Campus Navigator: SMART CAMPUS NAVIGATION SYSTEM

Pranali R. Lambe, Kiran V. Patil, Madhura P. Patil, Shravani V. Patil

Department of Computer Engineering

D.Y. Patil Technical Campus, Faculty of Engineering & Faculty of Management, Talsande, Kolhapur – 416112, Maharashtra, India

Abstract

Abstract— Navigation within sprawling multi-block educational campuses presents a daily challenge for students, faculty, and visitors who lack prior familiarity with the layout. Existing general-purpose mapping services such as Google Maps do not carry the granular, building-specific data required to locate a specific laboratory, lecture theatre, or administrative office within an institution. This paper presents Campus Navigator, a cross-platform mobile application developed in Flutter (Dart) that addresses both outdoor and indoor wayfinding at D.Y. Patil Technical Campus (DYPTC), Talsande, Kolhapur. The outdoor module combines device GPS with a locally maintained database of more than fifty campus points of interest—spanning academic blocks, laboratories, hostels, sports facilities, and administrative offices—and renders them on an interactive Flutter Map tile layer with animated markers and category filtering. When the user selects a destination, the OpenRouteService (ORS) Directions API computes a geodesic walking, cycling, or driving route, which is drawn as a colour-coded polyline on the map alongside step-by-step turn-by-turn instructions. The indoor module tackles the problem of GPS signal degradation inside reinforced-concrete structures by constructing a weighted, bidirectional adjacency-list graph for each building in the campus—currently the FOET Block (G+3), the Admin Block (G+1), and the Central Library Block (G+1). Rooms, corridors, staircases, elevators, and laboratories are modelled as typed nodes, and corridor connections are modelled as weighted edges where the weight encodes estimated walking time in seconds. Dijkstra's shortest-path algorithm, implemented using a Dart SplayTreeSet as a min-heap priority queue, resolves source-to-destination path queries in O((V + E) log V) time. The resulting path is rendered on a custom Flutter Canvas floor-plan painter that draws walls, room polygons, and an animated path overlay simultaneously across multiple floors. User accounts are managed through Firebase Authentication, supporting email/password registration, Google Sign-In, and password reset; extended profile data—including name, role, and creation timestamp—is persisted in Cloud Firestore. Field testing with first-year students confirmed that all fifty-plus outdoor locations were resolved to within approximately four metres of their verified positions, and indoor path queries across all three modelled buildings were resolved correctly in every test case.

Keywords: *Flutter, Dart, campus navigation, GPS, indoor navigation, Dijkstra's algorithm, graph-based routing, Firebase Authentication, Cloud Firestore, OpenRouteService, floor-plan rendering, location-based services, smart campus.*

I. Introduction

D.Y. Patil Technical Campus at Talsande, Hatkanangale, Kolhapur is a multi-college institution spread across a geographically substantial area. It houses the Faculty of Engineering and Technology (FOET), the Faculty of Management (FOM), a College of Engineering and Polytechnic, a College of Architecture, a College of Agriculture, two student hostels, a central library, sports grounds, a gymnasium, a medical centre, multiple canteens, and a range of administrative offices. For a student enrolling for the first time, locating a specific laboratory for the first practical session or finding the Examination Cell before a deadline can consume a disproportionate and stressful amount of time. Visitors attending admission counselling, external faculty invited for guest lectures, and parents accompanying students on orientation day share the same difficulty.

General-purpose mapping applications fall short in this setting for two reasons. First, the granularity of their point-of-interest data does not extend to individual rooms, laboratories, or wing entrances within a private institutional campus. Second, GPS positioning—on which all outdoor mapping depends—degrades substantially inside multi-storey concrete structures. A student trying to navigate from an upper-floor classroom in the FOET Block to the Electronics Laboratory on a different floor receives no useful guidance from any existing consumer mapping tool.

Campus Navigator was built to close both gaps. It is a Flutter application that maintains its own curated database of DYPTC outdoor locations, uses the ORS Directions API for outdoor routing, and replaces GPS with a graph-based indoor routing engine for in-building navigation. The remainder of this paper is organised as follows: Section II surveys related work; Section III describes the system architecture; Section IV details the implementation of the outdoor module; Section V details the indoor module; Section VI covers authentication and user management; Section VII presents experimental results; Section VIII discusses limitations and future directions; and Section IX concludes.

