

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет комп'ютерних наук і технологій

(повне найменування інституту, назва факультету)

Кафедра програмних засобів

(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)

магістра

(ступінь вищої освіти)

на тему ДОСЛІДЖЕННЯ ТА ПРОГРАМНА РЕАЛІЗАЦІЯ МЕТОДІВ
ПРОГНОЗУВАННЯ ДОСТУПНОСТІ ПАРКУВАЛЬНИХ МІСЦЬ
RESEARCH AND SOFTWARE IMPLEMENTATION OF METHODS FOR
PREDICTING AVAILABILITY OF PARKING LOTS

Виконав: студент(ка) 2 курсу, групи КНТ-121м

Спеціальності 121 Інженерія програмного
забезпечення

(код і найменування спеціальності)

Освітня програма (спеціалізація)

Інженерія програмного забезпечення

Онуфрієв М.С.

(прізвище та ініціали)

Керівник Скрупський 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 магістр
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)

Онуфрієва Михайла Сергійовича

(прізвище, ім'я, по батькові)

1. Тема проєкту (роботи) Дослідження та програмна реалізація методів
прогнозування доступності паркувальних місць
Research and Software Implementation of Methods for Predicting Availability of
Parking Lots

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 “4” листопада 2022 року № 363

2. Строк подання студентом проєкту (роботи) грудень 2022 року

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

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити) 1. Аналіз проблеми та постановка завдань дослідження.
2. Матеріали і методи. 3. Проєктування програмного забезпечення. 4. Основні
рішення щодо реалізації компонентів системи. 5. Експлуатація, тестування та
експериментальне дослідження 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
1-5 Основна частина	Скрупський С.Ю., доцент		
Нормоконтролер	Дейнега Л.Ю., ст. викладач		

7. Дата видачі завдання “1” вересня 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 та дослідження предметної області	2-5 тижні	Розділи 1, 2
3	Розробка архітектури програми	6 тиждень	Розділ 3
4	Розробка програми	7-9 тижні	Розділ 4
5	Тестування та експериментальне дослідження програми	10-11 тижні	Розділ 5
6	Оформлення пояснювальної записки та документів до неї. Нормоконтроль та рецензування	12 тиждень	Додатки
7	Захист роботи	13 тиждень	

Студент(ка)

(підпис) Онуфрієв М.С.
(прізвище та ініціали)

Керівник проєкту (роботи)

(підпис) Скрупський С.Ю.
(прізвище та ініціали)

РЕФЕРАТ

Пояснювальна записка до дипломної кваліфікаційної роботи магістра:
97 с., 26 рис., 3 табл., 3 дод., 16 джерел.

ПАРКУВАЛЬНИЙ ПРОСТІР, БРОНЮВАННЯ, ЗАДАЧА
ПРОГНОЗУВАННЯ, XGBOOST, ВЕБФРЕЙМВОРК.

Об'єкт роботи – процес планування паркування автомобілів. Предмет роботи – методи прогнозування доступності паркувальних місць.

Мета роботи – розробити програмне забезпечення бронювання і прогнозування доступності паркувальних місць для надання можливості водіям приймати рішення про паркування автомобілів через певний період часу.

Матеріали, методи та технічні засоби: мова Python, бібліотеки xgboost, pandas, фреймворк веброзробки Django, система керування базами даних PostgreSQL.

Результати. Наукова новизна роботи сформована в методі прогнозування доступності паркувальних місць, який виконує прогнозування мінімальної кількості доступних паркувальних місць протягом години, яка починається через годину від поточного моменту, використовуючи алгоритм машинного навчання XGBoost, що дозволяє в підсумку водію обґрунтовано приймати рішення про необхідність бронювання паркувального місця завчасно перед поїздкою або уникати такої необхідності.

Висновки. Представлене за результатами роботи програмне забезпечення охоплює підтримку керування даними про паркувальний простір, бронювання місць на паркувальному просторі й керування цими бронюваннями у вигляді вебдодатку та надає скрипти, що автоматично запускаються для прогнозування доступності місць та формування звітів.

Галузь використання – паркування автомобілів, паркувальні простори.

ABSTRACT

Explanatory note to the diploma qualifying work of the master: 97 pages, 26 figures, 3 tables, 3 appendixes, 16 sources.

PARKING SPACE, BOOKING, FORECASTING PROBLEM, XGBOOST, WEBFRAMEWORK.

The object of the work is the process of planning car parking.

The subject of the work are methods of predicting availability of parking lots.

The purpose of the work is to develop software for booking and predicting the availability of parking spaces to enable drivers to make decisions on parking cars in the future.

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

The results. The scientific novelty of the work is presented by the method of predicting the availability of parking lots, which performs the prediction of the minimum number of available parking lots during an hour which begins in an hour from the current moment, using the XGBoost machine learning algorithm, which ultimately allows a driver to make a decision on the need to book a parking lot in advance before traveling or to avoid such a need based on facts.

Conclusions. The resulting software supports parking space data management, parking space booking and management of these bookings as a web application, and provides auto-run scripts for space availability forecasting and reporting.

Field of use is car parking and parking spaces.

ЗМІСТ

	С.
Перелік скорочень та умовних познач.....	8
Вступ.....	9
1 Аналіз проблеми та постановка завдань дослідження	11
1.1 Проблема паркування в сучасному місті.....	11
1.2 Огляд програмного забезпечення для пошуку паркувальних просторів.....	14
1.3 Висновки за розділом 1	16
2 Матеріали і методи.....	17
2.1 Задача прогнозування доступності паркувальних місць	17
2.2 Існуючі методи прогнозування доступності паркувальних місць	17
2.3 Алгоритм XGBoost.....	19
2.4 Метод прогнозування доступності паркувальних місць.....	22
2.5 Висновки за розділом 2	23
3 Проектування програмного забезпечення	24
3.1 Вибір рішень для розробки програмного забезпечення.....	24
3.2 Визначення функціональних вимог до програми.....	25
3.3 Визначення структури бази даних	30
3.4 Висновки за розділом 3	41
4 Основні рішення щодо реалізації компонентів системи.....	42
4.1 Алгоритм функціонування програми.....	42
4.2 Основні реалізовані класи	44
4.3 Висновки за розділом 4	52
5 Експлуатація, тестування та експериментальне дослідження програми	53
5.1 Опис роботи з програмою	53
5.2 Експериментальне дослідження	56
5.3 Висновки за розділом 5	59
Висновки	60
Перелік джерел посилання	61

Додаток А Технічне завдання	63
Додаток Б Текст програми	68
Додаток В Слайди презентації.....	90

ПЕРЕЛІК СКОРОЧЕНЬ ТА УМОВНИХ ПОЗНАК

XGBoost – eXtreme Gradient Boost;

БД – база даних.

ВСТУП

Паркування автомобіля в сучасному місті переростає в окремий процес, який вимагає окремих ресурсів. Такі ресурси включають зокрема грошові ресурси, адже парковка може бути безкоштовною, а може і потребувати сплати, і часові ресурси, оскільки пошук паркувального місця поблизу місця призначення водія може бути достатньо тривалим, а після цього водій або пасажери ще мають витратити певний час на те, щоб дістатися (в найкращому випадку дійти) остаточного місця призначення. Тому якщо даний процес не планувати, то він може призвести до значної втрати таких ресурсів.

Переростання паркування автомобіля в окремий процес і відповідна необхідність розглядати його як процес прийняття рішення водієм пов'язано з тим, що місць у сучасному місті для паркування не вистачає. Це визначається тим, що автомобіль протягом доби або в більш тривалі періоди переміщується за певними маршрутами і потребує тимчасової стоянки в різних місцях. Розташування цих місць пов'язано з життєвими процесами людей і тим, як ці процеси і відповідні потоки влаштовані в різних містах. В одних містах ці потоки можуть потребувати переміщення на значні ділянки, наприклад, з роботи додому, до магазинів тощо. В інших містах ці потоки можуть бути не такими довгими і концентруватися в районах проживання людини. Проте і в тих, і в тих випадках, якщо людина користується автомобілем, автомобіль потрібно паркувати і різних місцях. Тільки в другому випадку підвищується ймовірність того, що людина уникне використання автомобіля. Проте в підсумку кількість автомобілів в одних місцях зазвичай є достатньо великою, а протягом доби автомобіль переміщується і відповідно точки концентрації автомобілів змінюється. У підсумку підвищена необхідність паркувальних місць виникає то на одній ділянці, то на іншій, а кількість таких ділянок через велику кількість автовласників у місті є достатньо великою.

Відповідно водію потрібна інформація про те, чи може він розраховувати на паркування на певній ділянці міста. Так він може вирішити, де йому краще залишити свій автомобіль. Для цього потрібний механізм прогнозування доступності паркувальних місць. Такий же механізм важливим є і для осіб, які володіють або керують парковками. В даній роботі потрібно створити такий механізм і забезпечити його практичне застосування за рахунок створення відповідного програмного забезпечення.

1 АНАЛІЗ ПРОБЛЕМИ ТА ПОСТАНОВКА ЗАВДАНЬ ДОСЛІДЖЕННЯ

1.1 Проблема паркування в сучасному місті

Паркування в сучасному місті є проблемою, яка охоплює цілий набір факторів:

- тривога щодо того, як знайти місце для паркування автомобіля на тривалий час, у водіїв викликає хвилювання та стрес, особливо коли водій запізнюється на важливу подію, у підсумку багато людей скаржаться на брак місць для паркування особливо у великому місті;

- проблема власного транспорту нині полягає не лише у завантаженості вулиць у місті, а й у нестачі паркувальних місць у різних мікрорайонах, у тому числі у країнах, в яких кількість таких паркувальних місць традиційно є достатньо великою, як наприклад, у Сполучених Штатах Америки, тому відсутність паркувальних місць можна вважати однією з найбільших проблем життя в жвавих районах;

- потенційними рішеннями проблеми можуть бути або збільшення кількості місць для паркування, або застосовування сучасних технологій для кращого управління ними: фактично проблеми з пошуком місць для паркування не завжди пов'язані з їх дефіцитом, тому важливим є використання відповідних технологій, які б дозволили знаходити місця для паркування замість пошуку їх на місці [1].

Існує розрив між попитом на паркування та пропозицією, який пояснюється декількома причинами:

- більшість старих та історичних міст, особливо столиць, були сплановані з вузькими вулицями, де не було машин, а були вози, які рухалися за допомогою коней, крім того щільність населення цих міст у той час була не такою високою порівняно з теперішньою щільністю тих самих міст і тих самих районів: оскільки міські вулиці найчастіше не можуть бути повністю змінені, вони за старого планування розміщують всі види транспортних засобів з високою щільністю руху та паркування, навантаження, яке перевищує їх заплановану пропускну здатність;

– концентрація діяльності та об'єктів, які вимагають великої кількості автомобілів в одній зоні, наприклад, концентрація комерційних об'єктів з офісними будівлями та державними установами в центральному районі міста;

– у нових містах і нових планованих міських районах часто існує неправильний розрахунок попиту на паркування, який очікується в цих районах, через несподіваний підвищений рівень володіння автомобілями, особливо серед населення з високим і середнім рівнем доходу: це пов'язано з тим, що система масового транспорту, запропонована в цих нових районах, не змусила людей цих класів використовувати її під час своїх поїздок на роботу чи для відпочинку;

– у планах нових міст і нових міських районів тенденція забезпечення паркомісцями часто залежить від бордюрів та паркувальних зон на рівні вулиці, немає земельних ділянок під багатоповерхові паркувальні споруди для поглинання зростаючої кількості автомобілів, які шукають місця для паркування, принаймні в місцях зосередженого об'єкта;

– порушення будівельних норм і правил зонування, які передбачають для кожного району конкретне використання будівель і конкретну кількість поверхів із забезпеченням гаражів у підвалах, – це порушення сприяє зміні всіх розрахунків проєктувальників щодо забезпечення достатніх місць для паркування автомобілів на цих територіях [2].

У кожному місті є простір, який займають самі автомобілі, при цьому середній автомобіль, який складається з двох тон сталі, на 80 % порожній, коли ним керує одна людина, при цьому більшу частину дня машини абсолютно порожні, не використовуються, але для цього потрібен простір для паркування: у Сполучених Штатах є мільярд паркувальних місць, по чотири на кожну наявну машину, крім того частина міста використовується на автомобільні дороги, що в підсумку призводить до того, що багато центральних частин міст виділяють від 50 до 60 % своєї дефіцитної площі на транспортні засоби [3].

На даний момент починають будуватися плани щодо створення міста майбутнього, яке включало б безпілотні автомобілі, для вирішення таких проблем, як затори та зміна клімату, і один з таких планів був представлений в Сан-Франциско, він характеризується наступним:

- зробити послуги спільного використання поїздок (включаючи, але не обмежуючись Uber і Lyft) більш зручними та доступними для жителів: якщо люди зможуть використовувати ці автомобілі для своїх транспортних потреб або поєднувати їх із варіантами громадського транспорту, їм не знадобиться стільки вуличних парковок, а оскільки кожна з цих машин обслуговує кількох пасажирів, вони займають менше місця в дорозі;

- визначення стимулів перевести людей із власних автомобілів на спільне використання автомобілів: це може означати визначення певних смуг дороги як доступних лише для спільного використання, що зробить їх швидшим варіантом, створення єдиного простого мобільного додатку, який поєднує маршрути, планування та оплату за послуги використання велосипедів, автомобілів, міського транспорту;

- зробити дані послуги доступнішими: це може передбачати надання мешканцям із низьким рівнем доходу доступу до смартфонів і банківських послуг, а також надання безкоштовного громадського Wi-Fi, зниження ціни на спільне використання автомобілів;

- перехід до автоматизованих електромобілів: якщо безпілотні автомобілі та автобуси зрештою стануть реальністю, їх також можна буде об'єднати в централізовану мережу, що спростить обмін, теоретично ці транспортні засоби можуть також зменшити кількість зіткнень зі смертельними наслідками, а також усунуть забруднення повітря [3].

Проте для вирішення проблеми паркування в сучасному місті в поточному його стані в підсумку необхідно:

- розробити програмне забезпечення, яке дозволяло б виконувати завчасно пошук паркувальних місць і за можливості бронювати ці місця, якщо це необхідно клієнту;

– визначати доступність паркувальних місць на відповідному просторі в майбутньому, щоб прийняти рішення, чи необхідне це бронювання, чи можна скористатися безкоштовним паркувальним простором, якщо такий є поблизу.

1.2 Огляд програмного забезпечення для пошуку паркувальних просторів

У процесі огляду програмного забезпечення для пошуку паркувальних просторів було розглянуто спершу Parkopedia (рис. 1.1) [4].

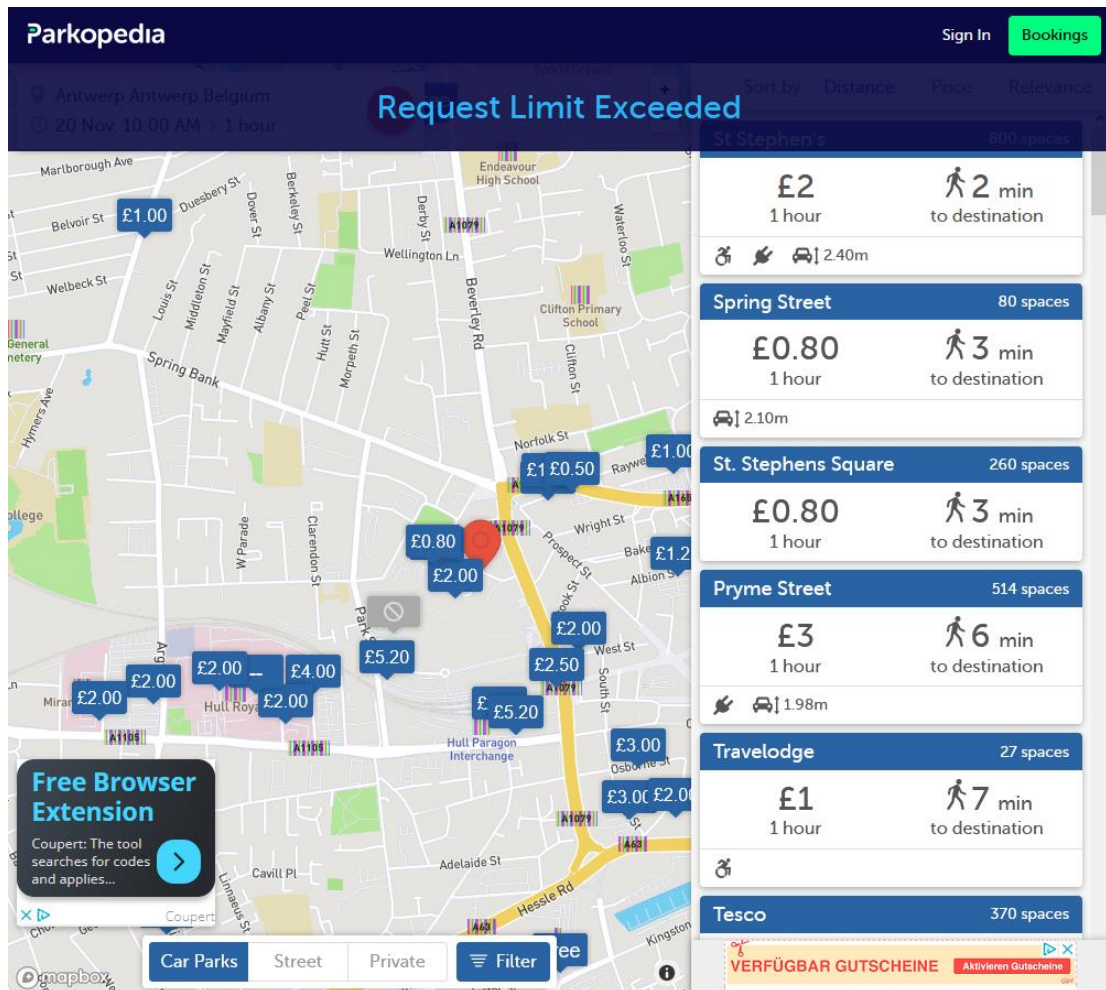


Рисунок 1.1 – Вебсайт Parkopedia

Даний вебсайт дозволяє виконувати пошук паркувальних просторів у заданому місті або за заданою адресою і на вказаний час. Для кожного результату визначається вартість такого періоду парковки, надається інформація про паркувальний простір, про тарифи на паркування загалом у різні періоди часу, про способи сплати за паркування, а також про години роботи. За авторизації або реєстрації можна також виконати бронювання місць.

Другим вебсайтом подібного спрямування є Parclick (рис. 1.2) [5]. Він має також мобільний додаток.

