

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій
(повне найменування інституту, факультету)

Кафедра програмних засобів
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ ОРЕНДИ СКУТЕРІВ
SCOOTER RENTAL SOFTWARE

Виконав: студент(ка) 4 курсу, групи КНТ-138
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)
Інженерія програмного забезпечення

Сурядний І.Ю.

(прізвище та ініціали)

Керівник Степаненко О.О.

(прізвище та ініціали)

Рецензент Казурова А.Є.

(прізвище та ініціали)

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

Інститут, факультет ФКНТ
 Кафедра програмних засобів
 Ступінь вищої освіти бакалавр
 Спеціальність 121 Інженерія програмного забезпечення
 (код і найменування)
 Освітня програма (спеціалізація) Інженерія програмного забезпечення
 (назва освітньої програми (спеціалізації))

ЗАТВЕРДЖУЮ

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

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА(КИ)

Сурядного Ігоря Юрійовича
 (прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення підтримки оренди скутерів
Scooter Rental Software

керівник проєкту (роботи) Степаненко Олександр Олексійович, к.т.н., доцент,
 (прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

затверджені наказом закладу вищої освіти від «27» квітня 2022 року № 102

2. Строк подання студентом проєкту (роботи) червень 2022 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. Розробка архітектури програми. 3. Розробка програми. 4. Експлуатація та тестування програми.

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

6. Консультанти розділів проекту (роботи)

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	прийняв виконане завдання
1-4 Основна частина	Степаненко О.О., доцент		
Нормоконтролер	Камінська Ж.К., асистент		

7. Дата видачі завдання « 4 » квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

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

Студент(ка)

(підпис)

Сурядний І.Ю.
(прізвище та ініціали)

Керівник проекту (роботи)

(підпис)

Степаненко О.О.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 83 с., 21 рис., 23 табл., 3 дод., 11 джерел.

СКУТЕР, ОRENDA СКУТЕРІВ, ОБ'ЄКТНО-ОРІЄНТОВАНЕ ПРОГРАМУВАННЯ, ФРЕЙМВОРК, БАЗА ДАНИХ.

Об'єкт роботи – процес оренди скутерів. Предмет роботи – програмне забезпечення підтримки оренди скутерів. Мета роботи – розробити програмне забезпечення підтримки оренди скутерів для забезпечення взаємодії між орендарями і службою оренди.

Матеріали, методи та технічні засоби: об'єктно-орієнтоване програмування, мови Python, JavaScript, HTML, CSS, вебфреймворк Django, система керування базами даних MySQL.

Результати. Проаналізовано програмні аналоги, сформульовано вимоги до програми, виконано вибір комплексу засобів для розробки, визначено структуру бази даних, логіку роботи програми, описано основні розроблені класи та представлено їх ієрархію, реалізовано програму і представлено процес роботи з програмою.

Висновки. Програмне забезпечення створено для надання через нього можливостей контролю за орендами зі сторони менеджерів компанії, для керування ними орендами, у той же час орендар також може користуватися програмою з метою створення попередніх замовлень, щоб бути впевненим у тому, що потрібний йому скутер буде в наявності, коли він прийде у відділення, а також для того, щоб відслідковувати власні оренди, включаючи ті, що вже були виконані, передзамовлені та ті, за якими він ще має виконати власні зобов'язання.

Галузь використання. Служба, яка надає скутери в оренду, на нетривалий період.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 83 pages, 21 figures, 23 tables, 3 appendixes, 11 sources.

SCOOTER, SCOOTER RENTAL, OBJECT-ORIENTED PROGRAMMING, FRAMEWORK, DATABASE.

The object of the work is a process of renting scooters. The subject of the work is a scooter rental support software. The purpose of the work is to develop scooter rental support software to ensure interaction between tenants and the rental service.

Materials, methods and technical means: object-oriented programming, Python languages, JavaScript, HTML, CSS, Django web framework, MySQL database management system.

Results. Software analogues were analyzed. Program requirements were formulated. A set of tools for the development was selected. Database structure, program operation logic were defined. The main developed classes were described and its hierarchy was presented. The software was implemented and program work process was presented.

Conclusions. The software is designed to control rentals by a company's managers, to manage rentals, while a tenant can also use the program to create pre-orders to make sure that the scooter he needs will be available, when he comes to the department, as well as to monitor his own rentals, including those rentals which have already been fulfilled, pre-ordered and those for which he has yet to fulfill his own obligations.

Field of use. A service which rents scooters for a short period.

ЗМІСТ

	С.
Перелік скорочень та умовних позначок.....	7
Вступ	8
1 Аналіз предметної області	9
1.1 Огляд особливостей оренди скутерів.....	9
1.2 Огляд програмного забезпечення для оренди скутерів.....	10
1.3 Висновки за розділом 1	12
2 Розробка архітектури програми.....	14
2.1 Вибір засобів проєктування і розробки.....	14
2.2 Визначення функціональних вимог до програмного забезпечення	15
2.3 Визначення структури бази даних	16
2.4 Проєктування логіки роботи програми.....	28
2.5 Висновки за розділом 2	29
3 Розробка програми.....	30
3.1 Структура реалізованих одиниць застосунку	30
3.2 Опис реалізованих одиниць вебзастосунку	31
3.3 Висновки за розділом 3	41
4 Експлуатація та тестування програми	42
4.1 Опис програми	42
4.2 Експлуатація програми	42
4.3 Тестування програми.....	47
4.4 Висновки за розділом 4.....	47
Висновки	48
Перелік джерел посилання	49
Додаток А Технічне завдання.....	51
Додаток Б Текст програми.....	55
Додаток В Слайди презентації	77

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

MVT – Model View Template;

БД – база даних.

ВСТУП

Сплеск цін на бензин неможливо не відчуту, і він в даний момент є в центрі уваги споживачів, оскільки рекламні повідомлення, статті в інформаційних агенствах постійно повідомляють, що бензин коштує в декілька разів вище ніж раніше, а в багатьох країнах і ще вище (так в Сполучених штатах вартість досягає 4 або навіть 5 доларів, а в деяких місцях навіть вище 6 доларів за галон), з такими рекордно високими цінами громадяни по всьому світу відразу відчують вплив на транспортні витрати, але вищі ціни на паливо також є зустрічним вітром для всієї економіки, не тільки для споживачів, які витрачають гроші безпосередньо на паливо, зростання вартості палива, особливо дизельного палива, означає, що це впливає на все, що перевозиться на вантажівці, поїзді чи кораблі, а витрати на паливо є основною причиною інфляції протягом десятиліть, тож ціни на всі види товарів і послуг зростають [1].

В результаті для транспортування містом люди все частіше намагаються використовуватися громадський транспорт або транспорт, який витрачає менше пального. Таким транспортом і є скутер. Це можуть бути і електричні моделі, і звичайні моделі, оскільки останні характеризуються невеликими витратами, а найпростіші доступні для водіння з правами категорії A1, які видаються з 16 років. Тож все це разом підтверджує, що саме в поточний момент зокрема, але і загалом в майбутньому оренда скутерів є напрямком, який дуже активно розвивається. Потреба в наявності програмного забезпечення для підтримки роботи служби оренди скутерів через це є великою і вона актуалізується більшою мірою в поточний момент. Тому в роботі і повинна бути розв'язана проблема розроблення програмного забезпечення підтримки оренди скутерів.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Огляд особливостей оренди скутерів

Оренда скутерів, особливо електричних, на даний момент стає все більш популярним способом переміщення по містах по всьому світу, при чому все це стосується багатьох країн світу, починаючи з США і продовжуючи містами України, де оренда скутерів набуває все більшої популярності: скутери забезпечують недорогу і зручну в експлуатації навігацію по місту, до того ж скутери є загалом достатньо екологічними рішеннями, а електричні скутери остаточно зменшують шкідливі викиди вуглекислого газу, а окрім того всі скутери зменшують затори на дорогах [2].

Опитування 7000 людей, проведене дослідницькою фірмою Populous, показало, що 3,6 % людей повідомили, що користувалися скутерами за відносно короткий час, коли вони були доступні, що є швидшим рівнем поширення, ніж у конкурентів, таких як велосипеди, при цьому майже 70 % людей позитивно ставляться до самокатів без докстанції [3].

На тлі на даний момент все ж таки певного розділення між тими, хто позитивно ставиться до скутерів і тих, хто проти них, відбувається активний рух до швидких змін у містах і громадських місцях, при цьому скутери нав'язують розмови про те, хто має право користуватися тротуарами, вулицями та бордюрами, а хто має платити за їх утримання, вони також викривають транзитні пустелі, показуючи, які території обслуговуються громадським транспортом краще, а які гірше, а те, що люди охоче взяли за скутери, свідчить про те, який прихований попит існує на швидкий і дешевий спосіб пересування містами [4]-[5].

Скутери також певним чином кидають виклик автомобілям: більшість автомобільних поїздок короткі, і якщо електричні скутери в кінцевому підсумку замінять деякі з них, вони можуть зменшити затори та допомогти довкіллю [4].

Усе це загалом вказує на те, що оренда скутерів є важливим і актуальним напрямком, який активно розвивається зокрема через наявність електричних скутерів. Підтримка такої оренди може здійснюватися по-різному. Одним з варіантів є існування класичних оренд, де видаються як електричні, так і класичні скутери. Для підтримки роботи такої служби потрібно відповідне програмне забезпечення.

1.2 Огляд програмного забезпечення для оренди скутерів

Одним з прикладів окремих сервісів оренди скутерів, що по суті є окремим відділенням, а не певною мережею, є Roma Rent Scooter (рис. 1.1)[6].

Roma Rent Scooter
Scooter and Bike in Rome



SCOOTER PRICE WHERE WE ARE UTILITY CONTACT



SCOOTERS call us for special offers!!									
CATEGORY	1 DAY	2 DAYS	3 DAYS	4 DAYS	5 DAYS	6 DAYS	7 DAYS	EXTRA DAY	1 HOUR
50 CC	€ 40,00	€ 80,00	€ 110,00	€ 150,00	€ 170,00	€ 190,00	€ 200,00	€ 15,00	€ 10,00
125 CC	€ 50,00	€ 100,00	€ 140,00	€ 180,00	€ 210,00	€ 230,00	€ 240,00	€ 20,00	€ 10,00
125 CONFORT	€ 60,00	€ 120,00	€ 160,00	€ 200,00	€ 230,00	€ 260,00	€ 280,00	€ 30,00	€ 10,00
125 PREMIUM	€ 65,00	€ 120,00	€ 180,00	€ 220,00	€ 260,00	€ 290,00	€ 320,00	€ 30,00	€ 15,00
150 CC	€ 75,00	€ 150,00	€ 210,00	€ 260,00	€ 300,00	€ 340,00	€ 370,00	€ 30,00	€ 15,00
300 CC MAXI	€ 85,00	€ 170,00	€ 240,00	€ 300,00	€ 360,00	€ 400,00	€ 440,00	€ 45,00	
VESPA 125 CC - 150 CC									
CATEGORY	1 DAY	2 DAYS	3 DAYS	4 DAYS	5 DAYS	6 DAYS	7 DAYS	EXTRA DAY	1 HOUR
125 CC	€ 65,00	€ 130,00	€ 180,00	€ 220,00	€ 260,00	€ 290,00	€ 320,00	€ 30,00	€ 15,00
150 CC	€ 65,00	€ 130,00	€ 180,00	€ 220,00	€ 260,00	€ 290,00	€ 320,00	€ 30,00	

Рисунок 1.1 – Roma Rent Scooter, інтерфейс вебсайту

Вебсайт надає можливість ознайомитися з наявними скутерами і вартістю оренди. Обидві ці функції виділені в окремі статичні сторінки. На сторінці зі скутерами тільки виводяться фотографії скутерів, навіть немає опису. На сторінці вартості виводиться таблиця з вартістю оренди різної тривалості за моделями скутерів. Ніяких можливостей з замовлення оренди вебсайт, як і більшість подібних, не має.

Вебсайт BikesBooking (рис. 1.2) [7] надає набагато ширші засоби для роботи над орендами скутерів.

The screenshot displays the BikesBooking website interface. At the top, there is a navigation bar with the logo, currency (EUR), language (English), and links for login/signup and adding a rental company. The main section is titled "Search rentals" with a background image of a person on a bicycle. Below this, a search form is visible with the following fields and options:

- Country:** Italy (dropdown)
- Pick-up City:** Rome (dropdown)
- Return to the same location:** ☒
- Pick-up Date:** 20/06/2022 (calendar icon)
- Drop-off Date:** 23/06/2022 (calendar icon)
- Time:** 09:00 (dropdowns)
- Type:** ☐ Bicycle, ☒ Scooter, ☐ Motorcycle, ☐ Quad bike
- New search** button

Below the search form, there is a section for "Scooter Vespa 50cc LX 4T". It includes a photo of the scooter, a rating of 9.2, and a "Show on the city map" link. The description states: "Automatic scooter. 2 persons. Easy to use. You can drive with an ordinary car (B) driver's license or A1/A license. Rates include: Unlimited mileage; Helmets; Chain and security lock; Free assistance within the Highway belt of Rome; Vehicle registration tax; Civil liability insurance; VAT; Anti-Flat tire; Emergency Assistance. Top case can't be installed on this scooter." The driver's licence category is "B (car)" and the minimum age for the driver is 18. The price is listed as €116,40 (€120,00 crossed out). It also mentions "Best Price Guarantee", "FREE Amendments", and "All taxes and fees included". A "More info" button is present, and a note states: "Rental terms and extra services will be shown at the next step".