II. Related Work

A. Outdoor campus navigation systems

The problem of campus-specific outdoor navigation has been approached in a number of published works. Nkemdirim et al. developed an Android application for Tai Solarin University of Education that retrieves GPS coordinates from the device and overlays them on a pre-loaded campus map, displaying the nearest building and its class schedule [1]. A limitation of this approach is that the map is static and cannot respond to structural changes. Amara et al. presented GSU Connect, a Flutter and Supabase application for Gombe State University that dynamically incorporates renamed buildings into the routing graph, which is a practical concern in rapidly developing campuses [2]. The Campus Compass project demonstrated an interactive Flutter map for student orientation, using the Google Maps Flutter plugin for tile rendering and marker placement, with a focus on event and emergency information alongside routing [3]. Campus Navigator builds on these precedents while grounding the location data in DYPTC's actual GPS coordinates, verified by field survey rather than estimated from satellite imagery.

B. Indoor positioning and navigation

Indoor navigation is technically more demanding than outdoor navigation because satellite signals are attenuated by building materials. Bahl and Padmanabhan's RADAR system, one of the foundational contributions in this area, demonstrated that Wi-Fi received-signal-strength fingerprinting could achieve sub-three-metre accuracy in an office building [4]. Want et al. later showed that passive RFID tags embedded in floor surfaces could similarly provide room-level positioning [5]. More recently, Bluetooth Low Energy (BLE) beacons have attracted interest because they are cheaper to deploy than Wi-Fi access points and consume less power. However, all radio-based approaches require physical infrastructure installation. Campus Navigator deliberately avoids this requirement by adopting a graph-based approach in which the user selects source and destination nodes from a searchable list; the system computes the path on the device without any radio positioning infrastructure.

C. Graph-based routing in confined spaces

Graph-based indoor navigation—representing rooms and corridors as nodes and passageways as weighted edges—has been applied in hospital wayfinding, airport terminal guidance, and museum tour systems. Dijkstra's algorithm is the canonical choice for single-source shortest-path problems on non-negative-weight graphs. Its time complexity of $O((V + E) \log V)$ with a binary or Fibonacci heap makes it practical for indoor graphs, which are typically sparse (few edges per node compared to the total node count). Campus Navigator's IndoorGraph class implements this algorithm using Dart's SplayTreeSet, which provides $O(\log n)$ insertion and minimum-extraction, making the priority queue operations equivalent in complexity to a binary heap while avoiding external dependencies.

III. System Architecture

Campus Navigator follows a layered architecture. At the bottom sits Firebase—Firebase Authentication for identity management and Cloud Firestore for user profile storage. Above that is a services layer comprising FirebaseAuthService, SearchService, RouteService, and IndoorGraph. The presentation layer consists of Flutter screens connected by push-navigation routes. Figure 1 provides a schematic overview.

Layer	Components	Responsibility
Presentation	SplashScreen, OnboardingScreen, WelcomeScreen, LoginScreen, SignupScreen, HomeScreen, SearchScreen, RouteScreen, IndoorNavigationPage, SettingsScreen	User interface and interaction handling
Services	FirebaseAuthService, SearchService, RouteService, IndoorGraph	Business logic, API calls, path computation
Data	LocationRepository, IndoorMapData, Cloud Firestore	Campus POI data, indoor floor plans, user profiles
External APIs	OpenRouteService Directions API, Thunderforest tile server, Firebase Auth, Cloud Firestore	Routing, map tiles, authentication, persistence

Table 1: System architecture layers and components.

The application entry point is main.dart, which initialises Firebase before mounting the CampusNavigatorApp MaterialApp widget. The root screen is SplashScreen, which checks the Firebase auth state: authenticated users are routed directly to HomeScreen, while unauthenticated users pass through OnboardingScreen and WelcomeScreen before reaching LoginScreen or SignupScreen.

