Cycle Bus

Layani Wathsala

B.Sc.(Hons) in Software Development

April 11, 2022

Final Year Project

Advised by: Dr Brian McGinley

Department of Computer Science and Applied Physics

Galway-Mayo Institute of Technology (GMIT)

Contents

About this project

Abstract Cycle bus is a community-based initiative to promote cycling among school children, which has become famous since it encourages children to begin the day healthily and energetically. Although cycle buses are already in place around Ireland, and efforts are applaudable, the safety of children who participate is still doubtful. The motivation of this project was to develop an application that can handle the main functionalities and ensure the safety of children. The project's requirements were finalized, taking into account of real-life user needs of the cycle bus community. Moreover, along with a functional system, a document on development process details, a review of technologies utilized, system design details, objectives and an evaluation are presented. Deliverables of this project include a mobile application to handle activities related to a cycle bus and a web-based administration site. Non-functional attributes targeted throughout the development cycle are customer experience, efficiency and performance. UI testing of the application was done manually to ensure user-friendly and efficient processes. API testing was carried out using Postman to conclude whether their functionality, performance and security are safeguarded. There is no doubt that the application can be enhanced more, but it administers most of the functional requirements of the cycle bus community

Chapter 1

Introduction

Cycle Bus is a community-based initiative to promote cycling among school children. This movement aims to make children experience how fun cycling is, encourage them to start the day energetically, and ensure the safety of children cycling to school. Children ride along a predecided route, guided by a Marshal, who could be a parent himself or an experienced volunteer. Marshal rides through a fixed route stopping at a few location points allowing children to join the bus. The need for this application came through its actual users. Currently, few Cycle Buses are functioning in the Galway region.[1].

1.1 Context

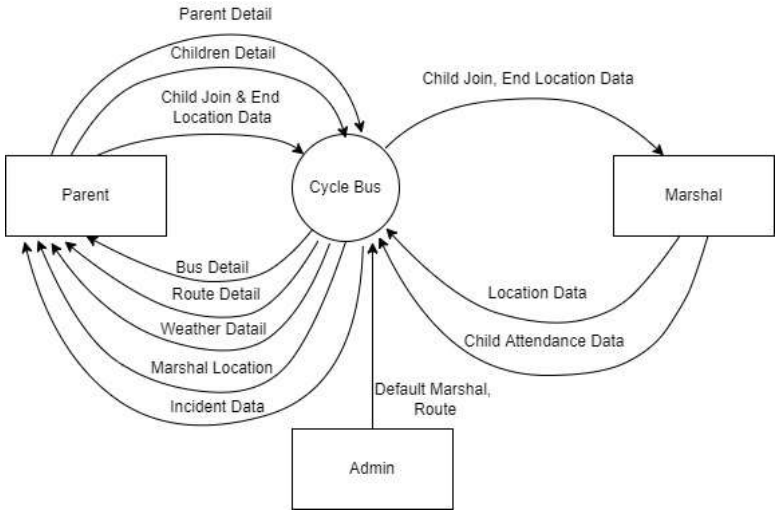

Figure 1.1: Context Diagram

As in Figure 1.1 above, three types of users can interact directly with the system developed. They are Parent(Can also be a Marshal), Marshal and Admin. A child is an indirect user. Parents register themselves and their children to the app providing all child data to the system. Then the parent adds the child to a bus providing the child's join location for each bus route. When Marshal starts the ride along the default route of the bus, he can see join locations of children who registered to the bus. Furthermore, Marshal can mark or check the attendance of a child. The parent can inform about the absence of a child, track the cycle bus's location, view the arrival time to the child's join location, and view weather data and incidents reported by Marshal during the ride. Admin is responsible for administrative tasks such as setting default Marshal and bus route. Labels on the arrows represent the data flow between the system and its users, which aids the system to function.

1.2 Functional Requirements of the System

Functional requirements of the system are

1. Registration of parents and children to the mobile app
2. Register the child to a cycle bus by the parent
3. Select the join location of the child for each route of the cycle bus
4. Starting a ride by marshal
5. Retrieve the GPS location of the marshal
6. Display map views to both parent and Marshal
7. Mark the participation of the child by the parent
8. Enabling parents to track the location of cycle bus
9. Incident reporting by marshal
10. Display data to parent(incident, weather data, and reach the time of cycle bus to join location of the child)
11. Mark and check the attendance of the child
12. Enabling Admin user to set default marshal and route for a cycle bus

