PRD / OneIITP Connect / Draco

Product Requirement Document: OneIITP App Enhancement Initiative (M2)

Version: 2.1

Author: Chaitanya Yash Jiya SHREYAS

Stakeholders: Students' Technical Council (STC), IIT Patna Student Body, IIT Patna Administration (IT Dept, Library, Transport, Dean of Student Affairs, Wellness Board, Security), Development Team (Student Volunteers), UI/UX Team (Student Volunteers), Finance Club (Challenge Initiator)

Last Updated: March 30, 2025

Status: M2 Planning - Ready for Refinement & Resourcing

> **This is a detailed PRD hand-fully crafted by 4 curious folks with outmost care and and a lot (!) of research. Please spend time reading this, nothing given here is unnecessary or just copy-pasted from other knowledge sources.** 🙏❤️
>
> **We believe if every page adds value and helps stakeholders understand the product, it's justified.**
>
> **Template Credit:** Manas J. Saloi (VP - Product @ Gojek Indonesia)

Table of Contents:

    - Cross-Platform Approach (React Native)
    - Backend Scalability
    - API Dependencies
    - Google Calendar Integration Approach
    - Report Harassment Implementation Considerations
    - Student Developer Context
4. How Do We Educate Users About Changes?
5. Non-Product Requirements
6. Out of Scope for this Phase (M2 Focus)
7. Rollout Plan (Incorporating Dogfooding)
8. Open Questions
9. MOMs / Appendices

> *When starting out, we asked our team only one question - "Why would people use OneIITP? What can make someone open the app and keep coming to it over and over?"*
>
> *And that stuck with us as our main playing arena - Make OneIITP indispensable in any IIT Patna's student core day to day workflow. Give them accessible and intuitive reasons to use the app rather than using a hundred different platforms.*

---

1. Introduction: The Current State & M2 Challenge

The OneIITP app, developed and maintained by the Students' Technical Council (STC), is currently live on the Google Play Store and Apple App Store. It serves as a foundational platform offering features like manual scheduling, event info, quick links, contacts, bus status display, and basic profile management including a QR code intended for gate pass usage. Although launched (considered M1), the app is facing significant challenges preventing it from achieving its primary goal: **to become the indispensable, go-to digital companion for every IIT Patna student.**

User feedback (Appendix B), RCA findings (Section 2), and observed usage patterns clearly indicate persistent issues with:

- **Performance and Stability:** Users experience notable lag, frequent crashes, and confusing update loops. Platform inconsistencies exist, hindering a uniform experience.
- **Data Reliability:** Critical information, particularly bus schedules, is often outdated due to reliance on manual update processes, eroding user trust.
- **Usability Friction:** Workflows like manual schedule entry are cumbersome and rarely used. Navigation lacks consistency, often missing clear back affordances. QR code management is problematic and intrusive reminders are reported. Interaction elements like forms and modals are perceived as difficult to use. Friction occurs when calling someone and it copy pastes.
- **Feature Gaps:** Critical convenience features, most notably deep integration with core campus systems (automated timetable via ERP, library integration), are missing. Important safety features (accessible harassment reporting) and requested productivity/utility tools (Lost & Found, Buy/Sell, Cab Share, To-dos, Calendar Sync) are not implemented.
- **Poor Onboarding:** The app currently fails to effectively communicate its value proposition or guide new users through setup, contributing to low activation and high churn rates.

This phase prioritizes delivering a **Minimum Viable Experience (MVE) instead of MVP** – a baseline level of quality, performance, essential utility, and usability – rather than introducing an excessive number of features prematurely.

---

2. Root Cause Analysis (RCA) Interview

Participants:

- **Interviewer:** PM
- **Interviewee:** STC Dev (Stakeholder responsible for the app)

(Interview Transcript Summary)

- **PM:** Thanks for meeting. We know OneIITP has potential, but adoption and consistent usage are low. What's your perspective on why students aren't using it daily?
- **STC Dev:** Honestly, it's frustrating. We built features we thought students needed – schedules, contacts, bus info. But people still rely on WhatsApp groups for timings or just don't bother opening the app after installing it. The feedback we get is often about bugs or it being slow, sometimes worse on one platform than another.
- **PM:** Okay, "slow" and "bugs." Let's dig into that. *Why* do you think it's perceived as slow or buggy?
- **STC Dev:** We've had different student teams working on it over semesters. Handover documentation isn't always perfect, so maybe the codebase isn't optimized? Technical debt is likely an issue. Plus, things like the bus schedule... it's manually updated by an STC member when they get the info. If they forget or are busy, the app shows wrong timings. That breaks trust immediately. Navigation also feels inconsistent sometimes.
- **PM:** So, potential technical debt, inconsistent development, lack of optimization contributing to performance issues, and manual data entry leading to unreliability. *Why* haven't we automated the bus schedule update?
- **STC Dev:** We asked the transport section, but there's no official API or digital feed they provide easily. It's usually just a PDF or a notice board update. Getting real-time data like GPS seemed even harder – it would need coordination with transport, maybe hardware costs for devices on buses, and someone to manage that system. It felt out of scope for past student teams given the time constraints.
- **PM:** Understood. Lack of accessible, reliable data sources is a key blocker for core utility. What about features like the academic schedule? Students *can* input their timetable manually, right? *Why* isn't that driving usage?
- **STC Dev:** It's just too much effort. Entering the whole timetable manually class by class, week by week? Nobody has time for that, especially when the official timetable exists in the ERP system. We explored ERP integration before, but getting secure API access and handling the complexities of different batches, branches, and electives seemed daunting for student teams to implement robustly within a limited timeframe.
- **PM:** So, high friction for manual tasks and difficulty integrating with official sources (ERP, Library) mean core academic utility is missing. Combined with the performance issues, unreliable data, and usability frustrations... *Why* would a student choose OneIITP over just checking ERP directly or asking on WhatsApp for bus timings?
- **STC Dev:** Exactly. Right now, there isn't a compelling *enough* reason for *daily* use. It doesn't reliably solve a major pain point significantly *better* than existing methods. To become essential, it needs to be rock-solid **stable**, the information *must* be **accurate**, the UX needs to be intuitive, and it needs to offer **convenience** they can't easily get elsewhere – like seeing their library books and official timetable automatically in one place. And that first impression is critical – if it crashes, or asks for an update that doesn't exist, or shows the wrong bus time, or feels clunky to use, they likely won't give it a second chance.

RCA Summary: Low adoption is not due to a lack of *potential* value, but stems primarily from:

1. **Poor Core Experience:** Instability, slow performance, confusing update loops, and inconsistent/unintuitive navigation erode trust and create significant user frustration.
2. **Unreliable Data:** Manually updated information (bus timings, potentially contacts) is frequently inaccurate, rendering key utility features untrustworthy and damaging credibility.
3. **High Friction & Missing Automation:** Manual tasks (e.g., timetable entry) are too cumbersome. Lack of integration with essential campus systems (ERP, Library) severely limits the app's unique value proposition. Poorly designed interactions (forms, modals) add to friction.
4. **Weak or Unclear Value Proposition:** The app currently fails to solve core student needs significantly better, faster, or more reliably than existing methods (ERP, WhatsApp, notice boards). The onboarding doesn't effectively communicate potential benefits or guide users smoothly.

3. Why Invest Further in OneIITP? (The Why - Post-RCA)

- **Addressing Proven Student Needs:** User feedback and observed workarounds (WhatsApp groups for timings, requests for integrations) clearly demonstrate a strong underlying demand for a centralized, *reliable*, and convenient campus app. Features addressing pain points (accurate bus info, automated timetable, library status, event consolidation, safety reporting, basic utilities like Lost & Found) are highly desired.
- **Improving the Overall Student "Experience":** A high-quality, integrated, and *usable* campus app significantly enhances daily student life by saving time, reducing frustration, centralizing information, and improving access to services. This directly impacts student satisfaction and reflects positively on the institute's digital ecosystem.
- **Creating a Single Source of Truth & Communication Hub:** OneIITP has the potential to become the definitive platform for essential campus information (schedules, verified contacts, official events, potentially announcements), reducing reliance on fragmented, unofficial, or hard-to-track communication channels, provided the information is accurate and accessible.
- **Enhancing Campus Safety & Support:** Implementing features like the improved 'Report Harassment' function provides a critical, accessible, and potentially more trusted channel for students needing support, demonstrating a tangible commitment to student safety and well-being.
- **Driving Efficiency for STC & Administration:** Features like a streamlined issue reporting system (COM-03) or a centralized event posting mechanism (COM-01) can automate and improve processes, freeing up STC/Admin time. Utilities like Lost & Found (CORE-07) could centralize currently fragmented processes.
- **Building a Platform for Future Innovation:** A stable, well-adopted, and *maintainable* app serves as a robust foundation upon which future STC initiatives, administrative services, community features, or specialized modules can be built and deployed more effectively and sustainably.
- **Invaluable Experiential Learning:** The project continues to provide unparalleled real-world experience in product management, software development (cross-platform frameworks like React Native), UI/UX design, API integration, testing, and team collaboration for the student volunteers involved in STC.

Choosing not to invest further risks rendering past efforts obsolete, perpetuating user frustration, and leaving a significant gap in the campus digital ecosystem. Addressing the core issues identified in the RCA and delivering key integrations, usability improvements, and safety features can transform OneIITP from a liability into an essential and valued tool for the IIT Patna community.

---

4. Competitor Analysis (InstiApp IITB)

Sheets Link for Competitor Analysis Table
Figma Link for Detailed Analysis

---

5. Goals for this Enhancement Phase (M2)

1. **Achieve Rock-Solid Reliability (MVE):** Eradicate major sources of instability (crashes, freezes), fix critical bugs (update loop, login failures, core navigation flaws), and significantly improve performance across platforms to establish a trustworthy foundation.
2. **Deliver High-Impact Utility & Convenience:** Integrate automated timetable fetching (from ERP) and library account information. Ensure bus schedule data is accurate and reliably updated. Implement the essential 'Report Harassment' feature and core campus utilities (Lost & Found, Buy/Sell, Cab Share).
3. **Reduce User Friction & Improve Usability:** Overhaul the user onboarding experience. Streamline profile QR code management and eliminate intrusive reminders. Ensure clear, consistent navigation (with visible back buttons). Redesign cumbersome UI elements (forms, modals) using native patterns. Implement Google Calendar synchronization. Improve core interaction flows (e.g., contact dialing).

4. **Increase Engagement & Habituation:** Drive weekly active usage (WAUU) by making the app indispensable for core, recurring student tasks (checking schedule, library, events, utilities) and providing timely, relevant (but non-fatiguing) notifications. Make the app feel like a natural part of the daily workflow.
5. **Enhance Platform Maintainability & Scalability:** Address accumulated technical debt within the cross-platform framework (React Native). Adopt improved development practices (documentation, code reviews, CI/CD) to facilitate smoother handovers and future development by rotating student teams. Ensure backend systems can support new features and increased load.

---

6. North Star Metric & Success Metrics

⭐ **North Star Metric (NSM): Weekly Active Utility Users (WAUU)**

- **Definition:** The number of unique users who open the app AND perform at least one core utility, engagement, or significant interaction action within a 7-day period.
- **Core Actions Include:** Viewing automated timetable, checking library info, checking *accurate* bus status, successfully submitting a harassment report, successfully reporting a bug/issue, viewing verified event details, successfully adding an event to calendar, adding/viewing personal To-Dos (if implemented), posting/viewing Lost & Found item, posting/viewing Buy/Sell item, posting/viewing Cab Share request, spending more than 5 minutes of cumulative screen time within the week.
- **Rationale:** This revised NSM still focuses on value delivery and habit formation. It emphasizes the use of features that solve real problems or provide significant convenience/utility, moving beyond just opens or basic navigation. Success means the app is integrated into the student's weekly routine for important tasks and interactions.

- **Key Success Metrics (Supporting NSM & Goals):**
  - **Adoption & Engagement:**
    - Increase WAUU (Target: +100% within 6 months post-MVE stabilization & key integrations launch).
    - Increase DAU/MAU ratio (Target: >30%, indicating improved stickiness - making it habit-forming).
    - Increase adoption rate of key features: % of WAUU using Automated Timetable (>60%), Library Info (>40%), Lost & Found (>15%), Buy/Sell (>10%), Report Harassment (measure submissions, qualitative feedback), GCal Sync (>25% if implemented).
    - Increase Notification CTR (Click-Through Rate) for relevant class/event reminders (if basic notifications remain). Track opt-out rate for master toggle.
    - *(Note: NPS tracking deemed unfeasible/premature for M2).*
  - **Performance & Stability:**
    - Crash-free session rate (Combined Android & iOS) > 99.7%.
    - Average App Load Time (Cold Start, P90, Android & iOS) < 1.5 seconds (excluding splash screen time, if splash screen duration is fixed). Aim for perceived instant loading post-splash.
    - Play Store / App Store Rating > 4.3.
  - **Task Success & Satisfaction:**
    - Reduction in user-reported bugs related to performance, data accuracy, login, navigation, usability by >80% (via COM-03).
    - High Task Completion Rate for: viewing timetable, checking library info, submitting harassment/bug reports, posting Lost/Found item, adding event to calendar.
    - Positive shift in qualitative feedback (App Store reviews, direct feedback) themes – less focus on bugs/performance/usability, more on feature utility/requests.
    - High satisfaction score on targeted user surveys focusing on usability and reliability post-launch.
  - **Retention:**

        - Improvement in Week 1 retention rate for new users > 50% (post-onboarding overhaul).
        - Improvement in Week 4 retention rate for new users > 30%.
    - **Safety Feature Metrics:**
        - Track number of harassment reports submitted (anonymized count).
        - Track usage of "Call Helpline" button (direct dial and via SAFE-02 double-tap).
        - Gather qualitative feedback on the ease of use, perceived safety, and accessibility of the reporting feature and emergency button.

---

7. Potential Metrics Watch Out / Impact

- **Increased Server Load:** New integrations (ERP, Library, GCal Sync), notifications, utility features (L&F, B&S, Cab Share), To-Do backend, and harassment report handling will significantly increase backend load and complexity. Requires robust backend development and scaling.
- **API Dependency Risk:** High reliance on internal IITP APIs (ERP, Library, Auth) which may have limitations, instability, or require significant negotiation/collaboration. Need fallback strategies. GCal API has its own quotas/requirements.
- **Cross-Platform Consistency (React Native):** While aiming for consistency, ensuring truly native feel and performance, especially for UI elements (native modals/pickers) and platform-specific interactions, requires careful implementation and testing within the React Native framework. Performance tuning might differ across platforms.
- **Student Developer Turnover:** This remains a critical risk. Excellent documentation (code, architecture, APIs, component library), modular architecture, CI/CD, and strong knowledge transfer processes are paramount for the React Native codebase.
- **Privacy & Security (Multiple Features):**
    - *Harassment Reporting:* Requires extreme care (anonymity, secure transmission, access control). High risk.
    - *Utilities (L&F, B&S, Cab Share):* Need clear guidelines on data sharing (e.g., contact info), content moderation policies, and potentially user reporting mechanisms for inappropriate content/scams.
- **Scope Creep:** With many desired features, maintaining focus on the MVE (stability, core integrations, safety, core utilities, basic UX fixes) before expanding is crucial. Prioritization must be strict.
- **Notification Fatigue:** Even with a master toggle (per ACA-06 change), the re-engagement notifications for lapsed users (COM-01) must be used judiciously and potentially offer a separate opt-out if they prove annoying.

---

8. DACI (Decision-Making Framework)

- **Driver (Accountable):** STC Technical Head / Designated Student Product Lead. (Drives the product vision, requirements, prioritization).
- **Approver(s) (Accountable):**
    - STC Council Leadership (Overall approval, resource allocation).
    - Faculty Advisor (If applicable, for guidance and oversight).
    - Dean of Student Affairs / Relevant Committee Head (Specifically for Harassment Reporting feature policy & workflow approval; potentially guidelines for utility features like B&S).
    - Head of IT Dept (For critical API access/integration approvals).
- **Consulted (Responsible/Consulted):**
    - Student Development Team Leads (Cross-Platform/Backend) (Technical feasibility, effort estimation, implementation details).
    - Student UI/UX Lead(s) (Design, usability, user flow, native component guidelines).

    - IITP IT Dept Liaisons (API details, authentication, security constraints).
    - Library / Transport Section Liaisons (Data accuracy, specific requirements).
    - Wellness Centre / Relevant Committee (Input on Harassment Reporting workflow, helpline info).
    - Campus Security (Input on Emergency button functionality, potentially L&F processes).
    - Student Focus Groups / Beta Testers (User feedback, usability testing).
- **Informed (Informed):** General Student Body, IIT Patna Administration (Wider), Other Clubs/Societies.

---

9. User Segmentation (Done using real users)

- *The Newbie Fresher*
- *The Engaged User (Platform Agnostic now)*
- *The Frustrated User (Focus on general performance/usability, not just iOS)*
- *The Disengaged Skeptic*
- *The Safety-Conscious User*

User Segmented Personas OneIITP (Access by clicking here)

---

10. Agile Epics Overview

We will be using Agile Methodology to build and ship M2.