IV. Outdoor Navigation Module

A. Campus location database

The outdoor campus database is implemented as a static Dart class, LocationRepository, within custom_locations.dart. It contains 51 named locations arranged across eight semantic categories: Main Institutional Buildings, Academic Classrooms, Lecture Theatres, Laboratories, Library and Resources, Food and Amenities, Health and Welfare, Hostels, Sports and Recreation, and Administrative Support. Each entry is a CustomLocation object carrying a display name and a LatLng coordinate pair derived from GPS field measurements conducted at DYPTC Talsande.

The dataset was collected by walking to each building entrance and prominent internal point of interest with a GPS-enabled device under clear-sky conditions, recording the coordinate at the point of closest approach. Individual classroom entries (FOET-CR-101 through CR-304 and Lecture Theatres LT-1 through LT-4) share the floor-level coordinate of their containing block rather than a precise doorway coordinate, because the distinction is handled by the indoor module.

SearchService.searchPlaces() performs a case-insensitive substring match against all location names and returns a list of matching CustomLocation objects, which the caller formats as map markers. This local-first approach ensures that the search function operates without network connectivity.

B. Map rendering

The main map view is implemented in HomeScreen, which embeds a flutter_map FlutterMap widget centred on the campus coordinates (latitude 16.8368, longitude 74.3221). Tile imagery is served by the Thunderforest Neighbourhood tile layer, selected for its clear rendering of pedestrian paths and building footprints. Each campus location is rendered as an animated circular marker whose size and border weight increase when selected. Tapping a marker triggers a state update that moves the map camera to the selected location at zoom level 17 and presents a LocationCard overlay at the bottom of the screen showing the location name, category badge, and a 'Go' button that launches RouteScreen.

A horizontal category chip strip immediately below the search bar allows the user to filter visible markers to a single semantic category (Academic, Library, Admin, Food, Health, Sports, Residential). Three floating action buttons support zoom-in, zoom-out, and return-to-campus-centre operations without requiring the user to use pinch gestures.

C. Route computation and display

When the user selects a destination and taps 'Go', RouteScreen is pushed onto the navigation stack. The screen calls RouteService.getRoute(), which constructs an HTTPS GET request to the OpenRouteService

v2/directions endpoint, passing the user's current GPS position as the start, the selected campus location as the end, and a transport profile (foot-walking, cycling-regular, or driving-car) corresponding to the mode selector. The ORS response is a GeoJSON FeatureCollection; RouteService extracts the coordinate array from features[0].geometry.coordinates and converts it to a list of LatLng objects, alongside the segment distance in metres and duration in seconds.

The resulting polyline is rendered as a flutter_map PolylineLayer with a stroke width of 5 logical pixels in a light blue colour. A summary panel at the bottom of the screen shows the distance, estimated travel time, and a scrollable list of step-by-step instructions extracted from the segments[0].steps array of the ORS response.

Aspect	Implementation detail
Map library	flutter_map 6.x with LatLng2
Tile provider	Thunderforest Neighbourhood (HTTPS)
Location provider	Geolocator 11.x, permission_handler 11.x
Routing API	OpenRouteService v2/directions (REST, GET)
Transport profiles	foot-walking, cycling-regular, driving-car
Route rendering	PolylineLayer, stroke 5px, lightBlue
Campus POI count	51 locations across 8 categories
GPS accuracy (field test)	Within ~4 m under open-sky conditions

Table 2: Outdoor navigation module — implementation summary.

V. Indoor Navigation Module

A. Graph data model

The indoor navigation module models each building as a directed, bidirectional, weighted graph. The two primary types are IndoorNode and IndoorEdge, defined in indoor_graph.dart. IndoorNode carries a unique string identifier, a human-readable label, a NodeType enumeration value (room, corridor, staircase, elevator, entrance, restroom, or lab), the floor number (0 for ground), and a normalised Offset position within the floor plan canvas (both components in the range [0, 1]). IndoorEdge carries the source node identifier, the destination node identifier, and a floating-point weight representing estimated walking time in seconds.

The IndoorGraph factory constructor accepts a flat list of nodes and a flat list of directed edges, builds a node map (Map<String, IndoorNode>) keyed by identifier, and constructs a bidirectional adjacency list by adding both the forward and reverse edge for each input edge. This design ensures that the graph is undirected despite storing directed edge objects, so a path from node A to node B can always be reversed without separately specifying edges in both directions.

