Parking system

ЗМІСТ

ВСТУП

Актуальність теми. Аналізуючи кількість авто в містах України, можна побачити, що вона постійно зростає. Це породжує ряд проблем для жителів міста. Зокрема найбільшою проблемою є пошук вільного паркувального місця біля великих супермаркетів, багатоповерхових житлових будинків чи інших місць скупчення людей. Для пошуку вільного паркувального місця зазвичай використовується чимало часу, що нерідко вносить свої корективи в розпорядок дня власника авто.

Актуальність обраної теми полягає в тому, що реалізація системи розпізнавання вільних місць для паркування автомобілів надасть можливість користувачу легко та швидко визначити, чи має конкретна парковка вільні місця та зорієнтуватися в якій частині парковки ці місця розміщені. Функціональні можливості системи значно полегшать паркувальний процес для жителів міста, які застосовують авто як засіб пересування по даному місту.

Метою роботи є проектування архітектури, розробка алгоритмів роботи та реалізація системи розпізнавання вільних місць для паркування авто та подальшого повідомлення результату через месенджер Telegram.

Визначена мета роботи обумовлює появу наступних завдань:

– аналіз елементів парковки та паркувального місця;

– проектування складових частин системи та їх взаємодію;

– обрати засоби реалізації програмної системи;

– реалізувати систему для розпізнавання вільних місць для паркування авто;

– описати можливості використання та налаштування системи.

Об’єктом дослідження є автоматизація процесу пошуку вільного паркувального місця для автомобіля.

Предметом дослідження є використання веб-технологій та технологій розпізнавання об’єктів на зображенні для автоматизації пошуку вільного місця для паркування авто.

Під час реалізації системи було застосовано методи об’єктно-орієнтованого проектування, веб-програмування, реляційних та нереляційних баз даних.

Розроблена програмна система надасть змогу користувачам здійснювати пошук вільних паркувальних місця на обраній парковці, витрачаючи при цьому мінімум часу.

РОЗДІЛ 1. АНАЛІЗ НАПРЯМКІВ ВИКОРИСТАННЯ ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ ДЛЯ ПОШУКУ ВІЛЬНОГО ПАРКУВАЛЬНОГО МІСЦЯ

1.1. Постановка задачі

Основною метою впровадження програмної системи є мінімізація часу на пошук вільного місця для паркування автомобіля. Впровадження системи допоможе користувачам, які пересуваються по місту на власному авто, проводити пошук вільних паркувальних місць, обираючи при цьому потрібну парковку зі списку доступних.

В даній роботі необхідно розробити наступні компоненти системи:

– чат-бот для месенджеру Telegram;

– панель адміністратора для управління системою;

– компонент для розпізнавання вільних паркувальних місць в реальному часі за допомогою IP-камери відеоспостереження.

Реалізація поставлених завдань для чат-бота складається з наступних етапів:

1. Розробити функціонал реєстрації користувачів при запуску бота через команду /start.
2. Розробити функціонал для показу всіх доступних парковок для пошуку вільних паркувальних місць.
3. Додати інформацію про бота та реалізувати вивід всіх доступних його команд через команду /help.
4. Реалізувати функціонал для можливості запиту на пошук вільного паркувального місця для обраної парковки.
5. Впровадити функціонал для обробки даних, отриманих від компоненту розпізнавання з подальшим відправленням інформації до користувача у вигляді тексту та фото.

6. Розробити можливість налаштування автоматичного пошуку вільного місця для заданої парковки в заданий час.

Реалізація поставлених завдань для адмін-панелі управління системою складається з наступних етапів:

1. Розробити функціонал авторизації в панель адміністратора через логін та пароль.
2. Впровадити можливість додавання нових користувачі-адміністраторів.
3. Розробити функціонал для управління парковками.
4. Розробити функціонал для управління даними, необхідними для налаштування доступу до IP камер на парковці.
5. Розробити функціонал для управління записами (редагування та видалення) «автоматичного пошуку вільного місця для заданої парковки в заданий час» користувачів.
6. Розробити верстку сайту та frontend логіку за допомогою HTML5, CSS3, JavaScript.

Реалізація поставлених завдань для скрипту розпізнавання вільних паркувальних місць складається з наступних етапів:

1. Розробити функціонал для підключення до IP камери відеоспостереження використовуючи необхідні дані з БД.
2. Розробити можливість отримання кадру для розпізнавання з відео потоку.
3. Впровадити функціонал для розпізнавання авто на кадрі та отримання координат «знайдених» авто.
4. Розробити алгоритм який визначає доступність паркувального місця по знайденим координатам.
5. Розробити можливість моніторингу запитів від чат-бота.
6. Впровадити функціонал для генерації відповіді для чат-бота.
7. Розробити функціонал для збереження фото з відміченим вільним паркувальним місцем.

Результатом реалізації поставлених завдань є система розпізнавання вільних місць для паркування автомобілів з інформуванням користувачів за допомогою чат-бота в месенджері Telegram, що включає в собі наступні 3 компоненти:

- чат-бот;
- панель адміністратора;
- функціонал для розпізнавання вільних паркувальних місць.

1.2. Аналіз аналогів програмного продукту

Програмна система розпізнавання вільних місць для паркування авто забезпечує швидке отримання інформації про наявність та розташування незайнятих паркувальних місць на обраній парковці, використовуючи месенджер Telegram. Крім того, дана система надає можливість налаштовувати автоматичний пошук вільного місця на заданій парковці в заданий час. На даний час можна знайти декілька подібних систем, що допомагають вирішенню задач, пов’язаних з пошуком вільного паркувального місця на відео. Для розгляду аналогів вибрані програмні продукти наведені в табл. 1.1.

Таблиця 1.1

Перелік розглянутих систем

Назва	Розробник	Необхідність встановлення спеціального обладнання
ParkEyes	ParkEyes	Так
Integra-Parking	Integra-S	Так
Acer Smart Parking	Acer	Так

Розглянемо аналоги даної програмної системи детальніше.

ParkEyes. Лідер систем паркувальної навігації на базі відеокамер з революційними функціями безпеки, ефективної навігації до вільного місця, індивідуального контролю вільних місць і пошуку місця парковки автомобіля.

Перевагами даної системи є: розпізнавання паркувальних місць, розпізнавання державних номерних знаків, запис та збереження відео 24/7. Головним недоліком даної системи є її складність та дороговизна в установці.

Integra-Parking. Перевагами даної системи є графічне відображення всіх місць для паркування в плані паркування та синхронізація звітів про автомобілі та автомобілі, закріплені відеокамерою, яка в'їхала на територію паркування картками. Недоліками системи є відсутність мобільного додатку та відсутність можливості автоматичного відправлення інформації користувачу в заданий період часу.

Acer Smart Parking. Перевагами цієї системи є наявність зручного мобільного додатку, можливість будувати маршрут до вільного паркувального місця, автоматичне повідомлення про наявність незайнятого місця. Недоліком даної системи є необхідність обладнання кожного паркувального місця паркоматом, що значно ускладнює установку такої системи на парковках міста.

Порівняємо наші аналоги за нефункціональними можливостями.

Таблиця 1.2.

Нефункціональні можливості

Програма	Версія для одного користувача	Мережева версія	Веб-інтерфейс	Мобільна версія
ParkEyes	Ні	Так	Так	Так
Integra-Parking	Ні	Так	Так	Ні
Acer Smart Parking	Ні	Так	Ні	Так

В таблиці 1.3 розглянуті програми-аналоги по функціональності.

Таблиця 1.3.

Функціональні можливості

Програма	Автоматичне оновлення	Запис відео	Бронювання місць	Пошук в заданий час	Розпізнавання державних номерів
ParkEyes	Так	Так	Так	Ні	Ні
Integra-	Так	Так	Так	Ні	Ні

Parking					
Acer Smart Parking	Так	Ні	Так	Ні	Так

Переглянувши функціонал аналогів та проаналізувавши проблему з паркуванням жителів міст України, було сформульовано функціональні та нефункціональні вимоги до системи. Насамперед система повинна бути простою та не дорогою в установці та експлуатації. Система переважно буде використовуватися за допомогою смартфону. Управління системою повинно здійснюватися адміністратором за допомогою веб-додатку.

