



# **Final Year Project**

## **Time Saving Parking**

**Arvids Ceceruks**

**G00351822**

**BEng (Honours) in Software & Electronic Engineering**

**Year 4**

**Galway-Mayo Institute of Technology**

**2019/2020**

## Project Poster



## Declaration

This project is presented in partial fulfilment of the requirements for the Honours degree of Bachelor of Engineering in Software & Electronic Engineering at Galway-Mayo Institute of Technology.

This project is my own work, except where otherwise accredited. Where the work of others has been used or incorporated during this project, this is acknowledged and referenced.

## Acknowledgements

I would like to acknowledge following people for contributions to my final year project.

Paul Lennon, Project supervisor at “GMIT” – for guiding me through development lifecycle of my project and suggesting tools that could be useful for project’s overall success.

Vladislavs Marisevs, Team Lead at “Visma” – for help in resolving the issue during development of persistence layer of the server program. Contributed to development of alternative JDBC template code that I am using in my server program.

## Table of Contents

1	Project overview .....	7
2	Project Architecture .....	8
3	System architecture and development tools .....	9
3.1	RESTful API Web server .....	9
3.2	Raspberry Pi 4 and Pi camera .....	9
3.3	Database .....	9
3.4	Android Mobile Application .....	9
3.5	Postman .....	9
3.6	NGROK .....	9
3.7	Maven .....	9
3.8	Balsamiq wireframes .....	9
4	DATABASE .....	10
5	RESTful API Web Server .....	12
5.1	REST API .....	12
5.2	SPRING BOOT .....	12
5.3	LOMBOK .....	13
5.4	Server architecture .....	13
5.5	Presentation Layer .....	14
5.6	Business Layer .....	15
5.7	Persistence Layer .....	16
5.8	Postman .....	16
5.9	NGROK TUNNEL .....	17
6	RASPBERRY PI 4 & Pi Camera .....	18
6.1	OBJECT RECOGNITION .....	18
6.1	OpenCV .....	18

6.1.2	YOLO.....	18
6.2	BASH SCRIPT .....	20
7	ANDROID MOBILE APPLICATION .....	21
7.1	Main activity.....	21
7.2	Welcome activity.....	23
7.3	View All locations activity.....	25
8	PROBLEM SOLVING .....	27
8.1	Server development issues and challenges .....	27
8.2	Mobile application development issues and challenges .....	27
8.3	Raspberry Pi/Object recognition issues and challenges .....	28
9	CONCLUSIONS .....	29
10	REFERENCES.....	30

## 1 Project overview

My final year project is “Time saving parking” – a parking slot finder system. I decided to develop such system because the number of cars and drivers on the roads is constantly increasing which is leading to heavy traffic delays, and heavier air pollution in cities and towns. Often It is very hard to know where the free parking slots are available in real time. In order, for drivers to park their car, driver must drive around city and look for one. This common activity is leading to more traffic congestion, pollution and wasted time.

The system that I developed is using computer vision, to implement this new way of parking. My project development consists of three main components – REST API Web Server, Raspberry Pi with camera and Android mobile application. I used Java programming language to develop my server. For object recognition I used Python programming language and YOLO (You Only Look Once) object recognition algorithm, and the application was developed in Android studio using Java.

In order to successfully complete the project – I have developed a project plan, allocated time for each of these main three project parts and did breakdown of work to be completed. For my completed work backup, version control and important documentation I used GitHub – code hosting platform.

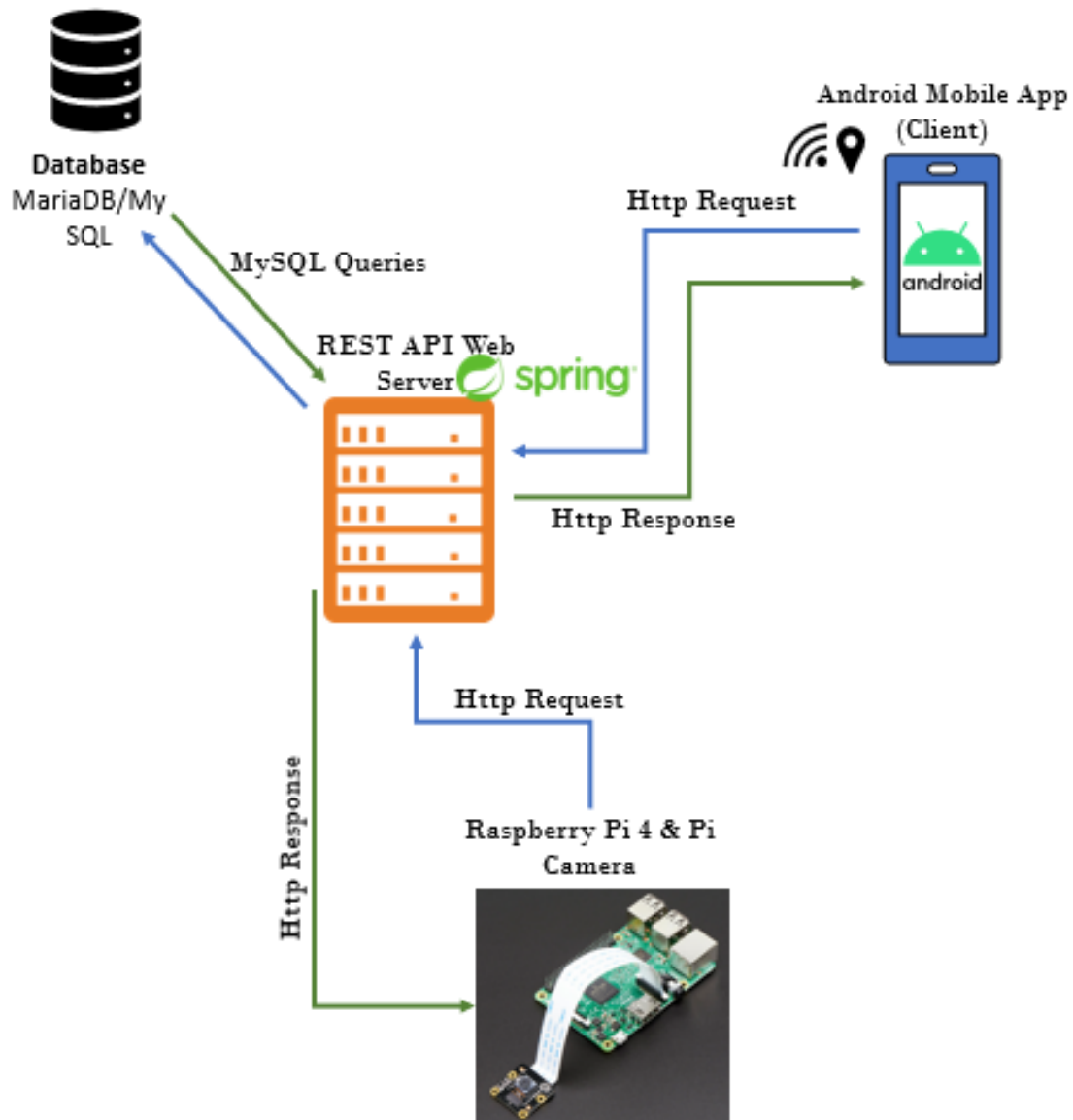
Final prototype of the system is completely autonomous and does not require any interaction from developer side once server is started and Raspberry Pi is executing a script.