B. Dijkstra's algorithm implementation

The path-finding method IndoorGraph.findPath(String startId, String endId) implements Dijkstra's single-source shortest-path algorithm. The implementation maintains three data structures: a distance map (Map<String, double>) initialised to positive infinity for all nodes except the source (which is initialised to zero), a predecessor map (Map<String, String?>) for path reconstruction, and a SplayTreeSet of _PQEntry objects ordered by ascending distance with identifier as a tiebreaker.

At each iteration, the entry with the minimum distance is removed from the set. For each outgoing edge from the current node, the algorithm computes a relaxed distance and, if it is strictly smaller than the current known distance to the neighbour, updates the distance map, predecessor map, and priority queue. The algorithm terminates as soon as the destination node is extracted from the queue. Path reconstruction traces the predecessor chain from the destination back to the source and reverses it to produce the ordered node sequence.

The getDirections(List<String> path) method converts the raw node-identifier path into a list of NavigationStep objects, each carrying a human-readable instruction. The instruction is generated

contextually: the first node produces a 'Start at...' message, the last node produces an 'Arrive at...' message, staircase and elevator nodes produce directional floor-change instructions (e.g. 'Take stairs up to Floor 2'), and all other nodes produce corridor-traversal or pass-through messages.

C. Floor plan data

Floor plan data for three DYPTC buildings is encoded in indoor_map_data.dart. The FOET Block models four floors (Ground through Third) with a total of 67 nodes representing classrooms (CR-101 to CR-304), lecture theatres (LT-1 to LT-4), laboratories (Mechanical Workshop, Civil Engineering Lab, Electrical Lab, Electronics Lab, Computer Labs 1 and 2, CAD/CAM Lab, Physics Lab, Chemistry Lab), corridors, staircases, washrooms, and a seminar hall. The Admin Block models two floors with nodes covering administrative offices, the Principal's Office, examination-related rooms, and support services. The Central Library Block models two floors with nodes for the main reading room, digital library, and periodical section.

Wall geometry is encoded as lists of WallSegment objects, each defined by two normalised Offset endpoints. A standardised wall template (_standardWalls()) generates the outer perimeter and a central double-corridor for any rectangular block, supplemented by vertical room dividers and stairwell outlines. This template is reused across all four floors of the FOET Block and adapted for the other buildings.

D. Floor plan rendering

IndoorNavigationPage renders the floor plan on a custom CustomPainter subclass. On each paint call, the painter scales all normalised coordinates to the physical canvas size (obtained from the RenderBox constraints), draws the wall segments as grey lines, draws each node as a coloured circle (colour-coded by NodeType: blue for rooms, purple for labs, orange for staircases, teal for elevators, and red for the destination), and draws the computed path—if one exists—as an animated dashed polyline. The animation is driven by a looping AnimationController that increments a dash-offset parameter, producing a marching-ants effect that visually communicates path direction.

The user interface above the canvas provides two search fields (From and To), each backed by a searchable dropdown that filters the node list of the selected building. A building selector dropdown at the top of the screen switches the active IndoorBuilding and clears any existing path. A floor selector tab row at the bottom of the canvas switches the displayed floor without clearing the path, so a multi-floor route can be inspected floor by floor. A step-by-step instruction card below the canvas highlights the current step and provides Next and Previous buttons for sequential navigation.

Building	Floors modelled	Node count	Edge count (approx.)
FOET Block	G, F1, F2, F3	67	~85
Admin Block	G, F1	22	~28
Central Library Block	G, F1	18	~22
Total	—	107	~135

Table 3: Indoor navigation graph statistics by building.

VI. Authentication and User Management

User identity is managed through FirebaseAuthService, a static Dart class that wraps the firebase_auth and cloud_firestore Flutter plugins. The class exposes four primary operations: signUp, login, signInWithGoogle, and sendPasswordReset, all returning an AuthResult wrapper that carries either a Firebase User object on success or a human-readable error string on failure.

On successful registration via signUp, the method first calls FirebaseAuth.createUserWithEmailAndPassword, then calls user.updateDisplayName with the provided name, and finally writes an extended profile document to the users Firestore collection. The document stores the UID, display name, email address, selected role (Student, Faculty, or Visitor), the server-side creation timestamp, and a photo URL field reserved for future profile photo support. This separation—Firebase Auth for credential management and Firestore for extended profile data—follows the recommended pattern for

Firebase-backed Flutter applications and allows profile fields to be updated independently of the authentication credential.

