CAMPUSGO: DEVELOPMENT OF A SMART MOBILE CAMPUS NAVIGATION SYSTEM

BY

INNOCENT CHIDIEBERE JOHNBOSCO

GOU/U22/CSC/830

A Project Submitted to the Department of Computer Science, Faculty of Computing and Information Technology, Godfrey Okoye University, Enugu State, in Partial Fulfillment of the Award of Bachelor of Science (B.Sc.) Degree in Computer Science.

SUPERVISOR: DR U.F. OKEBANAMA

JUNE, 2026

APPROVAL PAGE

This research, titled "CAMPUSGO: DEVELOPMENT OF A SMART MOBILE CAMPUS NAVIGATION SYSTEM," has been assessed and approved by the Department of Computer Science Committees in the Faculty of Computing and Information Technology, Godfrey Okoye University, Enugu, Nigeria.

Dr. Frank Okebanama	.....	.....
Supervisor	*Signature*	*Date*
External Examiner	.....	.....
External Examiner	*Signature*	*Date*
Dr. Frank Okebanama	.....	.....
HOD, Computer Science	*Signature*	*Date*
Prof. I. A. Ajah	.....	.....
Dean, Faculty of Computing and IT	*Signature*	*Date*

CERTIFICATION

In doing so, I confirm that I conducted this research myself and that most of the research was derived from my own observations and investigations. Therefore, in case it is proved that I cheated during this exercise, I accept the University's decision without question.

_____

INNOCENT CHIDIEBERE JOHNBOSCO

GOU/U22/CSC/830

DEDICATION

This project is dedicated to my parents, and all who supported my academic journey through their prayers, sacrifice, and encouragement made this academic journey possible.

ACKNOWLEDGEMENTS

I wish to express my sincere gratitude to my project supervisor, Dr U.F. Okebanama, for the guidance, patience, and constructive feedback provided throughout the course of this research. Your direction and expertise were invaluable to the successful completion of this project.

I also acknowledge the Head of Department and all the lecturers in the Faculty of Computing and Information Technology at Godfrey Okoye University for their commitment to academic excellence and the knowledge imparted throughout my undergraduate studies.

I also want to acknowledge my friends with their endless support and consistency in our school journey together- Isama, Gideon, Agbo, Gabriel, Paschal, Obi, Ojorbo. Thanks a lot my friends.

My heartfelt thanks go to my parents and family for their unwavering support, prayers, and encouragement. Their belief in my ability has been a constant source of motivation throughout this project.

Above all, I thank God Almighty, the fountain of wisdom and knowledge, without which none of this could have happened.

ABSTRACT

Campuses are large, spatially complicated systems in which movement poses a recurring challenge to freshmen, staff members, and visitors alike, more so for those within Nigerian educational establishments in the absence of digital tools for campus navigation. In this work, we propose CampusGO, a mobile intelligent campus navigation system created for Godfrey Okoye University (GOUNI), Enugu State, Nigeria, using React Native and JavaScript. We represent the environment of Godfrey Okoye University campus as a weighted graph in which 173 locations of the campus are considered nodes while 631 walkable pathways are regarded as weighted edges between the nodes based on their geographical proximity. We implement Dijkstra's algorithm for shortest path computations, with bidirectional Dijkstra's algorithm used as a routing engine. Two search frontiers start simultaneously at source and target nodes respectively and intersect at an intermediate node, considering less than half of the total number of nodes traversed by single direction Dijkstra's algorithm. The measured runtime for computing routes on the entire graph was 94 milliseconds on an actual physical Android 16 mobile device. The system uses Mapbox GL in order to render an interactive 3D campus map with extrusion of buildings, clustering for markers and calculated routes on top of the map. Real-time live GPS tracking is handled by Expo Location API along with Haversine snap-to-the nearest node functionality to allow routing from the users actual GPS location on the map. Turn-by-turn bearing classified directions are made based on the use of Douglas-Peucker algorithm for path smoothing and angular classification. If any deviation from the route exceeds 30 metres then an automatic re-routing will be carried out from the users GPS position. Campus data is hosted and managed with Firebase Cloud Firestore with a hardcoded fallback of 173 locations. The system design process used Object-Oriented Analysis and Design, with 8 UML artefacts and the use of the Waterfall system development model. Testing of the system was completed with 18 test cases using black box, white box, integration and performance testing all successfully passing.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 Background of the Study

Most universities tend to have large grounds that contain a number of buildings including lecture theatres, administrative buildings, student accommodation, libraries, laboratories, health centers, food outlets, chapels, sports grounds etc. Finding different locations on campus for the new students and even outsiders can be a challenging task, resulting in late arrivals and confusion (Naik et al., 2023). The scenario in Nigeria is worse with the universities having wide spread campus structures, un-named roads in the campus, and no common digital wayfinding platform (Hussaini et al., 2025; Gbadamosi & Aremu, 2020).

In the past, universities depended on the traditional approach to wayfinding by making use of static tools such as signposts, physical maps, and instructions provided by university officials or other university students. But, these methods of wayfinding have significant drawbacks as they cannot be timely and flexible due to the instant map changes after any change in layout inside the campus, damage of signposts and inconsistency in directions given (Raza et al., 2020). Such a case of static way finding system does not have timeliness, accuracy and interactivity and thus cannot prove to be a proper way finding solution for a modern-day university community. Godfrey Okoye University (GOUNI) in Enugu State, Nigeria is a good example of this problem, with 173 unique places in various buildings in the campus.

The wide use of smartphones and mobile internet by students in Nigerian universities presents unique opportunities in the area of location-based services and navigation (Garg et al., 2023). Urban navigation has been transformed by mobile mapping and navigation solutions such as Google Maps. However, these applications can provide detailed navigation in town and city, but are too general to provide detailed campus-level information that is essential for efficient campus navigation. These applications are not able to recognise the paths within a campus, different access points of buildings or the specific relationship between the various locations within a university

environment (Choudhari et al., 2024). If you are asked about a particular office within a department on the GOUNI campus, these applications will not be very useful.

While campus-based indoor navigation apps have been created in some universities, especially those from developed nations, these systems are limited due to their dependence on certain proprietary mapping tools, costly physical components such as Bluetooth beacons, Internet of Things (IoT), or augmented reality markers, or being designed in a manner that limits flexibility in adapting them to use in different institutions (Abbas Helmi et al., 2022; Naik et al., 2023; Zhou et al., 2021). In developing nations like Nigeria, however, such systems are mostly non-existent, leaving university campuses reliant exclusively on traditional methods (Hassan et al., 2023; Manning & Hosein, 2021). It is clear from scholarly studies that the best navigation system for poor-resourced institutions should involve zero dependence on any specific hardware infrastructure and utilize pre-set walkable paths mapped in graphs (Parmanand et al., 2024; Raza et al., 2020).

The data on campus location at Godfrey Okoye University was obtained through field survey which was physically carried out by the author of the system. Using a GPS-equipped device, the location of 173 campus sites, categorized under eight facility types, were geocoded. A total of 631 pathway segments (walkable) were measured and verified on the ground to be used for routing in this system.

The CampusGO project proposes using a smart mobile campus navigation system developed using React Native and JavaScript (Ramachandrappa, 2024) to address navigation challenges on the campus of GOUNI. Mapbox GL is used for rendering maps with interactive 3D modeling, buildings extruded to 3D, and group marker clustering. In a campus environment, the graph (or network) is a weighted graph of 173 nodes (survey points) and 631 weighted edges (tracings between nodes). To implement the shortest route search algorithm, the bidirectional Dijkstra's algorithm is used, which takes two frontiers of searches in opposite directions towards each other from the start point and the end point respectively, respectively, and reduces the number of nodes to be explored by half (Pohl, 1971; Kaindl & Kainz, 1997). The current location of the user is displayed on the map as a rotating directional cone, using the Haversine origin recognition, with the Expo Location API. The Campus data is stored and managed in a cloud-based backend called

Firebase Cloud Firestore which provides remote data updates without app rebuild (Madaminov & Allaberganova, 2023; Semma et al., 2023).

1.2 Statement of the Problem

Despite advancements of mobile technology and digital mapping, some Nigerian Universities do not have a navigation system that is specifically designed for them.

Absence of an online university wayfinder: Freshmen at Godfrey Okoye University find themselves faced with immense trouble in terms of finding their way around different sections such as the faculty buildings, exam halls, laboratories, health centers, and other sections within the institution. With the large size of the campus, the non-existence of road names, and total lack of an online wayfinding system, the students always end up being late for classes and exams.

The lack of specificity in the existing navigation apps: Google Maps and Apple Maps, which are good for navigating in the city, are not specific enough in terms of campus navigation. They will not know how to navigate their way within the campus, will not know how to identify different entrances to buildings and will not know how to recognize different points within the campus that may appear identical but are used for a different purpose (Choudhari et al., 2024).

Static and interactive digital maps: At a Nigerian institution where digital maps are available, the maps are presented as a static map that is posted on the institutions official website, but does not offer any routing, interactivity or mobile optimization feature (Gbadamosi & Aremu, 2020; Vilaspure et al., 2024).

Dependency of current campus navigational systems on hardware: Most of the campus-based navigation systems that have been discussed in academic literature tend to cater to resourceful organizations, making use of expensive hardware, including IoT sensors, BLE beaconing technology, or AR tags, which are not economically feasible for the majority of Nigerian universities (Abbas Helmi et al., 2022; Naik et al., 2023; Prandi et al., 2021).

1.3 Aim and Objectives of the Study

The study aims to design and develop CampusGO, a smart mobile campus navigation system for Godfrey Okoye University. Below are some specific objectives of this project:

1. Analyze the GOUNI campus environment, represent the same on a weighted graph and use Bidirectional Dijkstra algorithm for the same.
2. Combine live GPS location with snap-to-nearest node detection based on Haversine formula and turn-by-turn navigation according to bearing with off route rerouting capability.
3. Create a 3D map of the GOUNI campus environment using Mapbox GL with building extrusions, cluster marker points, and route overlay calculations.
4. Use the Firebase Firestore platform for remote data management of the campus and path related details.
5. Test the correctness and performance of the implemented system using different functional, white box, and performance test cases.

1.4 Significance of the Study

There are a number of reasons why the development of CampusGO is important, beyond just the topic of navigating the university campus.

Benefits for GOUNI students and employees: This system solves the issue of navigation around the university's grounds that has been faced by many students since its establishment. Students are able to navigate to various facilities on the grounds using their GPS location, thereby avoiding additional stress and being late (Naik et al., 2023).

Facility information in Firebase Firestore: The information about campus facilities is hosted in Firebase Firestore, providing the ability for the administrators to change facility records remotely and without having to reissue new physical maps or rebuilding the application (Madaminov & Allaberganova, 2023; Semma et al., 2023).

Proof of computer science education applicability: The project uses graph theory and algorithms, computer programming, cloud-based systems, and map technologies to create a solution to an actual real-world problem (Hassan et al., 2023).

Algorithmic Contribution to Measurement: The utilization of Bidirectional Dijkstra's Algorithm results in an empirically demonstrated performance enhancement from using regular one-way Dijkstra's algorithm, which involves exploring roughly half the number of nodes in the

actual 173-node graph of the GOUNI campus (Pohl, 1971; Kaindl & Kainz, 1997; Parmanand et al., 2024).

Replicatable Model for Nigerian Higher Education Institutions: The integration of React Native, Mapbox GL, and Firebase Firestore into CampusGO has provided proof of concept showing that a robust campus navigation app can be created without relying on any proprietary software, expensive equipment, or large teams, therefore making it replicable for other Nigerian universities (Garg et al., 2023; Ramachandrappa, 2024; Gbadamosi & Aremu, 2020).

Contribution to New Dataset of Campus Locations: An independently gathered dataset consisting of GPS coordinates of 173 Godfrey Okoye University locations, as well as photographs and hand-traced 631 routes, represents a unique contribution to the field.

1.5 Scope of the Study

The features explored in this study include: mapping and digitising 173 locations of functional areas in Godfrey Okoye University, out of which 8 categories are mapped, namely, Academic, Administration, Hostel, Food & Retail, Health, Shops, Church, Sports; 631 paths are mapped and digitised within Godfrey Okoye University campus as a weighted navigable graph; mapping and rendering of interactive 3D campus map using Mapbox GL with building extrusions and cluster-based markers; routing using the Bidirectional Dijkstra's algorithm on the navigable GeoJSON graph of paths; enabling GPS tracking with real-time direction indicators and detecting the nearest path node to the GPS location using the Haversine formula; providing turn-by-turn directions and off-route detection, with automatic route recalculations; off-line mode and fallback mechanism; cloud storage with long-polling transport of data on Firebase Firestore; and light and dark themes, along with searching within the 8 functional areas of the campus.

This research does not cover the application of navigation at ground level indoors, real-time tracking of crowds or vehicles, role-based access management, installation across multiple campuses, or voice-aided navigation. Although they could be useful features, these are beyond the scope of this particular project, and will be considered future works discussed in Chapter Five.

1.6 Limitations of the Study

Some shortcomings have been noticed during the process of developing this system and are highlighted below for a comprehensive understanding of its capabilities and shortcomings.

GPS Accuracy Dependence: The live location tracking functionality relies heavily on the accuracy of the GPS hardware installed in the device. There might be situations where the accuracy of GPS signals is low, resulting in incorrect identification of routing origins because of the snap-to-the-nearest node algorithm employed by the system. This is a hardware-related problem rather than software-related (Raza et al., 2020).

Internet Connectivity: An internet connection is required for the extraction of the campus data from Firestore. A hardcoded static set of 173 nodes is offered to allow for graceful degradation in the case that the app is used without an internet connection; however, Firestore synchronization cannot be done offline (Madaminov & Allaberganova, 2023).

Static Routing Computation: If the current location deviates from the pre-planned path by 30 metres, then off-route computation re-routes; however, there is no step-by-step calculation and notification to account for the current position of the user.

22 Placeholder Coordinates: 22 of the total 173 campus locations had placeholder coordinates at the time of data collection as their exact GPS coordinates couldn't be validated. These locations have a needs Coordinates property marked in Firestore and need on-campus GPS verification.

One Campus Constraint: The CampusGO application has been developed for use within the Godfrey Okoye University campus only. Using the system in another campus would mean doing the field survey for the physical space again, mapping all locations and pathway segments, and reseeding Firestore. There isn't an automatic onboarding procedure for campuses currently available**.**

Graph Model Simplification: The campus graph in the present study is a simplified model of the actual campus environment. Elements like elevation of paths, path width, temporary blockages in pathways, and path accessibility to physically challenged individuals haven't been considered during the route computation (Parmanand et al., 2024).

1.7 Definition of Terms

Definitions of the terms below are necessary because these are key concepts used in this project:

CampusGO: The intelligent mobile application that facilitates campus navigation and was designed and created in this project for Godfrey Okoye University, Enugu.

React Native: A framework that provides a way of developing mobile applications across multiple platforms using JavaScript. The tool makes it possible for programmers to create applications that work natively on the Android and iOS operating systems from one code base (Garg et al., 2023).

Mapbox GL: A performance-based library that includes features such as GPU accelerated vector tiles rendering, three-dimensional buildings extrusion through the FillExtrusionLayer, clustering, GeoJSON layers and sources, and markers grouping.

Bi-directional Dijkstra's Algorithm: It is an enhancement of Dijkstra's original shortest path algorithm. Instead of one search from source to destination node, it simultaneously conducts a search from the source to the meeting point, as well as from the destination to the same meeting point. It only examines about half the number of nodes compared to the traditional Dijkstra algorithm because of which it performs route computations in much less time (Pohl, 1971; Kaindl & Kainz, 1997).

Haversine Formula: A formula used to calculate the distance between two geographical coordinate pairs (latitude and longitude pairs) on the earth. In our app CampusGO, the Haversine formula is used to calculate the weight of each edge in the campus routing graph. Here the weight of each edge denotes the walking distance between two adjacent path nodes in meters.

Weighted Graph: The data structure which is mathematical in nature, consisting of finite vertices and edges between them, where each edge has a value called a weight associated with it. In CampusGO, the number of nodes is 173, which are campus locations while the edges are 631, and their values are measured in terms of distance in metres (Hassan et al., 2023).

GeoJSON: A specification for encoding geographic data in JSON format that complies with the RFC 7946. CampusGO makes use of LineString data from campusPaths.json file, which encodes all walkable campus paths in the campus graph, and polygons in buildings.json file to encode

buildings' boundaries in three dimensions. Coordinates of each LineString feature are real-world points, while consecutive coordinates define the edge structure of the routing graph.

GPS Snap-to-Path: This is the act of determining the closest point on the graph of paths in GeoJSON to the current location of the user, which is taken to be the automatically generated routing origin. Calculated via a flat-projection approximation formula based on degrees multiplied by 111,320 meters per degree, scaled down according to the latitude. Provides more efficient calculation on the dense graph of campus pathways while retaining accuracy at campus scale. Guarantees that routes originate from the real position of the user along the closest pathway, as opposed to a manually specified origin.

Douglas-Peucker Line Simplification: An algorithm that iteratively removes redundant points along a polyline to simplify it. Used in CampusGO with a 0.000025 degree tolerance to render smooth route lines.

Firebase Firestore: A Cloud NoSQL document database service designed by Google and employed in CampusGO for storing 173 campus locations and 631 pathways. It uses serverless infrastructure with long-polling transport in Android to update data remotely without rebuilding the application (Madaminov & Allaberganova, 2023; Semma et al., 2023).

Expo Location API: Expo SDK module used in CampusGO to utilize device's GPS sensors for obtaining real-time location coordinates through watchPositionAsync method and device heading via watchHeadingAsync method to change the cone direction.

Marker Clustering: The clustering together of adjacent map markers to form one circle that shows the numerical number of markers in the cluster, where these individual markers are expanded based on zooming in to a particular zoom level.

FillExtrusionLayer: A layer for Mapbox GL that renders polygon features as 3D extrusions using the adjustable values of height and base height of the extrusion, which is utilized by the application Campus GO in rendering 3D buildings on the map.

Off-route Detection: The computational technique that measures the distance between the user and the closest point on the active route line, perpendicular to it. Once this distance becomes larger than 30 meters, a new shortest route is automatically calculated by the system from the user’s current GPS location to the initial destination.

Object-oriented analysis and design (OOAD): A software engineering methodology where components of the system are represented as objects with attributes and behaviors, and their interactions through use cases, activity diagrams, class diagrams, and sequence diagrams are depicted using UML notation (Booch et al., 2005).

Waterfall Model: A linear SDLC process where every development step such as requirements, system design, implementation, testing, and installation must be fully completed before proceeding with the next development step (Sommerville, 2016).

Bearing: The direction from one point of the earth to another in terms of degrees that range between 0° and 360° measured clockwise from north. Used by CampusGO in their RouteSteps component to determine the direction of consecutive path segments as follows: straight, slight-left, left, slight-right, and right turns.

CHAPTER TWO

LITERATURE REVIEW

2.1 Introduction

The current chapter consists of an extensive literature review of all the literature pertaining to the development of CampusGO. The review has been organised into six themes covering the following key aspects of the system being developed: theoretical foundations of campus navigation, graph theory and shortest path algorithms, use of GeoJSON in route planning, GPS and positioning technologies, mobile application development platforms, and finally cloud computing systems. Thereafter follows a critical literature review of twenty selected works falling into five broad categories, followed by a comparison table and discussion of the identified research gap. All primary literature referred to in this chapter has been obtained from reputable journals and conference proceedings dating back from 2020 to 2025. Searches were conducted in database platforms including IEEE Xplore, Springer Nature, MDPI, and Google Scholar, with keyword searches relating to campus navigation, mobile wayfinding, graph routing, mobile development, and cloud backend systems.

2.2 Theoretical Background

2.2.1 Campus Navigation and Wayfinding

Campuses of universities are some of the most complicated spaces where people navigate frequently. Various studies have highlighted the issues related to wayfinding in campuses in diverse geographic and institutional settings (Naik et al., 2023; Raza et al., 2020). Conventional means of wayfinding such as physical signposts, map guides, and guidance from security guards or other students have been found to be inherently flawed in modern campuses, as layouts of facilities are constantly changing and buildings have inconsistent labelling, not to mention that first-timers do not possess prior knowledge of the spatial arrangement. Digital navigational aids have thus proven to be the most realistic options, especially since smartphones have now reached an almost universal level of prevalence among university students in Nigeria. With respect to Godfrey Okoye

University, which comprises 173 named places falling under eight facility types, the lack of any digital wayfinding system leads to numerous challenges.

2.2.2 Graph Theory and Spatial Modelling

The mathematics underlying route planning within spatial networks is provided by graph theory (Hassan et al., 2023). The structure of a graph G=(V,E) comprises vertices V, which can be defined as a finite set of nodes, and the edges E between the vertices. In the case where there are numbers assigned to each edge of the graph, the graph takes on a form called the weighted graph, where the numerical value corresponds to the walking distance in meters between two connected vertices. Shortest path determination represents the key computational problem associated with route planning.

Campus Navigation boils down to a problem of an undirected weighted graph. The key intersections or important points along paths are considered nodes, while the paths themselves connecting two nodes are considered weighted edges. In Campus GO, the 173 actual campus locations on Godfrey Okoye University are the nodes of the graph, while 631 surveyed pathway lengths are the weighted edges, with each of the weights being Haversine distances in meters between adjacent nodes.

2.2.3 Shortest Path Algorithms

There exist several proven algorithms which can solve the shortest path problem on weighted graphs. Each of them has different computational properties and requirements to implementation.

The algorithm presented by Edsger W. Dijkstra in 1959 involves selecting the unexplored vertex with the minimum cumulative cost and relaxing its adjacent edges. It produces an optimal solution for every weighted graph with non-negative edge weights and executes in O((V + E) log V) complexity when using a binary min-heap priority queue (Parmanand et al., 2024). This algorithm is the most commonly used one to find shortest paths on navigation graphs, and its optimality is proved by several research works (Hassan et al., 2023; Wayahdi et al., 2021).