Рисунок 1.2 – BikesBooking, інтерфейс вебсайту

Вебсайт дозволяє виконувати пошук скутерів за різними містами, датами, надаючи можливість шукати окрім скутерів ще велосипеди,

мотоцикли, квадроцикли. Надається можливість бронювання скутера онлайн, можливість отримання інформації про власні бронювання.

Вебсайт veliki.ua (рис. 1.3) [8] не є спеціалізованим прокатом скутерів, але надає послуги прокату саме в Україні. Прокат може здійснюватися велосипедів також, що є основним напрямком діяльності. На вебсайті надається інформація про наявність такого прокату і про відділення.

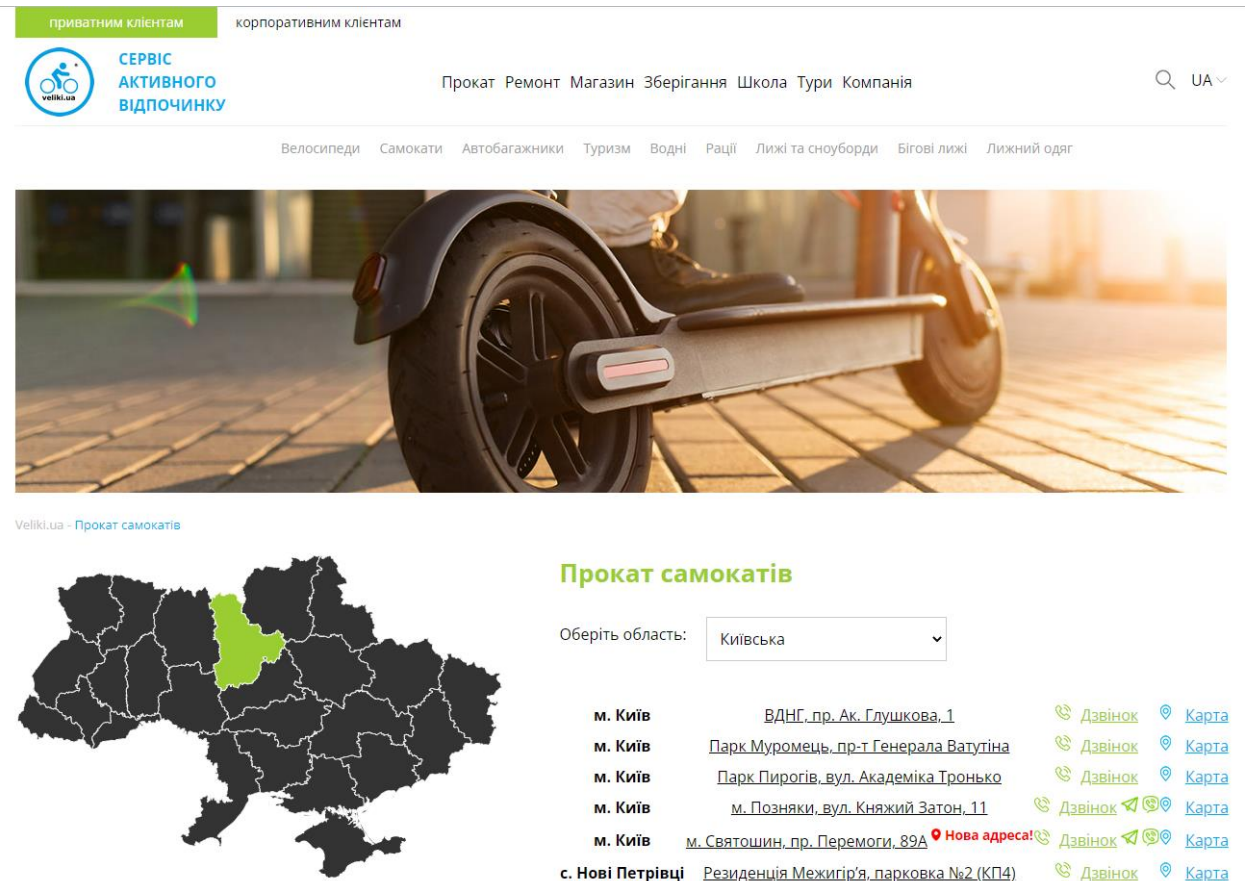


Рисунок 1.3 – Veliki.ua, інтерфейс вебсайту для оренди скутерів

1.3 Висновки за розділом 1

Огляд існуючого програмного забезпечення для оренди скутерів включав вебсайти Roma Rent Scooter, BikesBooking, Veliki.ua та інші. Проведений аналіз дозволив встановити, що в Україні подібні сервіси мають вебсайти, які не передбачають участі користувача окрім отримання інформації про оренду. Тому можна стверджувати, що важливо розробити

програму, яка вирішить цю проблему для деякої служби оренди скутерів, яка потенційно може включати декілька відділень.

Завданнями роботи для сукупного досягнення мети є:

- аналіз існуючого програмного забезпечення підтримки оренди скутерів;
- створення бази даних для роботи служби оренди скутерів;
- проєктування програми;
- створення програми.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Вибір засобів проєктування і розробки

Вибір засобів проєктування і розробки можна виконувати на основі мови програмування, але після вибору мови програмування у випадку веброзробки, якщо її виконувати на сучасному рівні та з забезпеченням невеликих строків розробки, потрібно обрати фреймворк веброзробки. Тому вибір фактично робиться в комплексі, а його результат визначає фреймворк та базові можливості мови. Вибір в роботі виконувався між фреймворками Django [9] і Laravel. Порівняння їх виконано в табл. 2.1. Результатом став вибір Django, а також автоматично мови програмування Python [10]-[11], а також додатково до цього засобів, необхідних для веброзробки, що включають мови JavaScript, HTML, CSS.

Таблиця 2.1 – Вибір фреймворка веброзробки

Показник	Django	Laravel
Повний стек веброзробки	Так	Так
Мова програмування	Python	PHP
Перевага зручності розробки мовою	Так	Ні
Вільні ліцензії розповсюдження	Так	Так
Кросплатформність	Так	Так
Підтримка об'єктно-реляційного відображення	Так	Так
Перевага швидкості роботи застосунків	Так	Ні

2.2 Визначення функціональних вимог до програмного забезпечення

Основними користувачами програми є орендар та менеджер, що представляють різні сторони даного процесу. Ще однією стороною є адміністратор, який повинен мати можливість керувати даними в системі. Оскільки цей потік роботи визначається структурою даних предметної області, то окремо не виділялися такі вимоги.

Функції програми для орендаря:

- перегляд інформації про скутер;
- замовлення оренди скутера;
- скасування оренди;
- перегляд переліку оренд скутерів цим орендарем;
- оцінювання скутера;
- реєстрація і авторизація;
- перегляд скутерів у відділенні;
- перегляд скутерів за моделлю.

Функції програми для менеджера:

- визначення видачі скутера в оренду;
- визначення повернення скутера;
- релокація скутера між відділеннями;
- визначення неповернення скутера;
- перегляд переліку оренд скутерів;
- пошук серед оренд;
- створення абонементу оренди;
- авторизація;
- встановлення вартості пакетів оренди;
- встановлення вартості абонементів;
- встановлення вартості страхувань;
- занесення страхувань;

- занесення пакетів оренди;
- занесення варіантів абонементів;
- занесення і редагування скутерів;
- занесення моделей та виробників скутерів;
- перегляд інформації про скутер;
- замовлення оренди скутера;
- скасування оренди;
- перегляд скутерів у відділенні;
- перегляд скутерів за моделлю.

2.3 Визначення структури бази даних

Роботу з базою даних (БД) реалізовано на основі використання MySQL. Проектування БД виконано за результатами аналізу предметної області.

У табл. 2.2 визначена структура таблиці Scootermode, що подає модель скутера.

Таблиця 2.2 – Структура таблиці Scootermode

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор моделі скутера
2	title	varchar(50)	назва моделі скутера
3	scootermaker	int	виробник скутера

Поле scootermaker – зовнішній ключ з таблиці Scootermaker, зв'язок один до багатьох.

У табл. 2.3 визначена структура таблиці Scootermaker, що подає виробника скутера.

Таблиця 2.3 – Структура таблиці Scootermaker

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор виробника скутерів
2	title	varchar(50)	назва виробника скутерів

У табл. 2.4 визначена структура таблиці Scooter, що подає самий скутер. Це не конкретний екземпляр скутера, але це вже певна специфікацій моделі. Тобто це реалізація фактично моделі скутера, що вже визначається певним паливом, характеристиками руху.

Таблиця 2.4 – Структура таблиці Scooter

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор скутера
2	scootermode	int	модель скутера
3	engine	varchar(100)	тип двигуна
4	fuel	float	потрібний обсяг палива для руху
5	capacity	float	потужність
6	fueltype	smallint	тип пального
7	speed	smallint	максимальна швидкість
8	driverage	smallint	мінімальний вік водія
9	license	smallint	потрібна категорія прав для керування
10	description	text	опис скутера
11	deposit	int	депозит, який необхідно внести для оренди
12	result	float	оцінка

Поле `scootermodel` – зовнішній ключ з таблиці `Scootermodel`, зв’язок один до багатьох.

У табл. 2.5 визначена структура таблиці `OfficeScooter`, що подає належність скутерів до офісу. Це ті дані, які необхідні для бронювання, адже під час бронювання не визначається конкретний екземпляр скутера, який отримає при видачі в прокат людина.

Таблиця 2.5 – Структура таблиці `OfficeScooter`

№	Ім’я поля	Тип	Опис
1	<code>id</code>	<code>int</code>	ідентифікатор прив’язування скутера до відділення
2	<code>office</code>	<code>int</code>	відділення
3	<code>scooter</code>	<code>int</code>	скутер
4	<code>quantity</code>	<code>int</code>	кількість таких скутерів
5	<code>accessed_quantity</code>	<code>int</code>	доступна на даний момент кількість скутерів

Поле `office` – зовнішній ключ з таблиці `Companyoffice`, зв’язок один до багатьох. Поле `scooter` – зовнішній ключ з таблиці `Scooter`, зв’язок один до багатьох.

У табл. 2.6 визначена структура таблиці `ScooterConcrete`, що подає екземпляр скутера.

Поле `officescooter` – зовнішній ключ з таблиці `OfficeScooter`, зв’язок один до багатьох.

У табл. 2.7 визначена структура таблиці `Rental`, що подає оренду скутера. Під орендою скутера розглядається сутність, яка охоплює одночасно і попереднє замовлення (бронювання) оренди, і подальшу фактичну оренду, коли вже скутер видається у відділенні.

Таблиця 2.6 – Структура таблиці ScooterConcrete

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор конкретного скутера
2	officescooter	int	визначення належності скутера до відділення
3	statenumber	varchar(10)	державний номер
4	vinnumber	varchar(20)	серійний номер скутера
5	state	smallint	стан скутера
6	color	varchar(30)	колір скутера

Таблиця 2.7 – Структура таблиці Rental

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор оренди
2	rentaler	int	орендар
3	booked	datetime	час, коли було створено оренду
4	state	int	стан
5	start	datetime	час, коли починається оренда
6	end	datetime	час, коли завершується оренда
7	planned_end	datetime	запланований час, коли мала бути завершена оренда
8	office	int	офіс, де реалізовано оренду
9	price	float	вартість оренди
10	deposit	int	депозит за оренду
11	prepayed	float	попередньо сплачена сума

Поле rentaler – зовнішній ключ з таблиці Rentaler, зв'язок один до багатьох. Поле office – зовнішній ключ з таблиці Companyoffice, зв'язок один до багатьох.

У табл. 2.8 визначена структура таблиці BookedItem, що подає складову оренди під час бронювання.

Таблиця 2.8 – Структура таблиці BookedItem

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор складової
2	officescooter	int	визначення належності скутера до відділення
3	rental	int	оренда
4	quantity	int	кількість таких скутерів
5	insurance	int	страхування
6	rentalpackage	int	пакет оренди

Поле officescooter – зовнішній ключ з таблиці OfficeScooter, зв'язок один до багатьох. Поле rental – зовнішній ключ з таблиці Rental, зв'язок один до багатьох. Поле insurance – зовнішній ключ з таблиці Insurance, зв'язок один до багатьох. Поле rentalpackage – зовнішній ключ з таблиці Rentalpackage, зв'язок один до багатьох.

У табл. 2.9 визначена структура таблиці RentalItem, що подає складову оренди вже під час видачі.

Таблиця 2.9 – Структура таблиці RentalItem

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор складової
2	scooterconcrete	int	екземпляр скутера, виданий в оренді
3	rental	int	оренда
4	insurance	int	страхування
5	rentalpackage	int	пакет оренди

Поле scooterconcrete – зовнішній ключ з таблиці Scooterconcrete, зв'язок один до багатьох. Поле rental – зовнішній ключ з таблиці Rental, зв'язок один до багатьох. Поле insurance – зовнішній ключ з таблиці Insurance, зв'язок один до багатьох. Поле rentalpackage – зовнішній ключ з таблиці Rentalpackage, зв'язок один до багатьох.

У табл. 2.10 визначена структура таблиці StateManagement, що подає дані про керування станами оренд.

Таблиця 2.10 – Структура таблиці StateManagement

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор зміни стану
2	rental	int	визначення належності скутера до відділення
3	saved	datetime	державний номер
4	state	int	серійний номер скутера
5	companymanager	int	менеджер, який виконав зміну
6	comment	varchar(100)	коментар

Поле rental – зовнішній ключ з таблиці Rental, зв'язок один до багатьох. Поле companymanager – зовнішній ключ з таблиці Companymanager, зв'язок один до багатьох.

У табл. 2.11 визначена структура таблиці Rentaler, що подає орендаря.