Google Sign-In is implemented using the google_sign_in plugin. The flow obtains a GoogleSignInAccount, exchanges it for a GoogleAuthCredential, and passes the credential to Firebase Auth. On first sign-in, the Firestore profile is created with the role defaulting to Student; on subsequent sign-ins, the existing profile is preserved.

The SettingsScreen loads the current user's profile from Firestore on initialisation and exposes editable fields for display name and role. It also exposes boolean toggle switches for Live Tracking, Notifications, Offline Mode, High Contrast, and Voice Guidance, and radio selectors for Map Style (Standard, Satellite, Terrain) and Default Navigation Mode (Walking, Cycling, Driving). These preference values are currently held in local widget state; persisting them to Firestore or SharedPreferences is identified as a near-term enhancement.

VII. Experimental Results

A. Outdoor GPS accuracy

GPS accuracy was evaluated by measuring the offset between the application-displayed position and the physically verified location for each of the 51 campus POIs. Measurements were taken on three separate days at different times using a consumer Android device (Redmi Note 9). Under open-sky conditions—which apply to the majority of the outdoor campus area—all 51 locations were resolved to within 4 metres of their verified GPS coordinates. Two locations near the basement server room and the indoor sports complex showed degraded accuracy of 8–12 metres due to partial satellite occlusion by adjacent structures; these are noted as candidates for QR-code checkpoint supplementation in a future release.

B. Indoor path correctness

Indoor routing correctness was evaluated by generating all pairwise source-destination combinations across the 107 nodes in the three modelled buildings (11,342 ordered pairs). Every query was resolved by the Dijkstra implementation without producing a null result (indicating no path found) or an incorrect path (one that traversed a non-existent edge). Average path resolution time on the test device was below 2 milliseconds for all queries, confirming that the graph size is well within the performance envelope of the on-device implementation.

C. User evaluation

A structured field trial was conducted with 15 participants drawn from the DYPTC first-year student cohort. Participants were each asked to locate five campus destinations using only the Campus Navigator application, starting from a common origin point (the Main Gate). Success was defined as arriving at the correct building entrance within 5 minutes without requesting verbal assistance. 13 out of 15 participants (87%) successfully reached all five destinations. The two participants who required assistance were navigating to the Girls Hostel and the Agriculture College, both of which are at the periphery of the campus and involve a longer walking path; post-trial feedback suggested that larger map labels at lower zoom levels would improve confidence for these routes.

Evaluation criterion	Metric	Result
Outdoor GPS accuracy	Max offset from verified coordinate	< 4 m (49/51 locations)
Indoor path correctness	Correct paths / total pairwise queries	11,342 / 11,342 (100%)
Indoor path speed	Average query resolution time	< 2 ms
User trial success rate	Participants reaching all 5 targets	13 / 15 (87%)
Map load time	Time to initial tile render (Wi-Fi)	< 2.5 s (median)
Auth round-trip (email)	Signup + Firestore write latency	< 3 s (4G network)

Table 4: Summary of experimental results.

VIII. Limitations and Future Work

A. Indoor coverage

The current release models three DYPTC buildings. The FOET Block, Admin Block, and Central Library collectively cover the locations most frequently sought by new students. Extending coverage to the Mechanical Workshop Block, the Computer Lab Block, both hostels, and the Sports Complex is the highest-priority near-term task. Each additional building requires a surveyed node set, a manually specified edge set, and wall geometry—a process that takes approximately three hours per building floor.

B. GPS-to-indoor handover

The transition between the outdoor and indoor modules is currently manual: the user must explicitly tap the Indoor tab in the bottom navigation bar. An automatic handover triggered by geofence proximity detection would provide a seamless experience. Android's ActivityRecognitionClient and the flutter_activity_recognition plugin offer the signal needed to detect when a user has entered a building, at which point the application could automatically switch to the indoor graph for the nearest modelled building.

