Jeepili: A Mobile Application for Enhancing Jeepney Commuting in Iligan City through Augmented Reality and Smart Routing Algorithm

Zeus Morley S. Pineda
James C. Ocao
Roel S. Bayola

Supervised by
Malikey M. Maulana

Department of Computer Science
College of Computer Studies
MSU - Iligan Institute of Technology

May, 2025

A thesis proposal submitted in partial fulfilment of the requirements in CSC199 - Undergraduate Thesis.

www.msuiit.edu.ph

To our Families

Thank you for always being there through the stress, deadlines, and quiet breakdowns. Your support meant everything.

To Our Friends And Mentors

Thanks for keeping us sane. Whether it was helping us brainstorm or just listening to us rant, we couldn't have done this without you.

And to the late nights, failed attempts, caffeine, and small wins along the way — this project came to life because we didn't give up.

Salamat kaayo.

Acknowledgements

First, huge thanks to **Sir Malikey M. Maulana**, our thesis adviser, for guiding us throughout this process — from idea to code to defense. You somehow made every correction feel like encouragement.

To the thesis panel, thank you for all the questions, comments, and suggestions. You saw things we didn't, and that made Jeepili better.

To the **Computer Science Department**, thanks for the deadlines (yes, really), the lab resources, and the space to make this happen.

To our families — thank you for the food, the patience, and for not getting mad when we were too busy to reply.

To our friends, especially those who kept checking on us or cheered us on even when we looked like zombies — you know who you are. We owe you snacks.

And lastly, to the bug fixes that worked on the fifth try, and to ourselves for not quitting when it got tough — thank you.

Abstract

This study addressed the inefficiencies in navigating jeepney routes in Iligan City, where traditional methods such as static maps, signage, or verbal directions often failed to meet the needs of commuters. These methods were time-consuming, prone to inaccuracies, and incapable of adapting to dynamic urban conditions like traffic congestion, rerouted jeepney services, or road closures. As a result, commuters faced significant challenges in making informed transportation decisions, particularly when navigating unfamiliar routes or managing multiple transfers. This study developed a mobile application that integrated augmented reality (AR) to transform how commuters interact with transportation information. AR was used to visualize hailing points, stopping points, walking paths, and jeepney routes directly in the user's environment, offering an immersive navigation experience. The system was powered by the Nearest-Neighbor Search algorithm, which efficiently computed optimal routes based on user preferences such as faster, cheaper, or alternate paths. Real-time location data and traffic updates ensured dynamic and accurate routing. This study highlighted the role of AR as a tool for enhancing urban mobility and improving commuter decision-making, alongside advancements in algorithmic efficiency and real-time data integration.

Keywords: AR (augmented reality), route optimization, Iterative Radius Search algorithm, public transportation.

Contents

List of Tables

List of Figures

List of Algorithms

List of Abbreviations

AR Augmented Reality
IDS Iterative Deepening Search
CPDO City Planning and Development Office
LTFRB Land Transportation Franchising and Regulatory Board

1

Research Description

In modern computational systems, the efficient organization and processing of data remain pivotal challenges. The realm of algorithm design, particularly in the field of search and optimization, plays a critical role in addressing these challenges. Efficient routing, one of the most prominent problems in algorithm research, is critical in applications ranging from network design to urban transportation systems (Hart et al. (1968); Yen (1971); Dhulipala et al. (2021)).

Traditional methods of understanding jeepney routes—such as asking friends or locals, or relying on pure knowledge—are often impractical, especially in real-time decision-making. Many systems, like Iligan City's transportation network, remain confusing, leaving commuters to navigate incomplete or outdated information. This would result in frustration, delays, and increased risk of errors, highlighting the need for a more reliable solution. These challenges emphasize the need for a solution that combines real-time data with efficient algorithms to deliver reliable and user-friendly transportation guidance.

The Iterative Radius Search algorithm, central to this study, offers a novel approach to solving such problems by incrementally expanding a search radius to identify the nearest relevant nodes within a graph structure. This approach builds on advancements in heuristic-based algorithms and leverages deep reinforcement learning to optimize transit networks. By integrating principles from heuristic algorithms with practical computational strategies, this algorithm is designed to handle sparse data and adapt dynamically to the constraints of a decentralized transportation network (Holliday et al. (2024); Guan et al. (2022)).

This research further explores how such algorithmic innovations can be combined

with modern technologies, including AR, to enhance usability and accessibility. AR serves as a powerful interface, allowing users to interact with the system in an immersive way. While prior studies focused on optimizing routing performance, this study extends their insights by incorporating AR to visualize markers, routes, and paths. By abstracting the complexities of real-world transportation systems into a graph-based model, the study aims to improve commuter experiences and set a foundation for innovative transportation solutions. Integrating AR in traveling has the potential to redefine how commuters interact with urban transportation, providing an engaging, accessible, and immersive experience. The findings contribute to the broader field of algorithmic optimization, with potential applications extending beyond transportation to network routing, logistics, and other domains requiring efficient pathfinding solutions (Cooke and Halsey (1996); Sutton and Barto (1998)).

1.1 | Background of the Study

Public transportation is a critical component of urban mobility, offering affordable and efficient travel options for city residents. In Iligan City, jeepneys serve as the primary mode of public transportation, yet navigating their routes remains a challenge for many, especially newcomers and infrequent commuters. The absence of standardized route maps and reliable information contributes to confusion, inefficiency, and delays for riders. This issue is particular for individuals who are unfamiliar with the city's very complicated road and jeepney networks, often leading to missed stops and unnecessary delays.

Traditional methods of understanding jeepney routes, such as relying on local advice or observing signage, often lack precision and are time-consuming. These limitations are particularly problematic in dynamic urban settings where road conditions, traffic, and route availability frequently change. As the city is growing and evolving and transportation methods grow more complex, these static, manual methods increasingly becomes insufficient for addressing the needs of modern commuters.

Despite these technological advancements, many transit systems in developing cities, including Iligan, still rely on traditional, paper-based route guides, which lack flexibility and fail to address the needs of modern commuters. This gap presents an opportunity to apply computational methods, mobile technology, and augmented reality (AR) to improve public transit accessibility and efficiency. This transition is not only about the technological improvement but also about aligning public transportation

systems, especially jeepneys', of today's technologically advanced and growing society.

With the integration of augmented reality (AR), this study aims to provide a more immersive experience for jeepney commuters by allowing users to visualize key information such as routes, markers, and paths directly in their surroundings. The augmented reality's immersiveness can potentially transform how commuters perceive and interact with such transportation systems, creating a more instinctive, and engaging experience This integration allows the users to use the application more efficiently while at the same time enhancing their user experience and making it enjoyable.

This study proposes the development of a mobile application that utilizes graph-based algorithms, geolocation technologies, augmented reality (AR), and real-time data to provide jeepney route recommendations in Iligan City. The system aims to guide users from their starting point to their destination by offering detailed information on jeepneys to take, boarding points, estimated travel times, and walking paths. By leveraging real-time traffic updates, the application will adapt to sudden changes in route availability, ensuring that commuters are always presented with the most efficient options. The integration of AR will enhance the user experience by allowing commuters to visualize routes, markers, and paths in their real-world environment. By combining computational methods with innovative visualization technologies, this study seeks to enhance urban mobility, improve commuter satisfaction, and contribute to the advancement of intelligent transportation systems,

1.2 | Statement of the Problem

Iligan City's current transportation and navigation applications do not support jeepney commuting and fail to provide essential features like real-time updates and adaptive route adjustments based on traffic or road conditions.

1.3 | Objectives

This study aims to address the challenges of public transportation routing in Iligan City through the development of an advanced computational model and algorithm while integrating AR to enhance user experience.

1.3.1 | General Objective

To design and develop an efficient and adaptable algorithm that optimizes jeepney route recommendations in Iligan City, incorporating AR features to provide commuters with an intuitive and visually interactive way of navigating public transportation systems.

1.3.2 | Specific Objectives

1. To collect data on jeepney routes, fares, and their schedules from city offices such as, City Planning and Development Office (CPDO) and the Land Transportation Franchising and Regulatory Board (LTFRB).
2. To create a graph-based model representing the jeepney network, incorporating nodes, edges, and real-time traffic data.
3. To design and implement an efficient, non-iterative proximity-based search algorithm capable of computing jeepney paths based on distance, transfer count, and accessibility.
4. To develop a mobile application that integrates these algorithms and provides a user-friendly platform for commuters to access.
5. To integrate dynamic real-time data updates and augmented reality functionalities to adapt to live urban conditions and visualize boarding points, stops, and routes.
6. To provide an administrator page that can modify and update routes.

1.4 | Scope and Limitation

The scope of this research is the design, implementation, and evaluation of the routing algorithm that will be used for the mobile application. The dataset will be derived from the existing jeepney routes within Iligan city. Augmented Reality (AR) will also be integrated into the application to help the commuters visualize the map elements into the real-world. The application will also include real-time updates for traffic or jeepney route changes.

With these, the users can have jeepney commuting guidance in Iligan city with real-time traffic or route changes updates, while having the option to take advantage of the augmented reality (AR) feature.

There are limitations in the research. The location is constrained to Iligan city only. The focus of the evaluation of the algorithm is the performance and accuracy, and the user experience is considered but is not the main focus of the evaluation. Some of the real-time data will be simulated as there are some information that is not available in our current public transportation.

1.5 | Significance of the Study

The significance of this research lies in addressing the gaps in jeepney commuting by designing a routing algorithm that is fit for the jeepney transportation system in Iligan city and developing a mobile application that is integrated with Augmented Reality (AR) to show the routing algorithm in action with real-time and real-world data. The designed routing algorithm can also contribute to the field of public transportation since it has the potential to be applied into the public transportation of other places. The vehicle does not have to be a jeepney but any vehicle as long as their routes are fixed, like the jeepneys of Iligan City. The integration of AR could also offer a more intuitive and immersive experience in jeepney commuting.

1.6 | Research Methodology

1.6.1 | Jeepney Routes Data Collection

The jeepney routes will be sourced or gathered from the relevant offices in the city hall where the data regarding jeepney routes will be available. The main office that this information would be gathered from is the City Planning and Development Office (CPDO). Gathering the data from this reliable and official source will ensure that the data collected is uniform and credible. Alternatively, several offices in the city hall possess this information but the best choice would be from CPDO.

1.6.2 | Jeepney Routes Data Conversion

The data gathered will be in the form of a map with drawings of the routes of jeepneys. For convenience, a program will be used where it takes two coordinates and returns all the coordinates between them. In the database, each jeepney will have a set of coordinates corresponding to their entire route. It will be in the form of latitude and longitude, like (8.237, 124.244).

Algorithm 1: Coordinates Enumerator

Input: $coordinates\_start, coordinates\_end, gap$ (in meters)
Output: $list\_of\_coordinates$

Function `calculate_bearing`$(start, end)$
  $lat1, lon1 \leftarrow math.radians(start[0]), math.radians(start[1])$;
  $lat2, lon2 \leftarrow math.radians(end[0]), math.radians(end[1])$;
  $bearing \leftarrow$
   $math.atan2(math.sin(lon2 - lon1) * math.cos(lat2), math.cos(lat1) *$
   $math.sin(lat2) - math.sin(lat1) * math.cos(lat2) * math.cos(lon2 - lon1))$;
  $bearing \leftarrow math.degrees(bearing)$;
  return $(bearing + 360)\%360$
Function `enumerate_coordinates`$(coordinates\_start, coordinates\_end, gap)$
  $start \leftarrow coordinates\_start$;
  $end \leftarrow coordinates\_end$;
  $total\_distance \leftarrow geodesic(start, end).meters$;
  $bearing \leftarrow calculate\_bearing(coordinates\_start, coordinates\_end)$;
  $coordinates\_list \leftarrow [coordinates\_start]$;
  $current\_distance \leftarrow 0$;
  while $current\_distance < total\_distance$ **do**
    $current\_distance += gap$;
    if $current\_distance >= total\_distance$ **then**
      $coordinates\_list.append(coordinates\_end)$;
      break;
    $next\_point \leftarrow geodesic(meters =$
     $current\_distance).destination(start, bearing)$;
    $coordinaes\_list.append((next\_point.latitude, next\_point.longitude))$;
  return $coordinates\_list$

Algorithm 1 shows the code for the coordinates enumerator where it will take two coordinates, and the gap as the input, where the gap is how far the enumerated coordinates are apart in meters, and returns all the coordinates between these two given coordinates. The enumerated coordinates can be set to be a given distance apart, like every 10 meters and so on.

In lines 2-3, the latitude and longitude of the start and end coordinates are converted to radians since by default they are in degrees.

In line 4, is the formula for calculating the bearing between two coordinates which is the equation below:

$$\theta = \text{atan2}\left(\sin(\Delta\lambda) \cdot \cos(\phi_2), \cos(\phi_1) \cdot \sin(\phi_2) - \sin(\phi_1) \cdot \cos(\phi_2) \cdot \cos(\Delta\lambda)\right) \tag{1.1}$$

Where:

- ϕ_1, ϕ_2 are the latitudes of the start and end points.
- λ_1, λ_2 are the longitudes of the start and end points.
- $\Delta\lambda = \lambda_2 - \lambda_1$ is the difference in longitudes.

In line 19, an object that represents a coordinate is created that is *current_distance* meters away from the *starting_coordinates* and at a bearing of *bearing*. Then line 20 takes the latitude and longitude of this object and appends it to the list.

1.6.3 | Jeepney Routes Data Storage

The data collected will be stored in a database by *Firebase* as it is cloud-based and fits the structure needed to store Jeepney routes.

1.6.4 | Jeepney Fares Data Collection

The fare rates will be sourced from the Land Transportation Franchising and Regulatory Board (LTFRB) office in the city. Collecting the data from this source

will ensure that the information used by the application is reliable and accurate. Alternatively, if the LTFRB office in the city is unresponsive for a significant amount of time, the data will be gathered from a recent social media post or website from the LTFRB.

1.6.5 | Jeepney Fares Data Usage

The regular and discounted fares along with their base price and price per kilometer after four kilometers will be added in the formula for fare estimation.

Algorithm 2: Fare Calculation

Input: $distance, type$
Output: $fare$
Function `calculate_fare`$(distance, type)$
    $fare\_details \leftarrow \{$
     $'regular' : 'base\_price' : 13, 'add\_per\_km' : 1.8,$
     $'discounted' : 'base\_price' : 11, 'add\_per\_km' : 1.4$
     $\}$ $base\_price \leftarrow fare\_details[fare\_type]['base\_price']$
    $add\_per\_km \leftarrow fare\_details[fare\_type]['add\_per\_km']$
    if $distance <= 4$ **then**
        return $base\_price$
    $extra\_distance \leftarrow distance - 4$
    $extra\_fare \leftarrow extra\_distance * add\_per\_km$
    $total\_fare \leftarrow base\_price + extra\_fare$
    return $round(total\_fare, 0)$

Algorithm 10 shows the sample code for the fare estimation of both regular and discounted(Senior Citizens, Students, PWDs) commuters.

2

Review of Related Literature

This chapter explores the theoretical and practical foundations relevant to the study, focusing on the computational challenges and algorithmic approaches to urban transit optimization. This synthesizes related works on graph-based algorithms, iterative search techniques, real-time data integration in transportation systems, and augmented reality, providing a clear academic context for this study

2.1 | Graph Algorithms in Transportation Systems

Graph algorithms serve as the foundation for modeling and solving transportation network problems. These algorithms enable efficient representation and analysis of transit systems by using nodes to represent stops and edges to denote routes, often weighted by time, distance, or cost.

2.1.1 | Overview of Graph-Based Models

Graph-based models offer a structured way to represent transportation systems. Nodes represent stops or intersections, while edges denote routes, often assigned weights corresponding to factors like travel time, distance, or cost. Such models facilitate computational analysis by abstracting complex systems into manageable components. Applications of these models include route optimization, network flow analysis, and identifying bottlenecks in transit systems. For example, public transit networks in metropolitan areas often use graph models to simulate and optimize service

patterns, ensuring efficient operations and user satisfaction (Cooke and Halsey (1996); Yen (1971)).

2.1.2 | Shortest Path Algorithm

Shortest path algorithms are essential tools for finding the most efficient routes within a graph. Classical algorithms like Dijkstra's and Bellman-Ford guarantee optimal solutions by exhaustively exploring all possible paths, albeit with significant computational demands. Heuristic methods such as A* improve efficiency by integrating domain-specific knowledge, reducing the search space while maintaining solution accuracy (Hart et al., 1968). These algorithms are widely used in navigation systems, logistics, and urban planning. For instance, A* has been employed in GPS applications to guide users through congested city networks by dynamically recalculating routes based on live traffic data. Additionally, variants of these algorithms cater to specific requirements, such as time-dependent or probabilistic travel scenarios (Korf, 1985).

2.1.3 | Dynamic and Real-Time Graph Models

Dynamic graph models extend the capabilities of static graph algorithms by incorporating real-time updates, such as changes in traffic conditions, road closures, or transit schedules (Cooke and Halsey, 1996). These models rely on adaptive algorithms that can adjust to live data inputs, ensuring that computed solutions remain relevant (Sutton and Barto, 1998). For example, Dynamic Shortest Path algorithms adaptively modify edge weights based on traffic fluctuations, offering users optimal paths under changing conditions (Cooke and Halsey, 1996). Applications of these models include ride-sharing platforms, public transit scheduling systems, and emergency evacuation planning. A notable challenge in this domain is maintaining computational efficiency while integrating high-frequency updates, which often requires distributed computing or edge-based processing (Sutton and Barto, 1998).

2.1.4 | Multi-Criteria Optimization in Graphs

Multi-criteria optimization algorithms address the need to balance competing factors, such as minimizing travel time, cost, or transfers. These algorithms extend

traditional shortest path approaches by considering multiple objectives, generating solutions that cater to diverse user preferences. Pareto optimization is a common method used in this context, providing a set of trade-off solutions rather than a single optimal result. For instance, multi-criteria approaches are employed in multimodal transit systems to optimize routes based on user-defined parameters like fare affordability, travel speed, or environmental impact (Yen (1971); Hart et al. (1968)). Despite their utility, these algorithms are computationally intensive, often necessitating approximation techniques or heuristic frameworks to ensure scalability in large networks.

2.1.5 | Nearest-Neighbor and Proximity-Based Search

Nearest-Neighbor Search (NNS) algorithms are widely applied in systems that rely on spatial data and real-time decision-making, especially in the context of urban mobility and transportation. In these systems, NNS helps determine which routes or stops are nearest to a user's location—an essential process for mobile applications that need to respond to the user's position in real time.

(Feng et al., 2024) introduced a nearest-neighbor algorithm tailored for public transportation networks. Their approach utilizes a pre-processed transit index that incorporates real-world factors such as transfers and stop timings. Instead of relying solely on circular radius-based queries, the algorithm considers the actual structure and flow of the network. This enables significantly faster and more accurate search results, making it well-suited for mobile systems that require immediate route recommendations. In Jeepili, a similar real-time need exists—users expect instant suggestions for nearby jeepneys after entering a location. The principles in Feng et al.'s method directly inform this requirement by emphasizing performance without compromising accuracy (Feng et al., 2024).

Meanwhile, (Abeywickrama et al., 2016) explored the use of k-Nearest Neighbor (k-NN) algorithms applied specifically to road network queries. Their system was designed to work entirely in-memory, enabling fast lookup speeds and minimal delay. Unlike expanding-radius methods, their design efficiently navigates road data using spatial indexing, which is beneficial when dealing with large datasets. This concept inspired Jeepili's use of a Proximity-based Route Matching Search (PRMS), a one-pass variant that reduces overhead by skipping iterative radius expansion. By operating in a single pass and leveraging dynamic thresholds, PRMS maintains speed regardless of

the user's walking distance to the nearest route (Abeywickrama et al., 2016).

Both studies demonstrate the utility of proximity-based search methods in transportation-focused systems, particularly when location-based results must be retrieved quickly and reliably. Additionally, their scalability aligns well with Jeepili's long-term goal of supporting a growing number of jeepney routes without incurring performance drops. These foundations strongly influenced the routing component of Jeepili, especially in its real-time path suggestion system where nearest-neighbor logic is critical.

2.2 | Advances in Iterative Search Technique

Iterative search techniques are pivotal in addressing computational challenges posed by large and complex transportation networks. These techniques optimize the search process by progressively refining the search space, ensuring efficient exploration without exhaustive computation. By iteratively improving solutions, these methods strike a balance between computational efficiency and accuracy, making them suitable for real-world applications.

2.2.1 | Iterative Deepening Search

Iterative Deepening Search IDS combines the advantages of depth-first and breadth-first search techniques. It systematically explores all possible paths within increasing depth limits, ensuring that the optimal solution is reached without exceeding memory constraints. IDS is particularly useful in scenarios where the depth of the solution is unknown, such as route optimization in vast transit networks. For example, IDS has been applied in pathfinding algorithms for multimodal transportation systems to identify feasible routes under complex constraints (Korf, 1985).

2.2.2 | Bidirectional Search

Bidirectional search enhances efficiency by simultaneously searching from the start and goal nodes, effectively halving the search space. This approach is especially valuable in transportation systems where multiple paths connect origin and destination points. By meeting in the middle, bidirectional search reduces computation time,

making it ideal for real-time applications like ride-sharing and logistics planning. Practical implementations include its use in ride-hailing platforms to match drivers and passengers along optimal routes. (Xiang et al., 2021)

2.2.3 | Adaptive Iterative Methods

Adaptive iterative methods dynamically adjust search parameters based on real-time data, ensuring relevance in rapidly changing environments. These methods integrate feedback loops to refine their searches, adapting to updates such as traffic fluctuations or route disruptions. Reinforcement learning techniques, such as Q-learning, are often employed to guide these adaptive searches, enabling them to learn from historical data and improve future performance (Sutton and Barto, 1998). For example, adaptive methods are used in dynamic transit scheduling systems to adjust bus frequencies and routes based on passenger demand and road conditions.

2.2.4 | Scalability and Efficiency in Iterative Searching

One of the significant challenges of iterative search techniques is ensuring scalability while maintaining efficiency. Methods such as hierarchical decomposition, where large networks are broken into smaller subgraphs, help mitigate computational overhead. Additionally, parallel processing and distributed computing frameworks further enhance the scalability of iterative search methods, enabling their application in large-scale urban transit networks. These approaches have proven effective in optimizing operations for metropolitan bus and rail systems. (Manohar et al., 2024)

2.3 | API Integration for Multi-Platform Systems

API integration is essential for enabling seamless communication and data exchange across multiple platforms in modern transportation systems. By leveraging well-designed APIs, transportation networks can efficiently share real-time data, improving service adaptability and scalability.

2.3.1 | Overview of APIs

APIs (Application Programming Interfaces) serve as intermediaries that allow software applications to interact with each other. RESTful APIs and GraphQL are two popular standards used in transportation systems, offering flexibility and efficiency in data retrieval and manipulation. These APIs enable systems to access and manage data streams, such as transit schedules and traffic conditions, in real time (Ofoeda et al., 2019).

2.3.2 | APIs in Transportation Systems

In transportation, APIs play a critical role in aggregating and delivering data from diverse sources. For example, platforms like Google Maps and OpenStreetMap rely on APIs to provide routing, traffic updates, and navigation services. By integrating multiple data sources, transportation systems can offer comprehensive and accurate transit information to users, enhancing decision-making and service reliability (Aung et al., 2024)

2.3.3 | Multi-Platform Integration

Multi-platform API integration facilitates communication between various stakeholders, such as public transit agencies, ride-sharing companies, and end-users. This integration enables real-time coordination of schedules, route planning, and service updates. Examples include API-driven systems that synchronize bus schedules with train arrivals, reducing transfer wait times and improving the overall user experience. However, challenges such as ensuring data consistency, minimizing latency, and maintaining secure connections across platforms remain areas of active development (Aung et al., 2024).

2.4 | Sustainability and Real-Time GPS Tracking Systems

Real-time GPS tracking systems have significantly enhanced the sustainability of public transportation by improving user experience and operational efficiency (Chan et al., 2020). These systems provide passengers with accurate, real-time data on vehicle

locations, schedules, and service changes, thereby encouraging the use of public transit and reducing reliance on private vehicles (Chan et al., 2020).

2.4.1 | User Experience Improvements

Studies show that real-time GPS systems increase user satisfaction by providing reliable information on transit schedules and reducing uncertainty. The introduction of mobile apps like UniBus in Sarawak, Malaysia, resulted in higher commuter satisfaction, as passengers were able to plan their trips more effectively and minimize waiting times (Chan et al., 2020). Features such as alerts for delays and re-routing ensure that users are informed of service changes, fostering trust in public transit systems.

2.4.2 | Operational Benefits

For transit operators, GPS tracking systems enhance route optimization and fleet management. Real-time data allows for dynamic scheduling, better crowd management, and more efficient use of resources. These improvements contribute to reduced operational costs and increased service reliability, making public transit more competitive with private vehicles. ((Upadhyay et al., 2024))

2.4.3 | Environmental and Social Impacts

The use of real-time GPS tracking promotes sustainable urban mobility by reducing traffic congestion and lowering greenhouse gas emissions. Encouraging public transit adoption leads to fewer vehicles on the road, decreasing air pollution and contributing to cleaner cities. Furthermore, improved accessibility to reliable transit services supports equitable mobility for all societal groups, including those without access to private vehicles. (Chan et al., 2020)

2.5 | Role of Real-Time GPS Tracking Applications in Public Transportation

Real-time Global Positioning System (GPS) tracking applications have become essential tools in modern public transportation systems, enhancing key factors such as accessibility, reliability, comfort, safety, and overall user satisfaction. By utilizing GPS data, these applications provide real-time updates on vehicle locations, schedules, and potential delays, thereby improving the commuting experience.

2.5.1 | Accessibility and Reliability

The implementation of real-time GPS tracking systems in public transportation has a direct impact on improving accessibility and reliability. (Chan et al., 2020) emphasized that these systems deliver precise information about nearby transit stops, which reduces the effort required for commuters to locate transportation options. This increased convenience encourages more people to use public transit. Additionally, real-time updates ensure that users can rely on accurate schedules, which minimizes uncertainty and reduces waiting times.

2.5.2 | Enhanced Comfort and Safety

The use of GPS tracking applications significantly improves the comfort and safety of public transportation services. By analyzing passenger density data, operators can allocate additional vehicles during peak hours, reducing overcrowding and ensuring a more comfortable ride for commuters. Safety is also enhanced, as these systems provide timely alerts about route changes or emergencies, fostering a sense of security among passengers. (Chan et al., 2020) highlighted that these advancements improve user perceptions of public transit systems.

