VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

SPECIALIZED PROJECT

BUILD A MOBILE APP FOR DATA COLLECTION AND URBAN TRAFFIC ESTIMATION

Major: Computer Science

THESIS COMMITTEE: Information System
SUPERVISORS: Assoc. Prof. Tran Minh Quang
Ms. Eng. Bui Tien Duc
REVIEWER: Dr. Phan Trong Nhan
STUDENT 1: Nguyen Minh Khoa - 2052538
STUDENT 2: Nguyen Nho Gia Phuc - 2052214

Ho Chi Minh City, December 17, 2023

Date of Signature

Associate Professor, Tran Minh Quang, PhD
Faculty of Computer Science and Engineering
Supervisor

DECLARATION OF AUTHENTICITY

We would like to state that to project "Build a mobile app for data collection and urban traffic estimation" is our own work under the guidance of Assoc. Prof. Tran Minh Quang and MSc. Bui Tien Duc. Related knowledge, information and research works that we refer to will be clearly noted in the references section. We are responsible for the truthfulness of our topic.

Ho Chi Minh, December 2023

Students

Nguyen Nho Gia Phuc

Nguyen Minh Khoa

ACKNOWLEDGEMENT

First of all, we would like to express our gratitude to Associate Professor Tran Minh Quang for his guidance and support throughout the course of the project. His expertise and knowledge have been invaluable in assisting us in overcoming the challenges we have faced. His help has been pivotal in shaping up our research direction and providing us with the necessary resources to complete our project.

We are also grateful to Master of Science Bui Tien Duc for his assistance in the development and deployment of the UTraffic server, as well as the access to necessary infrastructure for the project. His prompt and helpful responses to our inquiries have been extremely beneficial.

Finally, we would like to express our gratitude to the Faculty of Computer Science and Engineering for providing us with the opportunity to work on this project. We are grateful for the support and encouragement we have received from our professors and classmates.

Ho Chi Minh, December 2023

Students

Nguyen Nho Gia Phuc

Nguyen Minh Khoa

ABSTRACT

This project addresses the need to enhance the functionality of a dedicated mobile application designed for data collection and urban traffic estimation. The primary focus is on integrating bus and parking-related services, coupled with extending support to a broader range of operating systems. The overarching objective is to optimize the application's efficiency in furnishing real-time insights into urban traffic patterns. The study involves a meticulous examination of current app limitations and challenges, followed by the strategic design and implementation of upgrades.

The research methodology entails a comprehensive analysis of user requirements, feedback, and technological constraints. The integration of bus-related services involves data crawling from reliable sources, ensuring the provision of accurate and up-to-date information on routes, schedules, and other critical details. Simultaneously, the integration of parking services seeks to create a seamless user experience for locating, booking, and remunerating parking spaces within urban areas.

The anticipated outcomes of this project aim to encourage sustainable transportation choices and mitigate urban traffic congestion. The study's findings provide valuable insights into the intricacies of upgrading mobile applications for data collection within the domain of urban traffic estimation. The integrated services offer a comprehensive solution to urban mobility challenges, contributing to a more efficient and sustainable urban transportation ecosystem.

In this initial phase, crucial milestones have been achieved, including the conceptual design for parking services and the successful acquisition of bus data from online sources. The establishment of endpoints for various bus-related services is a major step toward the integration of bus services into mobile applications. Additionally, an early iOS frontend implementation has been initiated, showcasing the user interface and functionality for Apple devices. These accomplishments underscore substantial progress in enhancing the application's capabilities for data collection and urban traffic estimation while integrating essential bus services.

Contents

List of Figures

List of Tables

1 INTRODUCTION

1.1 Motivation

Ho Chi Minh City is a thriving metropolis known for its vibrancy and economic importance, but it also has a persistent problem with traffic congestion. An astounding estimated cost of 6 billion USD per year [1] highlights the crippling effects of traffic congestion on Ho Chi Minh City, indicating not only the financial toll on businesses but also the pressing need to put into practice practical solutions for sustainable urban development. Within this context, the project's driving force is the necessity to improve the current mobile application aimed at facilitating better traffic conditions in Ho Chi Minh City. [2]

Figure 1: Traffic congestion at the Hang Xanh intersection at the end of 2022.

The current measures taken by the government to alleviate traffic congestion in Ho Chi Minh City have mostly failed, despite their concerted efforts. The rapid urbanization and increasing vehicular density have posed challenges for infrastructure projects and traffic management initiatives to keep up with. Moreover, the current deficiencies in the public transportation system have hindered its capacity to function as a viable alternative for a significant portion of the population. The insufficient effectiveness of these measures underscores the intricacy of the challenge, requiring a thorough reassessment and inventive strategy to attain significant and enduring outcomes.

1.2 Available solutions

In response to the pressing issue of traffic congestion in Ho Chi Minh City, various measures have been implemented to alleviate the strain on the urban transportation system. One notable initiative involves ongoing investments in infrastructure development, with the construction of new roads and the expansion of existing thoroughfares aimed at accommodating the city's growing population and vehicular density. Additionally, the government has expanded public transportation services, such as buses and the metro system, as a means to encourage citizens to opt for more sustainable modes of commuting.

A different strategy for addressing traffic congestion involves the creation of specialized programs for the purpose of continuously monitoring the situation in real-time. These applications utilize technology to deliver real-time updates on traffic conditions, acknowledging the demand for immediate information. Through the collection and analysis of data from diverse sources, including GPS-enabled devices and surveillance cameras, these applications enable users to make well-informed decisions regarding their travels. An example is the TTGT HCMC application created by the Department of Transportation in Ho Chi Minh City. This application functions as a vital instrument, providing users with instant access to real-time updates on the status of routes and visual information through a network of 685 strategically positioned cameras throughout the city. These innovative technologies enhance the ability to effectively manage traffic congestion in urban environments by employing a dynamic and adaptable approach.

1.3 Problem statement

UTraffic [3] is a city traffic prediction system that relies on data gathered from the local population developed by Professor Tran Minh Quang, alongside his associates and students at Ho Chi Minh University of Technology. UTraffic's products aim to mitigate traffic congestion and enhance safety and convenience for users. Using big data analytics and machine learning, the solutions estimate and forecast traffic conditions accurately and quickly, warn users of traffic jams, support efficient routing that takes traffic conditions into account, and provide statistical information and forecasts to help managers make decisions. However, despite UTraffic's commendable objectives and the overarching focus on the mobile application and backend technologies, several challenges have surfaced in the system's current implementation that require careful consideration and resolution:

- In addition to impeding thorough data collecting and insights into traffic patterns in Ho Chi Minh City, the lack of an iOS version restricts our ability to interact with the sizable iPhone user base.
- In addition to relying on crowd-sourced data for efficient routing, our existing approaches to traffic congestion are not sufficiently diverse.

1.4 Scope and objectives

The UTraffic app and backend, developed by previous teams, serve as the basis for our efforts to enhance and innovate Ho Chi Minh City's traffic circumstances. The objective of the project is to expand the number of users by creating an iOS version of the current Android application. The objective of this iOS expansion is to enhance the accessibility of the UTraffic application, thereby encouraging a broader range of users. Objectively, the introduction of supplementary bus-related services is intended to encourage the greater use of public transportation. These services will provide users with up-to-date information about bus routes, schedules, and other information, improving the attractiveness of sustainable commuting choices. At the same time, with the emergence of research groups with similar interest in the traffic field, making the UTraffic mobile app in particular and the whole UTraffic system in general a place to showcase and deliver those new features will be one of our focus points. The project aims to make a substantial contribution to the improvement of urban mobility in Ho Chi Minh City through these collaborative efforts.

2 RELATED RESEARCH

2.1 Bus-Related Services

Previous groups' research has investigated traffic-related value-added services, such as nearby hospitals and automatic teller machines [4]. One major candidate for us to continue this research direction is the addition of buses and public transport as a whole. Our research investigates a range of existing applications that provide services related to buses. Through a thorough analysis of established solutions within this field, our objective is to acquire insights that will guide the improvement of our application by finding a compromise between the current system's strengths and weaknesses.

The existing solutions in question are the package offered by FPT in the names of Go!Bus TPHCM and the Buyt TPHCM website, as well as BusMap by Phenikaa MaaS JSC.

2.1.1 The FPT Ecosystem

Besides Go!Bus TPHCM, FPT offers a wide set of choices for citizens to track traffic data, and these services are linked together. Most notably, the TTGT Tp Ho Chi Minh app allows users to see public cameras for traffic. We place particular emphasis on Go!Bus as it aligns with our project's focus, and Buyt TPHCM holds significance for data collection, as detailed in subsequent sections. These applications play pivotal roles in informing our research and development efforts.

2.1.1.a Go!Bus

Fundamentally, Go!Bus offers solely bus-related services like bus stop searching, bus route searching, and routing. The app also has some support for Grab when routing. The home screen for the app is as follows:

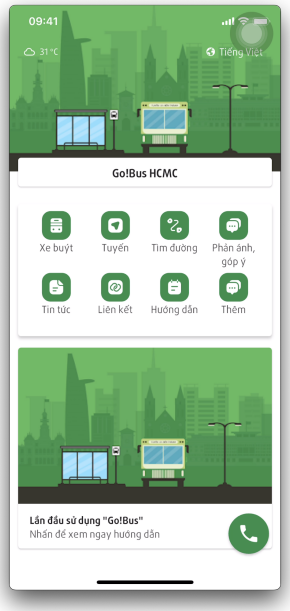
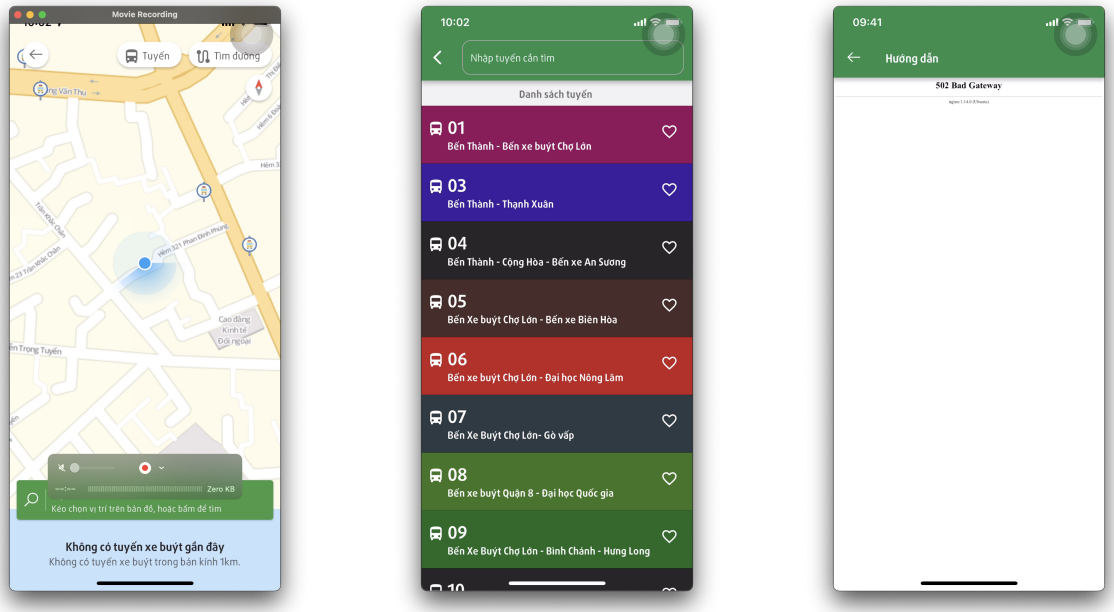


Figure 2: The home screen for Go!Bus.

Nevertheless, during our testing of these features, none of them functioned, or had inconsistent uptimes. Here are a few examples of these occurrences.



(a) No data for nearby routes.

(b) The stops do not show anything when pressed.

(c) Embedded web view with server error.

Figure 3: Go!Bus functionalities problems

It also portrayed some mobile app bad practices, such as embedding an excessive number of web views into the user interface. Apple, in particular, is not fond of this approach and publicly stated that apps using this will not be accepted in the future [5]. This is one of the points that we will take into consideration when developing our iOS app.

2.1.1.b Buyt TPHCM website

Buyt TPHCM website is an online resource designed to provide comprehensive information and tools for navigating the public bus system in Ho Chi Minh City. This website aims to enhance the accessibility and user experience of public transportation within the city by offering the following key functionalities:

- **Detailed route information:** The website provides users with comprehensive information on all bus routes in Ho Chi Minh City, including individual stop locations, schedules, and route maps.
- **Real-time bus arrival predictions:** To promote efficient travel planning, buyttphcm.com.vn offers real-time bus arrival predictions for all routes, allowing users to make informed decisions about their travel time.
- **Convenient tools and resources:** This website provides access to the Go!Bus mobile application, enabling users to easily purchase tickets and plan their bus journeys on the go. Additionally, users can find helpful information about the Public Transport Management Center and other relevant resources.
- **Up-to-date news and information:** buyttphcm.com.vn keeps users informed of the latest devel-

opments in the Ho Chi Minh City bus system, including temporary route changes, service disruptions, and new initiatives.

With its regularly updated resources, Buyt TP HCM will play a crucial role in our data collection process for bus stops and routes in the later stages of our project.

2.1.2 BusMap

BusMap rose as a prime solution for city navigation using buses. It also expanded into other regions and countries like Thailand and is overall doing well.

The fundamental bus services like searching and routing are executed well on BusMap, bar some UI lag that could be due to the excessive inclusion of map views. Users can see the bus locations that are coming their way with an estimated time of arrival. There are also other services offered by BusMap, such as JobMap, MotelMap, and driver license support. However, sections like "Blogs" and "News" might be unnecessary for most users.

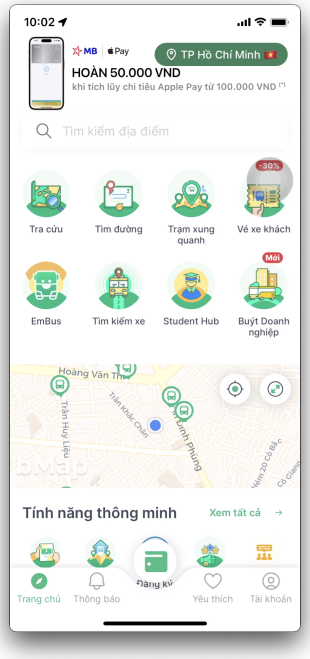


Figure 4: The home screen for BusMap

2.1.3 Conclusion

Both of these apps share a common feature of being equipped with an abundance of supplementary services. While Go!Bus TPHCM is compatible with Grab, BusMap has diversified its partnerships to include other businesses. One apparent limitation of UTraffic is the lack of access to real-time data, such as the current locations of buses and the duration until the next journey. Nevertheless, with sufficient effort and development, our application can excel as a dependable routing and search engine. Ultimately, the primary function of UTraffic is to process congestion data; therefore, we must be careful not to delve too thoroughly into other domains.

2.2 Parking services

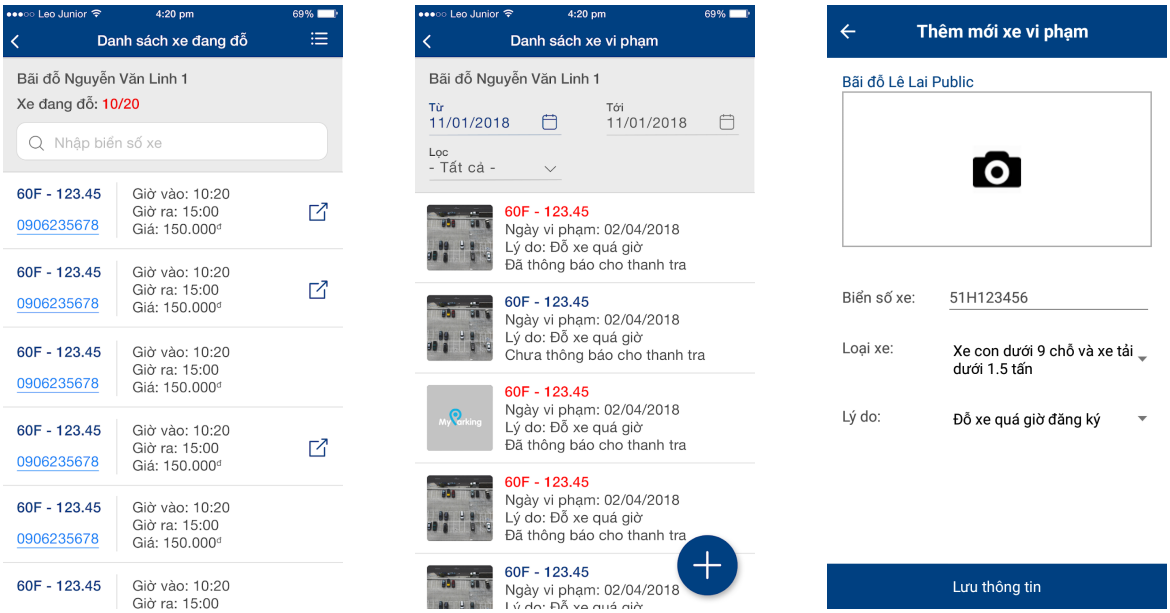
The need to add parking services to our application originates from the pervasive issue of road congestion, which is compounded by the extensive use of parked automobiles on streets. The existing condition complicates traffic flow and reduces road space availability. Parking services not only handle the difficulty of finding acceptable parking places but also provide customers with compelling incentives for app usage. The availability of simple and efficient parking options encourages consumers to make better use of allotted parking spots. This, in turn, improves the overall user experience while also helping to alleviate traffic congestion by freeing up valuable road real estate.