These features are displayed in Figure 1.2

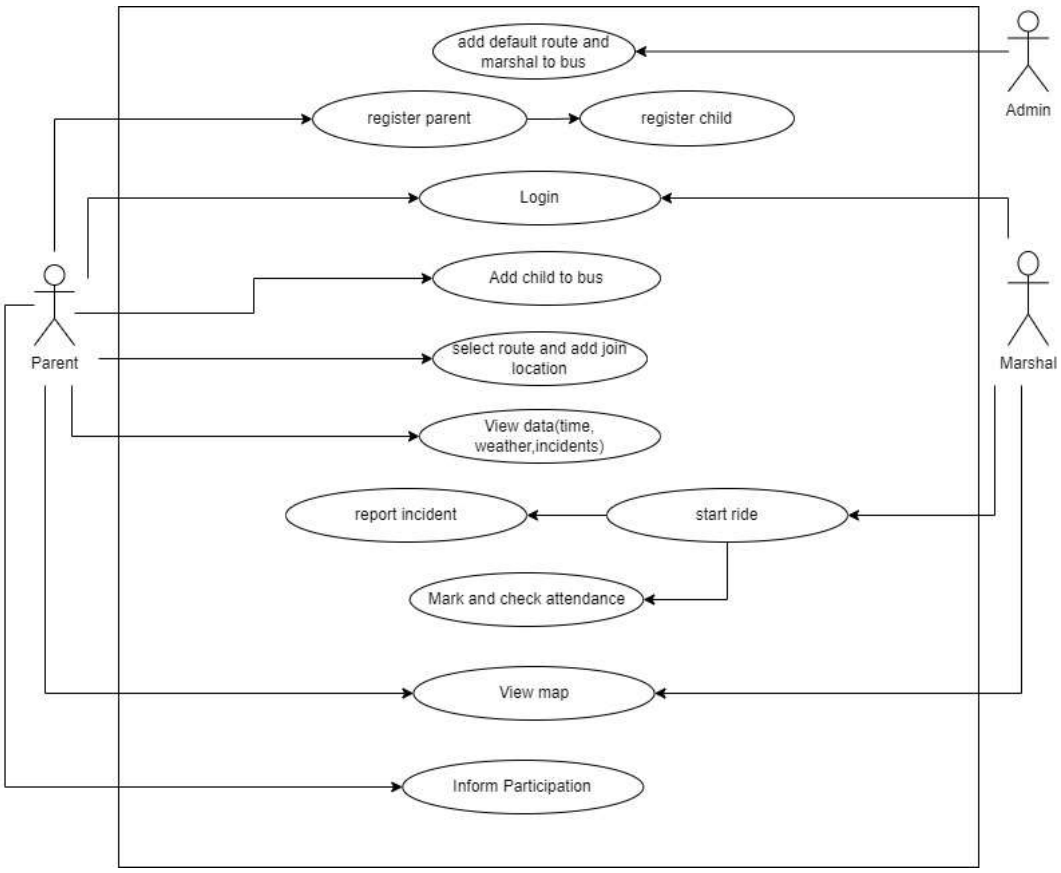

Figure 1.2: Usecase Diagram

1.3 Non Functional Requirements of the System

Non-functional requirements of the system include features such as performance, security and usability.

1.4 Objectives

The objectives of this project are to :

- Deliver a fully functioning mobile app, "Cycle Bus," to coordinate activities associated with cycle buses.
- A web-based administration site to carry out related administrative tasks.
- Ensure the safety of children by facilitating parents to track the real-time location of cycle bus and receive real-time incident reporting.
- Satisfy customers by delivering a user-friendly application with high performance and security, Document the system in various aspects such as specifications, design, methodology and Evaluation.

1.5 Content

- Methodology : This chapter consists of details on how the development process went on, the software development model followed, the testing approach and how issues got fixed.
- Technology Review : This chapter discusses technologies used in building and testing the application.
- System Design : System Architecture of the product, diagrams including Architecture Diagram, Sequence diagram, UML Class Diagram, Database Diagram and Use Case Diagram are included in this chapter.
- System Evaluation : This chapter has the Evaluation of the system against its objectives and limitations.
- Conclusion : This chapter states the summary of context and objectives and the project's outcome.

1.6 Project Resources

GitHub Address : `https://github.com/layaniw/CycleBus.git`

1.7 Main Elements of GitHub repository

- backend : This folder contains all backend files. The sub-folders are five Django apps(authentication, backend, user, bus, channels). Each sub-folder includes its models.py, seralizers.py, views.py, urls.py and apps.py files
- frontend : This folder has all frontend files. Navigation folder with react js files responsible for drawer menu, Views folder with react js files for views, assets folder where image files are stored, shared folder with react js script to set base URL and JSON Web Token and App.js files are sub-folders under the main folder. document : System-related documents and diagrams are here.
- README : Instructions for compiling, deploying and running the project are included in README

Chapter 2

Methodology

- The software development approach followed during this project combines incremental and prototyping models. The software product was divided into smaller components initially, and then prototypes were developed. Having prototypes made the learning and research process easy since the prototyping can begin much more straightforward and then improve along with the knowledge and coding skills gathered. The advantage of prototyping is having working versions of the application and continuously getting feedback from the client. Given his ideas and suggestions, simple prototypes turned into fully functional components and then combined or reused at the end. Since the project idea came from a real-life member of the Cycle Bus community, the requirements of the implemented system were evident. After a few sessions with the client, the project's scope was decided, and requirements were prioritized considering constraints such as timelines. System designing steps such as drawing database diagrams which helped later in implementing Django models, and drawing wire-frames, of the mobile interfaces, helped much in later stages of the development process. Meetings with the client once a week provided an opportunity to discuss already developed functionalities, missing functionalities, what can be improved, issues that emerged while implementing and the future tasks. Feedback from the client contributed much throughout the application's development life cycle.

- Mobile UI s were tested manually on three brands of mobile phones(Huawei, Nokia and Motorola). API testing was done using Postman, where authentication and response time of the requests were tested. To test whether the system retrieves and handles geolocation data accurately, an App called Lockito helped stimulate a fake GPS location with pa-

rameters such as speed.