Таким чином основними рисами розроблюваної системи мають бути:

- чат-бот в месенджері Telegram;
- панель адміністратора у вигляді веб-сайту;
- моніторинг парковки на предмет вільних паркувальних місць за допомогою встановлених там IP камер відео спостереження;
- відправка інформації про наявність вільних паркувальних використовуючи текстові та графічні повідомлення;
- налаштування авто пошуку для заданої парковки в заданий час.

Висновки до розділу 1

В даному розділі було визначено основне завдання та мету впровадження програмної системи для розпізнавання вільних паркувальних місць. Також було проведено пошук аналогів даного програмного продукту та проаналізовано їх функціональні та нефункціональні можливості.

Таким чином було вирішено, що програмна система повинна мати наступну функціональність:

- чат-бот в месенджері Telegram;
- панель адміністратора у вигляді веб-сайту;

- моніторинг парковки на предмет вільних паркувальних місць за допомогою встановлених там IP камер відео спостереження;
- відправка інформації про наявність вільних паркувальних використовуючи текстові та графічні повідомлення;
- налаштування авто пошуку для заданої парковки в заданий час.

РОЗДІЛ 2. ПРОЕКТУВАННЯ СИСТЕМИ ДЛЯ ПОШУКУ ВІЛЬНОГО ПАРКУВАЛЬНОГО МІСЦЯ

2.1. Визначення варіантів використання та об’єктно-орієнтованої структури системи

Програмна система для розпізнавання вільних паркувальних місць має основні цілі: система створюється з метою полегшення пошуку вільного місця для паркування автомобіля, затрачаючи при цьому мінімум часу.

Користувач повинен мати змогу користуватися системою, роблячи мінімум маніпуляцій та налаштувань. Крім того, програмна система повинна бути простою та швидкою в адмініструванні. Також вона повинна підтримувати одночасну роботу декількох користувачів-адміністраторів.

Зовнішні користувачі повинні мати змогу виконувати наступні дії:

1. Перегляд списку всіх доступних парковок для пошуку вільних паркувальних місць.

2. Перегляд інформації про чат-бот.

3. Перегляд списку доступних команд для виконання чат-ботом.

4. Запит на пошук вільного паркувального місця для обраної парковки.

5. Налаштування автоматичного пошуку вільного місця на заданій парковці в заданий час.

6. Автоматична реєстрація в системі в момент першого запуску чат-боту.

Користувачі-адміністратори повинні мати змогу керувати кількістю та станом парковок, налаштувань камер, редагувати та видаляти дані користувачів. Нижче наведено список функцій для користувачів-адміністраторів:

1. Управління даними користувачів.

2. Додавання/видалення нових користувачів-адміністраторів.

3. Управління списком доступних парковок.

4. Управління налаштуваннями камер.

При реалізації панелі адміністратора було використано мову програмування PHP та шаблон MCV. Інформацію про основні класи наведено нижче:

- Core – клас-ядро шаблону MVC. Має три методи: Init – ініціалізація сесії, бази даних та головного template файлу; Run – логіка виклику контролера та його функції в залежності від поточного url; Done – відображення результату у вигляді HTML розмітки;
- Template – клас для роботи з логікою відображення. Має наступні методи: getParam, setParam, setParams, які слугують для передачі параметрів та змінних в template файл. Метод getHTML підключає потрібний template файл та повертає його контент у вигляді рядка;
- DatabaseSignleton – клас для виконання прямих запитів до бази даних та роботи з нею в цілому;
- Admin_Controller – головний контролер для роботи панелі адміністратора. Цей клас використовує Admin_Model - для роботи з даними та Admin_View -–для логіки відображення;
- Admin_Model – клас-модель, за допомогою якого реалізована робота з базою даних для панелі адміністратора. Містить в собі методи для виконання CRUD операцій з базою даних. Також тут присутні методи, які відповідають за логіку авторизації та додавання нового користувача-адміністратора;
- Admin_View – складова MVC для логіки відображення. Клас використовується контролером та в залежності від його методу, виводить користувачу той чи інший template файл з параметрами;

При реалізації чат-боту та скрипта розпізнавання на мові програмування Python, було спроектовано наступні класи та методи:

- TelegramBot – клас, який відповідає за всю логіку роботи чат-бота Telegram. Містить в собі методи-команди чат бота, які починаються з

“command_”. Також містить метод initCommands, який ініціалізує вищесказані методи для виконання їх ботом;

- FreeParkingDetector – клас, який реалізовує алгоритми розпізнавання вільних паркувальних місць на відео з IP камери, та генерує об’єкт відповідь для відправки в чат-бот. Клас містить метод get_camera_path, який дістає необхідну інформацію з БД та генерує рядок підключення до IP камери. Метод process_available_parking відповідає за генерацію відповіді з необхідною інформацією, яка буде оброблена на стороні чат-бота. Головний метод класу, який і відповідає за логіку розпізнавання – run.
- MaskRCNNConfig – клас-конфігурація для Mask R-CNN моделі, яка задає параметри розпізнавання об’єктів на зображенні. Даний клас застосовується при створенні об’єкта моделі.
- MysqlClient – клас, який реалізує роботу з СУБД MySQL на стороні мови програмування Python. Використовується для вставки, редагування, видалення та отримання даних. Є своєрідним «єднальним елементом» між Python скриптами та панеллю адміністратора, реалізованою на мові програмування PHP.
- RedisClient – клас, який реалізує роботу із NoSQL сховищем даних Redis на мові програмування Python. За допомогою Redis здійснюється постійний зв'язок в реальному часі між Python скриптами. Найголовніші методи класу: process_bot_request та process_detection_respose, які слугують за обробку запитів та відповідей в чат-боті та скрипті розпізнавання.

Програмна система реалізує широкий спектр функцій, серед яких є авторизація користувачів та адміністраторів, розпізнавання вільного паркувального місця на конкретній парковці, менеджмент даних зі сторони адміністратора. Все це забезпечується класами та методами описаними вище.

2.2. Проектування та реалізація алгоритмів роботи системи

Основними трьома компонентами системи є чат-бот, скрипт розпізнавання вільного паркувального місця та панель адміністратора.

Після запуску Telegram чат-бота користувач може обрати наступні дії: переглянути інформацію про бот, вивести на екран список команд бота, чи показати доступні дії у вигляді кнопкового меню. Після того, як користувач обирає пункт «Знайти вільне місце» чи «Налаштувати авто пошук», перед ним з’являється кнопкове меню зі списком всіх доступних парковок. Після обирання однієї зі списку, в першому випадку користувачу приходить результат, а в другому випадку користувач повинен обрати час та дні тижня для автоматичного пошуку. Якщо користувач бажає завершити роботу з чат-ботом, він може вийти з бесіди, в іншому випадку можна обрати будь яку дію заново.

Другим важливим компонентом системи є скрипт розпізнавання вільного місця для паркування авто на парковці.

Скрипт розпізнавання запускається один раз і виконується до тих пір, поки є доступ до відео потоку IP камери. З відео береться один кадр і за допомогою алгоритму перевіряється чи є на парковці наявні вільні місця. В той же момент перевіряється чи є в наявності запити від чат-бота. В залежності від комбінації цих факторів, скрипт або генерує відповідну відповідь і відправляє її до чат-бота, або бере наступний кадр і починає описані дії заново.

Нижче наведено діаграму активності даного скрипта.

