

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 проекту (роботи)

бакалавра

(ступінь вищої освіти)

на тему ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ПОШУКУ ПОПУТНИКІВ ДЛЯ
ПОДОРОЖЕЙ АВТОМОБІЛЕМ
SOFTWARE FOR CAR TRAVEL COMPANION SEARCH

Виконав: студент(ка) 4 курсу, групи КНТ-129сп
Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Сердюк О.В.

(прізвище та ініціали)

Керівник Скруський С.Ю.

(прізвище та ініціали)

Рецензент Голуб Т.В.

(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)

Сердюка Олександра Валерійовича
(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Програмне забезпечення пошуку попутників для
подорожей автомобілем
Software for Car Travel Companion Search

керівник проєкту (роботи) Скрупський Степан Юрійович, 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 проєкту (роботи)

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

7. Дата видачі завдання «11» квітня 2022 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Постановка завдання роботи	1 тиждень	Завдання, ТЗ
2	Аналіз 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)

Сердюк О.В.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис)

Скрупський С.Ю.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи бакалавра: 84 с., 26 рис., 16 табл., 3 дод., 10 джерел.

ПОДОРОЖ АВТОМОБІЛЕМ, МАРШРУТ ПОДОРОЖІ, ПОПУТНИК, ПАСАЖИР, ВОДІЙ, ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ, БАЗА ДАНИХ.

Об'єкт дослідження – процес спільного подорожування автомобілем. Предмет дослідження – програмне забезпечення пошуку попутників для подорожей автомобілем. Мета дослідження – розробити програмне забезпечення пошуку попутників для подорожей автомобілем для спільного задоволення транспортних потреб людей.

Матеріали, методи та технічні засоби: мова Python, вебфреймворк Django, система керування базами даних PostgreSQL.

Результати. Програма, на створення якої була направлена робота, вирізняється можливістю пропонування альтернативного маршруту на певній ділянці подорожі пасажиром водію. Окрім того було проаналізовано інші можливості існуючих програм та виділено таким чином функціональні вимоги, що мають задовольняти потреби водіїв, пасажирів та адміністраторів. Визначено структуру бази даних у відповідності з цими вимогами. Визначено порядок роботи над пошуком подорожей, станами подорожі та запиту. Реалізовано програму, представлено логіку підтримки відповідних функцій на рівні коду.

Висновки. Програма дозволяє водіям, які мають вільне місце в автомобілі, надавати інформацію про власну подорож і пропонувати пасажирам долучитися до цієї подорожі, а пасажирам знаходити зручний для них спосіб переміщення і запитувати можливість долучитися до водія.

Галузь використання. Подорожі власним транспортом при наявності вільних місць в автомобілі.

ABSTRACT

Explanatory note to the diploma qualifying work of the bachelor: 84 pages, 26 figures, 16 tables, 3 appendixes, 10 sources.

TRAVEL BY CAR, TRAVEL ROUTE, TRAVELER, PASSENGER, DRIVER, SOFTWARE, DATABASE.

The object of research is the process of joint traveling by car. The subject of research is software to search for passengers to travel by car. The purpose of the study is to develop software for finding travel companions by car to jointly meet the transport needs of people.

Materials, methods and technical means: Python language, Django web framework, PostgreSQL database management system.

Results. The program, which was aimed at creating, is characterized by the possibility of offering an alternative route on a particular section of a journey by a passenger to a driver. In addition, other features of existing programs were analyzed and thus functional requirements were identified to meet the needs of drivers, passengers and administrators. The structure of the database in accordance to these requirements is determined. The order of work on search of travels, states of travel and inquiry is defined. The program is implemented, the logic of support of the corresponding functions at the code level is presented.

Conclusions. The program allows drivers who have a free seat in the car to provide information about their own trips and invite passengers to join these trips, and allows passengers to find a convenient way to move and request the opportunity to join the driver.

Field of use. Travelling by own transport if there are free places in the car.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	7
Вступ.....	8
1 Аналіз предметної області.....	9
1.1 Проблема спільних перевезень.....	9
1.2 Існуюче програмне забезпечення.....	9
1.3 Висновки за розділом 1 та завдання роботи.....	12
2 Розробка архітектури програми.....	13
2.1 Визначення функціональних вимог до програми.....	13
2.2 Вибір засобів розробки.....	16
2.3 Визначення станів основних сутностей.....	16
2.4 Проектування бази даних.....	18
2.5 Висновки за розділом 2.....	29
3 Розробка програми.....	30
3.1 Порядок роботи програми.....	30
3.2 Класи моделей програми.....	30
3.3 Класи представлень програми.....	36
3.4 Висновки за розділом 3.....	45
4 Експлуатація та тестування програми.....	46
4.1 Призначення програми.....	46
4.2 Умови виконання програми.....	46
4.3 Експлуатація програми.....	46
4.4 Тестування програми.....	50
4.5 Висновки за розділом 4.....	50
Висновки.....	51
Перелік джерел посилання.....	52
Додаток А Технічне завдання.....	53
Додаток Б Текст програми.....	57
Додаток В Слайди презентації.....	78

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

БД – база даних;

ПЗ – програмне забезпечення.

ВСТУП

Спільні подорожі автомобілем не так давно стали частиною суспільства загалом, хоча не можна сказати, що дана проблема повністю вирішена і розкрита. Кількість людей, які користуються такими сервісами, постійно збільшується, а далеко не всі на даний момент можуть користуватися такими сервісами, навіть якщо і знають про їх потенційне існування. Це призводить до того, що на даний момент весь час є групи користувачів, яких можна привернути для використання подібного програмного забезпечення (ПЗ).

Концепція спільних подорожей автомобілем є частиною концепції спільного використання економіки [1]-[3]. Це важлива концепція, адже вона дозволяє з одного боку знизити вплив на екологію за рахунок менших викидів, що зокрема яскраво виражено в спільних подорожах. З іншого боку це корисна концепція, адже зараз, коли дохід людей в нашій країні зменшується, велика кількість людей зіштовхнулась з транспортними проблемами, ціни на паливо ростуть у всьому світі, – все це разом призводить до того, що спільне використання стає ще важливішим, а спільні подорожі отримують ще більшу популярність через власну практичність. Кількість автомобілів на дорогах через збільшення вартості пального зменшується, а часом критично спадає через його значно обмежену доступність, а кількість потенційних пасажирів звичайно через це зростає, звідси і виникає потреба в розширенні діяльності в напрямку пошуку попутників для спільних подорожей.

У підсумку метою роботи є розробити ПЗ пошуку попутників для подорожей автомобілем для спільного задоволення транспортних потреб людей.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Проблема спільних перевезень

Спільні перевезення є частиною концепції економіки спільного використання. Поняття економіки спільного використання ґрунтується на обміні, розподілі та співпраці між окремими особами щодо товарів, послуг, ресурсів, часу чи знань, з грошовою підтримкою даних обмінів чи без нього, використовуючи зазвичай спеціальні платформи [1]-[3].

З'єднуючи водіїв з людьми, які хочуть їхати тим самим шляхом, сервіси спільних перевезень прагнуть зробити подорожі дешевшими, більш екологічними та більш соціальними: загалом концепція сучасних спільних перевезень побудована на принципі створення транспортної мережі, що складається з людей і довіри, яка їх об'єднує, тож на сьогодні найбільша подібна система, BlaBlaCar, має понад 30 мільйонів членів у 22 країнах Європи, розповсюджуючись до Мексики та Індії [4].

Однак у будь-якому випадку подібні сервіси вимагають певної сплати за власні послуги. Вони не є повністю безкоштовними. Наприклад, BlaBlaCar базово вимагає сплати приблизно 10 відсотків від загальної вартості поїздки.

1.2 Існуюче програмне забезпечення

Найвідомішим рішенням на даний момент у цій сфері є BlaBlaCar [5].

Це достатньо простий у використанні і популярний засіб пошуку попутників, що можна характеризувати таким чином:

- для пошуку на вебсайті достатньо вказати місто відправлення, бажаний пункт призначення та час відправлення, після цього відобразиться перелік результатів пошуку, представлений списком водіїв, які подорожують за цим маршрутом, їхні пункти відправлення, вартість та кількість місць у автомобілі;

- для створення поїздки потрібно натиснути на кнопку «Запропонувати поїздки» у верхній частині екрана, потім визначити маршрут подорожі, кількість доступних місць і будь-які інші пункти маршруту;
- програма надає систему взаємних відгуків: можна залишити відгук про поїздки, яку водій здійснив або в якій був пасажиром, і інші користувачі можуть цей відгук побачити;
- програма пропонує область налаштувань у профілі кожного користувача: у цій області можна вказати, чи можна брати з собою домашніх тварин, чи допускаються курці чи будь-які інші обмеження, у профілі також показано невеликий опис користувача;
- дозволяється також в профілі вказувати, на скільки водій або пасажир є балакучими: користувач може визначити, чи він говорить мало, чи розмовляє трохи з попутниками, чи любить говорити безперервно, це дозволить врахувати це в подальшому пошуку і зробити таким чином результат більш комфортним для кожної сторони, але загалом це не єдине подібне налаштування, так наприклад, схильність до прослуховування музики також є таким налаштуванням (усі такі налаштування враховуються під час пошуку);
- кожен користувач оцінюється 5 різними рівнями досвіду (новачок, середній, досвідчений, експерт і преміум);
- виконане бронювання можна скасувати, а також відправити повідомлення про скасування, проте зробити це потрібно вручну, тобто автоматичного відправлення повідомлення на випадок скасування як такого не передбачено;
- якщо бронювання скасував водій, то опублікувати інформацію про цю подорожу знову не можна буде, окрім того водій може бути оштрафований за надто часті скасування подорожей [6].

Однак з часом цей сервіс (рис. 1.1) став на стільки популярним і відомим у світі, що кількість певних обмежень стала зростати. До таких обмежень можна віднести і плату за бронювання, що активно просувається,

відповідно робить сервіс поступово фактично платним. До того ж версії для різних країн є достатньо розділеними, це призводить до того, що маючи обліковий запис на варіанті вебсервісу для певної країни, можна не мати доступу до подорожей в інших країнах, що звичайно є проблемою.

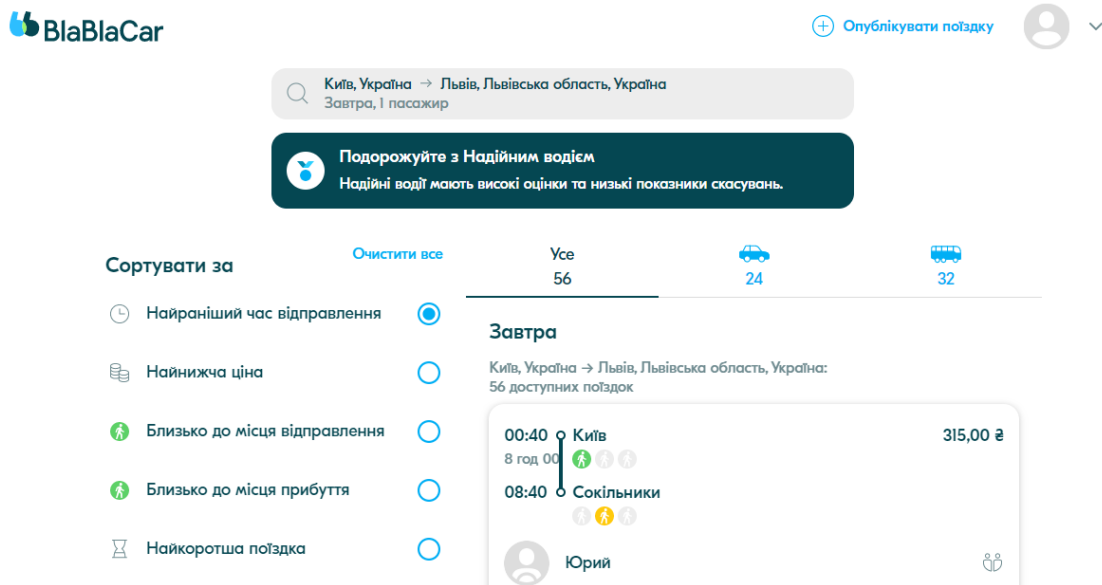


Рисунок 1.1 – BlaBlaCar

Усі ці перелічені фактори в результаті виражається в тому, що створюються інші варіанти ПЗ, які також діють в цій сфері. Однак вони також мають певні обмеження, наприклад компанія Google створила власне подібне рішення Waze Carpool, недоліком якого зокрема є початковий варіант роботи з нею. Адже для цього потрібно обов'язково встановити застосунок, а цей застосунок, наприклад, недоступний у нашій країні. В результаті будь-які проблеми з інтернетом, знаходячись в подорожі, розташування в країні, що не підтримується, призводять до того, що користувачі просто не можуть скористатися цим застосунком.

Poraride (рис. 1.2) [7] є одним з аналогів для пошуку попутників для спільних подорожей, що надає можливості пошуку попутників для подорожей в Сполучених Штатах і Канаді. Це додатково підкреслює важливість створення аналогів для певних країн. Застосунок має набагато менше фільтрів для пошуку порівняно з найвідомішим аналогом.

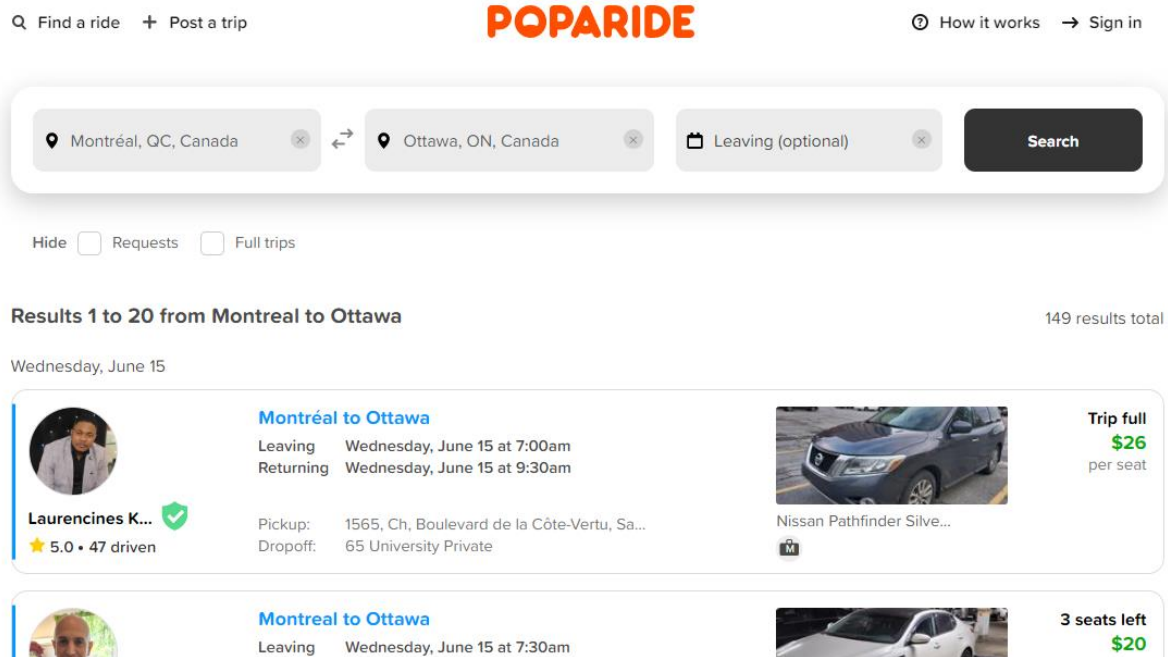


Рисунок 1.2 – Poraride

1.3 Висновки за розділом 1 та завдання роботи

Завдання роботи:

- дослідження проблеми пошуку попутників для подорожей автомобілем;
- визначення засобів розробки;
- визначення вимог до програми;
- проектування програми пошуку попутників для подорожей автомобілем;
- програмна реалізація.

Розглянуто проблему спільних подорожей, її особливості в сучасному світі, проаналізовано засоби, які надаються для підтримки спільних поїздок в існуючих програмах, що включають BlaBlaCar, Poraride. Основною відмінністю програми, що розробляється, запропоновано зробити можливість пропонування зміни маршруту в запиті від потенційного пасажера водію.

