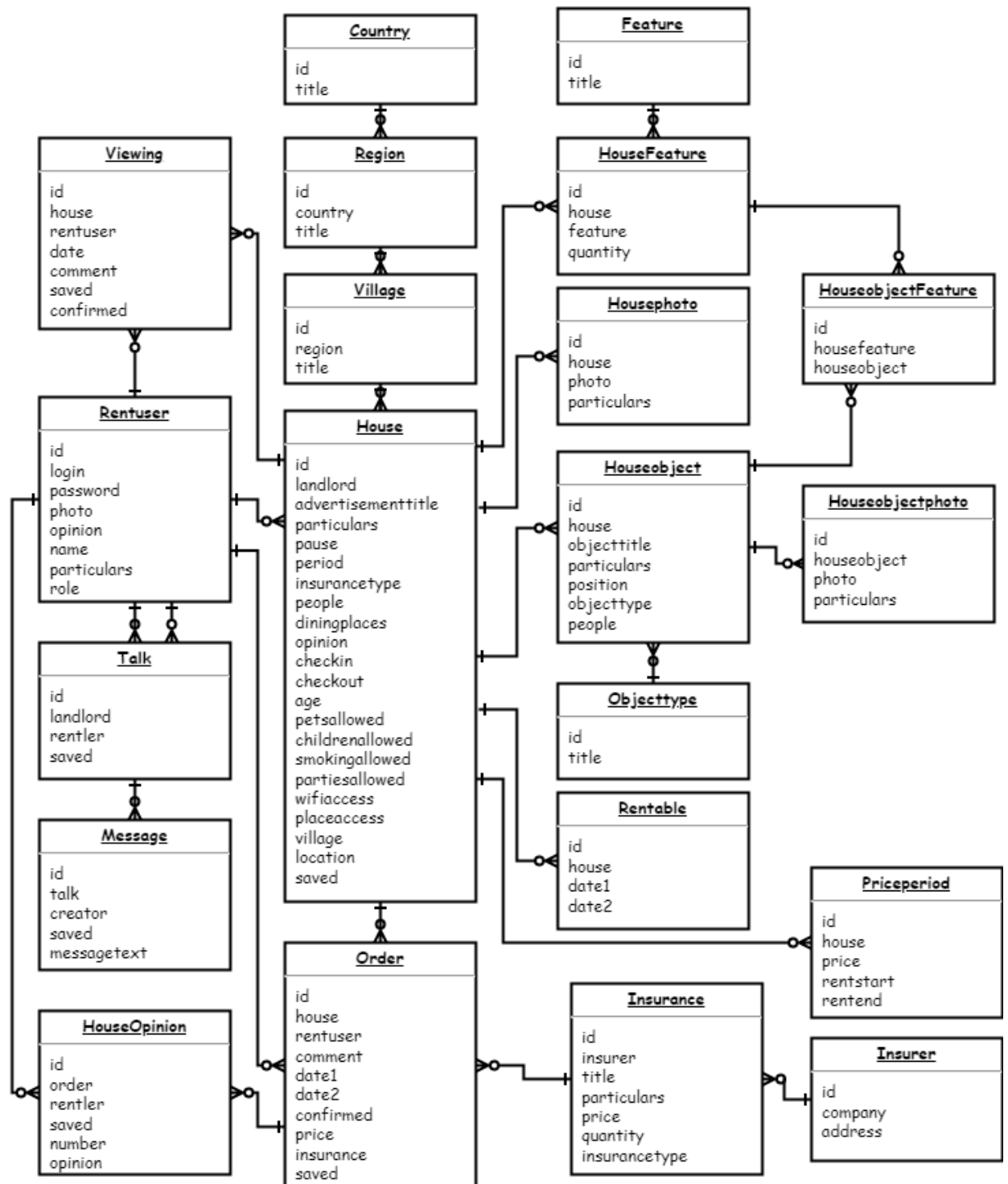


Рисунок 2.5 – Схема БД

Оскільки заміські будинки можуть орендуватися на тривалий період, то абсолютно стандартним у такому випадку може бути огляд будинку. Тобто

орендар перед тим, як орендувати будинок, бажає ознайомитися з ним, запитуючи відповідний огляд, а власник або погоджується на це, або ні. Дані переглядів замських будинків зберігаються в таблиці Viewing, яка до свого складу включає номер огляду id, номер замського будинку, огляд якого виконується, – поле house, яке зв’язує цю таблицю з таблицею House, орендар, який звернувся з цим запитом – поле rentuser, яке зв’язує цю таблицю з таблицею Rentuser, час проведення огляду – поле date, коментар до огляду – поле comment, дату створення запиту – поле saved, підтвердження огляду – поле confirmed (без відповіді, скасовано, підтверджено).

2.4 Висновки за розділом 2

У обраній структурі БД було враховано можливість різної структуризації регіонів у різних країнах, можливість деталізації інформації за окремими приміщеннями замського будинку, можливість необхідності визначення різноманітних властивостей за окремими приміщеннями, можливість змінювання ціни оренди в різні періоди часу, а також підтримку пакетів страхування, що визначаються страхувальниками. Необхідність забезпечення роботи з цією структурою даних користувача підкреслена сценаріями використання програми. Реалізацію програми має бути виконано мовою python.

3 РОЗРОБКА ПРОГРАМИ

3.1 Порядок функціонування програми

Програма головним чином побудована за допомогою меню, склад функцій в якому змінюється з ролями користувача (рис. 3.1).

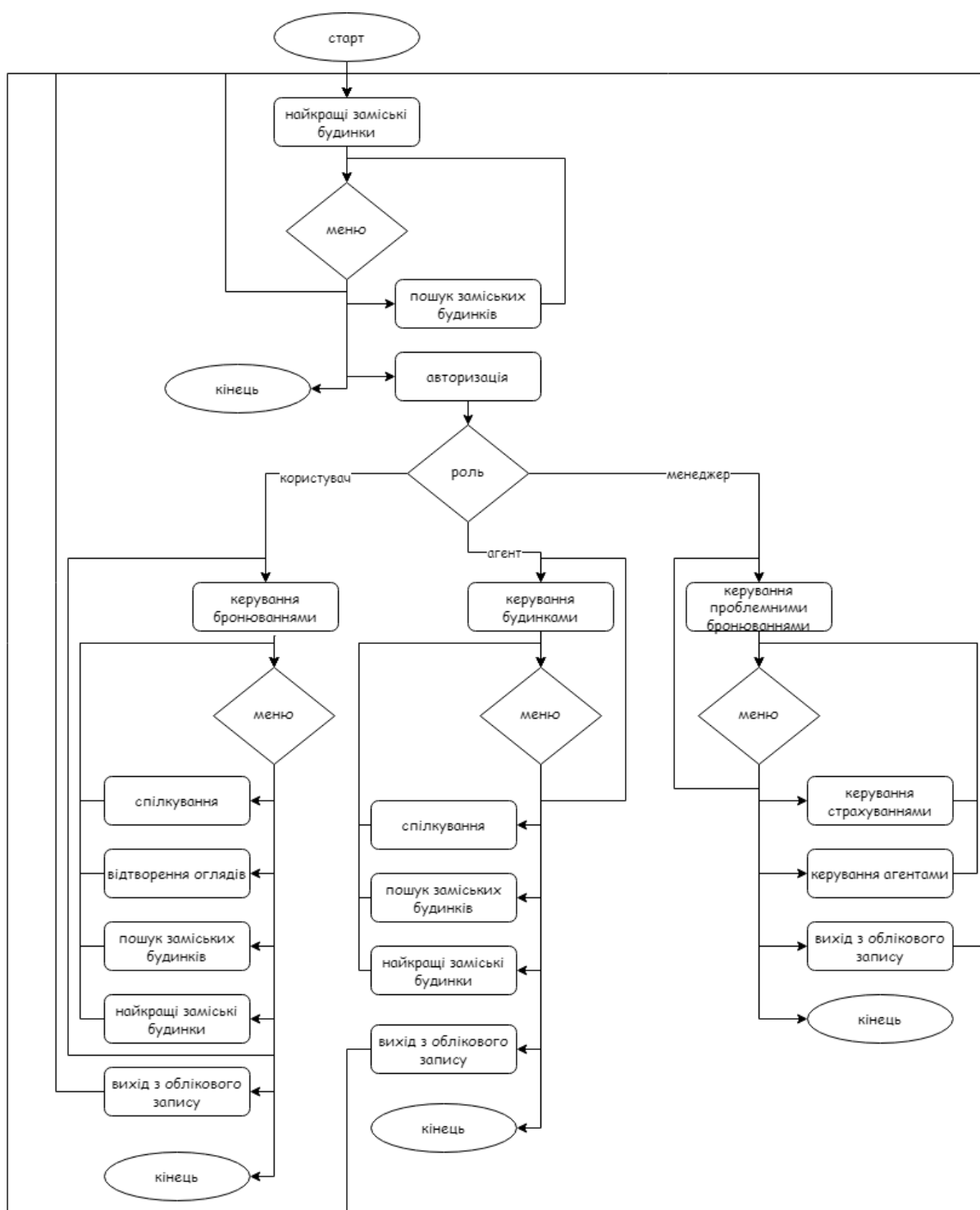


Рисунок 3.1 – Схема функціонування програми

3.2 Реалізація функцій програмної системи

Оскільки для реалізації ПЗ як вебдодатку потрібно виконувати обробку багатьох процесів, пов'язаних зі взаємодією з вебсервером, то для спрощення та приведення розробки до використання сучасних технологій, які як раз і дозволяють таке спрощення виконати, реалізація виконувалась з підтримкою фреймворку django.

Для забезпечення вихідних сторінок, які демонструються користувачу у веббраузері, були створені функції, які на основі взаємодії з БД визначають, які дані під конкретну функціональну вимогу потрібні та яким чином їх або внести в цю сторінку, або навпаки витягти з неї. Нижче описані такі розроблені функції. Створене ПЗ не обмежується тільки цими функціями, включаючи і самі шаблони сторінок, що необхідно для визначення того, якою буде розмітка кожної сторінки під виконання відповідної функції. Визначення самих стилів при створенні розмітки не було завданням роботи, тому було застосовано вільний CSS-шаблон [10], за допомогою якого вже було визначено розмітку кожної конкретної сторінки для підтримки відповідної функції.

Отримання переліку найкращих заміських будинків за регіоном передбачає витягання з БД всіх заміських будинків, які розташовані за зменшенням середньої оцінки з витяганням фотографій заміських будинків, які розташовані в регіоні з номером `regionId`, за допомогою функції `RegionHouseRatingView`. У результаті з шаблону `regionrating.html` доступний словник даних з ключами `houses` з самим переліком таких заміських будинків та `region` з даними про регіон.

Отримання переліку найкращих заміських будинків за населеним пунктом передбачає витягання з БД всіх заміських будинків, які розташовані за зменшенням середньої оцінки з витяганням фотографій заміських будинків, які розташовані в населеному пункті з номером `villageId`, за допомогою функції `VillageHouseRatingView`. У результаті з шаблону `villagerating.html`

доступний словник даних з ключами `houses` з самим переліком таких заміських будинків та `village` з даними про населений пункт.

Отримання переліку найкращих заміських будинків за країною передбачає витягання з БД всіх заміських будинків, які розташовані за зменшенням середньої оцінки з витяганням фотографій заміських будинків, які розташовані в країні з номером `countryId`, за допомогою функції `CountryHouseRatingView`. Оскільки країна в будинку на пряму не вказується, то відбувається відбирання тих будинків, які розташовані у населеному пункті, який знаходиться в регіоні, який знаходиться в заданій країні. У результаті з шаблону `countryrating.html` доступний словник даних з ключами `houses` з самим переліком таких заміських будинків та `country` з даними про країну.

Отримання переліку заміських будинків у керуванні агента передбачає витягання всіх заміських будинків, якими володіє або керує агент з номером `landlordId`, за допомогою функції `LandlordHousesView`. У результаті з шаблону `landlordhouses.html` доступний словник даних з ключами `houses` з самим переліком таких заміських будинків.

Окремо забезпечено отримання переліку заміських будинків у керуванні агента, який авторизувався, що передбачає витягання всіх заміських будинків, якими володіє або керує цей агент, інформація про якого витягається з даних авторизованого користувачього запису, за допомогою функції `MyHousesView`. У результаті з шаблону доступний словник даних з ключами `houses` з самим переліком таких заміських будинків.

Перевірка доступності та вартості оренди заміського будинку передбачає встановлення факту того, чи заміський будинок з номером `housed` у період дат з `rent1` до `rent2` доступний для оренди, та вартості такої оренди, якщо вона можлива, з урахуванням вартості страхування з номером `insuranceId`, якщо останнє встановлено, за допомогою функції `CheckHouseRentView`. У результаті з шаблону `checkrent.html` доступний словник даних з ключами `price` з визначеною вартістю такої оренди, `rentable` з бульовим значенням того, чи можна орендувати цей будинок в цей час,

rentables з переліком всіх відкритих для бронювання дат, але не раніше поточного часу, insurances з переліком доступних страхувань для цього заміського будинку.

Визначення пакету страхування передбачає за допомогою форми, що ґрунтується на моделі Insurance, створення відповідного об'єкту і внесення його в БД. Усі дані вносяться через форму, а збереження дозволяється тільки тоді, коли авторизований користувач є менеджером. Інакше, навіть отримавши необхідне посилання, користувач не зможе внести необхідні дані, а більше того – навіть отримати саму форму. Виводиться повідомлення на сторінці про виявлену проблему з доступом. Ця робота підтримується функцією InsuranceView.