The existing solutions predominantly reside within Viettel's myparking ecosystem, consisting of two mobile applications: "My parking - Security" and "My parking." The former is a parking management system for security guards, while the latter is a parking search engine for users. We will focus on the latter as it is more relevant to our research.

The system operates by Viettel advertising their parking lot management application to parking lot owners. Subsequently, they leverage the data collected to populate the database of the application, catering to the general public's parking lot search requirements.

2.2.1 Myparking for owners

While our exploration of the owner side of the ecosystem is constrained due to the group's inability to register as a parking lot owner, alternative avenues enable us to gain insights into the application. The app's website [6] informs us that they offer diverse services, including parking lot advertising, camera surveillance, and analytics. Moreover, the application's page on Google Play provides us with a brief preview of their UI.

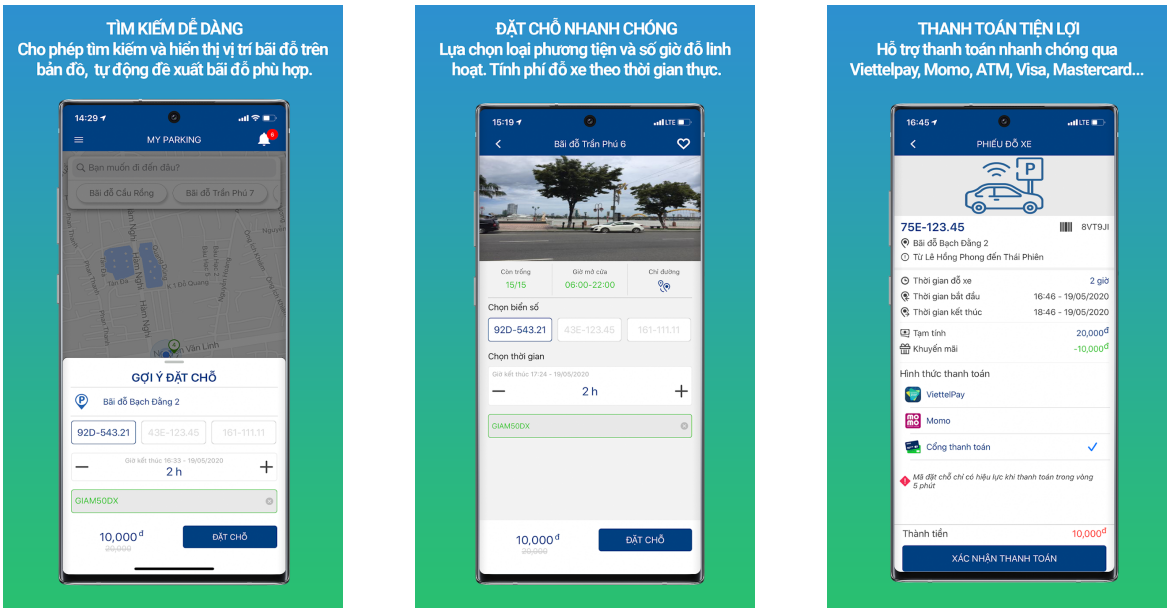


(a) Lot serveillance (b) List of vehicles with infractions (c) Add a new infration

Figure 5: Myparking for owners - UI

2.2.2 Myparking for general users

The application's intended functions are to provide users with a parking lot search engine, book empty lots, and digital payments. The illustrations for the features are as follow:



(a) Look for parking lots (b) Book empty lots (c) Make digital payments

Figure 6: Myparking's functionalities for general users

2.2.3 Conclusion

Viettel initially enticed parking lot owners by offering attractive incentives to integrate their services into the application, creating a comprehensive platform for users to search, book, and pay for parking spaces. Despite these promising features, the application has faced significant criticism stemming from inadequate administration and maintenance. Users have reported a range of issues, including the improper implementation of payment methods, inaccurate price and availability information, and instances where parking lot owners charge fees both through the app and directly. This dual charging system has led to frustration and mistrust among users, tarnishing the overall reputation of the application. The negative backlash underscores the importance of robust oversight and effective maintenance in ensuring the smooth operation and trustworthiness of such technology-driven services.

2.3 HCMUT TrafficView

HCMUT TrafficView is a project developed by the Ho Chi Minh City University of Technology's students that allow users to view the traffic cameras placed around the city. Via the techniques of data labeling and image processing, the project's product is able to assess the traffic status shown by a traffic camera via the traffic density, condition, and give a recommended velocity. The users can choose to view cameras along a path or an individual one. [7] Figure 7 shows a traffic camera near *Hai Thuong Lan Ong - Cau Cha Va* with moderate density, unstable flow of traffic condition, and a recommended velocity of 25 km/h.

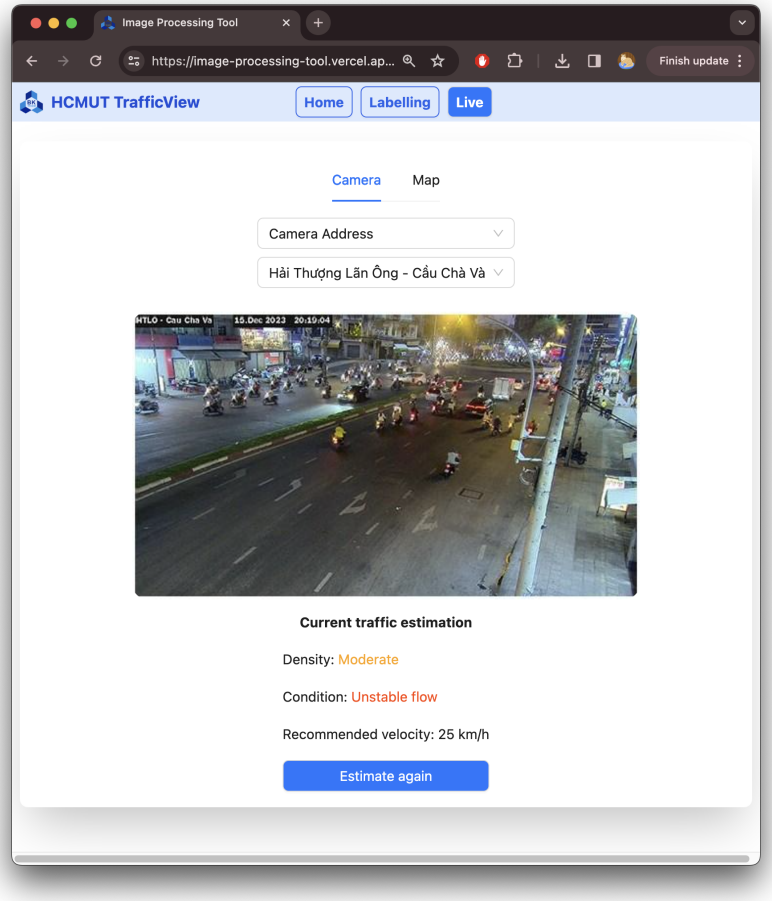


Figure 7: Demonstration of HCMUT TrafficView.

The core functionalities, as described by the authoring group, include:

- Live traffic camera feed: the system allowing users to select a location of interest and view real-time images.
- Congestion predictions: the HCMUT TrafficView predicts the density, level of service, and the average vehicle velocity, which was also demonstrated in Figure 7.
- Route optimization: the system calculates and recommends the shortest path between two locations with respect to the real-time traffic conditions.

The group used pretrained models to arrive to their conclusions, including Mobilenet (v3_large), GoogleNet, and Resnet18, with the reference times being 0.01108 second, 0.014 second, and 0.004 second, respectively.

This is a particular useful application that can have a significant impact on the traffic problem of Ho Chi Minh City. Therefore, the addition of HCMUT TrafficView into the UTraffic system and its mobile apps is an impactful and necessary work. We were tasked with supporting the authoring group in integrating their work into the grand system, and meanwhile, we have also developed a UI to see how the traffic camera feed and the traffic status of it would look like.

2.4 UTraffic's system analysis and design

2.4.1 Apple's MapKit framework

Since UTraffic is all about maps, finding the right mapping framework was our first priority. For iOS, we have the options of using Apple's MapKit or the Google Maps Software Development Kit (SDK). We have decided to go with MapKit for the following reasons:

- **Native support**: MapKit is a native framework of iOS, which means that it is developed and maintained by Apple. This ensures that the framework is always up-to-date with the latest iOS version and that it is optimized for the platform. We would be free from any SDK updates that would introduce breaking changes to the app. What is more, native support means that we have reduced development complexity, as we do not have to deal with third-party SDKs. This frees us to focus on the core functionalities of the app and the integration with the backend more easily.
- **Performance**: MapKit is optimized for iOS, which means that it is faster than other frameworks. This is especially important for UTraffic, as we are dealing with a large amount of data and we need to display them on the map as fast as possible.
- **Pricing**: Because MapKit is included in the iOS development tools, we are free from additional paid subscriptions or API calls. There is no dependence on Google's services or ecosystem, so we would have a greater flexibility.

While it is true that UTraffic web and its Android app uses Google Map SDK as their map rendering framework, we would not have to worry about incompatibility when rendering using MapKit. The fact that both MapKit and Google Maps use the same mapping standard, WGS-84, along with the OpenStreetMap data used on the UTraffic server being WGS-84 compatible, means that we can easily switch between the two frameworks without having to worry about the data being rendered incorrectly [8] [9] [10]. The World Geodetic System 1984 (WGS-84) is the Global Positioning System (GPS)'s reference, and has become the standard used in cartography, geodesy, and satellite navigation [11].

2.4.2 The GeoJSON format

When we started working on the iOS app of UTraffic, it was crucial to visualize the data onto the map as seamlessly as possible. We have previously chosen MapKit as the mapping framework after examining its compatibility with the WGS-84 mapping standard. However, we still need to find a way to convert the data from the server into a format that is compatible with MapKit. This is where GeoJSON comes in.

2.4.2.a GeoJSON definition

GeoJSON is an open standard geospatial data interchange format that represents simple geographic features and their nonspatial attributes. Based on JavaScript Object Notation (JSON), GeoJSON is a format for encoding a variety of geographic data structures. It uses a geographic coordinate reference system, World Geodetic System 1984, and units of decimal degrees [12].

```geojson
{
    "type": "FeatureCollection",
    "features": [
      {
        "type": "Feature",
        "geometry": {
          "type": "Point",
          "coordinates": [
            106.657698,
            10.772603
          ]
        },
        "properties": {
          "name": "Đại học Bách Khoa",
          "type": "Nhà chờ"
        }
      },
      {
        "type": "Feature",
        "geometry": {
          "type": "Point",
          "coordinates": [
            106.657829,
            10.771331
          ]
        },
        "properties": {
          "name": "Đại Học Bách Khoa(cổng trước)",
          "type": "Trụ dừng"
        }
      }
    ]
  }
```


Figure 8: An example of GeoJSON data.

2.4.2.b Feature and geometry types supported by GeoJSON

The following feature types are supported by GeoJSON:

- Point: addresses and locations.
- Line string: streets, highways, and boundaries.
- Polygon: countries, provinces, and tracts of land.
- Multipart collections of points, line strings, or polygons.

The following geometry types are supported by GeoJSON:

- Point.
- LineString.
- Polygon.
- MultiPoint.
- MultiLineString.
- MultiPolygon.

When rendered onto the map, the geometry types are illustrated in Figure 9.

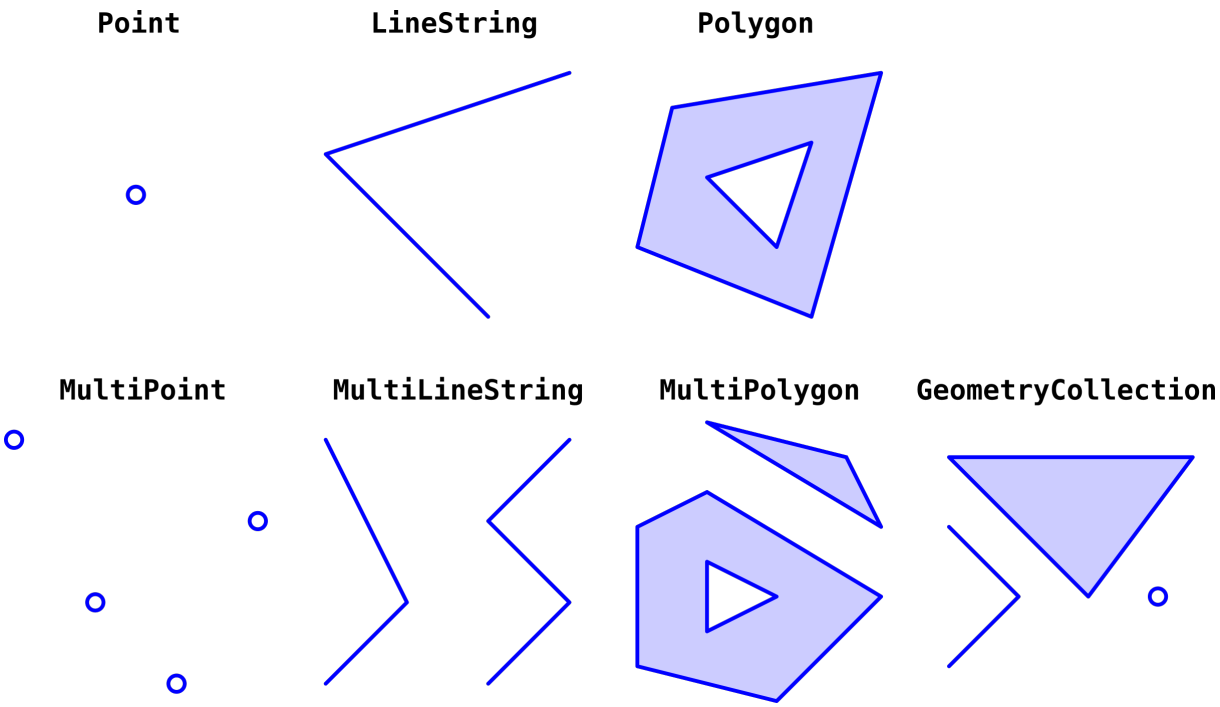


Figure 9: The geometry types supported by GeoJSON.

2.4.2.c The reason for choosing GeoJSON

We came to choose GeoJSON for the following reasons:

- **WGS-84 comaptible**: As the definition claims, GeoJSON is based on the WGS-84 mapping standard, which is the same standard used by MapKit, Google Maps, and OpenStreetMap. This means that we can easily convert the data from the server into a format that is compatible with MapKit.
- **Ease of understanding**: Because GeoJSON displays data in a structured and human-readable format, it is easy to understand and work with. This is especially important when we are working with a large amount of data, as we can easily visualize the data and see if there are any errors.
- **Ease of testing**: Since GeoJSON is a standard format, we can easily test the data using online tools such as geojson.io [13]. This is especially useful when we are testing the data returned from the server, as we can easily visualize the data and see if there are any errors. Figure 10 gives an example in how we can see the bus stops within the 500-meter radius of the Ho Chi Minh City University of Technology.