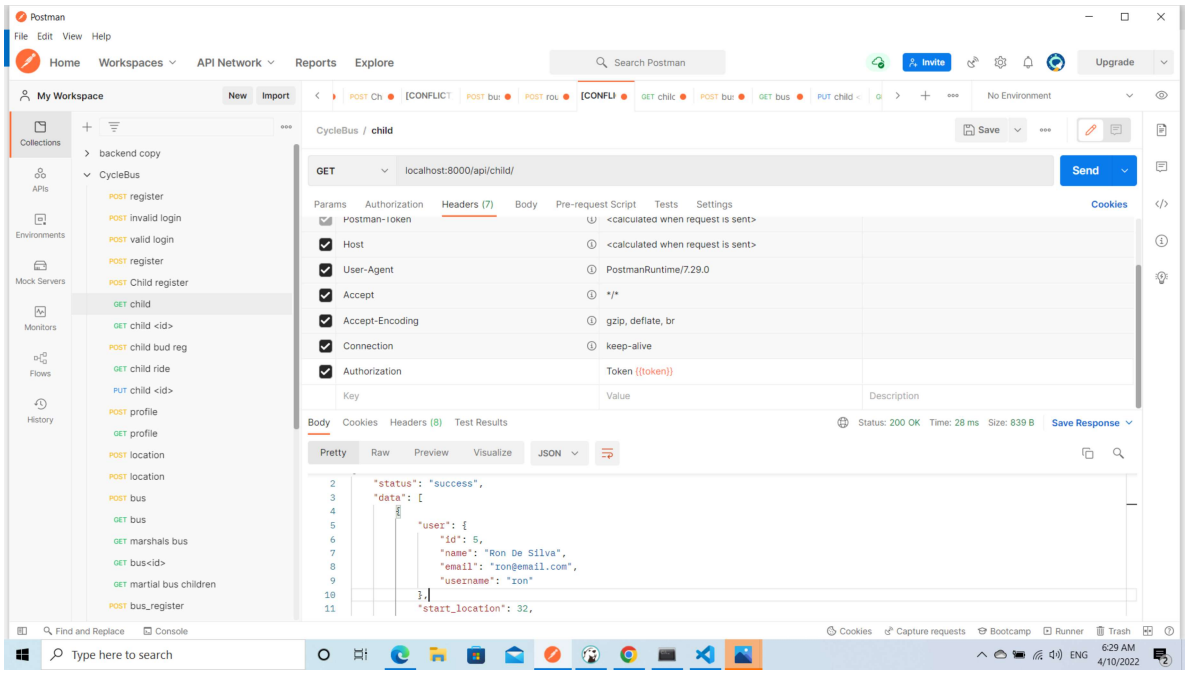

Figure 2.1: API testing with Postman

- The main factors considered to choosing programming languages are learning curve, readability, ability to use cross platforms, timelines and developer community support. Since Python is famous for its simple and clean code, handling backend logic was easier. React Native is a cross-platform language such that there was no need to code separately aiming at Android and iOS devices. Using Django as the Python framework was considered to write backend well structured, cleaned and maintainable. Without spending more time on coding, it helped focus on functionalities more. Then a reliable, fast, secure and extensible database system was needed, and PostgreSQL was a great match. Although Google Maps APIs and Open Weather APIs are not fully open-source, they helped bring together attractive map views and accurate weather data. These factors to consider when selecting these technologies are discussed in detail in chapter: Technology Review

Solving coding related issues was not so problematic since there is huge developer community support for Django and React Native. Django and React Native official cites also provide descriptive information. The client addressed the requirements related issues. Before coding the application, research was

done on selecting the technologies, system design and building system architecture. A considerable amount of research was carried out to decide whether to use the Django default authentication model or JWT Authentication. Basic knowledge of technologies was gained through research, but hands-on experience mostly paved the way to becoming familiar with the technologies.

Chapter 3

Technology Review

- The frontend is written in React Native
- The framework used to build the React Native app is Expo
- The backend is written in Python
- The framework used to write the backend in Python is Django
- Google Map APIs are used to render the map views and write frontend components
- OpenWeatherMap is used to obtain weather details
- PostgreSQL is used as database management system
- Rest API is used as architectural design for the APIs
- Websocket is used as the protocol to build the communication channel
- Postman is used to test APIs

3.1 React Native

React Native is a cross-platform framework for writing Android and iOS mobile applications. It has enabled the development of mobile interfaces for both Android and iOS simultaneously using a single language, JSX, comprised of JavaScript and XML-esque markup. The reason behind this is that React Native consists of React components converted to Native components according to the platform using Objective-C (for iOS) or Java (for Android). Moreover, React Native is based on React, which Facebook used to build user

interfaces. Therefore any developer who is familiar with React can quickly adapt to react native and develop web, iOS or Android applications.

Why use React Native?

- Can simultaneously develop code for all the platforms.
- Hot Reloading : There is no need to rebuild the application to reflect the changes in cooperated. Hitting " Command+R" refreshes your application, eliminating waiting time until the application builds.
- Live Reloading : Developers can code on a separate screen and view the results on another screen.
- Instead of building the libraries from the scratch frameworks such as Shoutem, Expo and native base can be utilized
- Developer support can be gained from huge developer community[2].

Fundamental Components of React Native

1. View

The view is a base component in React Native. It is similar to the **div** element in web development used to separate data into separate sections. Furthermore, a View is used to render elements inside a container and can be styled using CSS. And also, a View can have another View inside as a nested component[3]. e.g.

```
1  import React, { Component } from 'react';
2  import { Text, View } from 'react-native';
3
4  export default class HelloWorld extends Component {
5    render() {
6      return (
7        <View>
8          <Text>Hello, world!</Text>
9        </View>
10     );
11   }
12 }
```

The above code will render a text "Hello, world!" in the mobile view.

2. State

The state is used when data changes inside the component. The current state of data can be accessed inside the script using state. The state is generally initialized in the constructor and can be changed using **setState**. Changed data can be accessed inside the script using **this.state.variable_name**[3]. e.g.

```
import React, {Component} from 'react';
import { Text, View } from 'react-native';

export default class App extends Component {
    state = {
        name: ''
    }
    updateState = () => this.setState({name: 'George'})

    render() {
        return (
            <View>
                <Text onPress={this.updateState}> {this.state.name} </Text>
            </View>
        );
    }
}
```

In the above example name is set as empty when initializing and updated when clicking the Text component. The event **onPress** calls the **updateState** which updates the state of **name** using **setState**.