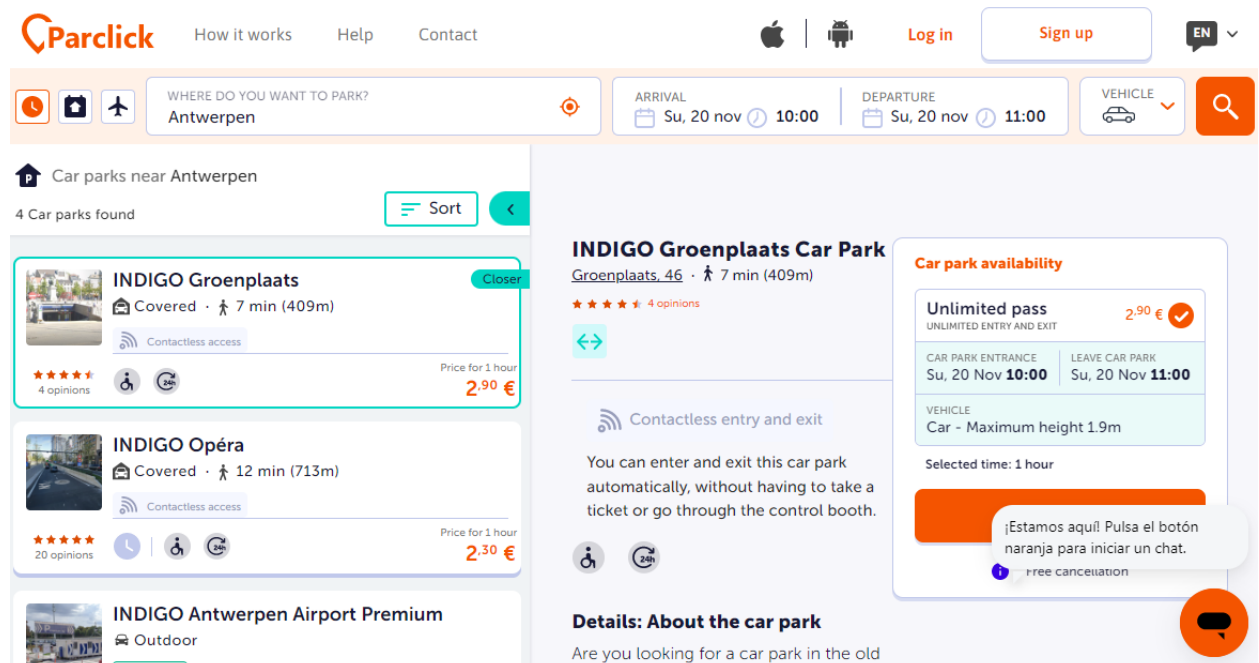


Рисунок 1.2 – Вебсайт Parclick

Parclick також дозволяє виконати пошук за адресою на вказаний час, розраховуючи вартість такої парковки. Також надається можливість забронювати паркувальне місце за додаткову плату. На відміну від попереднього вебсайту тут також клієнти можуть залишати власні відгуки про паркувальний простір. Якщо на Parkopedia під час пошуку в фільтрах основним обмеженням є висота автомобіля, то Parclick дозволяє також задати

тип автомобіля за розміром (автомобіль, автобус, мотоцикл, вантажівка до 6 метрів, вантажівка до 7 метрів, дім на колесах або фургон).

Однак необхідно визначити, що основною направленістю даних додатків є комерційне застосування (бронювання виконується за сплату) та загальне інформування (можливість пошуку місць). Але важливим обмеженням є те, що клієнт при цьому не має рекомендацій щодо того, чи зможе він знайти місце на паркувальному просторі у певний момент часу. Пошук реалізується тільки за фактом. Для розвитку даного напрямку необхідно забезпечити прогнозування доступності паркувальних місць.

1.3 Висновки за розділом 1

У відповідності з проблемою, що розглядається, за результатами виконання магістерської роботи необхідно забезпечити виконання наступних завдань:

- дослідити методи прогнозування доступності паркувальних місць;
- спроектувати програмне забезпечення бронювання і прогнозування доступності паркувальних місць;
- реалізувати програмне забезпечення.

У першому розділі виконано характеристику проблему паркування в сучасному місті, виділено основні існуючі проблемні ситуації, шляхи їх вирішення в подальшому та можливі шляхи вирішення на даний момент. Одним з таких шляхів є розробка програмного забезпечення, яке забезпечить пошук, бронювання та прогнозування доступності паркувальних місць на паркувальних просторах. Охарактеризовано існуючі вебсайти, які дозволяють вирішити проблему пошуку і бронювання паркувальних місць, виділено їх можливості.

2 МАТЕРІАЛИ І МЕТОДИ

2.1 Задача прогнозування доступності паркувальних місць

Задача прогнозування доступності паркувальних місць розглядається в даній роботі в першу чергу з точки зору водія автомобіля, який бажає прийняти рішення, чи варто йому розраховувати на безкоштовний паркувальний простір, чи необхідно скористатися платним простором і чи необхідно виконувати попередньо бронювання. Для цього водій повинен за певний період часу до приїзду отримати інформацію про те, якою є очікувана кількість вільних місць під час його приїзду. Оскільки акцент робиться на водія, то ця інформація має бути отримана за інтервал часу в першу чергу, а не на конкретну хвилину з точністю до хвилини. Це пояснюється тим, що різні додаткові фактори можуть впливати на те, в який саме час точно водій дістанеться певної точки в місці. Окрім того водія цікавить фактично те, яка кількість вільних місць є на паркувальному майданчику. Оскільки ця інформація має бути отримана про інтервал часу, то слід в підсумку говорити про мінімальну кількість доступних місць протягом періоду часу (години).

Таким чином, задача прогнозування доступності паркувальних місць полягає у прогнозуванні мінімальної кількості доступних місць на паркувальному просторі протягом години часу, що розпочинається через годину після поточної миті. Прогнозування відбувається на період через годину, щоб водій міг завчасно прийняти рішення, а на період в годину, тому що водій фактично дістанеться паркувального простору в певний інтервал часу відносно тієї миті, коли він початково це планує.

2.2 Існуючі методи прогнозування доступності паркувальних місць

На даний момент є декілька досліджень прогнозування доступності паркувальних місць, серед яких зокрема було виділено як основні [6], [7]

(табл. 2.1).

Таблиця 2.1 – Порівняння методів прогнозування доступності паркувальних місць

Критерій	Стаття [6]	Стаття [7]	Власний метод
Показник, прогнозування якого відбувається	Середня кількість зайнятих місць за годину	Кількість доступних місць на відповідну мить	Мінімальна кількість доступних (доступність) місць протягом години
Період прогнозу	Година	Година	Година
Вхідні параметри	Часовий ряд зайнятості місць, масштаб і тип паркувального простору	Номер години, день тижня, часовий ряд доступності місць протягом останніх хвилин, середнє значення і стандартне відхилення протягом минулого періоду, рівень трафіку	Район, номер години, день тижня, часовий ряд мінімальної доступності місць протягом останніх годин, рівень трафіку
Модель в основі методу	Метод опорних векторів	XGBoost	XGBoost

На відміну від існуючих методів метод, який створюється в роботі, направлений на прогнозування мінімальної кількості доступних місць протягом години, а не середню кількість [6] або фактичну кількість на певну мить [7], що дозволяє водію отримати більш корисну інформацію для його

рішень в конкретних життєвих ситуаціях. Тобто в роботах [6], [7] фактично розв'язується дещо інша задача прогнозування. Вхідними параметрами для запропонованого методу також має бути дещо інший набір порівняно з роботою [7], де також використовується алгоритм eXtreme Gradient Boosting (XGBoost). Зокрема в роботі [7] розглядається цілий набір вхідних параметрів, що стосується рівню трафіку і вхідного, і вихідного на різних дистанціях.

2.3 Алгоритм XGBoost

Дерево рішень – це тип керованої моделі машинного навчання, у якому дані постійно розподіляються відповідно до умов і який використовується як деревоподібна модель для множини рішень: дерево рішень починається з одного вузла, який далі розгалужується на можливі виходи, кожен вихід призводить до додаткових вузлів, які розгалужуються на більше можливостей, вся ця процедура надає структурі рішень деревоподібної форми [8].

XGBoost – реалізація алгоритму градієнтного бустинга на деревах рішень, яка відзначається наступним:

- хоча це послідовний алгоритм навчання, він включає паралельні обчислення в свою архітектуру для підвищення ефективності системи, а тому можна казати про те, що він оснащений як послідовними, так і паралельними архітектурами разом;
- алгоритм XGBoost є підкласом машинного навчання з учителем і є розширеним варіантом алгоритму Boosting Machine, який є підкласом алгоритму ансамбля дерев рішень, як Random Forest;
- алгоритм Random Forest запускає декілька незалежних дерев рішень паралельно та комбінує їхні результати шляхом усереднення всіх результатів, а цей підхід використовує випадкову початкову вибірку та часто називається

пакуванням, у цьому сенсі Random Forest — це алгоритм паралельного навчання;

- алгоритм Boosting Machine використовує адитивну стратегію: тобто додає одне нове дерево за раз: Boosting Machine послідовно запускає окремі прості дерева рішень, які називаються базовими учнями, тобто концептуально Boosting Machine побудована на архітектурі послідовного навчання;

- алгоритм Boosting Machine навчається послідовно, а Random Forest – паралельно;

- у рамках адитивної стратегії алгоритм Gradient Boosting Machine використовує градієнтний спуск для оптимізації: щоб зменшити обчислювальне навантаження, Gradient Boosting Machine апроксимує цільову функцію за допомогою члена першого порядку розкладання Тейлора та ігнорує будь-які члени вищого порядку для оптимізації навчання, іншими словами, він використовує першу похідну (градієнт) цільової функції (функції втрати), щоб визначити наступний предиктор слабкого навчання, таким чином, підсилення градієнта, зберігаючи існуючі слабкі предиктори, додає новий предиктор поверх них, щоб зменшити поточну помилку з метою поступового покращення продуктивності [9]-[11].

У задачах прогнозування, пов'язаних із неструктурованими даними (зображеннями, текстом тощо), штучні нейронні мережі, як правило, перевершують усі інші алгоритми, однак, коли задача полягає в роботі зі структурованими/табличними даними малого та середнього розміру, алгоритми на основі дерев рішень вважаються найкращими в своєму класі на даний момент [12].

Основними відмінностями саме даного алгоритму для роботи з деревами відносно інших існуючих, які обумовлюють його переваги, є:

- розпаралелювання: XGBoost підходить до процесу послідовної побудови дерева за допомогою розпаралеленої реалізації, це можливо завдяки взаємозамінній природі циклів, які використовуються для створення

базових учнів, так зовнішній цикл, який перераховує листові вузли дерева, і другий внутрішній цикл, який обчислює функції, таке вкладення циклів обмежує розпаралелювання, оскільки без завершення внутрішнього циклу (більш вимогливого до обчислень) зовнішній цикл не може бути запущений, тому, щоб покращити час виконання, порядок циклів змінюється за допомогою ініціалізації через глобальне сканування всіх екземплярів і сортування за допомогою паралельних потоків, це перемикає покращує продуктивність алгоритму, компенсуючи будь-які накладні витрати на розпаралелювання під час обчислень;

- обрізка дерев: критерій зупинки для розбиття дерев у рамках Gradient Boosting Machine є жадібним за своєю природою та залежить від критерію негативних втрат у точці розбиття, XGBoost використовує параметр `max_depth` замість «критерію спочатку» і починає обрізати дерева в зворотному напрямку, цей підхід «спочатку в глибину» значно покращує продуктивність обчислень;

- оптимізація апаратного забезпечення: цей алгоритм розроблено для ефективного використання апаратних ресурсів, це досягається шляхом усвідомлення кешу шляхом виділення внутрішніх буферів у кожному потоці для зберігання статистики градієнта, подальші вдосконалення оптимізують доступний дисковий простір, обробляючи великі порції даних, які не вміщуються в пам'ять;

- регуляризація: вона штрафує складніші моделі через регуляризацію LASSO (L1) і Ridge (L2), щоб запобігти перенавчанню;

- перехресна перевірка: алгоритм використовує вбудований метод перехресної перевірки на кожній ітерації, що позбавляє потреби явно програмувати цей пошук і вказувати точну кількість ітерацій посилення, необхідних за один запуск [12].

2.4 Метод прогнозування доступності паркувальних місць

Для розв'язання задачі прогнозування доступності паркувальних місць запропоновано використовувати дерева рішень на основі алгоритму XGBoost.

Для застосування дерев спочатку формується вибірка даних за виконаними спостереженнями за роботою паркувальних майданчиків, що охоплює наступні атрибути:

- район: для цього все місто має бути розбито на певні райони, для чого може використовуватися або стандартний поділ міста на райони, або відбуватися розподіл на зони активності (діловий центр, історичний центр, спальний район);
- номер години;
- номер дня тижня: в окрему категорію (номер) має бути виділено святкові дні;
- часовий ряд мінімальної доступності місць протягом останніх годин: фактично для вибірки, на якій виконувалось дослідження було запропоновано використовувати 12 останніх годин, що обумовлено тим, що менша кількість ускладнює дослідження в нічні години, коли кількість місць загалом є стабільною;
- рівень трафіку: середній рівень трафіку на навколишній території в 500 метрів;
- вихідна ознака – мінімальна кількість доступних місць, яка одномоментно була на паркувальному просторі протягом години часу, яка була 14-ою в послідовності (12 годин у часовому ряді, 13-а година пропущена, а 14-а година використовується як вихідна ознака).

Після того, як вибірку сформовано, її має бути нормалізовано, а після цього виконано навчання моделі на основі алгоритму XGBoost.

У подальшому застосовується навчена модель для прогнозування на початку години значення мінімальної кількості доступних місць протягом

години, яка розпочнеться через годину, на основі набору значень вхідних показників, які перед цим мають бути перетворені на основі правил, які застосовувались для навчальної вибірки. При цьому основний набір значень показників має бути отримано від паркувального простору, а значення трафіку має бути отримано від того сервісу, який використовувався для формування вибірки для визначення рівня трафіку (Google Maps, Here WeGo або інший). Наприклад, Here WeGo розрізняє 4 рівні трафіку:

- зелений для вільного руху;
- жовтий для ущільненого руху;
- червоний для повільного руху;
- чорний для корок або закритих доріг [13].

2.5 Висновки за розділом 2

Представлено задачу прогнозування доступності паркувальних місць, проаналізовано методи, які використовуються для розв’язання подібних задач, розглянуто алгоритм XGBoost.

Наукова новизна роботи сформована в запропонованому методі прогнозування доступності паркувальних місць, який виконує прогнозування мінімальної кількості доступних паркувальних місць протягом години, яка починається через годину від поточного моменту, використовуючи алгоритм машинного навчання XGBoost, що дозволяє в підсумку водію обґрунтовано приймати рішення про необхідність бронювання паркувального місця завчасно перед поїздкою або уникати такої необхідності.

3 ПРОЄКТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Вибір рішень для розробки програмного забезпечення

Вибір рішень для розробки програмного забезпечення виконувався комплексно, а тому порівнювались не тільки мови програмування, а і загалом обидві платформи для створення програм у всіх їх можливостях. Оскільки програма повинна бути вебдодатком, але при цьому працювати з даними, то порівнювались рішення на основі Python як найкращої мови для роботи з даними і PHP як дуже популярної мови для веброзробки.

Порівняння виконано в таблиці 3.1.

Таблиця 3.1 – Порівняння рішень для розробки вебдодатку

Критерій	Комплексне рішення на основі мови Python	Комплексне рішення на основі мови PHP
Складність написання коду	Просте вираження складних рішень	Складні рішення писати складніше, код читати також складніше
Підтримка дерев рішень	Scikit-learn, xgboost	Відсутня надійна бібліотека
Робота з даними	Велика перевага: pandas, numpy і велика кількість додаткових пакетів	Якісні рішення відсутні
Фреймворк веброзробки	Django	Laravel, Symphony, Yii

Оскільки обидві платформи мають фреймворки для веброзробки і ці фреймворки є достатньо відомими, популярними і застосовуваними, то говорити про велику перевагу за даним критерієм у PHP не слід. При цьому

варто відзначити перевагу в простоті написання і сприйняття коду особливо для достатньо складних рішень, що є на стороні Python. Тому за даним критерієм в підсумку паритет.

Відносно роботи з даними Python повністю виграє за рахунок наявності великої кількості зручних і популярних бібліотек для роботи з даними, до того ж як така популярна надійна бібліотека для роботи з деревами рішень на PHP відсутня, а на Python є навіть окрема бібліотека, що реалізує алгоритм XGBoost, що значно сприятиме розробці.

Загальний вибір прийнятий на користь створення рішення на основі мови Python і супутніх до цього фреймворка веброботки Django та бібліотек для роботи з даними і деревами рішень.

3.2 Визначення функціональних вимог до програми

Функціональні вимоги до програми є результатом аналізу аналогів, вивчення предметної області, представленого в роботі методу і повинні бути забезпечені можливостями програми:

- керувати даними паркувального простору, включаючи доступні місця, режими роботи та оплати;
- шукати паркувальні простори;
- шукати місця для паркування на заданий період;
- бронювати місця з паркувального простору;
- керувати виконаними бронюваннями;
- керування власними автомобілями, доступними для клієнта;
- надавати список бронювань клієнта;
- шукати бронювання клієнта за номером його телефону;
- надавати інформацію про зайнятість паркувальних місць у просторі;
- висловлення власних вражень про паркувальний простір після паркування;
- реєструвати і авторизовувати клієнтів у програмі;

- керувати доступом клієнтів і автомобілів;
- надавати список висловлювань про паркувальний простір за обмеженим інтервалом оцінок;
- надавати список власних висловлювань клієнта про паркувальні простори;
- авторизовувати і обмежено реєструвати операторів паркувальних просторів;
- прогнозувати наявність паркувальних місць у просторі на заданий період;
- формувати звіти про задіяність груп місць для паркування за період.

Базовим актором, який взаємодіє з програмою є користувач (рис. 3.1). Це фактично неавторизований користувач.



Рисунок 3.1 – Діаграма прецедентів користувача

Основний потік роботи користувача з програмою включає сценарії:

- пошук паркувальних просторів;
- пошук місць для паркування на заданий період;
- надання списку висловлювань про паркувальний простір;

- авторизація;
- відслідковування зайнятості паркувальних місць.

Авторизований користувач представлений актором Клієнт (рис. 3.2).

Даний актор наслідує сценарії роботи користувача.

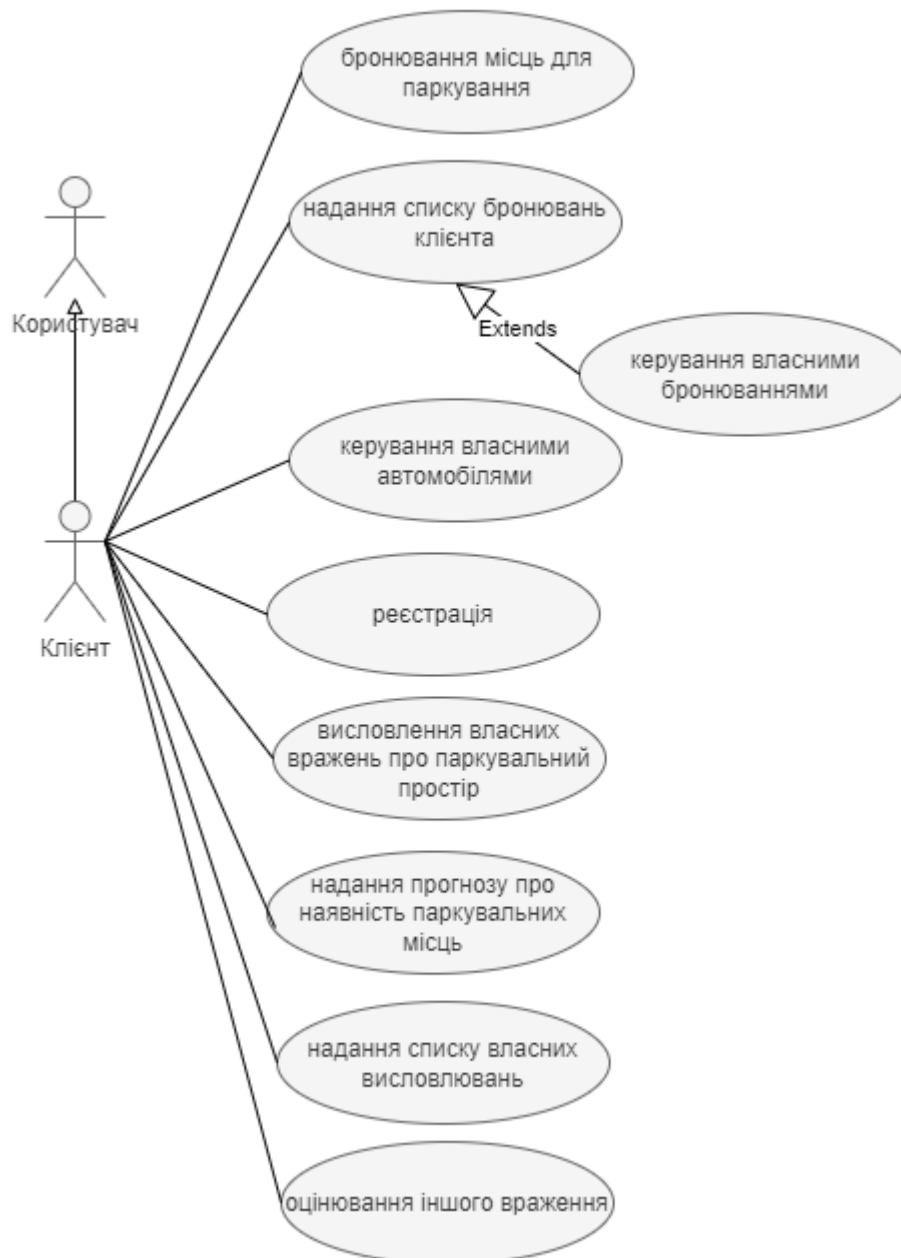


Рисунок 3.2 – Діаграма прецедентів клієнта

Основний потік роботи клієнта з програмою включає наступні сценарії, відмінні від сценаріїв роботи користувача:

- бронювання місць для паркування;

- надання списку бронювань клієнта;
- керування власними автомобілями;
- реєстрація;
- висловлення власних вражень про паркувальний простір;
- надання прогнозу про наявність паркувальних місць;
- надання списку власних висловлювань;
- оцінювання іншого враження.