Оновлення пакету страхування забезпечується за допомогою функції UpdateInsuranceView. Вона дуже подібна до порядку організації попередньої функції InsuranceView. Вона також ґрунтується на тій самій моделі та на тій самій формі. При цьому при відкритті сторінки форма заповнюється даними пакету страхування з номером insuranceId. Якщо дані внесені та зроблено це коректно, то дані зберігаються. Якщо користувач не є менеджером, то йому також не буде надано ніяких даних, форма не буде виведена. Ці ж принципи захисту використовуються у всіх функціях, які забезпечують внесення даних в БД.

Отримання переліку наявних пакетів страхування передбачає витягання з БД абсолютно всіх пакетів страхування. У результаті з шаблону insurances.html доступний словник даних з ключем insurances з самим переліком пакетів страхування.

Відображення всіх даних заміського будинку передбачає витягання всіх даних заміського будинку з номером houseId за допомогою функції HouseView. Для цього витягаються всі необхідні пов'язані дані, включаючи фотографії цього заміського будинку, дані агента, дані населеного пункту, регіону, країни, об'єкти (приміщення) будинку, фотографії цих об'єктів, властивості заміського будинку, властивості об'єктів будинку. Ці дані для

шаблону `house.html` вносяться в словник за ключем `house`, а за ключем `opinions` містяться оцінювання будинку, за ключем `insurances` всі пакети страхування, які можуть бути використані для того типу страхування будинку, до якого відноситься цей заміський будинок.

Пошук заміських будинків для оренди реалізований за допомогою створеної функції `HouseSearchView`. Вона реалізує поетапне визначення вимог до заміських будинків, що в підсумку дозволяє сформулювати повний набір вимог і за ним відібрати тільки ті заміські будинки, які ним відповідають. Вимоги визначаються параметрами функції. Параметр `villageId` визначає номер населеного пункту, в якому має знаходитись заміський будинок. Це стандартне визначення місця пошуку, але воно може бути і пропущене. Альтернативними варіантами є визначення номеру регіону за параметром `regionId` та номеру країни розташування заміського будинку за параметром `countryId`. Встановити значення декількох з цих параметрів фактично не дозволяє форма пошуку на сторінці `houses.html`: можна визначити тільки один з них. Параметр `people` встановлює кількість людей, які мають проживати в будинку: відповідно всі заміські будинки з меншою максимальною кількістю жителів відкидаються. Необхідна кількість місць у їдальні визначається параметром `diningplaces`. Параметр `opinionLevel` встановлює мінімальний рівень середньої оцінки заміського будинку. Параметр `age` визначає мінімальний вік орендарів. Параметр `pets` встановлює відношення в будинку до домашніх тварин: значення 1 – дозволено, 2 – заборонено, 0 – будь-яке значення. Параметр `children` встановлює вимоги до сімей з дітьми в будинку: значення 1 – дозволено перебувати з дітьми, 2 – заборонено, 0 – будь-яке значення. Параметр `smoking` встановлює правила щодо паління: значення 1 – дозволено, 2 – заборонено, 0 – будь-яке значення. Параметр `parties` встановлює правила щодо проведення вечірок: значення 1 – дозволено, 2 – заборонено, 0 – будь-яке значення. Параметр `wifi` визначає потребу в безпроводному доступі до Інтернету: якщо значення встановлено, то доступ має бути забезпечено. Параметр `place` визначає потребу в доступі до Інтернету: якщо значення

встановлено, то доступ має бути забезпечено. Параметр `checkin` визначає найраніший час, коли орендар готовий заселитися в будинок протягом дня. Параметр `checkout` визначає найпізніший час, коли орендар готовий виселитися з будинку протягом дня.

Після виконання відбору будинків за вказаними параметрами, далі виконується перевірка можливості саме оренди цих будинків. Для цього спочатку забезпечується перевірка того, що є вільні доступні для оренди дати на період оренди, вказаний в пошуку. Період оренди вказується параметрами `rentalBegin` і `rentalEnd`. Для перевірки використовується модель `Rentable`. Якщо знайдено вільний проміжок, який повністю охоплює вказаний період або якщо знайдено два проміжки, які охоплюють разом цей період, тобто закінчення одного проміжку є початком іншого, то результатом є підтвердження можливості оренди заміського будинку, він відповідно залишається в результатах до наступного етапу. Також перевіряється, що тривалість оренди в параметрах пошуку не менше мінімальної можливості, за якою агент дозволяє орендувати цей будинок. На наступному етапі виконується обчислення вартості оренди. Вартість порівнюється з інтервалом вартості, заданим у пошуку, від `bottomPrice` до `topPrice`. Якщо заміський будинок має відповідну вартість оренди, то він входить до підсумкових результатів пошуку, які впорядковуються за середньою оцінкою. Результати заповнюють шаблон сторінки через ключ `houses`.

Відображення досвіду агента забезпечується за допомогою функції `LandlordView`, яка визначає номер агента параметром `landlordId`. Окрім даних самого агента, які передаються в шаблон `landlord.html` за ключем `landlord`, до них додаються дані 5 заміських будинків цього агента, які мають найбільшу середню оцінку, за ключем `houses` та 5 останніх оцінювань одного з заміських будинків цього агента за ключем `opinions`.

Відображення досвіду користувача забезпечується за допомогою функції `RentlerView`, яка визначає номер користувача параметром `rentlerId`. Окрім даних самого користувача, які передаються в шаблон `rentler.html` за

ключем `rentler`, до них додаються дані 5 останніх оцінювань агентами цього користувача як орендаря за ключем `opinions`.

Оцінювання досвіду оренди заміського будинку забезпечується функцією `HouseOpinionView`, в параметрі `orderId` якої визначається бронювання, якого стосується це оцінювання. Оцінювання не може виконуватися без попереднього бронювання. В основі використовується форма для створення оцінювання, але визначення напрямку відбувається програмно. Якщо користувач є агентом, то при цьому перевіряється, що він є власником заміського будинку, який визначений в бронюванні, номер якого переданий як параметр. Якщо користувач є орендарем, то перевіряється, що він замовляв це бронювання. У випадку успішного проходження перевірки форми при збереженні даних окрім внесення оцінювання також перераховується середній бал заміського будинку та агента, якщо користувач був орендарем, або середній бал самого користувача-орендаря, якщо користувач, що створив оцінювання був агентом.

Бронювання оренди заміського будинку підтримується функцією `OrderView`, яка має параметр `housed` з номером заміського будинку та параметр `insuranceId` з номером пакету страхування, який може бути і пропущений. При виконанні бронювання реалізується перевірка доступності заміського будинку для оренди на вказані у формі дати. Для перевірки використовується модель `Rentable`. Якщо знайдено вільний проміжок, який повністю охоплює вказаний період або якщо знайдено два проміжки, які охоплюють разом цей період, тобто закінчення одного проміжку є початком іншого, то результатом є підтвердження можливості оренди. Також перевіряється, що тривалість оренди в параметрах пошуку не менше мінімальної можливості, за якою агент дозволяє орендувати цей будинок. Якщо заміський будинок можна орендувати, то обчислюється вартість його оренди на вказані дати з додаванням вартості страхування на ці дати. Виконані обчислення зберігаються в бронюванні в БД.

Встановлення строків можливої оренди заміського будинку реалізовано функцією `RentableView`, де параметр `houseId` визначає номер заміського будинку, для якого в формі встановлюються параметри строків. Використовується шаблон `rentable.html`. Оновлення строків можливої оренди заміського будинку реалізовано функцією `UpdateRentableView`. Параметр `rentableId` визначає номер строку, який було створено, а всі інші параметри задаються через ту ж форму, що і при створенні. Скасування вказаного строку можливої оренди заміського будинку реалізовано в функції `DeleteRentableView`, де строк оренди з номером `rentableId` вилучається з БД, якщо авторизований користувач є агентом цього заміського будинку, якого стосується цей строк.

Визначення оренди власного заміського будинку виконується функцією `AddHouseView`. Внесення всіх параметрів відбувається через форму `FormHouse`. Оновлення інформації про заміський будинок виконується функцією `UpdateHouseView`, параметр `houseId` якої визначає номер заміського будинку, оновлення інформації про який потрібно виконати.

Визначення окремих приміщень заміського будинку реалізовано функцією `AddHouseobjectView`. Працює за допомогою функції `FormHouseobject`, яка дозволяє вносити дані об'єктів (приміщень) заміського будинку. Номер будинку задається параметром `houseId`. Редагування інформації про окреме приміщення з номером `houseobjectId` заміського будинку відбувається за допомогою тієї самої форми та функції `UpdateHouseobjectView`. Вилучення цих об'єктів відбувається за допомогою функції `DeleteHouseobjectView`.

Отримання переліку бронювань оренди заміського будинку з підтримкою визначення всіх бронювань реалізується функцією `AllOrdersView`, яка має параметр `houseId`, що визначає номер самого заміського будинку. Перелік бронювань формується тільки для користувача, який є агентом і відповідає за цей будинок. Якщо це невірно, то перелік не формується, користувач не зможе отримати дані. Отримання переліку скасованих

бронювань оренди заміського будинку з номером `houseId` реалізується функцією `CancelledOrdersView`. Ці функції відповідні шаблони заповнюють за ключем `orders`. Отримання переліку підтверджених, але не розпочатих бронювань оренди заміського будинку з номером `houseId` реалізується функцією `ConfirmedOrdersView`. Шаблон `condirmedorders` заповнюється даними з ключем `orders`, де знаходяться підтверджені бронювання за заміським будинком з номером `houseId`. Окремо за ключем `oldorders` виділяються підтверджені бронювання, які не було розпочато, тобто їх виконання в минулому, але вони не були підтверджені жодною зі сторін. Отримання переліку поточних бронювань оренди заміського будинку з номером `houseId` реалізується функцією `StartedOrdersView`. За ключем `orders` у шаблоні сторінки `startedorders.html` формуються бронювання, які були визначені як розпочаті однією зі сторін або обома. Отримання переліку завершених бронювань оренди заміського будинку з номером `houseId` реалізується функцією `FinishedOrdersView`. Сюди відносяться всі бронювання, які знаходяться в стані завершених за думкою однієї зі сторін або обох одразу. Отримання переліку сплачених бронювань оренди заміського будинку з номером `houseId` реалізується функцією `PaidOrdersView`.

Отримання переліку всіх бронювань авторизованого користувача як орендаря реалізується функцією `AllRenterOrdersView`. Отримання переліку скасованих бронювань авторизованого користувача реалізується функцією `CancelledRenterOrdersView`. Отримання переліку завершених бронювань авторизованого користувача реалізується функцією `ConfirmedDoneRenterOrdersView`. Отримання переліку підтверджених, але не завершених повністю бронювань авторизованого користувача реалізується функцією `ConfirmedDoneRenterOrdersView`.

Скасування бронювання реалізовано функцією `CancelOrderView`, параметр `orderId` якої визначає номер бронювання, яке необхідно виконати. Скасування реалізується тільки в тих випадках, якщо користувач є орендарем та створив це бронювання, що знаходиться в стані підтвердженого або тільки

створеного, або якщо користувач є агентом, який відповідає за заміський будинок, до якого стосується це бронювання, та бронювання при цьому знаходиться в стані підтвердженого або тільки створеного.

Підтвердження бронювання з номером `orderId` за заявкою реалізується функцією `ConfirmOrderView`, яка виконується тільки якщо авторизований користувач є агентом, відповідальним за заміський будинок, який заброньовано, а бронювання є тільки створеним.

Підтвердження бронювання з номером `orderId` за початком оренди реалізується функцією `StartOrderView`, яка виконується тільки:

- якщо авторизований користувач є агентом, відповідальним за заміський будинок, який заброньовано, а бронювання є тільки сплаченим або розпочатим з точки зору орендаря (тобто він вже підтвердив цей факт стосовно замовлення);
- якщо авторизований користувач є орендарем, який створив це бронювання, а бронювання є тільки сплаченим або розпочатим з точки зору агента (тобто він вже підтвердив цей факт стосовно замовлення).

Підтвердження бронювання з номером `orderId` за завершенням оренди реалізується функцією `FinishOrderView`, яка виконується тільки:

- якщо авторизований користувач є агентом, відповідальним за заміський будинок, який заброньовано, а бронювання є підтвердженим обома сторонами як розпочате або підтвердженим орендарем як завершене;
- якщо авторизований користувач є орендарем, який створив це бронювання, а бронювання є підтвердженим обома сторонами як розпочате або підтвердженим агентом як завершене.

Запитування огляду будинку реалізовано функцією `AddViewingView`, яка за заміським будинком з номером `houseId` створює за підтримки форми `FormViewing` запис про запит на огляд будинку.

Підтвердження огляду заміського будинку реалізовано функцією `ConfirmOrderView`, яка має параметр `viewingId`, що визначає номер створеного запиту на огляд. Якщо користувач є агентом відповідальним за заміський

будинку, запит на огляд якого розглядається, то стан огляду змінюється і дані зберігаються в БД.

Скасування огляду заміського будинку реалізовано функцією `CancelOrderView`, яка має параметр `viewingId`, що визначає номер створеного запиту на огляд. Якщо користувач є агентом відповідальним за заміський будинок, запит на огляд якого розглядається, а огляд або тільки створений, або підтверджений, то його стан змінюється і дані зберігаються в БД.

Отримання переліку власноруч створених запитів на огляд з підтримкою визначення всіх бронювань реалізовано функцією `AllRenterViewingsView`, яка для авторизованого користувача як орендаря відбирає всі огляди, створені ним. Підтримка переліку тільки скасованих запитів на огляд реалізована функцією `CancelledRenterViewingsView`. Підтримка переліку тільки підтверджених запитів на огляд реалізована функцією `ConfirmedRenterViewingsView`.

Отримання переліку всіх запитів на огляд заміських будинків, якими керує авторизований агент, реалізовано функцією `AllLandlordViewingsView`. Отримання переліку всіх скасованих запитів на огляд заміських будинків, якими керує авторизований агент, реалізовано функцією `CancelledLandlordViewingsView`. Підтримка переліку тільки підтверджених запитів на огляд заміських будинків, якими керує авторизований агент, реалізована функцією `ConfirmedLandlordViewingsView`.