In mobile application, all features are accessible where person can see information about carpark locations, map, request to find nearest parking slot and get driving directions to slot location using google maps navigation.

## 2 Project Architecture

The image of an architecture diagram in figure below, shows the interaction between devices and components of my developed system.

More detailed information on development tools used in my project is described in next section.





### 3 System architecture and development tools

#### 3.1 RESTful API Web server

The server which is developed in IntelliJ IDEA using Spring Boot framework, Java programming language and Embedded Tomcat, processes incoming HTTP requests from devices, and queries the database.

#### 3.2 Raspberry Pi 4 and Pi camera

Raspberry Pi 4 is latest generation mini PC that operates using Raspbian Buster OS.

Developed system is applying computer vision to detect cars or their absence where system have been implemented. Development was performed using ThonnyIDE, Python scripting language, OpenCV and YOLO object recognition algorithm

#### 3.3 Database

A MySQL database developed in HeidiSQL (an open source MySQL database administration tool). Architecture of it is designed as relational database and consists of set of tables.

#### 3.4 Android Mobile Application

The application is developed in Android Studio v3.6 using Java language is providing user friendly interface.

#### 3.5 Postman

A software that is designed to work with simple and complex HTTP requests, allowing to work with CRUD (Create Read Update Delete) requests.

#### 3.6 NGROK

A multiplatform tunnelling software that establishes secure tunnels from public endpoint to a locally running network service.

#### 3.7 Maven

The tool for building, testing and managing Java-based projects.

#### 3.8 Balsamiq wireframes

This software allows to visualise look of application before functional development begins. It is a great tool, that allows developers to visualise their idea.

## 4 DATABASE

A database that I am using for my project is a MySQL database. I have developed it in HeidiSQL (an open source MySQL database administration tool). Architecture of it is designed as relational model of data and consists of set of tables. Using relational database is very common in engineering industry and allows me to link tables and manipulate data in tables taking other tables into reference.

Some columns in the tables, like login details, phone numbers, email addresses and car registration numbers are assigned *unique keys*. Such keys are used to indicate that any entry in that column must be unique and can not be duplicated. For example, there can not be two different cars with same car registration number.

The database model is a set of 6 tables where I am storing necessary data that is required for operation of 'Time Saving Parking' system.

**Database tables**

car_park	240.0 KiB
car_park_location	16.0 KiB
car_park_location_slot	2.0 KiB
car_park_slot_booking	80.0 KiB
user_info	48.0 KiB
user_vehicle	48.0 KiB
vehicle	16.0 KiB

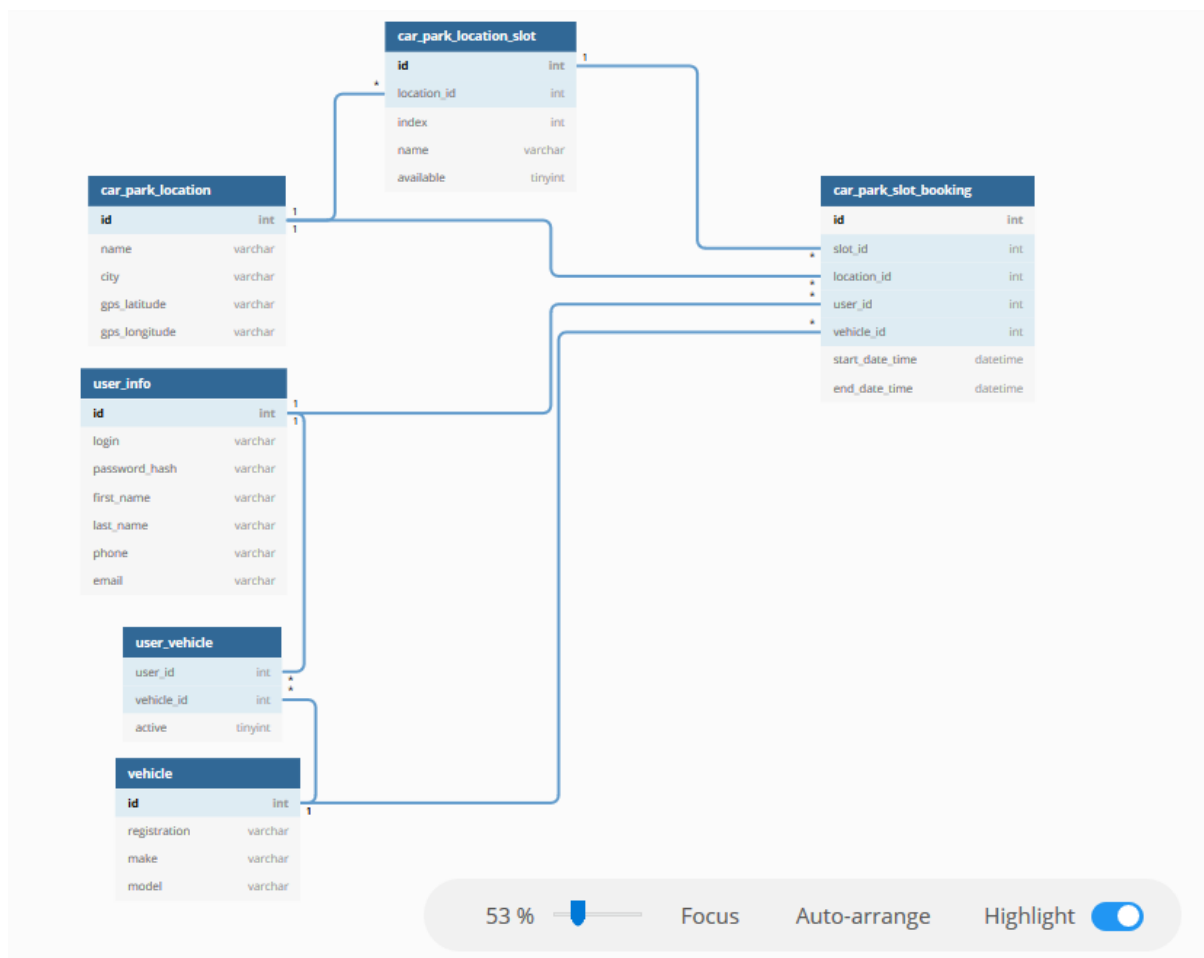
**Fig. 4-1** Database tables

- *car\_park\_location*: contains name of carpark, city where it is located, and GPS coordinates of each location.
- *car\_park\_location\_slot*: Tables primary function is to hold information about availability of each slot at every location.
- *car\_park\_slot\_booking*: contains information regarding parking slot booking. It is linked using 'One-to-many' relation.
- *user\_info*: contains information about users that are registered in the system. For demonstration purposes I have created three instances of registered users, names and other details of which are imaginary.
- *user\_vehicle*: The table is designed to hold information on vehicle ID's and user ID's. The ID's are linked as foreign keys to *user\_info* table, where ID is primary key.
- *vehicle*: contains information regarding vehicles of registered users. The vehicle details are imaginary and are for demonstration purposes only.

