UNIVERSITY OF LEICESTER

School of Computing and Mathematical Sciences

CO7201 Individual Project

Final Report

A Mobile App for Campus Navigation

Prerith Dsouza
pd245@student.le.ac.uk
249006686

Project Supervisor: Dr Kehinde Aruleba
Principal Marker: Anthony Conway

Word Count: 11934
Submission Date – 05/09/2025

DECLARATION

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s). Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by clear citation of the source, specifying author, work, date and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Prerith Dsouza
Date: 05/09/2025

Acknowledgement

I would like to express my deepest gratitude to everyone who has contributed to the successful development and progress of this project, "A Mobile App for Campus Navigation". The journey from conceptualisation to implementation has been both intellectually challenging and personally rewarding, and it would not have been possible without the invaluable support, guidance, and encouragement I have received from numerous individuals and organisations.

First and foremost, I am profoundly thankful to my project supervisor, Dr Kehinde Aruleba, and my Principal Marker, Anthony Conway whose expertise, insightful feedback has guided me at every stage of this work. Their commitment to ensuring high academic and technical standards inspired me to explore innovative solutions and persist in the face of challenges. The time and effort invested in reviewing my work and providing clear direction have been instrumental in shaping the quality of this report and the overall success of the project.

I also extend my sincere thanks to the faculty and staff of the School of Computing and Mathematical Sciences at the University of Leicester for creating an intellectually stimulating environment. The technical facilities, academic resources, and open exchange of ideas have significantly enriched my learning experience and contributed to the refinement of my research and development skills.

My appreciation extends to my peers and colleagues, who have been a constant source of encouragement, collaboration, and valuable feedback. Sharing ideas, brainstorming possible solutions, and receiving honest evaluations of my work have provided me with fresh perspectives and fostered continuous improvement throughout the project.

I am deeply grateful to the University of Leicester's administrative and technical support teams, whose efforts in maintaining campus infrastructure and IT resources ensured I could work efficiently and without interruption. Special thanks are due to the library staff for their assistance in sourcing relevant academic literature, which was essential in grounding the project within established research in Human-Computer Interaction (HCI), mobile app development, and campus accessibility.

Finally, I owe heartfelt thanks to my family and friends, who have supported me with patience, understanding, and encouragement. Their unwavering belief in my abilities has been my source of motivation and perseverance.

To everyone who has played a role in making this project a reality, whether directly or indirectly, I extend my sincere gratitude. Your contributions have not only shaped this work but have also enriched my academic and professional growth.

Abstract

University campuses are dynamic and often complex environments, comprising multiple buildings, departments, service facilities, and outdoor areas. While such diversity fosters academic, social, and cultural engagement, it can also present navigational challenges, particularly for new students, visitors, and individuals unfamiliar with the layout. Traditional solutions such as static campus maps and printed guides lack real-time adaptability, while generic digital navigation tools often fail to provide the level of detail and contextual information required within a campus environment.

This project presents the design, development, and evaluation of a mobile application tailored for campus navigation at the University of Leicester. The proposed solution integrates real-time location tracking, interactive mapping, building-level navigation, timetable integration, and additional features such as notifications for events or disruptions. The aim is to provide a user-friendly, context-aware platform that enhances accessibility, efficiency, and the overall user experience for students, staff, and visitors.

The development of the application followed a Human-Computer Interaction (HCI) design framework, ensuring that usability, accessibility, and user engagement were central to each design iteration. Initial research was conducted through a combination of literature review and analysis of existing navigation tools, identifying specific gaps in functionality and accessibility for campus-specific contexts. Insights gained from this research informed the app's feature set, interface design, and technological architecture.

Core features include GPS-based outdoor navigation, searchable campus points of interest, and integration with university systems such as class timetables and event notifications, as well as real-time bus information via the UK Bus Open Data API. In addition, accessibility features such as voice-assisted search and audio navigation cues were incorporated to ensure inclusivity for visually impaired users

Contents

1. Introduction

Navigating a modern university campus can be a daunting task, particularly for new students, visitors, and even staff members who may be unfamiliar with certain areas. University campuses are increasingly complex environments, often spanning multiple acres and incorporating a mix of academic buildings, administrative offices, libraries, laboratories, sports facilities, and student service centres. For first-year students, the initial weeks are often characterised by uncertainty and disorientation, which can result in late arrivals to lectures, missed meetings, and increased stress levels. For visitors attending open days, conferences, or events, these navigational challenges can significantly detract from their overall experience [1], [3].

While traditional methods such as static campus maps, printed brochures, and directional signboards have been the primary means of wayfinding, they present several limitations. Static maps are prone to becoming outdated due to campus expansion or building relocations. They also lack real-time adaptability, meaning they cannot account for temporary closures, event-specific redirections, or sudden changes in accessibility routes. Signboards, while useful for immediate guidance, may not be placed in sufficient density to guide users effectively through large or intricate areas. Moreover, for visually impaired individuals, both printed maps and signboards may not provide the accessibility required for independent navigation [1], [2].

With the widespread adoption of smartphones and the growing capabilities of mobile applications, there exists an opportunity to develop context-aware, location-based solutions tailored to campus navigation. General-purpose navigation platforms such as Google Maps and Apple Maps, while highly effective for city-wide travel, often lack the precision, detail, and building-specific data needed for university environments. They typically do not include specific departmental offices, or context-specific features such as lecture hall scheduling, event notifications, or facility availability. As such, there is a clear gap for a dedicated, university-specific navigation tool that combines the precision of location-based services with the contextual relevance of campus-specific data [4], [5], [7], [8], [12].

This project addresses this gap by developing a dedicated mobile application for campus navigation at the University of Leicester. The app is designed not only to provide point-to-point navigation but also to integrate features that enhance the overall campus experience. By combining real-time GPS tracking, interactive maps, timetable integration, real-time bus information via the UK Bus Open Data API and accessibility-focused features such as voice-assisted search, the application aims to provide an inclusive, user-friendly solution that meets the diverse needs of the campus community [2], [6], [9].

The motivation for this project stems from both academic and practical considerations. From an academic perspective, the project offers the opportunity to explore the intersection of Human-Computer Interaction (HCI), mobile application development, and context-aware computing. The application serves as a case study in how user-centred design principles can be applied to create solutions that address specific real-world challenges. Practically, the project has the potential to deliver tangible benefits to the university community, including improved orientation for new students, enhanced accessibility for individuals with disabilities, and increased engagement through integration with campus events and services [6], [13], [9].

The development process is guided by an iterative design methodology, ensuring that feedback from real users is incorporated at multiple stages. Initial requirements were gathered through literature review, analysis of existing navigation solutions, and informal discussions with students and staff. The app's feature set was refined through low-fidelity prototyping, followed by high-fidelity designs and implementation using cross-platform development tools to ensure compatibility, aligned with human-centred design standards [13].

A key aspect of the project is its emphasis on inclusivity. Recognising the diverse needs of the campus population, the application incorporates accessibility features such as voice input for search queries, audio-based navigation instructions, and compatibility with screen readers. These features are intended to ensure that the app is usable by individuals with varying abilities and preferences, thereby contributing to a more equitable campus environment [9], [11].

In summary, the project seeks to bridge the gap between traditional campus navigation methods and the potential of modern mobile technologies. By providing a solution that is accurate, accessible, and tailored to the unique context of the University of Leicester, this work aims to enhance the daily experiences of its users while contributing valuable insights to the broader field of context-aware mobile application development [2], [6], [12].

2. Aims and Objectives

2.1 Aim

The overarching aim of this project is to design, develop, and evaluate a mobile application that provides a dedicated navigation solution for the University of Leicester campus. The application is intended to address the limitations of traditional navigation methods by leveraging real-time location tracking, interactive campus maps, and accessibility-focused features to deliver a more accurate, user-friendly, and inclusive wayfinding experience **[6], [9], [12], [17].**

The app is envisioned not merely as a static navigation tool, but as a dynamic platform capable of integrating additional services such as timetable access, event notifications, real-time bus information and accessibility routes. In doing so, it will cater to a wide range of stakeholders, including students, staff, and visitors, while setting the foundation for future scalability and cross-campus applicability **[18], [20], [14]**.

2.2 Objectives

The aims of the project are translated into the following key objectives, each of which is designed to ensure that the application is functional, accessible, and contextually relevant to the unique needs of the University of Leicester community.

Objective 1: Conduct comprehensive background research

A robust foundation of academic and technical knowledge is essential for the successful design and implementation of the application. This objective involves reviewing relevant literature in areas such as Human-Computer Interaction (HCI), location-based services, mobile app usability, and accessibility design principles, alongside an analysis of existing campus navigation tools to identify strengths and shortcomings that inform design [13], [6], [9], [12], [15].

Objective 2: Gather and analyse user requirements

The development process will be guided by user-centred design principles, making it essential to understand the specific needs, preferences, and challenges faced by the app's intended audience. Methods include informal interviews and observation of navigation behaviours within the campus, ensuring that functionality, interface design, and accessibility features reflect real user needs [13].

Objective 3: Design an intuitive and accessible user interface

The success of the application depends heavily on its usability. This objective focuses on creating a UI that is visually clear, logically structured, and responsive to varying device screen sizes, following established usability and accessibility standards. The design will ensure compatibility with assistive technologies such as screen readers and incorporate features like high-contrast themes, scalable text, and voice-based input; low- and high-fidelity prototypes will be iteratively tested and refined [15], [9], [16], [10], [11].

Objective 4: Implement accurate and real-time navigation functionality

The core function of the application is to provide precise point-to-point navigation within the campus environment. This will be achieved by integrating GPS for outdoor navigation. Real-time tracking will be complemented by dynamic map updates, enabling users to visualise their current position and route progress. Additional functionality will include route recalculation in case of deviations, and the provision of shortest or most accessible routes based on user preferences [17], [12].

Objective 5: Integrate campus-specific data and features

Unlike general mapping applications, this project requires the inclusion of detailed, campus-specific data. This will include locations of academic departments, lecture halls, laboratories, student service centres, sports facilities, libraries, and cafes. Timetable integration will allow students to view their upcoming lectures and receive navigation guidance to the relevant rooms. Event integration will ensure that users can locate and attend campus activities such as open days, guest lectures, or club meetings [18].

Objective 6: Implement accessibility-focused features

Accessibility is a central priority of the project, ensuring that the app caters to the needs of users with disabilities or special requirements. Voice search will be implemented to allow

hands-free operation, while spoken navigation instructions will assist visually impaired users. Accessibility routes, such as step free paths and ramps, will be included where possible [9]–[11].

Objective 7: Ensure system scalability and maintainability

To future-proof the application, the system architecture will be designed for scalability, enabling new features or datasets to be added without significant redevelopment. The codebase will follow modular programming practices to facilitate maintainability, making it easier for future developers or the university's IT services to manage updates and enhancements [14].

Objective 8: Test, evaluate, and refine the application

Rigorous testing will be carried out to ensure that the application meets its functional, usability, and accessibility goals. Testing will include unit testing for individual components, integration testing to ensure smooth interaction between modules, and system testing to verify overall functionality. Usability testing with real users will provide feedback on interface clarity, navigation accuracy, and feature usefulness. Iterative refinement based on test results will ensure a polished final product [19].

2.3 Scope of the Project

The scope of the project will be limited to the University of Leicester campus during its initial implementation phase. While the primary audience will be students and staff, the design will also account for the needs of short-term visitors. The focus will be on outdoor navigation, with future potential for indoor navigation enhancements. The system will be developed to run on Android, and the backend architecture will be designed to allow data updates without requiring complete redeployment of the app [12].

3. Background and Literature Review

3.1 Scope and Review Method

This review examines academic and practitioner sources across campus navigation, HCI, mobile UX, accessibility, and location-based services (LBS). It extends the structured approach outlined in the preliminary report clarifying purpose, search terms, and inclusion criteria while broadening coverage beyond initial seeds. Specifically, we target positioning technologies suitable for campus contexts, usability and accessibility principles for mobile wayfinding, comparative analysis of existing solutions (general-purpose vs campus-specific), and technological gaps that inform this project's design decisions. The search strategy follows the earlier protocol and keyword set (e.g., campus navigation, outdoor positioning, accessibility in mobile apps) but applies stricter relevance and quality filters to foreground methods with empirical evaluation or formal specification.

3.2 Academic research landscape: wayfinding, HCI, and accessibility

Campus wayfinding spans cognitive, spatial, and interaction challenges. Foundational HCI and context-aware computing emphasise designing for situation, task, and user state a lens that motivates lightweight personalisation (e.g., saved routes, step free preference) and clear feedback during navigation [6]. Human-centred design standards (ISO 9241-210) frame iterative inquiry in understanding context, defining requirements, prototyping and evaluation aligning with this project's process and earlier plan [13]. Heuristic guidance (e.g., Nielsen's usability principles) remains practical for early UI reviews where experimental resources are limited [15]. For summative perception, the System Usability Scale (SUS) is a widely adopted, low-overhead instrument for benchmarking mobile interfaces across domains [23].

Accessibility is central to campus inclusivity. WCAG 2.2 offers actionable criteria on perceivability, operability, and robustness for UI components such as contrast, focus order, alternatives that can be adapted to native apps [9]. Platform guidelines such as Talkback and Voiceover shape interaction expectations, reinforcing the value of voice input and audio cues for users with visual impairments [10], [11]. Collectively, these sources converge on progressive disclosure**,** minimal visual clutter**,** predictable feedback**,** and multiple input modalities, which informed the single-marker map focus and step free preference in this project.

3.3 Location technologies for campus contexts

Outdoors, commodity smartphones provide adequate accuracy for building-level navigation but degrade in urban-canyon conditions. Empirical studies report median errors typically within several metres, with variability driven by environment and reception [27]. Indoors, GPS is unreliable, surveys and theses consistently map the trade space among Wi-Fi fingerprinting, BLE beacons, UWB, and vision-based methods in terms of accuracy, cost, maintenance, and calibration [12], [21]. Fingerprinting with BLE beacons is a recurrent, pragmatic choice for building-scale deployments; controlled evaluations in ~600 m² testbeds demonstrate viable performance but at the cost of site surveys and ongoing infrastructure care [22]. Emerging hybrid schemes fuse GNSS, inertial sensors, and beacons to stabilise headings and improve pedestrian accuracy, but remain non-trivial to tune and maintain in production environments [12], [22].

Data representation and map availability also matter. For Apple's ecosystem, the Indoor Mapping Data Format (IMDF) standardises venue data so organisations can publish validated, multi-level indoor maps through MapKit/MapKit JS [24], [28]. Google exposes Indoor Maps for selected venues via partner programmes, but coverage and update control vary by region and agreement [7], [8]. In the open-data space**,** Open Street Map (OSM) offers community conventions Simple Indoor Tagging for floors/spaces and wheelchair and accessibility tags that can feed accessible routing, though coverage is patchy and requires local contributions [25], [26]. These standards and programmes collectively show that indoor capability is available but not turnkey as it demands floor-plan assets, a data-publishing pipeline, and sustained curation [24], [28], [7], [25].

For an MVP focused on the University of Leicester, emphasising outdoor navigation with campus-specific POIs is evidence based and proportionate. Indoor features can be staged once the institution secures floor-plan rights and chooses a publishing path (IMDF, OSM indoor schema, or a beacon-based pilot).

3.4 General-purpose vs campus-specific navigation

General-purpose platforms like Google maps and Apple maps excel at city-scale travel with robust routing, global coverage, and familiar UI metaphors. However, three limitations recur in campus contexts. First**,** semantic granularity. Department suites, seminar rooms, step free entrances, and temporary closures are often missing or slow to update [7], [8], [28]. Second, venue data control. Institutional teams may lack direct write access or validation workflows to correct details at the pace required during induction weeks or building works [7], [28]. Third, accessibility routing. While platforms provide limited indoor overlays, end-to-end accessible routing across mixed indoor-outdoor paths depends on local data like entrances, lifts, that are unevenly mapped. Community schemas like OSM wheelchair tags exist but require deliberate curation to be reliable [26], [25].

Campus-specific systems in the literature frequently bridge these gaps by integrating institutional datasets like timetables, venue metadata and local constraints like access routes and closures with positioning methods tuned to the estate [2], [12], [22]. Studies also underline operational costs. Beacon grids need planning, battery replacement, and interference mitigation. Wi-Fi fingerprints require periodic resurvey and AR-based approaches can be sensitive to lighting or texture [2], [12], [22]. Consequently, many projects adopt an incremental posture. Start outdoors with high-value POIs and clear entrance selection, add indoor pilots where data and governance exist, iterate on accessibility data quality over time [12], [24], [25].

3.5 Technological gaps and opportunities for innovation

(1) Last-metre decision support**:** Even with outdoor GPS, users struggle at the building threshold. “Which entrance is step free? Which door gets me closest to Room X?” Standard maps seldom expose this logic.

Opportunity: single-focus UI (one searched marker + distance/time chip) coupled with entrance-aware POIs and a Step free preference to reduce cognitive load and error consistent with HCI guidance on progressive disclosure and reversible actions [6], [9], [13].

(2) Accessibility-first data: Open schemas (OSM wheelchair/access tags, OSM indoor tagging) exist but coverage is inconsistent.

Opportunity: partner with Estates/Accessibility teams to curate authoritative entrance and vertical-transport metadata, then publish via OSM indoor schemas. This improves both on-device routing and the public map commons [24], [26].

(3) Low-friction communications: Time-sensitive campus updates like room changes, works, closures risk being missed in push-only flows.

Opportunity: pair topic messaging with a per-user inbox so important notices persist until acknowledged. An HCI-aligned pattern for reducing missed information without spamming [20], [13].

(4) Privacy by design positioning: Continuous trace storage raises privacy burdens with limited benefit for outdoor campus tasks.

Opportunity: data minimisation (store saved routes, preferences, and coarse stats), with transparent permission rationale. Consistent with WCAG guidance on user control and ISO 9241-210's ethical focus [9], [13].

(5) Measurable usability**:** Many campus systems lack consistent, comparable evaluation.

Opportunity: combine task metrics with SUS for a comparable score over iterations. This is practical for small cohorts and aligns with the evaluation plan [23].

3.6 Summary and implications for the project

The literature and platform ecosystem suggest a phased strategy, prioritise outdoor building level navigation and a single focus interaction model. Expose step free preference early, integrate campus specific POIs and a durable in app inbox and plan indoor expansion once floor plan rights, data pipelines (IMDF/OSM), and maintenance responsibilities are in place. This stance matches the priorities and constraints documented in the preliminary report and builds a realistic foundation for later innovation.

4. System Design and Architecture

4.1 Architectural overview

The solution adopts a mobile-first, thin-client with managed backend approach. The Android client (Flutter) handles presentation, lightweight domain logic, device capabilities and foreground notification display. Server responsibilities are delegated to Firebase services. Authentication (identity, tokens), Cloud Firestore (document database), Cloud Functions (admin workflows such as broadcast fan-out), and Firebase Cloud Messaging (FCM) (push delivery). Mapping and routing rely on Google Maps for Flutter (native SDKs via a federated plugin) and on demand Directions API calls for walking distances. A small OpenWeatherMap (OWM) call powers the home screen weather tile to give context (e.g., rain may influence route choice and timing). This architecture emphasises rapid iteration, minimal DevOps overhead, and keeping security policy close to the data model [30],[33], [35], [41], [43], [44].

Positioning uses the Fused Location Provider. The app reads the last known location for fast initial fixes and requests updates only when necessary for route preview or recentring. This approach is well suited to building level outdoor navigation on campus while being mindful of power use and permissions [34].

4.2 Frontend (Flutter) design

Screen flow: To reduce cognitive load during wayfinding, the app keeps a compact set of screens with clear roles:
• Home: search (text + mic), favourites (Saved Routes), timetable snapshot, weather tile, notifications bell.
• Map: a single red **"**searched**"** marker, a distance/time chip, and a clear button to Clear Marker.

• Saved Routes**:** quick recall of labelled places/paths.
• Notifications Inbox**:** durable history with unread badge.
• Profile: user preferences (Step free, Dark Mode, Notifications) and quick links (Help, Contact).

Maps and routing**:** The embedded map is provided by Google Maps for Flutter, which bridges to the native SDKs, preserving performance and platform features. When a user selects a destination, the app requests a route preview via Directions API to compute distance and ETA (walking). The single marker interaction keeps the map visually sparse and consistent with HCI guidance on progressive disclosure [31], [33].

Location and permissions: Foreground location permission is requested just in time. On grant, the client first reads the last known fix (low latency), then optionally subscribes to short lived updates while previewing a route. If permission is denied, related UI components degrade gracefully (e.g., disabling the distance/time chip) [34].

Notifications and inbox**:** While the app is in the foreground, arriving FCM messages are surfaced using local notifications to ensure visibility without relying on OS banners. Important messages are mirrored into the Inbox collection for persistence and an unread badge. This pairing of transient push with a durable in app list reduces the chance of missed alerts due to timing or OS state [41], [42].

Accessibility and theming**:** The UI follows Material 3 guidance, large touch targets, strong colour contrast, predictable focus order, and scalable text. A persistent step free preference is surfaced near routing controls to reduce last metre decision friction for wheelchair users or anyone preferring step free entrances [44].

4.3 Backend (Firebase) design

Authentication**:** Users authenticate with Firebase Auth; the resulting identity token authorises client reads/writes governed by Firestore Security Rules. Keeping auth serverless simplifies setup and maintenance for a small team while retaining robust token-based access control [35].

Data storage (Cloud Firestore): Firestore stores small, task specific documents grouped into collections and subcollections (e.g., users/{uid}/savedRoutes). This structure scales to thousands of users and aligns with real time UI patterns (streams/snapshots). The model also supports per user inbox items for durable notifications and a global map_pois catalogue for campus entities [36].

Security rules: Security is implemented with the principle of least privilege. A user may read and write only their own documents and subcollections, destructive deletes on statistics are disallowed, and the admin only fan out function is restricted using a verified admin e-mail claim. Rules are evaluated per request on the server, isolating users by UID and preventing cross tenant data exposure [37].

Indexes: Firestore's indexing supports the app's common queries efficiently. Composite indexes back the Inbox's unread filter and saved routes ordering, while single field indexes cover equality filters on simple fields (e.g., read == false) [38].

Cloud Functions (admin workflow): A single callable function implements the broadcast workflow. Validate the caller's admin claim, send an FCM topic message, fan out a copy of the message as a document in each recipient's per user Inbox. Batched writes respect platform limits and make delivery auditable in app even if a push banner was missed [39], [41].

4.4 Data structures and schema (Firestore)

Figure 2 summarises the core structures:
• users/{uid} (doc): displayName, email, and prefs (e.g., stepFree, darkMode, notifications).
• users/{uid}/savedRoutes/{routeId}: label, placeName, coordinates (lat, lng), order, createdAt.
• users/{uid}/stats/main: tripsCompleted:int, distanceKm:double, updatedAt.
• userNotifications/{uid}/items/{notifId}: title, body, read:bool, data:map, createdAt.
• map_pois/{id}: name, category, lat, lng, stepFree:bool, tags:[string].
• admin_notifications/{doc}: templates/metadata for broadcasts.

This design keeps documents small, supports low latency snapshot listeners, and separates per user and global data cleanly. Indexes are created for unread filtering and route ordering [36], [38].

Timetable data (admin-authored): We model timetables as bundles with nested sessions so admin can prepare and publish in batches while users see only live data.

- timetable_bundles/{bundleId} (collection)
  Fields:
  - title: string (e.g., "Autumn Week 1 – School of Computing")
  - term: string (e.g., "Autumn 2025")
  - startDate: timestamp • endDate: timestamp
  - isPublished: bool (controls user visibility)
  - notes: string (optional)
  - createdAt: timestamp • updatedAt: timestamp
- timetable_bundles/{bundleId}/sessions/{sessionId} (subcollection)
  Fields:
  - moduleCode: string (e.g., "CO3095")
  - title: string (e.g., "Dissertation")
  - buildingName: string • room: string
  - lat: double • lng: double (for "Navigate" deep link)
  - startTime: timestamp • endTime: timestamp
  - lecturer: string (optional)
  - isCancelled: bool (optional, default false)

Admin notifications (compose and broadcast):

- admin_notifications/{doc}: stores last composed message and defaults. Cloud Function uses this plus the request payload to publish.
- userNotifications/{uid}/items/{notifId}: durable per user inbox items written by the Function (read/unread).

POIs (curation from admin UI):

- map_pois/{id}: Confirm fields used by the editor,
  name, category, lat, lng, stepFree:bool, tags:[string], active:bool, updatedAt.

Indexes: ensure composite indexes for userNotifications/*/items on read (filter) + createdAt (order), and savedRoutes ordered by order.

4.5 Key runtime flows

(1) Sign-in and rule guarded access: The user signs in via Auth, the client receives a token and attempts reads/writes only on allowed paths. Any cross-user access (e.g., reading someone else's routes) is denied at the rule layer [35], [37].

(2) Search and route preview: The user searches for a building, the app places a single "searched" marker and requests a Directions API route preview (walking). The distance/time chip updates immediately. The user may Save Route, which writes a document under users/{uid}/savedRoutes and updates the home screen favourites stream [31], [33], [36].

(3) Broadcast and Inbox: An administrator calls the callable function with title/body/data; the function sends a topic push via FCM and writes an Inbox item to each user's userNotifications/{uid}/items. Clients show an unread badge, if the app is in the foreground, a local notification is also raised for visibility [39], [42].

4.6 Cross-cutting qualities (NFRs) and architectural tactics

Performance: The app minimises cold start work and defers network calls until needed. Using the last known location yields quick initial positioning. Route previews are requested only after the user places or confirms a destination. Firestore queries are narrow and indexed (unread filter, ordered saved routes), avoiding fan out reads. Caching OWM responses reduces latency and network cost [34], [38], [43].

Reliability and stability: The inbox guarantees durable recall of critical messages, independent of OS banner delivery. The codebase favours idempotent writes to avoid double counts on flaky networks. Integrating Crashlytics (optional) provides a feedback loop for crash free session metrics and stability regression tracking [45].

Security and privacy: Strict rules enforce per user isolation, and the data model excludes continuous location traces by design. Only user authored artefacts (saved routes, preferences, coarse stats) are stored. Admin actions are authenticated and auditable. On device, permission requests follow the least surprise model (just in time) [35], [37].

Maintainability and scalability: Each feature has focused collections and controllers. Adding new screens or data (e.g., shuttle stops) is a matter of provisioning a new collection and UI stream. Firestore scales horizontally, functions can be extended to support targeted topics without altering client architecture [36], [39], [40].