Для клієнта наявний один альтернативний сценарій. Це сценарій керування власними бронюваннями. Передумовою даного сценарію є необхідність зміни раніше створеного клієнтом бронювання, яке ще не було реалізовано.

Актором, який представляє конкретний паркувальний простір, є оператор (рис. 3.3). Це фактично оператор конкретного паркувального простору.

Основний потік роботи оператора з програмою включає наступні сценарії:

- пошук бронювань клієнта за простором;
- відслідковування зайнятості паркувальних місць простору;
- керування доступом;
- керування даними паркувального простору;
- авторизація;
- надання звітів про задіяність паркувальних місць простору;
- надання прогнозу про наявність паркувальних місць за простором;
- керування бронюваннями за паркувальним простором.

Актором, який має найширше коло прав, є адміністратор (рис. 3.4). Він пов'язаний зв'язком узагальнення з оператором. При цьому він може працювати не тільки з власним паркувальним простором, але і з усіма наявними.

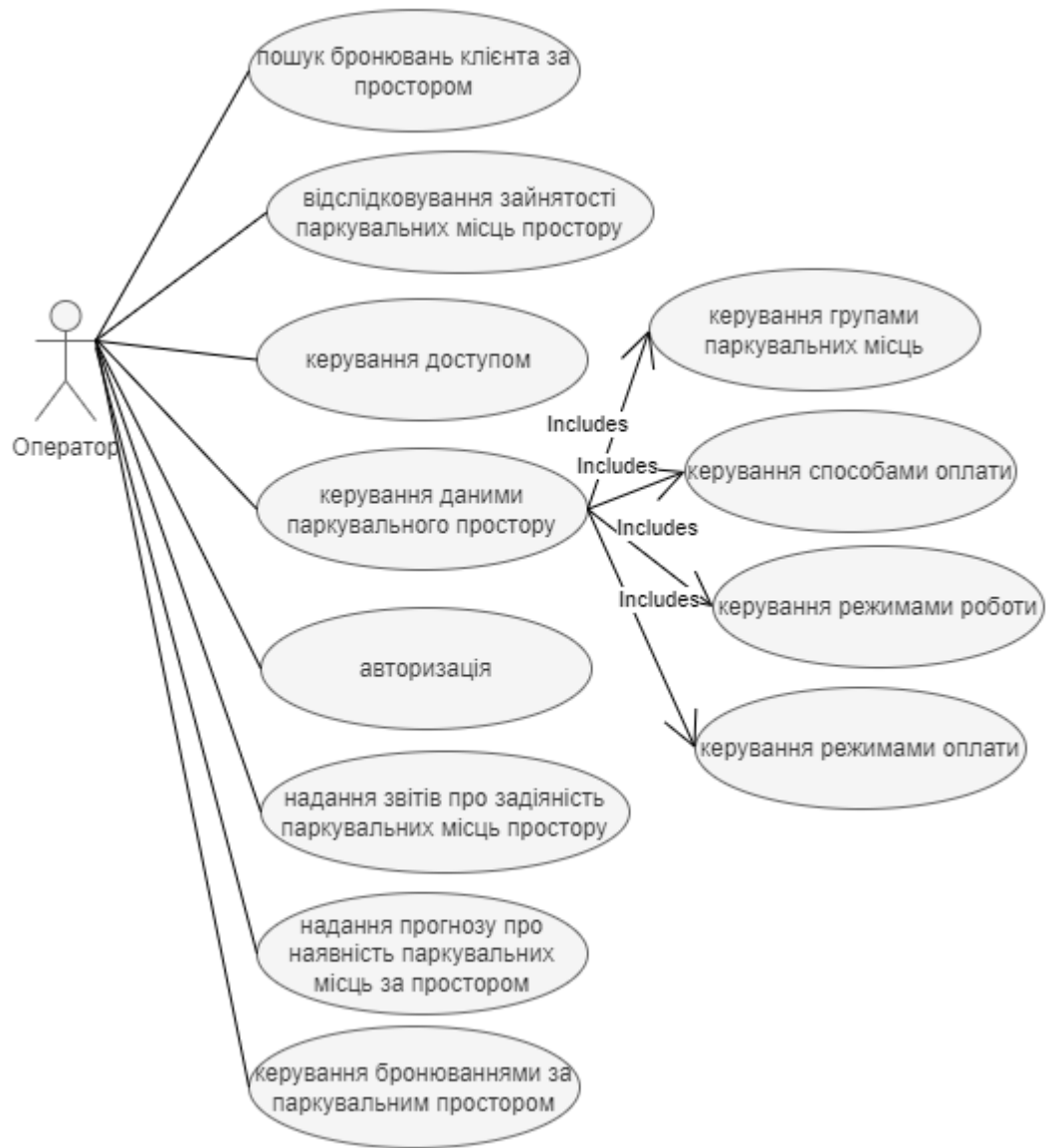


Рисунок 3.3 – Діаграма прецедентів оператора



Рисунок 3.4 – Діаграма прецедентів адміністратора

Основний потік роботи адміністратора з програмою включає наступні сценарії, відмінні від сценаріїв роботи оператора:

- реєстрація операторів;
- керування даними операторів;
- додавання нових паркувальних просторів.

3.3 Визначення структури бази даних

База даних (БД) для програми має використовуватися як для збереження даних про основні сутності предметної області, так і про звіти і прогнози, які є результатом підтримки функції прогнозування, яка є особливістю цієї програми.

Для реалізації роботи з БД використано систему керування БД PostgreSQL. Дана система є потужною, при цьому не вимагає витрачання коштів, є надійною і масштабованою, що в підсумку є важливим для предметної області [14].

Таблиця `parkapp_space` має бути створена для зберігання даних про паркувальні простори на основі таких полів:

- `id` – ідентифікатор паркувального простору;
- `capture` – назва паркувального простору;
- `location` – локація (адреса), за якою розташовано простір, у текстовому представленні;
- `coordinate1` – перша координата (довгота) простору у системі широта, довгота для відображення на мапі;
- `coordinate2` – друга координата (широта) простору у системі широта, довгота для відображення на мапі;
- `feature` – повна текстова презентація паркувального простору;
- `photo` – фотографія, яка буде використовуватися для презентації паркувального простору;

- places – загальна кількість місць на паркувальному просторі (всіх типів);

- type – тип паркувального простору (відкритий вуличний, простір під охороною, критий простір, окрема будівля, будівля з роботизованою системою);

- height – максимальна висота автомобіля, який може запаркуватися (для критих просторів, просторів у будівлях і т.д.);

- points – середня оцінка, виставлена простору клієнтами, які паркувалися на ньому;

- booked – визначає, чи можна бронювати місця на даному паркувальному просторі.

Таблиця `parkapp_raummode` має бути створена для зберігання даних про способи оплати паркувального простору на основі таких полів:

- id – ідентифікатор доступного способу сплати на паркувальному просторі;

- capture – спосіб оплати (не потребує оплати, готівкою монетами, готівкою, платіжною картою, Google Pay, Apple Pay);

- space – простір, до якого належить цей спосіб оплати.

Поле `space` є зовнішнім ключем у цій таблиці з таблиці `parkapp_space`, звідки кожен простір може мати декілька способів оплати, але не менше одного (має бути визначено щонайменше спосіб без оплати, якщо паркування безкоштовне). Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного простору. Це відповідає варіанту зв'язку один-до-багатьох.

Таблиця `parkapp_mode` має бути створена для зберігання даних про режими роботи або сплати за паркування в паркувальному просторі на основі таких полів:

- id – ідентифікатор режиму роботи або сплати за паркування в просторі;

- space – паркувальний простір, для якого визначено цей режим роботи або сплати;
- day – день тижня, коли діє такий режим роботи або сплати (або інший додатковий інтервал);
- fromtime – час, з якого діє такий режим;
- totime – час, до якого діє такий режим.

Поле space є зовнішнім ключем у цій таблиці з таблиці parkapp_space, звідки кожен простір може мати декілька режимів роботи або сплати, але не менше одного. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного простору. Це відповідає варіанту зв'язку один-до-багатьох.

Таблиця parkapp_lot має бути створена для зберігання даних про групи місць на основі таких полів:

- id – ідентифікатор групи місць;
- capture – назва групи місць;
- space – паркувальний простір, до якого належить ця група місць;
- places – кількість місць даної групи;
- feature – повний текстовий опис особливостей такої групи місць;
- sizetype – тип паркувального місця;
- photo – фотографія, яка буде використовуватися для презентації такої групи місць паркувального простору.

Поле space є зовнішнім ключем у цій таблиці з таблиці parkapp_space, звідки кожен простір може мати декілька груп місць, але не менше одного, оскільки місця бронюються за групою, тому хоча би одна група має бути створена для простору. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного простору. Це відповідає варіанту зв'язку один-до-багатьох.

Таблиця parkapp_busylot має бути створена для зберігання даних про недоступні (в ремонті) групи місць на паркувальному просторі на основі таких полів:

- id – ідентифікатор недоступності групи місць;
- lot – група місць;
- fromtime – дата і час, з яких не буде доступною ця група місць для паркування;
- totime – дата і час, до яких не буде доступною ця група місць для паркування;
- places – кількість місць з групи, що не буде доступною для паркування.

Поле lot є зовнішнім ключем у цій таблиці з таблиці parkapp_lot, звідки кожна група місць відповідного паркувального простору може мати декілька ситуацій, коли місця з такої групи не доступні для паркування через тимчасовий ремонт або інші фактори. Для групи місць може і не бути таких ситуацій. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався однієї конкретної групи місць з конкретного паркувального простору. Це відповідає варіанту зв'язку один-до-багатьох.

Таблиця parkapp_booking має бути створена для зберігання даних про бронювання паркувальних місць на основі таких полів:

- id – ідентифікатор бронювання;
- clientcar – автомобіль клієнта, для якого виконано бронювання;
- lot – група місць паркувального майданчика, до якої стосується бронювання;
- booked – дата і час, коли було заброньовано місце;
- updated – дата і час оновлення бронювання;
- state – стан бронювання;
- fromtime – дата і час, коли автомобіль може бути розташований на парковці;
- totime – дата і час, до якого автомобіль може бути розташований на парковці;
- price – вартість бронювання;
- payed – сплачена за бронювання сума.

Поле `lot` є зовнішнім ключем у цій таблиці з таблиці `parkapp_lot`, звідки кожна група місць відповідного паркувального простору може мати велику кількість бронювань в різні моменти часу. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався однієї конкретної групи місць з конкретного паркувального простору. Це відповідає варіанту зв'язку один-до-багатьох.

Поле `car` є зовнішнім ключем у цій таблиці з таблиці `parkapp_car`, звідки кожен автомобіль може мати велику кількість бронювань на різні моменти часу. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного бронювання. Це відповідає варіанту зв'язку один-до-багатьох.

Таблиця `parkapp_client` має бути створена для зберігання даних про клієнтів на основі таких полів:

- `id` – ідентифікатор клієнта програми;
- `first_name` – ім'я клієнта програми;
- `last_name` – прізвище клієнта програми;
- `password` – пароль, який використовується клієнтом для роботи з програмою;
- `username` – логін, який використовується клієнтом для роботи з програмою;
- `phone` – контактний номер телефону клієнта програми;
- `email` – електронна пошта клієнта програми;
- `forbidden` – заборона доступу до бронювання.

Таблиця `parkapp_clientcar` має бути створена для зберігання даних про автомобілі клієнта програми на основі таких полів:

- `id` – ідентифікатор клієнта програми;
- `client` – клієнт програми, якому належить або який використовує автомобіль;
- `car` – автомобіль.

Поле `client` є зовнішнім ключем у цій таблиці з таблиці `parkapp_client`, звідки кожен клієнт програми може мати в самій системі декілька автомобілів, бронювання паркування для яких він виконує. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного клієнта. Це відповідає варіанту зв'язку один-до-багатьох. Клієнт може не мати записів про власні автомобілі в поточний момент.

Поле `car` є зовнішнім ключем у цій таблиці з таблиці `parkapp_car`, звідки кожен автомобіль у програмі може мати декілька клієнтів, які ним керують. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного автомобіля. Це відповідає варіанту зв'язку один-до-багатьох.

Суть таблиці `parkapp_clientcar` полягає в тому, що вона реалізує зв'язок багато-до-багатьох між таблицями `parkapp_car` і `parkapp_client`, таким чином надаючи можливість бронювання паркувальних місць для одного й того самого автомобіля різними клієнтами. Таке рішення дозволяє бронювати місця, наприклад, членам сімей або помічникам при необхідності, а також враховувати, що одним і тим самим автомобілем можуть керувати і різних ситуаціях різні водії.

Таблиця `parkapp_car` має бути створена для зберігання даних про автомобілі в програмі на основі таких полів:

- `id` – ідентифікатор клієнта програми;
- `sizetype` – тип автомобіля;
- `regnumber` – реєстраційний номер автомобіля;
- `forbidden` – заборона бронювання паркувальних місць для даного автомобіля.

Таблиця `parkapp_comment` має бути створена для зберігання даних про висловлювання про паркувальні простори на основі таких полів:

- `id` – ідентифікатор висловлювання;
- `client` – клієнт, який створив висловлювання про паркувальний простір у програмі;

- space – паркувальний простір, про який створено висловлювання;
- created – дата і час створення висловлювання;
- view – текст висловлювання;
- liked – кількість клієнтів, які вважають дане висловлювання корисним для себе і загалом таким, що відповідає стану паркувального простору;
- disliked – кількість клієнтів, які вважають дане висловлювання некорисним для себе і загалом таким, що не відповідає стану паркувального простору;
- points – оцінка, яку було встановлено даному паркувальному простору.

Поле client є зовнішнім ключем у цій таблиці з таблиці parkapp_client, звідки кожен клієнт програми може мати різну кількість висловлювань. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного клієнта. Це відповідає варіанту зв'язку один-до-багатьох. Клієнт може не мати записів про власні висловлювання в поточний момент.

Поле space є зовнішнім ключем у цій таблиці з таблиці parkapp_space, звідки кожен паркувальний простір може мати декілька висловлювань про нього. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного простору. Це відповідає варіанту зв'язку один-до-багатьох. Простір може не мати висловлювань про нього в даний конкретний момент.

Таблиця parkapp_report має бути створена для зберігання даних сформованих звітів на основі таких полів:

- id – ідентифікатор звіту;
- lot – група місць паркувального майданчику, якої стосується такий звіт;
- period – період, за який сформовано цей звіт (день, тиждень і т.д.);
- createdfor – дата і час або тільки дата, яким відповідає звіт (фактично початок періоду);

- `max_free_places` – максимальна кількість вільних місць у будь-яку мить звітного періоду на паркувальному просторі;
- `min_free_places` – мінімальна кількість вільних місць у будь-яку мить звітного періоду на паркувальному просторі;
- `avg_free_places` – середня кількість вільних місць на паркувальному просторі;
- `booked_places` – кількість бронювань на місця паркувального простору протягом звітного періоду;
- `booked_time` – сумарний час у хвилинах, протягом якого були заброньовані місця паркувального простору.

Поле `lot` є зовнішнім ключем у цій таблиці з таблиці `parkapp_lot`, звідки кожна група місць відповідного паркувального простору може мати велику кількість звітів за весь час доступності для бронювання цієї групи в програмі. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався однієї конкретної групи місць з конкретного паркувального простору. Це відповідає варіанту зв'язку один-до-багатьох.

Таблиця `parkapp_prediction` має бути створена для зберігання даних про прогнозування наявності вільних місць на основі таких полів:

- `id` – ідентифікатор прогнозу;
- `createdfor` – час, на який виконано прогноз;
- `lot` – група паркувальних місць, якої стосується прогноз;
- `places` – мінімальна кількість місць, яка доступна протягом даного періоду в будь-яку його мить.

Таблиця `parkapp_price` має бути створена для зберігання даних про вартість паркування на основі таких полів:

- `id` – ідентифікатор запису про вартість паркування;
- `mode` – режим сплати, якого стосується даний запис;
- `lot` – група паркувальних місць, якої стосується запис про вартість паркування;

- `pricemode` – одиниця часу паркування, для якої працює даний режим;

- `pricemode` – вартість одиниці часу паркування.

Поле `mode` є зовнішнім ключем у цій таблиці з таблиці `parkapp_mode`, звідки кожен режим сплати може мати різну кількість записів про вартість паркування. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного режиму. Це відповідає варіанту зв'язку один-до-багатьох.

Поле `lot` є зовнішнім ключем у цій таблиці з таблиці `parkapp_lot`, звідки кожна група місць паркувального простору може мати різну кількість записів про вартість паркування. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався однієї конкретної групи місць. Це відповідає варіанту зв'язку один-до-багатьох.

Таблиця `parkapp_operator` має бути створена для зберігання даних про операторів паркувальних просторів на основі таких полів:

- `id` – ідентифікатор оператора паркувального простору;
- `first_name` – ім'я оператора паркувального простору;
- `last_name` – прізвище оператора паркувального простору;
- `space` – паркувальний простір, з яким працює даний оператор;
- `phone` – контактний номер телефону оператора паркувального простору;
- `password` – пароль, який використовується оператором паркувального простору для роботи з програмою;
- `username` – логін, який використовується оператором паркувального простору для роботи з програмою.

Поле `space` є зовнішнім ключем у таблиці `parkapp_operator` з таблиці `parkapp_space`, звідки кожен простір може мати декілька операторів, які відповідають за роботу з його даними в програмі. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного простору, тобто працювати з декількома просторами може тільки

адміністратор або адміністратори всієї системи. Це відповідає варіанту зв'язку один-до-багатьох.

Схема структури БД, яка була утворена за описаною логікою, подана на рис. 3.5.