Створення облікових запитів агентів реалізовано функцією `CreateRentuserView`, яка працює на основі форми `FormRentuser`. Створити обліковий запис агента може тільки менеджер.

Створення спілкування з агентом реалізовано функцією `CreateTalkView`, яка має параметр `landlordId`, що визначає номер агента. Спілкування створюється між авторизованим користувачем і агентом, якщо воно ще не було створено раніше. Якщо було, то відбувається перенаправлення до наступного представлення, де роботи вже з самими повідомленнями підтримується.

Надсилання повідомлення в спілкуванні реалізовано функцією `MessageView`, яка має параметр `talkId`, що визначає номер спілкування, робота з повідомленнями з якого виконується. Шаблон `message.html` заповнюється даними з ключем `form`, що визначає форму повідомлення, та з ключем `messages`, що містить перелік повідомлень у цьому спілкуванні, які вже були створені.

Отримання послідовності оренд і оглядів за заміським будинком з номером `houseId` реалізовано функцією `JobView`. Вона впорядковує одразу і оренди, і огляди за часом, починаючи з поточного моменту. Тобто це план роботи.

3.3 Висновки за розділом 3

Кожну заплановану функцію згідно з вимогами до програми реалізовано. Створено зокрема функції для підтримки сплати за бронювання та функції керування бронюваннями оренди та огляду заміських будинків.

4 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

4.1 Призначення програми

Програмна система призначена для забезпечення потреби в орендуванні замських будинків користувачами, які знаходяться в такому пошуку, з врахуванням особливостей як такого типу оренди загалом, так і замських будинків зокрема. Для забезпечення повноцінного такого процесу розробка забезпечує іншу сторону своїм другим призначенням – представляти інформацію про можливість оренди власних замських будинків, керувати бронюваннями щодо оренди замських будинків на різних етапах даного процесу.

4.2 Умови виконання програми

Технічні засоби для підтримки віддаленої роботи з наявністю обо'язкових програмних засобів, включаючи базовий набір засобів python і MySQL, мають включати мінімально:

- процесор, частота – 2 x 1,6 ГГц;
- оперативна пам'ять – 3,5 Гб;
- жорсткий диск – 40 Гб вільного простору (мінімально) з розширенням подальшим під ту кількість користувачів, які працюють як агенти і ту кількість фотографій, яку вони завантажують в програму.

4.3 Принципи експлуатації програми

Експлуатація програми має визначатися двома процесами. Перший пов'язаний з забезпеченням вимог, представлених у попередньому підрозділі, і запуском сервера. Другий реалізується вже самими користувачами в різних ролях за допомогою власних пристроїв з доступом до інтернет.

Експлуатація програми користувачами реалізується за представленим в розділі 3 порядком. Схему функціонування забезпечено за допомогою меню на сторінках вебдодатку. Меню різниться в залежності від етапу роботи і ролі, якій відповідає користувацький обліковий запис.

Звичайний неавторизований користувач з базової сторінки отримує доступ до рейтингу замських будинків. З меню такий неавторизований користувач може обрати цю роботу через пункт «Топ», а за допомогою зображення лупи перейти до пошуку замських будинків. Кнопка «Авторизація» дозволяє перейти до авторизації.

Після виконання авторизації кількість пунктів меню стає більшою. Пункт меню «Топ» також містить рейтинг найкращих замських будинків (рис. 4.1).

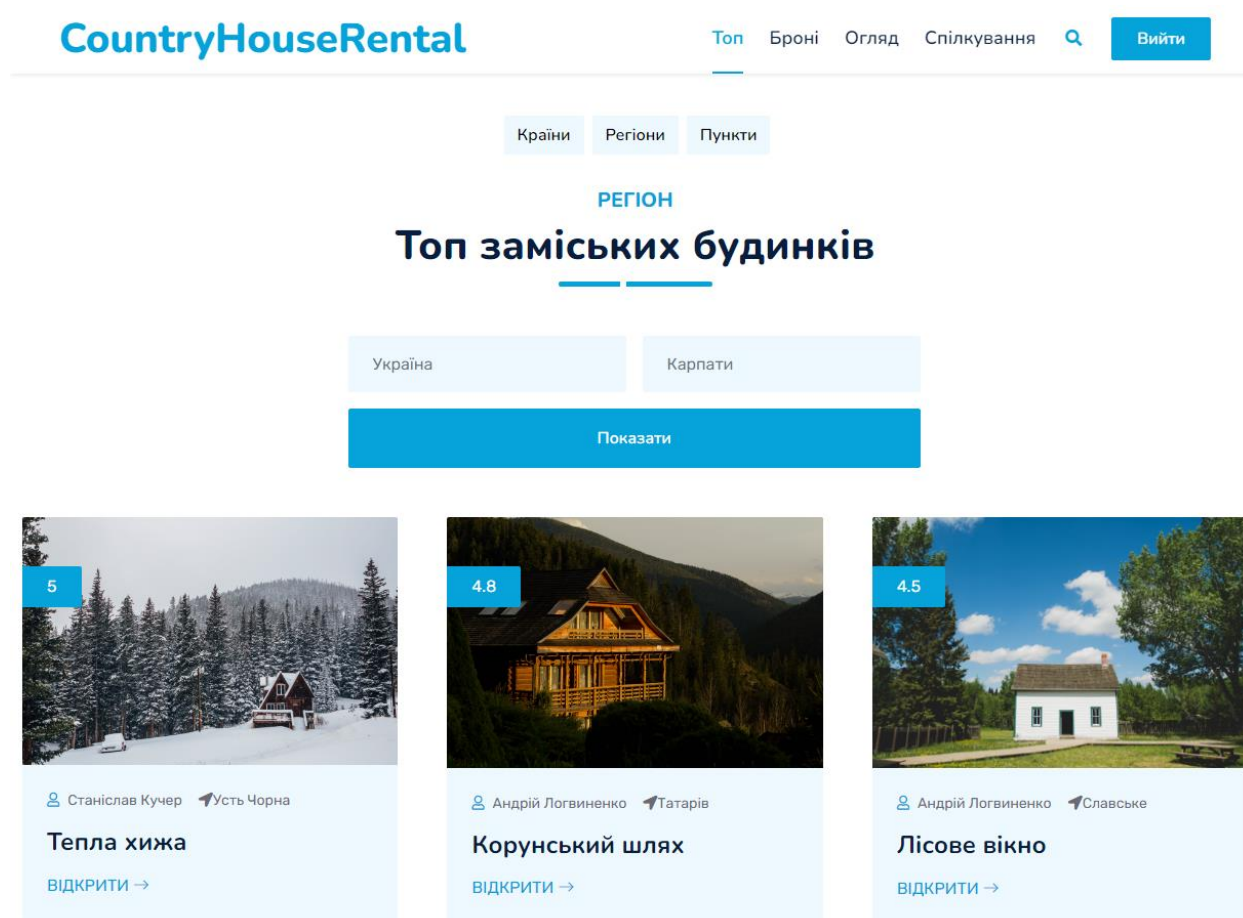


Рисунок 4.1 – Перелік найкращих замських будинків

Сторінка з переліком найкращих заміських будинків має 3 підпункти для вибору: країни, регіони, пункти. Країни дозволяють сформувати рейтинг найкращих у заданій країні заміських будинків. Регіони дозволяють сформувати рейтинг найкращих у заданому регіоні заміських будинків. Для цього потрібно ввести назву країни (випадаючий перелік формується автоматично за введеними даними) та назву регіону, потім натиснути на кнопку «Показати». Нижче відтворюється рейтинг за середньою оцінкою. На кожну позицію можна натиснути і відкрити детальні дані про такий заміський будинок. У загальному вигляді по ньому виводиться оцінка, агент, який за нього відповідає, назва будинку, населений пункт, де він розташований.

Після натискання на один з заміських будинків виводиться його сторінка (рис. 4.2).

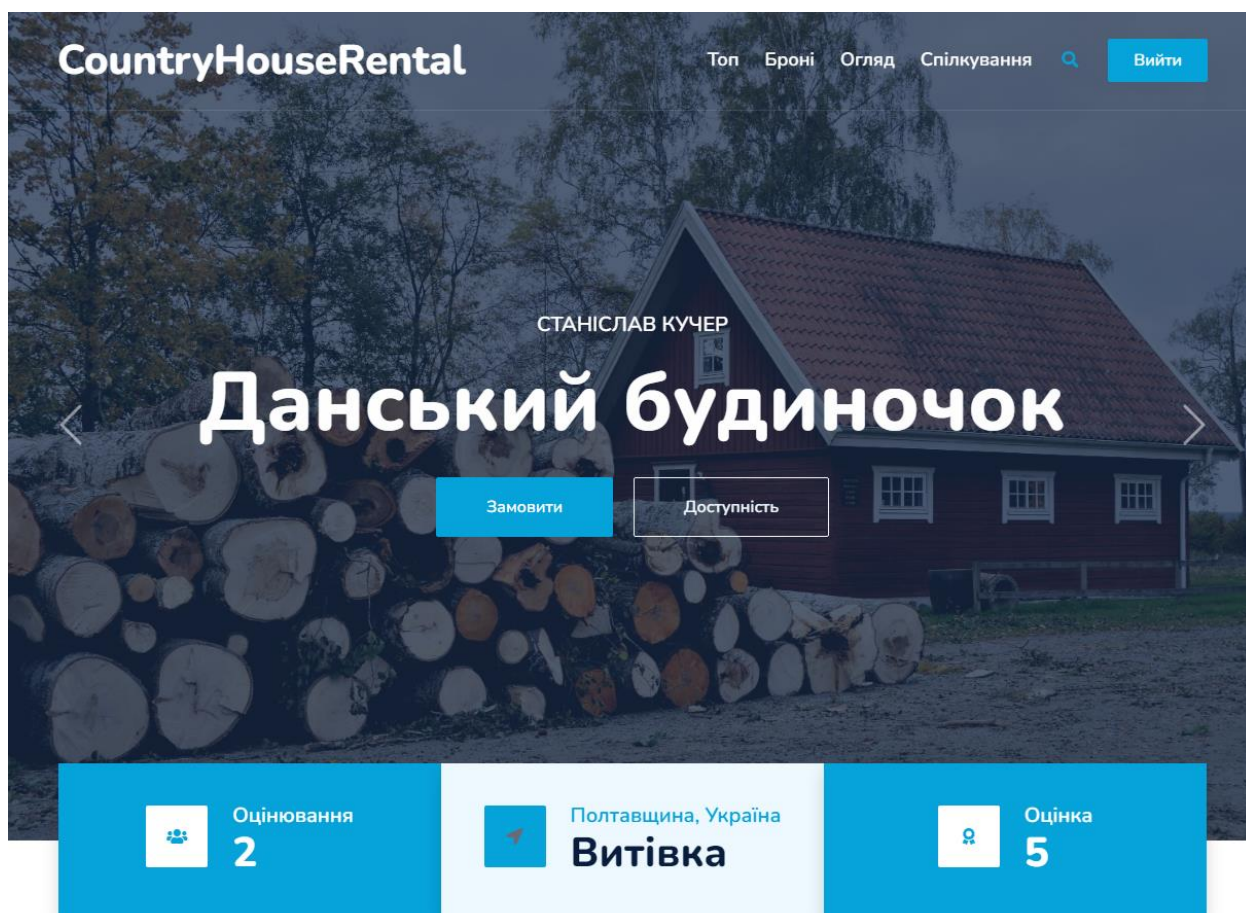


Рисунок 4.2 – Деталізація заміського будинку

На цій сторінці зверху знаходяться фотографії, які відповідають всьому будинку загалом. Між ними можна перемикається стрілками. Доступні кнопки, щоб виконати замовлення будинку або перевірити доступність будинку на потрібні дати. Також виводиться кількість виконаних оцінювань за цим будинком, місце розташування його та середня оцінка.

Сторінка ця є довгою і містить всю відповідну інформацію за замиським будинком. Так наступним блоком розташовано загальні дані про цей будинок (рис. 4.3), включаючи опис, значення різних характеристик, значення властивостей, які асоційовані з цим будинком (спальні місця, душ, холодильник, електропічка тощо). Також виводяться фотографії у вертикальному вигляді. При натисканні на фотографію вона відображається у повному вигляді поверх сторінки, надаються стрілки для переміщення між фотографіями.

CountryHouseRental

Топ

Броні

Огляд

Спілкування

Вийти

ЗАГАЛОМ

Горганський дім

В одному з доволі віддалених куточків Горган розташований будиночок, де все достатньо затишно і прям посеред дикого світу. Дійти треба пішки, але є можливість підвезти машиною, так що йти хвилин 25. Ми будемо раді поділитися власною схованкою з добрими людьми.

✓

Тварини дозволені

✓

Вечірки дозволені

✓

Можна з дітьми

✓

Паління заборонено

✗

Провідний інтернет

✓

Мінімальний вік: 20

✗

Безпроводний інтернет

✓

Заїзд з 14:00

✓

Місць: 3

✓

Виїзд до 13:00

✓

Харчування: 2

Мінімальна тривалість, діб

3

✓

Спальні місця: 2

✓

Мікрохвильовка: 1

✓

Душ: 1

✓

Холодильник: 1

Рисунок 4.3 – Загальні деталі замиського будинку

Наступним блоком сторінка містить послідовно (як окремими блоками) розташовані дані всіх приміщень, які було окремо виділені за цим замиським будинком (рис. 4.4). За кожним приміщенням надається його опис, інформація про кількість людей, які можуть знаходитись у приміщенні, про тип приміщенні, а також асоційовані властивості (диван, шафа, котел тощо).