To show ER (entity relationship) and database structure I used 'dbdiagram.io', which is a tool for drawing ER diagrams.

As an example, I will discuss relation between *car\_park\_location* and *car\_park\_location\_slots* tables. The diagram below shows the link between these two tables, which is a 'One-to-many' relation. The *id* field in *car\_park\_location* table is a primary key and is linked with *location\_id* in *car\_park\_location\_slots* table. By creating this link, I am "telling" the database that one location can have many slots, and the slot can have only one specific location.

### Database structure



**Fig. 4-2** Database structure

## 5 RESTful API Web Server

After reviewing the architecture and performance of my first server, that I was developing at the beginning of the academic year, I researched more sources that were providing information on development of servers and tools that are used to assist me in this process. In my studies I was referring to list of popular web resources such as [tutorialspoint.com](https://www.tutorialspoint.com), [stackoverflow.com](https://stackoverflow.com) and [youtube.com](https://www.youtube.com), but this list is not exhaustive.

For development of my new server I used Apache Maven build tool, which is designed for building and managing Java-based projects, Spring framework and Spring-boot (part of a Spring framework), which is a very popular framework for development of web applications.

The server is using embedded Tomcat server, which “is an application server designed to execute Java servlets and render web pages that use Java Server page coding [4].”

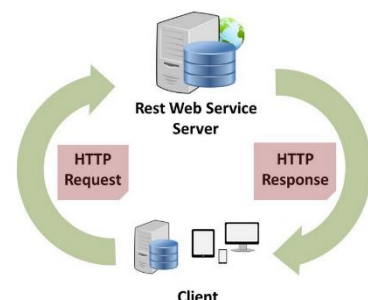
The RESTful server oversees tasks of processing HTTP requests coming from clients, querying the database and sending response to a client.

### 5.1 REST API

“**Representational State Transfer (REST)** is a software architectural style that defines a set of constraints to be used for creating web services. Web services that conform to the REST architectural style, called *RESTful* Web services, provide interoperability between computer systems on the Internet [6].”

In other words, the RESTful Architecture allows independent systems and devices to communicate over HTTP.

In my project I am using this RESTful architecture to establish communication between REST server (web service), android mobile app and Raspberry Pi (clients).



### 5.2 SPRING BOOT

A Spring-boot is a part of a Spring framework and is designed for developers of an applications to be able to generate a production-ready application and simplifies configuration process of all the basic configurations.

To generate a Spring-boot Maven project It only required me to visit [spring.io](https://spring.io) website and open a Spring initializer window, where I can configure initial project, such as naming the project and adding list of required dependencies. Once the Spring-boot project is generated It is ready for further development.

One very great feature of Spring-boot is that It has an embedded Apache Tomcat server. When an application is started, Spring-boot detects that application has Spring MVC controllers and starts up an embedded Tomcat server instance.

Spring-boot's embedded Apache Tomcat server is acting as a webserver and is listening for requests on *localhost* port *8080* by default, but this configuration can be changed in *Application.properties*.

### 5.3 LOMBOK

I am using Lombok library in my server program and in my mobile application. This is a great library that generates boilerplate code at runtime – no longer I need to generate getters or setters in my code, so the code becomes more efficient when executing, and is significantly more compact. In order to access features of this library, the library needs to be downloaded from official website <https://projectlombok.org/> or added to Maven project *pom.xml* file.

### 5.4 Server architecture

My server follows a 3-tier architecture, where all system layers are protected and isolated from each other. The server layers are: Presentation layer (Controller), Business layer(service) and Persistence layer (repository). Current design allows to manipulate individual parts of the microservice ensuring security and flexibility.

References in text:

DTO - Data Transfer Object. An object that is used to encapsulate data and send it from one layer of an application to another.

DAO – Data Access Object. An object that is used in persistence layer to access the repository.



**Fig.5-4-1** 3-Tier architecture. Access sequence

## 5.5 Presentation Layer

Presentation layer oversees tasks of validating requests coming from the client.

In the presentation layer I am creating public endpoints and providing a method of requests. These endpoints are part of the request URLs that server will be ready to process. Once the request is being sent to the microservice it is attempted to be processed.

It is often required to pass some parameters with the request. In the code snippet below (Fig.5-5) the endpoint is created for registering car at the specific location and slot, which is determined by slot ID. What also is required is the vehicle registration number to be included in request, in order to know vehicle details. This is achieved by using *@RequestBody* annotation which will take in the data transfer object *registerVehicleCarParkSlotDto*, which is a string e.g. 06LK1234.

```

14 @RestController
15 public class VehicleController {
16
17     @Autowired
18     private VehicleCarParkSlotService vehicleCarParkSlotService;
19
20     /**
21      * *****
22      * Vehicle is getting registered at car park
23      */
24     @RequestMapping(path = "/car-park-slot/{slotId}", method = RequestMethod.POST)
25     @ResponseStatus(code = HttpStatus.OK)
26     @PostMapping
27     public void registerVehicleCarParkSlot(@PathVariable Integer slotId, @RequestBody RegisterVehicleCarParkSlotDto registerVehicleCarParkSlotDto) {
28         vehicleCarParkSlotService.registerVehicleCarParkSlot(slotId, registerVehicleCarParkSlotDto.getVehicleRegistration());
29     }

```

**Fig. 5-5** Vehicle controller endpoint

By using annotation *@RestController*, Spring-boot is adding controller to the pool of controllers within a Spring framework. In order to wire(link) controller described above to the business layer I am using *@Autowired* annotation. In this case it is linked to business layer's *VehicleCarParkSlotService* class.

For easy management of code, I have separated controllers in two classes, where one class holds endpoints associated with locations and another is associated with vehicles. The locations controller has endpoints that are returning data and are GET method requests (Fig.5-5-1). Vehicle controller endpoints are PUT and POST method requests and are designed for updating and inserting data into database (Fig.5-5).

```

34 /**
35  * Endpoint which returns detailed info about specific Location
36  * @param locationId is an ID which determines the request
37  */
38 @RequestMapping(path = "/car-park-locations/{locationId}", method = RequestMethod.GET)
39 public RichCarParkLocationDto getCarParkLocationById(@PathVariable Integer locationId) {
40     CarParkLocation carParkLocation = carParkLocationService.getCarParkLocation(locationId);
41     return RichCarParkLocationDto.from(carParkLocation);
42 }

```

**Fig.5-5-1** Location controller endpoint

## 5.6 Business Layer

Business layer is the core of the server. At this layer, service is performing data manipulation and logic calculations. By taking the data received from presentation layer it is performing logic and encapsulates data for access in next layer.

