

# **QUICK-E-PARK**

**An online parking ticket web application**

**Submitted by**  
Thirumal Janakiraman

## Table of Contents

Abstract .....	vi
Abbreviations & Definitions .....	vii
1. Introduction and Motivation .....	1
1.1 Problem Statement .....	1
1.2 Proposed System .....	2
1.3 Purpose .....	2
1.4 Scope .....	3
1.5 Intended Audience.....	3
2. Market Research & Existing System .....	4
3. Requirement Analysis.....	7
3.1 User Requirements .....	7
3.2 Functional Requirements.....	7
3.3 Non-Functional Requirements .....	8
3.4 Use Case Diagrams .....	8
3.4.1 Customer/End-User Use Case.....	8
3.4.2 Checker Use Case .....	9
3.4.3 Admin Use Case.....	9
4. System Architecture.....	10
5. Database Architecture.....	11
6. Data flow Diagram .....	13
7. Technologies Used.....	14
7.1 Front-end Development.....	14
7.1.1 React.....	14
7.1.2 Axios .....	14
7.1.3 React-html5-camera-photo.....	14
7.2 Back-end Development .....	14
7.2.1 Python Flask Framework .....	14
7.3 Database .....	15
7.3.1 Postgres .....	15
7.4 Docker .....	15
7.5 Nginx .....	15
8. Functional Overview .....	17
8.1 Functionalities of User .....	17
8.2 Functionalities of Checker.....	17
8.3 Functionalities of Admin.....	18

9. Implementation .....	20
9.1 User Functionalities.....	20
9.1.1 Registration .....	20
9.1.2 Login .....	22
9.1.3 Buy Parking Ticket .....	24
9.1.4 View Active Ticket - User: .....	26
9.1.5 View Fines User.....	28
9.1.6 Extend Ticket .....	29
9.1.7 View Payment .....	30
9.1.8 Clear Payment .....	32
9.1.9 View Ticket History .....	33
9.1.10 Add Vehicles.....	35
9.1.11 Show Vehicles.....	37
9.1.12 Update Password.....	38
9.1.13 Update Profile .....	39
9.2 Checker Functionalities .....	40
9.2.1 Registration of Checker .....	40
9.2.2 Checker Login.....	41
9.2.3 Parking Ticket Validation .....	43
9.2.4 View Ticket Checker .....	44
9.2.5 Issue Fine .....	46
9.2.6 View Issued Fine.....	47
9.3 Notification and Ticket History.....	48
9.4 License Image Extraction.....	49
10. Deployment.....	55
11. Evaluation .....	57
11.1 Frontend User Interfaces .....	57
11.2 Backend Web Services .....	58
12. System Limitation .....	60
13. Conclusion .....	61
14. Future Scope .....	62
References.....	63
Appendix.....	65

## List of Figures

Figure 1: Parking Symbol [2].....	1
Figure 2: Parkscheinautomat [3] .....	1
Figure 3: Customer use case .....	8
Figure 4: Checker Use case.....	9
Figure 5: Admin use case.....	9
Figure 6: Architecture Diagram .....	10
Figure 7:User database.....	11
Figure 8: Checker & Admin database.....	11
Figure 9: Data Flow Diagram .....	13
Figure 10: User functionalities.....	17
Figure 11: Checker functionalities.....	18
Figure 12: Admin functionalities .....	19
Figure 13: End-User Successful Login .....	20
Figure 14: User Registration Page 1 .....	21
Figure 15: User Registration Page 2 .....	21
Figure 16: User Login Page .....	23
Figure 17: User Dashboard Page .....	23
Figure 18: Parking Ticket Tab - Buy Ticket and View Ticket .....	25
Figure 19: View Fines Tab.....	28
Figure 20: Payment Tab.....	31
Figure 21: Payment Portal Module .....	32
Figure 22: Ticket History Tab.....	34
Figure 23: Vehicles Tab.....	36
Figure 24: Account Profile Tab .....	38
Figure 25: Checker Login Page .....	41
Figure 26: Checker Dashboard Page.....	41
Figure 27: Scan and Check Tab .....	43
Figure 28: Upload and Check Tab .....	44
Figure 29: Fines tab for Checker.....	47
Figure 30: Sample Notification.....	49
Figure 31: Encoding and decoding of base64.....	50
Figure 32: Working of OCR .....	51
Figure 33: Conversion from one colour scale.....	52
Figure 34: Resizing of Image.....	52
Figure 35: Filtered Image.....	53
Figure 36: Edge Detection Formula [29] .....	53
Figure 37 : Edge detection .....	53
Figure 38: Contour detection .....	54
Figure 39: Extracted License Number .....	54
Figure 40 : Nginx Configurations File.....	55
Figure 41 : Docker Compose File .....	56
Figure 42:Ex:Login Screen Validation .....	57
Figure 43: Galaxy S5 Mobile Screen.....	58
Figure 44: iPad Screen .....	58

**List of Tables**

Table 1: Comparison table of existing solutions.....	6
------------------------------------------------------	---

## **Abstract**

The project aims to create an online parking application that simplifies the existing parking process and provides a user-friendly experience by reducing the dependency on the physical parking machines "Parkschein." The completion of this project will lead to a simplified parking ticket system for the end-users, which will help them to buy parking tickets in a matter of clicks on the computer or mobile devices. It will also benefit the Ordnungsamt by having remotely accessible parking data. The checking of parking tickets should also be done easily using this application.

The project's primary goal is to design and develop a web application with all the functionality necessary for buying parking tickets, extending or ending the parking sessions with an intuitive User Interface for the end-user. Another important goal of the project is to develop a module to scan the number plates of vehicles and extract vehicle registration numbers using image processing. This is meant for Ordnungsamt to make their job easier for validating the parking tickets. This project is developed using Python-based Microservices, Postgres DBMS, React library for front-end, and Docker for deployment. Some other tools used during development and for the successful completion of the project are Postman and GitHub.

## Abbreviations & Definitions

<b>Terms</b>	<b>Definitions/Abbreviation</b>
Checkers	The application term for Ordnungsamt
OCR	Optical Character Recognition
Web Application	Quick-E-Park website
App	Application
Parkschein	Parking sign
Parkscheinautomat	Parking ticket vending machine
SMTP	Simple Mail Transfer Protocol
API	Application Programming Interface
Admin	Administrator
RDBMS	Relational Database Management System
Ordnungsamt	Public Order Services
Users	Customers who would use the web application
HTTP	Hypertext Transfer Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
HLS	HTTP Live Streaming
HDS	HTTP Dynamic Streaming
IMAP	Internet Message Access Protocol
POP3	Post Office Protocol version 3
UI	User Interface
URL	Uniform Resource Locator
REST	Representational State Transfer
UID	Unique Identifier is created for each user at the time of Registration.
Employee ID	Unique Employee ID is given by Administrator for each employee for accessing web application.
HTML5	HyperText Markup Language Version 5
OpenCV	Open Source Computer Vision Library

# 1. Introduction and Motivation

In the current digital era, people are experiencing the digitalization of nearly every product and service which used to be offline. Digitalization helps in the ease of use and maintenance and in the enhancement or addition of new functionalities. However, an exceptional case like the Parking ticket system in Germany is not digitized. People living in Germany use the "Parkscheinautomat", a parking ticket machine placed at every street corner where parking is available. The driver has to park in a defined parking area and then buy a parking ticket from the machines manually. Although there are many digital parking ticket applications, most of them are not widely used due to the lack of all the features in a single application. Quick-E-Park is developed as an evolved form of these digital applications and will be managed by the city-council to ease parking ticket management in Germany.

There is a wide range of parking options available in Germany for parking a car, such as public parking, also known as on-street parking, garages, and private parking areas. For public parking, the vehicles can only be parked if the street has a "mit Parkschein" symbol board [1]. Once the parking spot is found and the car is parked, people have to reach the Parkscheinautomat to buy the parking tickets [1]. The parking ticket can be purchased for fixed hours and needs to be pre-paid accordingly using Parkscheinautomat [1]. Upon payment, a receipt is generated, which has to be placed on the car's windshield. Extending the parking ticket needs to be done manually. The amount will now directly go to the city council as they are the ones who are maintaining all these services.



Figure 1: Parking Symbol [2]



Figure 2: Parkscheinautomat [3]

For checking and validating parking tickets, the city council has the Ordnungsamt department, which takes care of all the verification process. The Ordnungsamt manually checks for the parking ticket on the car's windshield and validates if the ticket is valid. In case the parking ticket is invalid or expired, the owner will be issued with a fine by sticking a penalty notice on the car [1].

## 1.1 Problem Statement

In the existing parking system, the users have to assume the number of hours for which they want to park their car and accordingly buy a parking ticket from the Parkschein machines. The main problem here is that it is difficult to determine the exact number of hours for which the ticket has to be bought due to uncertainties. For example: a person parks his car and buys a parking ticket for an hour to visit



a nearby shop. But due to some reason, the shop has a large number of customers and people have to wait in a queue before going in. Here the large queue is uncertain and it is challenging to predict that. In this case, the parking ticket is going to expire before the person enters the shop. This way, he has to go back to the parking machine to extend the ticket and it is not convenient. Also, the person has to stand at the back of the queue after returning from the Parkschein machine. Another issue is that during busy hours, people have to queue up in front of the machines and touch the same screen to be able to use the service. During the pandemic time, this process is risky as people can come in contact with the disease. Coins are mostly required in order to pay for the ticket although some machines accept card payments. The paper receipt generation causes sustainability issues on a small scale like the wastage of paper. These problems can be resolved if the existing offline system can be replaced with an online system.

## **1.2 Proposed System**

Quick-E-Park is an online parking ticket system that enables real-time services for users who park their vehicles in public places. This online system is developed to replace the existing legacy parking machines to overcome limitations such as the buying and extension of tickets using the machine, queuing up in front of the machine, and sustainability. The application is developed in such a way that it should meet the following criteria:

- Minimize the need for a physical parking machine in the process of buying or extending the parking ticket.
- Users should be easily able to extend their vehicle parking time remotely.
- Can pay online and also get the history of the parking.
- Checkers can scan the number plate and get parking details.
- Checkers can issue fines and check for a history of fines online.
- Eco-friendly - Paperless
- Saves time - No more queues

## **1.3 Purpose**

The purpose of this project is to create an online parking management system that aims to simplify the current process by providing a user-friendly web-based parking management system. The web application enhances the parking experience and makes the parking process easier in Germany for both end-users and stakeholders. Additionally, all the parking ticket data should be available and accessible remotely for both end-users and stakeholders. The web application should be intuitive to the end-user. i.e., the interface should be user friendly so that the end-user can perform the process without any difficulty or confusion.

## **1.4 Scope**

This project's scope is to design and develop a web application of an online parking ticket management system. The users for this product are the customers, checkers, and the administrator of the system. The customers should be provided the functionalities such as buy, extend, and view parking tickets. The checkers should use the application to validate the parking ticket of vehicles. The administrator's responsibility is to manage the checker's data, view the parking tickets, and issue customers' fines. This web application eases the parking ticket system for all the users and eliminates the use of Parkscheinautomat.

- This application does not provide reserve parking spaces for the user, and it is only limited to On-Street Parking.
- The application is free of cost and owned by the city council.
- All the user details, transaction details, and location details are stored securely in the relational database system.

The project is considered as completed after creating the fully-functional working web application providing all the functionalities below:

- Buying parking tickets online.
- Extending parking tickets online.
- Paying for the parking ticket online.
- Paying fines online.
- Checking the history of the parked sessions.

## **1.5 Intended Audience**

The project's intended audiences are the parking management stakeholders and the customers who will use online parking ticket systems.

## 2. Market Research & Existing System

Based on the market research, it was found that there are several web systems and applications that offer a similar parking service. During the analysis of the systems such as Park Now, Easy Parking, Pay by Phone, and Peter Park, it is observed that people living in Germany are not widely accepting these applications because of certain flaws or irregularities in them.

### Parking Machines

Using Parkschein, the customers have to buy the parking tickets. The customer has to deposit the amount based on how much time the car needs to be parked. The user has to return to the vehicle before the ticket gets expired or to extend.

*Issues:*

- The extension can be done only physically.

### Park Now

For using this app, the users have to register and then have to get a parking card or sticker which acts as an access card for the vehicle. Then all the basic functionalities can be done via the app. They charge for their services, 3 euros a month [4].

*Issues:*

- Mobile App – Needs to be installed.
- High service charge.
- The same car cannot be managed on several devices (not good for sharing a car. i.e., In a family).

### Pay by phone

This app also allows parking services online but it can be used without registering the vehicle in the app. So, it can be misused in several ways. For payment, they have linked the mobile number of the user, and later all parking bills come in addition to the phone bill. In some places, the user also has to submit/inform the inspector about the parking by submitting the sticker to them. The arrival of the sticker can take up to 10 days [5].

*Issues:*

- Mobile App – Needs to be installed.
- Physical Sticker is required to identify.
- No vehicle registration validation – Security issues may arise.
- Through play store reviews, analysed payment issues.

### Peter Park

Their system is basically designed to automate the parking lots which have entry and exit doors. They aim to make the process automated by recognizing the vehicle number from the license plate and then open and closing of the gates will be done according to that [6].

Issues:

- For Businesses – set up needs to be installed.
- Hardware investment required.

### Easy Park

This app allows all the basic functionality to be performed online like buying, extending, or ending parking tickets. They have installed the cameras in the street which tells the user the possibility of finding the vacant parking space and is widely used in Scandinavian countries [7].

Issues:

- Mobile App – Needs to be installed.
- No vehicle registration validation – Security issues may arise.

<i>Name</i>	<i>Type of solution</i>	<i>Service charges</i>	<i>License Plate validation during registration</i>	<i>Additional requirements</i>	<i>Checking method</i>
<b><i>Parking Machines</i></b>	Hardware Machines Need Physical Presence	No	No	Hardware	Manual checking of Tickets
<b><i>Park Now</i></b>	Mobile app Installation Required	25 cents/Per session	No	--	Parking sticker, Garage card, Plate scanning
<b><i>Pay by phone</i></b>	Mobile app Installation Required	10% of the session	No	--	Details sent to the local authority

<b><i>Peter Park</i></b>	B2B, partnered application		No	Cameras for license plate recognition	Plate scanning
<b><i>Easy Park</i></b>	Mobile app Installation Required	Yes	No	Camera for visualizing the vacant spaces for parking	Plate scanning
<b><i>Quick-E- Park</i></b>	Web app No installation	No	Yes	No additional Requirements	Plate scanning

*Table 1: Comparison table of existing solutions*

### **3. Requirement Analysis**

Requirement analysis involved identifying the functional requirements and non-functional requirements. The functional requirements analysis identifies the activities that end-users (including both customers and checkers) can operate. Meanwhile, the non-functional requirements analysis involved security, performance, implementation, testing, and maintenance of the application.

#### **3.1 User Requirements**

- Customers should be able to make a free registration on the website to use the online parking system by filling in all the required data in the registration form.
- Customers should accept the terms and Conditions for accessing the website during the registration.
- Customers should be able to log in with the registered email and the password.
- Customers should be able to reset the password if they have forgotten the password.
- After a successful login to the application, the customers should be able to buy a ticket by choosing the location, starting time, and duration.
- Customers should be able to extend the parking ticket anytime.
- Customers should be able to view ticket history, fines, and payment status.
- Customers should be able to add multiple vehicles to his profile.
- Customers should be able to update his account profile.
- Customers should be allowed to log out of the system anytime.

#### **3.2 Functional Requirements**

- The application should allow checkers to issue fines if a parking ticket has expired or has not been taken.
- Once a parking ticket is about to expire, the application should send a notification to the customer.
- The application should move the user's active parking tickets to their ticket history once the parking tickets get expired.
- The application should allow the customers to make the payment of the ticket and fines anytime, 24/7.
- The application should let the checkers verify if the parking is valid by scanning the vehicle license plate.
- The application should have an admin who can manage to create and delete credentials for the checker and manage databases.

### 3.3 Non-Functional Requirements

- The application should allow checkers to issue fines if a parking ticket has expired or has not been taken.
- Once a parking ticket is about to expire, the application should send a notification to the customer.
- The application should move the user's active parking tickets to their ticket history once the parking tickets get expired.
- The application should allow the customers to make the payment of the ticket and fines anytime, 24/7.
- The application should let the checkers verify if the parking is valid by scanning the vehicle license plate.
- The application should have an admin who can manage to create and delete credentials for the checker and manage databases.

### 3.4 Use Case Diagrams

#### 3.4.1 Customer/End-User Use Case

Figure 3 illustrates the use case diagram of the customers.

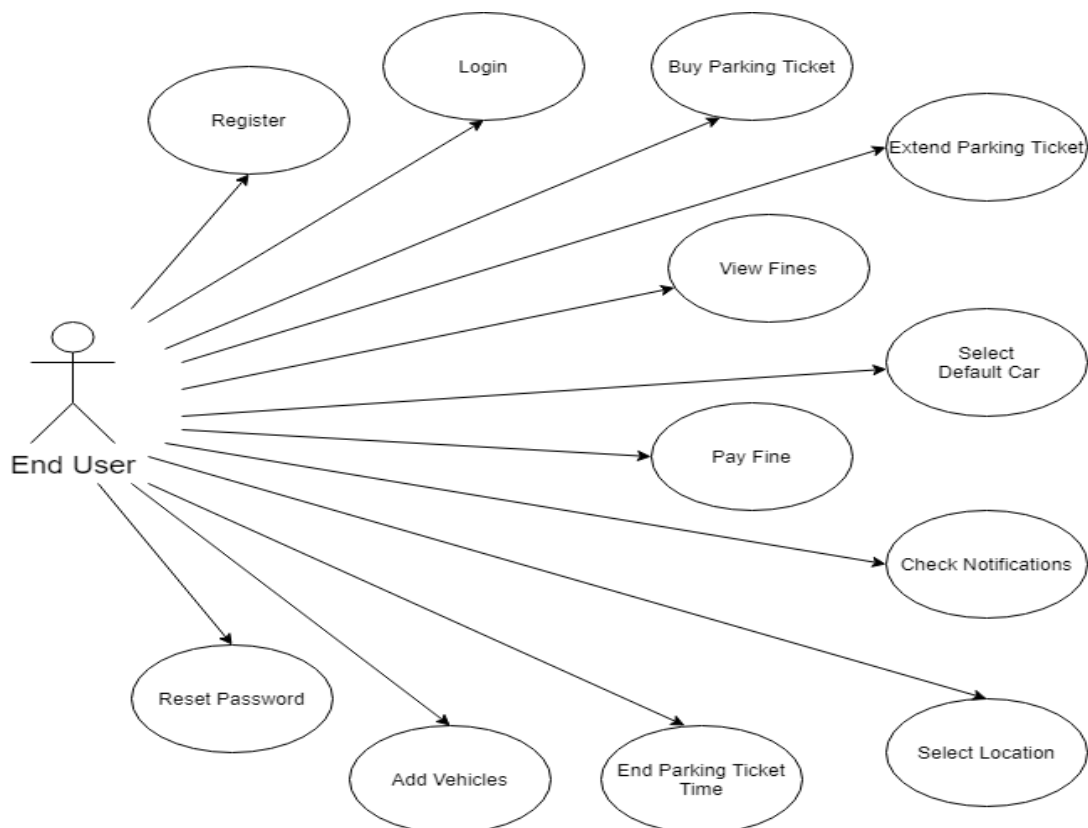


Figure 3: Customer use case

### 3.4.2 Checker Use Case

Figure 4 illustrates the use case diagram of the checkers.