3. Props

Properties of React Native components are known as **Props**. Parameters that can be passed at the creation of components are known as props. One such instance is **source** property of **Image** component. Props can also be utilized to pass data between components[3]. e.g

```
import React, { Component } from 'react';
import {Text, View } from 'react-native';
```

```
3
4  class Child extends Component {
5    render() {
6      return (
7          <View >
8            <Text >Hello {this.props.name}!</Text>
9          </View>
10     );
11   }
12 }
13
14 export default class Parent extends Component {
15   render() {
16     return (
17         <View >
18           <ChildClass name='George' />
19           <ChildClass name='Anne' />
20         </View>
21     );
22   }
23 }
```

Here **name** is passed as a **prop** to React Native base component **Text** Output of above example is

```
1  Hello George!
2  Hello Anne!
```

4. Style

All base components can be given a prop called **styles**. Styles are written in JavaScript. Styles can be coded inline or written using **StyleSheet** component. Similar to **CSS** in web development other than react native styles use names in camel case(font-size in **CSS**, fontSize in **JavaScript**)[3]. e.g.

```
1  import React, { Component } from 'react';
2  import { StyleSheet, Text, View } from 'react-native';
3
4  export default class Example extends Component {
```

```
    render() {
        return (
            <View>
                <Text style={{ backgroundColor: '#a7a6a9', color: 'yellow', fontSize: 20 }}>Th

                <Text style={styles.textStyle}>This is style passed using StyleSheet</Text>
            </View>
        );
    }
}
const styles = StyleSheet.create({
    textStyle: {
        color: 'gray',
        fontWeight: 'bold',
        fontSize: 30,
    },
});

```

Limitations of React Native

- Since React Native uses a bridge to convert code to native components during run time performance can be hindered compared to technologies such as Flutter [4]

3.2 Expo

Expo is a set of tools that can be used to develop, build and deploy React Native and Native code based applications on both Android and iOS devices. **Expo CLI** is the developer tool and **Expo Client**. The application can be shared through a link or by scanning a QR code where as in **React CLI** the .apk and .ipa files should be shared. Devices should be connected to the same LAN(Local Area Network) to access the development server when using Expo[5][6].

Why use React Native?

- Easy to set up.
- Can build and deploy applications for both Android and iOS.

- Can run applications without **Xcode** or **Android Studio**.
- A lot of native features such as Camera, file system, location, social authentication and push notifications are bundled with Expo. Therefore no need to in-cooperate those separately.
- Application gets updated quickly since no native code and easy to publish[5][7].

Drawbacks

- React Native Application which is abstracted layer is again wrapped in a abstract layer when using Expo[5].

3.3 Python

Python is an object-oriented language used by many famous companies such as Quora, Netflix, YouTube, Pinterest and Google and APIs used in the mobile application can be written using Python. Moreover, Python is straightforward to implement and use due to its clean structure and simplicity compared to other languages. The learning curve is also relatively less[8].

Why use Python for the backend

- Easy to adapt within limited time constraints due to its simplicity.
- High readability because of clean code and simple syntax.
- There are several libraries to achieve programming tasks.
- Python frameworks like Django and Flask can facilitate fast development and improve the quality since those offer great structuring to the code[9][10].

Drawbacks of using Python

- Python may utilize a huge amount of memory due to the flexibility of a variable's data type, making it unsuitable for memory-efficient applications.
- Python is slow at run time because it is an interpreted language. Unlike languages like Java, python code is executed line by line by the interpreter. Moreover, the data type of Python variables is assigned during the run time since there is no need to declare the variable data types when declaring.

3.4 Django

Django is an open-source framework to write clean and structured python code. It comes with many functionalities such as authenticating, URL routing, Object Relational Mapper and database migrations. Famous companies like Instagram use Django as their backend framework.[8].

Why use Django as the backend framework ?

- Django provides a **REST** framework for APIs, a Python library that enables writing an API in a few lines of code.
- **Django Channels** can be used to build asynchronous communication between the client and server, facilitating real-time updates.
- Another fascinating feature of the Django framework is the default Admin site, which can manage data in an application. It is easy to set up and can be customized using hooks.
- Secured authentication can be implemented since **JSON Web Token Authentication(JWT Authentication)** is supported by Django[11][12].

Important features in Django

1. Object Relational Mapper(ORM)

ORM can interact with the database, the same as SQL. The advantage here is developer does not need to write SQL; ORM automatically does this. It maps object attributes to respective database table fields. e.g.

```
class Parent(models.Model):

    name = models.CharField(max_length=200)
    email = models.EmailField(max_length=254)

class Child(models.Model):

    name = models.CharField(max_length=200)
    parent = models.ForeignKey(
        Parent,
        on_delete=models.CASCADE,
```

```
12          related_name='Vehicle')
13
```

2. View and URL

Django maps the requested URL to a URL from the set defined in the URLs file when a user requests a server. Then the matched URL calls the relevant View associated with itself. The View handles the request, connecting to the database by querying through model objects in the models file and constructing the response to be sent to the client e.g.

```
1  from django.shortcuts import render
2  from django.http import HttpResponse
3
4  def Home(request):
5      return HttpResponse('this is the view of the home')
6
```

```
1
2  from django.urls import path
3
4  from . import views
5  urlpatterns = [
6      path('home/', views.Home, name='home')
7  ]
```

Drawbacks of Django

- Django can not handle multiple requests at the same time[13].

