

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 ПІДТРИМКИ

(назва теми)

ТИМЧАСОВОГО ЗБЕРІГАННЯ РЕЧЕЙ

TEMPORARY STORAGE SOFTWARE

Виконав(ла): студент(ка) 4 курсу, групи КНТ-220сп

Спеціальності 122 Комп'ютерні науки

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Комп'ютерні науки

КАЛЮЖНИЙ І.К.

(ПРИЗВИЩЕ та ініціали)

Керівник ЛЬОВКІН В.М.

(ПРИЗВИЩЕ та ініціали)

Рецензент ТИМЕНКО А.В.

(ПРИЗВИЩЕ та ініціали)

2023

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет _____ ФКНТ _____
Кафедра _____ програмних засобів _____
Ступінь вищої освіти _____ бакалавр _____
Спеціальність _____ 122 Комп'ютерні науки _____
(код і найменування)
Освітня програма (спеціалізація) _____ Комп'ютерні науки _____
(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)

КАЛЮЖНОГО Іллі Костянтиновича

(ПРИЗВИЩЕ, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення підтримки тимчасового зберігання речей. Temporary Storage Software

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

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

2. Строк подання студентом проєкту (роботи) 1 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. 2. Проєктування програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

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

Слайди презентації

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	ЛЬОВКІН В.М., доцент		
Нормоконтроль	БЄЛОВА А.В., асистент		

7. Дата видачі завдання « 4 » 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)

_____ Ілля КАЛЮЖНИЙ
(підпис) (Ім'я ПРІЗВИЩЕ)

Керівник проєкту (роботи)

_____ Валерій ЛЬОВКІН
(підпис) (Ім'я ПРІЗВИЩЕ)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 93 с., 24 рис., 3 табл., 3 дод., 12 джерел.

ЗБЕРІГАННЯ РЕЧЕЙ, СХОВИЩЕ, РЕЗЕРВУВАННЯ, СПІЛЬНИЙ ДОСТУП, ВЕБРОЗРОБКА.

Об'єкт роботи – процес тимчасового зберігання речей. Предмет роботи – програмне забезпечення підтримки тимчасового зберігання речей. Мета роботи – розробити програмне забезпечення підтримки тимчасового зберігання речей для сприяння розширенню можливостей розміщення власних речей і оптимальному використанню власних приміщень.

Матеріали, методи та технічні засоби: мова програмування Python, засоби веброзробки, включаючи HTML, CSS, вебфреймворк Django, додаток Django PayPal та для керування базою даних система MySQL.

Результати. Розроблено програму, основним призначенням якої є забезпечення підтримки резервування для зберігання речей у орендованих орендарями сховищах, які надаються господарями на тимчасове використання.

Висновки. На даний момент потреба в забезпеченні приміщень для тимчасового зберігання речей на основі пропозицій господарів цих приміщень в світі не задоволена. Існує тільки один програмний засіб, який на території США вирішує цю проблему і активно розвивається. Створене програмне забезпечення дозволяє задовольнити цю потребу в Україні, надаючи господарям приміщень і людям, які знаходяться в пошуку сховищ, знайти можливість для взаємодії та взаємовигідно задовольнити потреби один одного.

Галузь використання. Побутові потреби людей, що включають необхідність пошуку приміщень для тимчасового зберігання речей.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 93 pages, 24 figures, 3 tables, 3 appendixes, 12 sources.

STORAGE OF THINGS, STORAGE, RESERVATION, SHARING, WEB DEVELOPMENT.

The object of the work is the process of temporary storage of things. The subject of the work is software for supporting the temporary storage of things. The purpose of the work is to develop software for supporting temporary storage of things to help expand the possibilities of placing one's own things and optimal use of one's own premises.

Materials, Methods and Technical Tools: Python programming language, web development tools including HTML, CSS, Django web framework, Django PayPal application and MySQL database management system.

The results. The program has been developed. Its main purpose is to provide backup support for storing things in storage facilities rented by tenants and provided by owners for temporary use.

Conclusions. At the moment the need to provide premises for temporary storage of things based on the proposals of the owners of these premises is not satisfied in the world. There is only one application which solves this problem in the USA and is actively developing. The created software makes it possible to satisfy this need in Ukraine, providing owners of premises and people who are looking for storage facilities with an opportunity to interact and mutually beneficially satisfy each other's needs.

Field of use. Household needs of people, including the need to find premises for temporary storage of things.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	10
1.1 Огляд існуючих програмних можливостей для підтримки тимчасового зберігання речей.....	10
1.2 Визначення особливостей програми, що розробляється	13
1.3 Висновки за розділом 1	15
2 Проєктування програми.....	16
2.1 Вибір засобів проєктування і розробки	16
2.2 Визначення функціональних вимог до програмного забезпечення	17
2.3 Проєктування структури бази даних.....	21
2.4 Висновки за розділом 2	31
3 Розробка програми	33
3.1 Порядок функціонування програми.....	33
3.2 Реалізація функцій додатку	34
3.3 Висновки за розділом 3	47
4 Експлуатація та тестування програми	48
4.1 Опис програми.....	48
4.2 Експлуатація програми.....	48
4.3 Тестування програми.....	53
4.4 Висновки за розділом 4	55
Висновки	56
Перелік джерел посилання	57
Додаток А Технічне завдання	59
Додаток Б Текст програми	64
Додаток В Слайди презентації.....	86

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

URL – Uniform Resource Locator;

БД – база даних.

ВСТУП

Загальновідомою є проблема оренди житла. Однак ця проблема має декілька фактично супутніх. Однією з супутніх проблем є оренда ділянки для паркування автомобіля. Другою є оренда місця для зберігання речей [1]. Наприклад, орендуючи житло, людина може мати необхідність розмістити певні власні речі, оскільки вони не вміщуються в орендоване місце проживання. Ця ж потреба може виникати і у тих, хто має власне житло. Іноді для розміщення автомобіля потрібен гараж. Це також фактично оренда сховища.

Часом людина може переїхати в інше місто або поїхати на відпочинок, при цьому не орендуючи житло в тому місті, звідки поїхала, а потребуючи в ньому зберігати речі. Це може бути не відпочинок, а робота за кордоном. Окрім того, коли людина приїжджає в інше місто, там в неї може виникати потреба залишити речі на певний період, наприклад, у випадку тривалої подорожі з різними пунктами в маршруті, щоб повернутися потім до місця зберігання речей. Звичайно, що в такому випадку можна використовувати централізовані сервіси збереження. Наприклад, на території вокзалів, однак такі варіанти мають певне коло обмежень, а загалом зазвичай є недешевими, а часом і ненадійними. Оренда у сторонніх господарів часто і дешевша, і зручніша [1]. Також можна шукати, вибираючи з ширшого набору варіантів, ніж у більшості централізованих сховищ. Тому застосування приміщень, які надаються господарями, є ідеєю, яка надає подальшого активного розвитку галузі зберігання речей. Будівельні матеріали, автомобілі, одяг – все це може потребувати знаходження відповідного варіанта. Господарі житла можуть мати окремі кімнати для зберігання речей біля своїх квартир і не використовувати їх, можуть мати льох на території приватної ділянки, а також підвал, приміщення на даху, все це може бути задіяно і приносити прибуток власнику.

Загальний перелік випадків, коли люди потребують винаймання приміщення для збереження речей, можна продовжувати, але в принципі це визначає, що потреба при зберіганні речей у винайманні для цього певного сховища є досить актуальною і потребує відповідного програмного забезпечення.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд існуючих програмних можливостей для підтримки тимчасового зберігання речей

Проведений огляд існуючих програмних можливостей для підтримки тимчасового зберігання речей спочатку був направлений на виявлення подібних програмних продуктів. Але в результаті було виявлено тільки одну програму, яка базується на тій же самій ідеї оренди сховищ від господарів для тимчасового зберігання речей. Ця програма називається Neighbor [2]-[3] (рис. 1.1).

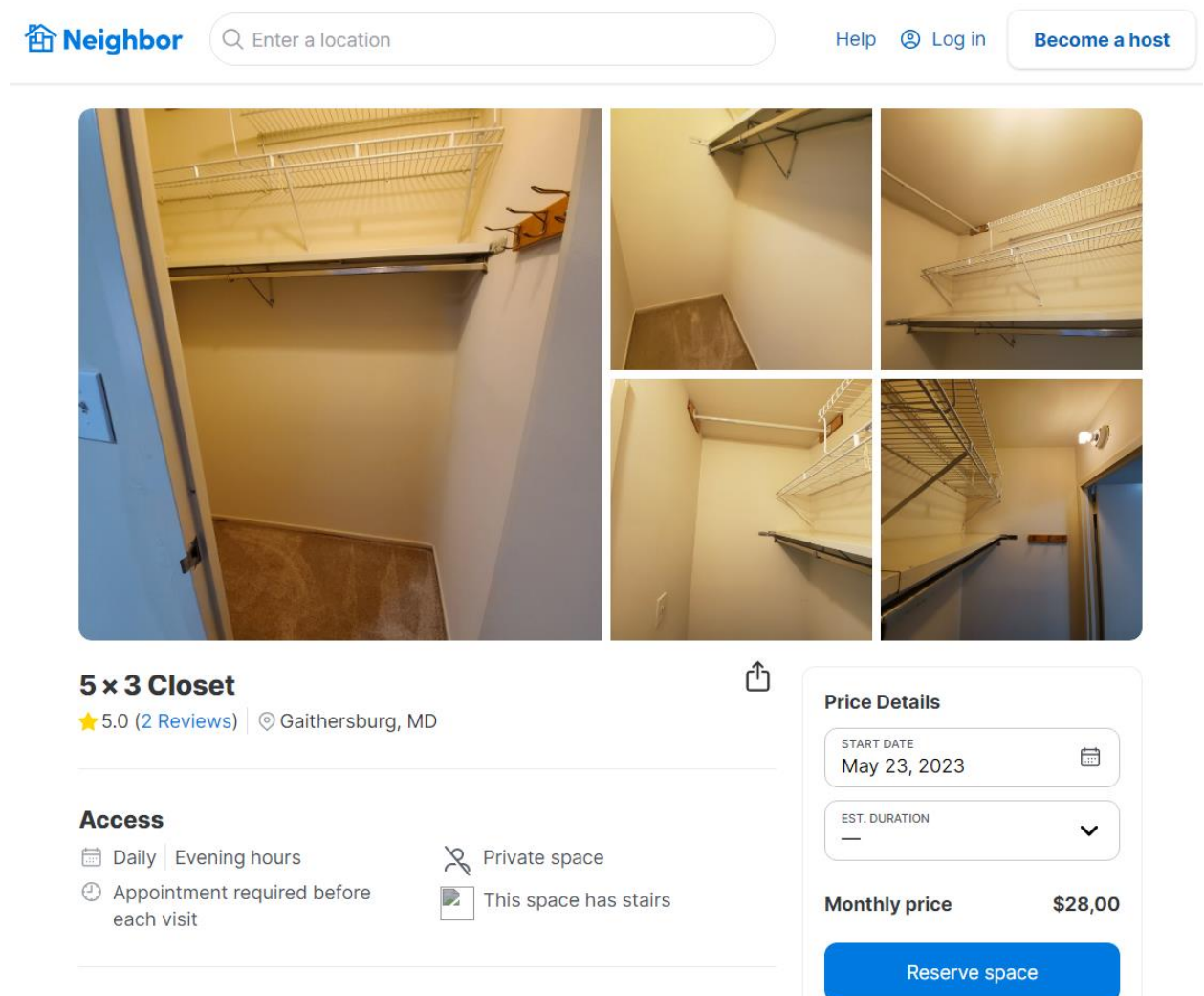


Рисунок 1.1 – Сторінка сховища для зберігання речей в програмі Neighbor

Наявність такого аналогу, який фактично в подальшому має використовуватися як прототип цієї розробки, дозволила встановити основні функціональні можливості, необхідні для роботи в цій предметній області на практиці. Однак окрім цього було встановлено, що ця програма використовується тільки в Сполучених Штатах Америки. Тобто всі оголошення, наявні на вебсайті, всі можливості пошуку обмежені тільки однією країною. Ні в яких інших країнах подібних програмних засобів не було виявлено. Це говорить про актуальність розробки, адже відповідно є величезний світовий ринок, який ще не охоплено наявними програмами. Звичайно, що в першу чергу в цій роботі цікавість викликає Україна. І зрозуміло, що в людей періодично виникає потреба в збереженні речей, але її важко вирішити на практиці на стільки ж легко, як орендувати житло. Звичайно, що можна стандартно орендувати приміщення (квартиру, будинок) для цього або знайти інший подібний варіант, але варто розуміти, що це значно більші витрати ніж орендувати окрему кімнату або спеціалізоване для цього приміщення, як наприклад, льох, підвал або подібний варіант. Відповідно фактично інструменту, який би міг використовуватися для вирішення цієї проблеми на даний момент в Україні немає.

Neighbor, як будь-яка подібна програма для оренди, зокрема оренди житла (Airbnb, Booking), в якості основної функції взаємодії з користувачем використовує пошук. Початково кожному користувачу окрім пошуку надається можливість стати господарем, тобто одразу перейти до виставлення власного сховища для оренди. Це звичайно одразу передбачає виконання реєстрації або авторизації. Окрім цієї функціональності доступна також інформація щодо сервісу, послуг і т.д., тобто загальна інформація.

Пошук також може бути представлений деякими швидкими варіантами, наприклад, переліком основних міст, типами збереження, але всі вони призводять до відкриття пошуку з наперед встановленими значеннями параметрів в залежності від обраного варіанта.

Проте після виконання пошуку надається доступ до цілого ряду споріднених функцій: перегляду самого оголошення, перегляду відгуків про сховище з оголошення, контактування з господарем, перегляду сторінки господаря зі всіма його доступними сховищами та відгуками про його сховища (рис. 1.2).

Відгуки про орендарів, які створили відгуки про сховище, недоступні, тобто зрозуміти орендарю, на скільки відгуку варто довіряти, важко.

Оренда сховищ для зберігання відбувається тільки на місячній основі, тобто мінімально можна орендувати приміщення на місяць. Тривалість оренди – кількість місяців.

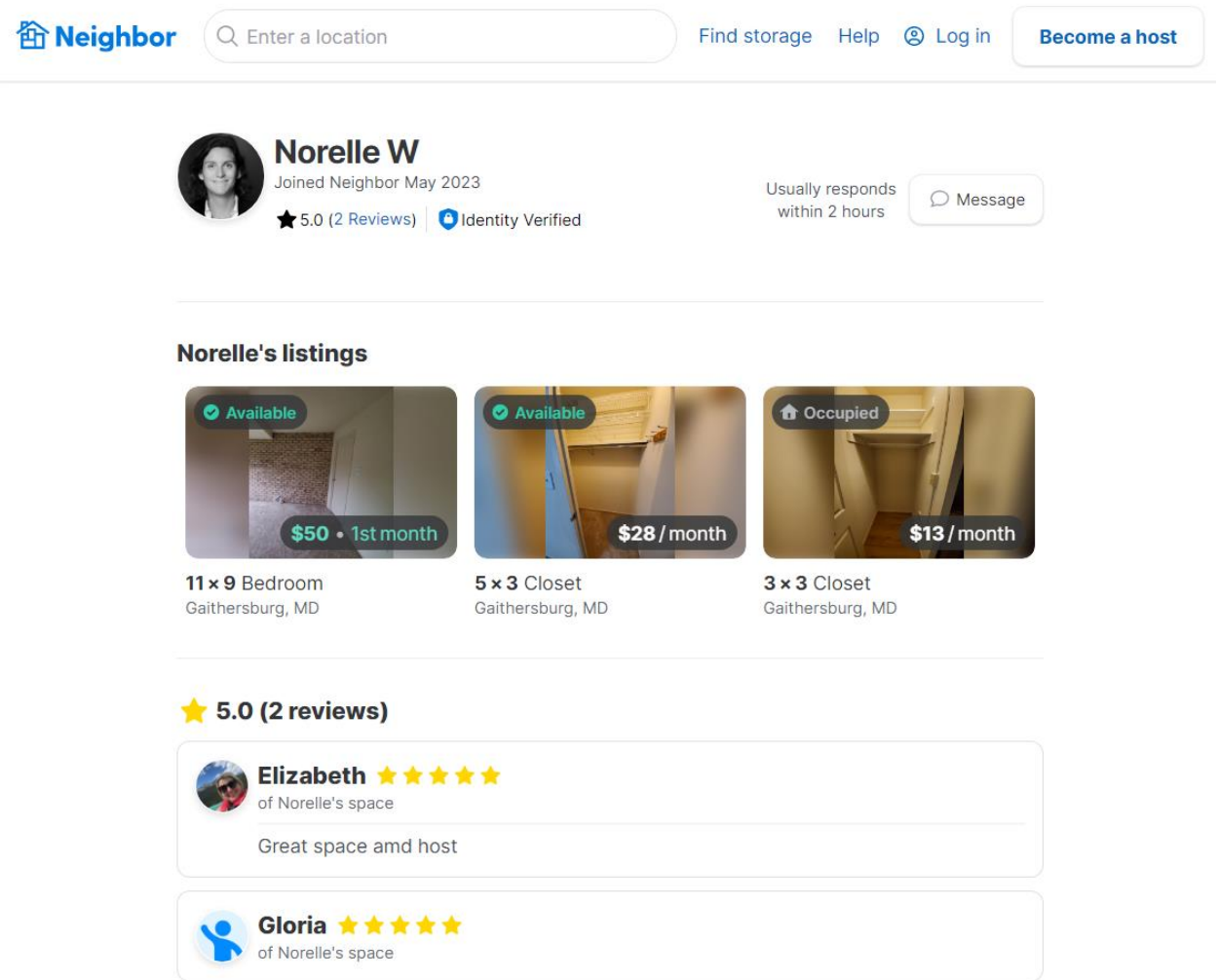


Рисунок 1.2 – Сторінка господаря сховища у програмі Neighbor

Окремо слід звернути на результати впровадження цієї програми. На даний момент це компанія вже з 5 роками на ринку [4]. За даними на 2019 рік, які було широко оприлюднено, компанія мала збільшення за фінансовими результатами в 5 разів на рік, що є величезним темпом [1]-[4]. У 2018 році компанія змогла зібрати 2,5 мільйони доларів на фінансування, що знову ж таки додатково підкреслює цінність ідеї, привабливість її в очах інвесторів [1]-[4]. Проте конкретні цифри фінансових результатів компанія не розкриває [4].

1.2 Визначення особливостей програми, що розробляється

Грунтуючись на отриманих результатах під час огляду, було вирішено у програмі, що розробляється в роботі, розширити підтримку тимчасового зберігання речей на Україну та надати додаткові можливості, які можуть сприяти привабливості програми для користувачів, які можуть виставити свої приміщення в якості сховищ для зберігання речей. Це є дуже важливим, оскільки у випадку успішності дозволяє розширювати базу користувачів, що є критичним для розвитку програми. Окрім того наявність пропозицій буде сприяти більшому залученню користувачів, які будуть ставати орендарями, адже для них критичним у свою чергу є можливість вибору і знаходження того варіанта, який відповідає їхнім запитам.

Отож прототипом розробки є Neighbor, але окрім впровадження ідеї в Україні передбачається розширення функціональних можливостей (табл. 1.1). Так у розробці передбачається розширити варіанти тривалості оренди, надаючи сховище в оренду і на день або деяку кількість днів, а не тільки на місяць як у випадку Neighbor. Це має сприяти тому, щоб люди, які відправляються у відпустку, які подорожують і навпаки бажають залишити свої речі в тому місті, куди вони приїхали, мали можливість це зробити.

Заплановано розширити систему дисконтів, адже в Neighbor зменшення вартості можна отримати тільки на перший місяць після резервування сховища на деяку кількість місяців. Передбачається, що в

розробці це буде можливість визначити дисконт і на перший період, і на всі наступні періоди, а також, що таких варіантів можна буде встановити декілька.