Таблиця 2.11 – Структура таблиці Rentaler

№	Ім'я поля	Тип	Опис
1	2	3	4
1	id	int	ідентифікатор орендаря
2	first_name	varchar(100)	ім'я орендаря
3	last_name	varchar(100)	прізвище орендаря

Продовження таблиці 2.11

1	2	3	4
4	username	varchar(100)	логін орендаря
5	password	varchar(100)	пароль орендаря
6	phone	varchar(13)	номер телефону орендаря

У табл. 2.12 визначена структура таблиці Comranumanager, що подає менеджера, який працює у деякому відділі оренди.

Таблиця 2.12 – Структура таблиці Comranumanager

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор менеджера
2	first_name	varchar(100)	ім'я менеджера
3	last_name	varchar(100)	прізвище менеджера
4	username	varchar(100)	логін менеджера
5	password	varchar(100)	пароль менеджера
6	phone	varchar(13)	номер телефону менеджера
7	email	varchar(100)	пошта менеджера
8	office	int	відділення, в якому працює менеджер

Поле office – зовнішній ключ з таблиці Office, зв'язок один до багатьох.

У табл. 2.13 визначена структура таблиці Pricecategoryscootermode, що подає дані про категорію вартості моделі скутера.

Поле pricecategory – зовнішній ключ з таблиці Pricecategory, зв'язок один до багатьох. Поле scootermode – зовнішній ключ з таблиці Scootermode, зв'язок один до багатьох.

Таблиця 2.13 – Структура таблиці Pricecategoryscootermodel

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор категорії вартості моделі скутера
2	scootermodel	int	модель скутера
3	pricecategory	int	категорія вартості

У табл. 2.14 визначена структура таблиці Pricecategory, що подає категорію вартості.

Таблиця 2.14 – Структура таблиці Pricecategory

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор категорії вартості
2	title	varchar(25)	назва категорії

У табл. 2.15 визначена структура таблиці Rentalpackage, що подає деякий пакет оренди скутера за тривалістю, який доступний в компанії.

Таблиця 2.15 – Структура таблиці Rentalpackage

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор пакета оренди
2	duration	int	максимальна тривалість оренди
3	union	smallint	одиниця вимірювання тривалості (день, година)

У табл. 2.16 визначена структура таблиці Rentalpackageprice, що подає дані про вартість пакетів оренди.

Таблиця 2.16 – Структура таблиці Rentalpackageprice

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор вартості пакета оренди
2	pricecategory	int	категорія вартості
3	rentalpackage	int	пакет оренди
4	price	float	вартість
5	extraprice	float	вартість у випадку перевищення часу оренди
6	union	int	одиниця вимірювання перевищення

Поле pricecategory – зовнішній ключ з таблиці Pricecategory, зв'язок один до багатьох. Поле rentalpackage – зовнішній ключ з таблиці Rentalpackage, зв'язок один до багатьох.

У табл. 2.17 визначена структура таблиці Companyoffice, що подає відділ компанії.

Таблиця 2.17 – Структура таблиці Companyoffice

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор відділу компанії
2	title	varchar(100)	назва відділу
3	address	varchar(255)	адреса відділу
4	phone	varchar(13)	номер телефону

У табл. 2.18 визначена структура таблиці Insurance, що подає страхування.

Таблиця 2.18 – Структура таблиці Insurance

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор страхування
2	title	varchar(100)	назва страхування
3	detailed	varchar(255)	детальний опис страхування

У табл. 2.19 визначена структура таблиці InsurancePrice, що подає вартість страхування.

Таблиця 2.19 – Структура таблиці InsurancePrice

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор вартості страхування
2	insurance	int	страхування
3	pricecategory	int	категорія вартості
4	price	float	вартість

Поле insurance – зовнішній ключ з таблиці Insurance, зв'язок один до багатьох. Поле pricecategory – зовнішній ключ з таблиці Pricecategory, зв'язок один до багатьох.

У табл. 2.20 визначена структура таблиці ScooterEstimation, що подає оцінювання скутера.

Таблиця 2.20 – Структура таблиці Estimation

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор оцінювання
2	scooter	int	скутер
3	rentaler	int	орендар
4	result	int	оцінка

Поле scooter – зовнішній ключ з таблиці Scooter, зв’язок один до багатьох. Поле rentaler – зовнішній ключ з таблиці Rentaler, зв’язок один до багатьох.

У табл. 2.21 визначена структура таблиці Subscription, що подає абонемент.

Таблиця 2.21 – Структура таблиці Subscription

№	Ім’я поля	Тип	Опис
1	id	int	ідентифікатор абонементу
2	period	smallint	тривалість дії абонементу
3	unit	smallint	одиниця вимірювання (день, тиждень, місяць)

У табл. 2.22 визначена структура таблиці Subscriptionprice, що подає вартість абонементів.

Таблиця 2.22 – Структура таблиці Subscriptionprice

№	Ім’я поля	Тип	Опис
1	id	int	ідентифікатор вартості абонементу
2	subscription	int	абонемент
3	pricecategory	int	категорія вартості
4	price	float	вартість

Поле subscription – зовнішній ключ з таблиці Subscription, зв’язок один до багатьох. Поле pricecategory – зовнішній ключ з таблиці Pricecategory, зв’язок один до багатьох.

Схема структури створеної БД представлена на рис. 2.1.



Рисунок 2.1 – Схема структуры БД

У табл. 2.23 визначена структура таблиці RentalerSubscriptionprice, що подає абонемент орендаря.

Таблиця 2.23 – Структура таблиці RentalerSubscriptionprice

№	Ім'я поля	Тип	Опис
1	id	int	ідентифікатор абонементу орендаря
2	rentaler	int	орендар
3	subscriptionprice	int	вартість абонементу
4	payed	float	сплачена сума
5	buyed	datetime	дата придбання
6	activied	datetime	дата початку дії

Поле rentaler – зовнішній ключ з таблиці Rentaler, зв'язок один до багатьох. Поле subscriptionprice – зовнішній ключ з таблиці Subscriptionprice, зв'язок один до багатьох.

2.4 Проєктування логіки роботи програми

Під час проєктування логіки роботи програми було встановлено, що основним ініціатором послідовності роботи програми є користувач, або точніше, користувачі різних ролей. Вони взаємодіють один з одним через оренди.

Стани оренди (рис. 2.2) визначають різні етапи роботи над програмою. До станів оренди відносяться замовлення, створено, скасовано, видане, повернено, не повернено.

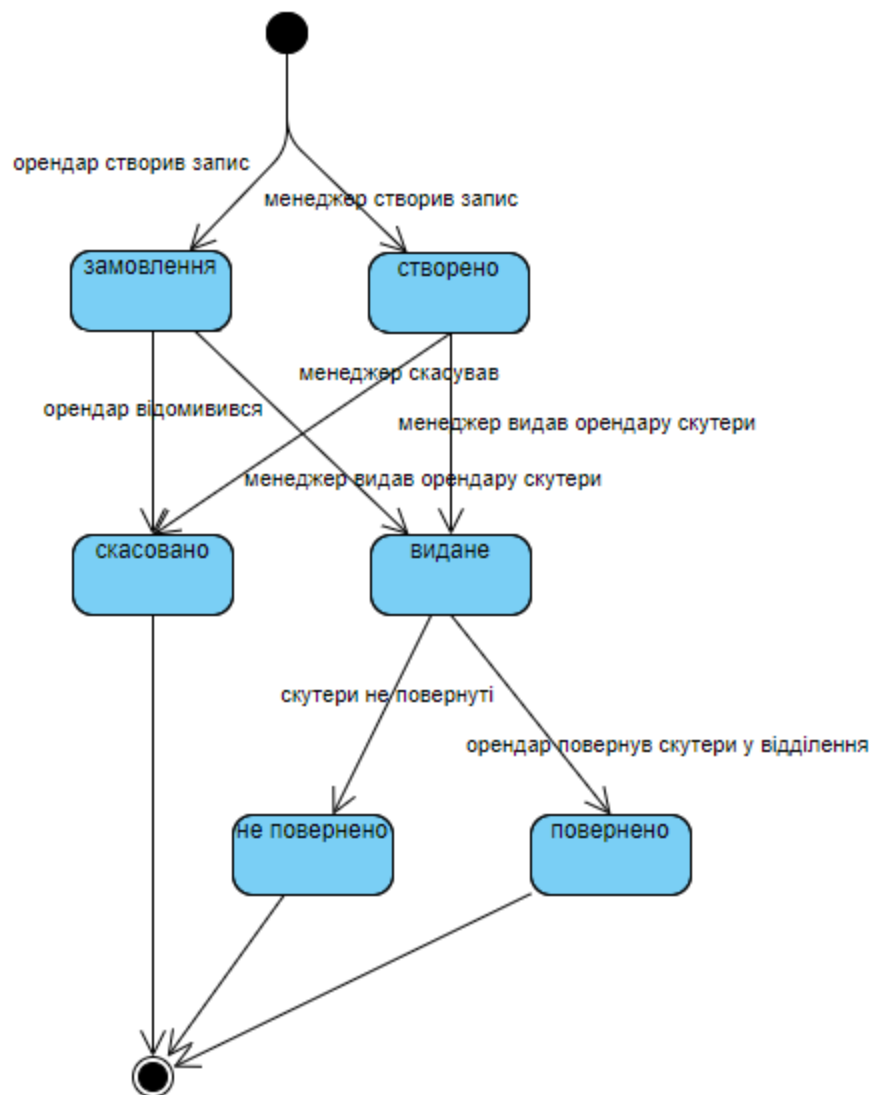


Рисунок 2.2 – Діаграма станів об'єкта оренди

2.5 Висновки за розділом 2

У другому розділі прийнято рішення щодо використання сполучення мови Python і фреймворку Django для створення програми, визначено вимоги до програми, виконано проєктування структури БД, виконано проєктування логіки роботи програми.

3 РОЗРОБКА ПРОГРАМИ

3.1 Структура реалізованих одиниць застосунку

Оскільки для реалізації програми використовувався фреймворк Django, адже він дозволяє спростити процес розробки і мати багато одразу наявних засобів, що прискорюють розробку, то структуру застосунку визначав стандартний для Django шаблон Model View Template (MVT) [11].

В шаблоні MVT центральними складовими є моделі та представлення (рис. 3.1).

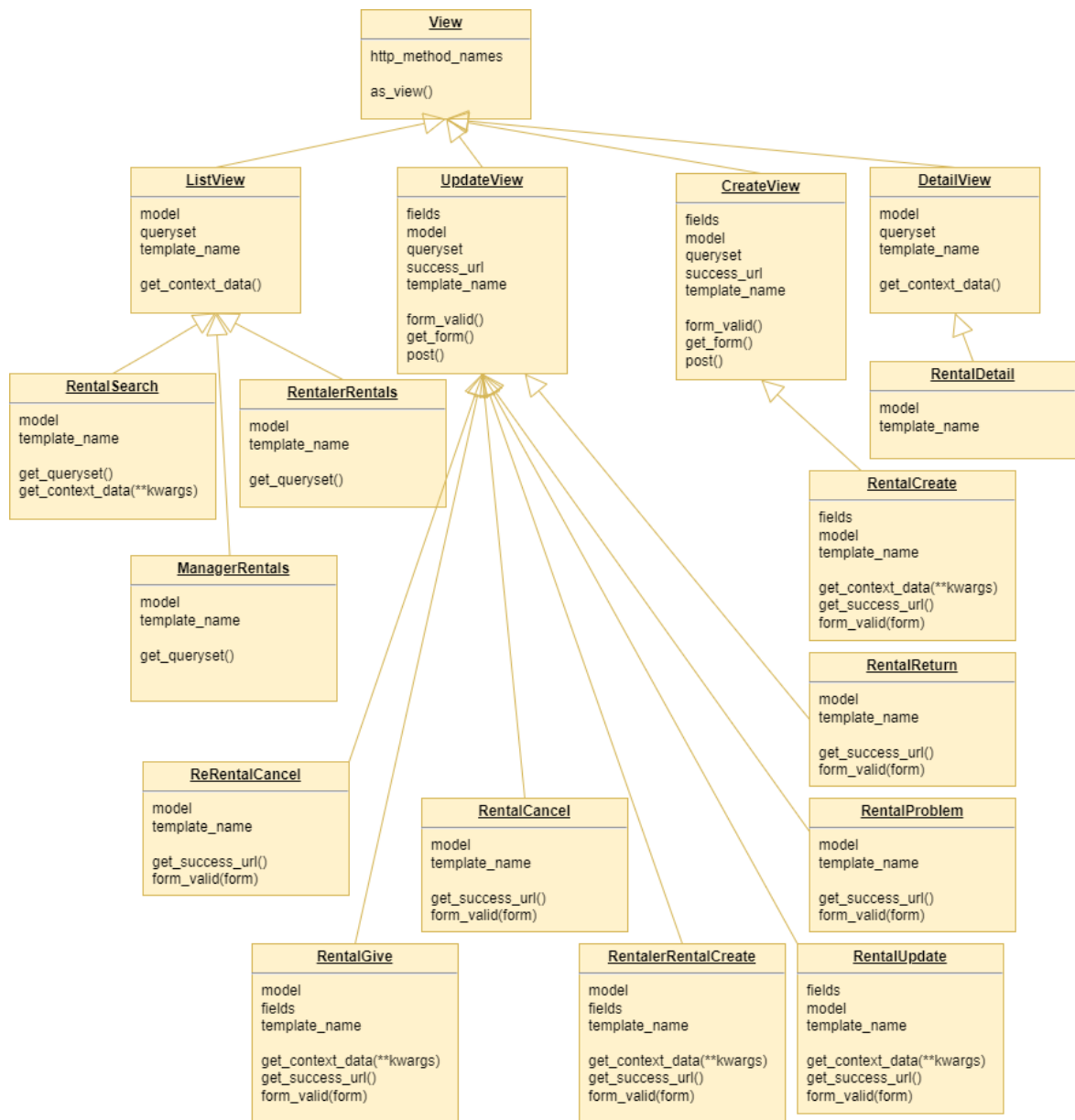


Рисунок 3.1 – Діаграма класів

Структура класів моделей відповідає структурі таблиць БД, тому її окремо не буде розглянуто. А ось структура класі представлень визначає функції і відповідні шаблони сторінок для роботи з цими функціями. На рис. 3.1 продемонстровано частину класів, які було розроблено. Це класи, які безпосередньо стосуються роботи з орендами скутерів загалом. Вони створюються наслідуванням стандартних класів Django, що були визначені на верхньому рівні на діаграмі.