2 РОЗРОБКА АРХІТЕКТУРИ ПРОГРАМИ

2.1 Визначення функціональних вимог до програми

У даному підрозділі було визначено сценарії використання ПЗ, які спочатку включали сценарії роботи водія з програмою (рис. 2.1).

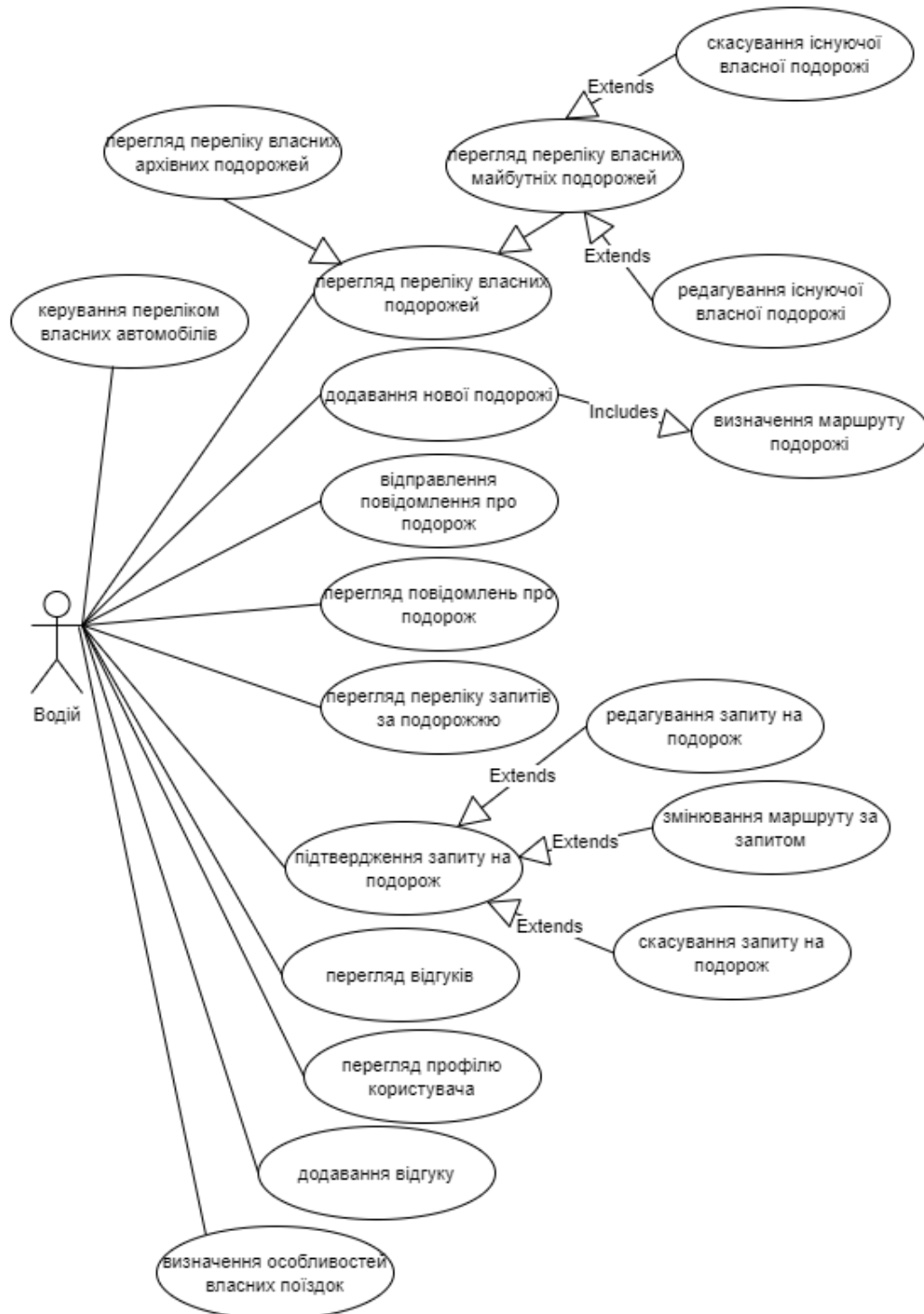


Рисунок 2.1 – Діаграма варіантів використання програми водієм

Реалізуючи пошук попутників для подорожей автомобілем, програма повинна забезпечувати роботу з функціями, що включають:

- перегляд переліку власних архівних подорожей;
- перегляд власних майбутніх подорожей;
- додавання нової подорожі;
- редагування існуючої власної подорожі;
- скасування існуючої власної подорожі;
- визначення маршруту подорожі;
- пошук автомобіля на потрібну подорож;
- додавання відгуку про водія або пасажирів;
- перегляд профілю користувача (водія або пасажирів);
- додавання нового автомобіля водія і його редагування;
- перегляд переліку власних автомобілів;
- визначення особливостей поїздок водія;
- перегляд відгуків про водія або пасажирів;
- відправлення повідомлення щодо подорожі;
- перегляд повідомлень щодо подорожі;
- створення запиту на подорож;
- редагування запиту на подорож;
- створення запиту на подорож зі зміненим маршрутом;
- перегляд переліку запитів за подорожжю;
- перегляд власних запитів;
- підтвердження запиту на подорож;
- скасування запиту на подорож;
- змінювання маршруту за запитом;
- керування користувачами;
- керування поїздками.

Сценарії роботи пасажирів і адміністратора приведені відповідно на рис. 2.2 та рис. 2.3.

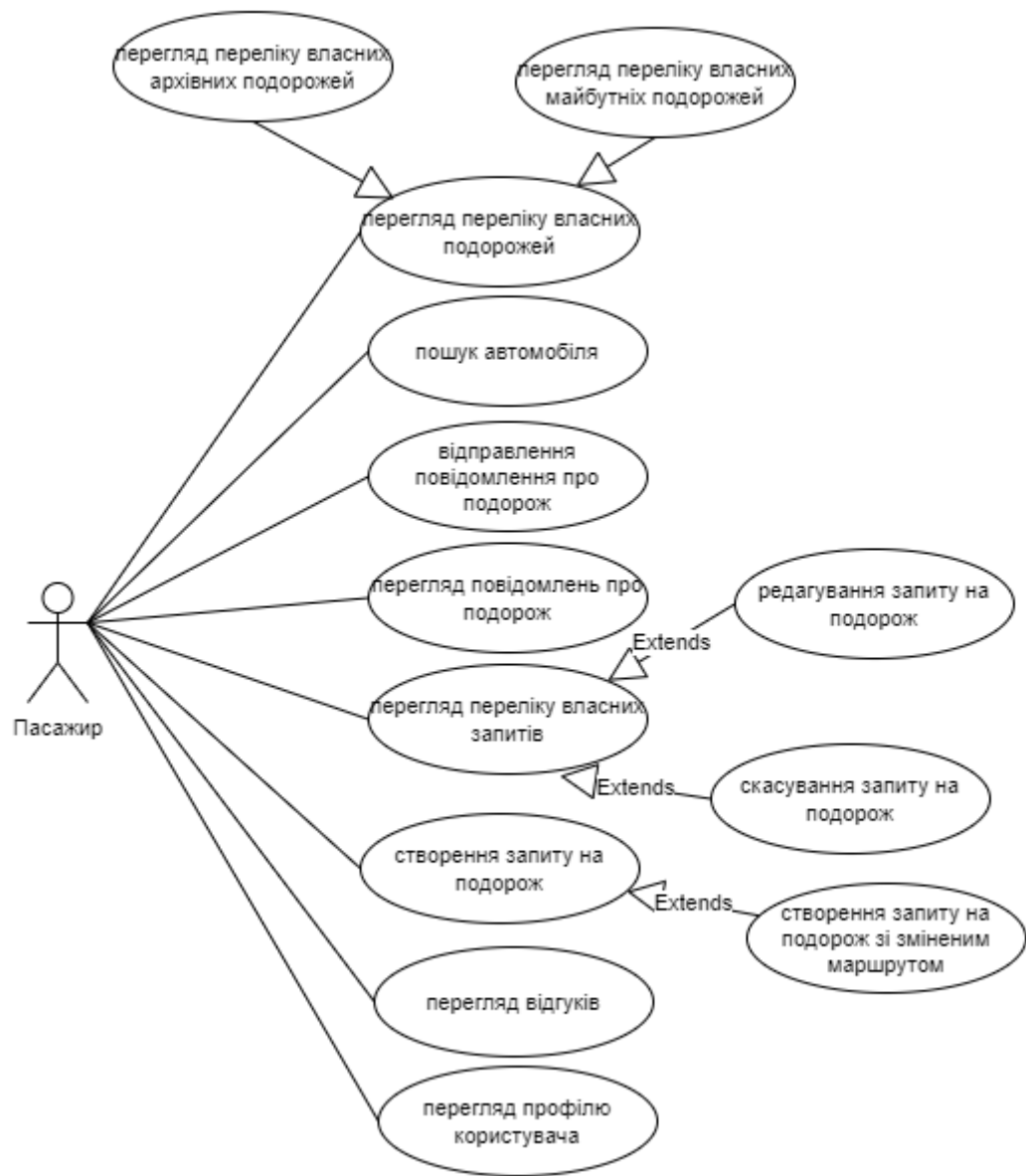


Рисунок 2.2 – Діаграма варіантів використання програми пасажиром



Рисунок 2.3 – Діаграма варіантів використання програми адміністратором

2.2 Вибір засобів розробки

Основний вибір засобів розробки полягав у тому, яку мову програмування обрати. Вибір (табл. 2.1) реалізовувався між Python і PHP. Вже після цього потрібно було обрати вебфреймворк. Було обрано мову Python [8]-[9], адже вона дозволяє легше читати і створювати код, має вищу захищеність застосунків за використання Django [10] як вебфреймворку порівняно з PHP, де для цього потрібен більший досвід розробників, має кращу розширюваність, що дозволяє до застосунку додавати функціональність легше.

Таблиця 2.1 – Вибір мови програмування

Показник	Python	PHP
Типізація	Динамічна	Динамічна
Читабельність коду	Висока	Середня
Захищеність застосунків	Вища за рахунок Django	Нижча
Рівень підтримки	Порівняний	Порівняний
Розширюваність	Вища	Порівняно дещо нижча

2.3 Визначення станів основних сутностей

До основних сутностей належать подорож і запити на подорож, оскільки саме вони представляють дві сторони взаємодії: водіїв та пасажирів.

Стани запиту на подорож входять до множини: новий, скасований та підтверджений (рис. 2.4). При цьому підтверджений запит водій також може скасувати.

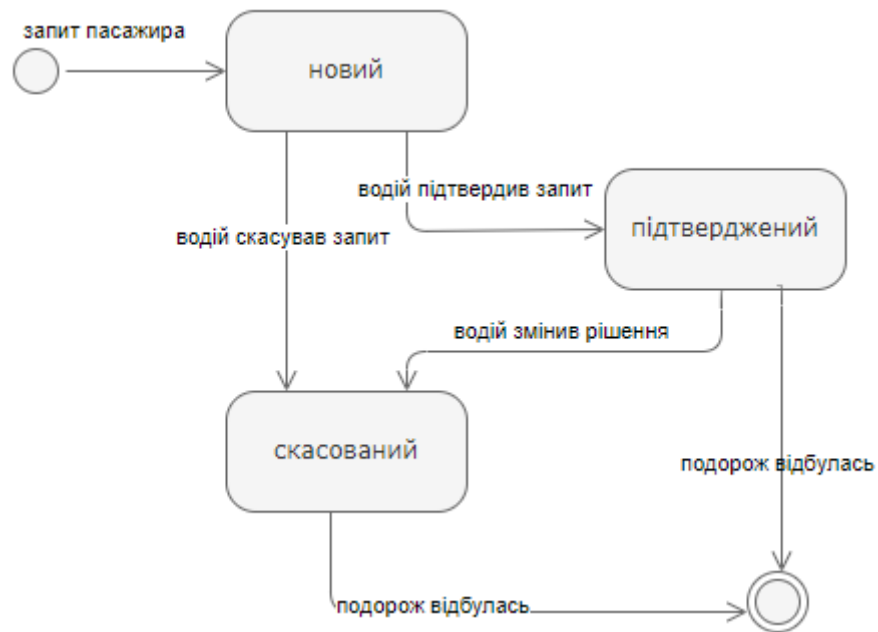


Рисунок 2.4 – Діаграма станів запиту на подорож

Стани подорожі входять до множини: нова, скасована, заповнена та архівована (рис. 2.5). При цьому заповнену подорож водій також може скасувати.



Рисунок 2.5 – Діаграма станів подорожі

2.4 Проєктування бази даних

Для роботи над базою даних (БД) була застосована система керування БД PostgreSQL. Використовуючи принципи її роботи, підтримувані типи даних, було відповідно виражено сутності предметної області.

Таблиця Driver дозволяє зберігати дані водіїв (табл. 2.2).

Таблиця 2.2 – Структура таблиці Driver

Но- мер	Ім'я поля	Тип поля	Опис поля
1	2	3	4
1	id	integer	номер водія
2	first_name	character (150)	ім'я водія
3	last_name	character (150)	прізвище водія
4	username	character (150)	логін водія
5	password	character (150)	пароль водія
6	phone	character (13)	номер телефону водія
7	trip_number	integer	кількість подорожей, яку вже провів водій
8	average_score	real	середня оцінка
9	reviews_number	integer	кількість відгуків на водія
10	positive_reviews	integer	кількість позитивних відгуків
11	negative_reviews	integer	кількість негативних відгуків
12	membered	timestamp	дата реєстрації

Продовження таблиці 2.2

1	2	3	4
13	passport	character (255)	шлях до файлу з паспортом
14	license	character (255)	шлях до файлу з водійським посвідченням
15	level	integer	рівень вправності у керуванні автомобілем, визначений самостійно

Таблиця Passenger дозволяє зберігати дані пасажирів (табл. 2.3).

Таблиця 2.3 – Структура таблиці Passenger

Но- мер	Ім'я поля	Тип поля	Опис поля
1	2	3	4
1	id	integer	номер пасажирів
2	first_name	character (150)	ім'я пасажирів
3	last_name	character (150)	прізвище пасажирів
4	username	character (150)	логін пасажирів
5	password	character (150)	пароль пасажирів
6	phone	character (13)	номер телефону пасажирів
7	trip_number	integer	кількість подорожей пасажирів
8	average_score	real	середня оцінка

Продовження таблиці 2.3

1	2	3	4
9	reviews_number	integer	кількість відгуків на пасажирів
10	positive_reviews	integer	кількість позитивних відгуків
11	negative_reviews	integer	кількість негативних відгуків
12	membered	timestamp	дата реєстрації
13	passport	character (255)	шлях до файлу з паспортом

Таблиця Trip дозволяє зберігати дані про подорожі (табл. 2.4).

Таблиця 2.4 – Структура таблиці Trip

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер подорожі
2	drivercar	integer	автомобіль водія для подорожі
3	created	timestamp	дата і час створення запису про подорож
4	start_time	timestamp	дата і час початку подорожі
5	end_time	timestamp	очікувані дата і час завершення подорожі
6	places	integer	кількість початково доступних місць для подорожі
7	active_places	integer	кількість доступних місць загалом на даний момент
8	state	character (1)	стан подорожі
9	price	real	вартість, яку стандартно потрібно сплатити за подорож

Поле `drivercar` таблиці `Trip` зв'язана з таблицею `Drivercar`: кожна подорож організовується на якомусь автомобілі одного з водіїв, автомобіль може використовуватися для різних подорожей, себто зв'язок належить до типу «один до багатьох».

Таблиця `TripPrice` дозволяє зберігати дані про особливі варіанти вартості подорожі між частинами маршруту (табл. 2.5).

Таблиця 2.5 – Структура таблиці `TripPrice`

Но- мер	Ім'я поля	Тип поля	Опис поля
1	<code>id</code>	<code>integer</code>	номер запису про вартість ділянки подорожі
2	<code>start_way</code>	<code>integer</code>	початковий пункт ділянки
3	<code>end_way</code>	<code>integer</code>	кінцевий пункт ділянки
4	<code>price</code>	<code>real</code>	вартість такого маршруту

Поле `start_way` таблиці `TripPrice` зв'язана з таблицею `Way`: кожен запис про вартість має один початковий пункт ділянки, сам пункт може входити до різних варіантів маршруту як початковий, себто зв'язок належить до типу «один до багатьох». Також полем `end_way` таблиці `TripPrice` зв'язана з таблицею `Way`: кожен запис про вартість має один кінцевий пункт ділянки, сам пункт може входити до різних варіантів маршруту як кінцевий, себто зв'язок належить до типу «один до багатьох».