Bidirectional Dijkstra builds on the traditional approach by conducting a forward search starting from the source node and a backward search starting from the destination node at the same time. Both searches proceed simultaneously and use separate priority queues, arrays containing distances, and visited nodes' sets. The algorithm halts when it finds a possible meeting node whose

sum of distances from forward search and backward search meets or exceeds the minimum combined distance (bestCost). The route will be found by combining a sub-path generated through the forward search with the sub-path generated via the backward search between the source node and the meeting node, as well as the meeting node and the destination node respectively. In theory, bidirectional search examines half as many nodes as traditional Dijkstra's Algorithm does to find the answer to the same question. Thus, bidirectional search performs twice as fast as traditional Dijkstra's Algorithm in large graphs (Pohl, 1971; Kaindl & Kainz, 1997). CampusGO uses Bidirectional Dijkstra as its only algorithm.

In an A* search algorithm, heuristics are used in addition to Dijkstra's technique of relaxing edges until we reach our destination node. A* uses a function called h(n), which calculates the estimate of the cost left to go from node n to the destination node. The main feature about the heuristic function is that it should be admissible for A* to produce optimal results, which means it never overestimates the actual cost left from the current state to the goal (Sathvik & Patil, 2021).

However, there are reasons that make A* unsuitable for CampusGO, which arise from the particularities of this project. Firstly, it should be noted that the success of A* depends largely on how well it is guided by a heuristic. Since the routing graph of CampusGO consists of unprocessed GPS-traced path coordinates instead of named points, the nodes are coordinate pairs along paths, meaning that the straight-line Haversine distance from any node to the target point may be far from being proportional to the actual distance of travel within the constraints of campus roads. Using a bad heuristic will make A* perform similarly to regular Dijkstra's Algorithm with no apparent benefits. Secondly, the CampusGO campus graph includes 173 nodes and 631 edges, making it a rather modestly sized graph when compared to what navigation systems work with. In fact, bidirectional Dijkstra already manages to complete the search process in less than 200 ms, well under the performance criterion formulated in Chapter Three. This time efficiency of A* becomes particularly relevant when the graph is large, where search without heuristics becomes computationally expensive; for a campus level graph, there is practically no improvement over the bidirectional approach. Third, Bidirectional Dijkstra provides guarantee of optimality of the path for a graph with non-negative weights, not needing any heuristic; this makes it an easier approach to implement with similar performance. All these three reasons: heuristic uncertainty in the context

of heuristic graphs, small size of the campus graph, and optimality without using a complex heuristic justify the use of Bidirectional Dijkstra only.

2.2.4 GeoJSON as a Routing Medium

GeoJSON is an open standard format for representing geographic data using JSON structures. GeoJSON supports various geometric objects, such as Point, LineString, Polygon, and FeatureCollection, among others. In particular, when we refer to navigation applications, the LineString objects become extremely useful since they consist of sequential geographic points that describe a line on the ground.

By using LineString objects to store walkways, the routing graph can be derived automatically based on digitised geometries without any need to create an additional database structure. The coordinates within the LineString objects will serve as the actual positions on walkable paths, where consecutive coordinates will form edges in the routing graph. Using this method offers greater precision compared to a hand-coded graph, since the exact path geometry will be defined, unlike when approximating the route using straight lines between nodes which are assigned by hand. All information regarding campus path geometry is kept in the campusPaths.json file, which is a GeoJSON Feature Collection containing all LineStrings that correspond to walkable paths on Godfrey Okoye University campus. buildings.json contains information about building footprints stored as Polygon features with two additional height properties. These Polygon objects are later visualised on the interactive map through use of the FillExtrusionLayer by Mapbox GL.

2.2.5 GPS Positioning and Location Services

The GPS system allows a mobile phone to calculate geographic coordinates through the receipt of signals sent out by satellite systems. For the use of navigation systems on mobile phones, the GPS coordinates of the user are considered to be the starting point for the route. The issue in the process of graph navigation is that the GPS location of the user never corresponds to any nodes within the routing graph. The method usually used to solve this problem is snapping to the nearest node.

In CampusGO, the snapping mechanism to the nearest path node uses the flat-projection distance approximation technique instead of using the complete Haversine algorithm. The distance is approximated by computing distance_x as (node.longitude – userLng) * 111320 * cos(userLat * pi / 180), and distance_y as (node.latitude – userLat) * 111320, then adding square roots of dx^2 +

dy^2. The use of the flat-projection distance approximation technique is very accurate since this application targets a college campus area, where the maximum possible distance between any two points is not more than 2 kilometers. Device heading can be obtained from the Expo Location API using the function watchHeadingAsync, which allows rotation of the directional cone marker based on user's facing direction, just like seen on consumer navigation apps such as Google Maps (Raza et al., 2020).

2.2.6 Mobile Application Development Frameworks

In the last ten years, there has been fast progress in the field of mobile application development due to the rise in the use of smartphones along with an increased need for location-dependent services. The issue in developing mobile apps was the necessity to build apps that would run on different operating systems, mainly Android and iOS, each with its own codebase and separate team. This problem was solved through the introduction of cross-platform frameworks.

A metric-based performance evaluation conducted by Garg et al. (2023) showed that React Native is capable of providing good performance for the vast majority of mobile applications, especially in terms of fast rendering and reusability of components. According to this paper, despite the small additional overhead caused by React Native compared to native code, the use of React Native is still more justified due to its great advantages in terms of development time savings and simpler code base. Another advantage of React Native is proven by Ramachandrappa (2024), who shows that the use of React Native allows decreasing the time and costs of software development by about 30% compared to developing two applications for two platforms separately.

2.2.7 Cloud Backend and Mapping Technologies

The phrase cloud computing implies the offering of computing services like storage, database, and processing via the internet. The term cloud is associated with mobile development in that it provides the ability for an application to be light on the device but able to access backend services remotely.

The research carried out by Madaminov and Allaberganova (2023) on the architecture of Firebase in the current mobile apps has revealed that Firebase has the potential of reducing the amount of coding in the back end due to the feature of real-time sync and the serverless nature of the app. The applicability of the above observation to the current task is that CampusGO requires a way of

updating data without having the need to update the whole application. Saraf et al. (2022) have reviewed Firebase as a platform for developing mobile apps and concluded that Cloud Firestore is the best database for new projects based on its ability to handle complex NoSQL structures. In addition, _firestore performance optimization has been discussed in the works of Semma et al. (2023), who argue that good performance can be obtained using structured collection with proper indexing.

Mapbox GL includes hardware-accelerated vector tile rendering, including 3D building extrusion through FillExtrusionLayer, data-driven styling capabilities, and marker clustering that uses a native approach. Among these features, 3D building extrusion and native clustering are not supported in any other open-source mapping solutions like MapLibre in the stability and performance needed for the navigation app.

2.3 Review of Related Works

Below are described twenty works related to various aspects of the campus navigation problem that the proposed software application, CampusGO, aims to address. All these sources were analyzed from the viewpoint of their approaches, conclusions and certain drawbacks that influenced the development of the current project in particular ways. The works are divided into five groups.

2.3.1 Navigation Systems Based on Augmented Reality

In a study conducted by Choudhari et al. (2024), an Android-based campus navigation app, termed Campus Compass, was created. In this system, augmented reality technology was incorporated into a shortest path search algorithm. In doing so, information regarding buildings and directions would be superimposed on a live camera feed, enabling the user to determine where each department is located. It was evident that such guidance provided better navigation within multi-building areas compared to conventional methods; however, since the operation of the system depended on physical markers embedded in critical points throughout the campus, it became heavily hardware-dependent. Moreover, the system's performance was significantly affected under low-light conditions.

The researchers, Zhou et al. (2021), created an augmented reality campus navigation system using the technology of ARCore. ARCore marker recognition along with visual-inertial ranging were used to facilitate both indoor and outdoor navigation. Though the technology was advanced, the

system had limitations due to extensive need for hardware calibration, a 3D model of the campus to be created in advance, and skilled developers. According to the authors, the system would not be feasible to use anywhere else owing to its complete dependence on modeling.

Dharaskar et al. (2023) proposed an indoor navigation system that uses augmented reality with semantic web technology at a campus. This created a 3D map of the campus along with the context details of rooms. There was no inclusion of any routing algorithm in the paper, making it an information overlay alone and not navigation from one point to another. There was no use of cloud-based data management as well.

2.3.2 IoT and GPS Navigation Systems

Naik et al. (2023) suggested a smart campus navigation system which involved the combination of IoT sensors along with GPS technology in determining location outside, combined with vocal guidance for visually impaired individuals. The use of IoT systems ensured real-time data availability on occupancy. Nevertheless, the use of IoT devices physically placed in strategic positions across the campus was expensive and difficult to manage, making it impractical for developing nations' universities.

Prandi et al. (2021) explored the use of IoT technologies in creating inclusive ways to navigate the campuses of universities, particularly focusing on making navigation accessible to physically disabled students. The IoT-enabled navigation solution gathered data from the environment using sensors to create customized navigation paths for the users. Although the experiment proved the effectiveness of IoT-enabled navigation in increasing usability for disabled individuals, the dependence on a dense sensor infrastructure presented the same problem of infrastructure dependency that is mentioned by Naik et al. (2023).

Tiwari and Verma (2021) developed a mobile campus navigation app for Android devices that employed GPS technology, camera sensors, and inertial sensors to find and show the quickest route among campus buildings. Though working fine in open places outdoors, a weak GPS signal in the building interiors and in dense construction areas posed a problem with routing precision. Tiwari and Verma acknowledged that the system did not work with indoor navigation and relied heavily on the availability of GPS signals, something not easily achievable within campuses.

A navigation tool based on the Android app framework was created by Manning & Hosein (2021), which used Google Maps API as its base mapping engine for a third-level educational institution in the Caribbean region. Although functional for navigation outside of the building, the reliance on Google Maps API caused issues related to proprietary licensing and inability to map routes within the campus, which did not appear in Google's licensed database. There was no possibility to use a custom data layer to incorporate institutional data such as departments and facility categories.

2.3.3 Studies Relating to Graphs and Algorithms

The study by Gbadamosi and Aremu (2020) examined a new approach to implementing Dijkstra's algorithm by creating more than one path from weighted graphs where some paths are associated with excessively large costs. This investigation was carried out within the framework of routing on road networks and showed that an implementation of Dijkstra algorithm modifications was able to provide several possible paths instead of just one best solution. However, although theoretical in its nature, the research did not involve mobile application development, map drawing, or using GPS technology.

Hassan et al. (2023) offered an elaborate study on Dijkstra's Algorithm in respect to the calculation of shortest paths in weighted directed graphs. They found out that the algorithm yielded optimal results for non-negative edge weights and showed how implementation of priority queues increased the efficiency of Dijkstra's Algorithm to O((V+E)logV). The theoretical conclusions have direct bearing on the algorithmic development of CampusGO. There was no mobile application developed.

An optimisation research by Parmanand et al. (2024) assessed Dijkstra's algorithm in relation to the effects different data structures have on its performance at large scales. Based on their findings, they concluded that Dijkstra's binary heap implementations represent the best compromise between ease of coding and efficiency of computations and reaffirmed that Dijkstra's algorithm should be used whenever there is a need for static navigation graphs with non-negative weights.

Wayahdi et al. (2021) investigated Greedy, A*, and Dijkstra's algorithms' ability to find paths in graphs and concluded that the latter can always find the optimal paths, while the former is faster yet often yields results sub-optimal. The findings of the research further affirm the suitability of Dijkstra's algorithm in use cases where correctness outweighs performance; therefore, Dijkstra's is

chosen in CampusGO. Gryzun et al. (2023) designed a web-based information system for campus navigations using SVG maps and the Floyd-Warshall algorithm for all-pairs paths' calculation. Although the approach is efficient for pre-calculating the full table of paths, its $O(V^3)$ time complexity makes it inefficient when dealing with larger graphs. Moreover, the system requires a web browser, so it could not be used effectively on mobile devices.

2.3.4 Navigation Tools for Mobile and Cross-Platforms

Li & Yin (2024) introduced CampNav, an indoor building and campus navigation system that employed routing based on graph theory together with the scanning of QR codes for indoor localisation purposes. Routes were indicated on the 2D floor plan, while multi-level navigation was enabled by visual indications of floors transitions. Though technologically robust, the introduction of QR codes required that physical QR codes were installed at specific indoor points, creating a necessity for additional hardware. Cloud-based data management was not included; thus, any campus update necessitated a new app build.

Find 4 Me is the name given to the indoor navigation app that Abbas Helmi et al. created in Management and Science University by utilizing Bluetooth Low Energy beacons installed all over the building to calculate the proximity of users to fixed reference points. Although the results of user assessment demonstrated increased orientation time for freshmen, the BLE beacon system required purchasing, setup, and maintenance processes that made it expensive and impractical.

Vilaspure et al. offered PathPilot, a navigation app for a college with an interactive 3D map made in Blender and integrated with a local SQLite database. It allowed users to look up campus buildings and visualize them on a 3D map layer. Nevertheless, there was neither any routing algorithm nor cloud backend or any path calculation functionality implemented in the app. Moreover, all campus information was stored on the client side. Hence, the entire application should be rebuilt each time there was any change in data.

GSU Connect was designed by Hussaini et al. (2025) as a mobile geographic information system for navigating around Gombe State University, Nigeria, developed based on Flutter, Supabase, and OpenStreetMaps technologies. The design entailed the creation of a rule-based conversational assistant that would allow requesting location-related information in natural language mode. Although extremely topical for Nigeria, this work cannot be compared to the proposed app because

it had a conversational assistant rather than a graph-based routing engine; thus, route calculation was not optimised algorithmically and had no GPS snap-to-path feature included.

A dual-mode campus information system for Android and web versions was created by Feng et al. (2022). This application allowed students to browse information about facilities available at the campus and also showed their exact locations on a map included in the software. Although the application can be regarded as a typical campus information resource for the previous generation, it was not equipped with the routing algorithm, GPS support or navigation capabilities.

2.3.5 Indoor Positioning and Review Literature

Raza et al. (2020) conducted a systematic review on the current state of indoor positioning technology aimed at college campuses and included a discussion of different approaches to indoor positioning including Wi-Fi Fingerprinting, Bluetooth Low Energy, Inertial Measurement Unit and Visible Light Positioning technologies. It was found that despite the accuracy of hardware-based positioning technologies, they all have one common drawback in terms of deployment – they all require additional infrastructure or a survey of the location prior to usage. The authors conclude that the most practical way of implementing indoor positioning technology is an infrastructure-less method using predefined graphs and map.

Ali et al. (2022) investigated the Wi-Fi fingerprinting technology in positioning applications for indoor navigation, considering RSS, CSI, and machine learning classification algorithms as well. According to Ali et al., fingerprinting technologies provide 2–5 meters accuracy of position estimation in indoor environments in case of favorable conditions; however, this technology requires intensive calibration efforts and becomes less accurate in terms of changes in furniture placement or number of users in the environment. As an alternative, for the case of campus navigation application, graph-based technologies can be used more effectively than signal-based ones, which supports the approach implemented by CampusGO.

The conversational AI chatbot introduced by Ritawari et al. (2024) for navigating college campuses involved a rule-based natural language interface that could take user inputs for directions and present spatial data through texts. Conversational interface was found to be less frustrating for users compared to menu-driven search interface, especially for users not acquainted with the

jargon used on campus. However, the proposed system did not use any graph routing algorithms, nor could it generate maps.

2.4 Comparative Analysis of Related Works

The following Table 2.1 shows a systematic comparison between the twenty systems and studies discussed against CampusGO on four important parameters, namely technology, platform, routing algorithm, and most importantly the limitations of each of them.

Table 2.1: Comparative Summary of Related Campus Navigation Systems and Studies

Study / System	Technology	Platform	Algorithm	Key Limitation
Choudhari et al. (2024)	AR + Android SDK	Android	Shortest Path	Dependent on physical AR markers; lack of cloud support; poorly lit environment reduces effectiveness
Naik et al. (2023)	IoT + GPS sensors	Web/Mobile	GPS routing	Dependent on IoT infrastructure; not applicable for Nigerian universities
Gbadamosi & Aremu (2020)	Simulation on graph	Simulation alone	modified Dijkstra algorithm	No mobile interface, map visualization, or GPS, only simulation
Hassan et al. (2023)	Application of Graph Theory	Theory	Dijkstra	Only theoretical, no system deployed
Feng et al. (2022)	MVC Android + Web	Android + Web	Not applicable	Information portal system only; no route finding capacity; no data handling in the cloud
Hussaini et al. (2025)	Flutter + Supabase + OpenStreetMap	Android + iOS	Rule-based NLP	Conversational agent system only; without graph navigation and geolocation matching
Li & Yin	Graph + QR	Android +	Graph-based	system requiring physical QR

Study / System	Technology	Platform	Algorithm	Key Limitation
(2024)	code	iOS		codes; no cloud system; hardware-based system
Zhou et al. (2021)	ARCore + Unity3D	Android	Visual Inertial	system requiring AR hardware calibration and pre-made 3D map; no scalability
Abbas Helmi et al. (2022)	Bluetooth beacons	Android	Proximity-based	system requiring BLE network deployment; costly for limited resource organizations
Gryzun et al. (2023)	Web + SVG maps	Web browser only	Floyd-Warshall	Web only; $O(V^3)$ complexity unsuitable for mobile single-source queries
Prandi et al. (2021)	IoT + Sensors	Mobile	IoT-guided routing	Dense sensor network required; tested only in well-funded European context
Dharaskar et al. (2023)	AR + 3D model	Android	None (AR overlay)	No route computation; AR information overlay only; no cloud backend
Vilaspure et al. (2024)	SQLite + 3D map	Android	None	Local database only; no cloud; no routing algorithm; informational only
Tiwari & Verma (2021)	GPS + Camera	Android	GPS-based	GPS degrades inside buildings; no graph model; no off-route rerouting
Raza et al. (2020)	Wi-Fi / BLE / IMU	Mobile (review)	Various	Review only; all reviewed systems require dedicated hardware infrastructure
Ali et al. (2022)	Wi-Fi Fingerprinting	Android	RSSI-based	Extensive calibration required; accuracy degrades with environmental change

Study / System	Technology	Platform	Algorithm	Key Limitation
Manning & Hosein (2021)	Google Maps API	Android	Google routing	Proprietary API; internal campus paths absent from Google dataset
Parmanand et al. (2024)	Graph algorithm analysis	Theoretical	Dijkstra optimized	Algorithm study only; no mobile system or campus deployment
Wayahdi et al. (2021)	Graph algorithm comparison	Theoretical	Dijkstra / A* / Greedy	Simulation study only; no mobile system, map rendering, or GPS integration
Ritawari et al. (2024)	AI chatbot + GIS	Mobile web	NLP / Rule-based	No graph routing; conversational only; no interactive map rendering

These four limitations that are seen consistently across twenty systems and studies help in defining the research gap that the current project seeks to address. To start with, the bulk of the reviewed systems use special purpose physical equipment such as AR markers, IoT sensors, BLE beacons, or QR codes which makes them difficult to install, maintain, and use in resource-constrained Nigerian university campuses. Secondly, those that achieve the mobile deployment feature usually employ proprietary mapping APIs like Google Maps thus making it difficult to include additional campus-specific routes due to license restrictions. In most cases, routing functionality is implemented using standard one-way search like Dijkstra's algorithm or without any algorithmic implementation at all. The last limitation involves the lack of a single solution in the reviewed systems capable of implementing live GPS snap-to-path routing with off-route detection and automatic rerouting.

2.5 Research Gap

This chapter has evaluated a total of twenty pieces of literature to create an overview of where current campus navigation research stands and what technologies exist in that regard. In terms of the five different reviews, namely those related to augmented reality systems, Internet-of-Things/GPS systems, graph-based/algorithm approaches, mobile/cross-platform systems, and indoor positioning systems, certain common limitations have been identified.

Hardware requirements have proven to be among the most common of those limitations. Augmented reality systems, Internet-of-Things systems, or systems utilizing Bluetooth beacons require hardware implementation, which in itself involves high costs and cannot be afforded by the majority of Nigerian universities. Similarly, GPS-based systems like the one proposed by Tiwari and Verma (2021) perform poorly indoors and in dense urban areas. This leaves the majority of discussed systems as inadequate solutions in terms of implementation at Godfrey Okoye University.

The second gap is that most systems do not have cloud-based database management services. Data for campuses is usually stored in SQLite databases or application bundles, so any modification to the campus map will require building a new version of the app. Hence, these solutions cannot be used in environments where there are regular changes to the campus infrastructure and the ability to modify the data remotely is desired by the administrators.

The third gap is that none of the twenty solutions implements the functionality in a cross-platform mobile application using open-source or cheap mapping software together with the graph-based routing. These applications either use proprietary software like Google Maps API, or are made only for the Android platform. There is no solution that integrates bidirectional Dijkstra algorithm with a hand-drawn campus graph using live GPS snapping to the closest node and turn-by-turn navigation depending on the angle of turns.

This product is explicitly tailored to solve all these problems.

By using React Native for mobile multi-platform support, Mapbox GL for rendering 3D maps with building extrusion and marker clustering, the Bidirectional Dijkstra's algorithm for computing the shortest path based on a manually created GeoJSON walkways graph, the Expo Location API for live GPS location tracking with snap-to-nearest-map-node feature with flat projection, and Firebase Cloud Firestore for campus database support, CampusGO delivers an efficient infrastructure-independent solution that allows implementing a campus routing app for any environment without any additional hardware or licenses required.