> Here's what that means: The epic is broken down into smaller pieces of work. Your team may call these smaller pieces product backlog items, user stories, issues, or something else. As conditions or customer requirements change over time, these smaller pieces can be modified, removed, or added to a team's product backlog with each sprint.
>
> *- Scrum Alliance Inc, Copyright © 2025*

1. **EPIC-CORE:** Core Experience, Performance & Utilities *(Expanded Scope)*
2. **EPIC-BUS:** Campus Bus Utility Enhancement
3. **EPIC-ACADEMICS:** Academic & Personal Productivity Integration *(Attendance Updated)*
4. **EPIC-ONBOARD:** Onboarding & User Education Overhaul
5. **EPIC-PROFILE:** Profile Management & QR Code
6. **EPIC-COMMUNITY:** Events, Engagement & Support
7. **EPIC-SAFETY:** Safety & Reporting Features
8. **EPIC-UX:** User Experience Refinement *(New Epic)*
9. **EPIC-MISC:** Miscellaneous Features & Fixes *(New Epic)*

---

11. Product Requirements (The What) - Detailed Breakdown

EPIC-CORE: Core Experience, Performance & Utilities

Focus: Fixing fundamental reliability, speed, navigation, and implementing core campus utility features.

Req CORE-01: App Stability Enhancement

- **Detailed Description:** Identify, diagnose, and fix the root causes of frequent application crashes, freezes, and unexpected terminations reported across platforms. Analyze crash logs (e.g., Requires Firebase Crashlytics). Goal is reliable usability.

- **Goals:** Reduce user frustration; Build trust; Establish stable baseline.
- **User Stories:**
  - *As a student needing info,* I want the app stable so I can rely on it.
  - *As a frequent user,* I want responsiveness, not freezes.
  - *As an STC developer,* I need robust crash reporting for quick fixes.
- **Functional Requirements:** Implement comprehensive crash reporting; Systematically address top crash/ANR causes; Test thoroughly on various OS versions/devices.
- **Non-Functional Requirements:** Target combined crash-free session rate > 99.7%;
- **Acceptance Criteria:** Crash rates meet target; User feedback on crashes reduces >80%; No stability regressions.
- **Priority: Must Have (MVE)**
- **Dependencies:** Crash reporting tools access.

Req CORE-02: Performance Optimization (Load/Transition Times)

- **Detailed Description:** Improve perceived and actual app speed: reduce launch time (cold/warm start), speed up screen transitions, ensure smooth scrolling in lists (events, contacts, utilities). Optimize API calls, database queries, image loading/caching, UI rendering (React Native optimization), and profile code for bottlenecks. Address artificial splash screen delay if applicable.
- **Goals:** Responsive/fluid feel; Reduce waiting; Improve UX quality.
- **User Stories:**
  - *As a student checking schedule,* I want instant load.
  - *As a user browsing lists,* I want smooth scrolling.
  - *As any user,* I want immediate tap response.
- **Functional Requirements:** Profile launch times/transitions; Optimize network requests (caching, compression); Optimize list rendering (e.g., FlatList optimization); Review database/async storage access; Optimize React Native bridge communication if needed.
- **Non-Functional Requirements:** Average Cold Start Time (P90) < 1.5s (post-splash); Smooth screen transitions (~60fps); P95 latency for critical APIs < 1s.
- **Acceptance Criteria:** Performance metrics met; User feedback confirms speed improvement; Load indicators used appropriately.
- **Priority: Must Have (MVE)**
- **Dependencies:** Backend API performance.

Req CORE-03: Update Mechanism Fix

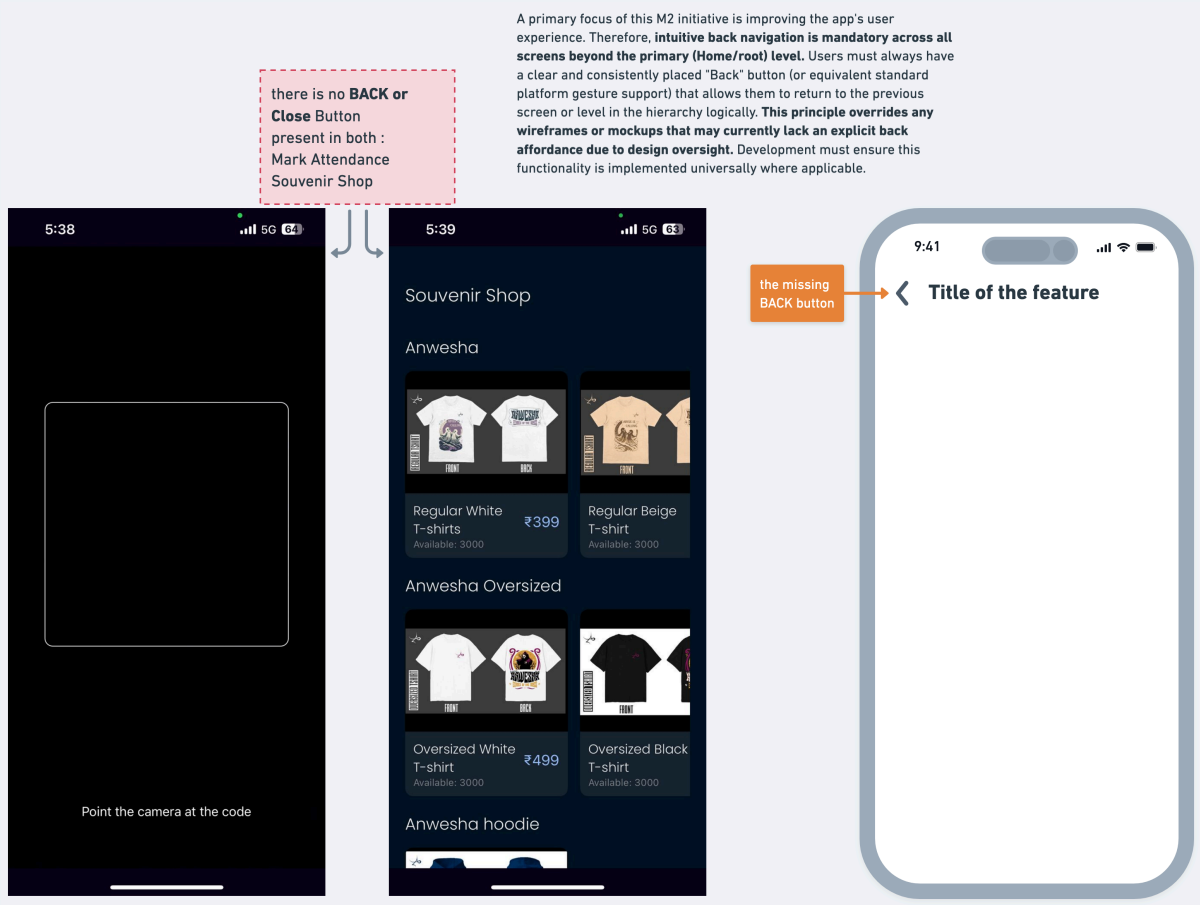
- **Detailed Description:** Fix the issue where the app forces/prompts an update, but none is available on the store. Correct version checking logic to accurately compare installed vs. latest published version. Implement robust error handling and clear user messaging.
- **Goals:** Eliminate major blocker; Ensure smooth update experience.
- **User Stories:**
  - *As a user,* I don't want false update blocks.
  - *As a user needing update,* I want correct store direction.
  - *As STC admin,* I want reliable version checks.
- **Functional Requirements:** Implement reliable store version fetching; Accurate version comparison; Trigger prompt only if update available/published; Clear optional/mandatory prompt; Direct store link; Graceful network error handling for version check.
- **Acceptance Criteria:** "Update loop" resolved; Version check logic accurate; Correct store redirection; App usable if check fails (non-critical).

- **Priority: Must Have (MVE)**
- **Dependencies:** Ability to query Play Store/App Store version info.

(We have skipped CORE-04 due to Out Of Scope Issues, does not affect anything)

Req CORE-05: Basic Navigation Consistency (Back Button)

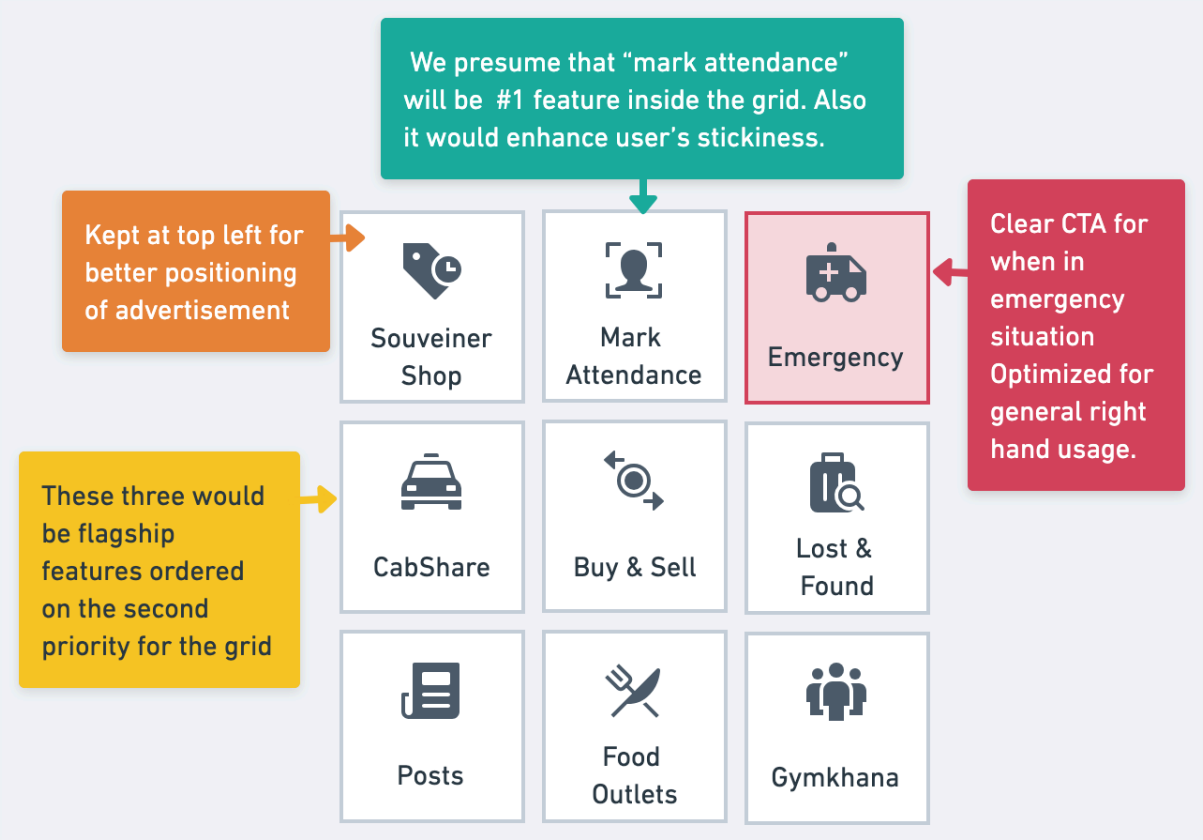
- **Detailed Description:** Ensure consistent and predictable navigation, especially fixing inconsistent back button behavior. Ensure a clear, visible back button/affordance is present on non-root screens. Review overall navigation structure (bottom tabs, etc.) for clarity.
- **Goals:** Intuitive app movement; Eliminate navigation frustration; Consistent interaction model.
- **User Stories:**
  - *As any user,* I expect back navigation to work logically.
  - *As any user,* I want to easily understand main navigation.
  - *As a user deep in a feature,* I want a clear way back.
- **Functional Requirements:** Ensure back actions (hardware/software) work predictably (go up hierarchy/back in history); Implement consistent, visible back buttons (< icon) on relevant screens; Review/refine main navigation (bottom bar).
- **Acceptance Criteria:** Back navigation works predictably; Main navigation is smooth/consistent; Users can easily navigate into/out of detail views; User feedback confirms clear navigation.
- **Priority: Must Have (MVE)**
- **Dependencies:** Clear definition of navigation hierarchy; UI/UX design for back button placement.



Back Button Missing

Req CORE-06: Home Screen Grid Order Configuration

- **Detailed Description:** Implement a system to allow authorized STC administrators to manually define the default order and visibility of feature shortcuts displayed in the main Home Screen grid (which includes items like Schedule, Bus, QR Code, Events, Lost & Found, etc.). This configuration should be manageable via a backend interface. The primary goal is to give STC control over the **standard layout** presented to all users. This is about setting a deliberate default layout, *not* about dynamic or time-based changes.
- **Goals:**
  - Allow STC to curate the default Home Screen layout based on their understanding of feature importance and general student needs.
  - Provide administrative control over which features are presented on the main grid and in what order.
  - Ensure a consistent default Home Screen experience for all users, as defined by STC.
- **User Stories:**
  - *As an STC admin,* I want to set a specific, fixed order for the home grid buttons (e.g., Schedule first, Bus second) that reflects our view of the most essential functions for students.
  - *As an STC admin,* I want the ability to hide a less critical feature from the main grid if needed, without removing the feature entirely from the app.
  - *As a student,* I expect the main shortcuts on the home screen to be organized logically and consistently based on general utility.
- **Functional Requirements:**
  - **Backend Configuration:** Develop a backend endpoint and an associated admin interface (e.g., simple web panel) allowing authorized STC admins to:
    - Define the list of available grid items (features).
    - Manually set a specific display order for these items.
    - Manually toggle the default visibility (show/hide) for specific items.
  - **Frontend Implementation:**
    - The app fetches this grid configuration (ordered list of items with visibility flags) from the backend upon app launch or refresh. This configuration represents the *single default layout* for all users.
    - The Home Screen grid UI dynamically renders the items based *only* on this fetched default configuration (respecting order and visibility).
    - (Layout details like number of columns, icon design etc. are based on overall UI design).
- **Acceptance Criteria:**
  - Home Screen grid items render dynamically in the precise order and visibility specified by the backend configuration.
  - Changes made to the order/visibility in the backend admin interface are reflected in the app (for all users) after the next data refresh or app launch.
  - The fetching and rendering process is performant (respecting CORE-02 targets).
  - The admin interface for managing the configuration is functional and usable by authorized STC personnel for setting the default layout.
- **Priority: Should Have** (Provides necessary administrative control over core layout)
- **Epic:** EPIC-CORE
- **Dependencies:** Backend endpoint and admin interface for configuration management; Basic Home Screen grid UI structure exists; Defined list of features potentially includable in the grid.



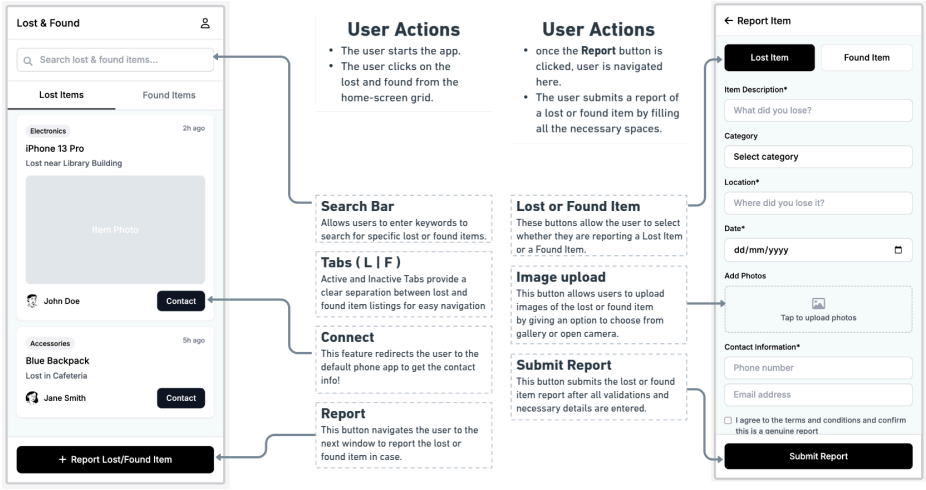
Homepage Grid Wireframe

Req CORE-07: Lost & Found Feature

- **Detailed Description:** Implement a dedicated "Lost & Found" section where students can post items they have lost or found on campus. Users should be able to create posts with details like item description, location (lost/found), date, an optional image, and their contact preference (e.g., display contact info, or require interested parties to message via an intermediary system if privacy is paramount - *initial version likely displays contact info*). Users should be able to browse/search listings.
- **Goals:** Centralize lost and found reporting; Help students recover lost items; Reduce reliance on scattered physical notices or social media posts.
- **User Stories:**
  - *As a student who lost my ID card,* I want to post a description, where I think I lost it, and my contact number in the app's Lost & Found section, hoping someone finds it.
  - *As a student who found a water bottle,* I want to post its description, location found, and maybe a picture, so the owner can contact me.
  - *As any student,* I want to easily browse or search recent Lost & Found postings.
- **Functional Requirements:**
  - Create "Lost & Found" section with options to "Report Lost Item" and "Report Found Item".
  - Form for posting: Item Description (required), Category (optional, e.g., Electronics, Stationery, Clothing), Location Lost/Found (required), Date Lost/Found (required), Image Upload (optional), Contact Information (e.g., Phone number/email - user enters, visibility confirmed).
  - Store postings in a backend database.
  - Display separate lists/tabs for "Lost Items" and "Found Items", sorted chronologically (newest first).
  - Implement basic keyword search within listings.

- Allow users to mark their own posts as "Resolved" or "Closed" (or delete them).
- Implement basic content moderation guidelines/reporting mechanism for inappropriate posts (link to COM-03 or separate).

- **Acceptance Criteria:** Users can successfully create Lost and Found posts with required details and optional image; Posts are displayed correctly in respective lists; Search function works; Users can mark their posts as resolved/delete them; Contact information is displayed as intended.
- **Priority: Should Have** (Core Utility)
- **Dependencies:** Backend infrastructure for storing/serving posts; Image storage solution; UI/UX design for forms and listings; Content moderation policy.