Таблиця 1.1 – Порівняння розробки з прототипом

Характеристика	Neighbor	Розробка
Країни пропозицій	США	Україна
Пошук сховищ для збереження речей від господарів	Так	Так
Мінімальний період оренди	Місяць	Місяць, день
Дисконт за тарифом	На перший місяць	Різні варіанти
Направленість відгуків	Сховище	Сховище, орендар
Спільний доступ до резервувань	Ні	Так
Спільний доступ до оголошень при пошуку сховищ	Ні	Так

Також важливим є визначення відгуків не тільки в один бік. Хоча звичайно критичними є відгуки про сховища, адже в цій предметній області, саме орендар залишає своє речі і потенційно ризикує тим, що ці речі можуть бути втрачені або вкрадені. Це відповідно змінює напрямок порівняно з орендуванням житла, де як раз більш критичними є відгуки про орендарів, адже вони можуть зіпсувати житло господаря. Проте і господарі повинні мати розуміння, кому вони надають свій простір для оренди, чи не виникали проблеми з цим орендарем раніше. Адже речі можуть містити і заборонені речі, наприклад.

Оскільки орендувати сховище може сім'я або декілька людей разом, то під час пошуку, а також після резервування зручним для них може бути надання спільного доступу, щоб вони бачили необхідну інформацію в зручному для всіх форматі, тому спільний доступ до резервувань і спільний доступ до оголошень при пошуку сховищ є відмінними функціями програми.

1.3 Висновки за розділом 1

Визначеними завданнями на роботу є:

- аналіз предметної області тимчасового зберігання речей;
- визначення вимог і особливостей програмного забезпечення підтримки тимчасового зберігання речей;
- проєктування програмного забезпечення підтримки тимчасового зберігання речей;
- реалізація програмного забезпечення підтримки тимчасового зберігання речей і тестування.

Даний розділ є результатом виконання завдання аналізу предметної області тимчасового зберігання речей. За результатами аналізу було виявлено тільки один наявний аналог програми, яка розробляється в роботі, у світі, яким є програма Neighbor. Вона дозволяє виконувати зберігання речей, орендуючи для цього приміщення тільки на території Сполучених Штатів Америки, відповідно в Україні таких можливостей немає. Також було вирішено надати ряд покращень щодо функціональних можливостей розробки порівняно з аналогом.

2 ПРОЄКТУВАННЯ ПРОГРАМИ

2.1 Вибір засобів проєктування і розробки

Python [5]-[6] за рахунок вебфреймворків став однією з найбільш активно використовуваних мов для розробки вебдодатків середнього і великого масштабу. Виходячи з результатів першого розділу, вебдодаток, який розробляється, належить саме до такого типу. Серед всіх вебфреймворків Python було виокремлено два, які мають найбільш придатні характеристики для вирішення цієї проблеми: Django [7]-[10] і TurboGears (табл. 2.1). Вони подібні за функціями і можливостями, але Django має більш потужні засоби захисту від різного роду атак, а в TurboGears потрібен додатковий аналіз, пошук додаткових інструментів або додаткові витрати на реалізацію додаткового коду, тому для роботи обрано саме Django.

Таблиця 2.1 – Порівняння вебфреймворків для розробки

Показник	Django	TurboGears
Тип	Повний стек	Повний стек
Вбудована адміністративна панель	Так	Так
Засоби безпеки	Засоби проти декількох напрямків атак, дослідження командою нових вразливостей	Вбудовані тільки мінімальні
Бібліотека шаблонів	Так, власна	Так, Kajiki
Засоби роботи з базою даних	Об'єктно-реляційне відображення	Об'єктно-реляційне відображення
Засоби забезпечення авторизації	Так	Так

2.2 Визначення функціональних вимог до програмного забезпечення

Користувач, який потребує зберігання речей в сховищі і для того виконує його пошук, називається орендарем. Йому доступні наступні функції:

- пошук сховища для збереження речей;
- відтворення всіх відгуків господаря;
- відтворення всіх відгуків за всіма сховищами господаря;
- відтворення всіх відгуків орендаря;
- відтворення повних даних орендаря;
- відтворення повних даних господаря;
- відтворення всіх відгуків за сховищем;
- відтворення всіх відгуків за орендарем;
- залишення відгуку про сховище;
- відтворення всіх сховищ господаря;
- резервування сховища господаря;
- відхилення резервування;
- розширення резервування сховища для зберігання речей;
- внесення сплати за наступний період резервування;
- надання повних даних про сховище;
- надсилання повідомлення господарю;
- відтворення розмови з господарем;
- відтворення розпочатих розмов з господарями;
- розділення доступу до резервування з іншим орендарем;
- відтворення резервувань зі спільного доступу;
- відтворення всіх доступних для зберігання речей орендарем резервувань;
- внесення сховища для детального розгляду;
- відтворення сховищ для детального розгляду;
- змінювання власних даних;

- відтворення сховищ для детального розгляду зі спільного доступу;
- вилучення сховища з детального розгляду;
- надання спільного доступу до сховища для детального розгляду іншому орендарю;
- вилучення спільного доступу до сховища для детального розгляду іншим орендарем;
- надсилання скарги за резервуванням (рис. 2.1).

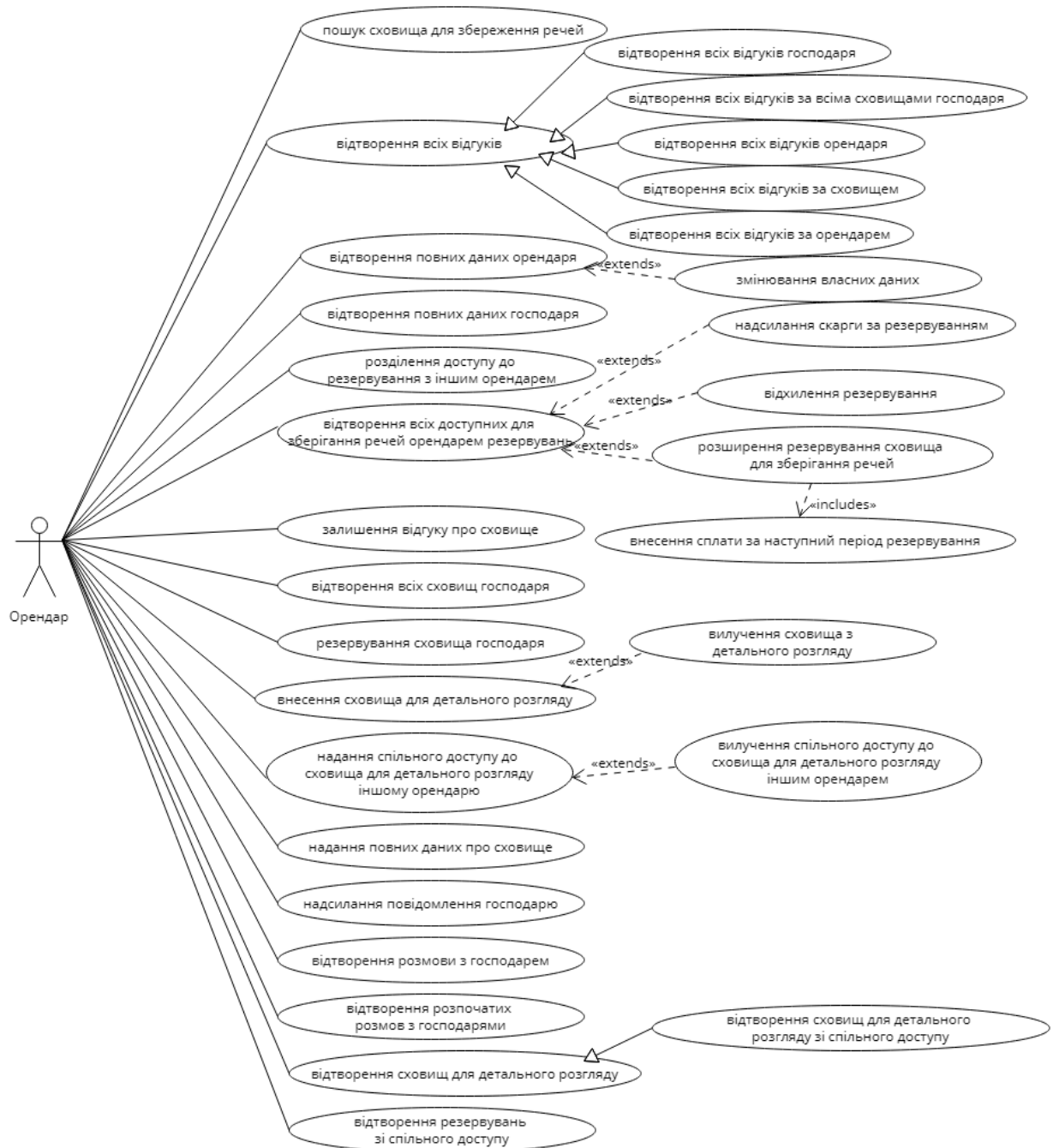


Рисунок 2.1 – Діаграма прецедентів орендаря

Користувачі, які виставляють власні приміщення для зберігання речей, називаються господарями і мають наступні функції:

- пошук сховища для збереження речей;
- відтворення всіх відгуків господаря;
- відтворення всіх відгуків за всіма сховищами господаря;
- відтворення всіх відгуків орендаря;
- відтворення повних даних орендаря;
- відтворення повних даних господаря;
- відтворення всіх відгуків за сховищем;
- відтворення всіх відгуків за орендарем;
- залишення відгуку про орендаря;
- змінювання даних про сховище для резервування;
- виставлення власного сховища для резервування для зберігання речей;
- змінювання власних даних;
- відтворення всіх сховищ господаря;
- відтворення резервувань за сховищем господаря;
- відтворення неопрацьованих господарем резервувань;
- підтвердження резервування;
- відхилення резервування;
- встановлення тарифу на зберігання речей у сховищі;
- встановлення вільних дат для резервування сховища;
- відтворення всіх актуальних тарифів на зберігання речей у сховищі;
- змінювання тарифу на зберігання речей у сховищі;
- встановлення дисконту за тарифом;
- відтворення можливих дисконтів за тарифом;
- вилучення тарифу на зберігання речей у сховищі;
- вилучення дисконту за тарифом;
- надання повних даних про сховище;
- надсилання повідомлення орендарю;

- відтворення розмови з орендарем;
- відтворення розпочатих розмов з орендарями;
- надсилання скарги за резервуванням (рис. 2.2).

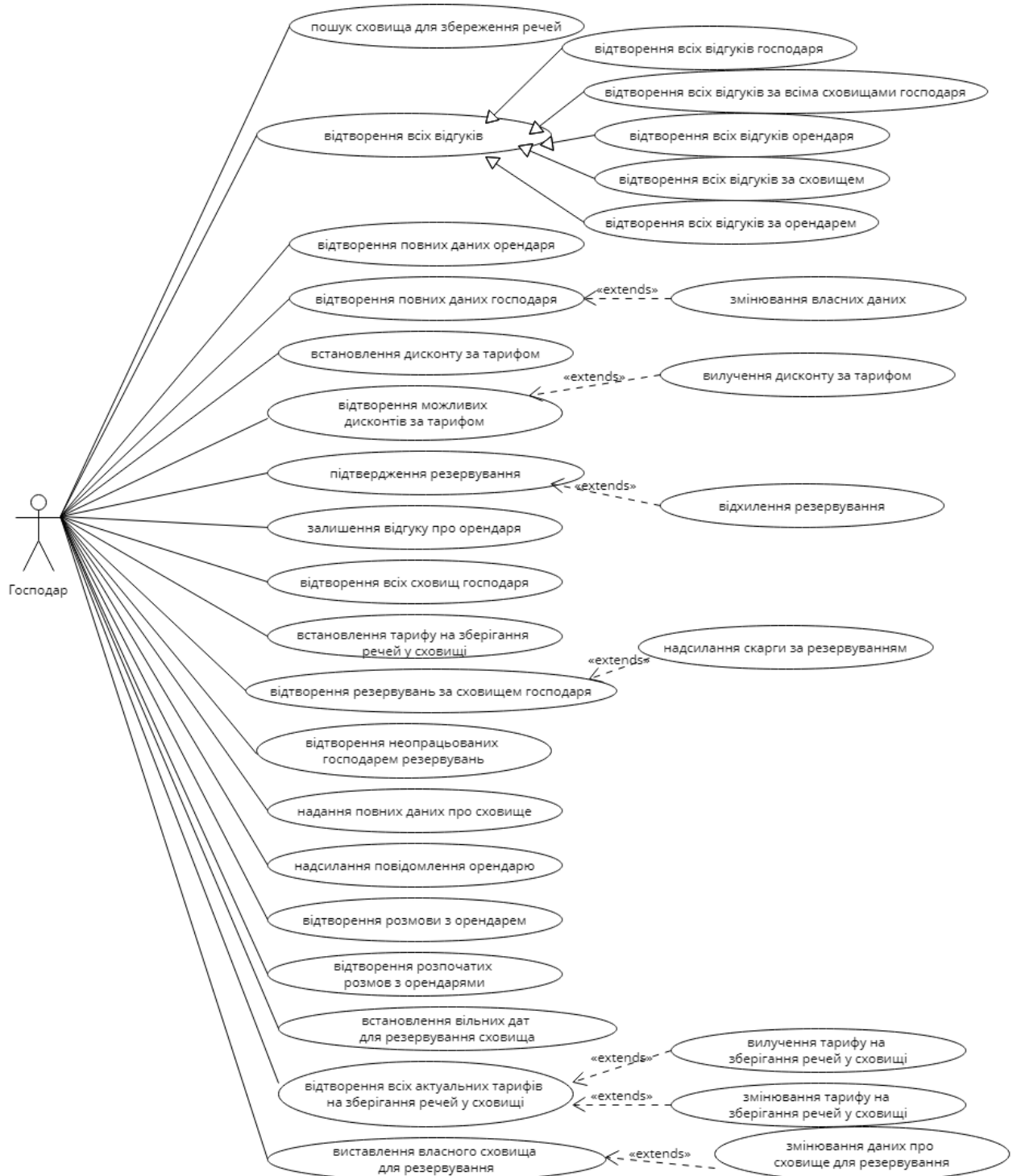


Рисунок 2.2 – Діаграма прецедентів господаря сховища

Менеджери виконують фактично керування конфліктами, які виникають, тому їм доступні функції:

- закриття доступу користувача (орендаря або господаря) після аналізу скарг;
- відтворення всіх скарг за резервуваннями;
- відтворення всіх непроаналізованих скарг за резервуваннями;
- надання відповіді на скаргу (рис. 2.3).



Рисунок 2.3 – Діаграма прецедентів менеджера

2.3 Проектування структури бази даних

При проектуванні бази даних (БД) важливим було врахуванням вимог до програми визначених особливостей програми, а також визначення використовуваних засобів, основним з яких була система керування БД MySQL [11].

Речі зберігаються у сховищах, які надаються їхніми власниками у тимчасове користування. Сховища характеризуються цілою множиною параметрів і зберігаються в таблиці Storage, яка включає такі поля:

- id – номер, що ідентифікує сховище;
- host – господар сховища, який визначається записом з таблиці Host, тобто є зовнішнім ключем;
- title – назва сховища, яка використовується на вебсторінці сховища для користувацької ідентифікації;
- slug – текстовий ідентифікатор сторінки, на якій розташовується інформація про сховище (частина адреси Uniform Resource Locator (URL), яка

фактично використовується замість номера з поля id з метою підтримки безпеки роботи з даними);

- description – опис цього сховища;
- storage_type – тип сховища: підвал, дах, скриня, вуличне сховище і т.д., зберігається як ціле число;
- vehicle – можливість запаркувати транспортний засіб у сховищі або поруч з ним;
- inner – можливість паркування транспортного засобу на вулиці або всередині приміщення, якщо парковка взагалі доступна (в полі vehicle);
- place_longitude – координата адреси сховища, довгота;
- place_latitude – координата адреси сховища, широта;
- place – місце, де розташовується сховище: фактично це місце або селище, або інший тип населеного пункту, визначається записом з таблиці Place, тобто є зовнішнім ключем;
- size_type – тип доступного розміру сховища, визначається записом з таблиці Sizetype, тобто є зовнішнім ключем;
- height – висота стелі у приміщенні сховища;
- private_entrance – наявність приватного входу в приміщення сховища, тобто можливість самостійно пройти у сховище;
- closed_territory – варіант доступу до сховища: закритий (сховище розташовується окремо, господар не має спільних речей зі сховищем, тобто щонайменше є деякі двері до цього приміщення), спільний (приміщення загалом відділяється, але господар також зберігає речі в тому самому приміщенні, хоча розміри сховища визначені саме ті, які доступні лише орендарю), відкритий;
- stairs – наявність сходів, які потрібно використовувати для того, щоб дістатися сховища;
- no_smoking – заборона палити у приміщенні;
- conditioner – наявність кондиціонера в приміщенні сховища;
- refrigerator – наявність холодильника в приміщенні сховища;

- freezer – наявність морозильної камери в приміщенні сховища;
- electricity – можливість підключення до електромережі в приміщенні сховища;
- fire_detector – наявність системи попередження про пожежу в приміщенні, зокрема в приміщенні сховища;
- alarm – наявність сигналізації в приміщенні сховища;
- locked – можливість закрити сховище на замок;
- no_pet – відсутність домашніх тварин у приміщенні, яке контактує зі сховищем або у самому сховищі;
- shelves – наявність полиць для збереження речей у приміщенні сховища;
- hangers – наявність вішалок для збереження речей у приміщенні сховища;
- light – наявність освітлення у приміщенні сховища;
- cctv – наявність відеоспостереження у приміщенні сховища;
- star – усереднена оцінка сховища;
- registration – необхідність встановлення домовленостей або інформування господаря для відвідування приміщення сховища;
- driveway – наявність під'їзду для прямого під'їзду до сховища або поруч з ним, щоб легше можна було переносити речі з автомобіля;
- access_type – правила щодо періодичності доступу: без обмеження, тільки в будні дні, раз на тиждень, раз на місяць;
- access_start – час, з якого протягом дня можна отримати доступ у сховище;
- access_end – час, до якого протягом дня можна отримати доступ у сховище;
- next_month – вартість сплати за наступний місяць: продовження резервування відбувається на місяць, це стандартна сума, яка в подальшому стягується за резервування цього сховища.

Для розуміння особливостей сховища окрім вказаних характеристик також використовуються фотографії. Фотографій може бути декілька для кожного сховища. Кожна фотографія зберігається в таблиці Storagephoto, яка включає такі поля:

- id – номер, що ідентифікує фотографію;
- storage – сховище, яке зображує фотографія, визначається записом з таблиці Storage, тобто є зовнішнім ключем;
- photo – сама фотографія сховища.

Для того щоб характеризувати розміри сховища, краще використовувати не конкретні значення розмірів, адже для орендаря в подальшому встановлювати конкретні розміри в сантиметрах не надто корисний варіант. А ось мати певний варіант, який визначає сховище, куди можуть поміститися тільки деякі скрині, сховище, куди може поміститися звичайний легковий автомобіль, сховище, куди може поміститися кемпер або подібний великий автомобіль, – це більш яскрава і важлива характеристика. Таким чином орендар може одразу співставляти тип розміру з варіантом використання такого сховища. Тип розміру сховища зберігається в таблиці Sizetype, яка включає такі поля:

- id – номер, що ідентифікує тип розміру сховища;
- length_size – базовий розмір такого типу розміру сховища за довжиною;
- width_size – базовий розмір такого типу розміру сховища за шириною;
- type_name – назва типу розміру сховища, що використовується для ідентифікації;
- description – опис призначення (використання) такого типу розміру сховища.