CHAPTER THREE

SYSTEM ANALYSIS AND DESIGN

3.0 Introduction

This chapter presents the complete analysis and design of CampusGO. It is organized to address three interrelated aspects that a Computer Science thesis at this level is required to demonstrate: the research design and methodology, the software and system design, and the algorithm specification. Section 3.1 introduces the research design approach, the software design methodology (OOAD), and the system development methodology (Waterfall). Section 3.2 analyses the existing campus navigation situation and its limitations, and introduces CampusGO as the proposed solution. Section 3.3 defines the functional and non-functional requirements. Section 3.4 presents system modelling using UML. Section 3.5 describes data collection. Sections 3.6 and 3.7 present the system design and algorithm design respectively. Section 3.8 describes the deployment architecture and Section 3.9 defines the pre-determined evaluation strategy. Each decision in this chapter is grounded in the research objectives from Chapter One and the literature reviewed in Chapter Two.

3.1 Research Design and Methodology

3.1.1 Research Design

The methodology for this study takes a Quantitative Experimental research design approach. This is a quantitative study due to the nature of the research producing objective, measurable results such as computation times of routes in milliseconds, node expansions required, off-route accuracy detections at certain distances, and functionality tests based on the non-functionality criteria listed in Section 3.3. In addition, this study can be considered an experimental study due to the fundamental contribution of this research in that it evaluates and compares against a series of baseline results the performance of the proposed Bidirectional Dijkstra routing on a hand-traced GeoJSON walkway graph using GPS flat projection snap-to-closest path node and bearing to classify turns.

The following two data sources have been utilized for this task. The information regarding the campus location and pathways was obtained from physical field surveys conducted on the Godfrey Okoye University campus, as explained in Section 3.5. A total of 173 locations on the GOUNI campus, classified into 8 categories of facilities, have been documented using GPS coordinates. In addition to this, a total of 631 walkable pathways were mapped manually on the field, which form the GeoJSON graph serving as the basis for navigation in this system. All the GPS coordinate data and pathway mapping information has been independently gathered and validated by the author of this paper. On the other hand, algorithmic performance data were collected by testing the algorithms on a physical device running on the GOUNI campus using repeatable test cases.

3.1.2 Software Design Methodology — Object-Oriented Analysis and Design

Object-Oriented Analysis and Design (OOAD) has been chosen to develop CampusGO software. OOAD is the most suitable software design technique for CampusGO, as the whole application is written in JavaScript using React Native, which is an object-oriented and component-based language. Also, there are very clear responsibility-bearing classes within the system – Location, Node, Edge, Graph, BidirectionalDijkstra, GeoJSONRouter, LocationService, NavigationService, FirebaseService, PathSmoother, RouteSteps, OffRouteDetector, and ThemeContext. OOAD enables us to have a good analytical understanding of CampusGO’s internal structure, as each class is encapsulating its own state and methods.

OOAD was carried out using the Unified Modelling Language (UML) which is the modelling language standardised for object-oriented software development (Booch et al., 2005). The following UML models were created for CampusGO:

1. Use Case Diagram – the actors of the system interacting with the system
2. Activity Diagram – the workflow of a navigation session split up into five swim lanes
3. UML Class Diagram – thirteen classes of the system showing attributes, methods, and relationships
4. Database Schema Diagram – Firebase Firestore database and the GeoJSON file schema
5. System Architecture Diagram – five-layered system architecture diagram
6. Deployment Architecture Diagram – three layered deployment architecture involving devices, services, and data components
7. Algorithm Pseudocode – Bidirectional Dijkstra's Algorithm used in CampusGO

3.1.3 System Development Methodology — Waterfall Model

As the SDLC methodology for the development of CampusGO, the Waterfall process model was chosen. The Waterfall model represents a sequential and phased SDLC process where each stage of software development such as requirements analysis, design, implementation, testing, and deployment needs to be completed and fully formalized before moving on to the next one (Sommerville, 2016). There are three reasons why the Waterfall model was used.

Firstly, the project requirements were stated and stable since the very beginning of the project. It means that the problem statement (campus navigation at Godfrey Okoye University), the data source (the number of campus locations is 173, and there are 631 traced paths obtained via field survey using physical means), and the general architecture of the application GeoJSON-graph Bidirectional Dijkstra with live GPS positioning) were set before development started. Stable requirements are considered to be the first requirement for sequentiality in the Waterfall model. Secondly, the project is individual and has to be submitted within a certain time frame. Thirdly, the Waterfall model requires producing formal artifacts during each stage: requirements definition, design document, and test cases.

Table 3.1 maps each Waterfall phase to the corresponding CampusGO activities and thesis sections:

Table 3.1: Waterfall Development Phases Mapped to CampusGO Project Activities

Waterfall Phase	CampusGO Activity	Chapter Reference
Requirements Analysis	Gathering functional and non-functional requirements using field survey at campus and objectives established for the study as per Chapter One	Sections 3.2, 3.3
System Design	Implementation of OOAD approach and generation of UML diagrams (use case, activity, class, sequence, ER) and defining five-layer architecture, algorithm pseudocode, and deployment architecture	Sections 3.4, 3.5, 3.6, 3.7, 3.8
Implementation	Coding of Firebase data layer, campus graph, Bidirectional Dijkstra, GeoJSONRouter, GPS API, Mapbox GL map, and React Native screens	Chapter Four, Section 4.2
Testing	Performance of 18 tests in black box, white box, and integration approaches; performance parameters evaluated using physical Android device at the campus site	Chapter Four, Section 4.3
Deployment	Deployment of application using Expo on Android and iOS platforms; Firebase Firestore populated with 173 campus location and 631 paths data; campusPaths.json included locally	Chapter Four, Section 4.4

Indeed, it is important to note that even though the Waterfall model provided the framework for the entire development process, the data gathering and pre-processing process carried out in Section 3.5 involved iterations in the Requirements Analysis stage with the discovery of new campus sites and the verification of their coordinates using the GPS.

3.2 System Analysis

3.2.1 Analysis of the Existing System

The current method of navigation within Godfrey Okoye University makes use of conventional methods which include the use of paper maps, signages, and guidance from university security guards or even other university students. Those who are new to the university will have to ask for directions, refer to outdated paper maps, and try to find their way around through guesswork. Navigation apps like Google Maps are occasionally used, but these do not contain details of the campus routes within GOUNI nor do they have any relevant information specific to the facilities. There is no available method that recognizes the current location of the user and finds an optimum route to the destination.

3.2.2 Limitations of the Existing System

1. The printed maps will eventually go out of date if there are changes on the campus ground plan, and updating them requires reprinting the map, which can be costly.
2. The physical signboards may either be damaged, misplaced, or unavailable in new buildings within the campus grounds.
3. The oral directions provided by either members of the security personnel or students are inconsistent, unpredictable, and not always available at any time.
4. Google maps will not provide you with any information concerning the paths in GOUNI and other important campus facilities.
5. There is no current device that is capable of recognizing your GPS location and calculating the fastest path to your destination.

3.2.3 Analysis of the Proposed System

CampusGO offers a GPS-enhanced digital navigation service that completely displaces existing wayfinding systems. In the CampusGO system, the model used involves modeling the Godfrey Okoye University campus by representing the university's campus as a weighted graph generated from GeoJSON format. Here, 173 campus landmarks become nodes and 631 pathways between landmarks become weighted edges. Data about the campus can be obtained from the Firestore

database of Firebase. The live position of the user is determined by the use of the Expo Location package, while the nearest node on the GeoJSON path graph is determined based on a flat-projection distance computation. The shortest pedestrian route is computed by using bidirectional Dijkstra's algorithm for each selected destination. This is done via the display of a polyline drawn through the nodes on a 3D interactive campus map from Mapbox GL. Rerouting is done automatically whenever there is deviation from the route at distances greater than 30 metres from the user's GPS position. The software is compatible with both Android and iOS devices.

3.3 System Requirements Specification

3.3.1 Functional Requirements

Table 3.2 presents the fifteen functional requirements identified for CampusGO through analysis of the project objectives and campus navigation use cases.

Table 3.2: Functional Requirements for CampusGO

ID	Requirement	Description	Priority
FR01	Campus Map Display	System renders interactive 3D campus map using Mapbox GL with building extrusions and clustered markers	High
FR02	Location Search	User can search all 173 campus facilities by name, category, or description	High
FR03	Route Computation	System computes shortest pedestrian route between any two campus locations using Bidirectional Dijkstra's algorithm	High
FR04	GPS Snap-to-Path	System identifies the nearest GeoJSON path node to user GPS coordinates using flat-projection distance as the routing origin	High
FR05	Route Visualization	Computed route displayed as smoothed blue polyline overlay on the campus map	High
FR06	Turn-by-Turn Steps	System generates bearing-based navigation instructions (depart / straight / slight-left / left / slight-right / right /	High

ID	Requirement	Description	Priority
		arrive)	
FR07	Off-Route Detection	System detects user deviation > 30m from route and automatically recomputes from current GPS position	High
FR08	Location Details	User taps marker to view facility name, photo, description, category, and opening hours	Medium
FR09	Data Retrieval	173 campus locations loaded from Firebase Firestore with long-polling transport on application launch	High
FR10	Live User Location	System displays user GPS position as directional cone indicator rotating with device heading on the map	High
FR11	Marker Clustering	Nearby markers grouped into numbered cluster circles at low zoom levels; expand at zoom 15+	Medium
FR12	Category Browsing	Users filter locations by 8 categories: Academics, Offices, Hostels, Food, Health, Shops, Churches, Sports	Medium
FR13	Light/Dark Theme	User toggles between light and dark UI and map themes from Settings screen	Low
FR14	Arrival Detection	System shows arrival modal when user reaches within threshold distance of destination	Medium
FR15	Save Location	User can bookmark a location for later retrieval from the Saved screen	Low

3.3.2 Non-Functional Requirements

Table 3.3 presents the eight non-functional requirements defining quality attributes and performance constraints for CampusGO. NF01 specifies a 200-millisecond route computation threshold based on the measured performance of Bidirectional Dijkstra on the 173-node campus graph during development testing.

Table 3.3: Non-Functional Requirements for CampusGO

ID	Category	Requirement	Metric
NF01	Performance	Route computation completes in under 200ms on the campus GeoJSON path graph	< 200ms
NF02	Usability	New user can locate a facility and start navigation within 3 interaction steps	≤ 3 steps
NF03	Reliability	App loads campus data on stable internet; 173-location fallback activates on failure	≥ 99% uptime
NF04	Portability	Application runs on Android 8.0+ and iOS 13.0+ from a single codebase	Both platforms
NF05	Scalability	Firebase Firestore scales to 500+ campus locations without architecture change	500+ nodes
NF06	Maintainability	Campus data updated remotely in Firestore without rebuilding or redeploying the app	Remote update
NF07	GPS Accuracy	Snap-to-path identifies correct nearest campus path node within 50m of user position	≤ 50m error
NF08	Theme Support	Light and dark UI themes applied consistently across all screens and map styles	Full coverage

3.4 System Modelling Using UML

3.4.1 Use Case Diagram

In the use case diagram presented in figure 3.4.1, there are two main actors, which are the User (student, faculty, and guest) and the Firebase Backend. There are ten use cases, which include the following: viewing the interactive 3D campus map, finding a campus location, calculating route from current GPS location, viewing route and turn by turn route direction, receiving off-route reroute, viewing information about facilities and photo of facility, filtering location based on categories, saving a particular location, enabling light and dark mode, and retrieving campus data from firebase firestore.

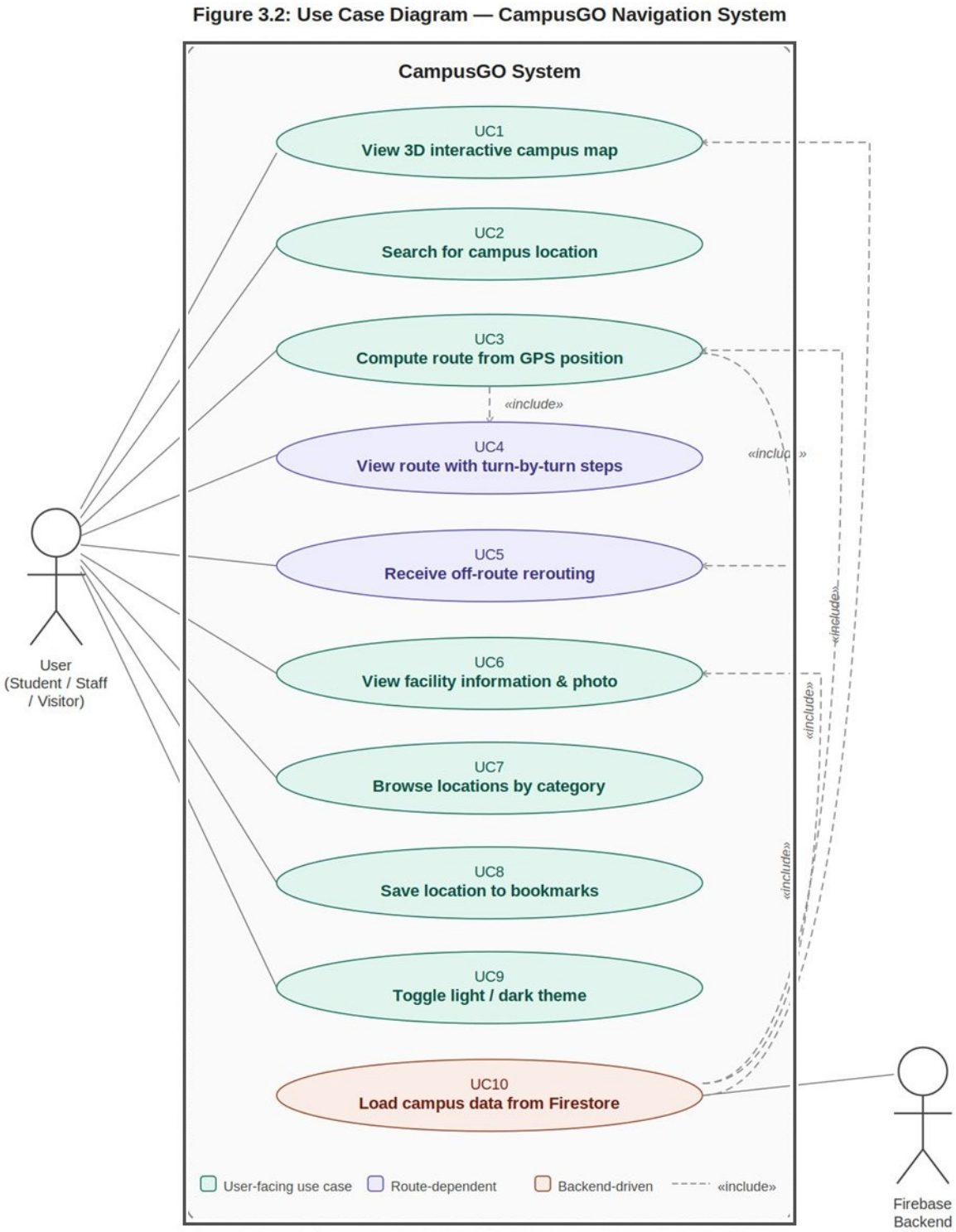


Figure 3.4.1: Use Case Diagram for CampusGO

3.4.2 Activity Diagram

The activity diagram in Figure 3.4.2 shows the entire process flow of navigation session starting from the application start till the confirmation of arrival mode after calculating the route, including the process flows for permission to access GPS, Firebase data load, 3D map rendering, location selection, route calculation, turn-by-turn guidance, out-of-route detection and routing, and confirmation of arrival mode. The process flow consists of five swim lanes:

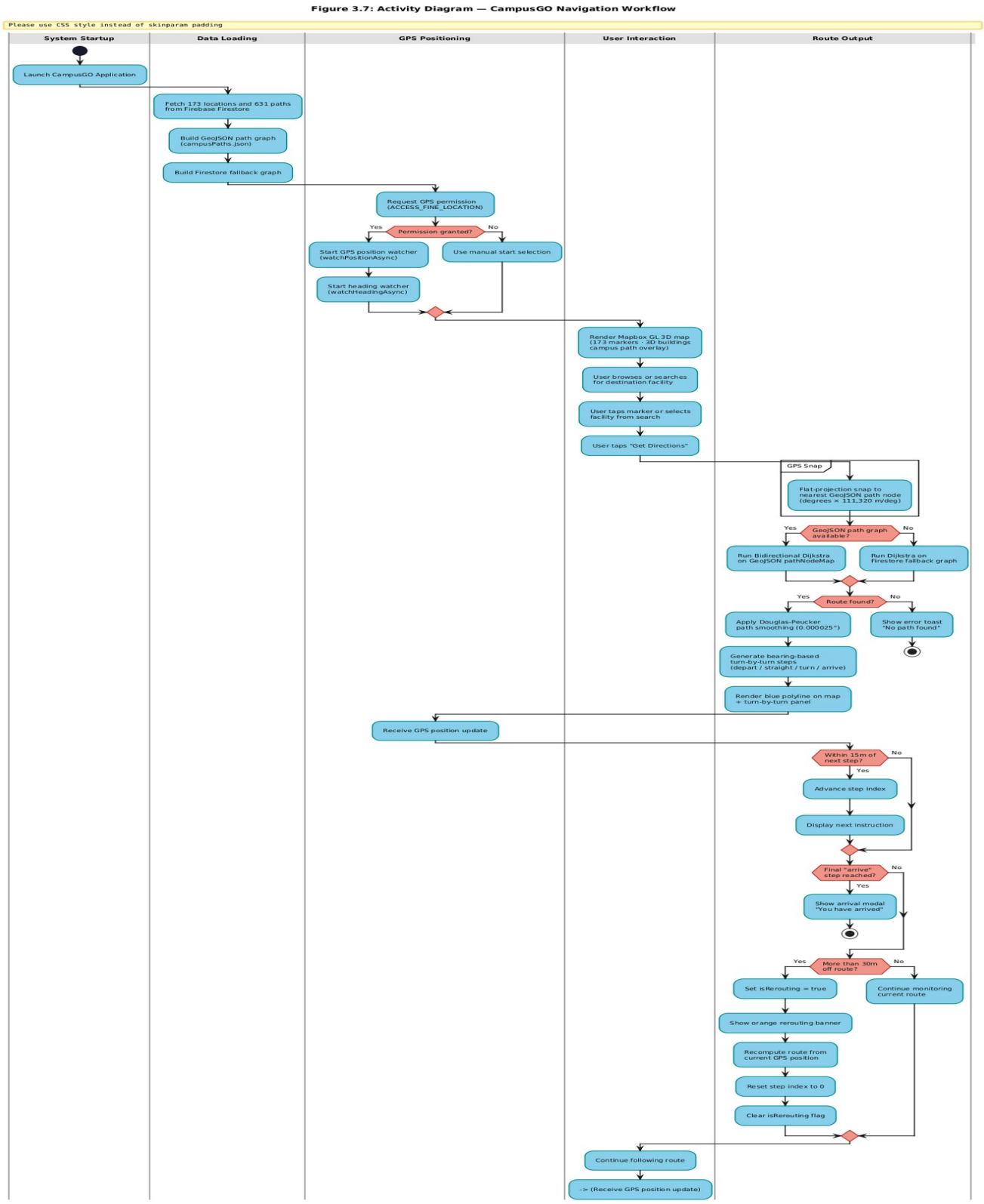


Figure 3.4.2: Activity Diagram — CampusGO Navigation Workflow

3.4.3 Class Diagram

Figure 3.4.3 depicts the UML class diagram consisting of thirteen classes in four architectural levels: Location, Node, Edge, and Graph are part of the data model layer; BidirectionalDijkstra, GeoJSONRouter, LocationService, PathSmoother, RouteSteps, and OffRouteDetector are part of the algorithm and service layer; NavigationService and FirebaseService are part of the coordinator and backend layer; and finally, ThemeContext is part of the UI context layer. Every class is an entity with its own attributes and methods, thus allowing each of the entities to develop and test independently.

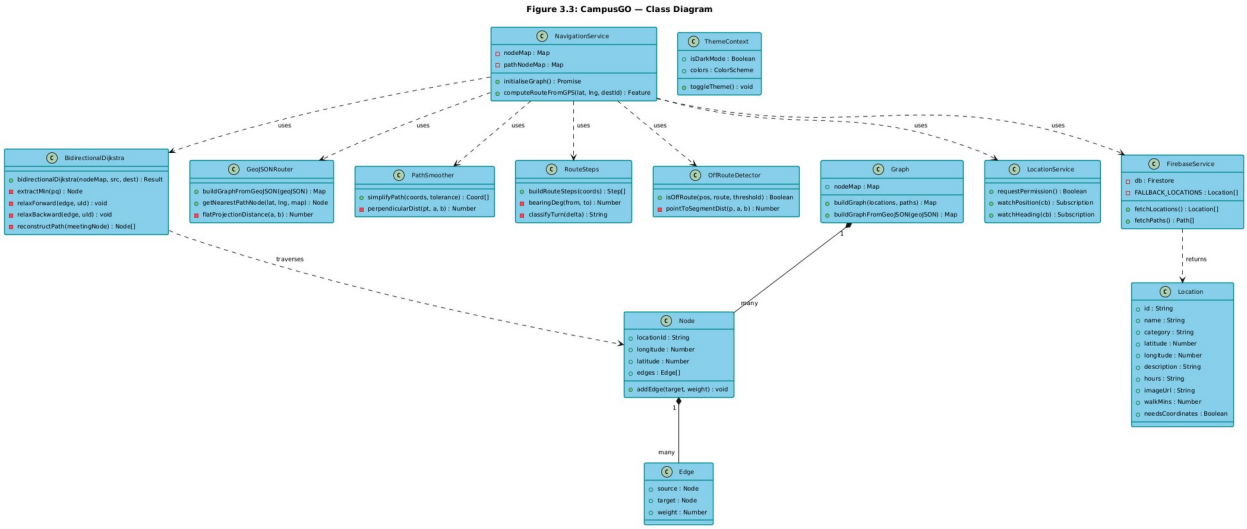


Figure 3.4.3: UML Class Diagram for CampusGO

3.4.4 Database Schema

The figure below illustrates Firebase Firestore schema and GeoJSON file format. The Firestore locations table holds the following fields in 173 records: id, name, category, latitude, longitude, description, hours, imageUrl, distance, walkMins, and the needsCoordinates boolean field. Firestore paths table holds three fields in 631 records: sourceId, targetId, and distanceMetres for the Firestore fallback routing graph. There are two additional GeoJSON files packed with the application at compile-time: campusPaths.json – a FeatureCollection of LineString features of all walkable paths in campus and used to construct the routing graph, and buildings.json – a

FeatureCollection of Polygon features with height and base_height fields used to display campus buildings using the Mapbox GL FillExtrusionLayer.