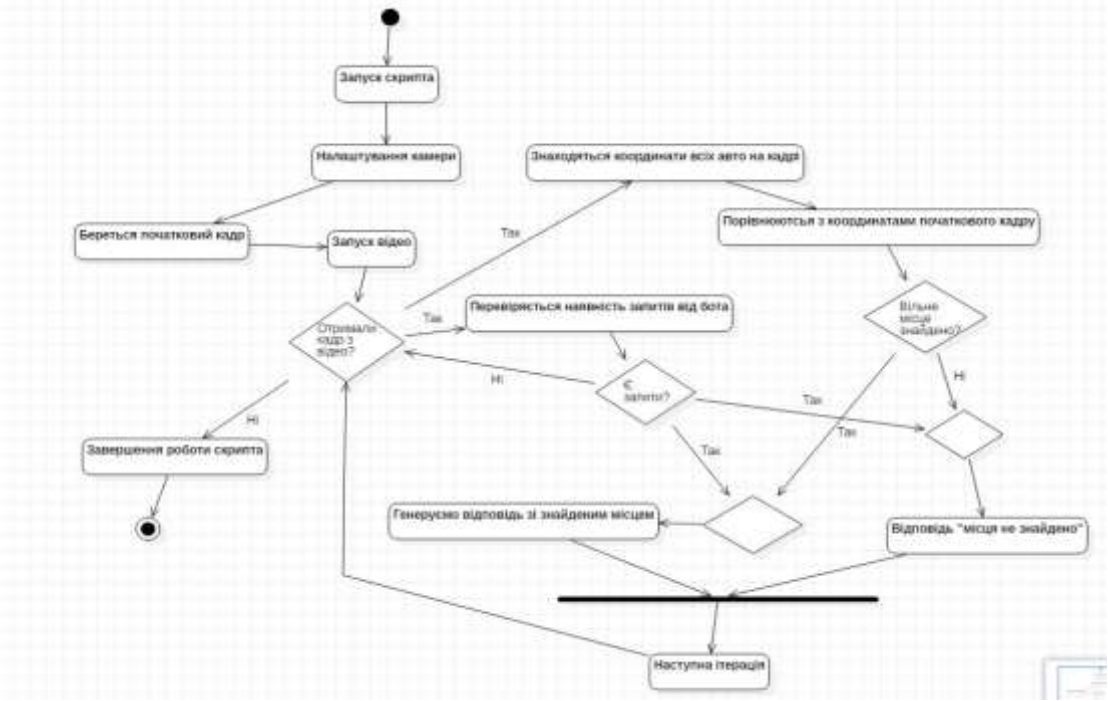

Рис. 2.1. Діаграма активності розпізнавання паркувального місця

Таким чином нами було розглянуто основні компоненти програмної системи для розпізнавання вільних місць для паркування авто та проаналізовано взаємодію класів системи під час роботи різних процесів системи.

2.3 Реалізація системи розпізнавання вільних місць для паркування авто

Для роботи з базою даних в мові програмування PHP було використано технологію доступу до БД PDO. Було створено клас «DatabaseSingleton», використовуючи патерн сінглтон[4], за допомогою якого здійснюються всі запити до бази даних з панелі адміністратора. Код найголовніших методів класу наведено нижче:

```
public static function getInstance() : DatabaseSingleton
{
if (null === static::$instance) {
static::$instance = new static();
}
return static::$instance;
```

```
}

public function SelectAllFromTable($table, $id = null) {
if ($id == null) {
$sql = "SELECT * FROM $table";
$sth = $this->Pdo->prepare($sql);
$sth->execute();
return $sth->fetchAll();
} else {
$sql = "SELECT * FROM $table WHERE id = ?";
$sth = $this->Pdo->prepare($sql);
$sth->execute(array($id));
return $sth->fetch();
}
}

public function checkAdmin($u, $p) {
$sql = "SELECT * from users where login = ?";
$sth = $this->Pdo->prepare($sql);
$sth->execute(array($u));
$user = $sth->fetch();
if ($user['password'] === md5($p)) {
return true;
} else {
return false;
}
}
```

Для реалізації логіки показу структури темплейтів в панелі адміністратора, було використано наступний метод:

```
public function Menu($actionName, $tableName = null, $id = null, $type = null) {
$tpl = new Template('template/admin/adminpanel.tpl');
switch ($actionName) {

case 'panel' : $tpl->SetParams(array('AdminContent' => $this->Dashboard())); break;
case 'addnew' : $tpl->SetParams(array('AdminContent' => $this->AddNew($type))); break;

case 'managelocation' : $tpl->SetParams(array('AdminContent' => $this->ManageLocation()));
break;
case 'managecamera' : $tpl->SetParams(array('AdminContent' => $this->ManageCamera())); break;
```

```
case 'manageuser' : $tpl->SetParams(array('AdminContent' => $this->ManageUser())); break;
case 'scheduling' : $tpl->SetParams(array('AdminContent' => $this->Scheduling($id))); break;
case 'edit' : $tpl->SetParams(array('AdminContent' => $this->Edit($tableName, $id))); break;
case 'useradd' : $tpl->SetParams(array('AdminContent' => $this->NewUser())); break;
}
return $tpl->GetHTML();
}
```

Для реалізації спілкування між 2-ома незалежними компонентами системи, а саме: чат-бота та скрипта розпізнавання, було застосовано NoSQL сховище даних Redis. Нижче наведено код основних методів класу «RedisClient» на мові програмування Python:

```
 def send_recognition_request(self, values):
try:
   self.r.publish("parkingCheckingRequest", json.dumps(values))
   return True
except:
   return False

def process_bot_request(self, frame, additional, callback=None):
   counter = 0
   while counter < 15:
      message = self.pReq.get_message(ignore_subscribe_messages=True)
      if message:
         parking_info = self.json_decode(message['data'])
         if isinstance(parking_info, dict) is False:
            continue
         if callback:
            response = callback(parking_info, frame, additional)
            if response:
               self.r.publish("parkingCheckingResponse", json.dumps(response))
      counter += 1

def process_detection_respose(self, userId, parkingId):

   counter = 0
   while counter < 15:
      message = self.pResp.get_message(ignore_subscribe_messages=True)
      if message:
         detection_response = self.json_decode(message['data'])
         if isinstance(detection_response, dict) is False:
            continue
         if (detection_response['success'] == False):
            return False
         if detection_response['userId'] == userId and detection_response['parkingId'] == parkingId:
            return detection_response
   return False
```

Для створення логіки роботи чат-бота, було використано мову програмування Python та її бібліотеку «telegram.ext». Нижче наведено код двох

найголовніших методів класу чат-бота, а саме: «check_available_parking_places» «show_result_to_user» та «show_available_parking_place»:

```
def check_available_parking_places(self, parking_name, bot, update, userId = None, parking_id = None,
schedule_id = None):
    if schedule_id is not None:
        self.mysqlClient.delete_schedule_by_id(schedule_id)
    if userId is None and parking_id is None:
        parking_id = self.get_parking_id_by_name(parking_name)
        userId = update.message.from_user.id
    isSuccess = self.redisClient.send_recognition_request({
        'userId': userId,
        'parkingId': parking_id
    })
    if isSuccess:
        self.show_result_to_user(userId, parking_id, bot, update)
    else:
        update.message.reply_text("Помилка відправлення запиту")

def show_result_to_user(self, userId, parkingId, bot, update):
    response = self.redisClient.process_detection_respose(userId, parkingId)
    if response:
      if response['success'] == True and response['foundPlaces'] == True:
          self.show_available_parking_place(response, bot, update)
      else:
          update.message.reply_text("Наразі на даній парковці вільних місць НЕ ЗНАЙДЕНО!")
          update.message.reply_text("Всього місць на парковці - " + str(response['allPlaces']))
    else:
        update.message.reply_text("Помилка обробки відповіді")

def show_available_parking_place(self, response, bot, update):
    imagePath = response['file']
    free_places = response['freePlaces']
    all_places = response['allPlaces']
    update.message.reply_text("Знайдено вільні паркувальні місця в кількості " + str(free_places))
    update.message.reply_text("Всього місць на парковці - " + str(all_places))

    bot.send_photo(
        chat_id=update.message.chat_id,
        photo=open(imagePath, "rb")
    )
```

У разі успішного відправлення повідомлення за допомогою Redis, скрипт починає прослуховування відповіді від скрипта розпізнавання у форматі JSON. Як тільки відповідь отримано, методи бота здійснюють її парсинг, та в залежності від результату відправляють відповідні повідомлення до користувача.

Для реалізації логіки розпізнавання вільних паркувальних місць в режимі реального часу, використовуючи відео потік з камери відео спостереження, було реалізовано наступні дії:

1. Здійснення підключення до IP камери відеоспостереження, використовуючи дані, занесені в систему через панель адміністратора.
2. Отримання початкового фото парковки, на якій заповнено автомобілями всі паркувальні місця. Дане фото повинне бути додано через панель адміністратора в момент додавання нової парковки.
3. Розпізнавання всіх авто на початковому фото за допомогою використання моделі «Mask R-CNN» в мові програмування Python, та отримання їх координат.
4. Отримання кадру з відео потоку.
5. Розпізнавання всіх авто на даному кадрі за допомогою використання моделі «Mask R-CNN» та отримання їх координат.
6. Знаходження значення IoU (Intersection Over Union) (рис. 2.8) між авто на початковому фото та авто на даному фото. Таким чином і визначається доступність паркувального місця. Якщо дане значення менше за деяке стале значення (припустимо 0,15), то дане паркувальне місце вважається вільним.
7. Вільні місця обводяться зеленими рамками, зайняті – червоними.