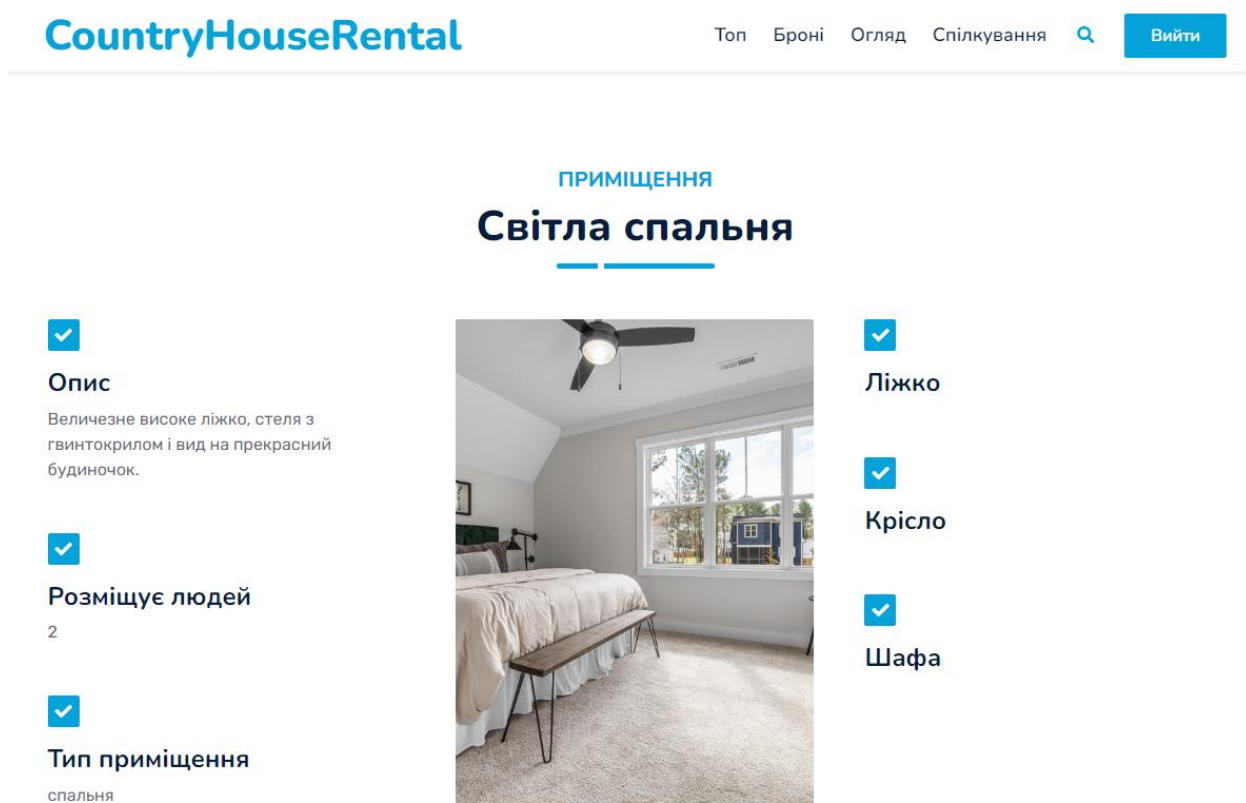


Рисунок 4.4 – Приміщення замиського будинку

Наступний блок включає інформацію про агента з можливістю відкрити сторінку з даними про його досвід та можливістю написати йому в спілкування, після чого відкривається сторінка з повідомленнями. Під ним знаходиться блок, який відображає мапу, на якій показано адресу, за якою розташовано цей будинок. Ще нижче блоком знаходяться пакети страхування, які можна використати для страхування при орендуванні за цим будинком, наступним блоком – оцінювання, які були залишені після орендування цього замиського будинку.

Для менеджера пункти меню включають «Проблеми» для відображення бронювань, які не були підтверджені обома сторонами протягом 5 днів після завершення строків оренди, «Страхування» для керування пакетами страхування, в тому числі додавання нових, оновлення, «Агенти» для перегляду внесених даних та створення нових облікових записів у вебдодатку.

Агент після авторизації окрім рейтингу та пошуку має пункти меню «Будинки» та «Спілкування». Спілкування дозволяє відповідати на повідомлення користувачів. Пункт «Будинки» є центральним щодо засобів керування (рис. 4.5).

CountryHouseRental
Топ Будинки Спілкування
Вийти

МАКСИМ ЩЕРБИНА

22.05.2023-05.06.2023

Коментар відсутній

✓ Підтверджено заїзд

✓ Вартість: ₴21000

✓ Страховка: APN, повний

Підтвердити завершення

ЄВГЕН РЕШКО

15.05.2023-22.05.2023

Я зможу заїхати тільки до 18:00.

✓ Завершене

✓ Вартість: ₴10500

✓ Страховка: APN, повний

Відгук




Рисунок 4.5 – Всі бронювання оренди замиського будинку

Цей пункт дозволяє спочатку вивести всі замиські будинки, якими керує цей агент. Доступне посилання для переходу до внесення нового будинку. За

кожним будинком відображається посилання для оновлення даних, посилання на періоди часу, в які доступна оренда за будинком, посилання на цінові пропозиції, на бронювання, огляди та на план оренд і оглядів (послідовні дати з визначенням моменту, коли починається і закінчується оренда та датами оглядів). Бронювання агенту виводяться за конкретним будинком. Після вибору бронювань за будинком відкривається сторінка з бронюваннями з можливістю переходити між різними станами з самої гори сторінки (всіх, скасованих, підтверджених, поточних, завершених).

За завершеними бронюваннями агент може залишити відгук. За заявками агент може підтвердити або скасувати бронювання. Після цього користувачу стає з його сторінки з бронюваннями доступна можливість оплатити за бронювання. Якщо оплата не виконана протягом певного періоду, агент може її скасувати зі сторінки з бронюваннями. Далі агент і користувач підтверджують факти початку оренди та її закінчення. Якщо останній факт не підтверджено вчасно (5 діб), то таке бронювання відображається менеджеру.

4.4 Висновки за розділом 4

Представлені описи принципів експлуатації програми неавторизованими користувачами, користувачами, менеджерами та агентами. Приведено конкретні приклади виконання функцій.

ВИСНОВКИ

Оренда замських будинків має певні особливості, включаючи можливість наявності декількох приміщень, достатньо різні властивості таких приміщень, проведення огляду будинків, які було визначено і забезпечено при виконанні роботи. У підсумку розроблено ПЗ оренди замських будинків, використовуючи мову python для підтримки основних можливостей веброзробки. Проводячи аналіз аналогів, встановлено, що серед аналогів існують програми, які дозволяють виконувати тільки перегляд оголошень про оренду замських будинків, як належать конкретному агентству, програми, які агрегують оголошення, та програми, де можна виставляти різні пропозиції на самому вебсайті. До останньої категорії належить розробка цієї роботи.

У роботі створено детальну структуру БД для врахування особливостей оренди замських будинків, визначено і розподілено за ролями функціональні вимоги до програми стосовно неавторизованих користувачів, користувачів, агентів, які володіють або керують замськими будинками, менеджерів з побудовою діаграм прецедентів, реалізовано програмну систему як вебсистему з описом отриманих результатів.

Програмну систему, яку розроблено в підсумку, можна застосовувати для бронювання оренди замських будинків і підтримки процесів, пов'язаних з бронюваннями. Програма дозволяє безпосередньо самому власнику замського будинку або агенту визначити параметри оренди будинку, включаючи дати, коли він доступний для оренди, різну вартість у різні періоди, керувати бронюваннями звичайних користувачів, які фактично є орендарями, керувати переглядами будинків. Користувачі можуть бронювати оренду житла, перегляд будинків, коли вони бажають завчасно ознайомитися з будинком та шукати замські будинки, які відповідають їх вимогам.

Усі завдання на роботу, вимоги до програми, які були визначені та заплановані для реалізації, виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Powell, F. Millennials Are Renting in Cities But Buying Country Houses [Electronic resource] / F. Powell. – Access mode : <https://www.thestreet.com/mortgages/millennials-are-renting-in-cities-but-buying-country-houses-13099629>.
2. Letting country homes: will rents continue to rise? [Electronic resource] / T. Kerr, T. Kruse, V. Foreman, C. Culot. – Access mode : <https://www.brown-co.com/view-insight/letting-country-homes-will-rents-continue-to-rise>.
3. Estate Agents Country House Experts Hampshire and West Sussex [Electronic resource]. – Access mode : <https://www.countryhousecompany.co.uk/>.
4. Cottage Gems. Luxury Holiday Cottages to Rent in the UK and Ireland [Electronic resource]. – Access mode : <https://www.cottagegems.com/>.
5. HolidayHomes. Search Holiday Lettings, Hotels & Flights [Electronic resource]. – Access mode : <https://www.holidayhomes.com/>.
6. Hotelmix.com.ua – Бронювання готелів онлайн – Забронювати Готель Дешево [Електрон. ресурс]. – Режим доступу : <https://hotelmix.com.ua/>.
7. Ernesti, J. Python 3 [Text] : The Comprehensive Guide to Hands-On Python Programming / Johannes Ernesti, Peter Kaiser. – Bonn : Rheinwerk Computing, 2022. – 1078 p.
8. Маттес, Е. Пришвидшений курс Python [Текст] / Е. Маттес. – Львів : Видавництво Старого Лева, 2021. – 600 с.
9. Hernandez, M. J. Database Design for Mere Mortals [Electronic resource] : 25th Anniversary Edition / M. J. Hernandez. – Boston : Addison-Wesley Professional, 2020. – 640 p.
10. Startup Free Website Template. Free CSS Templates. Free CSS [Electronic resource]. – Access mode : <https://www.free-css.com/free-css-templates/page288/startup>.

ДОДАТОК А
Технічне завдання

Вступ

Оренда замських будинків визначає необхідність взаємодії між людьми, які володіють такими будинками або представляють таких людей, тобто є агентами, та орендарями. Така необхідність має забезпечуватися використанням інформаційних технологій, при чому таких, щоб можна було забезпечити віддалену взаємодію. Тому при виконанні роботи таку необхідність потрібно забезпечити.

A.1 Підстави для розробки

Завдання на дипломну кваліфікаційну роботу сформульовано у відповідності з наказом № 87 від 4 квітня 2023 р. за Національним університетом «Запорізька політехніка», яким було визначено тему «Програмне забезпечення оренди замських будинків».

A.2 Призначення розробки

Розробка призначена для забезпечення потреби в орендуванні замських будинків користувачами, які знаходяться в такому пошуку, з врахуванням особливостей як такого типу оренди загалом, так і замських будинків зокрема.

Для забезпечення повноцінного такого процесу розробка має забезпечувати іншу сторону своїм другим призначенням – представляти інформацію про можливість оренди власних замських будинків, керувати бронюваннями щодо оренди замських будинків на різних етапах даного процесу.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Деталізація функціональних вимог до розробки у відповідності з її призначенням включає:

- пошук замських будинків для оренди;
- визначення оренди власного замського будинку;
- бронювання оренди замського будинку;
- встановлення строків можливої оренди замського будинку;
- оновлення строків можливої оренди замського будинку;
- скасування вказаного строку можливої оренди замського будинку;
- отримання переліку замських будинків у керуванні;
- оновлення інформації про замський будинок;
- визначення окремих приміщень замського будинку;
- редагування і вилучення інформації про окремі приміщення замського будинку;
- отримання переліку бронювань оренди замського будинку з підтримкою визначення всіх бронювань, тільки скасованих, тільки підтверджених, поточних, завершених;
- підтвердження бронювання за заявкою, за початком оренди, за завершенням оренди;
- скасування бронювання;
- запитування огляду будинку;
- підтвердження чи скасування огляду замського будинку;
- створення облікових запитів агентів (орендодавців);
- авторизація;
- отримання послідовності оренд і оглядів;
- створення спілкування з агентом (орендодавцем);
- надсилання повідомлення в спілкуванні;
- реєстрація користувачів (орендарів);

- отримання переліку запитів на огляд заміського будинку з підтримкою визначення всіх запитів, тільки скасованих, тільки підтверджених;
- оцінювання досвіду оренди заміського будинку;
- визначення пакету страхування;
- оновлення пакету страхування;
- отримання переліку наявних пакетів страхування;
- відображення всіх даних заміського будинку;
- відображення досвіду агента;
- відображення досвіду користувача;
- отримання переліку власноруч створених запитів на огляд з підтримкою визначення всіх бронювань, тільки скасованих, тільки підтверджених;
- отримання переліку найкращих заміських будинків за регіоном, за населеним пунктом, за країною.

A.3.2 Вимоги до складу та параметрів технічних засобів

Технічні засоби для підтримки віддаленої роботи мають включати мінімально:

- процесор, частота – 2 x 1,6 ГГц;
- оперативна пам'ять – 3,5 Гб;
- жорсткий диск – 40 Гб вільного простору (мінімально) з розширенням подальшим під ту кількість користувачів, які працюють як агенти і ту кількість фотографій, яку вони завантажують в програму.

A.4 Порядок контролю та приймання

Завдання на роботу визначає основні складові виконуваної роботи та порядок виконання різних етапів, у тому числі з зазначенням у календарному

плані етапу роботи, строків виконання кожного етапу та складової пояснювальної записки, яка повинна бути створеною за результатами.