First, the classes in service layer are marked with `@Service` spring-boot annotation. By using this annotation, the spring-boot will add the class to the pool of services.

Second, the methods within a class must be annotated as `@Transactional`. By doing this, spring-boot creates a proxy, and “a proxy provides a way for Spring to inject behaviors before, after, or around method calls into the object being proxied.[3]”

The code snippet (Fig.5-6) is a method that is registering slot booking. It is responsible for accessing/passing the data/parameters *slotId*, *userId*, *vehicleId*, *locationId* and time of booking with the DAO, and update fields in the database.

The method creates three new entity objects and is creating them by accessing user and vehicle details from database using *vehicleDao*, *userVehicleDao* and *carParkLocationSlotDao*.

It is then performing the check by using an *if* statement which is simply checking the status of the slot availability. If the value returned is *false*, it means that slot is already booked by other user, therefore runtime exception will be thrown, and service will stop processing ongoing request.

If slot is not booked, the service creates new *carParkSlotBookingEntity* object and sets the new values. When new entity is created, method is passing it as a parameter in DAO. And after is updating slot availability.

```

39  @Transactional
40  public void registerVehicleCarParkSlot(Integer slotId, String vehicleRegistration) {
41      VehicleEntity vehicleEntity = vehicleDao.getVehicleByRegistration(vehicleRegistration);
42      List<UserVehicleEntity> users = userVehicleDao.getVehicleActiveUsers(vehicleEntity.getId());
43      CarParkLocationSlotEntity slot = carParkLocationSlotDao.getCarParkSlot(slotId);
44      //CarParkLocationEntity location = carParkLocationDao.getCarParkLocation(locationId);
45      if (Boolean.FALSE.equals(slot.getAvailable())) {
46          throw new RuntimeException("Car park slot is busy");
47      }
48
49      users.forEach(user -> {
50
51          CarParkSlotBookingEntity carParkSlotBookingEntity = new CarParkSlotBookingEntity();
52          carParkSlotBookingEntity.setSlotId(slot.getId());
53          carParkSlotBookingEntity.setLocationId(slot.getLocationId());
54          carParkSlotBookingEntity.setUserId(user.getUserId());
55          carParkSlotBookingEntity.setVehicleId(user.getVehicleId());
56          carParkSlotBookingEntity.setStart(LocalDate.now());
57
58          carParkLocationSlotBookingDao.insertCarParkBooking(carParkSlotBookingEntity);
59          carParkLocationSlotDao.updateAvailability(slot.getId(), available: false);
60      });
61  }

```

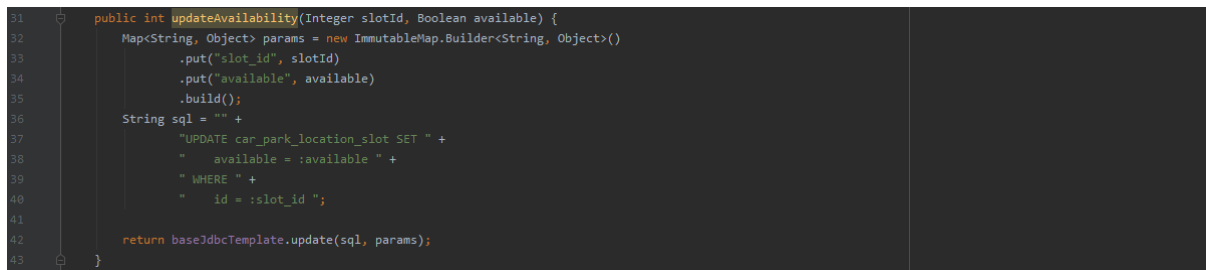
Fig. 5-6 Slot booking method

## 5.7 Persistence Layer

At persistence layer data is being accessed using data access objects.

In this layer I am accessing the data stored in MySQL database. To indicate that the layer is a data access layer, I am using Spring-boot annotation *@Repository*.

In order to be able to work directly with the database, the JDBC template must be used, which stands for Java Database Connectivity.



```

31 public int updateAvailability(Integer slotId, Boolean available) {
32     Map<String, Object> params = new ImmutableMap.Builder<String, Object>()
33         .put("slot_id", slotId)
34         .put("available", available)
35         .build();
36     String sql = "" +
37         "UPDATE car_park_location_slot SET " +
38         "    available = :available " +
39         " WHERE " +
40         "    id = :slot_id ";
41
42     return baseJdbcTemplate.update(sql, params);
43 }

```

**Fig. 5-7-1** SQL query

Having an *@Autowired BaseJdbcTemplate* provides automatic access to template's methods such as *insert*, *update* or *queryForObject*.

The code snippet above (Fig. 5-7-1) is a part of one of the data access classes. This public method is querying the database by passing two parameters *slotID* and *available* coming from the business layer to the SQL string. Values of these parameters are fully dependent on business layer's logic.

Another very important aspect of persistence layer is to provide the database models(entities). When data arrives back from database, it must be processed back to presentation layer, therefore it must be managed during this process. By creating an entity class, I am mapping it using entity class's objects.

## 5.8 Postman

In order to be able to work with HTTP requests and for development of my server I used 'Postman', which is a software that is designed to work with simple and complex HTTP requests, allowing to work with CRUD (Create Read Update Delete) method requests.

I used this software to test endpoints (URL's) created in server program and was monitoring response of database.

Some requests require additional data to be passed with request, therefore I could write data as JSON object and pass it with HTTP request.

Below is an example of my HTTP POST request to server (Fig.5-8-1). I am passing JSON object with key:vehicleRegistration and value:141D34123.



This request will register car (with passed vehicle registration number) in carpark slot and set registration time.

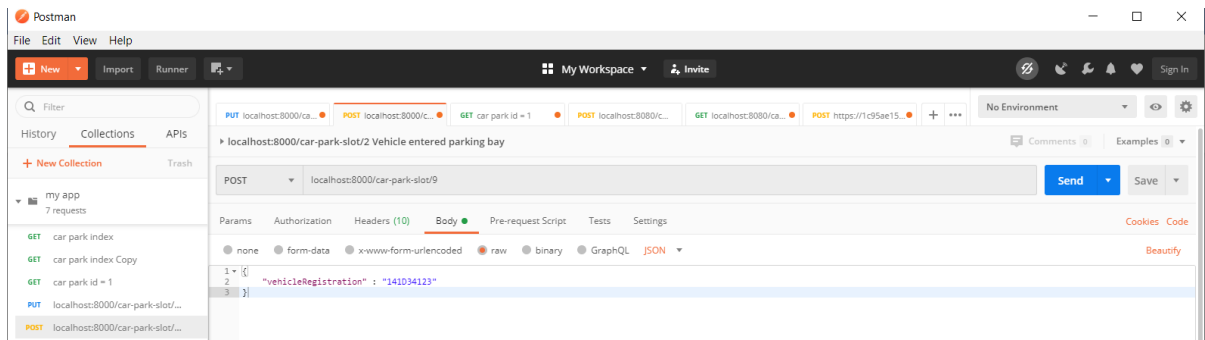


Fig. 5-8-1 Postman request

## 5.9 NGROK TUNNEL

An NGROK is a multiplatform tunnelling software that establishes secure tunnels from public endpoint to a locally running network service. I used ngrok service in order to provide tunnel for my server, so I can work on development of parts android mobile application where it requires connection with the server.