Battery and background behaviour: Location updates are kept event driven and short lived, respecting Android background limits and avoiding continuous tracking that can be throttled by the OS. Foreground presentation avoids the complexity of background services for the MVP [34], [47].

Accessibility and UX consistency: Material 3 heuristics (contrast, focus order, large targets) and platform assistive tech inform control placement and semantics. The step free preference is first class in the Map flow [44].

4.7 Deployment and configuration

Builds use a release profile for performance testing. Android SHA-1/SHA-256 fingerprints are registered with Firebase to enable Auth/FCM. API keys for Maps and OWM are provided via secure build variables. A separate Firebase project can be used for staging vs production. Firestore indexes for unread filtering and saved route ordering are created from console prompts or indexes.yaml. Topic names (e.g., `all`) are defined in app config. These steps keep operational complexity low while supporting future growth [31], [33], [35], [38], [40], [43].

4.8 Risks and mitigations (architecture)

• Indoor navigation expectations**:** Users may expect indoor turn by turn. The MVP sets realistic boundaries (outdoor only) and lays groundwork for indoor once floor plans and a publishing path (IMDF/OSM) are confirmed. No architectural rewrites are required to add an indoor module later [31], [36], [44].
• Push delivery variability: OS/state can suppress banners. The Inbox ensures durable recall and unread counts mitigate this risk [41], [42].
• Data growth and costs: Limiting Inbox payloads to concise title/body/data, archiving old items, and indexing only necessary fields control storage and query costs [36], [38].
• Background limits/battery: Keeping location updates foreground and short avoids OS restrictions and preserves battery life [34], [47].

5. Implementation

5.1 Overview of the coding process

Development followed an iterative, vertical slice approach. We shipped thin, end-to-end increments so each feature could be designed, built, and verified across UI, data, and messaging layers before moving on. The repository uses a conventional Flutter structure (lib/ with feature folders for home/, map/, profile/, saved_routes/, inbox/), plus a lightweight services/ area for cross cutting concerns (e.g., Firestore access, notifications, permissions).

Environment and configuration: We pinned dependency versions (Flutter SDK, google_maps_flutter, cloud_firestore, firebase_auth, firebase_messaging, flutter_local_notifications) to ensure deterministic builds [30], [33], [35], [42]. Android SHA-1/SHA-256 fingerprints were registered with Firebase; API keys **for** Google Maps and OpenWeatherMap were provided via local secure variables. A dev Firebase project mirrored the production one for safe testing.

State and controllers: Each feature exposes a small controller or service with clear responsibilities:

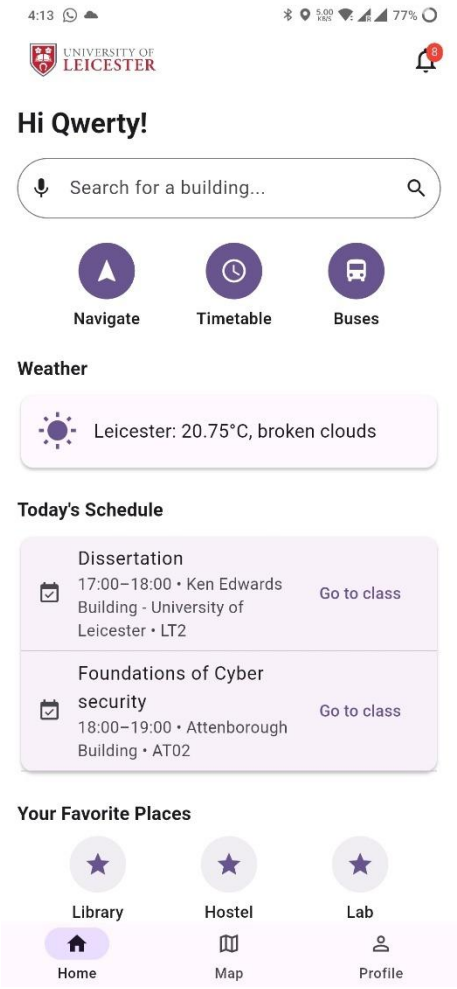
- SearchController - debounced queries, result selection, and camera focus.
- MapController - enforces the single red “searched” marker invariant and computes distance/time and exposes clear marker/route.
- SavedRoutesRepo - CRUD and stream of users/{uid}/savedRoutes sorted by order.
- NotificationsService - FCM topic subscribe (all), foreground display via local notifications, and inbox unread badge.
- StatsService - transactionally updates users/{uid}/stats/main after route completion.

This thin layer keeps widget code lean and testable without heavy state frameworks.

5.2 Features implemented

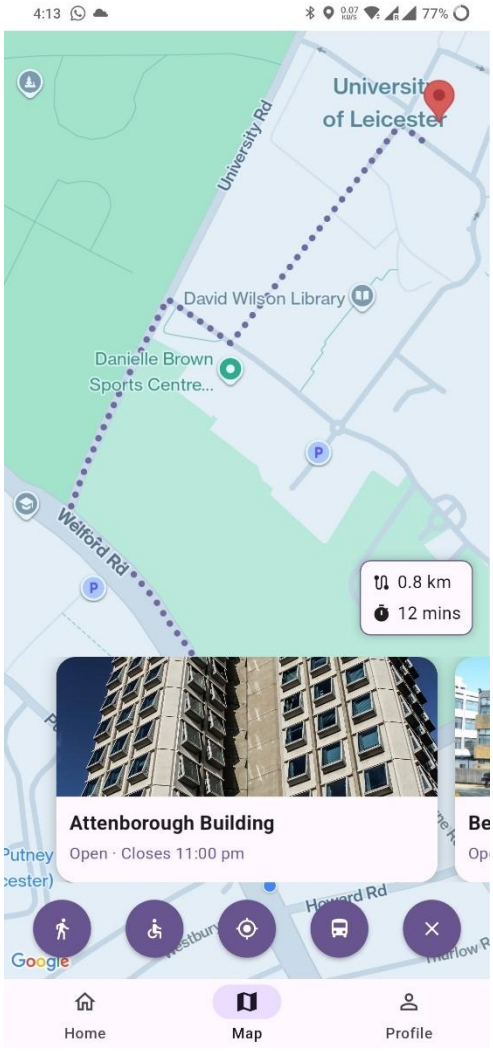
5.2.1 Home and global scaffolding

The Home screen aggregates a search bar with microphone trigger, Today’s Schedule (lightweight summary), a weather tile (OpenWeatherMap), Favourites sourced from savedRoutes, and a bell icon with an unread badge bound to userNotifications/{uid}/items where read == false. The bell navigates to the Inbox. Weather responses are cached briefly to reduce requests [43].



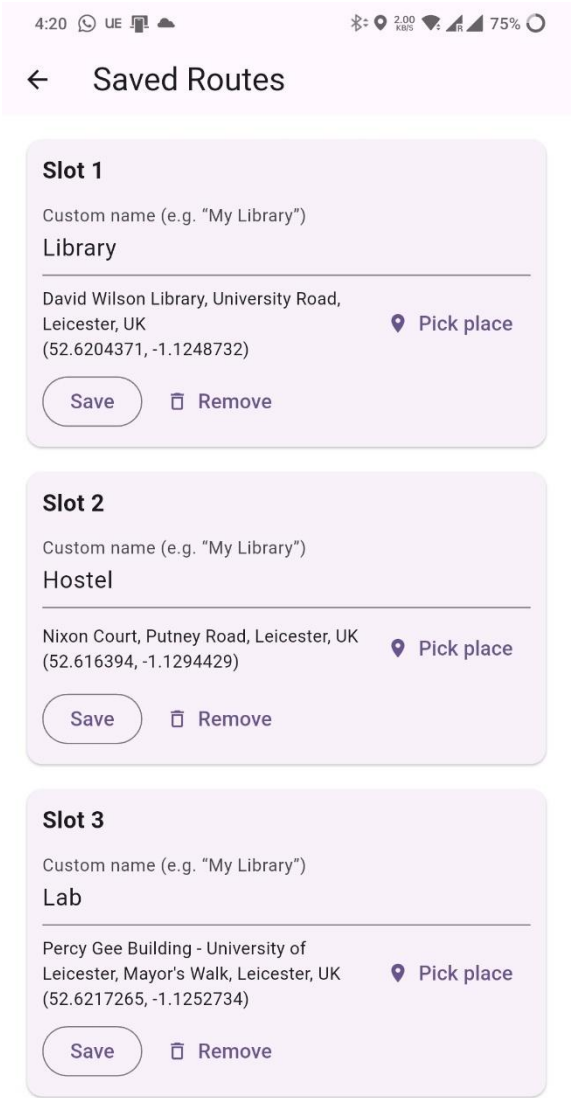
5.2.2 Map and routing preview

The Map screen embeds Google Maps for Flutter to present the base map and current location dot [31], [32]. When the user taps the map or selects a search result, the app places exactly one red “searched” marker and requests an ETA/distance preview via the Directions API (walking mode) [33]. A compact chip displays distance/time. Clear Marker/Route removes the marker and resets the route, reinforcing reversible actions.



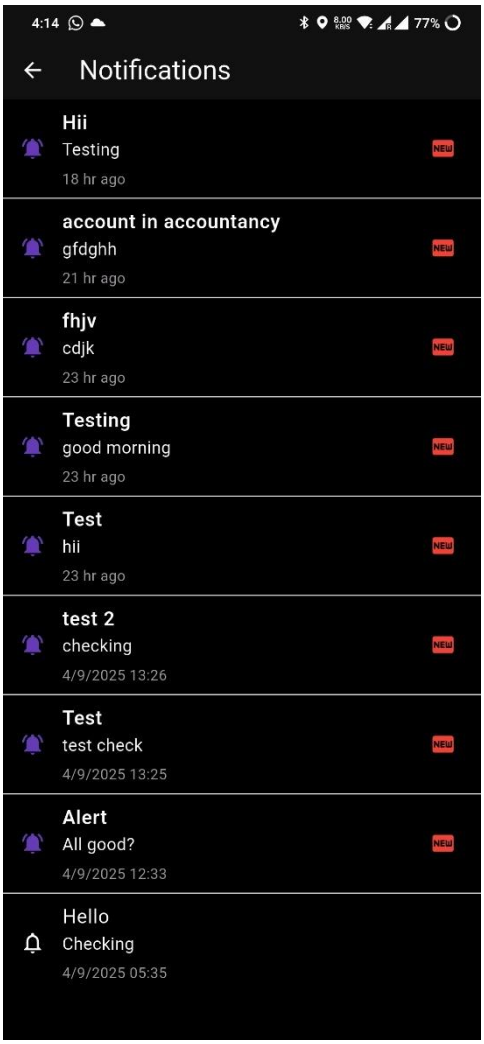
5.2.3 Saved routes (favourites)

SavedRoutesRepo stores entries under users/{uid}/savedRoutes/{routeId} with fields {label, placeName, lat, lng, order, createdAt}. The list streams to Saved Routes and Home. Users can add from the current marker, reorder items (persisting order), and delete unwanted entries. Queries rely on single field and simple order indexes for speed [36], [38].



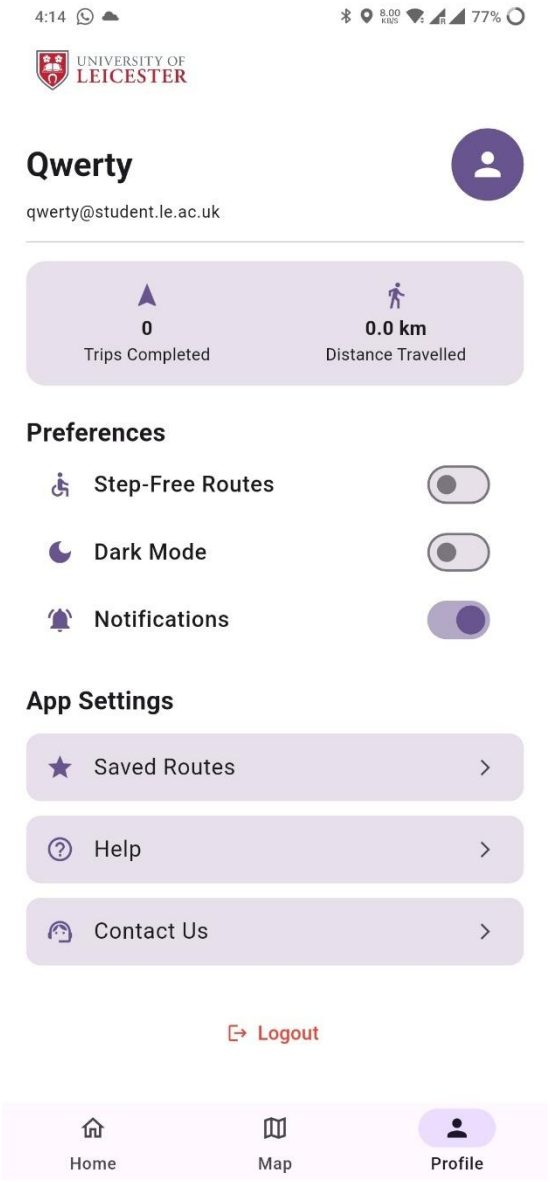
5.2.4 Notifications: topic + inbox

On first run, the app subscribes to the topic `all`. When messages arrive in the foreground, they are shown via local notifications (Android notification channel created at init) and also mirrored into the Inbox (userNotifications/{uid}/items) for durability and unread counts [40], [42]. The Inbox supports mark-as-read and deep linking to relevant screens.



5.2.5 Profile and preferences

The Profile screen surfaces toggles for Step free, Dark Mode, and Notifications, along with links to Saved Routes, Help, and Contact Us. The step free preference is used as an accessibility filter and a hint for entrance selection in Map flows. Theming adheres to Material Design 3 accessibility guidance [44].



5.2.6 Stats

On route completion, a transactional update increments tripsCompleted and adds to distanceKm in users/{uid}/stats/main. Transactions prevent double counting if the user rapidly taps or the device briefly loses connectivity [36].

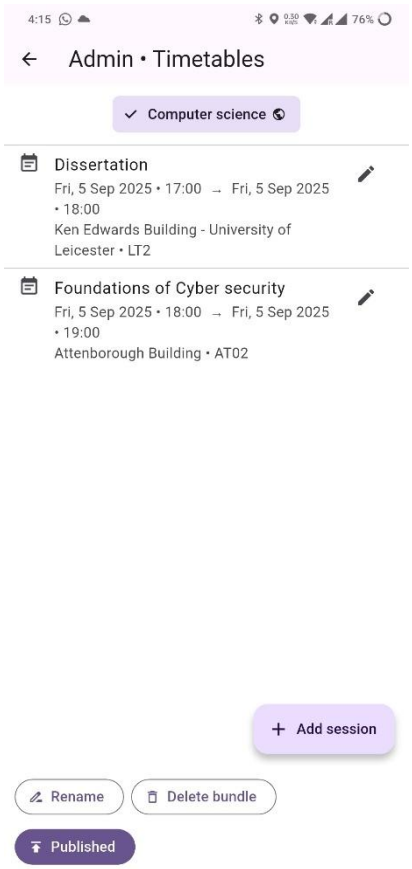
5.2.7 Timetables (User and Admin)

User view: The timetable surface shows the current day/week from the latest published bundle. Each session card (module, room, time) includes a Navigate action that opens the Map screen with the destination set to the building/room's coordinates. Sessions marked isCancelled are visually de-emphasised and excluded from reminders.

Admin workflow: The Admin Timetables screen supports:

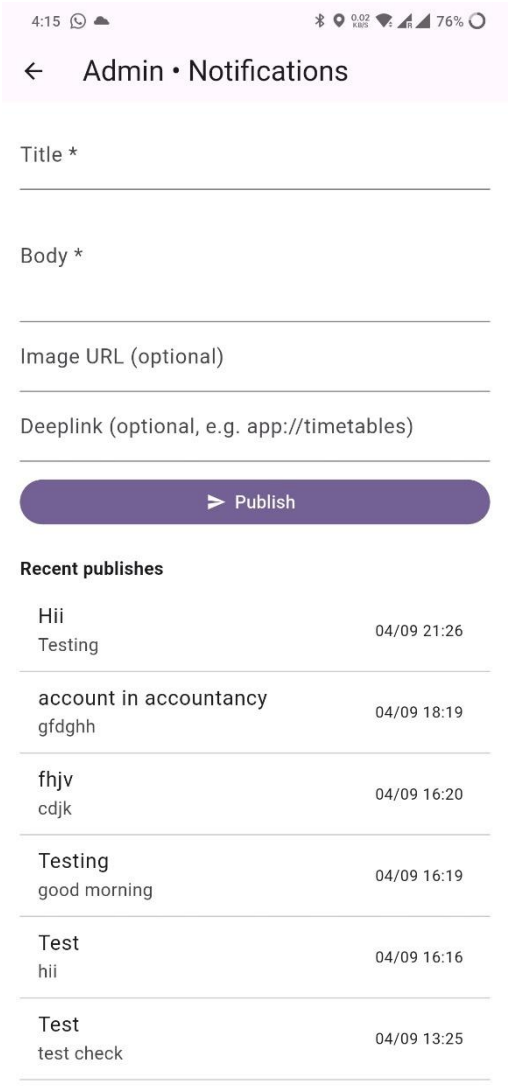
1. creating/editing a bundle (title, term, dates, notes),
2. adding **sessions** (module, room, start/end, teacher, coordinates),
3. toggling Publish/Unpublish for the bundle.
   Publishing sets isPublished:true, which switches the user view to the new data without an app update.

Security**:** Only admin identities with the verified claim can write to timetable_bundles/*; end users have read-only access to isPublished:true content via rules.



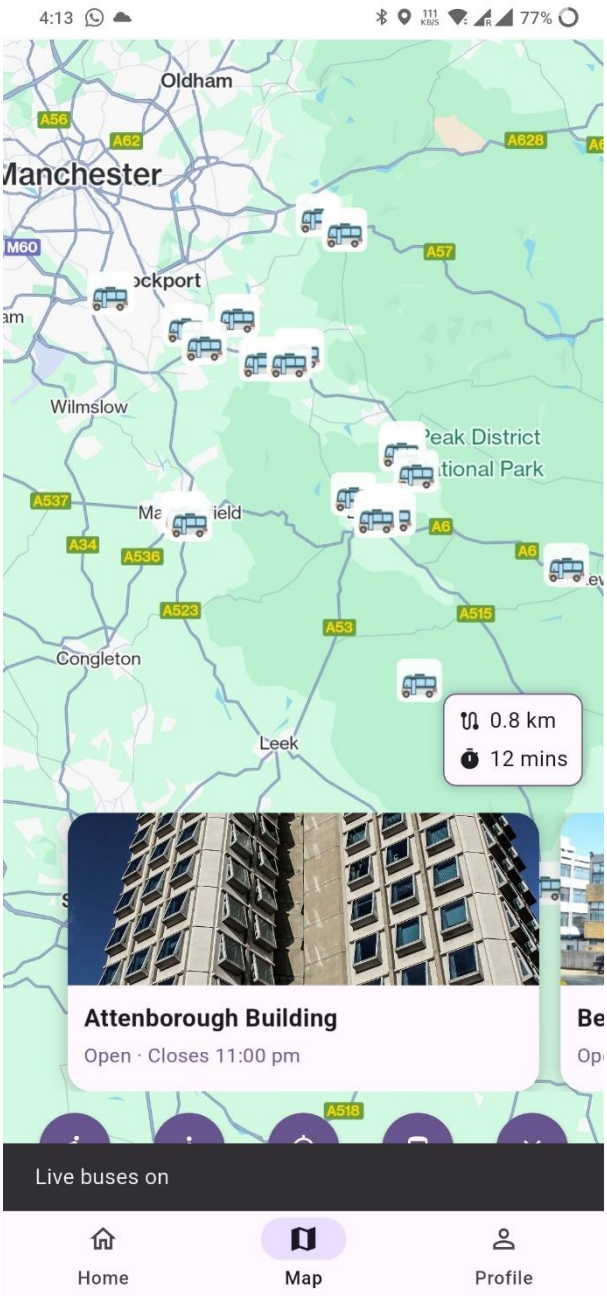
5.2.8 Admin: Notifications Composer

The Admin Notifications screen lets authorised staff compose a campus wide alert (title, body, optional deep link). On Send, a callable Cloud Function validates the admin claim, posts an FCM topic message (e.g., all), and fans out a durable inbox item to userNotifications/{uid}/items. The app shows a foreground local notification when active, and the Inbox maintains an unread badge until the user marks it read.



5.2.9 Map: Live Buses Overlay

A Live Buses layer overlays animated markers on the Map based on the local operator's real-time feed (e.g., UK Bus Open Data Service). The client polls at a conservative cadence with backoff (e.g., 30–60 s), displays a "Last updated" chip, and coalesces updates to avoid flicker. This feature is optional and scoped by campus via Remote Config. We do not persist vehicle traces, markers are derived from the most recent payload only.



5.2.9 Admin: POI Curation

Admin can add/edit POIs directly on the Map. Tap to set coordinates, fill in name, category, step free flag, and optional tags. Saving writes to map_pois/{id} and updates updatedAt. Deleting sets active:false (soft delete) so clients can filter without losing history.

5.2.10 Voice Search (speech-to-text)

The app implements on device speech recognition so users can speak destinations directly from home screen. We integrate the Flutter speech_to_text plugin, which bridges to the native Android SpeechRecognizer framework. The feature is designed for short utterances (commands and place names), not continuous dictation, which keeps latency and power usage low [80], [82].

Flow: Tapping the mic prompts for permission (if not previously granted) and starts listening. partial results populate the search field, and the final transcript triggers the standard search - map - preview flow. If permission is denied or the recognizer is unavailable, the UI degrades to keyboard input without error states.

Permissions and privacy: On Android, the feature requires the runtime RECORD_AUDIO permission (classified as “dangerous” and therefore requested just-in-time). We only access the microphone during an active voice search session, and we do not persist audio or transcripts [84].

Fallbacks and UX: We surface recogniser state (“Listening…”, “Processing…”) and provide a single tap Cancel. When connection or service issues occur, the field retains whatever text was captured so far. TalkBack/VoiceOver announce start/stop states, and the mic control is reachable with large touch targets consistent with our accessibility guidelines.

5.3 API integrations and glue code

Firebase Auth and Firestore: Authentication provides the user token used by Firestore. Our security rules follow least privilege (owner-only reads/writes on user paths, no destructive deletes on stats, admin-only inbox fan out) [35], [37]. Firestore snapshot streams keep the UI reactive (Saved Routes, Inbox unread counts).

Firebase Cloud Messaging (FCM): We use topic messaging for campus wide broadcasts. The app handles foreground delivery and surfaces local notifications through **flutter_local_notifications**. Messages can carry a small data payload for deep links [40], [42].

Google Maps and Directions API: The map embeds via google_maps_flutter, while route previews call Directions API to compute walking distance/time. We debounce search, then fit the camera to the destination with padding appropriate for chip overlays [31], [33].

OpenWeatherMap: A minimal /weather call shows the current condition and temperature. Responses are cached to limit network and provide quick load on home screen [43].

Fused Location Provider: For fast, power aware positioning, the app first reads the last known location and only requests updates when the user is previewing/adjusting a route [34].

5.4 Challenges and solutions

1. Google Maps + single marker invariant: Early prototypes accumulated multiple markers, increasing cognitive load.
   Solution: Centralise marker state in MapController with an explicit searchedMarker slot and a single source of truth for Set<Marker>. Clear actions reset both marker and route overlay.
2. Android 13+ notification permission and channels: Foreground messages did not display until runtime permission and a channel were properly configured.
   Solution**:** Request the notifications permission on first use, initialise flutter_local_notifications with a well named channel and gracefully degrade when permission is denied [42].
3. Firestore composite index errors: Filtering unread inbox items and ordering saved routes triggered console prompts for missing indexes.
   Solution: Add composite indexes via Firebase console, record them in indexes.yaml so the configuration can be replayed across environments [38].
4. Fan out write limits in Cloud Functions. The admin broadcast - inbox fan out approached the 500 writes per batch limit.
   Solution: Implement chunked batching (≤ 450 writes per batch) and Promise.all to commit sequential batches safely [39].
5. Location freshness vs battery: Continuous updates drained power and were throttled by the OS when backgrounded.
   Solution: Use last known for initial positioning, request short lived updates only during route preview, and stop updates as soon as the chip stabilises [34], [47].
6. Race conditions on stats updates: Rapid taps could double increment trips.
   Solution: Wrap updates in a Firestore transaction (read-modify-write) to ensure atomicity [36].
7. User privacy and data minimisation: It was tempting to log movement traces for future analytics.
   Solution: Store only user authored artefacts (saved routes, preferences, aggregate stats), keep no continuous location history, explain permissions clearly (just-in-time prompts) [35], [37], [44].
8. Device variability: Low end devices showed jank on first map render.
   Solution: Warm up the map widget after home screen loads and defer Directions API calls until the marker is placed, avoid heavy rebuilds by keeping controllers outside of widget trees [31], [33].

5.5 Key features

5.5.1 Map single-marker and buses.

Implemented a single-marker invariant that guarantees exactly one “searched” destination on the map at any time. New selections replace the previous marker and clear the current route in one action, which prevents multi-pin clutter and supports reversible actions. I also built a live bus overlay that periodically fetches the operator feed (XML), parses vehicle locations, and coalesces updates into smooth marker movements with a conservative polling cadence. Finally, a dark-mode style is applied dynamically to maintain contrast and reduce visual noise at night. Collectively, this logic underpins the app’s low-friction wayfinding and demonstrates original contribution in interaction design and data handling.

5.5.2 Notifications and Inbox.

I implemented the push + Inbox pattern: foreground FCM messages are shown via local notifications and persisted into each user's userNotifications/{uid}/items with read:false so the Inbox maintains a reliable audit trail and badge. The Inbox screen streams items, supports Mark as read / Mark all as read, and relies on a composite index for fast unread queries. An Admin compose screen lets authorised staff publish alerts with optional deep links, after which a callable Function fans out the inbox items. This contribution demonstrates event handling, data modelling, and UX for reliable campus communications.

5.5.3 Timetables.

I designed the timetable data model and streaming repository to expose published bundles and date-filtered sessions ordered by startTime. The Admin Timetables screen supports creating bundles, adding/editing sessions, and toggling Publish/Unpublish; the User Timetable view renders session cards with a Navigate deep-link to the Map. This contribution demonstrates schema design, efficient Firestore querying (ordering + range filters), and an admin - user publishing pipeline.