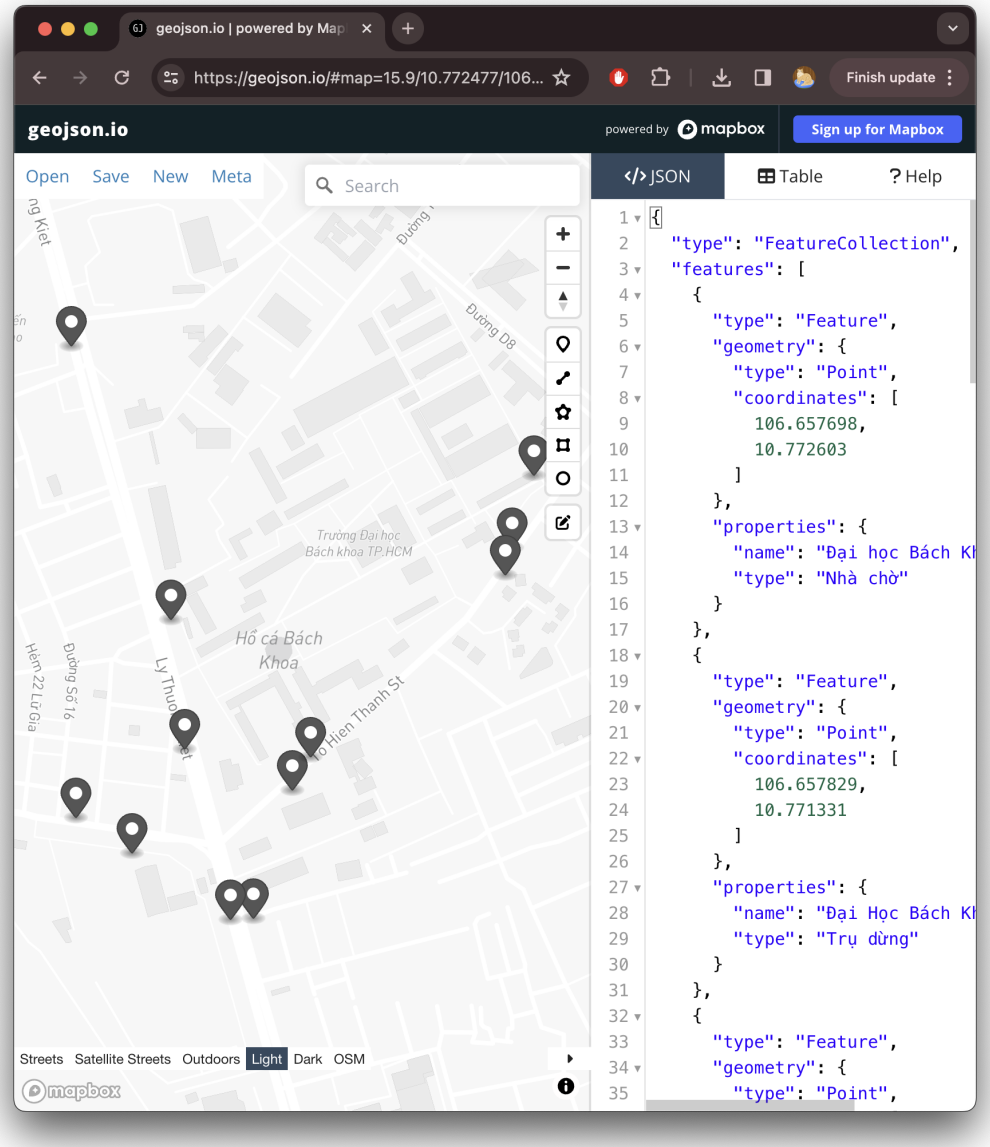


Figure 10: Testing GeoJSON data using geojson.io.

- **Support for iOS MapKit framework**: The iOS MapKit framework supports GeoJSON natively via the use of the `MKGeoJSONDecoder` class. This class is able to decode GeoJSON data into a list of `MKAnnotation` objects, which can then be displayed on the map. This is a great advantage for us, as we do not have to parse the GeoJSON data into our own classes, saving development effort and adhering to Apple's best practices [14]. Since it supports iOS 13.0 and newer versions, our target development version, iOS 16.0, is well within the supported range.

All in all, GeoJSON is part of our effort in standardizing the data format for UTraffic. We believe there are good reasons to choosing this format as we have explained, given our use cases and the advantages that it brings to the system.

2.4.3 Implemented APIs

During our research to improve the performance and coherence of the application, we have developed new API endpoints. They are tailored to work serve two major purposes when it comes to buses, which are the bus routes and the bus stops. The endpoints' description are as follows:

1. API endpoints related to bus stops:
    - **Get nearby bus stops**: This endpoint takes in a latitude value, a longitude value, and a radius value in meters. It then returns a list of bus stops that are within the radius of the given location and in the form of GeoJSON.
    - **Get nearby disability friendly bus stops**: This endpoint takes the same query paramaters like the nearby bus stops endpoint, but it returns a list of disability friendly bus stops instead.
    - **Get the details of a bus stop**: This endpoint takes in a bus stop ID and returns the details of that bus stop.
2. API endpoints related to bus routes:
    - **Get all bus routes**: This endpoint simply returns all the bus routes in the database to the requesting client. This helps in building a list view for browsing purposes.
    - **Get the details of a bus route**: This endpoint takes in a bus route ID and returns the details of that bus route.
    - **Get the bus stops along a bus route's path**: This endpoint takes in a bus route ID and the leg, either **forward** or **return**. The endpoint then returns the bus stops along that bus route's path, preserving the order of the stops. The response is in the form of GeoJSON.
    - **Get the path of a bus route**: This endpoint takes in a bus route ID, and the desired leg and then returns the path of that bus route in the form of GeoJSON.

The details of the API endpoints as well as the deployment efforts that we have done are discussed further in Section 4.

2.4.4 The expansion of the traffic status API endpoint

We also made changes to the currently deployed API endpoints. At the moment, the endpoint `/traffic-status/get-status-v2` receives the query paramaters which are the northeast and southwest bounds of the client's visible map and the zoom level in order to return the appropriate street segments. This endpoint is currently used by the Android app as well as the web version of UTraffic. We have implemented another endpoint, `/traffic-status/get-status-v3`, to handle the same functionality but with GeoJSON as the response. This is done by following the GeoJSON's standard format of being a FeatureCollection of Features, where each Feature is a Point with the coordinates of the traffic status. The following is the comparison between the two format when taking a node as an example:


```
{
  "_id": 62391,
  "polyline": {
    "coordinates": [
      [
        106.6509813,
        10.7442391
      ],
      [
        106.6514738,
        10.7445197
      ]
    ],
    "_id": "657a650dbf0f55d99d183678",
    "type": "LineString"
  },
  "priority": 2,
  "segment": 62391,
  "source": "base-data",
  "velocity": 30,
  "color": "#ffff00",
  "street": {
    "name": "Võ Văn Kiệt",
    "type": "trunk"
  }
}
```

```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [
            106.6509813,
            10.7442391
          ],
          [
            106.6514738,
            10.7445197
          ]
        ]
      },
      "properties": {
        "_id": 62391,
        "priority": 2,
        "segment": 62391,
        "source": "base-data",
        "velocity": 30,
        "color": "#ffff00",
        "street": {
          "name": "Võ Văn Kiệt",
          "type": "trunk"
        }
      }
    }
  ]
}
```


(a) The current response format. (b) The GeoJSON response format.

Figure 11: Comparison between the current response format and the GeoJSON format.

Both of these API endpoints use the map bounds along with the street level in order to limit the processing load of the backend as well as the size of the data returned:

- The map bounds: The northeast and southwest coordinates of the visible region forms a rectangle. Figure 12 illustrates a simplified map region with the coordinates.

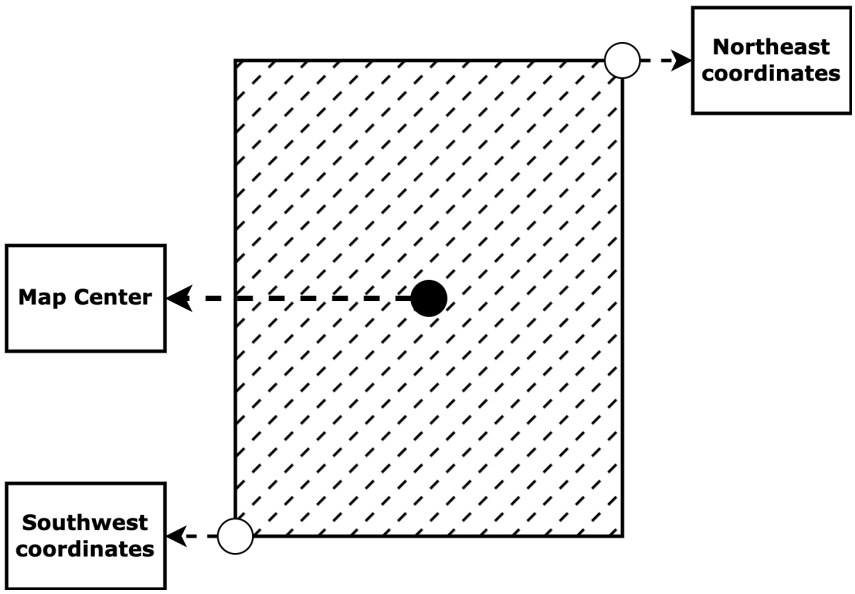


Figure 12: The northeast and southwest bounds of the visible region.

The idea is to find street segments that intersect with it, and then send the appropriate segnments back to the client that issued the API call. Figure 13 illustrates a simplified segment that intersects with the map bounds. In reality, however, it should be understood that a set of segments fall into the region, while the others do not.

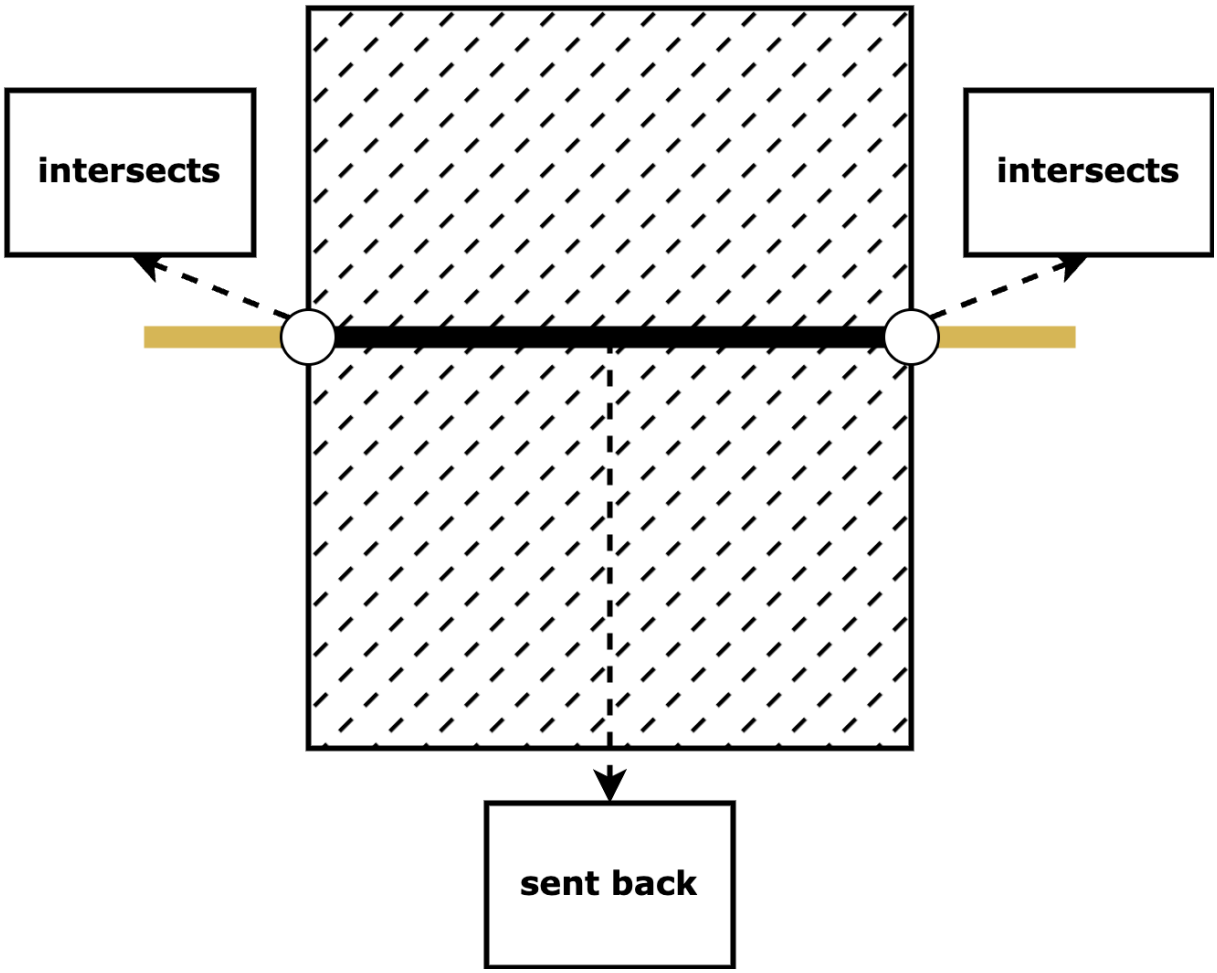


Figure 13: The intersection between the map bounds and the street segments.

- The street level: The street levels correspond to the zoom level of the visible map. When the map is zoomed out, the application will only display traffic reports for major roads. As the map is zoomed in, the application will include traffic reports for smaller roads as well. We will adopt the definition of street levels based on the research of the group of Duong Hoai Phong and Nguyen Mau Quoc Duong [15], which is as follows:
  - Level 1 - Zoom level 0 to 12: The application will only display traffic reports for trunk roads.
  - Level 2 - Zoom level 13 - 14: The application will onl display trunk and primary roads.
  - Level 3 - Zoom level 15 and higher: The application will display all types of roads.

With the bounds and street levels ready, we were able to display the traffic status on the iOS map, which is discussed about in greater detail in Section 4.

3 THEORETICAL BASIS

3.1 OpenStreetMap's data format

The OSM data file is structured around three fundamental elements:

- **Node**: Represents a singular point on the geographic map, defined by latitude, longitude, and a unique identifier (ID). Optional tag elements, children of a node, display metadata. Nodes can signify individual features (e.g., a specific bench) or combine to create paths or areas.
- **Way**: Comprises nodes forming a specific zone or line. Nodes within a way tag serve as vertices of a shape or path, referenced by the "ref" attribute. Ways can be open (with the initial and terminal nodes different) or closed (forming a closed shape). Open ways often represent linear features like roads or streams, while closed ways define boundaries, barriers, or territorial areas.
- **Relation**: An element with at least the "type=*" tag and a list of members (nodes, ways, or relations). Relations define logical or geographic connections between objects, such as a lake and its island or roads for a bus route. Members in a relation can have optional roles, describing their part within the relation. Members must be geometric elements or other relations, ensuring valid visibility on the map.

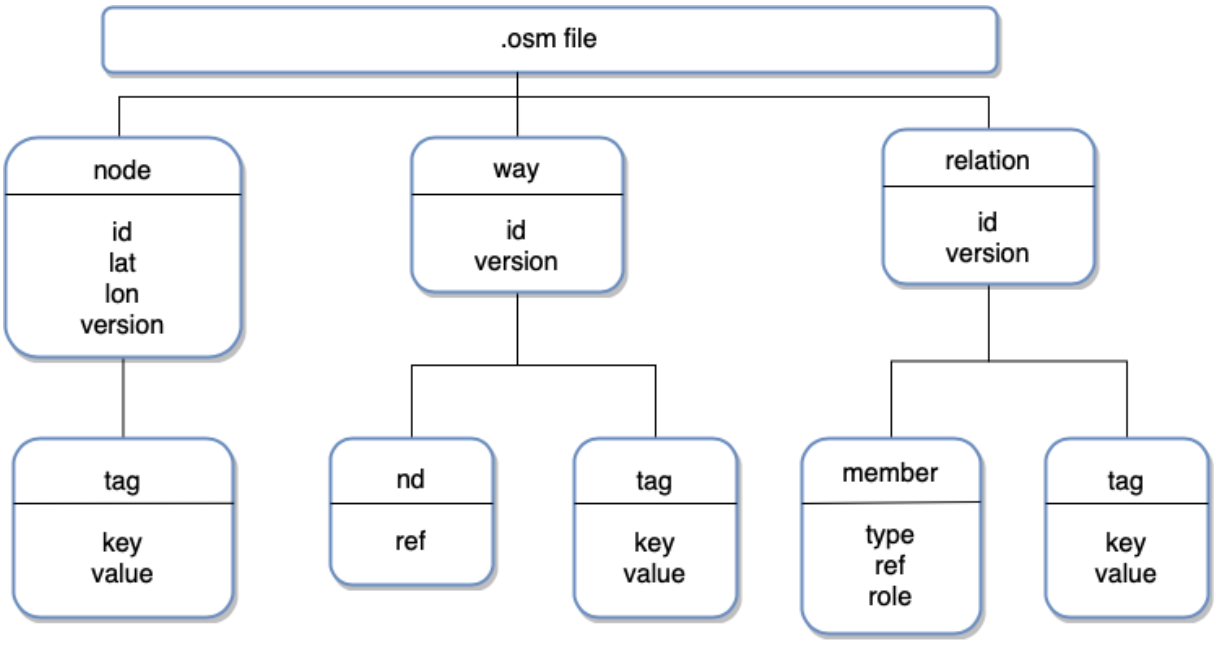


Figure 14: OSM data format [16]

3.2 iOS Application design patterns

3.2.1 Model-View-Controller (MVC)

MVC is an architectural design pattern for building applications that separates the application into three main logical components: the model, the view, and the controller. In the MVC model:

- **Model**: Represents the data and business logic of the application. It is the "unchanging essence" of the application and should be stable and long-lived.