NGROK service is not requiring any installation. It works by downloading and running a program on local machine and providing a port of my server. “It connects to the ngrok cloud service which accepts traffic on a public address and relays that traffic through to the ngrok process running on my machine and then on to the local address specified [7].”

The command below is creating a tunnel to my local server which is run on port 8080



The http address is generated and can be used to access the server (Fig. 5-9-1)

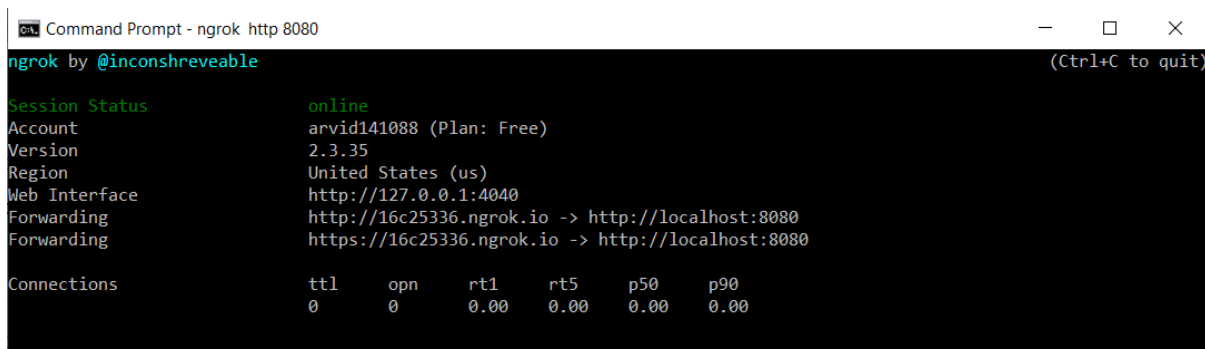


Fig.5-9-1 Ngrok secure tunnel

## 6 RASPBERRY PI 4 & Pi Camera

Developed system is applying computer vision to detect empty parking slots where system has been implemented. Raspberry Pi is taking images using camera, applying YOLO (You Only Look Once) algorithm and sends request containing information regarding current location to the Server. Execution of bash script oversees tasks of taking images using Pi Camera that is attached to mini PC, and execution of program code that is developed using Python scripting language.

To set up Raspberry Pi 4 for my project I needed to flash the SD card with the latest image of Raspbian OS. Raspbian OS is an official operating system that is specifically designed for Raspberry Pi mini PC's, although other Linux based operating systems can be installed. I used "Balena Etcher" utility tool in order to write image of OS to SD card.

### 6.1 OBJECT RECOGNITION

#### 6.1.1 OpenCV

"OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms [1]." The algorithms are used for various purposes such as object detection and recognition, color detection, face and eye recognition, image processing and correction and many more.

#### 6.1.2 YOLO

"YOLO (You Only Look Once) is a clever convolutional neural network (CNN) for doing object detection in real-time. The algorithm applies a single neural network to the full image, and then divides the image into regions and predicts bounding boxes and probabilities for each region [2]."

I decided to use YOLO algorithm for my project because It is designed for object recognition in real time and It works much faster comparing to other OR algorithms.

The code for object recognition, developed using Python programming language, is an open source and is available from [pyimagesearch.com](https://pypi.org/project/pyimagesearch/) [5]. I have made modifications to the code by including the communication between RESTful service and Raspberry Pi, and some logic calculations.

The program accesses image from specified location and processes it. Once image is processed, the program is saving instance of image to the machine. If any objects that are listed in the *config* file are recognized by the program, the program draws a box around them and setting the name at top of each box.

In my project I am only recognizing cars, therefore I have edited *config* file to have only car entries.

I have edited code to be able to interact with the server. For RESTful communication over HTTP I am using python 'requests' library. This library allows me to send request to the server and receive data (Fig.6-1-2).

```

4 import cv2
5 import os
6 import requests
7 import json
8 import subprocess
9
10 headers = {'Content-type': 'application/json', 'Accept': 'text/plain'}
11
12 #Endpoint request info about location (GET)
13 url2 = "https://1c95ae15.ngrok.io/car-park-locations/4"
14 r=requests.get(url2)
15 dataReceived=r.json()
16
17 #Total number of slots
18 slotsTotal=dataReceived['slots']
19 countTotal=len(slotsTotal)

```

Fig. 6-1-2

The request URL is returning a JSON object (Fig. 6-1-3) that contains detailed information regarding carpark at specific location. Python program is utilizing this information and extracting necessary data – I am interested in total number of slots at the carpark. Once Raspberry Pi received data from server, the code does the calculation of total slots. The calculation is needed because response JSON string contains a List within it. Therefore, I am separating the list on line 20,21(Fig.number) and getting an integer value *countTotal* which represents total number of slots.

Program then performs object recognition and is setting index values depending on how many objects were recognized. This index value is used later to see if at least one detection exists.

JSON	Raw Data	Headers
Save	Copy	Collapse All Expand All
id: 4		
name: "Center"		
city: "GALWAY"		
gpsLatitude: "53.274994"		
gpsLongitude: "-9.045973"		
▼ slots:		
▼ 0:		
id: 11		
index: 1		
available: true		
▼ 1:		
id: 12		
index: 2		
available: true		
▼ 2:		
id: 13		
index: 3		
available: true		

Fig.6-1-3

Further, the program is comparing if total number of slots is greater or equal than number of recognized objects. If it is , then program begins a 'for-loop' and sends POST requests to the server to register busy slots. The number of requests sent to the server is equal to number of recognized objects at a time (Fig. 6-1-4).

```

116
117 # ensure at least one detection exists
118 if len(idxs) > 0:
119     print("Cars recognized:")
120     finalCnt=countTotal-len(idxs)
121     if countTotal>len(idxs):
122         #print(len(idxs))
123         for count in range(finalCnt-1):
124             slotId = count+11
125             #Endpoint request to register busy slot (POST)
126             url1="https://1c95ae15.ngrok.io/car-park-slot-busy/"+str(slotId)
127             #Sending POST request
128             requests.post(url1)