3.2 Опис реалізованих одиниць вебзастосунку

Перегляд комплектаций скутерів, які є у заданому через параметр запиту відділенні, самим орендарем реалізується за допомогою класу OfficeScooters, який забезпечує формування переліку скутерів, впорядковуючи їх за назвою моделі. Заповнення цими даними виконується шаблону вебсторінки office_scooters.html.

Перегляд комплектаций скутерів, які є у відділенні, в якому працює менеджер, який увійшов в програму, тобто у відповідності з його обліковим записом, у якому міститься номер відділення, виконується за допомогою класу ManagerOfficeScooters. Заповнення цими даними виконується шаблону вебсторінки man_office_scooters.html.

Перегляд конкретних екземплярів скутерів, які є у відділенні, в якому працює менеджер, який увійшов в програму, реалізується через клас ManagerOfficeScooterConcretes. Цей влаштований таким чином, що спочатку відбираються всі комплектації скутерів, що є у відділенні, в якому працює цей менеджер, а далі відбираються всі екземпляри, які стосуються цих комплектацій:

```
scooters = Scooter.objects.filter(officescooters__office =
comoffice)
```

```
concretes =
ScooterConcrete.objects.filter(officescooter__in = scooters)
concretes = concretes.order_by('-id')
```

Заповнення цими даними виконується шаблону вебсторінки `man_office_scooterconcretes.html`.

Саме занесення екземпляра скутера реалізується за допомогою класу `ScooterConcreteCreate`. Підтримується зв'язування полів моделі та полів форми, включаючи державний, серійний номери, стан цього екземпляра та його колір. Для цього екземпляра або передається номер комплектації в конкретному відділенні, або передається номер відділення та номер комплектації скутера, а на основі цих даних встановлюється відповідно номер комплектації в конкретному відділенні. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `newscooterex.html`.

Відповідне редагування екземпляра скутера реалізується через клас `ScooterConcreteUpdate`. Підтримується зв'язування полів моделі та полів форми, включаючи державний, серійний номери, стан цього екземпляра та його колір. Для цього екземпляра у випадку необхідності переміщення на інше відділення або передається номер комплектації в конкретному відділенні, або передається номер відділення та номер комплектації скутера, а на основі цих даних встановлюється відповідно номер комплектації в конкретному відділенні. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `renewscooterex.html`.

Занесення скутера в систему реалізується за допомогою класу `ScooterCreate`. Підтримується зв'язування полів моделі та полів форми, включаючи опис двигуна, тип пального, потужність, витрату пального, максимальну швидкість, мінімальний вік водія, категорія прав, що потрібна для керування цим скутером, опис скутера та депозит, який потрібно внести за цей скутер. Для цього екземпляра також передається через запит номер

моделі скутера. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `newscooter.html`.

Відповідно редагування скутера в системі реалізується за допомогою класу `ScooterUpdate`. Підтримується зв'язування полів моделі та полів форми для випадків оновлення їх значень, включаючи опис двигуна, тип пального, потужність, витрату пального, максимальну швидкість, мінімальний вік водія, категорія прав, що потрібна для керування цим скутером, опис скутера та депозит, який потрібно внести за цей скутер. Для цього екземпляра також передається через запит номер моделі скутера, якщо він змінюється в процесі редагування. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `renewscooter.html`.

Перегляд всіх даних скутера менеджером реалізується за допомогою класу `ManagerScooterDetail`. Особливості перегляду саме менеджером визначають те, що до самих даних скутера (комплектації скутера) додається перелік конкретних екземплярів скутера в цьому відділенні. Заповнення цими даними виконується шаблона вебсторінки `man_scooter.html`.

Перегляд комплектації скутера орендарем реалізується за допомогою класу `ScooterDetail`, що за допомогою наслідування класу `DetailView` автоматично наповнюється відповідною моделлю. Використовується шаблон `scooter.html`.

Безпосередньо перегляд вже конкретного екземпляра скутера реалізується класом `ScooterConcreteDetail`. Це варіант перегляду менеджером, тому автоматично до даних цього екземпляра скутера додається перелік оренд, в складі яких міститься відповідний екземпляр скутера. Розглядаються тільки склад оренди, а не бронювання, тому що в бронюванні містяться тільки комплектації скутерів, а конкретні екземпляри містяться в складі всієї оренди:

```
rentals = Rental.objects.filter(items__contains =
self.object.pk)
rentals = rentals.order_by('-id')
```

```
scooter ['rentals'] = rentals
return scooter
```

Занесення моделі скутера в програму виконується за допомогою класу `ScootermodeCreate`. Підтримується зв'язування поля назви моделі та відповідного поля форми. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `newscootermode.html`. Окремо через сам запит до сторінки може передаватися виробник скутера.

Редагування моделі скутера в програмі виконується за допомогою класу `ScootermodeUpdate`. Підтримується зв'язування поля назви моделі та відповідного поля форми. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `newscootermode.html`. Окремо через сам запит до сторінки може передаватися виробник скутера, якщо його потрібно оновити.

Перегляд моделі скутера реалізується за допомогою класу `ScootermodeDetail`. Заповнюючи дані про модель скутера, одразу додається перелік комплектацій скутерів.

Занесення виробника скутера реалізується за допомогою класу `ScootermakerCreate`. Підтримується зв'язування поля назви виробника та відповідного поля форми. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `newscootermaker.html`.

Редагування виробника скутера реалізується за допомогою класу `ScootermodeUpdate`. Підтримується зв'язування поля назви виробника та відповідного поля форми. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `renewscootermaker.html`.

Перегляд виробника скутера реалізується за допомогою класу `ScootermakerDetail`. Заповнюючи дані про виробника скутера, одразу

додається перелік моделей скутерів, що виробляються цим самим виробником.

Перегляд оренд, які належать орендарю, що увійшов у програму, реалізується через клас `RentalerRentals`, в якому відбираються всі оренди, які були створені або належать авторизованому орендарю, впорядковуючи їх за датою бронювання, починаючи з останніх:

```
rentaler = self.request.user
rentalerc = Rentaler.objects.get(id = rentaler)
rentals = Rental.objects.filter(rentaler = rentalerc)
rentals = rentals.order_by('-booked')
```

Перегляд оренд у відділенні менеджера, який увійшов у програму, реалізується за допомогою класу `ManagerRentals`. Обираються всі оренди, які стосуються скутерів у відділенні, з яким працює менеджер, що авторизувався в програмі. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `manager_rentals.html`.

Перегляд деталей оренди виконується за допомогою класу `RentalDetail`. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `rental.html`.

Занесення варіанта вартості абонементу в систему реалізується за допомогою класу `Subscriptionprice`. Підтримується зв'язування поля моделі вартості та поля форми. Значення категорії вартості та абонементу реалізується через визначення значення в параметрі запиту. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `newsubscriptionprice.html`.

Відповідно варіанта вартості абонементу в системі реалізується за допомогою класу `SubscriptionpriceUpdate`. Підтримується зв'язування поля моделі та поля форми для випадків оновлення значення вартості. Оновленні значення абонементу та категорії вартості оренди передаються через

параметри запиту. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `renewsubscriptionprice.html`.

Замовлення оренди менеджером реалізується за допомогою класу `RentalCreate`. Підтримується зв'язування полів початку оренди, запланованого завершення оренди, відділення, до якого належить оренда, внесеного депозиту і передплатеної суми та відповідних полів форми. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `newsrental.html`. Варто зазначити, що це представлення є особливим в сенсі того, що воно має додатковий важливий параметр, яким є переліченням комплектацій скутера в замовленні, тобто склад замовлення. Він розбирається розділенням через коми значення, а тоді за кожним таким значенням, яке є ідентифікатором комплектації скутера, паралельно розбираються відповідні значення кількості комплектацій, а тоді такі пари зберігаються як склад замовлення в БД:

```
items = self.kwargs['items']
items = items.split(',')
quant = self.kwargs['quantities']
quant = quant.split(',')
k = 0
for item in items:
    item = Scooter.objects.get(id = item)
    BookedItem.objects.create(officescooter = self.office,
rental = self.object.pk, quantity = quant[k], insurance = ins)
    k = k + 1
```

Відповідним чином визначається стан замовлення, а також обчислюється сумарна вартість замовлення. При цьому також визначається страхування та орендар через параметри запиту до сторінки.

Редагування оренди менеджером реалізується за допомогою класу `RentalUpdate`. Підтримується зв'язування полів початку оренди, запланованого завершення оренди, відділення, до якого належить оренда, внесеного депозиту і передплатеної суми та відповідних полів форми.

Значення змінюються у випадку необхідності зміни в формі. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `updatesrental.html`. При цьому також визначається страхування та орендар через параметри запиту до сторінки.

Варто зазначити, що представлення має додатковий важливий параметр, яким є переліченням комплектацій скутера в замовленні, тобто склад замовлення. Він розбирається розділенням через коми значення, а тоді за кожним таким значенням, яке є ідентифікатором комплектації скутера, паралельно розбираються відповідні значення кількості комплектацій, а тоді такі пари зберігаються як склад замовлення в БД, перевіряючи, чи таке значення змінилось. Якщо кількість змінилась, вона змінюється в БД. Якщо зі складу оренди взагалі така комплектація зникла, вона зникає з БД, якщо цей запис не внесений в БД, то він навпаки вноситься.

Відповідно вилучення скутера з замовлення реалізується через клас `BookedItemDelete`, а вилучення конкретного екземпляра скутера з оренди реалізується через клас `RentalItemDelete`.

Замовлення оренди орендарем реалізується за допомогою класу `RentalerRentalCreate`. Підтримується зв'язування полів початку оренди, запланованого завершення оренди та відповідних полів форми. Також через параметр запиту вноситься інформація про відділення оренди та страхування, що використовується в оренді.

Варто зазначити, що представлення має додатковий важливий параметр, яким є переліченням комплектацій скутера в замовленні, тобто склад замовлення. Він розбирається розділенням через коми значення, а тоді за кожним таким значенням, яке є ідентифікатором комплектації скутера, паралельно розбираються відповідні значення кількості комплектацій, а тоді такі пари зберігаються як склад замовлення в БД.

Видача скутера в оренду реалізується за допомогою класу `RentalGive`. Підтримується зв'язування поля запланованого завершення оренди та відповідного поля форми. Також через параметр запиту вноситься

інформація про відділення оренди та страхування, що використовується в оренді. В цьому класі виконується перевірка часу початку. Час початку або задається як параметр запиту, або встановлюється автоматично на основі поточного моменту:

```

        if self.kwargs['start']:
            start =
datetime.datetime.strptime(self.kwargs['start'],
"%d.%m.%Y").date()
            self.start = start
        else:
            start = date.today()
            self.start = start

```

Повернення скутера з оренди реалізується за допомогою класу `RentalReturn`. Підтримка роботи представлення виконується за допомогою шаблону `rentalreturn.html`. Через параметри запиту визначається коментар, а також автоматично встановлюється відповідний стан оренди та фіксується час завершення оренди. Аналогічно описаному вище рішення, час завершення або передається через параметр запиту, або встановлюється автоматично на основі поточного моменту.

Фіксація неповернення скутера з оренди реалізується за допомогою класу `RentalProblem`. Через параметри запиту визначається коментар, а також автоматично встановлюється відповідний стан оренди.

Скасування скутера з оренди реалізується за допомогою класу `RentalCancel`. Підтримка роботи представлення виконується за допомогою шаблону `rentalcancel.html`. Через параметри запиту визначається коментар, а також автоматично встановлюється відповідний стан оренди.

Скасування оренди самим орендарем реалізується за допомогою класу `ReRentalCancel`. Підтримка роботи представлення виконується за допомогою шаблону `rentalercancel.html`. Зміна стану реалізується таким чином тільки

тоді, коли оренда тільки була створена користувачем як замовлення і в інші стани не була переведена.

Оцінювання скутера реалізується за допомогою класу `EstimationCreate`. Автоматично фіксується орендар, який виконав це оцінювання. Шляхом зв'язування поля форми оцінки з полем моделі вноситься відповідне значення.

Редагування моделі скутера реалізується за допомогою класу `ScootermodelUpdate`. Підтримується зв'язування поля назви моделі та відповідного поля форми. Значення змінюється у випадку необхідності зміни в формі.

Оновлення замовлення реалізується за допомогою класу `BookedItemUpdate`. Підтримується зв'язування поля кількості скутерів цієї комплектації та відповідного поля форми. Значення змінюється у випадку необхідності зміни в формі.

Пошук оренди реалізується за допомогою класу `RentalSearch`, який виконує підтримку шаблону `search.html`. При цьому через параметри запиту передаються параметри пошуку оренд, що має надаватися менеджеру відділення. До цих параметрів відносяться наступні.

Перший параметр – це номер телефону орендаря, за яким він виконав замовлення або який був зафіксований під час видачі скутерів (`phone`).