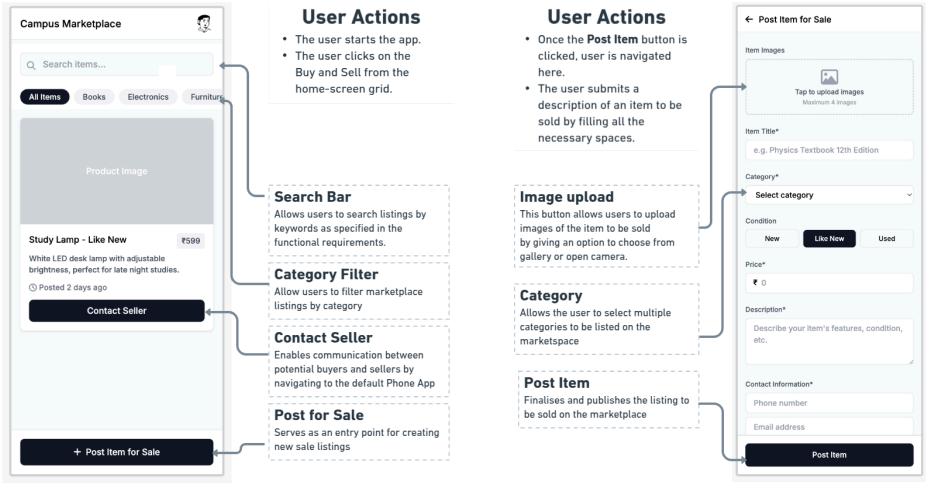
Lost And Found Wireframes

Req CORE-08: Buy & Sell Feature

- **Detailed Description:** Implement a dedicated "Buy & Sell" section for students to post items they want to sell or buy within the campus community (e.g., used books, lab coats, cycle parts, furniture). Users create posts with item description, price, category, optional image, and contact information. Other users browse/search listings and contact sellers directly.
- **Goals:** Facilitate peer-to-peer commerce within the campus; Provide a centralized marketplace for common student needs; Reduce reliance on external platforms or fragmented groups.
- **User Stories:**
  - *As a student finishing a course,* I want to sell my used textbook by posting its details, condition, price, and my contact info in the app.
  - *As a student looking for a cheap study lamp,* I want to browse the "Buy & Sell" section or search for lamps to see if anyone is selling one.
  - *As a user,* I want to easily contact the seller if I'm interested in an item.
- **Functional Requirements:**
  - Create "Buy & Sell" section with options like "Post Item for Sale" / "Post Buying Request" (Optional V2). Focus on Selling first.
  - Form for posting sale item: Item Title/Description (required), Category (e.g., Books, Electronics, Furniture, Other), Expected Price (required), Condition (optional), Image Upload (optional), Contact Information (required, e.g., phone/email).
  - Store postings in a backend database.
  - Display listings, potentially filterable by category, sorted chronologically.
  - Implement basic keyword search.

- Allow users to mark their posts as "Sold" or delete them.
- Ensure contact information (phone/email) is easily accessible/tappable on listings.
- Implement basic content moderation guidelines/reporting mechanism for inappropriate items/scams.

- **Acceptance Criteria:** Users can successfully post items for sale with details, price, image; Listings are displayed correctly with filtering/search; Contact info is accessible; Users can manage their posts (mark as sold/delete).
- **Priority: Should Have** (Core Utility)
- **Dependencies:** Backend infrastructure; Image storage; UI/UX design; Content moderation policy.



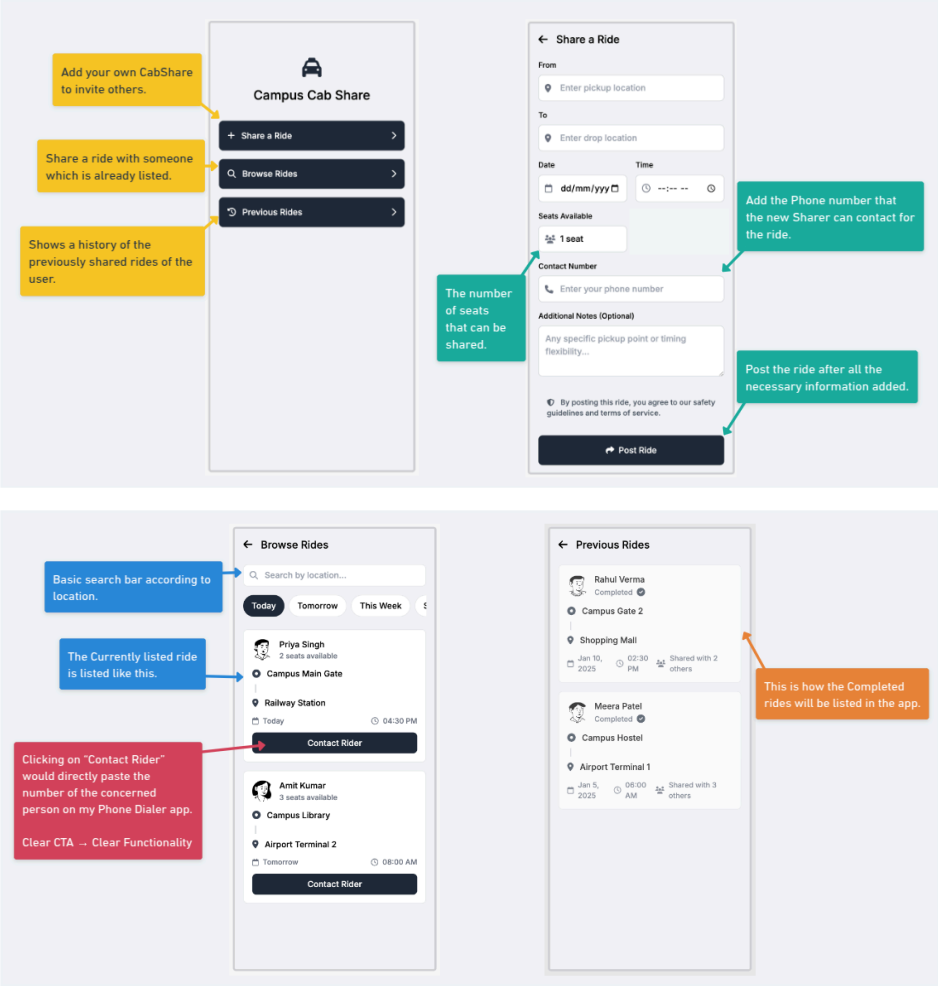
Buy And Sell Mockup

Req CORE-09: Cab Sharing Feature

- **Detailed Description:** Implement a "Cab Share" section allowing students to post requests for sharing cabs/taxis for common routes (e.g., to/from airport, railway station, city locations) on specific dates/times. Users post their travel plans (Origin, Destination, Date, Time, Number of seats needed/available) and contact information. Other users can browse requests and contact posters directly to coordinate sharing.
- **Goals:** Facilitate cost-effective travel for students by enabling cab pooling; Reduce individual travel costs; Centralize coordination for cab sharing.
- **User Stories:**
  - *As a student needing to go to the airport next Friday morning,* I want to post a request in the app looking for 1-2 others to share a cab, specifying the time and my contact details.
  - *As a student flexible with my travel time to the station,* I want to browse existing cab share requests for today/tomorrow to see if I can join someone and save money.
  - *As a user finding a matching request,* I want to easily contact the original poster to arrange the share.
- **Functional Requirements:**
  - Create "Cab Share" section.
  - Form for posting request: Origin (e.g., Campus Gate 1, Airport), Destination (e.g., Airport, Patna Jn), Date of Travel, Time of Travel (approximate window possible), Seats Needed / Seats Available in existing booking, Contact Information (required, phone/email).
  - Store requests in a backend database.
  - Display active requests, filterable by Date, Origin/Destination (optional). Sort chronologically by travel date/time.
  - Implement basic keyword search (e.g., location names).
  - Allow users to delete their requests once fulfilled or no longer needed.

- Ensure contact information is easily accessible/tappable on listings.
- Include a disclaimer regarding safety/verification (users share at own discretion).

- **Acceptance Criteria:** Users can successfully post cab share requests with necessary details; Requests are displayed clearly with filtering/search; Contact info is accessible; Users can delete their posts.
- **Priority: Should Have** (Core Utility)
- **Dependencies:** Backend infrastructure; UI/UX design; Clear disclaimer text.



Cab Share Wireframe

EPIC-BUS: Campus Bus Utility Enhancement

Focus: Making the bus information feature reliable and useful.

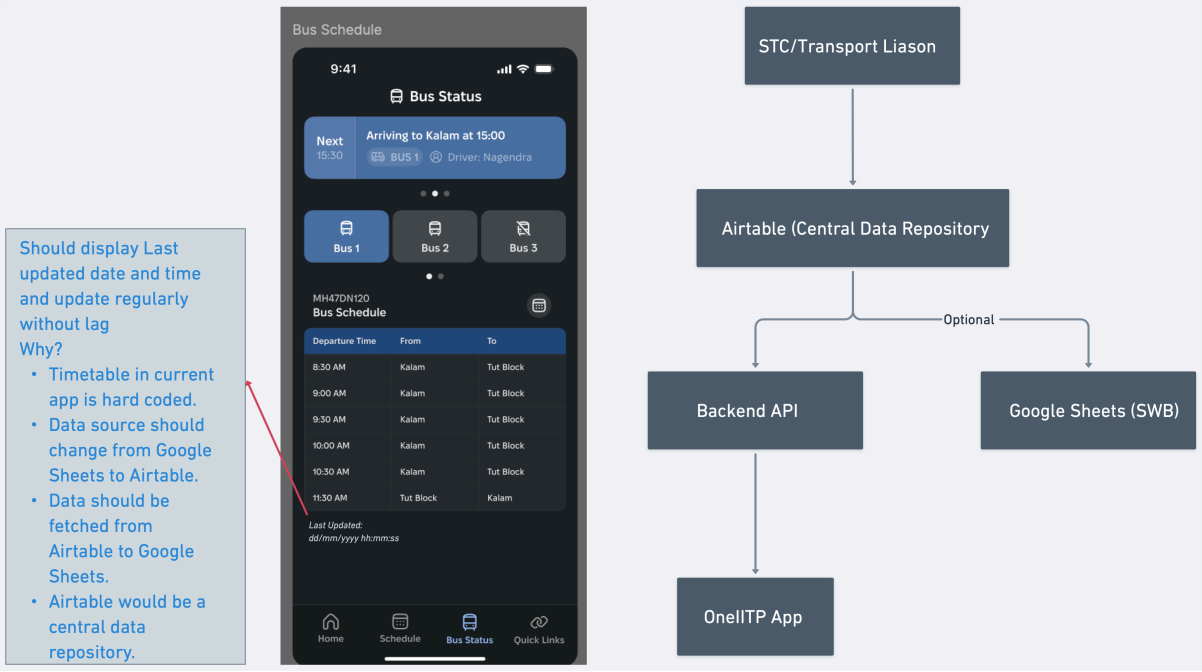
Req BUS-01: Bus Schedule Accuracy via Airtable & Reliable Updates

- **Detailed Description:** Implement a robust system to ensure the bus schedule displayed in the OneIITP app is accurate and reflects the latest official timings, addressing the current issue of hardcoded or unreliably updated data. **Airtable will serve as the single source of truth** for bus schedule information, managed and updated by authorized STC members or Transport liaisons. The OneIITP app will fetch the schedule data directly from this Airtable base via its API. Optionally (for existing workflows or broader visibility), data from Airtable can also populate the existing "SWB Sheet" Google Sheet. The app *must* prominently display a "Last Updated" timestamp corresponding to the data fetched from Airtable.

- **Rationale (The Why):**
  - The current hardcoded timetable is unsustainable and inaccurate.
  - Leveraging Airtable provides a structured, easily manageable database-like interface for STC/Transport to maintain the schedule, superior to direct Google Sheet editing for consistency and API access.
  - Using Airtable as the central repository ensures data consistency across different potential consumers (the app, potentially Google Sheets).
  - Direct API fetching from Airtable by the app is more reliable and structured than parsing a Google Sheet.
- **Goals:**
  - Establish Airtable as the definitive, easily maintainable source for accurate bus schedules.
  - Ensure the OneIITP app always displays the current, authoritative schedule fetched from Airtable.
  - Restore user trust by providing consistently reliable bus information.
  - Eliminate user complaints about outdated bus timings.
  - Provide transparency about data freshness via the "Last Updated" timestamp.
- **User Stories:**
  - *As a student relying on the bus,* I want the app to show the timings directly sourced from the official Airtable base, so I know it's the most current version available.
  - *As an STC member responsible for updates,* I want to update the bus schedule in one central place (Airtable) using its user-friendly interface, knowing it will automatically reflect in the app and potentially the SWB Google Sheet.
  - *As any user,* I want to see the "Last Updated" timestamp in the app, reflecting when the data was last fetched or modified in Airtable, so I can gauge its freshness.
- **Functional Requirements:**
  - **Airtable Base Setup:** Design and set up an Airtable base with appropriate tables and fields to store the bus schedule logically (e.g., Route Name, Stop Name, Day Type (Weekday/Weekend), Departure Time, Bus Number (if applicable)). Define clear data entry guidelines.
  - **Airtable API Integration:**
    - Configure secure API access to the Airtable base (using API keys/tokens).
    - Implement logic in the OneIITP backend (or directly in the app, though backend mediation is often better) to fetch schedule data from the relevant Airtable table(s) via the Airtable API.
    - Handle data parsing and formatting for display in the app.
  - **App Display:**
    - Display the full, fetched schedule clearly within the app's bus section.
    - Prominently display a "Last Updated" timestamp. This could be the timestamp of the last successful fetch *or* preferably, a 'last modified' timestamp retrieved from Airtable itself, if available via the API, for greater accuracy.
  - **Data Refresh:** Implement a mechanism for the app/backend to periodically refresh the schedule data from Airtable (e.g., every few hours, on app launch, or via manual refresh button).
  - **(Optional) Airtable to Google Sheets Sync:** If maintaining the SWB Google Sheet is still required, implement a mechanism (e.g., using Airtable Automations, Zapier, or a custom script) to automatically sync data *from* the master Airtable base *to* the SWB Google Sheet (read-only view preferably).
  - **Error Handling:** Gracefully handle errors during API fetching (e.g., network issues, Airtable API limits, invalid data), potentially showing cached data with a warning or an explicit error message.
- **Acceptance Criteria:**
  - The bus schedule displayed in the app accurately matches the data currently present in the master Airtable base.
  - The "Last Updated" timestamp is clearly visible and accurately reflects the data's freshness (either fetch time or Airtable modification time).

- The process for STC/Transport to update the schedule in Airtable is documented and functional.
- Changes made in Airtable are reflected in the app within the defined refresh interval or upon manual refresh.
- (If implemented) The SWB Google Sheet accurately reflects the data in Airtable after sync.
- User complaints about outdated bus timings reduce by >90%.

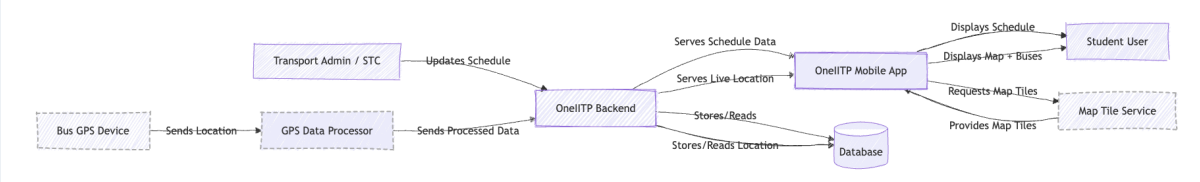
- **Priority: Must Have (MVE)**
- **Epic:** EPIC-BUS
- **Dependencies:** Airtable account and base setup; Airtable API Key; Backend or app logic to integrate with Airtable API; Clear process agreed upon with STC/Transport for maintaining the Airtable base.



Bus Last Updated Wireframe

Req BUS-02: Bus GPS Tracking (Live Location)

- **Detailed Description:** Implement real-time GPS tracking for campus buses. Requires GPS device installation/integration, backend system, and displaying live locations on an in-app map, updated frequently, with clear bus identification. Handle data unavailability gracefully.
- **Goals:** Precise real-time location info; Reduce waiting uncertainty; Major convenience feature.
- **User Stories:** *As a waiting student,* I want to see bus location on map. *As a student heading to stop,* I want to check location to gauge time. *As user unsure of bus,* I want clear identification.
- **Functional Requirements:** Integrate map view; Receive/display real-time GPS coords as moving icons; Update frequently (15-30s); Label buses; Show user location (permission needed); Handle signal loss gracefully.
- **Non-Functional Requirements:** Location accuracy ~50m; Responsive map; Optimized battery use.
- **Acceptance Criteria:** Live locations accurate; Updates automatic/frequent; Buses identifiable; Handles signal loss; Performance acceptable.
- **Priority: Could Have** (Moved down due to complexity/dependencies)
- **Dependencies:** GPS hardware/installation/availability; Backend infra; GPS data API; Potential costs.