Тарифи на резервацію сховищ зберігаються в таблиці Tariff, яка включає такі поля:

- id – номер, що ідентифікує тариф;

- storage – сховище, якого стосується тариф, визначається записом з таблиці Storage, тобто є зовнішнім ключем;
- unit – одиниця часу, за якою вимірюється тариф (фактично в програмі передбачається або день, або місяць);
- min_duration – мінімальна тривалість (днів або місяців відповідно), за якої використовується цей тариф;
- start_date – дата, з якої починається дія цього тарифу;
- end_date – дата, до якої діє цей тариф;
- cost – вартість одиниці часу за цим тарифом (добі або місяця).

До тарифів може за певних умов (певної тривалості резервації за цим тарифом) застосовуватися дисконт для зменшення вартості, щоб стимулювати довші резервації. Дисконти зберігаються в таблиці Discount, яка включає такі поля:

- id – номер, що ідентифікує дисконт;
- tariff – тариф, до якого застосовується такий дисконт, визначається записом з таблиці Tariff, тобто є зовнішнім ключем;
- duration – мінімальна кількість одиниць часу, вказаних у тарифі, за якої застосовується цей дисконт: наприклад, кількість місяців (якщо в тарифі вказано місяць), за якої починає діяти дисконт;
- start_type – застосовується тільки до першої одиниці часу або до всіх, тобто якщо одиницею часу є місяць, то дисконт застосовується до першого місяця або до всіх місяців резервації;
- percentage – відсоток, на який зменшується вартість тарифу.

Відгуки про орендарів зберігаються в таблиці TenantReview, яка включає такі поля:

- id – номер, що ідентифікує відгук;
- host – господар, який створив цей відгук, визначається записом з таблиці Host, тобто є зовнішнім ключем;
- tenant – орендар, про якого створено цей відгук, визначається записом з таблиці Tenant, тобто є зовнішнім ключем;

- creation_date – дата створення відгуку;
- star – оцінка у відгуку;
- review_text – текст у відгуку.

Відгуки про сховища зберігаються в таблиці StorageReview, яка включає такі поля:

- id – номер, що ідентифікує відгук;
- storage – сховище, про яке було створено відгук, визначається записом з таблиці Storage, тобто є зовнішнім ключем;
- tenant – орендар, який створив цей відгук, визначається записом з таблиці Tenant, тобто є зовнішнім ключем;
- creation_date – дата створення відгуку;
- star – оцінка у відгуку;
- review_text – текст у відгуку.

Внесення сховища для детального розгляду зберігаються в переліку орендаря в таблиці Storagelist, яка включає такі поля:

- id – номер, що ідентифікує входження до переліків орендарів;
- tenant – орендар, який вніс сховище для детального розгляду в подальшому, визначається записом з таблиці Tenant, тобто є зовнішнім ключем;
- storage – сховище, яке було внесено орендарем для детального розгляду в подальшому, визначається записом з таблиці Storage, тобто є зовнішнім ключем.

Детальний розгляд в подальшому може відбуватися не тільки орендарем, який виконав збереження, але і людьми, з якими він разом хоче приймати рішення щодо вибору сховища. Надання доступу до детального розгляду сховища іншим орендарям зберігається в таблиці Storagelisttenant, яка включає такі поля:

- id – номер, що ідентифікує надання доступу до детального розгляду сховища орендарю;

– storagelist – входження до переліку орендаря, до якого відкривається доступ, визначається записом з таблиці БД Storagelist, тобто є зовнішнім ключем;

– tenant – орендар, якому надано доступ до сховища для детального розгляду в подальшому, визначається записом з таблиці Tenant, тобто є зовнішнім ключем.

Місця (місто, селище), де можуть знаходитись сховища, зберігаються в таблиці Place, яка включає такі поля:

- id – номер, що ідентифікує місце;
- country – країна, в якій знаходиться це місце;
- name – назва місця (міста або селища, населеного пункту).

Відкриті до оренди дати за сховищем зберігаються в таблиці Freedate, яка включає такі поля:

- id – номер, що ідентифікує відкритий до оренди сховища інтервал часу;
- storage – сховище, за яким відкритий інтервал, визначається записом з таблиці Storage, тобто є зовнішнім ключем;
- start_date – дата, з якої починається інтервал часу;
- end_date – дата, до якої триває інтервал часу.

Резервування сховища зберігається в таблиці Reserve, яка включає такі поля:

- id – номер, що ідентифікує резервацію;
- storage – сховище, яке намагається зарезервувати орендар або яке зарезервував орендар, визначається записом з таблиці Storage, тобто є зовнішнім ключем;
- tenant – орендар, який виконав резервування, визначається записом з таблиці Tenant, тобто є зовнішнім ключем;
- amount – сумарна вартість резервування;
- creation_date – дата створення резервування;
- start_date – дата початку зберігання речей у сховищі;

- reserved_end_date – дата закінчення зберігання речей у сховищі, яка визначена спочатку в резервуванні;
- waited_end_date – очікувана на даний момент дата закінчення зберігання речей у сховищі (вже не початкова);
- result_end_date – фактична (остаточна) дата закінчення зберігання речей у сховищі;
- status – стан резервування.

Так само як і сховища для подальшого розгляду, орендар також може поділитися з іншими орендарями і резервуваннями. Це може використовуватися для спільної роботи, використовуючи для цього таблицю Reservetenant, яка включає такі поля:

- id – номер, що ідентифікує доступ орендаря до резервування;
- reserve – резервування, до якого надано доступ, визначається записом з таблиці Reserve, тобто є зовнішнім ключем;
- tenant – орендар, якому надано доступ до резервування, визначається записом з таблиці Tenant, тобто є зовнішнім ключем.

Господарі мають облікові записи в системі, але вони також мають і певну специфічну інформацію про себе, що зберігається в таблиці Host, яка включає такі поля:

- user_id – номер облікового запису, який господар використовує для роботи в програмі;
- respond – середній час очікування відповіді від господаря;
- bankmail – електронна пошта, що використовується в PayPal-акаунті користувача, куди будуть перенаправлені кошти;
- description – інформація, надана господарем про себе.

За резервуваннями виконуються платежі, при чому таких платежів може бути декілька, адже в подальшому резервування для збереження речей у певному сховищі може продовжуватися. Тоді потрібно, наприклад, кожен місяць вносити сплату. Сплати за зберігання речей у сховищі зберігаються в таблиці Payment, яка включає такі поля:

- id – номер, що ідентифікує сплату;
- reserve – резервування, за яким внесено сплату, визначається записом з таблиці Reserve, тобто є зовнішнім ключем;
- bank – код PayPal-платежу;
- amount – сума, яку було сплачено;
- pay_date – дата, коли було виконано або має бути виконано сплату;
- start_date – дата початку дії резервування за цією сплатою;
- end_date – дата завершення дії резервування за цією сплатою;
- status – стан сплати: кожен раз, якщо резервування не закінчено, створюється платіж, а орендар має його оплатити, відповідно поки він його не оплатить, зберігається фактично шаблон для нагадування орендарю, а після сплати він стає фактичним платежом.

Облікові записи, які використовуються господарями і орендарями для роботи з програмою, зберігаються в таблиці User, яка включає такі поля:

- id – номер, що ідентифікує користувача за обліковим записом;
- first_name – ім'я користувача;
- last_name – прізвище користувача;
- email – електронна пошта;
- username – логін користувача;
- password – пароль;
- is_staff – доступ до адміністративної панелі, що зокрема також визначає, чи це господар або орендар, чи це менеджер;
- is_active – обліковий запис активний або ні: може використовуватися для блокування користувачів;
- date_joined – дата реєстрації;
- last_login – дата останньої авторизації.

Орендарі мають свої власні характеристики і зберігаються в таблиці Tenant, яка включає такі поля:

- user_id – номер, що ідентифікує орендаря, визначається записом з таблиці User, тобто є зовнішнім ключем;

- description – інформація, надана орендарем про себе;
- star – усереднена оцінка орендаря.

Орендарі та господарі можуть обмінюватися повідомленнями, щоб орендар міг уточнити питання, які його цікавлять перед тим, як виконати резервування. Дані контактування зберігаються в таблиці Contact, яка включає такі поля:

- id – номер, що ідентифікує контактування;
- host – господар, який бере участь у контактуванні, визначається записом з таблиці Host, тобто є зовнішнім ключем;
- tenant – орендар, який бере участь у контактуванні, визначається записом з таблиці Tenant, тобто є зовнішнім ключем;
- author – автор повідомлення (господар або орендар);
- creation_date – дата, час відправлення повідомлення;
- contact_text – текст повідомлення (рис. 2.4).

На резервування можуть поступати скарги. Вони зокрема необхідні для вирішення спірних питань, проблем, що виконується менеджерами. Господар міг не надати приміщення, речі могли вкрасти і т.д. Для цього подаються скарги. Скарги зберігаються в таблиці БД Claim, яка включає такі поля:

- id – номер, що ідентифікує скаргу;
- reservation – резервування, якого стосується скарга, визначається записом з таблиці Reserve, тобто є зовнішнім ключем;
- author – сторона, яка є автором скарги: господар або орендар;
- manager – менеджер, який виконує керування скаргою;
- creation_date – дата створення скарги;
- solution_date – дата вирішення скарги;
- problem – опис проблеми зі сторони автора скарги;
- answer – відповідь за проблемою від менеджера.

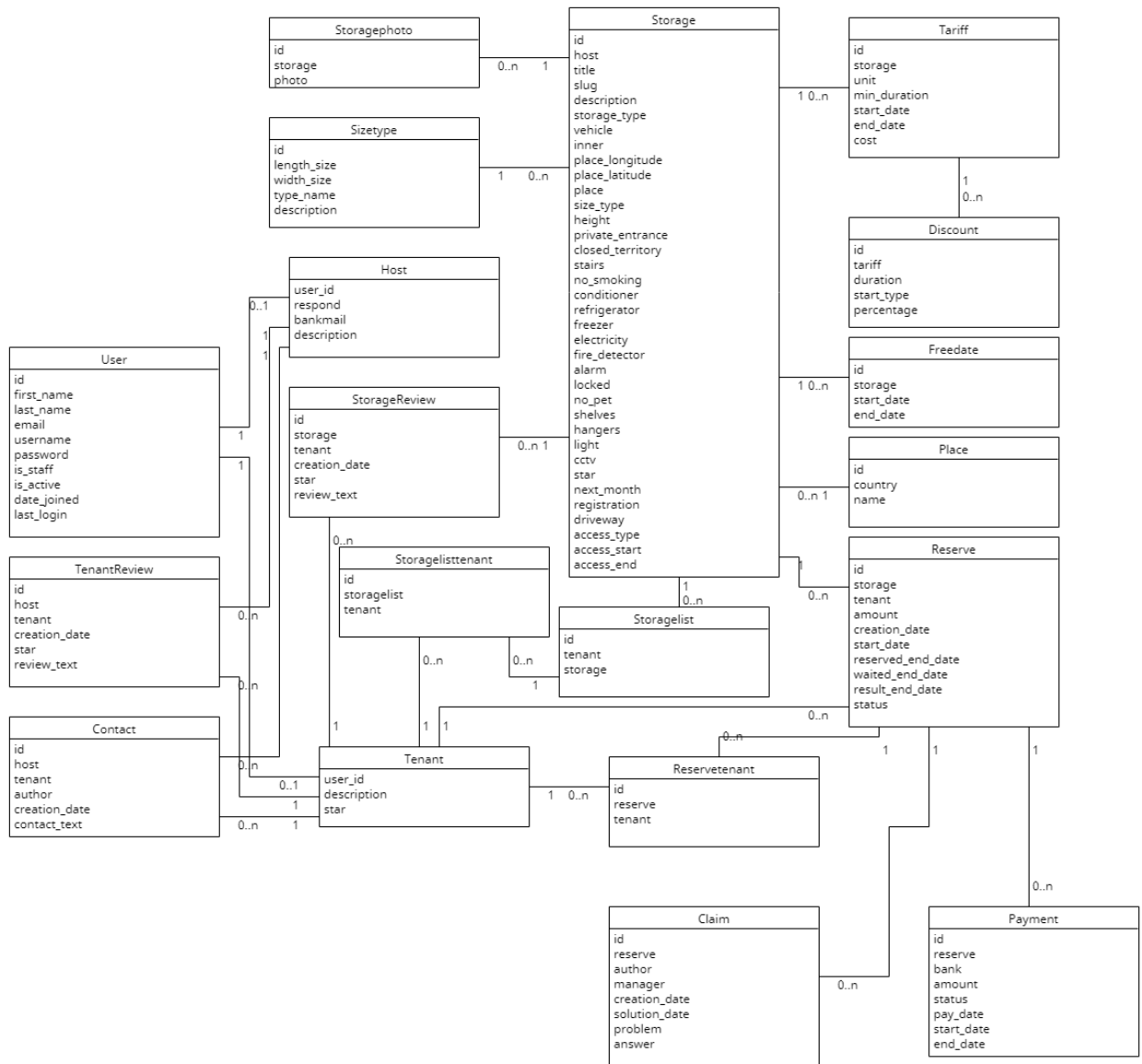


Рисунок 2.4 – Схема БД

2.4 Висновки за розділом 2

Вибір на користь Python і Django для реалізації вебдодатку дозволяє спростити процес розробки і створити в результаті надійний інструмент для підтримки процесу тимчасового зберігання речей.

Згідно з особливостями, відмінностями програми та базовими можливостями створено діаграми прецедентів для господаря, орендаря та менеджера.

Проектування роботи з даними призвело до створення схеми БД, що фактично у відповідності з архітектурою Django-додатків призвело до визначення структури моделей додатку.

3 РОЗРОБКА ПРОГРАМИ

3.1 Порядок функціонування програми

Для функціонування програми пошук сховищ для зберігання речей є центральним процесом, адже він фактично для користувача визначає доступ до майже всіх інших можливостей, які пов'язані вже з окремими сховищами і т.д. Те, на скільки зручним є цей пошук, впливає на те, чи знайде фактично користувач потрібне йому сховище. Тому в процесі відбувається відбір сховищ, які не відповідають заданим параметрам (рис. 3.1). Після перевірки за окремими параметрами, що відповідають полям таблиці БД, реалізується перевірка всіх відібраних сховищ. За кожним сховищем перевіряється, чи є за ним відкриті дати, чи вони не зайняті іншими резервуваннями та чи вартість резервування потрапляє в заданий інтервал.

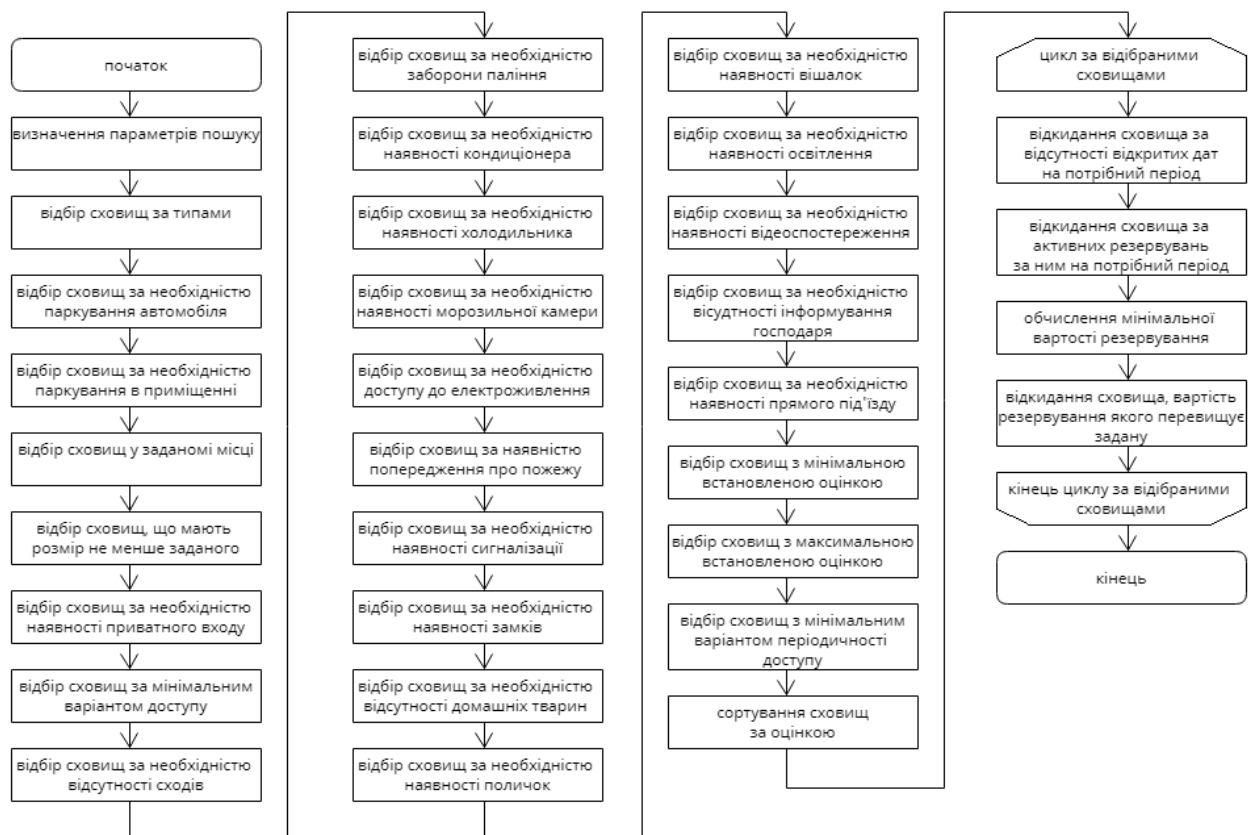


Рисунок 3.1 – Алгоритм пошуку сховища для зберігання речей

При обчисленні вартості мають враховуватися і тарифи місячні, і тарифи денні. За кожним тарифом, що підходить до параметрів потенційного резервування, заданого в пошуку (період резервування), обчислюється вартість з вибором найбільшого дисконту за цим тарифом. Зі всіх тарифів обирається мінімальна результуюча вартість. Вона перевіряється на входження в заданий інтервал вартості. Якщо вартість входить в інтервал, то сховище залишається в результатах. У протилежному випадку з результатів відкидається таке сховище.

3.2 Реалізація функцій додатку

Функції додатку забезпечуються через підтримку на основі відповідних складових додатку, що відповідають структурі додатку Django:

- моделей додатку, що включають класи, розроблені в чіткій відповідності, але з врахуванням особливостей даних класів, оскільки вони дозволяють забезпечити реалізацію в коді без створення окремої моделі зв'язку багато-до-багатьох: відповідно клас моделі Reserve має поле tenants, з якого можна отримати перелік орендарів, які мають спільний доступ з власником резервування до самого резервування, а клас моделі Storagelist має поле tenants, яке відповідно визначає перелік орендарів, які мають спільний доступ з власником до відповідного збереженого ним сховища;
- файла admin.py, в якому зареєстровано всі відповідні моделі для роботи з БД на основі адміністративної панелі;
- менеджера URL, в якому встановлено зв'язок фактично кожної функції, презентованої відповідним представленням, з URL-адресою, що використовується для звернення до представлення;
- шаблонів вебсторінок, що відповідають за роботу з кожною відповідною функцією та визначають позиції підвантаження даних, наданих відповідним представленням у кожному необхідну для цього позицію на сторінці;

– представлень, які реалізовані окремими функціями у програмі, кожна з яких визначає відповідну користувацьку функцію: їх розбито на окремі файли з представленнями функцій спільних для всіх ролей користувачів, з представленнями функцій менеджера, з представленнями функцій орендаря та представленнями функцій господарів.

Представлення відтворення всіх відгуків за орендарем забезпечено функцією `tenant_reviews(request, tenantUsername = None, star = 0)`, яка має параметри: `tenantUsername` визначає логін користувацького запису орендаря, за яким виділяються відгуки, та `star`, що визначає бали, виставлені у відгуку, які потрібно відобразити (при нульовому значенні відображаються всі відгуки). Функція працює і для варіанта, коли логін орендаря не задано, тоді використовуються дані авторизованого користувача. Тобто фактично це або перегляд відгуків про себе в режимі орендаря, або перегляд відгуків про заданого орендаря у всіх інших випадках. Підтримка цього представлення забезпечується через шаблон сторінки `reviews_tenant.html`. Шаблони вебсторінок створені за допомогою шаблону [12].