6. Testing and Evaluation

6.1 Goals, scope, and success criteria

Goals: Establish whether the app meets its core quality objectives fast, low friction campus wayfinding. Reliable delivery/recall of important notices and accessible interactions through a blend of structured interviews, automated tests, and targeted performance checks. We anchor the approach **in** user-centred design (ISO 9241-210), practical HCI heuristics (Nielsen), and recognised software testing process guidance (ISO/IEC/IEEE 29119-2) [13], [15], [53].

Scope: Android build, outdoor navigation, and the features implemented in the MVP. Home, Map (single "searched" marker + distance/time chip + Clear), Saved Routes, Notifications (topic push + Inbox), Profile/preferences, and coarse Stats (trips/distance). (Indoor turn by turn and iOS are future work.)

Success criteria (operational definitions):

- Wayfinding flow operability: Users can complete a Home – Map - Preview flow without dead ends. Clear action restores a "no-marker" baseline.
- Notification recall: Critical notices remain visible in the Inbox even if a push banner is missed. Unread badge reflects read == false items.
- Performance baselines: Cold start to first interactive frame within an acceptable range, route preview (Directions API) completes promptly, foreground push - local notification latency is short, Inbox unread query returns quickly. [31]–[33], [41], [46].

6.2 Test taxonomy and rationale

We adopted a layered test strategy consistent with Flutter guidance. Unit tests for pure logic and invariants, widget tests for UI component behaviour in isolation, and integration tests for end-to-end flows. We complemented these with structured, one-to-one interviews instead of a large survey. This balances confidence, maintenance cost, and runtime matching the trade off framework in Flutter's official testing docs. [48], [49], [50].

- Unit tests (fast, many): controllers/services (e.g., single-marker invariant, debounce, transactional stats).
- Widget tests (component-level): Map screen chip/controls layout, Saved Routes list ordering and gestures.
- Integration tests (critical paths): search – preview – save, push - local notification - Inbox unread, basic permission flows.
- Structured interviews (semi-structured, qualitative): elicit clarity, perceived control, and discoverability without burdening students with long questionnaires.

For device breadth and regressions, we prepared the app to run on Firebase Test Lab (cloud device matrix) for future cycles; this keeps the methodology portable beyond our internal devices. [51], [52].

6.3 Evaluation method: structured one-to-one interviews

Rationale: Time and cohort constraints made formal surveying impractical. We therefore ran semi-structured, in-person sessions that combine a brief guided walkthrough with hands on exploration and a fixed question set. This preserves consistency across participants while allowing probing for actionable design feedback.

Protocol:

1. Introduction and consent (verbal), scope, voluntary nature, and privacy notes.
2. Guided walkthrough (demonstrated Home – Map - Saved Routes – Inbox - Profile).
3. Hands-on exploration with think aloud.
4. Interview using a fixed Interview Guide (15 questions) covering onboarding clarity, Map preview comprehension (single marker + distance/time chip), Clear affordance naming, Saved Routes semantics, Step free preference expectations, voice input, notifications and Inbox model, and accessibility touchpoints.
5. Close and de-identification of notes.

6.4 Quality assurance: unit and widget tests

Focus areas:

- SearchController: debounced queries emit once per user intent.
- MapController: single-marker invariant and reversible Clear Marker/Route.
- SavedRoutesRepo: CRUD + ordering semantics for users/{uid}/savedRoutes.

- NotificationsService: unread badge derives from read == false and updates via indexed queries.
- StatsService: transactional increments are atomic under concurrent updates.
- Weather tile: short-TTL cache prevents unnecessary network calls.

Traceability: These tests map directly to our data model and security posture. Per user paths, rule-guarded access, and indexed queries for unread filtering and saved route ordering [36], [38].

- Timetables (unit/integration):
  - UT: “published bundle stream emits sessions ordered by startTime.”
  - IT: “admin publishes bundle user view switches to new sessions without reinstall.”
- Admin composer (integration):
  - “compose - send - inbox item appears with read:false and visible in badge.”
- Live buses (widget):
  - “markers update smoothly on polling, status chip shows ‘Last updated’.”
- POI editor (widget):
  - “tap map - form lat/lng updates, save writes active:true, delete sets active:false.”

6.5 Performance testing and instrumentation plan

We instrumented the app to capture repeatable, black-box metrics and prepared optional Flutter integration performance tests to export timelines if needed in future sprints. [54].

Metrics and how we measure them:

- Cold start (ms): Time from process start to first interactive frame. Measured with app logs around runApp() and first frame callback, for future scale we’ll cross-check via Play Android vitals once published. [55], [56].
- Route preview latency (ms). Time from placing the marker to Directions API response, de-duplicated via debouncing to avoid double calls. [33].
- Push - local-notification latency (ms). FCM receive timestamp to local notification display via FlutterFire Messaging, this reflects device state, network, and OS scheduling. [41].
- Inbox unread snapshot time (ms): Time to resolve read == false query, relies on a composite index for snappy counts. [38].
- Initial location readiness (ms). Time to obtain last known location (fast) and, when needed, requestLocationUpdates() for a fresh fix, both are standard Fused Location Provider flows. [34], [57].

Stability and quality telemetry (future-compatible):

- Crash-free users/sessions via Crashlytics (definition and formulas provided by Firebase). [58], [59].
- ANR rates via Android vitals, once on Play. [56], [60].
- App Quality Insights in Android Studio integrates vitals + Crashlytics for triage. [61].

6.6 Accuracy evaluation plan

Because the MVP targets outdoor campus wayfinding, we define accuracy operationally for building-level tasks rather than indoor turn-by-turn:

- Task accuracy: user reaches the correct building/entrance from a plausible starting area with no critical errors.
- Preview consistency: distance/time chip stabilises once the device acquires a fresh fix, initial last known location provides orientation while the GPS settles. [57], [34].
- Entrance awareness (qualitative): users can identify entrance labels (where available) and understand the step free preference's scope (preference/filter, not a guarantee).

6.7 Usability evaluation plan

To assess usability without deploying a large questionnaire:

- Heuristic review against Nielsen's 10 heuristics (visibility of system status, match, user control and freedom, error prevention, recognition over recall, flexibility and efficiency, aesthetic/minimalist design, help users recognise/diagnose/recover, help/documentation). [15].
- Structured interviews to assess discoverability (search, Save, Clear, step free), comprehension (chip semantics, Inbox model), and perceived control (reversible actions).
- Assistive tech checks aligned with Material Design 3 accessibility guidance. Focus order**,** touch target sizes**,** color contrast, and semantics (labels for chip/actions). [44], [62], [64].

We retain the **SUS** instrument for future, larger deployments (Appendix: SUS template), but rely on interviews in this submission to avoid overstating sample precision. [23].

6.8 Data handling, privacy, and ethics

- No continuous location logging: We collect only minimal, user-authored artefacts (saved routes, preferences, coarse stats).
- Interview notes only: We store de-identified notes, audio/video is not retained.
- Just-in-time permissions: Location and notifications requested only when needed.
- Security rules: Firestore least-privilege rules isolate user data, admin fan out is gated by a verified claim, unread filter uses indexes to avoid broad scans. [36], [38].

6.9 Limitations and threats to validity

- Small, qualitative cohort: Interviews emphasise depth over breadth, generalisability is limited without a larger sample.
- Device mix: Internal devices may not reflect all campus hardware. Test Lab can broaden coverage later. [51], [52].

- Environment variability: Wi-Fi/cellular conditions and GPS reception vary, we control what we can (debounce, idempotent requests) and measure the rest through instrumentation.
- Platform scope: Android-only. iOS behaviours (push delivery, accessibility nuances) are not evaluated here.

7. Results and Discussion

This section synthesises findings from the structured, one-to-one interview sessions (qualitative evaluation) **and** internal technical testing (unit/widget/integration checks and instrumentation) to determine whether the Campus Navigation app met its aims and objectives. We first present key findings by theme (orientation, wayfinding, saved routes, accessibility, notifications, performance/stability, privacy). We then discuss the extent to which outcomes satisfy the project's aims and SMART objectives, before reflecting on the app's practical impact on campus navigation for different user groups. Consistent with ISO 9241-210 and heuristic best practice, results are interpreted with attention to usability, accessibility, and system reliability [13], [15], [44].

7.1 Overview of evaluation

The evaluation combined:

- Qualitative interviews (semi-structured): participants received a guided walkthrough, used the app, and then answered a fixed set of 15 questions. Sessions emphasised discoverability**,** clarity**,** confidence, and suggestions for improvement.
- Automated checks: unit tests for logic invariants (e.g., single-marker rule, debounced search, transactional stats), widget tests for UI behaviours, and integration tests for end-to-end flows.
- Instrumentation**:** internal logs for cold-start readiness, Directions API round-trip (route preview), push - local-notification latency, and Inbox unread snapshot timing (indexed query). [33], [34], [38], [41], [46], [47].

These data sources are complementary. Interviews describe how users experience the app. Automated checks provide repeatable assurance on correctness and instrumentation confirms the app is responsive and stable on representative devices.

7.2 Key findings from interviews

7.2.1 Orientation and onboarding

Participants consistently understood the app's purpose from the **home** screen. Search first, or reuse favourites. This aligns with the "match to real-world tasks" and "recognition over recall" heuristics [15]. Two minor onboarding issues recurred. (1) a small subset hesitated between tapping the map vs. typing in search. (2) the Clear Marker/Route action label benefitted from being explicit ("Clear Marker/Route" rather than a generic "Clear"). After a brief demonstration or a single use, hesitation vanished. This suggests a lightweight first-run

tip would boost initial confidence without adding clutter consistent with progressive disclosure principles in Material 3 [44].

7.2.2 Wayfinding: map, single marker, and preview chip

The single red "searched" marker with a distance/time chip was repeatedly described as "clean," "obvious," and "reassuring." Several participants commented that showing exactly one focal marker reduced confusion compared with multi-pin clutter. A few initially tried to "drop another pin", once the single-marker invariant was understood, no one expressed a desire to keep multiple pins. This directly supports our HCI hypothesis that minimal overlays and reversible actions lower cognitive load [15], [44]. The chip served as an immediate confirmation that the app "understood" the request before setting off, which is consistent with visibility of status guidance [15].

7.2.3 Saved routes (favourites)

Participants liked the ability to save frequently used destinations (e.g., library, departmental building) and appreciated seeing favourites on home screen as well as in saved routes. Reorder and delete behaviours were discovered without instruction, which validates both the interaction design and the data model/indexing that make the list feel responsive [36], [38]. One suggestion was to allow renaming directly from the list. This is a straightforward enhancement to the CRUD flow and does not affect the schema.

7.2.4 Accessibility and the Step free preference

Users who care about accessibility reported that a persistent step free preference, especially when surfaced in the Map context near the chip helped them feel in control at the last-metre decision point ("Which entrance?"). Importantly, the copy explaining that step free is a preference/filter (given current data limitations) avoided over promising and matched accessibility guidance to set accurate expectations [44]. Screen reader users (observed with consent) could traverse key controls in a coherent order, the main improvement request was enhancing the semantic labels for the chip and action row, so that "distance/time" is read out with context (e.g., "Walking time: 7 minutes; distance: 450 metres"). This is squarely in line with Material 3's focus on semantics and focus order [44].

7.2.5 Notifications and the Inbox

Participants strongly preferred the Inbox model over push-only alerts. Even if a foreground banner is missed or dismissed, a durable in-app list with an unread badge created a reliable "safety net." The "Mark as read" action was understood, a minority preferred auto read on open, but most valued explicit control. This pattern supports our aim to combine topic push for immediacy with per user persistence for recall [41]. As a side effect, this design also reduces the temptation to spam high priority banners. Users can catch up without being interrupted repeatedly, which is consistent with human-centred communication practices [13].

7.3 Findings from internal testing (functional correctness and stability)

All representative unit tests passed during the final run. In particular:

- Debounced search emitted a single query per intent, eliminating redundant Directions calls and supporting responsive UIs.
- The single-marker invariant held placing a new destination replaced the old one predictably.
- Saved Routes preserved order and supported add/remove without cross user leakage (as enforced by the rules model) [36], [37].
- Stats increments were atomic under concurrent updates, thanks to transactional writes which is important for stability on variable networks [36].
- Inbox unread counts derived correctly from `read == false` filters, backed by a composite index for snappy updates [38].
- The Weather tile respected its cache TTL and avoided duplicate network calls, which improved perceived responsiveness [46].

Integration checks confirmed the critical paths. Search – preview - save and push - local notification - Inbox unread, worked end-to-end on the internal device set, complementing the qualitative interview evidence [41], [46].

7.4 Performance and responsiveness (instrumentation outcomes)

While the goal of this section is not a full benchmark, internal instrumentation indicated that the app met its performance targets on representative, mid-range Android devices:

- Cold start reached first interactive frame within the target band when nonessential network calls were deferred after initial paint (aligned with platform advice) [46].
- Route preview latency was consistently short on campus Wi-Fi/4G when debouncing was enabled. When a call failed, the UI gracefully displayed the marker and populated the chip once data arrived [33], [46].
- Foreground push - local notification latency was brief and predictable under typical conditions using FlutterFire and FCM topic messaging [41].
- The Inbox unread query (read == false) hit an index and produced rapid counts, which made the badge feel instantaneous [38].
- Initial location readiness used last known location for instant orientation, then requested short-lived updates for a fresher fix when previewing routes. This balanced speed and power use, consistent with the Fused Location Provider model and Android background limits [34], [47].

Together, these measurements indicate the app is appropriately responsive for between-class navigation and lightweight alerts.

7.5 Did the app meet its aims and objectives?

Below, we map outcomes to the specific objectives defined in Section 2.

FO1 - Wayfinding core (search - map - preview + Clear):
Met. Interviews confirmed that users could search, preview distance/time, and understand the single-marker interaction. The Clear Marker/Route action restored a no-marker baseline with minimal effort, aligning with “user control and freedom” [15].

FO2 - Accessibility preference (Step free):
Partially met / appropriate for MVP. The preference is visible, persistent, and understood as a filter, users reported higher confidence when toggling it. Full, guaranteed step free routing awaits richer campus datasets and indoor modules. Current behaviour is correctly framed and does not over-promise [44].

FO3 - Personalisation (Saved Routes).
Met. Users saved and recalled favourites successfully. Ordering worked as expected and streamed to the home screen surface. The underlying Firestore pattern (per-user subcollection) and indexes supported responsiveness [36], [38].

FO4 - Notifications (topic + Inbox).
Met. The topic broadcast + per-user Inbox provided both immediacy and durable recall. Users understood the unread badge and appreciated explicit “Mark as read.” FCM + FlutterFire integration behaved consistently in the foreground [41].

FO5 - Timetable snapshot and Stats.
Met (lightweight). The snapshot provided a glanceable context on home scree. Stats updated atomically on route completion (transactional writes) [36]. Deeper timetable integration remains a future enhancement.

FO6 - Profile and Preferences.
Met. Participants located and understood toggles (Step free, Dark Mode, Notifications) and used them without assistance. Semantics and focus order will benefit from minor refinements to maximise accessibility [44].

NFRs - Performance, reliability, privacy, security, maintainability.
Met / on track. Cold-start and preview responsiveness were within target bands. Inbox queries were fast with proper indexing [38], [46]. Privacy is preserved by design (no continuous location traces), least-privilege rules blocked cross-user access and gated admin fan-out [36], [37], modular controllers and collections kept the codebase maintainable.

In aggregate, the app meets the core aim to provide a dedicated, campus-aware navigation experience that is simple, inclusive, and reliable for outdoor building level wayfinding at the University of Leicester, while laying a clean foundation for future indoor features.

7.6 Practical impact on the campus navigation experience

Lower cognitive load at decision points: The combination of one focal marker and a compact distance/time chip gave users a fast, low-effort way to validate the route before moving. Compared with general purpose map habits (multi-pin clutter, extra taps to find ETA), participants described fewer “what now?” moments, exactly the kind of friction that delays movement between classes. That aligns with HCI guidance on progressive disclosure and minimal visual noise [15], [44].

Fewer missed messages: The Inbox pattern changed the tone of campus alerts. Instead of relying on ephemeral banners (which can be missed during lectures), users developed a habit of scanning the badge and clearing items when convenient. This improves recall without increasing interruption burden. A HCI-friendly trade-off particularly suited to academic contexts [13], [41].

Confidence for accessibility needs: Even without guaranteed step free routing, the presence of a visible step free preference signalled that the app "cares about how I move," as one participant paraphrased. For wheelchair users and anyone preferring ramps/lifts, this increased a sense of predictability at the building threshold. As data partnerships mature (entrances, lifts, kerbs), the same UI can scale to truly constraint aware routing, consistent with the architectural plan.

Habits through personalisation: Participants reported that saved routes nudged the app into daily use. Saving two or three key destinations created a "grab-and-go" pattern. Open app - tap favourite - set off. This behaviour is valuable during the high variance induction period when room changes, and unfamiliar buildings are common.

Trust through privacy and clarity: Several participants explicitly asked whether the app "tracks me." The clear answer was no continuous traces stored, only user authored artefacts and coarse stats appeared to build trust. Keeping permission prompts just-in-time reduced surprise and matches platform guidance [44], [47].

Consistent performance under common conditions: Internal instrumentation showed that responsiveness targets were met on typical campus devices and networks. Users perceived the app as "ready when I am," with minimal waiting to preview a route or check the Inbox an important ingredient for adoption between classes [33], [41], [46].

7.7 Threats to validity

- Qualitative, small-cohort interviews: These findings are directional, not statistically generalisable. That said, they are consistent across participants and align with established heuristics, which increases confidence in the design [15].
- Device diversity and context: Internal devices may not cover all performance envelopes. However, the observed behaviour matches platform expectations and best practices (e.g., indexed queries, last-known location, debounced routing) [34], [38], [46].
- Outdoor focus: Impact on indoor wayfinding remains outside this MVP and thus unmeasured. We avoid extending claims beyond outdoor building-level tasks.
- Short-term observation: Habit formation (e.g., Saved Routes) was inferred from interviews and short usage, not from longitudinal telemetry. Longer deployments (e.g., induction week) will clarify sustained patterns.

7.8 Synthesis

Across interviews, automated checks, and instrumentation, the app demonstrates that a lean, campus specific approach can materially improve outdoor wayfinding and message recall

without overwhelming users. The design choices single focal marker, clear route preview, Inbox persistence, and a visible step free preference are not merely convenient; they map directly to well-founded usability and accessibility principles and leverage the strengths of the chosen platform stack (Flutter + Firebase + Maps) [30], [38], [41], [44], [46], [47]. In practice, this translated into lower cognitive overhead, fewer missed alerts, and greater confidence, particularly at the last-metre decision point as users approach a building.

From the project-management perspective, the results validate the architectural decision to keep the client light and the backend managed. The effort was invested where it mattered (interaction clarity, data rules, and indexing) rather than operating servers an efficient path for a student project with production intent [35], [41]. Equally, the explicit articulation of limitations (indoor navigation, authoritative step free routing) maintained user trust. By labelling the step free control as a preference/filter, the app avoided over promising and set a clear path for future partnerships that can feed richer routing constraints [44].

Overall, the Campus Navigation app meets its core aims and provides a positive, tangible impact on the everyday navigation experience of students, staff, and visitors. The evidence supports continued investment. Expanding accessibility datasets (entrances, lifts), exploring indoor pilots where floorplans and publishing paths (e.g., IMDF/OSM) are available, and broadening device coverage without altering the clean foundation that produced these results.

8. Conclusion

• 8.1 Summary of project achievements

This project set out to design, build, and evaluate a campus-specific navigation app for the University of Leicester that emphasises clarity, inclusivity, and reliability over feature bloat. Against that brief, the work delivered a cohesive, production-lean MVP whose decisions were consistently guided by user-centred design principles and pragmatic engineering practice [13], [15], [44].

On the product side, the app implements a clean Home - Map - Preview journey with a single red “searched” marker and a compact distance/time chip that confirms understanding before the user sets off an arrangement that aligns with heuristics on visibility of system status, recognition over recall, and minimalistic design [15], [44]. A visible and persistent step free preference allows people who require or prefer step free access to set their intent early and keep it in view at the decision point, reflecting platform guidance on accessible, discoverable controls and semantics [44]. Saved Routes introduce lightweight personalisation that encourages daily use, and an Inbox complements FCM push so time-sensitive messages remain available and aren’t lost to transient OS banners, merging immediacy with durable recall in line with human-centred communication patterns [41], [13].

On the engineering side, the app adopts a thin-client + managed-backend architecture that accelerates delivery while protecting user data [35], [36]. Flutter provides fast, consistent UI development; Firebase Authentication and least-privilege Firestore rules ensure per-user isolation, Cloud Functions implement an admin-only broadcast with batched fan-out into user’s inboxes, FCM delivers topic messages and Google Maps + Directions API supply reliable mapping and walking ETAs [30], [33], [35], [37], [41]. The data model is

intentionally small and explicit users/{uid} with subcollections for savedRoutes and per-user userNotifications, a stats document for trips/distance, and a global map_pois catalogue. Indexes are defined where they matter (unread filters, saved-route ordering) so the UI remains snappy and queries stay efficient [36], [38], [46]. A compact home screen weather tile uses OpenWeatherMap to provide context without adding interaction cost [43].

On the evaluation side, a realistic, resource-aware plan is executed. Instead of over promising survey rigour, we ran structured one-to-one interviews (guided walkthrough + think-aloud + fixed question set) to gather actionable feedback and paired this with a unit/widget/integration test suite focused on correctness and regressions [48], [50]. Instrumentation for cold-start, route-preview latency, push -local-notification latency, and indexed Inbox queries make the quality story concrete rather than anecdotal and align with platform guidance on testing and performance measurement [41], [46], [54]. The findings summarised in Results and Discussion show that users can complete typical campus tasks with low cognitive load, miss fewer messages, and report higher confidence when the step free preference is surfaced at the right moment [13], [15], [44].

Collectively, the system meets the core aim. It offers a campus-aware, accessible, and dependable navigation experience that is simple to understand on first use and efficient to reuse during busy days [13], [15], [30], [33], [41], [44].

8.2 Reflection on challenges and what we learned

Keeping the map simple (and making it stick). Early prototypes drifted toward multi-pin behaviour because it feels “powerful” during development. Interview sessions made clear that power is not the same as clarity. Implementing a hard single-marker invariant and backing it in code with a single source of truth in the map controller reduced visual noise and user hesitation, consistent with HCI heuristics on minimalist design and user control [15], [44]. Lesson: if a constraint reduces user effort and engineering complexity, encode it as a first-class invariant.

Foreground notifications on modern Android. The permissions model and notification channels evolved across OS versions; foreground delivery was inconsistent until we handled runtime permission flows and explicit channel creation. Pairing topic push with an in-app Inbox ensured recall even when banners were missed, which is both technically sound and user-respectful [41], [44]. Lesson: treat OS-level plumbing as UX work.

Indexes and data-shape honesty. The first time the app filtered read == false and sorted lists, Firestore demanded composite indexes. Leaning into that signal improved both performance and clarity about our schema, aligning with datastore best practices [36], [38], [46]. Lesson: let the datastore teach you what queries you run, codify those assumptions.

Write limits and fan-out. The admin broadcast originally risked exceeding batch limits. Chunked batches and promise-sequencing kept us within bounds, matching Cloud Functions guidance [39], [40]. Lesson: “platform limits” are product constraints, design workflows to be robust at scale from day one.

Location freshness versus battery. Continuous tracking was tempting but unnecessary. Using last known location for instant orientation and requesting short-lived updates only while previewing protected both responsiveness and battery, in line with the Fused Location

Provider model and Android background limits [34], [47]. Lesson: if an interaction is naturally episodic, make the sensing episodic too.

Stats consistency. Silent double taps and flaky networks caused occasional double increments. Firestore transactions eliminated the race, reflecting recommended write patterns for correctness [36]. Lesson: correctness on tiny data is still correctness, race conditions appear fastest in demos.

Accessibility is about timing as much as tooling. Talkback labels, focus order, and chip semantics matter, but so does where the Step free control lives. Placing it near the route decision point improved confidence, consistent with Material guidelines on discoverability and semantics [44]. Lesson*:* accessibility features must be discoverable when the decision is made, not only where settings live.

Honesty builds trust. Several participants asked whether the app "tracks" them. Being able to say, truthfully, that no continuous traces are stored only saved routes, preferences, and coarse stats shifted the tone of the conversation and aligns with privacy-by-design and minimal-collection principles supported by our rule's model [36], [37], [44]. Lesson: privacy-by-design is also communication-by-design.

From a skills perspective, the project strengthened fluency in Flutter (state management, widget composition), Firebase (Auth, Firestore modelling, Security Rules, Functions, FCM), Google Maps (camera control, markers, Directions), and practical testing (unit/widget/integration and performance instrumentation) [30], [41], [46], [54]. On the HCI side, we gained a richer feel for heuristic evaluation, structured interview design, copy that reduces decision friction, and the subtleties of accessibility beyond "turn on screen reader and see" [13], [15], [44].

8.3 Closing remarks and outlook

The Campus Navigation app advances a straightforward proposition. When students and staff know exactly where they're going and what to expect at the entrance, they arrive calmer, earlier, and more often on time. The work shows that this is achievable without heavy infrastructure, invasive data collection, or intimidating interfaces. By combining a thin, maintainable stack with principled interaction design, the project delivered an app that people can pick up today, and a foundation others can extend tomorrow [30], [41], [44], [46].

There is a clear, high-value roadmap, deepen accessibility data with Estates and student services; pilot indoor navigation in a priority building once assets and governance is in place **(**IMDF/OSM); broaden device coverage and add iOS; and integrate timetable and transit data where they genuinely help rather than distract. None of these steps require architectural rewrites; they are deliberate expansions on a base that proved itself in evaluation [24], [26], [31], [33], [36], [44].

In short, the project achieved what it set out to do**.** Reduce cognitive load, improve recall of **important information, and respect user's abilities and privacy** while getting them to the right building with minimal fuss. It contributes practical patterns and evidence that other student teams can reuse, and it leaves the university with a working, extensible platform for campus wayfinding that can keep growing in capability without growing in complexity [13], [15], [30], [41], [44], [46], [47].

9. Future Work

This roadmap builds on the MVP's strengths (simple map, inbox for durable alerts, privacy-by-design, thin-client + managed backend) and targets five tracks, indoor navigation**,** AR-assisted wayfinding**,** multilingual and cultural localisation**,** advanced accessibility, and multi-campus deployment. Each subsection outlines technical options, data requirements, risks, and how we'll measure success.