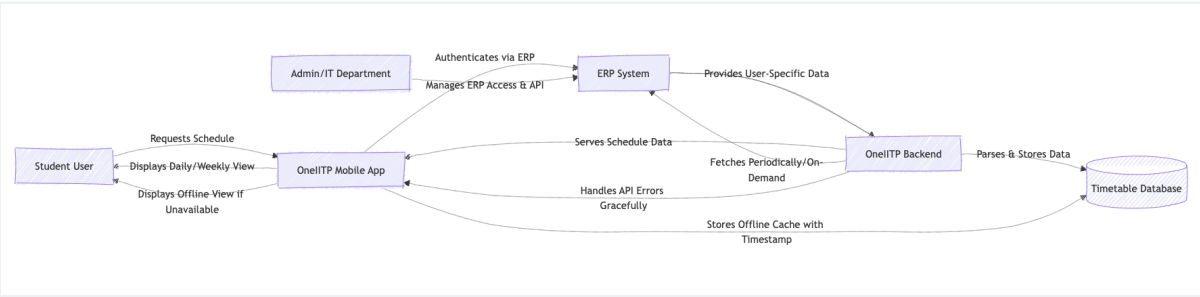
High Level Diagram for Live Bus Tracking

EPIC-ACADEMICS: Academic & Personal Productivity Integration

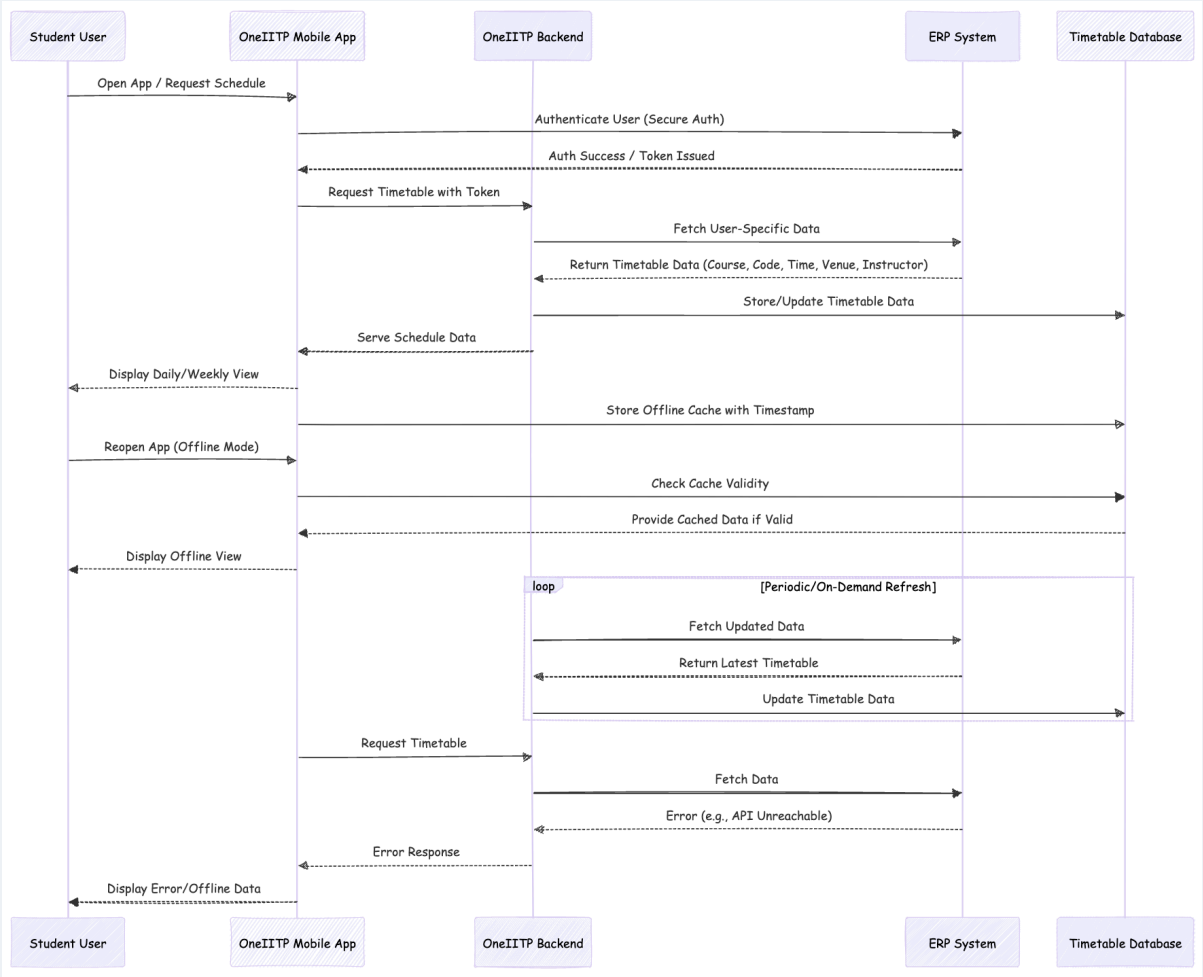
Focus: Making the app indispensable for core academic tasks and adding personal productivity tools.

Req ACA-01: Automated Timetable Integration (via ERP API)

- **Detailed Description:** Eliminate manual timetable entry via ERP integration. Securely authenticate, fetch/display personalized, up-to-date schedule (course, code, time, venue, instructor). Handle groups/electives/changes. Clear daily/weekly view.
- **Goals:** Effortless access to official timetable; Eliminate manual entry friction; Central hub for academic planning; Increase daily utility/stickiness.
- **User Stories:**
  - *As a student,* I want my schedule automatically.
  - *As student with electives,* I want only my courses. *As user,* I want changes reflected automatically. *As user planning,* I want easy daily/weekly view.
- **Functional Requirements:** Secure ERP auth; Fetch user-specific data; Parse/display clearly (daily/weekly view); Include all details; Handle complex structures; Refresh periodically/manually; Offline cache with timestamp; Graceful API error handling.
- **Acceptance Criteria:** Matches ERP schedule; All data present/legible; Handles complexity; Updates reflected timely (target < 6-7h/refresh); Offline view works; Performant.
- **Priority: Should Have** (High Impact, complex dependency)
- **Dependencies:** CRITICAL: Reliable/documented secure ERP API; IT Dept collaboration.



Technical High Level System Diagram for ACA-01 via ERP API



Sequence Diagram for TimeTable Integration

Req ACA-02: Central Library Integration (Issued Books, Due Dates, Notifications)

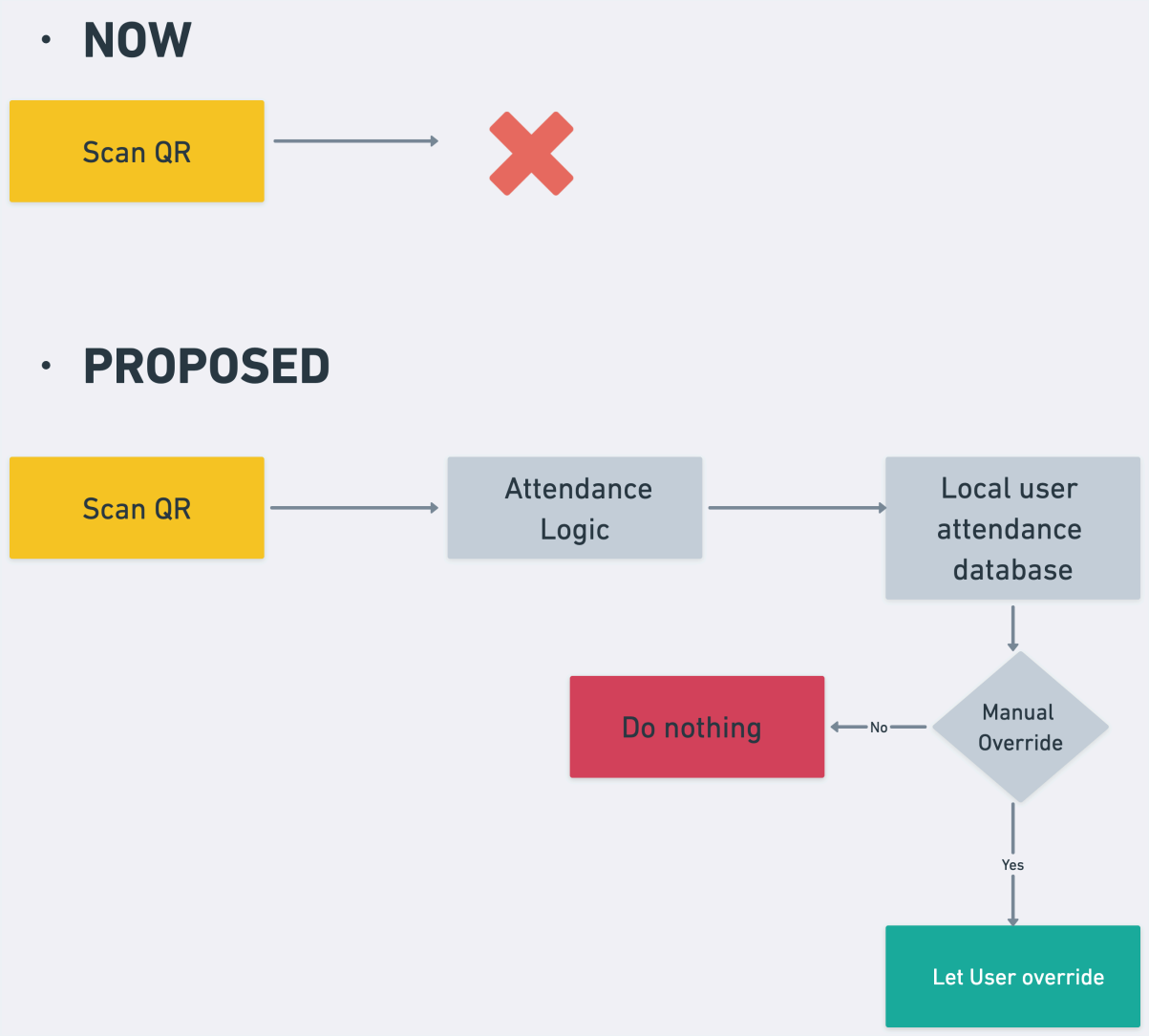
- **Detailed Description:** Integrate with Library Management System (LMS). Show issued books, due dates, fines. Implement *opt-in* push notifications for upcoming due dates (1-2 days prior).
- **Goals:** Convenient access to borrowing status; Help avoid fines via reminders; Increase academic utility.
- **User Stories:**
  - *As student borrower*, I want quick status check in-app.
  - *As forgetful student*, I want due date notifications.
  - *As user*, I want to see outstanding fines.
- **Functional Requirements:** Secure LMS API auth; Fetch user's issued books data; Display Title, Author (opt), Due Date, Fine (opt); Display total fines; Background job for due date check; Send push notifications; Setting to enable/disable library notifications; Graceful API error handling; Ensure privacy compliance.
- **Acceptance Criteria:** Matches library record; Fines accurate; Opt-in notifications reliable/timely; Setting works; Performant.
- **Priority: Should Have** (High Impact, complex dependency)
- **Dependencies:** CRITICAL: Reliable/documented secure LMS API; Library collaboration.

Library Management Wireframe

Req ACA-03: Integrate QR Scan with Existing Attendance Tracking

- **Detailed Description:** Integrate the existing QR code attendance scanning mechanism with the already implemented manual attendance tracking feature (which allows marking P/A, calculates percentages, and shows warnings). When a user successfully scans a valid class QR code, the app should automatically update the attendance status for that specific class session in the backend to 'Present' ('P'). Users should still retain the ability to manually override or edit this status later via the existing interface if needed (e.g., if they scanned by mistake or want to mark 'Leave' instead).
- **Goals:**
  - Provide an automated way ('QR Scan') to mark attendance as 'Present', reducing manual effort.
  - Ensure the automatic QR scan data seamlessly flows into the existing attendance tracking system.
  - Maintain user control by allowing manual overrides of QR-scanned entries.
  - Leverage existing percentage calculation and warning features with QR-sourced data.
- **User Stories:**
  - *As a student attending class,* I want to scan the class QR code quickly, and have the app automatically mark me as 'Present' for that session in my attendance tracker.
  - *As a student reviewing my attendance,* I want to see which sessions were marked 'Present' automatically via QR scan versus those I marked manually. (Optional: visual distinction).
  - *As a student who scanned the QR but later needed to apply for leave,* I want to be able to go back to that session in my schedule and manually change the status from 'Present' (scanned) to 'Leave'.
- **Functional Requirements:**
  - **QR Scan Integration:** Modify the existing QR scanning mechanism:
    - Upon successful scan of a valid class QR code, identify the corresponding user, course, and specific class session (date/time).

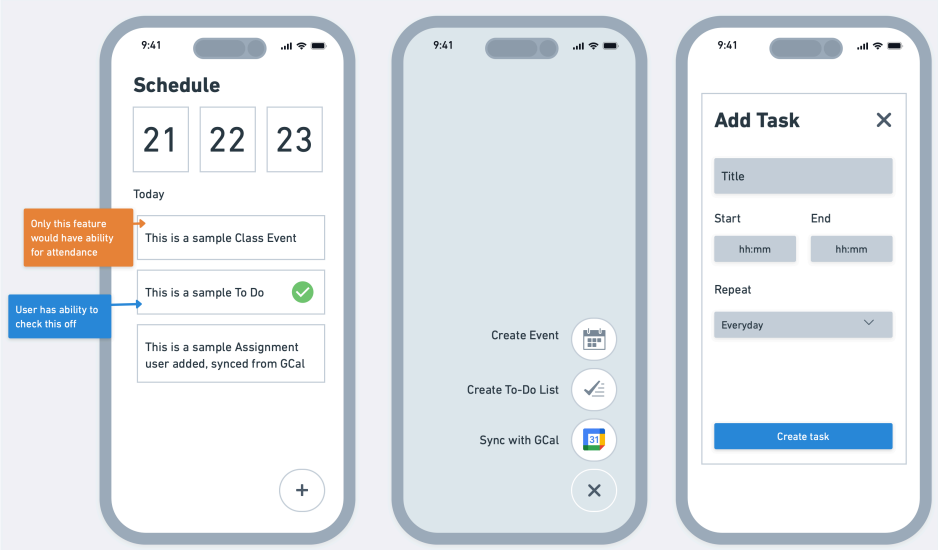
        - Trigger a backend update to set the attendance status for that specific user/session instance to 'Present' ('P').
    - **Visibility:** Ensure the 'Present' status set by a QR scan is correctly reflected in the existing schedule view/interface where users manually track attendance. (Optional: Consider a subtle visual indicator like a small QR icon next to the 'P' to show it was scanned automatically).
    - **Manual Override:** The existing functionality allowing users to manually select/edit the status (P/A for any session must remain functional and should override any status previously set by a QR scan for that session.
    - **Percentage Calculation:** Ensure the existing percentage calculation logic correctly incorporates the 'Present' status set by QR scans when determining the total number of 'P' sessions.
- **Acceptance Criteria:**
    - Successfully scanning a valid class QR code updates the attendance status for that session to 'Present' in the backend and is reflected in the UI.
    - The existing manual attendance marking interface remains functional.
    - Users can manually override a 'Present' status that was set by a QR scan (e.g., change it to 'A' or 'L').
    - The existing attendance percentage calculation correctly uses the 'Present' status from QR scans.
- **Priority: Should Have** (Integrates existing separate functionalities)
- **Epic:** EPIC-ACADEMICS
- **Dependencies:** Functional automated timetable (ACA-01); Existing QR scan mechanism operational; Existing manual attendance tracking system (P/A marking, percentage calculation, warnings) fully functional; Backend capable of storing/updating attendance status per session instance.



User Flow Diagram of Attendance Marking

Req ACA-04: Personal To-Do List with Calendar Integration

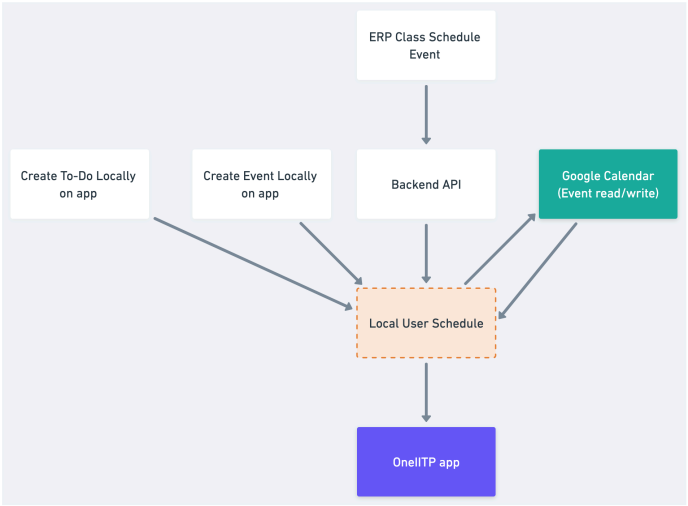
- **Detailed Description:** Introduce simple personal To-Do list. Add tasks with description, optional due date. View To-Dos alongside academic schedule in calendar/schedule view for unified planning.
- **Goals:** Convenient task management; Increase utility as daily planner; Delighter feature.
- **User Stories:**
  - *As student juggling tasks,* I want to add personal To-Dos in-app.
  - *As user planning,* I want To-Dos overlaid on my schedule.
  - *As user,* I want to mark tasks complete.
- **Functional Requirements:** Interface to add/view/edit/delete To-Dos (Desc required, Due Date opt); Secure backend storage; Mark complete (checkbox/swipe); Display date-based To-Dos in Schedule view (visually distinct); Persist data.
- **Acceptance Criteria:** CRUD operations work; To-Dos appear correctly in Schedule view; Visually distinct; Data persists.
- **Priority: Could Have** (Delighter Feature)
- **Dependencies:** Backend infra; UI/UX design.