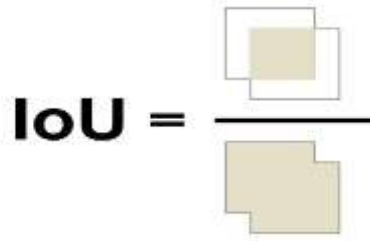

Рис. 2.2. Intersection Over Union

Нижче наведено код визначення доступності паркувального місця шляхом підраховування значення IoU для кожного зі знайдених автомобілів на фото:

```
overlaps = mrcnn.utils.compute_overlaps(parked_car_boxes, car_boxes)
```

```
# Currently we have not free places
        free_space = False
        all_parking_places = 0
        free_parking_places = 0
        for parking_area, overlap_areas in zip(parked_car_boxes, overlaps):

            # Get max IoU value
            max_IoU_overlap = np.max(overlap_areas)

            y1, x1, y2, x2 = parking_area
            all_parking_places += 1

            # Check availability of parking place by IoU value
            if max_IoU_overlap < 0.15:
                # Place is free. Draw green rectangle
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 1)
                free_parking_places += 1
                free_space = True
            else:
                # Place is not free. Draw red rectangle
                cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 1)
```

Висновки до розділу 2

Система розпізнавання вільних місць для паркування авто реалізує функціонал розпізнавання авто в реальному часі при використанні відео з IP камери відеоспостереження, можливість задавати дату та час для автоматичного розпізнавання, можливість переглядати список доступних парковок в чат-боті, можливість керувати інформацією за допомогою панелі адміністратора. Даний функціонал забезпечується використанням 2-ох мов програмування: Python та PHP. В частині системи, яка реалізована за допомогою PHP, використовується архітектурний шаблон MVC.

Було розглянуто та описано алгоритми та основні методи роботи системи розпізнавання вільних місць для паркування авто.

РОЗДІЛ 3. ІНТЕРФЕЙС ТА ПОРЯДОК РОБОТИ З СИСТЕМОЮ РОЗПІЗНАВАННЯ ВІЛЬНИХ МІСЦЬ ДЛЯ ПАРКУВАННЯ АВТО

3.1. Порядок встановлення та налаштування системи

Система розпізнавання вільних місць для паркування авто реалізована за допомогою мов програмування PHP та Python. Дана система використовує веб-сервер Apache2 та працює піж керівництвом ОС Linux. Програмний продукт використовує MySQL базу даних, та NoSQL сховище даних Redis.

Для встановлення та налаштування системи необхідно покроково виконати наступні дії:

1. Встановлення та налаштування ОС Linux версії 16.04 або вище на сервер.
2. Встановлення та налаштування веб-сервера Apache2.
3. Встановлення на сервері інтерпретаторів мов програмування PHP 7.0 та Python 3.6.
4. Встановлення та налаштування СУБД MySQL та додатку phpMyAdmin (для зручності керування базою даних).
5. Встановлення та налаштування NoSQL сховища даних Redis.
6. Створення бази даних «free_parking_detection_database» та імпорт даних в неї, застосовуючи файл-бекап (створений раніше).
7. Розміщення файлів системи на сервері, застосовуючи FTP клієнт.
8. Налаштування консольних команд (аліасів) для запуску системи.

Основні параметри доступу до бази даних наведені в файлі DatabaseCredentials.php.

З метою забезпечення можливості дистанційної роботи і незалежності від операційної системи користувача, даний програмний комплекс потребує наявності сервера, комп’ютера користувача-адміністратора та комп’ютера або смартфону користувача зі встановленим месенджером Telegram.

Дана система працює під керуванням Unix-подібних операційних систем встановлених на сервері. З точки зору користувача, система може працювати на всіх розповсюджених ОС, а саме: Windows, Linux, Android, iOS. Для використання системи необхідно мати встановлений додаток Telegram або використовувати веб-версію у будь-якому веб-браузері.

Зважаючи на те, що система для своєї роботи потребує значних апаратних ресурсів від сервера, його конфігурація повинна містити сучасні та потужні складові.

Наприклад в системі Linux Ubuntu для роботи програми потрібно встановити наступну конфігурацію на сервері:

64-розрядний (x64) чотириядерний процесор з тактовою частотою 3 ГГц або вище.

8 ГБ ОЗУ або вище.

50 ГБ вільного місця на жорсткому диску.

6 ГБ відеопам’яті.

Під час роботи система використовує значні ресурси процесора, відеокарти та оперативної пам’яті. Насамперед це потрібно для виконання «тяжких» методів для пошуку об’єктів на зображенні. Зважаючи на це і було вирішено встановлювати систему на сервері, що має описані апаратні ресурси.

3.2. Інтерфейс та порядок роботи з системою розпізнавання вільних місць для паркування авто

Розглянемо функціонал Telegram чат-бота. Для того, щоб знайти бот, необхідно в пошуку месенджера Telegram написати його назву

Рис. 3.1. Пошук бота

Після знаходження бота, необхідно його запустити, натиснувши кнопку «Start». Після цього користувача буде додано до бази даних системи. Відповідне повідомлення буде відправлено від імені чат-бота.

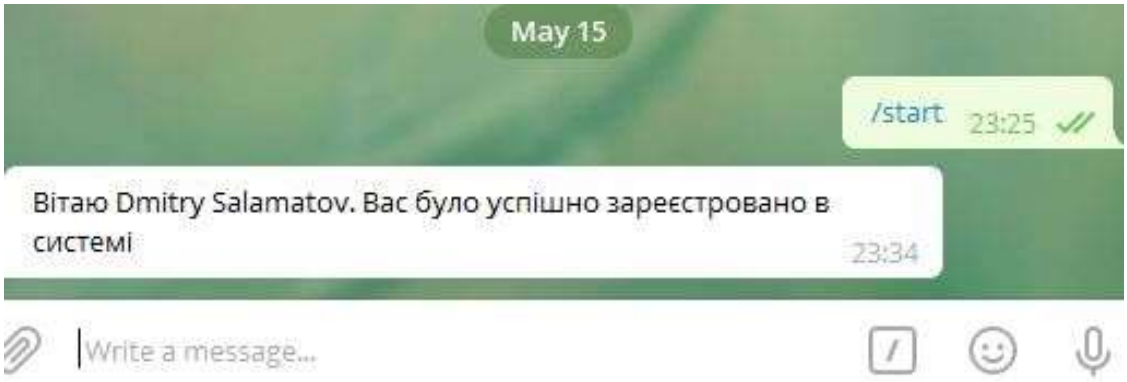

Рис. 3.2. Реєстрація в системі

Щоб переглянути інформацію про чат-бот, було реалізовано команду «/info». Виконавши цю команду, користувач зможе побачити коротку інформацію про бот та зорієнтуватися в подальших діях з ним.

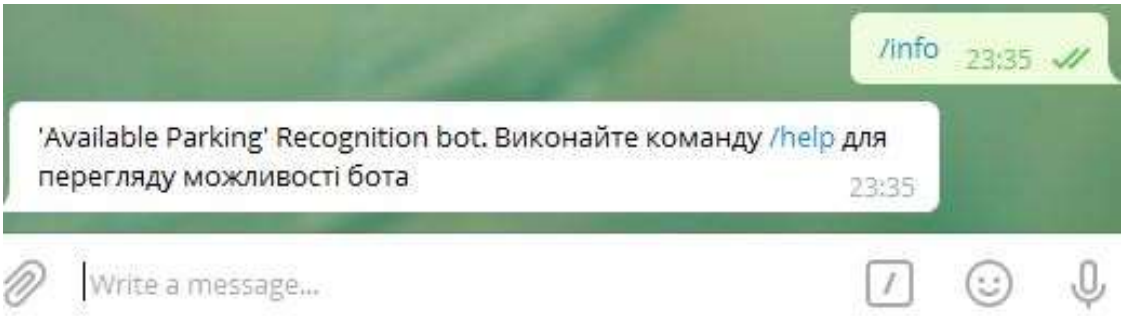

Рис. 3.3. Можливості команди /info

Для перегляду можливості чат-бота, необхідно виконати команду «/help». Використавши дану команду, користувач зможе побачити перелік всіх команд та функціоналу Telegram бота.