Відтворення всіх відгуків орендаря забезпечено представленням `by_tenant_reviews(request, tenantUsername = None, star = 0)`, яке має параметри: `tenantUsername` визначає логін користувацького запису орендаря, за яким виділяються відгуки, та `star`, що визначає бали, виставлені у відгуку, які потрібно відобразити. Для відображення всіх відгуків потрібно встановити значення `star` в 0. Функція працює і для варіанта, коли логін орендаря не задано, тоді використовуються дані авторизованого користувача. Виконується або перегляд відгуків, створених заданим орендарем, або перегляд відгуків, створених самим авторизованим користувачем. Підтримка цього представлення забезпечується через шаблон сторінки `reviews_by_tenant.html`.

Відтворення всіх відгуків за сховищем забезпечено представленням `storage_reviews(request, storageSlug, star = 0)`, яке має параметри: `storageSlug` визначає адресу сховища, відгуки про яке потрібно показати, та `star`, що визначає бали, виставлені у відгуку, які потрібно відобразити. Для

відображення всіх відгуків потрібно встановити значення `star` в 0. Підтримка цього представлення забезпечується через шаблон сторінки `reviews_storage.html`.

Представлення відтворення всіх відгуків за всіма сховищами господаря забезпечено функцією `host_reviews(request, hostUsername = None, star = 0)`, яке має параметри: `hostUsername` визначає логін господаря, відгуки про сховища якого потрібно показати, та `star`, що визначає бали, виставлені у відгуку, які потрібно відобразити. Для відображення всіх відгуків потрібно встановити значення `star` в 0. Функція працює і для варіанта, коли логін орендаря не задано, тоді використовуються дані авторизованого користувача. Виконується або перегляд відгуків про сховища, якими володіє заданий за логіном господар, або про сховища, якими володіє авторизований користувач, якщо логін не задано. Підтримка цього представлення забезпечується через шаблон сторінки `reviews_host.html`.

Відтворення всіх відгуків господаря забезпечено представленням `by_host_reviews(request, hostUsername = None, star = 0)`, яке має параметри: `hostUsername` визначає логін господаря, відгуки якого про орендарів потрібно показати, та `star`, що визначає бали, виставлені у відгуку, які потрібно відобразити. Для відображення всіх відгуків потрібно встановити значення `star` в 0. Функція працює і для варіанта, коли логін господаря не задано, тоді використовуються дані авторизованого користувача. Виконується або перегляд відгуків, створених заданим господарем, або перегляд відгуків, створених самим авторизованим користувачем. Підтримка цього представлення забезпечується через шаблон сторінки `reviews_by_host.html`.

Представлення відтворення повних даних орендаря забезпечено функцією `tenant_info(request, tenantUsername = None)`, яка має параметр `tenantUsername`, що визначає логін орендаря, за яким потрібно надати інформацію. Підтримка цього представлення забезпечується через шаблон сторінки `tenant.html`, дані в який передаються з представленням словником з ключами:

- `tenant` – об'єкт орендаря;
- `reviews` – 5 останніх відгуків про орендаря;
- `reviews_by` – 5 останніх відгуків, створених орендарем.

Представлення відтворення повних даних господаря забезпечено функцією `host_info(request, hostUsername = None)`, яка має параметр `hostUsername`, що визначає логін господаря, за яким потрібно надати інформацію. Підтримка цього представлення забезпечується через шаблон сторінки `host.html`, дані в який передаються з представленням словником з ключами:

- `host` – об'єкт господаря;
- `reviews` – 5 останніх відгуків про будь-яке сховище господаря;
- `reviews_by` – 5 останніх відгуків, створених господарем;
- `storages` – 3 сховища цього господаря.

Відтворення всіх сховищ господаря забезпечено представленням `host_storages(request, hostUsername = None, role = 0)`, яке має параметри: `hostUsername`, що визначає логін господаря, сховища якого потрібно відобразити, та `role`, що визначає роль користувача, в якій він звернувся до цієї функції. Роль необхідна для того, щоб розуміти, чи відтворення відбувається в режимі орендаря, чи в іншому режимі. Якщо це орендар, то він може зберегти сховища для подальшого детального розгляду, тому потрібно включати інформацію про те, чи збережено зараз це сховище, чи ні. Для цього кожне сховище, що в результаті повертається в словнику до шаблону сторінки `host_storages.html` за ключем `storages`, включає для орендаря атрибут `saved`, який містить значення `True`, якщо це сховище було вже збережено авторизованим орендарем, та значення `False` у протилежному випадку.

Залишення відгуку орендарем про сховище і залишення відгуку господарем про орендаря забезпечено єдиним представленням `review(request, object = None, role = 0)`, яке має параметр `object`, що визначає або адресу сховища, або логін орендаря, про якого потрібно залишити відгук. На те, якому саме варіанту відповідає цей параметр, впливає другий параметр – `role`,

який визначає роль, в якій відкривається сторінка залишення відгуку. Якщо це значення дорівнює 0, то створюється відгук про сховище, а якщо значення дорівнює, то створюється відгук про орендаря. У процесі перевіряється наявність резервувань, які відповідно були створені авторизованим орендарем про це сховище (перший варіант) або які були створені заданим орендарем і стосуються одного зі сховищ авторизованого господаря. Якщо такі резервування, при чому не в стані тільки створеного (без підтвердження або початкового скасування), є, то дозволяється створити відгук. Створення відгуку реалізується на основі форми `StorageReviewForm` для першого варіанта або на основі форми `TenantReviewForm` для другого варіанта. При збереженні самого відгуку переобчислюється середня оцінка відповідно для сховища (перший варіант) або для орендаря (другий варіант).

Також внесення даних на основі форми реалізується:

- представленням виставлення власного сховища для резервування для зберігання речей, що працює на основі функції `new_storage(request)`;
- надсилання скарги за резервуванням, що працює на основі функції `claim(request)`.
- резервування сховища господаря, що працює на основі функції `reserve(request, storage)`, що має параметр `storage`, який визначає адресу сховища, а всі параметри самого резервування задаються користувачем в формі, відповідно дані автора резервування встановлюються як дані авторизованого користувача, перевіряючи, що є такий орендар у програмі, а також виконується перевірка того, що на задані в формі дати відкрите резервування на сховище `storage`, що на ці дати немає підтверджених резервувань, та в підсумку виконується обчислення вартості, як це описано для пошуку, резервування, а отримане значення передається в представлення `payment`, основна робота якого підтримується через додаток `Django PayPal`;
- встановлення тарифу на зберігання речей у сховищі, що працює на основі функції `tariff(request, storage)`, що має параметр `storage`, який визначає адресу сховища;

- встановлення вільних дат для резервування сховища, що працює на основі функції `freedate(request, storage)`, що має параметр `tariff`, який визначає тариф, за яким створюється дисконт;

- встановлення дисконту за тарифом, що працює на основі функції `discont(request, tariff)`, що має параметр `tariff`, який визначає тариф, за яким створюється дисконт.

Фактично тими самими формами користуються і представлення, які забезпечують функцій змінювання даних:

- надання відповіді на скаргу, що працює на основі функції `claim_answer(request, claim)`, що має параметр `claim`, який визначає номер скарги, а оскільки може використовуватися тільки менеджером, то відбувається перевірка того, що авторизований користувач має менеджерський запис;

- змінювання даних про сховище для резервування, що працює на основі функції `edit_storage(request)`;

- змінювання тарифу на зберігання речей у сховищі, що працює на основі функції `edit_tariff(request)`;

- розширення резервування сховища для зберігання речей, що працює на основі функції `edit_reserve(request, reserve)`;

Представлення підтвердження резервування реалізується функцією `submit_reserve(request, reserve)`, що має параметр `reserve`, який визначає резервування, підтвердження якого виконується через зміну його стану і збереження внесених змін.

Представлення відхилення резервування реалізується функцією `cancel_reserve(request, reserve)`, що має параметр `reserve`, який визначає резервування, скасування якого виконується через зміну його стану і збереження внесених змін.

Вилучення тарифу на зберігання речей у сховищі реалізується представленням на основі функції `delete_tariff(request, tariff)`, що має параметр `tariff` з номером тарифу, за яким перевіряється, що він стосується сховища,

яким володіє як господар авторизований користувач, а після успішної перевірки реалізується вилучення з БД тарифу.

Вилучення дисконту за тарифом реалізується представленням на основі функції `delete_discount(request, discount)`, що має параметр `discount` з номером дисконту, за яким перевіряється, що він стосується сховища, яким володіє як господар авторизований користувач, а після успішної перевірки реалізується вилучення з БД тарифу.

Вилучення сховища з детального розгляду реалізується представленням на основі функції `delete_storagelist(request, storage)`, що має параметр `storage` з відповідною частиною URL-адреси сховища, за яким перевіряється, що він стосується сховища, яким володіє як господар авторизований користувач, а після успішної перевірки реалізується вилучення з БД тарифу.

Вилучення спільного доступу до сховища для детального розгляду іншим орендарем реалізується представленням на основі функції `delete_storagelisttenant(request, storage, tenant)`, що має параметри `storage` з відповідною частиною URL-адреси сховища та `tenant` з логіном орендаря, для якого потрібно прибрати спільний доступ, при цьому перевіряється, що цей спільний доступ існує і що він був створений авторизованим користувачем або що авторизований користувач має заданий логін, після успішної перевірки реалізується вилучення з БД запису.

Представлення пошуку сховища для збереження речей забезпечено функцією `find_storages(request)`, яка всі параметри пошуку отримує через запит, включаючи до них:

- `storage` – потрібні типи сховища, що мають бути перелічені відповідними номерами через кому;
- `vehicle` – необхідність запаркувати транспортний засіб у сховищі або поруч з ним (будь-яке встановлене значення відповідає такій необхідності);

- inner – необхідність паркування транспортного засобу всередині приміщення (будь-яке встановлене значення відповідає такій необхідності) або на вулиці (коли відсутнє значення параметра);
- place – місце, в якому необхідне сховище: задається його назвою;
- size_type – необхідний тип розміру сховища: відбираються всі сховища, довжина і ширина яких не менша ніж довжина і ширина вказаного типу сховища;
- entr – необхідність приватного входу в приміщення сховища (будь-яке встановлене значення відповідає такій необхідності);
- territor – варіант доступу до сховища: відбираються всі, що мають не менший варіант доступу ніж заданий;
- stairs – необхідність відсутності сходів (будь-яке встановлене значення вказує на таку необхідність);
- smoke – необхідність забори палити у приміщенні (будь-яке встановлене значення вказує на таку необхідність);
- condit – необхідність наявності кондиціонера в приміщенні сховища (будь-яке встановлене значення вказує на таку необхідність);
- refriger – необхідність наявності холодильника в приміщенні сховища (будь-яке встановлене значення вказує на таку необхідність);
- freeze – необхідність наявності морозильної камери в приміщенні сховища (будь-яке встановлене значення вказує на таку необхідність);
- electricity – необхідність наявності підключення до електромережі в приміщенні сховища (будь-яке встановлене значення вказує на таку необхідність);
- fire – необхідність наявності системи попередження про пожежу в приміщенні (будь-яке встановлене значення вказує на таку необхідність);
- alarm – необхідність наявності сигналізації в приміщенні сховища (будь-яке встановлене значення вказує на таку необхідність);
- locked – необхідність замикання сховища на замок (будь-яке встановлене значення вказує на таку необхідність);

– pets – необхідність відсутності домашніх тварин у приміщенні, яке контактує зі сховищем або у самому сховищі (будь-яке встановлене значення вказує на таку необхідність);

– shelves – необхідність наявності полиць для збереження речей у приміщенні сховища (будь-яке встановлене значення вказує на таку необхідність);

– hangers – необхідність наявності вішалок для збереження речей у приміщенні сховища (будь-яке встановлене значення вказує на таку необхідність);

– light – необхідність наявності освітлення у приміщенні сховища (будь-яке встановлене значення вказує на таку необхідність);

– cctv – необхідність наявності відеоспостереження у приміщенні сховища (будь-яке встановлене значення вказує на таку необхідність);

– reg – необхідність відсутності встановлення домовленостей або інформування господаря для відвідування приміщення сховища (будь-яке встановлене значення вказує на таку необхідність);

– way – необхідність наявності під'їзду для прямого під'їзду до сховища або поруч з ним, щоб легше можна було переносити речі з автомобіля (будь-яке встановлене значення вказує на таку необхідність);

– star – мінімальна усереднена оцінка сховища;

– star_to – максимальна прийнятна усереднена оцінка сховища;

– access – варіант періодичності доступу до сховища: відбираються всі, що мають не менший варіант періодичності доступу ніж заданий;

– from – час, з якого протягом дня потрібно орендарю обов'язково отримати доступ у сховище;

– to – час, до якого протягом дня потрібно отримати орендарю доступ у сховище;

– money – мінімальна необхідна вартість оренди сховища;

– top – максимальна допустима вартість оренди сховища на заданий період;

– begin – початок періоду, з якого необхідно виконати резервування сховища;

– finish – закінчення періоду, до якого необхідно виконати резервування сховища.

Для відбору сховищ на необхідні дати виконується спочатку через перевірку того, що господар приміщення вказав задані дати як відкриті для резервування. Якщо це так, то далі перевіряється, що немає жодного резервування зі станом не менше ніж скасоване або створене (без підтвердження), яке включає вказані дати. У випадку, якщо таких резервувань не знайдено, далі перевіряється вартість резервування за кожним сховищем, залишаючи його в результаті тільки тоді, якщо вартість резервування знаходиться в потрібному інтервалі. Для обчислення вартості виконується обчислення вартості за денних тарифів, обираючи з них найдешевший, а потім використовуються місячні тарифи, для кожного з яких застосовується найбільший прийнятний дисконт, а в підсумку з місячних тарифів також обирається найдешевший варіант. З цих двох варіантів обирається найменше значення і саме воно порівнюється з необхідним інтервалом вартості. Перевіряються при цьому і випадки, коли задані дати містяться в декількох тарифах (на різні дати, тобто період резервування розбивається на частини).

Надання повних даних про сховище забезпечено функцією `storage(request, storage, reserve_from = None, reserve_to = None, role = 0)`, яка має параметри `storage`, що визначає адресу сховища, `reserve_from`, що визначає дату початку резервування, `reserve_to`, що визначає дату завершення резервування, `role`, що визначає роль користувача, який переглядає сторінку сховища. Підтримка цього представлення забезпечується через шаблон сторінки `storage.html`. Представлення передбачає не тільки отримання даних про саме сховище, але і перевірку можливості резервування сховища на вказані дати. Вся необхідна інформація повертається за ключем `storage`. Аналогічно як і для пошуку виконується обчислення найменшої вартості резервування цього сховища на вказані дати, а отримане значення

повертається в атрибуті `cost`. Якщо зарезервувати це сховище на ці дати неможливо, то атрибут має значення `-1`. У атрибуті `saved` зберігається інформація про те, чи вніс авторизований користувач, якщо робота реалізується в режимі орендаря, це сховище для детального розгляду в подальшому.

Функції відтворення розмови з орендарем, відтворення розмови з господарем, надсилання повідомлення орендарю і надсилання повідомлення господарю реалізовані представленням `contact(request, username, role = 0)`, яке має параметри: `username`, що визначає логін користувача (орендаря для господаря, господаря для орендаря), контактування з яким стосується переписки, `role`, що визначає роль, в якій звертається до представлення користувач. Це представлення використовує в основі форму для створення повідомлення та завантажує перелік повідомлень між авторизованим користувачем і користувачем, логін якого передано в параметрі `username`. Додатково в результатах повертається об'єкт користувача, який є стороною автора повідомлення (ключ `author`), та об'єкт користувача, який є протилежною стороною (ключ `contact`).

Відтворення розпочатих розмов з господарями і відтворення розпочатих розмов з орендарями реалізовані представленням `contacts(request, role = 0)`, який має параметр `role`, що визначає роль користувача, в якій він звернувся до представлення. Відповідно за цією роллю орендарю надається перелік господарів, а господарю – перелік орендарів.

Відтворення всіх скарг за резервуваннями і відтворення всіх непроаналізованих скарг за резервуваннями реалізоване представленням `all_claims(request, status = 0)`. У представленні перевіряється, що авторизований користувач має менеджерський запис. Параметр `status` у випадку нульового значення визначає, що мають відтворюватися всі скарги, а всі інші значення вказують на необхідність відтворення тільки скарг, у яких не встановлена відповідь.

Відтворення резервувань за сховищем господаря реалізоване представленням `storage_reserves(request, storage)`, яке має параметр `storage`, що визначає адресу сховища. У представленні перевіряється, що у заданого сховища господарем є авторизований користувач.

Відтворення неопрацьованих господарем резервувань реалізоване представленням `undone_reserves(request)`. У представленні відбираються всі резервування, які стосуються сховищ, у яких господарем є авторизований користувач, а стан резервувань відповідає тільки створеному.

Відтворення всіх актуальних тарифів на зберігання речей у сховищі реалізоване представленням `storage_tariffs(request, storage)`, яке має параметр `storage`, що визначає адресу сховища, тарифи за яким відображаються. Здійснюється перевірка, що авторизований користувач є господарем цього сховища.

Відтворення можливих дисконтів за тарифом реалізоване представленням `tariff_discounts(request, tariff)`, що має параметр `tariff` з номером тарифу. Здійснюється перевірка, що авторизований користувач є господарем сховища, якого стосується тариф.

Відтворення резервувань зі спільного доступу реалізоване представленням `shared_reserves(request)`. Дані вибираються за авторизованим користувачем як орендарем.

Відтворення всіх доступних для зберігання речей орендарем резервувань реалізоване представленням `shared_reserves(request)`. Дані вибираються за авторизованим користувачем як орендарем. Вибираються всі резервування, які створив цей орендар або якими з ними поділилися інші орендарі.

Відтворення сховищ для детального розгляду реалізоване представленням `saved_storages(request)`. Дані вибираються за авторизованим користувачем як орендарем.

Відтворення сховищ для детального розгляду зі спільного доступу реалізоване представленням `shared_storages(request)`. Дані вибираються за авторизованим користувачем як орендарем.

Внесення сплати за наступний період резервування реалізоване представленням `payment(request, reserve, sum)`, яке має параметри: `reserve`, що визначає резервування, `sum`, що визначає суму, яку необхідно сплатити. Реалізація сплати підтримується через додаток Django PayPal.

Розділення доступу до резервування з іншим орендарем реалізоване представленням `save_reserve(request, reserve, tenantName)`, яке має параметри: `reserve`, що визначає резервування, доступ до якого необхідно розділити, `tenantName`, що визначає логін орендаря, для якого потрібно надати доступ до резервування. Представлення виконує перевірку, що таке розділення не відбулось раніше.

Внесення сховища для детального розгляду реалізоване представленням `save_storagelist(request, storage)`, яке має параметр `storage`, що визначає адресу сховища, яке необхідно зберегти. Представлення виконує перевірку, що таке збереження авторизованим користувачем за записом орендаря не відбулось раніше.

Надання спільного доступу до сховища для детального розгляду іншому орендарю реалізоване представленням `save_storage(request, storage, tenantName)`, яке має параметри: `storage`, що визначає адресу сховища, доступ до якого необхідно розділити, `tenantName`, що визначає логін орендаря, для якого потрібно надати доступ до сховища. Представлення виконує перевірку, що таке розділення не відбулось раніше.

Закриття доступу орендаря або господаря після аналізу скарг реалізоване представленням `cancel_user(request, username)`, яка має параметр `username`, що визначає логін користувача, якому менеджер вирішив закрити доступ, при цьому реалізується перевірка, що авторизований користувач має менеджерський доступ.