To-Do List

Rep ACA-05: Google Calendar Synchronization (Two-Way Push)

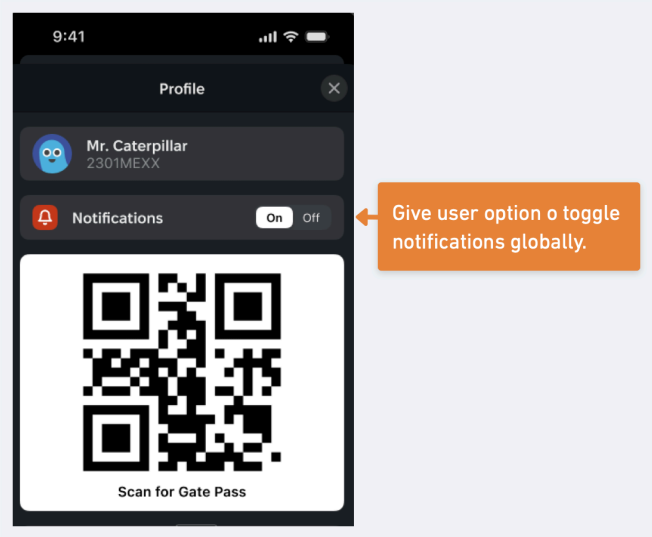
- **Detailed Description:** Allow users to connect Google Calendar account. Implement Phase 1 (Two-Way Push): Users can two-way sync Google Calendar inside OneIITP to access their classes, to-dos in GCal as well as in-app.
- **Goals:** Allow use of preferred GCal tool; Leverage GCal features (multiple repeat event, more flexibility in event making); Reduce manual re-entry.
- **User Stories:** *As GCal user,* I want my IITP schedule and GCal schedule to be synced together. *As user with To-Dos,* I want them synced too.
- **Functional Requirements (Phase 1):** Secure Google Auth (OAuth 2.0); Request read and write permission; UI option to initiate sync; Select target GCal; Format data correctly for GCal API; Use API to insert events; Handle errors/rate limits; Provide status feedback; Consider update mechanism (store Event IDs).
- **Acceptance Criteria (Phase 1):** Auth works; Export successful; Events correct in GCal; Status feedback clear; Complies with Google terms.
- **Priority: Should Have** (High value for GCal users)
- **Dependencies:** GCP project; GCal API libraries; OAuth knowledge; UI/UX; Requires ACA-01.



Calendar High Level System Diagram

Req ACA-06: Centralized Notification Toggle

- **Detailed Description:** Implement a single, master toggle switch in the app settings to enable or disable *all* push notifications sent by the OneIITP app (including potential class reminders, library alerts, event updates, lapsed user pings). Remove fine-grained controls for individual notification types for simplicity in M2. The backend logic for triggering different notification types (library, events, lapsed user) still needs to exist, but delivery is controlled by this single user-facing switch and OS-level permissions.
- **Goals:** Provide users with simple, ultimate control over receiving notifications; Reduce settings complexity for M2; Rely on OS permissions as primary gatekeeper.
- **User Stories:**
  - *As a user who finds notifications distracting,* I want one simple switch to turn off all notifications from the OneIITP app.
  - *As a user who wants reminders,* I want to ensure notifications are enabled via this switch and my OS settings.
- **Functional Requirements:**
  - Provide a single "Enable Notifications" toggle switch in the main Settings screen.
  - The state of this toggle should control whether the app attempts to send *any* notifications to the user's device via FCM/APNs (in addition to OS-level permission checks).
  - Ensure the toggle state is saved reliably per user.
  - Backend logic for triggering different notification types (library, event, lapsed user) remains implemented but respects this master toggle.
- **Acceptance Criteria:**
  - The master notification toggle switch exists in Settings and saves its state correctly.
  - When disabled, no push notifications are received from OneIITP (tested across different types).
  - When enabled (and OS permissions granted), relevant notifications (e.g., library due date if opted-in backend-wise, lapsed user event ping) are received.
- **Priority: Should Have** (Simplified control mechanism)
- **Epic:** EPIC-ACADEMICS, EPIC-COMMUNITY, EPIC-CORE
- **Dependencies:** Backend infrastructure for triggering notifications; Setup of FCM/APNs.



Notifications Toggle inside Profile Page

EPIC-ONBOARD: Onboarding & User Education Overhaul

Focus: Improving the crucial first-time user experience to better communicate the app's value and guide users, including mandatory QR code upload prompt.

Req ONB-01: Implement New Onboarding Flow with Mandatory QR Prompt

- **Detailed Description:** Redesign and implement the initial experience for first-time users (or after major updates). This flow includes welcome messaging, value proposition highlights, login, contextual permission requests, and an optional tour. **Crucially, integrate a mandatory step or persistent prompt for uploading the Profile QR code (PROF-01) during or immediately after onboarding.** If the user denies or skips the QR upload during onboarding, they should be re-prompted when accessing the Profile section or the Home Screen QR code grid item until the QR is uploaded.

> This eliminates the reported persistent "QR Gate Pass reminder" on every app open. We have got this issue many a times.

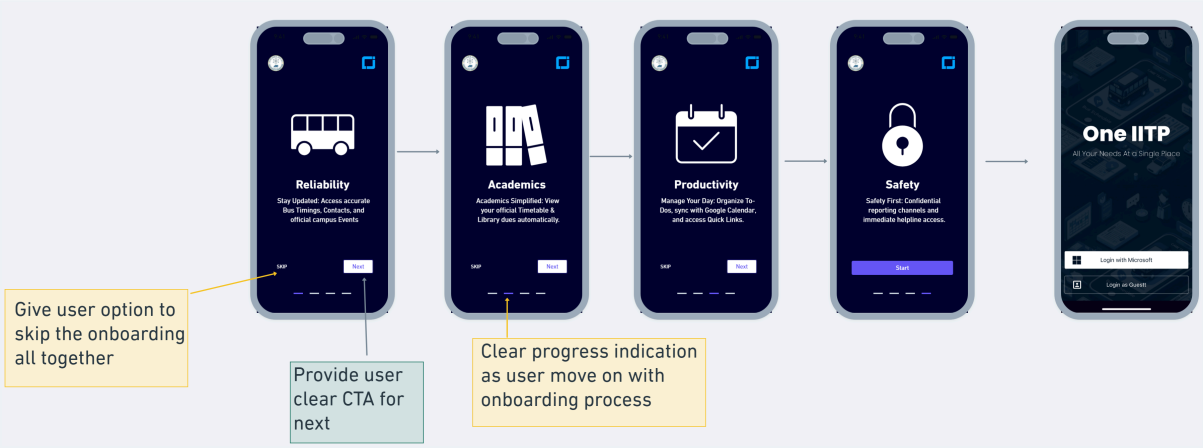
- **Goals:** Improve user activation/retention; Set clear value expectations; Smooth setup; Positive first impression; Ensure essential QR code is uploaded early; Remove annoying reminders.
- **User Stories:**
  - *As a new user,* I want a quick overview of the app's benefits before login.
  - *As any user,* I want login to be easy.
  - *As STC requiring QR for Gate Pass,* I need users to upload their QR code reliably during setup.
  - *As a user who skipped QR upload initially,* I understand being prompted again when I try to access my profile, but not constantly on every app open.
- **Functional Requirements:**

  Proposed Onboarding User Flow:

  1. **Step 1: Welcome Screen:** (Content: IITP/OneIITP branding, tagline "Your Essential Campus Companion". Interaction: "Get Started" button. Visuals: IITP branding).
  2. **Step 2: Value Proposition (Carousel/Swipeable Cards - Max 3-4):** (Purpose: Highlight key benefits with icons/minimal text. Cards: Reliability, Academics, Utilities/Convenience, Safety - *show based on implemented features*. Interaction: Swipe/dots, final card has "Next/Login").
  3. **Step 3: Login:** (Content: Prompt for official IITP credentials, link for help, "Remember Me" option. Interaction: Standard fields, Login button).
  4. **Step 4: Permissions Request (Contextual Preferred):** (Approach: Request when needed - Location for Bus map, Notifications for reminders - with clear explanation. Fallback: Explain upfront if essential).
  5. **Step 5: QR Code Upload Prompt (Mandatory Interaction):**
     - **Content:** Screen explaining the QR code is required (e.g., "Upload Your QR Code for Gate Pass").
     - **Interaction:** Prominent "Upload QR Code" button (triggers PROF-01 functionality - gallery picker). A "Skip for Now" option is present but leads to persistent reminders later (as per ONB-01).
  6. **Step 6: Quick Tour (Optional & Skippable):** (Purpose: Orient user to main nav/key sections. Implementation: Coach marks/tooltips highlighting bottom tabs, new features, safety, settings. Interaction: "Next", prominent "Skip Tour"/"Got It").
  7. **Step 7: Landing Page:** (Destination: Main Home screen or relevant starting screen).

  - **QR Upload Prompt Step:** Clearly explain the need for the QR code (e.g., "Required for Gate Pass Access"). Provide "Upload QR Code" button (linking to PROF-01 functionality) and potentially a "Skip for Now" option.
  - **Persistent Prompting (if skipped):** If QR upload is skipped during onboarding, display a prominent, non-blocking prompt (e.g., a banner in Profile, a modal on tapping QR grid item) reminding the user to upload their QR code. This prompt should only appear in relevant contexts (Profile, QR grid item tap) and disappear once the QR is uploaded.
  - **Remove Existing Reminder:** Ensure any logic causing a QR reminder on every app open is removed.

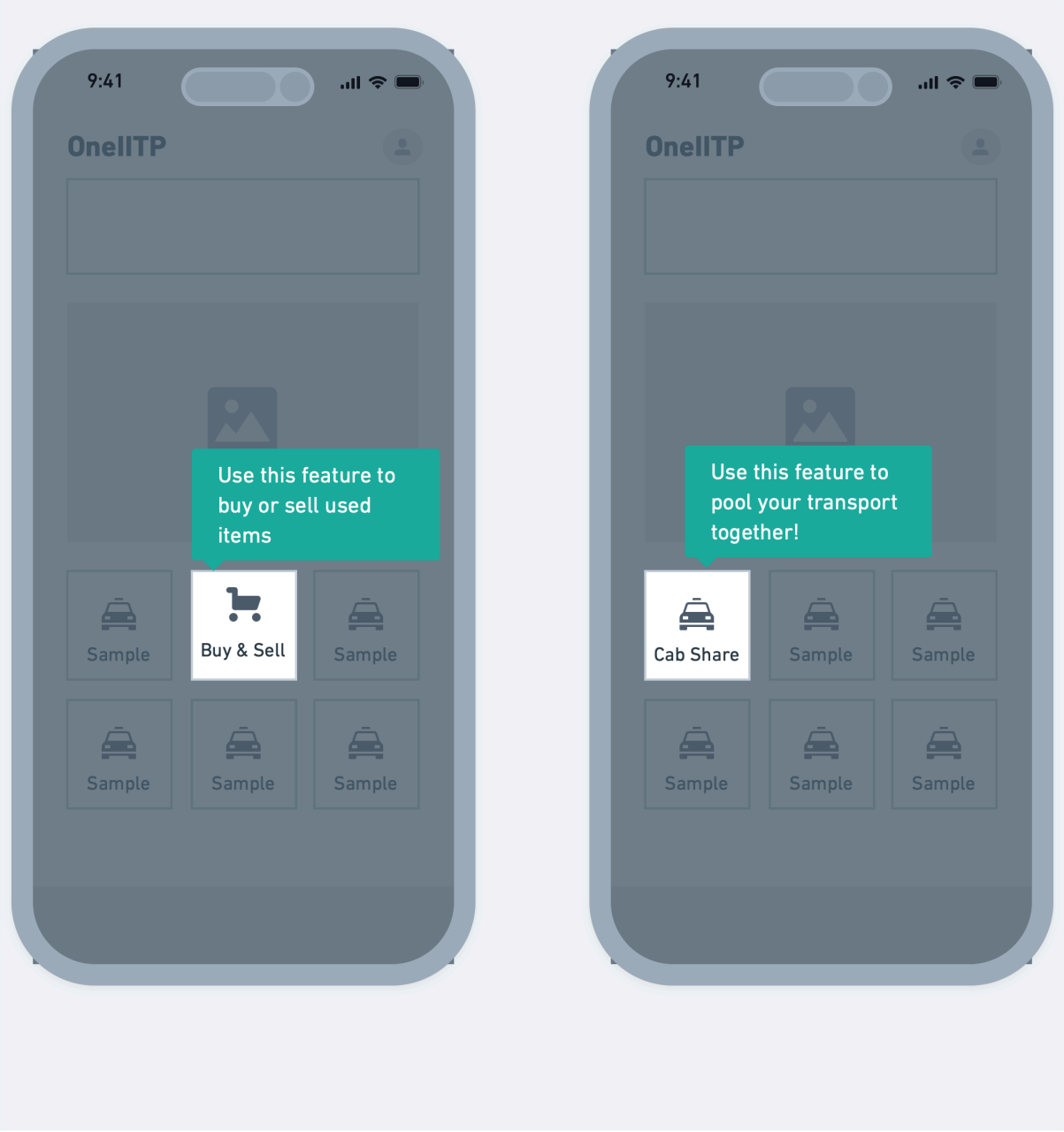
- **Acceptance Criteria:** Flow matches design; Steps functional; Value prop accurate; QR upload prompt appears during onboarding; QR upload works via prompt; Persistent prompt appears *only* in relevant sections if QR not uploaded, and disappears after upload; Existing annoying reminders are gone; Onboarding shown only once; High completion rate (analytics?).
- **Priority: Must Have (MVE)**
- **Dependencies:** UI/UX designs for onboarding & QR prompts; Stable login (CORE-04); Functional QR upload (PROF-01).



Onboarding User Flow Wireframes

Req ONB-02: Contextual Hints for New/Improved Features

- **Detailed Description:** Implement subtle, non-intrusive ways (badges, tooltips, dismissible banners) to guide existing users towards new or significantly improved features after an update, aiding discovery without annoyance.
- **Goals:** Increase adoption of new features; Help users understand changes; Guide without disruption.
- **User Stories:** *As an existing user,* I want subtle hints about major new features (like auto-timetable) after updating. *As a user,* a small confirmation of an improvement (like bus accuracy) would be nice.
- **Functional Requirements:** Display hints once per user/feature; Use non-modal UI; Easily dismissible; Configurable triggering based on version.
- **Acceptance Criteria:** Hints appear correctly for designated features post-update; Easily dismissible; No performance impact.
- **Priority:** Could Have
- **Dependencies:** List of features needing hints per release; UI designs for hints.



Onboarding Tooltips

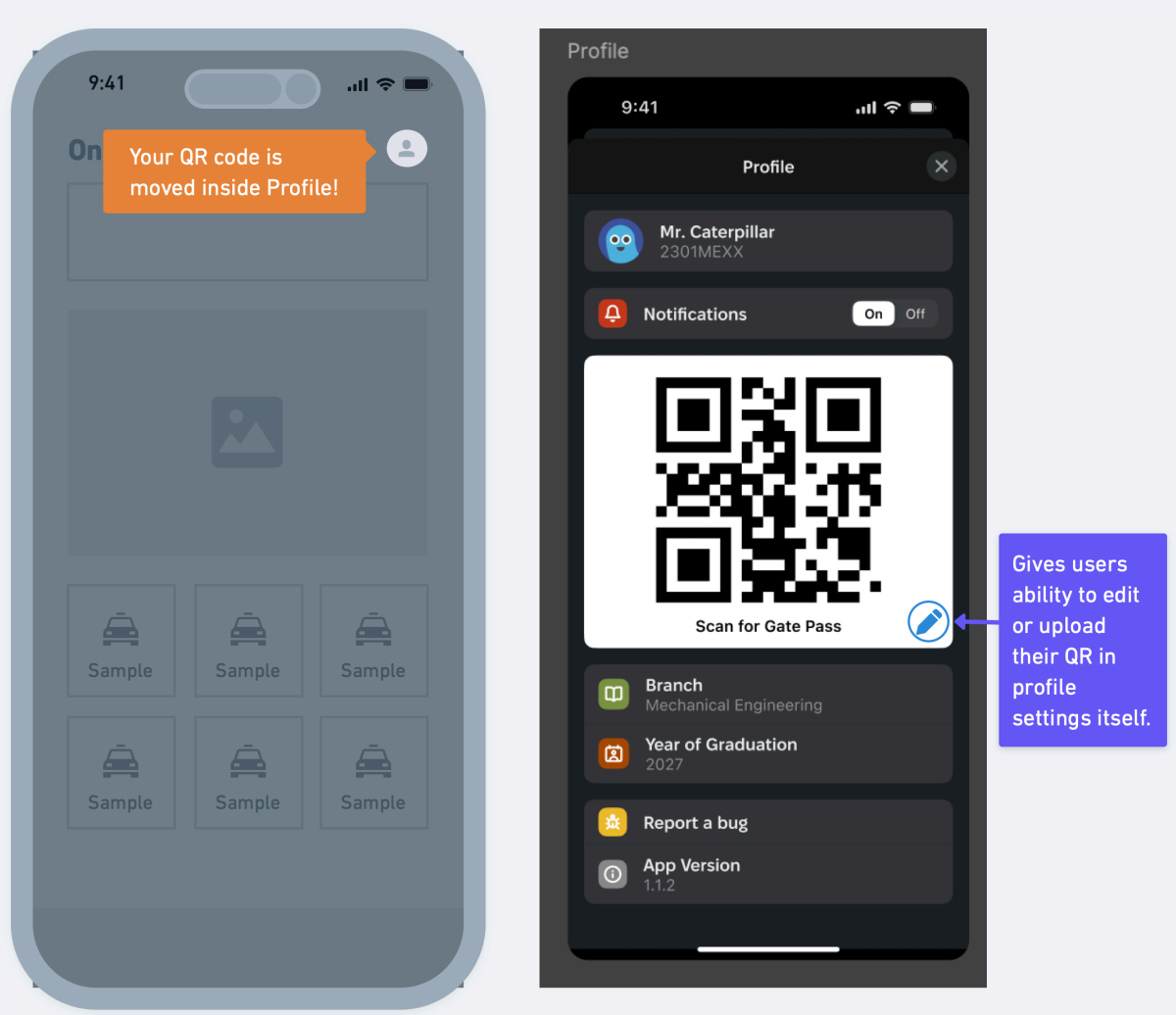
EPIC-PROFILE: Profile Management & QR Code

Focus: Improving the user profile section, focusing on QR code management and basic info display.