2.5.3 | Impact on Customer Satisfaction and Loyalty

GPS tracking applications play a vital role in increasing customer satisfaction and loyalty in public transportation. The transparency and convenience offered by these systems create a more positive overall experience, aligning with the expectations of

modern users. As observed by (Chan et al., 2020), customer satisfaction significantly increases after the integration of real-time GPS tracking systems, which promotes repeat usage and enhances loyalty to public transit services.

2.6 | Mobile Applications for Transportation service

Mobile applications have significantly transformed urban mobility by integrating advanced algorithms, real-time data, and user-friendly interfaces to provide commuters with efficient and accessible solutions. Key examples include:

International:

1. **Grab**: Known primarily for ride-hailing services, Grab uses GPS technology to connect passengers with drivers, offering features such as real-time tracking, route optimization, and fare calculation. While highly effective for point-to-point transit, its relatively high costs make it less accessible for users who depend on public mass transit systems. Additionally, Grab's user-friendly interface ensures seamless booking, but its focus remains on individualized transport rather than shared commuting solutions.
2. **Moovit**: A globally popular application, Moovit specializes in public transport navigation by providing detailed route planning, real-time updates, and service alerts for buses, trains, and other transit options. Despite its comprehensive features, first-time users may find its interface overwhelming due to the abundance of information it offers. However, its ability to integrate multimodal transportation makes it a versatile tool for commuters worldwide.
3. **Waze**: Best known for its community-sourced traffic data, Waze helps private vehicle drivers avoid congestion by providing real-time traffic updates and alternative routes. Unfortunately, it lacks functionality specific to public transportation, making it unsuitable for commuters relying on jeepneys, buses, or similar services. Waze's focus on private vehicles also limits its applicability for urban mass transit systems.

Local:

1. **Angkas**: A local motorcycle ride-hailing app in the Philippines, Angkas is widely used for navigating traffic-heavy urban areas. With its GPS-enabled

platform, Angkas connects passengers with motorbike riders, ensuring quick and efficient travel. While primarily focused on individual riders, its affordability and adaptability to narrow streets make it a practical option for commuters.

2. **Joyride**: Similar to Angkas, JoyRide offers motorcycle ride-hailing services but also caters to delivery needs. Equipped with real-time tracking features, JoyRide ensures timely rides and provides reliable communication between riders and customers. Its growing user base highlights its effectiveness in addressing the mobility challenges in urban settings.

3. **Maxim**: Maxim, an emerging player in the Philippine market, offers a range of services, from ride-hailing to delivery. Its platform supports real-time GPS tracking, providing users with updated information on ride availability and routes. The app's competitive pricing and versatility have made it a viable alternative for commuters looking for budget-friendly solutions.

Local applications like Angkas, JoyRide, and Maxim highlight the potential of technology in solving the Philippines' unique transportation challenges. By combining real-time GPS tracking, affordable services, and adaptability to urban conditions, these apps fill critical gaps left by international platforms. Their multifunctional features address both commuter and delivery needs, offering practical and scalable solutions tailored to local demands. This localized approach emphasizes the importance of technology-driven innovations in enhancing the efficiency and accessibility of urban transportation systems.

2.7 | Mobile Applications for Jeepney Guidance

There are a few mobile applications implemented in different cities in the Philippines where their function is to guide the people in their city with the city's public transportation:

1. **JTransit**: A mobile application focused on jeepney navigation in Cebu city. Offering over 60 jeepney routes, and calculation of fares for the entire trip.

2. **Sakay.ph**: A mobile application focused on commuting guidance in Metro Manila. Offering public transportation directions for Jeepneys, Trains, Buses, P2P, UV

Express, Pasig River Ferry, Libreng Sakay, and more. It also includes fare estimation for the entire trip.

The existence and availability of similar applications especially in populated cities in the Philippines highlights the potential for these applications to be implemented in other cities as well. Both applications receive positive feedback from their users who are mostly commuters who are not from these cities. It guides these visitors or even locals unfamiliar with their city's public transportation on how to navigate the city by using its public transportation.

2.8 | Augmented Reality in Transportation Systems

Augmented Reality (AR) has become an essential tool in transforming the way people interact with transportation systems. AR technology enhances real-world environments with digital information and interactive visual elements, offering a more immersive and intuitive experience for users. The integration of AR in both public transit and personal navigation systems can help improve commuter efficiency, safety, and satisfaction by offering dynamic, context-aware visual aids. As cities and transportation systems evolve with the help of modern technologies, AR plays a crucial role in bridging the gap between real-time data and user interaction.

2.8.1 | Overview of AR Technologies in Transportation

Augmented Reality (AR) technology has evolved into a powerful tool for enhancing transportation systems, offering innovative ways to improve navigation, user experience, and decision-making. In transportation, AR overlays digital information—such as maps, directions, and real-time traffic data—onto the physical environment, allowing users to interact with and visualize information directly in their surroundings. This integration of real-time data with the physical world is particularly useful in public transit, vehicle navigation, and pedestrian guidance systems (Mendoza-Ramírez et al., 2023).

One key form of AR in transportation is markerless AR, also known as location-based AR. Unlike traditional AR systems that rely on predefined markers, markerless AR uses environmental features like surfaces, patterns, or objects, combined

with technologies such as GPS, accelerometers, and computer vision, to accurately position and overlay digital content. For instance, AR navigation apps, such as Google Maps' Live View, utilize a combination of GPS, accelerometer, gyroscope, and machine learning to recognize landmarks and provide users with real-time, interactive directions in their physical environment. This technology enables seamless navigation by overlaying visual directions and key transit information directly onto the user's field of view, significantly improving wayfinding and commuter experience.

By leveraging these technologies, AR in transportation not only enhances the visibility and accessibility of important data but also ensures a more immersive and intuitive interaction with the transportation network, offering significant potential to optimize transit systems and improve user satisfaction (Mendoza-Ramírez et al., 2023)

2.8.2 | User Experience and Interaction with AR in Navigation

Augmented Reality (AR) in navigation systems has been shown to improve user experience by providing more intuitive and engaging ways to navigate, compared to traditional smartphone-based navigation. Smartphone navigation systems typically offer passive assistance, such as text or voice directions, which can distract users from their surroundings and hinder effective navigation. In contrast, AR systems integrate navigational information directly into the user's view of the real world, enhancing situational awareness and enabling users to receive real-time guidance while still focusing on the environment around them.

A comparative study conducted by (Lakehal et al., 2023) explored how smartphones and AR-based systems impacted user navigation in pedestrian environments. The study found that AR-based navigation systems improved the user's overall navigation experience by offering more intuitive, context-aware guidance. Users found it easier to follow directions and were less likely to make mistakes when navigating in unfamiliar areas with the help of AR systems. This is because the AR system presents directional cues in the user's line of sight, making it easier to align the digital guidance with physical landmarks, as opposed to having to shift attention between a screen and the environment.

The study's findings are in line with other research that has demonstrated the advantages of AR over traditional smartphone navigation in terms of user experience and navigation performance. Future studies with larger sample sizes could further confirm the benefits of AR in diverse real-world settings, especially in improving the

ease of navigation in complex environments.

2.9 | Review of Related Literature Summary

The literature highlights significant advancements in computational techniques and real-time data integration for urban transit optimization, particularly focusing on graph algorithms and iterative search methods. However, there is a gap in applying these approaches to enhance the integration of augmented reality (AR) and GPS-based mobile applications specifically for jeepneys in Iligan City. While existing studies emphasize real-time data and multi-criteria optimization, limited attention has been given to how AR and real-time tracking can improve jeepney navigation, user experience, and operational efficiency. This presents an opportunity to further explore how AR and real-time systems can optimize jeepney commuting and enhance commuter satisfaction in areas like Iligan City.

Features/Aspects	**Grab, Moovit, Waze, Angkas, Joyride, Maxim**	**JTransit**	**SakayPh**	**Proposed Approach**
Focus on Jeepney Routes	No	Yes	Yes	Yes
Routes Suggestions	No	Yes	Yes	Yes
Real-Time Data	Yes	Yes	Yes	Yes
Integrated with AR	No	No	No	Yes

Table 2.1: Features and Aspects Comparison

3

Theoretical Framework

3.1 | Graph-Based Model

We model each road intersection or junction as a node in a graph, and connect them via edges that follow the jeepney's path—sampled every 10 m along the actual route. This lets our proximity-based search algorithm determine which jeepneys will be nearest and instantly reroute when there are route changes with live traffic updates.

3.2 | Iterative Deepening Search (IDS)

What it is. Iterative Deepening Search performs a series of depth-limited depth-first searches, incrementally increasing the depth limit until a goal (e.g. a valid route) is found.

3.3 | Iterative Radius Search (IRS)

What it is. Iterative Radius Search applies the "deepening" concept to geographic distance: starting from a small radius around the user's location, it gathers candidate stops, then expands the radius until a route meeting the criteria is found.

Note. Although IRS aligned well with a distance-focused search, its repeated area expansions proved too slow in practice and has been superseded by PRMS.

3.4 | Nearest Neighbor Search (NNS)

What it is. Nearest Neighbor Search retrieves the k closest nodes in a metric space to a given query point—commonly implemented via spatial indexes such as k-d trees or R-trees.

Why we need it. NNS inspired our PRMS by finding the nearest jeepney points around the user.

3.5 | Proximity-based Route Matching Search (PRMS)

What it is. PRMS is an adaptation of the Nearest Neighbors Search(NNS) algorithm, specifically designed for geospatial route matching in public transportation systems. It finds the nearest jeepneys along the start and end location, and recommends them to the user.

Why we need it. By avoiding iterative expansions, PRMS computes optimal routes quickly. It performs a single-pass distance calculation of every jeepney towards the start and/or end location.

3.6 | Augmented Reality

What it is. We leverage device sensors (GPS, compass, gyroscope) together with the camera feed to overlay virtual markers—hailing points, stop area, walking path, and jeepney path onto the live view.

Why we need it. AR transforms abstract route information into spatial visuals, helping commuters intuitively "see" where to hail or stop their jeeps.

4

Research Methodology

This chapter shows the process of collecting the data, the technologies and development tools to be used, the system architecture, the implementation of the routing algorithm, the pseudocode of the implementation of the routing algorithm, the process flow of different setups, the implementation of Augmented Reality (AR), the development cycle, and the steps for evaluating the mobile application.

4.1 | Algorithm Design

The system models jeepney routes as graphs, where nodes represent coordinate points and edges connect them based on path sequence. This structure enables efficient route querying and analysis.

Alongside the search logic, the system includes classification rules that determine whether a route can be completed in a single ride, requires a transfer between two jeepneys, or needs alternate suggestions when preferred routes are excluded. These strategies allow the system to dynamically adjust to various commuting scenarios and user preferences.

4.1.1 | Iterative Radius Search

Iterative Radius search (IRS) is an adaptation of the Iterative Deepening search (IDS) algorithm. IDS iterates graphs by depth, similarly, IRS operates in a geospatial domain through expanding a radius in a 2D space. IRS searches for points of interest

(coordinates) within a radius, then continues on expanding until the desired conditions are met.

Algorithm 3: Iterative Deepening Search

```
Input: start_node, goal
Output: Result of search or None
Function IterativeDeepeningSearch(start_node, goal)
    depth ← 0;
    while True do
        result ← DLS(start_node, goal, depth);
        if result ≠ None then
            return result;
        depth ← depth + 1;
Function DLS(node, goal, depth_limit)
    if depth_limit == 0 then
        if node == goal then
            return node;
        else
            return None;
    else if depth_limit > 0 then
        foreach child Input: expand
        node do
            result ← DLS(child, goal, depth_limit − 1);
            if result ≠ None then
                return result;
    return None;
```

Algorithm 7 shows the code for the Iterative Deepening Search. It is adapted by Iterative Radius Search (Algorithm 4) to work for searching within maps.

Algorithm 4: Iterative Radius Search

Input: $start\_lat, start\_lon, jeepney\_routes\_db$

Output: List of nearby jeepneys

Function `HaversineDistance`$(lat1, lon1, lat2, lon2)$**:**
  Convert $lat1, lon1, lat2, lon2$ from degrees to radians;
  $dlat \leftarrow lat2 - lat1$;
  $dlon \leftarrow lon2 - lon1$;
  $a \leftarrow \sin^2(\frac{dlat}{2}) + \cos(lat1) \times \cos(lat2) \times \sin^2(\frac{dlon}{2})$;
  $c \leftarrow 2 \times \text{atan2}(\sqrt{a}, \sqrt{1-a})$;
  $radius\_of\_earth \leftarrow 6371000$; `// in meters`
  return $radius\_of\_earth \times c$;

Function `IterativeRadiusSearchRecursive`$(start\_lat, start\_lon, jeepney\_routes\_db, radius)$**:**
  $north \leftarrow start\_lat + \frac{radius}{111320}$;
  $south \leftarrow start\_lat - \frac{radius}{111320}$;
  $east \leftarrow start\_lon + \frac{radius}{111320 \times \cos(start\_lat)}$;
  $west \leftarrow start\_lon - \frac{radius}{111320 \times \cos(start\_lat)}$;
  $candidates \leftarrow []$;
  foreach $jeepney \in jeepney\_routes\_db$ **do**
    foreach $(lat, lon) \in jeepney.route$ **do**
      if $south \leq lat \leq north$ ***and*** $west \leq lon \leq east$ **then**
        $candidates.append((jeepney.id, lat, lon))$;
  $nearby\_jeepneys \leftarrow []$;
  foreach $(jeepney\_id, lat, lon) \in candidates$ **do**
    $distance \leftarrow$ `HaversineDistance`$(start\_lat, start\_lon, lat, lon)$;
    if $distance \leq radius$ **then**
      $nearby\_jeepneys.append(jeepney\_id)$;
  if *nearby_jeepneys is not empty* **then**
    return unique values of $nearby\_jeepneys$;
  return `IterativeRadiusSearchRecursive`$(start\_lat, start\_lon, jeepney\_routes\_db, radius + 10)$;

Function `Main`$(start\_lat, start\_lon, jeepney\_routes\_db)$**:**
  return `IterativeRadiusSearchRecursive`$(start\_lat, start\_lon, jeepney\_routes\_db, 10)$

Algorithm 4 shows the code for the Iterative Radius Search. Iterative Radius Search is an adaptation to Iterative Deepening Search and they are similar in many ways like: (1) The search depth is incrementally increased, it starts with a small search radius (10 meters) and increases it if no jeepney route coordinates are found. (2) The goal of both is to find a solution, in the case of IRS, to find a jeepney route coordinate within the search radius. (3) Like how IDS prevents searching too deep too early by going through depth by depth, IRS also starts at a smaller search radius to prevent exploring too large of a radius too early.

4.1.2 | Route Overlap Detection

By applying this structure, the system is able to perform efficient pathfinding and route recommendation. When the user provides a starting and destination location, the application identifies the jeepney routes whose coordinates are close to these two points. The algorithm then determines which routes overlap or connect the two locations by comparing nodes and their proximity. In cases where there is no direct route, the system finds a possible transfer between two jeepney lines by searching for common or nearby nodes between their paths.

4.1.3 | Scalability and Real-Time Adaptation

Modeling the transportation network as a graph also supports scalability and adaptability, as new routes can be added by inserting more nodes into the graph. This also provides the foundation for applying search algorithms and real-time updates, allowing the system to respond dynamically to change in route availability, traffic conditions, or road conditions.

4.1.4 | Nearest-Neighbor Search Algorithms

The route selection process involves searching for jeepney routes that are close to the user's start and destination locations. There are two different approaches considered for this process. The first is an initial adaptation of the Iterative Deepening Search (IDS) algorithm into a graph based variant called Iterative Radius Search (IRS), which was the initial design. The second is a simplified nearest-neighbor approach that was used for the final implementation due to its efficiency.

4.1.5 | Proximity-based Route matching Search

Proximity-based Route Matching Search (PRMS) is an adaptation of the K-Nearest Neighbors (KNN) algorithm, specifically designed for geospatial route matching in public transportation systems. While KNN traditionally finds the k closest points in a feature space, PRMS operates in a geospatial domain to find the most efficient routes between two points using public transportation.

Algorithm 5: K-Nearest Neighbors Search

Input: *query_point*, *dataset*, *k*
Output: *k* nearest neighbors

```
1 Function KNN (query_point, dataset, k)
2     distances ← empty list;
3     foreach point Input: d
4     ataset do
5         distance ← calculateDistance (query_point, point);
6         distances.append((point, distance));
7     distances ← sort (distances by distance);
8     return first k points from distances
```

Algorithm 8 shows the code for the K-Nearest Neighbors Search. It is adapted by Proximity-based Route Matching Search (PRMS) to work for finding the nearest jeepney routes.

Algorithm 6: Proximity-based Route Matching Search

Input: *start_lat*, *start_lon*, *jeepney_routes_db*

`// start_lat and start_lon could also be: end_lat and end_lon`

Output: List of nearby jeepneys with their distances

Function `HaversineDistance`$(lat1, lon1, lat2, lon2)$**:**
  Convert $lat1, lon1, lat2, lon2$ from degrees to radians;
  $dlat \leftarrow lat2 - lat1$;
  $dlon \leftarrow lon2 - lon1$;
  $a \leftarrow \sin^2(\frac{dlat}{2}) + \cos(lat1) \times \cos(lat2) \times \sin^2(\frac{dlon}{2})$;
  $c \leftarrow 2 \times \text{atan2}(\sqrt{a}, \sqrt{1-a})$;
  $radius\_of\_earth \leftarrow 6371000$ `// in meters`
  return $radius\_of\_earth \times c$;

Function `PRMS` *(start_lat, start_lon, jeepney_routes_db)***:**
  $min\_distance \leftarrow \infty$;
  $nearest\_coordinate \leftarrow$ null;
  $jeepneys\_at\_min \leftarrow []$;
  foreach $jeepney \in jeepney\_routes\_db$ **do**
    foreach $(lat, lon) \in jeepney.route$ **do**
      $distance \leftarrow$ `HaversineDistance`$(start\_lat, start\_lon, lat, lon)$;
      if $distance < min\_distance$ **then**
        $min\_distance \leftarrow distance$;
        $nearest\_coordinate \leftarrow (lat, lon)$;
        $jeepneys\_at\_min \leftarrow []$;
      if $distance = min\_distance$ **then**
        $jeepneys\_at\_min.append((jeepney.id, distance))$;
  return $(nearest\_coordinate, jeepneys\_at\_min)$;

Algorithm 9 shows the code for the Proximity-based Route Matching Search. PRMS is an adaptation of K-Nearest Neighbors and they are similar in many ways like: (1) Both algorithms perform a single pass through the dataset to find the nearest points, with PRMS finding the single nearest coordinate and tracking which jeepneys have coordinates at that minimum distance. (2) The goal of both is to find the closest points to a query point, in the case of PRMS, to find the nearest jeepney route coordinate from the start point or end point. (3) Like how KNN maintains a list of k nearest neighbors, PRMS maintains a list of jeepneys that have coordinates at the minimum

distance found, ensuring we capture all jeepneys that pass through the nearest point to the start location. This approach eliminates the need for iterative radius expansion, making it more efficient than the Iterative Radius Search algorithm.

4.1.6 | Comparison between IRS and PRMS

While both the Iterative Radius Search (IRS) and Proximity-based Route Matching Search (PRMS) algorithms aim to find optimal jeepney routes between two points, they employ fundamentally different approaches that lead to significant differences in performance. The IRS algorithm, inspired by Iterative Deepening Search, expands its search radius iteratively to find jeepneys near start and end points, collecting indexes where these jeepneys pass and connecting them to form routes. Its runtime scales with walking distance because a larger radius means more points along the jeepney routes are considered "near" the start/end points, resulting in more indexes to process. In contrast, PRMS, adapted from K-Nearest Neighbors, performs a single-pass proximity search with dynamic distance thresholding, eliminating the need for iterative radius expansion and maintaining constant time complexity regardless of walking distance. This fundamental difference makes PRMS particularly more efficient in scenarios with longer walking distances or denser transportation networks, where the IRS algorithm's performance would degrade due to its iterative nature and the increasing number of points to process.

4.1.7 | Time Complexity for Best, Average, and Worst cases

4.1.7.1 | Time Complexity of IRS

- Best Case: **O(n x c)** where:
  n = total number of jeepneys in the database
  c = average number of coordinates per jeepney route
  Occurs when jeepney routes are found within initial search radius

- Average Case: **O(n x c x k)** where:
  n = total number of jeepneys in the database
  c = average number of coordinates per jeepney route
  k = **(d-r)/i + 1**, number of radius expansions needed
  d = distance from user to nearest jeepney route

r = initial search radius
i = increment size for radius expansion

- Worst Case: **O(n x c x ((m-r)/i + 1))** where:
  n = total number of jeepneys in the database
  c = average number of coordinates per jeepney route
  m = maximum search radius
  r = initial search radius
  i = increment size for radius expansion

4.1.7.2 | Time Complexity of PRMS

- Best Case: **O(n x c)** where:
  n = total number of jeepneys in the database
  c = average number of coordinates per jeepney route
- Average Case: **O(n x c)**
- Worst Case: **O(n x c)**

The Best, Average, and Worst case all have the same time complexity because it performs a single pass through all the coordinates unlike the expanding radius.

4.1.8 | Actual runtime comparison

As of the time the screenshots were taken, the Jeepneys that are available in the database so far are the following: St Mary, Dalipuga, Palao, Del Carmen, Buruun, and Tubod. There are two for each of the three bounds: (1) Northbound, (2) Eastbound, (3) Westbound.

There are two runtime values for each example, the first is for the searching of nearby jeeps near the start location, and the second is for the searching of nearby jeeps near the end location. The searching of nearby jeeps starts as soon as a start or end marker is placed.

4.1.8.1 | Single Ride Examples

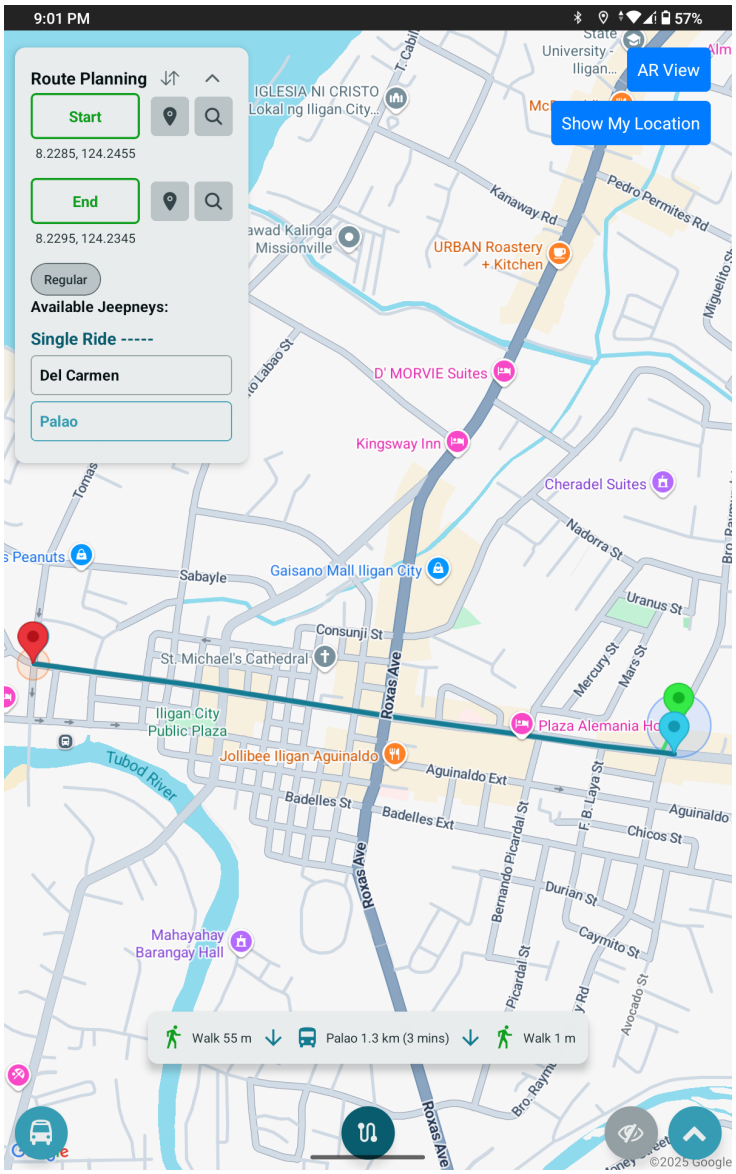

Figure 4.1: Single Ride 1

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	10.97	5.09	55
End Location Runtime (ms)	8.56	4.95	1

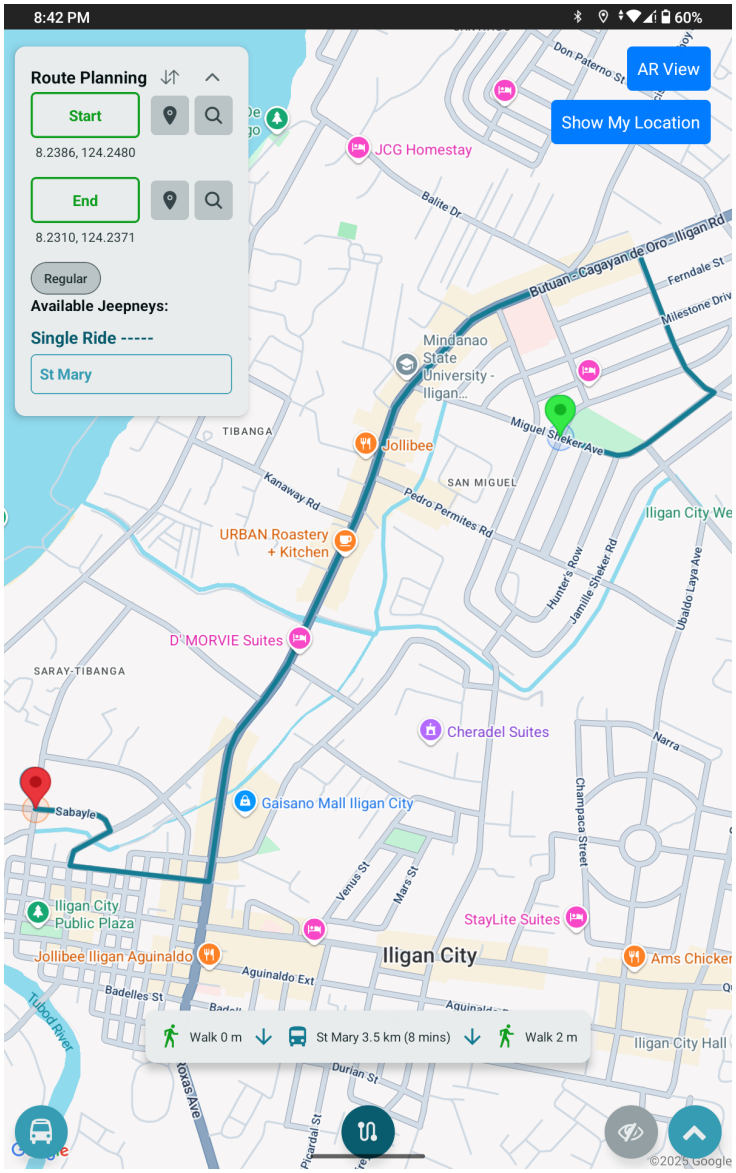

Figure 4.2: Single Ride 2

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	5.05	5.17	0
End Location Runtime (ms)	7.18	6.67	2