- **View**: Represents the user interface of the application. There can be multiple views for different devices, such as a web interface, mobile app, or command line interface.
- **Controller**: Handles user input and updates the model and view accordingly. It acts as an intermediary between the model and the view, processing business logic and managing data flow.

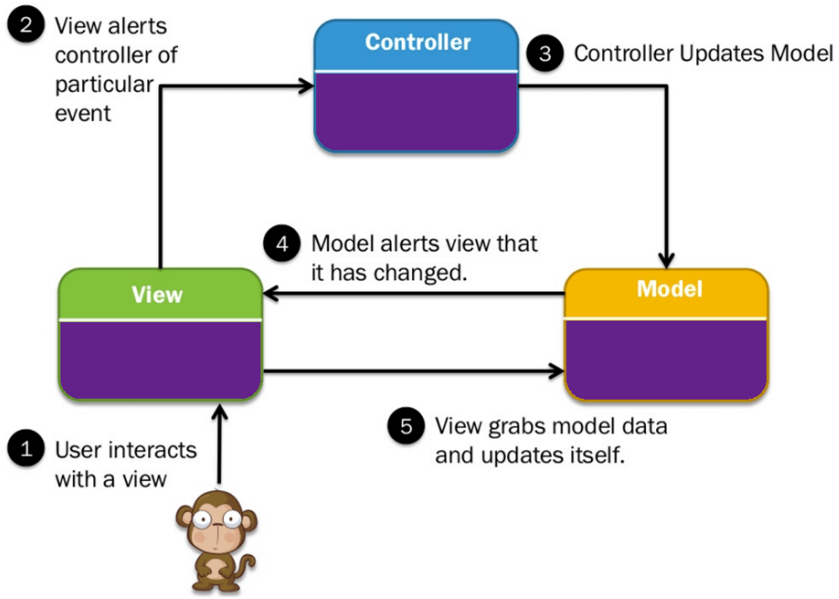


Figure 15: MVC architecture

3.2.2 Model-View-ViewModel (MVVM)

MVVM architecture offers two-way data binding between view and view-model. The view-model makes use of observer pattern to make changes in the view-model. The components of MVVM are:

- **Model**: Holds the data and business logic. The model is independant of UI or user interaction.
- **View**: Represents how the user interacts with the application.
- **View-model**: Acts as a bridge between the view and the model. It provides data from the model to the view and handles user interaction with the view.

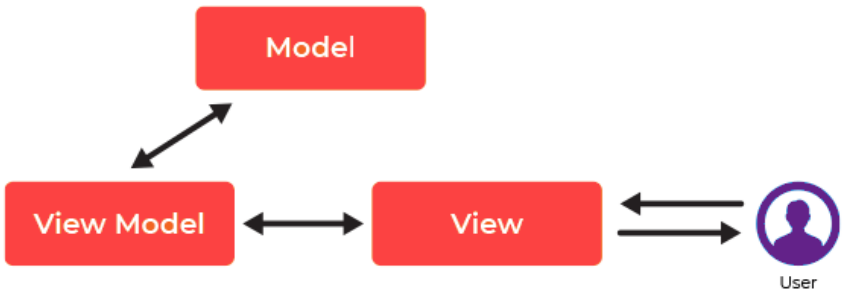


Figure 16: MVVM architecture

3.2.3 Conclusion

	Advantage	Disadvantage
MVC	• The development of the various components can be performed parallelly. • It avoids complexity by dividing an application into separate (MVC) units • It only uses a front controller pattern that processes web application requests using a single controller. • It provides a clean separation of concerns.	• Business logic is mixed with Ul • Hard to reuse and implement tests • There is a need for multiple programmers to conduct parallel programming.
MVVC	• Business logic is decoupled from Ul. • Easy to reuse components. • You can write unit test cases for both the viewmodel and Model layer without the need to reference the View.	• Maintenance of lots of codes in controller • Some people think that for simple UIs of MVVM architecture can be overkill

Table 1: Comparison between MVC and MVVM

Base on the comparison above, MVVM is more suitable for this project because it is easier to reuse components and write unit test cases.

3.3 Front-end

3.3.1 Objective-C

Objective-C currently ranks 22nd in popularity according to the TIOBE Index. While its use is declining, it remains significant in the Apple ecosystem, with around 60,000 active repositories. Introduced in the 1980s by Brad Cox and Tom Love, Objective-C is an object-oriented programming language primarily used for Apple platforms. It was licensed by NeXT Computer Inc., a company founded by Steve Jobs. Noteworthy features include classes, inheritance, dynamic binding, and a Smalltalk-like syntax for messaging objects. Despite declining usage, Objective-C remains one of the more popular programming languages in the market

Some of the great features of this language are:

- **Object-Oriented**: Supports classes, objects, and object-oriented concepts.
- **Dynamic Binding**: Resolves method calls at runtime for flexibility.
- **Message Passing**: Utilizes message passing for a natural coding style.
- **Smalltalk-style Syntax**: Simple and expressive syntax like Smalltalk for clear code.
- **Cross-Platform**: Works on macOS, iOS, and other Apple platforms, plus Windows and Linux.
- **Interoperability with C**: Integrates seamlessly with C libraries for collaboration.

3.3.2 Swift

Swift is a powerful and modern programming language developed by Apple for building applications across its ecosystem, including iOS, macOS, watchOS, and tvOS. Introduced in 2014, Swift was designed to be efficient, expressive, and easy to learn, addressing the shortcomings of its predecessor, Objective-C. With a syntax that is both concise and readable, Swift empowers developers to create robust and high-performance applications. Known for its safety features, dynamic capabilities, and versatility, Swift has quickly become the language of choice for many developers seeking to craft innovative and seamless experiences within the Apple ecosystem. In this introduction, we'll explore the key features and characteristics that make Swift a standout language in the world of software development.

3.3.3 SwiftUI

In summary, SwiftUI, developed by Apple, has become increasingly popular among iOS app developers for valid reasons. Introduced five years after Swift alongside Swift 5 and Xcode 11, it serves as a UI toolkit designed for creating software across various platforms, including iOS, macOS, watchOS, and tvOS. SwiftUI offers a declarative approach to UI design, enabling developers to describe layout and behavior using a simple and intuitive syntax. This results in more efficient and faster development of complex UIs, thanks to concise and easily readable code compared to traditional imperative approaches.

In summary, SwiftUI stands out with these key features:

- **Declarative Syntax**: Describes UI appearance easily, improving code readability.
- **Automatic Layout**: Handles UI layout automatically, reducing manual effort.
- **Dynamic UI**: Enables the creation of dynamic and interactive interfaces.
- **Cross-platform Support**: Works across all Apple devices with a single codebase.
- **Real-time Preview**: Provides live previews of UI changes during development.
- **Pre-built Components**: Offers ready-to-use UI components for quick and efficient development.

3.3.4 Conclusion

Feature	Objective C	Swift
Age	Developed in the early 1980s	Introduced in 2014
Syntax	Uses C-based syntax with Smalltalk-style messaging	Uses a modern and concise syntax
Performance	Slower than Swift due to overhead and lack of optimization	Faster than Objective-C due to optimization features
Memory Managment	Prone to memory leaks	No memory leaks due to being type-safe and memory-safe
Stability	Stable	Unstable as it is still growing

Table 2: Comparison between Objective C and Swift

Base on the comparison above, Swift is more suitable for this project because it is superior in multiple aspects when comparing to Objective C.

3.4 Back-end

3.4.1 Java

Java stands out as a widely utilized and potent programming language, particularly in the realm of back-end development. Launched in 1995, it has evolved into one of the most embraced languages for enterprise-level software development. The robustness, portability, and scalability of Java contribute to its prevalence in the back-end space. Leveraging an object-oriented programming model facilitates the creation of modular and reusable code, and its platform independence enables code execution across various operating systems. Java's appeal is further enhanced by its extensive standard library and a broad ecosystem of third-party libraries, facilitating the rapid and efficient development of intricate, feature-rich applications. Particularly suited for constructing large-scale applications demanding high performance and reliability, Java's garbage collection system ensures efficient memory management, and its support for multithreading enables the execution of parallel tasks with efficiency.

3.4.2 Python

Python, a favored programming language, is commonly employed in backend systems. Noted for its straightforward syntax, it is easy to learn and wield, featuring a broad spectrum of libraries and frameworks, rendering it versatile for diverse applications. Python excels in backend systems handling data processing and analysis, leveraging popular libraries such as NumPy, Pandas, and Matplotlib. These facilitate seamless handling of large datasets, intricate computations, and data visualization.

Moreover, Python is widely chosen for web applications, leveraging frameworks like Django and Flask. Its compatibility with ReactJS is notable, supporting commonly used web development protocols, including RESTful APIs, simplifying the creation of APIs consumable by ReactJS applications. Backed by a vast developer community, Python ensures accessibility to support and solutions for common challenges.

Python's versatility extends to embedding image processing and natural language processing technologies, exemplified by YOLO, RASA, and OpenAI. This is made possible through a suite of libraries and frameworks, such as OpenCV for computer vision, TensorFlow and PyTorch for machine learning, NLTK for natural language processing, and RASA for conversational AI. Additionally, Python can access the robust OpenAI API through HTTP requests, providing advanced natural language processing capabilities. In essence, Python's rich ecosystem makes it a robust choice for constructing backend systems that incorporate image processing and natural language processing technologies.

3.4.3 NodeJS

Node.js, originating in 2009, stands as a preeminent back-end development framework, leveraging the JavaScript programming language. Revered for its speed, efficiency, and scalability, Node.js is widely adopted for constructing scalable back-end applications, particularly in real-time domains like chat applications, social networks, and online gaming platforms.

Operating on an event-driven, non-blocking I/O model, Node.js facilitates high levels of concurrency and throughput. This design ensures applications can adeptly manage substantial traffic and data influx. The robust Node.js community has fostered the development of numerous valuable libraries and frameworks, streamlining the creation of intricate applications swiftly and effectively. Notable Node.js frameworks include Express.js, Koa, and Nest.js.

3.4.4 Conclusion

	Advantage	Disadvantage
Java	• Java's strict type-checking and error checking enhance stability for complex apps. • Efficiently handles large-scale applications with substantial data and high traffic. • Supports cross-platform development with its "write once, run anywhere" philosophy. • Java's extensive third-party libraries and frameworks enhance adaptability.	• Java's strict syntax and type system may pose challenges for beginners, impacting their effective use of the language. • Java applications can exhibit slower startup times compared to other languages, potentially affecting the performance of smaller applications. • Java's garbage collection system, while efficient, may lead to performance issues in specific scenarios.
Python	• Python's syntax is designed for easy comprehension, appealing to both beginners and seasoned developers. • Python serves a broad range of applications, from web development to data science and machine learning. • Python features a vast collection of third-party libraries and frameworks, extending its capabilities.	• Due to its interpreted nature and dynamic typing, Python may exhibit slower performance in comparison to some languages. • Python may face limitations in handling large datasets and high traffic, impacting suitability for large-scale applications. • Python's Global Interpreter Lock (GIL) can limit its ability to fully exploit multi-core processors.
NodeJS	• NodeJS prioritizes speed, making it well-suited for real-time applications. • NodeJS exhibits high scalability, handling substantial traffic and data volumes with ease. • A vibrant developer community has contributed to a plethora of libraries and frameworks, streamlining complex application development.	• While excelling in real-time scenarios, NodeJS might not be the optimal choice for applications requiring intricate data processing or machine learning. • NodeJS utilizes an asynchronous programming model, which could pose challenges for beginners. • Debugging NodeJS applications can be intricate due to its asynchronous model and event-driven architecture.

Table 3: Comparison between Java, Python, and NodeJS

When weighing the pros and cons, each language possesses unique strengths and weaknesses tailored to specific project requirements. In scenarios demanding exceptional performance, scalability, and reliability, Java emerges as a compelling choice. Furthermore, Java's platform independence ensures compatibility with various operating systems and hardware configurations, offering versatility for diverse project types.

3.5 Database

3.5.1 MySQL

MySQL is a widely used database for web-based applications, offered as freeware with regular upgrades to improve capabilities and security. The freeware edition emphasizes efficiency and dependability over a comprehensive range of functionalities. Prominent characteristics encompass the ability to select storage engines, a user-friendly interface, and effective handling of extensive datasets in batches, all while maintaining modest resource use.

	Advantage	Disadvantage
,MySQL	• MySQL is accessible for free, making it cost-effective for various applications. • Multiple user interfaces can be implemented, enhancing usability. • Supports both structured data (SQL) and semi-structured data (JSON).	• The process of setting up MySQL for specific activities may necessitate a greater investment of time and effort in comparison to systems that automate certain functions. • While support is available for the free version, premium support may incur additional costs.

Table 4: Advantages and disadvantages of MySQL

3.5.2 MongoDB

MongoDB is designed for applications that handle both structured and unstructured data, and it is offered in both free and paid versions. The highly adaptable database engine establishes connections between databases and applications using MongoDB database drivers, providing a wide array of options that are compatible with multiple programming languages. Although MongoDB may encounter performance challenges when heavily utilized for relational data models, it demonstrates exceptional capabilities in managing changeable, non-relational data. Successive iterations consistently incorporate functionalities to cater to various applications, bolster operational robustness, and prioritize the protection of data.

	Advantage	Disadvantage
MongoDB	• MongoDB is fast and easy to use. • The engine supports JSON and other NoSQL document formats. • Capable of effectively storing and retrieving data of any arrangement without rigid schema prerequisites. • Allows the modification of schema without downtime, facilitating flexibility.	• Tools translating SQL to MongoDB queries add an extra step in using the engine. • The default settings may lack sufficient security measures, necessitating further tweaking to provide optimal security.

Table 5: Advantages and disadvantages of MongoDB

3.5.3 Oracle

Oracle is a prominent and long-lasting database management solution that has been in use since the late 1970s. The different versions of the software are designed to meet the specific requirements of different organizations. The current long-term release focuses on providing comprehensive support and ensuring reliability. The most recent update incorporates cutting-edge functionalities like as autonomic management, AutoML, and improved multi-model compatibility, hence strengthening its attractiveness for future use.

	Advantage	Disadvantage
Oracle	• Oracle consistently deliver cutting-edge innovations. • Oracle's tools are highly robust, offering a wide range of capabilities to meet diverse requirements. • Supports various data models, including semistructured (JSON, XML), spatial, RDF, and structured data (SQL). • Provides multiple access patterns based on the data model.	• The cost of Oracle can be prohibitive, particularly for smaller organizations. • The installation process may need a substantial amount of resources, which could result in the need for hardware upgrades.

Table 6: Advantages and disadvantages of Oracle

3.5.4 PostgreSQL

To summarize, PostgreSQL is a popular and freely available database system that is extensively utilized for web-based databases. Being one of the pioneering database management systems, it supports both organized and unstructured data and is compatible with multiple platforms, including Linux. The most recent update incorporates improvements in compression choices, backing for organized server log output in JSON format, and general benefits in performance.

	Advantage	Disadvantage
PostgreSQL	• PostgreSQL has excellent scalability, enabling efficient management of terabytes of data. • The system is capable of handling JSON data format, which allows for versatile data administration. • Offers a range of predefined functions for diverse database operations. • Multiple interfaces are available for user interaction. • Functions as a multi-model database, supporting Spatial Data, Key-Value, Structured Data (SQL), and Semi-Structured Data (JSON, XML).	• The documentation may be incomplete or challenging to navigate • Speed may be affected during extensive bulk operations or read queries involving large datasets.

Table 7: Advantages and disadvantages of PostgreSQL

3.5.5 Conclusion

After analyzing the advantages and disadvantages mentioned, it is clear that each of the four databases has unique strengths and limitations that are suited to certain project needs. When it comes to situations where the most important factors are scalability, flexibility, and excellent performance, MongoDB is the ideal choice. MongoDB is widely recognized for its remarkable capacity to handle large amounts of data and heavy website traffic. MongoDB's document-oriented architecture is specifically tailored for unstructured data and complex data structures. This design provides efficient access and retrieval of data, resulting in improved overall performance.

4 IMPLEMENTATION

As developed by the previous groups, the UTraffic architecture consists of four main components: an API server, a computing server, a database server, and an application interface.