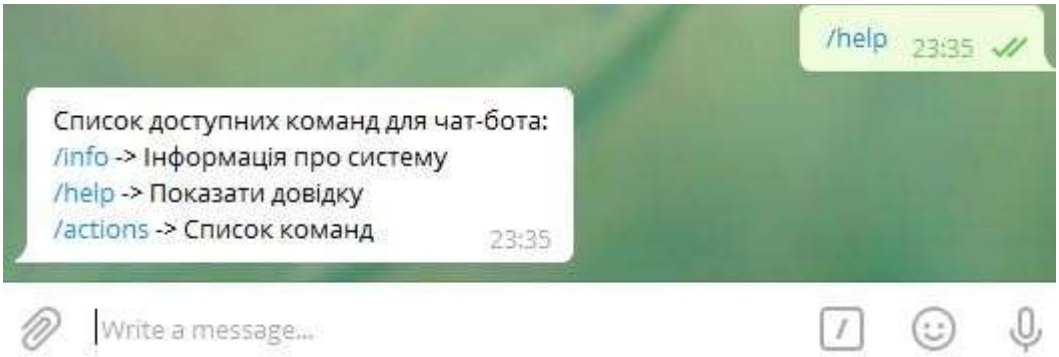

Рис. 3.4. Команда /info

Найважливіша команда даного чат-бота – це «/actions». За допомогою цієї команди користувач може відкрити кнопкове меню, в якому буде відображено основний функціонал системи.

Рис. 3.5. Команда /actions

Перед користувачем з’являється 3 кнопки, в яких зосереджено основні дії, які можуть бути здійснені в даній системі в ролі користувача.

Кнопка «Список парковок» слугує для відображення інформації про доступні парковки в системі. Ця інформація може бути корисною для користувачів, хто знає адресу або назву локації, але не впевнений чи працює дана система на цій локації.

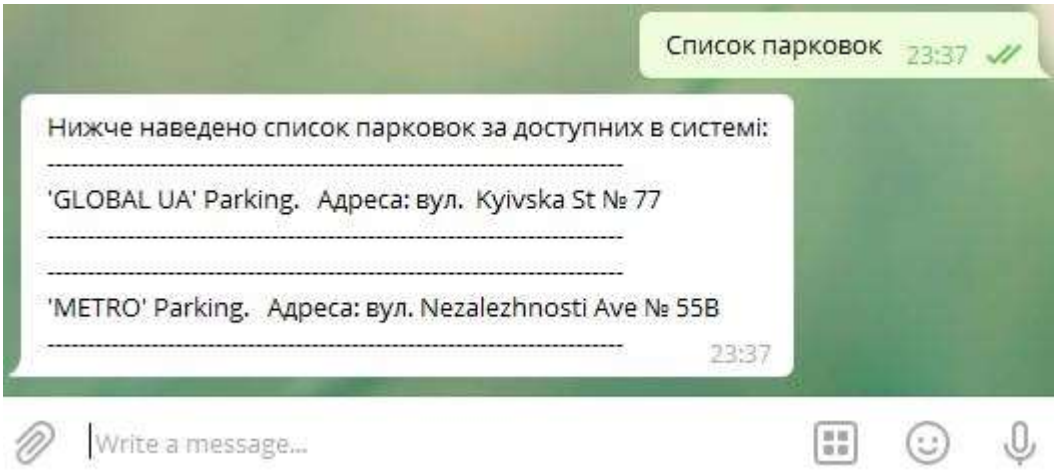

Рис. 3.6. Список парковок

Кнопка «Налаштувати автоматичне розпізнавання» потрібна для налаштування розпізнавання в заданий час для заданої локації. Це може бути особливо актуально для користувачів, які планують свої поїздки заздалегідь. Після натискання цієї кнопки, користувачу необхідно буде ввести дату та час як вказано в повідомленні. Якщо дата та час вказані вірно, користувачу буде запропоновано обрати парковку для здійснення автоматичного розпізнавання у вказаний ним час.

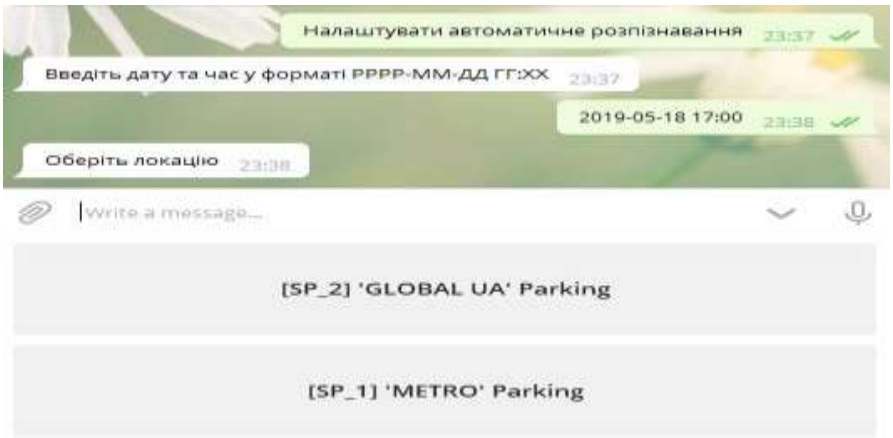

Рис. 3.7. Налаштування автоматичного розпізнавання

Після того, як користувач обрав одну із запропонованих йому локацій для розпізнавання, перед ним з’являється повідомлення про успішне налаштування даної функції Telegram чат-бота.

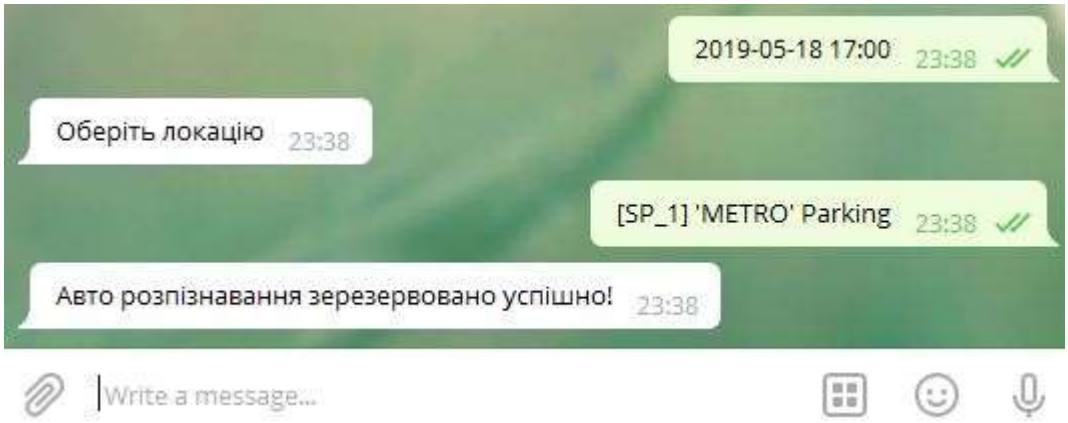

Рис. 3.8. Повідомлення успішного налаштування авто розпізнавання

Кнопка «Перевірити наявність вільних місць» - кнопка, за допомогою якої користувач може впевнитись, чи є на потрібній йому парковці вільні місця для паркування авто в даний час. Натиснувши її, користувачу необхідно обрати локацію зі списку запропонованих.