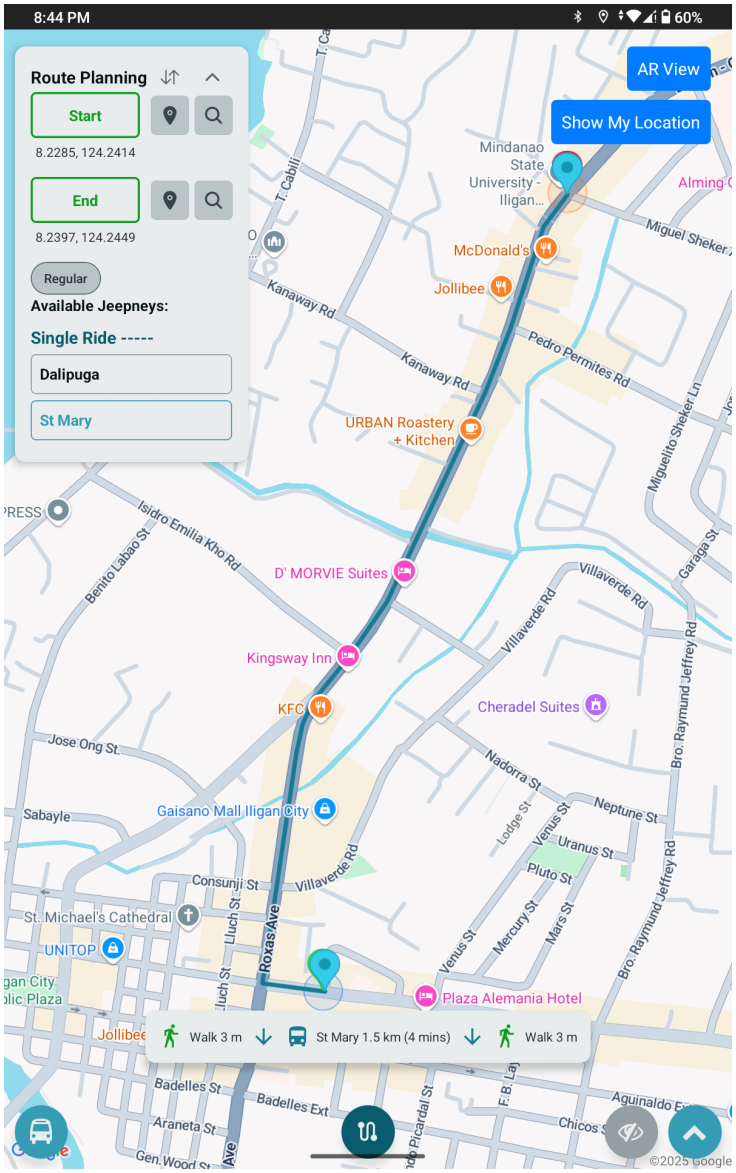

Figure 4.3: Single Ride 3

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	7.29	9.43	3
End Location Runtime (ms)	11.93	5.21	3

Figure 4.4: Single Ride 4

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	100.95	4.89	485
End Location Runtime (ms)	8.63	9.07	1

Figure 4.5: Single Ride 5

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	5.05	5.17	327
End Location Runtime (ms)	7.18	6.67	2

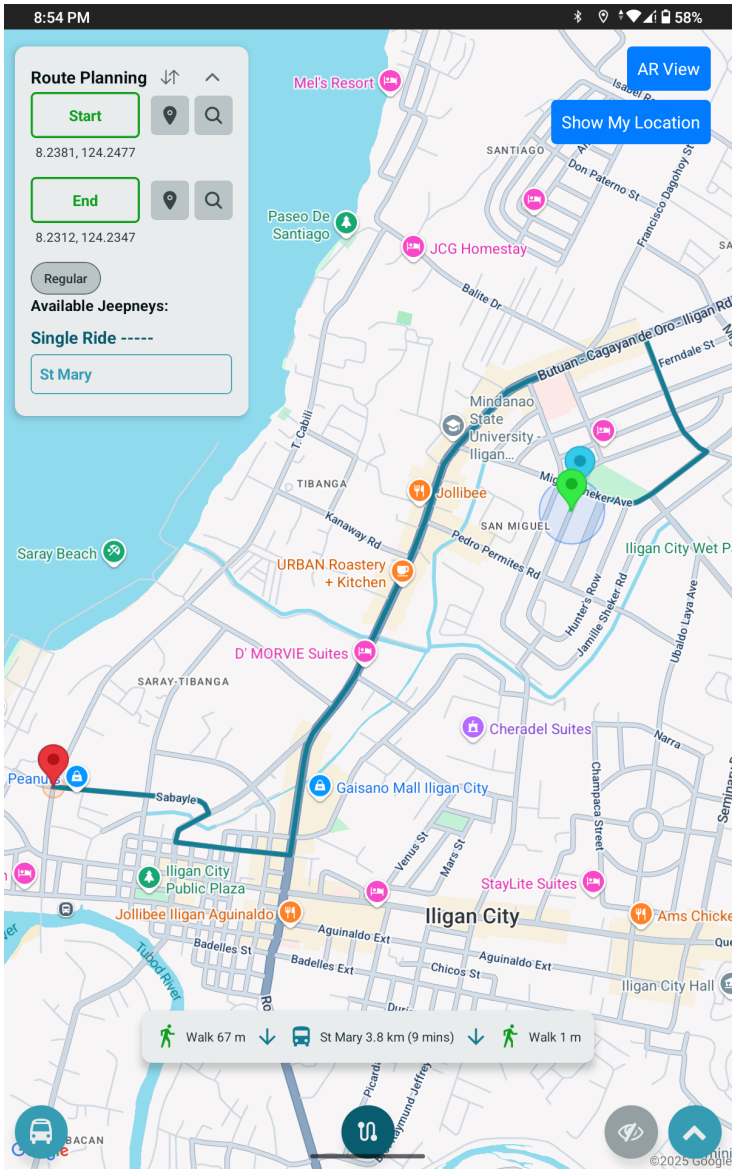

Figure 4.6: Single Ride 6

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	7.29	9.43	67
End Location Runtime (ms)	11.93	5.21	1

4.1.8.2 | Double Ride Examples

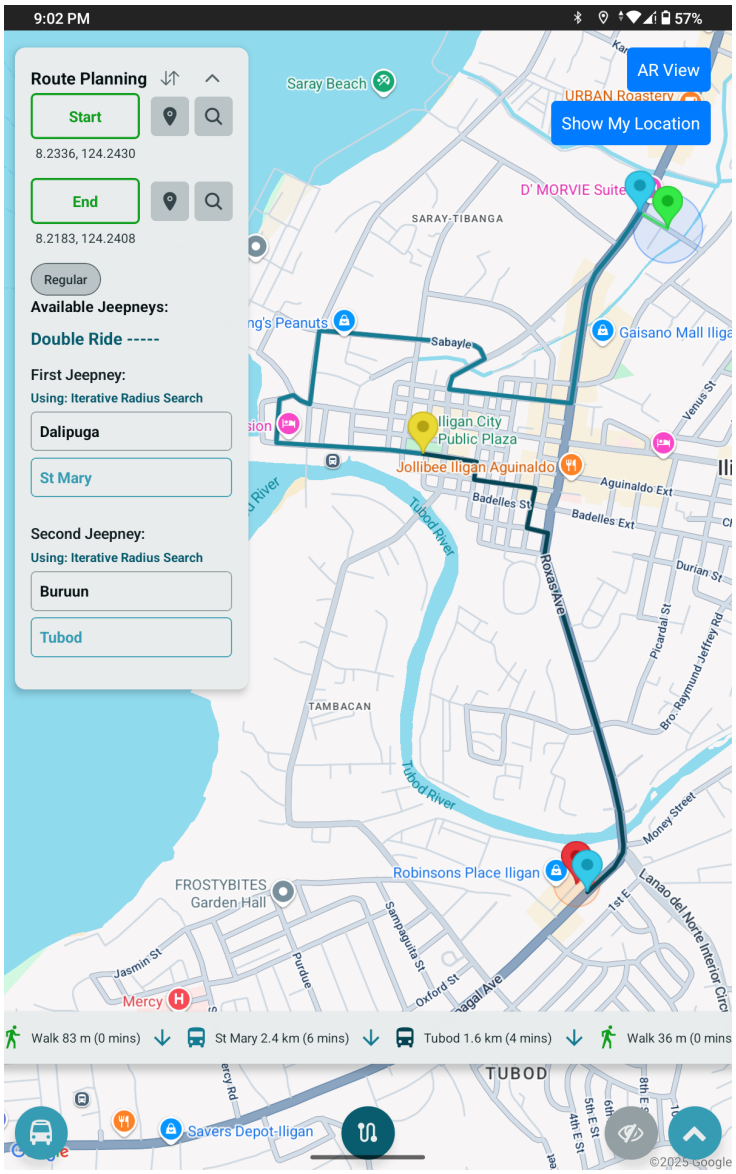

Figure 4.7: Double Ride 1

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	21.57	5.00	83
End Location Runtime (ms)	16.49	5.05	36

Figure 4.8: Double Ride 2

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	35.30	5.26	163
End Location Runtime (ms)	23.96	5.46	118

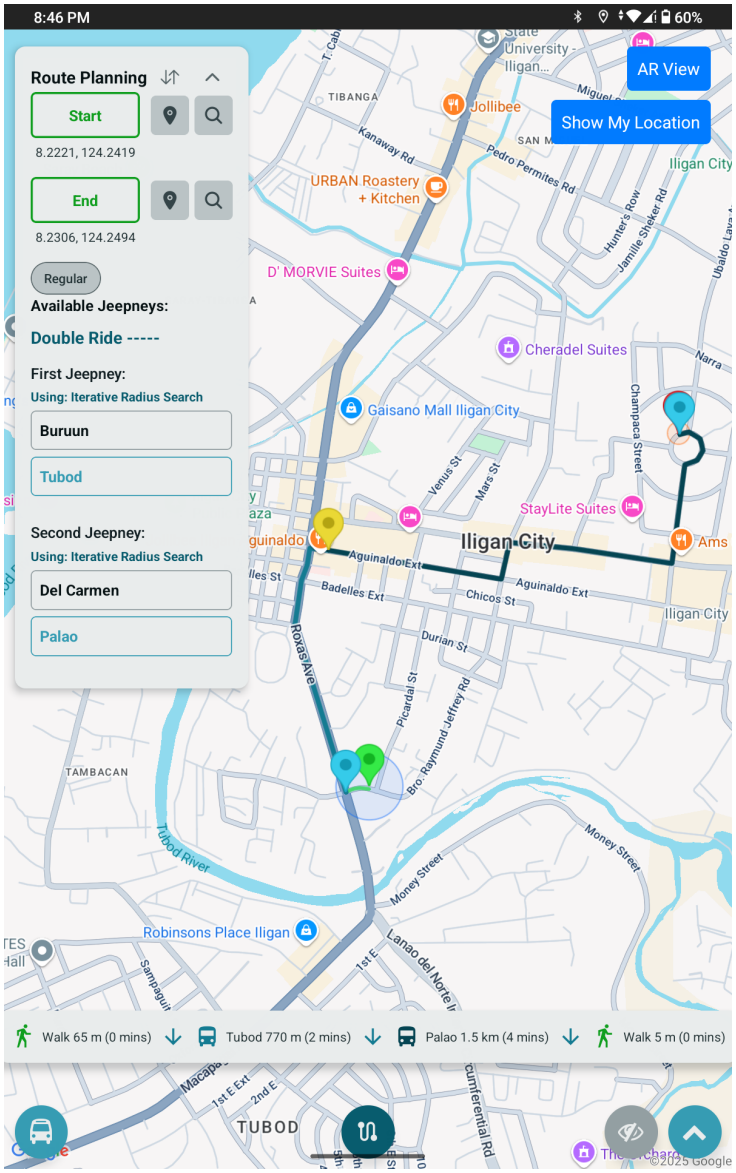

Figure 4.9: Double Ride 3

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	19.98	4.85	65
End Location Runtime (ms)	7.90	4.99	5

Figure 4.10: Double Ride 4

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	19.20	5.42	70
End Location Runtime (ms)	15.98	5.62	30

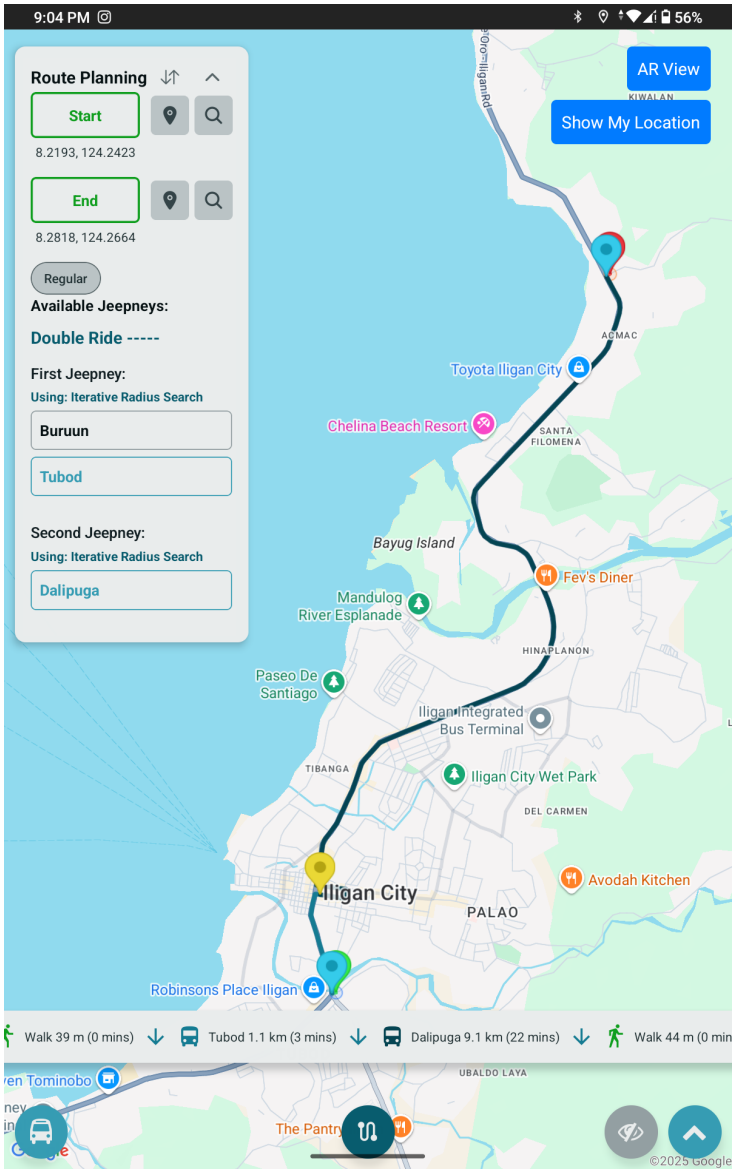

Figure 4.11: Double Ride 5

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	15.59	4.91	39
End Location Runtime (ms)	15.82	5.62	44

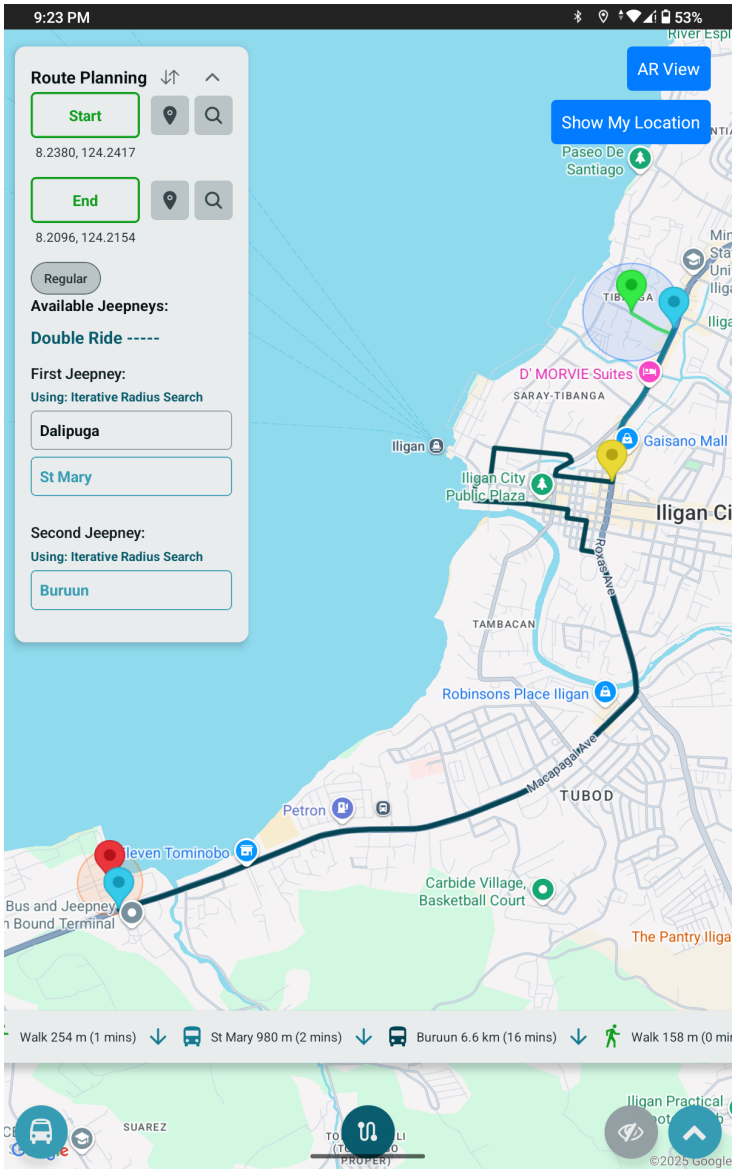

Figure 4.12: Double Ride 6

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	51.79	9.25	254
End Location Runtime (ms)	40.96	5.08	158

4.2 | Artificial Intelligence (AR) Implementation

Augmented reality (AR) was integrated into Jeepili as a core feature to enhance how users interact with the city's jeepney network. By blending virtual elements into the real-world view through the device's camera, commuters can intuitively locate walking paths, jeepney routes, and stopping points without relying on abstract map interfaces. This immersive presentation bridges the gap between spatial data and user navigation, helping commuters make decisions more quickly and confidently, especially in unfamiliar areas.

4.2.1 | AR Rendering Approach

ReactVision's ReactViro was used to implement AR it is a library for developers to build augmented reality (AR) and virtual reality (VR) experiences.

- We obtain the user's latitude, longitude and heading via `react-native-get-location` and the device compass.
- Routes and path coordinates are retrieved from Firestore and mapped to AR positions.
- The Jeepney Paths are represented by `ViroSphere`, with their world transforms recalculated in real time on each location/orientation update.
- The Markers are also represented by `ViroSphere` with a text on top represented by `ViroText` to indicate what is the purpose of this area.
- All AR content is managed inside `ViroARSceneNavigator`, which handles the live camera feed, lighting, and scene.

4.2.2 | Visual Elements and User Flow

To render helpful overlays for navigation, Jeepili uses the device's GPS to determine the user's location. This location data is used to calculate proximity to various points such as:

- The walking path leading to the nearest jeepney hailing spot.

- The stop point or transfer area for riding a second jeepney (in case of double rides).
- The stop point for disembarking and hiking toward either the destination or another route.
- The final destination marker.

The distance between the user and each marker determines the size of the marker; those closer to the user appear larger, while those farther away appear smaller. This size-scaling approach helps maintain spatial awareness and depth perception.

Since the device's orientation is constantly monitored, the system also knows which direction the user is facing. Markers are only rendered when they are within the user's forward-facing field of view. This prevents information overload and ensures only relevant visual guidance appears in the AR view.

4.3 | Testing and Validation

4.3.1 | Objective Testing

The guide questions for the objective testing will be:

1. Where it will point the user to wait for the jeepney/s.
   (Is it the closest possible waiting point for the jeepneys?).
2. The list of jeepney routes is displayed or suggested.
   (Are these the correct jeepneys that would pass through or nearby?).
3. Where it will point the user to stop.
   (Is this the closest possible stop for the user's destination?)
4. The travel information provided.
   (Are the fare calculations and travel time correct or accurate? Is the difference between the estimation and actual acceptable?).

4.3.2 | Subjective Testing

For the familiar residents of the city, the application's suggestion will be compared to how they would usually navigate. These people likely already practice the optimal methods of navigating the city by jeepneys, like where they will usually wait, and where they will make their stop. These feedbacks will be used to compare how the algorithm would suggest these information and to further improve the accuracy that it provides.

4.4 | Administrator Dashboard

The administrator Dashboard provides a centralized platform for route and traffic management. It enables admins to dynamically update routes, and add or remove blockages to reflect real-world conditions in real time.

4.4.1 | Admin Features

Admins can mark road segments as blocked or impassable, and create an alternate road as the detour. The system reroutes all affected jeepney routes and updates the ETAs and paths for users. The dashboard enables real-time monitoring, allowing administrators to view the current state of the jeepney network, including jeepney locations, and traffic conditions. The admin can also add a new jeepney route or remove an existing one.

4.4.2 | Implementation

There are three aspects for the dashboard. The interface features interactive maps for visualizing routes, blockages, and updates. Data synchronization ensures that changes made in the dashboard are reflected in the backend database and simulation in real time.

5

The Jeepili System

5.1 | System Overview

Jeepili is a mobile application designed to address the inefficiencies of jeepney commuting in Iligan City. At its core, the system enables commuters to input their origin and destination, and in return, recommends the most optimal jeepney route using proximity-based algorithms. Augmented Reality (AR) enhances this system by allowing commuters to visualize jeepney paths, hailing spots, walking segments, and stops directly within their surroundings using their device camera.

The application integrates several technologies including Google Maps, OSRM for walking directions, Firebase for cloud storage, and ViroReact for AR rendering. Through a modular architecture, the system features dynamic route computation, real-time path suggestions, fare and distance estimation, and immersive AR overlays. This layered design ensures a user-friendly experience while maintaining computational efficiency, particularly on mobile devices.

The system also supports an administrator dashboard that enables real-time editing of jeepney routes and simulation of road conditions such as reroutes during events. This capability ensures that the system remains responsive and adaptive to the city's evolving transportation patterns.

5.2 | System Objectives

- The system shall allow users to input both a starting point and destination using text or pin-drop on a map.
- The system shall search and recommend jeepney routes that minimize total travel time, transfer count, and walking distance.
- The system shall support single rides, double rides (one transfer), and alternate ride options when optimal routes are unavailable.
- The system shall calculate jeepney fare estimates based on the LTFRB's standard fare matrix.
- The system shall display walking directions to and from hailing and drop-off points using OSRM API.
- The system shall use AR to overlay route markers, hailing areas, stop points, and walking paths on the user's real-world environment.
- The system shall work on Android devices running version 8.0 or higher, and support AR features on ARCore-compatible devices.
- The system shall provide an admin dashboard for creating, editing, and deleting jeepney routes and simulating reroutes based on real-world events.
- The system shall store and retrieve route and fare data using Firebase Firestore.
- The system shall dynamically adapt to admin-made changes and update the interface in real-time.

These function requirements ensure that Jeepili operates reliably and efficiently in providing smart, location-aware, real-time data, transportation support to jeepney commuters in Iligan City.

5.3 | System Scope and Limitations

- **Location Input:** Users must select locations within Iligan City. Inputs outside this boundary are invalidated with an error message. **Assumption:** Users enable GPS and provide valid locations.

- **Route Recommendations:** Recommendations depend on available jeepney route data. Incomplete route data results in limited or no recommendations, with a clear notification shown. **Assumption:** The database is regularly maintained and updated.
- **Ride Options:** Single, double, and alternate rides depend on existing jeepney routes. If no viable routes exist, only default recommendations are given. **Assumption:** Adequate route overlap exists in the database.
- **Fare Calculation:** Fare estimates strictly adhere to LTFRB standard rates. Non-standard fare inputs are not accepted or computed. **Assumption:** LTFRB rates provided in the system are current and accurate.
- **Walking Directions:** Directions rely on OSRM API accuracy. Errors or data loss from OSRM result in a notification and the inability to display directions. **Assumption:** OSRM API service is available and accurate.
- **AR Features:** Only ARCore-compatible devices can access AR features. Devices not meeting requirements receive notifications and default to standard map mode. **Assumption:** Users have ARCore-compatible devices.
- **Device Compatibility:** The app supports Android 8.0 or higher. Attempts to install or run the app on older versions are prevented, and clear compatibility notifications are shown. **Assumption:** Users maintain compatible and updated devices.
- **Admin Dashboard:** Editing and event simulation require stable internet connectivity. Lack of connectivity triggers error messages and prevents updates. **Assumption:** Admin has reliable internet access.
- **Data Storage:** The system's functionality relies on accurate and complete data stored in Firebase. Incorrect configurations trigger notifications and limit functionality. **Assumption:** Firebase database is accurately configured and maintained.
- **Real-time Updates:** Updates occur only when devices are connected to the internet. Disconnected devices receive notifications about failed updates and will refresh content once reconnected. **Assumption:** Users maintain an active internet connection while using the app.

These constraints and assumptions define the boundary of Jeepili's functionality. If assumptions are not met (e.g., no internet, incompatible device), system performance will degrade gracefully without causing app failure, and appropriate error messages or fallback options will be shown.