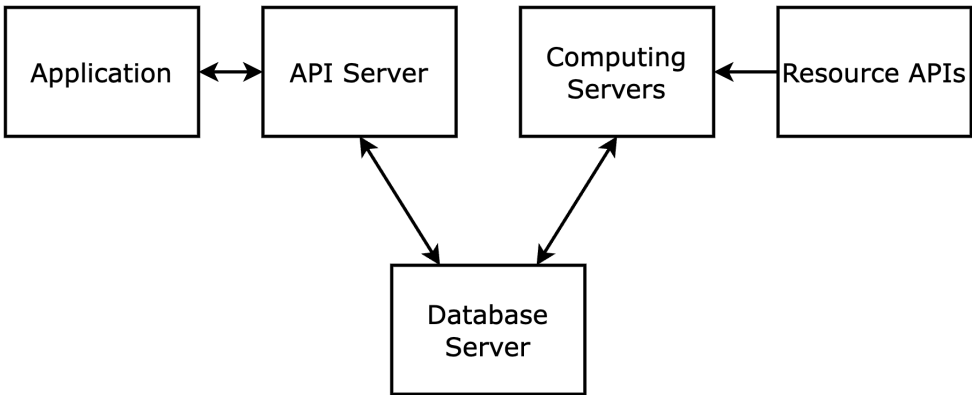


Figure 17: The system's architecture diagram.

- Database server: All the data of UTraffic is stored in a MongoDB database. The database server is responsible for storing and retrieving data. There is a lot of queries that concern geographical coordinates, and MongoDB well supports this type of data via the use of geospatial indexes.
- API server: The API server is a pivotal point of the UTraffic architecture, acting as a bridge to connect the application interface and the database server via API endpoints.
- Computing server: The computing server processes user data. This work consists of traffic status calculation, the user's credit, as well as the user's total traveled distance. We have not made any changes to this component in this phase of the project.
- Application: The application is the interface that the users interact with. Previous groups have developed the web application and the Android application. In this phase, we have developed a prototype iOS application.

Along with the main components, the Resource APIs component acts as a data mining source in the case of insufficient data.

4.1 Bus Services Integration

4.1.1 Getting a single route's information

The integrity of the data on the Buyt TPHCM website is guaranteed due to its commissioning and maintenance by the Ho Chi Minh City Department of Transport.

Several JavaScript source files power this site:

- **L.Config.js:** This is a configuration file to set up the map rendering for Buyt TPHCM. The tiles are loaded from http://map.stis.vn/bright/z/x/y.png. This means that FPT has forked a copy of the OpenStreetMap project and is hosting it on their own server.
- **L.RouteMap.js:** This source file is responsible for showing the stops and bus routes. We were able to get the coordinate sets that we wanted by placing breakpoints on this file.

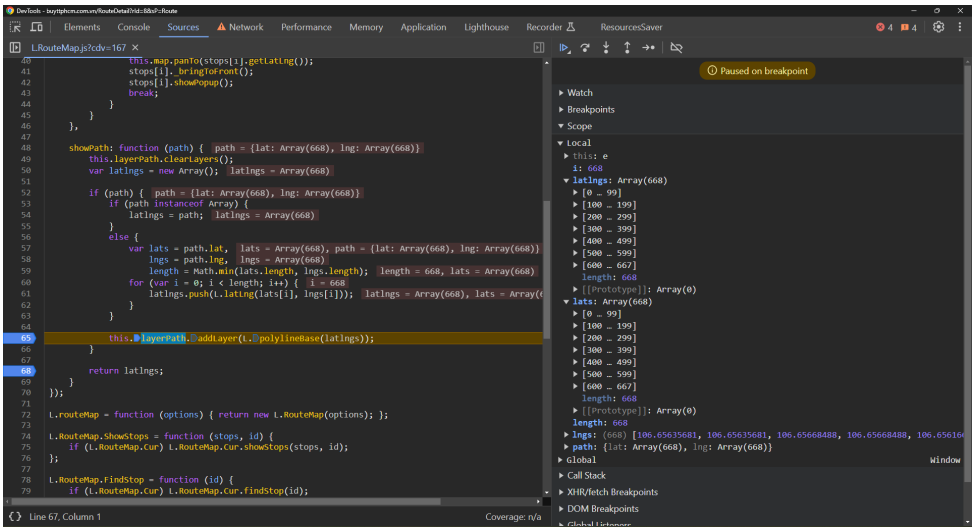


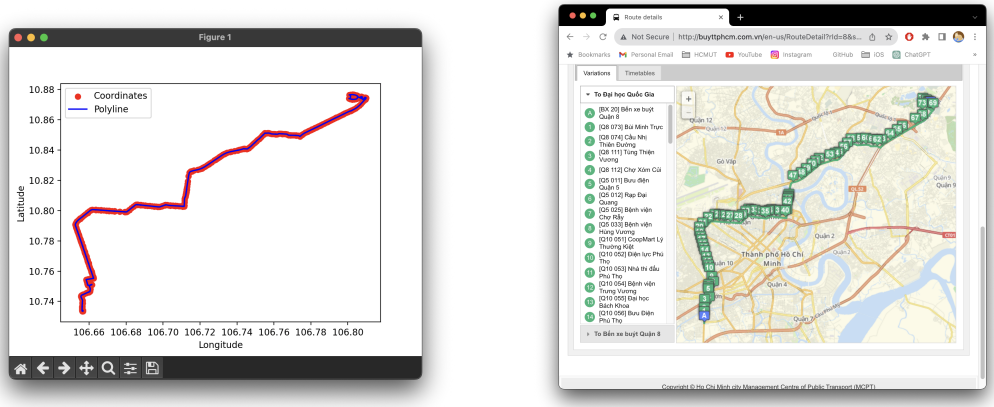
Figure 18: Capturing and observing the route data using the DevTool

By inserting breakpoints in L.RouteMap.js, we can easily obtain the coordinates of the route, as demonstrated on the right-hand side. It is worth mentioning that while the coordinates are given in both distinct arrays and a single array with integrated data, it is more advantageous to utilize the latitudes and longitudes in separate arrays. The reason for this is that the present system implementation utilizes MongoDB, which does not allow tuples due to its implementation of BSON.

When it is desirable to work with a single array of coordinates, the stored data can be retrieved and combined together. Below is an example where we zip the arrays and plot them using Matplotlib (please note that since there are over 600 elements in each array, 668 to be exact, a figurative number was put in the code excerpt).

```
lat = [10.001, 10.002] # the latitudes
lng = [106.001, 106.002] # the longitudes
coordinates = [(lat[i], lng[i]) for i in range(min(len(lat), len(lng)))]
# Extract latitudes and longitudes from the coordinates
latitudes, longitudes = zip(*coordinates)

# Create a scatter plot to mark the coordinates
plt.scatter(longitudes, latitudes, color='red', marker='o', label='
Coordinates')

# Create a line plot to draw the polyline
plt.plot(longitudes, latitudes, color='blue', label='Polyline')

# Set labels for the x and y axes
plt.xlabel('Longitude')
plt.ylabel('Latitude')

# Add a legend
plt.legend()

# Display the plot
plt.show()
```

The result is as follow:



(a) The Matplotlib representation of the bus route 8.

(b) The actual route 8 from Buyt TPHCM.

Similarly, information about the bus stops can be obtained by placing breakpoints in L.RouteMap.js. Each route comes with a list of stops and their corresponding data. This will be useful in designing our data schema later on when we integrate the gathered information into our database.

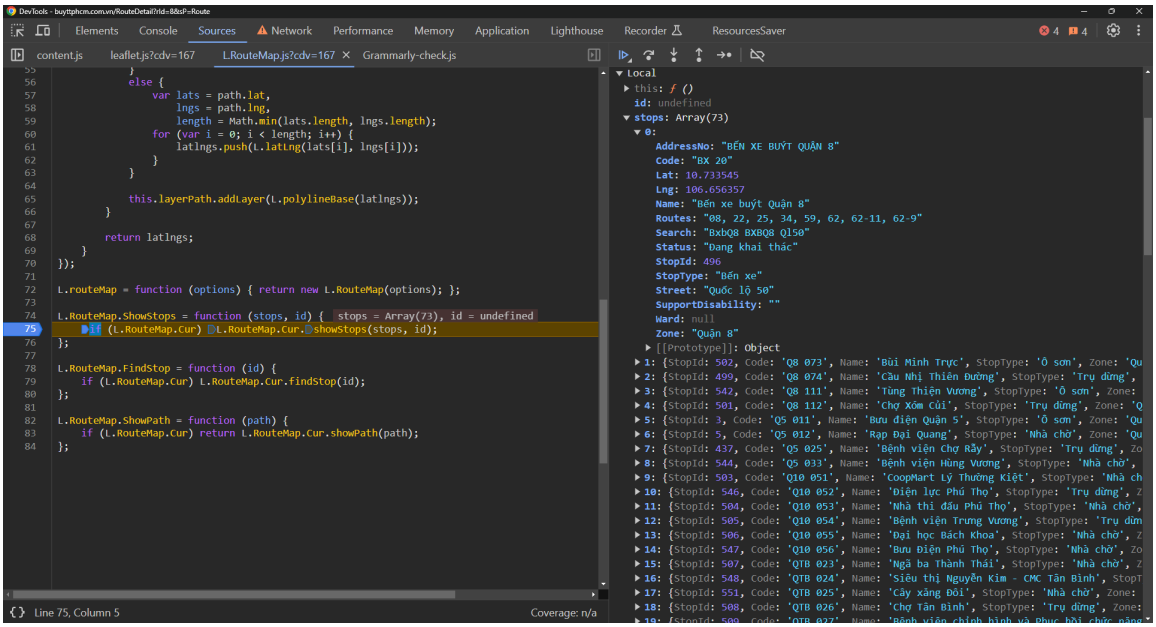


Figure 20: Capturing and observing the bus stops data using the DevTool

4.1.2 Automating the data collection process

Since there are about 130 routes currently displayed on the website, it is impossible to place breakpoints and collect data by hand. This necessitates the use of automatic scripts to gather data.

After careful examinations, we found several places where we could automate getting the data. This introduces us to the routevar.js file, which contains the three functions: `loadVarsByRou(rouId)`, `loadStopsByVar(prtId, rouId, varId)`, and `loadPathByVar(prtId, rouId, varId)`.

It is worth it for us to elaborate more on the given terms:

- `rouId`: This value represents a bus route's ID. It is important to note that this ID is not the same as the route number. For example, the route SWB1 is represented by the ID 337.
- `prtId`: This value represents a bus route's leg, which is either forward or return. The source from which we got the data does not use set values for all the routes, with some routes using 1 and 2 denoting each leg, while some other routes use 3, 4 or even 7, 8.
- `varId`: This value represents a bus route's variant. A variant's value is the combination of the `rouId` and `prtId`. Therefore, if the `rouId` is 8 and the `prtId` is 1, then the `varId` is *r8v1*.

The automated data retrieval can be described in the following steps:

1. Go to the source website and wait for all routes to load.
2. Capture all links to individual routes and store them in a temporary list.
3. Extract the necessary ID in each link and loop over the list to extract individual data.

Because the `rouId` is present in a bus route view's URL, we can extract this value and log the value returned by the `loadVarsByRou` function. After that, we can split the recorded string to get the `prtId` value. The data retrieval process then involves injecting the website with two custom JavaScript variables that can capture the data loaded by the `loadStopsByVar(prtId, rouId, varId)` and `loadPathByVar(prtId, rouId, varId)`. This is done after we have injected the website with our custom `routevar.js` script that assigns the source's data to the custom variables. This whole process is then automated using Selenium WebDriver. Figure 21 shows how the raw JSON data are stored after they are collected. This structured organization makes it easy for later scripts to iterate over the data and further process them.

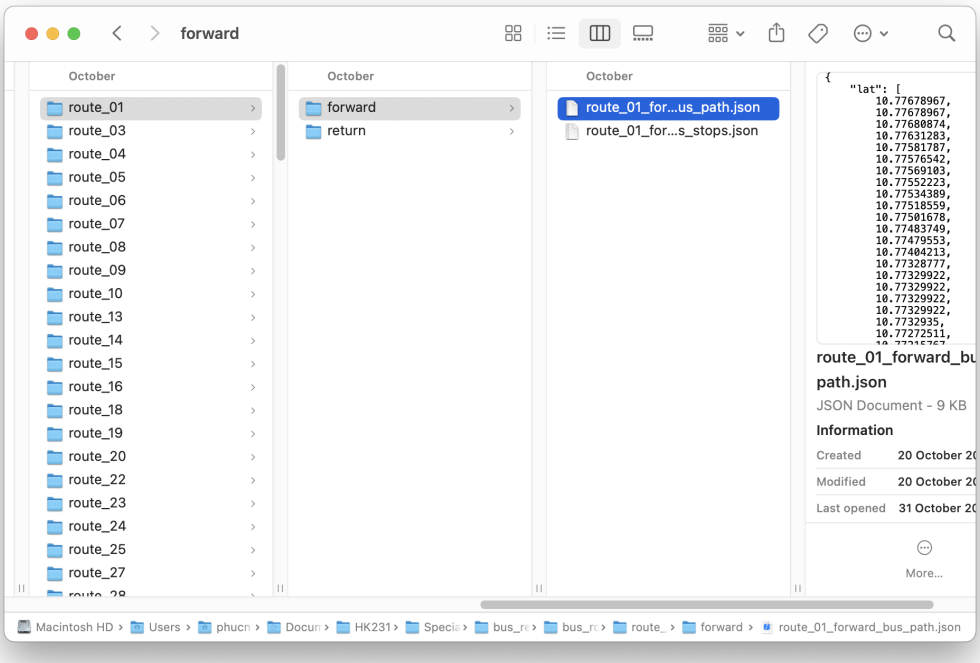


Figure 21: Organization of the data retrieval process.

```
[
    {
        "AddressNo": "BẾN XE BUÝT QUẬN 8",
        "Code": "BX 20",
        "Lat": 10.733545,
        "Lng": 106.656357,
        "Name": "Bến xe buýt Quận 8",
        "Routes": "08, 22, 25, 34, 59, 62, 62-11, 62-9",
        "Search": "BxbQ8 BXBQ8 Ql50",
        "Status": "Đang khai thác",
        "StopId": 496,
        "StopType": "Bến xe",
        "Street": "Quốc lộ 50",
        "SupportDisability": "",
        "Ward": null,
        "Zone": "Quận 8"
    },
    {
        "AddressNo": "193",
        "Code": "Q8 073",
        "Lat": 10.73646,
        "Lng": 106.656346,
        "Name": "Bùi Minh Trực",
        "Routes": "08, 25, 45, 47, 59, 62",
        "Search": "BMT 193 Ql50",
        "Status": "Đang khai thác",
        "StopId": 502,
        "StopType": "Ô sơn",
        "Street": "Quốc lộ 50",
        "SupportDisability": "",
        "Ward": null,
        "Zone": "Quận 8"
    },
    // ...
]
```


Figure 22: The bus stops of a route before processing.

For the details of a bus route, this was made easy since they are explicitly displayed on the source website. We were able to parse the existing HTML structures and put the data into a JSON file and automate the process using Selenium WebDriver.

4.1.3 Data storage of bus services

Given the structured nature of the data we collected, as well as the existing use of MongoDB, it was a straightforward process to load the collection JSON files into MongoDB. This process was made even simpler with MongoDB Compass, a GUI for MongoDB that allows us to import JSON and CSV files into the database with a few clicks.

However, the first thing to do was to preprocess the data for efficient and logical storage. In particular, for a route number, the raw data has the stops listed in full details, as in Figure 22. If we had gone and stored the data as-is to MongoDB, one clear problem would have been duplicated data. Therefore, our solution to this was to create a Python script to iterate over the raw JSON and pick out the unique stops and save them to a CSV file. For the association with the routes, we reduced the stops array to only contain the IDs. This also helped save the order of the stops along a certain path. Finally, when it comes to querying, the IDs can be used to retrieve the full details of the stops from the stops collection, improving performance.

We also implemented a `2dsphere` index on the locations of the bus stops. This helps tremendously in the querying processes that care about the radius from within a coordinate. Our *Get nearby bus stops* API endpoint is a prime example of how useful this index is, because if it was not implemented, the scanning process would have been very slow and inefficient. A column scan would have been used, and given that there are over 4000 stops in the database, the performance would have been unacceptable. The index creation was straightforward with the help of MongoDB Compass.