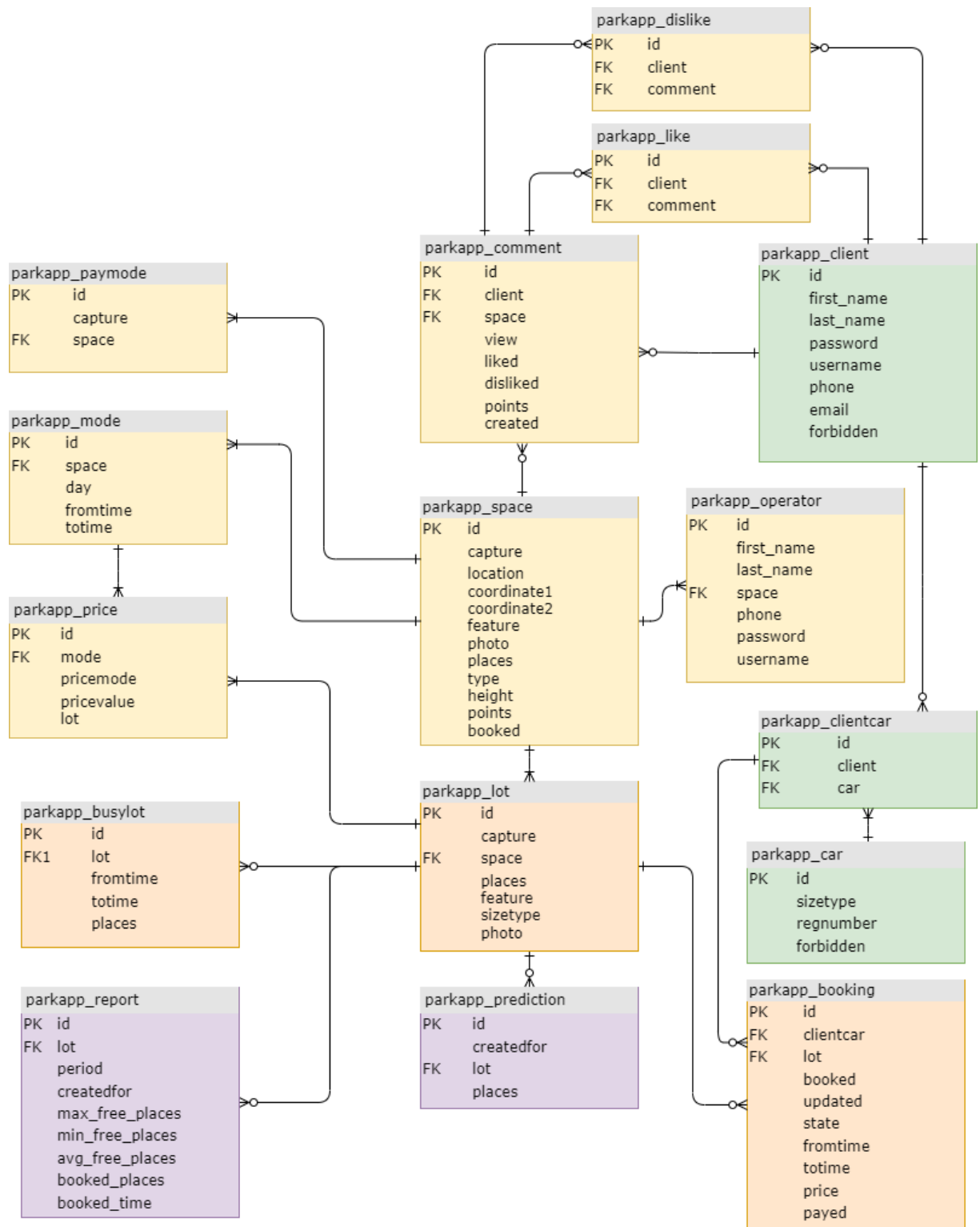


Рисунок 3.5 – Схема структури БД

Таблиця `parkapp_like` має бути створена для зберігання даних про уподобання висловлювань на основі таких полів:

- `id` – ідентифікатор запису про уподобання;
- `client` – клієнт, який уподобав висловлювання;
- `comment` – висловлювання, яке було уподобано.

Поле `client` є зовнішнім ключем у цій таблиці з таблиці `parkapp_client`, звідки кожен клієнт може мати різну кількість уподобаних висловлювань інших клієнтів. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного клієнта. Це відповідає варіанту зв'язку один-до-багатьох.

Поле `comment` таблиці `parkapp_like` є зовнішнім ключем у цій таблиці з таблиці `parkapp_comment`, звідки кожне висловлювання може мати різну кількість уподобань. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного висловлювання. Це відповідає варіанту зв'язку один-до-багатьох.

Таблиця `parkapp_dislike` має бути створена для зберігання даних про негативне враження про висловлювання на основі таких полів:

- `id` – ідентифікатор запису про негативне враження;
- `client` – клієнт, який має негативне враження про висловлювання;
- `comment` – висловлювання, яке було негативно оцінено.

Поле `client` є зовнішнім ключем у цій таблиці з таблиці `parkapp_client`, звідки кожен клієнт може мати різну кількість негативно оцінених висловлювань інших клієнтів. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного клієнта. Це відповідає варіанту зв'язку один-до-багатьох.

Поле `comment` є зовнішнім ключем у цій таблиці з таблиці `parkapp_comment`, звідки кожне висловлювання може мати різну кількість негативних оцінок. Записи в таблиці визначаються таким чином, щоб кожен з них стосувався одного конкретного висловлювання. Це відповідає варіанту зв'язку один-до-багатьох.

3.4 Висновки за розділом 3

У третьому розділі виконано і описано результати проєктування програми, що включають порівняння і вибір для подальшої реалізації мови Python, фреймворку Django, моделювання функціональних вимог до програми на основі діаграм прецедентів для користувача, клієнта, оператора і адміністратора, проєктування структури БД.

4 ОСНОВНІ РІШЕННЯ ЩОДО РЕАЛІЗАЦІЇ КОМПОНЕНТІВ СИСТЕМИ

4.1 Алгоритм функціонування програми

Початок роботи системи (рис. 4.1) визначається тоді, коли адміністратор запускає вебсервер і відповідно сам додаток Django на виконання, тоді запускаються два потоки процесів, які виконуються одночасно на сервері.

Перший потік належить до виконання самого вебдодатку і контролюється Django. Під час реалізації даного процесу кожен клієнтський пристрій виконує звернення до вебсервера, а той у відповідь формує основну вебсторінку вебдодатку, після чого зароджується цикл, в процесі виконання якого користувач (користувач, клієнт, оператор, адміністратор) задає дані на вебсторінці і/або визначає параметри запиту до сервера, а той у свою чергу вносить зміни в БД, якщо такі зміни передбачались запитом, після чого або одразу (якщо зміни не передбачались) формує у відповідь вебсторінку. Цикл взаємодії користувача з вебсервером продовжується до того моменту, поки його не завершить користувач.

Другий потік реалізується на основі скриптів, які виконуються за розкладом. В залежності від поточного моменту передбачено послідовне виконання наступних перевірок:

- якщо поточний момент відповідає початку нової години, то спочатку для всіх паркувальних просторів виконується прогнозування доступності паркувальних місць, а потім створюється звіт за минулу годину;
- якщо поточний момент є не тільки початком нової години, але і початком нового дня, то створюється звіт за минулу добу;
- якщо поточний момент є не тільки початком нової години і нового дня, але і початком нового місяця, то створюється звіт за минулий місяць;
- якщо поточний момент є не тільки початком нової години, дня, місяця, але і початком нового року, то створюється звіт за минулий рік.

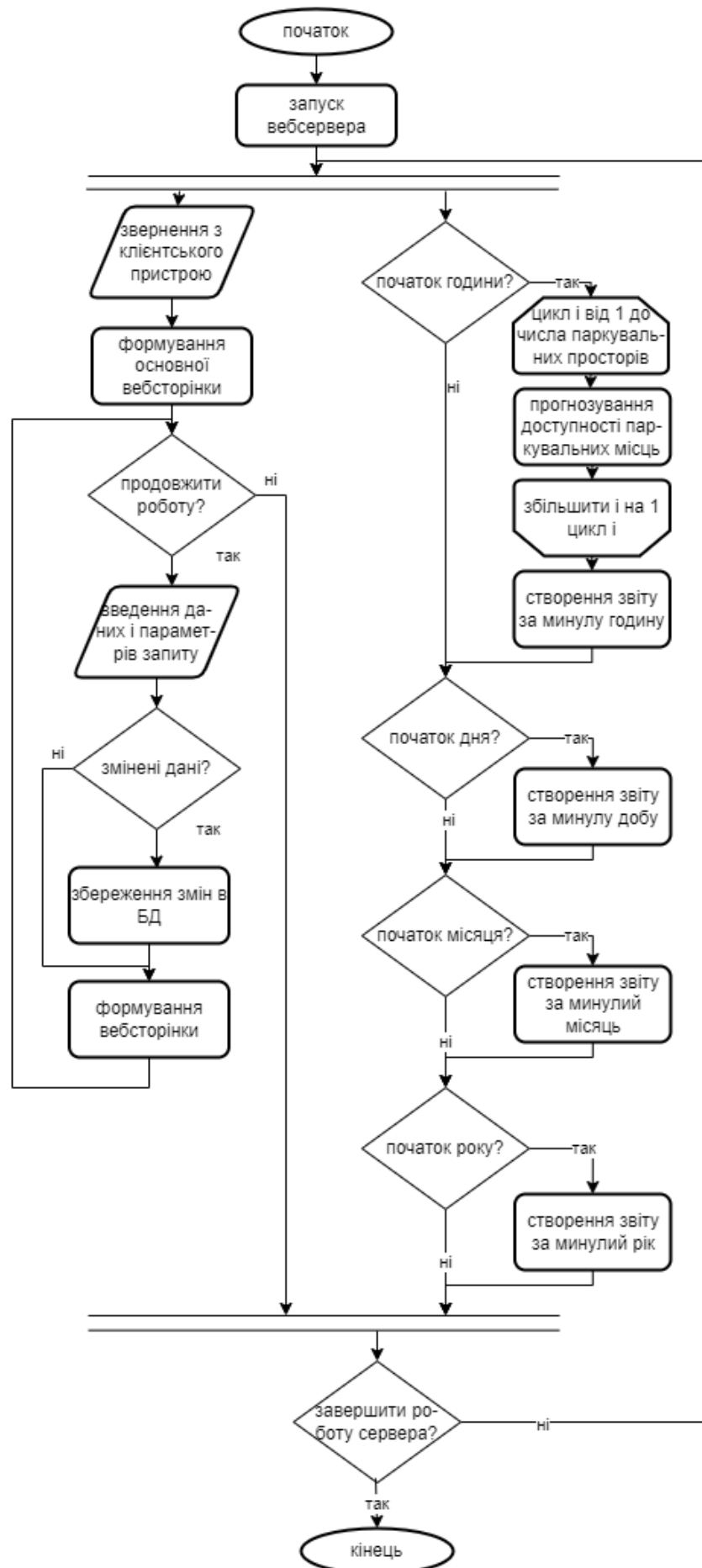


Рисунок 4.1 – Схема функціонування програми

4.2 Основні реалізовані класи

Основні класи вебдодатку охоплюють класи, що забезпечують створення представлень, форм та моделей вебдодатку. Оскільки моделі вебдодатку відповідають структурі БД, а вона була детально розглянута в розділі 2, то в даному розділі ці класи не розглядаються окремо повторно, адже визначені вони повністю на основі логіки структури таблиць і зв'язування цих таблиць між собою з додаванням тільки правил упорядкування даних в моделях, коли це необхідно.

Частина рішень вебдодатка реалізована не у вигляді класів. Вона охоплює зокрема менеджер URL, шаблони вебсторінок, та окремі скрипти, які визначають етапи, представлені на схемі функціонування програмного забезпечення. У подальшому основна увага приділена класам представлень, які вирішують питання наповнення сторінок, які відображаються користувачам в різних ролях, даними, проте розглянуто тільки частину з них, за допомогою яких описано принцип прийняття рішень щодо їх реалізації.

Робота адміністратора реалізована через адміністративну панель Django, а тому потребувала тільки реєстрації відповідних моделей у скрипті `admin.py`.

Перш за все потрібно зазначити, що підтримка користувачів для даного програмного забезпечення містить певну специфіку, яка зокрема була закладена і в БД, і в моделі вебдодатку. Ця специфіка пов'язана з тим, що підбір паркувальних майданчиків, підбір паркувальних місць виконується за віддаленістю користувача від самих паркувальних майданчиків, або за віддаленістю того місця, яке шукає користувач, від паркувальних майданчиків. Під час вирішення цієї проблеми, яка в значній мірі актуалізується під час пошуку паркувальних місць, було, по-перше, визначено, що впорядкування пропозицій буде здійснюватися за ступенем віддаленості цих місць від точки, яку шукає або в якій знаходиться користувач. Але друга частина визначала відповідну необхідність

обчислення дистанції між різними місцями. Для цього було використано цілий ряд бібліотек:

- бібліотека геокодування `geocoder`, яка фактично використовується для того, щоб отримати інформацію про розташування користувача у випадку, якщо його цікавить пошук відносно його поточної позиції: тоді потрібно встановити IP-адресу користувача і з неї визначити широту і довготу;

- бібліотека `haversine`, яка безпосередньо призначена і фактично використовується для того, щоб обчислити відстань між двома точками, визначеними їхніми широтою і довготою, у метрах (у протилежному випадку потрібно було б вручну реалізовувати відповідне обчислення): у даному випадку такими точками є поточне розташування або задана позиція користувачем і позиція розташування самого паркувального майданчика, визначені вказаними координатами;

- бібліотека `geору`, яка створює клієнт для вебсервісів геокодування і яка фактично використовується для того, щоб визначити за заданою адресою в місті, тобто містом, вулицею, номером будинку, координати, тобто довготу і широту, цього місця.

Описані бібліотеки і логіка їх використання застосовується для створення класів представлень з пошуку, які загалом охоплюють наступні.

Клас `SpaceSearchView` реалізує функціональність пошуку паркувальних просторів.

Клас `LotSearchView` реалізує функціональність пошуку паркувальних місць.

Обидва класи пошуку організовані через наслідування класу `ListView`, який є базовим класом Django, що допомагає упорядковувати листи об'єктів, що відповідають класам моделей. У випадку класу `SpaceSearchView` такою моделлю є модель `Space`, що визначає паркувальний простір, а у випадку класу `LotSearchView` такою моделлю є модель `Lot`, що визначає групу паркувальних місць. Пошук, керування, бронювання відбувається саме серед

груп місць, тобто однакові за загальними параметрами паркувальні місця не відрізняються одне від одного і відслідковуються групою. На якому саме місці в паркінгу розташовано таке місце не важливо, оскільки контроль відбувається при заїзді, відповідно важливо, щоб автомобіль, для якого було заброньовано місце і який відповідає вимогам щодо розміру (тобто це не має бути кемпер або автобус у випадку, якщо місце було заброньовано для звичайного легкового автомобіля), пересік пропускний пункт. Контролюється загальна кількість автомобілів кожного розміру, серед яких визначено:

- мотоцикл;
- легковий автомобіль;
- кемпер;
- автобус.

Робота класу LotSearchView контролюється через параметри GET-запиту:

- address – адреса в текстовому представленні (місто, вулиця, номер будинку);
- long – довгота координати, відносно якої потрібно виконувати пошук;
- lat – широта координати, відносно якої потрібно виконувати пошук;
- type – тип паркувального простору (ціле значення, що представляє відкритий вуличний простір, простір під охороною, критий простір, окрему будівлю, будівлю з роботизованою системою);
- size_type – тип розміру автомобіля (за замовчуванням використовується значення 0);
- height – висота автомобіля в метрах (за замовчуванням використовується значення 213);
- dist – відстань в метрах від точки, від якої виконується пошук (за замовчуванням використовується значення 50);

– points – мінімальна оцінка простору (за замовчуванням використовується значення 1);

– day – частина дати, для якої виконується пошук місць, що визначає номер дня в місяці;

– month – частина дати, для якої виконується пошук місць, що визначає місяць;

– year – частина дати, для якої виконується пошук місць, що визначає рік;

– hour – частина дати, для якої виконується пошук місць, що визначає рік;

– minute – частина дати, для якої виконується пошук місць, що визначає хвилину.

Ще одним додатковим питанням, яке вирішувалось під час організації класу LotSearchView, є питання визначення вартості паркування на відповідному паркувальному місці. Це пов'язано з тим, що вартість паркування в різний час і в різні дні може бути різною. У підсумку використовується така система визначення вартісного режиму:

- значення 1 відповідає вартості в понеділок;
- значення 2 відповідає вартості у вівторок;
- значення 3 відповідає вартості в середу;
- значення 4 відповідає вартості в четвер;
- значення 5 відповідає вартості в п'ятницю;
- значення 6 відповідає вартості в святковий день;
- значення 7 відповідає вартості в робочий день;
- значення 8 відповідає вартості в вихідний день;
- значення 9 відповідає вартості в будь-який день, тобто в такому випадку вартість завжди однакова.

Відповідне значення зберігається в полі моделі Mode, що має ім'я day.

Для того щоб визначити, якому саме варіанту відповідає конкретна дата, на яку виконується пошук, використовуються наступні бібліотеки:

- бібліотека `holidays`, яка дозволяє для потрібної країни визначити, чи є відповідна дата в країні святковою, чи ні;
- бібліотека `calendar`, яка використовується для визначення кількості днів у місяці;
- бібліотека `pandas`, яка використовується для перевірки заданої дати на те, чи є вона робочим днем.

Робота класу `SpaceSearchView` контролюється через параметри GET-запиту:

- `name` – назва паркувального простору чи фраза з цієї назви;
- `address` – адреса в текстовому представленні (місто, вулиця, номер будинку);
- `long` – довгота координати, відносно якої потрібно виконувати пошук;
- `lat` – широта координати, відносно якої потрібно виконувати пошук;
- `type` – тип паркувального простору (ціле значення, що представляє відкритий вуличний простір, простір під охороною, критий простір, окрему будівлю, будівлю з роботизованою системою);
- `height` – висота автомобіля в метрах (за замовчуванням використовується значення 213);
- `dist` – відстань в метрах від точки, від якої виконується пошук (за замовчуванням використовується значення 50);
- `points` – мінімальна оцінка простору (за замовчуванням використовується значення 1).

Клас `CommentLikeView` реалізує вподобання висловлювань іншими клієнтами, якщо вони вважають ці висловлювання корисними. Даний клас створений через наслідування класу `View`, що є базовим для представлення без особливостей, та класу `LoginRequiredMixin`, що передбачає необхідність попередньої авторизації. Завдяки цьому користувач, тобто неавторизований користувач, не зможе вподобати висловлювання, допоки він не авторизується в програмі і не почне виконувати роль клієнта.

При створенні уподобання важливим є перевірка того, що дане висловлювання цим клієнтом ще не було внесено до неуподобаних, тобто він попередньо не висловився, що це висловлювання є некорисним, а також того, що це висловлювання раніше вже було уподобано цим клієнтом. Останнє необхідно для того, щоб не можна було штучно змінювати кількість уподобань висловлювань.

Основна логіка роботи даного класу реалізується при натисканні на відповідну кнопку в інтерфейсі користувача, а тому визначена методом `post (self, request, pk)`. У випадку, якщо таке уподобання вже існує, генерується помилка як об'єкт класу `IntegrityError`. Також витягається об'єкт моделі `Dislike` і тільки, якщо такого об'єкту немає, то виконується спроба створити і зберегти об'єкт уподобання, який буде збережений без помилки тільки якщо такого об'єкта не було збережено в БД раніше, а також збільшення кількості уподобань даного висловлювання у випадку, якщо вдалось зберегти уподобання, та збереження оновленого об'єкта висловлювання. Аналогічним чином влаштований клас `CommentDislikeView`, який підтримує роботу представлення щодо внесення висловлювання до неуподобаних висловлювань.

Клас `SpaceView` реалізує представлення роботи з даними паркувального простору. Для цього використовується клас `DetailView`, через наслідування якого реалізовано даний. Оскільки це клас деталізації, то основною задачею є наповнення даними сторінки про відповідний об'єкт паркувального простору з номером `pk`, що реалізується через метод `get (self, request, pk)`.