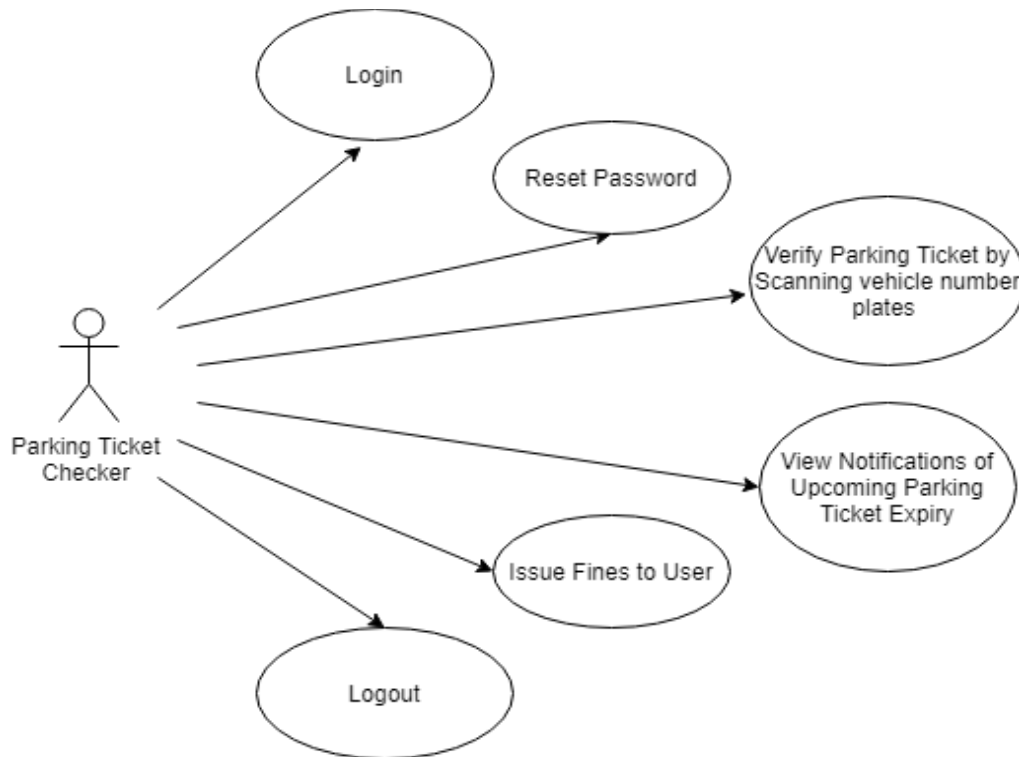


Figure 4: Checker Use case

### 3.4.3 Admin Use Case

Figure 5 illustrates the use case diagram of the system administrator.

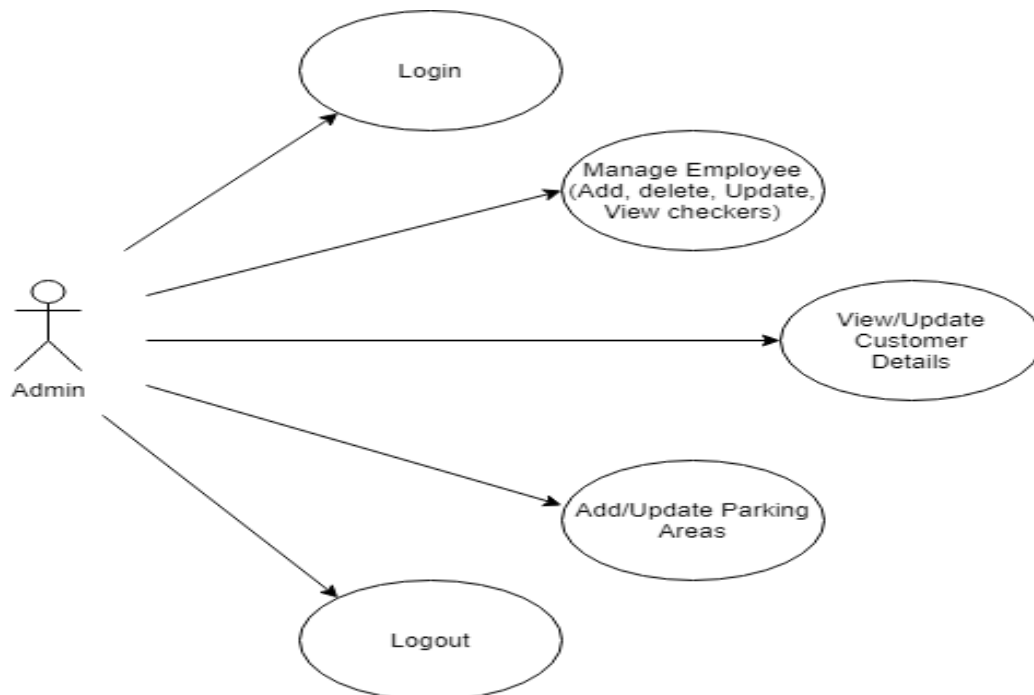


Figure 5: Admin use case



## 4. System Architecture

Quick-E-Park follows a client-server architecture. A client-server architecture has two parts namely client and server. Client is where all the front-end related operations occur. The client makes all the requests to the server, as the client does not have enough processing power. Clients receive all the responses from the server and show it in the front-end i.e., browser. So, a client acts as an interface between user and server [8].

A server can be defined as a processor which receives the request from the client, processes that request and gives the desired result back to the client in the form of the response. All the communication between the client and server takes place with the help of a network [8]. Furthermore, all the services running in the server are designed as microservices i.e., all the backend services are independent of each other [9].

Figure 6 shows the overall system architecture of the Quick-E-Park web application.

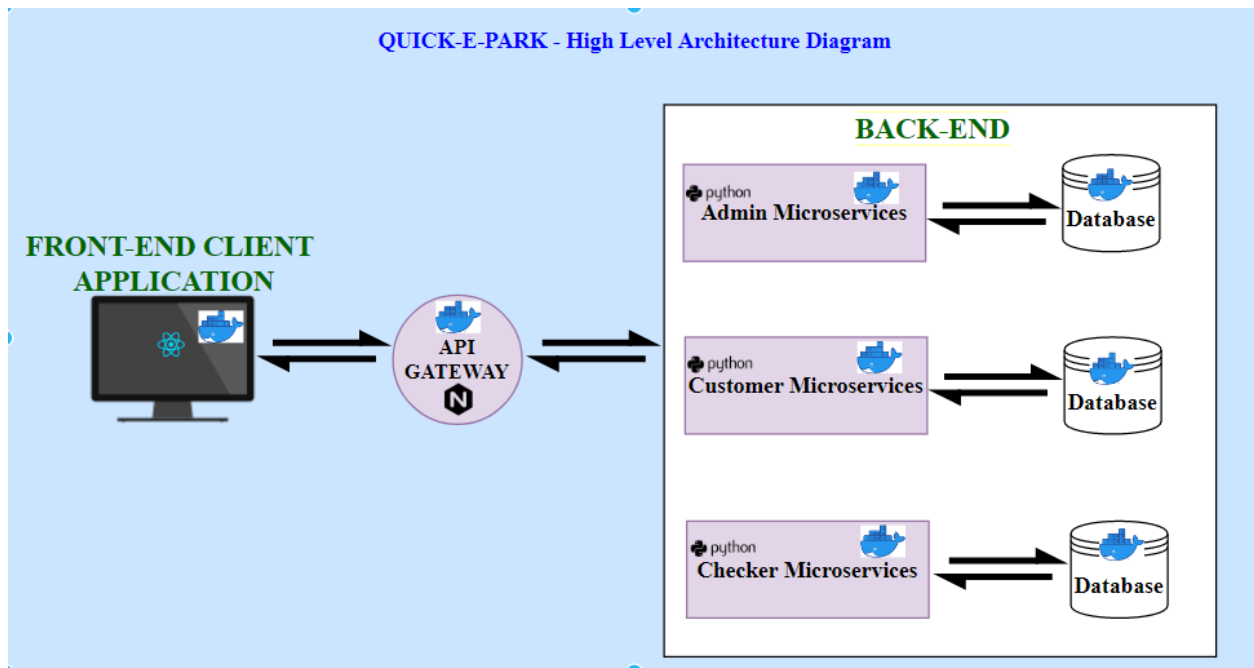


Figure 6: Architecture Diagram

## 5. Database Architecture

Quick-E-Park is implemented using RDBMS(PostgreSQL) for storing, manipulation and fetching of data by the application. The databases are classified into the Parking database, Checker database and Admin database as mentioned in the figure 7 and figure 8.

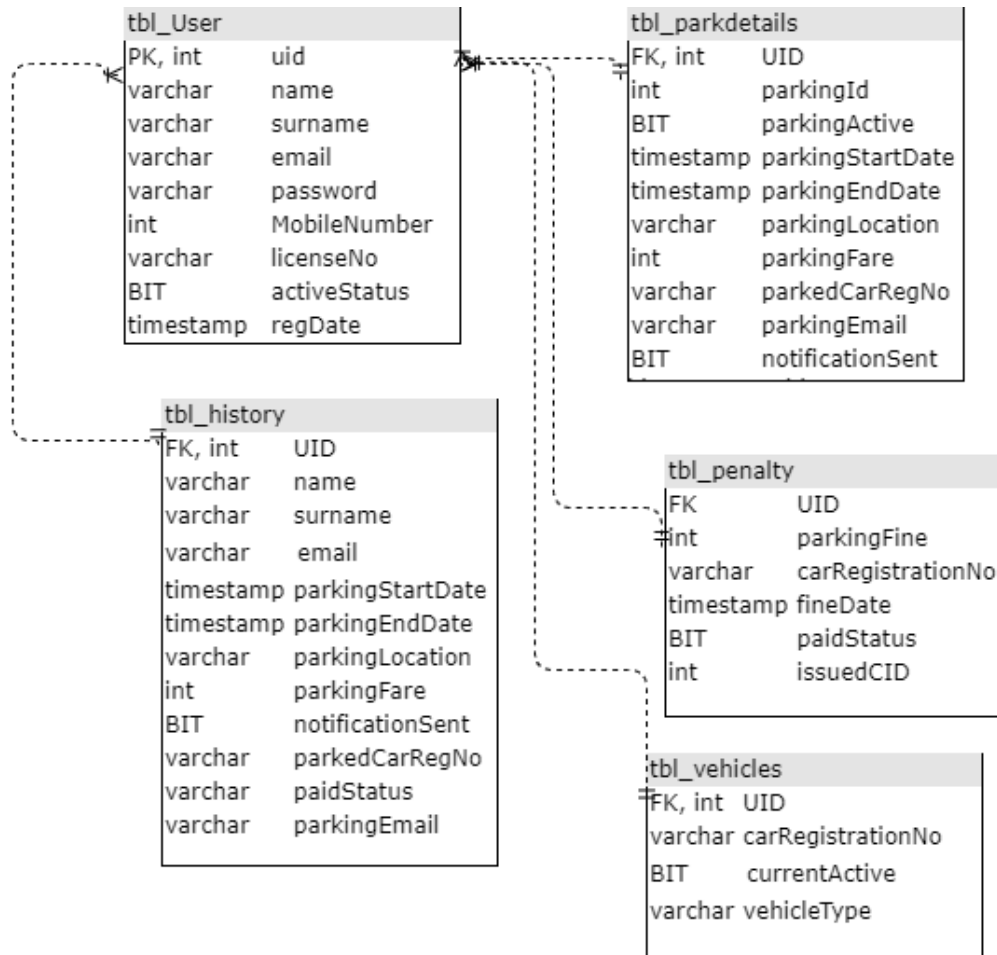


Figure 7: User database

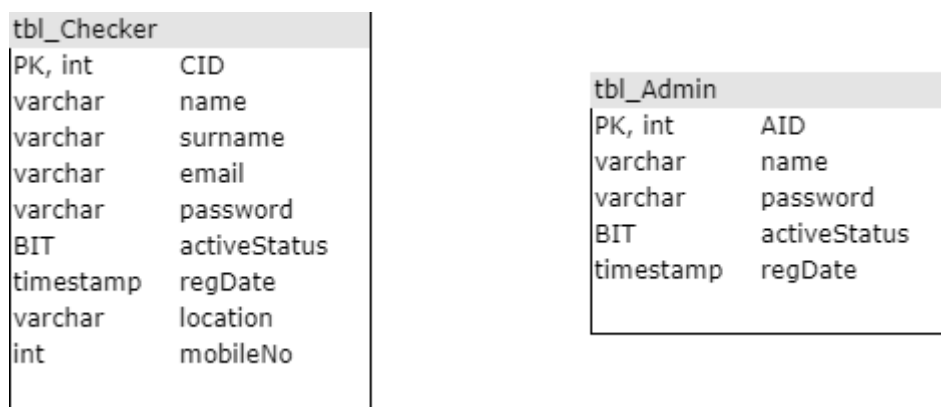


Figure 8: Checker & Admin database

*Parking Database:* The parking database is constructed such that all the customer and parking related data are stored in this database. This database consists of 5 tables, 2 views and 1 stored procedure.

The tables used are tbl\_user, tbl\_parkdetails, tbl\_vehicles, tbl\_history and tbl\_penalty. Table tbl\_user consists of data related to the user. Whenever a new user registers into the application, a new row is added in this table. The parking details are stored in tbl\_parkdetails and customer vehicles are stored in tbl\_vehicles. The history of parking is recorded in table tbl\_history and the parking fines are stored in table tbl\_penalty.

The views used are uv\_getparkdetails and uv\_totalfare. The stored procedure sp\_updateTransactions is used to update the history table tbl\_history.

*Checker Database:* The checker database consists of table tbl\_checkers for storing data related to the user checker.

*Admin Database:* The admin database is used to store data related to admin in table tbl\_admin.

## 6. Data flow Diagram

Figure 9 depicts the entire data flow of the application.

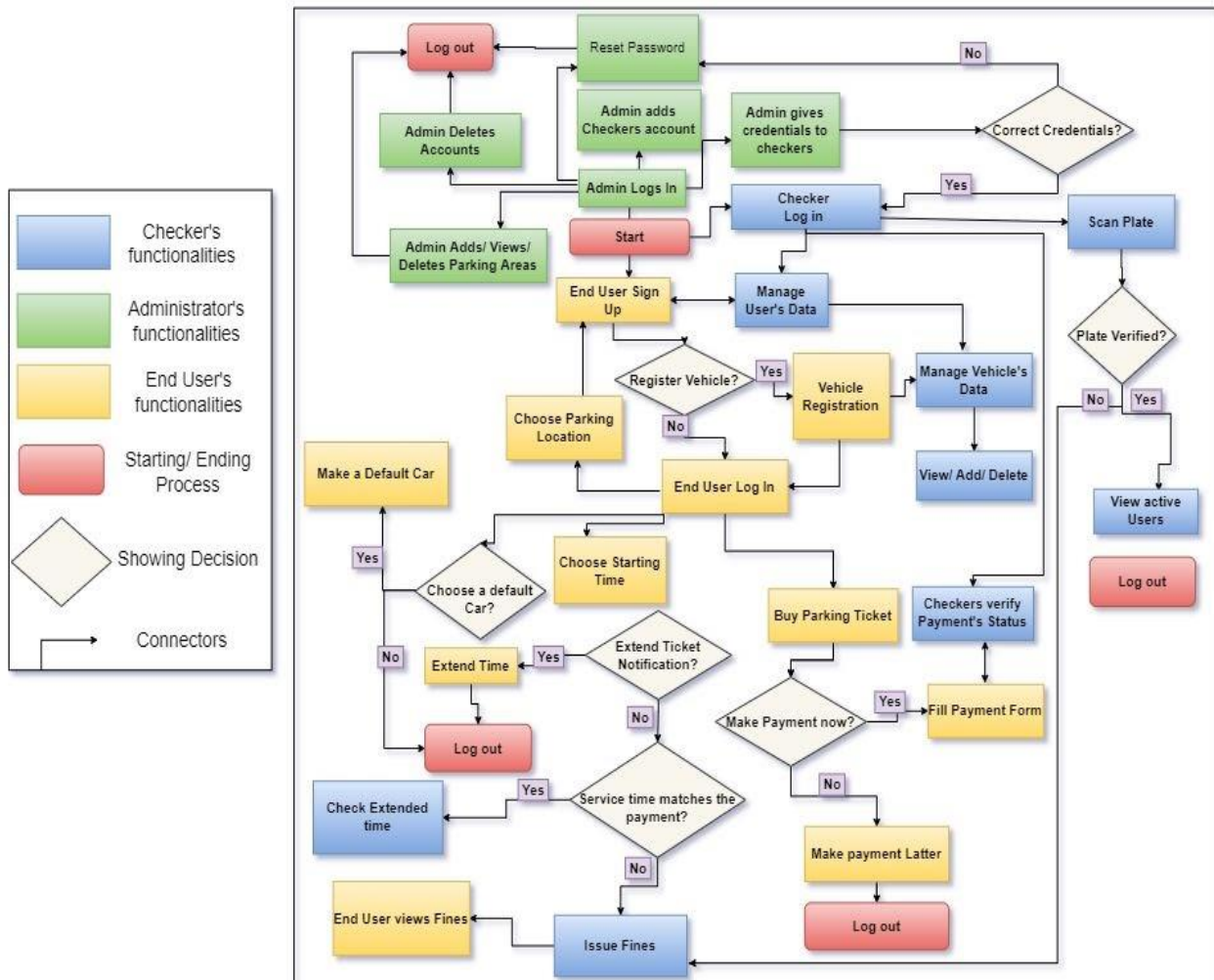


Figure 9: Data Flow Diagram

## **7. Technologies Used**

### **7.1 Front-end Development**

#### **7.1.1 React**

React is a standard, successful JavaScript library for creating simple, interactive, attractive, and even complex user interfaces with much ease. React is designed with a simple view and component-based logic for passing the data throughout the application instead of templates. React is working using the node libraries and runs on a node server [10]. Necessary libraries of React can be installed by using the npm tool (node package manager) [11]. Quick-E-Park contains numerous encapsulated components maintaining their state, which requires re-rendering the elements when the data changes, which can be done effectively through React. Quick-E-Park should be very responsive since it is a web app that can be accessed from both mobiles and laptops of different screen sizes. Creating responsive screens requires designing distinctive styling, but such screens can be developed efficiently using React theme, which we can apply to all the components [11].