Req PROF-01: Profile QR Code Management (Upload & Display)

- **Detailed Description:** Implement a reliable mechanism within the user profile section for managing the user's official QR code (required for Gate Pass). The QR code, once uploaded, should be displayed prominently within the profile screen. Provide a clear visual **edit icon** (pencil icon shown in the UI) placed near the QR code display area, allowing users to **upload initially or subsequently update** their QR code image from their device gallery. This code must be securely associated with the user's profile and persist reliably across sessions.
- **Goals:**

    - Allow reliable storage, access, and *updating* of the essential QR code directly within the Profile section.
    - Eliminate upload friction and ensure the Gate Pass feature is functional and dependable.
    - Remove annoying global reminders, replacing them with contextual prompts if needed.
    - Clearly communicate the location of the QR code management to users.
- **User Stories:**
    - *As a student needing my QR code for gate entry,* I want to upload the official code image via my Profile and have it displayed there clearly, so I can quickly access it.
    - *As a user whose QR code might change or was uploaded incorrectly,* I want an obvious 'edit' button next to my displayed QR code in the Profile to easily upload a new version.
    - *As a user,* I want the QR code I uploaded to be saved permanently and not require constant re-uploads.
    - *As a user,* I don't want constant reminders to upload my QR if I haven't yet, only when I try to access related functions.
- **Functional Requirements:**
    - **QR Code Display:** Prominently display the currently saved QR code image within the main Profile screen area (below Notifications, as shown). Include the text "Scan for Gate Pass" below it.
    - **Upload/Update Trigger:** Implement an edit icon (pencil icon) adjacent to the QR code display area. Tapping this icon should trigger the image selection process.
    - **Image Selection:** Allow image selection from the device's photo gallery upon tapping the edit icon. Perform basic image format validation.
    - **Storage:** Securely store the uploaded QR code image data (backend recommended for persistence).
    - **Display for Scanning:** Tapping the main QR code image itself (not the edit icon) should ideally present it in a larger, brighter view suitable for scanning.
    - **Reminder Removal & Contextual Prompts:** Remove any global reminders. Implement contextual prompts as defined in ONB-01 (trigger on profile access or QR grid item tap *only if* QR is not yet uploaded).
    - **(Optional) Informational Message:** Consider displaying a one-time, dismissible informational message ("Your QR code is moved inside Profile!") for existing users after the update introduces this profile-centric location.
- **Acceptance Criteria:**
    - The saved QR code is displayed correctly in the Profile section with the "Scan for Gate Pass" label.
    - Tapping the edit icon successfully initiates the gallery image selection for uploading/updating the QR code.
    - Users can successfully upload/update their QR code via this mechanism.
    - The code is stored reliably and persists.
    - (If implemented) The informational message about the move appears correctly once for existing users.
- **Priority: Should Have** (Fixing core profile functionality & usability based on UI)
- **Epic:** EPIC-PROFILE
- **Dependencies:** User profile section UI; Storage decision (backend preferred); Clarity on QR purpose ("Gate Pass" confirmation); Functional ONB-01 for contextual prompts; Edit icon asset.



QR in Profile UI

EPIC-COMMUNITY: Events, Engagement & Support

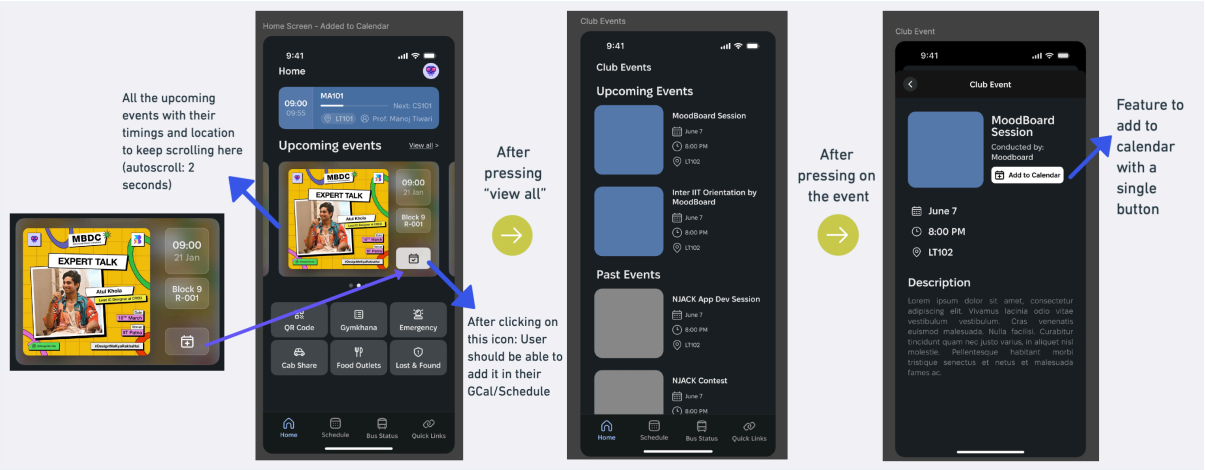
Focus: Making the app a better hub for campus happenings, providing support channels, and potentially increasing user engagement.

Req COM-01: Centralized Event Listing & Calendar Integration

- **Detailed Description:** Centralized display of upcoming/past campus events (submitted by authorized sources via workflow). Home screen shows visual upcoming event cards. Dedicated Events section lists upcoming/past. Tap for details. One-tap "Add to Calendar" using OS intent (pre-fills details, 30min default reminder). Implement re-engagement notifications for inactive users featuring relevant events.
- **Goals:** Clear view of events; Easy saving to personal calendar; Increase activity awareness; Drive usage via discovery/reminders; Re-engage lapsed users.
- **User Stories:** *As student,* see upcoming events on home. *As student,* tap for details. *As student,* tap 'Add to Calendar' to save easily. *As student,* expect phone calendar reminders. *As lapsed user,* event notifications might bring me back.
- **Functional Requirements:** Backend storage for events; Event submission workflow (with approval); Upcoming event cards on Home (scrollable, key info, Add to Cal button); Dedicated Events list section (Upcoming/Past); Event Detail screen (Image, Title, Org, Date/Time, Venue, Desc, Add to Cal button); "Add to Calendar" uses OS

intent (pre-fill, 30m reminder); Visually indicate added state in-app; Lapsed user re-engagement notifications (identify inactive users >10-14d, send targeted event pings via ACA-06 infra, limit frequency, deep-link).

- **Non-Functional Requirements:** Fast image load/cache; Smooth navigation; Reliable "Add to Calendar" intent.
- **Acceptance Criteria:** Events appear correctly; Details shown; "Add to Calendar" works (opens calendar app, pre-fills, reminder); Added state indicated; Inactive users receive targeted notifications; Notification tap deep-links.
- **Priority: Should Have**
- **Dependencies:** Backend infra; Submission workflow; Push notification service (ACA-06); UI/UX designs.

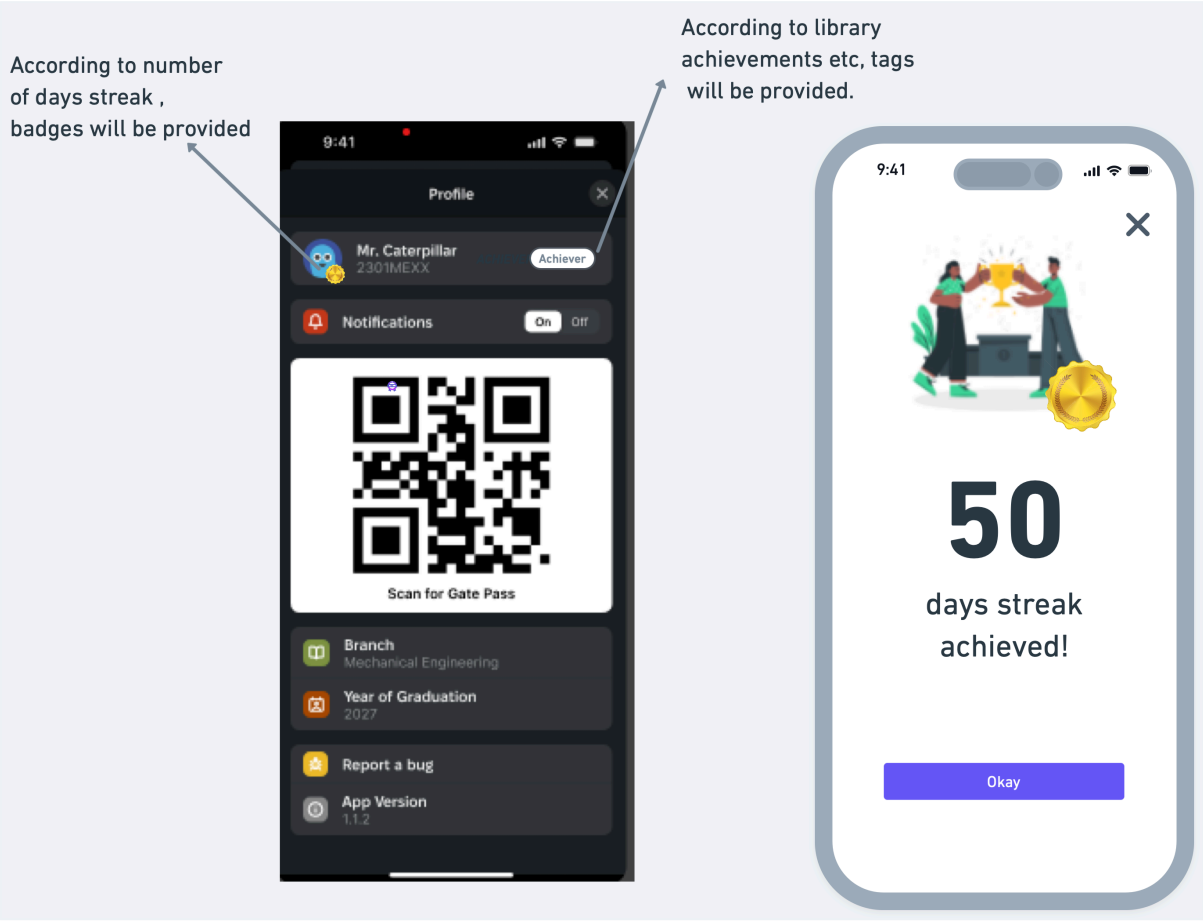


Events UI

Req COM-02: Gamification via Streaks & Achievement Tags

- **Detailed Description:** Implement lightweight gamification featuring Daily Login Streaks and Achievement Tags. Track consecutive daily logins, resetting if a day is missed. Award profile badges/icons (visual flair near profile pic) for reaching defined streak milestones (e.g., 7, 50 days). Separately, award distinct text-based tags (e.g., "Achiever", displayed near name) for completing specific valuable actions (library use, event adds, L&F resolution, bug reports). Trigger celebratory popups for significant streak milestones or earning new tags.
- **Goals:** Encourage daily use (streaks); Recognize feature engagement (tags); Add fun/personalization; Provide positive milestone feedback.
- **User Stories:**
  - As a user, seeing my daily login streak count increase (and potentially unlocking a visual flair/badge on my profile) motivates me to open the app daily.
  - As a user who regularly uses the library feature or helps others via Lost & Found, I want to potentially earn a tag like "Library Regular" or "Campus Helper" displayed on my profile as recognition.
  - As a user reaching a significant milestone like a 50-day streak, I want to see a fun confirmation popup acknowledging my achievement.
  - As a user viewing another's profile (if profiles become viewable later), these tags might give a glimpse into their campus engagement. (Future consideration)
- **Functional Requirements:**
  - Backend logic for tracking daily streaks (consecutive opens) & achievement triggers.
  - Define streak milestones (e.g., 7, 30, 50) & associated profile badges/icons.
  - Define achievement actions (e.g., return X books, add Y events) & associated profile tags (e.g., "Library Regular").
  - Display current streak badge/icon and earned tag(s) on the user profile.
  - Implement dismissible celebratory popups for milestones/new tags.

- **Non-Functional Requirements:** Minimal performance impact; Simple, non-distracting.
- **Acceptance Criteria:** Streaks calculated/reset correctly; Badges/tags awarded & displayed accurately; Popups triggered correctly/once per achievement; Profile reflects status.
- **Priority: Could Have**
- **Epic:** EPIC-COMMUNITY
- **Dependencies:** Backend tracking; Defined actions/rewards/milestones; UI/UX designs (badges, tags, popups); Dependent features for achievements (ACA-02, COM-01, CORE-07, COM-03).

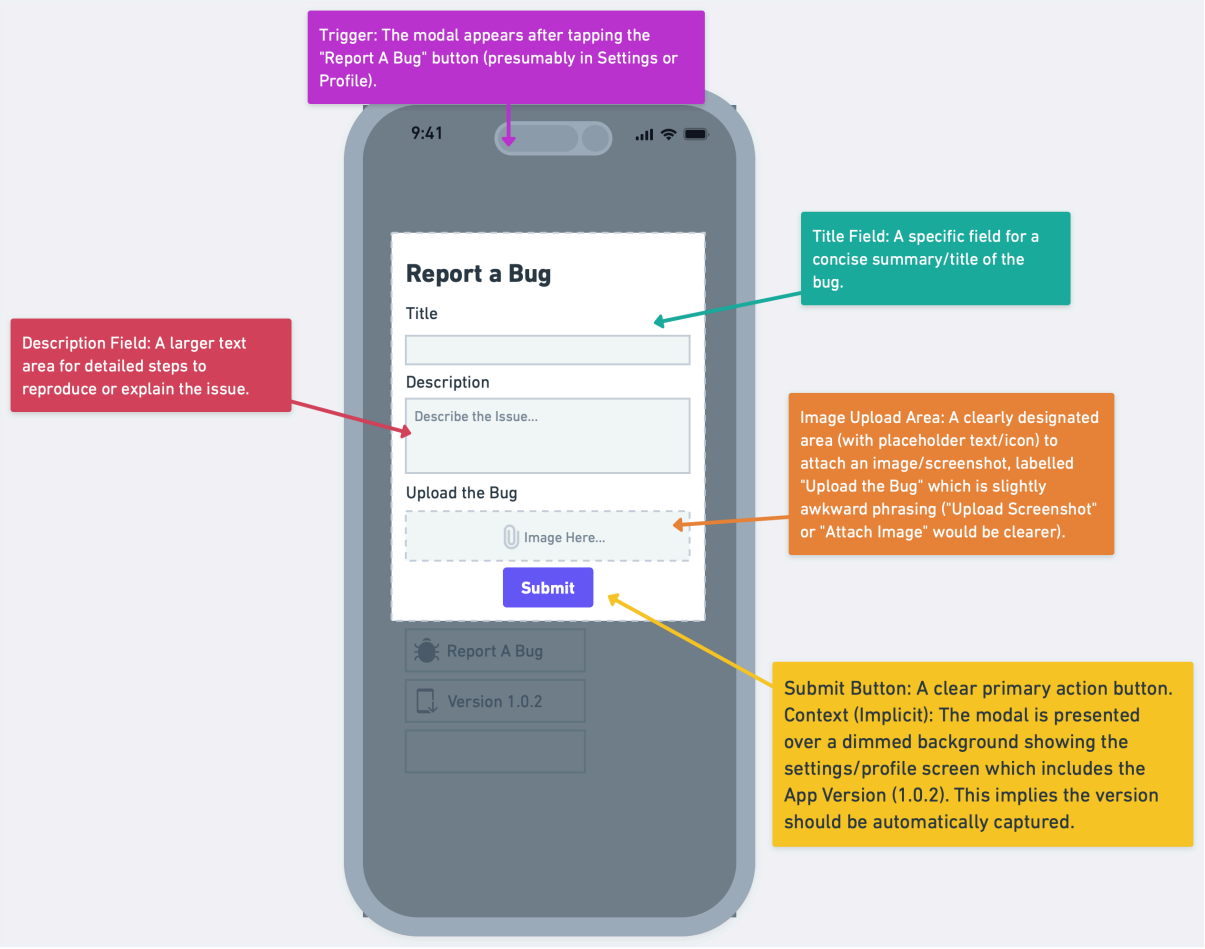


Gamification Wireframes

Req COM-03: Issue/Bug Reporting System (In-App)

- **Detailed Description:** Dedicated section (in Settings) for users to report app-specific issues (bugs, crashes, UI, performance) or suggestions. Allow classifying report type, description, optional screenshot. Route directly to STC tech team (Dev/Design/PM). Automatically include device/app context.
- **Goals:** Streamline bug/feedback reporting; Provide devs direct, contextual feedback; Improve app quality via user reports.
- **User Stories:**
  - *As user with crash,* I want easy in-app report with steps.
  - *As user with suggestion,* I want dedicated channel.
  - *As STC dev,* I want reports with context/screenshots.
- **Functional Requirements:** "Report Bug/Suggestion" section; Optional classification fields; Required description field; Optional screenshot attachment (gallery/capture); Auto-include context (AppVer, OSVer, Device); Submit securely to STC endpoint (email, issue tracker, DB).