У методі `get` відбувається витягання самого об'єкта з паралельним заповненням пов'язаного з ним набору об'єктів режимів сплати через відповідне зворотне поле моделі `raumodes`. При відтворенні цієї сторінки до набору даних входить також інформація про групи паркувальних місць. Стан кожної групи встановлюється на певний момент, а тому дані формуються або

на поточний момент, або на дату, яку передано через параметри GET-запиту, що включають:

- day для визначення дня в місяці;
- month для визначення місяця;
- year для визначення року;
- hour для визначення години;
- minute для визначення хвилини.

З переданих значень формується дата і час як об'єкт `datetime`, а тоді реалізується перевірка цієї дати, чи не є вона святом або чи є вона робочим днем. Тоді відбувається пошук режимів роботи для цих варіантів. Якщо результатів не знайдено, то відбувається пошук за номером робочого дня тижня. Якщо і цих результатів не знайдено, то реалізується пошук режимів для всього тижня, тобто постійного графіку.

Відповідно для кожної групи місць паркувального простору далі визначається:

- кількість місць, які недоступні на задану дату (ключ `busy_places` словника `lots` контексту даних сторінки, що повертається в результаті), тобто знаходяться в ремонті або з інших причин не можуть використовуватися;
- кількість місць яка взагалі є в даній групі місць на цьому паркувальному просторі;
- кількість місць, які заброньовані на задану дату (ключ `booked_places` словника `lots` контексту даних сторінки, що повертається в результаті);
- кількість вільних місць на задану дату (ключ `free_places` словника `lots` контексту даних сторінки, що повертається в результаті);
- інформація про всю групу паркувальних місць повертається через ключ `lot` словника `lots` контексту даних сторінки, що повертається в результаті;
- вартість для кожного режиму паркування повертається через ключ `prices` словника `lots` результуючого контексту даних.

Весь цей перелік груп паркувальних місць з відповідними параметрами, описаними вище повертається у складі параметра `lots` контексту даних, який передається в сторінку шаблону, а разом з ним у складі параметра `space` повертаються дані стосовно самого простору, у складі параметра `modes` повертається список режимів роботи паркувального простору, у складі параметра `comments` повертаються 3 останні висловлювання про цей простір.

Ряд класів представлень відповідає за перегляд висловлювань, наприклад, клас `AuthClientCommentsView` надає список висловлювань, які були створені авторизованим у даний момент клієнтом, впорядковані починаючи від останнього висловлювання за часом публікації.

Відповідні класи реалізовані для додавання даних основних сутностей, редагування, де це необхідно, і їх видалення.

Клас `PredictionView` реалізує підтримку перегляду прогнозів за паркувальним майданчиком і його групами місць, а класи `OperatorHourReportView`, `OperatorDayReportView`, `OperatorMonthReportView`, `OperatorYearReportView` реалізують підтримку роботи зі звітами за годину, цілий день (містить інформацію про кожну годину і загалом за день), місяць (містить інформацію про кожну добу і загалом за місяць) або рік (містить інформацію про кожен місяць і загалом за рік) відповідно, тобто з даними про паркування, які вже відбулись на відповідному паркувальному просторі, з яким працює оператор, що авторизувався в програмі.

Для реалізації методу, який запропоновано в роботі, використано наступні основні бібліотеки:

- бібліотека `pandas` використовується для роботи з вибіркою даних, яка зокрема таким чином завантажується з відповідного CSV-файлу;
- бібліотека `xgboost`, яка безпосередньо містить реалізацію алгоритму `XGBoost` і відповідної логіки роботи загалом.

4.3 Висновки за розділом 4

У четвертому розділі представлено основний порядок роботи вебдодатку і скриптів, які обслуговують його роботу, описано спосіб створення основних класів вебдодатку, основні використані бібліотеки під час реалізації.

5 ЕКСПЛУАТАЦІЯ, ТЕСТУВАННЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ДОСЛІДЖЕННЯ ПРОГРАМИ

5.1 Опис роботи з програмою

Для взаємодії користувачів різних ролей з вебдодатком створено графічний інтерфейс користувача на основі шаблону [15]. Визначено інтерфейс для всіх ролей користувачів, який оформлений у єдиному стилі.

Основна функціональність, з якою працює клієнт, після авторизації пов'язана з пошуком місць (рис. 5.1). У випадку якщо користувач неавторизований, то меню зменшене у відповідності з діаграмою прецедентів, а спроба бронювання призводить до переходу на сторінку авторизації.

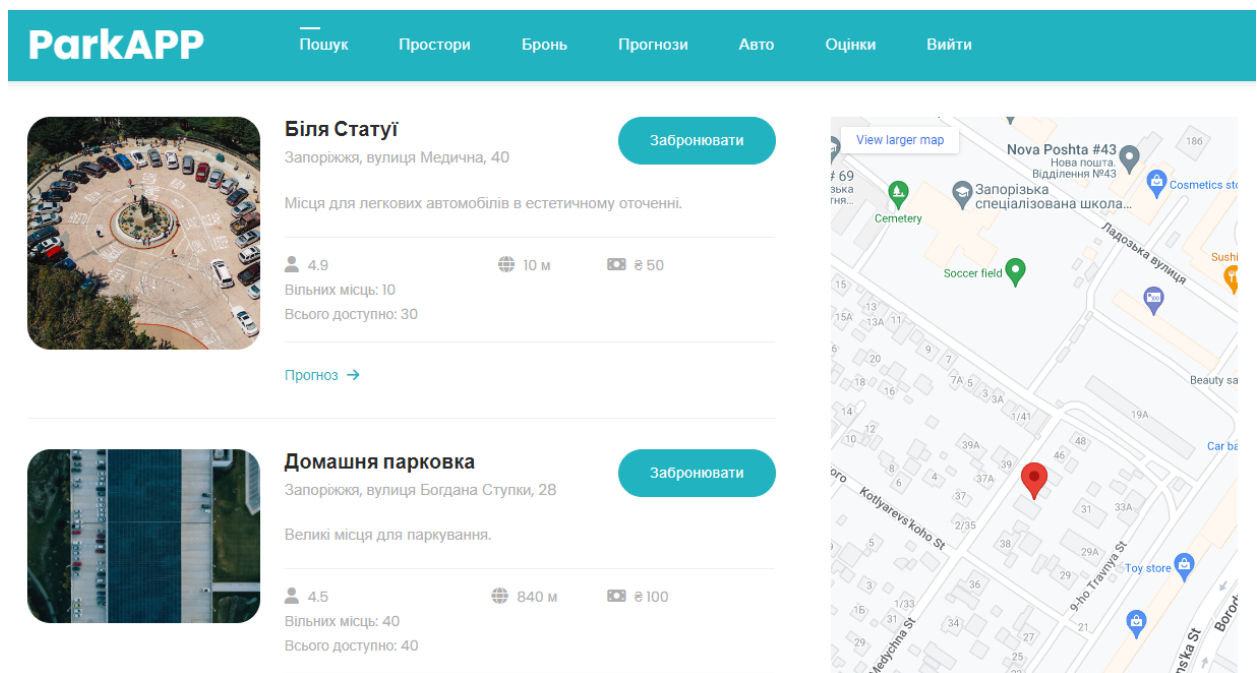


Рисунок 5.1 – Сторінка пошуку місць на паркувальних просторах

На сторінці пошуку місць на паркувальних просторах згорі розташовано форму для визначення параметрів пошуку, що включає:

– адресу в текстовому представленні (місто, вулиця, номер будинку), що вводиться вручну або обирається як координата на мапі (при натисканні

на галочку “на мапі” відображається ділянка з мапою), також може задаватися радіус пошуку (відстань в метрах від точки, від якої виконується пошук, за замовчуванням дорівнює 500 метрам);

- тип паркувального простору, що обирається з випадającego переліку і включає вуличний простір, простір під охороною, критий простір, окрему будівлю, будівлю з роботизованою системою;

- тип розміру, що обирається з випадającego переліку і включає мотоцикл, легковий автомобіль, автобус, фургон, вантажний автомобіль;

- висота автомобіля в метрах;

- мінімальна оцінка простору, що вибирається з випадającego переліку;

- дата і час, на які виконується пошук місць.

В результатах пошуку відображаються в такому випадку саме групи місць. Кожна група місць відповідає заданому типу автомобіля з параметрів пошуку, представляє назву простору, його фотографію, адресу, оцінку, віддаленість від заданої точки, вартість паркування на заданий в пошуку строк, кількість вільних місць на цей час та кількість місць, які загалом доступні для бронювання.

Для того щоб забронювати місце на паркувальному просторі, потрібно натиснути на кнопку з цією назвою, що відкриє сторінку з формою бронювання (рис. 5.2), звідки можна задати час паркування та обрати потрібний автомобіль з усіх, які були визначені клієнтом (з пункту меню “Авто”). Після внесення потрібних даних розраховується вартість бронювання, а після підтвердження відбувається перехід до сторінки сплати. На даний момент доступна сплата через Приватбанк.

Клієнт також може переглядати і за потреби скасовувати не виконані бронювання через меню “Бронь”. Меню “Прогнози” дозволяє переглянути прогноз щодо наявної кількості місць через годину на потрібному паркувальному просторі.

Якщо клієнт бажає просто ознайомитися з паркувальними просторами, то пошук з меню “Простори” є скороченим варіантом пошуку з меню “Пошук”, а в результаті пошук відбувається саме просторів, а не груп місць.

Переглянути власні висловлювання про простори можна з меню “Оцінки”. Створити нові можна зі сторінки власних бронювань.

Рисунок 5.2 – Сторінка бронювання місця на паркувальному просторі

Робота оператора після авторизації реалізується саме з тим паркувальним простором, до якого він належить.

Через меню “Прогноз” (рис. 5.3) оператор може переглянути прогноз про кількість паркувальних місць на його паркувальному просторі за кожною групою місць. Через кнопку “Редагувати” можна змінити інформацію про цю групу місць.

Меню “Бронь” дозволяє керувати всіма бронюваннями за цим простором та виконувати їх пошук за номером телефону клієнта.

Меню “Доступ” дозволяє заборонити доступ для бронювання клієнту або автомобілю.

Переглянути звіти, визначивши дату, про те, на скільки були зайнятими місця на паркувальному просторі, можна через меню “Звіти”.

ParkAPP
БроньПростірМісцяДоступЗвітиПрогнозВийти



Важка техніка
Вантажний автомобіль

Редагувати

Місця для вантажних автомобілів до 6 метрів.

Всього місць: 20
Заброньовано: 10
Недоступно: 0
Вільно: 10
Прогноз (мінімум наявних): 4



Автобусна гавань
Автобус

Редагувати

Місця для автобусів з доступом до мийки.

Всього місць: 10
Заброньовано: 4
Недоступно: 0
Вільно: 6
Прогноз (мінімум наявних): 3

Рисунок 5.3 – Сторінка з прогнозами задіяності місць паркувального простору

5.2 Експериментальне дослідження

Експериментальне дослідження виконано з використанням спеціальної вибірки даних [16], яка була створена на основі даних про паркування в місті Лісабон, Португалія, протягом 2020 року.

Дана вибірка містить такі характеристики:

- ідентифікатор паркувального простору;
- назва паркувального простору;
- максимальна кількість місць на паркувальному просторі;
- кількість зайнятих місць на паркувальному просторі в даний

момент;

- географічні координати паркувального простору;
- час, з якого була зафіксована дана інформація про кількість місць;
- час, до якого ця інформація про кількість місць була актуальною.

Під час вивчення вибірки було досліджено фактори, які впливають на кількість доступних місць. Отримані результати були покладені в основу створення методу прогнозування доступності паркувальних місць.

Під час даного вивчення спочатку було встановлено, що хоча є певна загальна залежність мінімальної кількості доступних місць на паркувальному майданчику від години доби, але вона не є чітко визначеною, оскільки загальна форма такої зміни змінюється в різні дні. Було встановлено, що для різних паркувальних просторів дана форма також може бути різною: так кількість вільних місць може вночі бути більшою ніж вдень, а може бути навпаки дещо меншою. Це залежить від того, де саме розташовано паркувальний простір. Приклад одного паркувального простору представлено на рис. 5.4, де помітно, що кількість вільних місць вночі є меншою ніж вдень. Але при цьому кількість вільних місць вдень може бути значно різною в різні дні і це не обумовлюється напряму сезоном.

Для прикладу було обрано три дати протягом року:

- 29 червня 2020 року характеризується більшою кількістю вільних місць вночі, але меншою їх кількістю вдень порівняно з попереднім днем (28 червня 2020 року);

- якщо загальне збільшення числа вільних паркувальних місць вдень характерно і для 28 червня, і для 29 червня 2020 року, то форма кривої, що визначає цю залежність між двома сусідніми датами відрізняється;

- 15 червня 2020 року мінімальна доступна кількість місць вночі і вранці була приблизно рівною цим двом обраним літнім датам, а ось всередині дня ця зміна була достатньо значною і сильно збільшувалась до 16 години, після чого відбувся різкий спад.

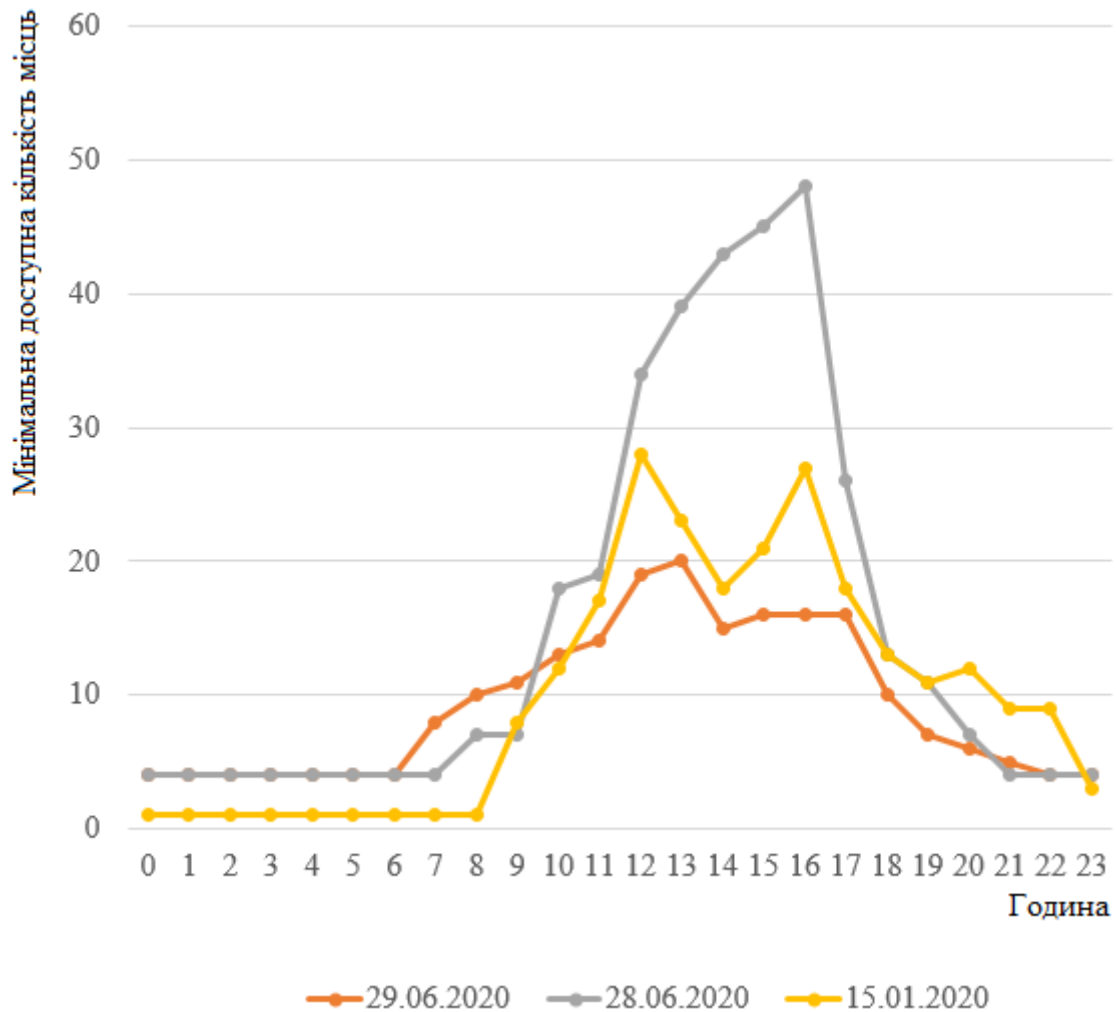


Рисунок 5.4 – Графік залежності мінімальної доступної кількості паркувальних місць від години на паркувальному майданчику

Дані дати були обрані випадково для ілюстрації загальної характеристики залежності, згідно з якою напряму сезоном або сусідніми датами важко напряму визначити залежність для визначення кількості наявних місць у майбутні періоди. Хоча така залежність і є, але вона значно змінюється від одного дня до іншого. І у випадку, коли на стоянці може не виявитися взагалі місць, така зміна може стати критичною і призвести до прийняття невірної рішення водієм.

Експериментальне дослідження було проведено для моделей прогнозування, які використовуються для роботи з часовими рядами і які були використані в дослідженнях, приведених в розділі 2: SARIMAX, метод

опорних векторів за роботою [6], XGBoost за роботою [7] та запропонований метод. Отримані результати представлені в табл. 5.1.

Таблиця 5.1 – Результати експериментального дослідження

Метод / модель	Коренева середньоквадратична похибка
SARIMAX	0,421
Метод опорних векторів за роботою [6]	0,373
XGBoost за роботою [7]	0,251
Запропонований метод	0,197

Найкращий результат з рівнем кореневої середньоквадратичної похибки в 0,197 дозволив отримати запропонований метод, він на 21,5 % виявився кращим за XGBoost за роботою [7], який з тими ж ознаками, що і в роботі [7], був використаний для прогнозування показника у відповідності з даною задачею. Окрім покращення значення похибки необхідно зазначити, що загалом запропонований метод дозволяє виконувати значно меншу кількість звернень до сервера, що є важливим для сервісу, з якого запитується трафік в місті. Це дозволяє за використання безкоштовних тарифів виконувати більшу кількість запитів на місяць, що дозволяє виконувати дослідження, наприклад, для різних міст, зокрема потенційно достатньо великої кількості міст світу.

5.3 Висновки за розділом 5

У п'ятому розділі виконано опис основних можливостей, які запитуються при роботі з програмою різними користувачами, проведено експериментальне дослідження, яке вказало на наявність переваг у запропонованого методу в точності прогнозування та кількості запитів.

ВИСНОВКИ

Основні результати виконаної роботи полягають у створеному програмному забезпеченні бронювання паркувальних місць, розробленому методі прогнозування доступності паркувальних місць, який було впроваджено до складу розробленого програмного забезпечення для надання фактичних даних і прогнозів з метою прийняття рішень водіями про необхідність дочасного бронювання паркувальних місць або відсутність такої необхідності, та проведеному експериментальному дослідженні проблеми прогнозування доступності паркувальних місць.

Для розроблення програмного забезпечення спочатку було проаналізовано наявні аналоги, на основі чого визначено прогнозування доступності паркувальних місць як особливість розробки та виділено функції, які програма має виконувати, після цього було промодельовано функціональні вимоги до програми, вирішено використовувати Python і Django для розробки, що було посилено за рахунок бібліотек для оброблення даних, інтелектуального аналізу даних, прийняття допоміжних прикладних рішень, визначено структуру БД з врахуванням необхідності як збереження основних даних стосовно паркувальних просторів і паркування на них, так і результатів використання методу прогнозування доступності паркувальних місць, визначено порядок роботи та в підсумку реалізовано програмне забезпечення.