#### **7.1.2 Axios**

To access the backend APIs, React provides a component called Axios with child function call-back. Axios contains features such as error, response, loading, async, await while accessing the APIs, along with call-back props such as onSuccess, onError, and OnLoading [12] [13]. Quick-E-Park web application requires consuming and displaying several APIs fetched from the backend, which is accomplished through the Axios.

#### **7.1.3 React-html5-camera-photo**

In React, the node package manager provides a package called react-html5-camera-photo, which is a beneficial and straightforward component for utilizing the device's camera services for an application. This react-html5-camera-photo component provides several inbuilt functions such as 'onCameraStart', 'onCameraStop', 'onCameraError', 'onTakePhoto', 'onTakePhotoAnimationDone', 'imageType' etc., which makes the development smooth [14]. This component will work only after seeking permission from the user. This component also provides image encoding using base64 encoder, making it easy to bind with the backend service API.

### **7.2 Back-end Development**

#### **7.2.1 Python Flask Framework**

Flask is a trending micro web framework in Python language. The flask framework design becomes popular because it eliminates the dependency on the monolithic architecture and structure used in Django Projects. Flask communicates utilizing the web server gateway interface called WSGI. The

web server interface helps with dynamic development and testing. Python provides various libraries such as Flask\_Restful, flask-sqlalchemy, psycopg2, etc., making back-end APIs and database setup like MYSQL, Postgres is much more comfortable than the other back-end frameworks. Python flask framework design is very lightweight, easy to use, and extend [15].

## **7.3 Database**

### **7.3.1 Postgres**

PostgreSQL is an open-source database management system [16]. There are many reasons for PostgreSQL to be chosen over other open-source database systems. The reason behind choosing PostgreSQL is that it has a large community of support and it's now owned by a single entity or company [16]. Thus, it is easier to debug errors and find support online. Also, the Postgres image is readily available for docker enabling easy deployment.

## **7.4 Docker**

Docker is a software platform based on containerization technology [17]. Docker is beneficial for developers in building and shipping applications or microservices or databases as containers. Containers are a standardized software unit that empowers developers to detach the application or microservice from the host environment and let them have their own environment [17]. In simplified terms, containerizing an application using docker makes the application runnable on any machine. Docker is a lightweight, user-friendly software that strengthens the various domains of application development [17]. Docker affords developers a remarkable opportunity to innovate their selection of application stacks, software tools, and deployment environments for every project [17]. Docker's workflow is based on a simple command-line interface that comes with various commands [18]. Using the docker commands, and 'dockerfile,' 'docker-compose file,' developers can build the applications, microservices, and databases as images and make them executable containers [17] [19].

In the Quick-E-Park application, docker is used for the deployment. In this application, docker is used to containerize the Nginx reverse proxy API gateway, react based user interfaces, python back-end web services, and Postgres database as separate containers.

## **7.5 Nginx**

Nginx is open-source software that contributes multiple services such as reverse proxying (as an API gateway), load balancing, HTTP caching, media streaming, and most significantly, works as a web server [20] [21]. Nginx is a complete package supporting various modernized web elements such as HTTP/2, WebSockets, including several media streaming forms such as HLS, HDS, etc [20]. Besides the HTTP server facilities, NGINX also works as a load balancer for HTTP, TCP, and UDP servers

and as a proxy server for email (IMAP, POP3, and SMTP) [20] [21]. Nginx will be appropriate for applications with more end-users since it aids in managing the massive volume of incoming connections and distributes it to slower upstream servers, including everything from microservices to database servers [20].

In this Quick-E-Park application, Nginx is used as the HTTP web server and reverse proxying the back-end web services. Since the application will be a parking application, Nginx can also be used as a load balancer in further development.

## 8. Functional Overview

### 8.1 Functionalities of User

The Quick-E-Park provides various functionalities for the user. Initially, the users must register into the web application before buying a parking ticket online. The Registration page is only for the users. After successful registration, the user must Sign-In with the registered email and password. After a successful login, the user can buy the parking ticket by mentioning the parking location, choosing the vehicle from their vehicle list, and selecting the parking duration. The user can view the active parking ticket with all the details mentioned above. This parking system enables the user to extend the parking ticket before the ticket expiry if necessary. The user receives email notifications before the parking ticket expires. This system allows the user to view their fines, which are pending or which are being paid. With the help of a payment gateway mechanism, the user can pay the fines and pay the amount of parking ticket. This system allows the user to add the new vehicle registration numbers into their vehicle list and view all the vehicles already added from their list. The user can update their profile and reset their password anytime.

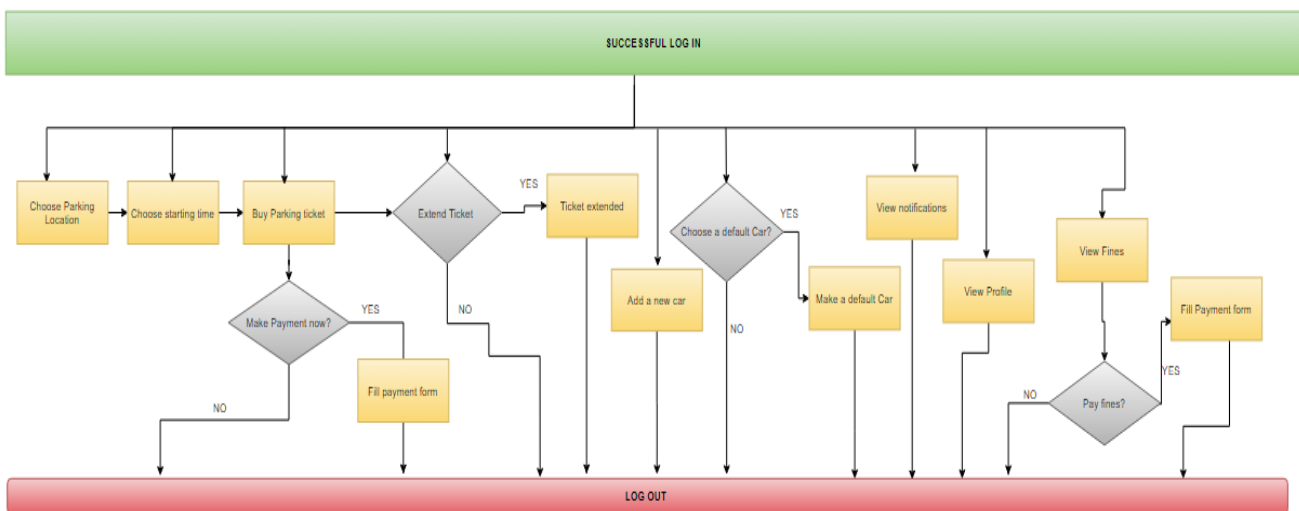


Figure 10: User functionalities

### 8.2 Functionalities of Checker

This Online parking system allows the checkers with different UI and functionalities. Initially, the administrator provides the checkers with the employee id and password for logging into the Quick-E-Park web application. After a successful login, the checker can access several functionalities. The checkers are allocated to specific zones, and they can verify the parking ticket of the vehicles of that particular zone by merely scanning the vehicle registration plate. The checker receives the notifications of the upcoming parking ticket expiry of their specific zones. This system also provides the additional functionality for the checker to issue fines for the user whose ticket got expired and



still the vehicle is in the parking zone. The checker can also view the complete history of issued fines. The checkers can log out of the web application after their working hours.

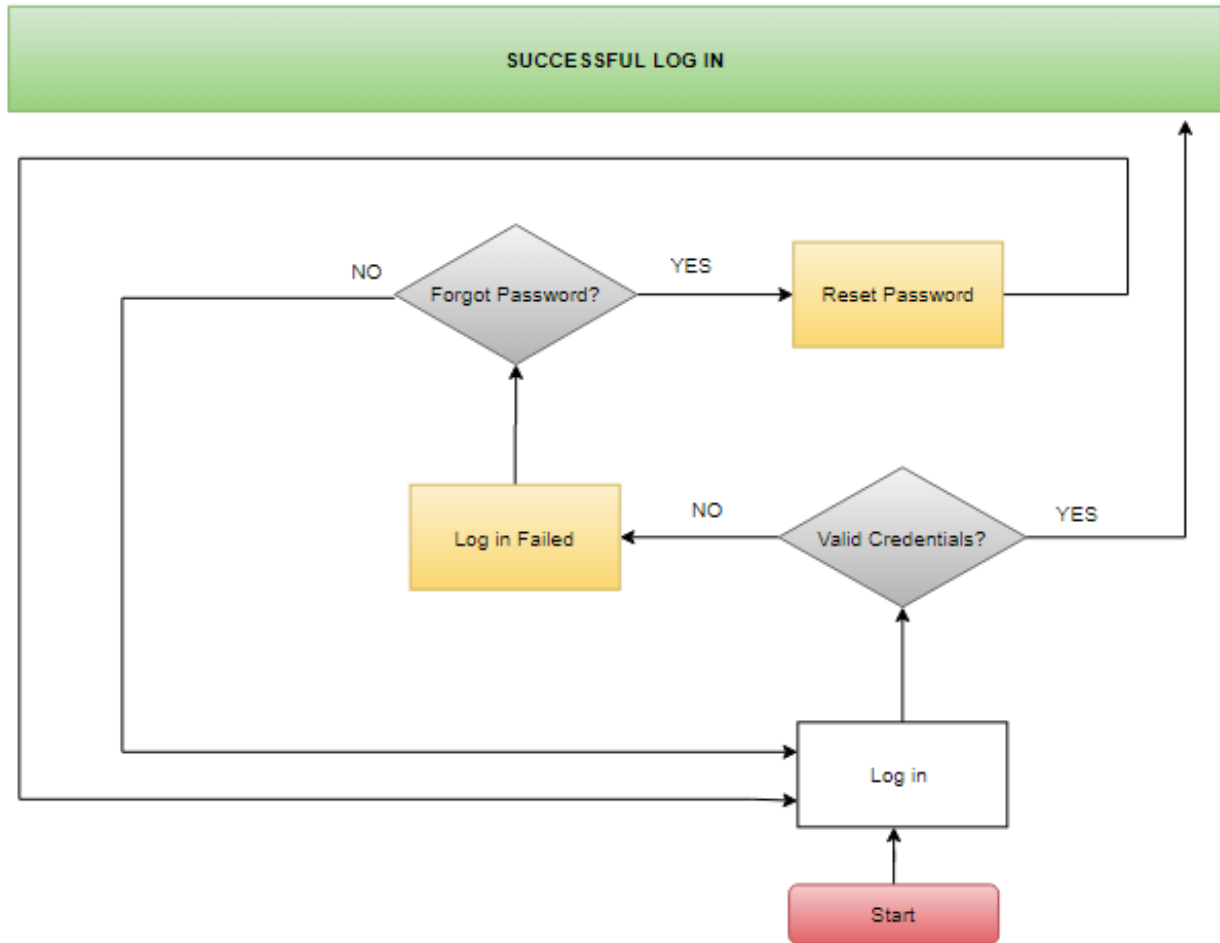


Figure 11: Checker functionalities

### 8.3 Functionalities of Admin

The Administrator of this system has complete access to the web application. The Administrator can directly login into the web application without any initial registration. They can view all the registered users, the active parking tickets and update the user details anytime. The Administrator is responsible for adding and removing the parking areas on the web application. The Administrator has complete control over the management of employees, such as adding the employee, removing an employee, and updating the employee's details. The Administrator is also responsible for providing the web application's access details for the employee and assigning them to specific zones.

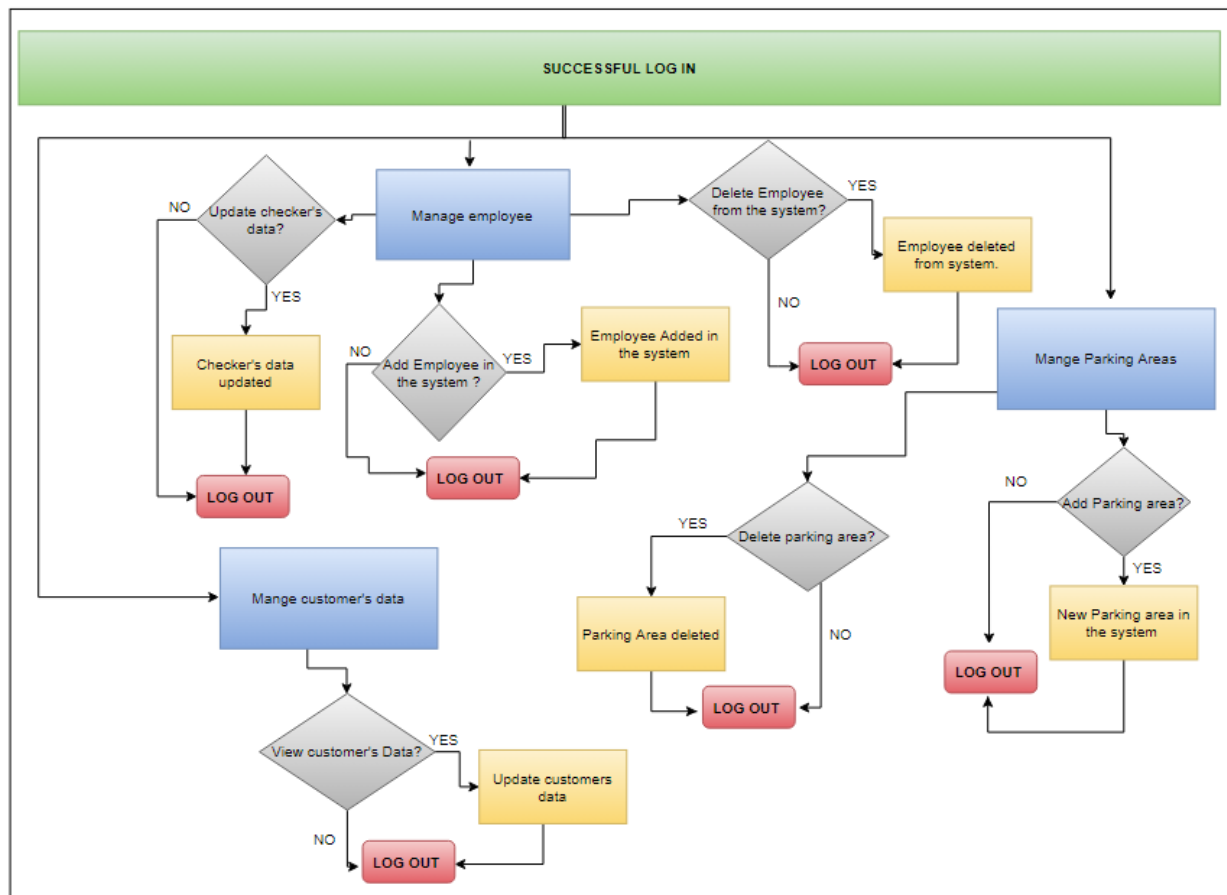


Figure 12: Admin functionalities

## 9. Implementation

The user interfaces and the backend web services were independently developed using their respective React library and Python flask web framework. The two interfaces were then integrated using the Nginx web server and deployed using docker containerization. The application is accessed using the Quick-E-Park URL post-deployment. The following content elaborates on how the front-end user interfaces and backend web services were implemented for different modules during the development process.

### 9.1 User Functionalities

The user module is the independent module that allows the user to perform various functionalities related to the end-users (customers) using the Quick-E-Park web application.