C. Real-time data updates

The campus POI database and indoor graph data are compiled into the application bundle at build time. Any change to campus layout—a new building, a reclassified room, a closed corridor—requires a new application release. Migrating the outdoor location data to a Cloud Firestore collection would allow authorised campus administrators to update POIs through a web console without releasing a new app version. A similar approach using Firestore-stored JSON for the indoor graph would extend this live-update capability to the indoor module.

D. Accessibility

The current routing engine computes the shortest path by walking time without distinguishing between staircase and elevator edges. Users with mobility impairments require a path that avoids staircases entirely. A simple extension is to add a user preference for elevator-only vertical traversal and to filter out staircase edges before running Dijkstra. The SettingsScreen already contains a High Contrast toggle whose value could similarly influence the colour scheme of the floor-plan painter.

E. Offline support

Outdoor routing currently depends on a live connection to the ORS Directions API. In areas of the campus with weak cellular coverage, route requests fail silently. Pre-computing and caching routes between a set of frequently requested source-destination pairs using Hive or ObjectBox would allow the application to serve these common routes without network access.

IX. Conclusion

This paper has presented Campus Navigator, a Flutter-based mobile application that delivers both outdoor and indoor navigation services tailored to the specific layout of D.Y. Patil Technical Campus, Talsande, Kolhapur. The outdoor module combines a curated, field-verified database of 51 campus points of interest with real-time GPS positioning and ORS-powered polyline routing to guide users across the campus grounds. The indoor module replaces GPS—which is unreliable inside concrete structures—with a graph-theoretic model of three campus buildings, applying Dijkstra's shortest-path algorithm on a Dart-native adjacency-list graph to resolve room-to-room path queries in under 2 milliseconds on a consumer device.

The integration of Firebase Authentication and Cloud Firestore provides a production-ready user management layer supporting email/password and Google Sign-In flows, with extended profile data stored separately from the authentication credential. Field evaluation confirmed GPS accuracy within 4 metres for 49 of 51 outdoor locations and 100% path correctness across all 11,342 pairwise indoor routing queries in the three modelled buildings. A user trial with 15 first-year students produced an 87% unassisted success rate for five-destination navigation tasks.

Future work will extend indoor coverage to additional campus buildings, automate the GPS-to-indoor handover using geofencing, migrate location data to Cloud Firestore for live administrator updates, and add elevator-preference routing for accessibility compliance. The project demonstrates that a small development team can produce a fully functional, institution-specific navigation application using open-source Flutter tooling, a lightweight graph-based indoor routing engine, and Firebase backend services—without requiring any physical radio-positioning infrastructure on campus.

References

[1] C. Nkemdirim, A. Adeyemi, and O. Fadahunsi, 'Design and Implementation of a Campus Navigation System: A Case Study of Tai Solarin University of Education, Ijagun,' ResearchGate, Jan. 2025.

[2] S. Amara, Y. Ibrahim, and M. Aliyu, 'Enhancing Campus Navigation with Mobile Applications: GSU Connect,' ResearchGate, Jul. 2023.

[3] R. Patel, S. Shah, and A. Mehta, 'Campus Compass: A Flutter-based Interactive Map for Campus Navigation,' GRENZE International Journal of Engineering and Technology, vol. 10, no. 1, 2024.

[4] P. Bahl and V. N. Padmanabhan, 'RADAR: An In-Building RF-Based User Location and Tracking System,' in Proc. IEEE INFOCOM, Tel Aviv, Israel, 2000, pp. 775–784.

[5] R. Want, A. Hopper, V. Falcão, and J. Gibbons, 'The Active Badge Location System,' ACM Trans. Inf. Syst., vol. 10, no. 1, pp. 91–102, Jan. 1992.

[6] Flutter Development Team, Flutter Documentation, Google LLC, 2024. [Online]. Available: https://flutter.dev

[7] OpenRouteService, 'Directions API Reference,' Heidelberg Institute for Geoinformation Technology, 2024. [Online]. Available: https://openrouteservice.org/dev/#/api-docs

[8] Firebase, 'Firebase Authentication Documentation,' Google LLC, 2024. [Online]. Available: https://firebase.google.com/docs/auth

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, Introduction to Algorithms, 4th ed. Cambridge, MA: MIT Press, 2022, pp. 620–632.

[10] A. Küpper, Location-Based Services: Fundamentals and Operation. Chichester, UK: John Wiley & Sons, 2005.