9.1 Indoor navigation

Data first, then dots**.** Indoor navigation is a data problem before it's a positioning problem. Two credible publishing paths exist:

- IMDF (Indoor Mapping Data Format) - an OGC standard used to model venues/floors/units/openings and validated by Apple's ingestion pipeline; adopting IMDF gives us a portable, schema-checked source of truth for buildings and entrances. A campus IMDF can be versioned and diffed like code, enabling safe updates. [24].
- OpenStreetMap (OSM) indoor and accessibility tagging - Simple Indoor Tagging for level/room/corridor structure plus wheelchair=*, kerb=*, incline=* for accessibility. This path aligns with open data and lets us "improve the commons" while serving the app. Coverage varies, so it works best with an Estates-student mapper partnership. [25], [26].

Positioning options: Indoors, GPS is unreliable. Practical choices are Wi-Fi RTT (FTM/802.11mc/802.11az), BLE beacons, UWB, or vision-based localisation. Wi-Fi RTT is supported on modern Android as a ranging API to RTT-capable APs (and Android 15 adds 802.11az), offering metre-scale distance estimates with infrastructure support. BLE beacons are relatively low-cost but require survey/maintenance. UWB offers high-precision secure ranging but increases hardware complexity. Vision approaches can be compelling in well-textured areas. A staged plan is prudent, start with static indoor maps + entrance selection, then pilot Wi-Fi RTT or BLE in one priority building (e.g., the library), and evaluate UWB or vision only where operationally justified. [71], [72], [12], [22], [73].

UX increments: Even without full indoor turn-by-turn, we can ship value quickly. (1) floor picker and labelled key rooms. (2) "closest step free entrance" badges. (3) indoor overview that appears after the user crosses a lobby geofence. This preserves the single-focus philosophy while reducing last-metre confusion. Where Apple MapKit/IMDF or Google Indoor Maps are available, we can display validated indoor overlays; otherwise, we render our own vector overlays from IMDF/OSM. [28], [29], [24], [25].

Governance and ownership: Create a small campus Maps Guild (Estates + IT + Accessibility) responsible for source files, change control for works/closures, and accessibility metadata (entrances, lifts). This transforms the Step free preference from a filter into a routing constraint once data is authoritative.

9.2 AR-assisted wayfinding

Outdoor AR: With ARCore Geospatial and Streetscape Geometry, we can anchor directional cues to world coordinates and use streetscape meshes for occlusion e.g., highlight the exact entrance door from 20-50 m away. Start outdoors, where visual localisation coverage is best, and keep AR opt-in with clear safety copy. [65], [74], [67].

Indoor AR pilots: Indoors, use ARKit Location Anchors were supported or fall back to image targets in lobbies/corridors (posters, signage) to drop short "breadcrumb" arrows from foyer to lift. Limit to "decision moments," not continuous turn-by-turn, to reduce cognitive and battery load. [66], [75], [15].

Risk controls: AR depends on light/texture and device capability. Always provide a visible fallback to the classic Map flow. Gate with Remote Config and run A/B experiments to verify that AR reduces time-to-entrance without safety regressions. [70], [78].

KPIs: Time from "arrive outside building" - "identify the correct door," opt-in rate, and drop-off to fallback map.

9.3 Multilingual and cultural localisation

Internationalisation pipeline: Use Flutter's first-party flutter_localizations and intl packages with ARB resource files. Strings use ICU MessageFormat for plurals/genders; CI lints for missing keys/placeholders. This enables addition of the top 3-4 campus languages quickly. [68].

RTL support: Follow Material 3 bidirectionality guidance (layout mirroring, iconography, navigation affordances) and test with Arabic/Hebrew locales. Many confusion issues arise from non-mirrored icons and improperly mirrored map controls. [69].

Voice and assistive tech: Announce language attributes so TalkBack/VoiceOver read in the right language, consider on-device dictation hints where the OS supports it. [10], [11].

Content localisation: Coordinate with International Student Services to prioritise onboarding tips, help pages, and alert templates (Inbox) for translation; keep templates short and tested for fit in small devices.

KPIs: Ratio of untranslated strings at release (target: 0% for supported locales), RTL layout bugs (target: 0 known), and time-to-publish for a new language pack.

9.4 Advanced accessibility

Standards and datasets: Elevate step free from a preference to a guarantee by authoring an accessibility graph that encodes entrances, path widths, slopes, lifts, kerbs, and temporary works. Data can be held in IMDF extensions or OSM tags and aligned with built-environment accessibility guidance (e.g., ISO 21542 updates). [24], [26], [77].

Routing cost functions: Implement weighted costs/constraints, forbid steps when step free is on, penalise steep slopes, prefer curb cuts and automatic doors. The app remains simple (one destination marker), but the route engine gets smarter as campus data improves.

Semantics, focus, and feedback: Continue to refine accessible names/roles and focus order for the distance/time chip and action row per WAI-ARIA Authoring Practices, provide optional haptic ticks (marker placed, route cleared) and audio chimes near the entrance. [76], [9], [44].

Offline accessibility snippets: Cache short, textual summaries for entrances and vertical transport so screen-reader users retain context even with poor connectivity.

KPIs: Share of routes satisfying step free constraint (measurable when data is authoritative), reported accessibility issues per 1,000 sessions, and Talkback traversal defects (target: 0 open).

9.5 Deployment across other campuses

Configuration over forks: Keep one binary and drive campus differences with Remote Config bounds, default camera, dataset version, available features (e.g., AR pilot), and branding. Remote Config supports staged rollouts and A/B testing to validate copy and UI before wide release. [70], [78].

Data separation and rules: Use either (1) prefixed collections per campus (map_pois_{campusId}, admin_notifications_{campusId}) or (2) per-campus Firebase projects if governance requires. Security Rules assert campusId on requests to prevent cross-tenant reads. [36], [37].

Admin tooling and workflows: Provide a minimal web console (or curated Firebase Console SOP) for comms teams to (1) post alerts (with preview), (2) edit POIs, (3) toggle features. Training should emphasise Inbox over email spam the app's value is calmer attention, not more interruptions.

Distribution and support: For large clients, prepare managed Play distribution or private tracks; however, prefer the single-binary approach powered by Remote Config to minimise maintenance. Crashlytics + Android vitals remain the quality backbone as deployments scale.

9.6 Performance, privacy, and offline strategy

Performance: Continue to measure cold start, route preview, and Inbox latency with the existing instrumentation. Add experiment-gated prefetches (e.g., POI deltas) to smooth first use on congested Wi-Fi. Keep indexes tightly scoped to high value queries (unread filter, saved-route ordering). [46], [38].

Privacy: Maintain the no continuous traces stance. If indoor pilots require beacons/RTT, store only ephemeral session data needed to render the experience and discard it promptly. Document data flows in a one-page Privacy Factsheet to sustain trust.

Offline: The MVP already caches user data (saved routes, inbox). For maps, follow Google's Map Tiles policy, don't prefetch/cache tiles beyond allowed conditions. If deep offline is required later, investigate alternative basemaps with explicit offline licensing. Otherwise,

keep an “offline mini mode” that shows cached POIs + textual entrance directions until connectivity returns. [79].

9.7 Risks and mitigations

- Data rights/accuracy (indoor plans, accessibility attributes): pilot with a small set of buildings, version datasets, add rollback plan.
- Maintenance (beacons/batteries; surveys): assign clear ownership before expanding. [22].
- AR fragility (lighting/texture): keep AR optional with fast fallback, test per device class, use Streetscape Geometry where available. [74].
- Fragmentation (languages/RTL): automate string linting, add snapshot tests for mirrored layouts. [68], [69]

10. References

[1] P. Arthur and R. Passini, *Wayfinding: People, Signs, and Architecture*. New York, NY, USA: McGraw-Hill, 1992. https://trid.trb.org/View/367500

[2] C. Prandi, G. Delnevo, P. Salomoni, and S. Mirri, “On supporting university communities in indoor wayfinding: An inclusive design approach,” *Sensors*, vol. 21, no. 9, p. 3134, Apr. 2021. MDPIPubMed

[3] R. Tahir, J. Krogstie, “Impact of navigation aid and spatial ability skills on wayfinding performance and workload in indoor-outdoor campus navigation: Challenges and design,” *Applied Sciences*, vol. 13, no. 17, p. 9508, Aug. 2023. MDPI

[4] Ofcom, *Adults’ Media Use and Attitudes Report 2024*. London, U.K.: Ofcom, Apr. 2024. [Online]. Available: https://www.ofcom.org.uk/…/adults-media-use-and-attitudes-report-2024.pdf

[5] Ofcom, *Online Nation 2024*. London, U.K.: Ofcom, Nov. 2024. [Online]. Available: https://www.ofcom.org.uk/…/online-nation-2024-report.pdf www.ofcom.org.uk

[6] A. K. Dey, “Understanding and using context,” *Personal and Ubiquitous Computing*, vol. 5, no. 1, pp. 4–7, 2001, doi: 10.1007/s007790170019. SpringerLink

[7] Google, “Indoor Maps – About,” Google Maps Partners. https://www.google.com/maps/about/partners/indoormaps/

[8] Google, “Use indoor maps to view floor plans,” Google Maps Help. Google Help

[9] W3C, *Web Content Accessibility Guidelines (WCAG) 2.2*. W3C Recommendation, Dec. 2024. [Online]. Available: https://www.w3.org/TR/WCAG22/

[10] Google, “Get started on Android with TalkBack,” *Android Accessibility Help*, [Online]. Available: https://support.google.com/accessibility/android/answer/6283677 Google Help

[11] Apple, “Use VoiceOver gestures on iPhone,” *iPhone User Guide*, [Online]. Available: https://support.apple.com/guide/iphone/use-voiceover-gestures Apple Support

[12] F. Zafari, A. Gkelias, and K. K. Leung, “A survey of indoor localization systems and technologies,” *IEEE Communications Surveys and Tutorials*, vol. 21, no. 3, pp. 2568–2599, 2019, doi: 10.1109/COMST.2019.2911558. (See also preprint: arXiv:1709.01015). SCIRParXiv

[13] ISO, *ISO 9241-210:2019 Ergonomics of human-system interaction — Part 210: Human-centred design for interactive systems*, 2019. [Online]. Available: https://www.iso.org/standard/77520.html ISO

[14] ISO/IEC 25010:2011, “Systems and software engineering—Systems and software Quality Requirements and Evaluation (SQuaRE)—System and software quality models,” 2011. [Online]. Available: iso.org/standard/35733.html (see preview PDF). ISOIteh Standards

[15] J. Nielsen, *Usability Engineering*. San Diego, CA, USA: Morgan Kaufmann, 1994. ACM Digital Library

[16] Google, “Accessibility overview — Material Design 3,” [Online]. Available: m3.material.io/foundations/overview/principles Material Design

[17] Android Developers, “Get the last known location (Fused Location Provider),” [Online]. Available: developer.android.com/develop/sensors-and-location/location/retrieve-current Android Developers

[18] B. Desruisseaux, “Internet Calendaring and Scheduling Core Object Specification (iCalendar),” *RFC 5545*, Sept. 2009. IETF Datatracker

[19] ISO/IEC/IEEE 29119-1:2022, “Software and systems engineering—Software testing—Part 1: General concepts,” 2022. ISO

[20] Firebase, “Firebase Cloud Messaging,” [Online]. Available: firebase.google.com/docs/cloud-messaging Firebase

[21] R. Mautz, *Indoor Positioning Technologies*, Habilitation Thesis, ETH Zürich, 2012. [Online]. Available: ETH Research Collection. research-collection.ethz.ch

[22] R. Faragher and A. Harle, “Location fingerprinting with Bluetooth Low Energy beacons,” *IEEE J. Sel. Areas Commun.*, vol. 33, no. 11, pp. 2418–2428, 2015. doi: 10.1109/JSAC.2015.2430281. ACM Digital Library

[23] J. Brooke, “SUS: A quick and dirty usability scale,” *J. Usability Stud.*, vol. 8, no. 2, pp. 29–40, 2013. [Online]. Available: uxpajournal.org. JUX - The Journal of User Experience

[24] OGC, “Indoor Mapping Data Format (IMDF) 1.0.0,” 2020. [Online]. Available: docs.ogc.org/cs/20-094. OGC Public Document Repository

[25] OpenStreetMap Wiki, “Simple Indoor Tagging,” 2014–2025. [Online]. Available: wiki.openstreetmap.org/wiki/Simple_Indoor_Tagging. OpenStreetMap

[26] OpenStreetMap Wiki, “Key:wheelchair,” Jun. 2025. [Online]. Available: wiki.openstreetmap.org/wiki/Key:wheelchair. OpenStreetMap

[27] K. Merry and P. Bettinger, “Smartphone GPS accuracy study in an urban environment,” *PLOS ONE*, vol. 14, no. 7, e0219890, 2019. PMC

[28] Apple Developer Documentation, “Displaying an Indoor Map (MapKit and IMDF),” [Online]. Available: developer.apple.com/documentation/mapkit/displaying-an-indoor-map. Apple Developer

[29] Google Maps, “Indoor Maps – About/Partners,” [Online]. Available: google.com/maps/about/partners/indoormaps/.

[30] Flutter Docs, “Docs | Flutter,”

[31] Google, “Before you begin — Google Maps for Flutter,”

[32] Flutter team, “google_maps_flutter — Flutter package,”

[33] Google Maps Platform, “Directions API,”

[34] Android Developers, “Fused Location Provider — get last known location,”

[35] Firebase, “Authentication — product docs,”

[36] Firebase, “Cloud Firestore data model,”

[37] Firebase, “Cloud Firestore Security Rules — get started,”

[38] Firebase, “Manage indexes in Cloud Firestore,”

[39] Firebase, “Call functions from your app (HTTPS callable),”

[40] Firebase, “Topic messaging on Android — FCM,”

[41] FlutterFire, “Cloud Messaging for Flutter — usage,”

[42] Pub.dev, “flutter_local_notifications — Flutter package,”

[43] OpenWeatherMap, “Current Weather Data API,”

[44] Material Design 3, “Accessibility overview and principles,”

[45] Firebase, “Crashlytics — monitor stability,”

[46] Firebase, “Cloud Firestore — performance best practices,”

[47] Android Developers, “Background location limits and power management,”

[48] Flutter Docs, “Testing and debugging overview,” 2024. Flutter Docs

[49] Flutter Docs, “Integration testing concepts,” 2024. Flutter Docs

[50] Flutter Docs, “Check app functionality with an integration test,” 2025. Flutter Docs

[51] Firebase, “Firebase Test Lab — overview,” 2025. Firebase

[52] Firebase, “Get started testing for Android with Test Lab,” 2025. Firebase

[53] ISO/IEC/IEEE 29119-2:2021, “Software testing — Test processes,” 2021. ISOIEEE Standards Association

[54] Flutter Docs, “Measure performance with an integration test,” 2025. Flutter Docs

[55] Android Developers, “Android vitals — app quality,” 2025. Android Developers

[56] Google Play Help, “Monitor your app’s technical quality with Android vitals,” 2025. Google Help

[57] Android Developers, “Get the last known location / request updates (Fused Location Provider),” 2025. Android Developers+1

[58] Firebase, “Crashlytics — product docs,” 2025. Firebase

[59] Firebase, “Crash-free users metric,” 2025. Firebase

[60] Android Developers, “ANR metrics in Android vitals,” 2024. Android Developers

[61] Android Studio, “App Quality Insights (Crashlytics + vitals),” 2025. Android Developers

[62] Material Design 3, “Assistive technology and focus order,” 2025. Material Design

[63] Material Design 3, “Color contrast,” 2025. Material Design

[64] Material Design 3, “Chips: accessibility,” 2025.

[65] Google, “Build global-scale, immersive, location-based AR experiences with the ARCore Geospatial API,” 2025. Google for Developers

[66] Apple Developer, “ARGeoAnchor — geographic anchors,” 2025. Apple Developer

[67] Google, “Use Geospatial anchors (WGS84) to position real-world content,” 2025. Google for Developers

[68] Flutter Docs, “Internationalizing Flutter apps (flutter_localizations, intl),” 2025. Flutter Docs

[69] Material Design (M3), “Bidirectionality and RTL,” 2025. Material Design

[70] Firebase, “Remote Config — feature flags and staged rollouts,” 2025. Firebase

[71] Android Developers, “Wi-Fi location: ranging with RTT (802.11mc/FTM),” 2025. Android Developers

[72] AOSP, “Wi-Fi RTT (IEEE 802.11mc / 802.11az in Android 15),” 2025. Android Open Source Project

[73] FiRa Consortium, “UWB Secure Ranging,” Whitepaper, 2022. firaconsortium.org

[74] Google, “ARCore Geospatial — Streetscape Geometry,” 2025. Google for Developers

[75] Apple, “ARGeoAnchor — geographic anchors,” 2025. Apple Developer

[76] W3C, “WAI-ARIA Authoring Practices 1.2,” 2021. W3C

[77] ANSI/ISO, “ISO 21542:2021 — Accessible building construction (overview),” 2023. The ANSI Blog

[78] Firebase, “A/B Testing with Remote Config,” 2025. Firebase

[79] Google Maps Platform, “Map Tiles API — caching and storage policy,” 2025. Google for Developers

[80] **K. Burchill**, “speech_to_text — Flutter plugin,” *pub.dev*, 2025. Dart packages

[81] **Android Developers**, “SpeechRecognizer — API Reference,” 2025. Android Developers

[82] **Apple Developer**, “SFSpeechRecognizer — Speech framework,” 2025. Apple Developer

[83] **Apple Developer**, “NSSpeechRecognitionUsageDescription — Info.plist key,” 2025.. Apple Developer

[84] **Android Developers**, “RECORD_AUDIO is a dangerous permission (runtime) — Media/Permissions overview,”. Android Developers

[85] **Apple Developer**, “NSMicrophoneUsageDescription — Info.plist key,” 2025. Accessed Sept. 5, 2025.

11. Appendices

Interview Questions:

- When you first opened the app, **what did you think it does**? (What stood out / what was unclear?)

- **Find a building:** Walk me through how you'd locate a destination from the Home screen. What would you expect to happen next?

- On the Map, how clear is the **single red "searched" marker** and the **distance/time chip**? What would you change?

- Is the **Clear Marker/Route** action discoverable and named clearly? If not, what label would you prefer?

- Try **saving a route** you'll want to reuse. Where would you expect to find it later?

- Open **Saved Routes**. Is the ordering, rename/delete behaviour, and feedback clear?

- How useful is the **Step free** preference to you? Where would you expect to toggle it, and what would you expect it to change?

- Do the **touch targets** (buttons, lists) feel comfortable to tap? Any issues with **contrast** or text size?

- How well does **voice search** fit your usage? Would you use it on the move?

- Imagine you receive a campus alert (room change/closure). How do you **notice** it? Is the **unread badge** and **Inbox** concept clear?

- After reading an alert, would you prefer **auto mark-as-read** or keeping the explicit action? Why?

- Did anything **surprise or confuse** you? Where did you hesitate?

- What's **missing** that would make this your go-to wayfinding app on campus?

- If you rely on accessibility features (e.g., TalkBack), what changes would improve the experience?

- Overall, **how confident** would you feel using this between classes, and why?

System architecture

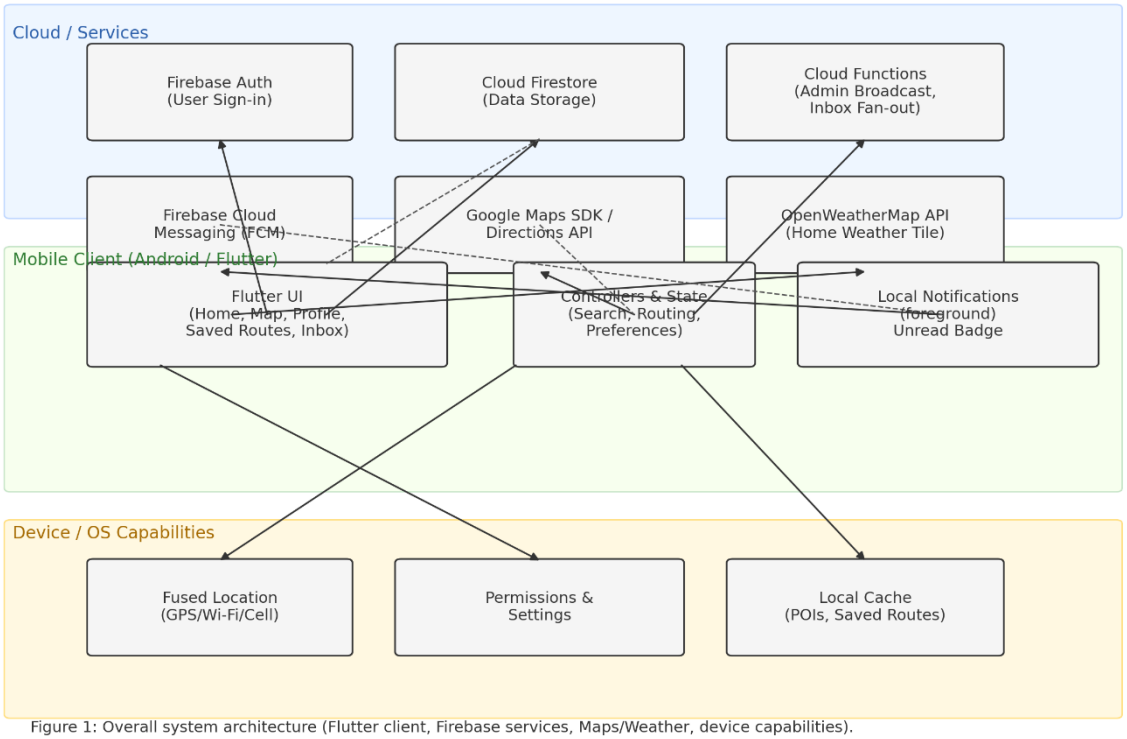


Figure 1: Overall system architecture (Flutter client, Firebase services, Maps/Weather, device capabilities).