Таблиця `Way` дозволяє зберігати дані про пункт маршруту подорожі (табл. 2.6).

Поле `trip` таблиці `Way` зв'язана з таблицею `Trip`: кожен пункт відноситься до певної подорожі, кожна подорож може включати багато пунктів на маршруті, себто зв'язок належить до типу «один до багатьох».

Таблиця 2.6 – Структура таблиці Way

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер запису про пункт маршруту
2	trip	integer	номер подорожі
3	num_in_trip	integer	позиція пункту в маршруті подорожі
4	pointx	real	координата на мапі (горизонтальна)
5	pointy	real	координата на мапі (вертикальна)
6	plan_arrive	timestamp	запланований час прибуття в пункт
7	plan_departure	timestamp	запланований час відправлення з пункту
8	address	character (255)	точна адреса місця, де в цій локації буде зупинка
9	map_point	integer	локація, де буде зупинка

Поле `map_point` таблиці `Way` зв'язана з таблицею `Point`: кожен пункт маршруту подорожі представляє певну локацію, тобто місто або селище, де ця зупинка на маршруті знаходиться, кожна локація може мати багато зупинок в різних маршрутах, себто зв'язок належить до типу «один до багатьох».

Таблиця `Point` дозволяє зберігати дані про локацію, якою може бути загалом місто, село (табл. 2.7).

Таблиця 2.7 – Структура таблиці Point

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер локації
2	title	character (255)	назва локації (міста)

Таблиця Note дозволяє зберігати дані про особливість подорожі (табл. 2.8). Такою особливістю може бути заборона на паління, наявність кондиціонера тощо.

Таблиця 2.8 – Структура таблиці Note

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер особливості
2	title	character (255)	повністю особливість

Таблиця DriverNote дозволяє зберігати дані про особливість подорожі, якої вимагає водій (табл. 2.9).

Таблиця 2.9 – Структура таблиці DriverNote

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер особливості подорожі, що вимагає водій
2	driver	integer	номер водія
3	note	integer	номер особливості подорожі

Таблиця DriverNote є реалізацією зв'язку між таблицями Driver і Note типу «багато до багатьох»: один водій може мати багато встановлених особливостей подорожі, кожна особливість може використовуватися різними водіями.

Таблиця Drivercar дозволяє зберігати дані про автомобіль водія (табл. 2.10).

Таблиця 2.10 – Структура таблиці Drivercar

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер автомобіля водія
2	driver	integer	номер водія
3	carmodel	integer	номер моделі автомобіля
4	color	character (30)	колір автомобіля
5	places	integer	кількість місць в автомобілі
6	info	character (255)	додатковий опис автомобіля

Поле driver таблиці Drivercar зв'язана з таблицею Driver: кожен водій може мати декілька автомобілів, якими він здійснює подорожі, кожен запис про автомобіль водія відповідає одному конкретному водію, себто зв'язок належить до типу «один до багатьох».

Поле carmodel таблиці Drivercar зв'язана з таблицею Carmodel: кожна модель автомобіля може мати багато автомобілів різних водіїв, кожен запис про автомобіль водія відповідає одній конкретній моделі, себто зв'язок належить до типу «один до багатьох».

Таблиця Carmodel дозволяє зберігати дані автомобіль водія (табл. 2.11).

Таблиця 2.11 – Структура таблиці Carmodel

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер моделі автомобіля
2	title	character (100)	назва моделі автомобіля
3	brand	integer	бренд автомобіля

Поле `brand` таблиці `Carmodel` зв'язана з таблицею `Brand`: кожен автомобільний бренд може мати багато моделей, кожна модель автомобіля представляє один конкретний бренд, себто зв'язок належить до типу «один до багатьох».

Таблиця `Brand` дозволяє зберігати дані про бренд автомобіля (табл. 2.12).

Таблиця 2.12 – Структура таблиці Brand

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер бренда
2	title	character (255)	назва бренда

Таблиця `Review` дозволяє зберігати дані про подорожі (табл. 2.13).

Поле `driver` таблиці `Review` зв'язана з таблицею `Driver`: кожен водій може мати багато відгуків, один відгук може стосуватися тільки одного водія, себто зв'язок належить до типу «один до багатьох».

Поле `passenger` таблиці `Review` зв'язана з таблицею `Passenger`: кожен пасажир може мати багато відгуків, один відгук може стосуватися тільки одного пасажирів, себто зв'язок належить до типу «один до багатьох».

Таблиця 2.13 – Структура таблиці Review

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер відгуку
2	driver	integer	водій, до якого відноситься або який є автором відгуку
3	passenger	integer	пасажир, до якого відноситься або який є автором відгуку
4	created	timestamp	дата і час створення відгуку
5	score	integer	оцінка

Таблиця Chat дозволяє зберігати дані про повідомлення між водіями і пасажиром щодо подорожі (табл. 2.14).

Таблиця 2.14 – Структура таблиці Chat

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер повідомлення
2	trip	integer	подорож, до якої відноситься повідомлення
3	passenger	integer	пасажир, якому адресовано або який створив повідомлення
4	created	timestamp	дата і час створення повідомлення
5	message	character (255)	текст повідомлення

Поле trip таблиці Chat зв'язана з таблицею Trip: кожне повідомлення відправляється тільки стосовно певної поїздки, кожна поїздка може мати багато повідомлень, себто зв'язок належить до типу «один до багатьох».

Поле passenger таблиця Chat зв'язана з таблицею Passenger: кожен пасажир може мати багато повідомлень, одне повідомлення може бути створено або стосуватися тільки одного пасажир, себто зв'язок належить до типу «один до багатьох».

Таблиця Request дозволяє зберігати дані про запити до подорожі (табл. 2.15).

Таблиця 2.15 – Структура таблиці Request

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер запиту
2	trip	integer	подорож, до якої відноситься запит
3	passenger	integer	пасажир, який створив запит
4	created	timestamp	дата і час створення запиту
5	state	character (1)	стан запиту
6	start_way	integer	пункт, з якого бажає їхати пасажир
7	end_way	integer	пункт, до якого бажає їхати пасажир
8	differ	boolean	визначає запит на подорож за існуючим маршрутом або за новим визначенням

Поле trip таблиця Request зв'язана з таблицею Trip: кожен запит стосується конкретної подорожі, кожна подорож може мати багато запитів, себто зв'язок належить до типу «один до багатьох».

Поле passenger таблиця Request зв'язана з таблицею Passenger: кожен пасажир може мати багато запитів, один запит створений конкретним пасажиром, себто зв'язок належить до типу «один до багатьох».

Таблиця Requestway дозволяє зберігати дані про пункти маршруту до запити до подорожі, якщо маршрут визначався самим пасажиром (табл. 2.16).

Таблиця 2.16 – Структура таблиці Requestway

Но- мер	Ім'я поля	Тип поля	Опис поля
1	id	integer	номер пункту маршруту до запиту
2	request	integer	запит, до якого відноситься пункт маршруту
3	num_in_trip	integer	позиція пункту в маршруті подорожі
4	pointy	real	координата на мапі (вертикальна)
5	pointx	real	координата на мапі (горизонтальна)
6	address	character (255)	бажана адреса в маршруті подорожі
7	map_point	integer	локація, до якої відноситься пункт подорожі

Поле request таблиці Requestway зв'язана з таблицею Request: кожен пункт маршруту запиту стосується тільки одного запиту, кожен запит може мати багато пунктів маршруту, себто зв'язок належить до типу «один до багатьох».

Поле map_point таблиці Requestway зв'язана з таблицею Point: кожен пункт маршруту запиту знаходиться тільки в одній локації, кожна локація може мати багато таких пунктів у різних маршрутах, себто зв'язок належить до типу «один до багатьох».

Схема структури БД показана на рис. 2.6.

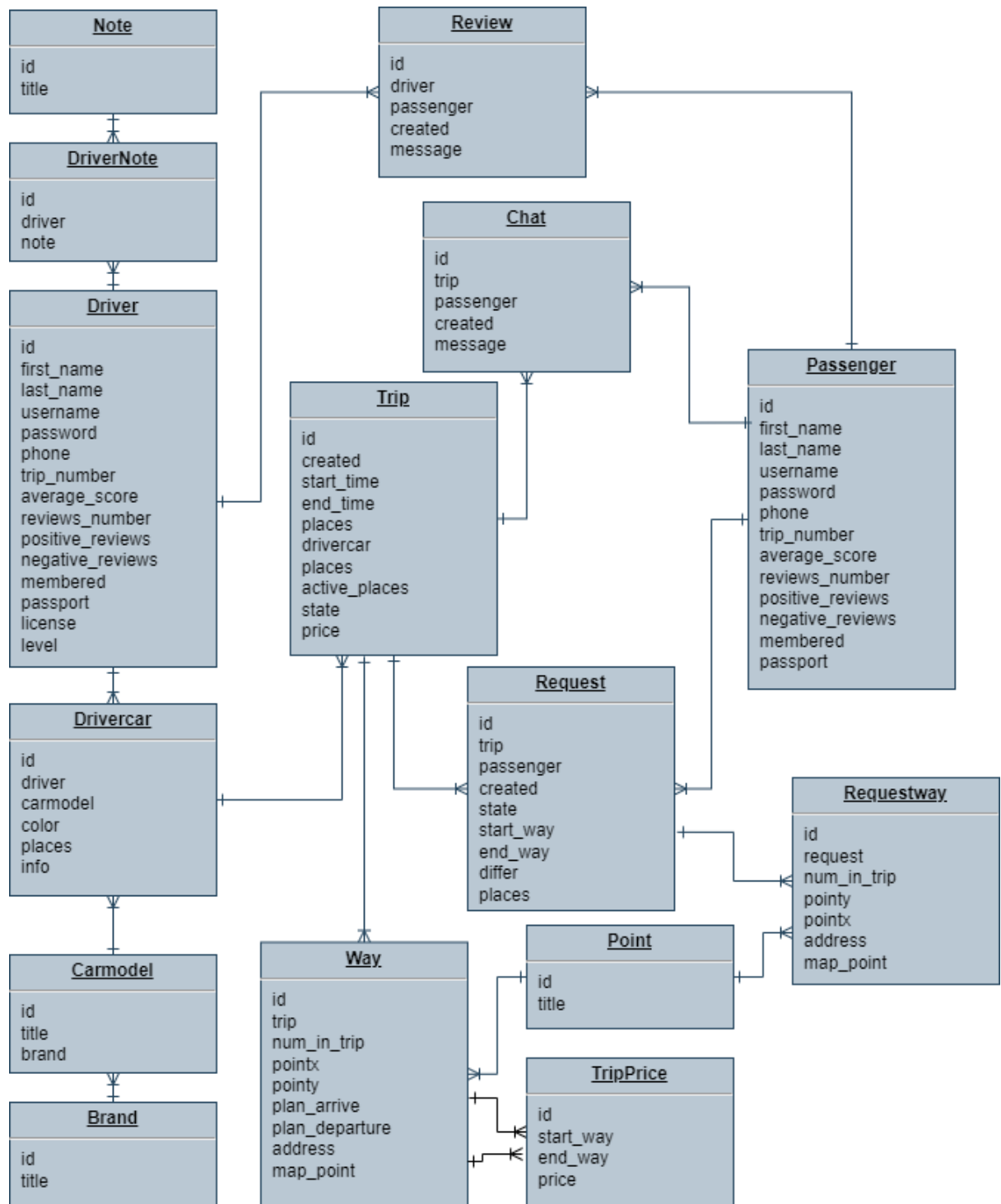


Рисунок 2.6 – Схема структури БД

2.5 Висновки за розділом 2

Виділено функціональні вимоги, що мають задовольняти потреби водіїв, пасажирів та адміністраторів. Визначено структуру БД. Визначено порядок роботи над станами подорожі та запиту.

3 РОЗРОБКА ПРОГРАМИ

3.1 Порядок роботи програми

Оскільки ключовим моментом взаємодії водіїв і пасажирів є саме подорож, то важливим для ефективної комунікації є знаходження адекватних варіантів подорожей. Тому порядок роботи з програмою було розглянуто на основі порядку функціонування процесу пошуку подорожей (рис. 3.1).

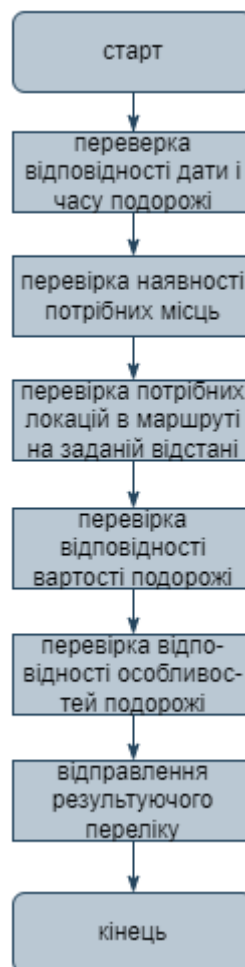


Рисунок 3.1 – Схема функціонування процесу пошуку подорожей

3.2 Класи моделей програми

Клас моделі Driver реалізовано через наслідування класу AbstractUser, який є стандартним класом Django-користувача, що дозволило таким чином

наслідувати основну частину необхідних полів класу. Додатково було визначено поля:

- phone – поле `models.CharField (max_length=13)`, що визначає номер телефону водія;
- trip_number – поле `models.IntegerField ()`, що визначає кількість подорожей, яку вже провів водій;
- average_score – поле `models.DecimalField(max_digits = 2, decimal_places = 1, default = 0)`, що визначає середню оцінку водія;
- reviews_number – поле `models.IntegerField ()`, що визначає кількість відгуків на водія;
- positive_reviews – поле `models.IntegerField ()`, що визначає кількість позитивних відгуків;
- negative_reviews – поле `models.IntegerField ()`, що визначає кількість негативних відгуків;
- membered – поле `models.DateTimeField()`, що визначає дату реєстрації;
- passport – поле `models.CharField (max_length=255)`, що визначає шлях до файлу з паспортом;
- license – поле `models.CharField (max_length=255)`, що визначає шлях до файлу з водійським посвідченням;
- level – поле `models.IntegerField ()`, що визначає рівень вправності у керуванні автомобілем;
- notes – поле `models.ManyToManyField(Note, related_name = 'drivernotes', db_table = 'DriverNote')`, що представляє набір особливостей, якими характеризуються подорожі з цим водієм.

Аналогічно побудовано Клас моделі `Passenger`. Додатково було визначено поля:

- phone – поле `models.CharField (max_length=13)`, що визначає номер телефону пасажера;

- trip_number – поле `models.IntegerField ()`, що визначає кількість подорожей пасажиром;
- average_score – поле `models.DecimalField(max_digits = 2, decimal_places = 1, default = 0)`, що визначає середню оцінку пасажиром;
- reviews_number – поле `models.IntegerField ()`, що визначає кількість відгуків на пасажиром;
- positive_reviews – поле `models.IntegerField ()`, що визначає кількість позитивних відгуків;
- negative_reviews – поле `models.IntegerField ()`, що визначає кількість негативних відгуків;
- membered – поле `models.DateTimeField()`, що визначає дату реєстрації;
- passport – поле `models.CharField (max_length=255)`, що визначає шлях до файлу з паспортом.

Клас моделі `Trip` має поля:

- drivercar – поле `models.ForeignKey(Drivercar)`, що визначає автомобіль водія для подорожі;
- created – поле `models.DateTimeField()`, що визначає дату і час створення запису про подорож;
- start_time – поле `models.DateTimeField()`, що визначає дату і час початку подорожі;
- end_time – поле `models.DateTimeField()`, що визначає очікувані дату і час завершення подорожі;
- places – поле `models.IntegerField ()`, що визначає кількість початково доступних місць для подорожі;
- active_places – поле `models.IntegerField ()`, що визначає кількість доступних місць загалом на даний момент;
- state – поле `models.CharField (max_length=1)`, що визначає стан подорожі;

– price – поле `models.DecimalField(max_digits = 7, decimal_places = 1, default = 0)`, що визначає вартість, яку стандартно потрібно сплатити за подорож.