3. JSON Web Token(JWT) Authentication

Django has its authentication model, a session cookie-based system which can manage users, user groups, and permissions. However, it supports JWT Authentication too. JWT authentication is a model consisted of a **accessToken**, by concatenating **userID** and **expiresIn** encrypted with **ACCESS_TOKEN_SECRET**. The following example shows how to generate a JSON Web Token, user's ID, and expiry date of 60 days is encrypted.

```
    def _generate_jwt_token(self):

        dt = datetime.now() + timedelta(days=60)

        token = jwt.encode({
            'id': self.pk,
            'exp': int(dt.strftime('%s'))
        }, settings.SECRET_KEY, algorithm='HS256')

        return token.decode('utf-8')
```

This accessToken is sent to the client by the server and client saves it on the client side. Then with every subsequent request client sends the accessToken. There is no need of a sessionDB to save accessToken in JWT authentication as in session cookie based authentication. Once client sends a request with accessToken in the header it is decrypted back to userID with ACCESS_TOKEN_SECRET. Therefore it is considered somewhat fast than session cookie based approach[14].

4. Django Channels

While supporting HTTP requests, Django also has **Django Channels** which can handle asynchronous long time connections by supporting protocols like **WebSockets**. Without waiting for the user to refresh the browser to see the updates, Django Channels keeps on pushing updates to its registered consumers [15][16].

5. Django REST APIs

REST API is the standard convention to build an API. **Django ResT Framework** in Django has made implementing REST APIs easy. Django Rest Framework URLs define how the end-user interacts with data using HTTP methods such as GET, POST, PUT and DELETE. Django Serializer is responsible for converting model objects to JSON format(Serializing) to be transferred via HTTP request and converting JSON data to model objects. e.g

```
from rest_framework import serializers
from .models import Child
```

```

class ChildSerializer(serializers.ModelSerializer):
    parent_name = serializers.RelatedField(source='parent', read_only=True)

    class Meta:
        model = Child
        fields = ('id', 'name', 'parent_name')
```

Fields can be validated here as parent_name is set as required unless the request has parent_name that request will not be processed. Then post(), get(), delete(), patch() methods can be used in the APIView class to perform operations on a model. e.g.

```
from rest_framework.views import APIView
from rest_framework.response import Response
from rest_framework import status
from .serializers import ChildSerializer
from .models import Child

class ChildAPIView(APIView):
    def post(self, request):
        serializer = ChildSerializer(data=request.data)
        if serializer.is_valid():
            serializer.save()
            return Response({"status": "success", "data": serializer.data}, status=status.HTTP
        else:
            return Response({"status": "error", "data": serializer.errors}, status=status.HTTP
```

Next the endpoints to use above post() method should be exposed and the ChildAPIView class registered as a view for the user[17][18].

Drawbacks of Django

- Django can not handle multiple requests at the same time[13].

3.5 Google Maps API

Google Maps API is a technology offered by Google. Maps, Routes and Places support provided by Google Maps API makes it easy to develop functionalities such as

- Geocoding : Converting an address, eircode in to coordinates and vice-versa.
- Mark a route along way-points according to many parameters like travel mode, optimize route and traffic data.
- Geolocation : Accessing the device and get an approximate location.
- Providing map views and marking locations in different styles.
- Distance Matrix: Providing travel time and distance for a set of origins and destinations[19].

3.6 Open Weather API

Open Weather API provides real-time and accurate climate, environmental, weather and ocean data through a single API for any location given a particular time. While having high performance with a response rate of only a few milliseconds, Open Weather API provides plenty of weather, climate, environmental and ocean parameters such as temperature, humidity, wind speed and evaporation rate[20].

Chapter 4

System Design

4.1 System Architecture

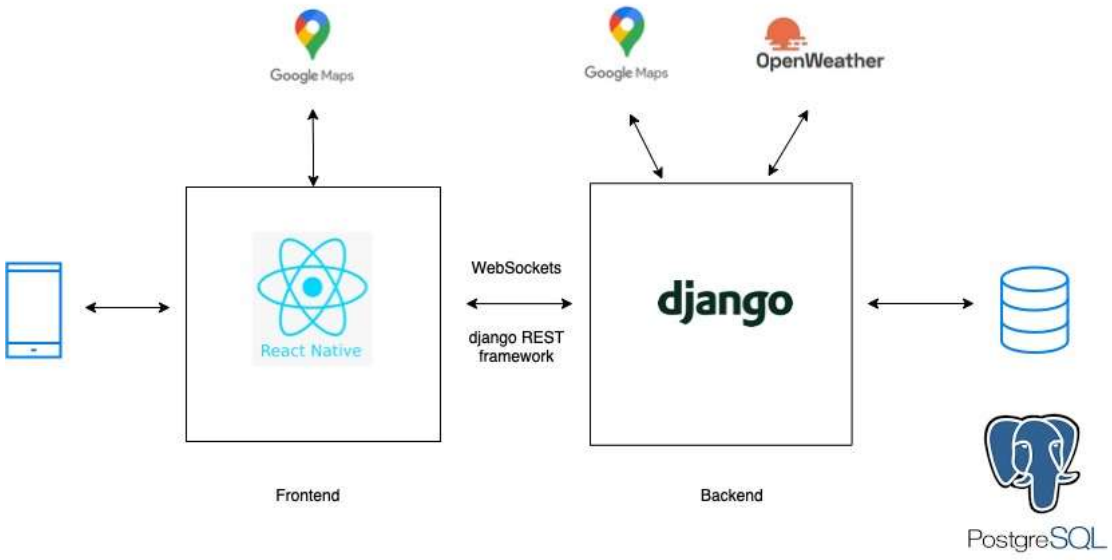

Figure 4.1: System Architecture

The implemented system almost follows **Model View Controller(MVC)** architecture where the Model interacts with the database, performing data updates or retrievals; the View represents actual data presented to the user and the Controller deals between Model and View, handling the logic. The backend framework is handled by Django, which corresponds to **Model Template View (MTV)** architectural pattern. The View in Django is similar to The controller in MVC, Model in Django, is the same as in MVC and

Templates in Django relate to View in MVC. However, in this project, Templates are not used; instead, Django Rest Framework handles the View[21].