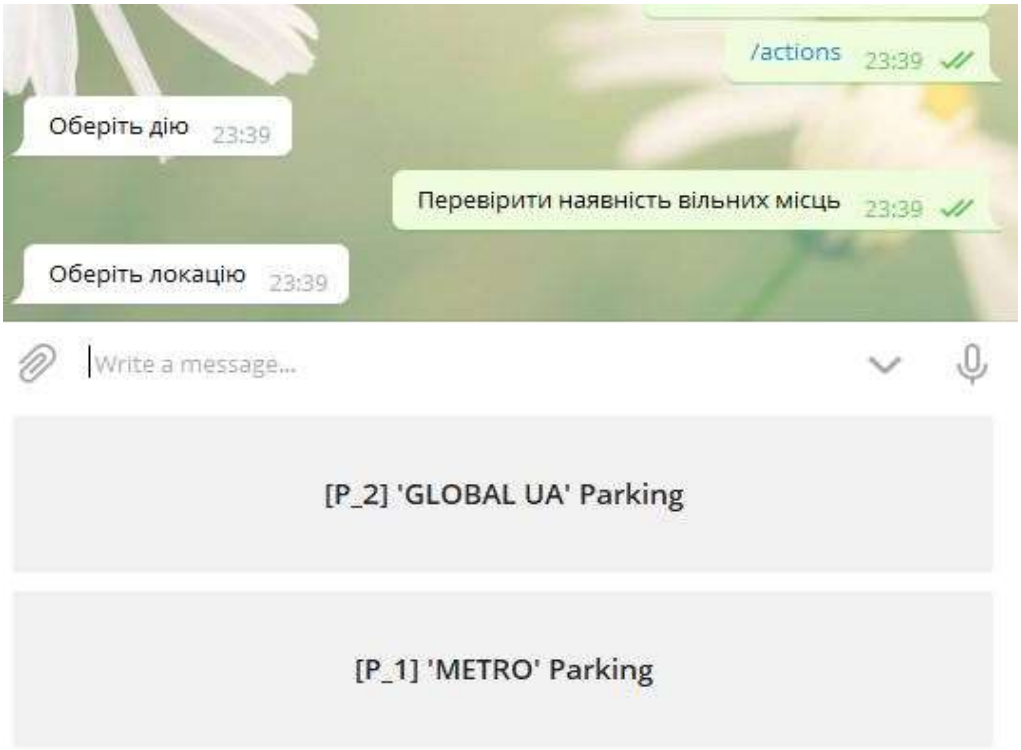

Рис. 3.9. Перевірка наявності вільних місць

Після того як користувач вибере одну із локацій, ботом буде відправлено запит на розпізнавання до скрипта, якій відповідає за розпізнавання вільних паркувальних місць на даній парковці.

У разі, якщо вільних місць на парковці не знайдено, буде виведено повідомлення з даною інформацією. Також користувачу буде показано кількість місць на даній парковці.

Рис. 3.10. Вільних місць не знайдено

Якщо ж скриптом було розпізнано хоча б одне вільне паркувальне місце, буде виведено відповідне повідомлення. Також користувачу буде показана інформація про кількість вільних місць, що дасть змогу користувачу детальніше проаналізувати ситуацію та впевнитись, що він зможе знайти вільне місце в найближчому майбутньому. До повідомлення про успішно знайдене паркувальне місце прикріплюється фото даного місця, яке дасть змогу користувачу швидко його на великих парковках.

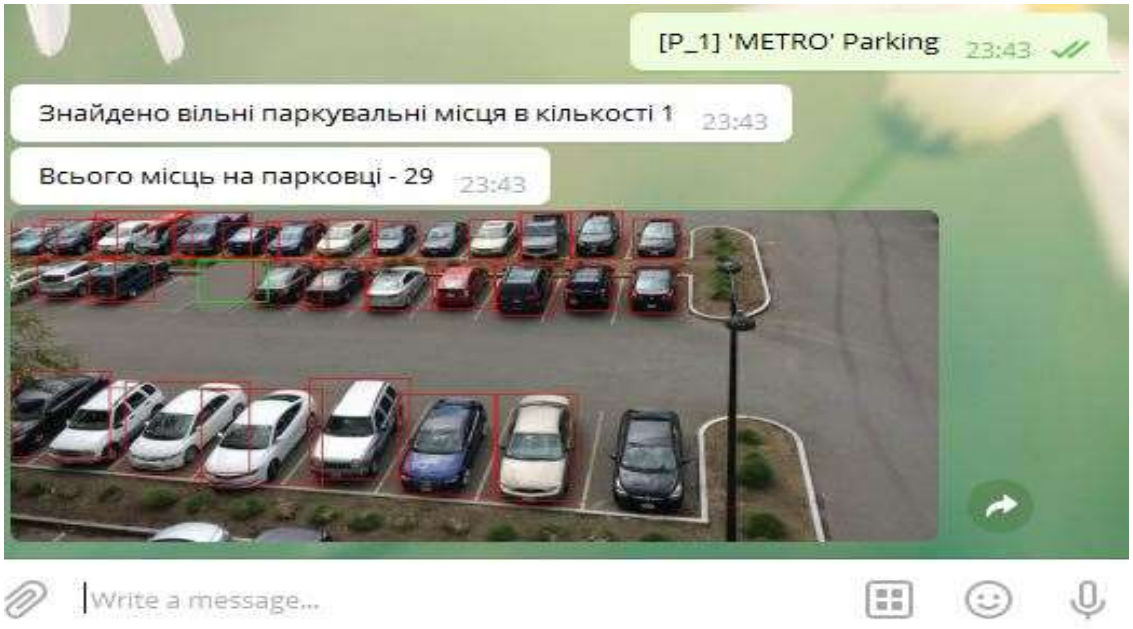

Рис. 3.11. Знайдено вільне місце

Висновки до розділу 3

В даному розділі було детально описано етапи встановлення та запуску програмної системи на сервері. Також було обґрунтовано вибір апаратних та програмних ресурсів для роботи системи як з точки зору адміністраторів системи, так і з точки зору кінцевих користувачів.

Реалізований програмний продукт включає в себе інтерфейси адміністратора та користувача. Для адміністраторів система наявна у вигляді веб-сайту для керування основною інформацією, такою як: локації, камери, користувачі чат-бота. З точки зору користувача, система являється чат-ботом в месенджері Telegram. Даний чат-бот обладнаний необхідними функціональними можливостями, які допомагають користувачу вирішувати проблеми пошуку вільного паркувального місця.

ВИСНОВКИ

Під час виконання роботи було спроектовано та реалізовано систему розпізнавання вільних місць для паркування авто. Для цього було виконано такі завдання:

1. Досліджено аналогічні системи ParkEyes, Integra-Parking, Aces Smart Parking та визначено, що майже всі програми мають стандартний набір функцій, які є досить дорогими в установці та обслуговуванні, що є значним недоліком цих систем. В даних системах відсутні робота в месенджері Telegram та можливість налаштування розпізнавання в заданій локації в заданий час.

2. Для реалізації системи було застосовано наступні інструменти: мову розмітки HTML5; таблиці стилів CSS3; браузерну мову програмування JavaScript; мову програмування PHP; мову програмування Python; систему управління базами даних MySQL; NoSQL сховище даних Redis; середовища розробки PhpStorm 2017.2 та PyCharm 2017.3.

3. Розроблено та реалізовано основні алгоритми роботи всіх складових системи, а саме: панелі адміністратора, чат-бота, скрипта розпізнавання.

4. Інтерфейс реалізованої системи складається із інтерфейсу панелі адміністратора та інтерфейсу чат-бота. Для адміністраторів система наявна у вигляді веб-сайту для керування основною інформацією, такою як: локації, камери, користувачі чат-бота. Для звичайних користувачів система являється чат-ботом в месенджері Telegram. Даний чат-бот є зручним та зрозумілим в користуванні. Також бот обладнаний необхідними функціональними можливостями, які допомагають користувачу вирішувати проблеми пошуку вільного паркувального місця.