Клас моделі `TripPrice` має поля:

– start_way – поле `models.ForeignKey(Way)`, що визначає початковий пункт ділянки;

– end_way – поле `models.ForeignKey(Way)`, що визначає кінцевий пункт ділянки;

– price – поле `models.DecimalField(max_digits = 7, decimal_places = 1, default = 0)`, що визначає вартість маршруту.

Клас моделі `Way` має поля:

– trip – поле `models.ForeignKey(Trip)`, що визначає подорож;

– num_in_trip – поле `models.IntegerField()`, що визначає вартість маршруту;

– pointx – поле `models.DecimalField(max_digits = 9, decimal_places = 6)`, що визначає горизонтальну координату на мапі;

– pointy – поле `models.DecimalField(max_digits = 9, decimal_places = 6)`, що визначає вертикальну координату на мапі;

– plan_arrive – поле `models.DateTimeField()`, що визначає запланований час прибуття в пункт;

– plan_departure – поле `models.DateTimeField()`, що визначає запланований час відправлення з пункту;

– address – поле `models.CharField(max_length=255)`, що визначає точну адресу місця, де в цій локації буде зупинка;

– map_point – поле `models.ForeignKey(Point)`, що визначає локацію, де буде зупинка.

Клас моделі `Point` має поле `title` – поле `models.CharField(max_length=255)`, що визначає назву локації.

Клас моделі `Note` має поле `title` – поле `models.CharField(max_length=255)`, що визначає назву особливості.

Клас моделі `Drivercar` має поля:

- `driver` – поле `models.ForeignKey(Driver)`, що визначає водія;
- `carmodel` – поле `models.ForeignKey(Carmodel)`, що визначає модель автомобіля;
- `places` – поле `models.IntegerField()`, що визначає кількість місць в автомобілі;
- `color` – поле `models.CharField(max_length=30)`, що визначає колір автомобіля;
- `info` – поле `models.CharField(max_length=255)`, що визначає додатковий опис автомобіля.

Клас моделі `Carmodel` має поля:

- `title` – поле `models.CharField(max_length=100)`, що визначає назву моделі автомобіля;
- `brand` – поле `models.ForeignKey(Brand)`, що визначає бренд автомобіля.

Клас моделі `Brand` має поле `title` – поле `models.CharField(max_length=255)`, що визначає назву бренду.

Клас моделі `Review` має поля:

- `driver` – поле `models.ForeignKey(Driver)`, що визначає водія, до якого відноситься або який є автором відгуку;
- `passenger` – поле `models.ForeignKey(Passenger)`, що визначає пасажирів, до якого відноситься або який є автором відгуку;
- `created` – поле `models.DateTimeField()`, що визначає дату і час створення відгуку;
- `score` – поле `models.IntegerField()`, що визначає оцінку.

Клас моделі `Chat` має поля:

- `trip` – поле `models.ForeignKey(Trip)`, що визначає подорож, до якої відноситься повідомлення;
- `passenger` – поле `models.ForeignKey(Passenger)`, що визначає пасажирів, якому адресовано або який створив повідомлення;

- `created` – поле `models.DateTimeField()`, що визначає дату і час створення повідомлення;

- `message` – поле `models.CharField (max_length=255)`, що визначає текст повідомлення.

Клас моделі `Request` має поля:

- `trip` – поле `models.ForeignKey(Trip)`, що визначає подорож, до якої відноситься запит;

- `passenger` – поле `models.ForeignKey(Passenger)`, що визначає пасажирів, який створив запит;

- `created` – поле `models.DateTimeField()`, що визначає дату і час створення запиту;

- `start_way` – поле `models.ForeignKey(Way)`, що визначає пункт, з якого бажає їхати пасажир;

- `end_way` – поле `models.ForeignKey(Way)`, що визначає пункт, до якого бажає їхати пасажир;

- `differ` – поле `models.BooleanField()`, що визначає це запит на подорож за існуючим маршрутом або за новим визначенням;

- `state` – поле `models.CharField (max_length=1)`, що визначає стан запиту.

Клас моделі `Requestway` має поля:

- `request` – поле `models.ForeignKey(Request)`, що визначає пункт маршруту до запиту;

- `num_in_trip` – поле `models.IntegerField ()`, що визначає позицію пункту в маршруті подорожі;

- `pointx` – поле `models.DecimalField(max_digits = 9, decimal_places = 6)`, що визначає горизонтальну координату на мапі;

- `pointy` – поле `models.DecimalField(max_digits = 9, decimal_places = 6)`, що визначає вертикальну координату на мапі;

- `address` – поле `models.CharField (max_length=255)`, що бажану адресу в маршруті подорожі;

– `map_point` – поле `models.ForeignKey(Point)`, що визначає локацію, до якої відноситься пункт подорожі.

3.3 Класи представлень програми

Усі представлення розроблені для підтримки роботи з відповідними шаблонами вебсторінок, кожна з яких фактично надає певну з функцій програми. Тому нижче будуть перелічені розроблені класи представлень.

Клас `ArchiveDriverTrips` реалізує представлення для вебсторінки перегляду переліку архівованих подорожей водія. Водій визначається на основі даних авторизованого користувача. Для цього спочатку перевіряється наявність запису про відповідного користувача як водія, і якщо такий запис є, то визначається перелік автомобілів цього водія. Далі для кожного автомобіля з цього переліку визначаються всі подорожі, які пов'язані з цим автомобілем, при чому час початку яких менший за поточний час, тобто вони вже мали відбутись. Визначений таким чином перелік упорядковується за датою створення подорожі, починаючи від останньої за часом створення подорожі:

```
driver_user = self.request.user
adriver = Driver.objects.get(id = driver_user.id)
driver_cars = Drivercar.objects.filter(driver = adriver)
currtime = date.today()
trips = Trip.objects.filter(drivercar__in = driver_cars,
start_time__lt = currtime)
trips = trips.order_by('-created')
```

Клас `ArchivePassengerTrips` реалізує представлення для вебсторінки перегляду переліку архівованих подорожей пасажира. Пасажир визначається на основі даних авторизованого користувача. Далі для такого пасажира запитуються всі його запити, які мають стан підтверджений, а подорожі за ними мають стан архівований, тобто це фактично запити, які були створені,

підтверджені, а вже потім самі подорожі архівовані, тому що час поїздки вийшов. Визначений таким чином перелік запитів упорядковується за датою створення подорожі, починаючи від останньої за часом створення подорожі.

Клас `FutureDriverTrips` реалізує представлення для вебсторінки перегляду переліку майбутніх подорожей водія. Водій визначається на основі даних авторизованого користувача. Для цього спочатку перевіряється наявність запису про відповідного користувача як водія, і якщо такий запис є, то визначається перелік автомобілів цього водія. Далі для кожного автомобіля з цього переліку визначаються всі подорожі, які пов'язані з цим автомобілем, але час початку яких не пізніший за поточний момент. Визначений таким чином перелік упорядковується за датою створення подорожі, починаючи від останньої за часом створення подорожі.

Клас `FuturePassengerTrips` реалізує представлення для вебсторінки перегляду переліку майбутніх подорожей пасажир. Пасажир визначається на основі даних авторизованого користувача. Далі для такого пасажир запитуються всі його запити, які мають стан підтверджений, а подорожі за ними ще не відбулись. Визначений таким чином перелік запитів упорядковується за датою створення подорожі, починаючи від останньої за часом створення подорожі.

Клас `CancelledDriverTrips` реалізує представлення для вебсторінки перегляду переліку скасованих подорожей водія. Водій визначається на основі даних авторизованого користувача. Для цього спочатку перевіряється наявність запису про відповідного користувача як водія, і якщо такий запис є, то визначається перелік автомобілів цього водія. Далі для кожного автомобіля з цього переліку визначаються всі подорожі, які пов'язані з цим автомобілем і які мають стан скасованого. Визначений таким чином перелік упорядковується за датою створення подорожі, починаючи від останньої за часом створення подорожі.

Клас `CancelledPassengerTrips` реалізує представлення для вебсторінки перегляду скасованих подорожей пасажир. Пасажир визначається на основі

даних авторизованого користувача. Далі для такого пасажера запитуються всі його запити, які мають стан підтверджений, а подорожі за ними мають стан скасованих. Визначений таким чином перелік запитів упорядковується за датою створення подорожі, починаючи від останньої за часом створення подорожі.

Клас `TripCreateView` реалізує підтримку створення подорожі водієм. Організований на основі наслідування стандартного класу представлення створення `CreateView`. Поля дати і часу початку і завершення заповнюються через форму, а також кількість місць, кількість початково доступних місць та стандартна вартість. Програмно при цьому визначається через параметр `drivercar`, переданий через запит до сторінки, автомобіль водія, який використовується в цій подорожі, а також програмно заповнюється перелік всіх наявних у цього водія, який є авторизованим користувачем, автомобілів, що можуть відповідно використовуватися в подорожі. Також задається одразу дата створення подорожі як поточний момент та стан подорожі як новоствореної.

Клас `TripUpdateView` реалізує підтримку редагування власної подорожі водієм. Організований на основі наслідування стандартного класу представлення створення `UpdateView`. Поля дати і часу початку і завершення, кількості місць, кількості початково доступних місць та стандартної вартості можуть змінюватися через форму. Програмно при цьому визначається через параметр `drivercar`, переданий через запит до сторінки, автомобіль водія, який використовується в цій подорожі, якщо він був змінений і відповідно переданий через запит.

Клас `TripCancelView` реалізує скасування власної подорожі водієм. Організований на основі наслідування стандартного класу представлення створення `UpdateView`. При цьому без зміни інших полів змінюється стан подорожі на скасований.

Клас `TripView` реалізує деталізацію подорожі, тобто детальний перегляд подорожі її власником, тобто водієм, який її організовує. Клас

організований на основі наслідування стандартного класу представлення деталізації `DetailView`. Окрім даних самого об'єкта подорожі до контексту даних додаються також дані:

- про маршрут подорожі, для чого вибираються всі записи про пункти подорожі та впорядковуються за полем номеру в подорожі (зберігаються в контексті під ключем `route`);
- про особливості ціноутворення на маршруті подорожі (зберігаються в контексті під ключем `prices`);
- про всі повідомлення, які були направлені стосовно цієї подорожі (зберігаються в контексті під ключем `chat`);
- про всі запити стосовно цієї подорожі (зберігаються в контексті під ключем `requests`).

Клас `DriverReviewCreateView` реалізує підтримку створення відгуку про водія пасажиром після подорожі. Організований на основі наслідування стандартного класу представлення створення `CreateView`. Поля водія і оцінки заповнюються через форму, а програмно вноситься інформація про пасажирів, який написав відгук, а також дату створення відгуку.

Клас `PassengerReviewCreateView` реалізує підтримку створення відгуку про пасажирів водієм після подорожі. Організований на основі наслідування стандартного класу представлення створення `CreateView`. Поля пасажирів і оцінки заповнюються через форму, а програмно вноситься інформація про водія, який написав відгук, а також дату створення відгуку.

Клас `DriverReviews` реалізує представлення для вебсторінки перегляду переліку відгуків про водія. Даний клас працює на основі параметрів, що передаються через параметри запиту, включаючи `driver`, що задає водія, відгуки про якого потрібно вивести, та `state`, що задає потрібну оцінку середню, за якою потрібно виводити відгуки. На основі цих параметрів відбираються всі відгуки, що відповідають цим параметрам, а також вони впорядковуються за датою створення, починаючи з тих відгуків, що є найсвіжішими.

Клас `PassengerReviews` реалізує представлення для вебсторінки перегляду переліку відгуків про пасажирів. Даний клас працює на основі параметрів, що передаються через параметри запиту, включаючи `passenger`, що задає пасажирів, відгуки про якого потрібно вивести, та `state`, що задає потрібну оцінку середню, за якою потрібно виводити відгуки. На основі цих параметрів відбираються всі відгуки, що відповідають цим параметрам, а також вони впорядковуються за датою створення, починаючи з тих відгуків, що є найсвіжішими.

Клас `OwnDriverReviews` реалізує представлення для вебсторінки перегляду переліку відгуків про водія, який авторизувався в програмі. Даний клас працює на основі параметра `state`, що задає потрібну оцінку середню, за якою потрібно виводити відгуки. На основі цих параметрів відбираються всі відгуки, що відповідають цим параметрам, а також вони впорядковуються за датою створення, починаючи з тих відгуків, що є найсвіжішими.

Клас `OwnPassengerReviews` реалізує представлення для вебсторінки перегляду переліку відгуків про пасажирів, який авторизувався в поточний момент. Даний клас працює на основі параметра `state`, що задає потрібну оцінку середню, за якою потрібно виводити відгуки. На основі цих параметрів відбираються всі відгуки, що відповідають цим параметрам, а також вони впорядковуються за датою створення, починаючи з тих відгуків, що є найсвіжішими.

Клас `OwnDriverCars` реалізує представлення для вебсторінки перегляду переліку автомобілів водія, який авторизувався в поточний момент.

Клас `DrivercarCreateView` реалізує підтримку додавання власного автомобіля водія. Організований на основі наслідування стандартного класу представлення створення `CreateView`. Поля кольору автомобіля, кількості місць та детального опису автомобіля заповнюються через форму, а програмно вноситься інформація про модель автомобіля, номер якої

передається через запит. Автоматично для підтримки роботи заповнюється перелік моделей та брендів автомобілів.

Клас `DrivercarUpdateView` реалізує підтримку редагування власного автомобіля водія. Організований на основі наслідування стандартного класу представлення редагування `UpdateView`. Поля кольору автомобіля, кількості місць та детального опису автомобіля за необхідності оновлюються через форму, а програмно вноситься інформація про модель автомобіля, номер якої передається через запит.

Клас `PassengerChatCreateView` реалізує підтримку відправлення пасажиром повідомлення про подорож. Організований на основі наслідування стандартного класу представлення створення `CreateView`. Поле тексту повідомлення заповнюється через форму, а програмно вноситься інформація про подорож, щодо якої це повідомлення, номер подорожі передається через запит.

Клас `DriverChatsView` реалізує представлення для вебсторінки перегляду переліку повідомлень про подорож для водія. Даний клас працює на основі параметра запита `trip`, що задає потрібну подорож. На основі цих параметрів відбираються всі повідомлення, що стосуються цієї подорожі, але тільки в тому випадку, якщо автомобіль, яким здійснюється подорож, належить авторизованому користувачу як водію:

```
trip = self.request.GET.get('trip')
ftrip = Trip.objects.get(id = trip)
driver = Driver.objects.filter(id = self.request.user)
if driver == ftrip.drivercar.driver:
    chats = Chat.objects.filter(trip = ftrip)
    chats = chats.order_by('-created')
    return chats
```

Клас `PassengerChatsView` реалізує представлення для вебсторінки перегляду переліку повідомлень про подорож для пасажирів. Даний клас працює на основі параметра запита `trip`, що задає потрібну подорож. На

основі цього параметра та запису про пасажирів, який відповідає авторизованому користувачу, відбираються всі повідомлення, що стосуються цієї подорожі і адресовані цьому пасажирові або створені цим пасажировим.

Клас `RequestCreateView` реалізує підтримку створення запиту на подорож. Організований на основі наслідування стандартного класу представлення створення `CreateView`. Поле подорожі, до якої відноситься запит, заповнюється програмно на основі номеру подорожі, переданої через параметр запиту. Таким саме чином задається пункт початку і пункт завершення подорожі пасажировим. Дата створення запиту задається програмно за поточним моментом, а стан за замовчуванням визначається як новий. Пасажир, від якого направлено запит, задається на основі даних користувача, який авторизувався в програмі.

Клас `RequestUpdateView` реалізує підтримку редагування запиту на подорож. Організований на основі наслідування стандартного класу представлення створення `UpdateView`. Поля пункт початку і пункт завершення подорожі пасажировим заповнюються програмно на основі номерів відповідних пунктів, переданих через параметр запиту.