As in Figure 4.3 Backend and Frontend components are connected together by the Django Rest Framework. Models in Django handle all behaviour and relationship attributes related to the data stored in the PostgreSQL database. When a user submits a form or views details, the HTTP request sent by the client will be mapped to an URL. Then it is passed to the relevant Django View, and HTTP requests and responses are handled by the View with the help of Django Serializer, responsible for serializing(converting JSON data to Django Model objects) and deserializing(converting Django Model objects to JSON).

WebSockets implemented using Django Channels are used for real-time updates of the locations. The application developed there Parents in the consumer group where Marshal pushes updates. The need here was to show the location of Marshal while he moves along a route without the need for a Parent to refresh the view. Furthermore, WebSockets are used for incident reporting. Parents get immediate updates of incidents reported by Marshal during the ride.

For the map views shown on mobile interfaces **React Native Map** components like **MapView** and **Marker**, powered by Google, are used in the frontend. This application uses **Directions** component for **react-native-maps**, an important feature powered by **Google Map Directions API** to mark the route of the Marshal when the initial location and destination are given. It can be imported and rendered under **MapView** component, and many types of props such as origin, destination, apikey, waypoints, mode and optimizeWaypoints can be given to obtain the direction and then mark a **MapView.Polyline** as the route[22]. Since the route of the Marshal needed to be lied along a set of locations to pick up children waypoints prop is used to pass an array of join locations of children, and optimizeWaypoints prop is set to false to avoid optimization of waypoints to mark the route. **Google Maps Distance Matrix API** is implemented in Django backend to retrieve time for Marshal to reach Child's join location, and parameters such as origins, destinations and mode can be given to customize the request[19]. Frontend access this data through an API. **Google Text Search API** is used in the backend to convert eircodes for start location and end location of Child sent by frontend via an API. Serializer converts these eirocdes to coordinates before saving them to the database. Text Search API can be given parameters like output(JSON or XML), and query(string on which to search)returns information on a set of places according to the string given[23].

Open Weather API is used to obtain weather conditions at a particular location(Marshal's location) and time(current time). This is also imple-

mented in the backend and accessed via an API. Open Weather API returns data for numerous parameters, but only wind speed, temperature, and rain data are used in this application.