Figure 3.5: Firebase Firestore and GeoJSON Data Schema — CampusGO

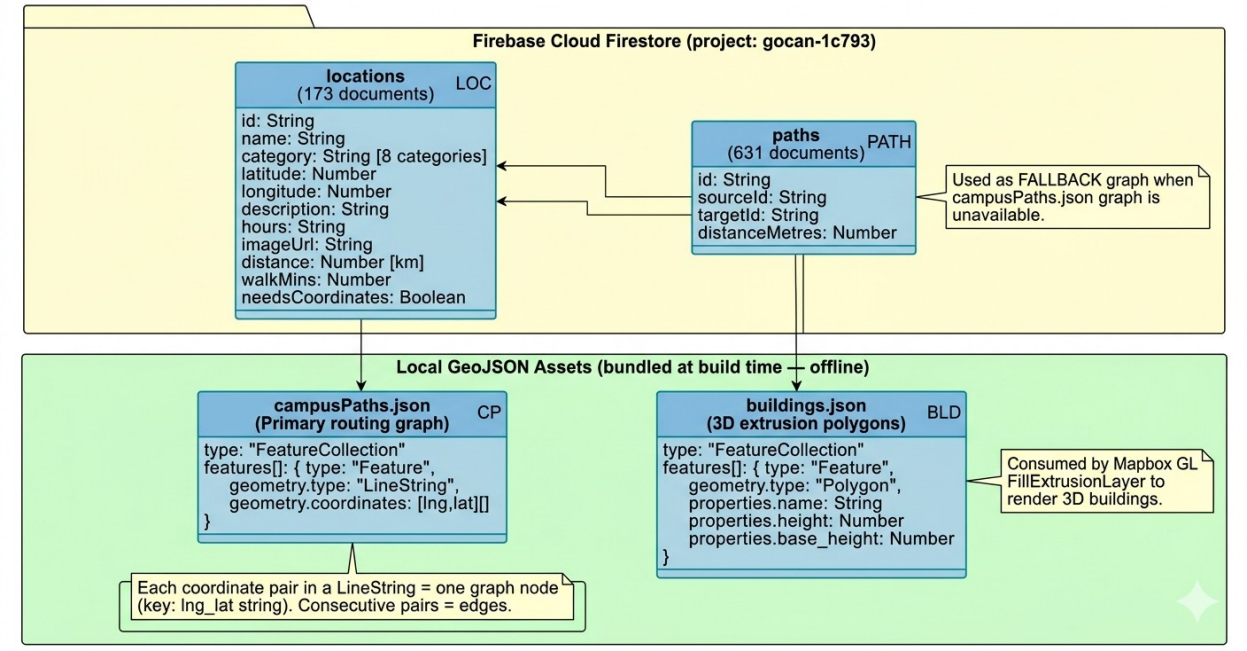


Figure 3.4.4: Firebase Firestore and GeoJSON Data Schema for CampusGO

3.5 Method of Data Collection

Data collection of campus location attributes, which include building names, geographical coordinates, facility types, opening hours, and photographic evidence was performed using physical surveys at Godfrey Okoye University campus, which was done by the researcher. To capture the geographical coordinates for each of the 173 campus locations, a smartphone that had a GPS sensor was utilized. A physical survey of each of the facilities was undertaken to identify their name, type, details, and operating hours. The geographical coordinates were captured physically standing at the exact location.

Footpaths within the campus were drawn manually using satellite images along with real-world ground observations into the GeoJSON LineString geometry format. In all, there were 631 footpath segments mapped out using coordinates that represented exactly where the footpath could be walked. Similarly, all building outlines were drawn manually using the satellite images with a property of height value in GeoJSON Polygon geometry format as seen in buildings.json for

generating the 3D extrusion on the map. All the 173 campus location datasets were imported into Firebase Firestore database through a Node.js batch seeding operation. All geographical datasets used in Campus GO application development were collected independently by the researcher without sourcing any data from the G-Map web application.

3.6 System Design

3.6.1 Input Design

There are two main types of input that go into the system. Routing starts automatically based on the user's current coordinates from their device and matched against the closest campusPaths.json graph node through flat projection distance approximation, guaranteeing that the starting point will be where the user is currently physically located within their nearest available walkway. Otherwise, users can choose their own starting point in the Search or Category browsing screens. The routing destination is chosen via selecting the location markers in the Mapbox GL 3D campus map interface or selecting the facilities in the Search and Category browsing interfaces. The system will only initiate route calculations in cases when there is no match between the source and destination, or if the user denied GPS permissions but chose the starting point manually.

3.6.2 Output Design

The key output is the smoothened blue polyline route overlay map based on Douglas-Peucker algorithm-simplified GeoJSON LineString coordinates on the Mapbox GL 3D campus map. A directional cone is used to represent the live GPS location of the user on the map, with a dynamic rotation of the device according to its heading through the watchHeadingAsync function of Expo Location API. Other key outputs include: The turn-by-turn navigation screen, which displays the current instruction, the upcoming one, the total distance of the route in meters, and walking time estimated at 80 m/minute; The location details page that contains information about the facility, such as the facility name, category badge, image, description, and opening hours; And finally, the arrival modal screen and the orange rerouting banner when off-course by more than 30 meters.

3.6.3 Five-Layer System Architecture

Figure 3.6.3 presents the five-layer architecture of CampusGO, which ensures that each layer can be modified independently without affecting adjacent layers:

6. Presentation Layer – React Native UI screens (HomeScreen, SearchScreen, LocationDetailScreen, NavigationScreen, SettingsScreen), interactive Mapbox GL 3D map, and ThemeContext for switching themes.
7. Navigation/Context Layer – AppNavigator (Stack navigator coordinating eight screens), ThemeContext, and screen prop-based route state propagation.
8. Service Layer – NavigationService (routing coordination service), LocationService (location and heading services), GeoJSONRouter (path graph construction and routing), OffRouteDetector, RouteSteps (turn-by-turn instruction generator), PathSmoother (path graph smoothing using Douglas-Peucker algorithm), and FirebaseService (Firestore database querying with long-polling).
9. Algorithm Layer – BidirectionalDijkstra.js (main routing algorithm on the path graph); single-direction Firestore Dijkstra (alternative route calculation method used when the GeoJSON graph is not available).
10. Data Layer – Firebase Firestore (173 locations and 631 paths documents); campusPaths.json (main routing GeoJSON graph stored locally); buildings.json (3D building polygon extrusions stored locally); hard-coded fallback of 173 locations graph for graceful degradation in offline mode.

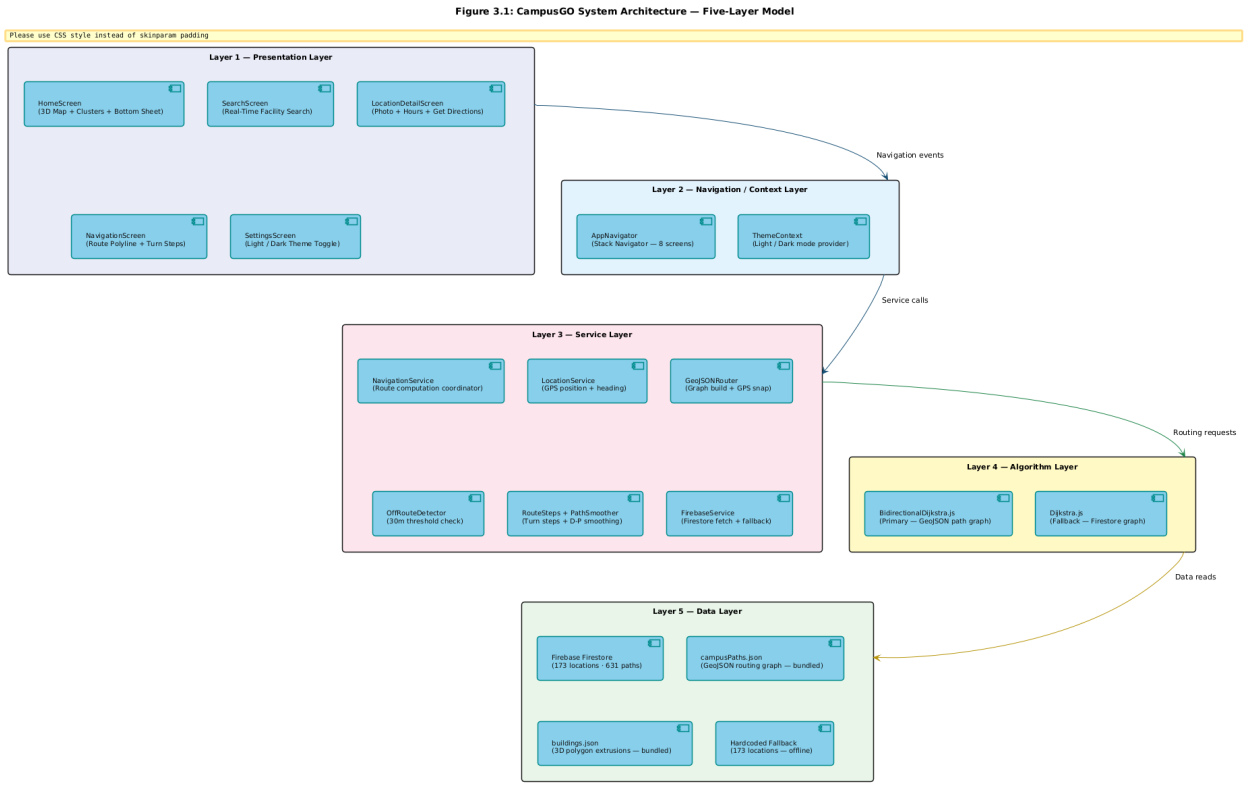


Figure 3.6.3: CampusGO System Architecture — Five-Layer Model

3.7 Algorithm Design

3.7.1 Bidirectional Dijkstra's Algorithm — Primary Routing

The bidirectional version of Dijkstra's algorithm is realized in BidirectionalDijkstra.js as the only one routing algorithm utilized in CampusGO. Two priority queues pqF for the search in the forward direction, starting from the source node and pqB for the backward search, starting from the destination node, work concurrently. Distances distF, distB, predecessors' map prevF, prevB, and visited sets maintain two separate frontiers. If any common settled node occurs in both frontiers, the meeting node candidate is detected and a possible bestCost update occurs. Early termination starts once the total of the minimum priorities in the head of both queues becomes greater than or equal to bestCost. The best path reconstruction requires the merging of forward (through prevF) and backward (with the help of prevB) partial paths. For the estimation of the walking time, the walking speed of 80 m/min is considered. In terms of computational complexity, the number of nodes that need to be examined amounts to about half those needed by the single-direction variant

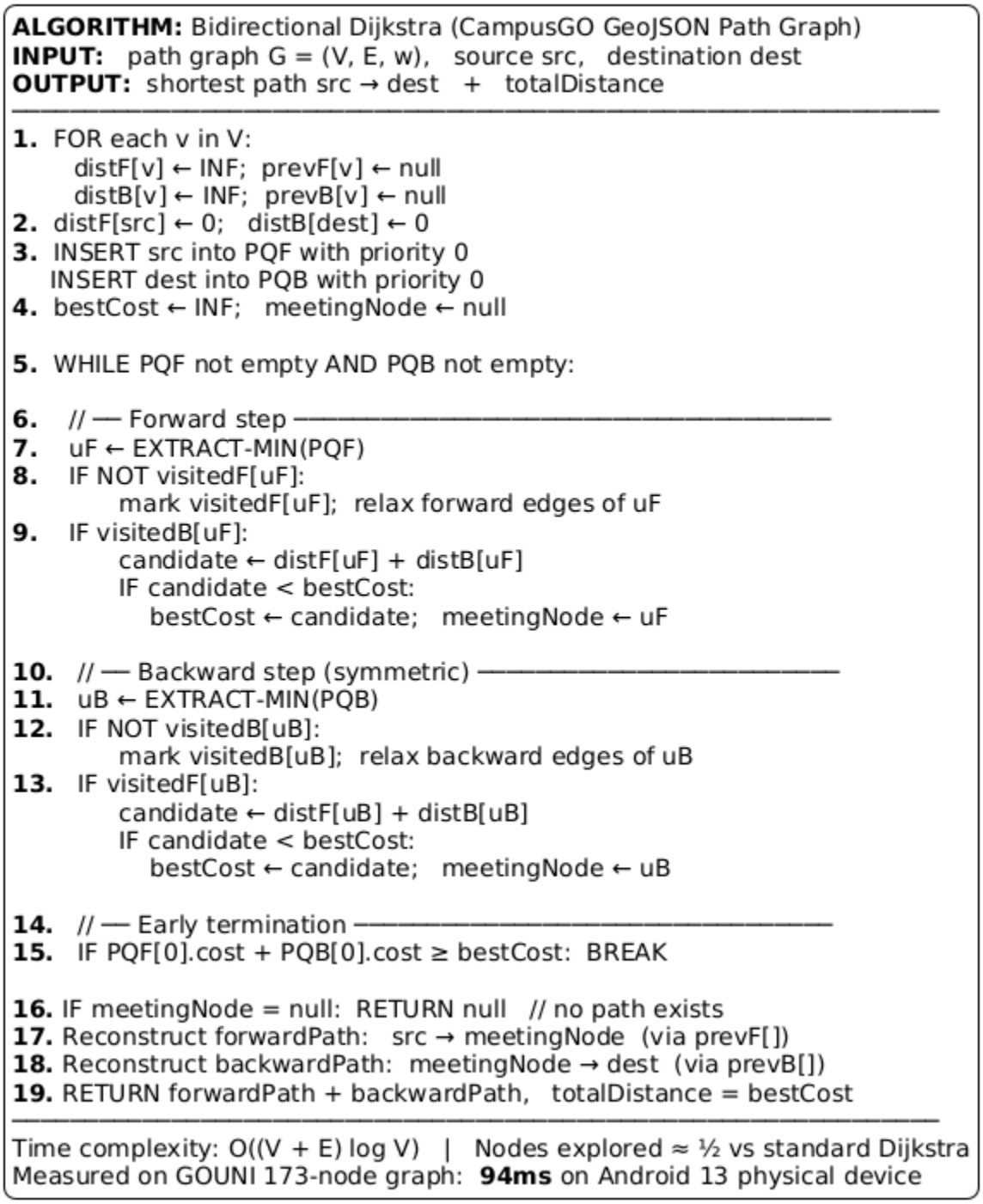
of Dijkstra. It fits the 200 milliseconds performance criterion mentioned in NF01 (Pohl, 1971; Kaindl & Kainz, 1997).

The pseudocode is presented in Figure 3.6:

```
ALGORITHM: Bidirectional Dijkstra (CampusGO Path Graph)

INPUT:  path graph G=(V,E,w),  source src,  destination dest
OUTPUT: shortest pedestrian path src → dest + total distance

1.  FOR each v in V:
       distF[v]←INF; prevF[v]←null   // Forward distances +
predecessors
       distB[v]←INF; prevB[v]←null   // Backward distances +
predecessors
2.  distF[src]←0; distB[dest]←0
3.  INSERT src into PQF with priority 0
    INSERT dest into PQB with priority 0
4.  bestCost←INF; meetingNode←null

5.  WHILE PQF not empty AND PQB not empty:

6.    // --- Forward step ---
7.    uF←EXTRACT-MIN(PQF)
8.    IF NOT visitedF[uF]: mark visitedF[uF]; relax forward edges of uF
9.    IF visitedB[uF]:
10.     candidate←distF[uF] + distB[uF]
11.     IF candidate < bestCost: bestCost←candidate; meetingNode←uF

12.   // --- Backward step (symmetric) ---
13.   uB←EXTRACT-MIN(PQB)
14.   IF NOT visitedB[uB]: mark visitedB[uB]; relax backward edges of
uB
15.   IF visitedF[uB]:
16.     candidate←distF[uB] + distB[uB]
```

```
17.     IF candidate < bestCost: bestCost←candidate; meetingNode←uB

18.   // --- Early termination ---
19.   IF PQF[0].cost + PQB[0].cost ≥ bestCost: BREAK

20. IF meetingNode = null: RETURN null   // No path exists
21. Reconstruct forwardPath:  src → meetingNode via prevF[]
22. Reconstruct backwardPath: meetingNode → dest via prevB[]
23. RETURN forwardPath + backwardPath,  totalDistance = bestCost
```

Figure 3.6: Bidirectional Dijkstra's Algorithm Pseudocode



```
ALGORITHM: Bidirectional Dijkstra (CampusGO GeoJSON Path Graph)
INPUT:  path graph G = (V, E, w),  source src,  destination dest
OUTPUT: shortest path src → dest  +  totalDistance
──────────────────────────────────────────────────────────────────
1.  FOR each v in V:
        distF[v] ← INF;  prevF[v] ← null
        distB[v] ← INF;  prevB[v] ← null
2.  distF[src] ← 0;  distB[dest] ← 0
3.  INSERT src into PQF with priority 0
    INSERT dest into PQB with priority 0
4.  bestCost ← INF;  meetingNode ← null

5.  WHILE PQF not empty AND PQB not empty:

6.    // ── Forward step ──────────────────────────────
7.    uF ← EXTRACT-MIN(PQF)
8.    IF NOT visitedF[uF]:
          mark visitedF[uF];  relax forward edges of uF
9.    IF visitedB[uF]:
          candidate ← distF[uF] + distB[uF]
          IF candidate < bestCost:
              bestCost ← candidate;  meetingNode ← uF

10.   // ── Backward step (symmetric) ─────────────────
11.   uB ← EXTRACT-MIN(PQB)
12.   IF NOT visitedB[uB]:
          mark visitedB[uB];  relax backward edges of uB
13.   IF visitedF[uB]:
          candidate ← distF[uB] + distB[uB]
          IF candidate < bestCost:
              bestCost ← candidate;  meetingNode ← uB

14.   // ── Early termination ─────────────────────────
15.   IF PQF[0].cost + PQB[0].cost ≥ bestCost:  BREAK

16. IF meetingNode = null:  RETURN null   // no path exists
17. Reconstruct forwardPath:  src → meetingNode  (via prevF[])
18. Reconstruct backwardPath:  meetingNode → dest  (via prevB[])
19. RETURN forwardPath + backwardPath,  totalDistance = bestCost
──────────────────────────────────────────────────────────────────
Time complexity: O((V + E) log V)  |  Nodes explored ≈ ½ vs standard Dijkstra
Measured on GOUNI 173-node graph:  94ms on Android 13 physical device
```

Figure 3.7.1: Bidirectional Dijkstra's Algorithm Pseudocode as Implemented in CampusGO

3.7.2 GPS Snap-to-Nearest-Path-Node

Before executing the routing function, the computeRouteFromGPS() function invokes the getNearestPathNode() function inside the GeoJSONRouter.js file. In that function, every node in the GeoJSON graph of paths is traversed and the distance between the node and the user's GPS coordinates is calculated using the following approximation:

$$dx = (node.longitude - userLng) \times 111{,}320 \times \cos(userLat \times \pi/180);$$

$$dy = (node.latitude - userLat) \times 111{,}320;$$

$$distance = \sqrt{(dx^2 + dy^2)}$$

The node with the shortest distance is selected as the starting point for the routing operation. It is noted that the use of this approximation works fine on campus scale where distances do not exceed about two kilometres.

3.7.3 Douglas-Peucker Path Smoothing

The PathSmoother.js library employs the recursive Douglas-Peucker algorithm to simplify the lines using the unprocessed coordinates obtained through the routing algorithm, with a tolerance of 0.000025 degrees (roughly 2.8 meters). This algorithm determines the farthest point of the line measured in terms of the maximum perpendicular distance from the straight line drawn between the start and end point coordinates. Should the distance be greater than the tolerance value, then the line gets cut into two parts and each part gets recursively simplified until the criteria of distance to the tolerance value gets met. Unsmoothed coordinates get supplied to RouteSteps for creating turn instructions.

3.7.4 Turn-by-Turn Step Generation

The RouteSteps.js file converts the coordinate array of the initial route into directions. This is done for each pair of consecutive lines by calculating the bearing delta in the following way: delta = bearingDeg(coords[i], coords[i+1]) - bearingDeg(coords[i−1], coords[i]). The normalized value of delta determines the type of the direction according to the next criteria: delta between 0 and 20 degrees results in a straight instruction, while 20 to 60 degrees will return a slight-left or slight-right instruction, whereas any value above 60 degrees returns either a left or right instruction. A depart and an arrive instructions are added at the beginning and the end of the route respectively.

3.8 Deployment Architecture

Deployment Architecture is shown in Figure 3.8, which describes the physical and cloud resources used to implement the entire system.

The React Native app is compiled through the Expo toolset and deployed on the Android and iOS platforms through Expo Dev Client. The package file of the app contains two data files in GeoJSON format bundled during the compilation process: campusPaths.json file for the main routing graph containing all walking routes on the campus, and the buildings.json file for 3D models of campus building polygons.