Клас `RequestAcceptView` реалізує підтримку підтвердження запиту на подорож, організований на основі наслідування стандартного класу представлення створення `UpdateView`. У випадку якщо це запит до подорожі, яка організовується авторизованим користувачем як водієм, то стан запиту змінюється на відповідний.

Клас `RequestCancelView` реалізує скасування запиту на подорож, організований на основі наслідування стандартного класу представлення створення `UpdateView`. У випадку якщо це запит до подорожі, яка організовується авторизованим користувачем як водієм, то стан запиту змінюється на відповідний скасований.

Клас `TripRequestsView` реалізує представлення для вебсторінки перегляду переліку запитів на подорож. Даний клас працює на основі параметра запиту `trip`, що задає потрібну подорож. На основі цього параметра

та запису про водія, який відповідає авторизованому користувачу, перевіряється, чи ця подорож організовується даним водієм, а далі, якщо так, то вибираються всі відповідні запити і впорядковуються за датою створення.

Клас `OwnRequestsView` реалізує представлення для вебсторінки перегляду переліку власних запитів на подорож, створених авторизованим пасажиром. Вибираються всі запити, створені цим пасажиром, та впорядковуються за датою створення.

Клас `OwnAcceptedRequestsView` реалізує представлення для вебсторінки перегляду переліку власних запитів на подорож, створених авторизованим пасажиром, які при цьому вже були підтверджені водієм. Вибираються всі запити, створені цим пасажиром, перевіряється їх стан та вони впорядковуються за датою створення.

Клас `DriverProfileView` реалізує деталізацію профілю водія. Клас організований на основі наслідування стандартного класу представлення деталізації `DetailView`.

Клас `PassengerProfileView` реалізує деталізацію профілю пасажирів. Клас організований на основі наслідування стандартного класу представлення деталізації `DetailView`.

Клас `RequestwayCreateView` реалізує додавання пункту маршруту до запиту зі зміною маршруту. Організований на основі наслідування стандартного класу представлення створення `CreateView`. Поля положення пункту в маршруті, координат цього пункту та адреси пункту заповнюються через підключення відповідних полів класу до полів форми. Поле локації, де знаходиться цей пункт маршруту, заповнюється програмно на основі номеру локації, таким саме чином заповнюються і дані про подорож, до якої належить цей пункт маршруту.

Клас `RequestwayUpdateView` реалізує редагування пункту маршруту до запиту, який був оформлений зі зміною маршруту. Організований на основі наслідування стандартного класу представлення редагування `UpdateView`. Поля положення пункту в маршруті, координат цього пункту та

адреси пункту заповнюються через підключення відповідних полів класу до полів форми. Поле локації, де знаходиться цей пункт маршруту, заповнюється програмно на основі номеру локації.

Клас `WayCreateView` реалізує додавання пункту маршруту подорожі, організований на основі наслідування стандартного класу представлення створення `CreateView`. Поля положення пункту в маршруті, координат цього пункту, запланованого часу прибуття і відправлення та адреси пункту заповнюються через підключення відповідних полів класу до полів форми. Поля подорожі, до якої відноситься маршрут, та локації, де знаходиться цей пункт маршруту, заповнюються програмно на основі номерів подорожі і локації, переданих через параметри запити.

Клас `WayUpdateView` реалізує редагування пункту маршруту подорожі, організований на основі наслідування стандартного класу представлення редагування `UpdateView`. Поля положення пункту в маршруті, координат цього пункту, запланованого часу прибуття і відправлення та адреси пункту заповнюються через підключення відповідних полів класу до полів форми. Поле локації, де знаходиться цей пункт маршруту, у випадку його зміни, заповнюється програмно на основі номеру локації, переданого через параметр запити.

Також розроблено класи представлення `PassengerCreateView`, що реалізує додавання нового облікового запису пасажирів, представлення `PassengerUpdateView`, що реалізує роботу над зміною профілю пасажирів, `DriverCreateView`, що реалізує роботу над створенням профілю водія в системі, та представлення `DriverUpdateView`, що реалізує роботу над зміною профілю водія.

Аналогічний набір представлень для адміністратора включає класи представлення перегляду переліку водіїв `DriversView`, пасажирів `PassengersView`, змінювання прав доступу пасажирів `PassengerFullUpdateView`, змінювання прав доступу водія `DriverFullUpdateView`, пошуку серед всіх подорожей `TripSearchView`, повного

керування подорожжю `TripFullUpdateView`, пошуку серед всіх запитів на подорож `RequestSearchView`, повного керування запитом на подорож `RequestFullUpdateView`.

3.4 Висновки за розділом 3

У третьому розділі виконано опис реалізації програми. Розглянуто стани, які використовувались для характеристики подорожі та запиту пасажирів на спільну подорож. Представлено розроблені класи моделей і представлень застосунку як основні класи для організації роботи з функціями застосунку.

4 ЕКСПЛУАТАЦІЯ ТА ТЕСТУВАННЯ ПРОГРАМИ

4.1 Призначення програми

Призначенням розробки є надання інструменту пошуку попутників для поїздок на автомобілі водіям та пошуку водія для власних переміщень пасажиром. Особливістю є те, що пасажир може пропонувати змінити водію маршрут.

4.2 Умови виконання програми

Мінімальний рівень технічних засобів для забезпечення роботи програмної системи включає процесор не менш ніж 4 ядра і частотою 2,9 ГГц, оперативну пам'ять у 8 Гб, жорсткий диск у 1 Тб.

4.3 Експлуатація програми

Експлуатація програми користувачами здійснюється звичайним шляхом через браузер введенням запиту до вебсервера. Для роботи пасажир, водія та адміністратора передбачені окремі інтерфейси, що доступні за окремими посиланнями.

Розглянемо принципи роботи пасажир з програмою.

Основним інструментом, доступним пасажир, є пошук поїздок. Це можливість, яка виділяє його порівняно з водієм. Вона доступна з відповідного пункту навігації.

Для пошуку пасажир має задати (рис. 4.1) дату поїздки, доступну для вибору через календар, час, не раніше якого бажає виїхати пасажир, час, до якого потрібно здійснити виїзд, час, до якого потрібно приїхати в потрібне місце, потрібна кількість місць, пункт виїзду (при введенні випадає перелік підходящих), пункт прибуття (аналогічно влаштований), мінімальну вартість поїздки, максимальну вартість, а також за необхідності задати радіус дії,

тобто кількість кілометрів, віддалення на яку від заданого пункту виїзду і прибуття допускається. Також нижче розташовані особливості, за якими потрібно задати, чи необхідні вони, чи ні під час подорожі пасажирів.

ШУКАТИ ПОДОРОЖІ ПОВІДОМЛЕННЯ ВІДГУКИ ПРОФІЛЬ ЗАЛИШИТИ

Дата подорожі
29.06.2022

Виїзд не раніше
--:--

Виїзд не пізніше
--:--

Прибуття не пізніше
--:--

Кількість місць

ШУКАТИ

Пункт виїзду

Пункт прибуття

Вартість від

Вартість до

Радіус дії, км

☒ не палити

☐ кондиціонер

☒ з тваринами

Рисунок 4.1 – Форма пошуку

В самому низу сторінки відображаються результати пошуку (рис. 4.2). При натисканні на кнопку пошуку сторінка переміщується на дану позицію. За кожним варіантом виводиться фотографія водія (потрібно натиснути, щоб перейти на сторінку профілю з середньою оцінкою, кількістю позитивних, негативних відгуків), загальний маршрут (потрібно натиснути, щоб відкрити повністю сторінку подорожі), час подорожі, доступна кількість місць і

максимальна можлива кількість місць, вартість за загальний маршрут, а також кнопка для формування запиту на поїздку.

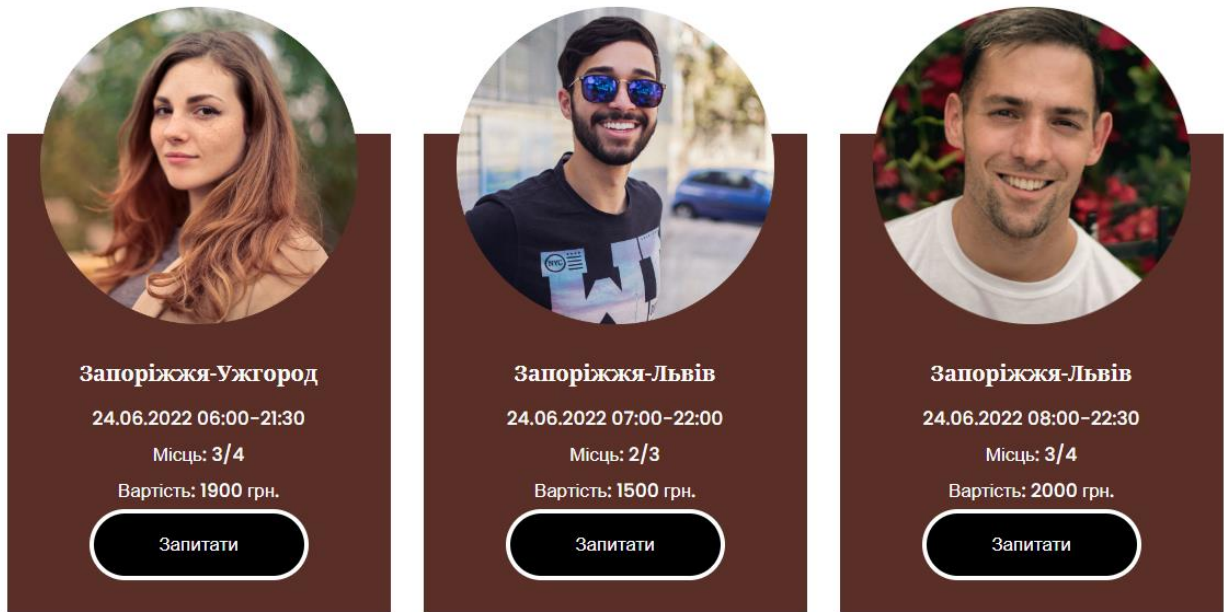


Рисунок 4.2 – Результати пошуку

З профілю водія доступний перехід до переліку відгуків (рис. 4.3).

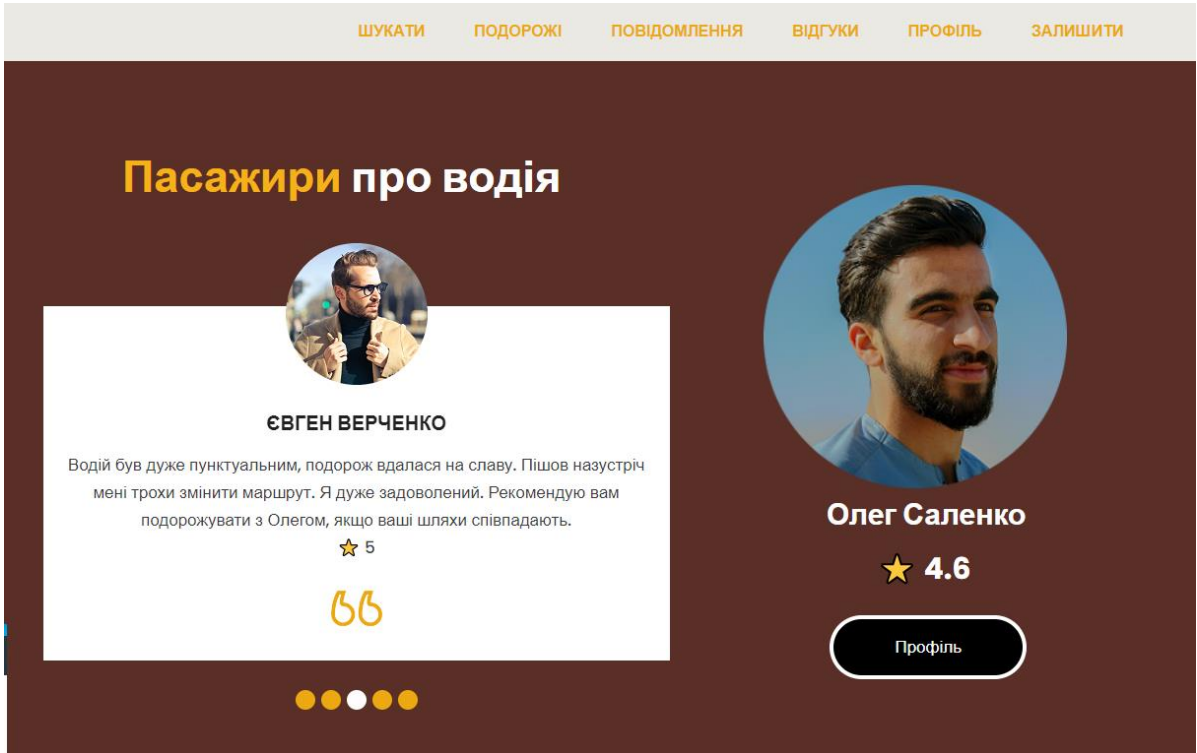


Рисунок 4.3 – Відгуки про водія

За кожним відгуком можна переміститися до профілю пасажир, а звідти до відгуків про нього.

Зі сторінки подорожі можна також написати водію повідомлення, а перелік написаних повідомлень відображається з відповідного пункту панелі навігації. В подорожах пасажир відображаються його подорожі на основі запитів. Звідси він може відображати майбутні, архівні подорожі. Також пасажир може переглянути відгуки про себе.

Водій має засоби керування своїми подорожами і запитами на них (рис. 4.4), що відповідно виділяє його на фоні пасажир.

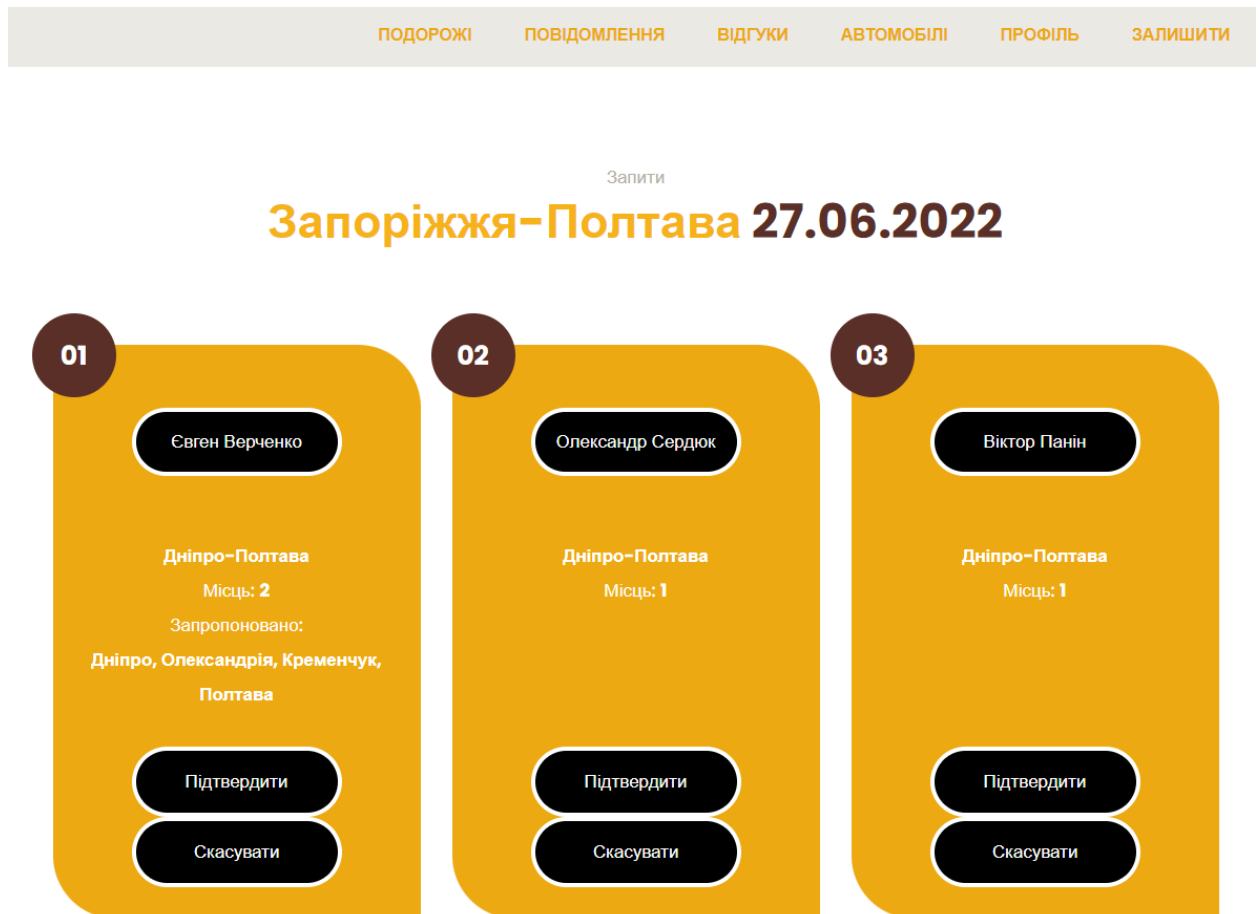


Рисунок 4.4 – Керування запитами водієм

Керуючи запитами, водій може їх підтверджувати, скасовувати, отримуючи за кожною пропозицією інформацію про запитану ділянку

маршруту, про запитану кількість місць, про альтернативний маршрут, якщо він був запитаний. Також звідси можна перейти до профілю пасажера.