5.4 | Architectural Design

The Jeepili system follows a modular design consisting of interconnected components that enable route computation, data storage, visualization, and system control. The architecture was initially conceptualized with Flutter for the frontend and Flask for the backend. However, during development, it was transitioned to React Native (Expo) with integrated backend logic. This shift allowed the team to streamline development using a single language and take advantage of Expo's built-in support for camera, location, and platform APIs essential for AR and mapping features.

The overall architecture is illustrated in Figure 4.1.

5.4.1 | System Architecture

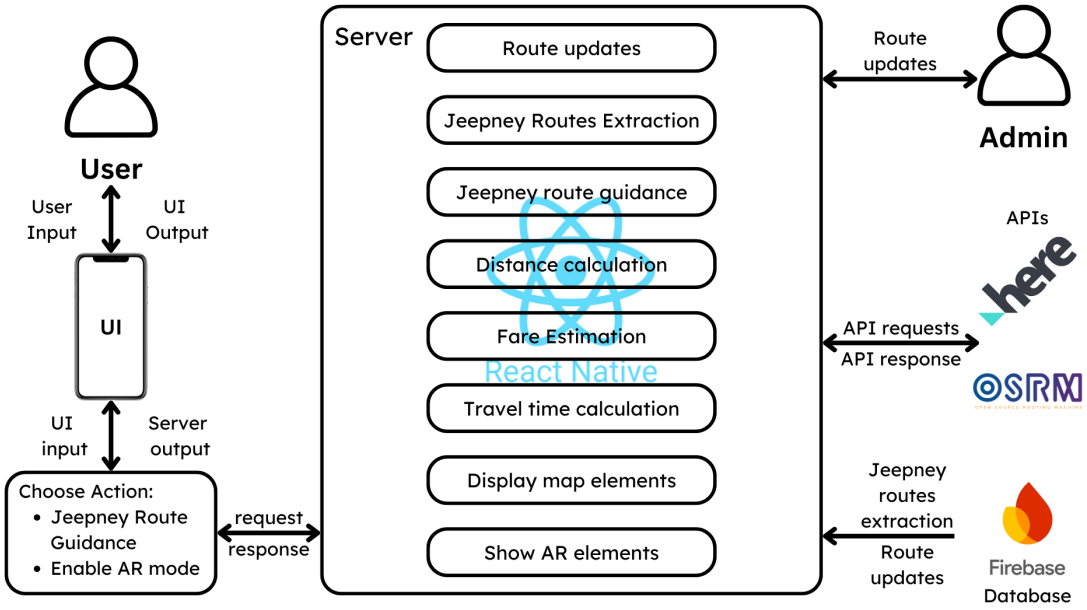

Figure 5.1: System Architecture

The system flow begins with the user interface (UI), where users can select their starting and destination points, choose actions (e.g., jeepney route guidance, AR mode), and view outputs. These inputs are processed through the internal logic embedded in the application, which handles route guidance, fare estimation, and time calculation.

The system makes API calls to HERE Maps and OSRM for geocoding and walking path generation, and interacts with Firebase Firestore to extract jeepney routes and fare structures. Server logic then evaluates the data and sends the appropriate responses back to the UI.

Administrators can dynamically modify jeepney routes or create traffic events through the admin dashboard. These changes are immediately pushed to the Firebase database and reflected across all user-facing components upon sync.

The major components are summarized below:

- **User Interface (Frontend):** Built using React Native, the UI allows users to interact with the system via map selection, toggle features, and view route and AR data.
- **Backend Logic:** Responsible for route matching, fare calculations, time estimations, and path finding using proximity-based search algorithms.
- **External APIs:** Includes HERE API (location search), OSRM (walking paths), and ViroReact (AR rendering).
- **Cloud Database:** Firebase Firestore stores route data, fare matrices, and admin-defined events.
- **Admin Dashboard:** Provides tools for modifying jeepney routes, simulating road blocks, and broadcasting updates.

This layered architecture enables real-time performance, modular feature updates, and compatibility across a wide range of Android devices.

5.5 | System Functions

5.5.1 | Single Ride - Default Route

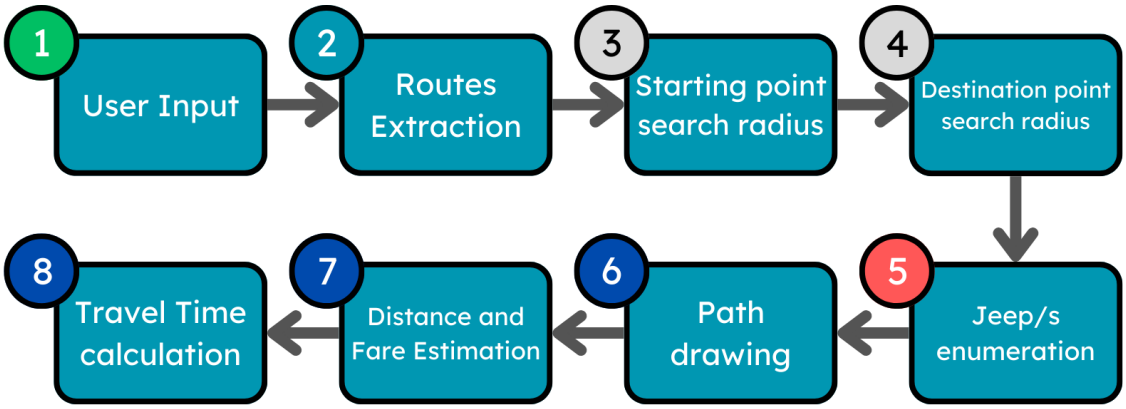

Figure 5.2: Single Ride Process Flow with Default Route

1. **User Input** - The user will input two locations, the start and destination point, these inputs will be either by dropping a pin or typing the address. Both of these locations will be stored as a coordinate variable (latitude, longitude).

2. **Routes Extraction** - The jeepney routes will be stored in a database, each jeepney route (e.g. Tambo) will have an array and the index will hold a coordinate variable. When the entire array is plotted on the map, it will draw the whole route of that jeepney route.

3. **Starting point search radius** - At the starting point, initially, there will be a defined circle with a radius (for example 10m). Now recall that we extracted all the jeepney routes from the database. Now in this search radius, it will look for coordinates that belong to jeeps, if it doesn't find any, it will increase by (for example 10m) and continue to do so until it finds at least one coordinate that belongs to a jeepney route. It will hold the list of jeepney routes found for Step 5.

4. **Destination point search radius** - Similar to Step 3, it will have an initial circular search radius that will look for jeepney route coordinates inside it. Now, it doesn't just look for any coordinates, it looks for coordinates that belonged to jeepney routes that were detected in Step 3. The search radius will continue to expand or increase until it finds a coordinate that satisfies the requirement.

5. **Jeep/s enumeration** - The Jeepney routes that were detected by both Step 3 and Step 4 will be displayed in the list. This means that the user can ride any of these jeepneys and they can stop or arrive near their destination point.

6. **Path drawing** - By default, the jeepney path shown will be the first one, which is usually the best choice, but the user can click a different jeep in case they choose to hail a different jeepney route.

   Recall, that in Step 3, it was looking for coordinate points that are near the Start location. There will be a walking path shown between the starting point and the nearest point to it where the jeepney route will pass through. This will indicate that the user must walk towards this place to hail the jeeps.

   Recall, that in Step 4, it was looking for coordinate points that are near the destination location. There will be a walking path shown between the jeep's closest coordinate to the destination point and the destination point. This will indicate that the user must stop or "para" near this point to walk the shortest path towards its destination.

7. **Distance and Fare Estimation** - The distance of the jeepney path will be calculated with the help of the Open Source Routing Machine (OSRM) API. The fare will be calculated based on the distance and whether the user belongs to the regular or discounted (PWDs, Students, Senior Citizens).

8. **Travel Time Estimation** - By default, the travel time will be calculated with the help of Google Maps API for the availability of the traffic heatmap, if this data however is unavailable or doesn't work for the time being, the alternate calculation will be the assumption that the jeepneys are constantly traveling not faster than the highway speed limit.

5.5.2 | Double Ride - Default Route

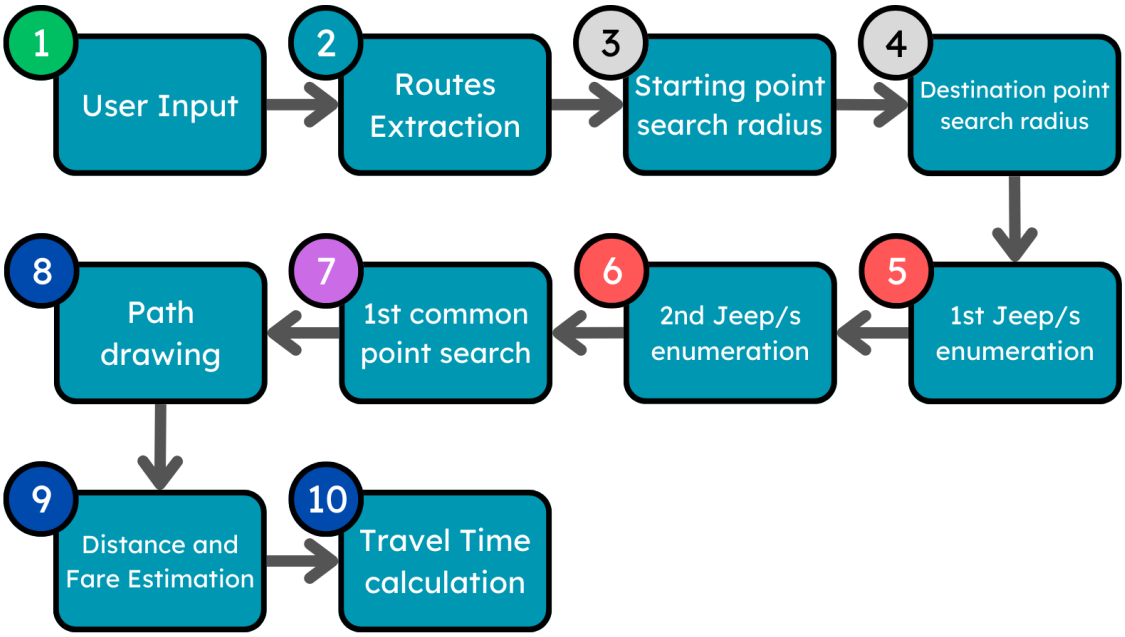

Figure 5.3: Double Ride Process Flow with Default Route

1. **User Input** - The user will input two locations, the start and destination point, these inputs will be either by dropping a pin or typing the address. Both of these locations will be stored as a coordinate variable (latitude, longitude).

2. **Routes Extraction** - The jeepney routes will be stored in a database, each jeepney route (e.g. Tambo) will have an array and the index will hold a coordinate variable. When the entire array is plotted on the map, it will draw the whole route of that jeepney route.

3. **Starting point search radius** - At the starting point, initially, there will be a defined circle with a radius (for example 10m). Now recall that we extracted all the jeepney routes from the database. Now in this search radius, it will look for coordinates that belong to jeeps, if it doesn't find any, it will increase by (for example 10m) and continue to do so until it finds at least one coordinate that belongs to a jeepney route. It will hold the list of jeepney routes found for Step 5.

4. **Destination point search radius** - Similar to Step 3, it will have an initial circular search radius that will look for jeepney route coordinates inside it. Now, it doesn't just look for any coordinates; it looks for coordinates that belonged to jeepney

routes that were detected in Step 3. The search radius will continue to expand or increase until it finds a coordinate that satisfies the requirement.

5. **1st Jeep/s enumeration** - The Jeepney routes that were detected by Step 3 will be displayed in the list. This means that the user can ride any of these jeepneys and they can stop or "para" near the second starting point to ride the second jeepney.

6. **2nd Jeep/s enumeration** - The Jeepney routes that were detected by Step 4 will be displayed in the list. This means that when the user stops or "para" near the second starting point they can ride any of the jeepneys in this list and they will arrive near their destination point.

7. **1st common point search** - Recall that Step 5 and Step 6 kept mentioning “second starting point”, this refers to the area where both the jeeps from the first list and second list will pass through or pass by. The common points of any of the jeeps from the first list and second list will be considered as the common area. This still depends on what the user chose as their first jeep and second jeep as some jeep routes may take a different route earlier than the rest.

8. **Path drawing** - By default, the jeepney path shown will be the first one, which is usually the best choice, but the user can click a different jeep in case they choose to hail a different jeepney route.

   Recall, that in Step 3, it was looking for coordinate points that are near the Start location. There will be a walking path shown between the starting point and the nearest point to it where the jeepney route will pass through. This will indicate that the user must walk towards this place to hail the jeeps.

   Recall, that in Step 4, it was looking for coordinate points that are near the destination location. There will be a walking path shown between the jeep’s closest coordinate to the destination point and the destination point. This will indicate that the user must stop or "para" near this point to walk the shortest path towards its destination.

   Recall that since this is for a double ride, there will be an additional point or path highlighted (for example: “stop here, for the 2nd ride”). This will indicate the user to stop near this area to ride their second jeep.

9. **Distance and Fare Estimation** - The distance of the jeepney path will be calculated with the help of the Open Source Routing Machine (OSRM) API. The fare will be calculated based on the distance and whether the user belongs to the regular or discounted (PWDs, Students, Senior Citizens).

10. **2nd Jeep/s enumeration** - By default, the travel time will be calculated with the help of Google Maps API for the availability of the traffic heatmap, if this data however is unavailable or doesn't work for the time being, the alternate calculation will be the assumption that the jeepneys are constantly traveling not faster than the highway speed limit.

5.5.3 | Alternate Route

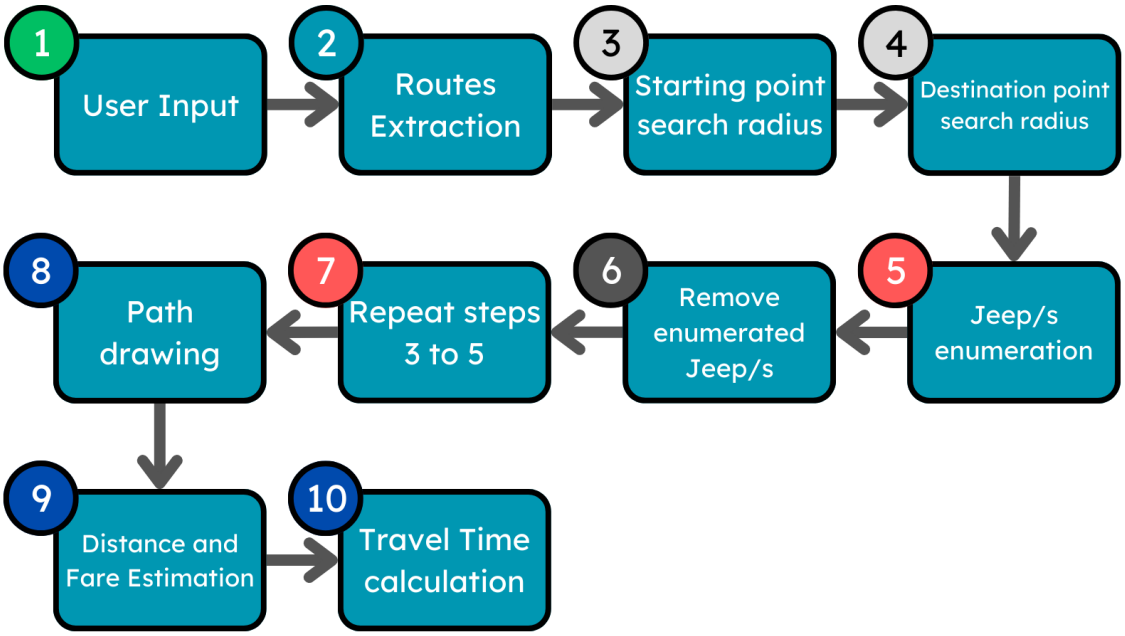

Figure 5.4: Process Flow of Alternate Route

1. **User Input** - The user will input two locations, the start and destination point, these inputs will be either by dropping a pin or typing the address. Both of these locations will be stored as a coordinate variable (latitude, longitude).

2. **Routes Extraction** - The jeepney routes will be stored in a database, each jeepney route (e.g. Tambo) will have an array and the index will hold a coordinate variable. When the entire array is plotted on the map, it will draw the whole route of that jeepney route.

3. **Starting point search radius** - At the starting point, initially, there will be a defined circle with a radius (for example 10m). Now recall that we extracted all the jeepney routes from the database. Now in this search radius, it will look for coordinates

that belong to jeeps, if it doesn't find any, it will increase by (for example 10m) and continue to do so until it finds at least one coordinate that belongs to a jeepney route. It will hold the list of jeepney routes found for Step 5.

4. **Destination point search radius** - Similar to Step 3, it will have an initial circular search radius that will look for jeepney route coordinates inside it. Now, it doesn't just look for any coordinates; it looks for coordinates that belonged to jeepney routes that were detected in Step 3. The search radius will continue to expand or increase until it finds a coordinate that satisfies the requirement.

5. **Jeep/s enumeration** - The Jeepney routes that were detected by both Step 3 and Step 4 will be displayed in the list. This means that the user can ride any of these jeepneys and they can stop or arrive near their destination point.

6. **Remove Enumerated Jeeps** - Recall that the jeeps listed in Step 5 are the best options because they will get you nearest to your destination, with minimal walking needed. Now Recall that in Step 2, all the jeepneys routes are extracted from the database. In this step, all the jeepneys enumerated by step 5 will be removed, as if they didn't exist. This supports the situation where the users needs to be close to the destination point as fast as possible for whatever reason they have. However, the stops of these alternate routes will make them take a longer walk compared to choosing the default option as the stops of these will be far.

7. **Repeat steps 3 to 5** - It will again search for jeeps near the start and destinations, and enumerate them in the list. The jeeps listed here will be the alternate options.

8. **Path drawing** - By default, the jeepney path shown will be the first one, which is usually the best choice, but the user can click a different jeep in case they choose to hail a different jeepney route.

   Recall, that in Step 3, it was looking for coordinate points that are near the Start location. There will be a walking path shown between the starting point and the nearest point to it where the jeepney route will pass through. This will indicate that the user must walk towards this place to hail the jeeps.

   Recall, that in Step 4, it was looking for coordinate points that are near the destination location. There will be a walking path shown between the jeep's closest coordinate to the destination point and the destination point. This will indicate that the user must stop or "para" near this point to walk the shortest path towards its destination.

9. **Distance and Fare Estimation** - The distance of the jeepney path will be calculated with the help of the Open Source Routing Machine (OSRM) API. The fare will be calculated based on the distance and whether the user belongs to the regular or discounted (PWDs, Students, Senior Citizens).

10. **Travel Time Estimation** - By default, the travel time will be calculated with the help of Google Maps API for the availability of the traffic heatmap, if this data however is unavailable or doesn't work for the time being, the alternate calculation will be the assumption that the jeepneys are constantly traveling not faster than the highway speed limit.

5.5.4 | Walking Path Guidance

Walking directions are computed to guide users between their actual position and the nearest jeepney path, both before boarding and after alighting. The system uses OSRM (Open Source Routing Machine) to generate walkable paths based on real-world road networks. These paths are displayed both on the map and through AR visualization.

The application ensures that walking paths do not traverse inaccessible terrain and are kept minimal, prioritizing straight-line proximity and actual road distance.

5.5.5 | Fare Calculation Logic

The fare estimation function uses the LTFRB fare matrix, calculating price based on:

- A base fare (first 4 km)
- An incremental rate per kilometer thereafter

Users are asked to specify whether they are regular or discounted commuters. The fare logic adjusts accordingly, using stored values for each fare tier. The computed fare is rounded off for display and includes both rides in double ride scenarios.

5.5.6 | AR Visualization

Jeepili includes an augmented reality mode that visualizes key commuting information in the user's environment. Using the device's compass, GPS, and camera, the system overlays the following elements in AR:

- Hailing areas
- Stop locations
- Walking paths
- Jeepney directions

AR markers dynamically adjust in size based on distance, and only appear when within the user's field of view. The AR system is implemented via ViroReact, and AR features are automatically disabled on unsupported devices.

5.5.7 | Admin Route Management

The administrator dashboard allows real-time updates to the system, including:

- Adding or modifying jeepney routes
- Plotting routes by selecting ordered snapzones
- Declaring alternate routes due to road closures or city events

Admins can also simulate traffic scenarios (e.g., night markets or construction). All changes are synced to Firebase Firestore and automatically reflected in the user interface. Error handling ensures that any admin-side issues (e.g., internet disconnection) are flagged without affecting user access.

These administrative tools ensure that the app remains accurate and adaptable to real-world changes in city transit.

5.6 | Physical Environment and Resources

To ensure smooth usage of Jeepili across a variety of devices, we identify two levels of system requirements depending on whether the user enables AR or not. The application is intended to be widely accessible, but certain features like Augmented Reality require more advanced hardware and software compatibility.

5.6.1 | Minimum Requirements for Standard Map Mode (No AR)

Devices running the basic map-based navigation (without AR) should meet the following specifications:

- **Android OS:** Version 8.0 (Oreo) and above
- **RAM:** 2GB or more
- **GPS/Location services:** Required
- **Internet Connection:**Required for real-time route, fare, and traffic data
- **Sensors:** Accelerometer and Compass (Recommended for better direction detection)

These requirements ensure that even lower-end smartphones can run the standard routing and mapping functionalities without AR. The core routing, fare estimation, and path guidance will work seamlessly under these constraints.

5.6.2 | Minimum Requirements for Augmented Reality Mode

For the Augmented Reality features of Jeepili to work properly, the user must have a device that meets the minimum ARCore support standards set by Google:

- **Android OS:** Version 10 and above
- **RAM:** At least 4GB
- **Camera:** Rear-facing camera with autofocus

- **Sensors:** Accelerometer, Gyroscope, and Compass (required for direction and stability)
- I**nternet Connection:** Required for real-time updates and route syncing
- **AR Compatibility:** Device must support ARCore (check official ARCore supported list)
- **Google Play Services for AR:** Must be installed and updated

Failure to meet any of these requirements may result in degraded performance or inability to launch AR view. While the application still works in map mode, AR markers and overlays will only appear on compatible devices.

These two tiers of requirements were designed to accommodate as many users as possible, while still enabling advanced features like immersive AR guidance for users with modern smartphones.

5.6.3 | Administrator Environment

The Admin Dashboard is designed for mobile access to ensure flexibility and ease of use in real-time conditions. Admins must use devices with the following specifications:

- **Android OS:** Version 8.0 or higher
- **RAM:** Minimum of 2GB
- **Browser:** Chrome, Firefox, or any modern mobile browser
- **Internet Connection:** Required for syncing edits, events, and route data to Firebase

All administrative features are optimized for mobile interfaces, allowing transport officers or administrators to make field updates directly from their smartphones.

5.6.4 | System Assumptions

- Users have location services enabled and grant the app GPS access.
- Users have stable mobile data or Wi-Fi while using the app.

- Admins perform updates using a secure and stable internet connection.

These requirements were designed to accommodate a wide user base while still supporting immersive and data-rich features for compatible devices.

6

Design and Implementation Issues

This chapter discusses the design and implementation of the major data structures and algorithms used in the software. It includes a discussion on the major issues and problems encountered, and the corresponding solutions and alternatives employed by the proponents. Parts of the design tools in the Technical Manual may be lifted as figures in this chapter.

6.1 | Development Challenges and Solutions

During development, the team encountered several challenges that led to modifications in the original plan and required adaptive solutions.

1. **Initial Tech Stack Limitations:** The initial stack using Flutter and Flask presented challenges in the compatibility with AR. Switching to React Native (Expo) allowed for improved efficiency in developing, and better integration with APIs and mobile features.

2. **AR Implementation Challenges:** The planned AR implementation using ARCore proved incompatible with React Native, as the required packages were either deprecated or incompatible with other modules. This limitation led to an innovative workaround using expo-camera, where we implemented a custom solution that displays the camera feed and overlays directional icons that dynamically adjust based on the device's location and orientation.

Despite these challenges, the system was successfully implemented with all major features functioning, including map rendering, route detection, and fare estimation. Each obstacle contributed to enhancing the overall structure, performance, and usability of the application.

6.2 | Route Matching Design Decisions

Originally, the system was built on an Iterative Radius Search (IRS) algorithm inspired by Iterative Deepening Search. However, this proved inefficient for real-time responsiveness and accurate filtering. It was replaced by the Proximity-based Route Matching Search (PRMS) algorithm.

PRMS improved runtime performance by limiting search space and directly evaluating route proximity to both start and end points. This shift significantly reduced unnecessary computation and enabled support for alternate ride suggestions.

6.2.1 | Iterative Radius Search

Iterative Radius search (IRS) is an adaptation of the Iterative Deepening search (IDS) algorithm. IDS iterates graphs by depth, similarly, IRS operates in a geospatial domain through expanding a radius in a 2D space. IRS searches for points of interest (coordinates) within a radius, then continues on expanding until the desired conditions are met.

Algorithm 7: Iterative Deepening Search

```
Input: start_node, goal
Output: Result of search or None
Function IterativeDeepeningSearch(start_node, goal)
    depth ← 0;
    while True do
        result ← DLS(start_node, goal, depth);
        if result ≠ None then
            return result;
        depth ← depth + 1;
Function DLS(node, goal, depth_limit)
    if depth_limit == 0 then
        if node == goal then
            return node;
        else
            return None;
    else if depth_limit > 0 then
        foreach child Input: expand
        node do
            result ← DLS(child, goal, depth_limit − 1);
            if result ≠ None then
                return result;
    return None;
```

Algorithm 7 shows the code for the Iterative Deepening Search. It is adapted by Iterative Radius Search to work for searching within maps.

6.2.2 | Nearest-Neighbor Search Algorithms

The route selection process involves searching for jeepney routes that are close to the user's start and destination locations. There are two different approaches considered for this process. The first is an initial adaptation of the Iterative Deepening Search (IDS) algorithm into a graph based variant called Iterative Radius Search (IRS), which was the initial design. The second is a simplified nearest-neighbor approach that was used for

the final implementation due to its efficiency.

6.2.3 | Proximity-based Route matching Search

Proximity-based Route Matching Search (PRMS) is an adaptation of the K-Nearest Neighbors (KNN) algorithm, specifically designed for geospatial route matching in public transportation systems. While KNN traditionally finds the k closest points in a feature space, PRMS operates in a geospatial domain to find the most efficient routes between two points using public transportation.