- **Acceptance Criteria:** Can navigate/submit reports; Includes description/screenshot; Reports received by STC with context; Process intuitive.
- **Priority: Should Have** (Essential feedback loop)
- **Dependencies:** Defined receiving endpoint; UI design for form.



Report A Bug Wireframe

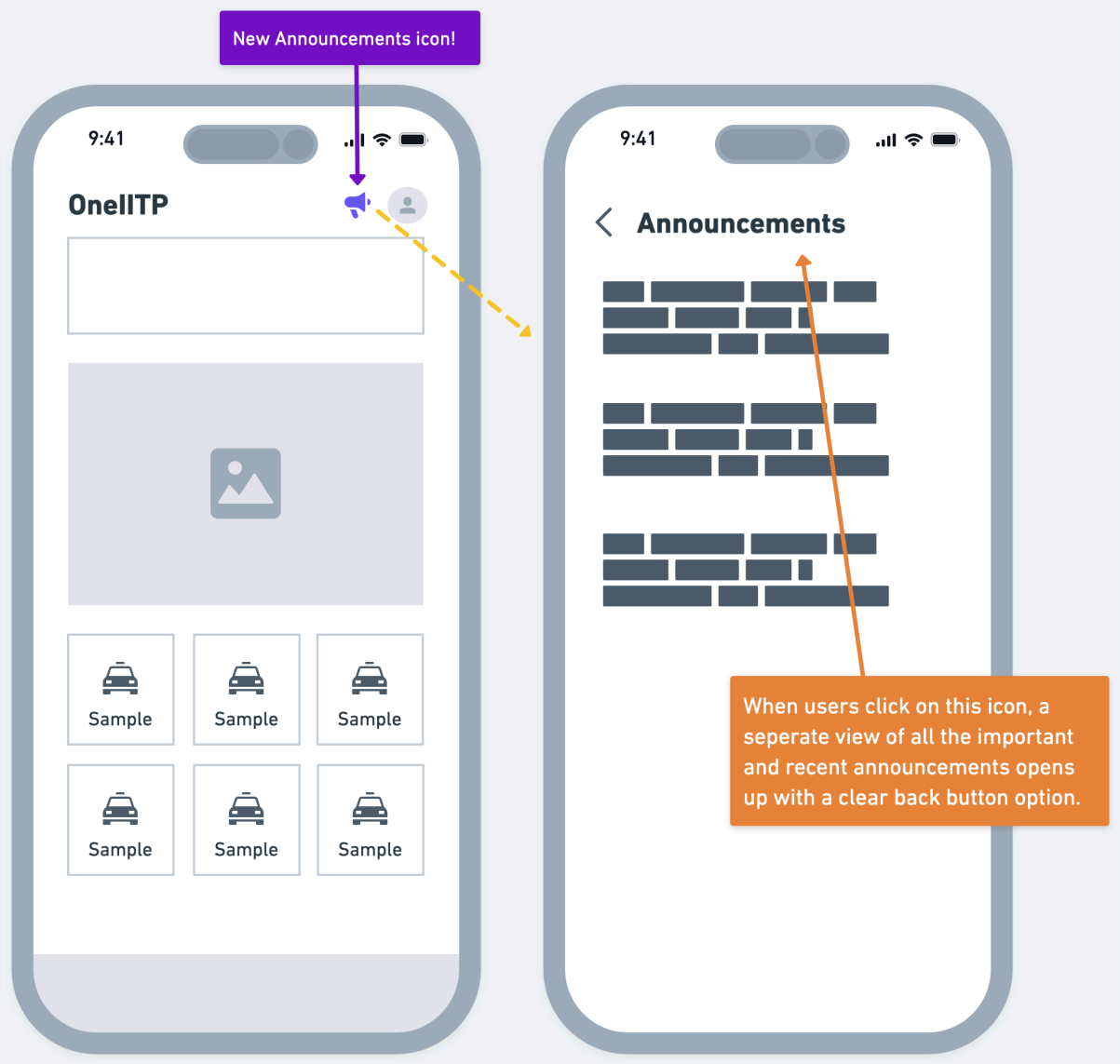
Req COM-04: Contact Information Verification & Completeness

- **Detailed Description:** Ensure "Contacts" section (Emergency, Gymkhana, Admin, Shops) is accurate, complete, regularly maintained. Initial verification, add missing, establish periodic review process (e.g., semesterly) by STC. Clear labels, relevant details (phone, email, location/hours). Tapping contact should initiate native action (dialer/email).
- **Goals:** Reliable contact directory; Ensure easy/correct contact usage; Build trust in app data accuracy.
- **User Stories:**
  - *As student needing hostel office,* find correct number/hours easily.
  - *As user needing emergency number,* find accurately, call immediately.
  - *As student needing STC rep,* find current contact.
- **Functional Requirements:** Verify all contacts; Add missing essentials; Structure logically (categories); Phone numbers tappable -> dialer intent (handled by UX-03); Email addresses tappable -> email intent; Include location/hours if relevant; Document/implement regular update process.
- **Acceptance Criteria:** Contacts verified/accurate; Well-organized; Tap actions work (via UX-03); Relevant info included; Update process established/followed.

- **Priority: Should Have**
- **Dependencies:** Department/club cooperation; Defined maintenance process; Relies on UX-03 for tap actions.

📣 Req COM-05: Institute Announcement Channel

- **Detailed Description:** Implement a dedicated section or feed within the app for displaying official announcements from the institute administration or STC. This serves as a centralized, authoritative channel for important updates, notices, circulars, etc., potentially replacing or supplementing emails/website notices. Requires a backend system and an interface for authorized personnel to post announcements.
- **Goals:** Provide a single, reliable source for official institute announcements; Improve visibility and accessibility of important information; Reduce reliance on email/other channels for critical updates.
- **User Stories:**
  - *As a student,* I want one place in the app to check for all official announcements from the administration or STC, so I don't miss important deadlines or policy changes.
  - *As an administrator,* I want a way to push important announcements directly to students via the official app.
  - *As a user,* I want announcements to be clearly dated and perhaps categorized.
- **Functional Requirements:**
  - Develop backend system to store announcements (Title, Content, Posting Date, Author/Department, Category - optional, Attachments - optional).
  - Create a secure interface/workflow for authorized personnel (Admin/STC) to create and publish announcements.
  - Display announcements in a dedicated section within the app, sorted chronologically (newest first).
  - Consider push notifications (using ACA-06 infrastructure, respecting master toggle) for critical announcements (requires careful policy/tagging).
  - Allow viewing announcement details, including attachments if supported.
- **Acceptance Criteria:** Authorized personnel can post announcements; Announcements appear correctly in the app, sorted chronologically; Details/attachments are viewable; Push notifications (if implemented) work for tagged critical items.
- **Priority: Could Have**
- **Epic:** EPIC-COMMUNITY
- **Dependencies:** Backend infrastructure; Posting interface/workflow; Authorization policy; Decision on push notifications for announcements.



Announcement Channel Wireframe

EPIC-SAFETY: Safety & Reporting Features

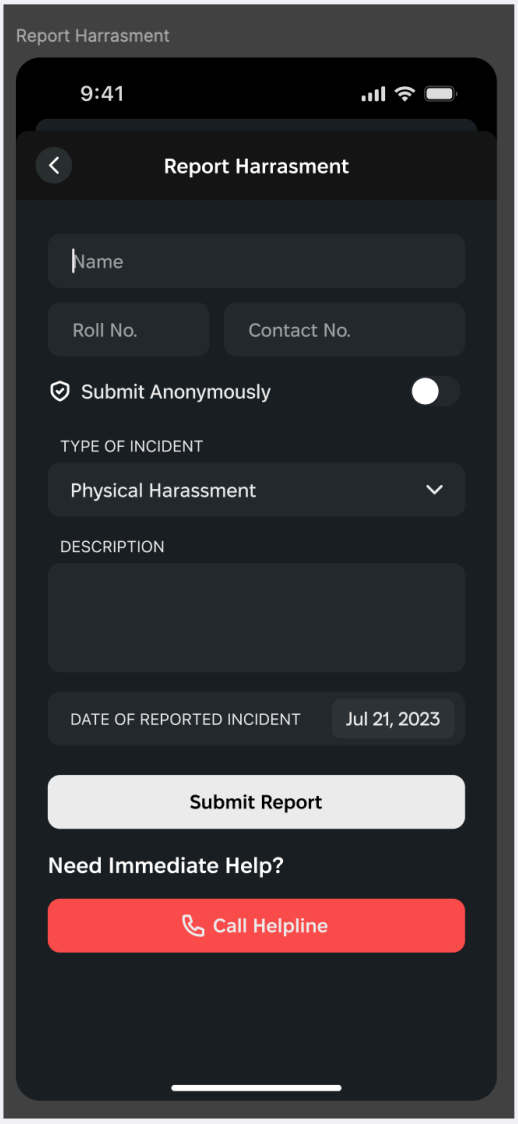
Focus: Implementing features dedicated to student safety and providing accessible reporting channels.

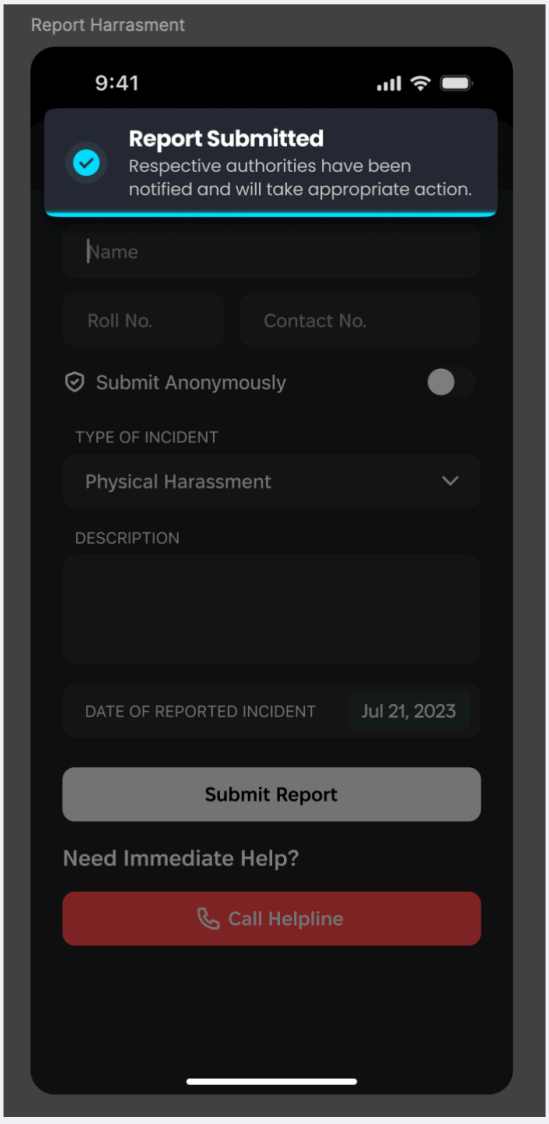
Req SAFE-01: Report Harassment Feature

- **Detailed Description:** Dedicated, secure, confidential feature to report harassment (physical, verbal, online, etc.). Allow description, type, date, location (opt). User chooses anonymity (Yes/No toggle). Submit *only* to designated, confidential authority (e.g., committee, Dean's office) per institute policy. Provide immediate "Call Helpline" button.
- **Goals:** Safe/confidential reporting channel; Ensure reports reach correct authority securely; Offer anonymity; Provide immediate support access; Demonstrate commitment to student safety.
- **User Stories:**
  - *As student experiencing/witnessing harassment*, I want safe/confidential in-app reporting.
  - *As reporter*, I want anonymity option.

    - *As user needing support,* I want quick helpline access.
    - *As authority,* I need secure report reception.
- **Functional Requirements:** Dedicated "Report Harassment" section; Input fields (Name/Roll/Contact - optional based on toggle, Anon toggle, Incident Type dropdown, Description required, Date, Location opt); Secure transmission (HTTPS); Configure endpoint *only* to designated authority; Confirmation message; Prominent "Call Helpline" button initiating call to official number.
- **Non-Functional Requirements:** Security/Confidentiality highest priority (E2E encryption, secure handling, preserve anonymity); Reliability.
- **Acceptance Criteria:** Form accessible/functional; Anon toggle works; Reports securely transmitted *only* to correct authority; Confirmation shown; Helpline button calls correct number; Complies fully with IITP policy.
- **Priority: Should Have** (High Importance Safety Feature)
- **Dependencies:** CRITICAL: Clear policy/designated authority; Official helpline number; Robust security plan; Legal/policy review.

This is a high level implementation of Report Harassment Feature UI made by Chaitanya for OneIITP.

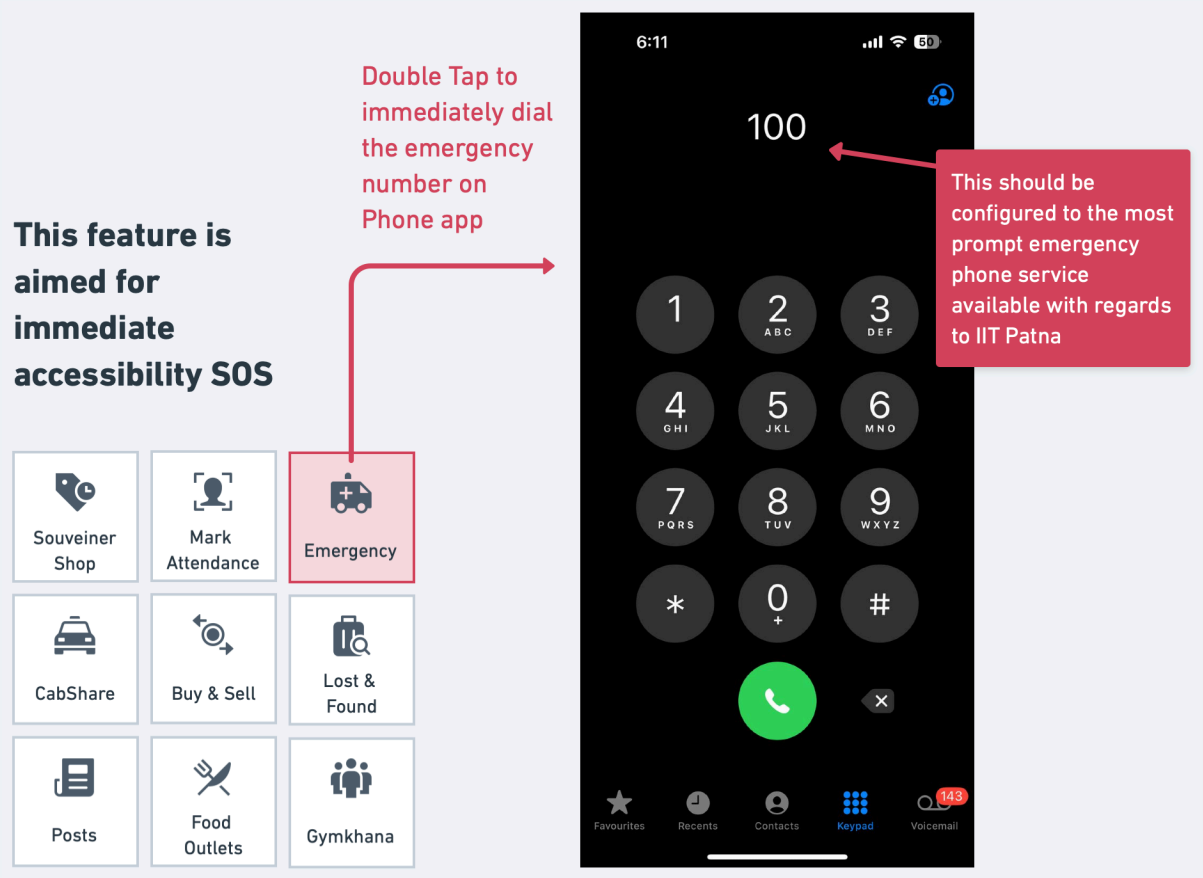




Reporting feature UI

Req SAFE-02: Emergency SOS Button (Double Tap)

- **Detailed Description:** Implement an easily accessible "Emergency" button on the main Home Screen grid. This button should have a distinct visual style (e.g., elevated, red color) to emphasize its purpose. A single tap on the button navigates the user to the main safety screen (where Report Harassment and Call Helpline options exist). A **double tap** (or potentially long press - requires UX evaluation) on the Home Screen Emergency button should *immediately* initiate a phone call to the primary IIT Patna emergency helpline number (e.g., Security). This provides a quicker way to call for help in urgent situations.
- **Goals:** Provide extremely fast access to call the emergency helpline in critical situations; Offer clear path to other safety features (reporting).
- **User Stories:**
  - *As a student in an urgent emergency situation,* I want to be able to call campus security instantly with just a quick double tap on a prominent button on the app's home screen.
  - *As a user unsure if I need to call or report,* I want a single tap on the Emergency button to take me to a screen with both options clearly laid out.
  - *As a user,* I want the Emergency button to look distinct and clearly signal its critical function.
- **Functional Requirements:**
  - Place an "Emergency" button on the Home Screen grid (order configurable via CORE-06).
  - Style the button distinctly (e.g., red background, elevated appearance - requires UI design).
  - **Single Tap Action:** Navigate the user to the main Safety screen (containing SAFE-01 Report Harassment / Call Helpline options).
  - **Double Tap Action (or Long Press):** Immediately initiate a phone call intent to the predefined primary IIT Patna emergency number (e.g., Security Main Gate). Requires confirmation of the number.
  - Ensure the double-tap detection is reliable and doesn't trigger accidentally too easily. (Long press might be safer from accidental triggers).
- **Acceptance Criteria:**
  - Emergency button is present on Home Screen grid with distinct styling.
  - Single tap correctly navigates to the Safety screen.
  - Double tap (or long press) successfully and immediately initiates a phone call to the correct emergency number.
  - Double tap/long press interaction feels responsive but avoids easy accidental calls.
- **Priority: Should Have** (High Importance Safety Feature)
- **Epic:** EPIC-SAFETY
- **Dependencies:** Confirmation of the primary emergency helpline number; UI/UX design for the button and interaction (double tap vs long press decision); Reliable tap detection logic.