Підсумковий результат перевіряється шляхом проведення захисту перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

```

from django.shortcuts import render
from django.shortcuts import redirect
from .models import Village
from .models import Region
from .models import Country
from .models import Rentuser
from .models import House
from django.shortcuts import get_object_or_404
from .models import Rentable
from .models import Priceperiod
from .models import HouseOpinion
from .forms import FormInsurance
from .models import Insurance
from .models import Order
from .models import Talk
from .models import Message
from .models import Viewing
from .forms import FormHouseOpinion
from .forms import FormOrder
from .forms import FormRentable
from .forms import FormHouse
from .forms import FormHouseobject
from .forms import FormViewing
from .models import Houseobject
from .forms import FormMessage
from .forms import FormRentuser
from django.conf import settings
from paypal.standard.ipn.signals import valid_ipn_received
from paypal.standard.models import ST_PP_COMPLETED

def VillageHouseRatingView(request, villageId):
    village = Village.objects.filter(id =
villageId).select_related('region').select_related('region__country').get()
    if village:
        village_houses = House.objects.filter(village =
village).prefetch_related("photos").order_by("-opinion")
        return render(request, "countryhouse/villagerating.html", {"houses":
village_houses, "village": village})

def RegionHouseRatingView(request, regionId):

```

```

    region = Region.objects.filter(id = regionId).select_related('country').get()
    if region:
        region_houses = House.objects.filter(village__region =
region).prefetch_related("photos").order_by("-opinion")
        return render(request,
                        "countryhouse/regionrating.html",
                        {"houses": region_houses,
                        "region": region
                        })

def CountryHouseRatingView(request, countryId):
    country = Country.objects.filter(id = countryId).get()
    if country:
        country_houses = House.objects.filter(village__region__country =
country).prefetch_related("photos").order_by("-opinion")
        return render(request,
                        "countryhouse/countryrating.html",
                        {"houses": country_houses,
                        "country": country
                        })

def LandlordHousesView(request, landlordId):
    landlord = Rentuser.objects.filter(id = landlordId).get()
    if landlord:
        landlord_houses = House.objects.filter(landlord =
landlord).prefetch_related("photos").select_related("village").select_related("village__region").select_related("village__region__country").order_by("-saved")
        return render(request,
                        "countryhouse/landlordhouses.html",
                        {"houses": landlord_houses
                        })
    else:
        return render(request,
                        "countryhouse/landlordhouses.html",
                        {"houses": []
                        })

def MyHousesView(request):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    return LandlordHousesView(request, landlord.id)

```

```

import datetime

def CheckHouseRentView(request, houseId, rent1, rent2, insuranceId = None):
    house = House.objects.filter(id = houseId).get()
    rentables = []
    insurances = []
    insurance = None
    if insuranceId:
        insurance = Insurance.objects.filter(id = insuranceId).get()
    if house:
        rentables = Rentable.objects.filter(house = house, date1__gte =
datetime.datetime.now())
        insurances = Insurance.objects.filter(insurancetype = house.insurancetype)
        rental_dates = False
        rentable = Rentable.objects.filter(house = house, date1__lte = rent1, date2__gte
= rent1)
        if rentable and rentable.date2 < rent2:
            for rent in rentable:
                lastpart = Rentable.objects.filter(house = house, date1 = rent.date2,
date2__gte = rent2)
                if lastpart:
                    rental_dates = True
            elif rentable:
                rental_dates = True
        resdays = rent2 - rent1
        if house and rental_dates and resdays.days <= house.period:
            priceperiod = Priceperiod.objects.filter(house = house, rentstart__lte = rent1,
rentend__gte = rent1)
            if priceperiod and priceperiod.date2 < rent2:
                days = priceperiod.date2 - rent1
                price = priceperiod.price * days.days
                rent1 = rentable.date2
                while rent1 < rent2:
                    lastpart = Rentable.objects.filter(house = house, date1 = rent1)
                    if lastpart.date2 < rent2:
                        days = lastpart.date2 - rent1
                        price = price + lastpart.price * days.days
                        rent1 = lastpart.date2
                    else:
                        days = rent2 - rent1

```

```

        price = price + lastpart.price * days.days
        rent1 = rent2
    elif priceperiod:
        days = rent2 - rent1
        price = priceperiod.price * days.days
    if insurance.insurancetype == house.insurancetype:
        if resdays.days % insurance.quantity == 0:
            days = resdays.days / insurance.quantity
        else:
            days = resdays.days / insurance.quantity + 1
        price = price + insurance.price * days
    return render(request,
                  "countryhouse/checkrent.html",
                  {"price": price,
                   "rentable": True,
                   "rentables": rentables,
                   "insurances": insurances
                  })
    return render(request,
                  "countryhouse/checkrent.html",
                  {"price": 0,
                   "rentable": False,
                   "rentables": rentables,
                   "insurances": insurances
                  })

def InsuranceView(request):
    forminsurance = None
    if request.user.is_staff:
        if request.method != "POST":
            forminsurance = FormInsurance()
        else:
            forminsurance = FormInsurance(request.POST)
            if forminsurance.is_valid():
                forminsurance.save()
                return redirect("allinsurancesview")
    return render(request,
                  "countryhouse/insurance.html",
                  {"form": forminsurance
                  })

```

```

def UpdateInsuranceView(request, insuranceId):
    forminsurance = None
    if request.user.is_staff:
        insurance = get_object_or_404(Insurance, pk = insuranceId)
        if request.method != "POST":
            forminsurance = FormInsurance(None, instance = insurance)

    else:
        forminsurance = FormInsurance(request.POST, instance = insurance)
        if forminsurance.is_valid():
            forminsurance.save()
            return redirect("allinsurancesview")
    return render(request,
                  "countryhouse/insurance.html",
                  {"form": forminsurance
                  })

def PaymentView(request, orderId):
    order = Order.objects.get(pk = orderId)
    paypal_payment = settings.paypaldata
    paypal_payment.invoice = order.pk
    paypal_payment.amount = order.price
    paypal_payment.item_name = "Country house rental"
    return render(request,
                  "countryhouse/payment.html",
                  {"paypal": paypal_payment
                  })

@receiver(valid_ipn_received)
def payment_getter(senderdata, **kwargs):
    if senderdata.payment_status == ST_PP_COMPLETED:
        try:
            order_paid = Order.objects.select_related("house").filter(pk =
senderdata.invoice).get()
            assert senderdata.mc_gross == order_paid.price and
senderdata.mc_currency == 'UAH'
        except Exception:
            pass
    else:
        order_paid.confirmed = 3

```

```

order_paid.save()
rentables = Rentable.objects.filter(date1__lte = order_paid.date1, date2__gte
= order_paid.date1)
for rentable in rentables:
    if rentable.date2 > order_paid.date2:
        end = rentable.date2
        rentable.date2 = order_paid.date1 - 1
        rentable.save()
    if order_paid.house.pause:
        rent = Rentable(house = order.house, date1 = order_paid.date2 + 1,
date2 = end)
        rent.save()
    else:
        rent = Rentable(house = order.house, date1 = order_paid.date2, date2
= end)
        rent.save()
    elif order_paid.house.pause:
        rentable.date2 = order_paid.date1 - 1
        rentable.save()
    else:
        rentable.date2 = order_paid.date1
        rentable.save()
rentables = Rentable.objects.filter(date2__lte = order_paid.date1, date2__gte
= order_paid.date2)
for rentable in rentables:
    if rentable.date1 < order_paid.date1:
        start = rentable.date2
        rentable.date2 = order_paid.date1 - 1
        rentable.save()
    if order_paid.house.pause:
        rent = Rentable(house = order.house, date1 = order_paid.date2 + 1,
date2 = end)
        rent.save()
    else:
        rent = Rentable(house = order.house, date1 = order_paid.date2, date2
= end)
        rent.save()
    elif order_paid.house.pause:
        rentable.date1 = order_paid.date2 - 1
        rentable.save()

```

```

        else:
            rentable.date1 = order_paid.date2
            rentable.save()

    def InsurancesView(request):
        insurances = Insurance.objects.all().select_related('insurer')
        .order_by("insurer__company")
        return render(request,
            "countryhouse/insurances.html",
            {"insurances": insurances
            })

    def LateOrdersView(request):
        if user.request.is_staff:
            end = datetime.datetime.now() - datetime.timedelta(days = 5)
            orders =
Order.objects.select_related('house').select_related('rentuser').filter(confirmed__in = [6, 7,
8], date2__lt = end).order_by("-date2")
            return render(request,
                "countryhouse/problemorders.html",
                {"orders": orders
                })

    def HouseView(request, houseId):
        house = House.objects.filter(pk =
houseId).select_related('landlord').select_related('village').select_related('village__region').
select_related('village__region__country').prefetch_related('photos').prefetch_related('objec
ts').prefetch_related('objects__photos').prefetch_related('features').prefetch_related('object
s__features').prefetch_related('objects__objecttype').get()
        opinions = HouseOpinion.objects.filter(order__house__id =
houseId).select_related('order__rentuser').order_by('-saved')
        insurances = Insurance.objects.filter(insurancetype =
house.insurancetype).select_related('insurer').order_by('title')
        return render(request,
            "countryhouse/house.html",
            {"house": house,
            "opinions": opinions,
            "insurances": insurances
            })

```

```

def HouseSearchView(request, rentalBegin, rentalEnd, topPrice = None,
bottomPrice = None, villageId = None, regionId = None, countryId = None, people = 1,
diningplaces = 1, opinionLevel = 1, checkin = None, checkout = None, age = None, pets =
None, children = None, smoking = None, parties = None, wifi = None, place = None,
houseFeatures = {}):
    houses_results =
House.objects.filter().select_related('landlord').select_related('village').select_related('villa
ge__region').select_related('village__region__country')
    if villageId:
        village = get_object_or_404(Village, pk = villageId)
        houses_results = houses_results.filter(village = village)
    if regionId:
        region = get_object_or_404(Region, pk = regionId)
        houses_results = houses_results.filter(village__region = region)
    if countryId:
        country = get_object_or_404(Country, pk = countryId)
        houses_results = houses_results.filter(village__region__country = country)
    if people > 1:
        houses_results = houses_results.filter(people__gte = people)
    if diningplaces > 1:
        houses_results = houses_results.filter(diningplaces__gte = diningplaces)
    if opinionLevel > 1:
        houses_results = houses_results.filter(opinion__gte = opinionLevel)
    if age:
        houses_results = houses_results.filter(age__lte = age)
    if pets and pets == 1:
        houses_results = houses_results.filter(petsallowed = True)
    elif pets and pets == 2:
        houses_results = houses_results.filter(petsallowed = False)
    if children and children == 1:
        houses_results = houses_results.filter(childrenallowed = True)
    elif children and children == 2:
        houses_results = houses_results.filter(childrenallowed = False)
    if smoking and smoking == 1:
        houses_results = houses_results.filter(smokingallowed = True)
    elif smoking and smoking == 2:
        houses_results = houses_results.filter(smokingallowed = False)
    if parties and parties == 1:
        houses_results = houses_results.filter(partiesallowed = True)
    elif parties and parties == 2:

```

```

        houses_results = houses_results.filter(partiesallowed = False)
    if wifi:
        houses_results = houses_results.filter(wifiaccess = True)
    if place:
        houses_results = houses_results.filter(placeaccess = True)
    if checkin:
        houses_results = houses_results.filter(checkin__gte = checkin)
    if checkout:
        houses_results = houses_results.filter(checkout__lte = checkout)
    houses_selected = []
    for house in houses_results:
        insurances = Insurance.objects.filter(insurancetype = house.insurancetype)
        rental_dates = False
        rentable = Rentable.objects.filter(house = house, date1__lte = rentalBegin,
date2__gte = rentalBegin)
        if rentable and rentable.date2 < rent2:
            for rent in rentable:
                lastpart = Rentable.objects.filter(house = house, date1 = rent.date2,
date2__gte = rentalEnd)
                if lastpart:
                    rental_dates = True
            elif rentable:
                rental_dates = True
        days = rentalEnd - rentalBegin
        if house and rental_dates and days.days <= house.period:
            priceperiod = Priceperiod.objects.filter(house = house, rentstart__lte =
rentalBegin, rentend__gte = rentalBegin)
            if priceperiod and priceperiod.date2 < rentalEnd:
                days = priceperiod.date2 - rentalBegin
                price = priceperiod.price * days.days
                rentalBegin = rentable.date2
                while rentalBegin < rentalEnd:
                    lastpart = Rentable.objects.filter(house = house, date1 = rentalBegin)
                    if lastpart.date2 < rentalEnd:
                        days = lastpart.date2 - rentalBegin
                        price = price + lastpart.price * days.days
                        rentalBegin = lastpart.date2
                    else:
                        days = rentalEnd - rentalBegin
                        price = price + lastpart.price * days.days

```

```

        rentalBegin = rentalEnd
    elif priceperiod:
        days = rentalEnd - rentalBegin
        price = priceperiod.price * days.days
        ok = True
    if topPrice and price > topPrice:
        ok = False
    if bottomPrice and price < bottomPrice:
        ok = False
    if ok:
        houses_selected.append(house.id)
    houses_results = House.objects.filter(pk__in = houses_selected).order_by("-
opinion")
    return render(request,
                  "countryhouse/houses.html",
                  {"houses": houses_results
                  })

def LandlordView(request, landlordId):
    landlord = get_object_or_404(Rentuser, pk = landlordId)
    if landlord.role == True:
        houses = House.objects.filter(landlord = landlord).order_by("-opinion")[:5]
        opinions = HouseOpinion.objects.filter(order__house__in = houses, rentler =
False).order_by("-saved")[:5]
        return render(request,
                      "countryhouse/landlord.html",
                      {"landlord": landlord,
                      "houses": houses,
                      "opinions": opinions
                      })
    else:
        return render(request,
                      "countryhouse/landlord.html",
                      {"landlord": landlord,
                      "houses": [],
                      "opinions": []
                      })

def RentlerView(request, rentlerId):
    rentler = get_object_or_404(Rentuser, pk = rentlerId)