5. Проведено тестування системи розпізнавання вільних місць для паркування авто. Було проведено наступні типи тестування: функціональне тестування, стресове тестування та тестування безпеки. Всі помилки та вразливості було виправлено.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Васильєв О. Програмування мовою Python / Олексій Васильєв. – Київ: Навчальна книга - Богдан, 2019. – 504 с.
2. Васильев А. Python на примерах / Алексей Васильев. – Киев: Наука и Техника, 2019. – 432 с. – (3).
3. Грофф Дж. SQL: Полное руководство / Грофф Дж., Вайнберг П.; пер. с англ. –[2-е изд., перероб. и доп.] -К.: Издательская группа BHV, 2001. - 816 с, ил.
4. Раджпут Д. Spring. Все паттерны проектирования / Динеш Раджпут. – Санкт-Петербург: Питер, 2019. – 320 с.
5. Симдянов И. PHP 7 / И. Симдянов, Д. Котеров. – Санкт-Петербург: БХВ-Петербург, 2016. – 1088 с.
6. Шварц Б. MySQL по максимуму / Б. Шварц, П. Зайцев, В. Ткаченко. – Санкт-Петербург: Питер, 2016. – 864 с.
7. Deep learning based Object Detection and Instance Segmentation using Mask R-CNN in OpenCV [Електронний ресурс] – Режим доступу до ресурсу: https://www.learnopencv.com/deep-learning-based-object-detection-and-instance-segmentation-using-mask-r-cnn-in-opencv-python-c/.
8. MVC — модель-представление-контроллер [Електронний ресурс] – Режим доступу до ресурсу: https://web-creator.ru/articles/mvc.
9. OpenCV Python [Електронний ресурс] – Режим доступу до ресурсу: https://pypi.org/project/opencv-python/.
10. PHP Data Objects [Електронний ресурс] – Режим доступу до ресурсу: https://www.php.net/manual/en/book.pdo.php.
11. Redis Pub/Sub [Електронний ресурс] – Режим доступу до ресурсу: https://redis.io/topics/pubsub.
12. Telegram Bot API Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://core.telegram.org/bots/api.

ДОДАТКИ

Додаток А

Лістинг коду чат-бота

```
from telegram.ext import Updater, CommandHandler, MessageHandler, Filters, JobQueue
from mysqlClient import MysqlClient
from redisClient import RedisClient
from datetime import datetime
from threading import Timer
import constants
import telegram
import inspect
import re


class TelegramBot:

    def __init__(self):
        self.updater = Updater(constants.BOT_UPDATER_TOKEN)
        self.job_queue = JobQueue(self.updater.bot)
        self.mysqlClient = MysqlClient()
        self.redisClient = RedisClient()
        self.detection_scheduling = {}

    def initCommands(self):
        methods = inspect.getmembers(self, predicate=inspect.ismethod)
        for method in methods:
            if (method[0][0:8] == "command_"):
                self.updater.dispatcher.add_handler(CommandHandler(method[0][8:],
method[1]))
        self.updater.dispatcher.add_handler(MessageHandler(Filters.text,
self.actions_control))

    def command_start(self, bot, update):
        userId = update.message.from_user.id
        userFirstName = update.message.from_user.first_name or ""
        userLastName = update.message.from_user.last_name or ""

        self.mysqlClient.register_new_telegram_user(userId, userFirstName + " " +
userLastName)
```

```
            update.message.reply_text(
                'Вітаю {}'.format(userFirstName + " " + userLastName) +
                ". Вас було успішно зареєстровано в системі")

        def command_info(self, bot, update):
            update.message.reply_text(
                "'Available Parking' Recognition bot. Виконайте команду /help для
перегляду можливості бота"
            )

        def command_help(self, bot, update):
            commands = [
                "/info -> Інформація про систему",
                "/help -> Показати довідку",
                "/actions -> Список команд"
            ]
            commandsString = ""
            for com in commands:
                commandsString += com + "\n"
            update.message.reply_text(
                "Список доступних команд для чат-бота:" + "\n" + commandsString
                )

        def command_actions(self, bot, update):
            keyboard = [
                [constants.PARKING_LOCATION_LIST_ACTION],
                [constants.PARKING_SELECT_LOCATION_FOR_DETECTING],
                [constants.SCHEDULE_DETECTION],
            ]
            reply_markup = telegram.ReplyKeyboardMarkup(
                keyboard,
                one_time_keyboard=True
            )
            bot.send_message(
                chat_id=update.message.chat_id,
                text="Оберіть дію",
                reply_markup=reply_markup
            )
```

```
        def actions_control(self, bot, update):
            action_text = update.message.text
            if self.is_string_datetime(action_text):
                success = self.date_time_control(bot, update, action_text)
                if success:
                    self.show_parking_locations_for_detection(bot, update, True)
            if action_text == constants.PARKING_LOCATION_LIST_ACTION:
                self.show_parking_list(bot, update)
            elif action_text == constants.SCHEDULE_DETECTION:
                self.schedule_detection(bot, update)
            elif action_text == constants.PARKING_SELECT_LOCATION_FOR_DETECTING:
                self.show_parking_locations_for_detection(bot, update)
            elif action_text[0:3] == "[P_":
                self.check_available_parking_places(action_text, bot, update)
            elif action_text[0:4] == "[SP_":
                self.generate_schedule(action_text, bot, update)

        def show_parking_list(self, bot, update):
            locations = self.mysqlClient.select_all_parking_locations()
            if not locations:
                return False
            list = "Нижче наведено список парковок за доступних в системі: \n"
            for l in locations:
                list += "-----------------------------------------------------------------
--------- \n"
                list  +=  l[constants.LOCATION_NAME]  +  ".       Адреса:  вул.  "  +
l[constants.LOCATION_STREET] + " № " + l[constants.LOCATION_HOUSE_NUM] + "\n"
                list += "-----------------------------------------------------------------
--------- \n"


            bot.send_message(
                chat_id=update.message.chat_id,
                text=list
            )

        def schedule_detection(self, bot, update):
            bot.send_message(
                chat_id=update.message.chat_id,
```

```
                text="Введіть дату та час у форматі РРРР-ММ-ДД ГГ:ХХ"
            )


        def auto_detection(self, bot, update):
            detection_scheduling = self.mysqlClient.select_auto_detection_scheduling()
            if detection_scheduling:
                detection_obj = []
                for d in detection_scheduling:
                    detect_after                                                    =
self.generate_detect_after(d[constants.SCHEDULING_DATE_TIME])
                    user_id = d[constants.SCHEDULING_USER_ID]
                    parking_id = d[constants.SCHEDULING_PARKING_ID]
                    detection_obj.append(
                        Timer(
                            detect_after,
                            lambda: self.check_available_parking_places("", bot, update,
user_id, parking_id)
                        )
                    )

                for d_obj in detection_obj:
                    d_obj.start()


        def date_time_control(self, bot, update, date_time_string):
            if self.is_datetime_valid(date_time_string):
                self.detection_scheduling[update.message.from_user.id]                  =
date_time_string
                return True
            else:
                bot.send_message(
                    chat_id=update.message.chat_id,
                    text="Введено невалідне занчення дати та часу"
                )
                return False

        def is_datetime_valid(self, date_time_string):
            try:
```

```
                date_time = datetime.strptime(date_time_string, "%Y-%m-%d %H:%M")
                if date_time <= datetime.now():
                    return False

                return True
            except:
                return False

        def is_string_datetime(self, date_time_string):
            try:
                date_time = datetime.strptime(date_time_string, "%Y-%m-%d %H:%M")
                return True
            except:
                return False

        def generate_detect_after(self, dtime):
            return (datetime.now() - dtime).total_seconds()

        def show_parking_locations_for_detection(self, bot, update, scheduling=False):
            prefix = "[P_"
            if scheduling:
                prefix = "[SP_"

            locations = self.mysqlClient.select_all_parking_locations()
            if not locations:
                return False
            buttons = []
            for l in locations:
                buttons.append([prefix + str(l[constants.LOCATION_ID]) + "] " +
l[constants.LOCATION_NAME]])

            reply_markup = telegram.ReplyKeyboardMarkup(buttons)
            bot.send_message(
                chat_id=update.message.chat_id,
                text="Оберіть локацію",
                reply_markup=reply_markup
            )
```

```
    def check_available_parking_places(self, parking_name, bot, update, userId =
None, parking_id = None, schedule_id = None):
        if schedule_id is not None:
            self.mysqlClient.delete_schedule_by_id(schedule_id)
        if userId is None and parking_id is None:
            parking_id = self.get_parking_id_by_name(parking_name)
            userId = update.message.from_user.id
        isSuccess = self.redisClient.send_recognition_request({
            'userId': userId,
            'parkingId': parking_id
        })
        if isSuccess:
            self.show_result_to_user(userId, parking_id, bot, update)
        else:
            update.message.reply_text("Помилка відправлення запиту")

    def generate_schedule(self, parking_name, bot, update):
        try:
            date_time = self.detection_scheduling[update.message.from_user.id]
            parking_id = self.get_parking_id_by_name(parking_name, True)
            self.mysqlClient.insert_to_scheduling(
                update.message.from_user.id,
                parking_id,
                date_time
            )
            update.message.reply_text("Авто розпізнавання зарезервовано успішно!")
        except:
            update.message.reply_text("Поимлка!")

    def show_result_to_user(self, userId, parkingId, bot, update):
        response = self.redisClient.process_detection_respose(userId, parkingId)
        if response:
           if response['success'] == True and response['foundPlaces'] == True:
                self.show_available_parking_place(response, bot, update)
           else:
                update.message.reply_text("Наразі на даній парковці вільних місць НЕ
ЗНАЙДЕНО!")
                update.message.reply_text("Всього місць на парковці - " +
str(response['allPlaces']))
```

```
            else:
                update.message.reply_text("Помилка обробки відповіді")

        def show_available_parking_place(self, response, bot, update):
            imagePath = response['file']
            free_places = response['freePlaces']
            all_places = response['allPlaces']
            update.message.reply_text("Знайдено вільні паркувальні місця в кількості " +
str(free_places))
            update.message.reply_text("Всього місць на парковці - " + str(all_places))

            bot.send_photo(
                chat_id=update.message.chat_id,
                photo=open(imagePath, "rb")
            )

        def get_parking_id_by_name(self, name, scheduling=False):
            parking_id = False
            first_symbols = 3
            if scheduling:
                first_symbols = 4
                founded_expr = re.search(r'\[SP_\d+\]', name).group()
            else:
                founded_expr = re.search(r'\[P_\d+\]', name).group()
            if founded_expr:
                parking_id = founded_expr[first_symbols:]
                parking_id = parking_id[:-1]

            return parking_id

        def run(self):
            self.initCommands()
            self.updater.start_polling()
            self.updater.idle()


    telegramBot = TelegramBot()
    telegramBot.run()
```