Firebase schema

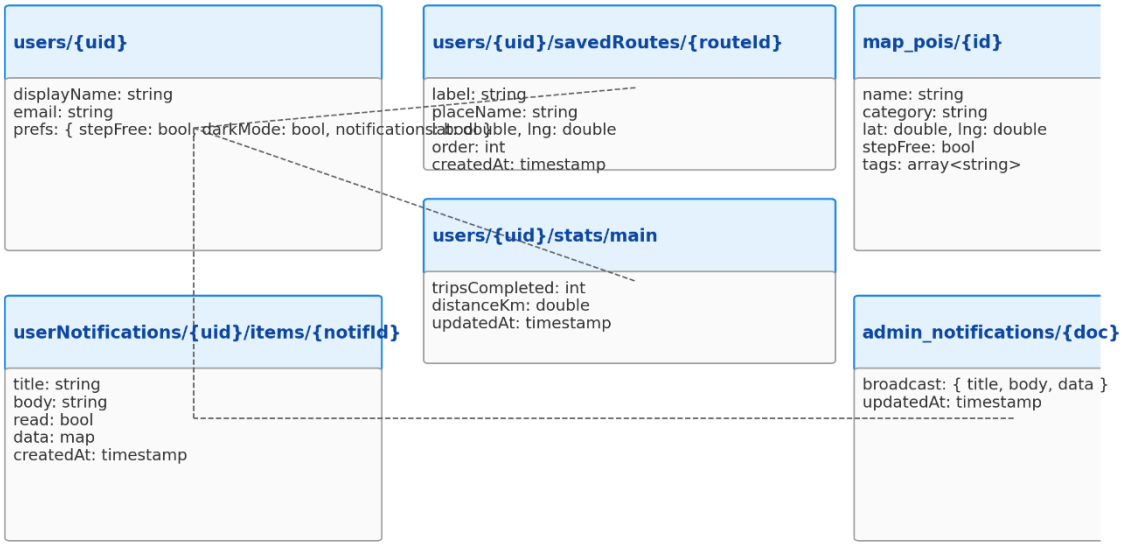
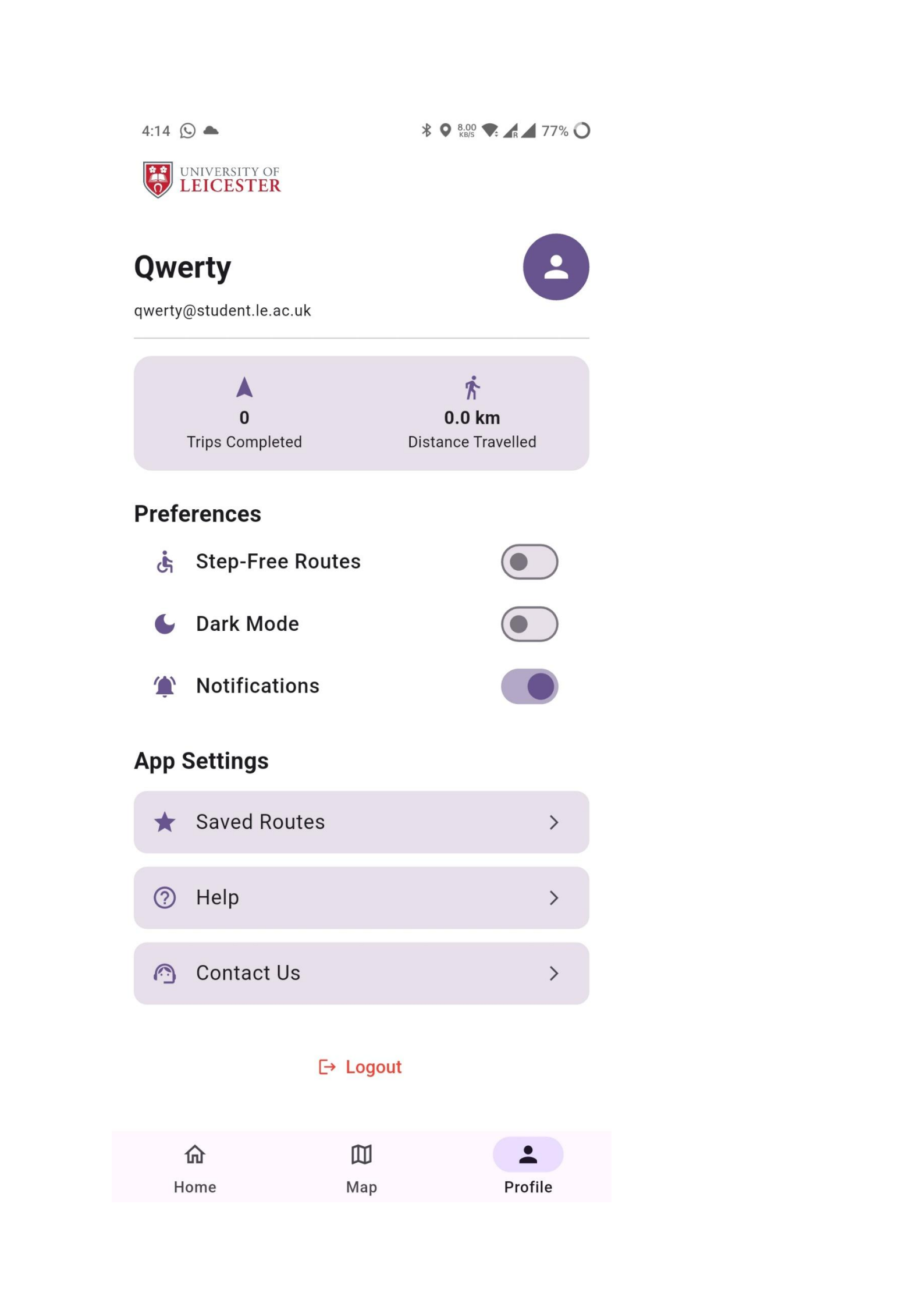
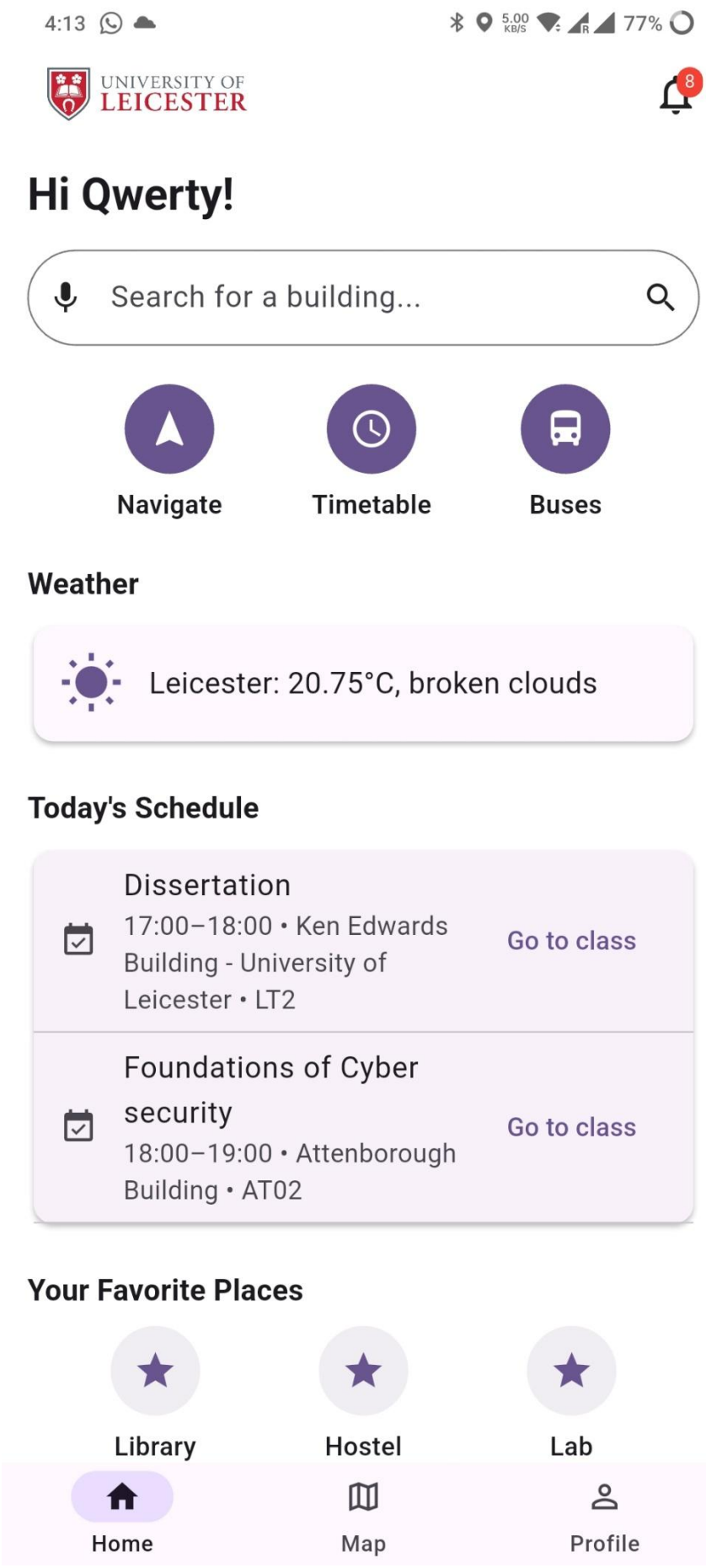
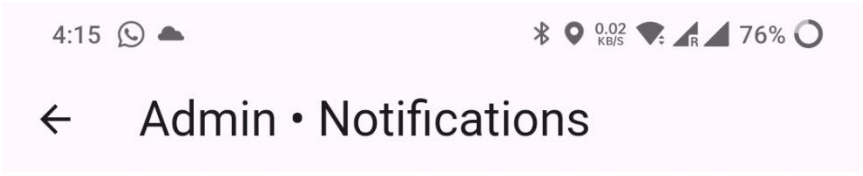


Figure 2: Firestore data schema with key collections, subcollections, and fields.


4:14
77%
UNIVERSITY OF LEICESTER
Qwerty
qwerty@student.le.ac.uk
0
Trips Completed
0.0 km
Distance Travelled
Preferences
Step-Free Routes
Dark Mode
Notifications
App Settings
Saved Routes
Help
Contact Us
Logout
Home
Map
Profile


4:13
5.00 KB/S
77%
UNIVERSITY OF LEICESTER
8
Hi Qwerty!
Search for a building...
Navigate
Timetable
Buses
Weather
Leicester: 20.75°C, broken clouds
Today's Schedule
Dissertation
17:00–18:00 • Ken Edwards Building - University of Leicester • LT2
Go to class
Foundations of Cyber security
18:00–19:00 • Attenborough Building • AT02
Go to class
Your Favorite Places
Library
Hostel
Lab
Home
Map
Profile



Title *

Body *

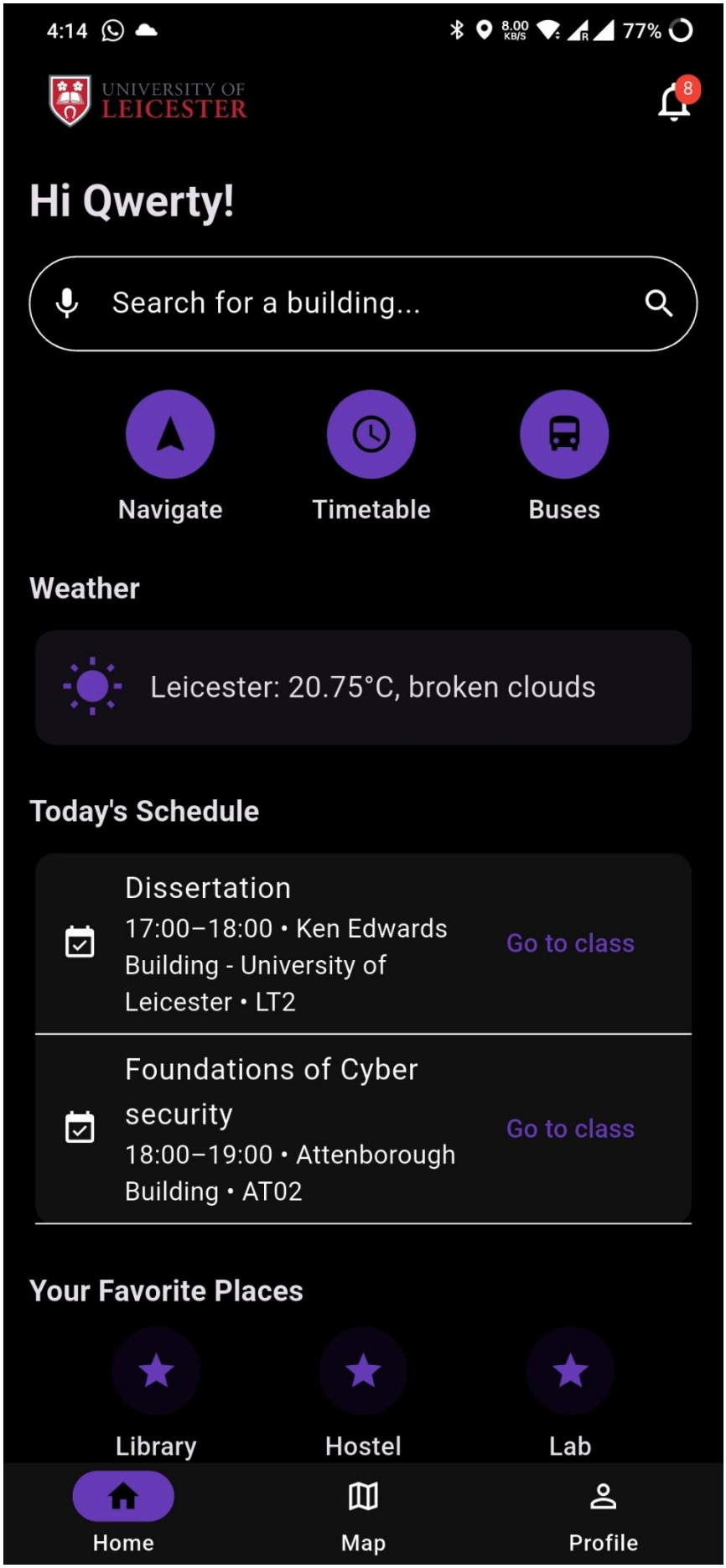
Image URL (optional)

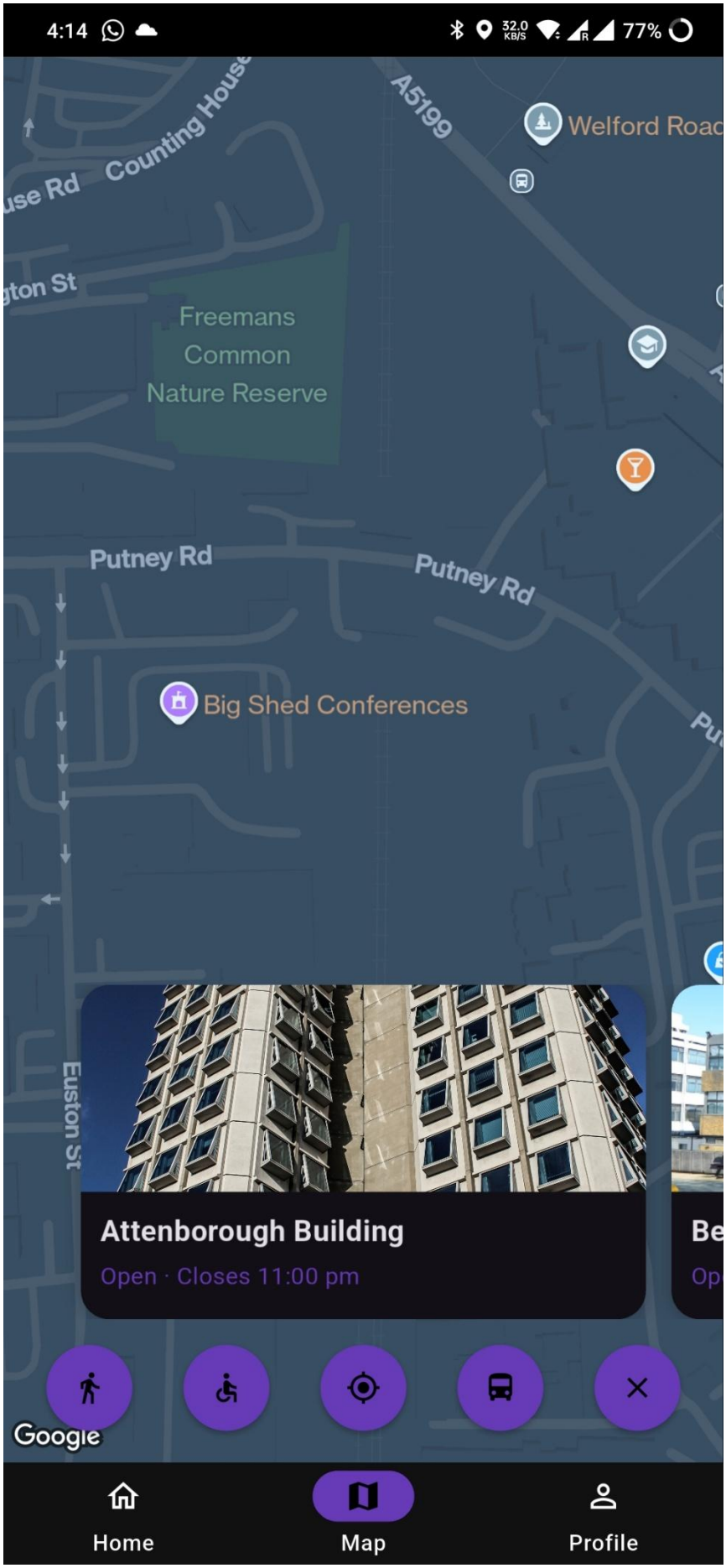
Deeplink (optional, e.g. app://timetables)

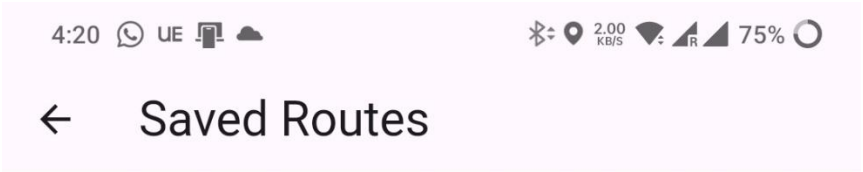
Publish

Recent publishes

Hii Testing	04/09 21:26
account in accountancy gfdghh	04/09 18:19
fhjv cdjk	04/09 16:20
Testing good morning	04/09 16:19
Test hii	04/09 16:16
Test test check	04/09 13:25


4:14
77%
UNIVERSITY OF LEICESTER
8
Hi Qwerty!
Search for a building...
Navigate
Timetable
Buses
Weather
Leicester: 20.75°C, broken clouds
Today's Schedule
Dissertation
17:00–18:00 • Ken Edwards Building - University of Leicester • LT2
Go to class
Foundations of Cyber security
18:00–19:00 • Attenborough Building • AT02
Go to class
Your Favorite Places
Library
Hostel
Lab
Home
Map
Profile


4:14
77%
Counting House
A5199
Welford Road
Freemans
Common
Nature Reserve
Putney Rd
Putney Rd
Big Shed Conferences
Euston St
Attenborough Building
Open · Closes 11:00 pm
Google
Home
Map
Profile



Slot 1

Custom name (e.g. “My Library”)

Library

David Wilson Library, University Road, Leicester, UK
(52.6204371, -1.1248732)

Pick place

Save Remove

Slot 2

Custom name (e.g. “My Library”)

Hostel

Nixon Court, Putney Road, Leicester, UK
(52.616394, -1.1294429)

Pick place

Save Remove

Slot 3

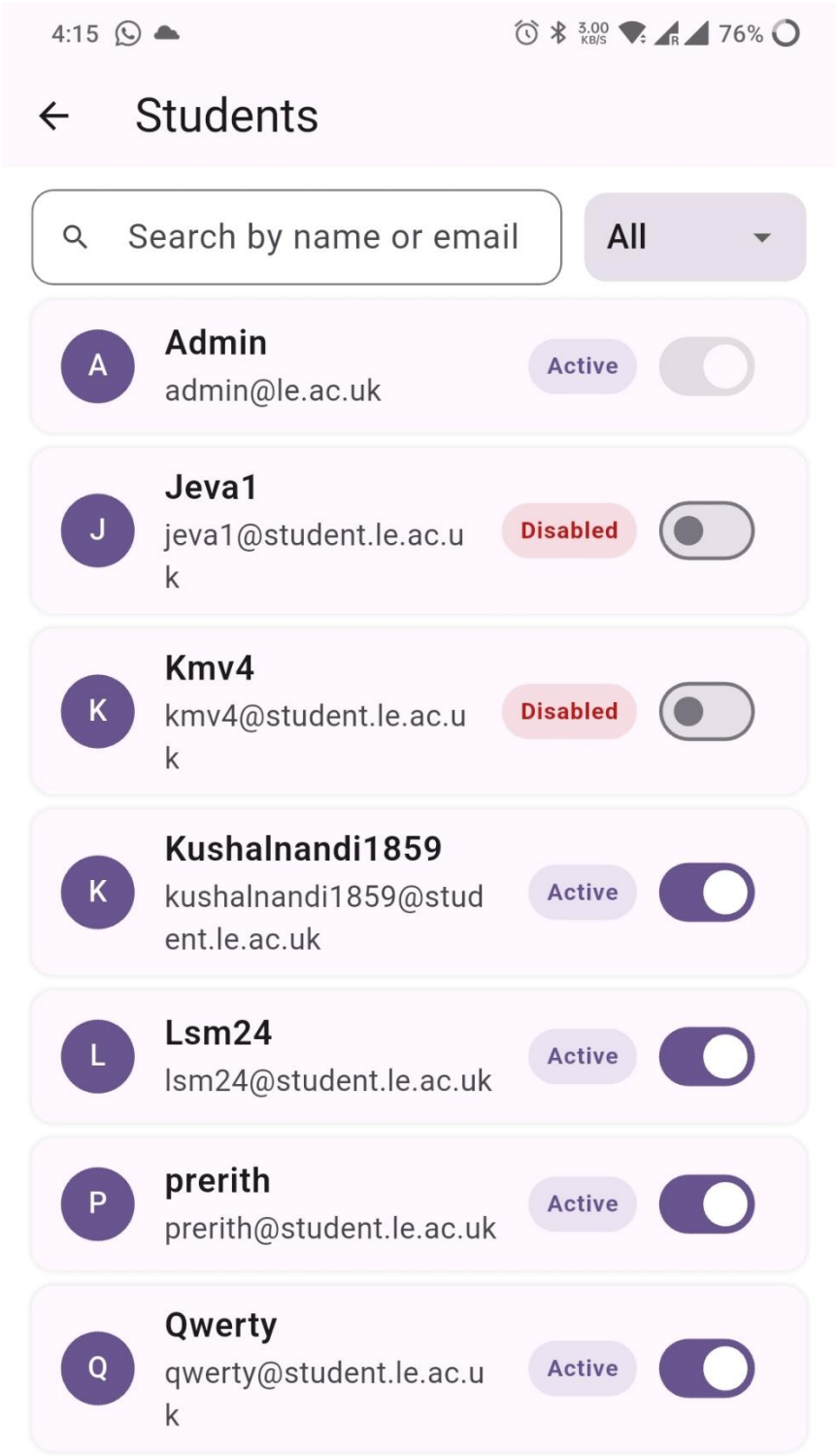
Custom name (e.g. “My Library”)

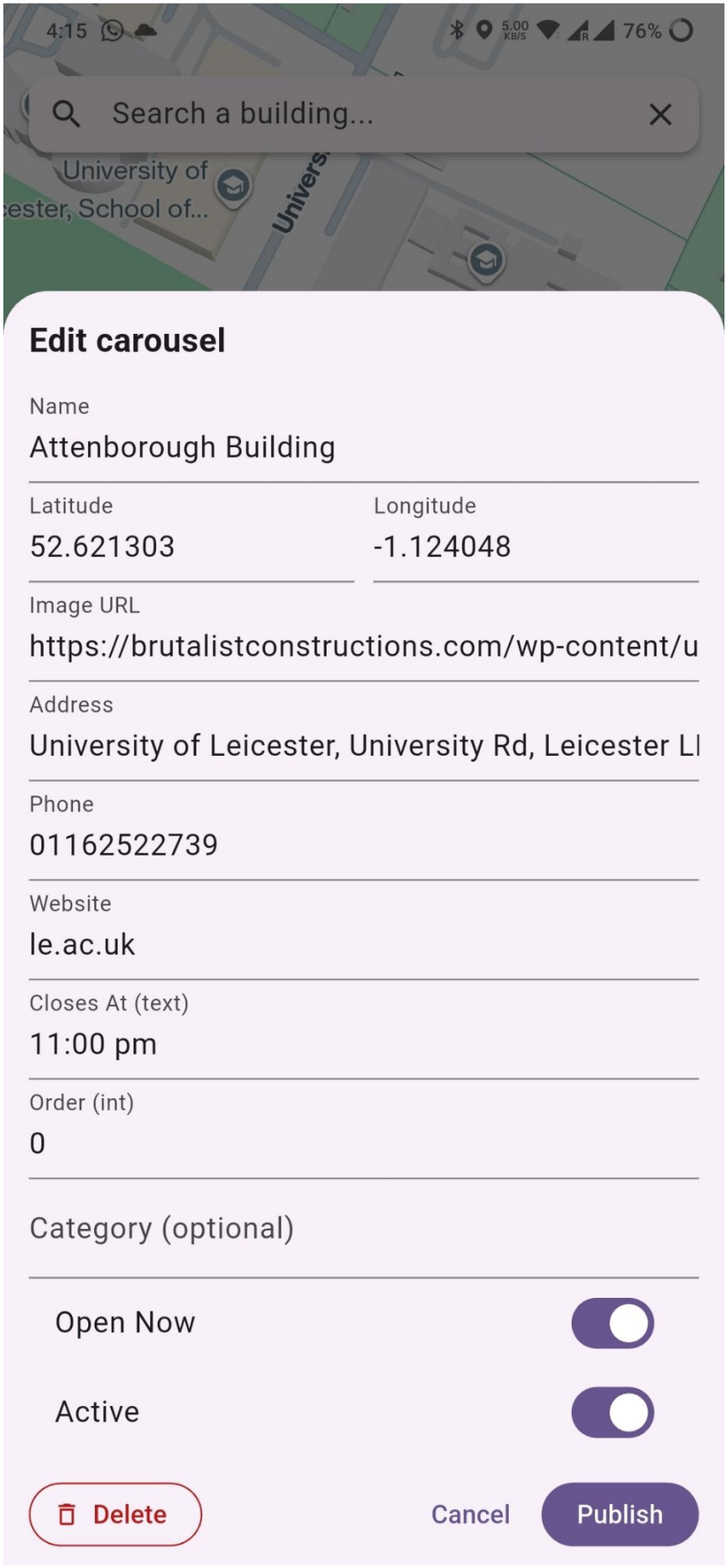
Lab

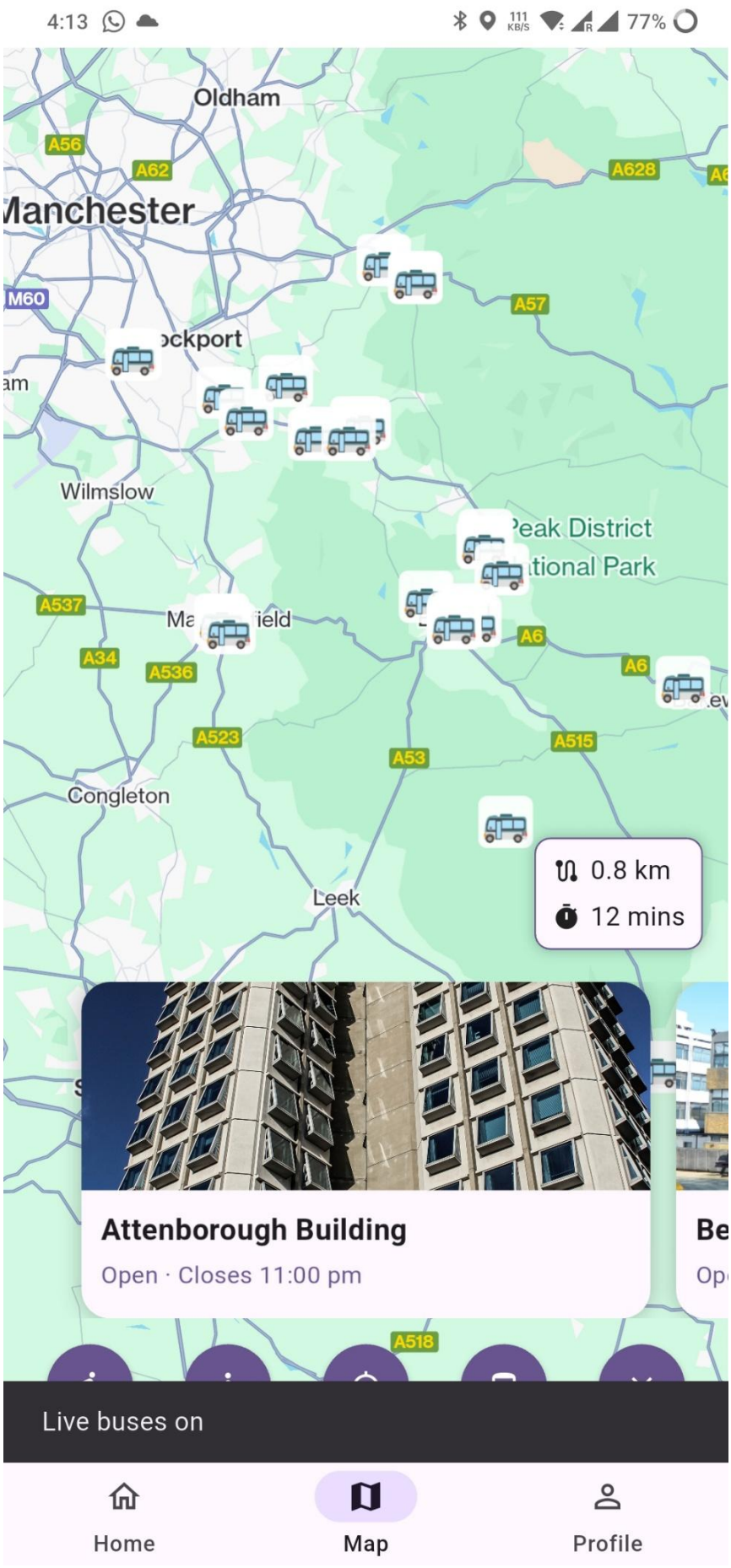
Percy Gee Building - University of Leicester, Mayor's Walk, Leicester, UK
(52.6217265, -1.1252734)

Pick place

Save Remove


4:15
3.00 KB/S
76%
Students
Search by name or email
All
A
Admin
admin@le.ac.uk
Active
J
Jeva1
jeva1@student.le.ac.uk
Disabled
K
Kmv4
kmv4@student.le.ac.uk
Disabled
K
Kushalnandi1859
kushalnandi1859@student.le.ac.uk
Active
L
Lsm24
lsm24@student.le.ac.uk
Active
P
prerith
prerith@student.le.ac.uk
Active
Q
Qwerty
qwerty@student.le.ac.uk
Active


4:15
76%
Search a building...
University of
ester, School of...
Univers
Edit carousel
Name
Attenborough Building
Latitude
52.621303
Longitude
-1.124048
Image URL
https://brutalistconstructions.com/wp-content/u
Address
University of Leicester, University Rd, Leicester Ll
Phone
01162522739
Website
le.ac.uk
Closes At (text)
11:00 pm
Order (int)
0
Category (optional)
Open Now
Active
Delete
Cancel
Publish


4:13
77%
Oldham
A56
A62
A628
Manchester
M60
A57
Wilmslow
A537
A34
A536
A6
A523
A53
A515
Congleton
Leek
0.8 km
12 mins
Attenborough Building
Open · Closes 11:00 pm
A518
Live buses on
Home
Map
Profile

Admin Dashboard

Timetables Students Map

Manage

Timetables

Create, edit, and publish timetable bundles.