```

```

        if landlord.role == False:
            opinions = HouseOpinion.objects.filter(order__rentuser = rentler, rentler =
True).order_by("-saved")[:5]
            return render(request,
                "countryhouse/rentler.html",
                {"rentler": rentler,
                "opinions": opinions
                })
        else:
            return render(request,
                "countryhouse/rentler.html",
                { "rentler": None,
                "opinions": []
                })

def HouseOpinionView(request, orderId):
    rentler = None
    formhouseopinion = None
    order = Order.objects.filter(id =
orderId).select_related('house').select_related('rentuser').select_related('house__landlord').
get()

    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    rentlerRole = True
    possible = False
    if rentuser.role:
        rentlerRole = False
        rentler = order.rentuser
        if order.house.landlord == rentuser:
            possible = True
    else:
        if order.rentuser == rentuser:
            possible = True
    if possible:
        if request.method != "POST":
            formhouseopinion = FormHouseOpinion()
        else:
            formhouseopinion = FormHouseOpinion(request.POST)
            if formhouseopinion.is_valid():
                houseopinion = formhouseopinion.save(commit = False)
                houseopinion.order = order

```

```

houseopinion.rentler = rentlerRole
houseopinion.save()
if not rentlerRole:
    houseopinions = HouseOpinion.objects.filter(rentler = order.rentuser)
    opinion, number = 0, 0
    for houseopinion in houseopinions:
        opinion = opinion + houseopinion.number
        number = number + 1
    rentler.opinion = opinion / number
    rentler.save()
    return redirect("rentlerview", rentler.id)
else:
    houseopinions = HouseOpinion.objects.filter(order__house =
order.house)
    opinion, number = 0, 0
    for houseopinion in houseopinions:
        opinion = opinion + houseopinion.number
        number = number + 1
    house = order.house
    house.opinion = opinion / number
    house.save()
    landlord = order.house.landlord
    houseopinions = HouseOpinion.objects.filter(order__house__landlord =
landlord)
    opinion, number = 0, 0
    for houseopinion in houseopinions:
        opinion = opinion + houseopinion.number
        number = number + 1
    landlord.opinion = opinion / number
    landlord.save()
    return redirect("houseview", order.house.id)
return render(request,
    "countryhouse/houseopinion.html",
    {"form": formhouseopinion,
    "role": rentlerRole,
    "rentler": rentler
    })

```

```

def OrderView(request, houseId, insuranceId = None):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)

```

```

formorder = None
insurance = None
if insuranceId:
    insurance = Insurance.objects.filter(id = insuranceId).get()
if request.method != "POST":
    formorder = FormOrder()
else:
    formorder = FormOrder(request.POST)
    if formorder.is_valid():
        house = get_object_or_404(House, pk = houseId)
        rent1 = form.cleaned_data['date1']
        rent2 = form.cleaned_data['date2']
        rentables = Rentable.objects.filter(house = house, date1__gte =
datetime.datetime.now())
        insurances = Insurance.objects.filter(insurancetype = house.insurancetype)
        rental_dates = False
        rentable = Rentable.objects.filter(house = house, date1__lte = rent1,
date2__gte = rent1)
        if rentable and rentable.date2 < rent2:
            for rent in rentable:
                lastpart = Rentable.objects.filter(house = house, date1 = rent.date2,
date2__gte = rent2)
                if lastpart:
                    rental_dates = True
            elif rentable:
                rental_dates = True
        resdays = rent2 - rent1
        if house and rental_dates and resdays.days <= house.period:
            priceperiod = Priceperiod.objects.filter(house = house, rentstart__lte =
rent1, rentend__gte = rent1)
            if priceperiod and priceperiod.date2 < rent2:
                days = priceperiod.date2 - rent1
                price = priceperiod.price * days.days
                rent1 = rentable.date2
                while rent1 < rent2:
                    lastpart = Rentable.objects.filter(house = house, date1 = rent1)
                    if lastpart.date2 < rent2:
                        days = lastpart.date2 - rent1
                        price = price + lastpart.price * days.days
                        rent1 = lastpart.date2

```

```

        else:
            days = rent2 - rent1
            price = price + lastpart.price * days.days
            rent1 = rent2
    elif priceperiod:
        days = rent2 - rent1
        price = priceperiod.price * days.days
    if insurance.insurancetype == house.insurancetype:
        if resdays.days % insurance.quantity == 0:
            days = resdays.days / insurance.quantity
        else:
            days = resdays.days / insurance.quantity + 1
        price = price + insurance.price * days
        order = formorder.save(commit = False)
        order.house = house
        order.rentuser = rentuser
        order.confirmed = 0
        order.price = price
        order.save()
        return redirect("ordersview")
    insurances = Insurance.objects.all()
    return render(request,
                  "countryhouse/order.html",
                  {"form": formorder,
                  "insurances": insurances
                  })

```

```

def RentableView(request, houseId):
    house = get_object_or_404(House, pk = houseId)
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    formrentable = None
    if landlord.role and house.landlord == landlord:
        if request.method != "POST":
            formrentable = FormRentable()
        else:
            formrentable = FormRentable(request.POST)
        if formrentable.is_valid():
            rentable = formrentable.save(commit = False)
            rentable.house = house
            rentable.save()

```

```

        return redirect("houseview", house.id)
    return render(request,
                  "countryhouse/rentable.html",
                  {"form": formrentable
                  })

```

```

def UpdateRentableView(request, rentableId):
    formrentable = None
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    if landlord.role:
        rentable = get_object_or_404(Rentable, pk = rentableId)
        if request.method != "POST":
            formrentable = FormRentable(None, instance = rentable)
        else:
            formrentable = FormRentable(request.POST, instance = rentable)
        if formrentable.is_valid():
            formrentable.save()
            return redirect("houseview", house.id)
    return render(request,
                  "countryhouse/rentable.html",
                  {"form": formrentable
                  })

```

```

def DeleteRentableView(request, rentableId):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    if landlord.role:
        rentable = get_object_or_404(Rentable, pk = rentableId)
        house = rentable.house.id
        if rentable.house.landlord == landlord:
            rentable.delete()
        return redirect("houseview", house)

```

```

def AddHouseView(request):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    formhouse = None
    if landlord.role:
        if request.method != "POST":
            formhouse = FormHouse()
        else:
            formhouse = FormHouse(request.POST)

```

```

    if formhouse.is_valid():
        house = formhouse.save(commit = False)
        house.landlord = landlord
        house.save()
        house.save_m2m()
        return redirect("houseview", house.id)
return render(request,
              "countryhouse/createhouse.html",
              {"form": formhouse
              })

```

```

def UpdateHouseView(request, houseId):
    formhouse = None
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    house = get_object_or_404(House, pk = houseId)
    if landlord.role and house.landlord == landlord:
        if request.method != "POST":
            formhouse = FormHouse(None, instance = house)
        else:
            formhouse = FormHouse(request.POST, instance = house)
        if formhouse.is_valid():
            formhouse.save()
            house.save_m2m()
            return redirect("houseview", house.id)
    return render(request,
                  "countryhouse/createhouse.html",
                  {"form": formhouse
                  })

```

```

def AddHouseobjectView(request, houseId):
    house = get_object_or_404(House, pk = houseId)
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    formhouseobject = None
    if landlord.role and house.landlord == landlord:
        if request.method != "POST":
            formhouseobject = FormHouseobject()
        else:
            formhouseobject = FormHouseobject(request.POST)
        if formhouseobject.is_valid():
            houseobject = formhouseobject.save(commit = False)

```

```

        houseobject.house = house
        houseobject.save()
        houseobject.save_m2m()
        return redirect("houseview", house.id)
    return render(request,
                  "countryhouse/houseobject.html",
                  {"form": formhouseobject
                  })

def UpdateHouseobjectView(request, houseobjectId):
    formhouseobject = None
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    houseobject = Houseobject.objects.filter(id =
houseobjectId).select_related('house').select_related('house__landlord').get()
    if landlord.role and houseobject.house.landlord == landlord:
        if request.method != "POST":
            formhouseobject = FormHouseobject(None, instance = houseobject)
        else:
            formhouseobject = FormHouseobject(request.POST, instance = houseobject)
            if formhouseobject.is_valid():
                formhouseobject.save()
                formhouseobject.save_m2m()
                return redirect("houseview", house.id)
    return render(request,
                  "countryhouse/houseobject.html",
                  {"form": formhouseobject
                  })

def DeleteHouseobjectView(request, houseobjectId):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    houseobject = Houseobject.objects.filter(id =
houseobjectId).select_related('house').select_related('house__landlord').get()
    house = houseobject.house
    if landlord.role and houseobject.house.landlord == landlord:
        rentable.delete()
    return redirect("myhousesview")

def AllOrdersView(request, houseId):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    house = get_object_or_404(House, pk = houseId)
    if landlord.role and house.landlord == landlord:

```

```

        allorders = Order.objects.filter(house__landlord =
landlord).select_related("rentuser").order_by("-saved")
        return render(request,
                        "countryhouse/houseorders.html",
                        {"orders": allorders
                        })
    else:
        return render(request,
                        "countryhouse/houseorders.html",
                        {"orders": []
                        })

def CancelledOrdersView(request, houseId):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    house = get_object_or_404(House, pk = houseId)
    if landlord.role and house.landlord == landlord:
        cancelledorders = Order.objects.filter(house__landlord = landlord, confirmed =
2).select_related("rentuser").order_by("-saved")
        return render(request,
                        "countryhouse/cancelledorders.html",
                        {"orders": cancelledorders
                        })
    else:
        return render(request,
                        "countryhouse/cancelledorders.html",
                        {"orders": []
                        })

def ConfirmedOrdersView(request, houseId):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    house = get_object_or_404(House, pk = houseId)
    if landlord.role and house.landlord == landlord:
        condirmedorders = Order.objects.filter(house__landlord = landlord, confirmed
= 1, date1__gte = datetime.datetime.now()).select_related("rentuser").order_by("date1")
        condirmedoldorders = Order.objects.filter(house__landlord = landlord,
confirmed = 1, date1__lt = datetime.datetime.now()).select_related("rentuser").order_by("-
date1")
        return render(request,
                        "countryhouse/condirmedorders.html",
                        {"orders": condirmedorders,

```

```

        "oldorders": confirmedoldorders,
    })
else:
    return render(request,
        "countryhouse/confirmedorders.html",
        {"orders": [],
         "oldorders": []
        })

def StartedOrdersView(request, houseId):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    house = get_object_or_404(House, pk = houseId)
    if landlord.role and house.landlord == landlord:
        startedorders = Order.objects.filter(house__landlord = landlord, confirmed__in
= [4,5,6]).select_related("rentuser").order_by("date1")
        return render(request,
            "countryhouse/startedorders.html",
            {"orders": startedorders
            })
    else:
        return render(request,
            "countryhouse/startedorders.html",
            {"orders": []
            })

def FinishedOrdersView(request, houseId):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    house = get_object_or_404(House, pk = houseId)
    if landlord.role and house.landlord == landlord:
        finishedorders = Order.objects.filter(house__landlord = landlord,
confirmed__in = [7,8,9]).select_related("rentuser").order_by("-date1")
        return render(request,
            "countryhouse/finishedorders.html",
            {"orders": finishedorders
            })
    else:
        return render(request,
            "countryhouse/finishedorders.html",
            {"orders": []
            })

```

```

def PaidOrdersView(request, houseId):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    house = get_object_or_404(House, pk = houseId)
    if landlord.role and house.landlord == landlord:
        paidorders = Order.objects.filter(house__landlord = landlord, confirmed__gte =
3).select_related("rentuser").order_by("-date1")
        return render(request,
                        "countryhouse/paidorders.html",
                        {"orders": paidorders
                        })
    else:
        return render(request,
                        "countryhouse/paidorders.html",
                        {"orders": []
                        })

def AllRenterOrdersView(request):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if not rentuser.role:
        allorders = Order.objects.filter(rentuser =
rentuser).select_related("house").select_related("house__village").select_related("house__vi
llage__region").select_related("house__village__region__country").order_by("-saved")
        return render(request,
                        "countryhouse/renterorders.html",
                        {"orders": allorders
                        })
    else:
        return render(request,
                        "countryhouse/renterorders.html",
                        {"orders": []
                        })

def CancelledRenterOrdersView(request):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if not rentuser.role:
        cancelledorders = Order.objects.filter(rentuser = rentuser, confirmed =
2).select_related("house").select_related("house__village").select_related("house__village__r
egion").select_related("house__village__region__country").order_by("-saved")
        return render(request,
                        "countryhouse/cancelledrenterorders.html",

```

```

        {"orders": cancelledorders
        })
    else:
        return render(request,
            "countryhouse/cancelledrenterorders.html",
            {"orders": []
            })

def ConfirmedDoneRenterOrdersView(request):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if not rentuser.role:
        condirmedorders = Order.objects.filter(rentuser = rentuser, confirmed =
9).select_related("house").select_related("house__village").select_related("house__village__r
egion").select_related("house__village__region__country").order_by("-date1")
        return render(request,
            "countryhouse/condirmedrenterorders.html",
            {"orders": condirmedorders
            })
    else:
        return render(request,
            "countryhouse/condirmedrenterorders.html",
            {"orders": []
            })

def ConfirmedNotDoneRenterOrdersView(request):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if not rentuser.role:
        condirmedorders = Order.objects.filter(rentuser = rentuser, confirmed__in =
[1,3,4,5,6,7,8]).select_related("house").select_related("house__village").select_related("hou
se__village__region").select_related("house__village__region__country").order_by("-date1")
        return render(request,
            "countryhouse/condirmednotrenterorders.html",
            {"orders": condirmedorders
            })
    else:
        return render(request,
            "countryhouse/condirmednotrenterorders.html",
            {"orders": []
            })