Другий параметр – найраніший час початку оренди (`period1`), а третій відповідно – найпізніший час (`period2`).

```
if period1:
    concretes = concretes.filter(start__gte = period1)
if period2:
    concretes = concretes.filter(start__gte = period2)
if phone:
    concretes = concretes.filter(rentaler__phone = phone)
return concretes.order_by('-id')
```

Четвертий параметр – номер оренди, якщо він чітко відомий менеджеру.

П'ятий параметр – номер комплектації скутера, якщо він відомий (передається номер, а в інтерфейсі виводиться назва).

За цими параметрами обирається перелік оренд і повертається як результат:

```
if rental_id:
    concretes = ScooterConcrete.objects.get(id = rental_id)
else:
    if scooterex:
        concretes = concretes.filter(Q(items__contains =
scooterex) | Q(bitems__contains = scooterex))
    elif scooter:
        scooterconcretes =
ScooterConcrete.objects.filter(officescooter__scooter = scooter)
        concretes = concretes.filter(reduce(operator.and,
(Q(items__contains = x) for x in scooterconcretes) |
Q(bitems__contains = scooter)))
```

Розміщення скутерів реалізується за допомогою класу `ScooterOfficeCreate`. Підтримується зв'язування полів кількості скутерів у відділенні та доступної кількості цих скутерів та відповідних полів форми. Номери офісу та комплектації скутера встановлюються через параметр запиту.

Релокація скутерів реалізується за допомогою класу `ScooterOfficeUpdate`. Підтримується зв'язування полів кількості скутерів у відділенні та доступної кількості цих скутерів та відповідних полів форми, що таким чином відповідно оновлюються у випадку необхідності. Номери офісу та комплектації скутера встановлюються через параметр запиту, якщо їх потрібно змінити.

Занесення варіанта абонементу в систему реалізується за допомогою класу `SubscriptionCreate`. Підтримується зв'язування полів моделі та полів

форми, включаючи період дії абонементу та одиницю виміру цієї дії. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `newsubscription.html`.

Відповідно редагування варіанта абонементу в системі реалізується за допомогою класу `SubscriptionUpdate`. Підтримується зв'язування полів моделі та полів форми для випадків оновлення їх значень, включаючи період дії абонементу та одиницю виміру цієї дії. Підтримка роботи з користувачем через це представлення реалізується за допомогою шаблону вебсторінки `renewsubscription.html`.

3.3 Висновки за розділом 3

У третьому розділі виконано реалізацію програми у відповідності з раніше визначеними вимогами до неї, а в підсумку описано, яким чином цю програмну реалізацію було виконано.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Опис програми

Програма призначена для підтримки роботи служби оренди скутерів та передзамовлення і стеження за власними орендами зі сторони орендарів.

Для підтримки роботи програми необхідно, щоб апаратна частина сервера включала:

- процесор (4 ядра, не менше 2,1 ГГц);
- оперативну пам'ять (від 4 Гб);
- жорсткий диск з наявним вільним простором (від 1 Гб).

Для клієнтської частини достатньо мати пристрій з підключенням до мережі, в якій розміщено вебсайт, та веббраузером.

4.2 Експлуатація програми

Для роботи з функціональністю адміністратора необхідно скористатися адміністративною панеллю, що доступна за стандартною адресою з додаванням admin вкінці. Її робота є типовою для Django, при цьому відкривається сторінка з якою можна отримати доступ до відповідних моделей даних, звідки доступні переліки даних за відповідною таблицею та можливості додавання і редагування.

У свою чергу всі інші користувачі працюють з програмою через одну і ту ж адресу. Застосунок містить можливість переходу між режимами, які відкриваються в залежності від ролі авторизованого користувача. Якщо він орендар, тобто звичайний користувач, то йому надається інтерфейс роботи над власними орендами. Цей інтерфейс дещо розширюється після авторизація, оскільки тоді стають доступними власні оренди.

Оскільки ключовою функцією для орендаря є оформлення замовлень на оренду скутерів, то кнопка Замовити винесена як окремий пункт в меню, що підсвічений синім кольором як кнопка.

З розділу Скутери користувач може обрати з переліку виробників потрібного, натиснути потрібного, потім обрати модель, після чого відкриється перелік з варіантами комплектації, а тоді можна відкрити окремо сторінку з варіантами комплектації скутера (рис. 4.1).

ЗАМОВИТИСКУТЕРИОРЕНДИВАРТІСТЬПРОФІЛЬВИЙТИ

VESPA VXL 150

Назад



Характеристики

Двигун

Опис: Одноциліндровий 4-тактний, з повітряним охолодженням

Потужність: 149.5

Максимальна швидкість: 90 км/г

Паливо

Витрати: 2 л

Пальне: Бензин А-92

Права

Категорія: А

Вік: 18

Опис

VESPA відома і омріяна, і якщо ти про неї коли-небудь чув, то це точно буде твій вибір. Зовнішній вигляд, прекрасна маневреність, добрі враження від водіння.

Середня оцінка: 9.5

Вартість

Замовити

Тривалість	Вимір	Вартість	При перевищенні
1	година	100 грн.	200 грн./год.
2	година	190 грн.	250 грн./год.
8	годин	700 грн.	200 грн./год.
1	доба	1000 грн.	200 грн./год.
7	доба	5000 грн.	800 грн./день

Рисунок 4.1 – Сторінка деталізації скутера

Зі сторінки деталізації можна повернутися назад за допомогою однойменної кнопки. На цій сторінці знаходиться фотографія скутера, набір

його характеристик (назва виробника, модель, опис двигуна, потужність, максимальна швидкість, тип і витрати пального, необхідна категорія прав, мінімальний вік, необхідний для керування цим скутером, опис скутера), також надається інформація про вартість оренди скутера, де виводяться всі можливі варіанти. Нижче також виводяться всі варіанти страхування скутера. За допомогою кнопки **Замовити** можна перейти до оформлення замовлення оренди саме даного скутера. У такому випадку не доведеться окремо обирати одразу модель. Для менеджера дана сторінка виглядає розширено, адже вона містить також перелік всіх екземплярів скутерів, які відповідають цьому варіанту скутера в відділенні, в якому працює менеджер.

З розділу **Вартість орендар** може отримати інформацію про всі варіанти вартості оренди для різних категорій скутерів за вартістю.

Менеджер має дещо інший варіант інтерфейсу, що відкривається після його авторизації (рис. 4.2).

ЗАМОВИТИ

ВИДАТИ

ОРЕНДИ

СКУТЕРИ

АБОНЕМЕНТИ

ВАРТІСТЬ

ПРОФІЛЬ

ВИХІД

ДОСТУПНІ ЗАМОВЛЕННЯ

Пошук

Всі

Пошук

Номер оренди

Номер телефону

Номер скутера

Модель скутера

Не раніше

Не пізніше

Всі

ДД.ММ.ГГГГ

ДД.ММ.ГГГГ

Шукати

Номер	Орендар	Стан	Дати	Вартість	Сплачено			
98	Сурядний Ігор	Нове	24.06.22 10:00 - 25.06.22 10:00	350 грн.	0 грн.	Відкрити	Видати	Скасувати
97	Єфимов Іван	Нове	21.06.22 14:00 - 21.06.22 18:00	80 грн.	0 грн.	Відкрити	Видати	Скасувати
95	Сурядний Ігор	Видане	14.06.22 15:00 - 14.06.22 18:00	150 грн.	150 грн.	Відкрити	Повернути	Проблема
94	Черненко Сергій	Видане	14.06.22 10:00 - 15.06.22 10:00	500 грн.	0 грн.	Відкрити	Повернути	Проблема
85	Сурядний Ігор	Завершено	13.06.22 15:30 - 13.06.22 18:53	210 грн.	210 грн.	Відкрити		
83	Черненко Сергій	Завершено	13.06.22 12:00 - 13.06.22 12:51	210 грн.	210 грн.	Відкрити		

Рисунок 4.2 – Керування орендами скутерів менеджером

Зрозуміло, що для менеджера процес керування орендами скутерів є ключовим. Тому для менеджера в окремі кнопки окрім замовлення одразу винесено кнопку Видати можливість видання скутерів за попереднім замовленням або без нього.

Після відкриття розділу Оренди менеджеру надається перелік оренд, якими можна керувати. Кнопки Пошук і Всі дозволяють першочергово керувати саме відображенням оренд. Варіант Всі дозволяє саме за порядком відобразити всі оренди скутерів. Варіант Пошук дозволяє виконати пошук серед всіх оренд або відобразити всі оренди за відділенням, в якому працює менеджер.

Менеджер може визначити пошук за такими фільтрами:

- номер оренди, якщо чітко відомий номер виконаної оренди;
- номер телефону орендаря, що дозволяє за заданим номером телефону орендаря одразу знайти всі виконані ним замовлення: не потрібні в такому випадку ні прізвище орендаря, ні його логін;
- номер скутера, якщо чітко відомо його ідентифікатор: потрібно відслідкувати саме за конкретним скутером і відомо, як він ідентифікується в системі;
- модель скутера;
- інтервал дат, між якими відбулася оренда.

Таблиця з орендами містить перелік всіх оренд, видаючи за ними номер, прізвище та ім'я орендаря, стан оренди, дати оренди (початку і завершення, якщо скутер повернуто, початку і запланованого завершення, якщо це більш ранні стани), вартість оренди, сплачену суму.

Кнопки керування залежать від стану оренди, але кнопка Відкрити відображається завжди, надаючи можливість відкрити повну сторінку окремого замовлення.

Менеджер може визначити повернення скутера з оренди, визначити, що скутер так і не був повернутися, видати скутер або скасувати оренду, якщо скутер ще не був виданий.

Якщо оренда вже завершена, то менеджер зможе тільки переглянути повністю оренду.

Розділ Скутери дозволяє переглянути переліки моделей, виробників, комплектаций подібно до орендаря, а також дозволяє додати інформацію про новий скутер. Для цього використовується кнопка додати при відображенні всіх скутерів. Форма додавання скутера представлена на рис. 4.3.

ДОДАТИ СКУТЕР

Виробник
Оберіть

Модель
Оберіть

Двигун

Пальне
Бензин

Витрати
на 100 км

Фотографія

Потужність
на 100 км

Максимальна швидкість
км/год

Потрібна категорія прав
A1

Вік
18

Опис

Зберегти

Рисунок 4.3 – Додавання нового скутера

Після визначення всіх параметрів скутера можна також завантажити фотографію (тоді потрібно натиснути на однойменну кнопку і з браузера обрати для завантаження файл, тоді шлях відобразиться біля кнопки, а за допомогою тієї ж кнопки можна завантажити файл на сервер).

Розділ Абонементи дозволяє вносити результати продажу абонементів на оренду скутерів. Також менеджер може переглядати вартість оренди за різними ціновими категоріями, а також аналогічно страхувань і абонементів. Відповідні кнопки дозволяють переходити між цими варіантами. Вносити ж конкретну вартість пакетів може адміністратор.

Таким чином виглядає логіка роботи з програмою користувачів.

Слід відзначити, що оцінювання доступно зі сторінки оренд орендаря. Звідти надається кнопка оцінити для тих скутерів, які були в оренді у орендаря.

4.3 Тестування програми

Тестування роботи проводилось на основі функціональних вимог до програми. Широкі сценарії використання програми були розроблені і таким чином встановлювалась їх відповідність. Під час тестування кожна функція перевірялась окремо. Використовувались варіанти оцінювання результатів тестування як пройдено повністю, пройдено частково та не пройдено. Усі функції були досліджені, перевірено очікуваний результат в порівнянні з отриманим. Тестування завершилось успіхом.

4.4 Висновки за розділом 4

У четвертому розділі зображено сценарії роботи менеджера і орендаря, описано основні можливості роботи з програмою для них та визначено загальні принципи, які дозволяють переміщуватися між ними у програмі. Тестування програми завершилось успіхом.

ВИСНОВКИ

У дипломній роботі розв'язано проблему розроблення програмного забезпечення підтримки оренди скутерів, для чого було проаналізовано програмні аналоги, які також надають засоби оренди скутерів, що дозволило сформулювати вимоги до програми і розглянути сценарії використання такої програми користувачами, що лягли в основному подальшого проєктування і реалізації програми, а також логіки взаємодії користувача з програмою під час опису програми. Окрім того було виконано вибір комплексу засобів для розробки, визначено структуру бази даних, логіку роботи програми, описано основні розроблені класи та представлено їх ієрархію у вигляді діаграми класів, реалізовано програму і представлено процес роботи користувачів з програмою.

У роботі було прийнято рішення щодо використання сполучення мови Python і фреймворку Django, а також мов JavaScript, HTML, CSS і системи керування базами даних MySQL для створення програми.

Зображено інтерфейси основних сценаріїв роботи менеджера і орендаря як основних користувачів з програмою, описано логіку їх взаємодії з програмою задля виконання тих функцій, які їм потрібні в їхній роботі. Створено типовий інтерфейс роботи адміністратора над наповненням програми даними.