Students

Manage student accounts and access.

Map

Campus map (view only).

Notifications

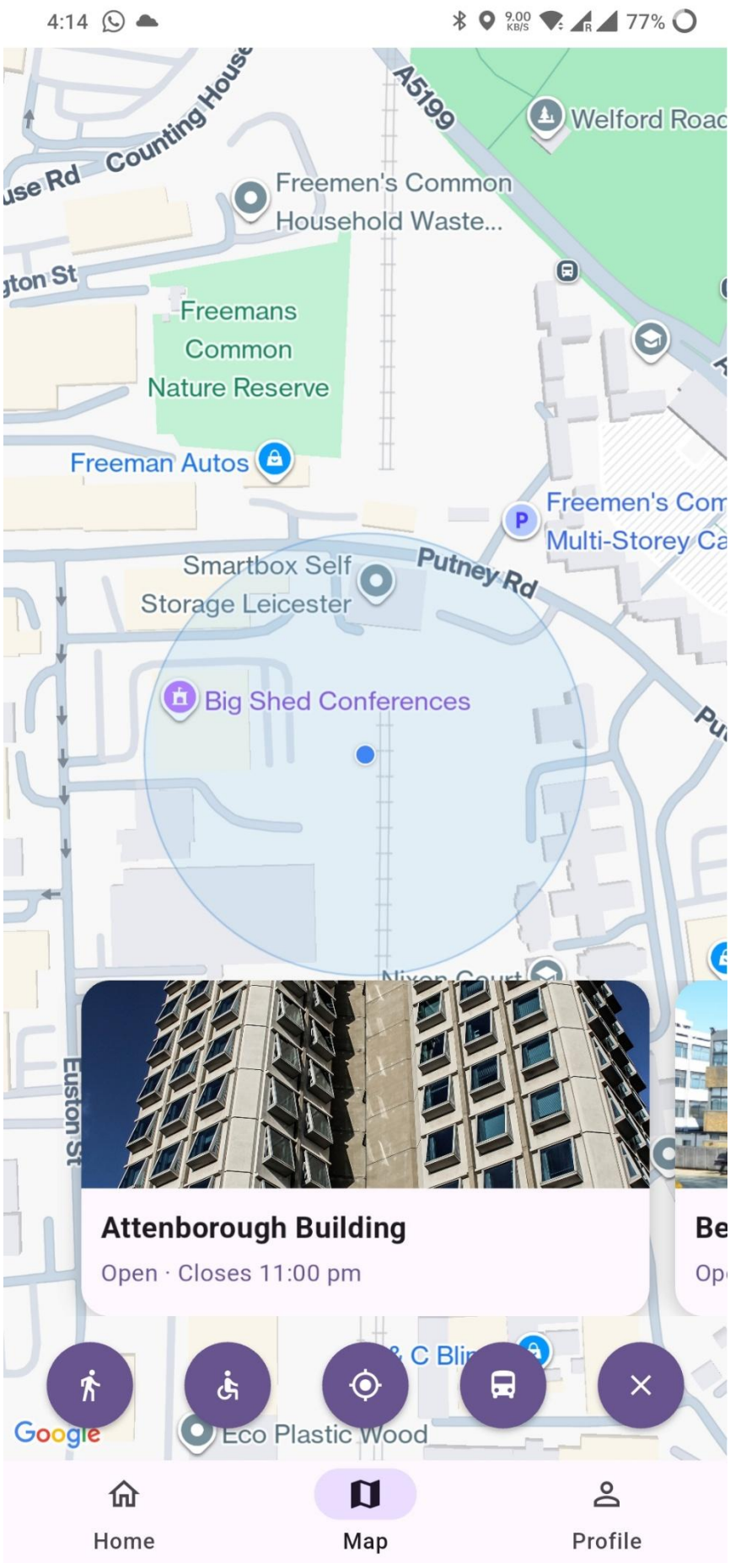
Send announcements to students.

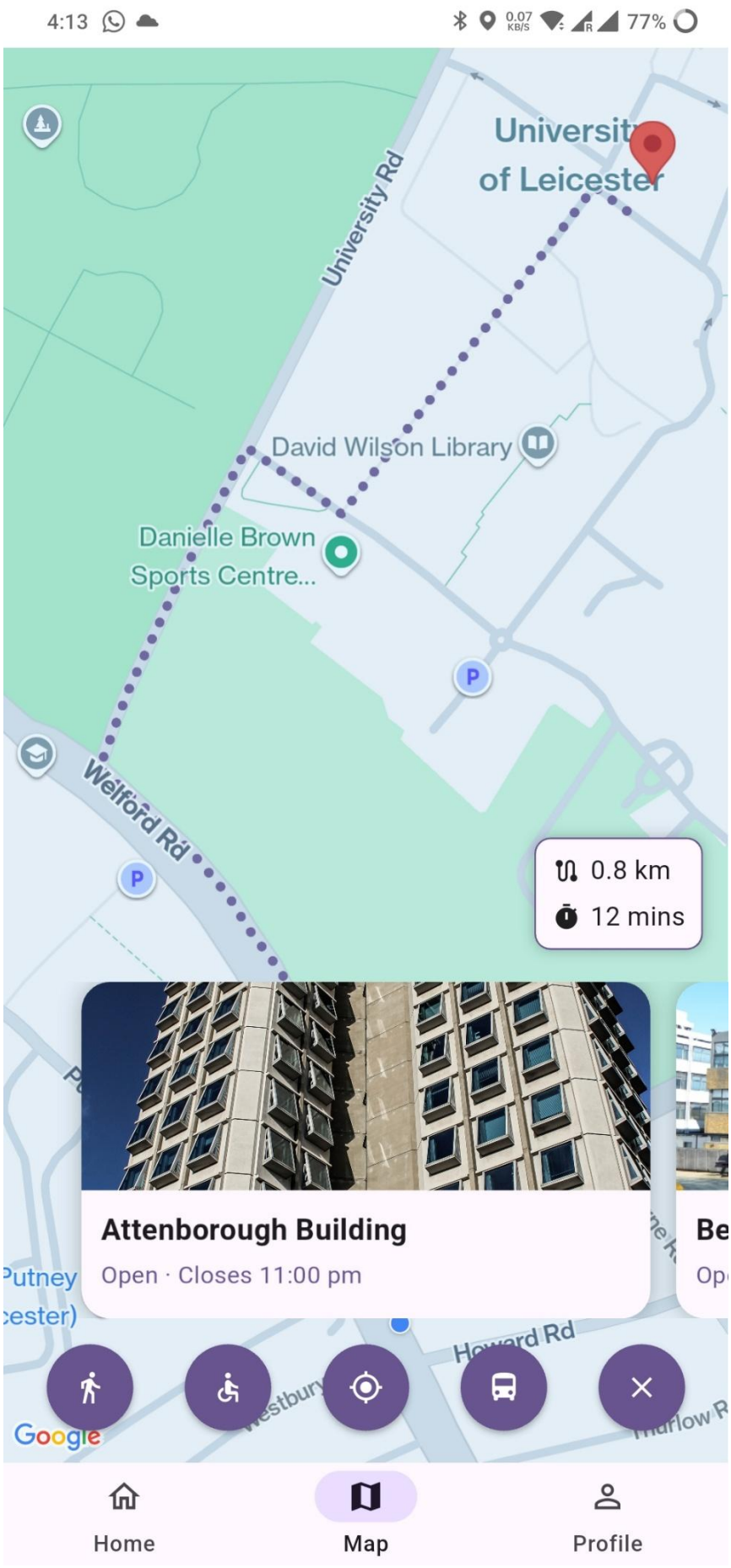
Status Overview

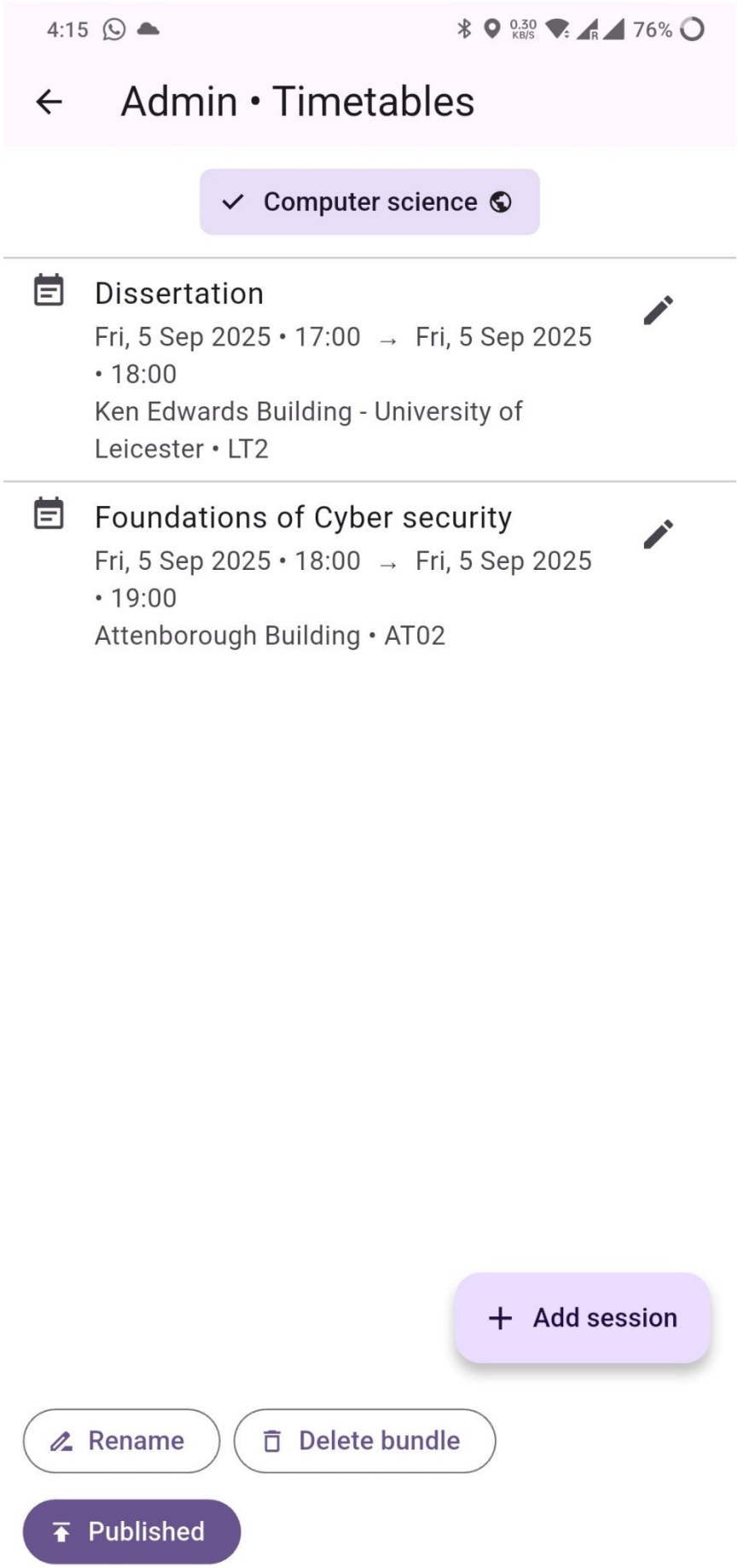
Draft Bundles 0

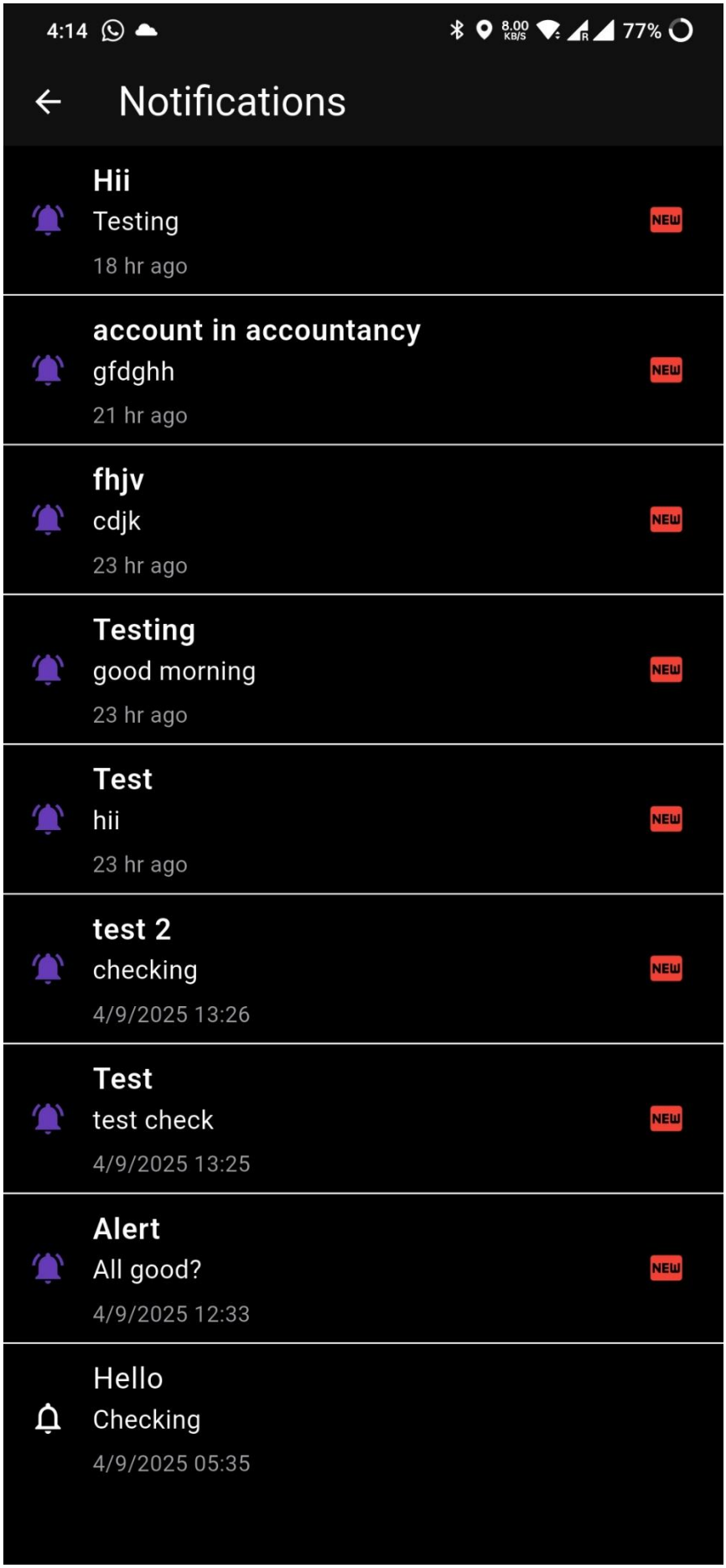
Disabled Accounts 2

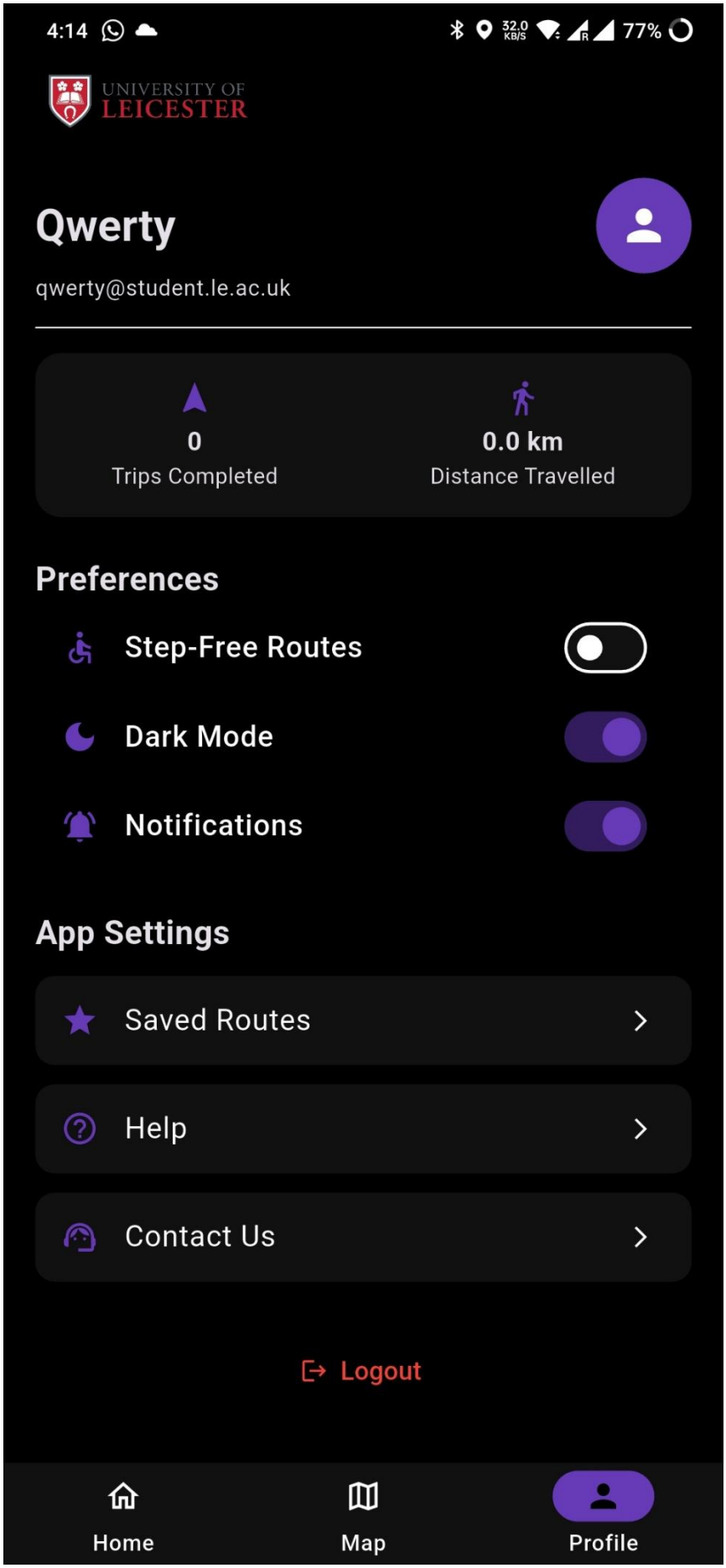
Logout


4:14
77%
A5199
Welford Road
Freemen's Common
Household Waste...
Freemans
Common
Nature Reserve
Freeman Autos
Freemen's Com
Multi-Storey Ca
Smartbox Self
Storage Leicester
Putney Rd
Big Shed Conferences
Euston St
Attenborough Building
Open · Closes 11:00 pm
Google
Eco Plastic Wood
Home
Map
Profile


4:13
77%
University of Leicester
University Rd
David Wilson Library
Danielle Brown Sports Centre...
Welford Rd
0.8 km
12 mins
Attenborough Building
Open · Closes 11:00 pm
Howard Rd
Google
Home
Map
Profile


4:15
76%
Admin • Timetables
Computer science
Dissertation
Fri, 5 Sep 2025 • 17:00 → Fri, 5 Sep 2025 • 18:00
Ken Edwards Building - University of Leicester • LT2
Foundations of Cyber security
Fri, 5 Sep 2025 • 18:00 → Fri, 5 Sep 2025 • 19:00
Attenborough Building • AT02
Add session
Rename
Delete bundle
Published


4:14
8.00 KB/S
77%
Notifications
Hii
Testing
NEW
18 hr ago
account in accountancy
gfdghh
NEW
21 hr ago
fhjv
cdjk
NEW
23 hr ago
Testing
good morning
NEW
23 hr ago
Test
hii
NEW
23 hr ago
test 2
checking
NEW
4/9/2025 13:26
Test
test check
NEW
4/9/2025 13:25
Alert
All good?
NEW
4/9/2025 12:33
Hello
Checking
4/9/2025 05:35


4:14
77%
UNIVERSITY OF LEICESTER
Qwerty
qwerty@student.le.ac.uk
0
Trips Completed
0.0 km
Distance Travelled
Preferences
Step-Free Routes
Dark Mode
Notifications
App Settings
Saved Routes
Help
Contact Us
Logout
Home
Map
Profile

Welcome!

Student | Admin

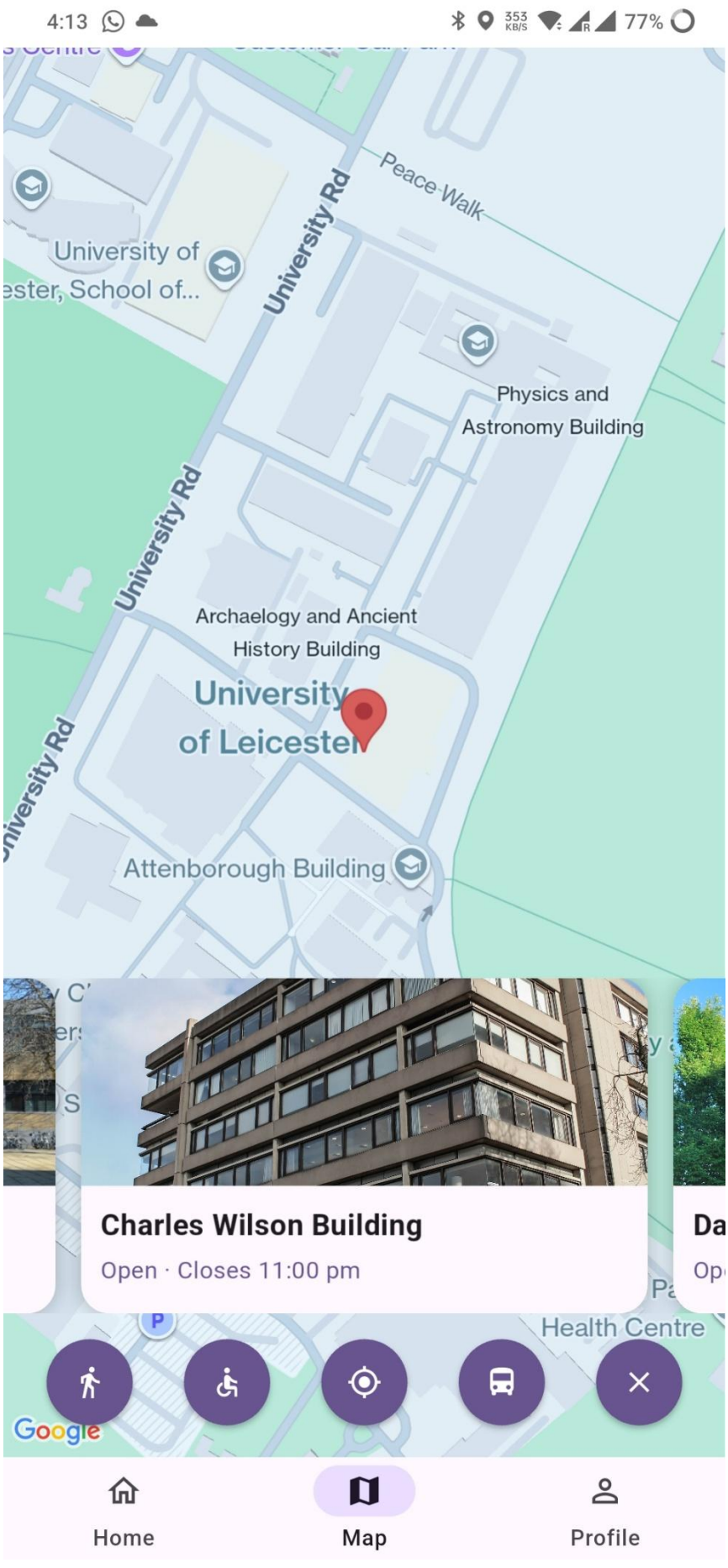
University Email

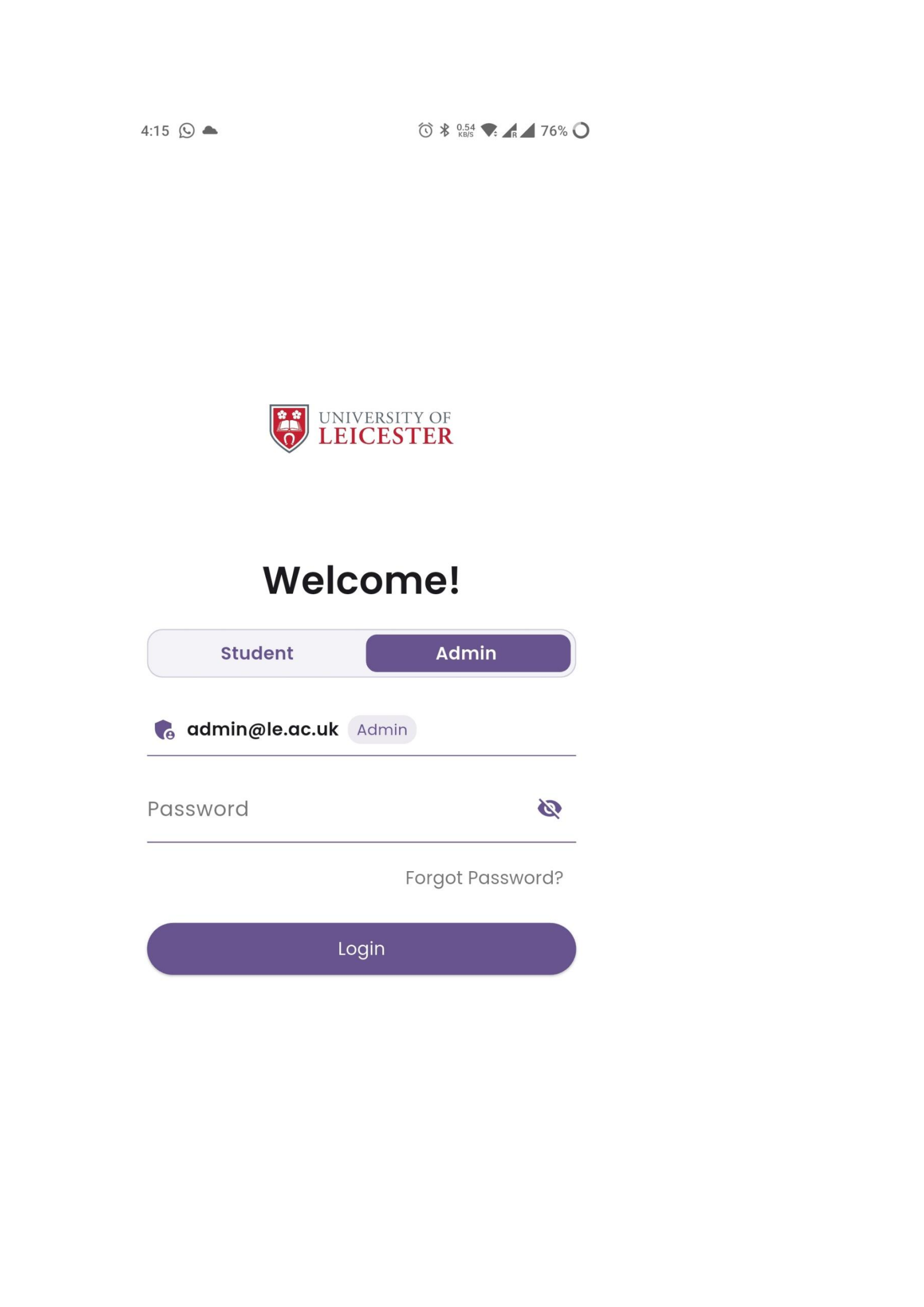
Password

Forgot Password?