```

Fig. 6-1-4

Regardless of how many objects were recognized the program will call a subprocess – which is an execution of bash script (Fig.6-1-5).

```

145 cv2.waitKey(0)
146 subprocess.call('runPi.sh')
147 else:
148     print("Cars NOT recognized:")
149     #finalCnt=countTotal-len(idxs)
150     #if countTotal==len(idxs):
151     for count in range(countTotal):
152         slotId = count+1
153         #Endpoint request to register free slot (POST)
154         url="https://3e833257.ngrok.io/car-park-slot-free/"+str(slotId)
155         #Sending POST request
156         requests.post(url)
157         print("posting"+str(slotId))
158 cv2.waitKey(0)
159 subprocess.call('runPi.sh')

```

**Fig. 6-1-5**

## 6.2 BASH SCRIPT

Execution of bash script oversees tasks of taking images using Pi Camera that is attached to Raspberry Pi, and execution of python program. The script is starting the whole process of interaction between camera and python program. It begins with acquiring an image after 10 seconds have elapsed and saving it in local directory where image will be available for processing. It runs the python code which is applying an object detection algorithm to the image provided. Further, it will delete image to keep storage of device clean (Fig. 6-2-1).

```

1  #!/bin/bash
2
3  #Giving 10 seconds before taking picture
4  #oncar.jpg and nocar.jpg are images for demonstration purposes
5  raspistill -t 10000 -o /home/pi/yolo-object-detection/images/acquiredImage.jpg
6  #python detect_car.py --image images/oncar.jpg --yolo yolo-coco
7  #python detect_car.py --image images/nocar.jpg --yolo yolo-coco
8  python detect_car.py --image images/acquiredImage.jpg --yolo yolo-coco
9  rm /home/pi/yolo-object-detection/images/acquiredImage.jpg
10

```

**Fig. 6-2-1 Bash script**

## 7 ANDROID MOBILE APPLICATION

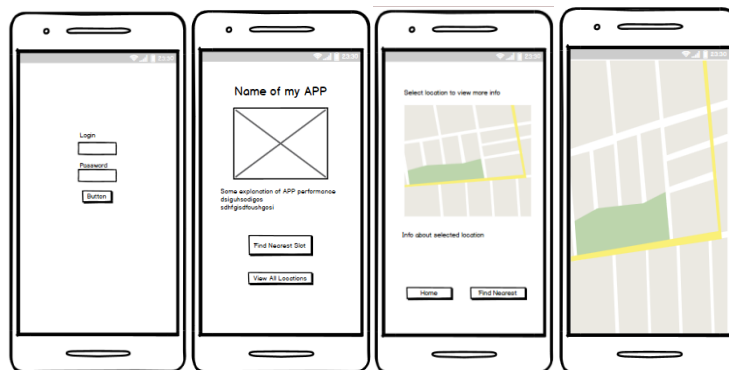
The mobile application (App) developed in Android Studio v3.6 using Java language is providing user friendly interface. For testing and debugging the application while developing I was using Android device that operates Android OS 10, API level 29.

Android studio has an option to develop an application using Kotlin programming language, but I was not familiar with it and have chosen Java language over it.

Mobile application's required minimum API level is 26 which means that it is designed for Android devices that operate on Android OS versions 8.0 or higher. According to Android Studio IDE, developed application can be run on >60% of android devices.

Mobile app consists of three Activities, two database model classes and two service classes.

For development of application's layout, I used Balsamiq Wireframes software. This software allows to visualise look of application before functional development begins. Using this software, I have developed a sketch of my application layout (Fig.7-1), and was referring to It during development of an app.



**Fig. 7-1** Pre-development layout

### 7.1 Main activity

The “Main activity’s” function is to provide user interface for login page. Once user of an app touches *Sign In* button, the app is sending GET request to the server, receives a response and redirecting to new activity.

Visual layout (fig.7-1-1):

- 2 EditText fields
- 3 ImageView boxes
- Button



Fig. 7-1-1

For RESTful communication over HTTP I am using *OkHttp client*. The client works on Android 5.0+ (API level 21+) and on Java 8+. The client builds a request using provided URL, sends it, and waits for a response. Once request is resolved, there is a callback function called *onResponse* that overrides, parsing the response body. Within this callback function I am creating a List of type *CarParkModel* (Is a class where I am replicating my database's columns) by using *ObjectMapperService's readValueAsList* method and passing response data to it.

The *ObjectMapperService* class is using Jackson Library which is used for serializing and deserialising JSON objects. I used [tutorialspoint.com] and [stackoverflow.com] websites where I have learnt essential basics of Jackson library (Fig.7-1-2) In my application I am using it to deserialize JSON string into a String java object.

```

15 public class ObjectMapperService {
16
17     private static final ObjectMapper objectMapper = new ObjectMapper();
18
19     public static <T> List<T> readValueAsList(String object, Class<T> classType) {
20         try {
21             CollectionType type = objectMapper.getTypeFactory().constructCollectionType(List.class, classType);
22             return objectMapper.readValue(object, type);
23         } catch (JsonProcessingException e) {
24             throw new RuntimeException(e);
25         }
26     }
27 }

```

Fig.7-1-2

Deserialized object is then saved using *CacheService* class. This class allows me to share resources between Android Activity screens instead of standard android resource bundles. Within that class I am creating a HashMap that is accepting parameters String as a key and object as a value, to make sure that any type of object can be used (Fig. 7-1-3).

```

7 public class CacheService {
8
9     public static final String CARPARK_LIST = "CARPARK_LIST";
10    // public static final String CARPARK_SLOT_LIST="CARPARK_SLOT_LIST";
11
12    private static Map<String, Object> cache = new HashMap<>();
13
14
15    public static void put(String key, Object object) {
16        cache.put(key, object);
17    }
18
19    public static <T> T get(String key, Class<T> valueType) {
20        return (T) cache.get(key);
21    }
22
23    public static <T> List<T> getList(String key, Class<T> valueType) {
24        return (List<T>) cache.get(key);
25    }
26
27 }

```

Fig. 7-1-3

After all, the new thread is created. *runOnUiThread* runs the specified action on the UI thread, which is creating new intent and starting new activity (Fig.7-1-4).

```

55 OkHttpClient client = new OkHttpClient();
56 String url = "https://0ce73247.ngrok.io/car-park-locations";
57 Request request = new Request.Builder()
58     .url(url)
59     .build();
60
61 client.newCall(request).enqueue(new Callback() {
62     @Override
63     public void onFailure(Call call, IOException e) { e.printStackTrace(); }
64
65     @Override
66     public void onResponse(Call call, Response response) throws IOException {
67         if (response.isSuccessful()) {
68             final String myResponse = response.body().string();
69
70             final List<CarParkModel> carParksInfo = ObjectMapperService.readValueAsList(myResponse, CarParkModel.class);
71             CacheService.put(CacheService.CARPARK_LIST, carParksInfo);
72
73             MainActivity.this.runOnUiThread() -> {
74                 //Creating new intent
75                 Intent welcomeIntent = new Intent( packageContext: MainActivity.this, WelcomeActivity.class);
76                 startActivity(welcomeIntent);
77             }
78         }
79     }
80 }

```

Fig. 7-1-4

## 7.2 Welcome activity

The “Welcome activity” is providing user of an application with two options – “find nearest slot” and “Show all locations”.