Програмне забезпечення створено для надання через нього можливостей контролю за орендами зі сторони менеджерів компанії, для керування ними орендами, у той же час орендар також може користуватися програмою з метою створення попередніх замовлень, щоб бути впевненим у тому, що потрібний йому скутер буде в наявності, коли він прийде у відділення, а також для того, щоб відслідковувати власні оренди, включаючи ті, що вже були виконані, передзамовлені та ті, за якими він ще має виконати власні зобов'язання.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Stevens, P. Rising fuel costs are a massive problem for business and consumers – Here's why they're so high [Electronic resource] / P. Stevens. – Access mode : <https://www.cnbc.com/2022/05/19/fuel-is-a-problem-for-business-and-consumers-why-prices-are-so-high.html>.
2. Everything about rental electric scooters models [Electronic resource]. – Access mode : <https://electricscootering.com/rental-electric-scooters-models/>.
3. Is It Better To Rent Or To Own An Electric Scooter? [Electronic resource]. – Access mode : <https://medium.com/@ride.taur/rent-or-own-an-electric-scooter-acf7aa586f1c>.
4. Irfan, U. Electric scooters' sudden invasion of American cities, explained [Electronic resource] / U. Irfan. – Access mode : <https://www.vox.com/2018/8/27/17676670/electric-scooter-rental-bird-lime-skip-spin-cities>.
5. Paphelson, S. Dockless Scooters Gain Popularity And Scorn Across The U.S. [Electronic resource] / S. Paphelson. – Access mode : <https://www.npr.org/2018/08/29/643058414/dockless-scooters-gain-popularity-and-scorn-across-the-u-s>.
6. Roma Rent Scooter. Scooter and Bike in Rome [Electronic resource]. – Access mode : <https://www.romarentscooter.it/>.
7. Scooter, motorcycle, bicycle rental in 65+ countries | BikesBooking.com [Electronic resource]. – Access mode : <https://bikesbooking.com/en/>.
8. Прокат самокатів – Veliki.ua [Електрон. ресурс]. – Режим доступу : <https://veliki.ua/ua/prokat-samokativ.html>.
9. Django MVT [Electronic resource]. – Access mode : <https://www.javatpoint.com/django-mvt>.
10. Ramalho, L. Fluent Python [Text] : Clear, Concise, and Effective Programming / L. Ramalho. – Massachusetts : O'Reilly Media, 2022. – 1012 p.

11. Shaw, Z. Learn Python 3 the Hard Way [Text] : A Very Simple Introduction to the Terrifyingly Beautiful World of Computers and Code / Z. Shaw. – Boston, Massachusetts : Addison-Wesley Professional, 2017. – 320 p.

ДОДАТОК А
Технічне завдання

Вступ

Скутери дозволяють достатньо швидко переміщатися в місті, а коли пальне стає дорожчим і більш дефіцитним, скутери стають транспортним засобом, який ще більше привертає увагу. Оренда скутерів – важливий інструмент підтримки такої мобільності в місті в поточних сучасних умовах.

А.1 Підстави для розробки

Дипломна кваліфікаційна робота виконується на основі завдання, виданого за темою «Програмне забезпечення підтримки оренди скутерів» у відповідності з наказом № 102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка».

А.2 Призначення розробки

Програма призначена для підтримки роботи служби оренди скутерів та передзамовлення і стеження за власними орендами зі сторони орендарів.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Набір функціональних вимог, які має забезпечувати програма, включає:

- перегляд інформації про скутер;
- замовлення оренди скутера;
- визначення видачі скутера в оренду;
- скасування оренди;
- визначення повернення скутера;

- релокація скутера між відділеннями;
- визначення неповернення скутера;
- перегляд переліку оренд скутерів;
- оцінювання скутера;
- створення абонементу оренди;
- пошук серед оренд;
- реєстрація і авторизація орендарів;
- реєстрація адміністратором працівників;
- авторизація працівників;
- встановлення вартості пакетів оренди;
- встановлення вартості абонементів;
- встановлення вартості страхувань;
- занесення страхувань;
- занесення пакетів оренди;
- занесення варіантів абонементу;
- занесення і редагування екземпляра скутера;
- перегляд скутерів у відділенні;
- занесення моделей та виробників скутерів;
- занесення скутерів.

A.3.2 Умови експлуатації

Умовами експлуатації є виділення менеджера або менеджерів для реалізації внесення даних в систему. Менеджер може також виконувати роль адміністратора, але краще виділити для цього окремо системного адміністратора, який також буде керувати базою даних.

A.3.3 Вимоги до складу та параметрів технічних засобів

Для підтримки роботи програми необхідно, щоб апаратна частина

включала:

- процесор (4 ядра, не менше 2,1 ГГц);
- оперативну пам'ять (від 4 Гб);
- жорсткий диск з наявним вільним простором (від 1 Гб).

А.4 Порядок контролю та приймання

Дипломна кваліфікаційна робота має виконуватися у відповідності з визначеним календарним планом згідно встановлених завдань, їх строків і результатів виконання.

ДОДАТОК Б
Текст програми

```

from django.views.generic.edit import CreateView
from django.views.generic import ListView
from django.views.generic import UpdateView
from django.views.generic.detail import DetailView
from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
from .models import Scooter, Companyoffice,
ScooterConcrete, OfficeScooter, Scootermodel, Scootermaker,
Rental, BookedItem, RentalItem, StateManagement, Rentaler,
Companymanager, Companyoffice, Estimation
import datetime
from datetime import date

#скутери, які є у відділенні, варіант для орендаря
class OfficeScooters (ListView):
    model = Scooter
    template_name = 'office_scooters.html'

    def get_queryset(self):
        office = self.request.GET.get('office')
        comoffice = Companyoffice.objects.get(id = office)
        scooters =
Scooter.objects.filter(officescooters__office = comoffice)
        scooters =
scooters.order_by('scootermodel__title')
        return scooters

    def get_context_data(self, **kwargs):
        scooters = super(OfficeScooters,
self).get_context_data(**kwargs)
        office = self.request.GET.get('office')
        scooters ['offices'] =
Companyoffice.objects.get(id = office)
        return scooters

#скутери (комплектації), які є у відділенні, в якому працює
менеджер, який увійшов в програму
class ManagerOfficeScooters (ListView):

```

```

model = Scooter
template_name = 'man_office_scooters.html'

def get_queryset(self):
    manager = self.request.user
    compmanager = Companymanager.objects.get(id =
manager)

    comoffice = compmanager.office
    scooters =
Scooter.objects.filter(officescooters__office = comoffice)
    scooters =
scooters.order_by('scootermode__title')
    return scooters

#екземпляри скутерів, які є у відділенні, в якому працює
менеджер, який увійшов в програму
class ManagerOfficeScooterConcretes (ListView):
    model = ScooterConcrete
    template_name = 'man_office_scooterconcretes.html'

    def get_queryset(self):
        manager = self.request.user
        compmanager = Companymanager.objects.get(id =
manager)

        comoffice = compmanager.office
        scooters =
Scooter.objects.filter(officescooters__office = comoffice)
        concretes =
ScooterConcrete.objects.filter(officescooter__in = scooters)
        concretes = concretes.order_by('-id')
        return concretes

#занесення екземпляра скутера
class ScooterConcreteCreate (LoginRequiredMixin,
CreateView):
    model = ScooterConcrete
    fields = ['statenumber', 'vinnumber', 'state', 'color']
    template_name = 'newscooterex.html'

    def get_success_url(self):

```

```

        return reverse_lazy('scooterexview', kwargs={'pk':
self.object.pk}))

    def get_context_data(self, **kwargs):
        officescooter = get_object_or_404(OfficeScooter,
id=self.kwargs['officescooter'])
        kwargs['officescooter'] = officescooter
        office = get_object_or_404(Office,
id=self.kwargs['office'])
        kwargs['office'] = office
        scooter = get_object_or_404(Scooter,
id=self.kwargs['scooter'])
        kwargs['scooter'] = scooter
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        if self.kwargs['officescooter']:
            self.officescooter =
get_object_or_404(OfficeScooter,
id=self.kwargs['officescooter'])
        else:
            officec = get_object_or_404(Office,
id=self.kwargs['office'])
            scooterc = get_object_or_404(Scooter,
id=self.kwargs['scooter'])
            self.officescooter =
OfficeScooter.objects.get(office = officec, scooter = scooterc)
        return super().form_valid(form)

#редагування екземпляра скутера
class ScooterConcreteUpdate (LoginRequiredMixin,
UpdateView):
    model = ScooterConcrete
    fields = ['statenumber', 'vinnumber', 'state', 'color']
    template_name = 'renewscooterex.html'

    def get_success_url(self):
        return reverse_lazy('scooterexview', kwargs={'pk':
self.object.pk}))

```

```

    def get_context_data(self, **kwargs):
        officescooter = get_object_or_404(OfficeScooter,
id=self.kwargs['officescooter'])
        kwargs['officescooter'] = officescooter
        office = get_object_or_404(Office,
id=self.kwargs['office'])
        kwargs['office'] = office
        scooter = get_object_or_404(Scooter,
id=self.kwargs['scooter'])
        kwargs['scooter'] = scooter
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        if self.kwargs['officescooter']:
            self.officescooter =
get_object_or_404(OfficeScooter,
id=self.kwargs['officescooter'])
        else:
            officec = get_object_or_404(Office,
id=self.kwargs['office'])
            scooterc = get_object_or_404(Scooter,
id=self.kwargs['scooter'])
            self.officescooter =
OfficeScooter.objects.get(office = officec, scooter = scooterc)
        return super().form_valid(form)

#занесення скутера
class ScooterCreate (LoginRequiredMixin, CreateView):
    model = Scooter
    fields = ['engine', 'fuel', 'capacity', 'fueltype',
'speed', 'driverage', 'license', 'description', 'deposit']
    template_name = 'newscooter.html'

    def get_success_url(self):
        return reverse_lazy('scooterview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        scootermode = get_object_or_404(Scootermode,
id=self.kwargs['scootermode'])

```

```

        kwargs['scootermodel'] = scootermodel
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.scootermodel =
get_object_or_404(Scootermodel, id=self.kwargs['scootermodel'])
        return super().form_valid(form)

#редагування скутера
class ScooterUpdate (LoginRequiredMixin, UpdateView):
    model = Scooter
    ['engine', 'fuel', 'capacity', 'fueltype', 'speed',
'driverage', 'license', 'description', 'deposit']
    template_name = 'renewscooter.html'

    def get_success_url(self):
        return reverse_lazy('scooterview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        scootermodel = get_object_or_404(Scootermodel,
id=self.kwargs['scootermodel'])
        kwargs['scootermodel'] = scootermodel
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.scootermodel =
get_object_or_404(Scootermodel, id=self.kwargs['scootermodel'])
        return super().form_valid(form)

#перегляд скутера менеджером
class ManagerScooterDetail (DetailView):
    template_name = 'man_scooter.html'
    model = Scooter

    def get_context_data(self, **kwargs):
        scooter = super().get_context_data(**kwargs)
        officec = get_object_or_404(Office,
id=self.kwargs['office'])
        kwargs['office'] = officec

```

```

        sc = OfficeScooter.objects.filter(office =
officeec, scooter = self.object.pk)
        scs =
ScooterConcrete.objects.filter(officescooter__in = sc)
        scooter ['concretes'] = scs
        return scooter

#перегляд скутера
class ScooterDetail (DetailView):
    template_name = 'scooter.html'
    model = Scooter

#перегляд екземпляра скутера
class ScooterConcreteDetail (DetailView):
    template_name = 'scooter_concrete.html'
    model = ScooterConcrete

    def get_context_data(self, **kwargs):
        scooter = super().get_context_data(**kwargs)
        rentals = Rental.objects.filter(items__contains =
self.object.pk)
        rentals = rentals.order_by('-id')
        scooter ['rentals'] = rentals
        return scooter

#занесення моделі скутера
class ScootermodeCreate (LoginRequiredMixin, CreateView):
    model = Scootermode
    fields = ['title']
    template_name = 'newscootermode.html'

    def get_success_url(self):
        return reverse_lazy('scootermodeview',
kwargs={'pk': self.object.pk})

    def get_context_data(self, **kwargs):
        scootermaker = get_object_or_404(Scootermaker,
id=self.kwargs['scootermaker'])
        kwargs['scootermaker'] = scootermaker
        return super().get_context_data(**kwargs)

```

```

        def form_valid(self, form):
            self.scootermaker =
get_object_or_404(Scootermaker, id=self.kwargs['scootermaker'])
            return super().form_valid(form)

#редагування моделі скутера
class ScootermodelUpdate (LoginRequiredMixin, UpdateView):
    model = Scootermodel
    fields = ['title']
    template_name = 'renewmodel.html'

    def get_success_url(self):
        return reverse_lazy('scootermodelview',
kwargs={'pk': self.object.pk})

    def get_context_data(self, **kwargs):
        scootermaker = get_object_or_404(Scootermaker,
id=self.kwargs['scootermaker'])
        kwargs['scootermaker'] = scootermaker
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.scootermaker =
get_object_or_404(Scootermaker, id=self.kwargs['scootermaker'])
        return super().form_valid(form)

#перегляд моделі скутера
class ScootermodelDetail (DetailView):
    template_name = 'scootermodel.html'
    model = Scootermodel

    def get_context_data(self, **kwargs):
        scooter = super().get_context_data(**kwargs)
        scooters = Scooter.objects.filter(scootermodel =
self.object.pk)
        scooters = scooters.order_by('title')
        scooter ['scooters'] = scooters
        return scooter

```

```

#занесення виробника скутера
class ScootermakerCreate (LoginRequiredMixin, CreateView):
    model = Scootermaker
    fields = ['title']
    template_name = 'newscootermaker.html'

#редагування виробника скутера
class ScootermodeUpdate (LoginRequiredMixin, UpdateView):
    model = Scootermaker
    fields = ['title']
    template_name = 'renewscootermaker.html'

#перегляд виробника скутера
class ScootermakerDetail (DetailView):
    template_name = 'scooternaker.html'
    model = Scootermaker

    def get_context_data(self, **kwargs):
        scooter = super().get_context_data(**kwargs)
        scooters =
Scootermode.objects.filter(scootermaker = self.object.pk)
        scooters = scooters.order_by('title')
        scooter ['models'] = scooters
        return scooter

#перегляд оренд, які належать орендарю, що увійшов у
програму
class RentalerRentals (ListView):
    model = Rental
    template_name = 'rentaler_rentals.html'

    def get_queryset(self):
        rentaler = self.request.user
        rentalerc = Rentaler.objects.get(id = rentaler)
        rentals = Rental.objects.filter(rentaler =
rentalerc)

        rentals = rentals.order_by('-booked')
        return rentals