Algorithm 8: K-Nearest Neighbors Search

Input: *query_point*, *dataset*, *k*
Output: *k* nearest neighbors

```
1 Function KNN (query_point, dataset, k)
2     distances ← empty list;
3     foreach point Input: d
4     ataset do
5         distance ← calculateDistance (query_point, point);
6         distances.append((point, distance));
7     distances ← sort (distances by distance);
8     return first k points from distances
```

Algorithm 8 shows the code for the K-Nearest Neighbors Search. It is adapted by Proximity-based Route Matching Search (PRMS) to work for finding the nearest jeepney routes.

Algorithm 9: Proximity-based Route Matching Search

Input: *start_lat*, *start_lon*, *jeepney_routes_db*

`// start_lat and start_lon could also be: end_lat and end_lon`

Output: List of nearby jeepneys with their distances

Function `HaversineDistance`$(lat1, lon1, lat2, lon2)$**:**
    Convert $lat1, lon1, lat2, lon2$ from degrees to radians;
    $dlat \leftarrow lat2 - lat1$;
    $dlon \leftarrow lon2 - lon1$;
    $a \leftarrow \sin^2(\frac{dlat}{2}) + \cos(lat1) \times \cos(lat2) \times \sin^2(\frac{dlon}{2})$;
    $c \leftarrow 2 \times \text{atan2}(\sqrt{a}, \sqrt{1-a})$;
    $radius\_of\_earth \leftarrow 6371000$ `// in meters`
    return $radius\_of\_earth \times c$;
Function `PRMS` *(start_lat, start_lon, jeepney_routes_db)***:**
    $min\_distance \leftarrow \infty$;
    $nearest\_coordinate \leftarrow$ null;
    $jeepneys\_at\_min \leftarrow []$;
    foreach $jeepney \in jeepney\_routes\_db$ **do**
        foreach $(lat, lon) \in jeepney.route$ **do**
            $distance \leftarrow$ `HaversineDistance`$(start\_lat, start\_lon, lat, lon)$;
            if $distance < min\_distance$ **then**
                $min\_distance \leftarrow distance$;
                $nearest\_coordinate \leftarrow (lat, lon)$;
                $jeepneys\_at\_min \leftarrow []$;
            if $distance = min\_distance$ **then**
                $jeepneys\_at\_min.append((jeepney.id, distance))$;
    return $(nearest\_coordinate, jeepneys\_at\_min)$;

Algorithm 9 shows the code for the Proximity-based Route Matching Search. PRMS is an adaptation of K-Nearest Neighbors and they are similar in many ways like: (1) Both algorithms perform a single pass through the dataset to find the nearest points, with PRMS finding the single nearest coordinate and tracking which jeepneys have coordinates at that minimum distance. (2) The goal of both is to find the closest points to a query point, in the case of PRMS, to find the nearest jeepney route coordinate from the start point or end point. (3) Like how KNN maintains a list of k nearest neighbors, PRMS maintains a list of jeepneys that have coordinates at the minimum

distance found, ensuring we capture all jeepneys that pass through the nearest point to the start location. This approach eliminates the need for iterative radius expansion, making it more efficient than the Iterative Radius Search algorithm.

6.2.4 | Comparison between IRS and PRMS

While both the Iterative Radius Search (IRS) and Proximity-based Route Matching Search (PRMS) algorithms aim to find optimal jeepney routes between two points, they employ fundamentally different approaches that lead to significant differences in performance. The IRS algorithm, inspired by Iterative Deepening Search, expands its search radius iteratively to find jeepneys near start and end points, collecting indexes where these jeepneys pass and connecting them to form routes. Its runtime scales with walking distance because a larger radius means more points along the jeepney routes are considered "near" the start/end points, resulting in more indexes to process. In contrast, PRMS, adapted from K-Nearest Neighbors, performs a single-pass proximity search with dynamic distance thresholding, eliminating the need for iterative radius expansion and maintaining constant time complexity regardless of walking distance. This fundamental difference makes PRMS particularly more efficient in scenarios with longer walking distances or denser transportation networks, where the IRS algorithm's performance would degrade due to its iterative nature and the increasing number of points to process.

6.2.5 | Time Complexity for Best, Average, and Worst cases

6.2.5.1 | Time Complexity of IRS

- Best Case: **O(n x c)** where:
  n = total number of jeepneys in the database
  c = average number of coordinates per jeepney route
  Occurs when jeepney routes are found within initial search radius
- Average Case: **O(n x c x k)** where:
  n = total number of jeepneys in the database
  c = average number of coordinates per jeepney route
  k = **(d-r)/i + 1**, number of radius expansions needed
  d = distance from user to nearest jeepney route

r = initial search radius
i = increment size for radius expansion

- Worst Case: **O(n x c x ((m-r)/i + 1))** where:
  n = total number of jeepneys in the database
  c = average number of coordinates per jeepney route
  m = maximum search radius
  r = initial search radius
  i = increment size for radius expansion

6.2.5.2 | Time Complexity of PRMS

- Best Case: **O(n x c)** where:
  n = total number of jeepneys in the database
  c = average number of coordinates per jeepney route
- Average Case: **O(n x c)**
- Worst Case: **O(n x c)**

The Best, Average, and Worst case all have the same time complexity because it performs a single pass through all the coordinates unlike the expanding radius.

6.2.6 | Actual runtime comparison

As of the time the screenshots were taken, the Jeepneys that are available in the database so far are the following: St Mary, Dalipuga, Palao, Del Carmen, Buruun, and Tubod. There are two for each of the three bounds: (1) Northbound, (2) Eastbound, (3) Westbound.

There are two runtime values for each example, the first is for the searching of nearby jeeps near the start location, and the second is for the searching of nearby jeeps near the end location. The searching of nearby jeeps starts as soon as a start or end marker is placed.

6.2.6.1 | Single Ride Examples

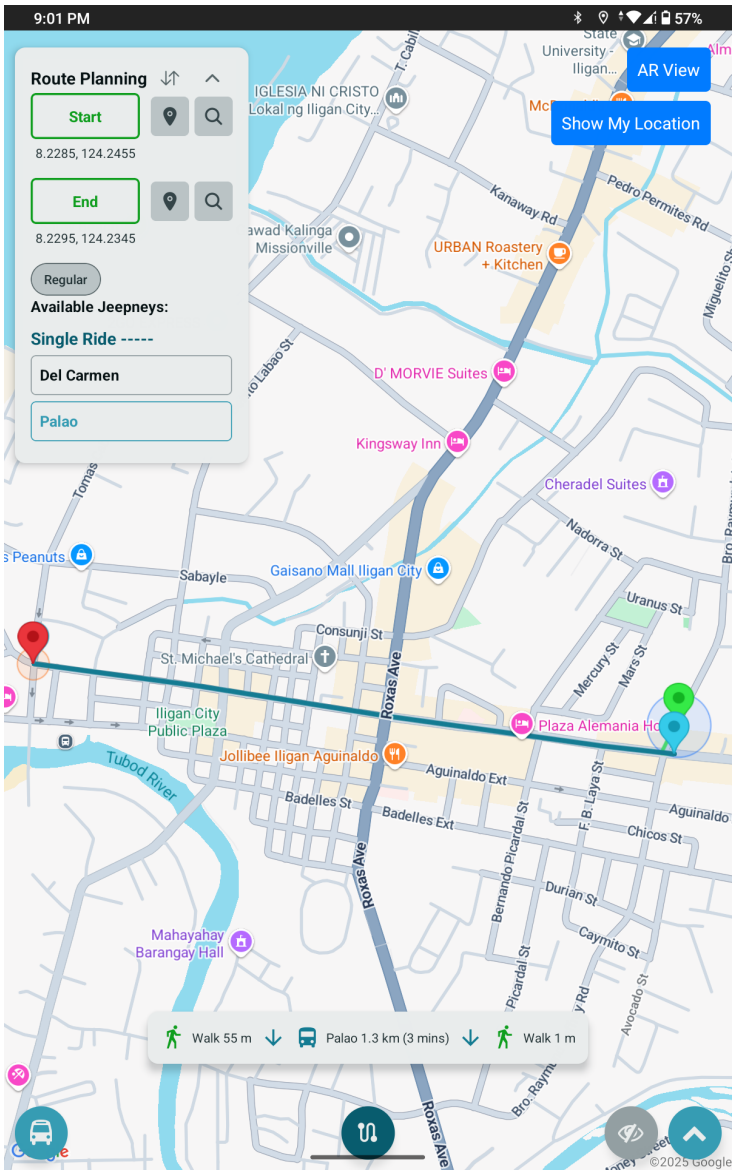

Figure 6.1: Single Ride 1

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	10.97	5.09	55
End Location Runtime (ms)	8.56	4.95	1

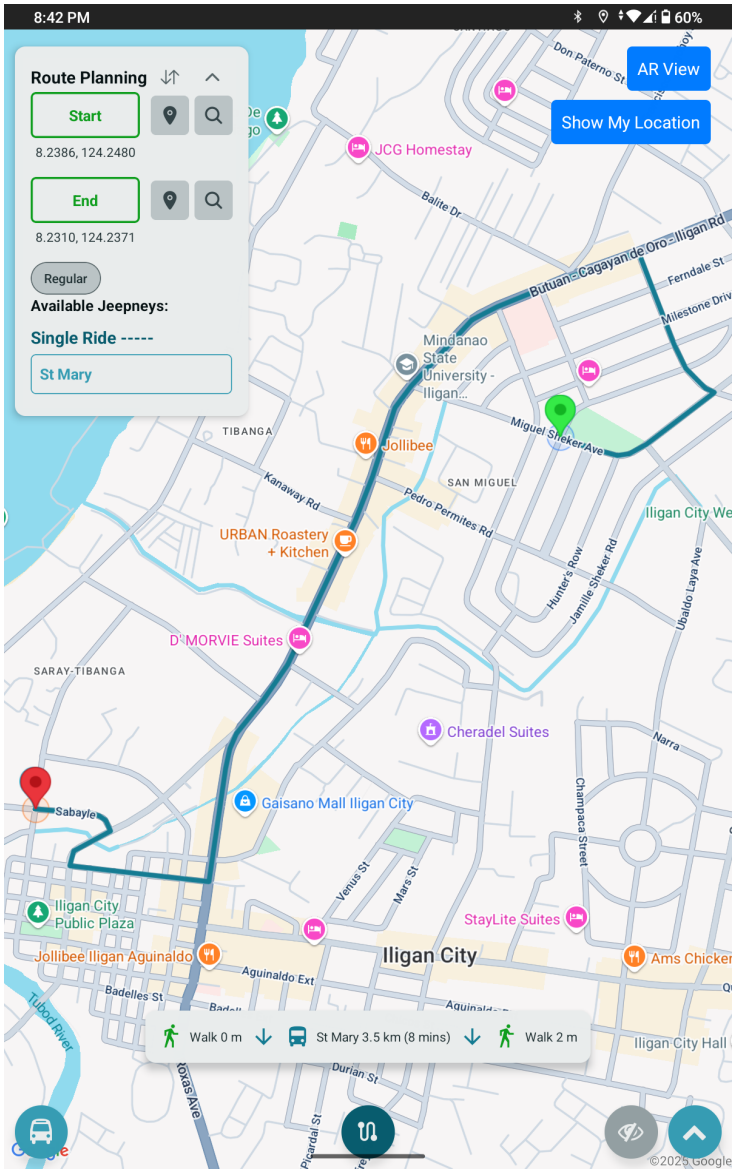

Figure 6.2: Single Ride 2

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	5.05	5.17	0
End Location Runtime (ms)	7.18	6.67	2

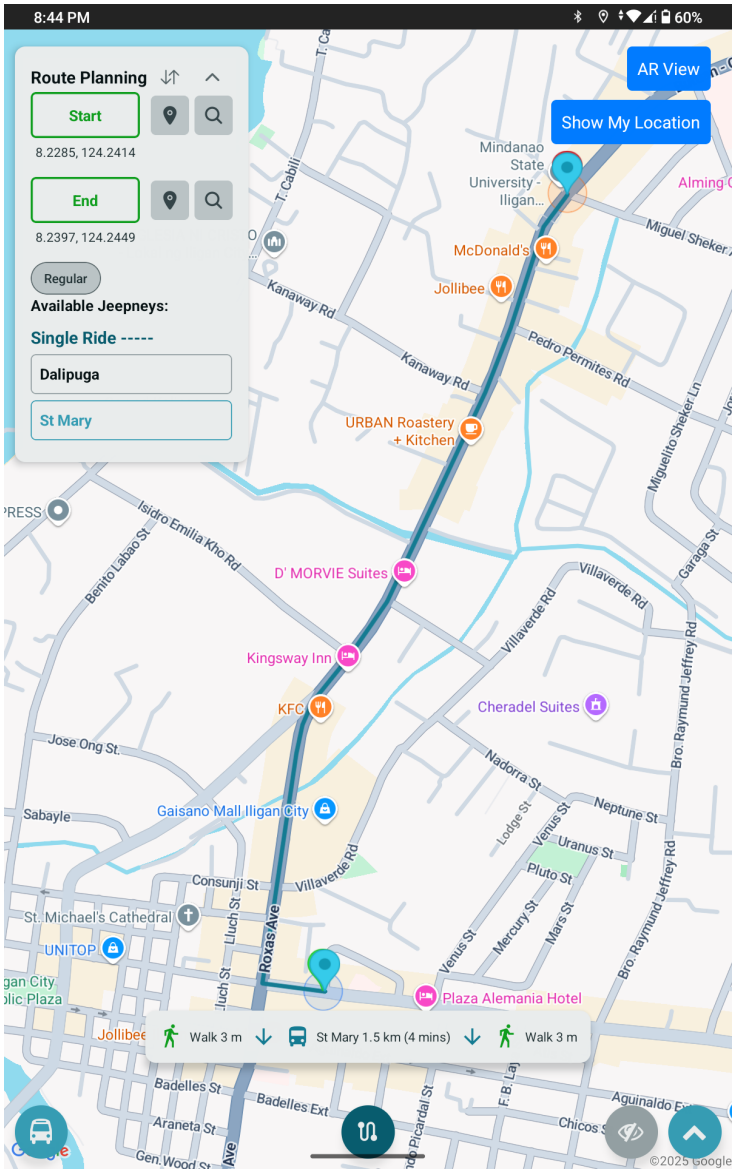

Figure 6.3: Single Ride 3

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	7.29	9.43	3
End Location Runtime (ms)	11.93	5.21	3

Figure 6.4: Single Ride 4

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	100.95	4.89	485
End Location Runtime (ms)	8.63	9.07	1

Figure 6.5: Single Ride 5

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	5.05	5.17	327
End Location Runtime (ms)	7.18	6.67	2

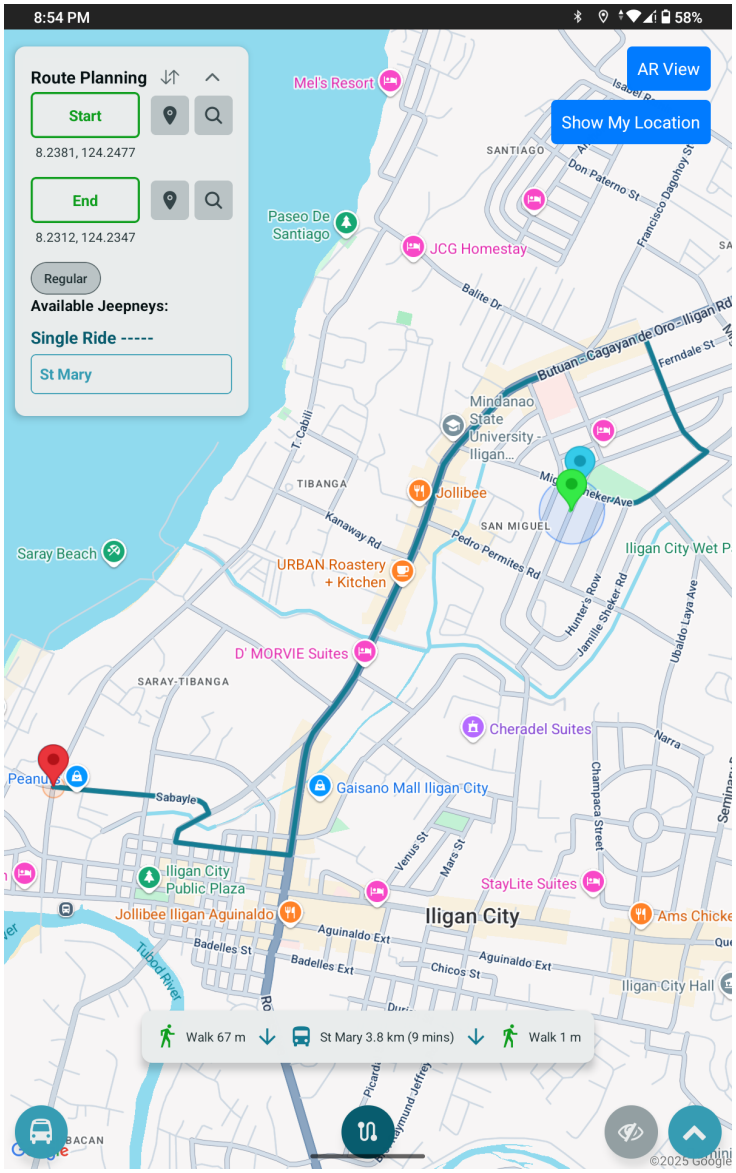

Figure 6.6: Single Ride 6

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	7.29	9.43	67
End Location Runtime (ms)	11.93	5.21	1

6.2.6.2 | Double Ride Examples

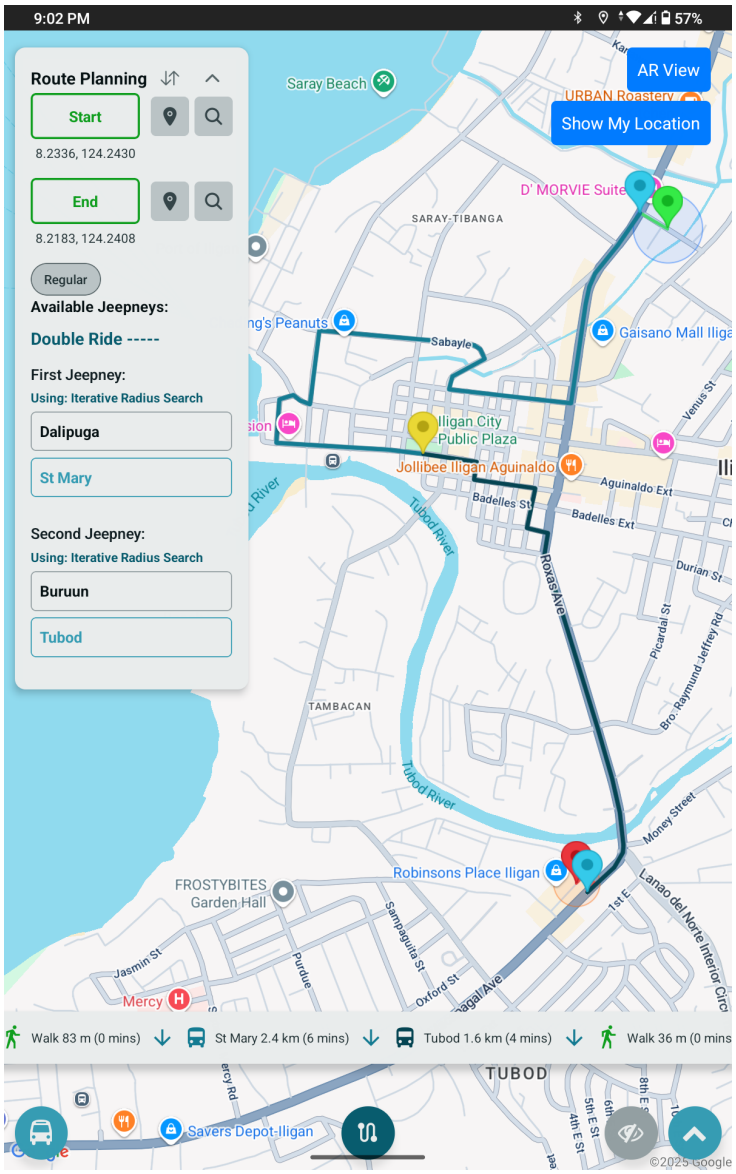

Figure 6.7: Double Ride 1

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	21.57	5.00	83
End Location Runtime (ms)	16.49	5.05	36

Figure 6.8: Double Ride 2

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	35.30	5.26	163
End Location Runtime (ms)	23.96	5.46	118

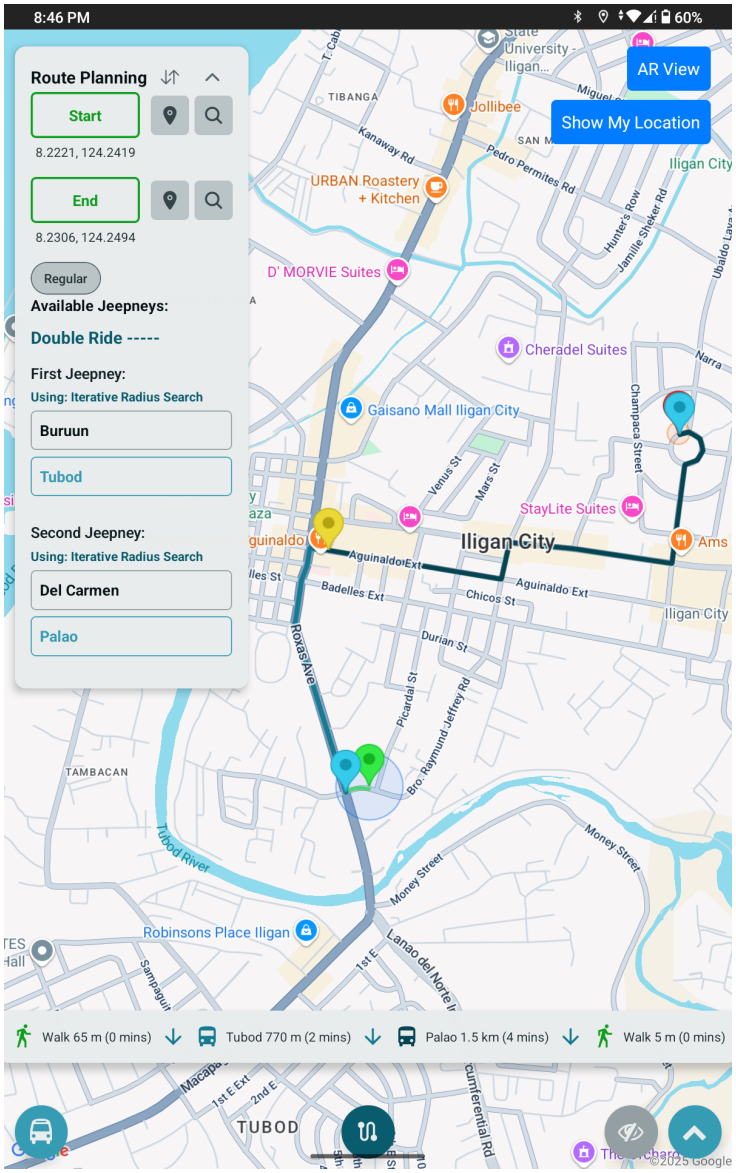

Figure 6.9: Double Ride 3

	IRS	PRMS	Distance (m)
Start Location Runtime (ms)	19.98	4.85	65
End Location Runtime (ms)	7.90	4.99	5

Figure 6.10: Double Ride 4

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	19.20	5.42	70
End Location Runtime (ms)	15.98	5.62	30

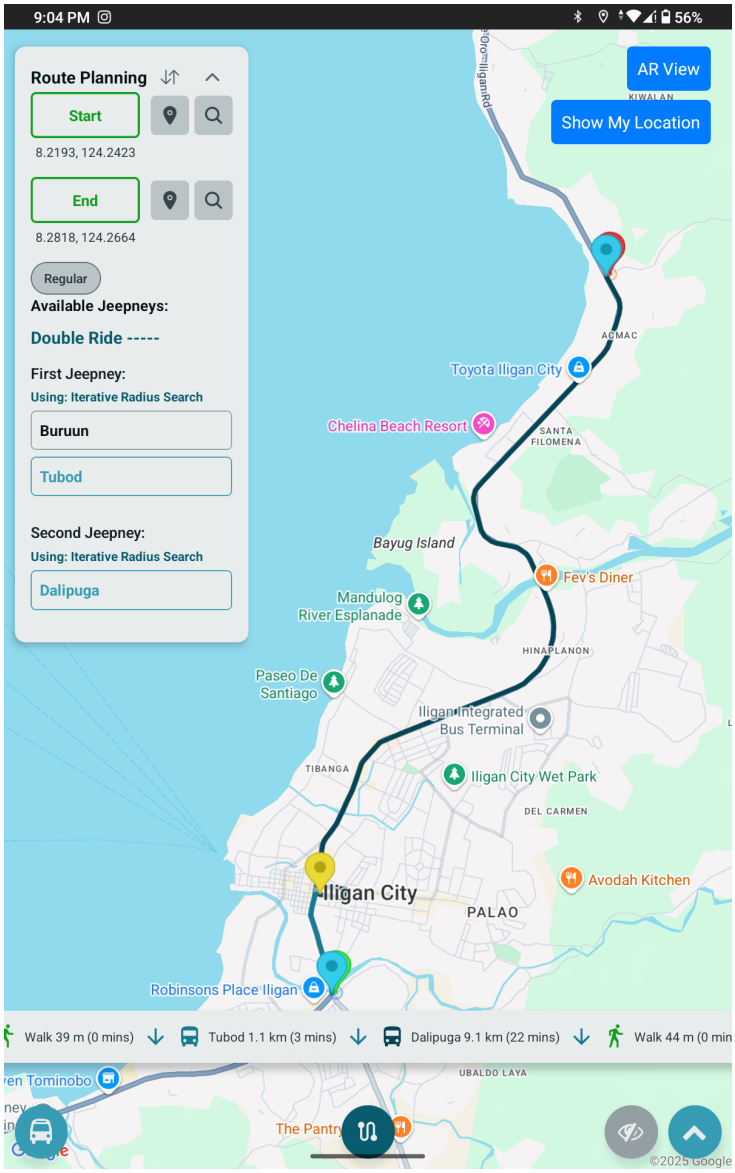

Figure 6.11: Double Ride 5

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	15.59	4.91	39
End Location Runtime (ms)	15.82	5.62	44

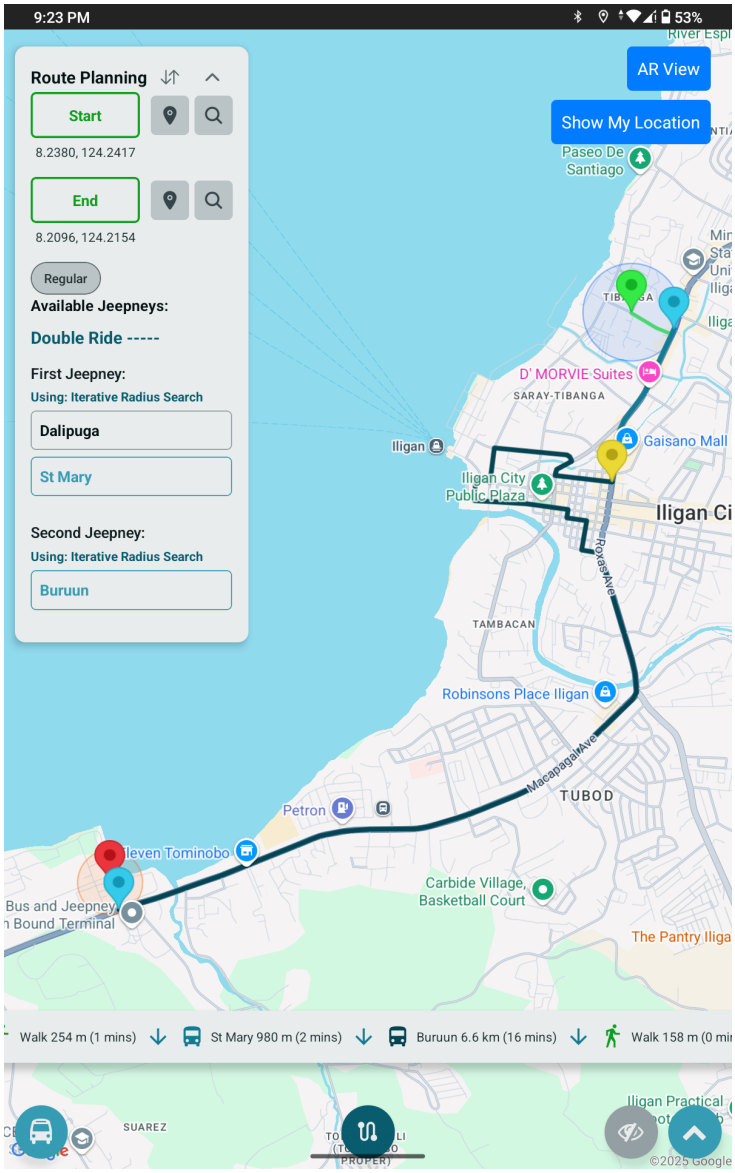

Figure 6.12: Double Ride 6

	IRS	**PRMS**	Distance (m)
Start Location Runtime (ms)	51.79	9.25	254
End Location Runtime (ms)	40.96	5.08	158

6.3 | Supporting Algorithm Design

Two essential features, walking guidance and fare computation, required supporting algorithms:

- **Walking Path Computation:** Walking directions are computed using the Open Source Routing Machine (OSRM), which generates navigable road paths between the user's location and the jeepney hailing or drop-off points.
- **Fare Estimation Logic:** Fare is calculated based on the LTFRB base fare structure. A base rate covers the first four kilometers, with additional charges applied per succeeding kilometer. The fare logic adjusts automatically for discounted user types.