We decided to have 4 new collections within the existing MongoDB database. They are as follows:

- **Bus_Associations**: This collection associates bus routes with their stops. The stops are stored as an array of IDs, preserving the order of the stops along a certain route.
- **Bus_Metadata**: This collection contains all information about bus routes, including their names, operating hours, operating agencies, the daily total trips, and the trip spacings.
- **Bus_Path_Coordinates**: This collection stores the paths of the bus routes in 4 arrays of forward and return latitudes and longitudes.
- **Bus_Stops**: This collection stores all the bus stops in the database. The details of a stop are its official street address, its location, its type, whether it is disability-friendly, and the routes that pass through it.

Figure 23 showcases the Bus_Stops collection in MongoDB Compass with 4414 documents.

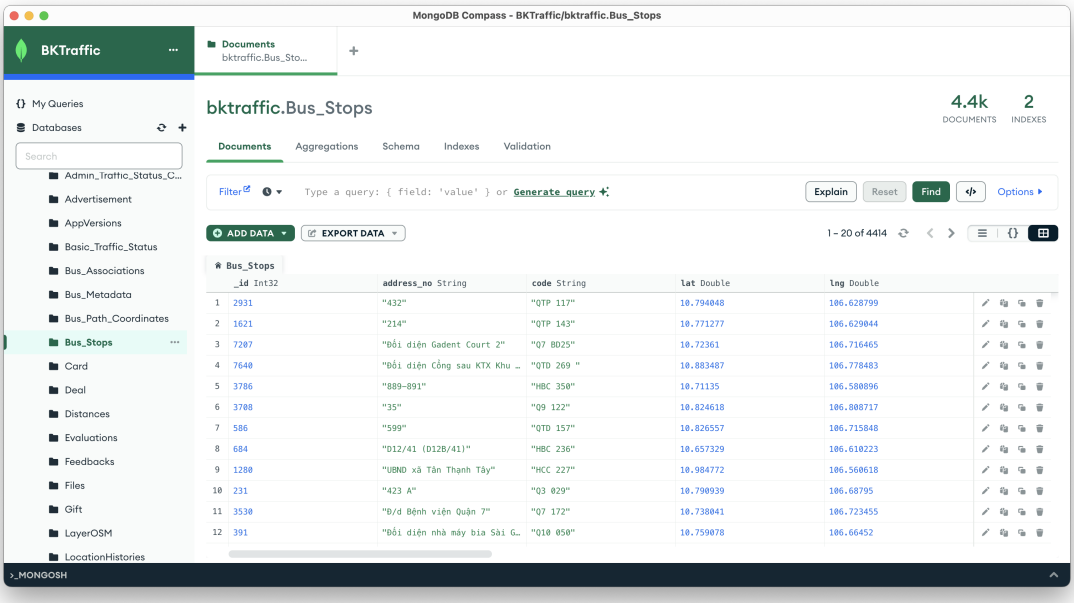


Figure 23: The Bus_Stops collection in MongoDB Compass.

4.2 Bus API Development and Server Deployment

We have previously discussed the introduction of the API additions in the UTraffic system, and we have taken the steps to deploy them on the live server of UTraffic. The endpoints are now live and working as intended.

4.2.1 The Bus API

The following are the details of the new API endpoints. An example URL and the corresponding response are also provided for each endpoint.

- **Get nearby bus stops**.
  - URL: https://api.bktraffic.com/api/bus/nearby-stops.
  - Method: GET.
  - Query parameters: `lat`, `lng`, `radius`.
  - Example request URL: https://api.bktraffic.com/api/bus/nearby-stops?lat=10.7721&lng=106.6579&radius=500
  - Example response:


```
{
    "code": 200,
    "message": "success",
    "data": {
        "type": "FeatureCollection",
        "features": [
            {
                "type": "Feature",
                "geometry": {
                    "type": "Point",
                    "coordinates": [
                        106.657698,
                        10.772603
                    ]
                },
                "properties": {
                    "name": "Đại học Bách Khoa",
                    "type": "Nhà chờ"
                }
            },
            {
                "type": "Feature",
                "geometry": {
                    "type": "Point",
                    "coordinates": [
                        106.657829,
                        10.771331
                    ]
                },
                "properties": {
                    "name": "Đại Học Bách Khoa(cổng trước)",
                    "type": "Trụ dừng"
                }
            },
            // ...
        ]
    }
}
```


Figure 24: The response containing bus stops within 500m of the coordinate (10.7721, 106.6579).

- **Get nearby disability friendly bus stops.**
  - URL: https://api.bktraffic.com/api/bus/disability-friendly-stops.
  - Method: GET.
  - Query parameters: `lat`, `lng`.
  - Example request URL: http://api.bktraffic.com/api/bus/disability-friendly-stops?lat=10.77037&lng=106.69868
  - Example response:


```
{
    "code": 200,
    "message": "success",
    "data": [
        {
            "_id": 1344,
            "address_no": "65G (75)",
            "code": "Q1 119",
            "lat": 10.766785,
            "lng": 106.696011,
            "name": "Cầu Ông Lãnh",
            "routes": "139, 140, 31, 46, 72",
            "status": "Đang khai thác",
            "stop_type": "Nhà chờ",
            "street": "Nguyễn Thái Học",
            "support_disability": true,
            "ward": "Phường Cầu Ông Lãnh",
            "zone": "Quận 1",
            "location": {
                "type": "Point",
                "coordinates": [
                    106.696011,
                    10.766785
                ]
            }
        },
        {
            "_id": 7276,
            "address_no": "277-279-275Y",
            "code": "Q1 190",
            "lat": 10.76767,
            "lng": 106.690941,
            "name": "Tôn Thất Tùng",
            "routes": "03, 04, 102, 109, 140, 18, 19, 20,
28, 34, 36, 39, 46, 52, 65, 69, 72, 75, 88, 93, D4",
            "status": "Đang khai thác",
            "stop_type": "Trụ dừng",
            "street": "Phạm Ngũ Lão",
            "support_disability": true,
            "ward": "Phường Phạm Ngũ Lão",
            "zone": "Quận 1",
            "location": {
                "type": "Point",
                "coordinates": [
                    106.690941,
                    10.76767
                ]
            }
        },
        // ...
    ]
}
```


Figure 25: The response containing bus stops that are disability-friendly near the coordinate (10.77037, 106.69868).

- **Get the details of a bus stop**.
  - URL: https://api.bktraffic.com/api/bus/stop-details.
  - Method: GET.
  - Query parameters: `id`.
  - Example request URL: https://api.bktraffic.com/api/bus/stop-details?id=2931.
  - Example response:


```
{
    "code": 200,
    "message": "success",
    "data": {
        "_id": 2931,
        "address_no": "432",
        "code": "QTP 117",
        "lat": 10.794048,
        "lng": 106.628799,
        "name": "Tân Sơn Nhì",
        "routes": "41",
        "status": "Đang khai thác",
        "stop_type": "Trụ dừng",
        "street": "Tân Sơn Nhì",
        "support_disability": false,
        "ward": "Phường Tân Sơn Nhì",
        "zone": "Quận Tân Phú",
        "location": {
            "type": "Point",
            "coordinates": [
                106.628799,
                10.794048
            ]
        }
    }
}
```


Figure 26: The response containing the details of the bus stop with the ID 2931.

- **Get all bus routes**.
  - URL: https://api.bktraffic.com/api/bus/routes.
  - Method: GET.
  - Query paramaters: none.
  - Example request URL: https://api.bktraffic.com/api/bus/routes.
  - Example response:


```
{
    "code": 200,
    "message": "success",
    "data": [
        {
            "_id": "01",
            "route_name": "Bến Thành - Bến xe buýt Chợ Lớn"
        },
        {
            "_id": "03",
            "route_name": "Bến Thành - Thạnh Xuân"
        },
        {
            "_id": "04",
            "route_name": "Bến Thành - Cộng Hòa - Bến xe An Sương"
        },
        // ...
}
```


Figure 27: The response containing all the bus routes of Ho Chi Minh City.

- **Get the details of a bus route**.
  - URL: https://api.bktraffic.com/api/bus/route-details.
  - Method: GET.
  - Query parameters: `id`.
  - Example request URL: http://api.bktraffic.com/api/bus/route-info?id=08
  - Example response:


```
{
    "code": 200,
    "message": "success",
    "data": {
        "_id": "08",
        "route_name": "Bến xe buýt Quận 8 - Đại học Quốc gia",
        "agencies": "Hợp tác xã vận tải xe buýt Quyết Thắng, ĐT:
(028)38.642.712",
        "route_type": "Phổ thông - Có trợ giá",
        "distance": "32.70 km",
        "vehicle_type": "68 seats",
        "operation_time": "Operation time: 04:40 - 20:30",
        "fare_price": "- Vé lượt trợ giá: 7000 VNĐ\n      - Vé lượt
trợ giá HSSV: 3000 VNĐ\n      - Vé tập: 157500 VNĐ",
        "total_trips": "348 Trip/day",
        "trip_time": "80 - 90 minutes",
        "trip_spacing": "3 - 10 minutes"
    }
}
```


Figure 28: The response containing the details of the bus route with the ID 08.

- **Get the bus stops along a bus route's path**.
  - URL: https://api.bktraffic.com/api/bus/route-stops.
  - Method: GET.
  - Query parameters: `id`, `leg`.
  - Example request URL: https://api.bktraffic.com/api/bus/route-stops?id=08&leg=forward.
  - Example response:

```
{
    "code": 200,
    "message": "success",
    "data": {
        "type": "FeatureCollection",
        "features": [
            {
                "type": "Feature",
                "geometry": {
                    "type": "Point",
                    "coordinates": [
                        106.656357,
                        10.733545
                    ]
                },
                "properties": {
                    "name": "Bến xe buýt Quận 8",
                    "stopType": "Bến xe"
                }
            },
            {
                "type": "Feature",
                "geometry": {
                    "type": "Point",
                    "coordinates": [
                        106.656346,
                        10.73646
                    ]
                },
                "properties": {
                    "name": "Bùi Minh Trực",
                    "stopType": "Ô sơn"
                }
            },
            // ...
        ]
    }
}
```


Figure 29: The response containing the array of the bus stop IDs along the route 08.

- **Get the path of a bus route**.
  - URL: https://api.bktraffic.com/api/bus/route-path.
  - Method: GET.
  - Query paramaters: `id`, `leg`.
  - Example request URL: https://api.bktraffic.com/api/bus/route-path?id=08&leg=forward.
  - Example response:

```
{
    "code": 200,
    "message": "success",
    "data": {
        "type": "Feature",
        "geometry": {
            "type": "LineString",
            "coordinates": [
                [
                    106.65635681,
                    10.7335453
                ],
                [
                    106.65635681,
                    10.7335453
                ],
                [
                    106.65668488,
                    10.73355579
                ],
                // ...
            ]
        }
    }
}
```


Figure 30: The response containing the line string path of the bus route 08.

4.2.2 Deployment of the Bus API

The deployment process was made possible first by accessing the server via the SSH protocol. The server is running Ubuntu 20.04 LTS with the NodeJS version 12.22.12 installed. It was crucial to first upgrade this NodeJS version to a more recent one due to this version being out of the active support. We have upgraded the NodeJS version to version 16.16.0 (LTS) instead of the newer version 18 or above, because the server hardware is also not very powerful, and the newer versions of NodeJS are not compatible. We also noticed that there was no Node Version Manager (NVM) installed on the server. This is a tool that allows us to install and manage multiple versions of NodeJS on the same machine, therefore, we installed the new node version with NVM.

The next step was to upgrade PM2. PM2 is a process manager for NodeJS applications. It allows us to keep the application running in the background, restart the application automatically when it crashes, and monitor the application's resource usage. We have upgraded PM2 to accompany the new NodeJS version successfully.

We use GitLab to host our source code. After the code was ready on the GitLab repository, we issued a `git pull` on the server's terminal to pull the latest code to the correct directory. We then installed the dependencies using `npm install` and restarted the process with `pm2 restart BKTrafficApi`. Figure 31 shows the running server process.

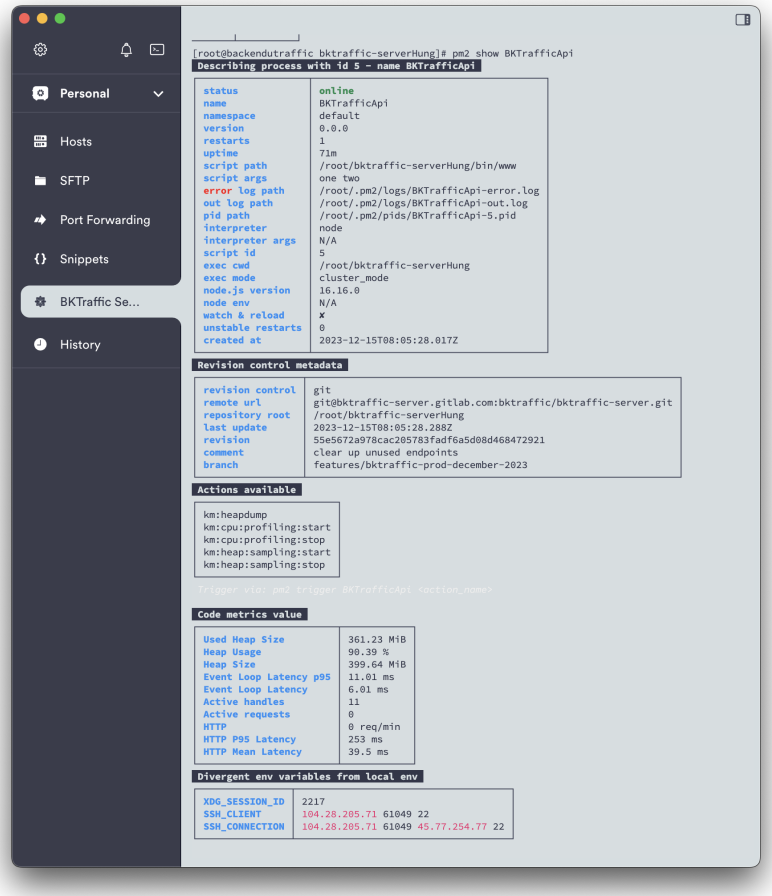


Figure 31: The server information and the running NodeJS process.

4.3 Concepts for parking services

In addition to advancing bus services, our group aims to implement functionalities related to parking. Due to the absence of a centralized data source, a model similar to the one adopted by Viettel (referenced in 2) is deemed more suitable for populating our database with information on available parking lots. However, to circumvent financial complexities, incentives offered to parking lot owners will be confined to analytics and the capability to promote their parking spaces to the application's user base.

4.3.1 Functional requirements

1. **User Registration and Authentication**:
   - Users (parking lot owners) must be able to register and create accounts securely.
   - Authentication mechanisms, such as email verification or two-factor authentication, should be in place.
2. **Space Listing**: Parking lot owners should be able to list their available parking spaces, specifying details such as location, pricing, availability, and any special features.
3. **Photo Upload**: Users should be able to upload images of their parking spaces to provide visual information to potential customers.
4. **Search and Filterting**:
   - Users should be able to search for parking spaces based on location, availability, pricing, and other criteria.
   - Filtering options should be provided to refine search results.
5. **Real-Time Availability Information**: The module should display real-time availability status of parking spaces, indicating whether a space is currently vacant or booked.

4.3.2 Non-functional requirements

1. **Performance**: The module should be responsive and perform efficiently, even during peak usage times
2. **Security**:
   - Ensure the security of user data, payment information, and sensitive details.
   - Implement secure authentication and authorization mechanisms.
3. **Scalability**: The system should be able to scale with an increasing number of users, parking spaces, and transactions.
4. **Usability**: The user interface should be intuitive and user-friendly for both parking lot owners and renters.
5. **Reliability**: Ensure that the system is available and reliable, with minimal downtime for maintenance.
6. **Cross-Platform Compatibility**: The module should be accessible on various devices and platforms, including web browsers and mobile applications.

7. **Geolocation Accuracy**: Ensure that the geolocation feature accurately identifies the location of parking spaces on the map.
8. **Error Handling**: Implement robust error handling to provide clear and informative error messages to users

4.3.3 Use cases

1. **List parking spaces**
   - **Actor**: Parking lot owner
   - **Description**: Parking lot owners can list their available parking spaces by providing details such as location, pricing, availability, and any special features. They can also upload photos of the space.
2. **Manage parking listings**
   - **Actor**: Parking lot owner
   - **Description**: The parking lot owner can view, edit, update, or remove their parking space listings. They can adjust pricing, mark spaces as unavailable, or delete listings as needed.
3. **Search for parking spaces**
   - **Actor**: Customer
   - **Description**: Users looking for parking spaces can search for available options based on location, date, time, and price. They can also use filters to narrow down the search results.
4. **View parking space details**
   - **Actor**: Parking lot owner, customers
   - **Description**: Both parking lot owners and users can see the real-time availability status of parking spaces
5. **Access account settings**
   - **Actor**: Parking lot owner, customers
   - **Description**: Users can access their account settings to view and edit their profile information and change their password.