```

```

#перегляд оренд у відділенні менеджера, який увійшов у
програму
class ManagerRentals (ListView):
    model = Rental
    template_name = 'manager_rentals.html'

    def get_queryset(self):
        manager = self.request.user
        managerc = Companymanager.objects.get(id =
manager)

        off = manager.office
        rentals = Rental.objects.filter(office = off)
        rentals = rentals.order_by('-booked')
        return rentals

#перегляд деталей оренди
class RentalDetail (DetailView):
    template_name = 'rental.html'
    model = Rental

#замовлення оренди менеджером
class RentalCreate (LoginRequiredMixin, CreateView):
    model = Rental
    fields = ['start', 'planned_end', 'office', 'deposit',
'prepayed']
    template_name = 'newsrental.html'

    def get_success_url(self):
        return reverse_lazy('rentalview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        rentaler = get_object_or_404(Rentaler,
id=self.kwargs['rentaler'])
        kwargs['rentaler'] = rentaler
        ins = get_object_or_404(Insurance,
id=self.kwargs['insurance'])
        kwargs['insurance'] = ins
        return super().get_context_data(**kwargs)

```

```

def form_valid(self, form):
    self.rentaler = get_object_or_404(Rentaler,
id=self.kwargs['rentaler'])
    manager = self.request.user
    managerc = Companymanager.objects.get(id =
manager.id)

    self.state = 'B'
    commentc = self.kwargs['comment']
    items = self.kwargs['items']
    items = items.split(',')
    quant = self.kwargs['quantities']
    quant = quant.split(',')
    ins = self.kwargs['insurance']
    ins = Insurance.objects.get(id = ins)
    k = 0
    for item in items:
        item = Scooter.objects.get(id = item)
        BookedItem.objects.create(officescooter =
self.office, rental = self.object.pk, quantity = quant[k],
insurance = ins)

        k = k + 1
    today = date.today()
    StateManagement.objects.create(rental =
self.object.pk, state = 'B', companymanager = managerc, comment
= commentc, saved = today)
    items = self.items
    pr = 0
    for item in self.items:
        pr = pr + item.quantity * scooterprice(item,
self.start, self.planned_end, self.rentaler)
        pr = pr + item.quantity *
insuranceprice(item, ins, self.start, self.planned_end,
self.rentaler)

    self.price = pr
    return super().form_valid(form)

```

#редагування оренди менеджером

```

class RentalUpdate (LoginRequiredMixin, UpdateView):
    model = Rental

```

```

        fields = ['start', 'planned_end', 'office', 'deposit',
'prepayed']
        template_name = 'updatesrental.html'

    def get_success_url(self):
        return reverse_lazy('rentalview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        rentaler = get_object_or_404(Rentaler,
id=self.kwargs['rentaler'])
        kwargs['rentaler'] = rentaler
        ins = get_object_or_404(Insurance,
id=self.kwargs['insurance'])
        kwargs['insurance'] = ins
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.rentaler = get_object_or_404(Rentaler,
id=self.kwargs['rentaler'])
        items = self.kwargs['items']
        items = items.split(',')
        quant = self.kwargs['quantities']
        quant = quant.split(',')
        allitems = []
        man = self.request.user
        manager = Companymanager.objects.filter(id =
man.id)

        k = 0
        ins = self.kwargs['insurance']
        ins = Insurance.objects.get(id = ins)
        for item in items:
            scooterc = Scooter.objects.get(id = item)
            os = OfficeScooter.objects.get (office =
manager.office, scooter = scooterc)
            if item not in self.bitems:
                BookedItem.objects.create(officescooter
= self.office, rental = self.object.pk, quantity = quant[k],
insurance = ins)
            elif item.quantity != quant[k]:

```

```

        bi =
BookedItem.objects.filter(officescooter = os, rental =
self.object.pk)

        bi.quantity = quant[k]
        bi.save()
        allitems.append(item)
        k = k + 1
    for item in self.bitems:
        if item not in allitems:
            BookedItem.objects.filter(id =
item.id).delete()
        pr = 0
        for item in self.items:
            pr = pr + item.quantity * scooterprice(item,
self.start, self.planned_end, self.rentaler)
            pr = pr + item.quantity *
insuranceprice(item, ins, self.start, self.planned_end,
self.rentaler)
        self.price = pr
        return super().form_valid(form)

#вилучення скутера з замовлення
class BookedItemDelete (DeleteView):
    model = BookedItem

#вилучення скутера з оренди
class RentalItemDelete (DeleteView):
    model = RentalItem

#замовлення оренди орендарем
class RentalerRentalCreate (LoginRequiredMixin,
CreateView):
    model = Rental
    fields = ['start', 'planned_end']
    template_name = 'newrental.html'

    def get_success_url(self):
        return reverse_lazy('rentalview', kwargs={'pk':
self.object.pk})

```

```

def get_context_data(self, **kwargs):
    office = get_object_or_404(Office,
id=self.kwargs['office'])
    kwargs['office'] = office
    ins = get_object_or_404(Insurance,
id=self.kwargs['insurance'])
    kwargs['insurance'] = ins
    return super().get_context_data(**kwargs)

def form_valid(self, form):
    self.office = get_object_or_404(Office,
id=self.kwargs['office'])
    ins = self.kwargs['insurance']
    ins = Insurance.objects.get(id = ins)
    rentaler = self.request.user
    rentaler = Rentaler.objects.get(id = rentaler.id)
    self.rentaler = rentaler
    self.state = 'A'
    today = date.today()
    self.booked = today
    items = self.kwargs['items']
    items = items.split(',')
    quant = self.kwargs['quantities']
    quant = quant.split(',')
    k = 0
    for item in items:
        item = Scooter.objects.get(id = item)
        BookedItem.objects.create(officescooter =
self.office, rental = self.object.pk, quantity = quant[k],
insurance = ins)
        k = k + 1
    pr = 0
    for item in self.items:
        pr = pr + item.quantity * scooterprice(item,
self.start, self.planned_end, self.rentaler)
        pr = pr + item.quantity *
insuranceprice(item, ins, self.start, self.planned_end,
self.rentaler)
    self.price = pr
    return super().form_valid(form)

```

```

#видача скутера в оренду
class RentalGive (LoginRequiredMixin, UpdateView):
    model = Rental
    fields = ['planned_end']
    template_name = 'rentalgive.html'

    def get_success_url(self):
        return reverse_lazy('rentalview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        office = get_object_or_404(Office,
id=self.kwargs['office'])
        kwargs['office'] = office
        ins = get_object_or_404(Insurance,
id=self.kwargs['insurance'])
        kwargs['insurance'] = ins
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        if self.kwargs['start']:
            start =
datetime.datetime.strptime(self.kwargs['start'],
"%d.%m.%Y").date()
            self.start = start
        else:
            start = date.today()
            self.start = start
        self.office = get_object_or_404(Office,
id=self.kwargs['office'])
        manager = self.request.user
        managerc = Companymanager.objects.get(id =
manager.id)
        today = date.today()
        ins = self.kwargs['insurance']
        ins = Insurance.objects.get(id = ins)
        commentc = self.kwargs['comment']
        StateManagement.objects.create(rental =
self.object.pk, state = 'C', companymanager = managerc, comment
= commentc, saved = today)

```

```

        items = self.kwargs['items']
        items = items.split(',')
        k = 0
        for item in items:
            item = Scooterconcrete.objects.get(id = item)
            RentalItem.objects.create(scooterconcrete =
item, rental = self.object.pk, insurance = ins)
            k = k + 1
        pr = 0
        for item in self.bitems:
            pr = pr + item.quantity * scooterprice(item,
self.start, self.planned_end, self.rentaler)
            pr = pr + item.quantity *
insuranceprice(item, ins, self.start, self.planned_end,
self.rentaler)
        self.price = pr
        return super().form_valid(form)

#повернення скутера з оренди
class RentalReturn (LoginRequiredMixin, UpdateView):
    model = Rental
    template_name = 'rentalreturn.html'

    def get_success_url(self):
        return reverse_lazy('rentalview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        manager = self.request.user
        managerc = Companymanager.objects.get(id =
manager.id)
        today = date.today()
        commentc = self.kwargs['comment']
        StateManagement.objects.create(rental =
self.object.pk, state = 'D', companymanager = managerc, comment
= commentc, saved = today)
        if self.kwargs['end']:
            end =
datetime.datetime.strptime(self.kwargs['end'],
"%d.%m.%Y").date()

```

```

        self.end = end
    else:
        end = date.today()
        self.end = end
    return super().form_valid(form)

#неповернення скутера з оренди
class RentalProblem (LoginRequiredMixin, UpdateView):
    model = Rental
    template_name = 'rentalproblem.html'

    def get_success_url(self):
        return reverse_lazy('rentalview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        manager = self.request.user
        managerc = Companymanager.objects.get(id =
manager.id)

        today = date.today()
        commentc = self.kwargs['comment']
        StateManagement.objects.create(rental =
self.object.pk, state = 'F', companymanager = managerc, comment
= commentc, saved = today)
        return super().form_valid(form)

#скасування оренди
class RentalCancel (LoginRequiredMixin, UpdateView):
    model = Rental
    template_name = 'rentalcancel.html'

    def get_success_url(self):
        return reverse_lazy('rentalview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        manager = self.request.user
        managerc = Companymanager.objects.get(id =
manager.id)

        today = date.today()

```

```

        commentc = self.kwargs['comment']
        StateManagement.objects.create(rental =
self.object.pk, state = 'E', companymanager = managerc, comment
= commentc, saved = today)
        return super().form_valid(form)

#скасування оренди самим орендарем
class ReRentalCancel (LoginRequiredMixin, UpdateView):
    model = Rental
    template_name = 'rentalercancel.html'

    def get_success_url(self):
        return reverse_lazy('rentalview', kwargs={'pk':
self.object.pk})

    def form_valid(self, form):
        rentaler = self.request.user
        rentaler = Rentaler.objects.get(id = rentaler.id)
        rental = Rental.objects.get(id = self.object.pk)
        if rental.rentaler == rentaler and rental.state ==
'A':

            self.state = 'E'
            return super().form_valid(form)

#оцінювання скутера
class EstimationCreate (LoginRequiredMixin, CreateView):
    model = Estimation
    fields = ['result']
    template_name = 'estimate.html'

    def get_success_url(self):
        return reverse_lazy('scooterview', kwargs={'pk':
self.object.pk})

    def get_context_data(self, **kwargs):
        scooter = get_object_or_404(Scooter,
id=self.kwargs['scooter'])
        kwargs['scooter'] = scooter
        return super().get_context_data(**kwargs)

```

```

def form_valid(self, form):
    rentaler = self.request.user
    rentaler = Rentaler.objects.get(id = rentaler)
    self.rentaler = rentaler
    return super().form_valid(form)

#редагування моделі скутера
class ScootermodeUpdate (LoginRequiredMixin, UpdateView):
    model = Scootermode
    fields = ['title']
    template_name = 'renewmodel.html'

    def get_success_url(self):
        return reverse_lazy('scootermodeview',
kwargs={'pk': self.object.pk})

    def get_context_data(self, **kwargs):
        scootermaker = get_object_or_404(Scootermaker,
id=self.kwargs['scootermaker'])
        kwargs['scootermaker'] = scootermaker
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.scootermaker =
get_object_or_404(Scootermaker, id=self.kwargs['scootermaker'])
        return super().form_valid(form)

#оновлення замовлення
class BookedItemUpdate (LoginRequiredMixin, UpdateView):
    model = BookedItem
    fields = ['quantity', 'rental']
    template_name = 'bookitemupdate.html'

    def get_success_url(self):
        return reverse_lazy('rentalview', kwargs={'pk':
self.object.rental.pk})

#пошук оренди
class RentalSearch (ListView):
    model = Rental

```

```

template_name = 'search.html'

def get_queryset(self):
    phone = self.request.GET.get('phone')
    period1 = self.request.GET.get('period1')
    period1 = datetime.datetime.strptime(period1,
"%d.%m.%Y").date()
    period2 = self.request.GET.get('period2')
    period2 = datetime.datetime.strptime(period2,
"%d.%m.%Y").date()
    rental_id = self.request.GET.get('number')
    scooter = self.request.GET.get('scooter')
    scootrex = self.request.GET.get('scootrex')
    concretes = ScooterConcrete.objects.all()
    if rental_id:
        concretes = ScooterConcrete.objects.get(id =
rental_id)
    else:
        if scootrex:
            concretes =
concretes.filter(Q(items__contains = scootrex) |
Q(bitems__contains = scootrex))
        elif scooter:
            scooterconcretes =
ScooterConcrete.objects.filter(officescooter__scooter = scooter)
            concretes =
concretes.filter(reduce(operator.and, (Q(items__contains = x)
for x in scooterconcretes) | Q(bitems__contains = scooter)))
        if period1:
            concretes = concretes.filter(start__gte
= period1)
        if period2:
            concretes = concretes.filter(start__gte
= period2)
        if phone:
            concretes =
concretes.filter(rentaler__phone = phone)
    return concretes.order_by('-id')

def get_context_data(self, **kwargs):