Algorithm 10: Fare Calculation

Input: $distance, type$
Output: $fare$

```
Function calculate_fare(distance, type)
    fare_details ← {
     'regular' : 'base_price' : 13,' add_per_km' : 1.8,
     'discounted' : 'base_price' : 11,' add_per_km' : 1.4
     } base_price ← fare_details[fare_type]['base_price']
    add_per_km ← fare_details[fare_type]['add_per_km']
    if distance <= 4 then
        return base_price
    extra_distance ← distance − 4
    extra_fare ← extra_distance ∗ add_per_km
    total_fare ← base_price + extra_fare
    return round(total_fare, 0);
```

Algorithm 10 shows the sample code for the fare estimation of both regular and discounted(Senior Citizens, Students, PWDs) commuters.

6.4 | Data Management and Real-Time Updates

Jeepili uses Firebase Firestore to store all jeepney routes, fare matrices, and system configurations. Admin-side changes — such as adding or removing routes — are instantly pushed to Firestore and synced across all client devices.

This real-time update capability allows administrators to manage detours and event-based road changes without interrupting user experience. The app listens for changes using snapshot listeners and immediately updates the relevant route paths and recommendations.

7

Results and Discussion

This chapter presents the outputs of the Jeepili mobile application and discusses how well the system met its intended goals. It includes actual screenshots of the implemented features and how the system responded during testing. The discussion focuses on the accuracy of route recommendations, AR visualizations, fare estimation, and user interaction with the app.

7.1 | Feature Completeness

Below are the finalized features that were successfully implemented in Jeepili. These features serve as the foundation for the testing and evaluation covered in the following sections.

Completed Features:

- Smart route detection based on start and end points
- Support for single rides, double rides, and alternate routes
- Accurate fare estimation using LTFRB-based logic
- Walking path generation and rendering
- Augmented Reality (AR) support for:
    - Hailing areas
    - Stopping points

    - Jeepney route visualization
- Admin dashboard for:
    - Route editing
    - Road blockage simulation
    - Traffic rerouting
- Real-time responsiveness to coordinate and traffic input changes

This list of features was the basis of the testing procedures described in the following sections.

7.2 | Objective Testing Results

The following screenshots illustrate the outputs from various test cases performed by the developers. Each screenshot captures how specific features functioned during internal testing, offering visual proof that the system behaved as expected across different scenarios. These demonstrations were used to verify route logic, AR rendering accuracy, fare estimation, and the dynamic responsiveness of the app when conditions such as traffic or route availability were altered.

7.2.1 | Map View of Jeepney Routes

This shows the plotted jeepney routes over Google Maps. Each route is stored as an array of coordinates.

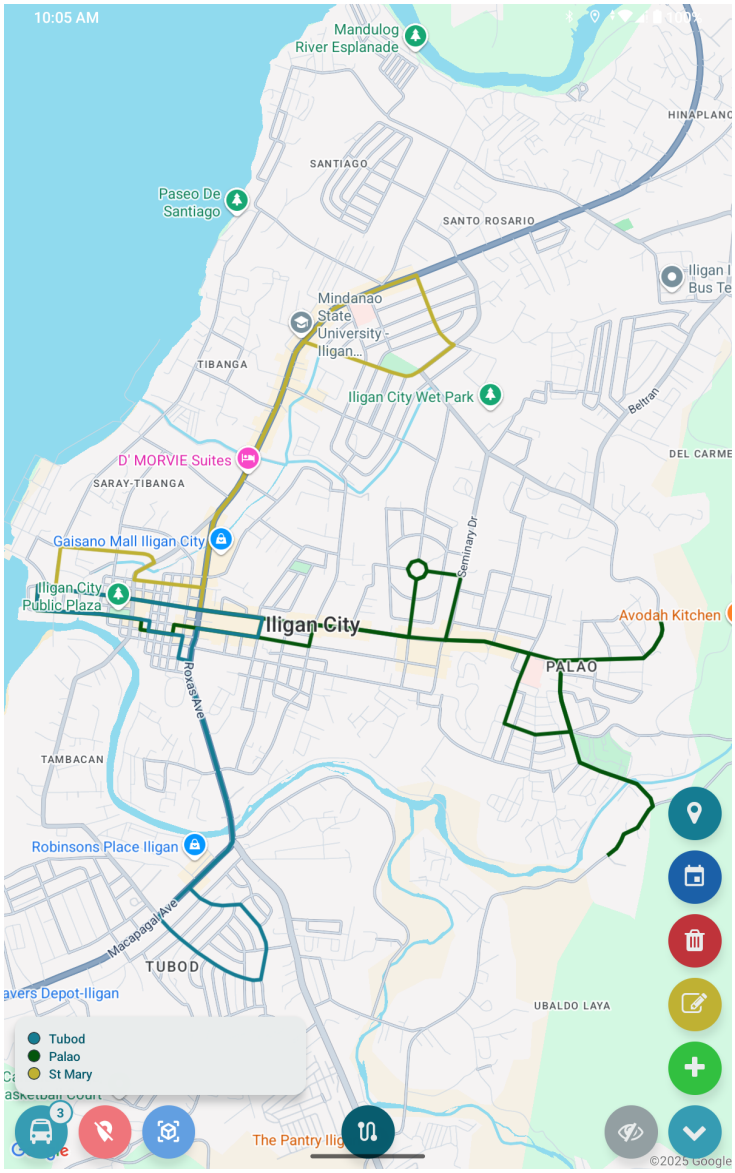

Figure 7.1: Jeepney Routes

Figure 7.1 shows some of the Jeepney Routes displayed on the map. Only a maximum of 5 can be displayed at a time to prevent a cluttered view of the map.

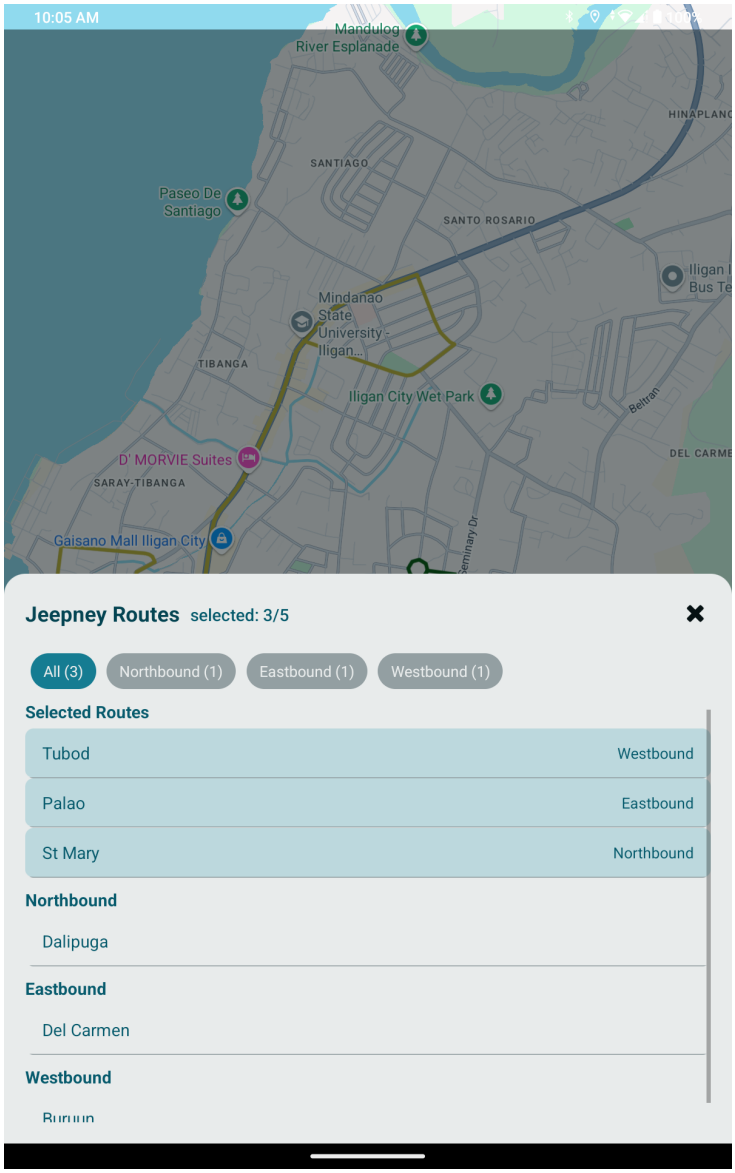

Figure 7.2: Routes list

Figure 7.2 shows the list of Jeepneys, as of the screenshot only 6 of the Jeepneys are available, two for each of the bounds: Northbound, Westbound, and Eastbound.

7.2.2 | Walking Paths

7.2.2.1 | Hailing Point

The app displays a green line from the user's location to the nearest Jeepney stop for hailing.

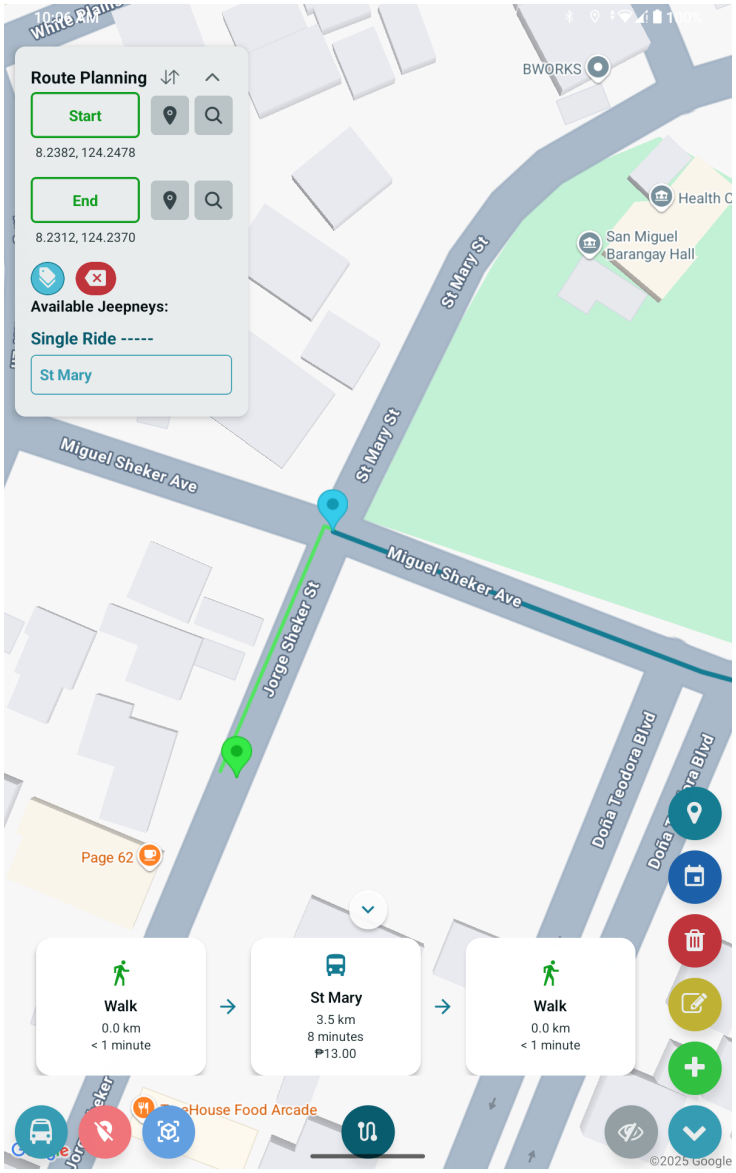

Figure 7.3: Hailing Point

Figure 7.3 shows the walking path the user will have to take in order to arrive near the hailing point.

7.2.2.2 | Destination

A red line guides the user from the Jeepney stop to their final destination.

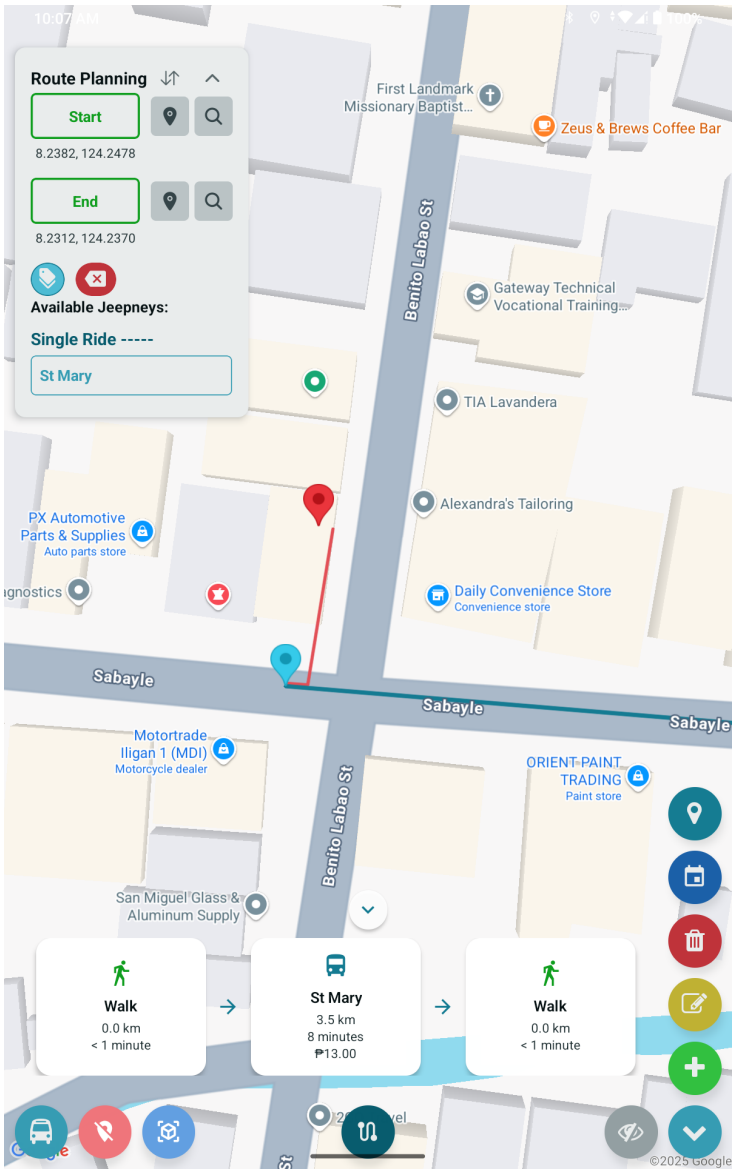

Figure 7.4: Destination Point

Figure 7.4 shows the walking path the user will have to take in order to arrive to their destination.

7.2.3 | Fare Estimation View

Figure 7.3 and Figure 7.4 displays the Fare Estimation, which includes the travel time, estimated fare, and distance.

7.2.4 | Route Results

7.2.4.1 | AR View

Visualizes jeepney paths, walking paths, hailing areas, and stop points in the user's real-world environment using AR.

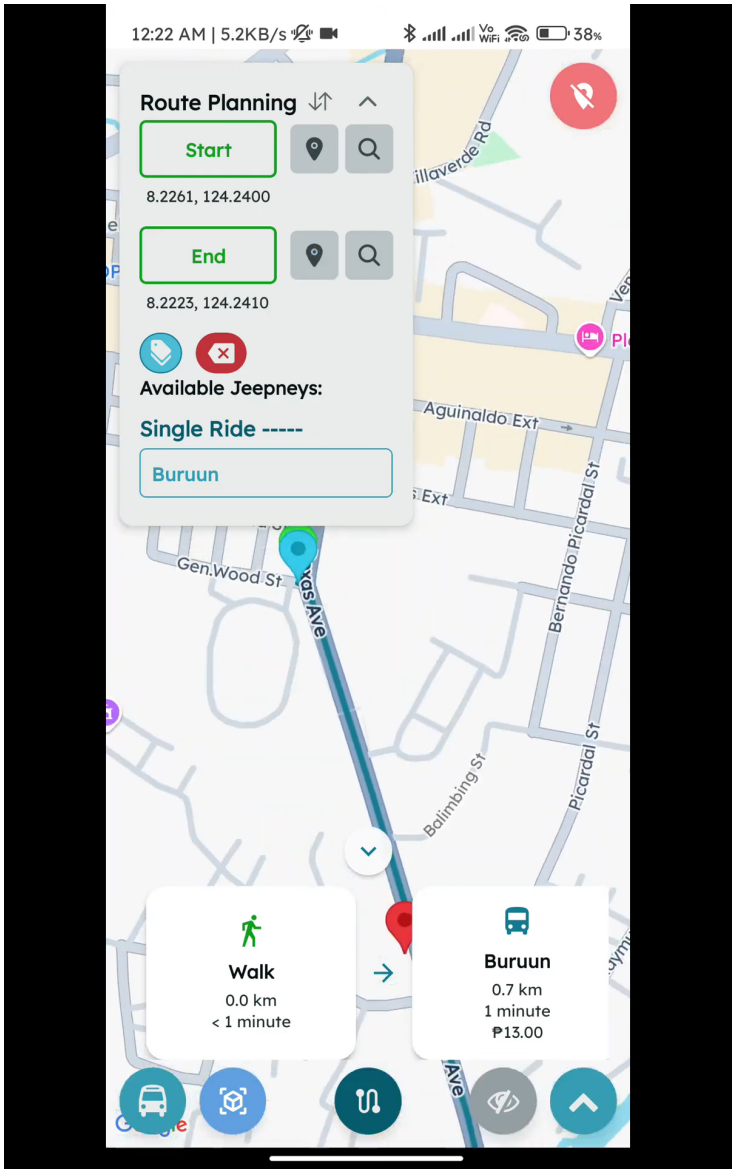

Figure 7.5: AR View Map

Figure 7.5 shows the Jeepney route initiated by the user to be viewed during AR view.

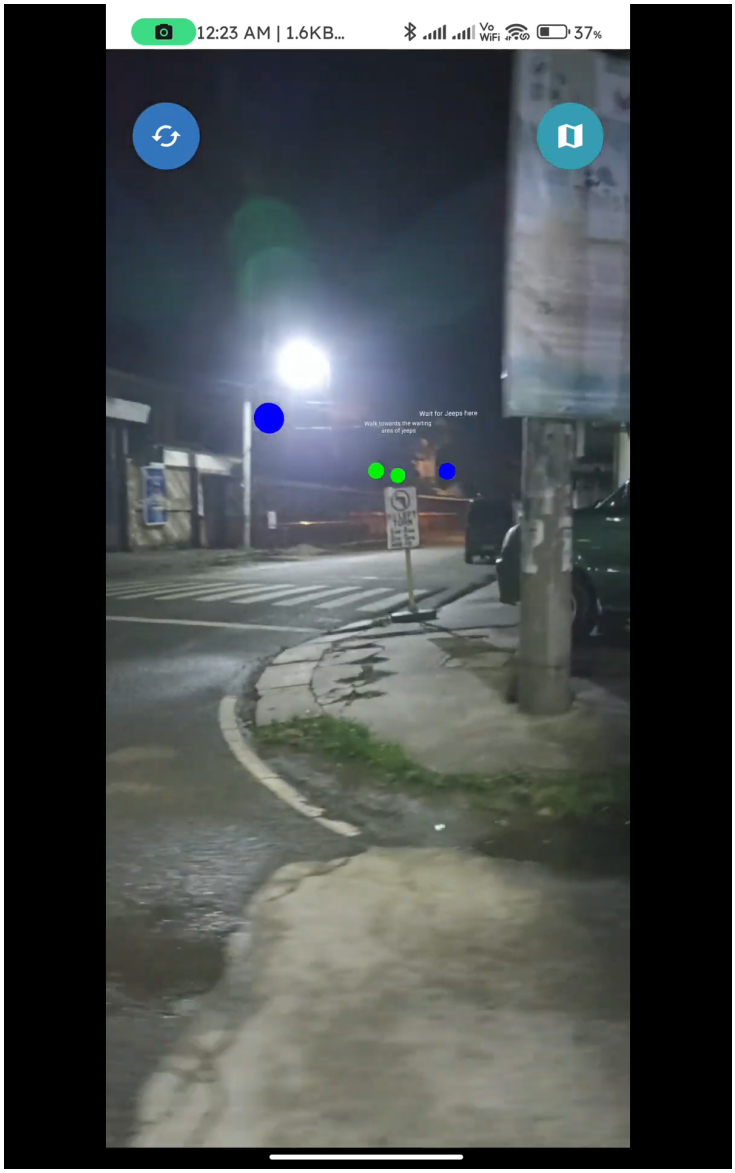

Figure 7.6: AR View Start

Figure 7.6 shows the walking path (in green) leading to the Jeepney hailing area, and the Jeepney route itself (in blue). At the start of the walking path there will be a text saying "Walk towards the waiting area of the jeeps" and at the start of the Jeepney route there will be a text saying: "Wait for Jeeps here", The Markers are quite far from the user in Figure 7.6 to make the text visible, but in Figure 7.7 these texts can be more readable since they are near.