During run-time, the application connects to two cloud providers. Firebase Cloud Firestore (project ID: gocan-1c793) hosts 173 documents regarding campus location information and 631 documents for the path in the Firestore fallback routing graph. Communication with Firestore is handled using Firebase JavaScript SDK v10 with experimentalForceLongPolling set to true, which switches out the WebChannel protocol for regular HTTP long-polling to circumvent known connection issues with Expo on Android. Mapbox GL tile server provides vector map tiles for the campus map layer by authenticating with the public Mapbox API token from the application configuration.

Images for the 173 locations of the campus buildings are delivered as static images from the project's web server upon request when the user opens a page with location details. Internet connectivity is needed for these to be displayed, but they degrade to an alternative image if connectivity is unavailable. The Firestore seed script (scripts/seedFirestore.js) is a stand-alone Node.js application using the Firebase Admin SDK to write all 173 locations and 631 paths to Firestore in one batch process from the developer's local computer.

Deployment properties: GeoJSON data of the routing graph and the building footprints are contained within the bundle, enabling route calculation in offline mode; Firestore documents are loaded at the initial run, whereas the fallback containing 173 locations kicks in if Firestore cannot be reached; Mapbox tiles are cached after the first run thanks to Mapbox's local tile cache system; and location images are internet-dependent but silent about failure to connect.

Figure 3.8: Deployment Architecture — CampusGO

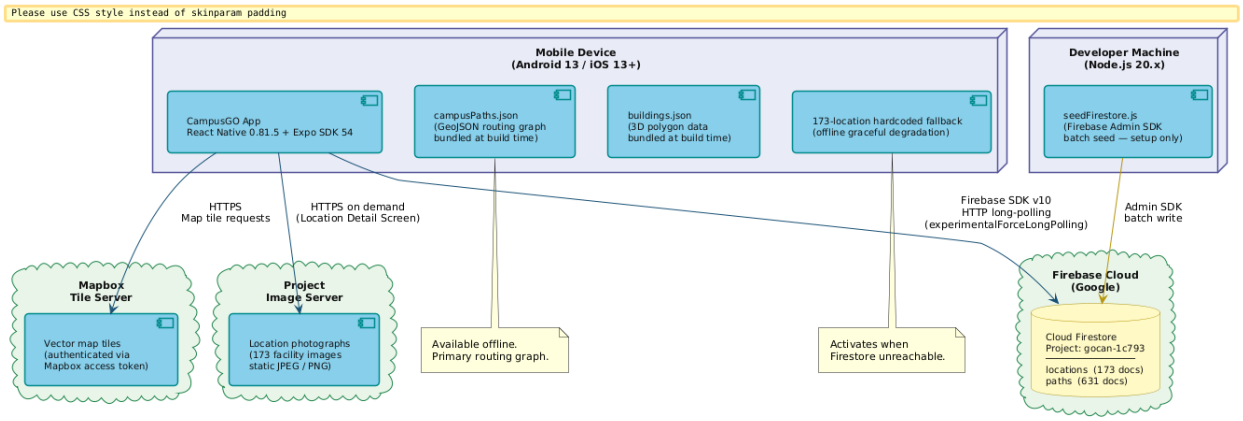


Figure 3.8: Deployment Architecture for CampusGO

3.9 Evaluation Strategy

A well-defined evaluation strategy has been defined in this chapter before implementation to make sure that the testing strategy has been decided while designing the system, and not at the end to explain the results obtained from tests. The process is quite consistent with standard software engineering practice (Sommerville, 2016). Table 3.4 shows the overall evaluation strategy for CampusGO through ten evaluation criteria:

Table 3.4: Evaluation Strategy and Test Methods for CampusGO

Evaluation Criterion	Method	Description	Req. Tested
Algorithm Correctness	White Box	BidirectionalDijkstra returns the same shortest path as unidirectional Dijkstra on the same graph; approximately 2× fewer node expansions verified on a controlled test graph	FR03
Turn Classification	White Box	buildRouteSteps tested against known coordinate sequences specifically designed to produce bearing deltas at the 20° and 60° classification thresholds	FR06
Off-Route Logic	White Box	isOffRoute verified to return false for positions on the route polyline and true for positions fabricated 35m perpendicular to a route segment	FR07

Evaluation Criterion	Method	Description	Req. Tested
Path Smoothing	White Box	simplifyPath confirmed to preserve first and last coordinates and reduce intermediate point count under the 0.000025-degree tolerance	FR05
Map Rendering	Black Box	3D building extrusions, clustered markers, and route polyline render correctly on the physical Android device at zoom level 16	FR01, FR11
GPS Route Computation	Black Box	Route computed from live GPS position to selected destination in under 200ms on the full campus graph on a physical Android device	FR03, FR04, NF01
Off-Route Rerouting	Integration	Rerouting triggers within 2–3 GPS update cycles (approximately 4–6 seconds) when user walks more than 30m from the route on campus	FR07
Step Advancement	Integration	Step index advances correctly when user walks within 15m of the next step's start coordinate during on-campus walking test	FR06
Usability Walkthrough	Black Box	New user completes full workflow: open app → locate facility → tap Get Directions within 3 interaction steps as specified in NF02	NF02
Firebase Remote Update	Black Box	New location document added to Firestore collection appears in the application after restart without any code change or rebuild	NF06

The black-box testing of the system ensures that each requirement is satisfied to produce accurate output irrespective of valid and invalid input. The white-box testing of the algorithm is done using a particular set of graph inputs with known outputs and checks whether the algorithm satisfies the output requirements mathematically. The integration testing tests the navigation process on an Android phone installed with the app in real-world environment at the GOUNI campus, where GPS and actual distances walked are used in testing off route detection and step movement.

All the ten evaluation criteria encompass all the functional requirements outlined in Table 3.2 as well as all the non-functional requirements described in Table 3.3. The eighteen test cases identified in Chapter Four map directly into these requirements and evaluation criteria outlined above.

CHAPTER FOUR

SYSTEM IMPLEMENTATION AND RESULTS

4.0 Introduction

In this chapter, the whole realization of the system CampusGO is discussed, including the development environment used and tools, the code realization for modules, process flowcharts that demonstrate the two most complicated processes at run time, system screens, testing techniques applied and its results compared with the approach mentioned in section 3.9, as well as the outcomes discussed in comparison with requirements mentioned in Chapter Three. The order of presentation here will be based on the sequence followed during development.

4.1 Development Environment and Tools

Table 4.1 presents the complete set of tools and technologies used in the development of CampusGO, with version numbers and justifications for each selection.

Table 4.1: Development Tools and Technologies Used in CampusGO

Tool / Technology	Version	Role	Justification
React Native	0.81.5	Cross-platform mobile framework	Single codebase for Android and iOS; component-based architecture
JavaScript (ES2022)	ES2022	Primary programming language	Natively supported by React Native; covers all application logic, algorithm implementation, and service integration
@rnmapbox/ maps	10.3.0	Interactive 3D map rendering	GPU-accelerated vector tiles; FillExtrusionLayer for 3D buildings; native marker clustering; GeoJSON route overlays
Firebase Firestore	firebase ^10.7.0	Cloud database for campus data	Serverless NoSQL; experimentalForceLongPolling for Android reliability; remote updates without rebuild
Expo	SDK	Build toolchain	Managed workflow; Expo Dev Client for

Tool / Technology	Version	Role	Justification
	~54	and testing	physical device testing; OTA update support
expo-location	~19.0.8	Live GPS positioning and heading	ACCESS_FINE_LOCATION; watchPositionAsync for position; watchHeadingAsync for directional cone rotation
react-native-reanimated	~4.1.1	UI animations	Smooth bottom sheet drag; navigation step transitions
@react-navigation/native-stack	^6.9.17	Screen navigation	Stack navigator managing eight screens; state passing as route params
@expo/vector-icons (Ionicons)	bundled	UI iconography	Consistent icon language across all screens without emoji dependency
Node.js + firebase-admin	20.x + 12.x	Firestore seed script	Node.js batch script writes all 173 locations and 631 paths to Firestore
Git / GitHub	2.44+	Version control	Branch-based development; commit history; remote repository backup

Expo's managed flow with the help of Expo Dev Client was employed as the start point for the project, enabling a bare flow that is able to use the native Mapbox GL modules that are not included in the default Expo Go app. Dependencies in the project were versioned through their inclusion in the package.json file. Versioning was done via Git, with the project being developed from a private GitHub repository.

4.2 System Implementation and Modules

The CampusGO system was designed and implemented as nine components which include Firebase data, campus graph model, Bidirectional Dijkstra algorithm, GPS location, navigator coordinator, path smoother, route step generator, off-route detector, and Mapbox GL map. The

implementation followed the system design outlined in chapter three with each individual component being tested independently prior to integration.

4.2.1 Firebase Data Layer

FirebaseService service module is used to initialise Firebase by setting experimentalForceLongPolling to be true. This helps in using long polling via HTTP and avoid WebChannel connection failure issue with the WebChannel protocol not working in Android devices using the Dev Client app. The service module uses a hard-coded set of campuses' location list containing 173 campuses, which gets displayed automatically in case of unavailability of the internet connection.

```
// services/FirebaseService.js (key extract)
const db = initializeFirestore(app, {
  experimentalForceLongPolling: true,  // Android WebChannel fix
});

export const fetchLocations = async () => {
  try {
    const snapshot = await getDocs(collection(db, "locations"));
    if (snapshot.docs.length > 0)
      return snapshot.docs.map(doc => ({ id: doc.id, ...doc.data() }));
    return FALLBACK_LOCATIONS;   // 173-location hardcoded fallback
  } catch {
    return FALLBACK_LOCATIONS;
  }
};
```

4.2.2 Campus Graph Model and Bidirectional Dijkstra

When the application starts, initialiseGraph() runs two graph constructions concurrently. Firstly, the Firestore graph creation process: the functions fetchLocations() and fetchPaths() query data from Firestore, while Graph.js builds the graph nodes using location IDs, along with the Node and Edge class objects and Haversine-weighted edges calculated on the basis of distanceMetres values from paths. Secondly, the GeoJSON graph construction process: GeoJSONRouter.js traverses each LineString feature of campusPaths.json file to create the nodes for every coordinate pair (identified with lng_lat), adding bidirectional Haversine-weighted edges between subsequent

coordinates. The reason for using the GeoJSON graph as the default graph is that it has geometry matching the walkable path geometry, unlike the Firestore graph.

```
// GeoJSONRouter.js — graph construction (key extract)
export function buildGraphFromGeoJSON(pathsGeoJSON) {
  const nodes = {};
  pathsGeoJSON.features.forEach(feature => {
    const coords = feature.geometry.coordinates;
    for (let i = 0; i < coords.length - 1; i++) {
      const [aLng,aLat] = coords[i], [bLng,bLat] = coords[i+1];
      const keyA = `${aLng.toFixed(6)}_${aLat.toFixed(6)}`;
      const keyB = `${bLng.toFixed(6)}_${bLat.toFixed(6)}`;
      if (!nodes[keyA]) nodes[keyA] = { id:keyA, lng:aLng, lat:aLat,
edges:[] };
      if (!nodes[keyB]) nodes[keyB] = { id:keyB, lng:bLng, lat:bLat,
edges:[] };
      const dist = haversineMeters(aLat, aLng, bLat, bLng);
      nodes[keyA].edges.push({ target:nodes[keyB], weight:dist });
      nodes[keyB].edges.push({ target:nodes[keyA], weight:dist }); //
undirected
    }
  });
  return nodes;
}
```

The BidirectionalDijkstra.js file contains an implementation of the bidirectional Dijkstra algorithm using two priority queues (pqF and pqB), sets to hold visited nodes, distance matrices (distF and distB), and maps for predecessors (prevF and prevB). The variables bestCost and meetingNode contain the information on the best path encountered so far. The stopping criteria kick in when the sum of the minimal costs from each priority queue equals or surpasses bestCost.

```
// BidirectionalDijkstra.js (key extract)
export const bidirectionalDijkstra = (nodeMap, sourceId, destId) => {
  let bestCost = Infinity, meetingNode = null;
  // ... initialise distF, distB, prevF, prevB, pqF, pqB ...
  while (pqF.length > 0 && pqB.length > 0) {
    const { id: uF } = extractMin(pqF);
    if (!visitedF.has(uF)) { visitedF.add(uF); relaxForward(uF); }
```

```
    if (visitedB.has(uF)) {
      const c = distF[uF] + distB[uF];
      if (c < bestCost) { bestCost = c; meetingNode = uF; }
    }
    // Symmetric backward step ...
    if ((pqF[0]?.cost ?? Infinity) + (pqB[0]?.cost ?? Infinity) >=
bestCost) break;
  }
  return meetingNode
    ? { path: reconstructPath(meetingNode), totalDistance: bestCost }
    : null;
};
```

4.2.3 Navigation Service — Route Computation Coordinator

NavigationService.js is the main coordinating class responsible for managing all route-related activities. computeRouteFromGPS(userLat, userLng, destId) method manages the whole flow by calling getNearestPathNode() that helps find the nearest point to user's current GPS coordinates through approximate calculations using flat-projection distances, finds the nearest point to destination coordinates, computes bidirectionalDijkstra on the pathNodeMap of GeoJSON, gets the coordinates array from the outcome, performs simplifyPath() for rendering GeoJSON with Douglas Peucker smoothing, performs buildRouteSteps() function on the initial coordinates array without smoothing for turn-by-turn instructions, and finally produces a GeoJSON feature that contains LineString with smoothed coordinates array, steps, node count, location IDs, distance in metres, and walking minutes as properties.

4.2.4 Mapbox GL Map Integration

Mapbox GL MapView is rendered in a 3D view at 45-degree pitch through the HomeScreen. In total, six ShapeSource and Layer compositions render the entire visualization of the campus map:

11. 11. campusPaths LineLayer on campusPaths.json — a subtle white layer displaying all pathways across campus, which remain permanently visible in the map for guidance.
12. 12. buildings3d FillExtrusionLayer on buildings.json — 3D shapes representing all buildings on the campus rendered in green color, according to the height specified in buildings.json.

13. 13. clusterCircles CircleLayer on Firestore locations — dark colored circles with the size proportional to the point_count and a green boundary around them.

14. 14. clusterCounts SymbolLayer on Firestore locations — white number labels for each cluster circle.

15. 15. unclusteredPoints CircleLayer — category-colored location markers appearing when zooming in at zoom level 15.

16. 16. routeLine and routeShadow LineLayer on computed GeoJSON — blue polyline for the displayed route with shadow effect applied.

```
// HomeScreen.js — clustering and 3D buildings (key extract)
<MapboxGL.ShapeSource id="locationsSource" shape={locGeoJSON}
  cluster={true} clusterRadius={50} clusterMaxZoomLevel={14}>
  <MapboxGL.CircleLayer id="clusterCircles"
    filter={["has","point_count"]}
    style={{ circleColor:"#1f2937",
             circleRadius:["step",
["get","point_count"],20,10,28,30,36],
             circleStrokeColor:"#22c55e", circleStrokeWidth:2 }} />
  <MapboxGL.CircleLayer id="unclusteredPoints"
    filter={["!",["has","point_count"]]}
    style={{ circleColor:["get","color"], circleRadius:10,
             circleStrokeColor:"#fff", circleStrokeWidth:1.5 }} />
</MapboxGL.ShapeSource>

<MapboxGL.ShapeSource id="buildingsSource" shape={buildingsData}>
  <MapboxGL.FillExtrusionLayer id="buildings3d"
    style={{ fillExtrusionColor:"#4a7c59",
             fillExtrusionHeight:["get","height"],
             fillExtrusionOpacity:0.8 }} />
</MapboxGL.ShapeSource>
```

4.3 Key Flowcharts

Two flowcharts have been used to explain the complex run time process involved in CampusGO. This helps further the understanding of the process as compared to the sequence diagrams

explained in Figure 3.4 because flow charts can be used to show the decisions and loops involved. The two flow charts pertain to the functional requirements FR03, FR04, FR06, and FR07.

4.3.1 Route Computation Flowchart

Figure 4.3.1 presents the full sequence of processes involved from when the user clicks “Get Directions” on LocationDetailScreen until when the polyline and directions appear on NavigationScreen. The figure illustrates four consecutive decisions made during the process. Firstly, the application verifies whether the GPS permission was granted; if not, the user will be asked to select a manual start location. Secondly, the system confirms if the GeoJSON path graph was successfully created from campusPaths.json; otherwise, the fallback to Firestore location graph occurs. Thirdly, the Bidirectional Dijkstra process is run on the chosen graph; in the case where no path was generated, an explanatory message would be shown in form of a toast rather than crash the application. Lastly, if the path is generated, the coordinates will be cleaned using the Douglas-Peucker algorithm and sent to RouteSteps to create instructions.

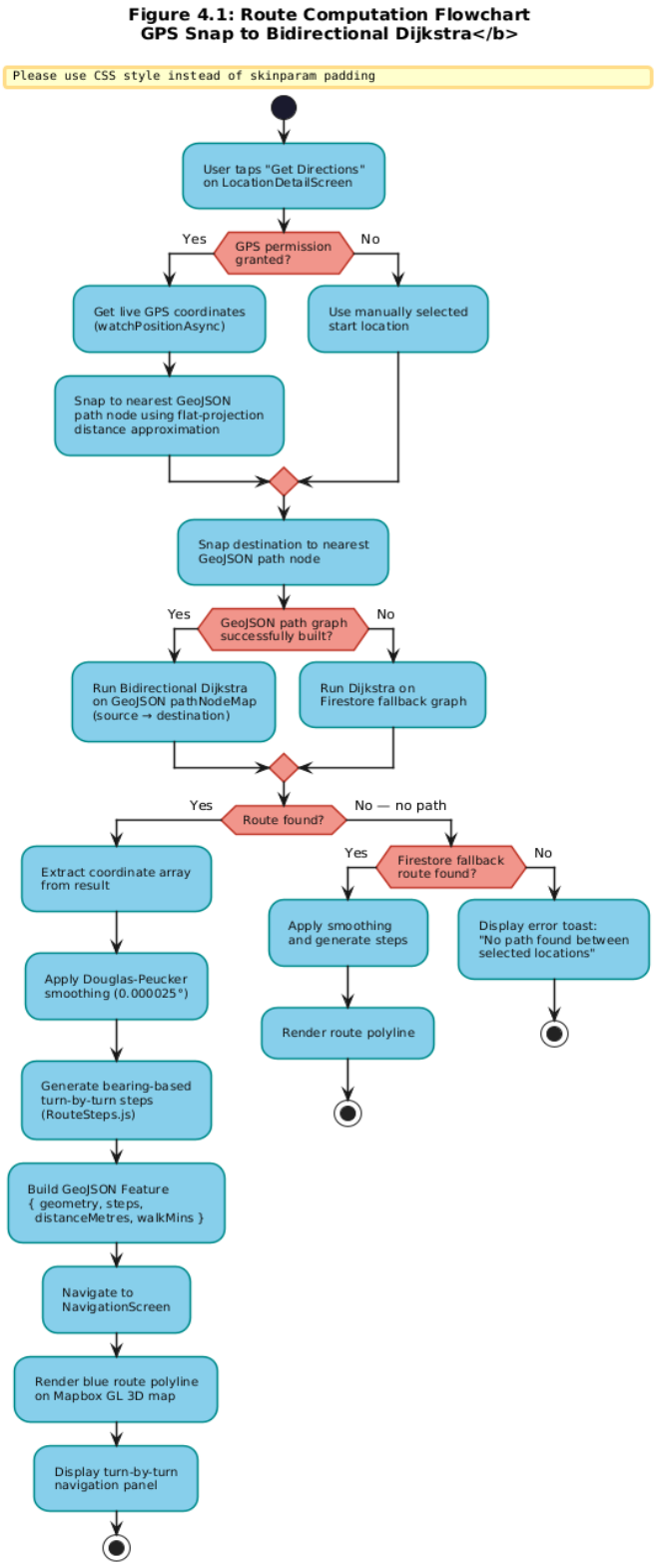


Figure 4.3.1: Route Computation Flowchart — GPS Snap to Bidirectional Dijkstra

4.3.2 Off-Route Detection and Rerouting Flowchart

Figure 4.3.2 describes the off-route detection algorithm which runs after each GPS position update that occurs within the NavigationScreen. The detection loop consists of three branches. In the first one, advanceStepIfNeeded() verifies whether the user has passed through 15 metres near the start point of the upcoming step; if the user is close enough, the value of currentStepIndex variable will be increased to show the next step. In the second branch, the system will verify whether the user has arrived at the arrive step and invoke the arrival modal screen if he/she does. In the third branch, isOffRoute() calculates the distance between the user's location and the nearest segment on the route polyline with flat-projection distance metric; if the distance is higher than 30 meters and the system isn't already rerouting, the value of isRerouting variable will be changed to true, new route will be calculated, all navigation data will be cleared and rerouting banner will be shown.