3.3 Висновки за розділом 3

Вебдодаток реалізовано повністю. Усі функціональні вимоги забезпечено створенням функцій представлень, менеджера URL, класів моделей, шаблонів вебсторінок.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Опис програми

Основним призначенням програми є забезпечення підтримки резервування для зберігання речей у орендованих орендарями сховищах, які надаються господарями на тимчасове використання.

4.2 Експлуатація програми

Експлуатація програми потребує від користувачів базових вмінь роботи з вебдодатками, адже всі функції представлені візуально та є загалом інтуїтивно зрозумілими. Після авторизації будь-який користувач може бути представлений в двох ролях: і орендаря, і господаря (рис. 4.1).

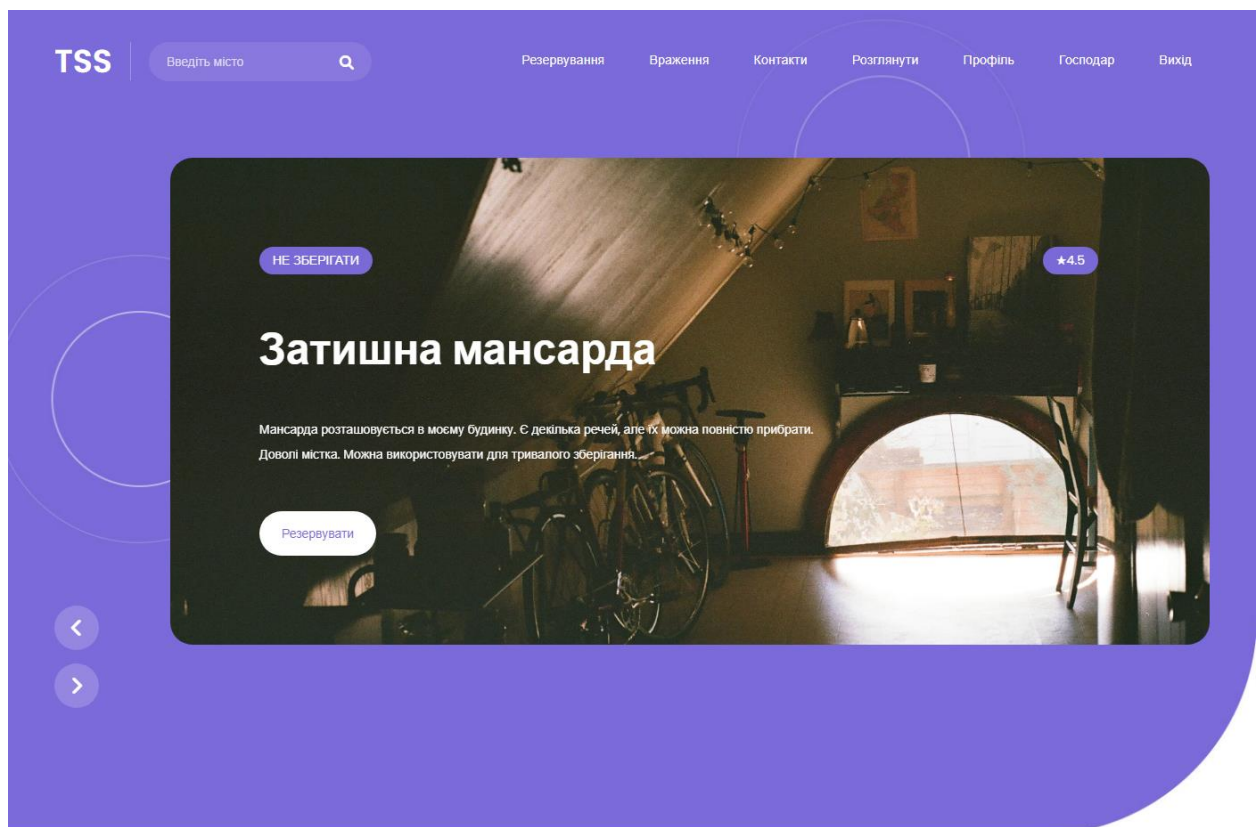


Рисунок 4.1 – Сторінка з оголошенням сховища

Для вибору режиму потрібно тільки перемкнутися між режимами, натискаючи на посилання «Господар» для переходу з режиму орендаря в режим господаря або «Орендар» для переходу з режиму господаря в режим орендаря. Таким чином користувач не потребує окремої авторизації і реєстрації, може легко перемикатися між режимами і бачити те, що зручно для відповідного режиму.

Для виконання пошуку виділено окреме поле, яке завжди відображається згори сторінки, де потрібно задати місце (місто або населений пункт), в якому потрібно виконати пошук. На сторінці з результатами можна задати розширені параметри пошуку, деталізуючи його, як це було описано в розділі 3.

Доступ до власних резервувань сховищ для збереження речей можна отримати за посиланням «Резервування» згори. Звідси можна також перейти до створення відгуку за сховищем з резервування. Переглянути свої відгуки (створені собою та про себе) можна за посиланням «Враження». На сторінці потрібно обрати, які саме відгуки цікавлять орендаря.

Посилання «Контакти» дозволяє переглянути розмови з господарями (для орендаря) або з орендарями (для господаря). Посилання «Розглянути» дозволяє виконати подальший аналіз і перейти до тих сховищ, які були раніше збережені. За кожним таким сховищем можна його вилучити, можна надати доступ до нього іншому користувачу, виконуючи його пошук за логіном. З цієї сторінки також можна перейти до сховищ, якими поділився з авторизованим орендарем інший орендар.

При відкритті окремого оголошення на його сторінці згори відображається фотографія сховища. Перейти до інших фотографій сховища можна за допомогою стрілок зліва від фотографії. Тут же виводиться назва, опис. Згори відображається позначка, чи збережено орендарем для подальшого детального аналізу це сховище, чи ні. Натискаючи на це посилання, можна зберегти незбережене сховища або вилучити збережене раніше.

Також виводиться оцінка і кнопка «Резервувати» для переходу до визначення параметрів резервування. Нижче розташована сама форма для визначення таких параметрів (рис. 4.2). В ній потрібно задати дати початку і закінчення резервування, тоді буде розраховано вартість такого резервування. Якщо орендар бажає, потрібно натиснути на кнопку «Резервувати» і перейти до сплати.

Зліва відображаються основні параметри сховища. Залежно від визначених параметрів господарем, вони можуть відрізнятися. Нижче розташована інформація про господаря. Щоб написати йому, потрібно натиснути на зображення листа. Щоб перейти до його сторінки, подивитись на відгуки про нього, потрібно натиснути на його ім'я. Відгуки про сховище розташовані нижче. Звідти можна перейти до сторінки зі всіма відгуками про сховище.

The screenshot displays the TSS website interface. At the top is a purple navigation bar with the TSS logo, a search bar labeled 'Введіть місто', and links for 'Резервування', 'Враження', 'Контакти', 'Розглянути', 'Профіль', 'Господар', and 'Вихід'.

The main content area is divided into two sections. On the left, a light purple box contains details about a storage unit:

- МІСЦЕ**: Запоріжжя
- ТИП**: Мансарда
- РОЗМІР ПРИБЛИЗНО**: 3x5 м
- ВИСОТА СТЕЛІ**: 1.9 м
- ЗАМОК**: встановлено
- ПРИВАТНИЙ ВХІД**: відсутній
- ДОДАТКОВО**: Кондиціонер, Електроенергія, Сповіщення про пожежу

 At the bottom of this box is a 'ГОСПОДАР' section with an email icon and the name 'Василь Макєснко', followed by a right-pointing arrow.

On the right, a dark purple box contains the reservation form:

- Хочу заїхати**: A date picker showing 30.05.2023.
- Хочу виїхати**: A date picker showing 30.07.2023.
- Вартість: 3000€**
- A white button labeled **Резервувати**.

Рисунок 4.2 – Опис сховища на сторінці оголошення

Потрапити на цю сторінку можна з різних розділів і сторінок, але спочатку це виконується через результати пошуку (рис. 4.3).

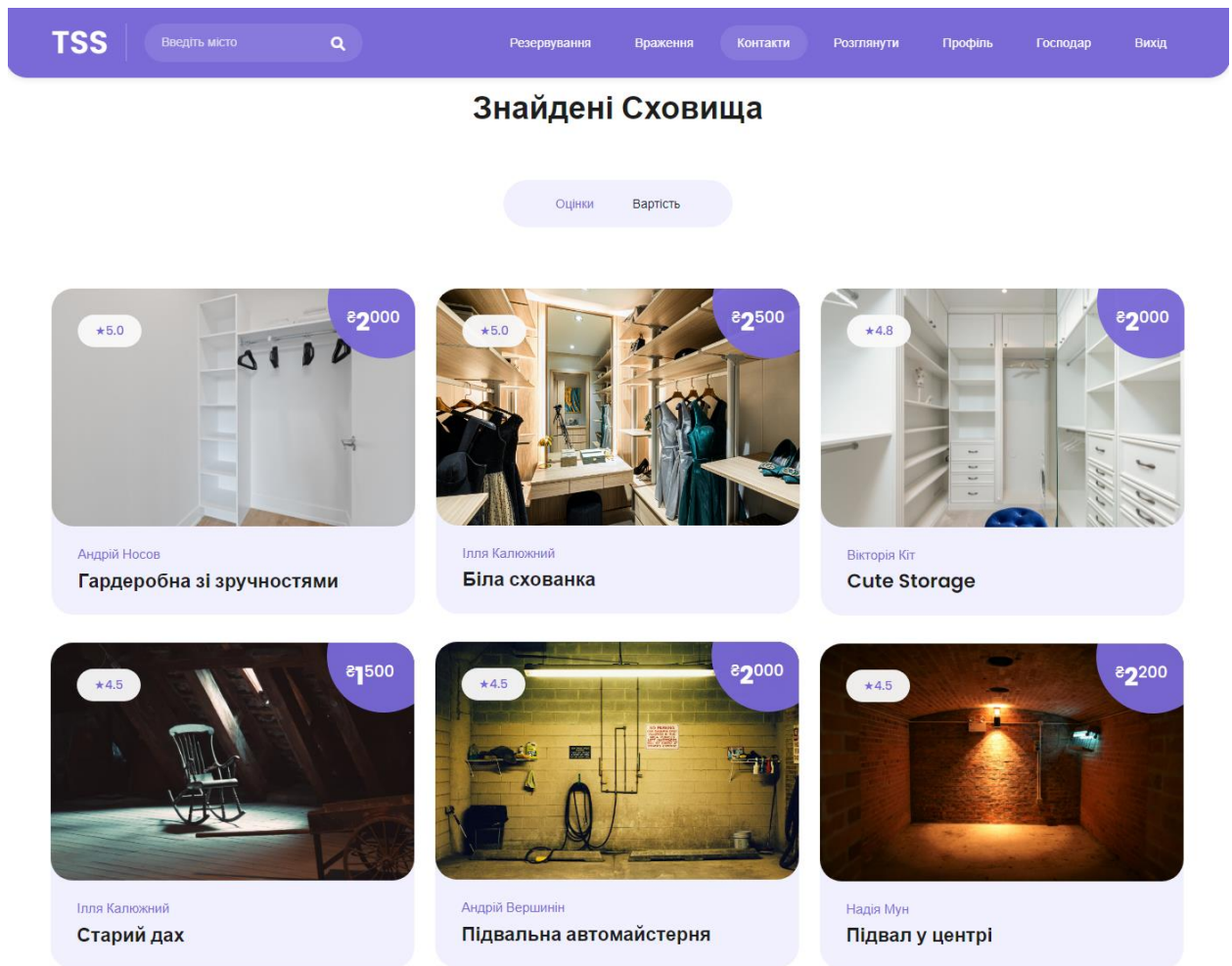


Рисунок 4.3 – Пошук сховищ

У результатах пошуку відображаються оголошення про сховища, які відповідають заданим параметрам, виводячи середню оцінку, фотографію сховища, вартість резервування за заданими датами, ім'я господаря, назву сховища (можна перейти до сторінки сховища звідти) та позначка про те, чи збережено вже для подальшого аналізу сховище. Якщо ні, то позначка не відображається. Для того щоб внести сховище в такий перелік, потрібно відкрити сторінку оголошення, а звідси натиснути на відповідне посилання.

Господар бачить трохи інші посилання згори. Одним з особливих посилань є «Сховища», звідки можна виставити нове сховище в оренду або

керувати вже виставленими, переходити до резервувань за конкретним сховищем, визначати тарифи, а від тарифів переходити до дисконтів.

Посилання «Резервування» відтворює неопрацьовані резервування за всіма сховищами господаря (рис. 4.4).

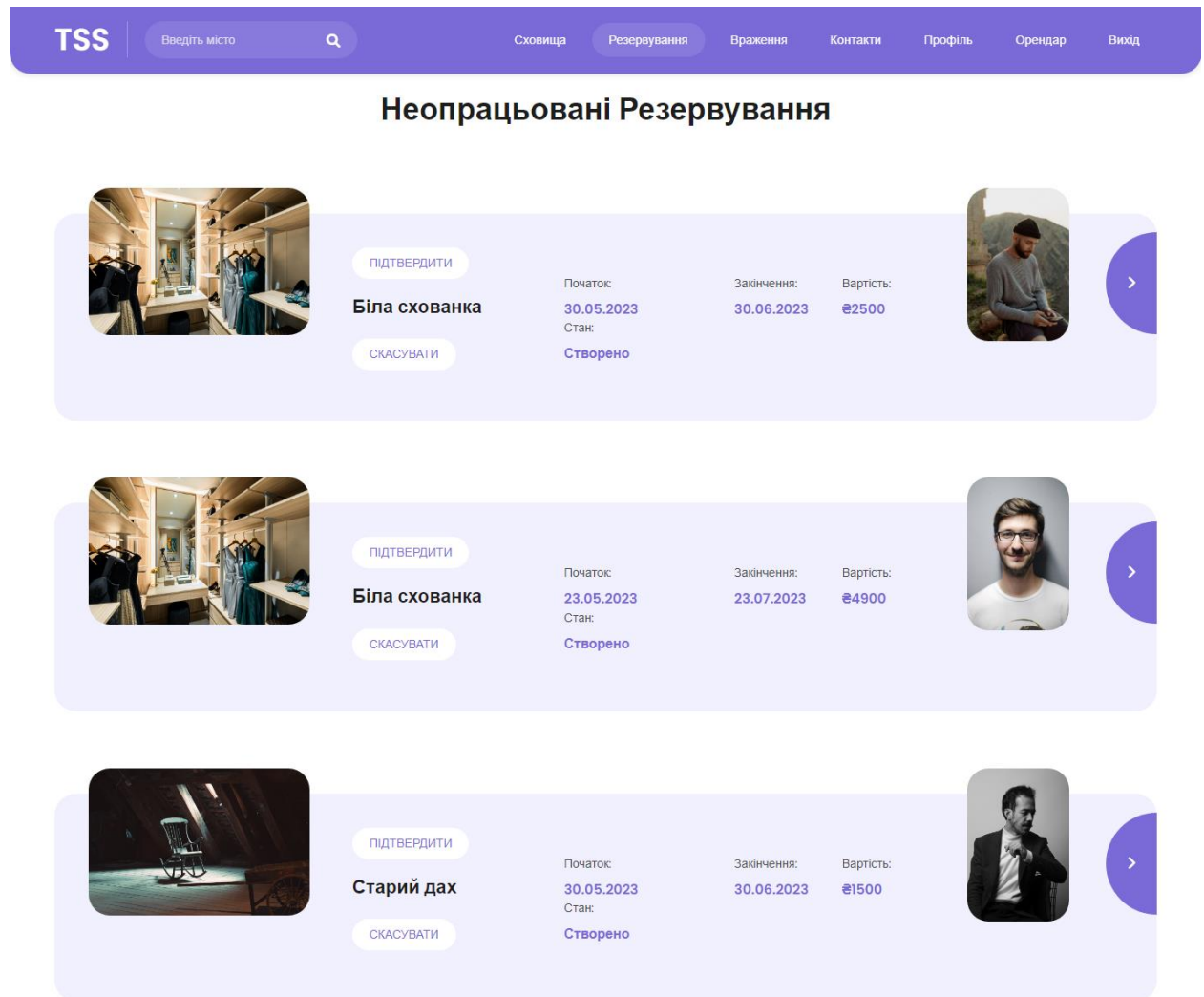


Рисунок 4.4 – Резервування за сховищем

За кожним резервуванням зліва виводиться фотографія сховища, справа – фотографія орендаря, який створив резервування. Спочатку за резервуванням виводяться посилання «Підтвердити» і «Скасувати». Після підтвердження це посилання зникає. Після початку строків підтвердженого резервування виникає посилання «Поскаржитись». Для підтвердженого резервування в подальшому відображаються також дати очікуваного

завершення та фактичного завершення. Відповідно змінюється і стан. Перейти до орендаря можна, натискаючи на стрілочку справа.

4.3 Тестування програми

Для тестування програми було створено чеклист, в який було включено всі сценарії, які за результатами моделювання програма мала надавати користувачам. Для кожного сценарію перевірялось, користувачу в якій саме ролі він має надаватися. Після цього виконувалось тестування конкретного сценарію для кожної такої ролі. Для кожного сценарію перевірялось, чи результат відповідає необхідному. Якщо він відповідав необхідному для кожної ролі, якій він має бути доступним, то в чеклист вносились позначка «+». За іншого результату мав вноситись «-». Тестування успішно завершено (табл. 4.1), програма реалізує необхідні сценарії.

Таблиця 4.1 – Чеклист тестування програми

Сценарій	Підсумок
1	2
Пошук сховища для збереження речей	+
Відтворення повних даних орендаря	+
Відтворення повних даних господаря	+
Залишення відгуку орендарем про сховище	+
Залишення відгуку господарем про орендаря	+
Виставлення власного сховища для резервування для зберігання речей	+
Змінювання даних про сховище для резервування	+
Відтворення всіх сховищ господаря	+
Резервування сховища господаря	+
Відтворення резервувань за сховищем господаря	+

Продовження таблиці 4.1

1	2
Встановлення тарифу на зберігання речей у сховищі	+
Відтворення неопрацьованих господарем резервувань	+
Підтвердження резервування	+
Відхилення резервування	+
Встановлення вільних дат для резервування сховища	+
Відтворення всіх актуальних тарифів на зберігання речей у сховищі	+
Змінювання тарифу на зберігання речей у сховищі	+
Встановлення дисконту за тарифом	+
Відтворення можливих дисконтів за тарифом	+
Вилучення тарифу на зберігання речей у сховищі	+
Вилучення дисконту за тарифом	+
Розширення резервування сховища для зберігання речей	+
Внесення сплати за наступний період резервування	+
Надання повних даних про сховище	+
Надсилення повідомлення орендарю	+
Надсилення повідомлення господарю	+
Відтворення розмови з орендарем	+
Відтворення розмови з господарем	+
Відтворення розпочатих розмов з господарями	+
Відтворення розпочатих розмов з орендарями	+
Розділення доступу до резервування з іншим орендарем	+
Відтворення резервувань зі спільного доступу	+
Відтворення всіх доступних для зберігання речей орендарем резервувань	+
Внесення сховища для детального розгляду	+
Відтворення сховищ для детального розгляду	+

Кінець таблиці 4.1

1	2
Надання відповіді на скаргу	+
Відтворення сховищ для детального розгляду зі спільного доступу	+
Вилучення сховища з детального розгляду	+
Відтворення всіх непроаналізованих скарг за резервуваннями	+
Надання спільного доступу до сховища для детального розгляду іншому орендарю	+
Вилучення спільного доступу до сховища для детального розгляду іншим орендарем	+
Закриття доступу орендаря або господаря після аналізу скарг	+
Змінювання власних даних	+
Надсилання скарги за резервуванням	+
Відтворення всіх скарг за резервуваннями	+
Відтворення всіх відгуків господаря	+
Відтворення всіх відгуків за сховищем	+
Відтворення всіх відгуків за всіма сховищами господаря	+
Відтворення всіх відгуків за орендарем	+
Відтворення всіх відгуків орендаря	+

4.4 Висновки за розділом 4

Роботу з програмою описано зі сторони користувачів, до яких входять господар і орендар та менеджер. Усі потрібні сценарії роботи забезпечені користувацькими інтерфейсами. Тестування реалізовано за кожним сценарієм і проведено успішно.