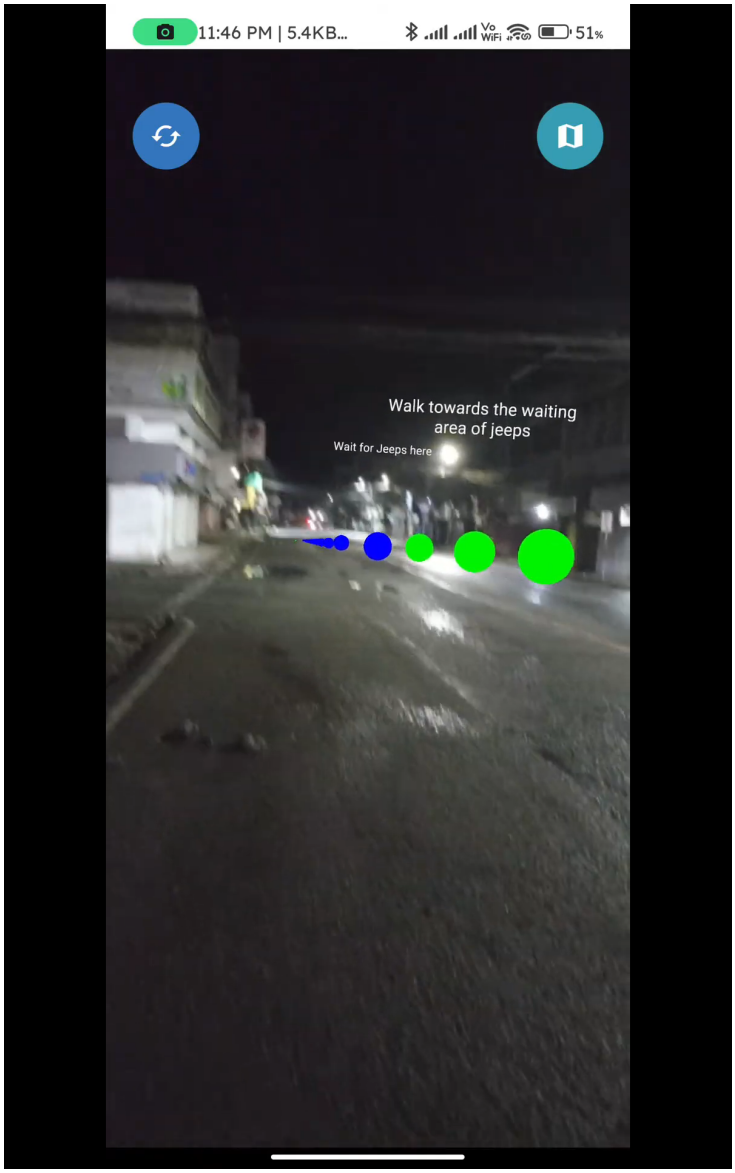

Figure 7.7: AR View Start Example

Figure 7.7 shows the walking path(in green) with the text: "Walk towards the waiting area of the jeeps" and the Jeepney Path (in blue) with the text: "Wait for Jeeps here"

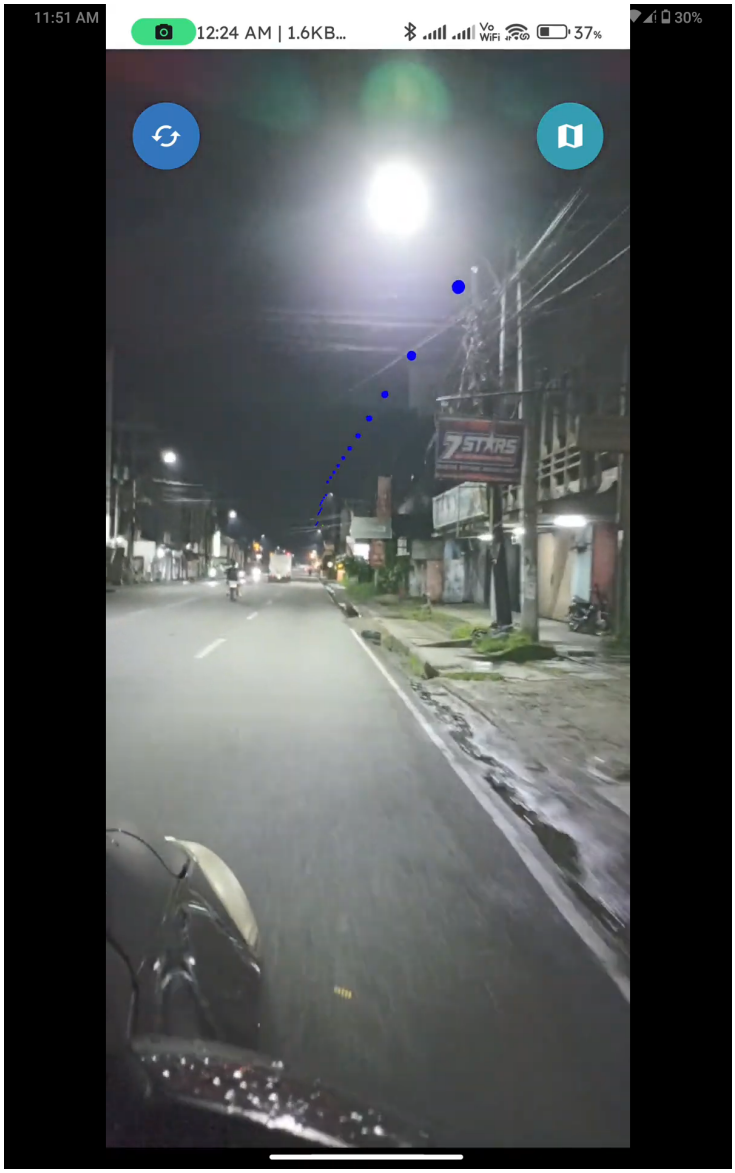

Figure 7.8: Jeepney Path in AR

Figure 7.8 shows the path of the Jeepney(in blue) during the journey.

Figure 7.9: AR View End

Figure 7.9 shows the end of the Jeepney Path (in blue) with the text saying "Stop Here and Walk towards your Destination". The Markers are quite far from the user in Figure 7.9 to make the text visible, but in Figure 7.10 these texts can be more readable since they are near.

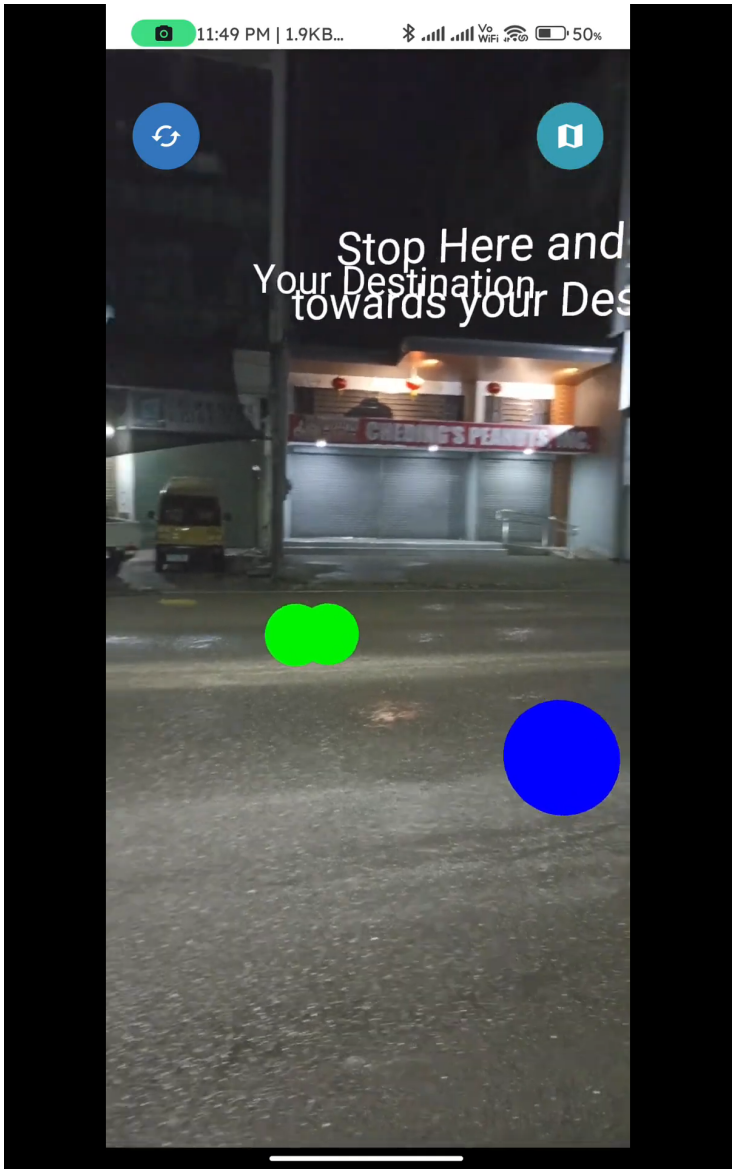

Figure 7.10: AR View End Example

Figure 7.10 shows the end of the Jeepney Path (in blue) with the text saying "Stop Here and Walk towards your Destination". The end of the walking path (in green) shows a text saying "Your Destination".

7.2.4.2 | Alternate Route

Shows how the system reacts and reroutes when a user removes the default jeepney.

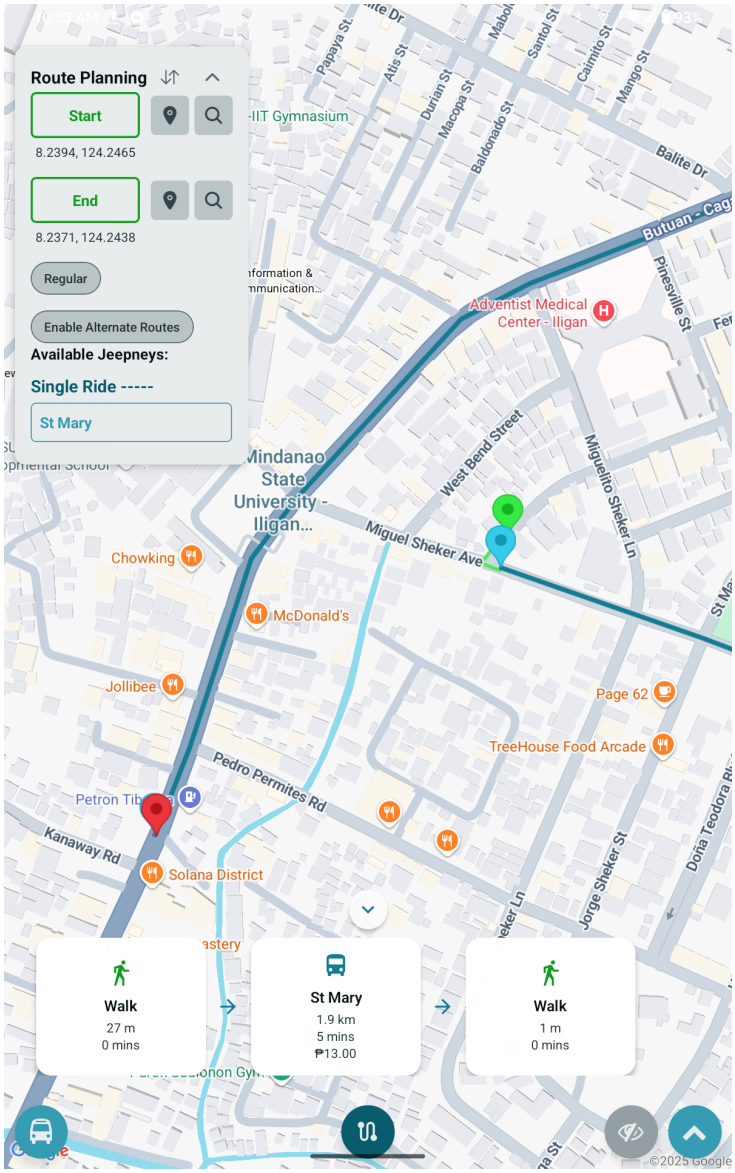

Figure 7.11: Before Alternate

Figure 7.11 shows that based on the start and end location, only one Jeepney is available which is St Mary. Therefore alternate can be enabled in case the only Jeepney is unavailable or taking too long.

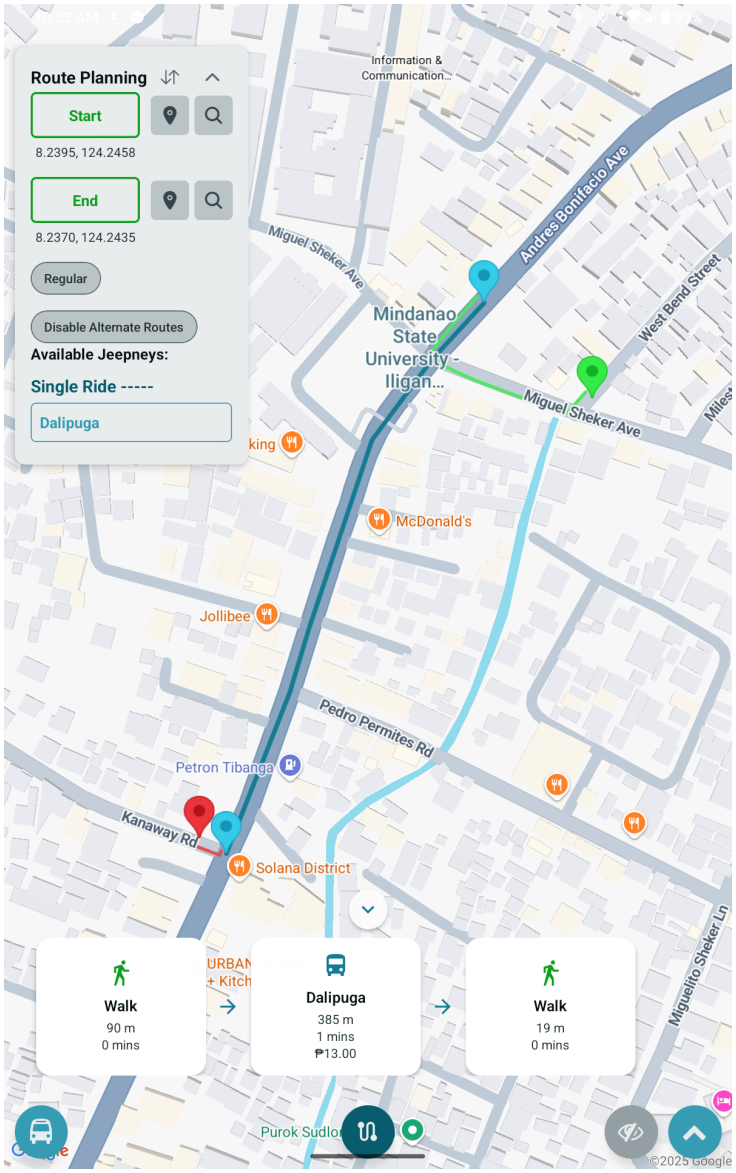

Figure 7.12: After Alternate

Figure 7.12 shows what happens to the walking suggestions when alternate is enabled; it suggests the user to walk near the highway to wait for the other jeeps. As of the screenshot, only Dalipuga is the one recommended but the suggestion of Dalipuga represents the suggestion of the other Northbound jeeps.

7.2.4.3 | Double Ride

Shows the transfer point between the first and second jeepneys.

Figure 7.13: Common Point

Figure 7.13 shows the location marker in yellow where the user should stop their first ride Jeepney, and hail for their second ride Jeepney in order to arrive to their destination.

7.2.4.4 | Admin Dashboard

Overview of the administrator dashboard used for managing routes and simulating road conditions.

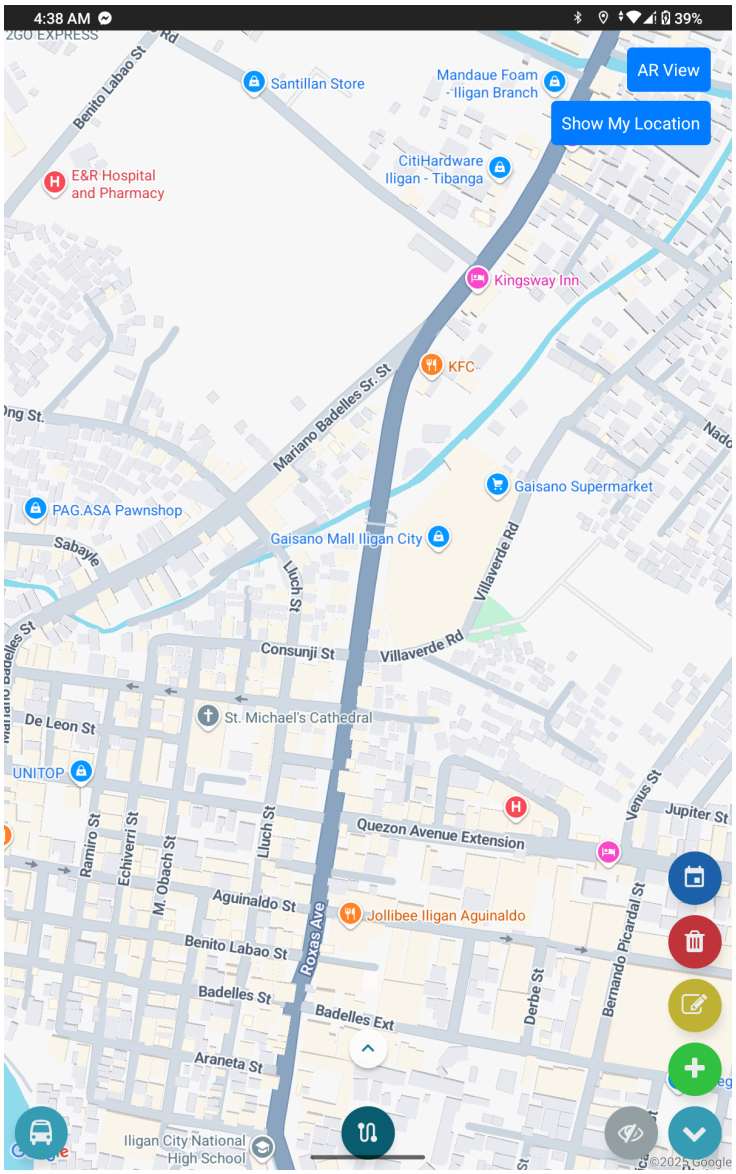

Figure 7.14: Admin Buttons

Figure 7.14 shows the admin buttons on the bottom right side. It allows the admin to: create a Jeepney route, edit a Jeepney route, delete a Jeepney route, and add an Event, which will cause the Jeepneys to take a detour.

7.2.4.5 | Admin Add Jeep

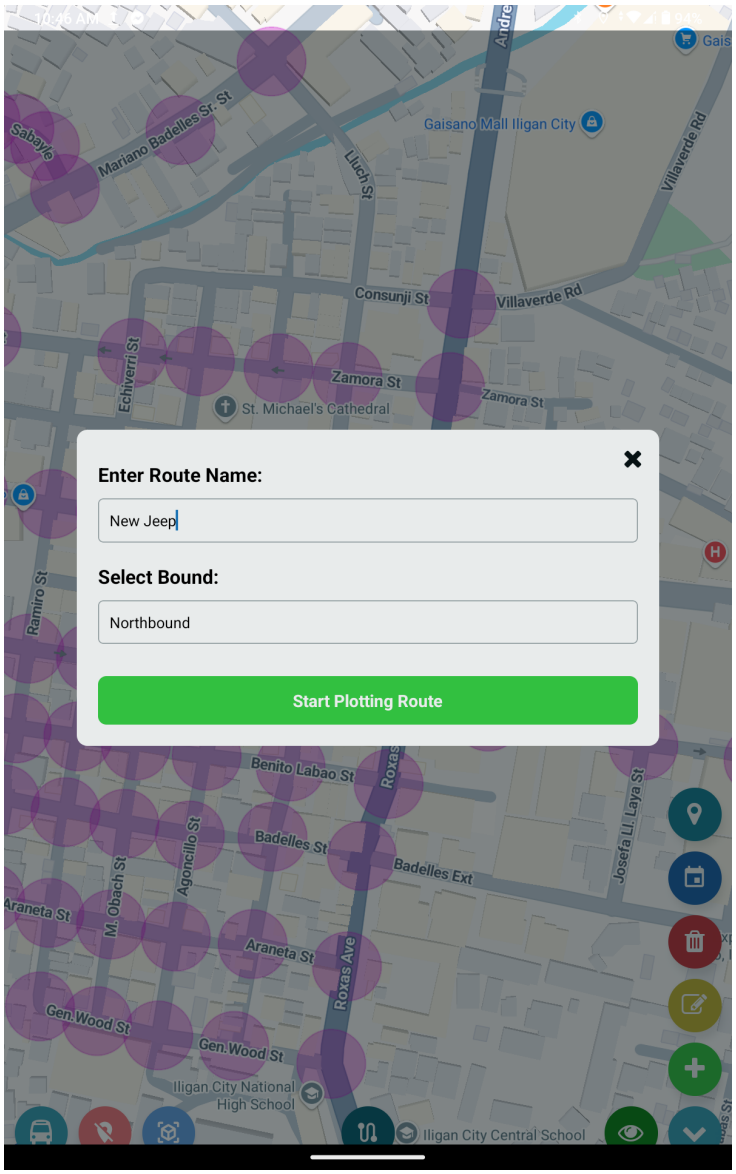

Figure 7.15: Admin Add Jeep

Figure 7.15 shows the Jeepney creation mode. After confirming the name and bound, the admin will then need to click all the snapzones (in pink) in the correct order to plot the full path of the Jeepney.

7.2.4.6 | Admin Add Plot Route

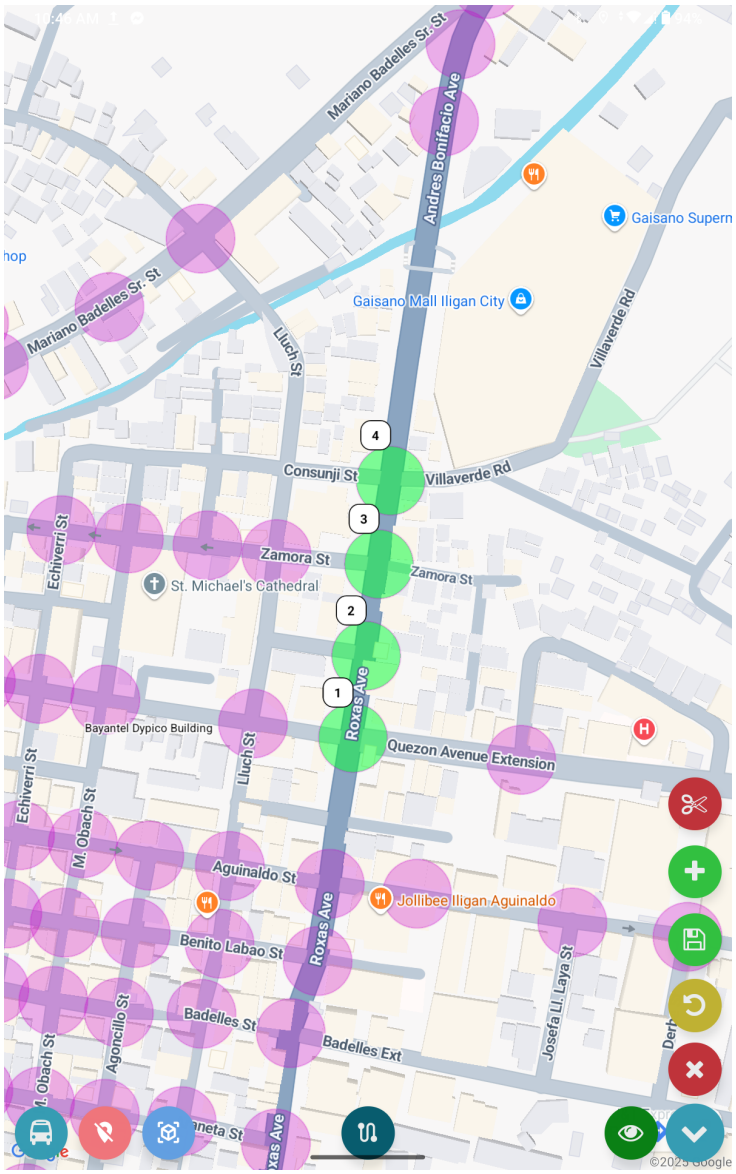

Figure 7.16: Admin Add Plot Route

Figure 7.16 shows the snapzones the admin will have to click in order to declare the path the created Jeepney will take. The snapzones that are available and are not selected are in pink, while the selected snapzones will be in green together with their order number.

7.2.4.7 | Admin Edit List

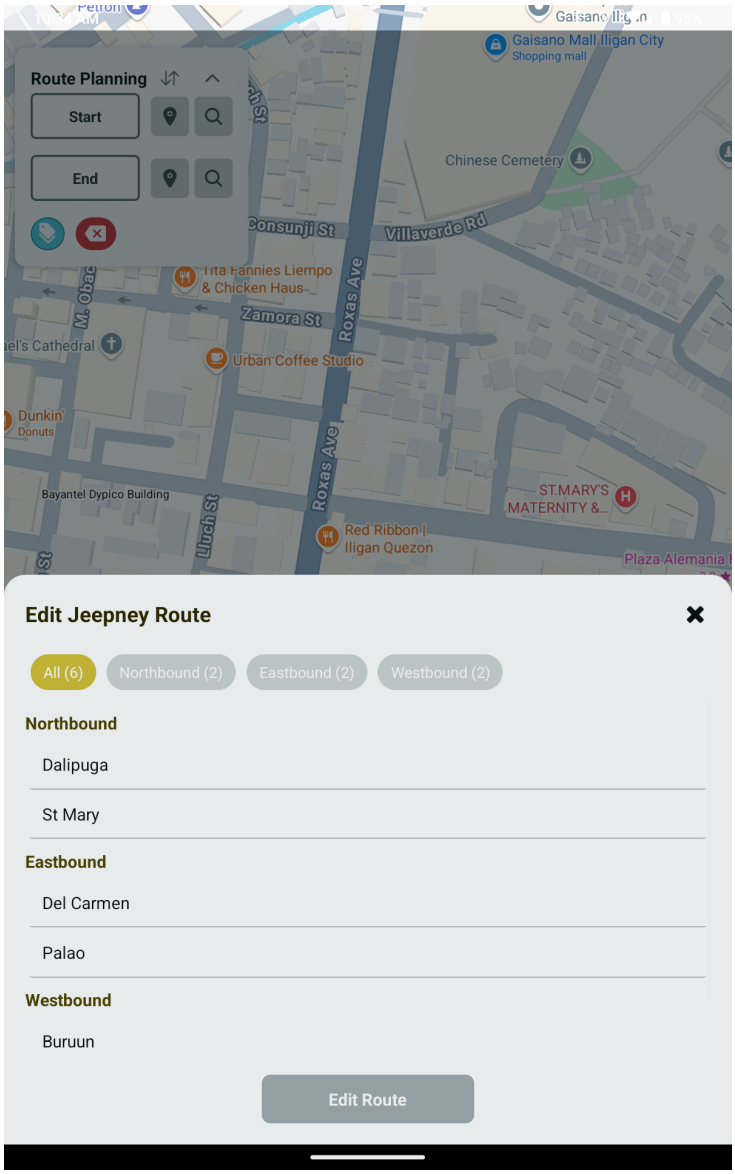

Figure 7.17: Admin Edit List

Figure 7.17 shows the Jeepney edit mode. The admin will select the Jeepney they wish to edit.