Visual layout (fig.7-2-1):

- 2 Text fields
- 1 ImageView fields
- 2 Buttons



Fig. 7-2-1

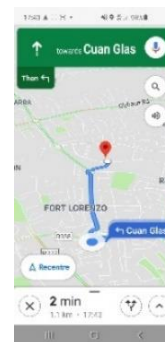


Fig. 7-2-2 Navigation

- “Find nearest slot” button will request user to turn on GPS location service, in order to grant permission for app to access user location information. Then activity will switch to new window, that will display dropped pin on the google map (destination point) at the nearest location of a carpark that has available slots and provide driving directions (Fig. 7-2-2).
- “Show all locations” button will open a new activity in android app and display information about all locations where system was implemented. This information is pulled from database and will expand if system will be implemented in other locations. Also, activity will display a map in split screen displaying markers(pins) at locations.

In order to work with GPS coordinates and locations I am using *Android.location* library which is designed for such operations.

I am using *LocationListener* interface which contains four methods and overriding *onLocationChanged* method (Fig.7-2-3), which is called when location has changed.

In this method I need to compare my current location GPS data with data that is stored in database. The information received in previous activity was stored in cache service, therefore I am creating a list and calling a *CacheService* class's method *getList*, which will return information on all carpark in database.

Also, I am creating another list where distance values will be stored. This data is analyzed and used further in the code.

Then, I am using for-each iterator to iterate through every element in the list create new *double* variables and parse information of GPS latitude and longitude. To compare and calculate minimal travel distance to each car park location I am using library's *distanceTo* method by passing a *loc* object to it, which consists of two *double* variables –latitude and longitude and which is returning a float value. This float value is travel distance to each car park in meters. I am saving all these distances to the list.

In order to know which location is nearest, I need to know its index number in the List. Using *Collections.min(distances)* method allows me to compare all values in my list of distances, find the smallest number and assign it to *minDist* variable.

Further, I am using for loop and setting it to loop amount of times which is equal to size of distances list. In this for-loop I am comparing *minDist* with list of distances. Once, there is a match, the id of location is determined.

```

155 locationManager = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
156 locationListener = new LocationListener() {
157     @Override
158     public void onLocationChanged(Location location) {
159         List<CarParkModel> carparkList = CacheService.getList(CacheService.CARPARK_LIST, CarParkModel.class);
160
161         List<Float> distances = new ArrayList<>();
162         for(CarParkModel cpModel : carparkList){
163             double lat=Double.parseDouble(cpModel.getGpsLatitude());
164             double lon= Double.parseDouble(cpModel.getGpsLongitude());
165             Location loc = new Location( provider: "dest");
166             loc.setLatitude(lat);
167             loc.setLongitude(lon);
168             float distance = location.distanceTo(loc);
169             distances.add(distance);
170         }
171         //Acquiring minDistance value and getting id of location with minDistance
172         float minDist = Collections.min(distances);
173         for(int i=0;i<distances.size();i++){
174             if (distances.get(i)== minDist)
175                 distanceId = i;
176         }

```

Fig.7-2-3

The *viewMap* method (Fig.7-2-4) is run when user touches the “Find nearest” button and will open new activity called ACTION\_VIEW.

I am utilizing the ID of location previously acquired by data manipulation in method described above and passing it as argument to *carparkList.get* method. In order to place



a pin on the map and provide driving directions to the location I am creating a *String uriGeoCode*. This String must be of format that is accepted by google maps API and is requiring gps latitude and longitude values. Also, additional parameters can be added – like zoom that I am using to zoom in the map.

```

178 private void viewMap() {
179     //Dropping pin on google map for nearest slot available
180     List<CarParkModel> carparkList = CacheService.getList(CacheService.CARPARK_LIST, CarParkModel.class);
181     CarParkModel carParkModel = carparkList.get(distanceId);
182     String uriGeoCode = "geo:0,0?q="+carParkModel.getGpsLatitude()+" "+ carParkModel.getGpsLongitude()+"&z=16";
183     //Creating new intent for google maps
184     Intent intent = new Intent(Intent.ACTION_VIEW, Uri.parse(uriGeoCode));
185     startActivity(intent);

```

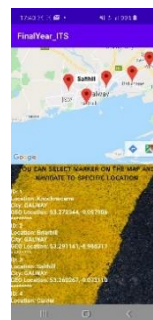
**Fig.7-2-4** View map method

### 7.3 View All locations activity

The activity is displaying information about all locations where system was implemented in, using *scrollView* (text area that can be scrolled through). This information is pulled from database and will expand if system will be implemented in other locations. Also, activity is displaying a map of all carpark locations by dropping pins on google map, using *mapView* (a field to display an interactive map).

Visual layout (Fig.7-3):

- Scrollview field
- MapView field



**Fig. 7-3**

For this activity to function I am using Google API that requires an API key.

The API key is a unique identifier that is used to authenticate requests associated with the project. And it is also used to track and control use of API.

A MapView displays a map (with data obtained from the Google Maps service). When focused, it will capture keypresses and touch gestures to move the map.

In order to use a MapView I must forward all the lifecycle methods from the activity containing this view to the corresponding ones in this class. Required methods that must be forwarded are:

- onCreate(Bundle)
- onStart()
- onPause()
- onStop()

- onDestroy()
- onSaveInstanceState()
- onLowMemory()

To acquire a GoogleMap I am using *getMapAsync* callback function. The MapView automatically initializes the maps system and view.

By re-using list of car parks, I have created three separate lists in order to segregate data. By using lists, I am ensuring that the program will work efficiently. Currently there are five car parks registered in the database – and I could write 3 lines of code for each location, for markers to appear on the map. Three lines for each location would add up to 15 lines of code in total. If database would hold information on 100 locations – program would be 300 lines longer, which is not efficient way of writing code.

For-each iterator is used to iterate through every element in collection and extract required data. I am parsing GPS coordinates as *double* type variables, separating them in two lists (*geoLat* and *geoLon*) and separating names of locations into third list.