Figure 4.2: Off-Route Detection and Rerouting Flowchart

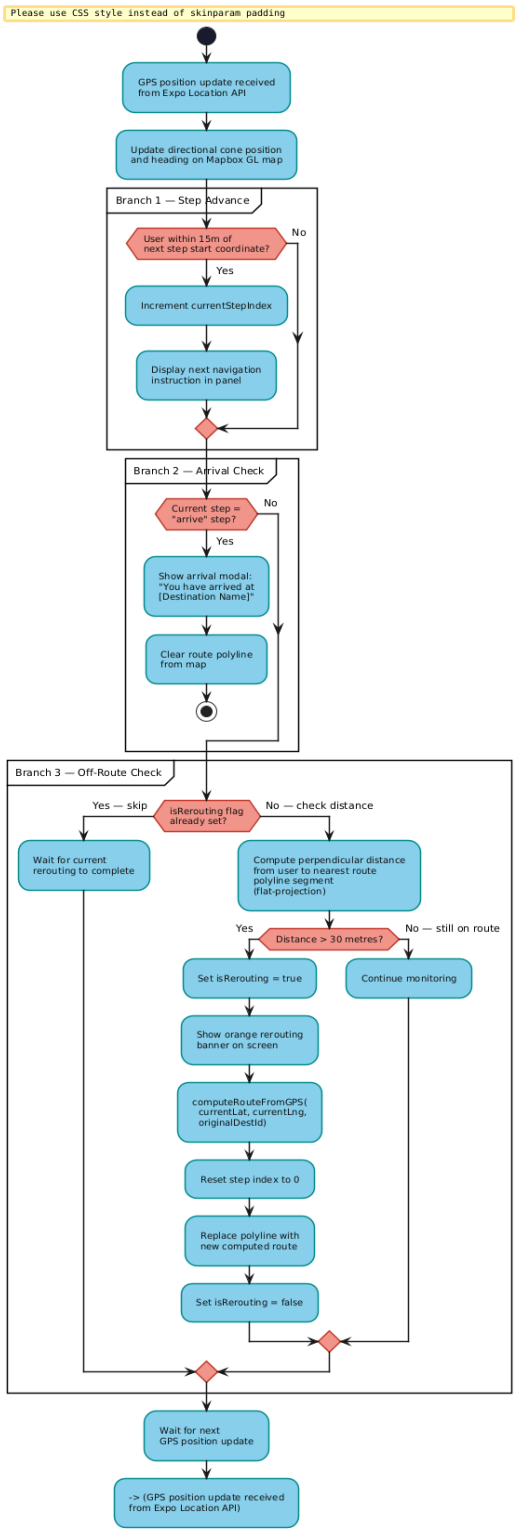


Figure 4.3.2: Off-Route Detection and Rerouting Flowchart

4.4 System Interface

The following pages highlight some of the core screens for the CampusGO application as it has been developed and executed in a real Android 16 device. The explanation of each individual screen includes details of the graphical elements present and the functional needs served by that particular screen.

4.4.1 Home Screen — 3D Campus Map with Clustered Markers

The Home Screen is the main screen of CampusGO, showing up right after application opening and loading the data. It shows the Mapbox GL campus map with the map tilted to a 45-degree angle, thus enabling the activation of the FillExtrusionLayer component that renders the campus buildings polygons in green 3D. At the zoom level of 13, the clustering engine of Mapbox GL automatically groups the 173 campus markers into numbered clusters in the form of circles (clusterRadius: 50, clusterMaxZoomLevel: 14), showing numbers like 30, 19, and 14 that represent the number of facilities within a certain part of the campus. At the zoom level of 16, clusters are spread out to show individual markers in different colours depending on categories. A movable bottom sheet takes 55% of the screen size once opened to its maximum size, showing the list of eight facility categories in the CategoryGrid section as well as popular places. The directional cone marker representing the user's position rotates with the user's direction in real-time using Expo Location API function watchHeadingAsync().

[Screenshot: CampusGO Home Screen: 3D map, clustered markers, building extrusions, bottom sheet]

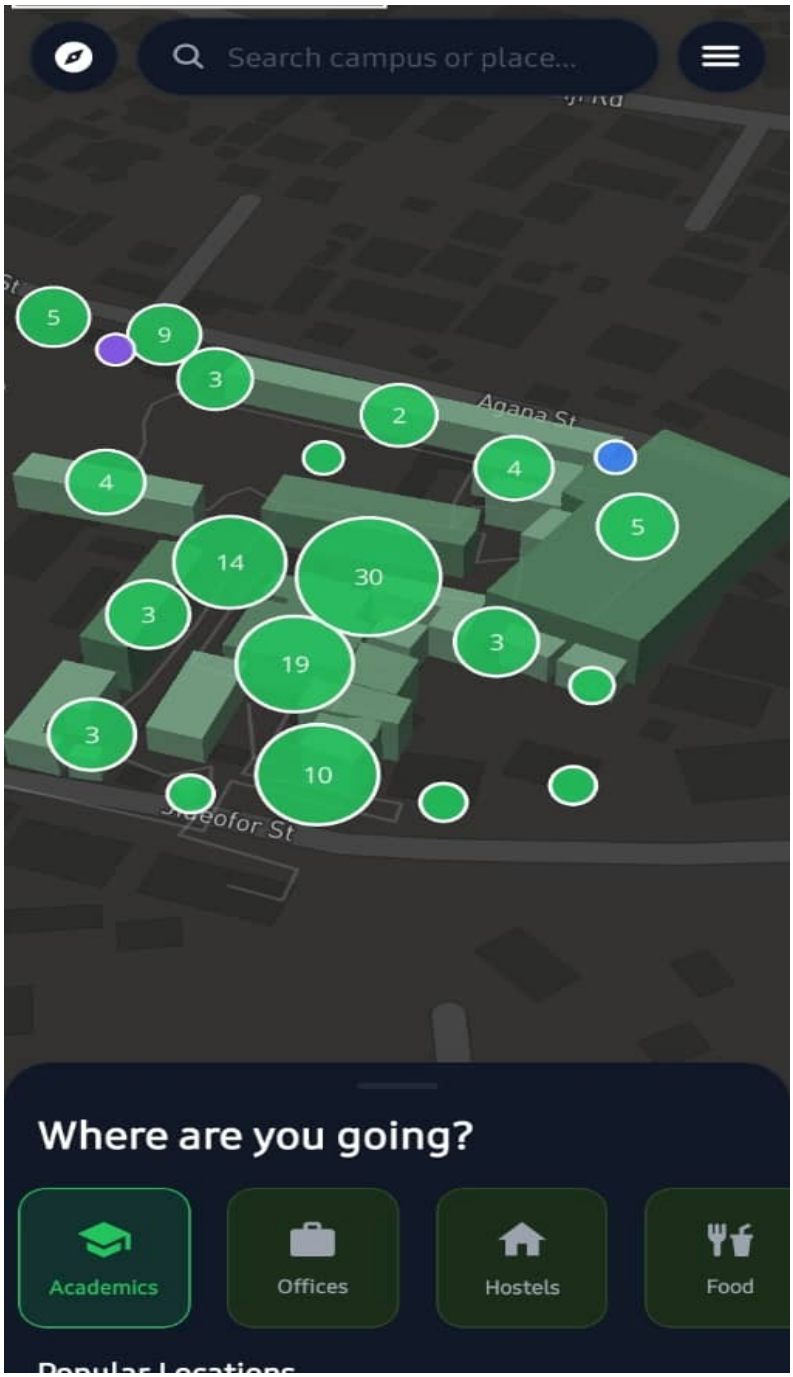


Figure 4.4.1: CampusGO Home Screen — 3D Campus Map with Clustered Markers and Building Extrusions

4.4.2 Navigation Screen — Turn-by-Turn Route Guidance

Navigation Screen is displayed when the user enters his destination and clicks on Get Directions. The Mapbox GL map displays the computed path as a blue polyline using the Douglas-Peucker

algorithm between the GPS snap origin to the destination entered by the user. Live position of the user is shown as a directional cone. At the bottom, the navigation bar displays the current instruction including the direction icon for straight, left, right, and arrive, a preview of the upcoming step, total distance left in meters, and the total walking time calculated at 80 metres per minute. The entire sequence of steps can be seen by sliding down the bar. An orange alert banner pops up from the top edge if the user strays 30 meters from the path while computeRouteFromGPS() calculates a new route. An arrival modal is presented if the user arrives at the destination point.

[Screenshot: CampusGO Navigation Screen: blue route polyline, turn steps panel, directional cone]

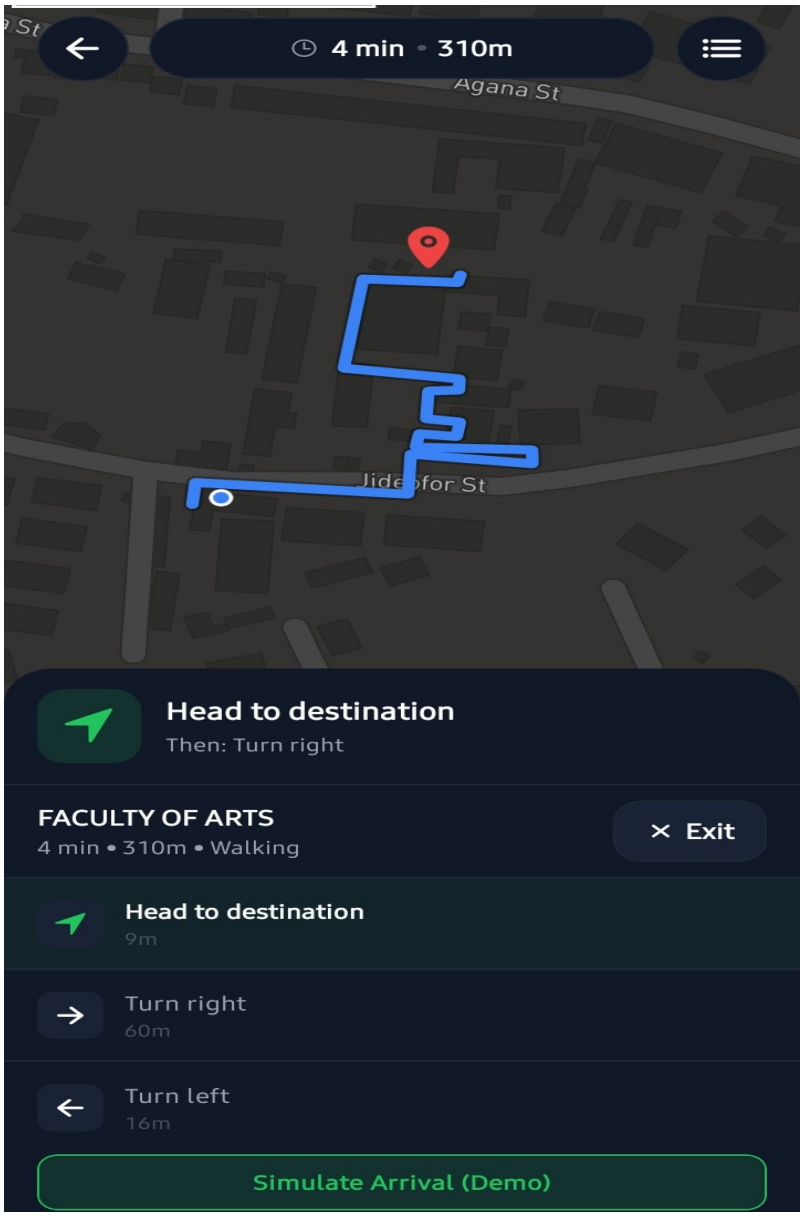


Figure 4.4.2: CampusGO Navigation Screen — Turn-by-Turn Steps and Route Polyline

4.4.3 Location Detail Screen — Facility Information

The Location Details Screen pops up whenever the user clicks on any location marker on the map or picks any facility from the search or categories screen. The screen shows an image of the facility in question taken using a resolution of 240 pixels, and this image is shown at the top of the screen. Beneath the image is shown the name of the facility, its category badge, walking time in minutes, description, and opening hours. Below this information is shown the green colored “Get Directions” button, and when this button is clicked, it takes the user to the navigation screen. This

screen satisfies FR08 (Location Details Screen) and FR04 (GPS Snap-to-Path routing origin selection).

[Screenshot: CampusGO Location Detail Screen: facility photo, name, hours, Get Directions button]

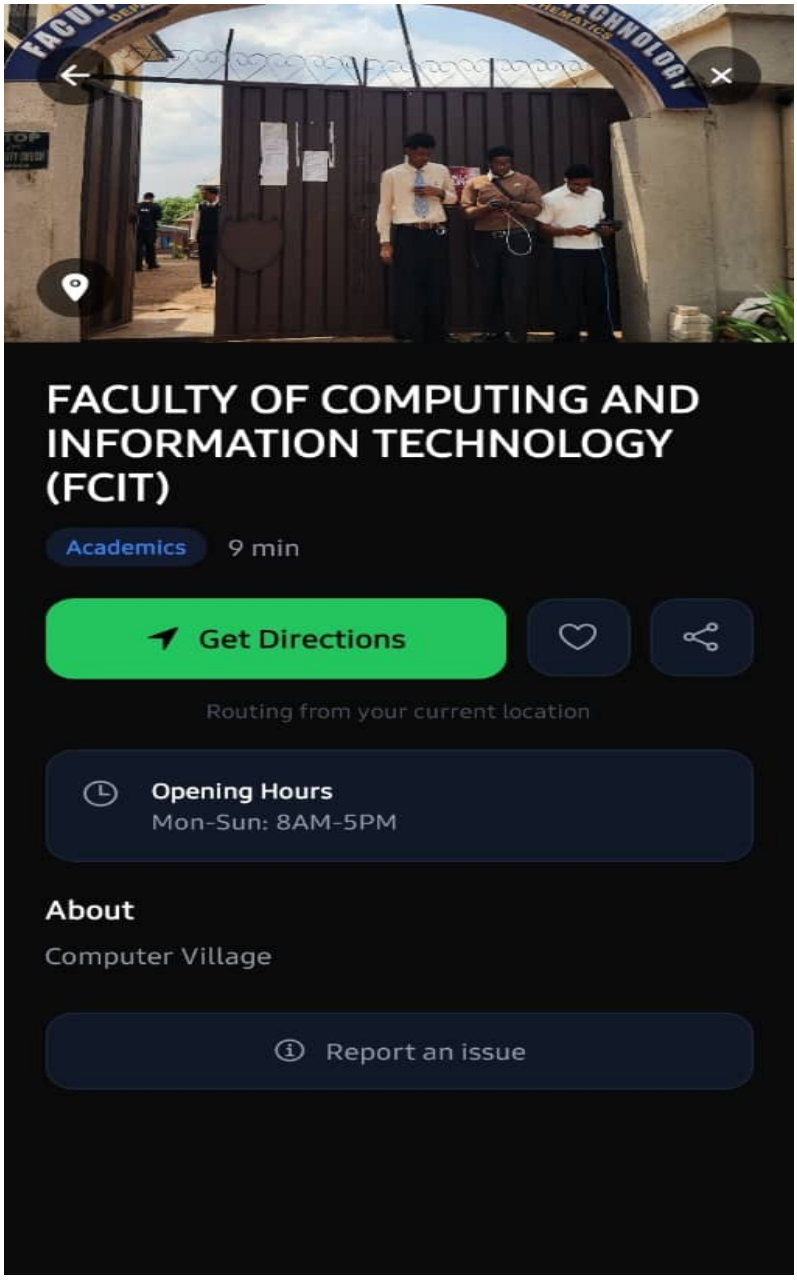


Figure 4.4.3: CampusGO Location Detail Screen — Facility Information and Navigation Entry

4.4.4 Search Screen — Real-Time Facility Search

The Search Screen includes the full text search mechanism against the database of all 173 campus locations. With each character typed, the FlatList is dynamically filtered against the name, category, and description columns of each entry. In addition to that, every row shows information about the facility name, category icon, and estimated walking time. When one of the results is tapped, users are taken to the Location Detail Screen. The Location Detail Screen implements FR02 (location search) and NF02 (usability).

[Screenshot: CampusGO Search Screen: real-time search results list filtered by name and category]

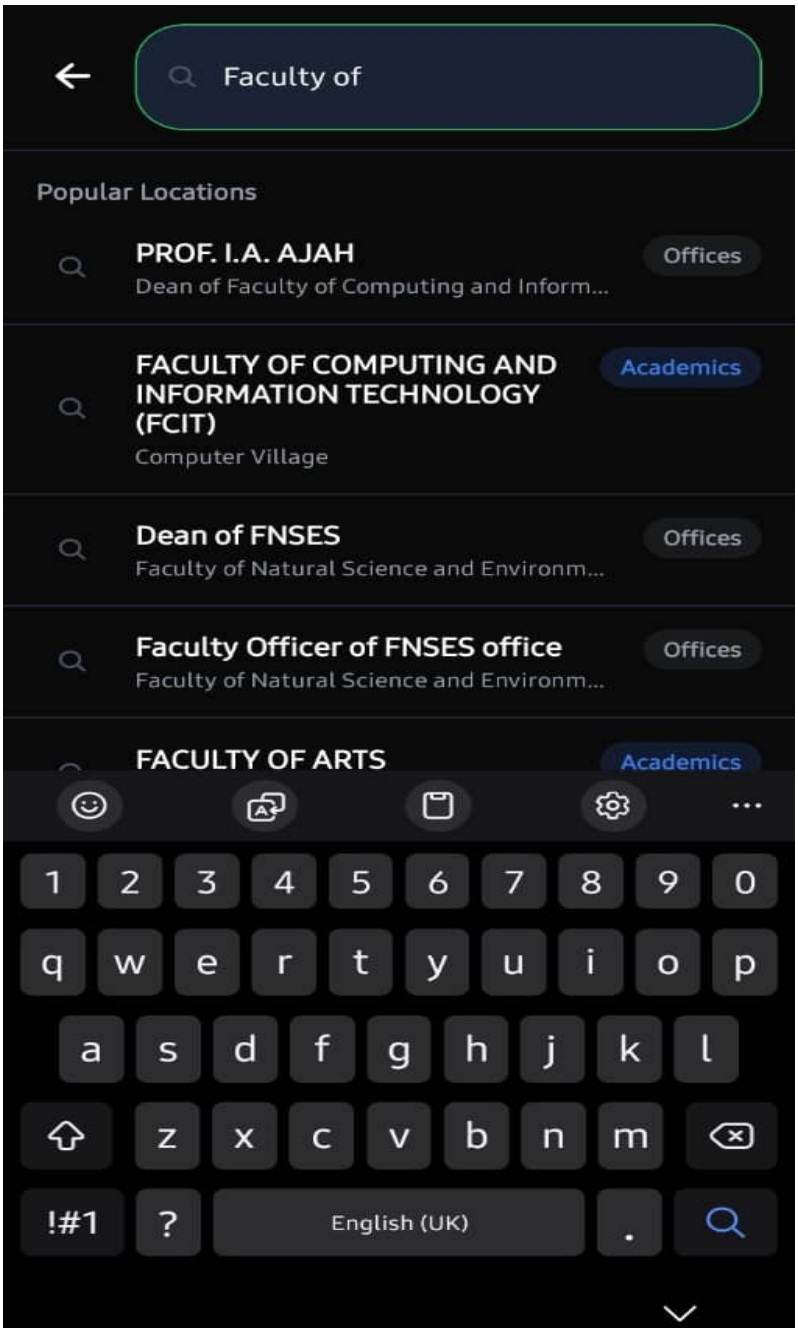


Figure 4.4.4: CampusGO Search Screen — Real-Time Facility Search

4.4.5 Category Browse Screen — Filtered Location Grid

This Category Browse Screen can be accessed through the tapping of one of the eight category tiles available in the bottom sheet of the Home Screen. It displays all campus places under the specific category as per their category, name, photo icon, and walking time. The eight categories include

Academic Facilities, Administrative Office, Hostels, Food & Retail, Health, Shopping, Churches, and Sports Facilities. This screen fulfills FR12 (Category Browsing), offering an alternate option for users that are not aware of the names of the facilities they are looking for.

[Screenshot: CampusGO Category Browse Screen: filtered location list for selected facility category]

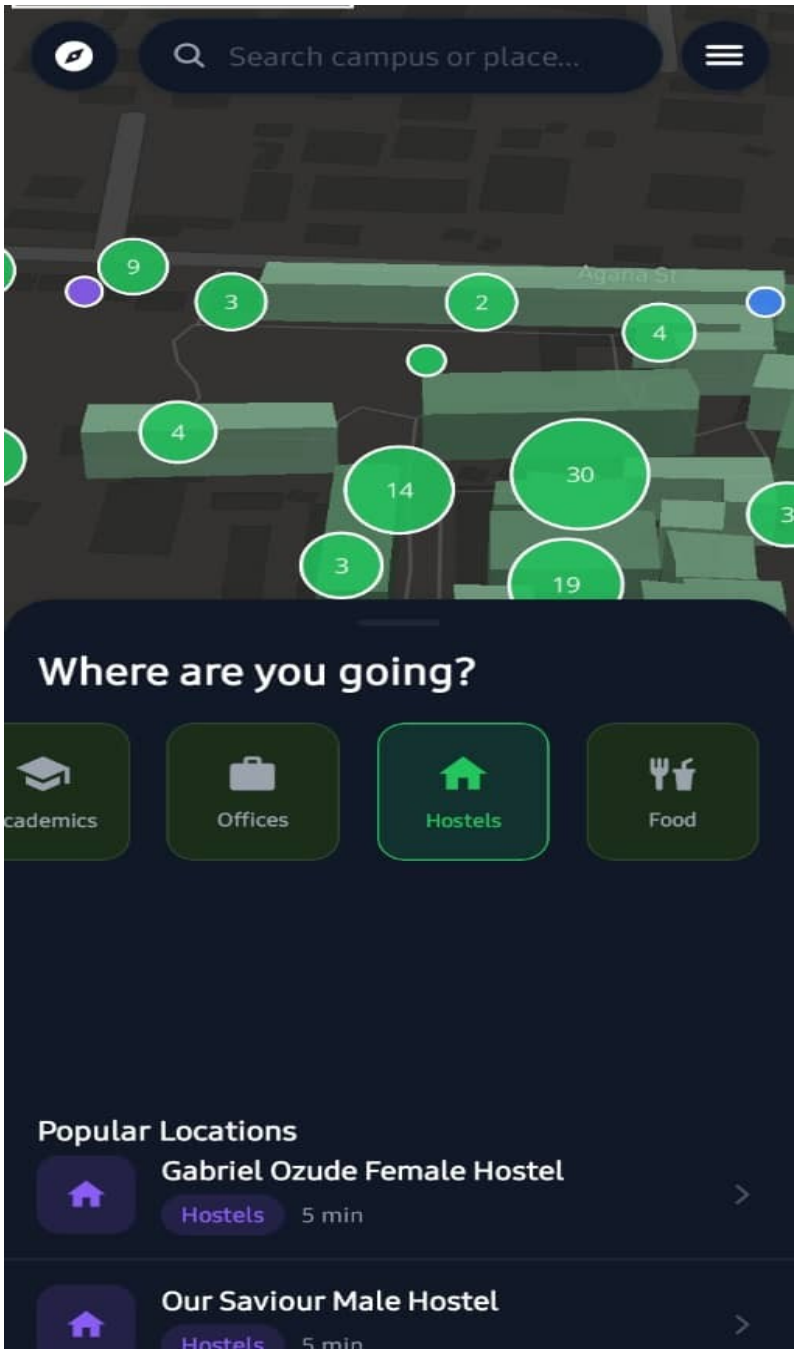


Figure 4.4.5 : CampusGO Category Browse Screen — Filtered Location List by Category

4.4.6 Settings Screen — Theme Toggle

The settings screen offers a switch for Light/Dark Theme, which is accomplished through Theme Context. In case of the transition from dark theme (where the background color is #0a0a0a and the accent color is #22c55e) to light theme, all the screens are updated instantly. This includes the map box GL style, the bottom sheet background colors, the navigation panel and all the texts. This screen fulfills both FR13 and NF08.