ВИСНОВКИ

На даний момент потреба в забезпеченні приміщень для тимчасового зберігання речей на основі пропозицій господарів цих приміщень в світі не задоволена. Існує тільки один програмний засіб, який на території США сприяє вирішенню цієї проблеми і активно розвивається. Тому мета роботи, яку було в результаті виконання роботи повністю виконано, полягала в розробці програмного забезпечення підтримки тимчасового зберігання речей для сприяння розширенню можливостей розміщення власних речей і оптимальному використанню власних приміщень.

Виконуючи проєктування програми, зроблено вибір на користь Python і Django для реалізації вебдодатку, що дозволяє спростити процес розробки і створити в результаті надійний інструмент для підтримки процесу тимчасового зберігання речей. Згідно з особливостями, відмінностями програми та базовими можливостями створено діаграми прецедентів для господаря, орендаря та менеджера. Проєктування роботи з даними призвело до створення схеми БД, що фактично у відповідності з архітектурою Django-додатків призвело до визначення структури моделей додатку.

Вебдодаток реалізовано повністю. Усі функціональні вимоги забезпечено створенням функцій представлень, менеджеру URL, класів моделей, шаблонів вебсторінок. Усі потрібні сценарії роботи забезпечені користувацькими інтерфейсами. Тестування реалізовано за кожним сценарієм і проведено успішно.

Створене програмне забезпечення дозволяє задовольнити потребу в тимчасовому зберігання речей різного роду, сприяючи розвитку цих засобів на території України, надаючи господарям приміщень і людям, які знаходяться в пошуку сховищ, знайти можливість для взаємодії та взаємовигідно задовольнити потреби один одного.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Neighbor Storage Review [Electronic resource] : Legit Airbnb for Storage or Logistical Nightmare? – Access mode : <https://moneydoneright.com/passive-income/other-ideas/neighbor-storage-review/>.
2. Neighbor. The Cheaper, Closer & Safer Storage Marketplace [Electronic resource]. – Access mode : <https://www.neighbor.com/>.
3. Neighbor – Self Storage – Apps on Google Play [Electronic resource]. – Access mode : <https://play.google.com/store/apps/details?id=com.neighbor.js&hl=en&gl=US>.
4. Olick, D. Neighbor is out to disrupt self-storage [Electronic resource] : It's like Airbnb for your stuff / D. Olick. – Access mode : <https://www.cnbc.com/2019/08/20/neighborcom-is-out-to-disrupt-self-storage-its-like-airbnb-for-stuff.html>.
5. Our Documentation. Python.org [Electronic resource]. – Access mode : <https://www.python.org/doc/>.
6. Bernard, J. Python Recipes Handbook [Text] : A Problem-Solution Approach / J. Bernard. – New York : Apress, 2016. – 155 p.
7. The web framework for perfectionists with deadlines. Django [Electronic resource]. – Access mode : <https://www.djangoproject.com/>.
8. Percival, H. Test-Driven Development with Python [Text] : Obey the Testing Goat : Using Django, Selenium, and JavaScript / H. Percival. – Sebastopol, USA : O'Reilly Media, 2017. – 624 p.
9. Forcier, J. Python Web Development with Django [Text] / J. Forcier. – Boston : Addison-Wesley Professional, 2008. – 402 p.
10. Bendoraitis, A. Django 3 Web Development Cookbook [Text] : Actionable solutions to common problems in Python web development / A. Bendoraitis, J. Kronika. – Birmingham : Packt Publishing, 2020. – 608 p.
11. Pettit, T. The MySQL Workshop [Text] : A practical guide to working with data and managing databases with MySQL / T. Pettit, S. Cosentino. –

Birmingham : Packt Publishing, 2022. – 726 p.

12. Topic Listing Template [Electronic resource]. – Access mode :
<https://templatemo.com/tm-590-topic-listing>.

ДОДАТОК А
Технічне завдання

Вступ

Загальновідомою є проблема оренди житла. Однак ця проблема має декілька фактично супутніх.

Однією з супутніх проблем є оренда ділянки для паркування автомобіля. Другою є оренда місця для зберігання речей. Наприклад, орендуючи житло, людина може мати необхідність розмістити певні власні речі, оскільки вони не вміщуються в орендоване місце проживання. Ця ж потреба може виникати і у тих, хто має власне житло. Іноді для розміщення автомобіля потрібен гараж. Це також фактично оренда сховища.

Часом людина може переїхати в інше місто або поїхати на відпочинок, при цьому не орендуючи житло в тому місті, звідки поїхала, а потребуючи в ньому зберігати речі. Це може бути не відпочинок, а робота за кордоном. Окрім того, коли людина приїжджає в інше місто, там в неї може виникати потреба залишити речі на певний період, наприклад, у випадку тривалої подорожі з різними пунктами в маршруті, щоб повернутися потім до місця зберігання речей. Цей перелік випадків можна продовжувати, але загалом це визначає, що потреба зберігання речей в винайманні для цього певного сховища є досить актуальною і потребує відповідного програмного забезпечення.

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», за яким було визначено тему «Програмне забезпечення підтримки тимчасового зберігання речей».

А.2 Призначення розробки

Основним призначенням розробки є забезпечення підтримки резервування для зберігання речей у орендованих орендарями сховищах, які надаються господарями на тимчасове використання.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Забезпечення призначення програми виконується через реалізацію сценаріїв використання програми на основі наступних функцій:

- пошук сховища для збереження речей;
- відтворення всіх відгуків за сховищем;
- відтворення всіх відгуків господаря;
- відтворення всіх відгуків орендаря;
- відтворення повних даних господаря;
- відтворення всіх відгуків за господарем;
- відтворення повних даних орендаря;
- відтворення всіх відгуків за орендарем;
- залишення відгуку орендарем про сховище;
- залишення відгуку господарем про орендаря;
- виставлення власного сховища для резервування для зберігання речей;
- змінювання даних про сховище для резервування;
- відтворення всіх сховищ господаря;
- резервування сховища господаря;
- відтворення резервувань за сховищем господаря;
- відтворення неопрацьованих господарем резервувань;
- підтвердження резервування або його відхилення;
- встановлення тарифу на зберігання речей у сховищі;

- відтворення всіх актуальних тарифів на зберігання речей у сховищі;
- змінювання тарифу на зберігання речей у сховищі;
- встановлення дисконту за тарифом;
- відтворення можливих дисконтів за тарифом;
- вилучення тарифу на зберігання речей у сховищі;
- вилучення дисконту за тарифом;
- встановлення вільних дат для резервування сховища;
- розширення резервування сховища для зберігання речей;
- внесення сплати за наступний період резервування;
- надання повних даних про сховище;
- надсилання повідомлення від господаря орендарю або навпаки;
- відтворення розмови господаря з орендарем або навпаки;
- відтворення розпочатих розмов між орендарем та власниками або навпаки;
- розділення доступу до резервування з іншим орендарем;
- відтворення резервувань зі спільного доступу;
- відтворення всіх доступних для зберігання речей орендарем резервувань;
- внесення сховища для детального розгляду;
- відтворення сховищ для детального розгляду, визначених орендарем та зі спільного доступу;
- вилучення сховища з детального розгляду;
- надання спільного доступу до сховища для детального розгляду іншому орендарю;
- вилучення спільного доступу до сховища для детального розгляду іншим орендарем;
- закриття доступу орендаря або господаря після аналізу скарг;
- надсилання скарги за резервуванням;
- відтворення всіх скарг за резервуваннями;
- надання відповіді на скаргу;

– відтворення всіх непроаналізованих скарг за резервуваннями.

A.3.2 Умови експлуатації

Експлуатація програми на сервері потребує підтримки, що включає навички роботи з БД, зокрема MySQL, виконання міграцій для Django-додатків та внесення даних через адміністративний інтерфейс при необхідності розширення або зміни місць, типів розмірів сховищ або інших каталогів стандартних даних.

A.3.3 Вимоги до складу та параметрів технічних засобів

Набір мінімальних технічних засобів, що мають забезпечуватися на сервері для підтримки роботи вебдодатку, включає:

- процесор не менше ніж Xeon сімейства 3000 або Core 2 Duo;
- оперативна пам'ять обсягом у 4 Гб;
- простір на жорсткому диску з доступним вільним обсягом не менше ніж 3 Гб, але для можливості збереження файлів зображень бажано не менше 20 Гб;
- мережна карта і підключення до мереж Інтернет.

A.4 Порядок контролю та приймання

Для виконання дипломної кваліфікаційної роботи критичним є виконання визначеного завдання, що включає безпосередньо завдання на роботу і технічне завдання, за темою, яка зафіксована наказом, у строки, які визначені завданням на роботу.

У підсумку виконується захист роботи за оприлюдненим наперед графіком перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

```

        from .models import Tenant, StorageReview, TenantReview,
        User, Storage, Host, Reserve, Place, Sizetype, Freedate, Tariff,
        Discount, Storagelist
        from .forms import StorageReviewForm, TenantReviewForm,
        ContactForm
        from django.shortcuts import redirect, render
        from django.db.models import Sum
        import datetime

    def tenant_reviews(request, tenantUsername = None, star =
0):
        reviews_of_tenant = []
        if tenantUsername:
            tenant_user = User.objects.get(username =
tenantUsername)
        else:
            tenant_user = request.user
        if tenant_user:
            tenant_object = Tenant.objects.get(user_id =
tenant_user.id)
            reviews_of_tenant =
TenantReview.objects.filter(tenant = tenant_object)
            if star != 0:
                star_int = int(star)
                if star_int > 0 and star_int <= 5:
                    reviews_of_tenant =
reviews_of_tenant.filter(star = star_int)
            reviews_of_tenant = reviews_of_tenant.order_by("-
creation_date")
            return render(request,
"tempstoragesoft/reviews_tenant.html", {"tenant": tenant_object,
"reviews": reviews_of_tenant })
        else:
            return render(request,
"tempstoragesoft/reviews_tenant.html", {"tenant": None,
"reviews": [] })

    def by_tenant_reviews(request, tenantUsername = None, star
= 0):
        reviews_by_tenant = []
        if tenantUsername:
            tenant_user = User.objects.get(username =
tenantUsername)
        else:
            tenant_user = request.user
        if tenant_user:
            tenant_object = Tenant.objects.get(user_id =
tenant_user.id)

```

```

        reviews_by_tenant =
StorageReview.objects.filter(tenant = tenant_object)
        if star != 0:
            star_int = int(star)
            if star_int > 0 and star_int <= 5:
                reviews_by_tenant =
reviews_by_tenant.filter(star = star_int)
                reviews_by_tenant = reviews_by_tenant.order_by("-
creation_date")
            return render(request,
"tempstoragesoft/reviews_by_tenant.html", {"tenant":
tenant_object, "reviews": reviews_by_tenant })
        else:
            return render(request,
"tempstoragesoft/reviews_by_tenant.html", {"tenant": None,
"reviews": [] })

def storage_reviews(request, storageSlug, star = 0):
    storage_reviews = []
    if storageSlug:
        storage_object = Storage.objects.get(slug =
storageSlug)
    else:
        return render(request,
"tempstoragesoft/reviews_storage.html", {"storage": None,
"reviews": [] })
    if storage_object:
        storage_reviews =
StorageReview.objects.filter(storage = storage_object)
        if star != 0:
            star_int = int(star)
            if star_int > 0 and star_int <= 5:
                storage_reviews =
storage_reviews.filter(star = star_int)
                storage_reviews = storage_reviews.order_by("-
creation_date")
            return render(request,
"tempstoragesoft/reviews_storage.html", {"storage":
storage_object, "reviews": storage_reviews })
        else:
            return render(request,
"tempstoragesoft/reviews_storage.html", {"storage": None,
"reviews": [] })

def host_reviews(request, hostUsername = None, star = 0):
    host_reviews = []
    if hostUsername:

```

```

        host_user = User.objects.get(username =
hostUsername)
    else:
        host_user = request.user
    if host_user:
        host_object = Host.objects.get(user_id =
host_user.id)
        host_reviews =
StorageReview.objects.filter(storage__host = host_object)
        if star != 0:
            star_int = int(star)
            if star_int > 0 and star_int <= 5:
                host_reviews = host_reviews.filter(star =
star_int)
            host_reviews = host_reviews.order_by("-
creation_date")
        return render(request,
"tempstoragesoft/reviews_host.html", {"host": host_object,
"reviews": host_reviews })
    else:
        return render(request,
"tempstoragesoft/reviews_host.html", {"host": None, "reviews":
[] })

def by_host_reviews(request, hostUsername = None, star =
0):
    reviews_by_host = []
    if hostUsername:
        host_user = User.objects.get(username =
hostUsername)
    else:
        host_user = request.user
    if host_user:
        host_object = Host.objects.get(user_id =
host_user.id)
        reviews_by_host = TenantReview.objects.filter(host
= host_object)
        if star != 0:
            star_int = int(star)
            if star_int > 0 and star_int <= 5:
                reviews_by_host =
reviews_by_host.filter(star = star_int)
            reviews_by_host = reviews_by_host.order_by("-
creation_date")
        return render(request,
"tempstoragesoft/reviews_by_host.html", {"host": host_object,
"reviews": reviews_by_host })
    else:

```

```

        return render(request,
"tempstoragesoft/reviews_by_host.html", {"host": None,
"reviews": [] })

    def tenant_info(request, tenantUsername = None):
        if tenantUsername:
            tenant_user = User.objects.get(username =
tenantUsername)
        else:
            tenant_user = request.user
        if tenant_user:
            tenant_object = Tenant.objects.get(user_id =
tenant_user.id)
            reviews_of_tenant =
TenantReview.objects.filter(tenant = tenant_object)
            reviews_of_tenant = reviews_of_tenant.order_by("-
creation_date")[:5]
            reviews_by_tenant =
StorageReview.objects.filter(tenant = tenant_object)
            reviews_by_tenant = reviews_by_tenant.order_by("-
creation_date")[:5]
            return render(request,
"tempstoragesoft/tenant.html", {"tenant": tenant_object,
"reviews": reviews_of_tenant, "reviews_by": reviews_by_tenant })
        else:
            return render(request,
"tempstoragesoft/tenant.html", {"tenant": None, "reviews": [],
"reviews_by": [] })

    def host_info(request, hostUsername = None):
        if hostUsername:
            host_user = User.objects.get(username =
hostUsername)
        else:
            host_user = request.user
        if host_user:
            host_object = Host.objects.get(user_id =
host_user.id)
            reviews_by_host = TenantReview.objects.filter(host
= host_object)
            reviews_by_host = reviews_by_host.order_by("-
creation_date")[:5]
            reviews_of_host =
StorageReview.objects.filter(storage__host = host_object)
            reviews_of_host = reviews_of_host.order_by("-
creation_date")[:5]
            host_storages = Storage.objects.filter(host =
host_object)

```

```

        host_storages = host_storages.order_by("-id")[:3]
        return render(request, "tempstoragesoft/host.html",
{"host": host_object, "reviews": reviews_of_host, "reviews_by":
reviews_by_host, "storages": host_storages })
    else:
        return render(request, "tempstoragesoft/host.html",
{"host": None, "reviews": [], "reviews_by": [], "storages": []
})

def host_storages(request, hostUsername = None, role = 0):
    if role == 0:
        actual_user = request.user
        actual_object = Tenant.objects.get(user_id =
actual_user.id)
        if hostUsername:
            host_user = User.objects.get(username =
hostUsername)
        else:
            host_user = request.user
        if host_user:
            host_object = Host.objects.get(user_id =
host_user.id)
            host_storages = Storage.objects.filter(host =
host_object)
            host_storages = host_storages.order_by("-id")
            if role == 0:
                for storage in host_storages:
                    storages =
Storagelist.objects.filter(tenant = actual_object, storage =
storage)

                    if storages:
                        storage.saved = True
                    else:
                        storage.saved = False
            return render(request,
"tempstoragesoft/host_storages.html", {"host": host_object,
"storages": host_storages })
        else:
            return render(request, "tempstoragesoft/host.html",
{"host": None, "storages": host_storages })

def review(request, object = None, role = 0):
    if request.user.is_active and role == 0:
        tenant_user = request.user
        if tenant_user:
            tenant_object = Tenant.objects.get(user_id =
tenant_user.id)
        else:

```



```

        marks =
StorageReview.objects.filter(storage =
storage_review.storage).aggregate(Sum("mark"))
        number =
StorageReview.objects.filter(storage =
storage_review.storage).count()
        storage = storage_review.storage
        storage.star = marks / number
        storage.save()
        redirect("storagereviews",
storage_review.storage.slug)
    else:
        storage_review_form = StorageReviewForm()
        return render(request,
"tempstoragesoft/review.html", {"form": storage_review_form,
"object": storage_object })
    elif role == 1:
        if request.method == "POST":
            tenant_review_form =
TenantReviewForm(request.POST)
            if tenant_review_form.is_valid():
                tenant_review =
tenant_review_form.save(commit=False)
                tenant_review.host = host_object
                tenant_review.save()
                marks = TenantReview.objects.filter(tenant
= tenant_review.tenant).aggregate(Sum("mark"))
                number = TenantReview.objects.filter(tenant
= tenant_review.tenant).count()
                tenant = tenant_review.storage
                tenant.star = marks / number
                tenant.save()
                id = tenant_review.tenant.user_id
                username = User.objects.get(id = id)
                redirect("tenantreviews", username)
            else:
                tenant_review_form = TenantReviewForm()
                return render(request,
"tempstoragesoft/review.html", {"form": tenant_review_form,
"object": tenant_object })
        else:
            return render(request,
"tempstoragesoft/review.html", {"form": None, "object": None })

def find_storages(request):
    storage_type = self.request.GET.get('storage', None)
    storages = None
    if storage_type:

```

```

        variants = storage_type.split(',')
        storages = Storage.objects.filter(storage_type__in
= variants)
    vehicle = self.request.GET.get('vehicle', None)
    if vehicle and storages:
        storages = storages.filter(vehicle = True)
    elif vehicle:
        storages = Storage.objects.filter(vehicle = True)
    inner = self.request.GET.get('inner', None)
    if inner and storages:
        storages = storages.filter(inner = True)
    elif inner:
        storages = Storage.objects.filter(inner = True)
    place = self.request.GET.get('city', None)
    if place:
        place = Place.objects.get(name = place)
        if storages:
            storages = storages.filter(place = place)
        else:
            storages = Storage.objects.filter(place =
place)
    size_type = self.request.GET.get('size', None)
    if size_type:
        size = Sizetype.objects.get(type_name = size_type)
        if size:
            size_length = size.length_size
            width_size = size.width_size
            if storages:
                storages =
storages.filter(size_type__length_size__gte = size_length,
size_type__width_size__gte = width_size)
            else:
                storages =
Storage.objects.filter(size_type__length_size__gte =
size_length, size_type__width_size__gte = width_size)
        entrance = self.request.GET.get('entr', None)
        if entrance and storages:
            storages = storages.filter(private_entrance = True)
        elif inner:
            storages = Storage.objects.filter(private_entrance
= True)
        closed = self.request.GET.get('territor', None)
        if closed and storages:
            storages = storages.filter(closed_territory__gte =
closed)
        elif storages:
            storages =
Storage.objects.filter(closed_territory__gte = closed)