```

```

def CancelOrderView(request, orderId):
    order = get_object_or_404(Order, pk = orderId)
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if not rentuser.role and order.rentuser == rentuser and order.confirmed in [0, 1]:
        order.confirmed = 2
        order.save()
        return redirect("cancelledrenterordersview")
    elif rentuser.role and order.house.landlord == rentuser and order.confirmed in
[0, 1]:
        order.confirmed = 2
        order.save()
        return redirect("cancelledlandordersview")

def ConfirmOrderView(request, orderId):
    order = get_object_or_404(Order, pk = orderId)
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if rentuser.role and order.house.landlord == rentuser and order.confirmed == 0:
        order.confirmed = 1
        order.save()
        return redirect("confirmedlandordersview")

def StartOrderView(request, orderId):
    order = get_object_or_404(Order, pk = orderId)
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if not rentuser.role and order.rentuser == rentuser and order.confirmed == 3:
        order.confirmed = 5
        order.save()
        return redirect("confnotrenterordersview")
    elif not rentuser.role and order.rentuser == rentuser and order.confirmed == 4:
        order.confirmed = 6
        order.save()
        return redirect("confnotrenterordersview")
    elif rentuser.role and order.house.landlord == rentuser and order.confirmed ==
3:
        order.confirmed = 4
        order.save()
        return redirect("startedlandordersview")
    elif rentuser.role and order.house.landlord == rentuser and order.confirmed ==
5:
        order.confirmed = 6

```

```

order.save()
return redirect("startedlandordersview")

```

```

def FinishOrderView(request, orderId):
    order = get_object_or_404(Order, pk = orderId)
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if not rentuser.role and order.rentuser == rentuser and order.confirmed == 6:
        order.confirmed = 8
        order.save()
        return redirect("confrenterordersview")
    elif not rentuser.role and order.rentuser == rentuser and order.confirmed == 7:
        order.confirmed = 9
        order.save()
        return redirect("confrenterordersview")
    elif rentuser.role and order.house.landlord == rentuser and order.confirmed ==
6:
        order.confirmed = 7
        order.save()
        return redirect("finishedlandordersview")
    elif rentuser.role and order.house.landlord == rentuser and order.confirmed ==
8:
        order.confirmed = 9
        order.save()
        return redirect("finishedlandordersview")

```

```

def AddViewingView(request, houseId):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    house = get_object_or_404(House, pk = houseId)
    formviewing = None
    if request.method != "POST":
        formviewing = FormViewing()
    else:
        formviewing = FormViewing(request.POST)
        if formviewing.is_valid():
            viewing = formviewing.save(commit = False)
            viewing.house = house
            viewing.save()
            return redirect("viewingsview")
    return render(request,
                  "countryhouse/viewing.html",

```

```

    {"form": formviewing
    })

```

```

def ConfirmOrderView(request, viewingId):
    viewing = Viewing.objects.filter(id =
viewingId).select_related('house').select_related('house__landlord').get()
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if rentuser.role and viewing.house.landlord == rentuser and order.confirmed ==
0:
        viewing.confirmed = 2
        viewing.save()
        return redirect("confirmedviewingsview")

```

```

def CancelOrderView(request, viewingId):
    viewing = Viewing.objects.filter(id =
viewingId).select_related('house').select_related('house__landlord').get()
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if rentuser.role and viewing.house.landlord == rentuser and order.confirmed_in
= [0, 2]:
        viewing.confirmed = 1
        viewing.save()
        return redirect("cancelledviewingsview")

```

```

def AllRenterViewingsView(request):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if not rentuser.role:
        allviewings = Viewing.objects.filter(rentuser =
rentuser).select_related("house").select_related("house__village").select_related("house__vi
llage__region").select_related("house__village__region__country").order_by("-date")
        return render(request,
                        "countryhouse/allrenterviewings.html",
                        {"viewings": allviewings
                        })
    else:
        return render(request,
                        "countryhouse/allrenterviewings.html",
                        {"viewings": []
                        })

```

```

def RentusersView(request):

```

```

if request.user.is_staff:
    rentusers = Rentuser.objects.all().defer("password")
    return rentusers

```

```

def CancelledRenterViewingsView(request):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if not rentuser.role:
        allviewings = Viewing.objects.filter(rentuser = rentuser, confirmed =
1).select_related("house").select_related("house__village").select_related("house__village__r
egion").select_related("house__village__region__country").order_by("-date")
        return render(request,
                        "countryhouse/cancelledrenterviewings.html",
                        {"viewings": allviewings
                        })
    else:
        return render(request,
                        "countryhouse/cancelledrenterviewings.html",
                        {"viewings": []
                        })

```

```

def ConfirmedRenterViewingsView(request):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    if not rentuser.role:
        allviewings = Viewing.objects.filter(rentuser = rentuser, confirmed =
2).select_related("house").select_related("house__village").select_related("house__village__r
egion").select_related("house__village__region__country").order_by("-date")
        return render(request,
                        "countryhouse/confirmedrenterviewings.html",
                        {"viewings": allviewings
                        })
    else:
        return render(request,
                        "countryhouse/confirmedrenterviewings.html",
                        {"viewings": []
                        })

```

```

def AllLandlordViewingsView(request):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    if landlord.role:

```

```

        allviewings = Viewing.objects.filter(house__landlord =
landlord).select_related("house").select_related("house__village").select_related("house__vi
llage__region").select_related("house__village__region__country").order_by("-date")
        return render(request,
                        "countryhouse/allviewings.html",
                        {"viewings": allviewings
                        })
    else:
        return render(request,
                        "countryhouse/allviewings.html",
                        {"viewings": []
                        })

def CancelledLandlordViewingsView(request):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    if landlord.role:
        allviewings = Viewing.objects.filter(rentuser = rentuser, confirmed =
1).select_related("house").select_related("house__village").select_related("house__village__r
egion").select_related("house__village__region__country").order_by("-date")
        return render(request,
                        "countryhouse/cancelledviewings.html",
                        {"viewings": allviewings
                        })
    else:
        return render(request,
                        "countryhouse/cancelledviewings.html",
                        {"viewings": []
                        })

def CreateRentuserView(request):
    formrentuser = None
    if request.user.is_staff:
        if request.method != "POST":
            formrentuser = FormRentuser()
        else:
            formrentuser = FormRentuser(request.POST)
            if formrentuser.is_valid():
                formrentuser.save()
                return redirect("rentusersview")

```

```

def ConfirmedLandlordViewingsView(request):
    landlord = get_object_or_404(Rentuser, login = request.user.login)
    if landlord.role:
        allviewings = Viewing.objects.filter(rentuser = rentuser, confirmed =
2).select_related("house").select_related("house__village").select_related("house__village__r
egion").select_related("house__village__region__country").order_by("-date")
        return render(request,
                        "countryhouse/confirmedviewings.html",
                        {"viewings": allviewings
                        })
    else:
        return render(request,
                        "countryhouse/confirmedviewings.html",
                        {"viewings": []
                        })

def CreateTalkView(request, landlordId):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    landlord = get_object_or_404(Rentuser, id = landlordId)
    talks = Talk.objects.filter(landlord = landlord, rentler = rentuser)
    talk = talks[0]
    if talks:
        return redirect("talkview", talk.id)
    else:
        talk = Talk(landlord = landlord, rentler = rentuser)
        talk.save()
        return redirect("talkview", talk.id)

def MessageView(request, talkId):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    talk = get_object_or_404(Talk, pk = talkId)
    if rentuser.role:
        talks = Talk.objects.filter(landlord = rentuser)
    else:
        talks = Talk.objects.filter(rentler = rentuser)
    formmessage = None
    if request.method != "POST":
        formmessage = FormMessage()
    else:
        formmessage = FormMessage(request.POST)

```

```

if formmessage.is_valid():
    message = formmessage.save(commit = False)
    if message.role:
        message.creator = True
    else:
        message.creator = False
    message.save()
    messages = Message.objects.filter(talk = talk).order_by("-saved")
    return render(request,
                  "countryhouse/message.html",
                  {"form": formmessage,
                  "messages": messages,
                  "talks": talks
                  })
messages = Message.objects.filter(talk = talk).order_by("-saved")
return render(request,
              "countryhouse/message.html",
              {"form": formmessage,
              "messages": messages,
              "talks": talks
              })

def JobView(request, houseId):
    rentuser = get_object_or_404(Rentuser, login = request.user.login)
    house = get_object_or_404(House, pk = houseId)
    if rentuser.role:
        orders = Order.objects.filter(house = house, date1__gte =
datetime.datetime.now()).order_by("date1")
        viewings = Viewing.objects.filter(house = house, date__gte =
datetime.datetime.now()).order_by("date")
        end = False
        results = []
        i, j = 0, 0
        while i < len(orders) or j < len(viewings):
            if i < len(orders)
                current = orders[i]
                i = i + 1
            else:
                current = None
            if j < len(viewings)

```

```
        current2 = viewings[j]
        j = j + 1
    else:
        current2 = None
    if current and current2:
        if current.date1 < current2.date:
            results.append(current)
        else:
            results.append(current2)
    else:
        if current:
            results.append(current)
        else:
            results.append(current2)
    return render(request,
                  "countryhouse/job.html",
                  {"results": results
                  })
```

ДОДАТОК В
Слайди презентації

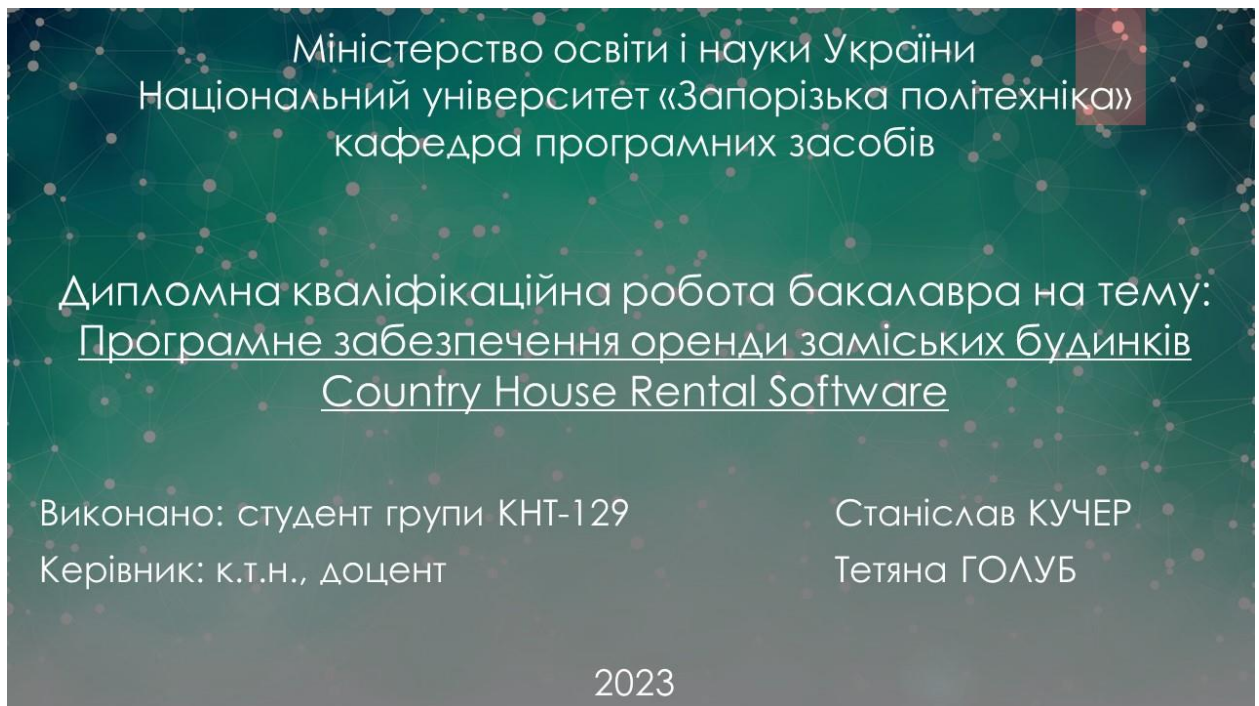


Рисунок В.1 – Початковий слайд

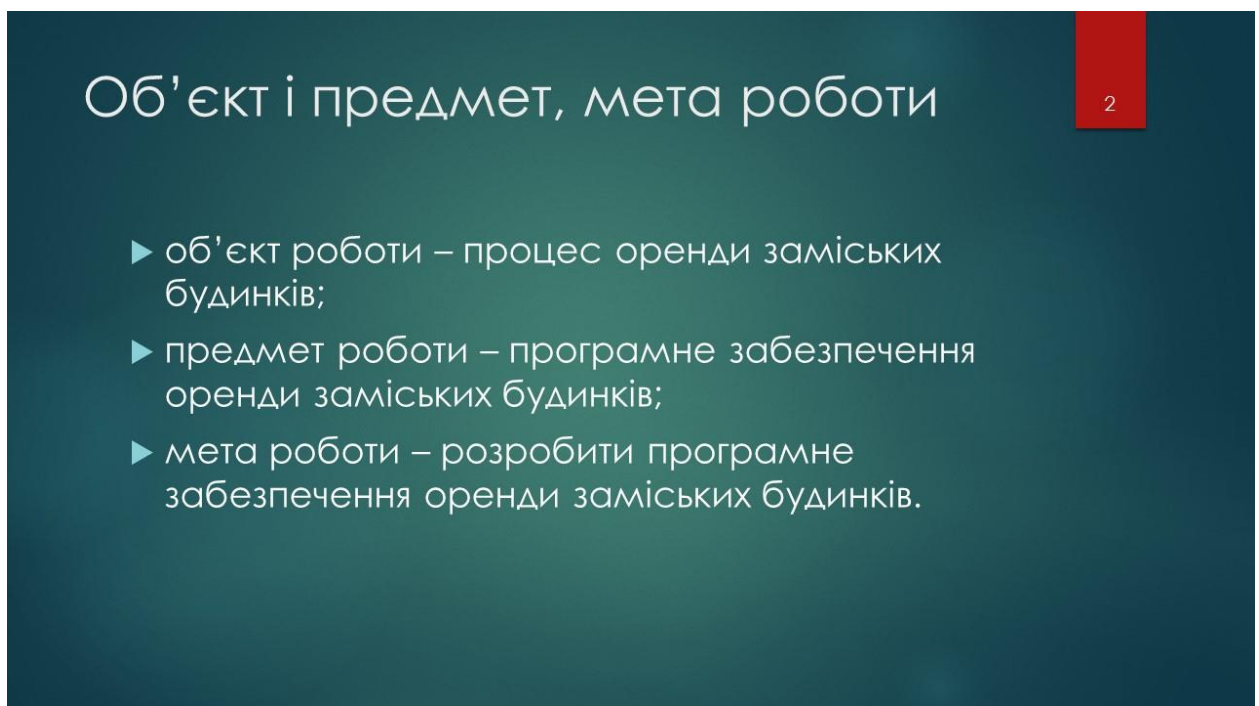


Рисунок В.2 – Об'єкт і предмет, мета роботи

Аналіз предметної області оренди замських будинків

3

Показник	CountryHouse-Company	Cottage-Gems	Holiday-Homes	Дипломна розробка
Пошук серед наявних замських будинків	–	+	+	+
Бронювання оренди	–	–	+	+
Виставлення власних пропозицій через додаток	–	–	+	+
Відгуки	–	+	+	+
Підтримка огляду будинків	–	–	–	+
Визначення деталей приміщень зі структуризацією	–	–	–	+