При керуванні подорожами, знову ж таки, можна відображати майбутні, архівні подорожі. Водію також доступна можливість додавання власних автомобілів з відповідного пункту панелі навігації.

4.4 Тестування програми

Тестування програми було виконано шляхом створення модульних тестів для перевірки основних функцій, що включали пошук подорожей, відображення власних запитів, підтвердження, скасування запитів пасажирів, створення подорожей. Окрім того після завершення реалізації програми було також перевірено роботу програми щодо реалізації функцій шляхом взаємодії користувача з інтерфейсом програми. Тестування було успішним.

4.5 Висновки за розділом 4

Виконано опис основних інструментів, що надаються для роботи водія, пасажера в програмі. Основною особливістю програми є можливість задавати альтернативний маршрут на певних ділянках маршруту початкової подорожі, що була запропонована водієм.

ВИСНОВКИ

Метою роботи було розробити програмне забезпечення пошуку попутників для подорожей автомобілем для спільного задоволення транспортних потреб людей.

Програма, на створення якої була направлена робота, вирізняється можливістю пропонування альтернативного маршруту на певній ділянці подорожі пасажиром водію. Окрім цієї можливості було проаналізовано інші можливості існуючих програм та виділено таким чином функціональні вимоги, що мають задовольняти потреби водіїв, пасажирів та адміністраторів.

Визначено структуру бази даних у відповідності з вимогами. Визначено порядок роботи над пошуком подорожей, станами подорожі та запиту. Реалізовано програму, представлено логіку підтримки відповідних функцій на рівні коду. Детально представлено принципи організації пошуку подорожей, які підходять під вимоги пасажирів.

При розробці програми використано мову Python, вебфреймворк Django, систему керування базами даних PostgreSQL.

Створена програма дозволяє водіям, які мають вільне місце в автомобілі, надавати інформацію про власну подорож і пропонувати пасажирів долучитися до цієї подорожі, а пасажирів знаходити зручний для них спосіб переміщення і запитувати можливість долучитися до водія. Розширеною функціональністю є пропонування коригування маршруту під власні потреби пасажирів. Окрім того надаються можливості створення і перегляду відгуків, відправлення і читання повідомлень стосовно подорожей, визначення власних автомобілів водієм, керування подорожами і запитами на ці подорожі у відповідності з визначеними діаграмами станів принципами.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. The sharing economy [Electronic resource] : Definition, examples and advantages. – Access mode : <https://climate.selectra.com/en/environment/sharing-economy>.
2. Lovick, S. What Is The Sharing Economy? [Electronic resource] / S. Lovick. – Access mode : <https://www.businessbecause.com/news/insights/6736/what-is-the-sharing-economy>.
3. Sharing Economy [Electronic resource]. – Access mode : <https://www.investopedia.com/terms/s/sharing-economy.asp>.
4. BlaBlaCar. Born to ride (share) [Electronic resource]. – Access mode : <https://breakthrough.unglobalcompact.org/briefs/blablacar-born-to-ride-share-frederic-mazzella/>.
5. BlaBlaCar – пошук надійних попутників [Електрон. ресурс]. – Режим доступу : <https://www.blablacar.com.ua/>.
6. BlaBlaCar [Electronic resource] : 15 things you need to know before using it. – Access mode : <https://www.apptuts.net/en/tutorial/android/blablacar-things-need-to-know/>.
7. Poparide – Reliable City-to-City Carpooling in Canada [Electronic resource]. – Access mode : <https://www.poparide.com/>.
8. Ramalho, L. Fluent Python [Text] : Clear, Concise, and Effective Programming / L. Ramalho. – Massachusetts : O'Reilly Media, 2022. – 1012 p.
9. 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.
10. The web framework for perfectionists with deadlines. Django [Electronic resource]. – Access mode : <https://www.djangoproject.com/>.

ДОДАТОК А
Технічне завдання

Вступ

У роботі необхідно розробити програмне забезпечення пошуку попутників для подорожей автомобілем.

A.1 Підстави для розробки

Розробка в межах дипломної кваліфікаційної роботи реалізується на підставі наказу № 102 від 27 квітня 2022 р. за Національним університетом «Запорізька політехніка» за темою «Програмне забезпечення пошуку попутників для подорожей автомобілем».

A.2 Призначення розробки

Призначенням розробки є надання інструменту пошуку попутників для поїздки на автомобілі водіям та пошуку водія для власних переміщень пасажиром.

A.3 Основні вимоги до програми

A.3.1 Вимоги до функціональних характеристик

Функціональні характеристики, які повинна мати програма, охоплюють:

- перегляд переліку власних архівних подорожей;
- перегляд власних майбутніх подорожей;
- додавання нової подорожі;
- редагування існуючої власної подорожі;
- скасування існуючої власної подорожі;
- визначення маршруту подорожі;
- пошук автомобіля на потрібну подорож;

- додавання відгуку про водія;
- перегляд профілю користувача (водія або пасажир);
- додавання нового автомобіля водія і його редагування;
- перегляд переліку власних автомобілів;
- визначення особливостей поїздок водія;
- перегляд відгуків про водія або пасажир;
- відправлення повідомлення щодо подорожі;
- перегляд повідомлень щодо подорожі;
- створення запиту на подорож;
- редагування запиту на подорож;
- створення запиту на подорож зі зміненим маршрутом;
- перегляд переліку запитів за подорожжю;
- підтвердження запиту на подорож;
- скасування запиту на подорож;
- змінювання маршруту за запитом;
- керування користувачами;
- керування поїздками.

А.3.2 Умови експлуатації

Експлуатація програми має забезпечуватися з одного боку звичайними користувачами, які працюють з даними, наповнюють їх, а з іншого боку – адміністраторами системи та БД.

А.3.3 Вимоги до складу та параметрів технічних засобів

Мінімальний рівень технічних засобів для забезпечення роботи програмної системи включає процесор не менш ніж 4 ядра і частотою 2,9 ГГц, оперативну пам'ять у 8 Гб, жорсткий диск у 1 Тб.

А.4 Порядок контролю та приймання

Для контролю виконання роботи використовується календарний план, а приймання роботи виконується під час проведення процедури захисту роботи перед екзаменаційною комісією.

ДОДАТОК Б
Текст програми

```

import datetime
import haversine as hav
from django.contrib.auth.mixins import LoginRequiredMixin
from django.urls import reverse_lazy
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 .models import Trip, Driver, Request, Passenger, Drivercar
from datetime import date

#перелік архівованих подорожей водія
class ArchiveDriverTrips (ListView):
    model = Trip
    template_name = 'archivedtrips.html'

    def get_queryset(self):
        driver_user = self.request.user
        adriver = Driver.objects.get(id = driver_user.id)
        driver_cars = Drivercar.objects.filter(driver =
adriver)

        currtime = date.today()
        trips = Trip.objects.filter(drivercar__in =
driver_cars, start_time__lt = currtime)
        trips = trips.order_by('-created')
        return trips

#перелік майбутніх подорожей водія
class FutureDriverTrips (ListView):
    model = Trip
    template_name = 'futuredtrips.html'

    def get_queryset(self):
        driver_user = self.request.user
        adriver = Driver.objects.get(id = driver_user.id)
        driver_cars = Drivercar.objects.filter(driver =
adriver)

        currtime = date.today()
        trips = Trip.objects.filter(drivercar__in =
driver_cars, start_time__gte = currtime)
        trips = trips.order_by('-created')
        return trips

```

```

#перелік архівованих подорожей пасажира
class ArchivePassengerTrips (ListView):
    model = Trip
    template_name = 'archiveptrips.html'

    def get_queryset(self):
        passenger_user = self.request.user
        apassenger = Passenger.objects.get(id =
passenger_user.id)
        requests = Request.objects.filter(passenger =
apassenger, state = 'A', trip__state = 'A')
        currtime = date.today()
        requests = requests.filter(trip__start_time__lt =
currtime)

        requests = requests.order_by('-trip__created')
        return requests

#перелік майбутніх подорожей пасажира
class FuturePassengerTrips (ListView):
    model = Trip
    template_name = 'futureptrips.html'

    def get_queryset(self):
        passenger_user = self.request.user
        apassenger = Passenger.objects.get(id =
passenger_user.id)
        requests = Request.objects.filter(passenger =
apassenger, state = 'A')
        currtime = date.today()
        requests = requests.filter(trip__start_time__gte =
currtime)

        requests = requests.order_by('-trip__created')
        return requests

#перелік скасованих подорожей водія
class CancelledDriverTrips (ListView):
    model = Trip
    template_name = 'cancelledtrips.html'

    def get_queryset(self):
        driver_user = self.request.user
        adriver = Driver.objects.get(id = driver_user.id)
        driver_cars = Drivercar.objects.filter(driver =
adriver)

```

```

        trips = Trip.objects.filter(drivercar__in =
driver_cars, state = 'C')
        trips = trips.order_by('-created')
        return trips

#перелік скасованих подорожей пасажира
class CancelledPassengerTrips (ListView):
    model = Trip
    template_name = 'cancelleptrips.html'

    def get_queryset(self):
        passenger_user = self.request.user
        apassenger = Passenger.objects.get(id =
passenger_user.id)
        requests = Request.objects.filter(passenger =
apassenger, state = 'A')
        requests = requests.filter(trip__state = 'C')
        requests = requests.order_by('-trip__created')
        return requests

#створення подорожі водієм
class TripCreateView (LoginRequiredMixin, CreateView):
    model = Trip
    fields = ['start_time', 'end_time', 'places',
'active_places', 'price']
    template_name = 'addtrip.html'

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

    def get_context_data(self, **kwargs):
        dcar = get_object_or_404(Drivercar,
id=self.kwargs['drivercar'])
        kwargs['dcar'] = dcar
        driver_user = self.request.user
        adriver = Driver.objects.get(id = driver_user.id)
        driver_cars = Drivercar.objects.filter(driver =
adriver)

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

    def form_valid(self, form):
        self.created = date.today()

```

```

        self.drivercar = get_object_or_404(Drivercar,
id=self.kwargs['drivercar'])
        self.state = 'N'
        return super().form_valid(form)

#редагування подорожі водієм
class TripUpdateView (LoginRequiredMixin, UpdateView):
    model = Trip
    fields = ['start_time', 'end_time', 'places',
'active_places', 'price']
    template_name = 'updatetrip.html'

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

    def get_context_data(self, **kwargs):
        dcar = get_object_or_404(Drivercar,
id=self.kwargs['drivercar'])
        kwargs['dcar'] = dcar
        driver_user = self.request.user
        adriver = Driver.objects.get(id = driver_user.id)
        driver_cars = Drivercar.objects.filter(driver =
adriver)

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

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

#скасування подорожі водієм
class TripCancelView (LoginRequiredMixin, UpdateView):
    model = Trip
    template_name = 'canceltrip.html'

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

    def form_valid(self, form):
        self.state = 'C'
        return super().form_valid(form)

```

```

#деталізація подорожі
class TripView (DetailView):
    template_name = 'trip.html'
    model = Trip

    def get_context_data(self, **kwargs):
        tripdata = super().get_context_data(**kwargs)
        ways = Way.objects.filter(trip = self.object.pk)
        ways = ways.order_by('num_in_trip')
        tripdata ['route'] = ways
        prices = TripPrice.objects.filter(Q(start_way__in =
ways) | Q(end_way__in = ways))
        tripdata ['prices'] = prices
        chats = Chat.objects.filter(trip = self.object.pk)
        chats = chats.order_by('-created')
        tripdata ['chat'] = chats
        requests = Request.objects.filter(trip =
self.object.pk)
        requests = requests.order_by('-created')
        tripdata ['requests'] = requests
        return tripdata

#створення відгуку про водія
class DriverReviewCreateView (LoginRequiredMixin, CreateView):
    model = Review
    fields = ['driver', 'score']
    template_name = 'addreview.html'

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

    def form_valid(self, form):
        self.created = date.today()
        passenger_user = self.request.user
        apassenger = Passenger.objects.get(id =
passenger_user.id)
        self.passenger = apassenger
        return super().form_valid(form)

#створення відгуку про пасажира
class PassengerReviewCreateView (LoginRequiredMixin,
CreateView):
    model = Review
    fields = ['passenger', 'score']

```

```

        template_name = 'addpreview.html'

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

    def form_valid(self, form):
        self.created = date.today()
        driver_user = self.request.user
        adriver = Driver.objects.get(id = driver_user.id)
        self.driver = adriver
        return super().form_valid(form)

#перелік відгуків про водія
class DriverReviews (ListView):
    model = Review
    template_name = 'driverreviews.html'

    def get_queryset(self):
        driver = self.request.GET.get('driver')
        adriver = Driver.objects.get(id = driver)
        astate = self.request.GET.get('state')
        reviews = Review.objects.filter(driver = adriver)
        astate1 = astate + 1
        astate2 = astate - 1
        reviews = reviews.filter(score__lt = astate1,
score__gt = astate2)
        reviews = reviews.order_by('-created')
        return reviews

    def get_context_data(self, **kwargs):
        reviewdata = super(DriverReviews,
self).get_context_data(**kwargs)
        driver = self.request.GET.get('driver')
        reviewdata ['driver'] = Driver.objects.get(id =
driver)

        astate = self.request.GET.get('state')
        reviewdata ['state'] = astate
        return reviewdata

#перелік відгуків про пасажирів
class PassengerReviews (ListView):
    model = Review
    template_name = 'passengerreviews.html'

```

```

def get_queryset(self):
    passenger = self.request.GET.get('passenger')
    apassenger = Passenger.objects.get(id = passenger)
    astate = self.request.GET.get('state')
    astate1 = astate + 1
    astate2 = astate - 1
    reviews = Review.objects.filter(passenger =
apassenger, score__lt = astate1, score__gt = astate2)
    reviews = reviews.order_by('-created')
    return reviews

def get_context_data(self, **kwargs):
    reviewdata = super(PassengerReviews,
self).get_context_data(**kwargs)
    passenger = self.request.GET.get('passenger')
    reviewdata ['passenger'] = Passenger.objects.get(id =
passenger)

    astate = self.request.GET.get('state')
    reviewdata ['state'] = astate
    return reviewdata

#перелік відгуків про авторизованого водія
class OwnDriverReviews (ListView):
    model = Review
    template_name = 'odriverreviews.html'

def get_queryset(self):
    adriver = Driver.objects.get(id = self.request.user)
    astate = self.request.GET.get('state')
    reviews = Review.objects.filter(driver = adriver)
    astate1 = astate + 1
    astate2 = astate - 1
    reviews = reviews.filter(score__lt = astate1,
score__gt = astate2)
    reviews = reviews.order_by('-created')
    return reviews

def get_context_data(self, **kwargs):
    reviewdata = super(DriverReviews,
self).get_context_data(**kwargs)
    astate = self.request.GET.get('state')
    reviewdata ['state'] = astate
    return reviewdata

#перелік відгуків про авторизованого пасажирів