```

```

stairs = self.request.GET.get('stairs', None)
if stairs and storages:
    storages = storages.filter(stairs = False)
elif stairs:
    storages = Storage.objects.filter(stairs = False)
smoking = self.request.GET.get('smoke', None)
if smoking and storages:
    storages = storages.filter(no_smoking = True)
elif smoking:
    storages = Storage.objects.filter(no_smoking =
True)

conditioner = self.request.GET.get('condit', None)
if conditioner and storages:
    storages = storages.filter(conditioner = True)
elif conditioner:
    storages = Storage.objects.filter(conditioner =
True)

refrigerator = self.request.GET.get('refriger', None)
if refrigerator and storages:
    storages = storages.filter(refrigerator = True)
elif refrigerator:
    storages = Storage.objects.filter(refrigerator =
True)

freezer = self.request.GET.get('freeze', None)
if freezer and storages:
    storages = storages.filter(freezer = True)
elif freezer:
    storages = Storage.objects.filter(freezer = True)
electricity = self.request.GET.get('electricity', None)
if electricity and storages:
    storages = storages.filter(electricity = True)
elif electricity:
    storages = Storage.objects.filter(electricity =
True)

fire_detector = self.request.GET.get('fire', None)
if fire_detector and storages:
    storages = storages.filter(fire_detector = True)
elif fire_detector:
    storages = Storage.objects.filter(fire_detector =
True)

alarm = self.request.GET.get('alarm', None)
if alarm and storages:
    storages = storages.filter(alarm = True)
elif alarm:
    storages = Storage.objects.filter(alarm = True)
locked = self.request.GET.get('lock', None)
if locked and storages:
    storages = storages.filter(locked = True)

```

```

elif locked:
    storages = Storage.objects.filter(locked = True)
pet = self.request.GET.get('pets', None)
if pet and storages:
    storages = storages.filter(no_pet = True)
elif pet:
    storages = Storage.objects.filter(no_pet = True)
shelves = self.request.GET.get('shelve', None)
if shelves and storages:
    storages = storages.filter(shelves = True)
elif shelves:
    storages = Storage.objects.filter(shelves = True)
hangers = self.request.GET.get('hangers', None)
if hangers and storages:
    storages = storages.filter(hangers = True)
elif hangers:
    storages = Storage.objects.filter(hangers = True)
light = self.request.GET.get('light', None)
if light and storages:
    storages = storages.filter(light = True)
elif light:
    storages = Storage.objects.filter(light = True)
cctv = self.request.GET.get('cctv', None)
if cctv and storages:
    storages = storages.filter(cctv = True)
elif cctv:
    storages = Storage.objects.filter(cctv = True)
registration = self.request.GET.get('reg', None)
if registration and storages:
    storages = storages.filter(registration = False)
elif registration:
    storages = Storage.objects.filter(registration =
False)

driveway = self.request.GET.get('way', None)
if driveway and storages:
    storages = storages.filter(driveway = True)
elif driveway:
    storages = Storage.objects.filter(driveway = True)
star_from = self.request.GET.get('star', None)
if star_from and storages:
    storages = storages.filter(star__gte = star_from)
elif star_from:
    storages = Storage.objects.filter(star__gte =
star_from)
elif storages:
    storages = storages.filter(star__gte = 0)
else:
    storages = Storage.objects.filter(star__gte = 0)

```

```

star_to = self.request.GET.get('star_to', None)
if star_to:
    storages = storages.filter(star__lte = star_to)
access_type = self.request.GET.get('access', None)
if access_type:
    storages = storages.filter(access_type__gte =
access_type)
access_from = self.request.GET.get('from', 10)
access_to = self.request.GET.get('to', 22)
if access_from and access_to:
    access_from = int(access_from)
    access_to = int(access_to)
    storages = storages.filter(access_start__lte =
access_from, access_end__gte = access_to)
    costs_from = self.request.GET.get('money', 0)
    costs_to = self.request.GET.get('top', None)
    reserve_from = self.request.GET.get('begin', None)
    reserve_from_components = reserve_from.split('.')
    reserve_from =
datetime.datetime(int(reserve_from_components[2]),
int(reserve_from_components[1]),
int(reserve_from_components[0]))
    reserve_to = self.request.GET.get('finish', None)
    reserve_to_components = reserve_to.split('.')
    reserve_to =
datetime.datetime(int(reserve_to_components[2]),
int(reserve_to_components[1]), int(reserve_to_components[0]))
    conditioned_storages = []
    storages = storages.order_by('-star')
    for storage in storages:
        free = Freedate.objects.filter(storage = storage,
start_date__lte = reserve_from, end_date__gte = reserve_to)
        if free:
            no_reservations = True
            reserves = Reserve.objects.filter(storage =
storage, start_date__lte = reserve_from, waited_end_date__gte =
reserve_from, status__gte = 2)
            if not reserves:
                reserves = Reserve.objects.filter(storage =
storage, start_date__gte = reserve_from, start_date__lte =
reserve_to, status__gte = 2)
            if reserves:
                no_reservations = False
        else:
            no_reservations = False
    if no_reservations:
        interval = reserve_to - reserve_from
        moment = datetime.now()

```

```

        tariffs = Tariff.objects.filter(storage =
storage, unit = 1, min_duration__gte = interval.days,
end_date__gte = moment)
        min_cost = None
        for tariff in tariffs:
            calculates_cost = 0
            if tatiff.start_date <= reserve_from
and tariff.end_date >= reserve_from and tariff.end_date <=
reserve_to:
                days = reserve_to -
tatiff.start_date
                days = hours.days
                calculates_cost = calculates_cost +
days * tariff.cost
            start_tariff =
tariffs.filter(start_date__lte = reserve_from, end_date__gte =
tatiff.start_date).order_by('cost')[0]
            days = start_tariff.end_date -
reserve_from
            days = days.days
            calculates_cost = calculates_cost +
hours * start_tariff.cost
            if not min_cost:
                min_cost = calculates_cost
            elif calculates_cost < min_cost:
                min_cost = calculates_cost
            calculates_cost = 0
            if tatiff.start_date >= reserve_from
and tariff.end_date >= reserve_to:
                days = tariff.end_date - reserve_to
                days = days.days
                calculates_cost = calculates_cost +
days * tariff.cost
            end_tariff =
tariffs.filter(start_date__lte = tariff.end_date, end_date__gte
= reserve_to).order_by('cost')[0]
            days = reserve_to - tariff.end_date
            days = days.days
            calculates_cost = calculates_cost +
days * end_tariff.cost
            if calculates_cost < min_cost:
                min_cost = calculates_cost
            calculates_cost = 0
            if tatiff.start_date <= reserve_from
and tariff.end_date >= reserve_to:
                days = reserve_to - reserve_from
                days = days.days

```

```

calculates_cost = calculates_cost +
days * tariff.cost

    if calculates_cost < min_cost:
        min_cost = calculates_cost
    tariffs = Tariff.objects.filter(storage =
storage, unit = 0, min_duration__gte = interval.months,
end_date__gte = moment)
    min_cost = None
    for tariff in tariffs:
        calculates_cost = 0
        if tatiff.start_date <= reserve_from
and tariff.end_date >= reserve_from and tariff.end_date <=
reserve_to:
            months = reserve_to -
tatiff.start_date
            months = months.months
            calculates_cost = calculates_cost +
months * tariff.cost
            start_tariff =
tariffs.filter(start_date__lte = reserve_from, end_date__gte =
tatiff.start_date).order_by('cost')[0]
            months = start_tariff.end_date -
reserve_from
            months = months.months
            calculates_cost = calculates_cost +
months * start_tariff.cost
            discounts =
Discount.objects.filter(tariff = tariff, duration__gte =
interval.months)
            max_discount = None
            for disc in discounts:
                dis = disc.start_type
                if dis:
                    dis = tariff.cost *
disc.percentage
                else:
                    dis = calculates_cost *
disc.percentage
            if (max_discount and dis >
max_discount) or (not max_discount):
                max_discount = dis
            if max_discount:
                calculates_cost =
calculates_cost - max_discount
            if not min_cost:
                min_cost = calculates_cost
            elif calculates_cost < min_cost:
                min_cost = calculates_cost

```

```

        calculates_cost = 0
        if tatiff.start_date >= reserve_from
and tariff.end_date >= reserve_to:
            months = tariff.end_date -
reserve_to

            months = hours.months
            calculates_cost = calculates_cost +
months * tariff.cost

            end_tariff =
tariffs.filter(start_date__lte = tariff.end_date, end_date__gte
= reserve_to).order_by('cost')[0]
            months = reserve_to -
tariff.end_date

            months = months.months
            calculates_cost = calculates_cost +
months * end_tariff.cost

            discounts =
Discount.objects.filter(tariff = end_tariff, duration__gte =
interval.months)

            max_discount = None
            for disc in discounts:
                dis = disc.start_type
                if dis:
                    dis = end_tariff.cost *
disc.percentage

                else:
                    dis = calculates_cost *
disc.percentage

                if (max_discount and dis >
max_discount) or (not max_discount):
                    max_discount = dis
            if max_discount:
                calculates_cost =
calculates_cost - max_discount

                if calculates_cost < min_cost:
                    min_cost = calculates_cost
                calculates_cost = 0
                if tatiff.start_date <= reserve_from
and tariff.end_date >= reserve_to:
                    months = reserve_to - reserve_from
                    months = months.months
                    calculates_cost = calculates_cost +
months * tariff.cost

                    discounts =
Discount.objects.filter(tariff = tariff, duration__gte =
interval.months)

                    max_discount = None
                    for disc in discounts:

```

```

        dis = disc.start_type
        if dis:
            dis = tariff.cost *

disc.percentage

        else:
            dis = calculates_cost *

disc.percentage

        if (max_discount and dis >
max_discount) or (not max_discount):
            max_discount = dis
            if max_discount:
                calculates_cost =
calculates_cost - max_discount
            if calculates_cost < min_cost:
                min_cost = calculates_cost
            storage.cost = min_cost
            if min_cost >= costs_from:
                if (costs_to and min_cost <=
int(costs_to)) or (not costs_to):

conditioned_storages.append(storage)
        role = self.request.GET.get('role', None)
        if role:
            actual_user = request.user
            actual_object = Tenant.objects.get(user_id =
actual_user.id)
            for storage in conditioned_storages:
                storages =
StorageList.objects.filter(tenant = actual_object, storage =
storage)

                if storages:
                    storage.saved = True
                else:
                    storage.saved = False
            return render(request, "tempstoragesoft/find.html",
{"storages": conditioned_storages})

    def storage(request, storage, reserve_from = None,
reserve_to = None, role = 0):
        storage = Storage.objects.get(slug = storage, None)
        reserve_from_components = reserve_from.split('.')
        reserve_from =
datetime.datetime(int(reserve_from_components[2]),
int(reserve_from_components[1]),
int(reserve_from_components[0]))
        reserve_to_components = reserve_to.split('.')

```

```

        reserve_to =
datetime.datetime(int(reserve_to_components[2]),
int(reserve_to_components[1]), int(reserve_to_components[0]))
        if not storage_object:
            return render(request,
"tempstoragesoft/storage.html", {"storage": None})
            storage.cost = -1
            free = Freedate.objects.filter(storage = storage,
start_date__lte = reserve_from, end_date__gte = reserve_to)
            if free:
                no_reservations = True
                reserves = Reserve.objects.filter(storage =
storage, start_date__lte = reserve_from, waited_end_date__gte =
reserve_from, status__gte = 2)
                if not reserves:
                    reserves = Reserve.objects.filter(storage =
storage, start_date__gte = reserve_from, start_date__lte =
reserve_to, status__gte = 2)
                    if reserves:
                        no_reservations = False
                else:
                    no_reservations = False
            if no_reservations:
                interval = reserve_to - reserve_from
                moment = datetime.now()
                tariffs = Tariff.objects.filter(storage =
storage, unit = 1, min_duration__gte = interval.days,
end_date__gte = moment)
                min_cost = None
                for tariff in tariffs:
                    calculates_cost = 0
                    if tatiff.start_date <= reserve_from and
tariff.end_date >= reserve_from and tariff.end_date <=
reserve_to:
                        days = reserve_to - tatiff.start_date
                        days = hours.days
                        calculates_cost = calculates_cost +
days * tariff.cost
                        start_tariff =
tariffs.filter(start_date__lte = reserve_from, end_date__gte =
tatiff.start_date).order_by('cost')[0]
                        days = start_tariff.end_date -
reserve_from
                        days = days.days
                        calculates_cost = calculates_cost +
hours * start_tariff.cost
                    if not min_cost:
                        min_cost = calculates_cost

```

```

        elif calculates_cost < min_cost:
            min_cost = calculates_cost
        calculates_cost = 0
        if tatiff.start_date >= reserve_from and
tariff.end_date >= reserve_to:
            days = tariff.end_date - reserve_to
            days = days.days
            calculates_cost = calculates_cost +
days * tariff.cost
            end_tariff =
tariffs.filter(start_date__lte = tariff.end_date, end_date__gte
= reserve_to).order_by('cost')[0]
            days = reserve_to - tariff.end_date
            days = days.days
            calculates_cost = calculates_cost +
days * end_tariff.cost
            if calculates_cost < min_cost:
                min_cost = calculates_cost
            calculates_cost = 0
            if tatiff.start_date <= reserve_from and
tariff.end_date >= reserve_to:
                days = reserve_to - reserve_from
                days = days.days
                calculates_cost = calculates_cost +
days * tariff.cost
            if calculates_cost < min_cost:
                min_cost = calculates_cost
            tariffs = Tariff.objects.filter(storage =
storage, unit = 0, min_duration__gte = interval.months,
end_date__gte = moment)
            min_cost = None
            for tariff in tariffs:
                calculates_cost = 0
                if tatiff.start_date <= reserve_from and
tariff.end_date >= reserve_from and tariff.end_date <=
reserve_to:
                    months = reserve_to - tatiff.start_date
                    months = months.months
                    calculates_cost = calculates_cost +
months * tariff.cost
                    start_tariff =
tariffs.filter(start_date__lte = reserve_from, end_date__gte =
tatiff.start_date).order_by('cost')[0]
                    months = start_tariff.end_date -
reserve_from
                    months = months.months
                    calculates_cost = calculates_cost +
months * start_tariff.cost

```

```

        discounts =
Discount.objects.filter(tariff = tariff, duration__gte =
interval.months)
        max_discount = None
        for disc in discounts:
            dis = disc.start_type
            if dis:
                dis = tariff.cost *
disc.percentage
            else:
                dis = calculates_cost *
disc.percentage
            if (max_discount and dis >
max_discount) or (not max_discount):
                max_discount = dis
            if max_discount:
                calculates_cost = calculates_cost -
max_discount
            if not min_cost:
                min_cost = calculates_cost
            elif calculates_cost < min_cost:
                min_cost = calculates_cost
            calculates_cost = 0
            if tariff.start_date >= reserve_from and
tariff.end_date >= reserve_to:
                months = tariff.end_date - reserve_to
                months = months.months
                calculates_cost = calculates_cost +
months * tariff.cost
            end_tariff =
tariffs.filter(start_date__lte = tariff.end_date, end_date__gte
= reserve_to).order_by('cost')[0]
            months = reserve_to - tariff.end_date
            months = months.months
            calculates_cost = calculates_cost +
months * end_tariff.cost
        discounts =
Discount.objects.filter(tariff = end_tariff, duration__gte =
interval.months)
        max_discount = None
        for disc in discounts:
            dis = disc.start_type
            if dis:
                dis = end_tariff.cost *
disc.percentage
            else:
                dis = calculates_cost *
disc.percentage

```

```

        if (max_discount and dis >
max_discount) or (not max_discount):
            max_discount = dis
        if max_discount:
            calculates_cost = calculates_cost -
max_discount

            if calculates_cost < min_cost:
                min_cost = calculates_cost
            calculates_cost = 0
            if tatiff.start_date <= reserve_from and
tariff.end_date >= reserve_to:
                months = reserve_to - reserve_from
                months = months.months
                calculates_cost = calculates_cost +
months * tariff.cost

            discounts =
Discount.objects.filter(tariff = tariff, duration__gte =
interval.months)

            max_discount = None
            for disc in discounts:
                dis = disc.start_type
                if dis:
                    dis = tariff.cost *
disc.percentage

                else:
                    dis = calculates_cost *
disc.percentage

            if (max_discount and dis >
max_discount) or (not max_discount):
                max_discount = dis
            if max_discount:
                calculates_cost = calculates_cost -
max_discount

            if calculates_cost < min_cost:
                min_cost = calculates_cost
            storage.cost = min_cost
        if role == 0:
            actual_user = request.user
            actual_object = Tenant.objects.get(user_id =
actual_user.id)
            storages = Storagelist.objects.filter(tenant =
actual_object, storage = storage)
            if storages:
                storage.saved = True
            else:
                storage.saved = False
        return render(request, "tempstoragesoft/storage.html",
{"storage": storage })

```

```

def contact(request, username, role = 0):
    user = request.user
    if role == 0:
        tenant_object = Tenant.objects.get(user_id =
user.id)

        if not tenant_object:
            return render(request,
"tempstoragesoft/contact.html", {"form": None, "messages": None
}

            host_user = User.objects.get(username = username)
            if host_user:
                host_object = Host.objects.get(user_id =
host_user.id)

                if not host_user or not host_object:
                    return render(request,
"tempstoragesoft/contact.html", {"form": None, "messages": None
}

                elif role == 1:
                    host_object = Host.objects.get(user_id = user.id)
                    if not host_object:
                        return render(request,
"tempstoragesoft/contact.html", {"form": None, "messages": None
}

                    tenant_user = User.objects.get(username = username)
                    if tenant_user:
                        tenant_object = Tenant.objects.get(user_id =
host_user.id)

                        if not tenant_user or not tenant_object:
                            return render(request,
"tempstoragesoft/contact.html", {"form": None, "messages": None
}

                    if request.user.is_active:
                        contact_messages = Contact.objects.filter(host =
host_object, tenant = tenant_object).order_by("-creation_date")
                        if request.method == "POST":
                            new_contant_form = ContactForm(request.POST)
                            if new_contant_form.is_valid():
                                new_contact =
new_contant_form.save(commit=False)
                                new_contact.tenant = tenant_object
                                new_contact.host = host_object
                                new_contact.author = role
                                new_contact.save()
                                redirect("contact", username = username,
role = role)

                    else:
                        new_contact_form = ContactForm ()
                        if role == 0:

```

```

        return render(request,
            "tempstoragesoft/review.html", {"form": new_contact_form,
            "messages": contact_messages, "author": tenant_object,
            "contact": host_object})
        else:
            return render(request,
                "tempstoragesoft/review.html", {"form": new_contact_form,
                "messages": contact_messages, "author": host_object, "contact":
                tenant_object })

    def contacts(request, role = 0):
        user = request.user
        if role == 0:
            tenant_object = Tenant.objects.get(user_id =
user.id)
            if not tenant_object:
                return render(request,
                    "tempstoragesoft/contacts.html", {"form": None, "contacts": None
                    }
            elif role == 1:
                host_object = Host.objects.get(user_id = user.id)
                if not host_object:
                    return render(request,
                        "tempstoragesoft/contacts.html", {"form": None, "contacts": None
                        }
                contacts_list = []
                if request.user.is_active:
                    if role == 0:
                        contacts_list = Contact.objects.filter(tenant =
tenant_object).distinct("host")
                    elif role == 1:
                        contacts_list = Contact.objects.filter(host =
host_object).distinct("tenant")
                    if role == 0:
                        return render(request,
                            "tempstoragesoft/review.html", {"form": new_contact_form,
                            "messages": contact_list, "author": tenant_object})
                        return render(request, "tempstoragesoft/review.html",
                            {"form": new_contact_form, "messages": contact_list, "author":
                            host_object})