Below is a diagram recapping these usecases

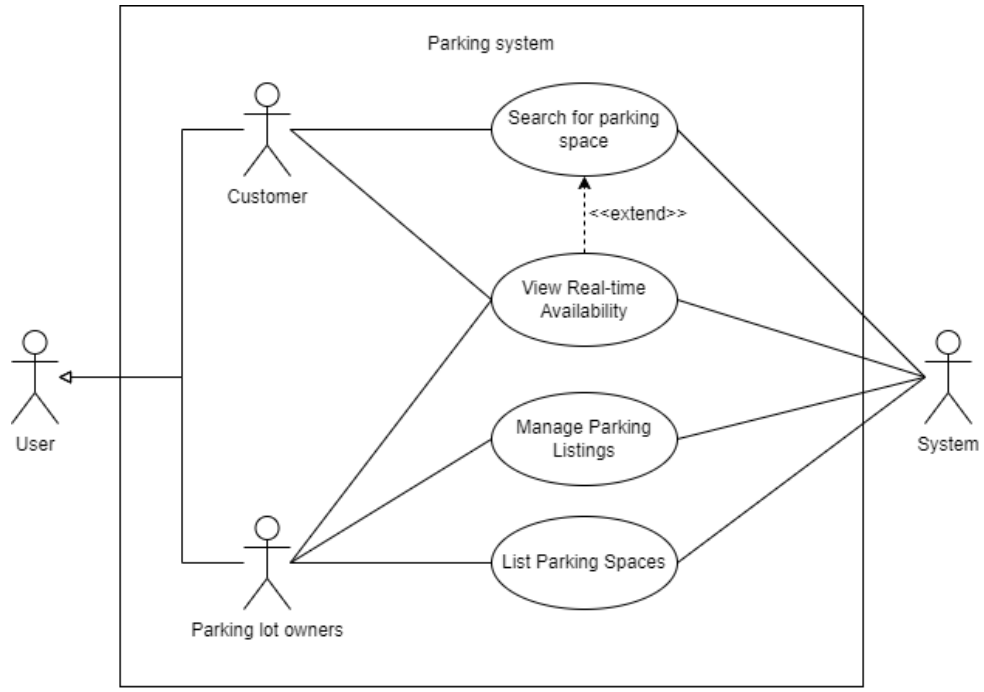


Figure 32: Activity diagram of the module

4.3.4 Diagrams

4.3.4.a Sequence diagrams

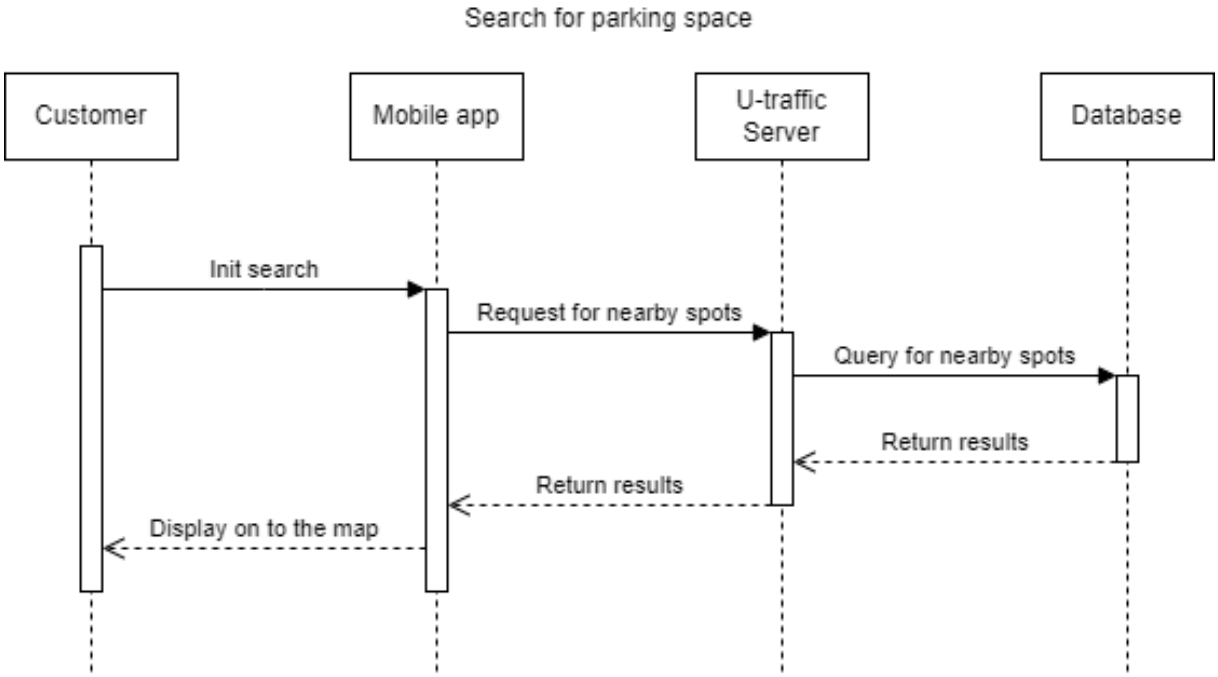


Figure 33: Sequence diagram for searching nearby parking lots

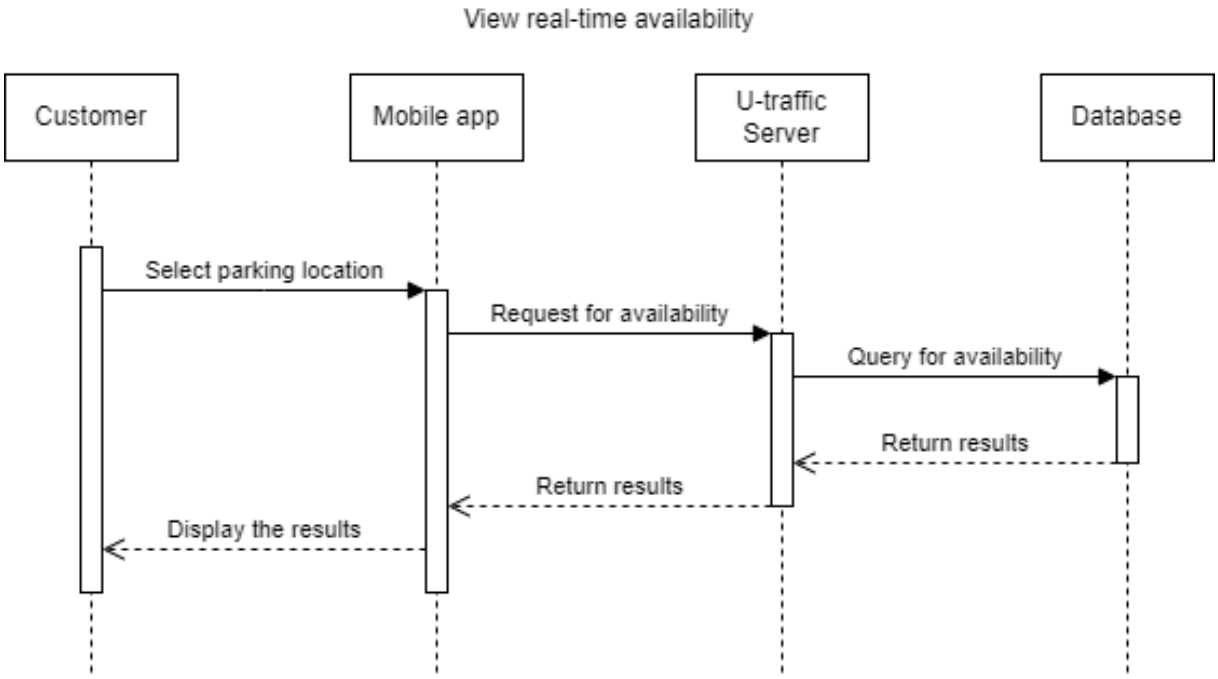


Figure 34: Sequence diagram for checking parking lot availability

The ability for users to promote their own parking spaces for rental is akin to the previous features, with an additional verification process to ensure essential information is provided

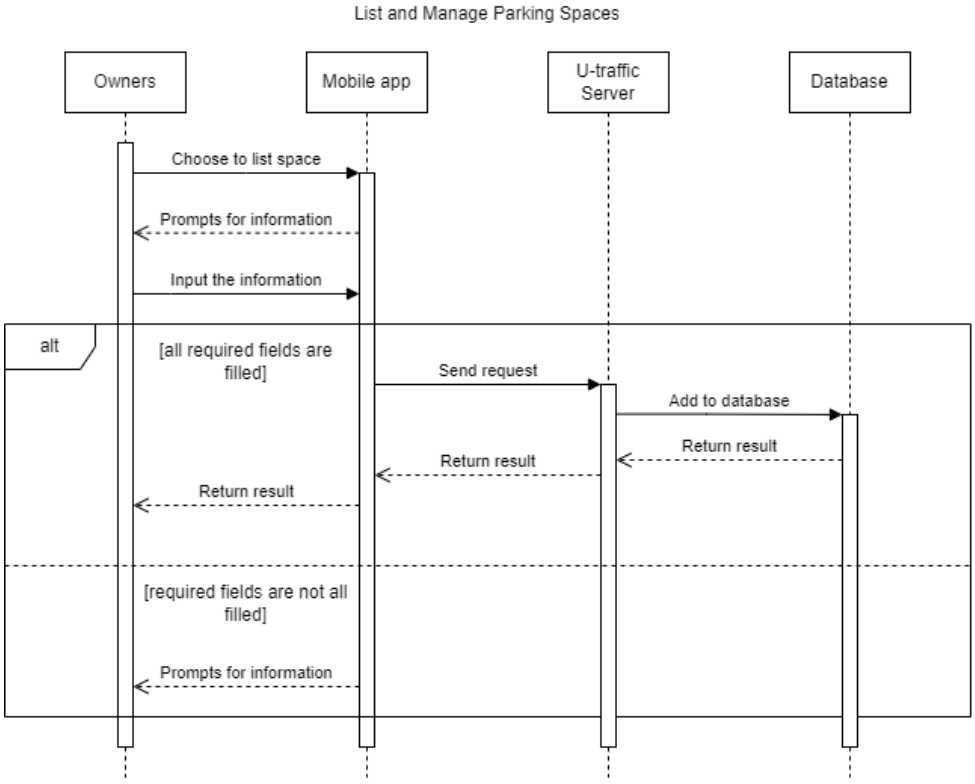


Figure 35: Sequence diagram for advertising parking spaces

4.3.4.b Activity diagram

For the act of searching and observing various parking spots, activity diagrams are unnecessary because these tasks involve querying without decision-making. Activity diagrams are provided for adding new parking lots and modifying the number of available parking spots

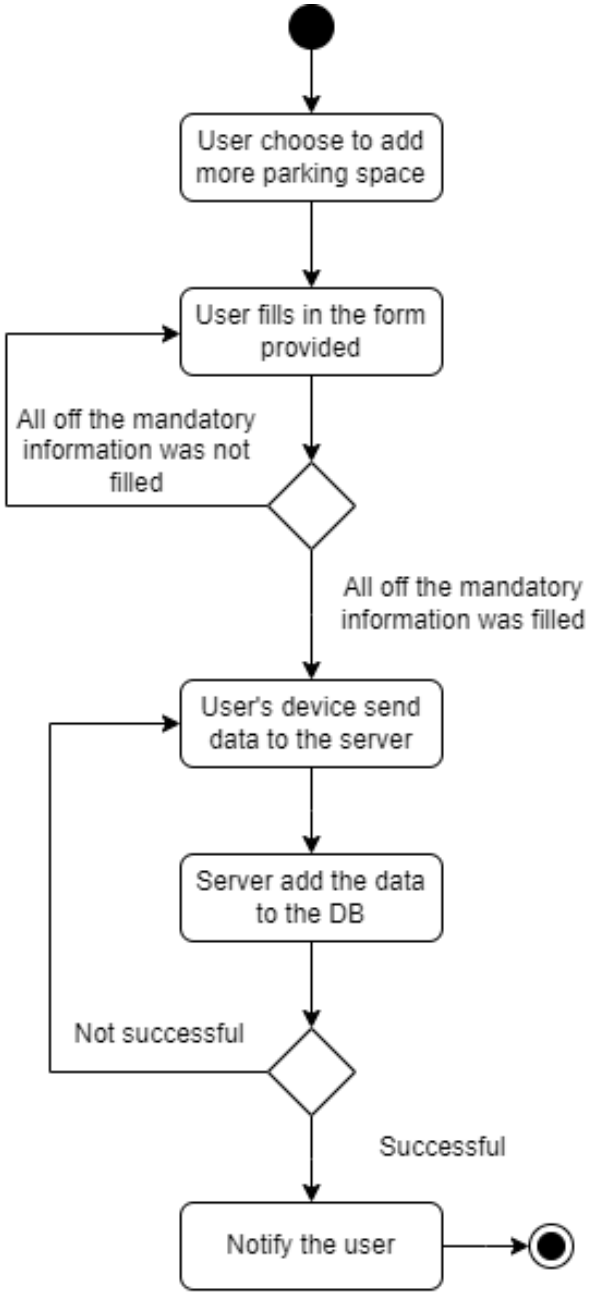


Figure 36: Acitivity diagram fo adding new parking lots

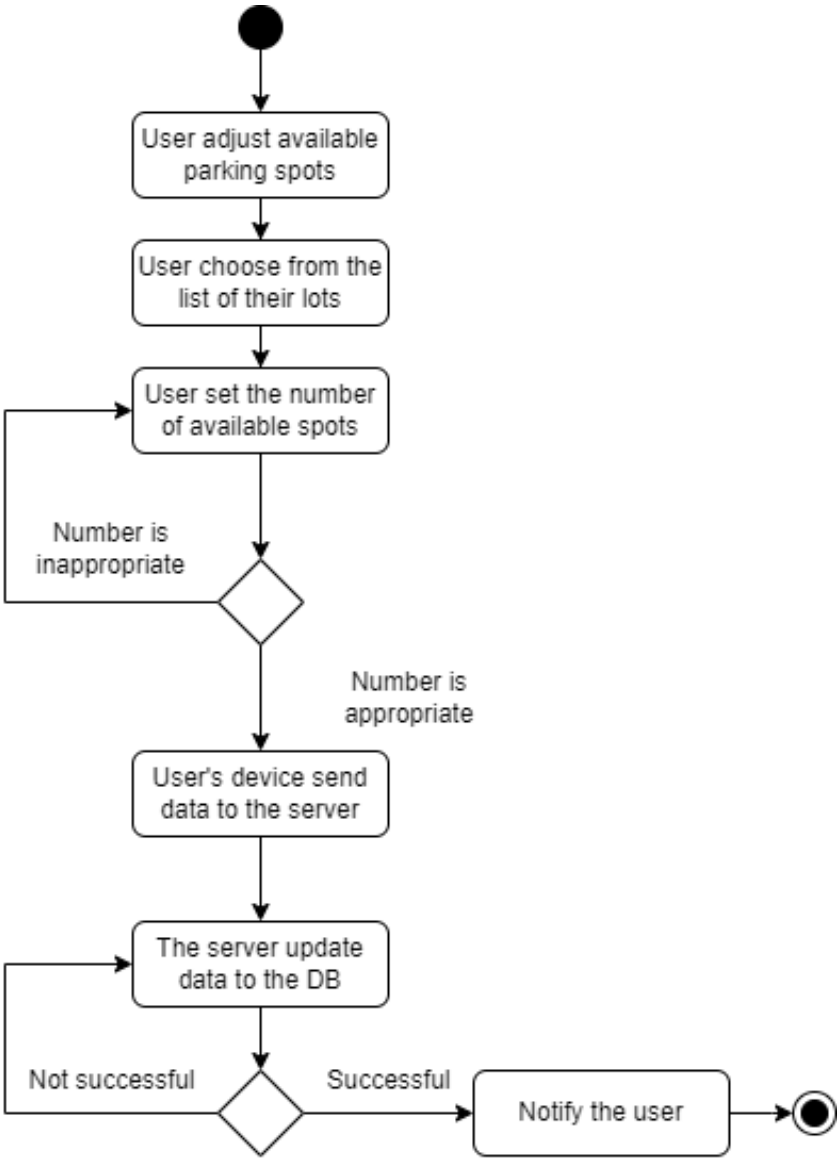


Figure 37: Adjusting the number of available spots

4.3.4.c ERD diagram

In the realm of parking lot management, a well-structured Entity-Relationship Diagram (ERD) plays a crucial role in database system development. This ERD is thoughtfully crafted to address the specific requirements of parking lot management, offering a clear depiction of the connections between various entities. Central to this ERD is the ”owns” relationship, serving as a linchpin to associate weak entities, representing parking lots, with their respective owners. These parking lots exhibit a diverse range of attributes, including name, address, geographical coordinates, the count of available parking spaces, and even the capacity to store visual references through photos. Although the server do have other information for our users, this diagram only includes their UID to serve as a primary key.