```

```

class OwnPassengerReviews (ListView):
    model = Review
    template_name = 'opassengerreviews.html'

    def get_queryset(self):
        apassenger = Passenger.objects.get(id =
self.request.user)
        astate = self.request.GET.get('state')
        reviews = Review.objects.filter(passenger =
apassenger, score = astate)
        reviews = reviews.order_by('-created')
        return reviews

    def get_context_data(self, **kwargs):
        reviewdata = super(PassengerReviews,
self).get_context_data(**kwargs)
        astate = self.request.GET.get('state')
        reviewdata ['state'] = astate
        return reviewdata

#перелік автомобілів авторизованого водія
class OwnDriverCars (ListView):
    model = Drivercar
    template_name = 'owndrivercars.html'

    def get_queryset(self):
        adriver = Driver.objects.get(id = self.request.user)
        cars = Drivercar.objects.filter(driver = adriver)
        cars = cars.order_by('-id')
        return cars

#внесення автомобіля водія
class DrivercarCreateView (LoginRequiredMixin, CreateView):
    model = Drivercar
    fields = ['color', 'places', 'info']
    template_name = 'adddrivercar.html'

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

    def get_context_data(self, **kwargs):
        dcar = get_object_or_404(Carmodel,
id=self.kwargs['model'])
        kwargs['model'] = dcar

```

```

        models = Carmodel.objects.all()
        kwargs['models'] = models
        brands = Brand.objects.all()
        kwargs['brands'] = brands
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.driver = Driver.objects.get(id =
self.request.user)
        self.carmodel = get_object_or_404(Carmodel,
id=self.kwargs['model'])
        return super().form_valid(form)

#редагування автомобіля водія
class DrivercarUpdateView (LoginRequiredMixin, UpdateView):
    model = Drivercar
    fields = ['color', 'places', 'info']
    template_name = 'updateddrivercar.html'

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

    def get_context_data(self, **kwargs):
        dcar = get_object_or_404(Carmodel,
id=self.kwargs['model'])
        kwargs['model'] = dcar
        models = Carmodel.objects.all()
        kwargs['models'] = models
        brands = Brand.objects.all()
        kwargs['brands'] = brands
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.carmodel = get_object_or_404(Carmodel,
id=self.kwargs['model'])
        return super().form_valid(form)

#відправлення водієм повідомлення про подорож
class DriverChatCreateView (LoginRequiredMixin, CreateView):
    model = Drivercar
    fields = ['message']
    template_name = 'adddchat.html'

    def get_success_url(self):

```

```

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

    def get_context_data(self, **kwargs):
        passeng = get_object_or_404(Passenger,
id=self.kwargs['passenger'])
        kwargs['passenger'] = passeng
        trip = get_object_or_404(Trip, id=self.kwargs['trip'])
        kwargs['trip'] = trip
        return super().get_context_data(**kwargs)

    def form_valid(self, form):
        self.passenger = get_object_or_404(Passenger,
id=self.kwargs['passenger'])
        self.trip = get_object_or_404(Trip,
id=self.kwargs['trip'])
        return super().form_valid(form)

#відправлення пасажиром повідомлення про подорож
class PassengerChatCreateView (LoginRequiredMixin, CreateView):
    model = Drivercar
    fields = ['message']
    template_name = 'addpchat.html'

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

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

    def form_valid(self, form):
        self.passenger = Passenger.objects.get(id=
self.request.user)
        self.trip = get_object_or_404(Trip,
id=self.kwargs['trip'])
        return super().form_valid(form)

#повідомлення про подорож для водія
class DriverChatsView (ListView):
    model = Chat
    template_name = 'driverchats.html'

```

```

def get_queryset(self):
    trip = self.request.GET.get('trip')
    ftrip = Trip.objects.get(id = trip)
    driver = Driver.objects.filter(id = self.request.user)
    if driver == ftrip.drivercar.driver:
        chats = Chat.objects.filter(trip = ftrip)
        chats = chats.order_by('-created')
    return chats

def get_context_data(self, **kwargs):
    chatdata = super(DriverChatsView,
self).get_context_data(**kwargs)
    trip = self.request.GET.get('trip')
    ftrip = Trip.objects.get(id = trip)
    chatdata ['trip'] = ftrip
    return chatdata

#повідомлення про подорож для пасажирів
class PassengerChatsView (ListView):
    model = Chat
    template_name = 'passchats.html'

def get_queryset(self):
    passenger = self.request.user
    apassenger = Passenger.objects.get(id = passenger)
    ftrip = Trip.objects.get(id = trip)
    chats = Chat.objects.filter(trip = ftrip, passenger =
apassenger)

    chats = chats.order_by('-created')
    return chats

def get_context_data(self, **kwargs):
    chatdata = super(PassengerChatsView,
self).get_context_data(**kwargs)
    trip = self.request.GET.get('trip')
    ftrip = Trip.objects.get(id = trip)
    chatdata ['trip'] = ftrip
    return chatdata

#пошук подорожей
class SearchTripView (ListView):
    model = Trip
    template_name = 'searchtrip.html'

def get_queryset(self):

```

```

        tripdate = self.request.GET.get('tripdate')
        tripdate = datetime.datetime.strptime(tripdate,
"%d.%m.%Y").date()
        trips = Trip.objects.filter(start_time__date =
tripdate)

        minstart = self.request.GET.get('minstart')
        minstart = datetime.datetime.strptime(minstart,
"%d.%m.%y %H:%M")
        maxstart = self.request.GET.get('maxstart')
        maxstart = datetime.datetime.strptime(maxstart,
"%d.%m.%y %H:%M")
        maxend = self.request.GET.get('maxend')
        maxend = datetime.datetime.strptime(maxend, '%d.%m.%y
%H:%M')

        aplaces = self.request.GET.get('places')
        aplaces = int(aplaces)
        trips = trips.filter(active_places__gte = aplaces)
        maxdistance = self.request.GET.get('maxdistance')
        minprice = self.request.GET.get('minprice')
        minprice = float(minprice)
        maxprice = self.request.GET.get('maxprice')
        maxprice = float(maxprice)
        startpoint = self.request.GET.get('startpoint')
        point1 = Point.objects.filter(id = startpoint)
        endpoint = self.request.GET.get('endpoint')
        point2 = Point.objects.filter(id = endpoint)
        gtrips = []
        for gtrip in trips:
            mindist = None
            startpointn = None
            mindist2 = None
            endpointn = None
            ways = Way.objects.filter(trip = gtrip)
            for way in ways:
                p1 = (way.pointx, way.pointy)
                p2 = (point1.pointx, point1.pointy)
                p3 = (point2.pointx, point2.pointy)
                dist = hav.haversine(way, p2)
                dist2 = hav.haversine(way, p3)
                if mindist and dist < mindist:
                    mindist = dist
                    startpointn = way
                if mindist2 and dist2 < mindist2:
                    mindist2 = dist2
                    endpointn = way

```

```

        res = False
        if mindist <= maxdistance and mindist2 <=
maxdistance and startpointn.num_in_trip < endpointn.num_in_trip and
startpointn.plan_departure >= minstart and startpointn.plan_departure
<= maxstart and endpointn.plan_arrive <= maxend:
            res = True
            if not (res and gtrip.price <= maxprice):
                rps = TripPrice.objects.filter(start_way =
point1, end_way = point2, price__lte = maxprice, price__gte =
minprice)

                if not rps:
                    res = False
            if res:
                gtrips.append(gtrip)
trips = gtrips
notes = self.request.GET.get('notes')
notes = notes.split(',')
sepnoses = []
for note in notes:
    gnote = Note.objects.get(id = note)
    sepnoses.append(gnote)
newtrips = []
for trip in trips:
    gdriver = trip.drivercar.driver
    dnotes = DriverNote.objects.filter(driver =
gdriver)

    res = True
    for snote in sepnoses:
        if snote not in dnotes:
            res = False

    if res:
        newtrips.append(trip)
trips = newtrips
trips = trips.order_by('start_time')
return trips

def get_context_data(self, **kwargs):
    tripdata = super(SearchTripView,
self).get_context_data(**kwargs)
    startpoint = self.request.GET.get('startpoint')
    point1 = Point.objects.filter(id = startpoint)
    endpoint = self.request.GET.get('endpoint')
    point2 = Point.objects.filter(id = endpoint)
    tripdata ['startpoint'] = point1
    tripdata ['endpoint'] = point2

```

```

        return tripdata

#створення запиту на подорож
class RequestCreateView (LoginRequiredMixin, CreateView):
    model = Request
    template_name = 'addrequest.html'

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

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

    def form_valid(self, form):
        self.start_way = Way.objects.get(id =
self.kwargs['start_way'])
        self.end_way = Way.objects.get(id =
self.kwargs['end_way'])
        self.trip = Trip.objects.get(id = self.kwargs['trip'])
        self.created = date.today()
        self.state = 'N'
        passenger = self.request.user
        apassenger = Passenger.objects.get(id = passenger)
        self.passenger = apassenger
        self.differ = False
        return super().form_valid(form)

#редагування запиту на подорож
class RequestUpdateView (LoginRequiredMixin, UpdateView):
    model = Request
    template_name = 'request.html'

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

    def form_valid(self, form):
        self.start_way = Way.objects.get(id =
self.kwargs['start_way'])
        self.end_way = Way.objects.get(id =
self.kwargs['end_way'])
        return super().form_valid(form)

```

```

#підтвердження запиту на подорож
class RequestAcceptView (LoginRequiredMixin, UpdateView):
    model = Request
    template_name = 'acceptrequest.html'

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

    def form_valid(self, form):
        driver = self.request.user
        gdriver = Driver.objects.get(id = driver)
        req = Request.objects.get(id = self.pk)
        if gdriver == req.trip.drivercar.driver:
            self.state = 'A'
        return super().form_valid(form)

#скасування запиту на подорож
class RequestCancelView (LoginRequiredMixin, UpdateView):
    model = Request
    template_name = 'acceptrequest.html'

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

    def form_valid(self, form):
        driver = self.request.user
        gdriver = Driver.objects.get(id = driver)
        req = Request.objects.get(id = self.pk)
        if gdriver == req.trip.drivercar.driver:
            self.state = 'C'
        return super().form_valid(form)

#перелік запитів на подорож
class TripRequestsView (ListView):
    model = Request
    template_name = 'triprequests.html'

    def get_queryset(self):
        trip = self.request.GET.get('trip')
        ftrip = Trip.objects.get(id = trip)
        driver = self.request.user
        gdriver = Driver.objects.get(id = driver)
        if gdriver == ftrip.drivercar.driver:

```

```

        requests = Request.objects.filter(trip = ftrip)
        requests.order_by('-created')
        return requests

    def get_context_data(self, **kwargs):
        tripdata = super(TripRequestsView,
self).get_context_data(**kwargs)
        trip = self.request.GET.get('trip')
        ftrip = Trip.objects.get(id = trip)
        tripdata ['trip'] = ftrip
        return tripdata

#власні запити на подорож авторизованого пасажира
class OwnRequestsView (ListView):
    model = Request
    template_name = 'ownrequests.html'

    def get_queryset(self):
        passenger = self.request.user
        apassenger = Passenger.objects.get(id = passenger)
        requests = Request.objects.filter(passenger =
apassenger)
        requests = requests.order_by('-created')
        return requests

#власні підтверджені запити
class OwnAcceptedRequestsView (ListView):
    model = Request
    template_name = 'ownacrequests.html'

    def get_queryset(self):
        passenger = self.request.user
        apassenger = Passenger.objects.get(id = passenger)
        requests = Request.objects.filter(passenger =
apassenger, state = 'A')
        requests = requests.order_by('-created')
        return requests

#профіль водія
class DriverProfileView (DetailView):
    template_name = 'driver.html'
    model = Driver

#профіль пасажира
class PassengerProfileView (DetailView):

```

```

        template_name = 'passenger.html'
        model = Passenger

#додавання пункту маршруту до запиту зі зміною маршруту
class RequestwayCreateView (LoginRequiredMixin, CreateView):
    model = Requestway
    fields = ['num_in_trip', 'pointx', 'pointy', 'address']
    template_name = 'addreqway.html'

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

    def form_valid(self, form):
        request = self.request.GET.get('request')
        self.request = request
        request.differ = True
        map_point = self.request.GET.get('point')
        self.map_point = map_point
        return super().form_valid(form)

#редагування пункту маршруту з запиту зі зміною маршруту
class RequestwayUpdateView (LoginRequiredMixin, UpdateView):
    model = Requestway
    fields = ['num_in_trip', 'pointx', 'pointy', 'address']
    template_name = 'updatereqway.html'

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

    def form_valid(self, form):
        map_point = self.request.GET.get('point')
        self.map_point = map_point
        request.differ = True
        return super().form_valid(form)

#додавання пункту маршруту до подорожі
class WayCreateView (LoginRequiredMixin, CreateView):
    model = Way
    fields = ['num_in_trip', 'pointx', 'pointy', 'plan_arrive',
'plan_departure', 'address']
    template_name = 'addway.html'

    def get_success_url(self):

```

```

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

    def form_valid(self, form):
        trip = self.request.GET.get('trip')
        self.trip = trip
        point = self.request.GET.get('point')
        self.point = point
        return super().form_valid(form)

#редагування пункту маршруту в подорожі
class WayUpdateView (LoginRequiredMixin, UpdateView):
    model = Way
    fields = ['num_in_trip', 'pointx', 'pointy', 'plan_arrive',
'plan_departure', 'address']
    template_name = 'updateway.html'

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

    def form_valid(self, form):
        point = self.request.GET.get('point')
        self.point = point
        return super().form_valid(form)

#робота над додаванням пасажира
class PassengerCreateView (LoginRequiredMixin, CreateView):
    model = Passenger
    fields = ['username', 'password', 'first_name', 'last_name',
'phone']
    template_name = 'passreg.html'

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

    def form_valid(self, form):
        self.trips = 0
        self.average_score = 0
        self.reviews_number = 0
        self.positive_reviews = 0
        self.negative_reviews = 0
        self.membered = date.today()
        return super().form_valid(form)

```

```

#робота над зміною профілю пасажира
class PassengerUpdateView (LoginRequiredMixin, UpdateView):
    model = Passenger
    fields = ['password', 'first_name', 'last_name', 'phone',
'passport']
    template_name = 'passupdate.html'

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

#робота над створенням профілю водія
class DriverCreateView (LoginRequiredMixin, CreateView):
    model = Driver
    fields = ['username', 'password', 'first_name', 'last_name',
'phone', 'level']
    template_name = 'driverreg.html'

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

    def form_valid(self, form):
        self.trips = 0
        self.average_score = 0
        self.reviews_number = 0
        self.positive_reviews = 0
        self.negative_reviews = 0
        self.membered = date.today()
        return super().form_valid(form)

#робота над зміною профілю водія
class DriverUpdateView (LoginRequiredMixin, UpdateView):
    model = Driver
    fields = ['password', 'first_name', 'last_name', 'phone',
'passport', 'license', 'level']
    template_name = 'driverupdate.html'

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

    def form_valid(self, form):
        notes = self.kwargs['drivernotes']
        sepnotes = []

```

```

    for note in notes:
        gnote = Note.objects.get(id = note)
        sepnotes.append(gnote)
    driver = self.request.user
    gdriver = Driver.objects.get(id = driver)
    for gnote in sepnotes:
        dc = Drivercar (driver = gdriver, note = gnote)
        if not dc:
            dc = Drivercar.objects.create(driver =
gdriver, note = gnote)
            dc.save()
    notes = Drivercar.objects.filter(id = gdriver)
    for gnote in notes:
        if gnote not in sepnotes:
            Drivercar.objects.get(driver = gdriver, note
= gnote).delete()
    return super().form_valid(form)

```

ДОДАТОК В
Слайди презентації

Національний університет «Запорізька політехніка»
Кафедра програмних засобів

Програмне забезпечення пошуку попутників для подорожей автомобілем

Виконав
студент групи
КНТ-129сп

О.В. Сердюк

Керівник
к.т.н., доцент

С.Ю. Скрупський

2022

Рисунок В.1 – Слайд № 1

Об'єкт, предмет, мета роботи

Об'єкт дослідження – процес спільного подорожування автомобілем.

Предмет дослідження – програмне забезпечення пошуку попутників для подорожей автомобілем.