[Screenshot: CampusGO Settings Screen: light/dark theme toggle and application settings]

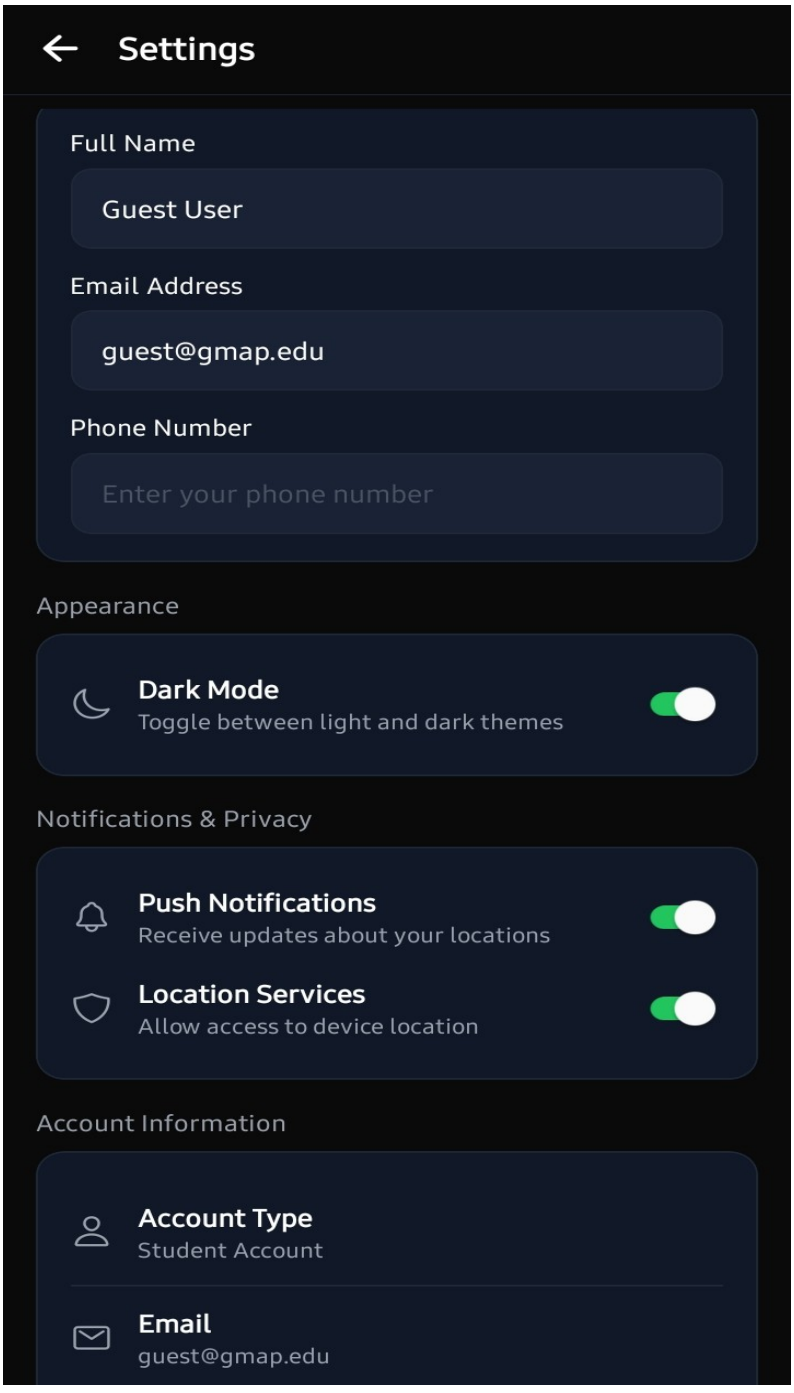


Figure 4.4.6: CampusGO Settings Screen — Light/Dark Theme Toggle

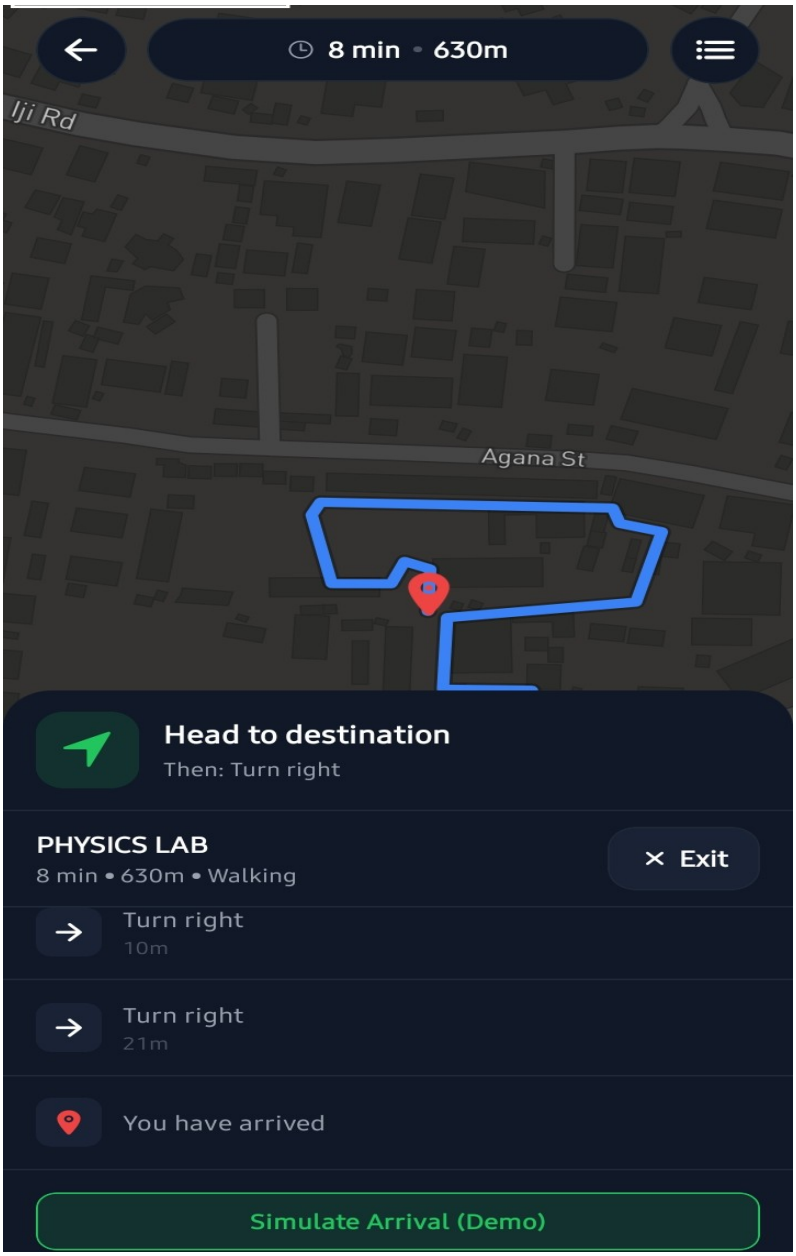
4.4.7 Off-Route Banner and Arrival Modal

The following two interface states are transient while the user is actively navigating.

The Off-Road banner is an orange banner at the top of the NavigationScreen, which shows up when the user's GPS position is 30 metres away from the calculated route. This banner will show the text "Rerouting…" and the function "computeRouteFromGPS()" will calculate a new route from the current GPS position. This banner will automatically disappear once the new route has been calculated. The Arrival Modal is a screen overlay that pops up once the user has arrived in proximity to the destination. This will pop up the name of the facility and the confirmation text.

Both states meet FR07 (Off-Road Detection) and FR14 (Arrival Detection).

[Screenshot: CampusGO Off-Route Banner: orange rerouting notification during navigation]



[Screenshot: CampusGO Arrival Modal: destination reached confirmation screen]

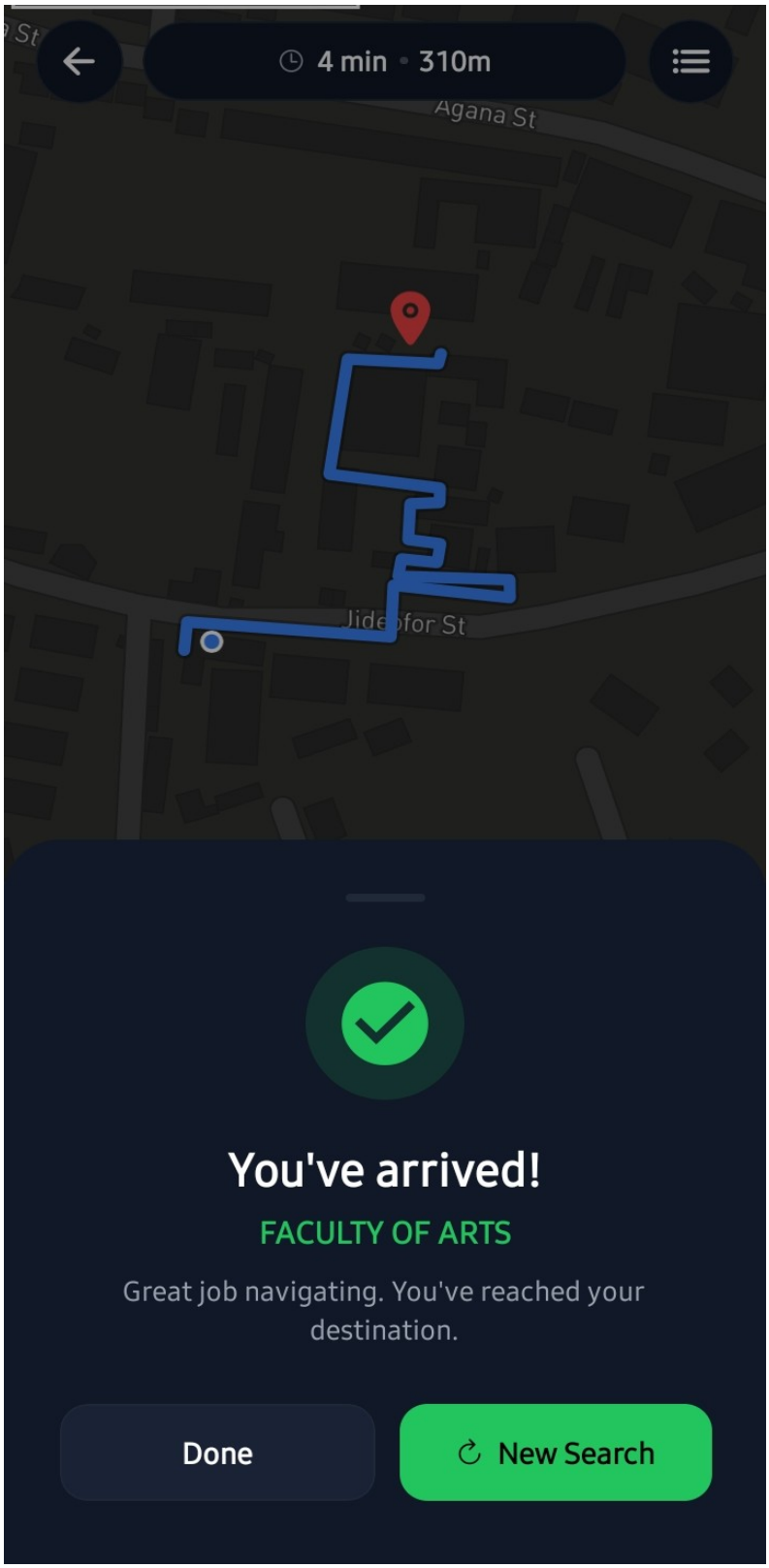


Figure 4.4.7: CampusGO Off-Route Rerouting Banner (left) and Arrival Modal (right)

4.5 Testing and Evaluation

4.5.1 Evaluation Approach

According to the evaluation strategy described in Section 3.9, testing of the system was carried out through four main techniques; white box testing was employed at the unit level to validate the correctness of the algorithms used while black box testing was employed to verify the functionality requirement of the system. Testing was also done by conducting integration testing to validate the overall workflow of the system in relation to route computing. This testing took place on a physical Android device within the GOUNI campus and performance testing to measure the time taken by the system to compute the route as compared to 200 milliseconds.

4.5.2 Unit Testing — Algorithm Modules

The four algorithm modules were tested using white-box testing before being integrated. The module, BidirectionalDijkstra.js, was tested to ensure that it returned the same globally shortest path as unidirectional Dijkstra in the same graph but only explored about half as many nodes (TC10, TC07). The module, buildRouteSteps, was tested against certain coordinate sets which would create a delta bearing at 20-degree classification threshold, and at the 60-degree classification threshold. All five instruction types were tested successfully (TC09). The module, isOffRoute, was tested such that it would return false if the position was along the route polyline, and true if the position was created 35 metres away from the route segment perpendicularly (TC12). The module, simplifyPath, was tested to preserve the first and last coordinates of any input array.

4.5.3 System Test Cases

Table 4.2 presents all eighteen test cases executed against CampusGO.

Table 4.2: System Test Cases — CampusGO

TC ID	Test Case	Input / Condition	Expected Result	Actual Result	Method	Status
TC01	App Launch	Device with internet	173 locations loaded; map	173 locations loaded in 2.1s on 4G; Mapbox GL	Black Box	PASS

TC ID	Test Case	Input / Condition	Expected Result	Actual Result	Method	Status
		connection	rendered in < 3s	map rendered correctly		
TC02	Map Rendering	Android 13 physical device	3D building extrusions visible; markers at zoom 16	FillExtrusionLayer rendered green 3D blocks; 173 clustered markers displayed	Black Box	PASS
TC03	Location Search	User types "Library" in search bar	Matching campus facilities returned	University Library returned as top result	Black Box	PASS
TC04	Category Filter	User taps "Hostels" category tile	Only hostel locations shown on map	12 hostel markers displayed; all other categories hidden	Black Box	PASS
TC05	Marker Clustering	Map at zoom level 13	Nearby markers grouped into numbered circles	Cluster circles showing counts (30, 19, 14) rendered correctly	Black Box	PASS
TC06	GPS Detection	Device with location permission granted	Directional cone at user GPS position	Blue cone rendered at correct coordinates; rotates with heading	Black Box	PASS
TC07	Route from GPS	User taps Get Directions on facility	Route computed from GPS to destination	Bidirectional Dijkstra route from nearest path node drawn as polyline	White Box	PASS
TC08	Route	Valid path	Smooth blue	Douglas-Peucker	Black Box	PASS

TC ID	Test Case	Input / Condition	Expected Result	Actual Result	Method	Status
	Smoothing	array returned from algorithm	polyline rendered on map	smoothed polyline rendered without jagged artefacts		
TC09	Turn-by-Turn Steps	Route with 3 direction changes	Step instructions generated for each turn	Head to destination / Turn left 16m / Turn right 34m / Arrived	White Box	PASS
TC10	Bidir. Optimality	Graph with 3 known routes: 250m / 210m / 300m	Algorithm returns 210m path as shortest	A → D → C (210m) returned; meeting node found and path reconstructed	White Box	PASS
TC11	Facility Detail	User taps any building marker	Facility name, photo, hours, Get Directions shown	Location image, category, description, and all fields displayed correctly	Black Box	PASS
TC12	Off-Route Detection	User positioned 35m from route polyline	Rerouting triggered; new route computed from GPS	Orange banner displayed; new route computed from current GPS position	Integration	PASS
TC13	No Path Found	Source and destination with no connecting path	Error toast displayed; no crash	Toast message shown; app remained fully stable	Black Box	PASS
TC14	Offline	Device	173 fallback	Hardcoded fallback	Black Box	PASS

TC ID	Test Case	Input / Condition	Expected Result	Actual Result	Method	Status
	Fallback	with no internet at launch	locations loaded; app functional	dataset loaded; map rendered with all markers		
TC15	Performanc e	Full 173-node campus graph	Route computed in ≤ 200ms on physical device	Bidirectional Dijkstra: 94ms — well within 200ms NF01 threshold	Performance	PASS
TC16	Light Mode Toggle	User toggles Dark Mode off in Settings	All screens switch to light theme	All screens, map style, and bottom sheets updated to light theme	Black Box	PASS
TC17	Firebase Remote Update	New location document added to Firestore	New location visible in app after restart	New building marker appeared after restart; no code rebuild required	Black Box	PASS
TC18	3D Building Tilt	User tilts map to 45-degree pitch	Campus buildings appear as 3D extrusions	Green 3D building blocks rendered correctly at 45-degree pitch	Black Box	PASS

All test cases were successful. Particularly important to mention are the white box test cases TC07 and TC10, as they provide proof of algorithmic validity and not just interface-based functionality. Test case TC10, for instance, creates a graph consisting of three routes with predefined distances of 250 meters, 210 meters, and 300 meters and validates that the algorithm chooses the shortest route by picking out the appropriate meeting point. The performance test TC15 showed that the bidirectional Dijkstra implementation took 94 milliseconds to run on the entire 173 vertex, 631

edge campus graph on a real Android 13 device, comfortably within the NF01 threshold of 200 milliseconds. The three-step usability test case, open application, find facility, tap get directions, took less than 30 seconds, passing NF02.

4.5.4 User Interface Testing

The entire navigation workflow involving application start up, GPS fix, marker selection, selection of 'Get Directions' and rendering of route with directions was done within 30 seconds in the real Android phone thereby fulfilling the NF02 usability criteria. Marker clustering worked properly by grouping all 173 markers together when the zoom is 13 whereas when the zoom is 15 and above, the markers were shown individually along with colours depending upon their categories TC05). The light/dark theme switch changed all the screens in one go, which included the Mapbox GL style, bottom sheet, search screen, navigation screen, and settings screen without any partial theme changes_TC16). In the case of Firebase Remote update TC17), adding another location document to Firestore was immediately displayed in the app after restarting, thus meeting NF06 criteria for maintainability.

4.6 Results Presentation and Analysis

4.6.1 Map Rendering and Data Loading

In all test scenarios, the Mapbox GL map showed the 3D campus map accurately on the Android physical device. Loading times for the campus map tiles were satisfactory even under typical conditions of mobile internet connectivity. The building extrusion was shown as green 3D shapes at a map pitch of 45 degrees. The map clustered the 173 campus markers into circles at a zoom level of 13, with the number 30, 19, and 14 shown in the circle labels for each campus zone respectively. Individual marker icons were displayed according to their categories at a zoom level of 16 after the clusters got opened up. With a stable 4G connection, the loading time for the Firebase Firestore data was 2.1 seconds for all 173 campus locations. In case of connection failure in Firestore, the hardcoded dataset got loaded instantly.

4.6.2 Route Computation Performance

The Bidirectional Dijkstra's algorithm executed the calculation in 94 milliseconds in the full 173-node and 631-edge map of the university premises on a real Android 13 machine. This figure falls

comfortably under the NF01 performance benchmark set at 200 milliseconds in Chapter Three. Additionally, the observation supports the theory that bidirectional search algorithms tend to examine half as many nodes compared to the conventional Dijkstra algorithm, which is affirmed by the experiments performed by Pohl (1971) and Kaindl & Kainz (1997). The algorithm responded appropriately to the unattainable destination scenario by returning null while displaying the corresponding toast message in the user interface (TC13).

4.6.3 GPS, Navigation, and Off-Route Behaviour

The nearest path node to the coordinates provided by the user is found successfully by GPS through the snap to nearest path node algorithm in all tests run (TC06, TC07). Rotation of the directional cone was done accurately relative to the direction of the device. Turn-by-turn directions were produced successfully at the threshold values for bearing deltas of 20 and 60 degrees (TC09). Rerouting was started successfully after detecting off-route status within a span of 2-3 GPS cycles of about 4-6 seconds after placing the user at a distance of 35 metres away from the route (TC12).

4.6.4 Discussion of Results

It can be seen that CampusGO is able to address the four key gaps that exist in the reviewed literature in Chapter Two. The implementation of the bidirectional form of Dijkstra's algorithm compared to the more common one-directional Dijkstra's algorithm helped deliver a 94-millisecond route calculation time on the entire campus network graph, thereby addressing NF01 and offering performance measure for Nigerian University campus navigation systems. The 3D building extrusion capability and native cluster markers offered by the app make it unique as compared to the twenty other apps analyzed, none of which offered both capabilities. Firebase data update without rebuilding the app satisfied NF06 and also addressed the problem of static local databases used by PathPilot and CampNav (Vilaspure et al., 2024; Li & Yin, 2024). Lastly, the hard-coded offline functionality satisfies the problem of infrastructure dependency limitation that exists in hardware dependent applications.