4.2 Sequence Diagram

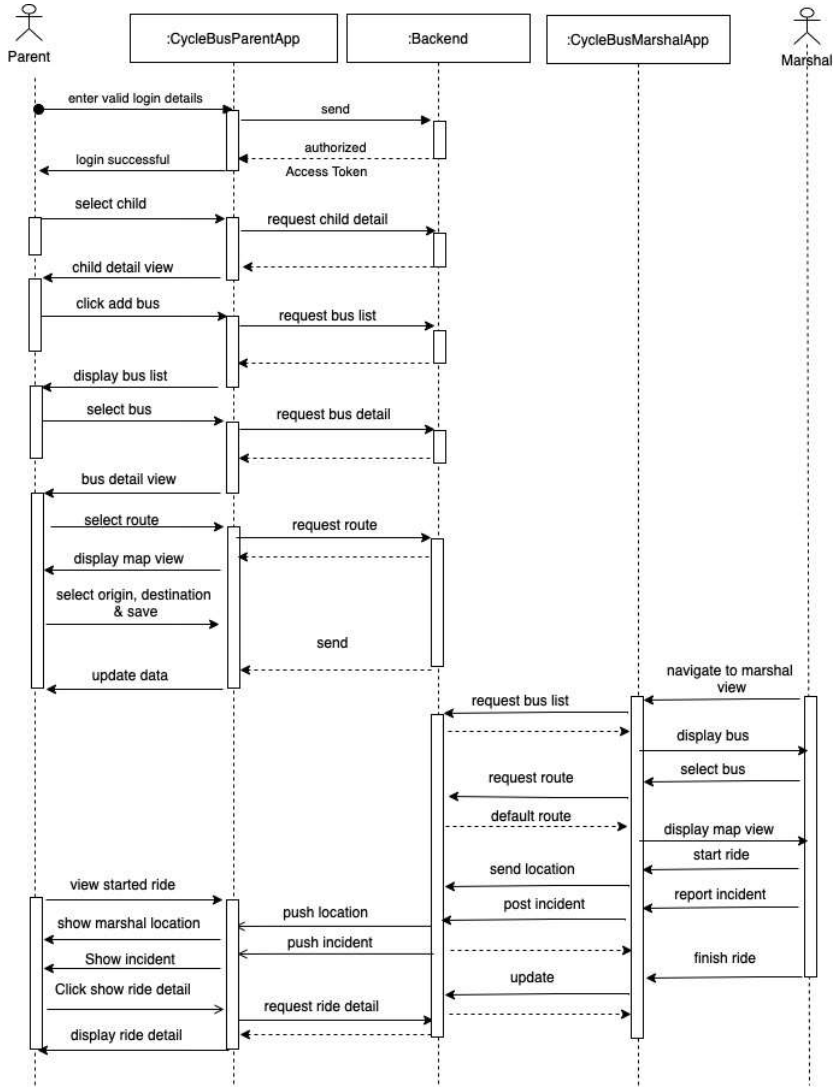

Figure 4.2: Sequence Diagram

4.3 UML Class Diagram

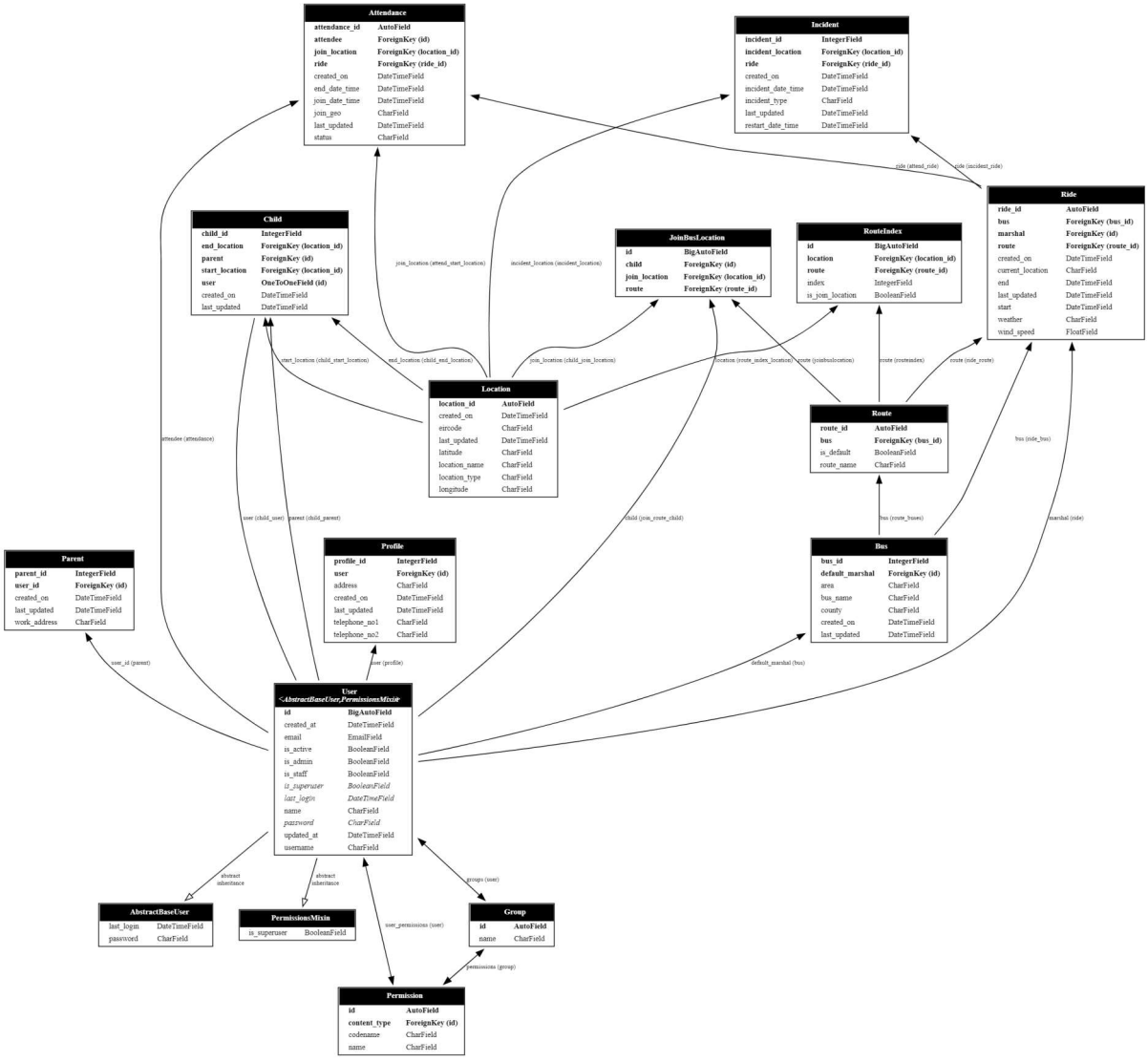

Figure 4.3: UML Class Diagram

Chapter 5

System Evaluation

- A manual test approach was followed to test mobile user interfaces. Functional testing to verify whether the features match the system's requirements and non-functional testing to ascertain the user-friendliness, consistency in styling and whether the system performs as intended were carried out. Styles were consistent on various brands of mobile phones.
- API performance testing was performed using **Postman**. It was verified that response time for every request is approximately less than five hundred milliseconds. Found out that the authentication system is working fine by trying to send requests without access token in the headers.This proved that only authenticated users can gain the access to server through APIs.

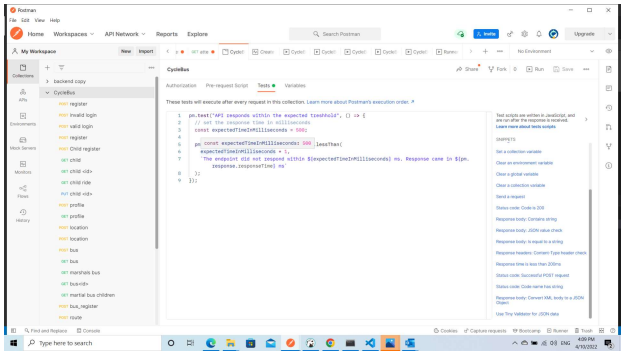

Figure 5.1: Code to Check API Response Time

- A demonstration of the working application was delivered to real-life members of the cycle bus community to get feedback that falls into usability testing.