7.2.4.8 | Admin Edit Route

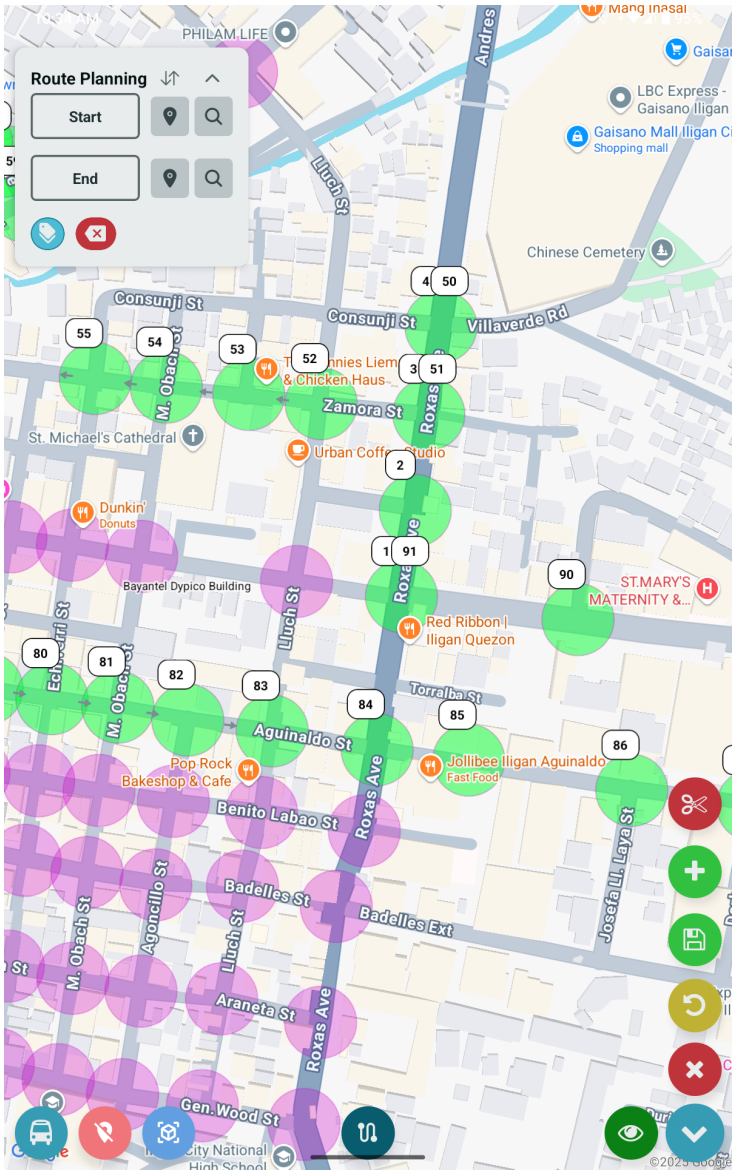

Figure 7.18: Admin Edit Route

Figure 7.18 shows the path of the selected Jeepney in their correct order. The admin can then wish to tweak the path, if necessary.

7.2.4.9 | Admin Events List

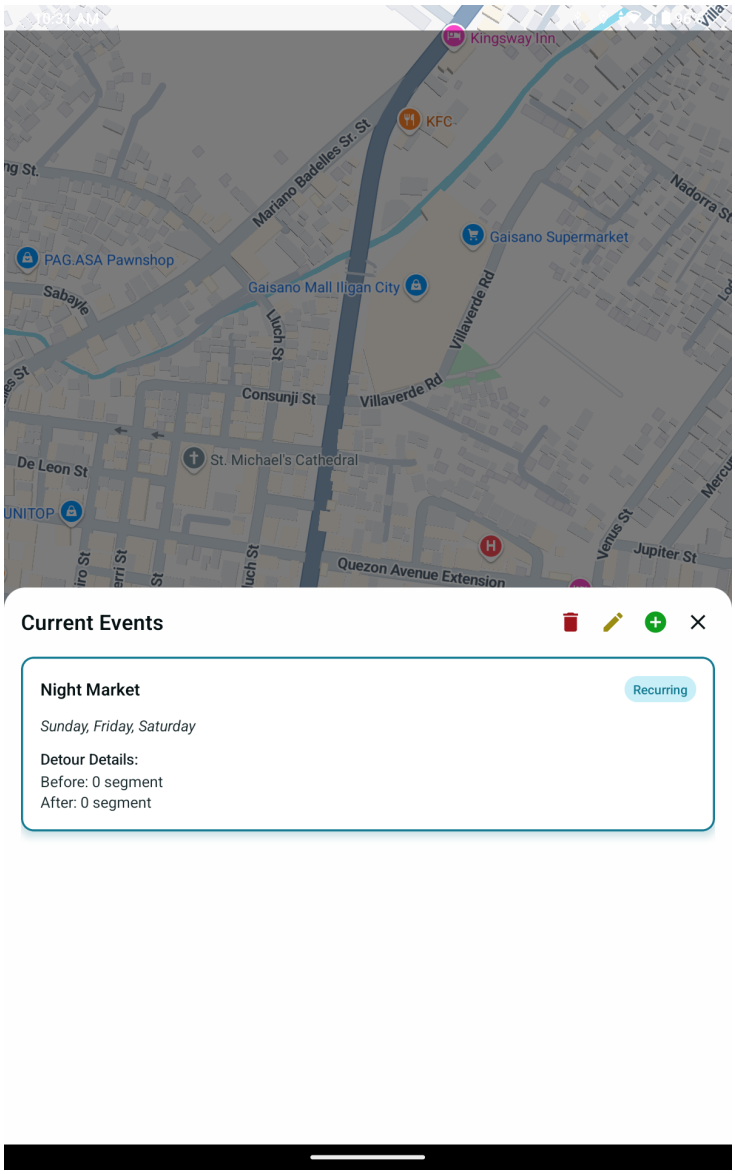

Figure 7.19: Admin Events List

Figure 7.19 shows the list of events; from here the admin can add, edit, or delete events.

7.2.4.10 | Admin Add Event

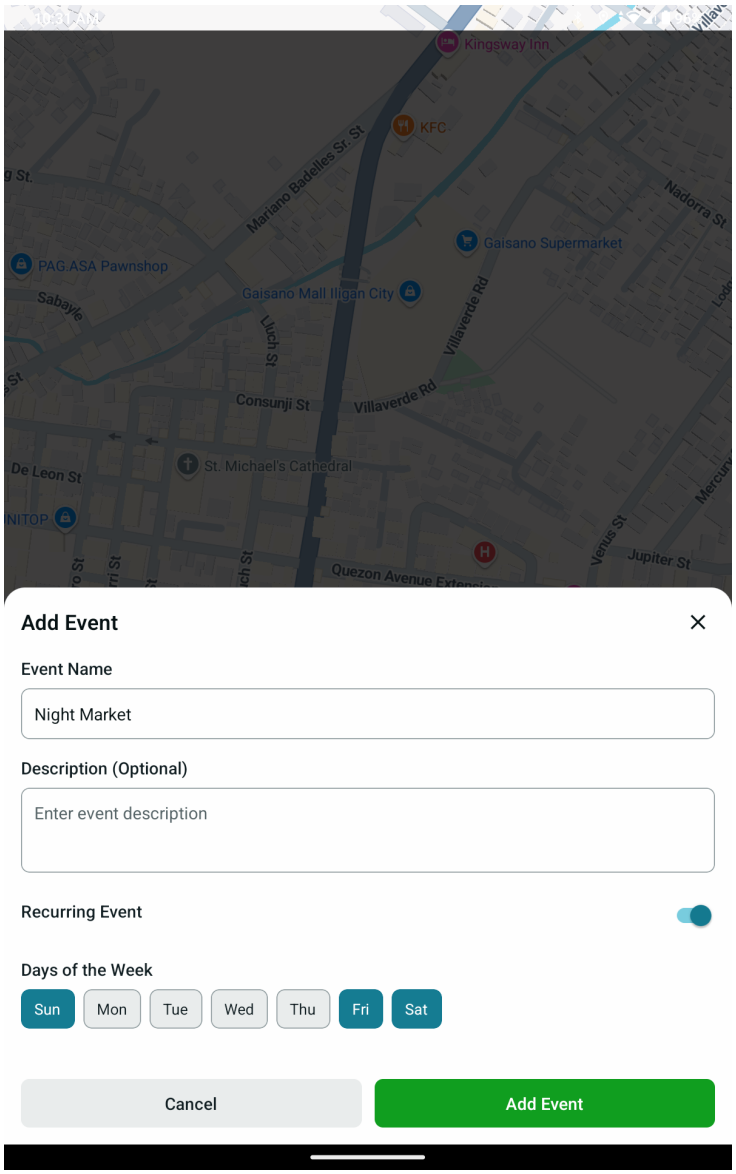

Figure 7.20: Admin Add Event

Figure 7.20 shows the add event panel, here the user can set the title of the event that will make the Jeepneys detour from their route. They can also set if this detour is recurring, like Night Market for example, which happens every Friday, Saturday, and Sunday.

7.2.4.11 | Admin Event Define Normal

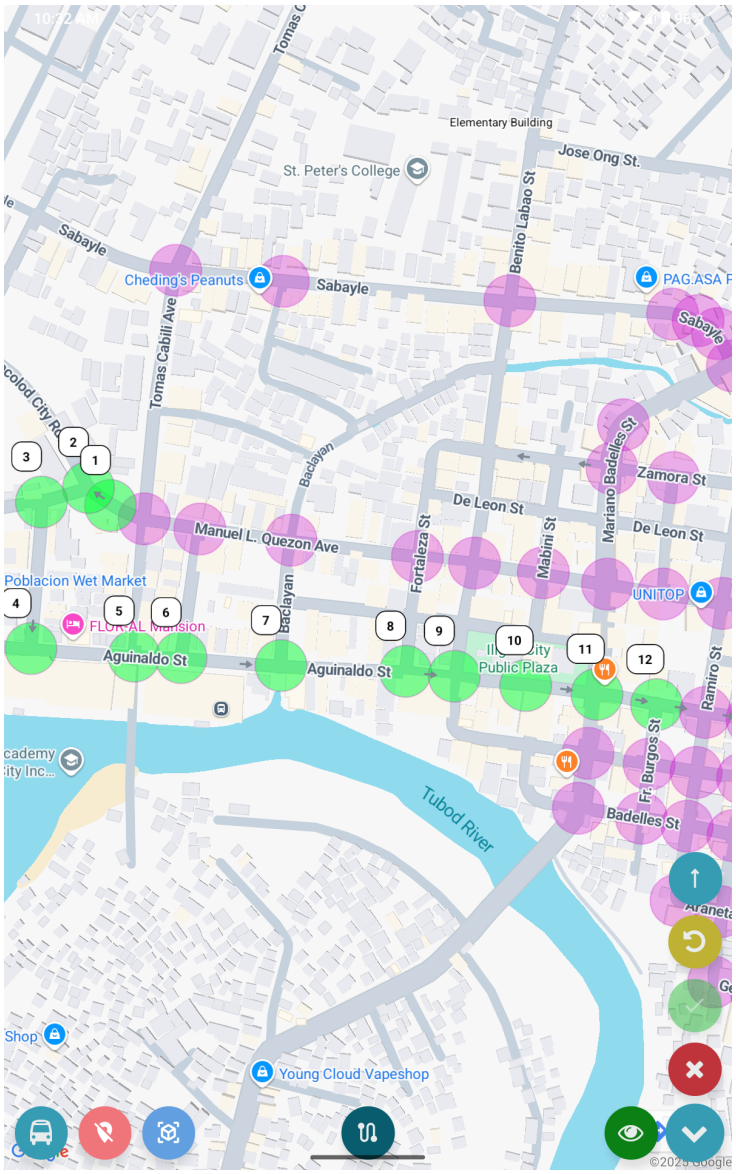

Figure 7.21: Admin Event Define Normal

Figure 7.21 After the admin wants to change the detour setup, they will need to first define what is the normal path. In Figure 7.21, the example depicts the normal path of Jeeps.

7.2.4.12 | Admin Event Define Alternate

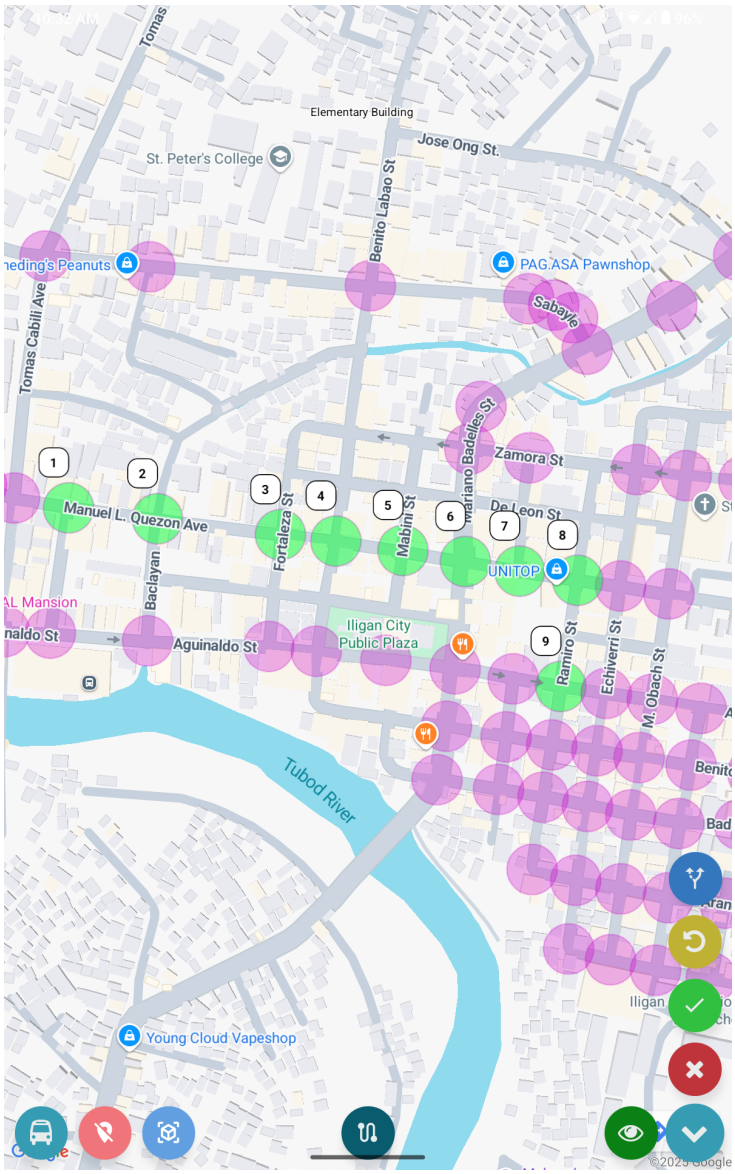

Figure 7.22: Admin Event Define Alternate

Figure 7.22 After the admin defines the normal, they will then need to define the detour. In 7.22, the example depicts the alternate path or the detoured path of the Jeepneys when it is Friday, Saturday, and Sunday due to the Night Market.

7.2.4.13 | Event Before and After

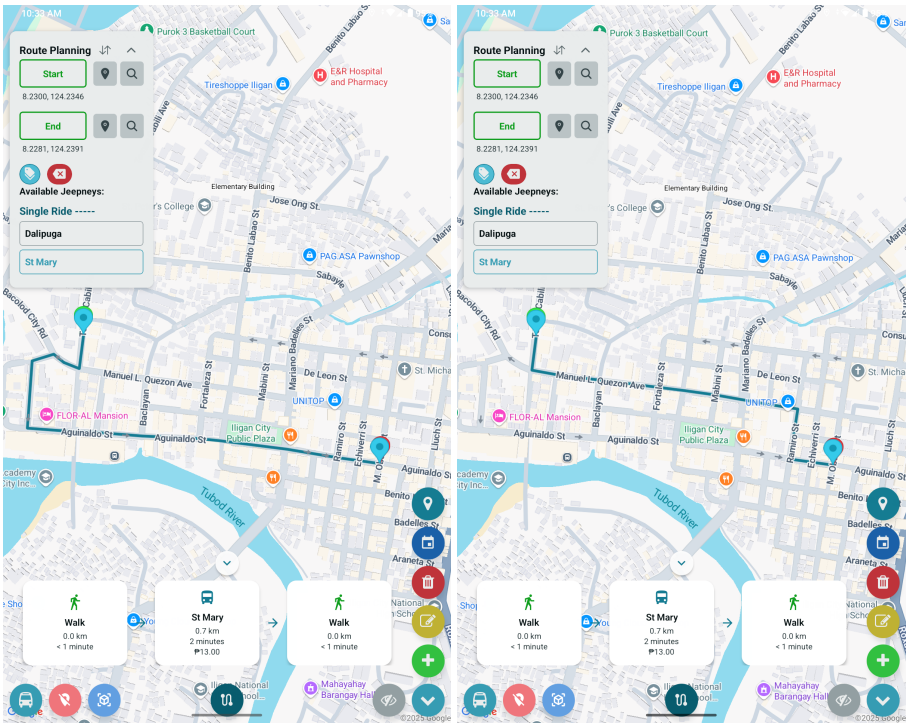

Figure 7.23: Event Before and After

Figure 7.23 is currently viewing a journey, the image on the left shows the normal path of the Jeepney, and the image on the right shows the detoured path of the Jeepney due to an Event. All Jeepneys that will get affected by an event will automatically get rerouted or detoured.

Each screenshot confirms that the app responded based on user input and real-time route conditions, with PRMS powering most of the matching and filtering logic.

Each screenshot confirms that the visual elements were rendered correctly and that the app responded based on user input and real-time data.

7.3 | System Testing

During the transition from development to testing, the Jeepili system was evaluated on ARCore-supported Android devices with active GPS and internet connectivity. This

phase aimed to validate the performance of the PRMS algorithm and the updated AR rendering logic.

7.3.1 | Route Detection Accuracy

Tests were conducted to verify if the PRMS logic could correctly identify jeepney routes near the user's start and destination locations. PRMS operates in a single pass, which was tested for consistency across different ride types:

- Detection of the nearest jeepney coordinates at both origin and destination
- Matching correct route IDs for single rides, double rides, and alternate routing logic
- Handling of edge cases with overlapping jeepney paths or long walking distances

7.3.2 | Walking Path Accuracy

The walking path generator was validated for precision and realism, now using PRMS-generated coordinates as start and end anchors.

- Paths were checked for real-world alignment with sidewalks and known walkable segments
- Waypoints from PRMS were confirmed to fall within accessible range from user input
- Visual placement in both Map and AR views was inspected for errors

7.3.3 | AR Visualization Stability

The AR system was tested using the updated orientation-aware rendering. The logic includes:

- Dynamic scaling of marker size based on the user's distance to each marker
- Visibility toggling based on device orientation and heading

- Position adjustments within the AR view depending on field of view and relative direction
- Smooth appearance/disappearance transitions during movement and rotation

7.4 | System Evaluation

The overall behavior of the system was reviewed under various commuting scenarios to determine its reliability and responsiveness in real-world conditions. This included evaluating how well the system adjusted to unexpected changes and whether its guidance aligned with local commuting norms.

- **Accuracy of Suggestions:** Most suggestions match how Iligan locals would commute. The waiting points and stop points generally reflect the best real-world choices.
- **Adaptability to Changes:** When preferred jeepneys are removed, the system correctly re-evaluates and offers alternate paths even if it means more walking.
- **Dynamic Adjustments:** When the simulated traffic module is enabled, the routes and ETAs adapt in real-time to reflect those changes.
- **AR Integration:** The AR experience aligns well with the physical environment. Paths and markers appear in the right places when tested using ARCore-supported devices.
- **Admin Adjustments:** Edits made in the admin dashboard reflect immediately on the client side, showing successful live updates of route and traffic conditions.

7.5 | Summary

Testing confirmed that Jeepili successfully delivered its core functionalities across a range of scenarios using ARCore-compatible Android smartphones. The system consistently identified appropriate jeepney routes for single, double, and alternate rides using the Proximity-based Route Matching Search (PRMS), a one-pass algorithm optimized for fast, scalable geospatial queries.

AR visualizations performed reliably, with features like marker resizing, visibility filtering based on device orientation, and smooth transition animations enhancing the overall experience. Fare estimations aligned with LTFRB standards, and walking paths appeared naturally anchored to accessible street points.

The admin dashboard enabled dynamic simulation of road conditions and route changes, with updates reflected in real-time on the user's screen. Overall, PRMS and AR integration significantly improved user responsiveness, clarity of navigation, and commuter decision-making — setting Jeepili apart from traditional transit apps.

8

Conclusions

8.1 | Achieved Objectives

The project successfully met its primary objectives: (1) to assist Iligan City commuters through a smart routing mobile application, and (2) to enhance the navigation experience using augmented reality (AR). Jeepili correctly identified and displayed jeepney routes based on user-specified locations. It supported single rides, double rides, and alternate route recommendations. The fare estimation system followed standard pricing models, while AR overlays improved the intuitiveness of stop and route guidance. The admin dashboard also enabled route edits and traffic scenario simulations in real-time.

8.2 | Critiques and Limitations

During internal testing, several limitations were observed. On lower-end devices, some AR elements occasionally experienced lag, reduced frame rate, increase the mobile's temperature significantly, or just not working at all.. GPS fluctuations also caused minor inaccuracies in the placement of walking paths, especially in dense urban areas where signal reflections were more frequent. Additionally, traffic simulations were based on preset conditions and not connected to any live traffic data source, which limited the realism of real-time route adjustments. Although these issues did not significantly hinder app usability, they highlight areas for further system optimization.

8.3 | Recommendations and Future Works

To expand Jeepili's capabilities and reach, future development should focus on deeper system optimization, broader compatibility, and improved user experience. Optimizing AR performance across a wider range of Android devices—especially mid-range models—would help improve accessibility for more users. Introducing offline mode could be valuable for commuters in areas with unstable mobile connectivity. Furthermore, incorporating localized user feedback from a broader commuter base will help refine route logic, fare accuracy, and path visualizations beyond internal testing. Enhancing the admin dashboard with more detailed analytics, traffic trend simulations, and route control options could also empower transport authorities or institutions to adapt Jeepili for larger-scale implementation.

8.4 | Final Remarks

Jeepili demonstrates the potential of combining smart routing algorithms with AR technology to modernize commuting in local public transportation. It addresses a specific urban challenge with a functional, scalable, and user-oriented solution. With refinement and continued development, Jeepili can become a staple tool for navigating jeepney commutes in Iligan and potentially other Philippine cities.

A

Resource Persons

- **Maulana, Malikey M.**
  Adviser
  MSU-Iligan Institute of Technology
  malikey.maulana@g.msuiit.edu.ph

B

Personal Vitae: The Researchers

- **Pineda, Zeus Morley S.**
  4th year BS Computer Science
  MSU-Iligan Institute of Technology
  Torralba St. Poblacion, Iligan City,
  Lanao del Norte, 9200, Philippines
  (+63)976 308 8043
  zeumorley.pineda@g.msuiit.edu.ph

- **Bayola, Roel S.**
  4th year BS Computer Science
  MSU-Iligan Institute of Technology
  Acmac, Iligan City,
  Lanao del Norte, 9200, Philippines
  (+63)976 171 9699
  roel.bayola@g.msuiit.edu.ph

- **Ocao, James C.**
  4th year BS Computer Science
  MSU-Iligan Institute of Technology
  Carbide Village, Tubod, Iligan City,
  Lanao del Norte, 9200, Philippines
  (+63)966 382 0717
  james.ocao@g.msuiit.edu.ph

References

Tharindu Abeywickrama, Ankur K. Sahu, and Shanika Fernando. k-nearest neighbors on road networks: A journey in experimentation and in-memory implementation. *Proceedings of the VLDB Endowment*, 9(7): 492–503, 2016.

Lynn Htet Aung, Soe Thandar Aung, Nobuo Funabiki, Htoo Htoo Sandi Kyaw, and Wen-Chung Kao. An implementation of web-based answer platform in the flutter programming learning assistant system using docker compose. *Electronics*, 13(24), December 2024. doi: 10.3390/electronics13244878.

Wei Chiang Chan, Wan Hashim Wan Ibrahim, May Chiun Lo, Mohamad Kadim Suaidi, and Shiaw Tong Ha. Sustainability of public transportation: An examination of user behavior to real-time gps tracking application. *Sustainability*, 12(22), November 2020. doi: 10.3390/su12229541.

Kenneth L Cooke and Eric Halsey. The shortest route through a network with time-dependent internodal transit times. *Mathematical Analysis and Applications*, 14(3):493–498, June 1996. doi: 10.1016/0022-247X(66)90009-6.

Laxman Dhulipala, Guy E Blelloch, and Julian Shun. Theoretically efficient parallel graph algorithms can be fast and scalable. *ACM Trans. Parallel Comput.*, 8(1), April 2021. doi: 10.1145/3434393.

Zhenghua Feng, Ziyuan Li, Dong Wang, Hongzhi Li, and Yu Zheng. Efficient knn search in public transportation networks. *Proceedings of the VLDB Endowment*, 17(11):3402–3414, 2024.

Hongzhao Guan, Beste Basciftci, and Pascal Van Hentenryck. Heuristic algorithms for integrating latent demand into the design of large-scale on-demand multimodal transit systems. *Optimization and Control*, December 2022. doi: 10.48550/arXiv.2212.03460.

Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Transaction on Systems Science and Cybernetics*, 4(2):100–107, July 1968. doi: 10.1109/TSSC.1968.300136.

Andrew Holliday, Ahmed El-Geneidy, and Gregory Dudek. Learning heuristics for transit network design and improvement with deep reinforcement learning. *Machine Learning*, April 2024. doi: 10.48550/arXiv.2404.05894.

Richard E Korf. Depth-first iterative-deepening: An optimal admissible tree search. *Artifical Intelligence*, 27 (1):97–109, September 1985. doi: 10.1016/0004-3702(85)90084-0.

Aymen Lakehal, Sophie Lepreux, Christos Efstratiou, Christophe Kolski, and Pavlos Nicolaou. Spatial knowledge acquisition for pedestrian navigation: A comparative study between smartphones and ar glasses. *Information*, 14(7):353, June 2023. doi: 10.3390/info14070353.

Magdalen Dobson Manohar, Zheqi Shen, Guy E Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. Parlayann: Scalable and deterministic parallel graph- based approximate nearest neighbor search algorithms. *Information Retrieval*, February 2024. doi: 10.48550/arXiv.2305.04359.

Carlos E Mendoza-Ramírez, Juan C Tudon-Martinez, Luis C Félix-Herrán, Jorge de J Lozoya-Santos, and Adriana Vargas-Martínez. Augmented reality: Survey. *Applied Sciences*, 13(18):10491, September 2023. doi: 10.3390/app131810491.

Joshua Ofoeda, Richard Boateng, and John Effah. Application programming interface (api) research: A review of the past to inform the future. *Enterprise Information Systems*, 15(3):76–95, July 2019. doi: 10.4018/IJEIS.2019070105.

Richard S Sutton and Andrew G Barto. Reinforcement learning. 1998.

Ram K Upadhyay, Sunil K Sharma, and Vikram Kumar. Intelligent transportation system and advanced technology. 2024. doi: 10.1007/978-981-97-0515-3_11.

Yusheng Xiang, Kailun Liu, Tianqing Su, Jun Li, Shirui Ouyang, and Samuel S Mao. An extension of bim using ai: a multi working-machines pathfinding solution. *IEEE*, pages 124583–124599, September 2021. doi: 10.1109/ACCESS.2021.3110937.

Jin Y Yen. Finding the k shortest loopless paths in a network. *Management Science*, 17(11):712–716, July 1971.