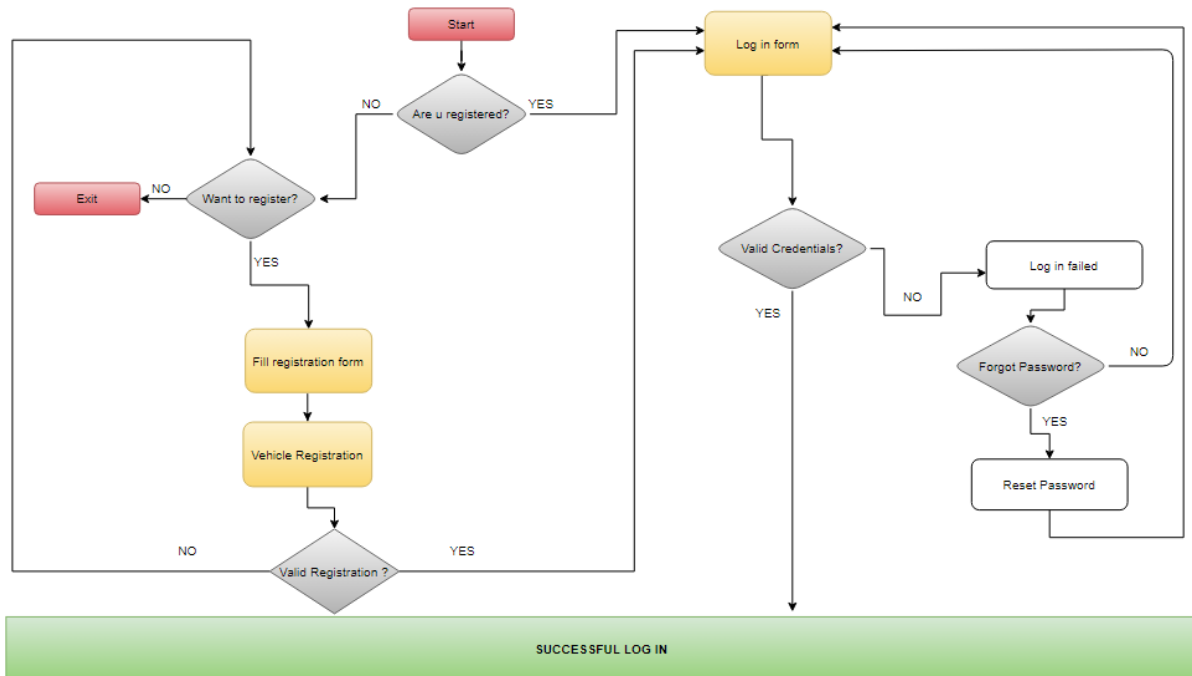


Figure 13: End-User Successful Login

#### 9.1.1 Registration

The customers are the users who would use the Quick-E-Park web app for purchasing the parking ticket online. Firstly, the user has to register into the web application to use the web app's functionalities; for this, registration REST API is called. The user should provide the following details for successful registration, first name, last name, email, password, mobile number, and license number.

#### User Interface:

In the case of the new user, a signup page link is provided on the login page as mentioned in figure 16. On click of the signup, a registration page will be displayed, as mentioned in figure 14.

The registration page contains two forms. The first form contains fields such as first name, last name, email address, mobile number, password, and confirm password as mentioned in figure 14. The second form contains the mandatory field for the license number, which will be validated, whether it is a valid license number or not as shown in figure 15. A new registration will be successful only with a valid license number and not already registered with the application. On successful registration, the user will be redirected to the login page.

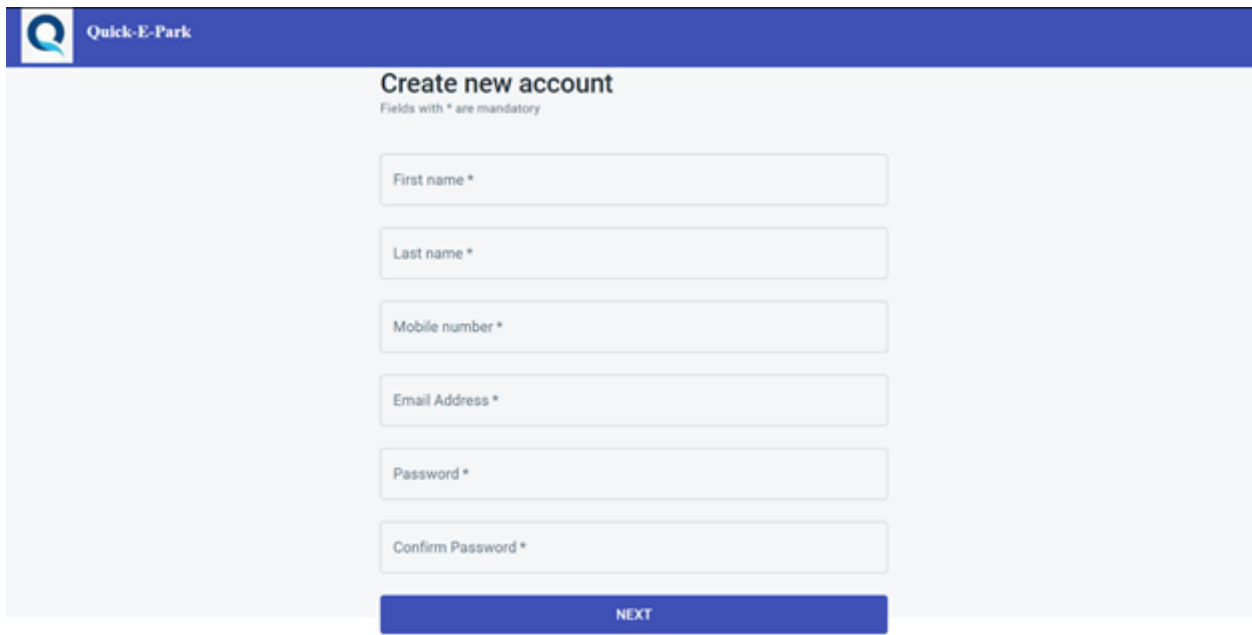


Figure 14: User Registration Page 1

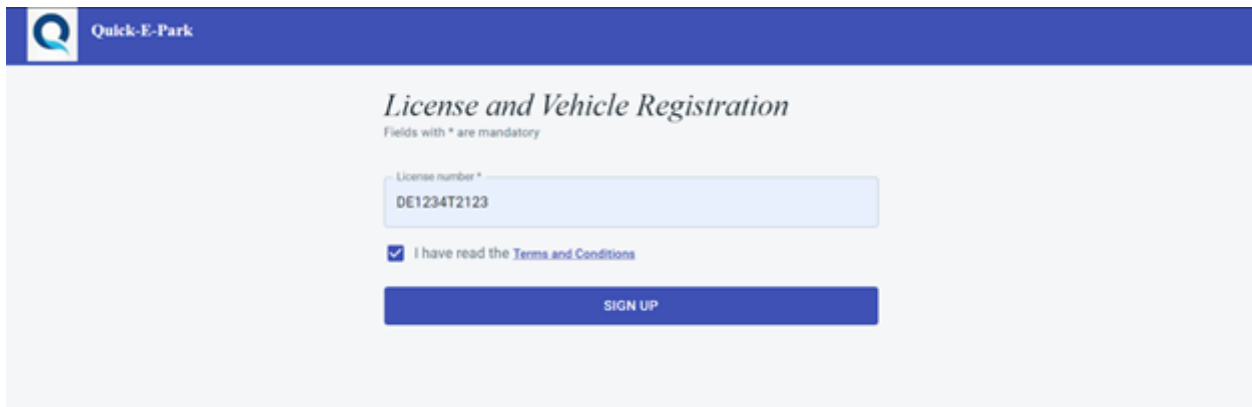


Figure 15: User Registration Page 2

After filling out the registration forms, using Axios, the back-end add new registration web service is called on click of the sign-up page. Based on the response received from the registration web service, either the success message or error message will be displayed in the user interface.

### Back-end Web Service:

Request type: POST

Request URL: `/addNewRegistration`

Request Structure:

```
{
  "firstName": <String>,
  "lastName": <String>,
  "email": <String>,
  "password": <String>,
  "mobileNumber": <String>,
  "licenseNumber": <String>
}
```

Success Request Response:

```
{
  "data": "Registered Successfully.",
  "statusCode": "200",
  "statusDesc": "Registration Successful.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

Failure Request Response:

```
{
  "data": "EmailID or License number already exists.",
  "statusCode": "200",
  "statusDesc": "Registration Unsuccessful, license number or email already exists.",
  "statusMsg": "Error",
  "statusType": 1
}
```

*Validations covered:*

1. The connection of the Database should be active.
2. Email or License number should be unique and does not exist in the database already.
3. Password is encrypted and stored in the database.
4. The length of the license number is validated.

*Actions in the system:*

This API allows the user to create a new entry in the `tbl_users` of the Quick-E-Park database. It checks if the email and license are unique or not, if it already exists in the table, this API returns an error response.

### 9.1.2 Login

After successfully registering on the Quick-E-Park web application, the user can log in by providing the registered email and password. The web application authenticates the email and password and then allows the user to access the application's various functionalities.

## User Interface:

The login page is as shown in figure 16. Users can enter their registered email and password and login into the application.

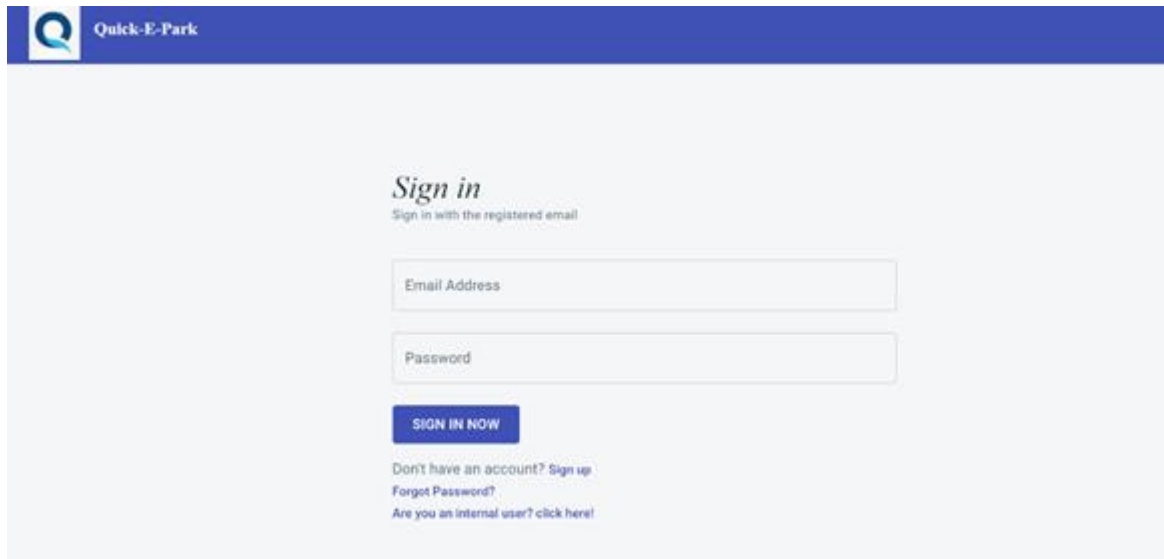


Figure 16: User Login Page

In the case of a successful login, the user will see the dashboard page. On the left, the user will have various options such as parking ticket, ticket history, view fines, account, payment, settings, logout as mentioned in figure 17.



Figure 17: User Dashboard Page

After entering the registered email id and password, using the Axios, the back-end login web service will be called on the click of the sign-in button. Based on the response received from the login web service, the user will be re-routed to the dashboard page in case of successful authentication, or an error message will be displayed.

**Back-end Web Service:**

Request Type: POST

Request URL: /loginvalid

Request Structure:

```
{
  "email":<String>,
  "password":<String>
}
```

Success Request-Response:

```
{
  "data": {
    "authentication": "true",
    "email": "sheharaz207@gmail.com",
    "firstName": "Sheharaz2",
    "lastName": "sheik",
    "licenseNumber": "1234",
    "mobileNumber": "None"
  },
  "statusCode": "200",
  "statusDesc": "Login Successful.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

Failure Request-Response:

```
{
  "data": "-1",
  "statusCode": "201",
  "statusDesc": "Input credentials incorrect",
  "statusMsg": "Error",
  "statusType": 1
}
```

*Validations covered:*

1. The database connection should be active.
2. Validation of email and password with the database.

*Actions in the system:*

This API allows the user to login into the web application. Initially, this API validates the email and password entered by the user with the user entries which are already existing in the tbl\_user table of Quick-E-Park database. If the user details exist in the database then it allows the user for successful login, otherwise this API returns an error response to the user.

**9.1.3 Buy Parking Ticket**

The Quick-E-Park web application enables the user to buy the parking ticket in the parking ticket section after successfully logging into the application. The user has to choose the desired parking

location, vehicle registration number from his/her vehicle list, and duration of the parking time from the drop-down list to park his/her vehicle for purchasing the parking ticket. And the user has an additional option for receiving the parking ticket notifications to an alternate email by clicking the checkbox, enabling the user to enter the alternate email id. Otherwise, the user gets parking ticket notifications to the registered email id. After submitting the above details, the user receives the parking ticket.

### User Interface:

On click of the parking ticket tab from the sidebar, the user can view two modules. One is to show currently active parking tickets, and another module is to buy a new parking ticket, as shown in the figure 18. The buy parking ticket module on the right side of figure 18 contains a basic form with input fields such as parking location, option to choose the priorly added vehicle number, parking duration, and an additional field for mentioning the specific email for the particular parking ticket notification. On the successful purchase of a parking ticket, a success message will be displayed, and the active parking ticket module is refreshed to view the active tickets.

The screenshot displays the Quick-E-Park web application. On the left is a sidebar with a user profile 'Thirumal' and navigation links: Home, Parking Ticket (selected), Ticket History, Account, Vehicles, Payment, Fines, Settings, and Logout. The main content area is divided into two sections. The top section, 'Active Parking Ticket 1', shows details for a ticket at Kassel 34127, vehicle DE 1234, starting at 15:35:00, with a 1-hour duration and a 1 Euro fare. It includes an 'EXTEND TICKET' button and a 'CONFIRM' button. The bottom section, 'Active Parking Ticket 2', shows details for a ticket at Kassel 34167, vehicle FH 3251. The right section, 'Buy New Parking Ticket', contains a form with fields for 'Parking Location \*', 'Choose vehicle' (a dropdown menu), 'Parking Duration' (a dropdown menu), and an optional checkbox for 'Use logged in user email for notifications' with a corresponding 'Email Address for Notification' field. A 'BUY TICKET' button is at the bottom of the form.

Figure 18: Parking Ticket Tab - Buy Ticket and View Ticket

After filling the required input fields, using Axios, the buy ticket web service is called on click of the buy ticket button. Based on the response received from the buy ticket web service, a success alert or error alert will be displayed.

### Back-end Web Service:

Request Type: POST



Request URL: **/buyticket**

Request Structure:

```
{
  "email":<String>,
  "parkedCarRegNo":<String>,
  "parkingDuration":<int>,
  "parkedLocation":<String>,
  "parkingEmail":<String>
}
```

Success Request-Response:

```
Output:
{
  "data": {
    "parkingFare": "0.083333333333333333 EUR"
  },
  "statusCode": "200",
  "statusDesc": "Parking Successful",
  "statusMsg": "Success.",
  "statusType": 0
}
```

*Validations covered:*

1. The database connection should be active.
2. The user shall choose the vehicle registration number only from his/her vehicle list.
3. The users cannot buy the multiple parking tickets for the same vehicle registration number for the same duration of time.
4. The user shall enter alternate email id for receiving the active ticket parking notifications.

*Actions in the system:*

This API enables the user to buy the parking ticket online on the Quick-E-Park web application. When the user buys the parking ticket, this API creates a new entry in the tbl\_parkdetails table of Quick-E-Park database. Initially before creating a new entry, this API checks whether the selected vehicle exists in tbl\_parkdetails table. If the selected vehicle already exists, this API returns an error response and the user cannot buy the parking ticket for the same vehicle which already has an active parking ticket. If the vehicle is not present in the tbl\_parkdetails, it creates a new entry in the table and then this API returns a success message with the fare amount and ticket.

#### **9.1.4 View Active Ticket - User:**

The active parking tickets bought through Quick-E-Park web application can be viewed in the parking ticket section of the application. Once the user buys a parking ticket, all the active tickets get updated in the webpage. This page displays details of parking including parking location, parking start and remaining time, parking fare, notification email and parked vehicle number. Thus, the users can verify the parking details after buying the parking ticket.

**User Interface:**

On click of the active parking ticket tab from the dashboard's sidebar as shown in figure 18, "viewTicketUser" back-end web service will be called. The component will display if the response has active tickets; else will show no active tickets dialog. The active parking ticket module on the left of figure xx will show the active tickets. This active parking ticket module displays all the details such as parking location, parking start time, parking remaining time, notification email for the particular ticket, parked vehicle number, and parking fare incurred.