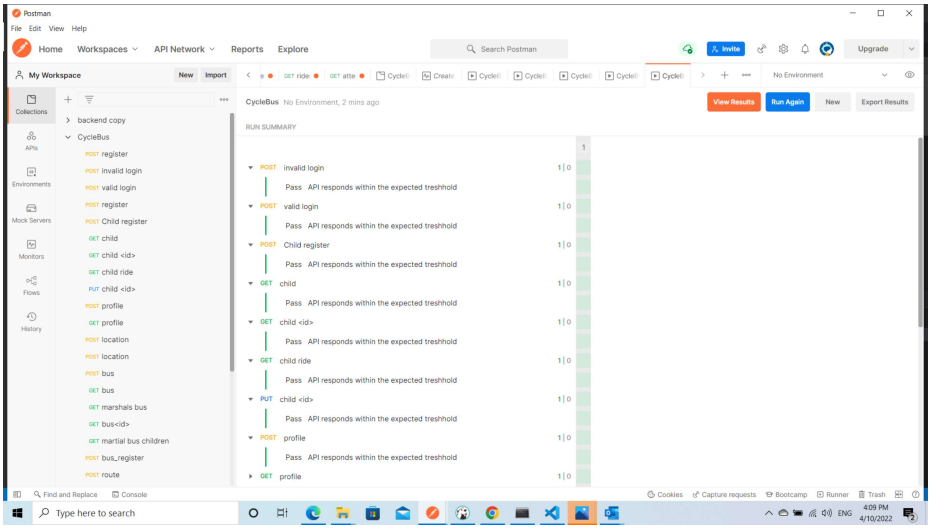

Figure 5.2: Results of API Response Time Testing

- As per objectives mobile application that accommodates the defined requirements could be delivered along with a web-based Django administration site.

5.1 Limitations

- Currently, obtaining GPS location only works when the application is open. This should be improved such that the system can retrieve GPS location even when the app is killed or running in the background.
- Marshal can be changed only by the admin user. This should be facilitated in a way that Marshal can request changes and be approved by the administrator.
- If needed application can be modified to show the child's location to the parent, a feature this system does not currently provide.

Chapter 6

Conclusion

The main objective of this software development project was to develop a mobile application that can accommodate the requirements of the cycle bus community and safeguard the child's security by allowing the parent to track the location of the marshal, which is the approximate location of the child. This primary purpose is successfully built in the Cycle Bus app. Real-time data such as incidents, weather details, and arrival times are also displayed to parents during the ride. Marshal can mark the participants while or after the ride again, facilitating the child's security. The administration site is also implemented to admit tasks such as setting the bus's default marshal and bus's default route. The knowledge acquired is immense by researching technologies, various approaches, and best practices for software development.

6.1 Outcomes

- A mobile application that can support cycle bus community along with web based administration site.
- Document on concepts such as system design, software development approach and methodology followed during development.
- Knowledge gained by research and hands-on coding experience.
- Experience of a development environment similar to the software industry stimulated by interaction with the customer needs and feedback.

Currently the system is not hosted in a server which will fall into future works.

Bibliography

[1] G. Schools, "Cycle bus network." https://www.meteomatics.com/en/weather-api/?

[2] K. Shah, "Advantages and disadvantages of react native development in 2022." https://www.thirdrocktechkno.com/blog/pros-and-cons-of-react-native-development-in-2021/.

[3] C. Facebook Supported, "React native." https://reactnative.dev.

[4] J. K, "React native: Features, limitations, pros and cons." https://acodez.in/react-native-features/.

[5] A. Ravichnadran, "Building react native apps - expo or not?." https://adhithiravi.medium.com/building-react-native-apps-expo-or-not-d49770d1f5b8.

[6] Expo, "Introduction to expo." https://docs.expo.dev.

[7] O. Bespalko, "React native init vs expo 2021: What are the differences?." https://fulcrum.rocks/blog/react-native-init-vs-expo/.

[8] K. Hill, "Why has python become a popular choice for mobile app development in 2020?." https://dzone.com/articles/why-python-has-become-a-popular-choice-for-mobile.

[9] H. Technologies, "Why use python for developing your product's backend?." https://medium.com/@Henote/why-use-python-for-developing-your-products-backend-b467fbad8b0f.

[10] MicroPyramid, "Why choose python as backend development?." https://micropyramid.com/blog/why-choose-python-as-backend-development/.

[11] S. Daftari, "10 reasons why django web development with python is most popular for backend web development." https://www.kelltontech.com/kellton-tech-blog/why-django-web-development-with-python-for-backend-web-development.

[12] Django, "Getting started with django." https://www.djangoproject.com/start/.

[13] S. Bhatt, "Pros and cons of django framework for app development." https://dzone.com/articles/pros-and-cons-of-django-framework-for-app-developm.

[14] thinkster, "Setting up jwt authentication."

[15] G. Singhal, "Django channels and websockets." https://blog.logrocket.com/django-channels-and-websockets/.

[16] RealPython, "Getting started with django channels." https://realpython.com/getting-started-with-django-channels/.

[17] RealPython, "Django rest framework – an introduction." https://realpython.com/django-rest-framework-quick-start/.

[18] N. Jatu, "Creating a rest api with django rest framework." https://stackabuse.com/creating-a-rest-api-with-django-rest-framework/.

[19]

[20] meteomatics, "Our weather api." https://www.meteomatics.com/en/weather-api/?

[21] A. N. Oyom, "Understanding the mvc pattern in django." https://medium.com/shecodeafrica/understanding-the-mvc-pattern-in-django-edda05b9f43f.

[22] npm, "react-native-maps-directions." https://www.npmjs.com/package/react-native-maps-directions.

[23] Google, "Text search." https://developers.google.com/maps/documentation/places/web-service/search-text.

Chapter 7

Appendices

GitHub Repository URL : `https://github.com/layaniw/CycleBus.git`

7.1 Installation Steps

Follow the README in above GitHub repository to set up the project in your computer and device. `https://github.com/layaniw/CycleBus/blob/d3852dfe60db2cc1a782ed8c7c288039e4c50001/README.md`