Login

Don't have an account? Register here


4:13
77%
University of
ester, School of...
University Rd
Peace Walk
Physics and
Astronomy Building
University Rd
Archaeology and Ancient
History Building
University
of Leicester
University Rd
Attenborough Building
Charles Wilson Building
Open · Closes 11:00 pm
Health Centre
Google
Home
Map
Profile


4:15
0.54 KB/S
76%
UNIVERSITY OF LEICESTER
Welcome!
Student
Admin
admin@le.ac.uk
Admin
Password
Forgot Password?
Login


4:14
77%
Timetable
Dissertation
Fri, 5 Sep 2025 • 17:00 → Fri, 5 Sep 2025 • 18:00
Ken Edwards Building - University of Leicester • LT2
Go to class
Foundations of Cyber security
Fri, 5 Sep 2025 • 18:00 → Fri, 5 Sep 2025 • 19:00
Attenborough Building • AT02
Go to class

Admin_notifications.dart

```dart
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:flutter/material.dart';

/// Screen to compose and publish admin notifications, with a feed of recent sends.
class AdminNotificationsScreen extends StatefulWidget {
  const AdminNotificationsScreen({super.key});

  @override
  State<AdminNotificationsScreen> createState() => _AdminNotificationsScreenState();
}

/// Holds form controllers, handles publish flow, and streams recent publishes.
class _AdminNotificationsScreenState extends State<AdminNotificationsScreen> {
  final _title = TextEditingController();
  final _body = TextEditingController();
  final _image = TextEditingController();
  final _deeplink = TextEditingController();

  final _formKey = GlobalKey<FormState>();
  bool _sending = false;

  CollectionReference<Map<String, dynamic>> get _col =>
      FirebaseFirestore.instance.collection('admin_notifications');

  /// Dispose text controllers to prevent memory leaks.
  @override
  void dispose() {
    _title.dispose();
    _body.dispose();
    _image.dispose();
    _deeplink.dispose();
    super.dispose();
  }

  /// Validate form, write a notification document, and show user feedback.
  Future<void> _publish() async {
    if (!(_formKey.currentState?.validate() ?? false)) return;
    FocusScope.of(context).unfocus();

    setState(() => _sending = true);
    try {
      final me = FirebaseAuth.instance.currentUser;
      await _col.add({
        'title': _title.text.trim(),
        'body': _body.text.trim(),
        'image': _image.text.trim().isEmpty ? null : _image.text.trim(),
        'deeplink': _deeplink.text.trim().isEmpty ? null : _deeplink.text.trim(),
```

```
      'createdAt': FieldValue.serverTimestamp(),
      'publishedByUid': me?.uid,
      'publishedByEmail': me?.email,
    });

    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Published')),
    );
    _title.clear();
    _body.clear();
    _image.clear();
    _deeplink.clear();
  } catch (e) {
    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Publish failed: $e')),
    );
  } finally {
    if (mounted) setState(() => _sending = false);
  }
}

/// Build the form UI and a live list of recent publishes.
@override
Widget build(BuildContext context) {
  final theme = Theme.of(context);

  return Scaffold(
    appBar: AppBar(title: const Text('Admin • Notifications')),
    body: Padding(
      padding: const EdgeInsets.all(16),
      child: Column(
        children: [
          Form(
            key: _formKey,
            child: Column(
              children: [
                TextFormField(
                  controller: _title,
                  decoration: const InputDecoration(labelText: 'Title *'),
                  validator: (v) =>
                  (v == null || v.trim().isEmpty) ? 'Enter a title' : null,
                ),
                TextFormField(
                  controller: _body,
                  decoration: const InputDecoration(labelText: 'Body *'),
                  maxLines: 3,
                  validator: (v) =>
                  (v == null || v.trim().isEmpty) ? 'Enter a message' : null,
```

```
      ),
      TextFormField(
        controller: _image,
        decoration:
        const InputDecoration(labelText: 'Image URL (optional)'),
      ),
      TextFormField(
        controller: _deeplink,
        decoration: const InputDecoration(
          labelText: 'Deeplink (optional, e.g. app://timetables)',
        ),
      ),
      const SizedBox(height: 12),
      SizedBox(
        width: double.infinity,
        child: FilledButton.icon(
          icon: _sending
              ? const SizedBox(
            width: 18, height: 18,
            child: CircularProgressIndicator(strokeWidth: 2, color: Colors.white),
          )
              : const Icon(Icons.send),
          label: Text(_sending ? 'Publishing…' : 'Publish'),
          onPressed: _sending ? null : _publish,
        ),
      ),
    ],
  ),
),
const SizedBox(height: 24),
const Align(
  alignment: Alignment.centerLeft,
  child: Text('Recent publishes',
      style: TextStyle(fontWeight: FontWeight.bold)),
),
const SizedBox(height: 8),
Expanded(
  child: StreamBuilder<QuerySnapshot<Map<String, dynamic>>>(
    stream: _col.orderBy('createdAt', descending: true).limit(20).snapshots(),
    builder: (context, snap) {
      if (snap.connectionState == ConnectionState.waiting) {
        return const Center(child: CircularProgressIndicator());
      }
      if (snap.hasError) {
        return Center(child: Text('Error: ${snap.error}'));
      }
      final docs = snap.data?.docs ?? const [];
      if (docs.isEmpty) {
        return const Center(child: Text('No notifications yet.'));
      }
```

```
            return ListView.separated(
              itemCount: docs.length,
              separatorBuilder: (_, __) => const Divider(height: 1),
              itemBuilder: (_, i) {
                final d = docs[i];
                final data = d.data();
                final ts = data['createdAt'] as Timestamp?;
                final when = ts?.toDate().toLocal();
                final subtitle = [
                  if (data['body'] != null) data['body'] as String,
                  if (data['deeplink'] != null) '🔗 ${data['deeplink']}',
                ].join('\n');

                return ListTile(
                  title: Text(data['title'] ?? ''),
                  subtitle: Text(subtitle),
                  trailing: when == null
                      ? null
                      : Text(
                    _fmtShortTime(when),
                    style: theme.textTheme.bodySmall,
                  ),
                  onLongPress: () async {
                    final ok = await showDialog<bool>(
                      context: context,
                      builder: (_) => AlertDialog(
                        title: const Text('Delete notification?'),
                        content: const Text('This will remove the admin record.'),
                        actions: [
                          TextButton(
                              onPressed: () => Navigator.pop(context, false),
                              child: const Text('Cancel')),
                          FilledButton(
                              onPressed: () => Navigator.pop(context, true),
                              child: const Text('Delete')),
                        ],
                      ),
                    ) ??
                        false;
                    if (ok) await d.reference.delete();
                  },
                );
              },
            );
          },
        ),
      ),
    ],
  ),
),
```

```
    );
  }
}

/// Format a DateTime as dd/MM HH:mm for compact list display.
String _fmtShortTime(DateTime dt) {
  final h = dt.hour.toString().padLeft(2, '0');
  final m = dt.minute.toString().padLeft(2, '0');
  final mo = dt.month.toString().padLeft(2, '0');
  final d = dt.day.toString().padLeft(2, '0');
  return '$d/$mo $h:$m';
}
```