**Back-end Web Service:**

Request Type: POST

Request URL: `/viewTicketUser`

Request Structure:

```
{
  "email":<String>,
}
```

**Success Request-Response:**

Output:

```
{
  "data": [
    {
      "parkedCarRegNo": "DE123456",
      "parkingEmail": "sheharaz07@gmail.com",
      "parkingFare": "0.5 EUR ",
      "parkingLocation": "Duisburg",
      "parkingStartDate": "2021-01-10 16:40:19.585693",
      "remainingParkingDuration": "0 Hours 12 minutes"
    }
  ],
  "statusCode": "200",
  "statusDesc": "GENERIC SUCCESS.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

**Validations covered:**

1. Only active tickets for the user should be shown.

**Actions in the system:**

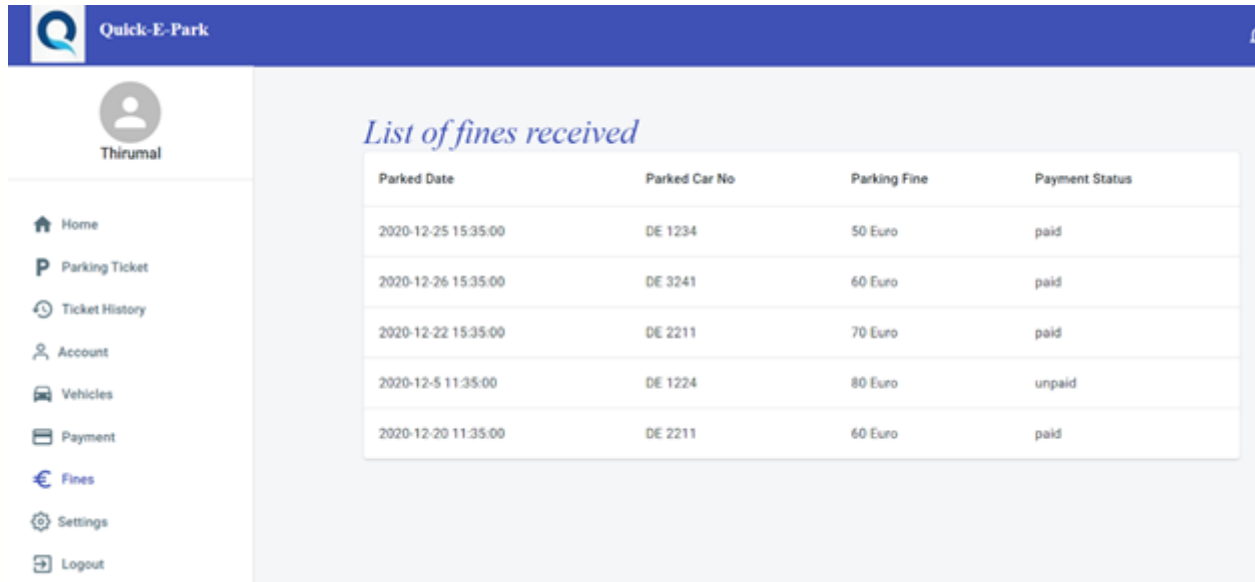
This API enables the user to view the active parking ticket online on the Quick-E-Park web application. This API checks the view uv\_getparkdetails for all the active parking tickets corresponding to the logged in user. If no active parking exists for the user in the view, then the API returns a response that no parking ticket exists for the user.

### 9.1.5 View Fines User

The users are issued with fines for illegal parking and can be viewed in the parking ticket section of the application.

#### User Interface:

On click of the fines tab, the logged-in user can view the fines that he received as shown in figure 19. These fines are issued by the checker for illegal parking.



Parked Date	Parked Car No	Parking Fine	Payment Status
2020-12-25 15:35:00	DE 1234	50 Euro	paid
2020-12-26 15:35:00	DE 3241	60 Euro	paid
2020-12-22 15:35:00	DE 2211	70 Euro	paid
2020-12-5 11:35:00	DE 1224	80 Euro	unpaid
2020-12-20 11:35:00	DE 2211	60 Euro	paid

Figure 19: View Fines Tab

#### Back-end Web Service:

Request Type: POST

Request URL: `/viewFinesUser`

Request Structure:

```
{
  "email":<String>,
}
```

Success Request-Response:

```
Output:
{
  "data": [
    {
      "carRegistrationNo": " DE YY1235",
      "fineDate": "2020-12-29 10:42:51.398743",
      "paidStatus": "Paid",
      "parkingFine": "1"
    }
  ],
  "statusCode": "200",
  "statusDesc": "GENERIC SUCCESS.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

*Validations covered:*

1. Only active tickets for the user should be shown.

*Actions in the system:*

This API enables the user to view the active parking ticket online on the Quick-E-Park web application. This API checks the view uv\_getparkdetails for all the active parking tickets corresponding to the logged-in user. If no active parking exists for the user in the view, then the API returns a response that no parking ticket exists for the user.

**9.1.6 Extend Ticket**

The Quick-E-Park web application enables the users to extend the active parking tickets. The user can extend the active parking ticket in the parking ticket section of the web application. The extend ticket option is only available if there is an active parking ticket existing for the user. Once the user selects extend ticket option, it allows the user to choose the duration of parking time to be extended and confirm the extension.

**User Interface:**

The parking ticket tab, along with buying parking tickets and viewing parking tickets, also allows the users to extend their already purchased parking ticket with the extend ticket button. This functionality to extend is provided along with the view ticket. The extend ticket button can be found beside each active parking ticket, as shown in figure 18.

After choosing the additional duration to extend the ticket, on click of the confirm extension button, using Axios, the “extendTicket” back-end web service will be called. Based on the successful response from the web service, the active view ticket component will get refreshed. In case of error response, respective error alerts will be displayed in the user interface.

**Back-end Web Service:**

Request Type: POST

Request URL: /extendTicket

Request Structure:

```
{
  "email": "sheharaz207@gmail.com",
  "parkedCarRegNo": "DE YY1235",
  "parkingEmail": "thirumaltce@gmail.com",
  "timeToExtend": 15
}
```

**Success Request-Response:**

```
{
  "data": [
    {
      "parkedCarRegNo": " DE YY1235",
      "parkingEmail": "thirumaltce@gmail.com",
      "parkingFare": "0. 33 EUR ",
      "parkingLocation": "Duisburg",
      "parkingStartDate": "2020-12-31 15:23:51.848346",
      "remainingParkingDuration": "0"
    }
  ],
  "statusCode": "200",
  "statusDesc": "Parking Successful",
  "statusMsg": "Success.",
  "statusType": 0
}
```

**Validations Covered:**

1. The database connection should be active.
2. The parking ticket shall be extended when there already exists an active parking ticket for the user.

**Actions to be done:**

This API allows the user to extend the active parking tickets in the Quick-E-Park web application. Initially this API checks for the active parking tickets existing for the user in the tbl\_parkdetails table of the QuickEpark database. If the user has an active parking ticket in the tbl\_parkdetails table, then this API extends the parking ticket by updating the tbl\_parkdetails table by increasing the remaining time of duration for the user and returns the success response. If there is no active parking ticket for the user, this API does not allow the user to extend the ticket and returns an error response.

**9.1.7 View Payment**

The user can view the remaining balance payment separately, one for the parking tickets which are already bought and the other is for parking fines which are received and it can be viewed in the payment section of the Quick-E-Park web application. The payment page gets updated automatically with the fare and fine, when the user buys the parking ticket and when the user receives the fine from the checkers.

**User Interface:**

On click of the payment tab, the logged-in user can view the pending ticket amount and pending fine amount incurred for the user as shown in figure 20. Subsequently, the same tab is used for clearing these payments.

Figure 20: Payment Tab

During the loading of the payment component, the "getPendingPayment" back-end web service will be called. The module will display the pending ticket amount and pending fine amount based on the service's response.

### Back-end Web Service:

Request Type: POST

Request URL: `/getPendingPayment`

Request Structure:

```
{
  "email":<String>
}
```

Success Request-Response:

```
{
  "data": {
    "fineAmount": "1 EUR",
    "ticketAmount": "0 EUR"
  },
  "statusCode": "200",
  "statusDesc": "GENERIC SUCCESS.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

Validations covered:

1. The payment page shall display only the total fine amount and ticket amount which are not paid by the user.
2. The database connection should be active.

#### *Actions to be done:*

This API allows the users to view their remaining balance payment amount for the fines and parking tickets in the payment section of Quick-E-Park web application. This API initially checks the total fines for the user in the tbl\_penalty table and total amount for the parking tickets in the uv\_totalfare table of Quick-E-Park database. If there exists an entry in the tbl\_penalty table, it displays the total fine amount to the user, else it displays a pending fine amount as zero in the payment page. If there exists an entry in the uv\_totalfare table, it displays the total amount for parking tickets in the payment page of the web application, else it displays the parking ticket amount as zero.

### 9.1.8 Clear Payment

The user can clear the pending amount for the parking tickets that are already bought and the fines received from the checkers. The user can clear payment in the payment section of the Quick-E-Park web application. The user can clear the pending amount for total parking fares and fines separately.

#### **User Interface:**

The payment tab is used to make the payment for both parking fares and parking fines. The payment portal will have an option to select the payment method and pending amount type to clear the pending bills of the account as mentioned in the figure 21.

Figure 21: Payment Portal Module

After providing the inputs, using Axios, the "clearPayment" back-end web service will be called on click of the pay now button. Based on the service's response, respective success or error alerts will be displayed.

### Back-end Web Service:

Request Type: POST

Request URL: /clearPayment

Request Structure for clearing payment for parking fare:

```
{
  "email":<String>,
  "pendingAmountType":"Fare"
}
```

Request Structure for clearing payment for parking fines:

```
{
  "email":<String>,
  "pendingAmountType":"Fine"
}
```

Success Request-Response:

```
{
  "data": [],
  "statusCode": "200",
  "statusDesc": "GENERIC SUCCESS.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

*Validations Covered:*

1. The user shall make the payment for fares and fines separately.

*Actions to be done:*

This API allows the user to clear the pending balance amount for the parking tickets and parking fines. Initially this API checks the pending amount type whether it is parking fare or parking fine. If it is parking fare, then this API updates the payment amount in the tbl\_parkdetails table of QuickEPark database and displays the success response as payment completed for the parking ticket to the user. If the pending ticket amount type is parking fine, then this API updates the payment amount in the the tbl\_penalty of Quick-E-Park database and displays the success response as payment completed for the fines to the user.

### 9.1.9 View Ticket History

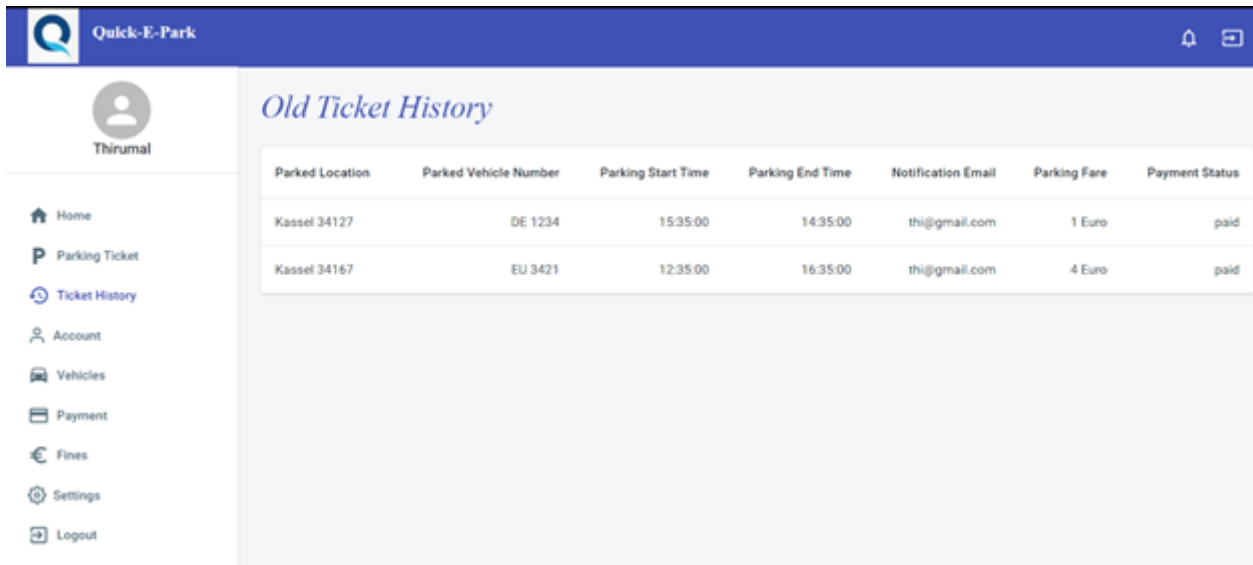
The user can view the complete history of the parking tickets which are already bought using the Quick-E-Park web application in the View Ticket History page. The view ticket history displays the complete details of the parking tickets which includes first name, last name, email, parking start date,



parking end date, parking location, parking fare, vehicle registration number and payment status for all the tickets which are bought by the user till date.

### User Interface:

On click of the ticket history from the sidebar of the application, the user can view their old expired parking tickets with all the details including the payment status of each ticket as mentioned in the figure 22.



Parked Location	Parked Vehicle Number	Parking Start Time	Parking End Time	Notification Email	Parking Fare	Payment Status
Kassel 34127	DE 1234	15:35:00	14:35:00	thi@gmail.com	1 Euro	paid
Kassel 34167	EU 3421	12:35:00	16:35:00	thi@gmail.com	4 Euro	paid

Figure 22: Ticket History Tab

On click of the ticket history tab from the dashboard page's sidebar, the "getTicketHistory" back-end web service will be called. Based on the web service response, the ticket history list will be displayed.

### Back-end Web Service:

Request Type: POST

Request URL: `/getTicketHistory`

Request Structure:

```
{
  "email": <String>
}
```

Success Request-Response:

```
{
  "data": [
    {
      "email": "test",
      "name": "test",
      "parkedCarRegNo": "test",
      "parkingEmail": "test",
      "parkingEndDate": "2020-12-30 18:43:51.963812",
      "parkingFare": "10",

```

```

        "parkingLocation": "test",
        "parkingStartDate": "2020-12-30 18:43:51.963812",
        "paymentStatus": "1"
    },
    {
        "email": "sheharaz207@gmail.com",
        "name": "Sheharaz2",
        "parkedCarRegNo": "DE56754",
        "parkingEmail": "sheharaz207@gmail.com",
        "parkingEndDate": "2020-12-30 21:07:23.278144",
        "parkingFare": "3",
        "parkingLocation": "Kassel",
        "parkingStartDate": "2020-12-30 18:07:23.278144",
        "paymentStatus": "1"
    }
],
"statusCode": "200",
"statusDesc": "GENERIC SUCCESS.",
"statusMsg": "Success.",
"statusType": 0
}

```

#### *Validations checked:*

1. The database connection should be active.
2. The view ticket history page shall display all the tickets which are already bought by the user along with payment status.

#### *Actions to be done:*

This API allows the user to view the complete history of tickets which are already bought using Quick-E-Park web application. This API displays all the entries of the user from the tbl\_history table of Quick-E-Park database based on the user email.

### **9.1.10 Add Vehicles**

The user can access the Vehicles page after logging into the Quick-E-Park web application. The user can add the new vehicle registration number into his/her vehicle list and choose the vehicle type as the owner or temporary in the vehicle page of the application.

#### **User Interface:**

On click of the vehicles tab, the users can access the add vehicle functionality of the application as shown in figure 23.

After providing the vehicle details, the customer can add the vehicles to their profile by clicking the add vehicle button which in turn calls the “addVehicleData” back-end web service using Axios. Based on the service response, success or error messages will be displayed.

The screenshot shows the 'Quick-E-Park' web application interface. On the left is a sidebar menu with options: Home, Parking Ticket, Ticket History, Account, Vehicles (selected), Payment, Fines, Settings, and Logout. The main content area is titled 'List of Vehicles in the Profile' and contains a table with two columns: 'Vehicle License Plate Number' and 'Vehicle Type'. The table lists two vehicles: one with license plate 'DE1235' of type 'Owner', and another with license plate 'DE56754' of type 'Temporary'. Below the table is a section titled 'Adding New Vehicle to Profile' with a note 'Fields with \* are mandatory'. It includes a text input field for 'Vehicle reg number \*', a checkbox for 'Are you the owner of the vehicle?', and an 'ADD VEHICLE' button.

Vehicle License Plate Number	Vehicle Type
DE1235	Owner
DE56754	Temporary

**Adding New Vehicle to Profile**  
Fields with \* are mandatory

Vehicle reg number \*

☐ Are you the owner of the vehicle?

ADD VEHICLE

Figure 23: Vehicles Tab

**Back-end Web Service:**

Request Type: POST

Request URL: /addVehicleData

Request Structure:

```
{
  "carRegNumber":<String>,
  "vehicleType":<String>,
  "email":<String>
}
```

**Success Request-Response:**

```
{
  "data": {
    "registrationMsg": "Vehicle registered successfully"
  },
  "statusCode": "200",
  "statusDesc": "New Car Registration number added successfully.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

**Validations Covered:**

1. The Database connection should be active.
2. The user vehicle type is validated. If the owner for the vehicle registration number already exists in the table, the other users cannot add the same vehicle registration number as the owner in his/her vehicle list rather the users can add the vehicle number as type temporary.
3. The same user cannot add the same vehicle registration number with different vehicle types either owner or temporary if the vehicle number already exists in the table.

### *Actions in the system:*

This API allows the user to add a new vehicle into the tbl\_vehicle table name of the quickepark database. Initially, this API checks if the vehicle number is already present in the table or not. If the vehicle is present, then it checks for the vehicle type of the user either owner or temporary. If the owner already exists, the other user cannot add the same vehicle to his list as the owner but they can add as the temporary. The UID is the foreign key for the tbl\_vehicles table and is the primary key for the tbl\_users table which allows the user to add the vehicles in the tbl\_vehicles table with the user email.

#### **9.1.11 Show Vehicles**

The user can view all the vehicles and vehicle types that are added to their vehicle list by clicking on the show vehicles button in the vehicles page of the Quick-E-Park web application.

#### **User Interface:**

On click of the vehicles tab, the users can access the view added vehicle functionality of the application as shown in the figure 23.

#### **Back-end Web Service:**

Request Type: POST

Request URL: /showVehicleData

Request Structure:

```
{
    "email": <String>
}
```

#### **Success Request-Response:**

```
{
  "data": [
    {
      "carRegNumber": "TJ2311111",
      "vehicleType": "owner"
    },
    {
      "carRegNumber": "TJ999991",
      "vehicleType": "temporary"
    },
    {
      "carRegNumber": "TJ999992",
      "vehicleType": "owner"
    }
  ],
  "statusCode": "200",
  "statusDesc": "Data fetched successfully from PostgreSQL table.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

*Validation covered:*

1. The database connection should be active.

*Actions in the system:*

This API allows the user to display all the vehicles which are already added into the tbl\_vehilces table from the quickepark database. With UID as the foreign key in the tbl\_vehicles table and as the primary key in the tbl\_users, it will fetch all the vehicles which are added to the user email.

### 9.1.12 Update Password

The Quick-E-Park web application allows the users to update the login password anytime. The update password can be done in the profile page of the web application. The user has to enter the current password and new password for successfully updating the new password.

#### User Interface:

The user also has the option to view and update his profile by clicking the account tab from the sidebar. The account tab provides functionalities like updating the password, updating the user details which is shown in the figure 24.

*Figure 24: Account Profile Tab*

After providing the password and confirm password values, on click of the update password button, the “updatePassword” back-end web service will be called using Axios from the react component. Based on the response, a success or error message will be displayed in the user interface.