```

```

        scooters = super(RentalSearch,
self).get_context_data(**kwargs)
        scooterex = self.request.GET.get('scooterex')
        scooters ['scooterex'] =
ScooterConcrete.objects.get(id = scooterex)
        scoot = self.request.GET.get('scooter')
        scooters ['scooter'] = Scooter.objects.get(id =
scoot)

        return scooters

#розміщення скутерів
class ScooterOfficeCreate (LoginRequiredMixin, CreateView):
    model = OfficeScooter
    fields = ['quantity', 'accessed_quantity']
    template_name = 'locate.html'

    def get_success_url(self):
        return reverse_lazy('scooterview', kwargs={'pk':
self.object.scooter.pk})

    def get_context_data(self, **kwargs):
        office = get_object_or_404(Office,
id=self.kwargs['office'])
        kwargs['office'] = office
        scooter = get_object_or_404(Scooter,
id=self.kwargs['scooter'])
        kwargs['scooter'] = scooter
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.office = get_object_or_404(Office,
id=self.kwargs['office'])
        self.scooter = get_object_or_404(Scooter,
id=self.kwargs['scooter'])
        return super().form_valid(form)

#релокація скутерів
class ScooterOfficeUpdate (LoginRequiredMixin, UpdateView):
    model = OfficeScooter
    fields = ['quantity', 'accessed_quantity']

```

```

        template_name = 'relocate.html'

    def get_success_url(self):
        return reverse_lazy('scooterview', kwargs={'pk':
self.object.scooter.pk})

    def get_context_data(self, **kwargs):
        office = get_object_or_404(Office,
id=self.kwargs['office'])
        kwargs['office'] = office
        scooter = get_object_or_404(Scooter,
id=self.kwargs['scooter'])
        kwargs['scooter'] = scooter
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.office = get_object_or_404(Office,
id=self.kwargs['office'])
        self.scooter = get_object_or_404(Scooter,
id=self.kwargs['scooter'])
        return super().form_valid(form)

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

**«ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПІДТРИМКИ ОРЕНДИ
СКУТЕРІВ»**

ДИПЛОМНА КВАЛІФІКАЦІЙНА РОБОТА
БАКАЛАВРА

студент групи КНТ-138
І.Ю. Сурядний

к.т.н., доцент
О.О. Степаненко

2022

Рисунок В.1 – Слайд № 1

Об'єкт, предмет, мета роботи

Об'єкт роботи – процес оренди скутерів.

Предмет роботи – програмне забезпечення підтримки оренди скутерів.

Мета роботи – розробити програмне забезпечення підтримки оренди скутерів для забезпечення взаємодії між орендарями і службою оренди.

2

Рисунок В.2 – Слайд № 2

Завдання роботи

- ✓ аналіз існуючого програмного забезпечення підтримки оренди скутерів;
- ✓ створення бази даних для роботи служби оренди скутерів;
- ✓ проектування програми;
- ✓ створення програми.

3

Рисунок В.3 – Слайд № 3

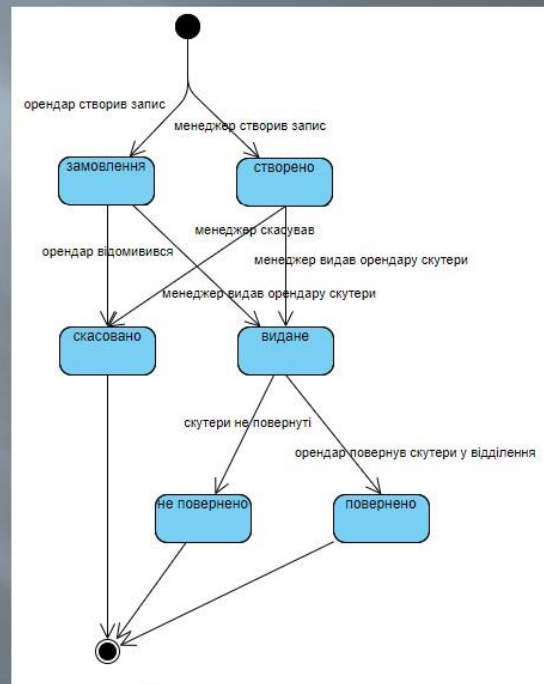
Вибір фреймворка веброзробки

Показник	Django	Laravel
Повний стек веброзробки	Так	Так
Мова програмування	Python	PHP
Перевага зручності розробки мовою	Так	Ні
Вільні ліцензії розповсюдження	Так	Так
Кросплатформність	Так	Так
Підтримка об'єктно-реляційного відображення	Так	Так
Перевага швидкості роботи застосунків	Так	Ні

4

Рисунок В.4 – Слайд № 4

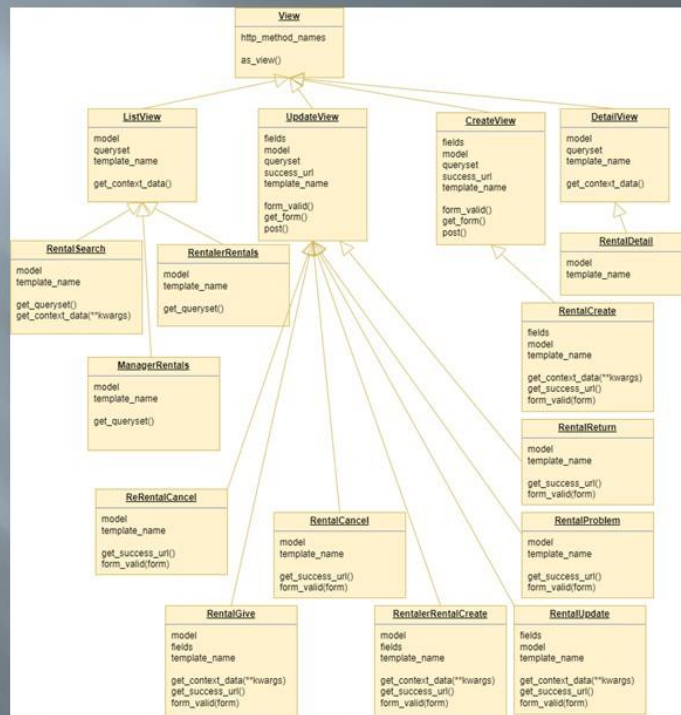
Діаграма станів об'єкта оренди



7

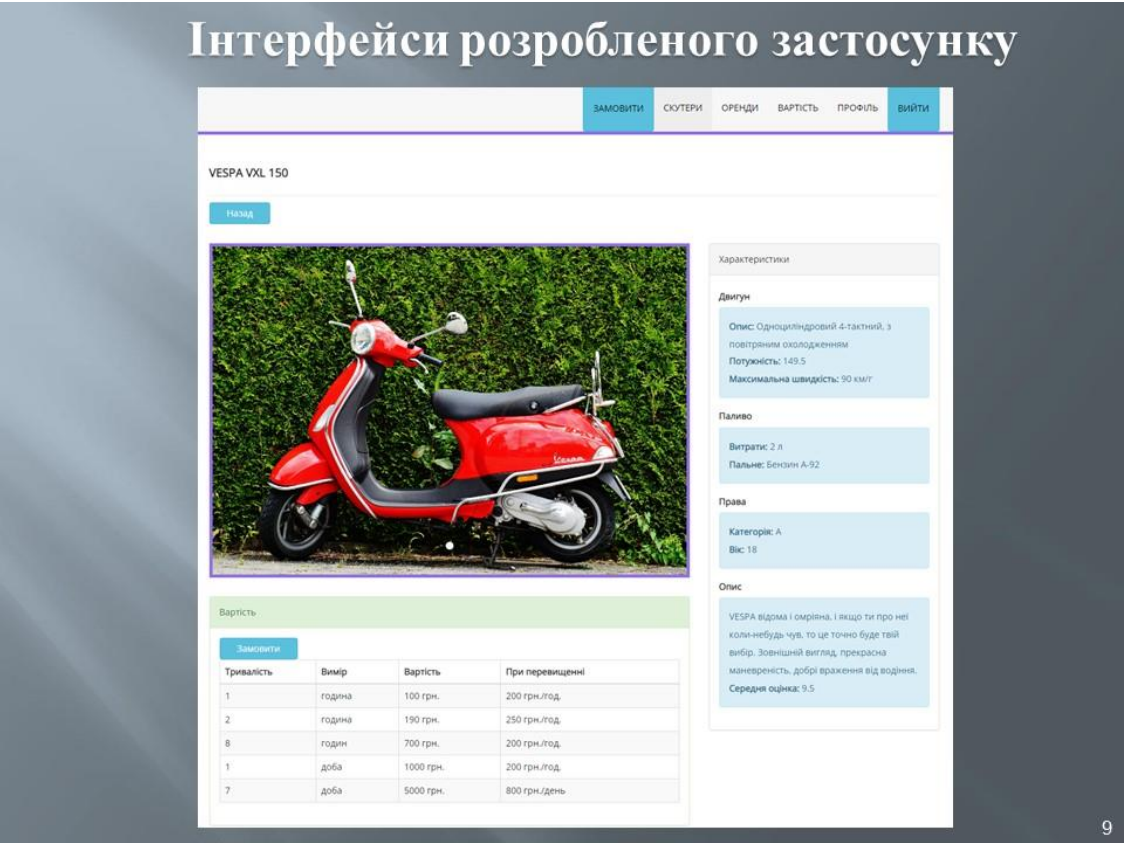
Рисунок В.7 – Слайд № 7

Діаграма класів



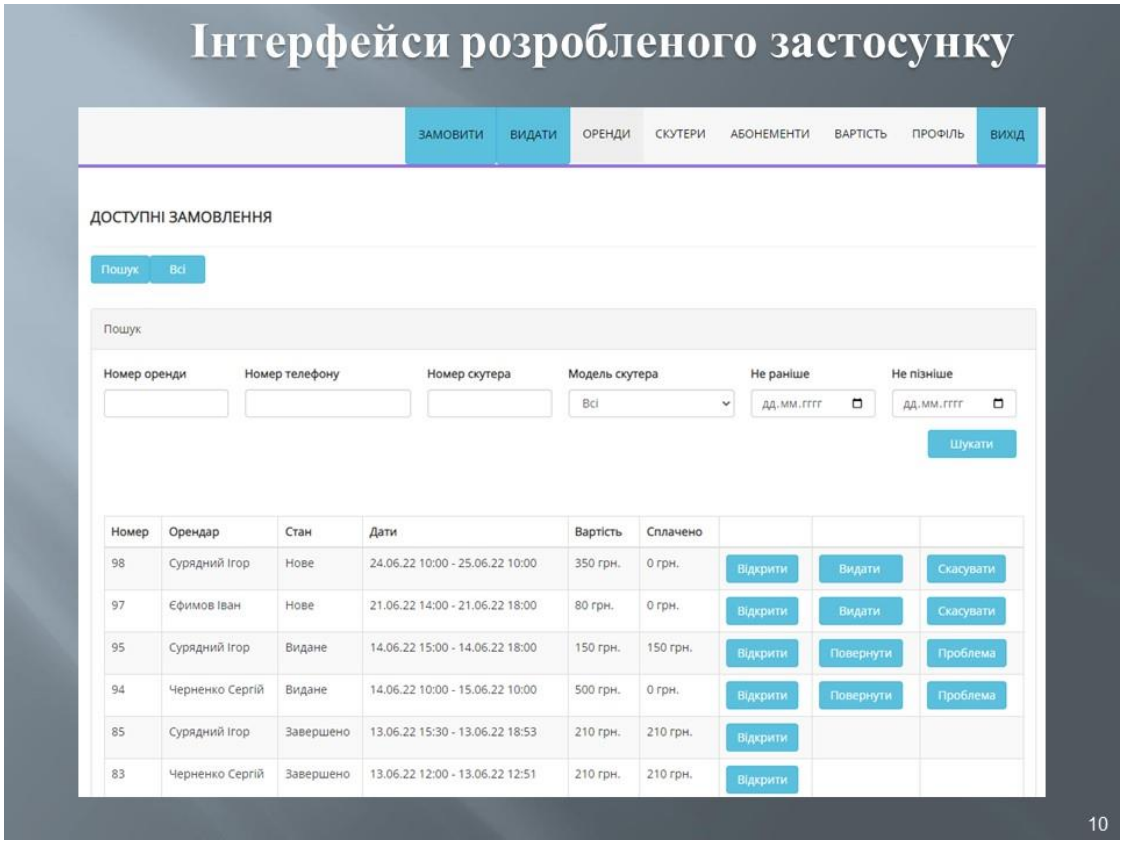
8

Рисунок В.8 – Слайд № 8



9

Рисунок В.9 – Слайд № 9



Інтерфейси розробленого застосунку

11

Рисунок В.11 – Слайд № 11

Висновки

У дипломній роботі розв'язано проблему розроблення програмного забезпечення підтримки оренди скутерів, для чого було проаналізовано програмні аналоги, які також надають засоби оренди скутерів, що дозволило сформулювати вимоги до програми і розглянути сценарії використання такої програми користувачами, що лягли в основному подальшого проектування і реалізації програми, а також логіки взаємодії користувача з програмою під час опису програми. Окрім того було виконано вибір комплексу засобів для розробки, визначено структуру бази даних, логіку роботи програми, описано основні розроблені класи та представлено їх ієрархію у вигляді діаграми класів, реалізовано програму і представлено процес роботи користувачів з програмою.

Програмне забезпечення створено для надання через нього можливостей контролю за орендами зі сторони менеджерів компанії, для керування ними орендами, у той же час орендар також може користуватися програмою з метою створення попередніх замовлень, щоб бути впевненим у тому, що потрібний йому скутер буде в наявності, коли він прийде у відділення, а також для того, щоб відслідковувати власні оренди, включаючи ті, що вже були виконані, передзамовлені та ті, за якими він ще має виконати власні зобов'язання.

12

Рисунок В.12 – Слайд № 12