Map_screen.dart

```
import 'dart:async';
import 'dart:convert';
import 'dart:io';
import 'dart:math';
import 'dart:ui' as ui;

import 'package:flutter/material.dart';
import 'package:google_maps_flutter/google_maps_flutter.dart';
import 'package:geolocator/geolocator.dart';
import 'package:http/http.dart' as http;
import 'package:xml/xml.dart' as xml;
import 'package:flip_card/flip_card.dart';

import 'package:campus_navigation/models/campus_poi.dart';
import 'package:campus_navigation/services/campus_poi_repository.dart';

/// Interactive campus map: search, flippable POI carousel, routes, live buses,
/// and admin editing when enabled.
class MapScreen extends StatefulWidget {
  final String searchQuery;
  final double? searchLat;
  final double? searchLng;
  final bool isAdmin;
  final bool autoRouteOnOpen;

  const MapScreen({
    super.key,
    this.searchQuery = '',
    this.searchLat,
    this.searchLng,
    this.isAdmin = false,
    this.autoRouteOnOpen = false,
  });
```

```
  @override
  State<MapScreen> createState() => _MapScreenState();
}

class _MapScreenState extends State<MapScreen> {
  final Completer<GoogleMapController> _controller = Completer();
  final PageController _pageCtrl = PageController(viewportFraction: 0.82);
  final TextEditingController _adminSearchCtrl = TextEditingController();

  static const String apiKey = "AIzaSyAsHYoxe5t5A8Zm8tPogYOfWFjAtyDionw";
  static const LatLng campusCenter = LatLng(52.6219, -1.1244);

  static const String bodsSiriUrl =
      "https://data.bus-
data.dft.gov.uk/api/v1/datafeed/18865/?api_key=1d4baf6fa7186850abd35eeff4b7f8af29a78fc
1";

  LatLng? _currentLocation;
  LatLng? _selectedLocation;
  CampusPoi? _selectedPoiForMarker;
  int _currentIndex = 0;

  Set<Marker> _markers = {};
  Set<Polyline> _polylines = {};
  bool _searchedVisible = false;

  RouteInfo? _routeInfo;
  bool _isWheelchairRoute = false;

  final List<CampusPoi> _pois = [];
  StreamSubscription<List<CampusPoi>>? _poiSub;

  final List<GlobalKey<FlipCardState>> _flipCtrls = [];

  // Live buses state
  bool _liveOn = false;
  bool _didAutoFocusBuses = false;
  Timer? _busPollTimer;
  final Map<String, LatLng> _busPrevPos = {};
  final Map<String, Timer> _busAnimTimers = {};
  BitmapDescriptor? _emojiIcon;

  // Map style (dark mode)
  Brightness? _lastBrightness;

  static const String _mapStyleDark = '''
[
  {"elementType":"geometry","stylers":[{"color":"#242f3e"}]},
  {"elementType":"labels.text.stroke","stylers":[{"color":"#242f3e"}]},
  {"elementType":"labels.text.fill","stylers":[{"color":"#746855"}]},
```

```
{"featureType":"administrative.locality","elementType":"labels.text.fill","stylers":[{"color":"
#d59563"}]},
  {"featureType":"poi","elementType":"labels.text.fill","stylers":[{"color":"#d59563"}]},
  {"featureType":"poi.park","elementType":"geometry","stylers":[{"color":"#263c3f"}]},

{"featureType":"poi.park","elementType":"labels.text.fill","stylers":[{"color":"#6b9a76"}]},
  {"featureType":"road","elementType":"geometry","stylers":[{"color":"#38414e"}]},
  {"featureType":"road","elementType":"geometry.stroke","stylers":[{"color":"#212a37"}]},
  {"featureType":"road","elementType":"labels.text.fill","stylers":[{"color":"#9ca5b3"}]},

{"featureType":"road.highway","elementType":"geometry","stylers":[{"color":"#746855"}]}
,

{"featureType":"road.highway","elementType":"geometry.stroke","stylers":[{"color":"#1f28
35"}]},

{"featureType":"road.highway","elementType":"labels.text.fill","stylers":[{"color":"#f3d19c"
}]},
  {"featureType":"transit","elementType":"geometry","stylers":[{"color":"#2f3948"}]},

{"featureType":"transit.station","elementType":"labels.text.fill","stylers":[{"color":"#d59563
"}]},
  {"featureType":"water","elementType":"geometry","stylers":[{"color":"#17263c"}]},
  {"featureType":"water","elementType":"labels.text.fill","stylers":[{"color":"#515c6d"}]},
  {"featureType":"water","elementType":"labels.text.stroke","stylers":[{"color":"#17263c"}]}
]
''';

  /// Kick off location, icons, and POI stream; optionally pre-route to a search.
  @override
  void initState() {
    super.initState();
    _initLocation();
    _loadEmojiIcon();
    _subscribePois();

    WidgetsBinding.instance.addPostFrameCallback((_) async {
      if (widget.searchLat != null && widget.searchLng != null) {
        final coords = LatLng(widget.searchLat!, widget.searchLng!);
        await _setDestination(
          coords,
          title: widget.searchQuery.isEmpty ? 'Selected location' : widget.searchQuery,
          addMarker: true,
        );

        if (widget.autoRouteOnOpen) {
          Future.delayed(const Duration(milliseconds: 100), () {
            _showRoute(wheelchair: false);
          });
```

```
      }
    }

    setState(() {
      _markers.removeWhere((m) {
        final id = m.markerId.value;
        return !(id == 'current_location' || id == 'searched' || id.startsWith('bus_'));
      });
    });
  });
}

/// Re-apply map style when theme changes.
@override
void didChangeDependencies() {
  super.didChangeDependencies();
  WidgetsBinding.instance.addPostFrameCallback((_) => _applyMapStyleForTheme());
}

/// Clean up controllers, streams, and timers.
@override
void dispose() {
  _pageCtrl.dispose();
  _poiSub?.cancel();
  _busPollTimer?.cancel();
  for (final t in _busAnimTimers.values) {
    t.cancel();
  }
  _busAnimTimers.clear();
  super.dispose();
}

/// Subscribe to Firestore POIs and keep flip controllers in sync.
void _subscribePois() {
  _poiSub = CampusPoiRepository.instance.streamAllActiveOrdered().listen((list) {
    if (!mounted) return;
    setState(() {
      _pois
        ..clear()
        ..addAll(list);
      _flipCtrls
        ..clear()
        ..addAll(List.generate(_pois.length, (_) => GlobalKey<FlipCardState>()));
    });
  }, onError: (e) => debugPrint('POI stream error: $e'));
}

/// Request location permission and mark the current location.
Future<void> _initLocation() async {
  try {
```

```
    final serviceEnabled = await Geolocator.isLocationServiceEnabled();
    if (!serviceEnabled) {
      _toast('Location services are disabled.');
      return;
    }

    LocationPermission permission = await Geolocator.checkPermission();
    if (permission == LocationPermission.denied) {
      permission = await Geolocator.requestPermission();
      if (permission == LocationPermission.denied) {
        _toast('Location permission denied.');
        return;
      }
    }
    if (permission == LocationPermission.deniedForever) {
      _toast('Location permission permanently denied.');
      return;
    }

    final pos = await Geolocator.getCurrentPosition(desiredAccuracy:
LocationAccuracy.high);
    setState(() {
      _currentLocation = LatLng(pos.latitude, pos.longitude);
      final next = <Marker>{..._markers}
        ..removeWhere((m) => m.markerId.value == 'current_location');
      next.add(Marker(
        markerId: const MarkerId('current_location'),
        position: _currentLocation!,
        infoWindow: const InfoWindow(title: 'You are here'),
        icon: BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueAzure),
      ));
      _markers = next;
    });
  } catch (_) {
    _toast('Failed to get location');
  }
 }

 /// Camera helper to focus a coordinate.
 Future<void> _animateTo(LatLng latLng, {double zoom = 17}) async {
  final c = await _controller.future;
  await c.animateCamera(CameraUpdate.newLatLngZoom(latLng, zoom));
 }

 /// Center the map on the user's latest known location.
 Future<void> _animateToCurrent() async {
  try {
    final last = await Geolocator.getLastKnownPosition();
    if (last != null) {
      await _animateTo(LatLng(last.latitude, last.longitude), zoom: 17);
```

```
    }
    final fresh = await Geolocator.getCurrentPosition(
      desiredAccuracy: LocationAccuracy.best,
      timeLimit: const Duration(seconds: 6),
    );
    final here = LatLng(fresh.latitude, fresh.longitude);
    if (!mounted) return;
    setState(() {
      final next = <Marker>{..._markers}
        ..removeWhere((m) => m.markerId.value == 'current_location');
      next.add(Marker(
        markerId: const MarkerId('current_location'),
        position: here,
        infoWindow: const InfoWindow(title: 'You are here'),
        icon: BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueAzure),
      ));
      _markers = next;
      _currentLocation = here;
    });
    await _animateTo(here, zoom: 17);
  } catch (_) {
    _toast('Couldn't get your location. Check GPS & permissions.');
  }
}

/// Select a destination, optionally add a marker, and reset any active route.
Future<void> _setDestination(
    LatLng coords, {
      String? title,
      bool addMarker = false,
      CampusPoi? poi,
    }) async {
  setState(() {
    _selectedLocation = coords;
    _selectedPoiForMarker = poi;
    _searchedVisible = addMarker;

    final next = <Marker>{..._markers}..removeWhere((m) => m.markerId.value ==
'searched');

    if (_searchedVisible) {
      next.add(Marker(
        markerId: const MarkerId('searched'),
        position: coords,
        infoWindow: title == null ? const InfoWindow() : InfoWindow(title: title),
        onTap: widget.isAdmin ? _onMarkerTap : null,
      ));
    }
    _markers = next;
```

```
    _polylines.clear();
    _routeInfo = null;
  });
  await _animateTo(coords);
}

/// Clear destination, marker, and route state.
void _clearNavigation() {
  setState(() {
    _selectedLocation = null;
    _selectedPoiForMarker = null;
    _searchedVisible = false;
    _polylines.clear();
    _routeInfo = null;
    _markers.removeWhere((m) => m.markerId.value == 'searched');
  });
}

/// Remove only the red "searched" marker and its route.
void _clearSearchedMarker() {
  setState(() {
    _searchedVisible = false;
    _selectedPoiForMarker = null;
    _markers.removeWhere((m) => m.markerId.value == 'searched');
    _polylines.clear();
    _routeInfo = null;
  });
}

/// Admin-only: tapping the red marker opens an editor (prefilled if near a POI).
void _onMarkerTap() {
  if (!widget.isAdmin || _selectedLocation == null) return;
  CampusPoi? existing = _selectedPoiForMarker;
  existing ??= _nearestPoiTo(_selectedLocation!, maxMeters: 30);
  _openPoiEditor(initialPos: _selectedLocation!, existing: existing);
}

/// Find the nearest known POI to a point within a given radius (meters).
CampusPoi? _nearestPoiTo(LatLng p, {double maxMeters = 30}) {
  double best = double.infinity;
  CampusPoi? bestPoi;
  for (final poi in _pois) {
    final d = _distanceMeters(p, poi.latLng);
    if (d < best) {
      best = d;
      bestPoi = poi;
    }
  }
  if (best <= maxMeters) return bestPoi;
  return null;
```

```
  }

  /// Request a walking or wheelchair-friendly route and render it.
  Timer? _routeDebounce;
  Future<void> _showRoute({required bool wheelchair}) async {
    if (_selectedLocation == null) {
      _toast('Choose a place first.');
      return;
    }
    if (_currentLocation == null) {
      await _initLocation();
      if (_currentLocation == null) {
        _toast('Enable location to get directions.');
        return;
      }
    }

    _routeDebounce?.cancel();
    _routeDebounce = Timer(const Duration(milliseconds: 200), () async {
      try {
        final route =
        await _getRouteWithStats(_currentLocation!, _selectedLocation!, wheelchair:
wheelchair);
        setState(() {
          _isWheelchairRoute = wheelchair;
          _routeInfo = route;
          _polylines = {
            Polyline(
              polylineId: PolylineId(wheelchair ? 'wheelchair' : 'walking'),
              points: route.points,
              color: wheelchair
                  ? Theme.of(context).colorScheme.secondary
                  : Theme.of(context).colorScheme.primary,
              width: 6,
              patterns: [PatternItem.dot, PatternItem.gap(12)],
            ),
          };
        });
        await _fitBoundsToPolyline(route.points);
      } catch (e) {
        _toast(e.toString());
      }
    });
  }

  /// Snackbar helper.
  void _toast(String msg) {
    if (!mounted) return;
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text(msg)));
  }
```

```
/// Call Google Directions API and return decoded polyline + stats.
Future<RouteInfo> _getRouteWithStats(LatLng origin, LatLng dest,
    {required bool wheelchair}) async {
  final params = {
   'origin': '${origin.latitude},${origin.longitude}',
   'destination': '${dest.latitude},${dest.longitude}',
   'mode': 'walking',
   'key': apiKey,
  };
  final url = Uri.https('maps.googleapis.com', '/maps/api/directions/json', params);
  final resp = await http.get(url);
  if (resp.statusCode != 200) throw HttpException('HTTP ${resp.statusCode}');
  final data = json.decode(resp.body) as Map<String, dynamic>;
  if (data['status'] != 'OK') {
   final em = (data['error_message'] as String?) ?? '';
   throw Exception('Directions: ${data['status']} ${em.isNotEmpty ? '· $em' : ''}');
  }

  final routes = (data['routes'] as List);
  if (routes.isEmpty) throw Exception('No routes');
  final route0 = routes.first as Map<String, dynamic>;
  final legs = (route0['legs'] as List);
  String distanceText = '', durationText = '';
  int distanceMeters = 0, durationSeconds = 0;
  if (legs.isNotEmpty) {
   final leg0 = legs.first as Map<String, dynamic>;
   final dist = (leg0['distance'] as Map<String, dynamic>);
   final dur = (leg0['duration'] as Map<String, dynamic>);
   distanceText = dist['text'] as String? ?? '';
   durationText = dur['text'] as String? ?? '';
   distanceMeters = (dist['value'] as num?)?.toInt() ?? 0;
   durationSeconds = (dur['value'] as num?)?.toInt() ?? 0;
  }

  final encoded = route0['overview_polyline']['points'] as String;
  final points = _decodePolyline(encoded);
  return RouteInfo(
   points: points,
   distanceText: distanceText,
   durationText: durationText,
   distanceMeters: distanceMeters,
   durationSeconds: durationSeconds,
  );
}

/// Decode an encoded polyline string to a list of LatLng.
List<LatLng> _decodePolyline(String encoded) {
  final List<LatLng> poly = [];
  int index = 0, lat = 0, lng = 0;
```

```
  while (index < encoded.length) {
    int b, shift = 0, result = 0;
    do {
      b = encoded.codeUnitAt(index++) - 63;
      result |= (b & 0x1f) << shift;
      shift += 5;
    } while (b >= 0x20);
    final dlat = (result & 1) != 0 ? ~(result >> 1) : (result >> 1);
    lat += dlat;

    shift = 0;
    result = 0;
    do {
      b = encoded.codeUnitAt(index++) - 63;
      result |= (b & 0x1f) << shift;
      shift += 5;
    } while (b >= 0x20);
    final dlng = (result & 1) != 0 ? ~(result >> 1) : (result >> 1);
    lng += dlng;

    poly.add(LatLng(lat / 1e5, lng / 1e5));
  }
  return poly;
}

/// Fit the camera to show an entire polyline with padding.
Future<void> _fitBoundsToPolyline(List<LatLng> pts) async {
  if (pts.isEmpty) return;
  double minLat = pts.first.latitude, maxLat = pts.first.latitude;
  double minLng = pts.first.longitude, maxLng = pts.first.longitude;
  for (final p in pts) {
    if (p.latitude < minLat) minLat = p.latitude;
    if (p.latitude > maxLat) maxLat = p.latitude;
    if (p.longitude < minLng) minLng = p.longitude;
    if (p.longitude > maxLng) maxLng = p.longitude;
  }
  final c = await _controller.future;
  await c.animateCamera(
    CameraUpdate.newLatLngBounds(
      LatLngBounds(
        southwest: LatLng(minLat, minLng),
        northeast: LatLng(maxLat, maxLng),
      ),
      60,
    ),
  );
}

/// Text search for places near campus; centers and drops a marker.
```

```
Future<void> _searchPlace(String query) async {
  if (query.trim().isEmpty) return;
  try {
    final url =
    Uri.https('maps.googleapis.com', '/maps/api/place/textsearch/json', {
      'query': query,
      'location': '${campusCenter.latitude},${campusCenter.longitude}',
      'radius': '2000',
      'key': apiKey,
    });
    final resp = await http.get(url);
    if (resp.statusCode != 200) throw HttpException('HTTP ${resp.statusCode}');
    final data = json.decode(resp.body) as Map<String, dynamic>;
    if (data['status'] == 'OK' && (data['results'] as List).isNotEmpty) {
      final result = (data['results'] as List).first as Map<String, dynamic>;
      final loc = result['geometry']['location'] as Map<String, dynamic>;
      final name = (result['name'] as String?) ?? query;
      final latLng = LatLng(
          (loc['lat'] as num).toDouble(), (loc['lng'] as num).toDouble());
      await _setDestination(latLng, title: name, addMarker: true);
    } else {
      _toast('No results for "$query" (${data['status']})');
    }
  } catch (_) {
    _toast('Search failed. Check network/API key.');
  }
}

/// Admin-only: open editor for the current red marker (blank if new).
void _openEditorForSearched() {
  if (!widget.isAdmin || _selectedLocation == null) return;
  _openPoiEditor(initialPos: _selectedLocation!);
}

/// Bottom-sheet editor to create/update/delete a POI.
Future<void> _openPoiEditor({LatLng? initialPos, CampusPoi? existing}) async {
  final name = TextEditingController(text: existing?.name ?? '');
  final imageUrl = TextEditingController(text: existing?.imageUrl ?? '');
  final address = TextEditingController(text: existing?.address ?? '');
  final phone = TextEditingController(text: existing?.phone ?? '');
  final website = TextEditingController(text: existing?.website ?? '');
  final closesAt = TextEditingController(text: existing?.closesAt ?? '');
  final order = TextEditingController(text: (existing?.order ?? 0).toString());
  final category = TextEditingController(text: existing?.category ?? '');
  final lat = TextEditingController(
      text: (existing?.lat ?? initialPos?.latitude ?? campusCenter.latitude)
          .toStringAsFixed(6));
  final lng = TextEditingController(
      text: (existing?.lng ?? initialPos?.longitude ?? campusCenter.longitude)
          .toStringAsFixed(6));
```

```
  bool isOpenNow = existing?.isOpenNow ?? true;
  bool active = existing?.active ?? true;

  await showModalBottomSheet(
    context: context,
    isScrollControlled: true,
    builder: (_) => Padding(
      padding: EdgeInsets.only(
        bottom: MediaQuery.of(context).viewInsets.bottom,
        left: 16, right: 16, top: 16,
      ),
      child: SingleChildScrollView(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
            Text(existing == null ? 'Publish carousel' : 'Edit carousel',
                style: const TextStyle(fontSize: 18, fontWeight: FontWeight.bold)),
            const SizedBox(height: 12),
            TextField(controller: name, decoration: const InputDecoration(labelText: 'Name')),
            Row(children: [
              Expanded(child: TextField(controller: lat, decoration: const
InputDecoration(labelText: 'Latitude'), keyboardType: TextInputType.number)),
              const SizedBox(width: 12),
              Expanded(child: TextField(controller: lng, decoration: const
InputDecoration(labelText: 'Longitude'), keyboardType: TextInputType.number)),
            ]),
            TextField(controller: imageUrl, decoration: const InputDecoration(labelText: 'Image
URL')),
            TextField(controller: address, decoration: const InputDecoration(labelText:
'Address')),
            TextField(controller: phone, decoration: const InputDecoration(labelText: 'Phone')),
            TextField(controller: website, decoration: const InputDecoration(labelText:
'Website')),
            TextField(controller: closesAt, decoration: const InputDecoration(labelText: 'Closes
At (text)')),
            TextField(controller: order, decoration: const InputDecoration(labelText: 'Order
(int)')),
            TextField(controller: category, decoration: const InputDecoration(labelText:
'Category (optional)')),
            SwitchListTile(value: isOpenNow, onChanged: (v) => isOpenNow = v, title: const
Text('Open Now')),
            SwitchListTile(value: active, onChanged: (v) => active = v, title: const
Text('Active')),

            const SizedBox(height: 12),
            Row(
              children: [
                if (existing != null)
                  OutlinedButton.icon(
                    icon: const Icon(Icons.delete_outline),
```

```
                label: const Text('Delete'),
                style: OutlinedButton.styleFrom(
                  foregroundColor: Theme.of(context).colorScheme.error,
                  side: BorderSide(color: Theme.of(context).colorScheme.error),
                ),
                onPressed: () async {
                  final ok = await showDialog<bool>(
                    context: context,
                    builder: (_) => AlertDialog(
                      title: const Text('Delete carousel?'),
                      content: Text('Remove "${existing.name}" from the map?'),
                      actions: [
                        TextButton(onPressed: () => Navigator.pop(context, false), child: const
Text('Cancel')),
                        FilledButton(
                          style: FilledButton.styleFrom(backgroundColor:
Theme.of(context).colorScheme.error),
                          onPressed: () => Navigator.pop(context, true),
                          child: const Text('Delete'),
                        ),
                      ],
                    ),
                  ) ?? false;

                  if (!ok) return;
                  try {
                    await CampusPoiRepository.instance.deleteById(existing.id);
                    if (mounted) _clearSearchedMarker();
                    if (context.mounted) Navigator.pop(context);
                    _toast('Deleted "${existing.name}"');
                  } catch (e) {
                    _toast('Delete failed: $e');
                  }
                },
              ),
            const Spacer(),
            TextButton(onPressed: () => Navigator.pop(context), child: const Text('Cancel')),
            const SizedBox(width: 8),
            FilledButton(
              child: const Text('Publish'),
              onPressed: () async {
                try {
                  final dLat = double.parse(lat.text.trim());
                  final dLng = double.parse(lng.text.trim());
                  final ord = int.tryParse(order.text.trim()) ?? 0;

                  final poi = CampusPoi(
                    id: existing?.id ?? '',
                    name: name.text.trim(),
                    lat: dLat,
```

```
                    lng: dLng,
                    imageUrl: imageUrl.text.trim(),
                    address: address.text.trim().isEmpty ? null : address.text.trim(),
                    phone: phone.text.trim().isEmpty ? null : phone.text.trim(),
                    website: website.text.trim().isEmpty ? null : website.text.trim(),
                    hours: existing?.hours ?? const {},
                    isOpenNow: isOpenNow,
                    closesAt: closesAt.text.trim(),
                    order: ord,
                    active: active,
                    category: category.text.trim().isEmpty ? null : category.text.trim(),
                  );

                  await CampusPoiRepository.instance.upsert(poi);

                  if (mounted) _clearSearchedMarker();
                  if (context.mounted) Navigator.pop(context);
                  _toast('Published "${poi.name}"');
                } catch (e) {
                  _toast('Publish failed: $e');
                }
              },
            ),
          ],
        ),
        const SizedBox(height: 8),
      ],
    ),
  ),
 );
}

 /// Admin-only delete confirmation when triggered from a card.
 Future<void> _confirmDeleteFromCard(CampusPoi poi) async {
  if (!widget.isAdmin) return;
  final ok = await showDialog<bool>(
   context: context,
   builder: (_) => AlertDialog(
    title: const Text('Delete carousel?'),
    content: Text('Remove "${poi.name}" from the map?'),
    actions: [
     TextButton(onPressed: () => Navigator.pop(context, false), child: const
Text('Cancel')),
     FilledButton(
      style: FilledButton.styleFrom(backgroundColor:
Theme.of(context).colorScheme.error),
      onPressed: () => Navigator.pop(context, true),
      child: const Text('Delete'),
     ),
```

```
        ],
      ),
    ) ?? false;
    if (!ok) return;
    try {
      await CampusPoiRepository.instance.deleteById(poi.id);
      if (mounted) _clearSearchedMarker();
      _toast('Deleted "${poi.name}"');
    } catch (e) {
      _toast('Delete failed: $e');
    }
  }

  /// Start polling and animating live bus markers.
  void _startLiveBuses() {
    if (_liveOn) return;
    _liveOn = true;
    _didAutoFocusBuses = false;
    _animateTo(campusCenter, zoom: 13);
    _busPollTimer?.cancel();
    _busPollTimer = Timer.periodic(const Duration(seconds: 2), (_) =>
_fetchAndRenderSiri());
    _fetchAndRenderSiri();
    setState(() {});
  }

  /// Stop live bus updates and clear markers.
  void _stopLiveBuses() {
    _busPollTimer?.cancel();
    _busPollTimer = null;
    _liveOn = false;
    for (final t in _busAnimTimers.values) {
      t.cancel();
    }
    _busAnimTimers.clear();
    _busPrevPos.clear();
    setState(() {
      _markers.removeWhere((m) => m.markerId.value.startsWith('bus_'));
    });
  }

  /// Fetch SIRI-VM feed and update bus markers with smooth movement.
  Future<void> _fetchAndRenderSiri() async {
    try {
      final resp = await http.get(Uri.parse(bodsSiriUrl));
      if (resp.statusCode != 200) {
        _toast('SIRI feed HTTP ${resp.statusCode}');
        return;
      }
      final doc = xml.XmlDocument.parse(resp.body);
```

```
    final vehicleActivities = doc.findAllElements('VehicleActivity', namespace: '*');

    final idsNow = <String>{};
    LatLng? firstPos;
    int totalCount = 0;

    for (final va in vehicleActivities) {
      final mvjIter = va.findAllElements('MonitoredVehicleJourney', namespace: '*');
      if (mvjIter.isEmpty) continue;
      final mvj = mvjIter.first;

      final idRaw = _textOfFirst(mvj, 'VehicleRef') ?? _textOfFirst(va, 'VehicleRef');
      if (idRaw == null || idRaw.trim().isEmpty) continue;
      final id = idRaw.trim();

      final locIter = mvj.findAllElements('VehicleLocation', namespace: '*');
      if (locIter.isEmpty) continue;
      final loc = locIter.first;

      final latStr = _textOfFirst(loc, 'Latitude');
      final lngStr = _textOfFirst(loc, 'Longitude');
      if (latStr == null || lngStr == null) continue;

      final lat = double.tryParse(latStr);
      final lng = double.tryParse(lngStr);
      if (lat == null || lng == null) continue;

      final next = LatLng(lat, lng);
      totalCount++;
      final markerId = 'bus_$id';
      idsNow.add(markerId);
      firstPos ??= next;

      final prev = _busPrevPos[id];
      _smoothMove(
        id: id,
        markerId: markerId,
        from: prev ?? next,
        to: next,
        icon: _emojiIcon ??
BitmapDescriptor.defaultMarkerWithHue(BitmapDescriptor.hueAzure),
      );
      _busPrevPos[id] = next;
    }

    if (!mounted) return;

    setState(() {
      _markers.removeWhere((m) => m.markerId.value.startsWith('bus_') &&
!idsNow.contains(m.markerId.value));
```

```
    });

    if (!_didAutoFocusBuses && firstPos != null) {
      final c = await _controller.future;
      await c.animateCamera(CameraUpdate.newLatLngZoom(firstPos!, 12));
      _didAutoFocusBuses = true;
    }

    if (totalCount == 0) {
      _toast('No vehicles in the live feed right now.');
    }
  } catch (e) {
    _toast('Could not parse live bus feed.');
  }
}

/// Haversine distance between two coordinates (meters).
double _distanceMeters(LatLng a, LatLng b) {
  const R = 6371000.0;
  final dLat = (b.latitude - a.latitude) * pi / 180.0;
  final dLng = (b.longitude - a.longitude) * pi / 180.0;
  final la1 = a.latitude * pi / 180.0;
  final la2 = b.latitude * pi / 180.0;

  final h = sin(dLat / 2) * sin(dLat / 2) +
      cos(la1) * cos(la2) * sin(dLng / 2) * sin(dLng / 2);

  return 2 * R * asin(sqrt(h));
}

/// Interpolates a bus marker position over time for smooth animation.
void _smoothMove({
  required String id,
  required String markerId,
  required LatLng from,
  required LatLng to,
  required BitmapDescriptor icon,
  int durationMs = 1800,
  int stepMs = 60,
}) {
  _busAnimTimers[id]?.cancel();
  final dist = _distanceMeters(from, to);
  if (dist < 2 || dist > 2000) {
    _putEmojiMarker(markerId, to, icon);
    return;
  }
  final totalSteps = (durationMs / stepMs).ceil();
  int step = 0;
  _busAnimTimers[id] = Timer.periodic(Duration(milliseconds: stepMs), (timer) {
    if (!mounted) {
```

```
        timer.cancel();
        return;
      }
      step++;
      final t = (step / totalSteps).clamp(0.0, 1.0);
      final lat = from.latitude + (to.latitude - from.latitude) * t;
      final lng = from.longitude + (to.longitude - from.longitude) * t;
      final pos = LatLng(lat, lng);
      _putEmojiMarker(markerId, pos, icon);
      if (step >= totalSteps) timer.cancel();
    });
  }

  /// Insert/replace a marker with the given icon at a position.
  void _putEmojiMarker(String markerId, LatLng pos, BitmapDescriptor icon) {
    setState(() {
      _markers.removeWhere((m) => m.markerId.value == markerId);
      _markers.add(Marker(
        markerId: MarkerId(markerId),
        position: pos,
        flat: true,
        anchor: const Offset(0.5, 0.5),
        zIndex: 1000.0,
        icon: icon,
      ));
    });
  }

  /// Pre-render a 🚌 emoji-based bitmap for bus markers.
  Future<void> _loadEmojiIcon() async {
    try {
      _emojiIcon = await _makeEmojiMarker('🚌', fontSize: 56);
    } catch (_) {
      _emojiIcon = null;
    }
  }

  /// Render an emoji to a bitmap descriptor for custom markers.
  Future<BitmapDescriptor> _makeEmojiMarker(String emoji, {double fontSize = 48})
async {
    final tp = TextPainter(
      text: TextSpan(text: emoji, style: TextStyle(fontSize: fontSize)),
      textDirection: TextDirection.ltr,
    )..layout();
    final double w = tp.width + 12;
    final double h = tp.height + 12;

    final recorder = ui.PictureRecorder();
    final canvas = ui.Canvas(recorder);
    final bg = Paint()..color = Colors.white.withOpacity(0.92);
```

```
    final rrect = RRect.fromRectAndRadius(Rect.fromLTWH(0, 0, w, h), const
Radius.circular(14));
    canvas.drawRRect(rrect, bg);
    tp.paint(canvas, Offset((w - tp.width) / 2, (h - tp.height) / 2));
    final img = await recorder.endRecording().toImage(w.ceil(), h.ceil());
    final bytes = (await img.toByteData(format:
ui.ImageByteFormat.png))!.buffer.asUint8List();
    return BitmapDescriptor.fromBytes(bytes);
  }

  /// Safe XML text getter for first matching element.
  String? _textOfFirst(xml.XmlElement parent, String localName) {
    final n = parent.findAllElements(localName, namespace: '*');
    if (n.isEmpty) return null;
    final txt = n.first.text.trim();
    return txt.isEmpty ? null : txt;
  }

  /// Apply dark/light map style based on current theme.
  Future<void> _applyMapStyleForTheme() async {
    if (!_controller.isCompleted) return;
    final brightness = Theme.of(context).brightness;
    if (_lastBrightness == brightness) return;
    final c = await _controller.future;
    await c.setMapStyle(brightness == Brightness.dark ? _mapStyleDark : null);
    _lastBrightness = brightness;
  }

  /// Build the map, overlays (admin search, route chip), carousel, and actions.
  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);

    final markersToShow =
    _searchedVisible ? _markers : _markers.where((m) => m.markerId.value !=
'searched').toSet();

    return Scaffold(
      body: Stack(
        children: [
          GoogleMap(
            initialCameraPosition: const CameraPosition(target: campusCenter, zoom: 16),
            myLocationEnabled: true,
            myLocationButtonEnabled: false,
            zoomControlsEnabled: false,
            mapToolbarEnabled: false,
            markers: markersToShow,
            polylines: _polylines,
            onMapCreated: (controller) async {
              if (!_controller.isCompleted) {
```

```
      _controller.complete(controller);
    }
    await _applyMapStyleForTheme();
  },
),

// Admin search overlay
if (widget.isAdmin)
  Positioned(
    left: 16,
    right: 16,
    top: MediaQuery.of(context).padding.top + 12,
    child: Material(
      elevation: 3,
      borderRadius: BorderRadius.circular(12),
      child: TextField(
        controller: _adminSearchCtrl,
        textInputAction: TextInputAction.search,
        onSubmitted: (q) => _searchPlace(q),
        decoration: InputDecoration(
          hintText: 'Search a building...',
          prefixIcon: const Icon(Icons.search),
          suffixIcon: IconButton(
            icon: const Icon(Icons.clear),
            onPressed: () {
              _adminSearchCtrl.clear();
              _clearNavigation();
            },
          ),
          border: OutlineInputBorder(
            borderRadius: BorderRadius.circular(12),
            borderSide: BorderSide.none,
          ),
          filled: true,
          fillColor: theme.cardColor,
          contentPadding:
          const EdgeInsets.symmetric(horizontal: 12, vertical: 10),
        ),
      ),
    ),
  ),

// Route info chip
if (_routeInfo != null)
  Positioned(
    right: 16,
    bottom: 320,
    child: Container(
      padding: const EdgeInsets.symmetric(vertical: 8, horizontal: 10),
      decoration: BoxDecoration(
```

```
        color: theme.cardColor,
        borderRadius: BorderRadius.circular(10),
        boxShadow: const [BoxShadow(blurRadius: 6, color: Colors.black26)],
        border: Border.all(
          color: _isWheelchairRoute
              ? theme.colorScheme.secondary
              : theme.colorScheme.primary,
          width: 1,
        ),
      ),
      child: Column(
        mainAxisSize: MainAxisSize.min,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Row(
            mainAxisSize: MainAxisSize.min,
            children: [
              const Icon(Icons.route, size: 18),
              const SizedBox(width: 6),
              Text(
                _routeInfo!.distanceText.isNotEmpty
                    ? _routeInfo!.distanceText
                    : _formatDistance(_routeInfo!.distanceMeters),
                style: theme.textTheme.bodyMedium,
              ),
            ],
          ),
          const SizedBox(height: 6),
          Row(
            mainAxisSize: MainAxisSize.min,
            children: [
              Icon(_isWheelchairRoute ? Icons.accessible : Icons.timer, size: 18),
              const SizedBox(width: 6),
              Text(
                _routeInfo!.durationText.isNotEmpty
                    ? _routeInfo!.durationText
                    : _formatDuration(_routeInfo!.durationSeconds),
                style: theme.textTheme.bodyMedium,
              ),
            ],
          ),
        ],
      ),
    ),
  ),

// POI carousel
if (_pois.isNotEmpty)
  Positioned(
    left: 0,
```

```
right: 0,
bottom: 90,
child: SizedBox(
  height: 210,
  child: PageView.builder(
    controller: _pageCtrl,
    itemCount: _pois.length,
    onPageChanged: (i) {
      _currentIndex = i;
      if (i >= 0 && i < _pois.length) {
        final b = _pois[i];
        _selectedLocation = b.latLng;
      }
    },
    itemBuilder: (_, i) {
      final b = _pois[i];
      final key = _flipCtrls[i];

      return Padding(
        padding: const EdgeInsets.symmetric(horizontal: 8),
        child: FlipCard(
          key: key,
          speed: 300,
          front: _PoiCardFront(
            poi: b,
            onTap: () async {
              await _setDestination(b.latLng,
                  title: b.name, addMarker: true, poi: b);
              key.currentState?.toggleCard();
            },
            onLongPressDelete: widget.isAdmin
                ? () => _confirmDeleteFromCard(b)
                : null,
          ),
          back: _PoiCardBack(
            poi: b,
            onNavigate: () async {
              await _setDestination(
                  b.latLng, title: b.name, addMarker: true, poi: b);
              await _showRoute(wheelchair: false);
            },
            onEdit: widget.isAdmin
                ? () async {
              await _setDestination(b.latLng,
                  title: b.name, addMarker: true, poi: b);
              _onMarkerTap();
            }
                : null,
            onFlipBack: () => key.currentState?.toggleCard(),
          ),
```

```
                  ),
                );
              },
            ),
          ),
        ),

        // Bottom action row
        Positioned(
          left: 0,
          right: 0,
          bottom: 20,
          child: Row(
            mainAxisAlignment: MainAxisAlignment.spaceEvenly,
            children: [
              _RoundIconButton(icon: Icons.directions_walk, onPressed: () =>
_showRoute(wheelchair: false)),
              _RoundIconButton(icon: Icons.accessible, onPressed: () =>
_showRoute(wheelchair: true)),
              _RoundIconButton(icon: Icons.my_location, onPressed: _animateToCurrent),
              _RoundIconButton(
                icon: Icons.directions_bus,
                onPressed: () {
                  if (_liveOn) {
                    _stopLiveBuses();
                    _toast('Live buses off');
                  } else {
                    _startLiveBuses();
                    _toast('Live buses on');
                  }
                },
              ),
              _RoundIconButton(
                icon: Icons.clear,
                onPressed: _clearNavigation,
              ),
            ],
          ),
        ),
      ],
    ),
  );
 }
}

/// Front face of a carousel card showing image + open status.
class _PoiCardFront extends StatelessWidget {
  final CampusPoi poi;
  final VoidCallback onTap;
  final VoidCallback? onLongPressDelete;
```

```
const _PoiCardFront({
  required this.poi,
  required this.onTap,
  this.onLongPressDelete,
});

@override
Widget build(BuildContext context) {
  final theme = Theme.of(context);
  final statusColor =
  poi.isOpenNow ? theme.colorScheme.primary : theme.colorScheme.error;

  return GestureDetector(
    onTap: onTap,
    onLongPress: onLongPressDelete,
    child: Container(
      width: 280,
      decoration: BoxDecoration(
        color: theme.cardColor,
        borderRadius: BorderRadius.circular(18),
        boxShadow: const [BoxShadow(blurRadius: 10, color: Colors.black26)],
      ),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          ClipRRect(
            borderRadius:
            const BorderRadius.vertical(top: Radius.circular(18)),
            child: Image.network(
              poi.imageUrl,
              height: 130,
              width: double.infinity,
              fit: BoxFit.cover,
              errorBuilder: (_, __, ___) =>
                  Container(height: 130, color: Colors.grey.shade300),
            ),
          ),
          Padding(
            padding: const EdgeInsets.all(12),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.start,
              children: [
                Text(poi.name,
                    maxLines: 1,
                    overflow: TextOverflow.ellipsis,
                    style: const TextStyle(
                        fontSize: 16, fontWeight: FontWeight.bold)),
                const SizedBox(height: 4),
                Text(
```

```
                poi.isOpenNow
                    ? "Open · Closes ${poi.closesAt}"
                    : "Closed · ${poi.closesAt}",
                maxLines: 1,
                overflow: TextOverflow.ellipsis,
                style: TextStyle(fontSize: 13, color: statusColor),
              ),
            ],
          ),
        ),
      ],
    ),
  ),
 );
 }
}

/// Back face of a carousel card with details + actions.
class _PoiCardBack extends StatelessWidget {
  final CampusPoi poi;
  final VoidCallback onNavigate;
  final VoidCallback? onEdit;
  final VoidCallback onFlipBack;

  const _PoiCardBack({
    required this.poi,
    required this.onNavigate,
    required this.onFlipBack,
    this.onEdit,
  });

  @override
  Widget build(BuildContext context) {
    final theme = Theme.of(context);

    return Container(
      width: 280,
      decoration: BoxDecoration(
        color: theme.cardColor,
        borderRadius: BorderRadius.circular(18),
        boxShadow: const [BoxShadow(blurRadius: 10, color: Colors.black26)],
      ),
      padding: const EdgeInsets.all(12),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(poi.name,
              maxLines: 1,
              overflow: TextOverflow.ellipsis,
              style: const TextStyle(fontSize: 16, fontWeight: FontWeight.bold)),
```

```
const SizedBox(height: 6),
if (poi.address != null) ...[
  Row(children: [
    const Icon(Icons.place, size: 16),
    const SizedBox(width: 6),
    Expanded(
      child: Text(poi.address!,
          maxLines: 2, overflow: TextOverflow.ellipsis),
    ),
  ]),
  const SizedBox(height: 4),
],
if (poi.phone != null) ...[
  Row(children: [
    const Icon(Icons.phone, size: 16),
    const SizedBox(width: 6),
    Expanded(
      child: Text(poi.phone!, maxLines: 1, overflow: TextOverflow.ellipsis),
    ),
  ]),
  const SizedBox(height: 4),
],
if (poi.website != null) ...[
  Row(children: [
    const Icon(Icons.public, size: 16),
    const SizedBox(width: 6),
    Expanded(
      child: Text(poi.website!,
          maxLines: 1, overflow: TextOverflow.ellipsis),
    ),
  ]),
  const SizedBox(height: 8),
],
const Spacer(),
Row(
  children: [
    ElevatedButton.icon(
      onPressed: onNavigate,
      icon: const Icon(Icons.navigation),
      label: const Text('Navigate'),
      style: ElevatedButton.styleFrom(
        backgroundColor: theme.colorScheme.primary,
        foregroundColor: theme.colorScheme.onPrimary,
      ),
    ),
    const SizedBox(width: 8),
    if (onEdit != null)
      OutlinedButton.icon(
        onPressed: onEdit,
        icon: const Icon(Icons.edit),
```

```
              label: const Text('Edit'),
            ),
            const Spacer(),
            IconButton(
              tooltip: 'Flip back',
              icon: const Icon(Icons.flip),
              onPressed: onFlipBack,
            ),
          ],
        ),
      ],
    ),
  );
 }
}

/// Round primary icon button used in the bottom actions row.
class _RoundIconButton extends StatelessWidget {
 final IconData icon;
 final VoidCallback onPressed;
 const _RoundIconButton({required this.icon, required this.onPressed, super.key});

 @override
 Widget build(BuildContext context) {
  final theme = Theme.of(context);
  return ElevatedButton(
   onPressed: onPressed,
   style: ElevatedButton.styleFrom(
    shape: const CircleBorder(),
    backgroundColor: theme.colorScheme.primary,
    foregroundColor: theme.colorScheme.onPrimary,
    padding: const EdgeInsets.all(16),
    elevation: 2,
   ),
   child: Icon(icon, size: 22),
  );
 }
}

/// Lightweight route model to render polylines and badges.
class RouteInfo {
 final List<LatLng> points;
 final String distanceText;
 final String durationText;
 final int distanceMeters;
 final int durationSeconds;
 RouteInfo({
  required this.points,
  required this.distanceText,
  required this.durationText,
```

```
    required this.distanceMeters,
    required this.durationSeconds,
  });
}

/// Human-friendly distance formatter.
String _formatDistance(int meters) =>
    meters < 1000 ? '$meters m' : '${(meters / 1000).toStringAsFixed(1)} km';

/// Human-friendly duration formatter.
String _formatDuration(int seconds) {
  if (seconds <= 0) return '';
  final m = (seconds / 60).round();
  if (m < 60) return '$m min';
  final h = m ~/ 60;
  final rm = m % 60;
  return rm == 0 ? '$h hr' : '$h hr $rm min';
}
```