Для розроблення методу прогнозування доступності паркувальних місць виконано дослідження існуючих методів прогнозування, в тому числі експериментальне на основі застосування вибірки даних, представлено власний метод, який підтвердив отримання кращих результатів порівняно з іншими існуючими.

Виконання роботи було побудовано на основі сформульованого переліку завдань, кожен пункт якого було виконано.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Lack of Parking in Big Cities [Electronic resource] : Reasons, Causes, and Potential Solutions. – Access mode : <https://studycorgi.com/lack-of-parking-in-big-cities-reasons-causes-and-potential-solutions/>.
2. Ahmed, I. S. Car Parking Problem in Urban Areas, Causes and Solutions [Electronic resource] / I. S. Ahmed // Towards A Better Quality of Life : The 1st International Conference, 24-26 November 2017. – El Gouna, Egypt : Technische Universitat Berlin Campus El Gouna, 2017 – Access mode : https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3163473.
3. Plumer, B. Cars take up way too much space in cities. New technology could change that [Electronic resource] / B. Plumer. – Access mode : <https://www.vox.com/a/new-economy-future/cars-cities-technologies>.
4. Find Parking Lots, Parking Garages, Parking Meters, Private Garages – Reserve Parking [Electronic resource]. – Access mode : <https://www.parkopedia.com/>.
5. Online Parking Reservations. Parclick [Electronic resource]. – Access mode : <https://parclick.com/>.
6. Zhao, Z. A Comparative Study of Parking Occupancy Prediction Methods considering Parking Type and Parking Scale [Electronic resource] / Z. Zhao, Y. Zhang, Y. Zhang // Journal of Advanced Transportation. – 2020. – Volume 2020. – Access mode : <https://www.hindawi.com/journals/jat/2020/5624586/>.
7. Parking Availability Prediction Using Traffic Data Services [Electronic resource] / I. Klandev, M. Tolevska, K. Mishev, D. Trajanov // ICT Innovations 2020 : Conference. – 2020. – Access mode : https://www.researchgate.net/publication/344808348_Parking_Availability_Prediction_Using_Traffic_Data_Services.
8. Understanding & Implementing XGBoost – A Machine Learning Algorithm In Data Science [Electronic resource]. – Access mode :

<https://www.turing.com/kb/understanding-and-implementing-xgboost-in-data-science>.

9. Sugino, M. XGBoost [Electronic resource] : its Genealogy, its Architectural Features, and its Innovation / M. Sugino. – Access mode : <https://towardsdatascience.com/xgboost-its-genealogy-its-architectural-features-and-its-innovation-bf32b15b45d2>.

10. XGBoost [Electronic resource]. – Access mode : <https://www.geeksforgeeks.org/xgboost/>.

11. XGBoost Documentation [Electronic resource]. – Access mode : <https://xgboost.readthedocs.io/en/stable/>.

12. Morde, V. XGBoost Algorithm: Long May She Reign! [Electronic resource] / V. Morde. – Access mode : <https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d>.

13. Guide – HERE Traffic API – HERE Developer [Electronic resource]. – Access mode : https://developer.here.com/documentation/traffic/dev_guide/topics/tiles.html.

14. Advantages of PostgreSQL – CYBERTEC. Data Science [Electronic resource]. – Access mode : <https://www.cybertec-postgresql.com/en/postgresql-overview/advantages-of-postgresql/>.

15. Woxx Travel Template [Electronic resource]. – Access mode : <https://templatemo.com/tm-580-woox-travel>.

16. Usage of Parking Lots in Lisbon Portugal [Electronic resource]. – Access mode : <https://www.kaggle.com/datasets/aryvini/usage-of-parking-lots-in-lisbon-portugal>.

ДОДАТОК А
Технічне завдання

Вступ

Паркування власних автомобілів є однією з найактуальніших урбаністичних проблем. Водії автомобілів стикаються з проблемою розміщення власного автомобіля дуже часто, особливо, якщо мешкають, працюють в мегаполісі. Час на паркування в такому випадку зростає. Тому водію, у випадку, якщо він знаходиться в місцях, де може спостерігатися нестача паркувальних місць, важливим є завчасне отримання інформації про те, чи може він розраховувати на паркування на певній ділянці міста. Так водій може вирішити, де йому краще залишити свій автомобіль. Для цього потрібний механізм прогнозування доступності паркувальних місць. Такий же механізм важливим є і для осіб, які володіють або керують парковками. В даній роботі потрібно створити такий механізм і забезпечити його практичне застосування за рахунок створення відповідного програмного забезпечення.

A.1 Підстави для розробки

Розробка дипломної кваліфікаційної роботи магістра виконується на основі завдання за темою «Дослідження та програмна реалізація методів прогнозування доступності паркувальних місць», затвердженою наказом № 363 від 4 листопада 2022 р. за Національним університетом «Запорізька політехніка».

A.2 Призначення розробки

Програма, розробка якої має бути виконана в роботі, призначена для підтримки процесу бронювання місць на паркувальних просторах, пошуку таких просторів і місць, відслідковування їх фактичної та прогнозованої доступності з метою прийняття подальших рішень клієнтами.

А.3 Основні вимоги до програми

А.3.1 Вимоги до функціональних характеристик

Для забезпечення власного призначення програма повинна дозволяти виконувати функції:

- керувати даними паркувального простору, включаючи доступні місця, режими роботи та оплати;
- шукати паркувальні простори;
- шукати місця для паркування на заданий період;
- бронювати місця з паркувального простору;
- керувати виконаними бронюваннями;
- керування власними автомобілями, доступними для клієнта;
- надавати список бронювань клієнта;
- шукати бронювання клієнта за номером його телефону;
- надавати інформацію про зайнятість паркувальних місць у просторі;
- висловлення власних вражень про паркувальний простір після паркування;
- реєструвати і авторизовувати клієнтів у програмі;
- керувати доступом клієнтів і автомобілів;
- надавати список висловлювань про паркувальний простір за обмеженим інтервалом оцінок;
- надавати список власних висловлювань клієнта про паркувальні простори;
- авторизовувати і обмежено реєструвати операторів паркувальних просторів;
- прогнозувати наявність паркувальних місць у просторі на заданий період;
- формувати звіти про задіяність груп місць для паркування за період.

А.3.2 Вимоги до складу та параметрів технічних засобів

Доступ з клієнтського пристрою користувача, який виконує будь-яку роль у вебдодатку, виконується за наявності підключення до мереж інтернет та за наявності актуальної версії будь-якого сучасного браузера.

Для запуску вебсервера, який буде підтримувати роботу вебдодатку безпосередньо, зокрема запускаючи і виконуючи метод, який має бути запропонований в роботі, необхідно скористатися послугами хостингу з можливостями підтримки роботи з системою керування БД PostgreSQL і запуску скриптів за розкладом, або підтримувати власний вебсервер, при цьому він повинен бути з оперативною пам'яттю з мінімальним обсягом 4 Гб, жорстким диском з мінімальним обсягом 1 Гб та щонайменше чотирьохядерним процесором (частота не менше 2 ГГц).

А.3.3 Умови експлуатації

Для розгортання вебдодатку на сервері з подальшим доступом користувачів знадобиться вебсервер. В якості стандартного рішення даної проблеми для додатків, розроблених за допомогою Django, використовується uWSGI. Вебсервер використовується як шлюз і фактично перенаправляє запити, які надходять до нього з клієнтських пристроїв, у Django-додаток.

А.3.4 Вимоги до програмної документації

Технічне завдання і текст програми, що повинен вміщувати фрагменти програми, оформлюються як окремі програмні документи, а опис роботи з програмою представляється в розділі 5 пояснювальної записки.

А.4 Порядок контролю та приймання

Контроль і приймання роботи мають відбуватися за календарним планом, який входить до складу завдання на дипломну кваліфікаційну роботу магістра.

ДОДАТОК Б
Текст програми

```

from django.contrib.auth.mixins import LoginRequiredMixin
from django.http import HttpResponse
from django.shortcuts import render, redirect,
get_object_or_404
from django.urls import reverse_lazy
from django.views.generic import DetailView, ListView
from django.views import View
from parkapp.forms import SpaceForm, SpaceSearchForm,
LotSearchForm, LotForm, CommentForm, PaymodeForm, ModeForm,
BusylotForm, PriceForm
from parkapp.models import Space, Lot, Prediction, Busylot,
Mode, Paymode, Booking, Comment, Price, Operator, Client, Like,
Dislike
from django.urls import reverse
import datetime
import geocoder
import haversine as hs
import holidays
import pandas as pd
from calendar import monthrange
from datetime import date
from geopy.geocoders import Nominatim
from haversine import Unit

class SpaceView(DetailView):
    model = Space
    template_name = 'parkapp/parking_space.html'

    def get(self, request, pk):
        space_object = Space.objects.
select_related('paymodes').get(id = pk)
        lots = Lot.objects.select_related('space')
        lots = lots.filter(space = space_object)
        lots_with_places = []
        day = self.kwargs['day']
        month = self.kwargs['month']
        year = self.kwargs['year']
        hour = self.kwargs['hour']
        minute = self.kwargs['minute']
        if day:
            values = [year, month, day, hour, minute, '00']
            now_time = datetime.datetime(*map(int, values))
            now_date = date(year, month, day)
        else:
            now_time = datetime.datetime.now()

```

```

        now_date = date(now_time.year, now_time.month,
now_time.day)
        ua_holidays = holidays.country_holidays('UA')
        date_type = -1
        if now_date in ua_holidays:
            date_type = 6
        busidess_day = len(pd.bdate_range(now_date,
now_date))
        if bool(busidess_day):
            date_type = 7
        nday = now_date.weekday()
        if date_type == -1 and nday >= 5:
            date_type = 8
        else:
            date_type2 = nday
        parking_modes = Mode.objects.filter(space =
space_object, day = date_type)
        if not parking_modes:
            parking_modes = Mode.objects.filter(space =
space_object, day = date_type2)
        if not parking_modes:
            parking_modes = Mode.objects.filter(space =
space_object, day = 9)
        parking_modes = parking_modes.filter(fromtime__lte
= now_time, totime__gte = now_time)
        for parking_lot in lots:
            busy = Busylot.objects.filter(lot =
parking_lot, fromtime__gte = now_time, totime__lte = now_time)
            busyplaces = busy.places[0]
            places = parking_lot.places
            places = places - busyplaces
            bookings = Booking.objects.filter(lot =
parking_lot, fromtime__gte = now_time, totime__lte = now_time)
            booked_places = bookings.count()
            free_places = places - booked_places
            lot_places = {}
            lot_places ['lot'] = parking_lot
            lot_places ['free_places'] = free_places
            lot_places ['booked_places'] = booked_places
            lot_places ['busy_places'] = busyplaces
            prices = Price.objects.filter(lot =
parking_lot, mode__in = parking_modes)
            lot_places ['prices'] = prices
            lots_with_places.append(lot_places)
        comments = Comment.objects.filter(space =
space_object).order_by('-created')[0:3]

```

```

        modes = Mode.objects.filter(space = space_object)
        page_context = { 'space': space_object, 'lots':
lots_with_places, 'comments': comments, 'modes': modes }
        return render(request, self.template_name,
page_context)

class NewSpaceView(LoginRequiredMixin, View):
    template_name = 'parkapp/new_space_form.html'

    def get(self, request, pk = None):
        space_form = SpaceForm()
        page_context = {'form': space_form }
        return render(request, self.template_name,
page_context)

    def post(self, request, pk = None):
        space_form = SpaceForm (request.POST, request.FILES
or None)

        if not space_form.is_valid():
            page_context = {'form': space_form }
            return render(request, self.template_name,
page_context)

        space = space_form.save(commit = False)
        space.save()
        return redirect(reverse_lazy('parkapp:spaces'))

@method_decorator(csrf_exempt, name='dispatch')
class CommentLikeView(LoginRequiredMixin, View):

    def post(self, request, pk):
        comment_object = get_object_or_404(Comment, id =
pk)

        client_object = Client.objects.get(user =
self.request.user)
        dislike = Dislike.objects.get(client =
client_object, comment = comment_object)
        if not dislike:
            like = Like(client = client_object, comment =
comment_object)

            try:
                like.save()
                comment.liked = comment.liked + 1
                comment.save()
            except IntegrityError as e:
                pass
        return HttpResponse()

```

```

@method_decorator(csrf_exempt, name='dispatch')
class CommentDislikeView(LoginRequiredMixin, View):

    def post(self, request, pk) :
        comment_object = get_object_or_404(Comment, id =
pk)

        client_object = Client.objects.get(user =
self.request.user)
        like = Like.objects.get(client = client_object,
comment = comment_object)
        if not like:
            dislike = Dislike(client = client_object,
comment = comment_object)
            try:
                dislike.save()
                comment.disliked = comment.disliked + 1
                comment.save()
            except IntegrityError as e:
                pass
            return HttpResponseRedirect()

class EditSpaceView (LoginRequiredMixin, View):
    template_name = 'parkapp/new_space_form.html'

    def get(self, request, pk):
        space = get_object_or_404(Space, id = pk)
        space_form = SpaceForm (instance = space)
        page_context = {'form': space_form }
        return render(request, self.template_name,
page_context)

    def post(self, request, pk = None):
        space = get_object_or_404(Space, id = pk)
        space_form = SpaceForm (request.POST, request.FILES
or None, instance = space)
        if not space_form.is_valid():
            page_context = {'form': space_form }
            return render(request, self.template_name,
page_context)
        space = space_form.save(commit = False)
        space.save()
        redirect(reverse_lazy('parkapp:spaces'))

class SpaceSearchView(ListView):
    model = Space
    template_name = 'parkapp/search_space.html'

```

```

def get_queryset(self, *args, **kwargs):
    spaces = super().get_queryset(*args, **kwargs)
    capture = str(self.request.GET.get('name'))
    if capture:
        spaces = spaces.filter(capture__icontains =
capture)

    location = str(self.request.GET.get('address'))
    if location:
        geo_nomin = Nominatim(user_agent="parkapp")
        full_address = geo_nomin.geocode(location)
        coord_lat = full_address.latitude
        coord_long = full_address.Longitude
    if not coord_lat:
        coord_lat = float(self.request.GET.get('lat'))
        coord_long =
float(self.request.GET.get('long'))
    if not coord_lat:
        coo = geocoder.ipinfo('me')
        ll = coo.latlng
        coord_lat = ll[0]
        coord_long = ll[1]
    dist = float(self.request.GET.get('dist'))
    user_coord = (coord_lat, coord_long)
    if dist:
        for space in spaces:
            space_coord = (space.coordinate1,
space.coordinate2)

            meters = hs.haversine(space_coord,
user_coord, unit = Unit.METERS)
            if meters > dist:
                spaces = spaces.exclude(id = space.id)
    space_type = int(self.request.GET.get('type'))
    space_type_restr =
int(self.request.GET.get('type_restr'))
    if space_type_restr and capture:
        spaces = spaces.filter(type__gte = space_type)
    else:
        spaces = spaces.filter(type = space_type)
    height = float(self.request.GET.get('height'))
    if height:
        spaces = spaces.filter(height__gte = height)
    min_points = float(self.request.GET.get('points'))
    if min_points:
        spaces = spaces.filter(points__gte =
min_points)

    return spaces

```

```

        def get_context_data(self, **kwargs):
            page_context = super(SpaceSearchView,
self).get_context_data(**kwargs)
            page_context ['search_form'] = SpaceSearchForm
(initial={
                'capture': self.request.GET.get('name', ''),
                'location': self.request.GET.get('address',
''),
                'coord_long': self.request.GET.get('long', ''),
                'coord_lat': self.request.GET.get('lat', ''),
                'space_type': self.request.GET.get('type',
'0'),
                'height': self.request.GET.get('height',
'213'),
                'dist': self.request.GET.get('dist', '50'),
                'min_points': self.request.GET.get('points',
'1')
            })
            return page_context

class NewLotView(LoginRequiredMixin, View):
    template_name = 'parkapp/new_lot_form.html'

    def get(self, request, pk = None):
        lot_form = LotForm()
        page_context = {'form': lot_form }
        return render(request, self.template_name,
page_context)

    def post(self, request, pk = None):
        lot_form = LotForm (request.POST, request.FILES or
None)

        if not lot_form.is_valid():
            page_context = {'form': lot_form }
            return render(request, self.template_name,
page_context)

        lot = lot_form.save(commit = False)
        lot.save()
        return redirect(reverse('parkapp:space', args =
[lot.space.id]))

class EditLotView (LoginRequiredMixin, View):
    template_name = 'parkapp/new_lot_form.html'

    def get(self, request, pk):

```

```

        lot = get_object_or_404(Lot, id = pk)
        lot_form = LotForm (instance = lot)
        page_context = {'form': lot_form }
        return render(request, self.template_name,
page_context)

    def post(self, request, pk = None):
        lot = get_object_or_404(Lot, id = pk)
        lot_form = LotForm (request.POST, request.FILES or
None, instance = lot)
        if not lot_form.is_valid():
            page_context = {'form': lot_form }
            return render(request, self.template_name,
page_context)
        lot = lot_form.save(commit = False)
        lot.save()
        return redirect(reverse('parkapp:space', args =
[lot.space.id]))

class PredictionView(LoginRequiredMixin, DetailView):
    model = Space
    template_name = 'parkapp/space_prediction.html'

    def get(self, request, pk):
        space_object = Space.objects.get(id = pk)
        lots = Lot.objects.filter(space = space_object)
        lots_with_places = []
        day = self.kwargs['day']
        month = self.kwargs['month']
        year = self.kwargs['year']
        hour = self.kwargs['hour']
        minute = self.kwargs['minute']
        if not day:
            now_time = datetime.datetime.now()
            values = [now_time.year, now_time.month,
now_time.day, '00', '00', '00']
            now_time = datetime.datetime(*map(int, values))
        else:
            values = [year, month, day, '00', '00', '00']
            now_time = datetime.datetime(*map(int, values))
        for parking_lot in lots:
            busy = Busylot.objects.filter(lot =
parking_lot, fromtime__gte = now_time, totime__lte = now_time)
            busyplaces = busy.places[0]
            places = parking_lot.places
            places = places - busyplaces

```

```

        bookings = Booking.objects.filter(
            lot = parking_lot, fromtime__gte = now_time, totime__lte = now_time)
        booked_places = bookings.count()
        free_places = places - booked_places
        lot_places = {}
        lot_places ['lot'] = parking_lot
        lot_places ['free_places'] = free_places
        lot_places ['booked_places'] = booked_places
        lot_places ['busy_places'] = busyplaces
        minplaces = Prediction.objects.get(
            lot = parking_lot, createdfor = now_time)
        lot_places ['predicted_places'] = minplaces
        lots_with_places.append(lot_places)
        page_context = { 'space' : space_object, 'lots':
            lots_with_places }
        return render(request, self.template_name,
            page_context)

```

```

class NewPaymodeView(LoginRequiredMixin, View):
    template_name = 'parkapp/add_paymode_form.html'

    def get(self, request, pk = None):
        paymode_form = PaymodeForm()
        page_context = {'form': paymode_form }
        return render(request, self.template_name,
            page_context)

    def post(self, request, pk = None):
        paymode_form = PaymodeForm (request.POST)
        if not paymode_form.is_valid():
            page_context = {'form': paymode_form }
            return render(request, self.template_name,
                page_context)
        paymode = paymode_form.save(commit = False)
        paymode.save()
        return redirect(reverse('parkapp:space', args =
            [paymode.space.id]))

```

```

class DeletePaymodeView(LoginRequiredMixin, DeleteView):
    model = Paymode

    def get_queryset(self):
        paymodes = super(DeletePaymodeView,
            self).get_queryset()
        operator = get_object_or_404(Operator,
            user=self.request.user)