Мета дослідження – розробити програмне забезпечення пошуку попутників для подорожей автомобілем для спільного задоволення транспортних потреб людей.

2

Рисунок В.2 – Слайд № 2

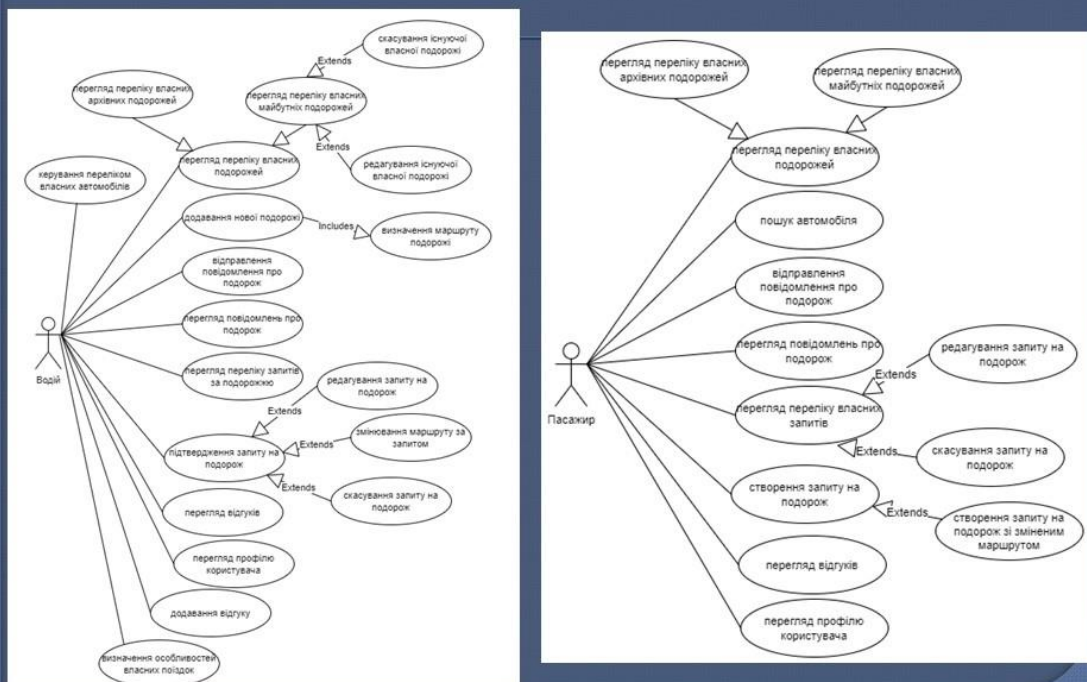
Завдання роботи

- ✓ дослідження проблеми пошуку попутників для подорожей автомобілем;
- ✓ визначення засобів розробки;
- ✓ визначення вимог до програми;
- ✓ проектування програми пошуку попутників для подорожей автомобілем;
- ✓ програмна реалізація.

3

Рисунок В.3 – Слайд № 3

Діаграми варіантів використання



4

Рисунок В.4 – Слайд № 4

Вибір мови програмування		
Показник	Python	PHP
Типізація	Динамічна	Динамічна
Читабельність коду	Висока	Середня
Захищеність застосунків	Вища за рахунок Django	Нижча
Рівень підтримки	Порівняний	Порівняний
Розширюваність	Вища	Порівняно дещо нижча

5

Рисунок В.5 – Слайд № 5



6

Рисунок В.6 – Слайд № 6



Рисунок В.7 – Слайд № 7



Рисунок В.8 – Слайд № 8

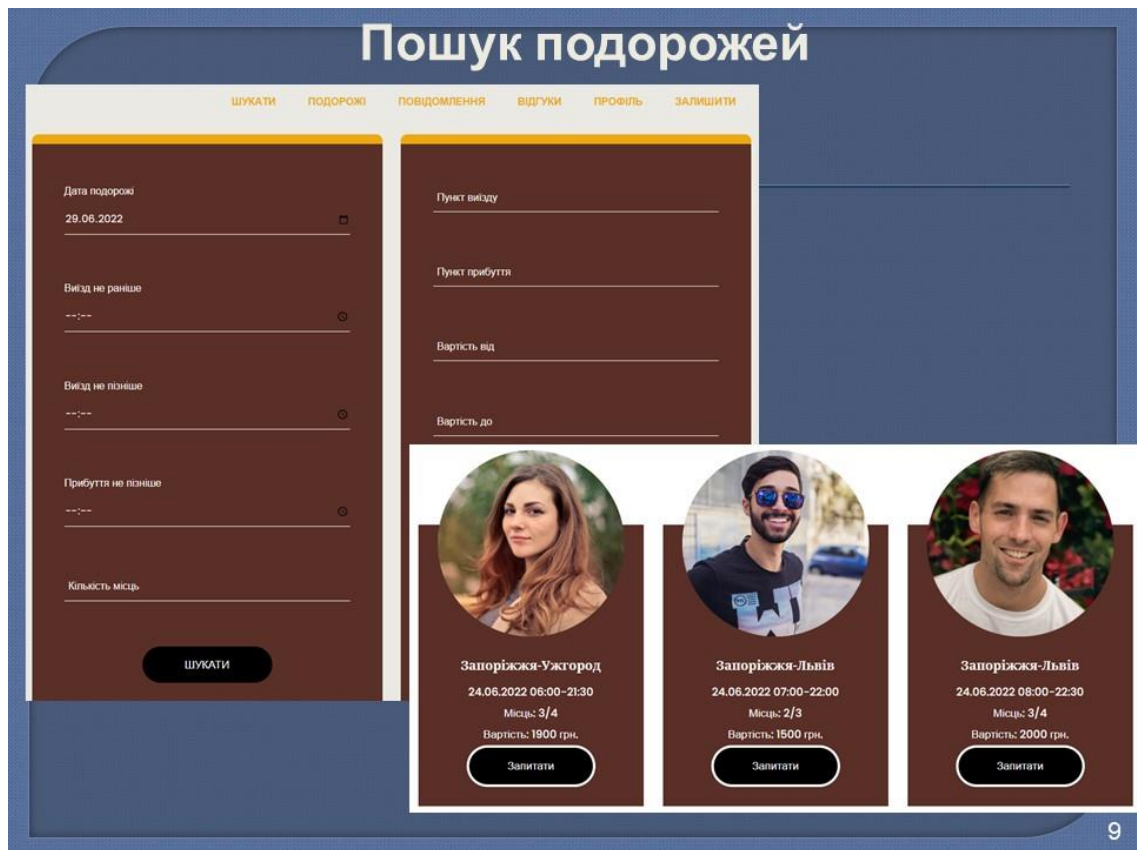


Рисунок В.9 – Слайд № 9

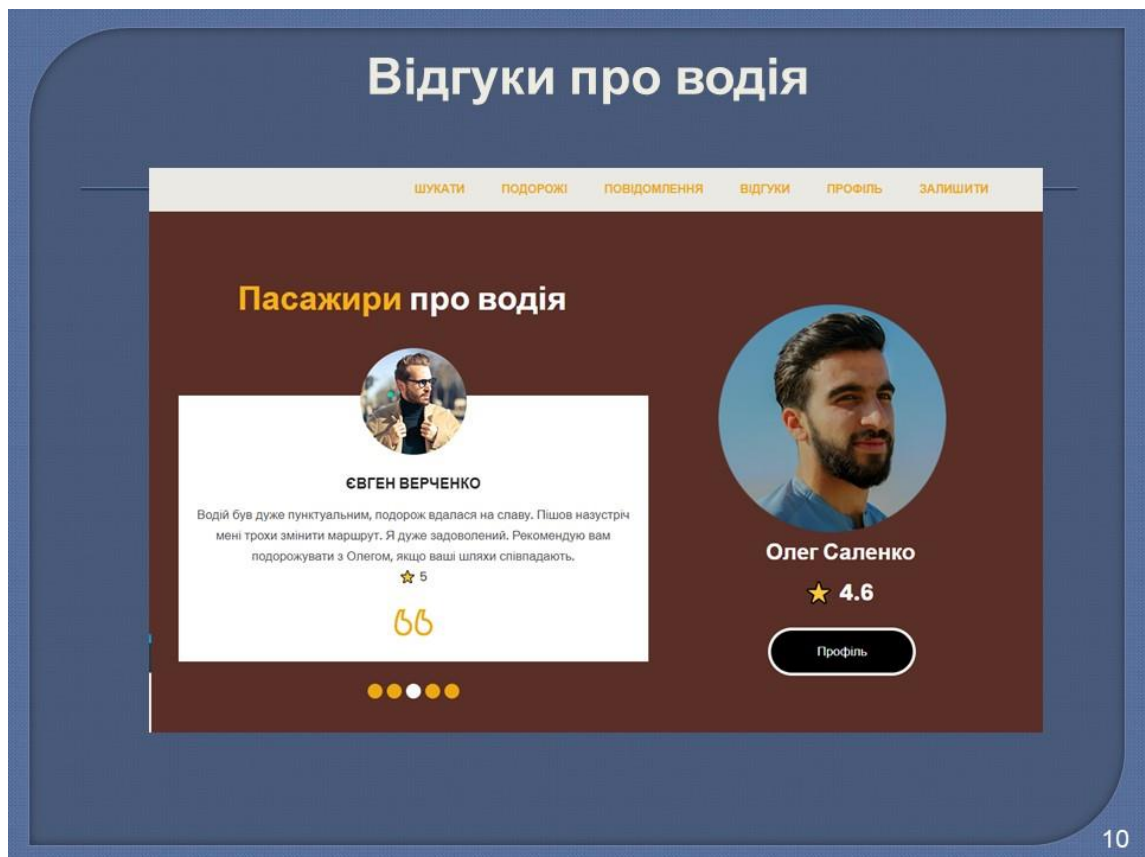


Рисунок В.10 – Слайд № 10

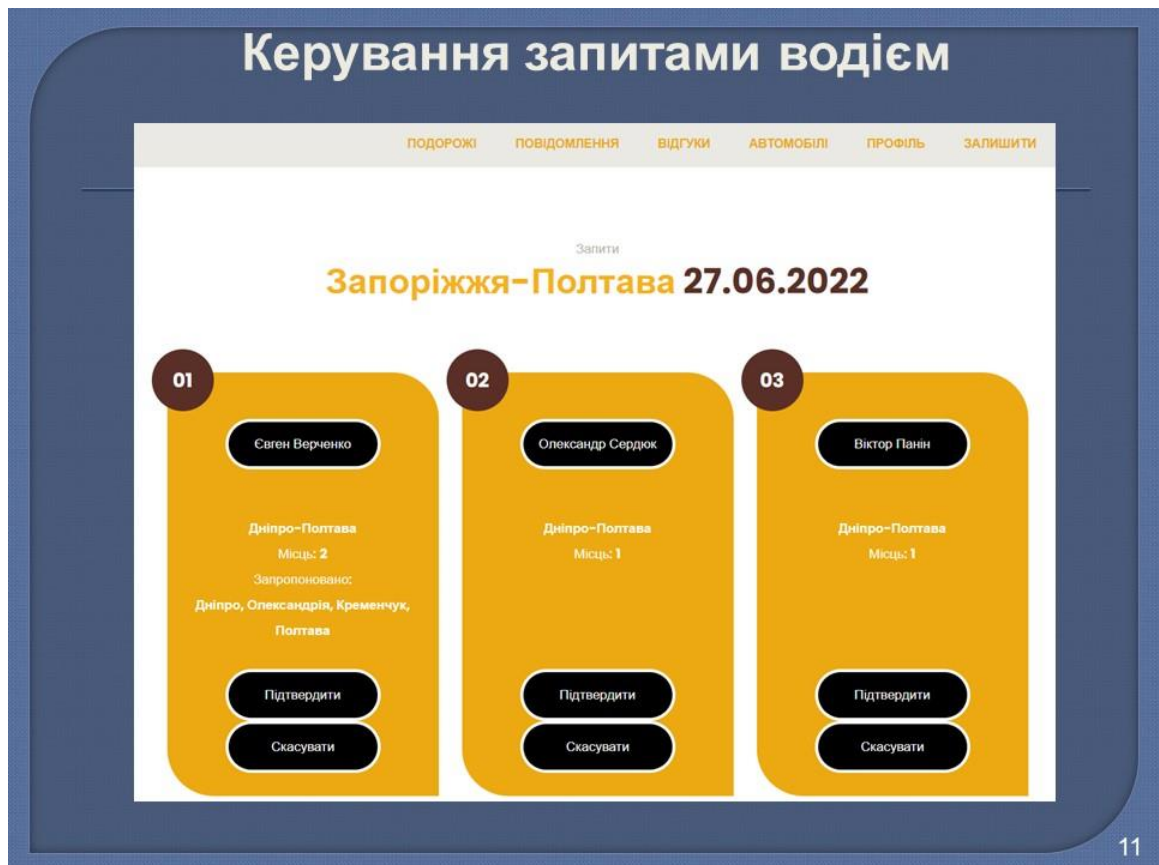


Рисунок В.11 – Слайд № 11

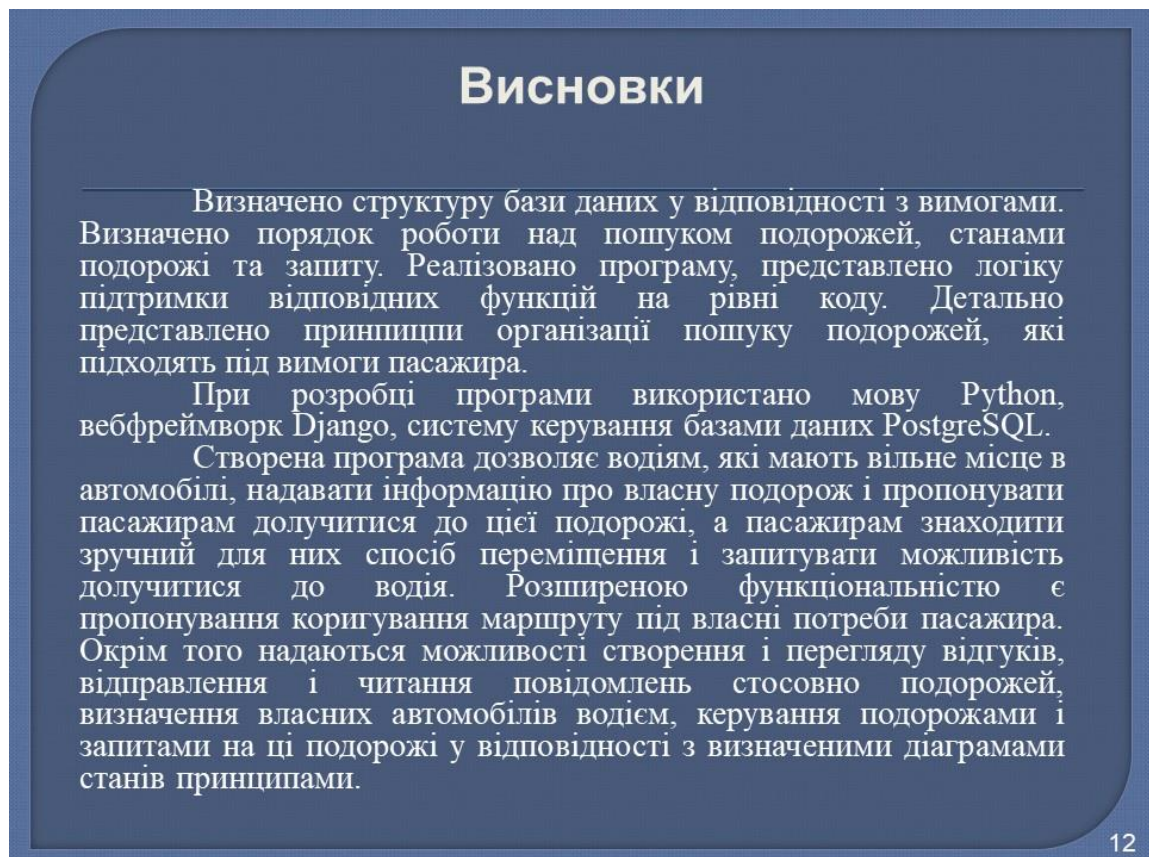


Рисунок В.12 – Слайд № 12