Рисунок В.3 – Аналіз предметної області оренди замських будинків

Порівняння технологій веброзробки

4

Показник	python	NodeJS
Тип технології	Мова програмування	Програмна платформа
Кросплатформність	+	+
Простота читання і написання коду	++	+–
Проста відслідковування помилок	++	+–
Швидкість виконання	+–	+
Документування	++	++

Рисунок В.4 – Порівняння технологій веброзробки

Діаграми прецедентів неавторизованого і авторизованого користувачів

5

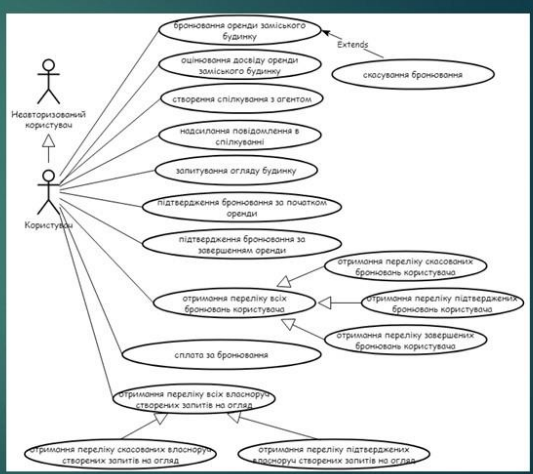


Рисунок В.5 – Діаграми прецедентів неавторизованого і авторизованого користувачів

Діаграми прецедентів агента і менеджера

6

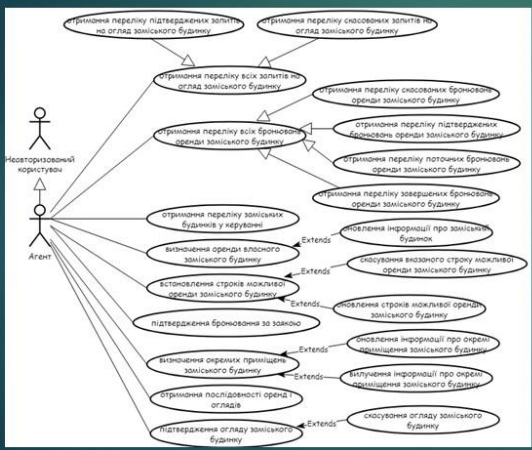


Рисунок В.6 – Діаграми прецедентів агента і менеджера

Логічна схема бази даних

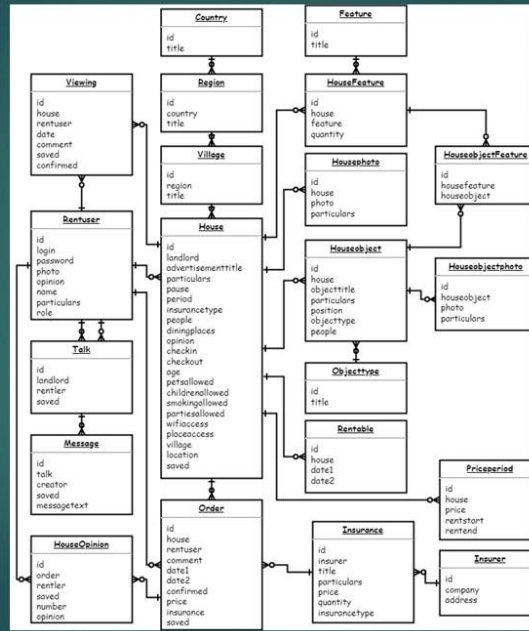


Рисунок В.7 – Логічна схема бази даних

Схема функціонування програми

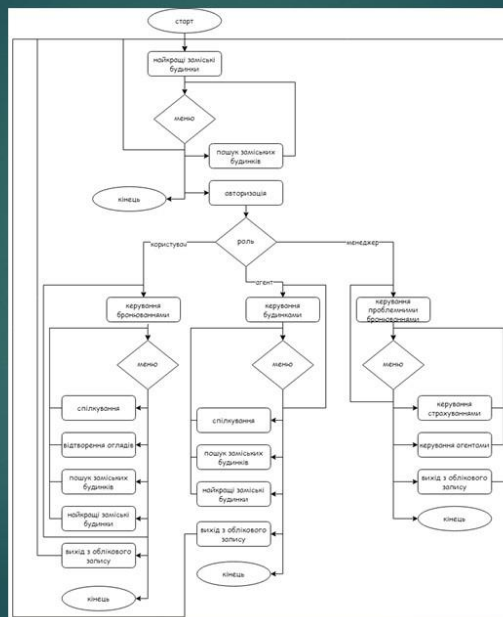


Рисунок В.8 – Схема функціонування програми

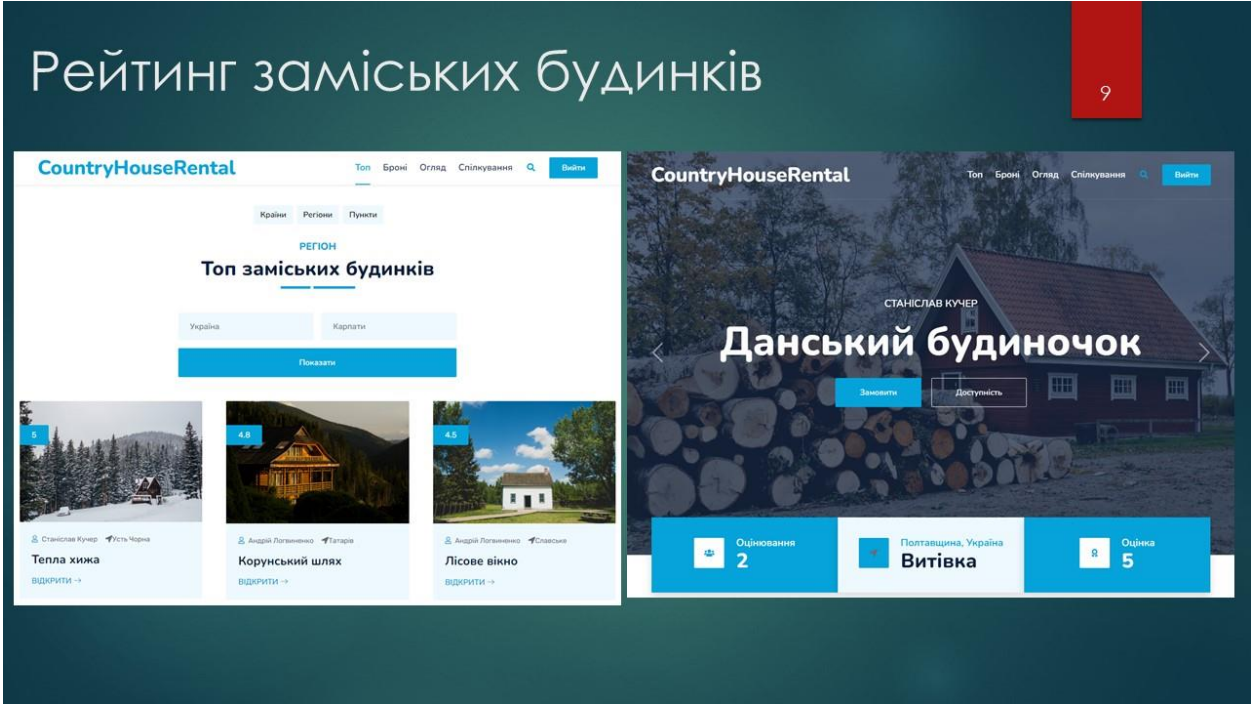


Рисунок В.9 – Рейтинг заміських будинків

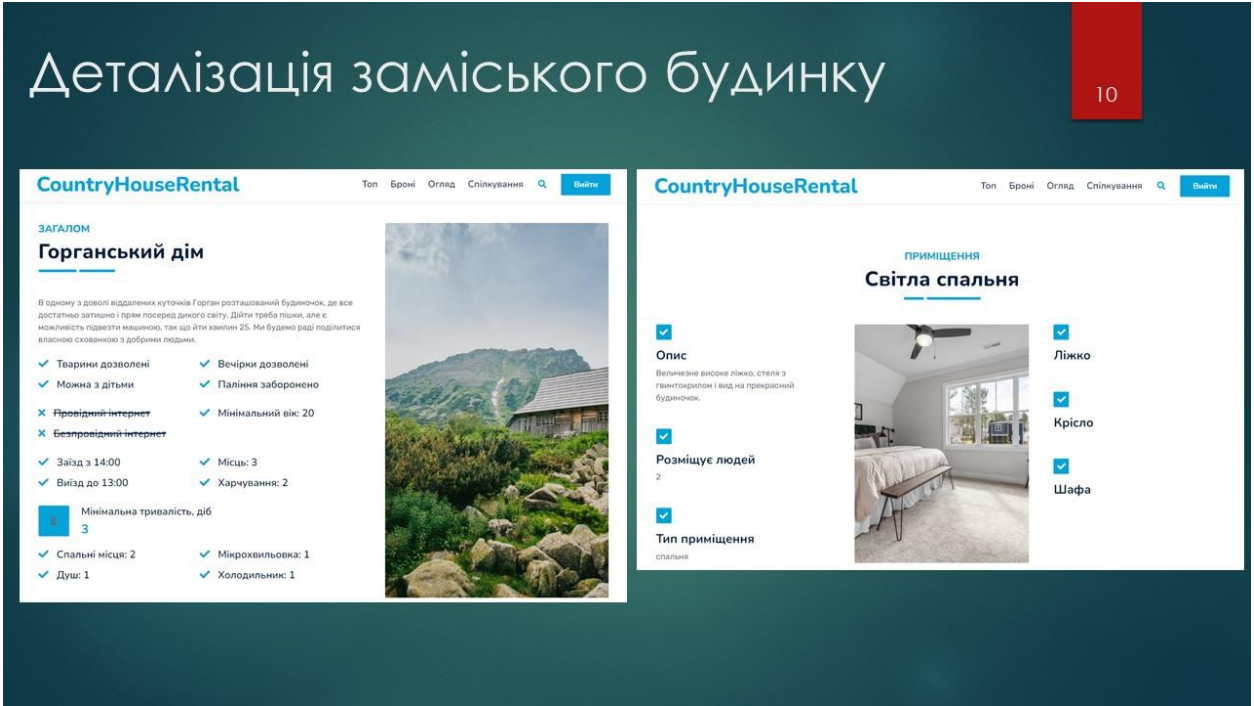


Рисунок В.10 – Деталізація заміського будинку

Робота агента з програмою

11

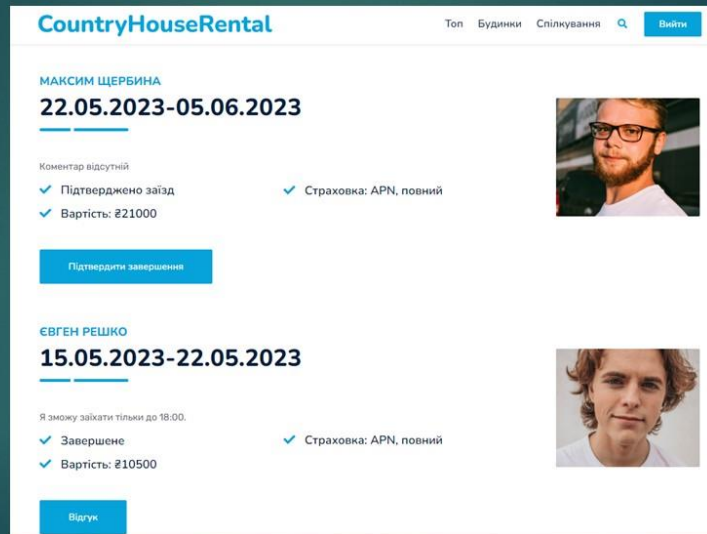


Рисунок В.11 – Робота агента з програмою

Висновки

12

- ▶ виконано аналіз програмних аналогів;
- ▶ створено детальну структуру бази даних для врахування особливостей оренди замських будинків, визначено і розподілено за ролями функціональні вимоги до програми стосовно неавторизованих користувачів, користувачів, агентів, які володіють або керують замськими будинками, менеджерів з побудовою діаграм прецедентів;
- ▶ реалізовано програмну систему як вебсистему з описом отриманих результатів.

Рисунок В.12 – Висновки