```

```

        space_id = operator.space.id
        space_object = Space.objects.
select_related('paymodes').get(id = space_id)
        return paymodes.filter(space = space_object)

class NewModeView(LoginRequiredMixin, View):
    template_name = 'parkapp/add_mode_form.html'

    def get(self, request, pk = None):
        mode_form = ModeForm()
        page_context = {'form': mode_form }
        return render(request, self.template_name,
page_context)

    def post(self, request, pk = None):
        paymode_form = ModeForm (request.POST)
        if not mode_form.is_valid():
            page_context = {'form': mode_form }
            return render(request, self.template_name,
page_context)

        mode = mode_form.save(commit = False)
        mode.save()
        return redirect(reverse('parkapp:space', args =
[mode.space.id]))

class DeleteModeView(LoginRequiredMixin, DeleteView):
    model = Mode

    def get_queryset(self):
        modes = super(DeleteModeView, self).get_queryset()
        operator = get_object_or_404(Operator,
user=self.request.user)
        space_id = operator.space.id
        space_object = Space.objects.
select_related('paymodes').get(id = space_id)
        return modes.filter(space = space_object)

class NewPriceView(LoginRequiredMixin, View):
    template_name = 'parkapp/add_price_form.html'

    def get(self, request, pk = None):
        price_form = PriceForm()
        page_context = {'form': price_form }
        return render(request, self.template_name,
page_context)

```

```

    def post(self, request, pk = None):
        price_form = PriceForm (request.POST)
        if not price_form.is_valid():
            page_context = {'form': price_form }
            return render(request, self.template_name,
page_context)

        price = price_form.save(commit = False)
        price.save()
        lot = price.lot
        return redirect(reverse('parkapp:space', args =
[lot.space.id]))

class DeletePriceView(LoginRequiredMixin, DeleteView):
    model = Price

    def get_queryset(self):
        prices = super(DeleteModeView, self).get_queryset()
        operator = get_object_or_404(Operator,
user=self.request.user)
        space_id = operator.space.id
        space_object = Space.objects.
select_related('paymodes').get(id = space_id)
        return prices.filter(lot__space = space_object)

class OperatorSpaceView(LoginRequiredMixin, View):
    template_name = 'parkapp/oparking_space.html'

    def get(self, request, pk = None):
        operator = get_object_or_404(Operator,
user=self.request.user)
        space_id = operator.space.id
        space_object = Space.objects.
select_related('paymodes').get(id = space_id)
        lots = Lot.objects.select_related('space')
        lots = lots.filter(space = space_object)
        lots_with_places = []
        now_time = datetime.datetime.now()
        now_date = date(now_time.year, now_time.month,
now_time.day)
        ua_holidays = holidays.country_holidays('UA')
        date_type = -1
        if now_date in ua_holidays:
            date_type = 6
        busidess_day = len(pd.bdate_range(now_date,
now_date))
        if bool(busidess_day):

```

```

        date_type = 7
    nday = now_date.weekday()
    if nday >= 5:
        date_type = 8
    else:
        date_type2 = nday
    parking_modes = Mode.objects.filter(space =
space_object, day = date_type2)
    if not parking_modes:
        parking_modes = Mode.objects.filter(space =
space_object, day = date_type)
    parking_modes = parking_modes.filter(fromtime__lte
= now_time, totime__gte = now_time)
    for parking_lot in lots:
        busy = Busylot.objects.filter(lot =
parking_lot, fromtime__gte = now_time, totime__lte = now_time)
        busyplaces = busy.places[0]
        places = parking_lot.places
        places = places - busyplaces
        bookings = Booking.objects.filter(lot =
parking_lot, fromtime__gte = now_time, totime__lte = now_time)
        booked_places = bookings.count()
        free_places = places - booked_places
        lot_places = {}
        lot_places ['lot'] = parking_lot
        lot_places ['free_places'] = free_places
        lot_places ['booked_places'] = booked_places
        lot_places ['busy_places'] = busyplaces
        prices = Price.objects.filter(lot =
parking_lot, mode__in = parking_modes)
        lot_places ['prices'] = prices
        lots_with_places.append(lot_places)
    comments = Comment.objects.filter(space =
space_object).order_by('-created')[0:3]
    modes = Mode.objects.filter(space = space_object)
    page_context = { 'space': space_object, 'lots':
lots_with_places, 'comments': comments, 'modes': modes }
    return render(request, self.template_name,
page_context)

```

```

class NewCommentView(LoginRequiredMixin, View):
    template_name = 'parkapp/new_comment_form.html'

```

```

    def get(self, request, pk = None):
        comment_form = CommentCreateForm()
        page_context = {'form': comment_form }

```

```

        return render(request, self.template_name,
page_context)

    def post(self, request, pk = None):
        comment_form = CommentCreateForm ()
        if not comment_form.is_valid():
            page_context = {'form': comment_form }
            return render(request, self.template_name,
page_context)
        comment = comment_form.save(commit = False)
        comment.client = Client.objects.get(user =
self.request.user)
        comment.save()
        space_about = comment.space
        space_comments = Comment.objects.filter(space =
space_about)
        points = 0
        for comment in space_comments:
            points = points + comment.points
        space_about.points = points
        space_about.save()
        return redirect(reverse('parkapp:space', args = [
space_about.id]))

class AuthClientCommentsView(LoginRequiredMixin,ListView):
    model = Comment
    template_name = 'parkapp/client_comments.html'

    def get_queryset(self, *args, **kwargs):
        comments = super().get_queryset(*args, **kwargs)
        current_client = get_object_or_404(Client, user =
self.request.user)
        space_id = operator.space.id
        comments = comments.filter(client = current_client)
        comments = comments.order_by('-created')
        return comments.values()

class ClientCommentsView(LoginRequiredMixin,ListView):
    model = Comment
    template_name = 'parkapp/client_comments.html'

    def get_queryset(self, *args, **kwargs):
        comments = super().get_queryset(*args, **kwargs)
        client_id = int(self.request.GET.get('client'))
        comments = comments.filter(client__id = client_id)
        comments = comments.order_by('-created')

```

```

        return comments.values()

class NewBusylotView(LoginRequiredMixin, View):
    template_name = 'parkapp/add_busylot_form.html'

    def get(self, request, pk = None):
        busylot_form = BusylotForm()
        page_context = {'form': busylot_form }
        return render(request, self.template_name,
page_context)

    def post(self, request, pk = None):
        busylot_form = BusylotForm (request.POST)
        if not busylot_form.is_valid():
            page_context = {'form': busylot_form }
            return render(request, self.template_name,
page_context)

        busylot = busylot_form.save(commit = False)
        busylot.save()
        lot = busylot.lot
        return redirect(reverse('parkapp:space', args =
[lot.space.id]))

class DeleteBusylotView(LoginRequiredMixin, DeleteView):
    model = Busylot

    def get_queryset(self):
        lots = super(DeleteModeView, self).get_queryset()
        operator = get_object_or_404(Operator,
user=self.request.user)
        space_id = operator.space.id
        space_object = Space.objects.
select_related('paymodes').get(id = space_id)
        return lots.filter(lot__space = space_object)

class SpaceCommentsView(LoginRequiredMixin, ListView):
    model = Comment
    template_name = 'parkapp/space_comments.html'

    def get_queryset(self, *args, **kwargs):
        comments = super().get_queryset(*args, **kwargs)
        space_id = int(self.request.GET.get('space'))
        comments = comments.filter(space__id = space_id)
        points_min = int(self.request.GET.get('points'))
        points_max =
int(self.request.GET.get('points_max'))

```

```

        if points_min:
            comments = comments.filter(points__gte =
points_min)
        if points_max:
            comments = comments.filter(points__lte =
points_max)

        comments = comments.order_by('-created')
        return comments.values()

class OperatorModesView(ListView):
    model = Mode
    template_name = 'parkapp/space_modes.html'

    def get_queryset(self, *args, **kwargs):
        modes = super().get_queryset(*args, **kwargs)
        operator = get_object_or_404(Operator, user =
self.request.user)
        space_id = operator.space.id
        space_obj = Space.objects.get(id = space_id)
        modes = modes.filter(space =
space_obj).order_by('fromtime')
        return modes

class OperatorPaymodesView(ListView):
    model = Paymode
    template_name = 'parkapp/space_paymodes.html'

    def get_queryset(self, *args, **kwargs):
        paymodes = super().get_queryset(*args, **kwargs)
        operator = get_object_or_404(Operator, user =
self.request.user)
        space_id = operator.space.id
        space_obj = Space.objects.get(id = space_id)
        paymodes = paymodes.filter(space =
space_obj).order_by('capture')
        return paymodes

class OperatorPricesView(ListView):
    model = Price
    template_name = 'parkapp/space_prices.html'

    def get_queryset(self, *args, **kwargs):
        prices = super().get_queryset(*args, **kwargs)
        operator = get_object_or_404(Operator, user =
self.request.user)
        space_id = operator.space.id

```

```

        space_obj = Space.objects.get(id = space_id)
        prices = prices.filter(lot__space =
space_obj).order_by('mode__fromtime')
        return prices

class OperatorBusylotsView(ListView):
    model = Busylot
    template_name = 'parkapp/space_busy.html'

    def get_queryset(self, *args, **kwargs):
        lots = super().get_queryset(*args, **kwargs)
        operator = get_object_or_404(Operator, user =
self.request.user)
        space_id = operator.space.id
        space_obj = Space.objects.get(id = space_id)
        lots = lots.filter(lot__space =
space_obj).order_by('-fromtime')
        return lots

class LotSearchView(ListView):
    model = Lot
    template_name = 'parkapp/search_lot.html'

    def get_queryset(self, *args, **kwargs):
        lots = super().get_queryset(*args, **kwargs)
        spaces = Space.objects.all()
        location = str(self.request.GET.get('address'))
        if location:
            geo_nomin = Nominatim(user_agent="parkapp")
            full_address = geo_nomin.geocode(location)
            coord_lat = full_address.latitude
            coord_long = full_address.Longitude
        if not coord_lat:
            coord_lat = float(self.request.GET.get('lat'))
            coord_long =
float(self.request.GET.get('long'))
        if not coord_lat:
            coo = geocoder.ipinfo('me')
            ll = coo.latlng
            coord_lat = ll[0]
            coord_long = ll[1]
        dist = float(self.request.GET.get('dist'))
        user_coord = (coord_lat, coord_long)
        if dist:
            for space in spaces:

```

```

        space_coord = (space.coordinate1,
space.coordinate2)
        meters = hs.haversine(space_coord,
user_coord, unit = Unit.METERS)
        if meters > dist:
            spaces = spaces.exclude(id = space.id)
        space_type = int(self.request.GET.get('type'))
        space_type_restr =
int(self.request.GET.get('type_restr'))
        if space_type_restr and capture:
            spaces = spaces.filter(type__gte = space_type)
        else:
            spaces = spaces.filter(type = space_type)
        height = float(self.request.GET.get('height'))
        if height:
            spaces = spaces.filter(height__gte = height)
        min_points = float(self.request.GET.get('points'))
        if min_points:
            spaces = spaces.filter(points__gte =
min_points)

        size_type = int(self.request.GET.get('size_type'))
        lots = lots.filter(space__in = spaces)
        if size_type:
            lots = lots.filter(sizetype = size_type)
        lots_searched = []
        day = self.request.GET.get('day')
        month = self.request.GET.get('month')
        year = self.request.GET.get('year')
        hour = self.request.GET.get('hour')
        minute = self.request.GET.get('minute')
        if day:
            values = [year, month, day, hour, minute, '00']
            now_time = datetime.datetime(*map(int, values))
            now_date = date(year, month, day)
        else:
            now_time = datetime.datetime.now()
            now_date = date(now_time.year, now_time.month,
now_time.day)
        ua_holidays = holidays.country_holidays('UA')
        date_type = -1
        if now_date in ua_holidays:
            date_type = 6
        busidess_day = len(pd.bdate_range(now_date,
now_date))

        if bool(busidess_day):
            date_type = 7

```

```

        nday = now_date.weekday()
        if nday >= 5:
            date_type = 8
        else:
            date_type2 = nday
            parking_modes = Mode.objects.filter(space__in =
spaces, day = date_type2)
            if not parking_modes:
                parking_modes = Mode.objects.filter(space__in =
spaces, day = date_type)
            parking_modes = parking_modes.filter(fromtime__lte
= now_time, totime__gte = now_time)
            for parking_lot in lots:
                busy = Busylot.objects.filter(lot =
parking_lot, fromtime__gte = now_time, totime__lte = now_time)
                busyplaces = busy.places[0]
                places = parking_lot.places
                places = places - busyplaces
                bookings = Booking.objects.filter(lot =
parking_lot, fromtime__gte = now_time, totime__lte = now_time)
                booked_places = bookings.count()
                free_places = places - booked_places
                lot = {}
                lot ['lot'] = parking_lot
                lot ['free_places'] = free_places
                lot ['booked_places'] = booked_places
                lot ['busy_places'] = busyplaces
                prices = Price.objects.filter(lot =
parking_lot, mode__in = parking_modes)
                lot ['prices'] = prices
                if free_places > 0:
                    lots_searched.append(lot)
            return lots_searched

```

```

    def get_context_data(self, **kwargs):
        page_context = super(LotSearchView,
self).get_context_data(**kwargs)
        page_context ['search_form'] = SpaceSearchForm
(initial={
            'size_type': self.request.GET.get('size_type',
'0'),
            'location': self.request.GET.get('address',
''),
            'coord_long': self.request.GET.get('long', ''),
            'coord_lat': self.request.GET.get('lat', ''),

```

```

        'space_type': self.request.GET.get('type',
'0'),
        'height': self.request.GET.get('height',
'213'),
        'day': self.request.GET.get('day', '0'),
        'hour': self.request.GET.get('hour', '0'),
        'minute': self.request.GET.get('minute', '0'),
        'month': self.request.GET.get('month', '0'),
        'year': self.request.GET.get('year', '0'),
        'dist': self.request.GET.get('dist', '50'),
        'min_points': self.request.GET.get('points',
'1')
    })
    return page_context

```

```

class OperatorHourReportView(LoginRequiredMixin, View):
    template_name = 'parkapp/ospace_report.html'

    def get(self, request, pk = None):
        operator = get_object_or_404(Operator,
user=self.request.user)
        space_id = operator.space.id
        space_object = Space.objects.get(id = space_id)
        reports = Report.objects.filter(lot__space =
space_object)
        day = self.kwargs['day']
        month = self.kwargs['month']
        year = self.kwargs['year']
        hour = self.kwargs['hour']
        values = [year, month, day, hour, '00', '00']
        now_time = datetime.datetime(*map(int, values))
        report_hour = reports.get(createdfor = now_time,
period = 0)
        page_context = { 'report': report_hour }
        return render(request, self.template_name,
page_context)

```

```

class OperatorDayReportView(LoginRequiredMixin, View):
    template_name = 'parkapp/ospace_day_report.html'

    def get(self, request, pk = None):
        operator = get_object_or_404(Operator, user =
self.request.user)
        space_id = operator.space.id
        space_object = Space.objects.get(id = space_id)

```

```

        reports = Report.objects.filter(lot__space =
space_object)
        day = self.kwargs['day']
        month = self.kwargs['month']
        year = self.kwargs['year']
        values = [year, month, day, '00', '00', '00']
        now_time = datetime.datetime(*map(int, values))
        report_day = reports.get(createdfor = now_time,
period = 1)
        report_hours = []
        for hour in range(0, 24):
            values = [year, month, day, '00', '00', '00']
            now_time = datetime.datetime(*map(int, values))
            rep = reports.get(createdfor = now_time, period
= 0)
            report_hours.append(rep)
        page_context = { 'report': report_day, 'reports':
report_hours }
        return render(request, self.template_name,
page_context)

class OperatorMonthReportView(LoginRequiredMixin, View):
    template_name = 'parkapp/ospace_month_report.html'

    def get(self, request, pk = None):
        operator = get_object_or_404(Operator, user =
self.request.user)
        space_id = operator.space.id
        space_object = Space.objects.get(id = space_id)
        reports = Report.objects.filter(lot__space =
space_object)
        month = self.kwargs['month']
        year = self.kwargs['year']
        values = [year, month, '1', '00', '00', '00']
        now_time = datetime.datetime(*map(int, values))
        report_month = reports.get(createdfor = now_time,
period = 2)
        report_days = []
        days_range = monthrange(year, month)
        days_number = days_range[1]
        for day in range(0, days_number):
            values = [year, month, day, '00', '00', '00']
            now_time = datetime.datetime(*map(int, values))
            rep = reports.get(createdfor = now_time, period
= 1)
            report_days.append(rep)

```

```

        page_context = { 'report': report_month, 'reports':
report_days }
        return render(request, self.template_name,
page_context)

class OperatorYearReportView(LoginRequiredMixin, View):
    template_name = 'parkapp/ospace_month_report.html'

    def get(self, request, pk = None):
        operator = get_object_or_404(Operator, user =
self.request.user)
        space_id = operator.space.id
        space_object = Space.objects.get(id = space_id)
        reports = Report.objects.filter(lot__space =
space_object)
        year = self.kwargs['year']
        values = [year, '1', '1', '00', '00', '00']
        now_time = datetime.datetime(*map(int, values))
        report_year = reports.get(createdfor = now_time,
period = 3)

        report_months = []
        for month in range(1, 13):
            values = [year, month, day, '00', '00', '00']
            now_time = datetime.datetime(*map(int, values))
            rep = reports.get(createdfor = now_time, period
= 2)

            report_months.append(rep)
        page_context = { 'report': report_year, 'reports':
report_months }
        return render(request, self.template_name,
page_context)

class OperatorPredictionView(LoginRequiredMixin, View):
    template_name = 'parkapp/ospace_prediction.html'

    def get(self, request, pk = None):
        operator = get_object_or_404(Operator, user =
self.request.user)
        space_id = operator.space.id
        space_object = Space.objects.get(id = space_id)
        lots = Lot.objects.filter(space = space_object)
        lots_with_places = []
        day = self.kwargs['day']
        month = self.kwargs['month']
        year = self.kwargs['year']
        hour = self.kwargs['hour']

```

```

minute = self.kwargs['minute']
if not day:
    now_time = datetime.datetime.now()
    values = [now_time.year, now_time.month,
now_time.day, now_time.hour, '00', '00']
    now_time = datetime.datetime(*map(int, values))
else:
    values = [year, month, day, hour, '00', '00']
    now_time = datetime.datetime(*map(int, values))
for parking_lot in lots:
    busy = Busylot.objects.filter(lot =
parking_lot, fromtime__gte = now_time, totime__lte = now_time)
    busyplaces = busy.places[0]
    places = parking_lot.places
    places = places - busyplaces
    bookings = Booking.objects.filter(lot =
parking_lot, fromtime__gte = now_time, totime__lte = now_time)
    booked_places = bookings.count()
    free_places = places - booked_places
    lot_places = {}
    lot_places ['lot'] = parking_lot
    lot_places ['free_places'] = free_places
    lot_places ['booked_places'] = booked_places
    lot_places ['busy_places'] = busyplaces
    minplaces = Prediction.objects.get(lot =
parking_lot, createdfor = now_time)
    lot_places ['predicted_places'] = minplaces
    lots_with_places.append(lot_places)
    page_context = { 'space' : space_object, 'lots':
lots_with_places }
    return render(request, self.template_name,
page_context)