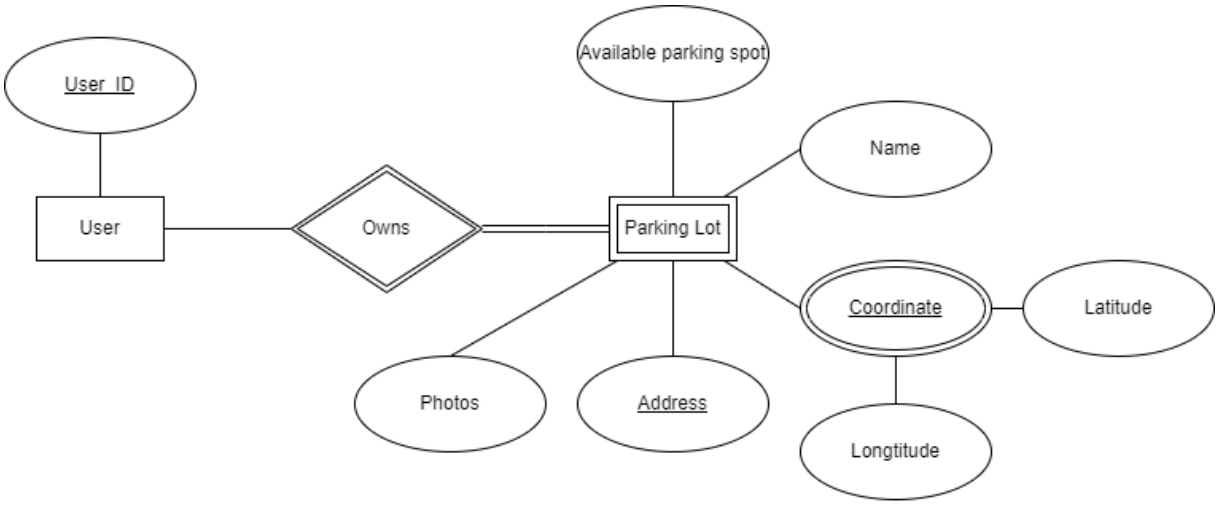


Figure 38: The ERD for storing parking lots and their relationships

4.4 iOS App Development

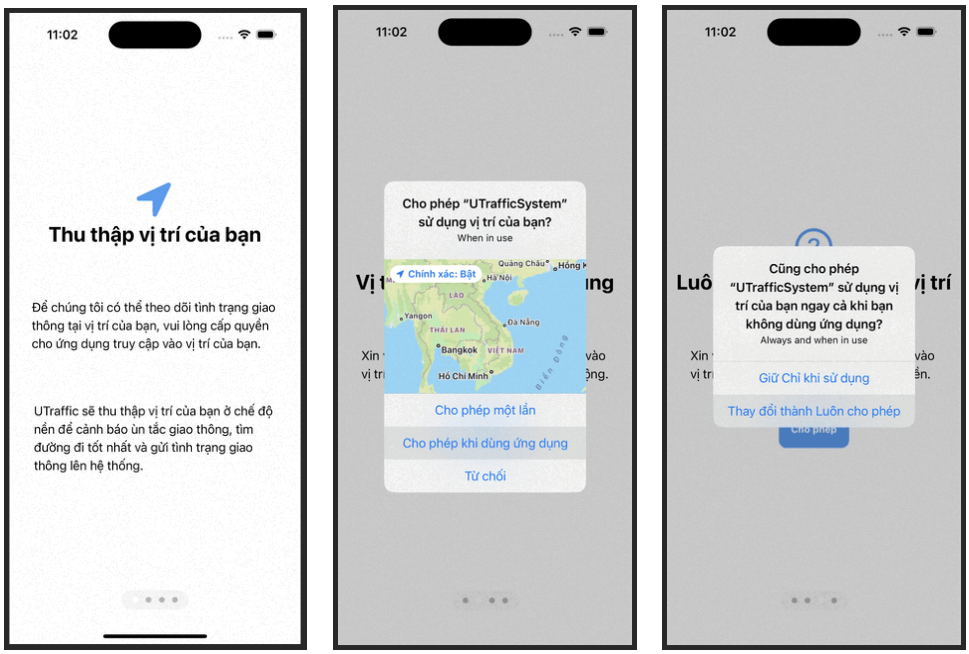
The iOS app is developed using Swift 5.9 and Xcode 15. The app is compatible with iOS 16.0 and above.

First and foremost, this app must at least offer the basic functionalities like that of the Android counterpart. This includes authentication, viewing the traffic status report, and the ability to communicate with the server. Other requirements are that the app must also be lightweight, performant, and consistent with the ongoing design of the Android app to an extent. During the development of this prototype, we have also taken into consideration the possibility of future expansion, such as adding more features and improving the user experience, as well as following Apple's best practices when it comes to iOS app development.

However, the similarities between the two versions should only be implemented to an extent, given the core differences between the two operating systems. We do not expect the complete iOS app to be *identical* to its Android counterpart, nor the Android app's new functionalities such as the bus services to be the same as in iOS.

4.4.1 The landing page and permission requests

Because the UTraffic app must know the user's real-time location in order to gather GPS data and also display nearby traffic status, it is crucial that we ask the users for location permissions upon the first app launch. This is done by presenting a dialog box that asks for the user's permission to access their location. The choice is saved into `UserDefaults`. `UserDefaults` is an interface to the user's defaults database, where key-value pairs of data are stored persistently across launches of the app. [17]. Therefore, this process will not be repeated on subsequent app launches. The onboarding process is divided into three main phases: greetings, location request, and always-on location request. They are as follows (the texts are in Vietnamese to first accommodate the Android app's design, but we are adding an option for language change in the next phase):

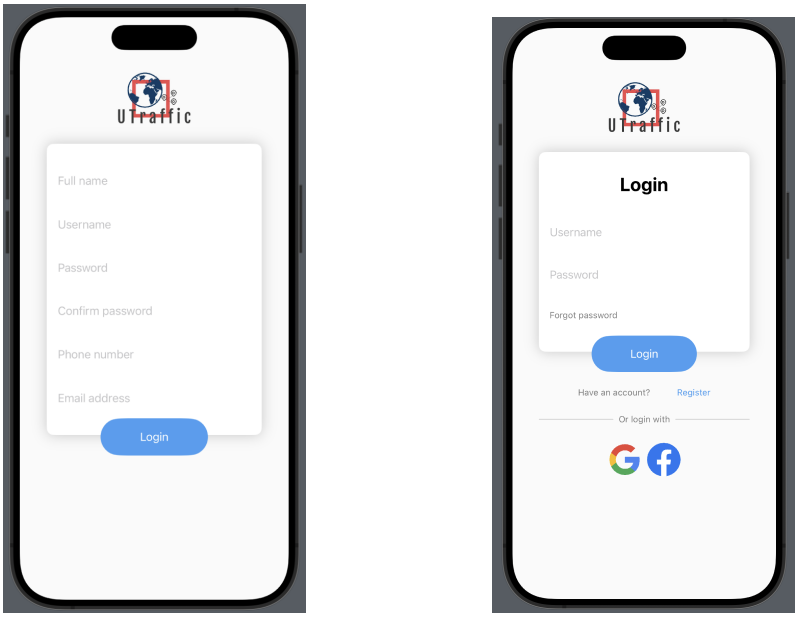


(a) Greetings. (b) Asking for location permission. (c) Asking for always-on location permission.

Figure 39: The onboarding process of the iOS UTraffic app.

4.4.2 Account registration and login

We adapt most of the user interface from the Android app to the iOS app in order to provide a consistent experience of the app across platforms. The account registration and login screens are as follows:



(a) The iOS account registration view. (b) The iOS app's login view.

Figure 40: The authentication views on the iOS UTraffic app.

4.4.3 Viewing the traffic status report

We adopted the display logic from the Android app, where the northeast and southwest bounds of the map's visible area, along with the street level, are taken into account. However, since the Android app uses the original response data provided by the `/get-status-v2` endpoint, using it on the iOS app means we have to develop additional classes to parse the data into the format that the iOS app can understand. We implemented the `/get-status-v3` for this exact reason to support the iOS app better.

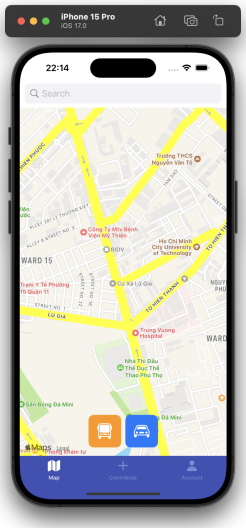


Figure 41: The home screen with the traffic status displayed.

4.4.4 Basic bus browsing

In this phase, we have implemented the bus list view within our iOS app. It is a view that displays all the routes within the database via the `/routes` endpoint.

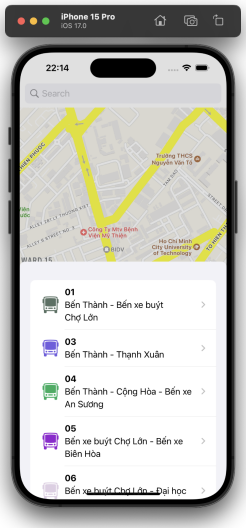


Figure 42: The bus list view displayed in a bottom sheet.

4.4.5 Integration with HCMUT TrafficView

We have developed an UI to see how the elements of HCMUT TrafficView would look like on iOS. The result is as follows:

Figure 43: The traffic camera view within the iOS app.

Because of mobile devices' limited processing power, it is best that we only show a snapshot of the camera at the time the user issues the request. This is to avoid programming complexity of having to deal with stream data, as well as to avoid the app from crashing due to memory issues.

5 WORK EVALUATION AND FUTURE PLANS

5.1 Work Evaluation

No.	Content	Duration
1	**Study existing systems**	**1 week**
1.1	Study UTraffic's services	0.5 week
1.2	Study other existing services	0.5 week
2	**Prepare necessary knowledge**	**4 weeks**
2.1	Prepare necessary Front-end knowledge	1 week
2.2	Prepare necessary Back-end knowledge	1 week
2.3	Prepare necessary Database knowledge	1 week
2.4	Prepare necessary Model knowledge	1 week
3	**Implementation**	**6 weeks**
3.1	Gather bus-related data	2 weeks
3.2	Bus services APIs	2 weeks
3.3	Implement application's iOS Front-end	2 weeks

5.2 Future Plans

No.	Content	Duration
1	**Implement bus services**	4 weeks
1.1	Add appropriate APIs to the system	2 weeks
1.2	Implement features for searching and filtering bus stops and routes	1 week
1.3	Implement features for suggesting bus routes between locations	1 week
2	**Implement parking services**	4 weeks
2.1	Add appropriate APIs to the system	2 weeks
2.2	Implement features for registering, managing, and searching for parking lots	2 weeks
3	**Implement iOS application**	4 weeks
3.1	Import existing features from Android	2 weeks
3.2	Implement bus services	2 weeks

5.2.1 Development strategy

We aim to deliver our iOS app to the Apple App Store, which would involve acquiring an Apple Developer Account and tailoring our app to suit Apple's developer guidelines. Because this process can take a long time due to the tedious nature of the Apple's review process, this work will be our major concern for the next phase of the project.

At the same time, after having access to the Google Developer console for the existing Android app, we were able to determine two necessary tasks, which are the data collection declaration to Google

regarding what UTraffic will collect from the users and the ability for the users to request for account deletion. This is explicitly required by both Google and Apple, and since UTraffic offers account creation, it must offer account deletion as well [18] [19]. Furthermore, we will work closely with the developing team of HCMUT TrafficView to bring their product into the UTraffic system in the next phase of the project.

Finally, we are working to close the gaps between the functionalities offered by the iOS and Android apps. This aligns with our ultimate goal of having the apps on the respective app stores.

5.2.2 Considerations for data collection

Because our bus data is static and retrieved via crawling instead of API calls, it is crucial to ensure that the data is up-to-date. While real-time data is a concern, we have a plan to implement tools for the UTraffic administrator to semi-automate the update process using a web extension. The extension will use the same logic of our scraping scripts but more intuitive and allow for easy data exports. We imagine a click of a button and then necessary JSON files will be generated.

We will implement the administration functionality to upload newer data and have them persist in the database as well. This will be done in the next phase of the project.

REFERENCE

[1] T. Ha and T. Dung, *Traffic congestion costs ho chi minh city $6 billion each year*, vi, https://tuoitrenews.vn/news/society/20221001/traffic-congestion-costs-ho-chi-minh-city-6-billion-each-year/69343.html, Accessed: 2023-12-13, Oct. 2022.

[2] *Utraffic mobile*, vi, https://bktraffic.com/home/mobile-app, Accessed: 2023-12-13.

[3] *UTraffic - homepage*, vi, https://bktraffic.com/home/, Accessed: 2023-12-13.

[4] N. H. Hoàng and N. T. Viễn, *Developing enhanced services based on data on the smart transportation application platform*, 2022.

[5] Apple Inc, *Updating apps that use web views - latest news - apple developer*, en, https://developer.apple.com/news/?id=12232019b, Accessed: 2023-12-13.

[6] MyParking, *My parking*, en, https://myparking.vn/, Accessed: 2023-12-16.

[7] D. Tran, H. Le, A. Van, K. Nguyen, and Q. Nguyen, *Hcmut trafficview: Automatically detecting traffic congestion for smooth routes*, Nov. 2023.

[8] *Cllocationcoordinate2d*, Apple Developer Documentation. [Online]. Available: https://developer.apple.com/documentation/corelocation/cllocationcoordinate2d (visited on 12/14/2023).

[9] *Openstreetmap (wgs84)*, hub.arcgis.com, Jun. 2019. [Online]. Available: https://hub.arcgis.com/maps/esri::openstreetmap-wgs84/about (visited on 12/14/2023).

[10] *Map and tile coordinates | maps javascript api*, Google Developers. [Online]. Available: https://developers.google.com/maps/documentation/javascript/coordinates.

[11] G. Geography, *World geodetic system (wgs84)*, GIS Geography, Oct. 2015. [Online]. Available: https://gisgeography.com/wgs84-world-geodetic-system/ (visited on 12/15/2023).

[12] *Geojson—arcgis online help | documentation*, doc.arcgis.com. [Online]. Available: https://doc.arcgis.com/en/arcgis-online/reference/geojson.htm (visited on 12/14/2023).

[13] MapBox, *Geojson.io*, geojson.io. [Online]. Available: https://geojson.io/ (visited on 12/15/2023).

[14] *Mkgeojsonobject*, Apple Developer Documentation. [Online]. Available: https://developer.apple.com/documentation/mapkit/mkgeojsonobject (visited on 12/14/2023).

[15] N. M. Q. D. Dương Hoài Phong, *Xây dựng ứng dụng di động cảnh báo tình trạng giao thông*, vi, 2020.

[16] V. A. N. Nguyễn Chính Khôi and L. D. Hưng, *Build a mobile app for data collection and urban traffic estimation*, en, 2023.

[17] *Userdefaults - foundation | apple developer documentation*, Apple.com, 2019. [Online]. Available: https://developer.apple.com/documentation/foundation/userdefaults.

[18] *Offering account deletion in your app - support - apple developer*, developer.apple.com. [Online]. Available: https://developer.apple.com/support/offering-account-deletion-in-your-app/ (visited on 12/15/2023).

[19] *Understanding google play's app account deletion requirements - play console help*, support.google.com. [Online]. Available: https://support.google.com/googleplay/android-developer/answer/13327111?hl=en (visited on 12/15/2023).

[20] *Trang chủ*, vi, http://buyttphcm.com.vn/, Accessed: 2023-12-14.

[21] *MongoDB manual*, vi, https://www.mongodb.com/docs/v3.0/reference/bson-types/, Accessed: 2023-12-12.

[22] M. Martin, *MVC vs MVVM – difference between them*, en, https://www.guru99.com/mvc-vs-mvvm.html, Accessed: 2023-12-13, Nov. 2023.

[23] Nimble App Genie and N. Sharma, *Objective c vs swift : Which one is the best?*, en, https://www.nimbleappgenie.com/blogs/swift-vs-objective-c/, Accessed: 2023-12-13, May 2023.

[24] *Educative answers - trusted answers to developer questions*, en, https://www.educative.io/answers/swift-vs-objective-c, Accessed: 2023-12-13.

[25] *The pros and cons of 8 popular databases*, en, https://www.keycdn.com/blog/popular-databases, Accessed: 2023-12-14.