Running a for-loop ensures I am placing a marker(pin) and its name on the map by creating new markers and passing values stored in the lists discussed above.

```

67     mapView.onCreate(savedInstanceState);
68     mapView.getMapAsync(new OnMapReadyCallback() {
69         @Override
70         public void onMapReady(final GoogleMap map) {
71
72             /**
73              * Separating Latitude , Longitude and Location names into arrayList
74              */
75             List<Double> geoLat = new ArrayList<>();
76             List<Double> geoLon = new ArrayList<>();
77             List<String> locationNames = new ArrayList<>();
78             for(CarParkModel model : carparkList){
79                 coordsLat = Double.parseDouble(model.getGpsLatitude());
80                 coordsLon= Double.parseDouble(model.getGpsLongitude());
81                 locName=model.getName();
82                 geoLat.add(coordsLat);
83                 geoLon.add(coordsLon);
84                 locationNames.add(locName);
85             }
86             for(int i=0;i<geoLat.size();i++) {
87                 LatLng coordinates = new LatLng(geoLat.get(i), geoLon.get(i));
88                 map.addMarker(new MarkerOptions().position(coordinates).title(locationNames.get(i)));
89                 map.moveCamera(CameraUpdateFactory.newLatLngZoom(coordinates, 12));
90             }
91             mapView.onResume();
92         }
93     });
94

```

## 8 PROBLEM SOLVING

### 8.1 Server development issues and challenges

Connecting to the database – persistence layer was the most challenging part. I have encountered problems during development of Persistence layer using standard JDBC template provided by Spring framework. To be more specific, the issues were occurring during execution of SQL statements.

The code part of the layer was not giving any errors and I was able to run the server, but when I was testing HTTP requests using ‘Postman’ was getting HTTP response status 500, which is an internal server error. This error was accompanied with exceptions thrown such as *IndexOutOfBoundsException*.

After using a debugger and investigating the issue I had no progress made. I had to contact one of my friends to have a look into it with me and maybe find the issue, because server was running fine.

The problem was the types of Lists that I have assigned throughout the code.

As a solution, he suggested to alter the JDBC template, instead of altering the whole server code. He kindly helped me to do it and I have acknowledged his help regarding this matter in the “Acknowledgements” section at the top of this report.

### 8.2 Mobile application development issues and challenges

#### Locally hosted server

When I began developing android application, I encountered one major issue where I had to test communication between app and a RESTful server. The issue was, that my server was hosted locally on my computer and I could only access it from localhost. I have researched web resources to find a way of making locally hosted server be available for access from public. The ngrok software was my solution. This software is very simple to use, and it provides secure tunnels from public endpoint to a locally running network service.

While working with this issue I also discovered OkHttp client. Which provided me with tools for building a http requests and sending them.

#### MapView

The challenging part of developing android application was In *ViewAllLocations* activity. I had to ‘split’ the screen in two parts. Where one part would contain text information about car parks and other part would have mapped locations of these locations. In order, to be able to implement MapView I had to watch video tutorials on youtube.com and refer to android developer website. One part of correct implementation of MapView was to acquire unique Google API key for my project. The API key is a unique identifier that is used to

authenticate requests associated with the project. And it is also used to track and control use of API.

### 8.3 Raspberry Pi/Object recognition issues and challenges

During project work on Raspberry Pi, I had encountered two major problems, which were hardware and software.

#### Hardware:

After successful installation of operating system, OpenCV and other required components to work with object recognition, the Raspberry Pi suddenly stopped booting.

To troubleshoot the issue, I checked most common reasons of this issue, which is damaged power supply, ports or PS cable. After confirming no fault occurred in mentioned hardware components, I stepped further and examined LED indicator flashing pattern.

I used web resources to check whether the flashing pattern matches any of errors associated with the board.

The error was indicating that SD card was unable to be read – I have checked SD card on another device, which confirmed faulty card.

Finally, I had to re-install all the software onto another card, because I had no backup image of previous OS setup.

#### Software:

After flashing a new SD card with latest image of Raspbian OS (Operating system) that I downloaded from official Raspberry Pi website, I noticed significant change in operating speed of OS – the board was constantly ‘freezing’ without any reason. Before troubleshooting the issue, I flashed card once again to eliminate possibility that operating system was corrupted during flashing process. Confirming, that issue still persists I moved on to troubleshoot issue by checking web resources for any information regarding latest OS from Raspberry Pi. Finding no information online, I decided to flash SD card once again, but using older version(pre-update) of OS.

The issue was resolved, and It appeared to be that image of OS provided on official website was corrupted.

## 9 CONCLUSIONS

The outcome of a project is a prototype of the system which is completely autonomous and does not require any interaction from developer side once server is started and Raspberry Pi is executing a script. The server accepts, processes and responds client's requests coming from Raspberry Pi and Android mobile application as intended.

In my opinion, the idea of the project is worth to be developed further. The world is moving fast into direction of creating 'smart' cities where technologies play great role.

The greatest advantage of my system is its cost and relatively simple installation comparing to complex systems that are used currently, which are using sensors, such as proximity sensors.

Good features of the system are low cost implementation, flexibility in use, access from any device with internet connection and geo sensor. The system can also implement a 'Pay as much, as you use' scheme, where drivers are paying for exact time that they are parked at location. And in my opinion this system can be an innovative replacement for parking pay-machines.

During development of the 'Time Saving System' I have researched and discovered new technologies, development tools, frameworks and development problem solutions. I have greatly improved my knowledge of Java programming language, databases and my expertise in technical writing.

## 10 REFERENCES

- [1] About OpenCV. (2020). OpenCV team. [ONLINE] <https://opencv.org/about/>. [Accessed 17 May 2020].
- [2] ODSC - Open Data Science. (Sep.25, 2018). Overview of the YOLO Object Detection Algorithm. [ONLINE] <https://medium.com/@ODSC/overview-of-the-yolo-object-detection-algorithm-7b52a745d3e0>. [Accessed 17 April 2020].
- [3] Abhimanyu Prasad. (May 12,2016). Understanding Transactional annotation in Spring.[ONLINE] <https://www.javacodegeeks.com/2016/05/understanding-transactional-annotation-spring.html>. [Accessed 21 April 2020].
- [4] Matthew Davis. (Jun. 17,2014). Five Reasons You Should Use Tomcat. [ONLINE] <https://www.futurehosting.com/blog/five-reasons-you-should-use-tomcat/> .[Accessed 17 April 2020].
- [6] Wikipedia. (May 13,2020). Representational State Transfer. [https://en.wikipedia.org/wiki/Representational\\_state\\_transfer](https://en.wikipedia.org/wiki/Representational_state_transfer) .[Accessed 16 May 2020].
- [7] Ngrok. (2020). Product. [ONLINE]. <https://ngrok.com/product>. [Accessed 17 April 2020].

### Books and example code

- [5] Adrian Rosenbrock. (Nov. 12,2018). YOLO Object detection with OpenCV. Object recognition source code. [ONLINE] <https://www.pyimagesearch.com/2018/11/12/yolo-object-detection-with-opencv/> .

Felipe Gutierrez. (2016) Pro Spring Boot. Apress. [Example code].

### Websites used during development

Baeldung.com / Spring.io / Tutorialspoint.com / Stackoverflow.com / Pyimagesearch.com/ Youtube.com / MVNRepository.com