```

ДОДАТОК В
Слайди презентації

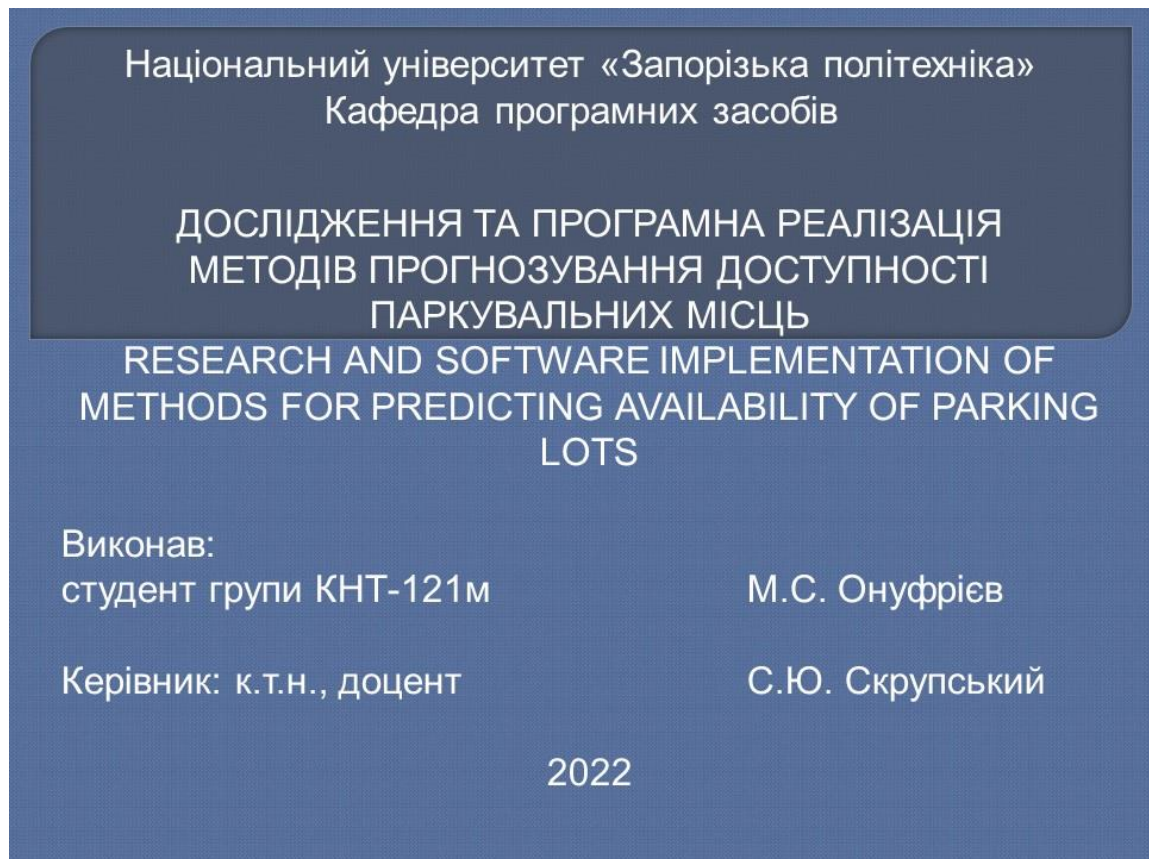


Рисунок В.1 – Титульний слайд

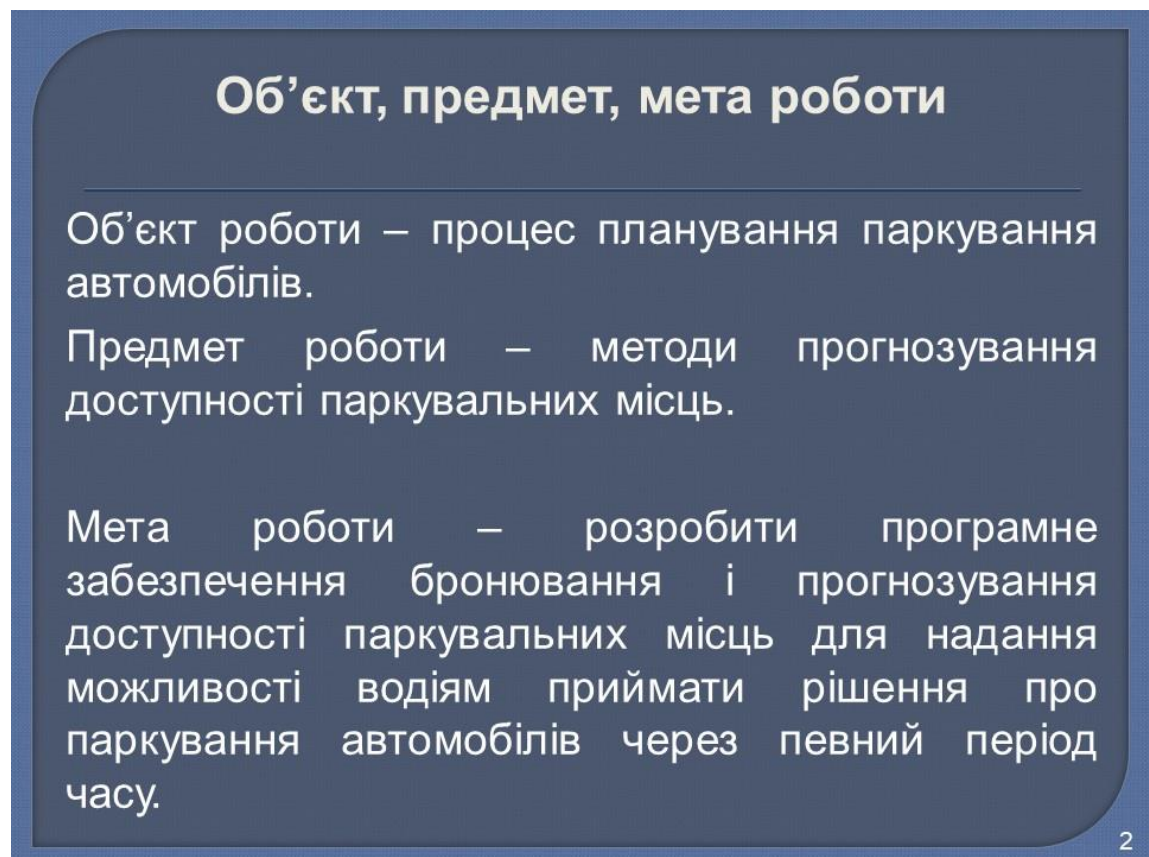


Рисунок В.2 – Об'єкт, предмет, мета роботи

Завдання роботи

- ✓ дослідити методи прогнозування доступності паркувальних місць;
- ✓ спроектувати програмне забезпечення бронювання і прогнозування доступності паркувальних місць;
- ✓ реалізувати програмне забезпечення.

3

Рисунок В.3 – Завдання роботи

Порівняння методів прогнозування доступності паркувальних місць			
Критерій	Стаття [Z. Zhao, Y. Zhang, Y. Zhang]	Стаття [I. Klandev, M. Tolevska, K. Mishev, D. Trajanov]	Власний метод
Показник, прогнозування якого відбувається	Середня кількість зайнятих місць за годину	Кількість доступних місць на відповідну мить	Мінімальна кількість доступних (доступність) місць протягом години
Період прогнозу	Година	Година	Година
Вхідні параметри	Часовий ряд зайнятості місць, масштаб і тип паркувального простору	Номер години, день тижня, часовий ряд доступності місць протягом останніх хвилин, середнє значення і стандартне відхилення протягом минулого періоду, рівень трафіку	Район, номер години, день тижня, часовий ряд мінімальної доступності місць протягом останніх годин, рівень трафіку
Модель в основі методу	Метод опорних векторів	XGBoost	XGBoost

4

Рисунок В.4 – Порівняння методів прогнозування доступності паркувальних місць

Метод прогнозування доступності паркувальних місць

Виконання формування вибірки даних за виконаними спостереженнями за роботою паркувальних майданчиків, що охоплює наступні атрибути:

- район: для цього все місто має бути розбито на певні райони, для чого може використовуватися або стандартний поділ міста на райони, або відбуватися розподіл на зони активності (діловий центр, історичний центр, спальний район);

- номер години;

- номер дня тижня;

- часовий ряд мінімальної доступності місць протягом останніх годин;

- рівень трафіку;

- вихідна ознака – мінімальна кількість доступних місць, яка одночасно була на паркувальному просторі протягом години часу.

Оброблення вибірки шляхом її нормалізації, а після цього навчання моделі на основі алгоритму XGBoost.

Застосування навченої моделі для прогнозування на початку години значення мінімальної кількості доступних місць протягом години, яка розпочнеться через годину, на основі набору значень вхідних показників, які перед цим мають бути перетворені на основі правил, які застосовувались для навчальної вибірки. При цьому основний набір значень показників має бути отримано від паркувального простору, а значення трафіку має бути отримано від того сервісу, який використовувався для формування вибірки для визначення рівня трафіку (стандартно 4 рівні трафіку).

5

Рисунок В.5 – Метод прогнозування доступності паркувальних місць

Порівняння рішень для розробки вебдодатку

Критерій	Комплексне рішення на основі мови Python	Комплексне рішення на основі мови PHP
Складність написання коду	Просте вираження складних рішень	Складні рішення писати складніше, код читати також складніше
Підтримка дерев рішень	Scikit-learn, xgboost	Відсутня надійна бібліотека
Робота з даними	Велика перевага: pandas, numpy і велика кількість додаткових пакетів	Якісні рішення відсутні
Фреймворк веброзробки	Django	Laravel, Symphony, Yii

6

Рисунок В.6 – Порівняння рішень для розробки вебдодатку



Рисунок В.7 – Діаграми прецедентів



Рисунок В.8 – Схема структури бази даних



Рисунок В.9 – Схема функціонування програми

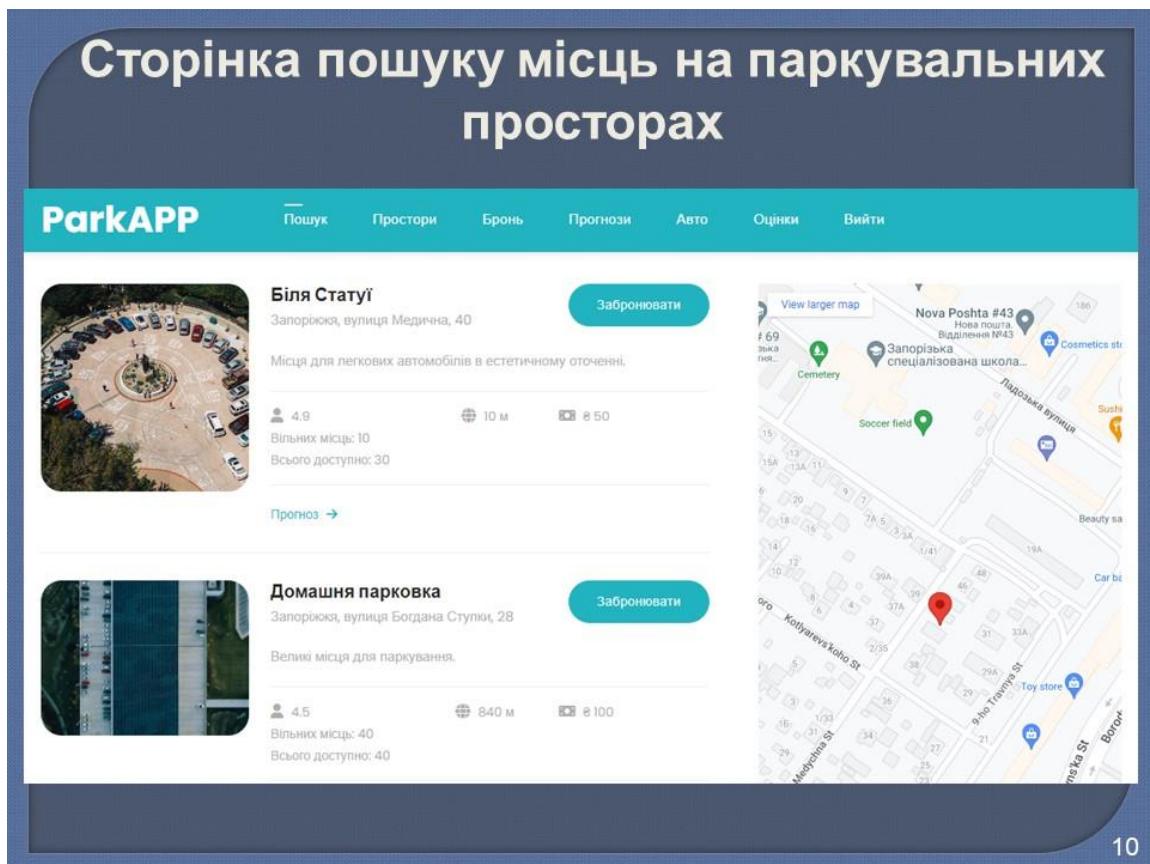


Рисунок В.10 – Сторінка пошуку місць на паркувальних просторах

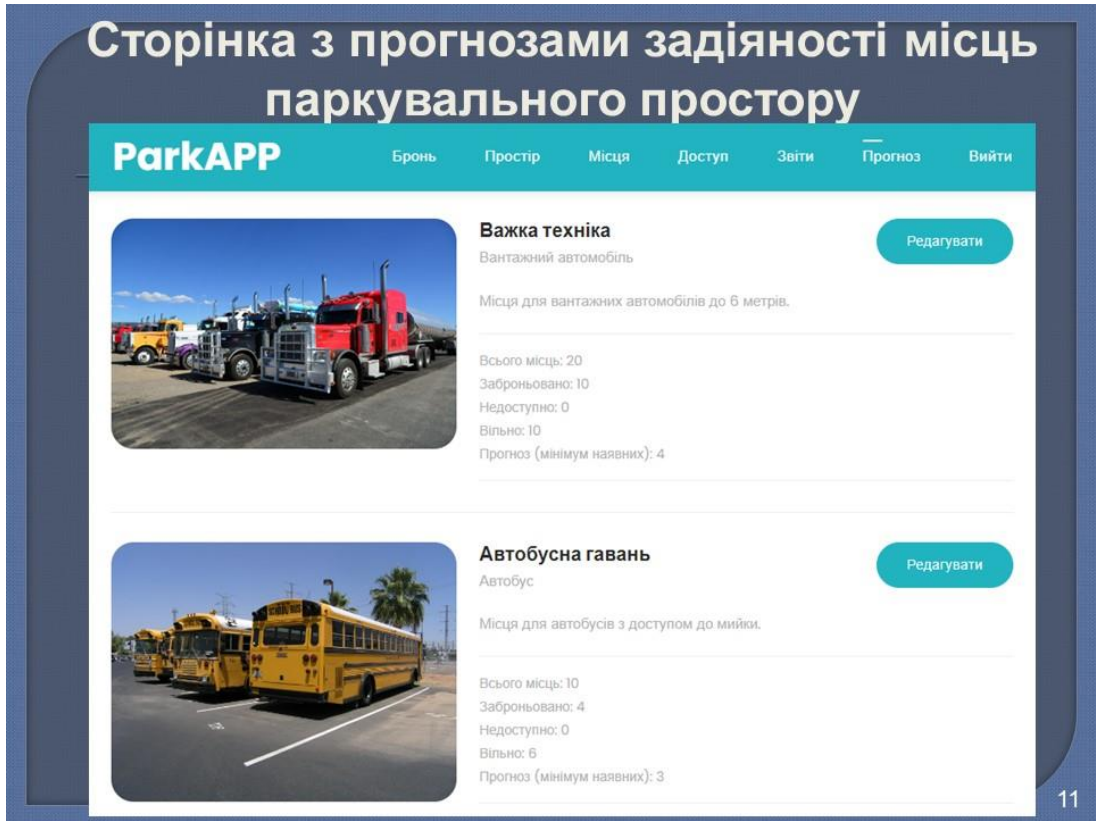


Рисунок В.11 – Сторінка з прогнозами задіяності місць паркувального простору

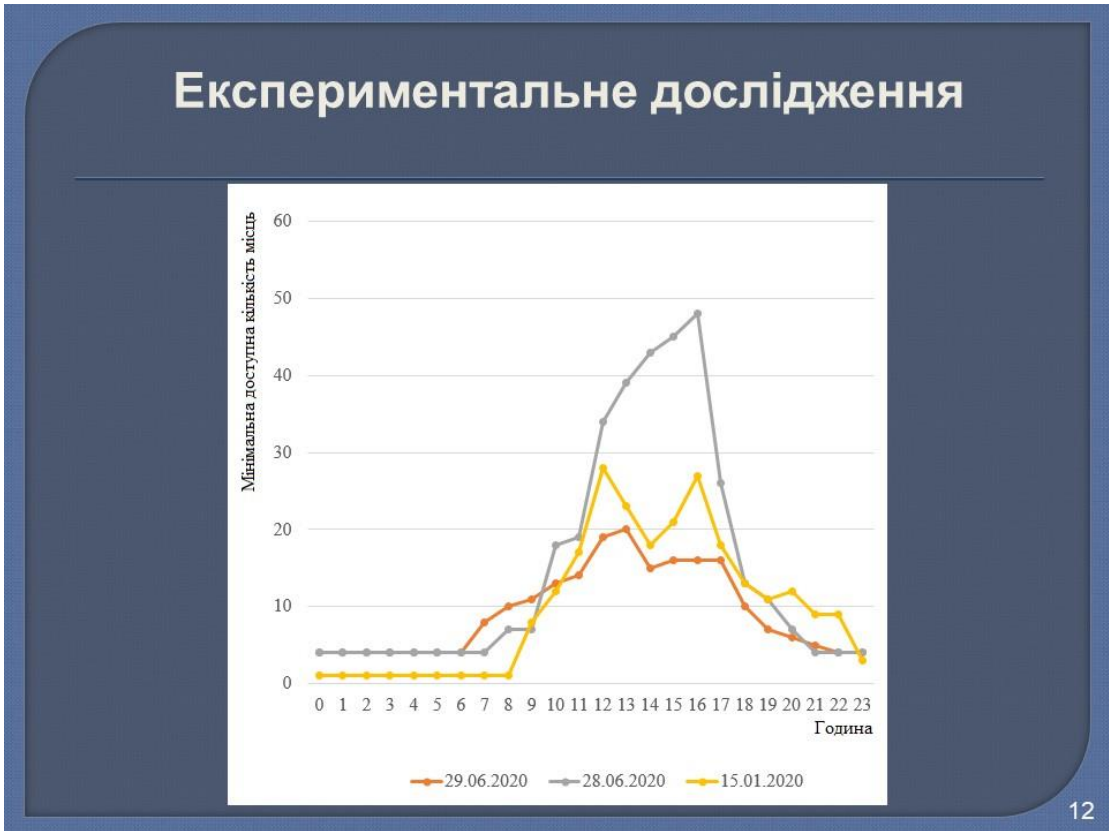


Рисунок В.12 – Експериментальне дослідження

Результати експериментального дослідження

Метод / модель	Коренева середньоквадратична похибка
SARIMAX	0,421
Метод опорних векторів	0,373
XGBoost	0,251
Запропонований метод	0,197

13

Рисунок В.13 Результати експериментального дослідження

Висновки

Для розроблення програмного забезпечення спочатку було проаналізовано наявні аналоги, на основі чого визначено прогнозування доступності паркувальних місць як особливість розробки та виділено функції, які програма має виконувати, після цього було промодельовано функціональні вимоги до програми, вирішено використовувати Python і Django для розробки, що було посилено за рахунок бібліотек для оброблення даних, інтелектуального аналізу даних, прийняття допоміжних прикладних рішень, визначено структуру БД з врахуванням необхідності як збереження основних даних стосовно паркувальних просторів і паркування на них, так і результатів використання методу прогнозування доступності паркувальних місць, визначено порядок роботи та в підсумку реалізовано програмне забезпечення.

Для розроблення методу прогнозування доступності паркувальних місць виконано дослідження існуючих методів прогнозування, в тому числі експериментальне на основі застосування вибірки даних, представлено власний метод, який підтвердив отримання кращих результатів порівняно з іншими існуючими.

14

Рисунок В.14 – Висновки