ДОДАТОК Б

Лістинг коду скрипта розпізнавання

```
import numpy as np
import cv2
import mrcnn.utils
from mrcnn.model import MaskRCNN
from mrcnn.config import Config
from pathlib import Path
from redisClient import RedisClient
from mysqlClient import MysqlClient
import constants


# Mask-RCNN config declaration
class MaskRCNNConfig(Config):
    NAME = "coco_pretrained_model_config"
    IMAGES_PER_GPU = 1
    GPU_COUNT = 1
    NUM_CLASSES = 1 + 80
    DETECTION_MIN_CONFIDENCE = 0.6
    VALIDATION_STEPS = 5
    STEPS_PER_EPOCH = 50


class FreeParkingDetector:

    def __init__(self):
        self.mysqlClient = MysqlClient()
        self.redisClient = RedisClient()
        self.PARKING_ID = 1
        self.ROOT_DIR = Path(".")
        self.MODEL_DIR = self.ROOT_DIR / "logs"
        self.COCO_MODEL_PATH = self.ROOT_DIR /
"coco20190321T1800/mask_rcnn_coco_0001.h5"
        self.IMAGE_SOURCE = None
        self.VIDEO_SOURCE = self.get_camera_path()
        self.SAVED_IMAGES_DIR = "savedImages"
```

```
        def get_camera_path(self):
            path = 0
            configuration = self.mysqlClient.select_parking_camera_configuration(str(self.PARKING_ID))
            if configuration:
                for c in configuration:
                    if c[constants.CAMERA_FIRST_IMAGE]:
                        self.IMAGE_SOURCE = c[constants.CAMERA_FIRST_IMAGE]
                    if c[constants.CAMERA_TEST_CONFIG]:
                        return c[constants.CAMERA_TEST_CONFIG]
                    path = "rtsp://" + c[constants.CAMERA_USERNAME] + ":" + c[constants.CAMERA_PASS] + "@" + c[constants.CAMERA_IP]
                    return path
            return path

        # Delete 'not cars' from found objects
        def getCarsObjects(self, objects, classes):
            car_boxes = []
            # 3 - car, 6 - bus, 8 - truck
            cars_classes = [3, 6, 8]

            for i, obj in enumerate(objects):
                if classes[i] in cars_classes:
                    car_boxes.append(obj)

            return np.array(car_boxes)

        def process_available_parking(self, info, frame, additional):
            userId = info['userId']
            parkingId = info['parkingId']
            if int(parkingId) == self.PARKING_ID:
                savedFileName = self.SAVED_IMAGES_DIR + "/usr" + str(userId) + "_p" + str(parkingId) + ".jpg"
                cv2.imwrite(savedFileName, frame)
                return {
                    'success': True,
                    'foundPlaces': True,
                    'allPlaces': additional['allPlaces'],
                    'freePlaces': additional['freePlaces'],
```

```
                    'file': savedFileName,
                    'userId': userId,
                    'parkingId': parkingId
                }
            return {
                'success': False
            }

        def process_unavailable_parking(self, info, frame, additional):
            userId = info['userId']
            parkingId = info['parkingId']
            if int(parkingId) == self.PARKING_ID:
                return {
                    'success': True,
                    'foundPlaces': False,
                    'allPlaces': additional['allPlaces'],
                    'freePlaces': additional['freePlaces'],
                    'userId': userId,
                    'parkingId': parkingId
                }
            return {
                'success': False
            }

        def run(self):
            # Create Mask-RCNN model.
            model       =       MaskRCNN(mode="inference",       model_dir=self.MODEL_DIR,
config=MaskRCNNConfig())

            # Load pre-trained model
            model.load_weights(self.COCO_MODEL_PATH.__str__(), by_name=True)

            parked_car_boxes = None

            video_capture = cv2.VideoCapture(self.VIDEO_SOURCE)
            first_image = cv2.imread(self.IMAGE_SOURCE)
            free_space_frames = 0
            results = []
            frames_counter = 0
```

```
check_requests_counter = 0


# Get single capture in loop.
while video_capture.isOpened():
    success, frame = video_capture.read()
    if parked_car_boxes is None:
        frame = first_image
    frames_counter += 1
    check_requests_counter += 1
    if not success:
        break

    # Convert img to RGB
    rgb_image = frame[:, :, ::-1]

    # Use Mask R-CNN detect method for getting results of distinguishing
    if frames_counter > 40 or not results:
        results = model.detect([rgb_image], verbose=0)
        frames_counter = 0
    r = results[0]

    if parked_car_boxes is None:
        # Get start parking condition. All parking place are unavailable
        parked_car_boxes = self.getCarsObjects(r['rois'], r['class_ids'])
    else:
        # Check free parking places

        # Get all cars on current capture.
        car_boxes = self.getCarsObjects(r['rois'], r['class_ids'])

        # Count overlaps between parked_cars condition and current condition
        overlaps = mrcnn.utils.compute_overlaps(parked_car_boxes, car_boxes)

        # Currently we have not free places
        free_space = False
        all_parking_places = 0
        free_parking_places = 0
        for parking_area, overlap_areas in zip(parked_car_boxes, overlaps):
```

```
        # Get max IoU value
        max_IoU_overlap = np.max(overlap_areas)

        y1, x1, y2, x2 = parking_area
        all_parking_places += 1

        # Check availability of parking place by IoU value
        if max_IoU_overlap < 0.15:
            # Place is free. Draw green rectangle
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 255, 0), 1)
            free_parking_places += 1
            free_space = True
        else:
            # Place is not free. Draw red rectangle
            cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 255), 1)

    additional_parking_info = {
        "allPlaces": all_parking_places,
        "freePlaces": free_parking_places
    }
    if free_space:
        free_space_frames += 1
    else:
        free_space_frames = 0

    if free_space_frames > constants.WAIT_FRAMES_COUNT:
        # Free parking place found.
        self.redisClient.process_bot_request(
            frame,
            additional_parking_info,
            self.process_available_parking
        )
    elif check_requests_counter > constants.WAIT_FRAMES_COUNT:
        self.redisClient.process_bot_request(
            frame,
            additional_parking_info,
            self.process_unavailable_parking
        )
```

```
                cv2.imshow('Video', frame)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                break

        video_capture.release()
        cv2.destroyAllWindows()


detector = FreeParkingDetector()
detector.run()
```