#### Back-end Web Service:

Request Type: POST

Request URL: `/updatePassword`

Request Structure:

```
{
  "email": "rd@gmail.com",
  "password": "@12111",
  "updatePassword": "22222"
}
```

Success Request-Response:

```
{
  "data": {
    "Msg": "Password updated successfully."
  },
  "statusCode": "200",
  "statusDesc": "Updated data successfully in Database.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

Failure Request-Response:

```
{
  "data": "-1",
  "statusCode": "201",
  "statusDesc": "Updating password failed.",
  "statusMsg": "Error",
  "statusType": 1
}
```

*Validations Covered:*

1. Checks for the current password match.
2. Checks the minimum character length for the new password.

*Actions to be done:*

This API allows the users to update the login password in the `tbl_users` table of QuickEPark database. Initially this API checks for the match of user current password with the password which is already existing in the `tbl_users` table. If the user current password matches, then this API updates the new login password in the `tbl_users` table and displays the success response as the password successfully updated to the user, else it displays the failure response as the updated password failed to the user.

### 9.1.13 Update Profile

Every user can access his/her profile page after successful login. The profile page consists of the first name, last name, email, and mobile number of the user. The user can update his/her details in the profile page anytime.

**User Interface:**

The user can update the user details by selecting the account tab as shown in the figure 24.

### Back-end Web Service:

Request Type: POST

Request URL: `/updateProfileData`

Request Structure:

```
{
  "firstName": <String>,
  "lastName": <String>,
  "email": <String>,
  "mobileNumber": <String>
}
```

Success Request-Response:

```
{
  "data": "Data Updated Successfully.",
  "statusCode": "200",
  "statusDesc": "Updated data successfully in Database.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

Validations Covered:

1. The Database connection got disconnected.

*Actions in the system:*

This API updates the user details in the `tbl_users` table of the Quick-E-Park database. This API allows the user to update all the fields at once, and also the user can update the individual field.

## 9.2 Checker Functionalities

The checker's module is also independent. The primary purpose of this module is to perform the functionalities of the checker's responsibilities.

### 9.2.1 Registration of Checker

The checker has to be registered in the Quick-E-Park system before using the application. Keeping the special privileges of checkers in mind and avoiding misuse of the application, the checkers are not allowed to register using the application's user interface. Instead, the checker's registration is done by the admin users of Quick-E-Park, who are going to be affiliated with the city council. The city council with admin privileges can add and remove checker accounts. The checker accounts are associated with a unique Employee Id and temporary Password directly shared with the respective checker. Other details like first name, last name, date of birth, email, and likewise are also filled by the admin while adding checker.

### 9.2.2 Checker Login

The checkers are the “Ordnungsamt” who check the legality of parking tickets associated with the vehicles. The checkers can login into Quick-E-Park web application by using employee id and password issued by the admin.

#### User Interface:

The checker can login into the application using the internal user’s login page as shown in figure 25. The checker can log in using their employee id and password.

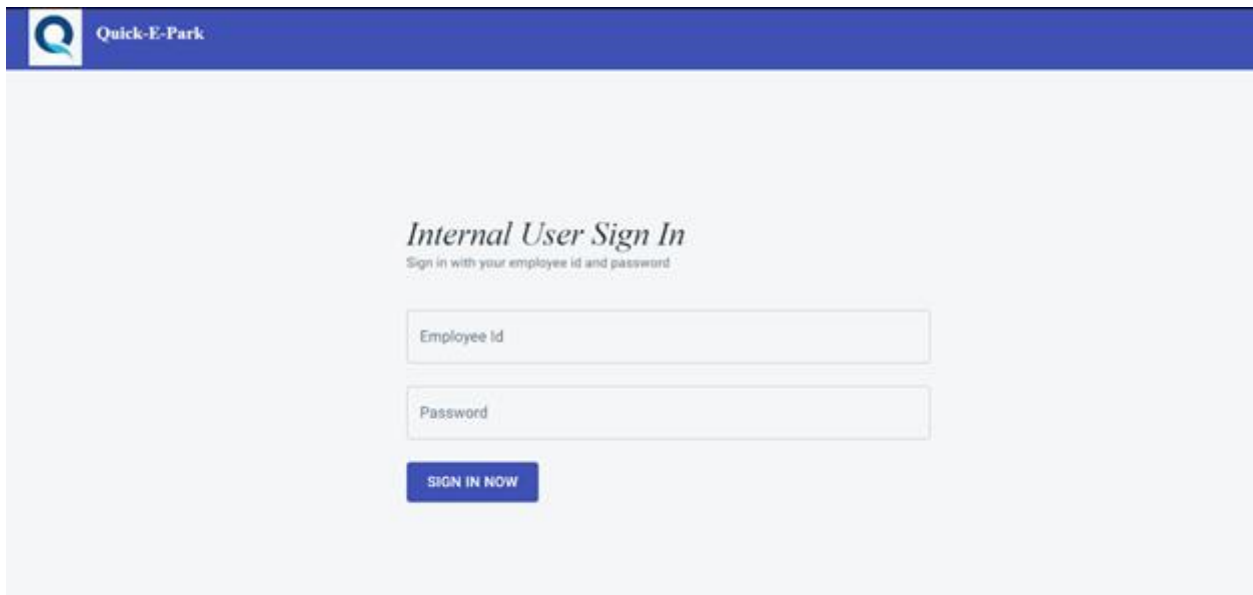


Figure 25: Checker Login Page

On Successful login, the checker will be able to view the dashboard page with the specific functionalities that are related to the role of the checker as shown in figure 26.

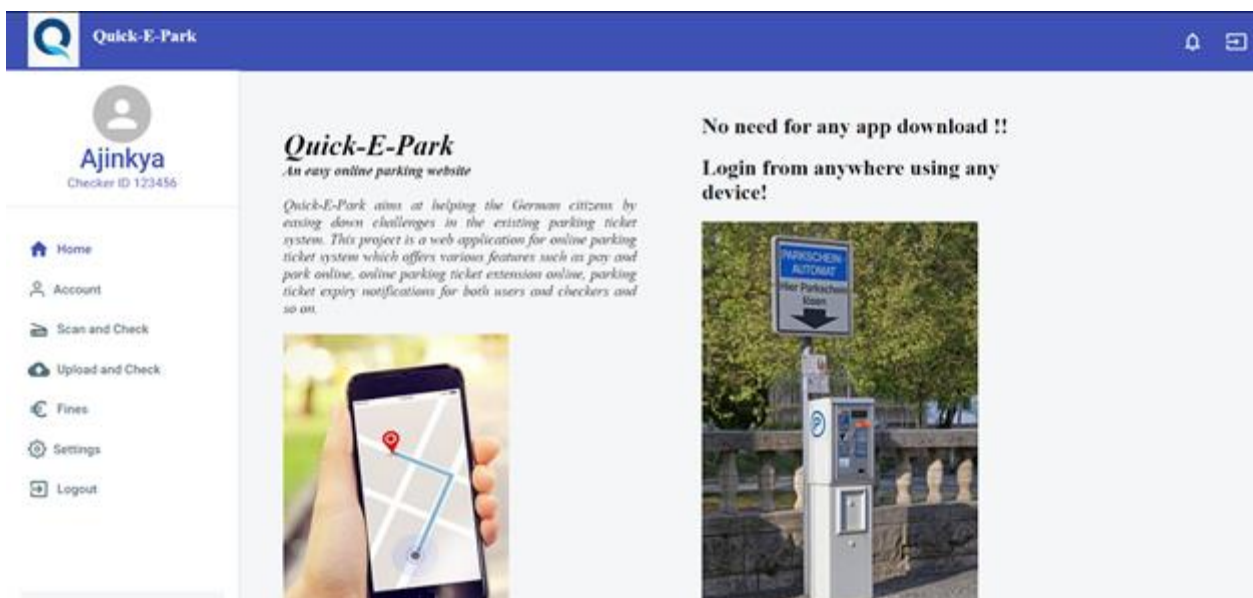


Figure 26: Checker Dashboard Page



After providing the employee Id and password in the login screen, on click of the sign-in button, the "loginValidChecker" back-end web service will be called using React's Axios component. If authentication is successful, the checker is re-routed to the dashboard page; otherwise, the error message will be displayed.

### Back-end Web Service:

Request Type: POST

Request URL: /loginValidChecker

Request Structure:

```
{
  "empId":<string>,
  "password":<string>
}
```

### Success Request-Response:

```
{
  "data": {
    "authentication": "true",
    "email": "test@test.com",
    "empId": "123456",
    "firstName": "test",
    "lastName": "test",
    "location": "test",
    "mobileNumber": "None"
  },
  "statusCode": "200",
  "statusDesc": "Login Successful.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

### Failure Request-Response:

```
{
  "data": "-1",
  "statusCode": "211",
  "statusDesc": "Input credentials incorrect",
  "statusMsg": "Error",
  "statusType": 1
}
```

### Validations Covered:

1. Login credentials match with the database.
2. Checker has an active flag of 1. Active flag 1 indicates active checker (employee). Old employees are not allowed to login.

### Actions to be done:

This API allows the checker to login into the web application. The API validates the employee id and password entered by the user with the user entries which are already existing in the tbl\_checker table

of Quick-E-Park database. If the checker details exist in the database and the checker active status flag is 1, then it allows the user for successful login, otherwise this API returns an error response to the user.

### 9.2.3 Parking Ticket Validation

The checkers can scan or upload images of the vehicle registration number and get the vehicle registration number automatically using Quick-E-Park's image processing functionality. The vehicle registration number is extracted and fed to the view ticket API to validate if the vehicle is parked with a valid parking ticket.

#### User Interface:

On click of the scan vehicle number in Scan and Check tab, the checker is asked for the device camera permission. Once the permission is allowed, the checker can capture the image of the vehicle number plate as shown in the figure 27. Alternatively, the checker can upload the image of the vehicle number plate using upload and check tab as shown in figure 28. These images are processed using image processing to capture vehicle registration number. The checkers can preview the vehicle registration number before confirming as shown in figures 27 and 28. On confirmation of vehicle registration number, parking ticket details are displayed if found in the database.

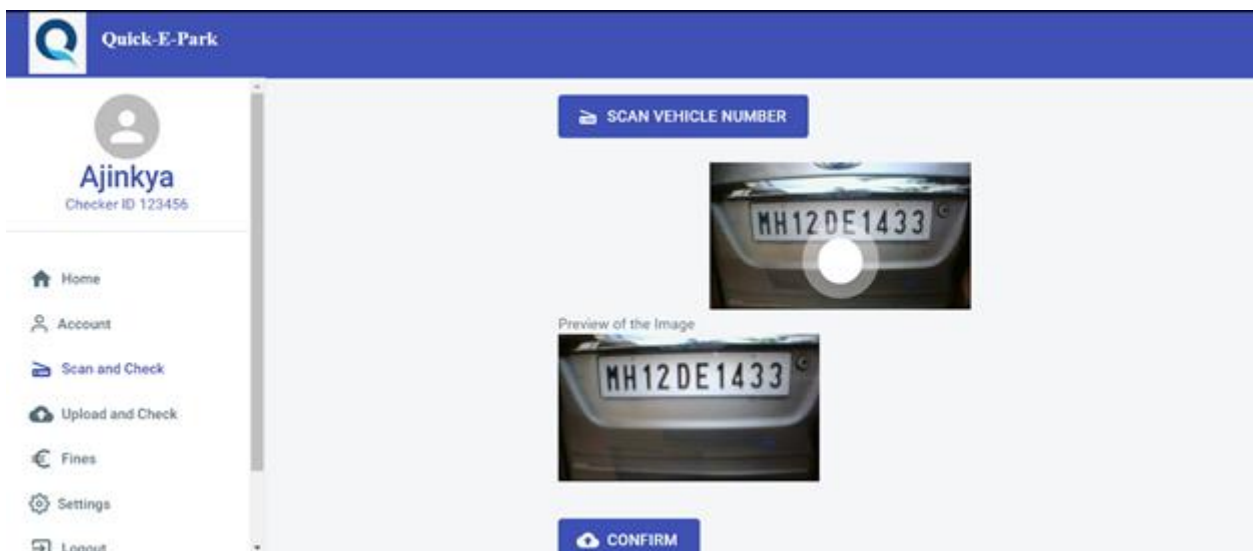


Figure 27: Scan and Check Tab

If the vehicle is registered in the application and doesn't show any valid parking ticket, the checker can issue a fine immediately using the application. Using React-html5-camera-photo component, checker's device camera is utilized for taking the pictures. Once the license plate picture is taken, it will be converted into base64 image format and will be sent to the back-end web service, "License Plate Extraction" which will be called using React Axios component.

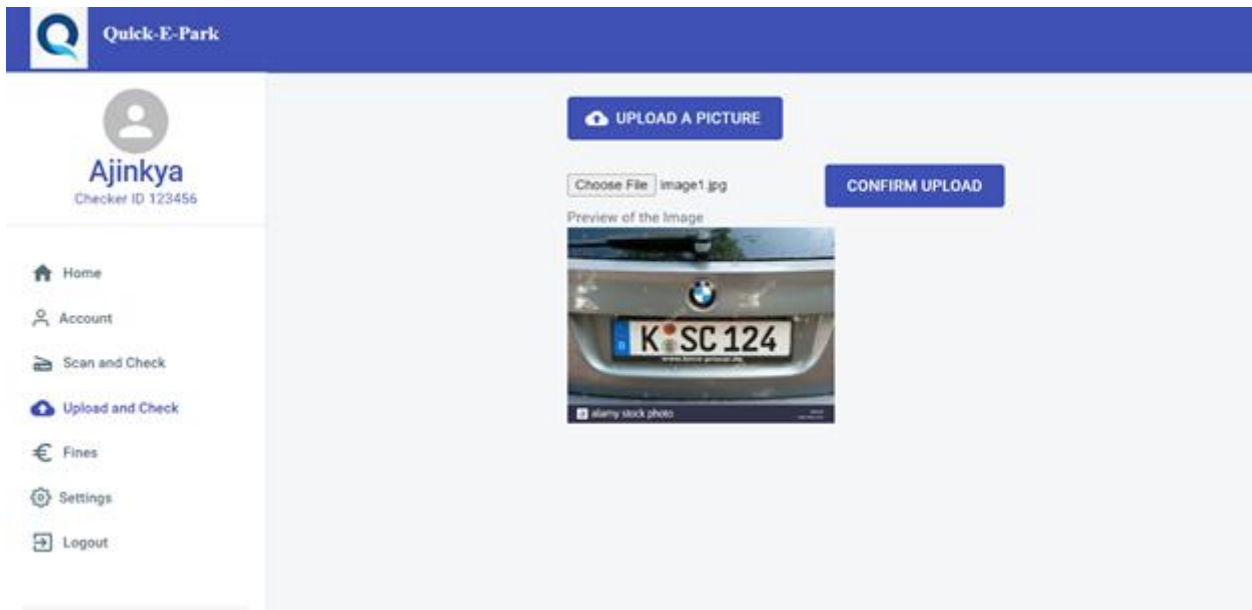


Figure 28: Upload and Check Tab

### Back-end Web Service:

Request Type: POST

Request URL: `/getLicenseNumber`

Request Structure:

```
{
  Base64Image: <base64 string of the image>
}
```

Success Request-Response:

```
{
  "data": {
    "extractedLicenseNumber": "MU BI 40"
  },
  "statusCode": "200",
  "statusDesc": "GENERIC SUCCESS.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

A detailed explanation of the License number extraction from the image is given in the [section 9.4](#).

## 9.2.4 View Ticket Checker

### User Interface:

Once the license number is retrieved from the back-end web service, it will be sent to the checker's user interface. The checker can confirm the license number. Once the checker ensures the license number, another API named "viewTicketChecker" will be called using the React Axios component. Based on the service response, now the checker can see whether the car has a valid ticket or not. If the vehicle has a valid ticket, then the retrieved active ticket will be displayed to the checker; else, the ticket checker will have an option to issue a penalty notice to the vehicle owner.

**Back-end Web Service:**

Request Type: POST

Request URL: `/viewTicketChecker`

Request Structure:

```
{
  "parkedCarRegNo": "DE YY1235"
}
```

Success Request-Response:

```
{
  "data": {
    "email": "sheharaz07@gmail.com",
    "parkedCarRegNo": "DE YY1235",
    "parkingEmail": "sheharaz07@gmail.com",
    "parkingFare": "2020-12-31 15:57:36.503201",
    "parkingLocation": "Duisburg",
    "parkingStartDate": "2020-12-31 15:57:36.503201",
    "remainingParkingDuration": "0"
  },
  "statusCode": "200",
  "statusDesc": "GENERIC SUCCESS.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

Failure Request-Response:

```
{
  "data": "-1",
  "statusCode": "245",
  "statusDesc": "0 rows fetched.",
  "statusMsg": "Error",
  "statusType": 1
}
```

Validations Covered:

1. Checks if the vehicle parking exists in the database with duration involving scan device current time.

Actions to be done:

This API allows the checker to validate if a vehicle is parked with a valid parking ticket. The API checks if the vehicle registration is present in `uv_getparkdetails` view. If the vehicle has an active parking ticket, then it returns the complete detail of parking and parked user by using SQL join of table `tbl_parkdetails`, table `tbl_user` and view `uv_getparkdetails`. If the parking details doesn't exist for the vehicle, API returns an error response to the user interface.

### 9.2.5 Issue Fine

The checkers, post scanning the vehicle registration number, obtain the parking details of the vehicle. If the vehicle is illegally parked, the checkers can issue a fine to the registered owner of the vehicle on Quick-E-Park application. Thus, the illegal parking is dealt using the application.

#### User Interface:

The checker can provide the fine amount and issue fine to the retrieved owner using app. After providing the details, on click of issue fine, the vehicle owner will be issued with the fine.

Once the checker clicks issue fine, the back-end web service “issueFine” will be called using the React Axios component and retrieves the response. Based on the service response, success or error messages will be displayed.

#### Back-end Web Service:

Request Type: POST

Request URL: /issueFine

Request Structure:

```
{
  "parkedCarRegNo": "DE YY1235",
  "parkingFine": "1",
  "empId": "123456"
}
```

Success Request-Response:

```
{
  "data": {},
  "statusCode": "200",
  "statusDesc": "GENERIC SUCCESS.",
  "statusMsg": "Success.",
  "statusType": 0
}
```

*Validations Covered:*

1. Checks the owner of the parked vehicle and issues the fine to the owner.
2. Store the employee id of the checker alongside fines for future reference of issued fine.

*Actions to be done:*

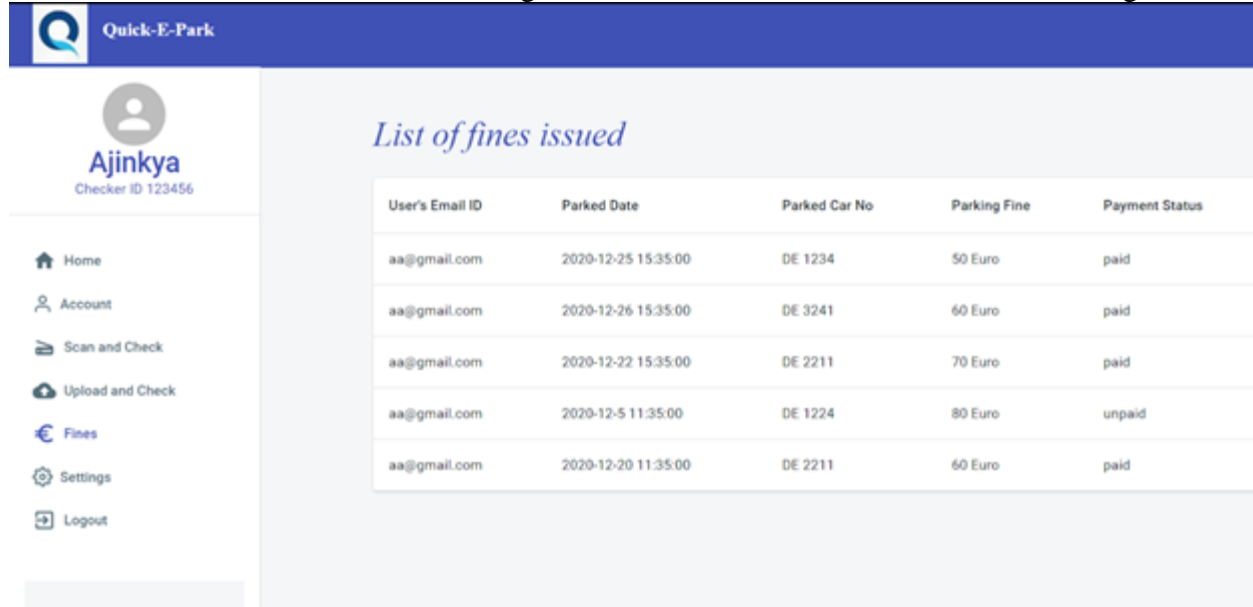
This API allows the checker to issue fine to the user, if a vehicle is illegally parked. Initially, this API checks for the owner of the illegally parked vehicle in the table tbl\_vehicles of the Quick-E-Park database. Next, the API inserts the fine amount, vehicle registration number, time of issued fine and checker employee id into the tbl\_penalty table. The fines can be viewed by both users and checkers in the future using the Quick-E-Park web application.

## 9.2.6 View Issued Fine

The checkers can view the issued fines using the fines tab of the Quick-E-Park web application. The purpose of this functionality is that the checkers can track the fines issued and use it for internal validation and documentation purposes. This functionality also allows the checker to check if any issued fine is unpaid.

### User Interface:

The checkers can view issued fines using the fines tab on the sidebar as shown in the figure 30.



User's Email ID	Parked Date	Parked Car No	Parking Fine	Payment Status
aa@gmail.com	2020-12-25 15:35:00	DE 1234	50 Euro	paid
aa@gmail.com	2020-12-26 15:35:00	DE 3241	60 Euro	paid
aa@gmail.com	2020-12-22 15:35:00	DE 2211	70 Euro	paid
aa@gmail.com	2020-12-5 11:35:00	DE 1224	80 Euro	unpaid
aa@gmail.com	2020-12-20 11:35:00	DE 2211	60 Euro	paid

Figure 29: Fines tab for Checker

On click of the fines tab in the dashboard sidebar, “viewIssuedFine” back-end web service will be called from the react using Axios component and based on the response, success or error message will be displayed.

### Back-end Web Service:

Request Type: POST

Request URL: `/viewIssuedFine`

Request Structure:

```
{
  "empId": "123456"
}
```

Success Request-Response:

```
{
  "data": [
    {
      "carRegistrationNo": " DE YY1235",
      "email": "sheharaz207@gmail.com",
      "fineDate": "2020-12-29 10:42:51.398743",
      "mobileNo": "None",
      "name": "Sheharaz2",
      "paidStatus": "Paid",
      "parkingFine": "1 EUR"
    }
  ]
}
```

```

    },
    "statusCode": "200",
    "statusDesc": "GENERIC SUCCESS.",
    "statusMsg": "Success.",
    "statusType": 0
}

```

Failure Request-Response:

```

{
  "data": "-1",
  "statusCode": "245",
  "statusDesc": "0 rows fetched.",
  "statusMsg": "Error",
  "statusType": 1
}

```

*Actions to be done:*

This API allows the checker to view the issued fines. Initially, this API checks if the employee id has issued any fines in the past by checking in the table `tbl_penalty`. If there are fines issued by the checker in the past, then a success response is returned. Otherwise, a failure response is returned.

### 9.3 Notification and Ticket History

#### Ticket History

The history of tickets is stored by Quick-E-Park application in a separate table to avoid load and deadlock on the main tables `tbl_user` and `tbl_parkdetails`. The parking history details are stored in table `tbl_history`. This is achieved using the stored procedure `sp_updateTransactions()`. This stored procedure checks if the table `tbl_parkdetails` contains an expired ticket. If an expired parking ticket is found in the `tbl_parkdetails`, then the user and parking details are inserted into table `tbl_history` and the row is deleted from the `tbl_parkdetails`. This ensures that there is no load on `tbl_parkdetails` table and the table `tbl_history` is independently viewed without affecting SQL transactions on main tables. The procedure `sp_updateTransactions()` is triggered every 2 minutes by the python script `notification.py`.

#### Notification

The users are notified in advance regarding parking tickets that are about to expire. This is achieved using `notification.py`. This python script runs in an infinite loop with a sleep time interval of 120 seconds. It checks in the view `uv_getparkdetails` for the active parking ticket. This view contains a flag column called `notificationSent`. If this flag is set to 0 and the remaining parking duration for a vehicle is less than 10 minutes, then a notification is sent to the user, warning about the expiring parking ticket. The email is sent using Simple Mail Transfer Protocol (SMTP) [22]. A Gmail account with an email address of “`quickepark@gmail.com`” is created and configured with SMTP access for

sending notifications related to Quick-E-Park application. Thus, users can view the email notification and extend the ticket using the Quick-E-Park web application. Once the notification is sent to the user, the flag `notificationSent` is set to 1, so that the notification is not sent multiple times and spamming of the user inbox is avoided.

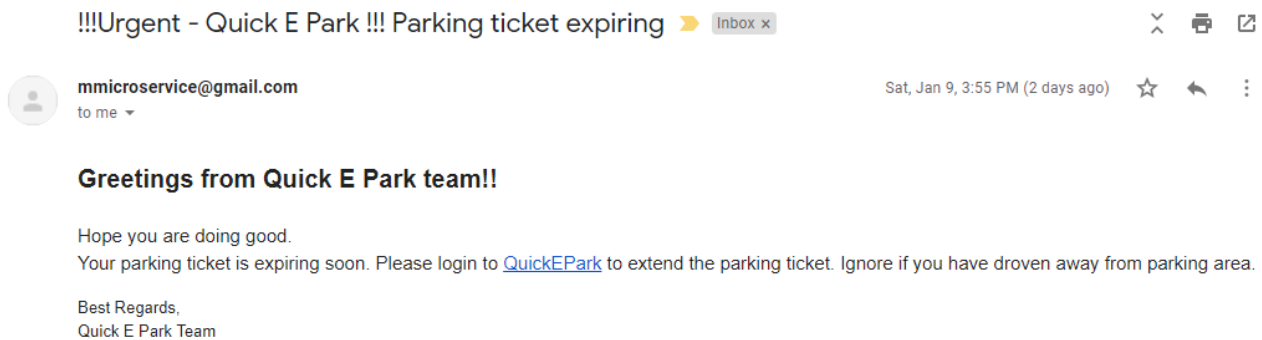


Figure 30: Sample Notification

## 9.4 License Image Extraction

### Image Processing

The image is taken from the checkers side (i.e., Client) and all the processes related to retrieving the text from the image occurs in the backend (Server), so the image needs to be sent from the client side to the server side.

There are several methods which can be used to transfer data between client and server and all the different methods have their own advantages and disadvantages based on the different architecture and the usage. For transferring data between client and server it was found that there are two methods which are suitable for this application.

- The first was to transfer data between client and server using the sockets, but the main problem with sockets was the downfalls of the sockets programming, like opening a socket and connecting with it and once the data is exchanging the socket must be closed properly and all of these things should happen in a sequence, which sometimes becomes tricky to handle in the code and then exceptions or crash will become usual with the application [23].
- The second was to encode the image to a string and to send it to the backend with the help of an API call. This process is simple and easy to implement and also the size of the data transferred (Image) in the application is not of large size (Sockets are preferred while sending large chunks of data), therefore it was also reliable to use this method. So, the encoding method that we used here is Base64 Conversion.

### Base64



Base64 is one of the most common methods of converting a binary data/file to a string [24]. The process of converting the binary data to a string or test is called encoding [24]. The encoded data is then sent over the network and once it reaches the destination then a decoding process happens which converts the string into the original file [24]. The most common example of Base64 encoding can be seen in the Email attachments [24].

After the click of the button on the UI, it will capture the complete image of the license plate and then React will call the backend API (Image Extraction) and will send the Base64 encoded image to the server. Here the UI has completed its part and now the server (Backend) will process the image further.

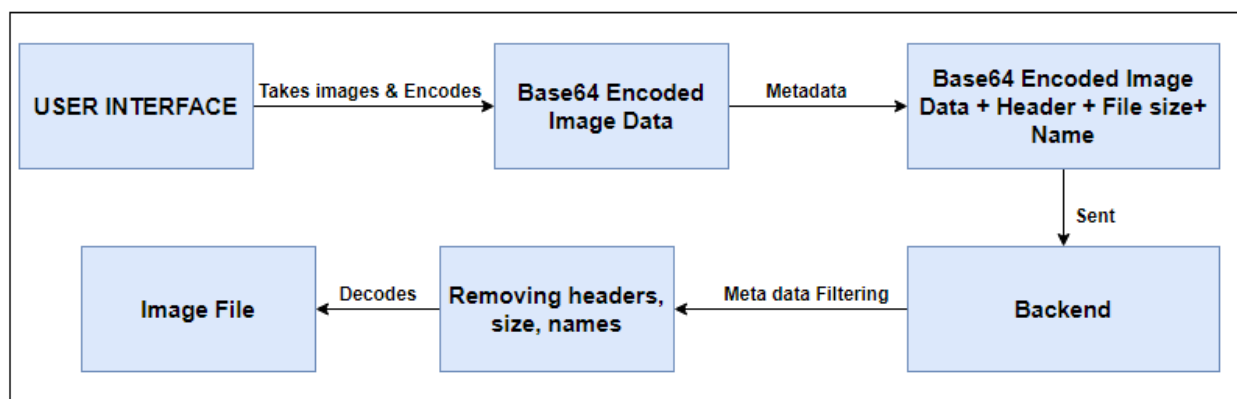


Figure 31: Encoding and decoding of base64

The encoded Base64 data of the image which came to the backend has some metadata (headers, image name, etc) that needs to be filtered before decoding it to the original image.

So, in the backend, the server will remove the metadata from the Base64 images and then will decode the encoded image using the function `b64decode (ImageStr)` function. Once the image is decoded, it's going to be saved in a particular location by a particular name (`./Images/Image.jpg`) and

The Image will be fetched from this location for further processing which is done by OCR.

## OCR

It stands for optical character recognition and it is used to extract the text from an Image. It consists of several sub-processes to convert an image into text. There are several open-source OCR algorithms available, but for this application, Google's Tesseract OCR engine has been used. For using this the Tesseract OCR, it should be installed in the system. The Tesseract OCR exe is the executable file that takes the image as input and converts it into the string.

It's free open-source software distributed under an Apache license and was introduced by Hewlett-Packard and then later acquired and developed and maintained by Google [25]. Most of the methods in the tesseract are written in C and C++ [25]

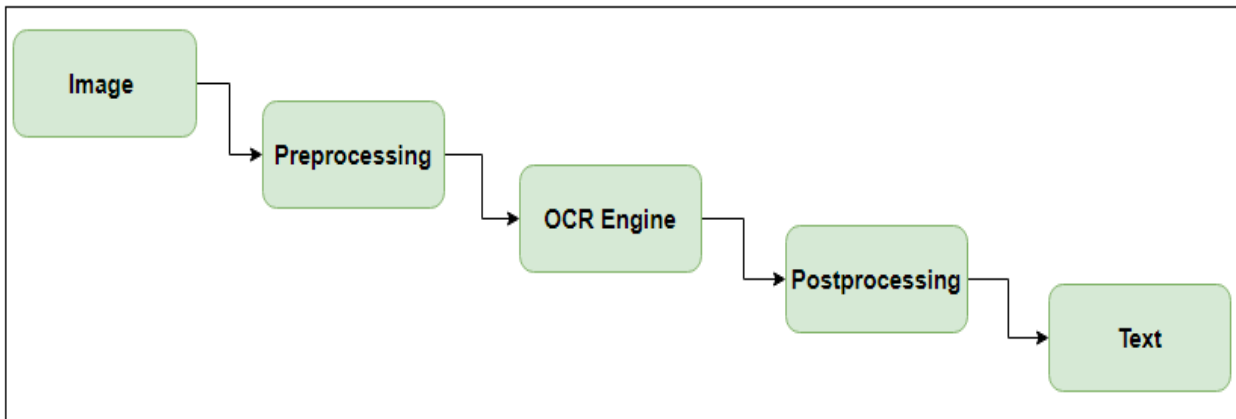


Figure 32: Working of OCR

The TesseractOCR engine alone is not enough to process the Image as it takes input in the form of a command line argument which is provided through the Pytesseract library of the python. The Pytesseract library has all of the inbuilt functions for Image Processing.

Now the TesseractOCR engine has to take an image (Vehicle License Plate Image) as an input to retrieve the text (License number) from it, but the Image cannot be directly passed as an input to the TesseractOCR engine. Before that there is some pre-processing that needs to be done on the image to make it more suitable for processing by the TesseractOCR. Also, a region has to be selected in the image which contains the license plate number, as there might be many texts (like vehicle brand, personal names etc) in the image which are not of interest. The marked region in the image is called the Region of Interest and OCR will work only on that region to retrieve the text.

The pre-processing steps that are going to perform are as follows: For pre-processing of the Image, python's OpenCV library has been used.

### **Conversion of Image from one colour scale to another**

Basically, this step is done to remove the colour from the image and make it gray, to remove the colour dependencies because sometimes the bright or dull colours make a significant effect on the processing of the image, therefore it is made of a neutral colour [26]. For this conversion, OpenCV library is used and the function is `cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)` [26]. This function converts the image to grayscale.



Figure 33: Conversion from one colour scale

## Resizing

In this pre-processing step, the size of the image is decreased a bit, because the accuracy of the tesseract OCR is not as good among big size images as compared to medium size images [27]. The function used to perform this operation is `cv2.resize(image, None, fx=0.5, fy=0.5)` [27]. It will reduce the x and y dimensions of the image to half.

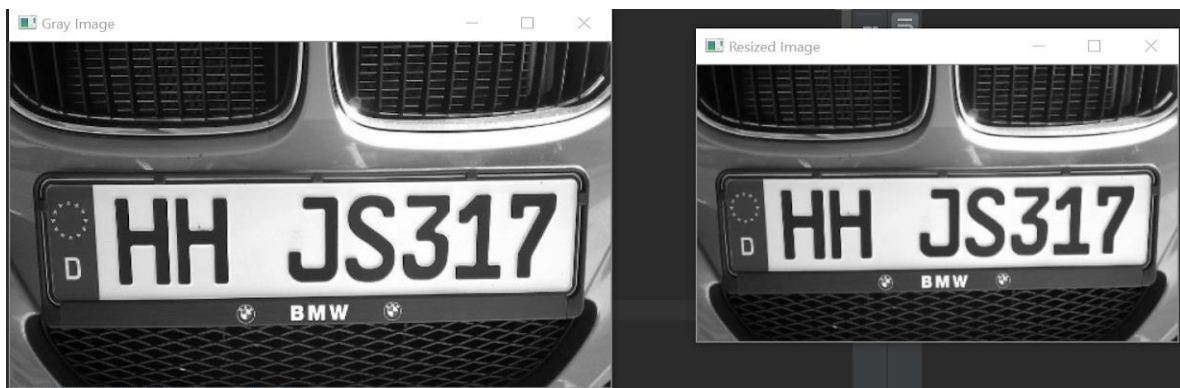


Figure 34: Resizing of Image

## Filtering the Image

This step is important to remove the excess noise from the image by removing the high-frequency pixels by passing through it a low pass filter which in the end makes the image smoother [28]. There are several filtering methods available but the one that is used here is bilateral filtering because this filter removes the noise while keeping the edges sharp on the image and the edges will be used in the further steps, while in the other filtering methods like Gaussian or Median blurring the edges are not kept sharp [28]. The function that has used is `cv2.bilateralFilter(Image)`.

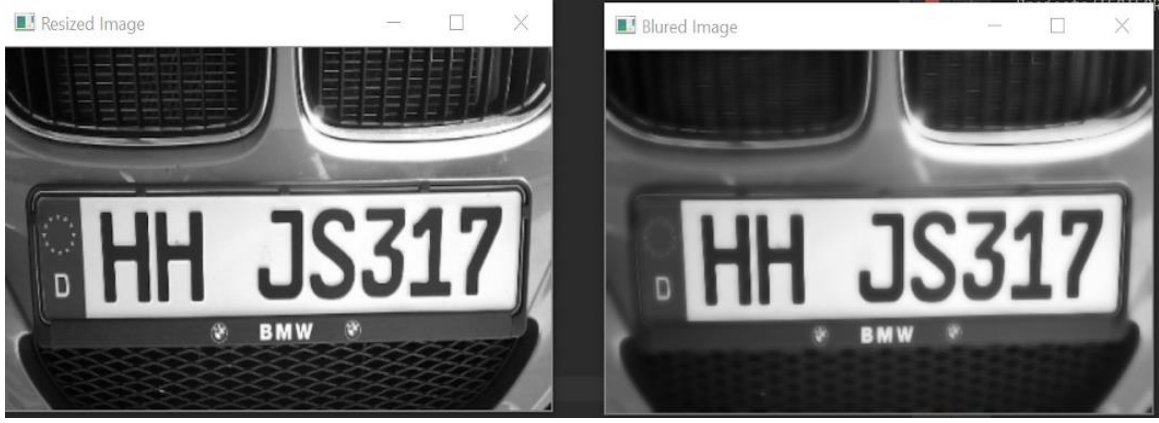


Figure 35: Filtered Image

### Edge Detection

In this step, the image is marked with the edges. The edges are helpful in detecting the region of interest in the image [29]. The image is passed to a Sobel kernel in both horizontal and vertical directions to get the derivatives and the angle of the edges [29]. The open CV2 function that is used for it is `cv2.Canny(blur, MinIntensity, MaxIntensity)`.

$$Edge\_Gradient (G) = \sqrt{G_x^2 + G_y^2}$$

$$Angle (\theta) = \tan^{-1} \left( \frac{G_y}{G_x} \right)$$

Figure 36: Edge Detection Formula [29]

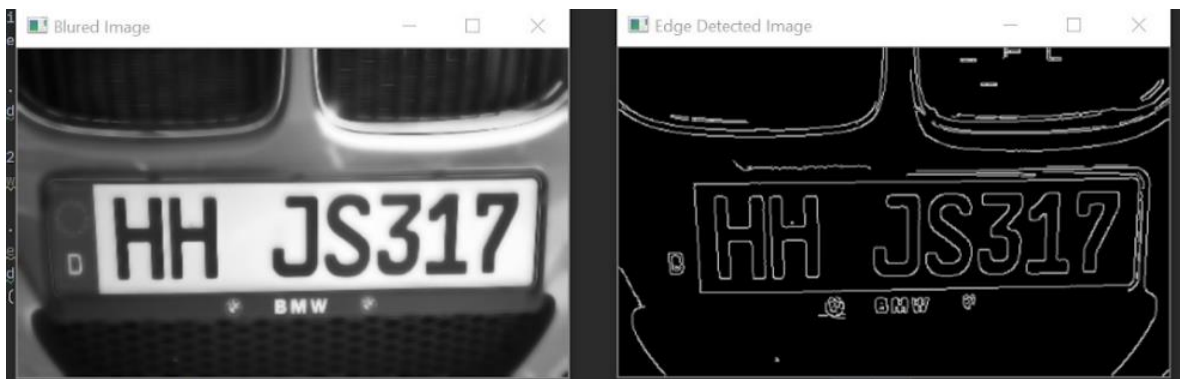


Figure 37 : Edge detection

### Contours detection

In this pre-processing step, the contours will be identified. Contours can be described as the continuous point of an object [30]. It is quite useful in detecting any small particular shape from a bigger image [30]. So, this technique is used to detect the license plate from the image. For detection

of contour, the function used is `cv2.drawContours(image)`. This function detects all the contours present in the image.



*Figure 38: Contour detection*

### Trimming out the License plate

Until here, all the pre-processing of the image has been done. Here logical operations are performed to detect the license plate from the bigger image. Now from the list of all contours, a rectangular box (the shape of the license plate) will be selected. For that looping has been done to all the contours (all the present shapes in the Image) with the help of the OpenCV function named `cv2.approxPolyDP()` [31]. This method helps to find out all the rectangles in the Image and among those rectangles a condition is kept of width to height ratio to successfully select the license plate rectangle. and is saved by the name of `Plate.png`.



*Figure 39: Extracted License Number*

### Tesseract OCR

This step is the crucial role of the tesseract OCR. Once all the pre-processing of the image is done, the pre-processed Image of the License Plate (`Plate.png`) is ready to be given as an input to the Tesseract OCR. For this, there is also a function called `pytesseract.image_to_string(Image)` which does the magic and converts the image to text and returns a string of it. Now the license number is retrieved and is sent to the user interface.

## 10. Deployment

For the deployment of the Quick-E-Park application, various functionalities of docker are used. The application is deployed in the docker as a multi-container application. The application used both the Dockerfile and Docker-compose file functionalities for the containerization. Docker-compose file is a .yml file for the configurations of the various containers of the application [18] [19] [20]. This Docker-compose file comprises four containers such as nginx, front-end, back-end, database.

Nginx is pulled as a docker image from the docker hub and created as a container for reverse proxying and web serving. "Nginx.conf" file contains the configurations such as ports, web service urls, etc., for providing the interaction between the front-end user interface and the back-end web services. Figure 39 depicts some of the important configurations of the Nginx.