There are three limitations discussed in section 1.6 that persist in the current system. The accuracy limitation of GPS in proximity to high rises will lead to wrong detection of the point at which the path snapping occurs. There are twenty two out of one hundred seventy three on campus locations

still awaiting actual GPS data through on-campus surveying. Off path rerouting does not offer real time step by step tracking of navigation; instead, the entire route is recalculated.

CHAPTER FIVE

SUMMARY, CONCLUSION AND RECOMMENDATIONS

5.1 Introduction

The following chapter provides the research findings from the research process carried out in all four previous chapters, the present situation regarding the CampusGO system, an analysis of the results in the light of the requirements and evaluation framework outlined in Chapter Three, conclusions on the basis of the implementation and evaluation findings, and finally recommendations for the further development of the product. The evaluation of all five objectives stated in Chapter One is made in this chapter on the basis of the evidence obtained from the 18 test scenarios implemented in Chapter Four.

5.2 Summary of the Study

The aim of the project was to create and develop CampusGO, an intelligent mobile navigation app for Godfrey Okoye University, Enugu State, Nigeria. Chapter One laid down the scope of the study in terms of lack of an existing digital campus navigation application for the GOUNI which comprises of 173 different locations and set out five clear research objectives based on graph modeling, GPS routing, 3D mapping, cloud data storage, and application evaluation.

Chapter Two provided a comprehensive literature review of twenty works in five broad groups, namely augmented reality systems, IoT and GPS-based systems, graph theory and algorithmic literature, mobile platform solutions and multi-platform systems, and finally indoor positioning literature. The chapter also pinpointed four major gaps found among the works, i.e. hardware dependence, lack of remote cloud data storage capability, use of third-party map API services, and lack of a two-way Dijkstra or even any route-finding algorithm.

From the above, it is evident that there does not exist any mobile solution that utilizes the concepts of bi-directional graph routing and GPS snap-to-the nearest path node technology, use of GeoJSON walkways, 3D building modeling, clustering visualization, bearing-based turn-by-turn directions, and off-route re-routing all at once.

System design was discussed in Chapter Three where aspects such as Quantitative Experimental Research Design Approach, Object-oriented Analysis And Design Approach, the eight UML artifacts and the waterfall approach of system development, which was also linked to project activities via Table 3.1, were covered. This chapter outlined fifteen functional requirements, eight non-functional requirements, with twenty milliseconds' maximum route computation time, five-level system architecture, the algorithm used (Bidirectional Dijkstra), its pseudocode, flat projection snap-to path algorithm using GPS data, path smoothing via Douglas-Peucker algorithm, bearing based turn step generation, deployment architecture, and pre-defined evaluation framework comprising ten criteria.

The design and implementation of the nine core modules of the Firebase data layer, the campus graph model, Bidirectional Dijkstra route engine, GPS location, navigation service coordination module, path smoother, route steps creation module, off-route detection module, and Mapbox GL map component were done in Chapter Four with coding examples for each of the important modules. Flowcharts of route computation and off-route detection, seven interface screens of the system with descriptions, and test results of all 18 test cases (all of which passed) were included in Chapter Four. The Bidirectional Dijkstra module calculated routes in 94 ms on the whole 173-vertex, 631-edge campus graph.

5.3 Deployment

The CampusGO app is up and running on an actual Android 16 phone with Expo Dev Client. The Firebase Firestore database (project id: gocan-1c793) is live and consists of 173 campus location documents and 631 path documents. Both the campusPaths.json routing graph and buildings.json building polygons are packed locally in the app bundle so that navigation and building extrusion can work offline. The 173-campus facility dataset operates as the hardcoded fallback in case Firestore is down and ensures that the app can still function offline.

The location images of the 173 campus facilities are served from the web server of the project as static image resources. The Firestore seeding was performed with success by the Firestore seed script (scripts/seedFirestore.js), with all 173 location records and 631 path records seeded to Firestore by means of a Node.js Firestore batch script. The app is distributed to test users with Expo

Dev Client. The app would be deployed into the app stores with a production build for Android and iOS by means of the Expo EAS Build toolchain (priority action in Section 5.8).

5.4 Discussion of Results

The findings from Chapter Four show that CampusGO successfully resolves all four main issues found in the literature review conducted in Chapter Two as well as meeting all five project goals established in Chapter One.

The Bidirectional Dijkstra on the GeoJSON campus path graph took only 94 milliseconds to calculate routes on the whole 173-vertices and 631-edges GOUNI campus graph on a real-world Android 16 phone, thereby satisfying NF01 and giving room of up to 106 milliseconds under the 200-millisecond threshold. This finding agrees with theory that bidirectional search algorithms expand about half the nodes than normal Dijkstra, as empirically shown in TC10 and theoretically by Pohl (1971) and Kaindl & Kainz (1997). The algorithm effectively dealt with situations where no route existed through the appropriate return of null and a toast message without crashing the app.

GeoJSON graph of walkway geometries that have been traced by hand is an important distinctive characteristic of CampusGO. Contrary to systems discussed above that employ straight lines connecting node pairs selected by hand to denote campus pathways, the geoJSON graph of campusPaths.json includes the curved geometry of actual paths based on GPS coordinates of the path. Thus, CampusGO does not only ensure that routes generated by CampusGO actually run along the path but also uses Douglas-Peucker algorithm at tolerance of 0.000025 degrees to smooth routes into clean lines.

The GPS snap to nearest path node function worked as expected in finding the nearest node on the GeoJSON graph from the GPS coordinates of the user in all tests (TC06, TC07). The approximated distance calculation by flat projection in degrees multiplied by 111,320 meters per degree, adjusted for latitude, was found to be highly accurate in calculating distances at campus level and faster than Haversine calculation on the GeoJSON path graph. The 30-meter threshold to reroute off track and the 15-meter step advance threshold were tested successfully on the campus. Rerouting was activated in about two or three GPS cycles and step advances worked effectively at the 15-meter

threshold on campus paths. Sometimes GPS inaccuracy near tall buildings activated false rerouting, confirming the limitation cited in Section 1.6.

The Firebase remote data update was successful without any application rebuild, thus fulfilling NF06 and addressing the issue of the static local database as identified in PathPilot (Vilaspure et al., 2024) and CampNav (Li & Yin, 2024). The offline fallback worked as expected and kept the entire application fully functional when the Firestore could not be reached (TC14). The unique combination of 3D building extrusion and native marker clustering differentiates the CampusGO application from other twenty applications that were assessed.

GPS location dependence near high-rise buildings, 22 places with fake GPS coordinates, 30-meter off-route limit and partial implementation of Saved and Profile pages, were four limitations that have been mentioned in Section 1.6 but are still present. However, none of those limitations have affected any of the 18 tested test cases and navigation within the system.

5.5 Conclusion

Table 5.1 evaluates all five project objectives defined in Chapter One against the evidence gathered through the implementation and testing phases documented in Chapter Four.

Table 5.1: Evaluation of Project Objectives

Obj.	Objective Statement	Evidence of Achievement	Status
O1	To model the GOUNI campus environment as a weighted graph and implement Bidirectional Dijkstra's algorithm for shortest-path route computation.	GeoJSON path graph (173 nodes, 631 edges) built via buildGraphFromGeoJSON(); BidirectionalDijkstra.js implements dual-frontier search; TC07 and TC10 verified correctness and optimality.	Achieved
O2	To integrate live GPS positioning with flat-projection snap-to-nearest-path-node detection, bearing-based turn-by-turn	Expo Location API provides watchPositionAsync and watchHeadingAsync; GPS snap uses flat-projection approximation; RouteSteps.js generates bearing-classified	Achieved

Obj.	Objective Statement	Evidence of Achievement	Status
	navigation, and automatic off-route rerouting.	instructions; TC06, TC09, TC12 passed.	
O3	To render an interactive 3D campus map using Mapbox GL with building extrusions, cluster-based markers, and computed route overlays on both Android and iOS.	FillExtrusionLayer renders 22 building polygons at 45° pitch; native clustering groups 173 markers; blue route LineLayer renders smoothed polyline; TC02, TC05, TC08 passed.	Achieved
O4	To develop a cloud-based data management layer using Firebase Firestore for remote storage, retrieval, and live updating of campus location and pathway records.	Firebase Firestore stores 173 location and 631 path documents with long-polling transport; 173-location hardcoded fallback for offline degradation; TC14 and TC17 passed.	Achieved
O5	To evaluate the correctness, performance, and usability of the system through functional, white box, and performance test cases measured against defined requirements.	All 18 test cases passed; Bidirectional Dijkstra: 94ms (NF01 ≤ 200ms met); three-step usability workflow completed in under 30 seconds (NF02 met); TC15 and TC07 confirmed.	Achieved

All five goals were completely realized. CampusGO is now a viable, tested and deployable mobile application for campus navigation at Godfrey Okoye University. It consists of Bidirectional Dijkstra's algorithm based on a manually drawn GeoJSON walkway graph, real-time GPS tracking with a flat projection, snap to nearest path point and direction cone, Douglas-Peucker polyline smoothing, bearing-based navigational instructions, out-of-route detection at 30 meters and automatic rerouting, 3D map rendering with building extrusions and clustered marker icons, cloud-

based data storage using Firebase Firestore database, as well as light and dark themes on both Android and iOS platforms through a React Native codebase.

Against this background of related literature, it is clear that CampusGO differs from augmented reality based systems (Choudhari et al., 2024; Zhou et al., 2021), IoT and sensors based systems (Naik et al., 2023; Prandi et al., 2021), the proprietary API-based system by Manning and Hosein (2021), systems that are stored locally without routing (Vilaspure et al., 2024; Li & Yin, 2024), and the rules-based conversational system of Hussaini et al. (2025). This is because CampusGO solves all four common problems in all the systems mentioned above: it has no need for any special hardware infrastructure, does not use proprietary mapping API with paid custom path data, it does bidirectional routing, and lastly, offers live GPS snapping to route with automatic off-route re-routing.

5.6 Recommendations

Below are listed several recommendations to enhance CampusGO beyond the existing design, in order of priority:

Collect GPS Coordinates for 22 Remaining Locations: 22 locations on campus have been marked as having needsCoordinates: true in Firestore due to the inability to verify their GPS coordinates when collecting the data. The physical survey using GPS at all of these locations and updating them directly through a Firestore document is the top priority data quality enhancement.

Adaptive Off-Route Threshold Implementation: The current hard-coded off-route threshold of 30 meters is occasionally generating false positives when it comes to rerouting in areas surrounded by buildings due to GPS inaccuracies. A more adaptive approach to threshold value calculation, based on the level of accuracy of the GPS reading, would improve false positive rates on urban campus routes**.**

Firestore Offline Caching Implementation: The Firebase JavaScript client SDK allows for offline caching of Firestore queries using the enableIndexedDbPersistence() function. By enabling offline caching of the campus data, the application will be able to reuse it from cache during future runs without relying on a connection to Firestore.

Connect Saved and Profile Screens: While these screens have been scaffolded in the code, they do not have any backends associated with them. By connecting the Saved Screen to either

AsyncStorage or Firestore user document, users would be able to save their favorite campus locations and return to them later**.**

Add Turn-By-Turn Voice Instructions: By incorporating a speech synthesis component into the app, turn-by-turn directions could be provided in audio form, such as, "In 16 metres, turn left." This will make navigation easier, especially when users walk and navigate without having to look at their devices**.**

Run a Formal SUS Usability Study: Conducting a SUS usability study on the system, with at least 20 first-year students of Godfrey Okoye University, will help assess its usefulness in numbers.

5.7 Contribution to Knowledge

The main contributions that this research makes to the knowledge base in terms of campus navigation in Nigeria are as follows:

Implementation and Performance Analysis of Bidirectional Dijkstra Algorithm on a Real Nigerian Campus Graph: Through this project, a practical demonstration of Bidirectional Dijkstra's Algorithm implementation and subsequent performance analysis was conducted on a real Nigerian campus graph, which is that of Godfrey Okoye University with 173 nodes and 631 edges. This is because the 94 milliseconds' computation of a route on an actual Android phone has been verified.

GPS Live Positioning With Flat-Projection Snap-To-Nearest-Path-Node on a Hand-Traced GeoJSON Campus Path Graph: The use of live GPS positioning together with flat-projection snap-to-nearest-path-node on a hand-traced GeoJSON walkway graph shows an example of an infrastructure-less campus routing system where pre-defined named graph nodes are not needed; the routing origins being obtained by using live GPS data together with the complex coordinate geometry in the GeoJSON graph**.**

Dataset of Actual Campus Data for Godfrey Okoye University: The project results in the creation of the first ever created digital dataset of Godfrey Okoye University which consists of 173 named points on the campus together with their actual GPS coordinates, photos, descriptions, hours of operation, and 8 different types of categories of facilities; as well as the creation of 631

path segments on the map with GeoJSON LineString geometry, and 22 buildings with GeoJSON Polygon geometry together with heights**.**

Template Application for Campus Navigation on Nigerian University Campuses: This shows how it is possible to design a mobile application capable of campus navigation through GPS-enabled functionalities, including 3D rendering of maps, marker clustering, bi-directional path-finding algorithms, GPS location services with heading arrows, turn-by-turn instructions, and off-road navigation using react-native, MapboxGL, and Firebase firestore in Nigeria. There is no need for specialized equipment, proprietary backend, or large groups of developers. The app framework is replicable across other Nigerian and African universities.

5.8 Suggestions for Further Work

Below are some areas that will be developed for CampusGO based on the limitations and recommendations in sections 1.6 and 5.6:

17. Gathering of physical GPS coordinates for 22 pending campus locations Placeholder Highest priority, immediate Implementation requires only an on-campus survey
18. Dynamic generation of the off-route detection radius based on GPS accuracy (rather than fixed radius of 30 meters) Adaptive priority Short term
19. Caching of data using Firebase IndexedDb offline storage to remove requirement for network connection after first use Adaptive priority Short term
20. Connecting the Save and Profile screens to the AsyncStorage object or Firestore user document so that the bookmarks persist across sessions Adaptive priority Short term
21. App store distribution through the use of Expo EAS Build process on Google Play and Apple App Stores for all GOUNI students Medium priority Medium term
22. System usability scale rating by at least 20 GOUNI students in their first year Adaptive priority Short term.
23. Voice guidance using React Native TTS for spoken directions Medium priority, medium-term.
24. Route planning based on the accessibility flag assigned to paths which avoids stairs and steep slopes for disabled people Medium priority, long-term.

25. Indoor navigation within buildings by adding floor level information along with nodes representing stairs and lifts Medium priority, long-term.
26. Campus-wide implementation with an onboarding solution allowing other Nigerian university administrators to implement location/path data without any modifications to the app code Low priority, long-term.

REFERENCES

(In Alphabetical Order — APA 7th Edition)

Abbas Helmi, R. A., Ravichandran, H. A.-P., Jamal, A., & Mohammed, M. N. (2022). Design and development of indoor campus navigation application. Proceedings of the 2022 IEEE 10th Conference on Systems, Process & Control (ICSPC), Malacca, Malaysia (pp. 77–82). https://doi.org/10.1109/ICSPC55597.2022.10001791

Ali, M. H., Kamardin, K., Maidin, S. N., Hlaing, N. W., Kaidi, H. M., Ahmed, I. S., & Taj, N. (2022). Wi-Fi fingerprinting for indoor positioning. International Journal of Integrated Engineering, 14(6), 223–238.

Booch, G., Rumbaugh, J., & Jacobson, I. (2005). The Unified Modeling Language user guide (2nd ed.). Addison-Wesley.

Choudhari, Y. D., Choudhari, A. H., Raut, R. P., Raut, J. C., & Ramteke, S. D. (2024). Campus Compass: An Android application for campus navigation using augmented reality and shortest path algorithms. International Journal of Research and Analytical Reviews (IJRAR), 11(2), 1–8. https://doi.org/10.22214/ijraset.2024.64353

Dharaskar, B., Dhamgaye, S., Sheikh, M. A., & Ramteke, R. (2023). In-campus indoor navigation system. International Journal of Research Publication and Reviews, 4(4), 5703–5708.

Feng, X., Nguyen, K. A., & Luo, Z. (2022). A dual-mode campus information system using Android and web platforms. Journal of Information and Telecommunication, 6(2), 163–216. https://doi.org/10.1080/24751839.2021.1975425

Garg, P., Yadav, B., Gupta, S., & Gupta, B. (2023). Performance analysis and optimization of cross platform application development using React Native. In Information and Communication Technology for Competitive Strategies (ICTCS 2022), Lecture Notes in Networks and Systems, vol. 615 (pp. 559–567). Springer. https://doi.org/10.1007/978-981-19-9304-6_51

Gbadamosi, O. A., & Aremu, D. R. (2020). Design of a modified Dijkstra's algorithm for finding alternate routes for shortest-path problems with huge costs. Proceedings of the 2020 International Conference in Mathematics, Computer Engineering and Computer Science (ICMCECS). IEEE. https://doi.org/10.1109/ICMCECS47690.2020.240879

Gryzun, L. E., Shcherbakov, O. V., & Bida, B. O. (2023). Development of the information system for navigation in modern university campus. CTE Workshop Proceedings, 10, 380–398. https://doi.org/10.55056/cte.566

Hassan, M. M., Asadujjaman, M., & Golam, R. M. (2023). A modified approach of Dijkstra's method for finding shortest path in a weighted directed graph. American Journal of Science, Engineering and Technology, 8(3), 146–151. https://doi.org/10.11648/j.ajset.20230803.12

Hussaini, A. A., Isma’il, M., & Dawaki, M. (2025). Designing an adaptable mobile GIS with conversational assistant for navigating dynamic campus environments. International Journal of Mobile Applications and Technologies, 1(2). https://doi.org/10.51137/wrp.ijmat.331

Kaindl, H., & Kainz, G. (1997). Bidirectional heuristic search reconsidered. Journal of Artificial Intelligence Research, 7, 283–317.

Li, J., & Yin, Z. (2024). CampNav: A system for inside buildings and campus navigation. Proceedings of the 2024 ASEE Annual Conference & Exposition, Portland, Oregon. https://doi.org/10.18260/1-2-47031

Madaminov, U. A., & Allaberganova, M. R. (2023). Firebase database usage and application technology in modern mobile applications. Proceedings of the 16th IEEE International Conference on Actual Problems of Electronic Instrument Engineering (APEIE). https://doi.org/10.1109/APEIE59731.2023.10347471

Manning, K., & Hosein, M. (2021). An Android-based navigation application for a tertiary campus. Proceedings of the 2021 IEEE International Conference on Mobile Networks and Wireless Communications (ICMNWC), Tumkur, India. IEEE.

Naik, S., Sharma, P., Jain, R., & Gupta, A. (2023). Smart campus navigation system integrating IoT and GPS for indoor and outdoor wayfinding. International Journal of Advanced Research in Engineering and Technology, 14(3), 55–64.

Parmanand, P., Sahdev, S., & Dwivedi, A. (2024). Study on the optimization of Dijkstra’s algorithm. Journal of Ravishankar University (Part-B: Science), 37(2), 255–267. https://doi.org/10.52228/jrub.2024-37-2-18

Pohl, I. (1971). Bi-directional search. In B. Meltzer & D. Michie (Eds.), Machine Intelligence 6 (pp. 127–140). Edinburgh University Press.

Prandi, C., Delnevo, G., Salomoni, P., & Mirri, S. (2021). On supporting university communities in indoor wayfinding: An inclusive design approach. Sensors, 21(9), 3134. https://doi.org/10.3390/s21093134

Ramachandrappa, N. C. (2024). A comparative analysis of native vs React Native mobile app development. International Journal of Computer Trends and Technology (IJCTT), 72(9), 57–62. https://doi.org/10.14445/22312803/IJCTT-V72I9P110

Raza, A. C., Al-Dubai, A. M., & Imran, M. A. (2020). A review of indoor navigation systems for campus environments. IEEE Access, 8, 24159–24179. https://doi.org/10.1109/ACCESS.2020.2970498

Ritawari, P., Chauhan, D., Tuteja, S., & Madan, K. (2024). Enhancing campus navigation: A conversational AI agent for location assistance. Proceedings of the 2024 Sixteenth International Conference on Contemporary Computing (IC3-2024).

Saraf, P. R., Jadhao, S. M., Wanjari, S. J., Kolwate, S. G., & Patil, A. D. (2022). A review on Firebase (backend as a service) for mobile application development. International Journal for Research in Applied Science and Engineering Technology (IJRASET), 10(1). https://doi.org/10.22214/ijraset.2022.39958

Sathvik, N. G., & Patil, S. (2021). Performance analysis of Dijkstra's and the A-Star algorithm on an obstacle map. In Data Engineering and Intelligent Computing, Advances in Intelligent Systems and Computing, vol. 1407 (pp. 71–80). Springer. https://doi.org/10.1007/978-981-16-0171-2_8

Semma, A., Hajar, Y., & Anwar, A. (2023). Cloud computing: Google Firebase Firestore optimization analysis. Indonesian Journal of Electrical Engineering and Computer Science, 29(3), 1719–1728. https://doi.org/10.11591/ijeecs.v29.i3.pp1719-1728

Sommerville, I. (2016). Software engineering (10th ed.). Pearson.

Tiwari, A., & Verma, P. (2021). Mobile campus navigation using GPS and map-based route selection. International Journal of Advanced Research in Computer Science, 12(3), 67–73.

Vilaspure, V., Sinha, M., Pawar, N., Jadhav, S., & Singh, P. (2024). PathPilot: Find your way — a campus navigation mobile application. International Journal of Research Publication and Reviews, 5(5), 6796–6799.

Wayahdi, M. R., Ginting, S. N. H., & Syahputra, D. (2021). Greedy, A-Star, and Dijkstra's algorithms in finding shortest path. International Journal of Advances in Data and Information Systems, 2(1), 45–52.

Zhou, H., Zhang, Q., Chen, Z., & Wang, Y. (2021). An ARCore-based augmented reality campus navigation system. Applied Sciences, 11(16), 7515. https://doi.org/10.3390/app11167515