```

ДОДАТОК В
Слайди презентації

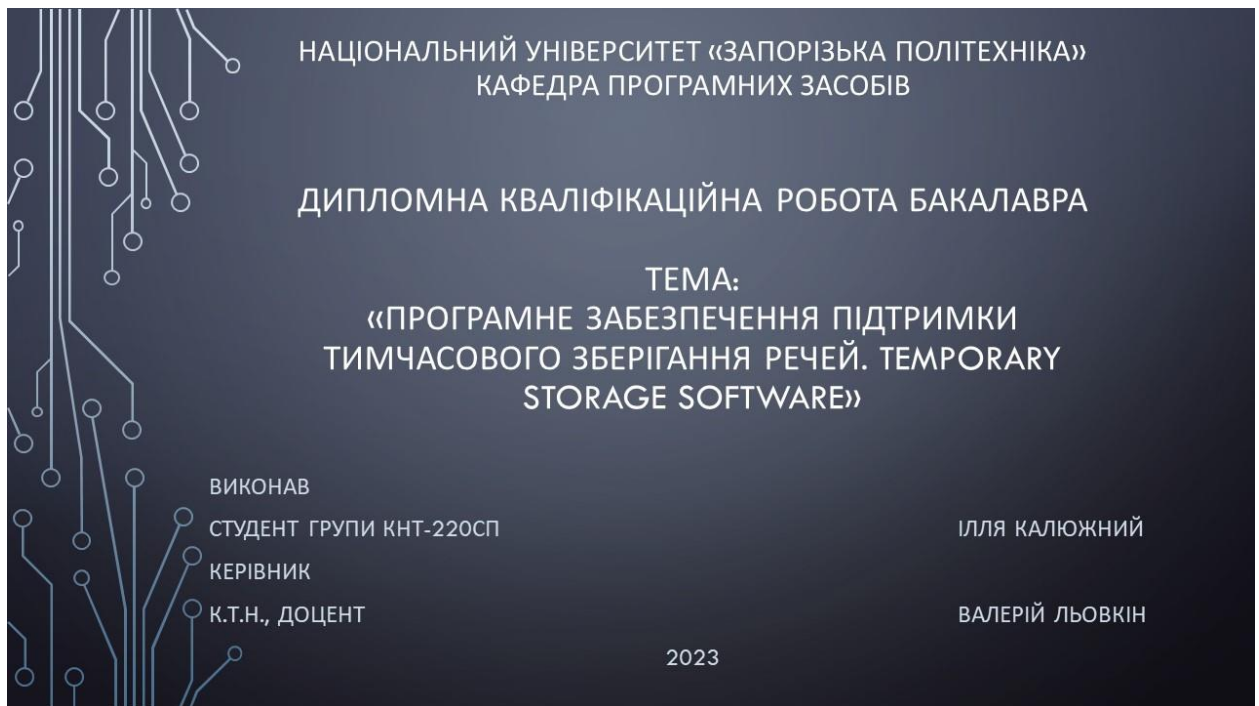


Рисунок В.1 – Початковий слайд

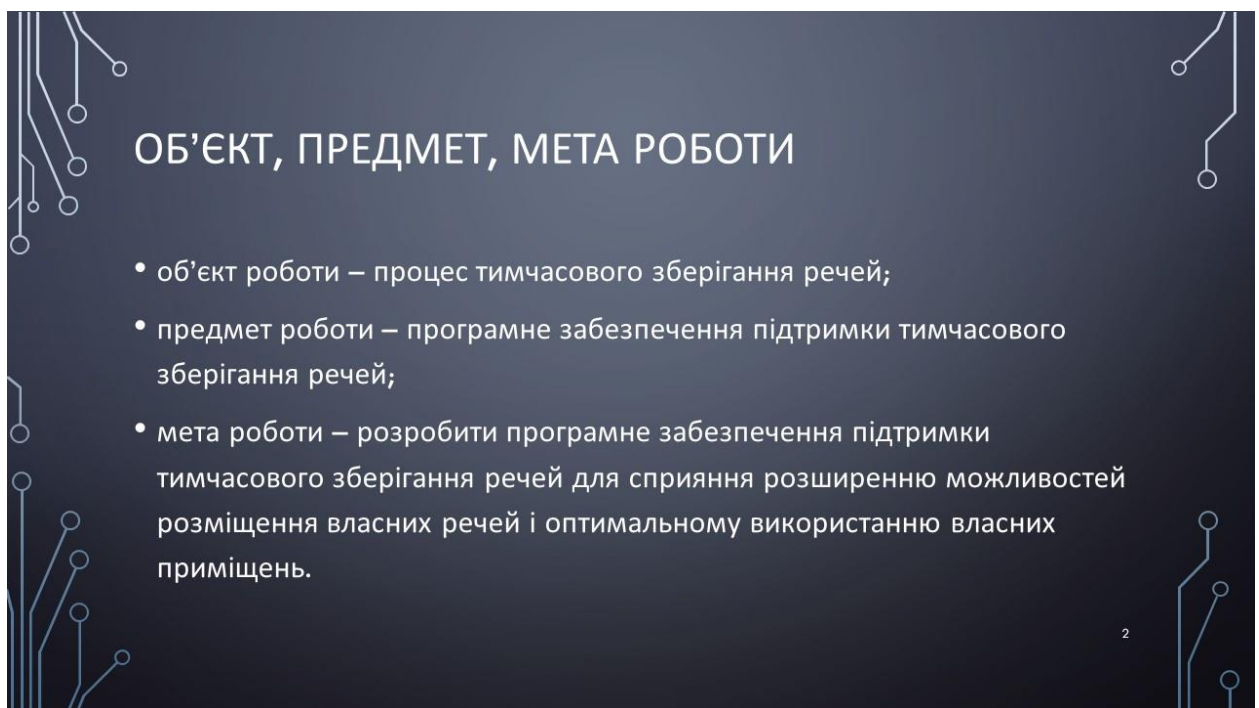


Рисунок В.2 – Мета роботи

ЗАВДАННЯ РОБОТИ

- аналіз предметної області тимчасового зберігання речей;
- визначення вимог і особливостей програмного забезпечення підтримки тимчасового зберігання речей;
- проєктування програмного забезпечення підтримки тимчасового зберігання речей;
- реалізація програмного забезпечення підтримки тимчасового зберігання речей і тестування.

3

Рисунок В.3 – Завдання роботи

ПОРІВНЯННЯ РОЗРОБКИ З ПРОТОТИПОМ

Характеристика	Neighbor	Розробка
Країни пропозицій	США	Україна
Пошук сховищ для збереження речей від господарів	Так	Так
Мінімальний період оренди	Місяць	Місяць, день
Дисконт за тарифом	На перший місяць	Різні варіанти
Направленість відгуків	Сховище	Сховище, орендар
Спільний доступ до резервувань	Ні	Так
Спільний доступ до оголошень при пошуку сховищ	Ні	Так

4

Рисунок В.4 – Порівняння розробки з прототипом

ПОРІВНЯННЯ ВЕБФРЕЙМВОРКІВ ДЛЯ РОЗРОБКИ		
Показник	Django	TurboGears
Тип	Повний стек	Повний стек
Вбудована адміністративна панель	Так	Так
Засоби безпеки	Засоби проти декількох напрямків атак, дослідження командою нових вразливостей	Вбудовані тільки мінімальні
Бібліотека шаблонів	Так, власна	Так, Kajiki
Засоби роботи з базою даних	Об'єктно-реляційне відображення	Об'єктно-реляційне відображення
Засоби забезпечення авторизації	Так	Так

Рисунок В.5 – Порівняння вебфреймворків для розробки



Рисунок В.6 – Діаграми прецедентів



АЛГОРИТМ ПОШУКУ СХОВИЩА ДЛЯ ЗБЕРІГАННЯ РЕЧЕЙ



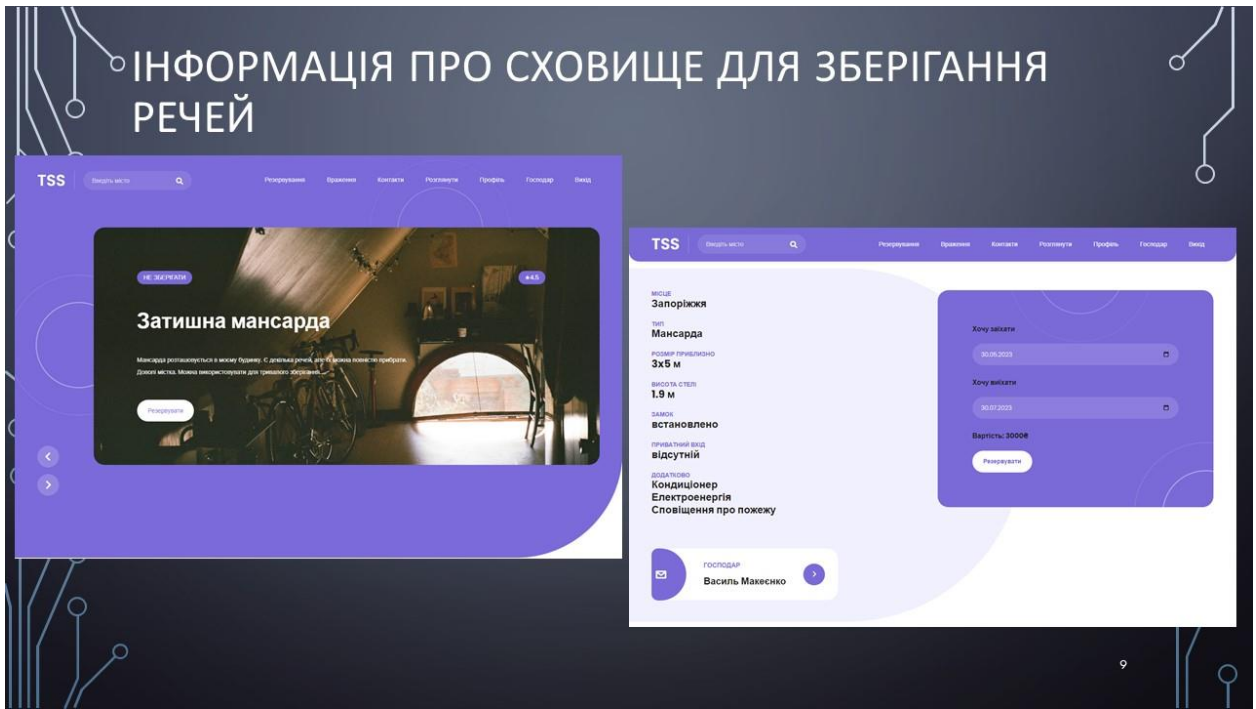


Рисунок В.9 – Інформація про сховище для зберігання речей

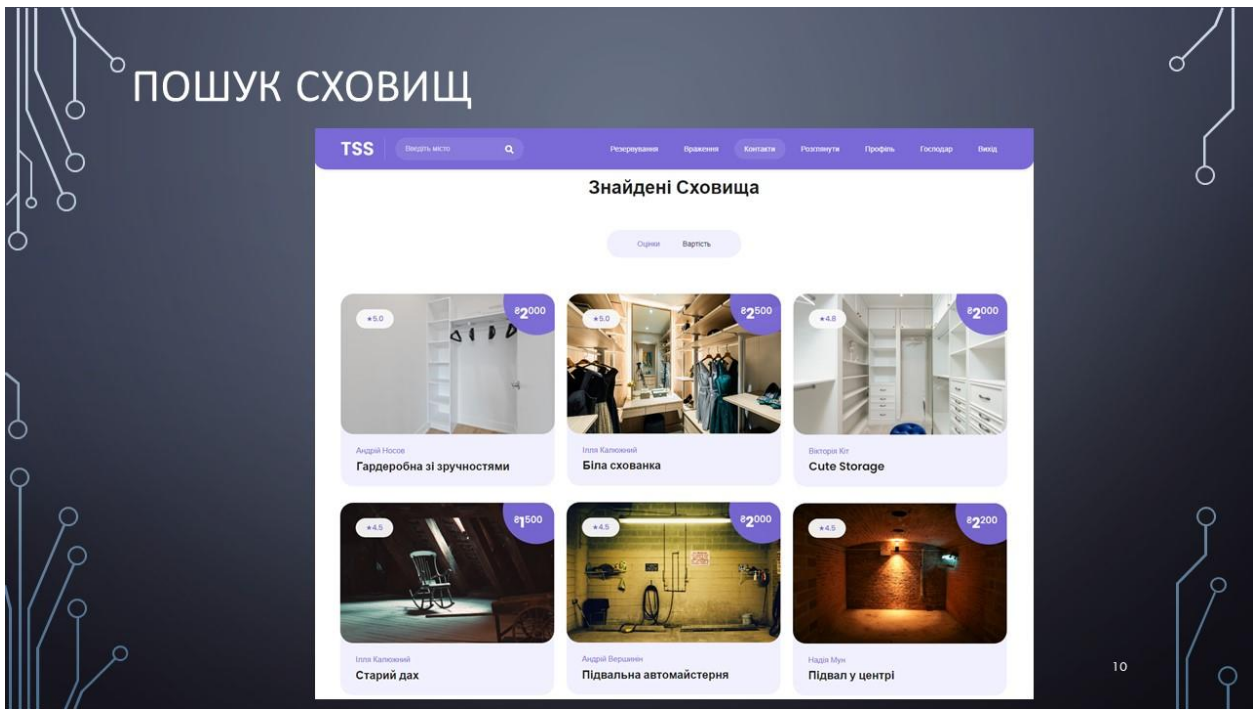


Рисунок В.10 – Пошук сховищ

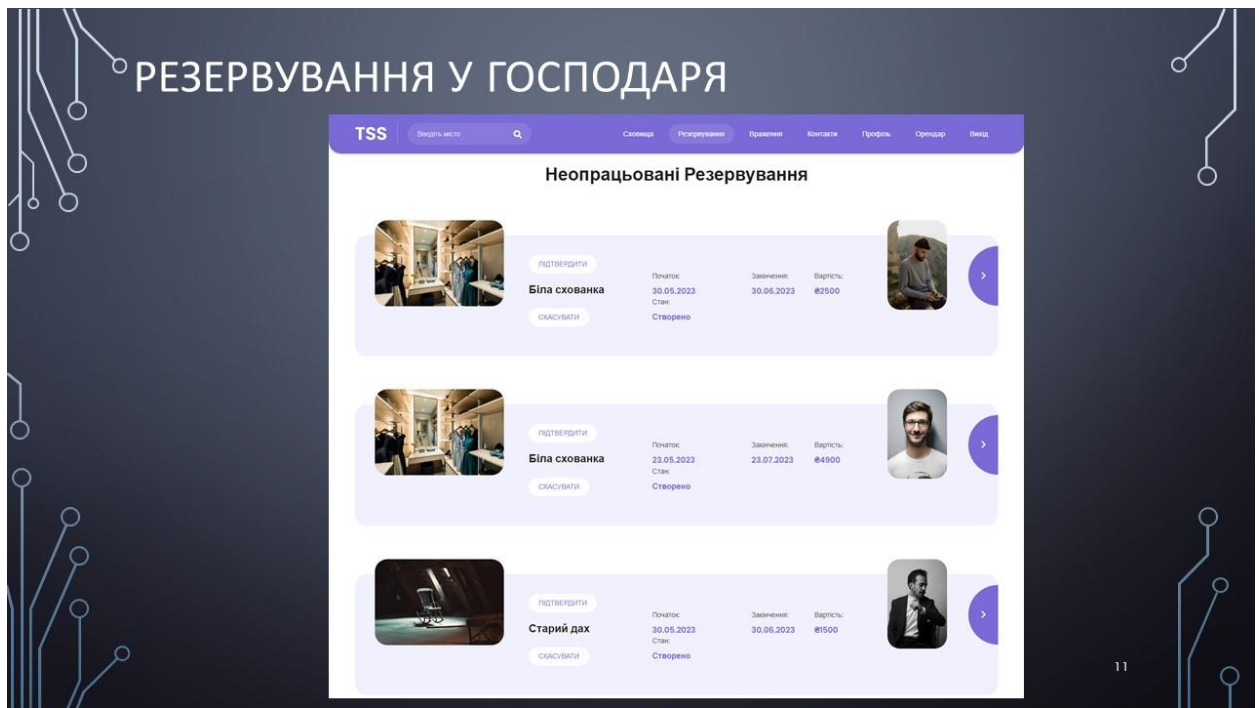


Рисунок В.11 – Резервування у господаря

ТЕСТУВАННЯ ПРОГРАМИ		Сценарій	Підсумок
		Встановлення тарифу на зберігання речей у сховищі	+
		Встановлення вільних дат для резервування сховища	+
Пошук сховища для збереження речей	+	Відтворення всіх актуальних тарифів на зберігання речей у сховищі	+
Відтворення повних даних орендаря	+	Змінювання тарифу на зберігання речей у сховищі	+
Відтворення повних даних господаря	+	Встановлення дисконту за тарифом	+
Залишення відгуку орендарем про сховище	+	Відтворення можливих дисконтів за тарифом	+
Залишення відгуку господарем про орендаря	+	Вилучення тарифу на зберігання речей у сховищі	+
Виставлення власного сховища для резервування для зберігання речей	+	Вилучення дисконту за тарифом	+
Змінювання даних про сховище для резервування	+	Розширення резервування сховища для зберігання речей	+
Відтворення всіх сховищ господаря	+	Внесення сплати за наступний період резервування	+
Резервування сховища господаря	+	Надання повних даних про сховище	+
Відтворення резервувань за сховищем господаря	+	Надсилання повідомлення орендарю	+
Відтворення неопрацьованих господарем резервувань	+	Надсилання повідомлення господарю	+
Підтвердження резервування	+	Відтворення розмови з орендарем	+
Відхилення резервування	+	Відтворення розмови з господарем	+
Надання відповіді на скаргу	+	Відтворення розпочатих розмов з господарями	+
Відтворення всіх непроаналізованих скарг за резервуваннями	+	Відтворення розпочатих розмов з орендарями	+
Надання спільного доступу до сховища для детального розгляду іншому орендарю	+	Розділення доступу до резервування з іншим орендарем	+
Вилучення спільного доступу до сховища для детального розгляду іншим орендарем	+	Відтворення резервувань зі спільного доступу	+
Закриття доступу орендаря або господаря після аналізу скарг	+	Відтворення всіх доступних для зберігання речей орендарем резервувань	+
Змінювання власних даних	+	Внесення сховища для детального розгляду	+
Надсилання скарги за резервуванням	+	Відтворення сховищ для детального розгляду	+
Відтворення всіх скарг за резервуваннями	+	Відтворення сховищ для детального розгляду зі спільного доступу	+
Відтворення всіх відгуків господаря	+	Вилучення сховища з детального розгляду	+
Відтворення всіх відгуків за сховищем	+	Відтворення всіх відгуків за орендарем	+
Відтворення всіх відгуків за всіма сховищами господаря	+	Відтворення всіх відгуків орендаря	+

Рисунок В.12 – Тестування програми

ВИСНОВКИ ЗА РОБОТОЮ

Виконуючи проектування програми, зроблено вибір на користь Python і Django для реалізації вебдодатку, що дозволяє спростити процес розробки і створити в результаті надійний інструмент для підтримки процесу тимчасового зберігання речей. Згідно з особливостями, відмінностями програми та базовими можливостями створено діаграми прецедентів для господаря, орендаря та менеджера. Проектування роботи з даними призвело до створення схеми бази даних, що фактично у відповідності з архітектурою Django-додатків призвело до визначення структури моделей додатку.

Вебдодаток реалізовано повністю. Усі функціональні вимоги забезпечено створенням функцій представлень, менеджеру URL, класів моделей, шаблонів вебсторінок. Усі потрібні сценарії роботи забезпечені користувацькими інтерфейсами. Тестування реалізовано за кожним сценарієм і проведено успішно.

Створене програмне забезпечення дозволяє задовольнити потребу в тимчасовому зберіганні речей різного роду, сприяючи розвитку цих засобів на території України, надаючи господарям приміщень і людям, які знаходяться в пошуку сховищ, знайти можливість для взаємодії та взаємовигідно задовольнити потреби один одного.

13

Рисунок В.13 – Висновки за роботою