```

user www-data;
worker_processes auto;
pid /run/nginx.pid;
include /etc/nginx/modules-enabled/*.conf;
events {
    worker_connections 1024;
}
http {
    # Weather Report Reverse Proxy
    server {
        listen 80;
        server_name localhost 127.0.0.1;
        location /addNewRegistration {
            proxy_pass          http://back-end:5000/addNewRegistration;
            proxy_set_header    X-Forwarded-For $remote_addr;
        }
        location /login {
            proxy_pass          http://back-end:5000/loginvalid;
            proxy_set_header    X-Forwarded-For $remote_addr;
        }
    }
}

```

Figure 40 : Nginx Configurations File

The docker file called "service.Dockerfile" is created with required configurations for the python services for the back-end docker container. The docker file called "web.Dockerfile" is made with the necessary configurations to run the react client application. Postgres database is pulled from the docker hub as an image and configured with docker volumes, and created as a container. Docker volumes are used to bind the databases, tables, views, and database schemas of the application and mount them into the containers. After creating the docker compose file as shown in figure 40, using the "docker-compose up --build" command, all the application containers were built and executed. Using the exposed ports in the "docker-compose.yml file," the application is accessed.

```

version: "3.7"
services:
  # Proxies requests to internal services
  nginx:
    image: nginx:1.17.10
    container_name: reverse_proxy
    volumes:
      - ./nginx/nginx.conf:/etc/nginx/nginx.conf
    ports:
      - 80:80

  front-end:
    build:
      context: ./frontend
      dockerfile: web.Dockerfile
    ports:
      - 8080:80
    restart: on-failure

  back-end:
    build:
      context: ./backend
      dockerfile: service.Dockerfile
    ports:
      - 5000:5000
    restart: on-failure

  database:
    # Pull the latest 'postgres' image
    image: "postgres"
    container_name: "quickeparkdb"
    # Postgres environment data saved in local volumes
    volumes:
      - quickepark-database:/var/lib/postgresql/data
      - ./database/init.sql:/docker-entrypoint-initdb.d/init.sql
    # Create environment
    environment:
      - POSTGRES_USER=postgres
      - POSTGRES_PASSWORD="Password"
    # Bind mount the default Postgres port
    ports:
      - 5432:5432

#MOUNTING VOLUMES
volumes:
  quickepark-database:

```

Figure 41 : Docker Compose File

## 11. Evaluation

### 11.1 Frontend User Interfaces

The required validations, error messages, and warning alerts are implemented at various screens during the user interface development. During the evaluation, all such scenarios are tested.

- If the customer clicks the sign-in or register button on the respective screens of login and registration without providing the required inputs, validation, and proper error messages are displayed. For example, the figure 41 depicts the validation messages on login screen.

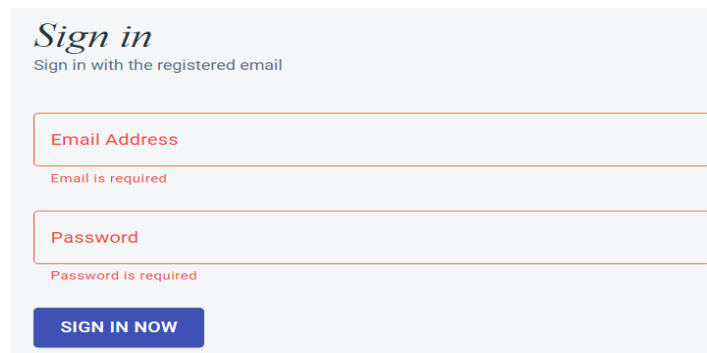


Figure 42:Ex:Login Screen Validation

- If the customer tries to buy a parking ticket for a vehicle number with an already active session, a warning info alert message is displayed.
- The application is seeking permission to use the checker's device camera if the checker wishes to scan the license plate in the checker's module.
- Since Quick-E-Park is a web application, it is implemented in a responsive way to support different types of devices such as various types of mobiles, laptops, and tablets. The figure 42 and 43 shows Quick-E-Park's responsive screens.



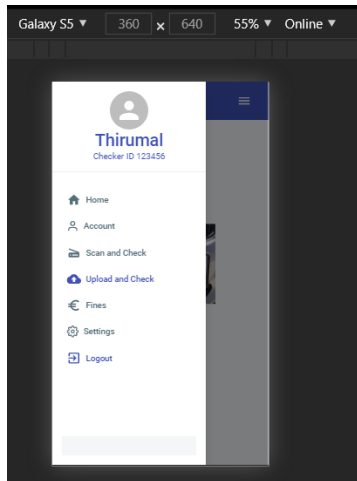


Figure 43: Galaxy S5 Mobile Screen

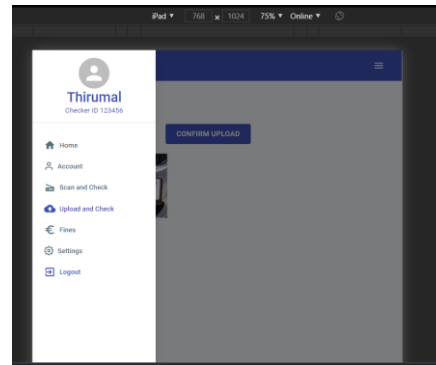


Figure 44: iPad Screen

## 11.2 Backend Web Services

The backend APIs of the Quick-E-Park were thoroughly tested and evaluated before deployment. The functional testing of APIs was achieved using the postman application. The input and output of all APIs are in JSON format. The major evaluation carried out are as follows:

- Input to the API is evaluated before performing actual function. If the input is in incorrect format, a response stating JSON input incorrect is returned.
- The database is tested for connection before performing actual function. The corresponding error response is returned if database connection fails.
- APIs are evaluated for mandatory inputs and error response is returned in case of missing mandatory input.
- The registration of the user with the same email and license number is evaluated and an error response is returned stating the user already exists.
- For all 3 users, the input credentials should be correct. Otherwise, an error response is returned stating invalid credentials.
- If a vehicle is already actively parked with an entry in the database, another attempt to park the same vehicle is not allowed. Error response stating vehicle parking already exists is returned.
- Notification is sent to the user in case the parking ticket is expiring within 10 minutes.
- For getting pending payment details, the sum of only expired unpaid tickets and unpaid fines are calculated and returned separately by the API. The paid ones are ignored during the calculation.

- One vehicle is allotted with only one owner. If a user tries to add already existing vehicle registration with vehicle type as owner, an error response is returned stating owner already exists. This vehicle can be added under temporary vehicle type.
- While updating the password using accounts tab, the API evaluates the old password and allows password change only if the old password entered is correct. Otherwise, an error response stating invalid credentials is returned.

## 12. System Limitation

The Quick-E-Park is a prototype implemented within a limited time, so there are few limitations in this system, which are as follows:

- The Google Map API used here is the free version, so it has limited functionality.
- The Image used to extract the vehicle's license number should be of good resolution; else, the extracting process becomes complicated.
- This application is just a prototype, so it is still not tested in the real world. All the testing has been done locally with the sample data.
- This application does not provide reserve parking spaces for the user, and it is only limited to On-Street Parking.

### **13. Conclusion**

Quick-E-Park is a simple, user friendly and light weighted web application, successfully implemented and tested. With the checkers module, issuing and managing fines will be completely online. Everything can be managed easily as all the details will be stored in the system. Although the system has certain limitations, with some more research and development, the web application can be very helpful for the end-users as well as city council by making their work easier and convenient.

## **14. Future Scope**

- The paid version of the Google APIs is required for geocoding and location features, which later can be used in more features like showing vacant parking spots, showing the distance between users and their parked car.
- This application can provide reserved parking spaces for the user in the future.
- Hardware, i.e., cameras, can be integrated with the application to automate the private parking lots, including entry and exit doors.

## References

- [1] (n.d.). Retrieved January 13, 2021, from <http://www.gettingaroundgermany.info/parken.shtml>
- [2] Stadtplanungs- und Bauordnungsamt. (n.d.). Retrieved January 13, 2021, from [https://www.dortmund.de/de/leben\\_in\\_dortmund/planen\\_bauen\\_wohnen/stadtplanungs\\_und\\_bauordnungsamt/stadtplanung/verkehrsplanung/gesamtstaedtische\\_verkehrsplanung/parken/wer\\_darf\\_auf\\_welchen\\_Stellplatz\\_parken.html](https://www.dortmund.de/de/leben_in_dortmund/planen_bauen_wohnen/stadtplanungs_und_bauordnungsamt/stadtplanung/verkehrsplanung/gesamtstaedtische_verkehrsplanung/parken/wer_darf_auf_welchen_Stellplatz_parken.html)
- [3] <https://www.mopo.de/hamburg/neue-geraete-in-hamburg-parkscheinautomaten-haben-jetzt-eine-wichtige-funktion-36221908>
- [4] Mobile parking with PARK NOW - How it works. (2020, December 21). Retrieved January 13, 2021, from <https://de.park-now.com/en/how-it-works/>
- [5] CanoantiSpam('#antispam-2086705670', B. (2016, August 23). New Pay by Phone APP - smartBluezone. Retrieved January 13, 2021, from <https://www.parking-net.com/parking-news/came-parkare/new-pay-by-phone-app-smartbluezone>
- [6] M.chin, Kaja, & Styles, C. (n.d.). Peter Park System GmbH. Retrieved January 13, 2021, from <https://www.parking-net.com/parking-industry/peter-park-system-gmbh>
- [7] (n.d.). Retrieved January 13, 2021, from <https://easypark.de/howItWorks/en>
- [8] Client-server architecture. (n.d.). Retrieved January 13, 2021, from <https://www.britannica.com/technology/client-server-architecture>
- [9] Microservice Architecture pattern. (n.d.). Retrieved January 13, 2021, from <https://microservices.io/patterns/microservices.html>
- [10] React – A JavaScript library for building user interfaces. (n.d.). Retrieved January 13, 2021, from <https://reactjs.org/>
- [11] Getting Started. (n.d.). Retrieved January 13, 2021, from <https://create-react-app.dev/docs/getting-started/>
- [12] DigitalOcean. (2021, January 13). How To Use Axios with React. Retrieved January 13, 2021, from <https://www.digitalocean.com/community/tutorials/react-axios-react>
- [13] About The AuthorShedrack Akintayo is a software engineer from Lagos, & Author, A. (2020, June 03). Consuming REST APIs In React With Fetch And Axios. Retrieved January 13, 2021, from <https://www.smashingmagazine.com/2020/06/rest-api-react-fetch-axios/>
- [14] React-html5-camera-photo. (n.d.). Retrieved January 13, 2021, from <https://www.npmjs.com/package/react-html5-camera-photo>
- [15] Flask (web framework). (2020, December 22). Retrieved January 13, 2021, from [https://en.wikipedia.org/wiki/Flask\\_\(web\\_framework\)](https://en.wikipedia.org/wiki/Flask_(web_framework))
- [16] About. (n.d.). Retrieved January 13, 2021, from <https://www.postgresql.org/about/>
- [17] Orientation and setup. (2021, January 08). Retrieved January 13, 2021, from <https://docs.docker.com/get-started/>
- [18] Docker. (2021, January 08). Retrieved January 13, 2021, from <https://docs.docker.com/engine/reference/commandline/docker/>

- [19] (n.d.). Retrieved January 13, 2021, from [https://hub.docker.com/\\_/nginx](https://hub.docker.com/_/nginx)
- [20] F5, L. (2020, February 07). Deploying NGINX Plus as an API Gateway, Part 1. Retrieved January 13, 2021, from <https://www.nginx.com/blog/deploying-nginx-plus-as-an-api-gateway-part-1/>
- [21] Use NGINX as a Front-end Proxy and Software Load Balancer. (n.d.). Retrieved January 13, 2021, from <https://www.linode.com/docs/guides/use-nginx-as-a-front-end-proxy-and-software-load-balancer/>
- [22] SMTP protocol Explained (How Email works?). (2019, August 19). Retrieved January 13, 2021, from <https://www.afternerd.com/blog/smtp/>
- [23] Pacerier, PacerierPacerier 73.8k8484 gold badges318318 silver badges593593 bronze badges, Surivsuriv 62688 silver badges1313 bronze badges, & Uwe KeimUwe Keim 36.2k3636 gold badges154154 silver badges259259 bronze badges. (1960, June 01). Disadvantages of websockets. Retrieved January 13, 2021, from <https://stackoverflow.com/questions/6224771/disadvantages-of-websockets>
- [24] Amin, A. (n.d.). Encoding and Decoding Base64 Strings in Python. Retrieved January 13, 2021, from <https://stackabuse.com/encoding-and-decoding-base64-strings-in-python/>
- [25] Tesseract-Ocr. (n.d.). Tesseract-ocr/tesseract. Retrieved January 13, 2021, from <https://github.com/tesseract-ocr/tesseract>
- [26] Changing Colorspaces¶. (n.d.). Retrieved January 13, 2021, from [https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_imgproc/py\\_colorspaces/py\\_colorspaces.html](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_colorspaces/py_colorspaces.html)
- [27] Python OpenCV cv2 Resize Image. (n.d.). Retrieved January 13, 2021, from <https://pythonexamples.org/python-opencv-cv2-resize-image/>
- [28] Smoothing Images. (n.d.). Retrieved January 13, 2021, from [https://docs.opencv.org/master/d4/d13/tutorial\\_py\\_filtering.html](https://docs.opencv.org/master/d4/d13/tutorial_py_filtering.html)
- [29] Canny Edge Detection¶. (n.d.). Retrieved January 13, 2021, from [https://opencv-python-tutroals.readthedocs.io/en/latest/py\\_tutorials/py\\_imgproc/py\\_canny/py\\_canny.html](https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html)
- [30] Contour Approximation Method. (n.d.). Retrieved January 13, 2021, from [https://docs.opencv.org/master/d4/d73/tutorial\\_py\\_contours\\_begin.html](https://docs.opencv.org/master/d4/d73/tutorial_py_contours_begin.html)

## Appendix