Double Tap Emergency

EPIC-UX: User Experience Refinement

Focus: Improving the overall usability, consistency, and visual polish of the app based on user feedback and design principles.

Req UX-01: Consistent Navigation & Visual Polish

- **Detailed Description:** Conduct a comprehensive UI/UX audit across the entire application. Ensure consistent use of navigation patterns (back buttons - CORE-05, bottom bar behavior), terminology, typography, color palettes, and spacing. Apply proper UX spacing principles (e.g., 9pt grid system or similar) for readability and accessibility. Address general UI inconsistencies and improve visual hierarchy. The goal is a cohesive, intuitive, and aesthetically pleasing user experience.
- **Goals:** Improve overall app usability and learnability; Ensure visual consistency and professional appearance; Enhance readability and accessibility.
- **User Stories:**
  - *As a user navigating different sections,* I want the app to feel consistent in how buttons look, where menus are, and how spacing is used, making it easier to learn and use.
  - *As a user,* I want text to be easily readable with appropriate margins and line spacing.
  - *As a user,* I want the app to look polished and modern, not haphazardly put together.
- **Functional Requirements:**
  - Establish or adopt a basic style guide (colors, typography, spacing units).
  - Review all major screens and components for consistency with the style guide.

    - Refactor UI components to enforce consistent spacing, alignment, and padding (use standardized spacing values).
    - Ensure consistent use of fonts, font sizes, and weights for different text elements (headers, body, captions).
    - Verify consistent implementation of navigation elements (back buttons, titles).
- **Acceptance Criteria:**
    - Key screens demonstrate consistent application of defined styles (color, typography, spacing).
    - Visual hierarchy is clear on major screens.
    - Navigation elements are consistently implemented.
    - User feedback indicates improved visual polish and ease of use.
- **Priority: Should Have** (Impacts overall perception and usability)
- **Epic:** EPIC-UX
- **Dependencies:** Defined or adopted style guide/design system principles; UI/UX design capacity for audit and refinement.

Req UX-02: Redesign Forms/Modals using Native Patterns

- **Detailed Description:** Redesign in-app forms, modals, popups, and pickers (like date pickers, dropdown selectors used in Harassment Report, Event Submission, etc.) to utilise standard native OS patterns and components (e.g., native date/time pickers, bottom sheets for selections on iOS/Android) instead of potentially clunky custom implementations. Ensure clear Call-to-Actions (CTAs) on all interactive modals/forms.
- **Goals:** Improve usability and familiarity of interactive elements; Reduce user friction when inputting data or making selections; Leverage native accessibility features.
- **User Stories:**
    - *As a user filling out the harassment report form,* I want to use the standard date picker I'm familiar with on my phone, making it quicker and less error-prone.
    - *As a user presented with options in a popup,* I want clear buttons like "OK", "Cancel", "Submit" that follow standard platform conventions.
    - *As a user,* I find custom form elements confusing; native ones are easier to use.
- **Functional Requirements:**
    - Identify all custom form elements, modals, pickers currently used.
    - Replace custom implementations with appropriate native components provided by the OS or standard React Native libraries that wrap native components (e.g., @react-native-community/datetimepicker, native alert dialogs, bottom sheet libraries).
    - Ensure all interactive modals/forms have clear, standard CTA buttons (e.g., Submit, Cancel, Save, Done).
    - Test thoroughly on both Android and iOS to ensure native look, feel, and behavior.
- **Acceptance Criteria:**
    - Date pickers, time pickers, selection dropdowns use native OS controls.
    - Modals and popups use standard platform presentation styles (e.g., bottom sheets, alert dialogs).
    - All interactive overlays have clear and standard CTAs.
    - User feedback indicates improved ease of use for forms and selections.
- **Priority: Should Have** (Addresses specific usability pain point)
- **Epic:** EPIC-UX
- **Dependencies:** UI/UX design guidance on native component usage; Developer familiarity with accessing native modules/components in React Native.

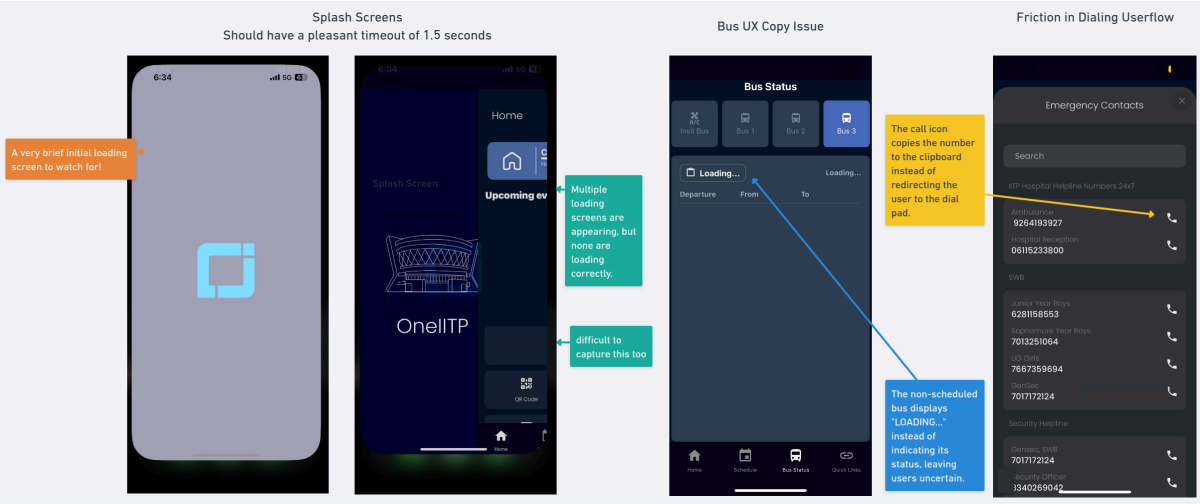
Clunky modals

Req UX-03: Refine Interaction Flows (Splash Screen, Bus Info, Contacts)

- **Detailed Description:** Address specific interaction flow issues identified in feedback:
  - **Splash Screen:** Adjust timing so it feels more natural and less abrupt, potentially allowing it to display until initial essential data is loaded (within performance targets of CORE-02). Avoid fixed artificial delays if possible; transition smoothly.
  - **Bus Information:** In the bus view (SWB), if trip data fails to load or no trips are available, display a clear message like "No Trips Available Currently" or "Failed to load bus data. Please try again." instead of leaving the section blank or showing a perpetual loader. Add essential identifying info like Bus Number Plate and potentially Driver Name (if available and permissible) to displayed bus details to increase trust and utility.
  - **Contact Dialing:** Ensure that tapping a phone number in the Contacts section (COM-04) directly opens the device's native phone dialer app with the number pre-filled, ready for the user to initiate the call.
- **Goals:** Improve perceived smoothness and feedback; Provide better error/empty state handling; Enhance trust and information in bus details; Reduce friction in contacting people.
- **User Stories:**
  - *As a user,* I want the splash screen transition to feel smooth, not jarringly fast or artificially slow.
  - *As a user checking bus info,* if there are no buses running or data fails, I want the app to tell me clearly instead of showing nothing. Knowing the bus number plate would also be helpful.
  - *As a user wanting to call a contact from the app,* I want tapping the number to take me straight to my phone's dialer with the number ready, saving me copy-pasting.
- **Functional Requirements:**
  - **Splash Screen:** Optimize loading logic; tie splash screen dismissal to app readiness or a smooth animation transition, respecting overall load time targets (CORE-02).
  - **Bus View:** Implement clear empty/error states for the trip list. Fetch and display Bus Number Plate/Driver Name if available via API/data source.
  - **Contact Dialing:** Use appropriate React Native linking mechanism (Linking.openURL('tel:${phoneNumber}')) to trigger the native dialer intent when a phone number is tapped in the Contacts section.
- **Acceptance Criteria:**
  - Splash screen transition feels natural and respects performance goals.

- Bus view displays informative messages for empty/error states. Bus number plate/driver name displayed if data available.
- Tapping phone numbers in Contacts correctly opens the native dialer with the number pre-filled.

- **Priority: Should Have** (Addresses specific usability flaws)
- **Epic:** EPIC-UX
- **Dependencies:** Refined splash screen logic; Access to Bus Number Plate/Driver Name data; Correct implementation of linking for dialing.



Refine Interaction Flow Examples

EPIC-MISC: Miscellaneous Features & Fixes

Focus: Addressing specific features or fixes that don't neatly fit into other core Epics.

Req MISC-01: Implement "Login as Guest" with Restricted Access

- **Detailed Description:** Provide "Continue as Guest" option on login. Guest users access limited public info only (e.g., Events, Bus Schedule, Emergency Contacts). Restrict access to student-only sections (Profile QR, Schedule, Library, Gymkhana, Safety features, Utilities). Prompt guests to log in for full access.
- **Goals:** Allow limited public access; Enhance security for student data; Preview app value.
- **User Stories:** *As visitor,* I want public bus schedule without login. *As STC,* I want Gymkhana info protected. *As prospective student,* I want to browse public info.
- **Functional Requirements:** Add "Guest" option; Implement role-based access control (Guest vs. Logged-in); Define public/private sections; Block restricted access for guests with login prompt; No persistent guest data.
- **Acceptance Criteria:** Guest login works; Only public sections accessible to guests; Restricted sections blocked with prompt; No security loopholes.
- **Priority: Should Have** (Important for security/visitors)
- **Dependencies:** Clear definition of public/private content; Robust auth logic (CORE-04).

---

12. User Flows (Examples for Key Changes)

1. **Flow: Checking Schedule with Attendance Marking:**
   - User opens app -> Taps "Schedule" tab -> App instantly displays today's view with ERP classes (ACA-01).

- User attends a class, scans QR code presented -> App confirms "Attendance Marked" for that session (via ACA-03 integration).
- Later, user reviews schedule -> Taps on a past class session (e.g., "MA101").
- A modal/bottom sheet appears showing class details and attendance status (e.g., "Present - Scanned").
- User realizes they missed scanning for another class -> Taps that class session -> Modal shows "Status: Unmarked" -> User taps "Mark As -> Present" -> Modal updates -> Status saved.
- User navigates to see percentage displayed near course name -> App shows calculated attendance % (e.g., "85%") and a "Low Attendance" warning pill if below threshold (e.g., <75%).

2. **Flow: Reporting Harassment (Using Native Modals - UX-02):**
   - User navigates to Safety section -> Taps "Report Harassment."
   - App displays the reporting form. User reads confidentiality info.
   - User taps "Type of Incident" -> Native OS picker (dropdown/bottom sheet) appears -> User selects type -> Picker dismisses.
   - User taps "Date of Incident" -> Native OS date picker appears -> User selects date -> Picker dismisses.
   - User fills description, decides anonymity -> Taps "Submit Report" (Clear native-style button).
   - App securely transmits data -> Displays native confirmation alert dialog: "Report Submitted. Authorities notified." with an "OK" button.
3. **Flow: Using Emergency SOS Button (SAFE-02):**
   - User perceives an emergency -> Opens OneIITP app Home Screen.
   - User double-taps the distinct, red "Emergency" button.
   - *App immediately initiates the phone call intent tel:<EmergencyNumber>.*
   - Device's native phone dialer app opens with the Security number pre-filled, ready to call.
   - (Alternative) User single-taps "Emergency" button -> App navigates to Safety screen with "Report Harassment" and standard "Call Helpline" buttons.

---

13. Technical Considerations & Recommendations

- **Cross-Platform Approach (React Native):** Continue leveraging React Native but focus on:
  - **Performance Optimization:** Profile React Native bridge communication, optimize component rendering (use React.memo, useCallback, FlatList optimizations), minimize unnecessary re-renders.
  - **Native Modules:** Utilize native modules/community packages that wrap native functionality for performance-critical tasks or when needing truly native UI elements (pickers, modals - UX-02, contact dialing - UX-03).
- **Backend Scalability:** Remains critical. Needs to support increased load from utilities (L&F, B&S, Cab Share - potentially many reads/writes), integrations, To-Dos, attendance tracking, etc. Consider database indexing, caching strategies, potentially serverless functions.
- **Google Calendar Integration Approach:** (Phase 1 Push focus remains) Use official Google API libraries via React Native bridge/packages. Implement secure OAuth 2.0. Format events correctly. Handle API errors.
- **Report Harassment Implementation Considerations:** (Security paramount) Use HTTPS. Ensure backend endpoint is secure & *only* accessible by designated authority. Preserve anonymity rigorously if chosen. Consider secure logging policies. Reliable submission essential.
- **Utility Features Backend:** L&F, B&S, Cab Share require backend APIs for posting, retrieving, searching, and potentially managing listings (delete/mark sold). Need database schema design, API endpoints, and potentially image storage solution (e.g., S3, Firebase Storage).
- **Student Developer Context:** (Critical) Excellent documentation, component storybooks (optional but helpful), mandatory code reviews, CI/CD pipelines, structured knowledge transfer plans are essential for React Native project sustainability with rotating teams.

14. How Do We Educate Users About Changes?

- **Pre-Launch Buzz:** Use STC channels for announcements, focusing on solutions (e.g., "Tired of losing things? Lost & Found coming soon!").
- **Release Notes:** Detailed, benefit-oriented notes listing major bug fixes, new features (Timetable, Library, Safety, Utilities, UX improvements), and changes.
- **In-App Onboarding (ONB-01):** Essential for new users, highlights key value props including new utilities and safety.
- **Contextual Hints (ONB-02):** Sparingly for major new features/UX changes (e.g., native modals).
- **"What's New" Screen:** In-app summary of latest version changes.
- **Targeted Communication:** Specific comms for Harassment Reporting (from Admin/Wellness). Promote utility features via STC channels.
- **Feedback Channels (COM-03):** Actively promote in-app bug reporting.

---

15. Non-Product Requirements

- **Policy & Workflow Definition (Admin):** Formal approval for Harassment Reporting workflow. Guidelines/terms of use for Buy/Sell, Cab Share, Lost & Found (content moderation, liability disclaimers).
- **API Access & Collaboration (Admin/IT):** Formal agreements/support for ERP, Library, Auth APIs. Transport liaison for schedule updates.
- **Legal & Privacy Review (Admin):** Review data handling for Reporting, Utilities (contact info display), GCal sync.
- **Marketing & Re-Launch Campaign (STC):** Plan campaign post-MVE stabilization.
- **Student Dev Team Support (STC/Faculty Advisor):** Tools, test devices, backend resources, mentorship (esp. React Native performance, native modules), process management.
- **User Support Training (STC):** Train members on new utilities, reporting, common issues.
- **Content Moderation Plan (STC):** Define process for handling inappropriate content in utility features (L&F, B&S).

---

16. Out of Scope for this Phase (M2 Focus)

- Full-fledged social networking features (feeds, direct messaging).
- Advanced AI/ML features.
- Hostel/Roommate Finder, Detailed Mess Menu system.
- Complex Gamification (beyond basic points/badges/streaks).
- Direct Payment Integrations.
- Faculty-specific portal/features.
- Real-time Bus GPS Tracking (BUS-02 - deferred to Could Have).
- Two-Way Google Calendar Sync (Phase 2 of ACA-05).
- In-app messaging/chat between users for utility features (rely on direct contact info for M2).

---

17. Rollout Plan (Incorporating Dogfooding)

1. **Phase 2a: MVE Stabilization & Core Fixes**
    - **Development Focus:** MVE requirements across CORE (Inc. Nav fixes), BUS (BUS-01), ACADEMICS (ACA-03 - link fix), PROFILE (PROF-02), UX (critical fixes identified in audit).
    - **Continuous Dogfooding:** STC uses internal builds daily. Dedicated feedback channel.

- **Alpha Testing:** Wider STC + trusted testers (1-2 weeks). Focus on stability, performance, basic usability.
- **Beta Testing:** Volunteer students via Play Store/TestFlight (2-3 weeks). Quantitative & qualitative feedback.
- **Production Release (Phased MVE):** Gradual rollout (5% -> 100%), monitor metrics/feedback closely.

2. **Phase 2b/3: High-Value Integrations & Utilities**
   - **Development Focus:** Prioritize **Should Have** features: ACA-01(Timetable), ACA-02(Library), SAFE-01(Harassment), ACA-07(Attendance Track), CORE-07/08/09(Utilities), ONB-01(Onboarding+QR), PROF-01(QR Manage), COM-01(Events), COM-03(Bug Report), COM-04(Contacts), SAFE-02(SOS), UX refinements (UX-01, UX-02, UX-03), MISC-01(Guest).
   - **Iterative Rollout:** Release in smaller bundles (e.g., Bundle 1: Academics; Bundle 2: Utilities; Bundle 3: Safety+UX). Each follows Dogfooding->Alpha->Beta->Phased Production.
   - **Feature Flags:** Use extensively for new integrations and backend-dependent features.

---

18. Open Questions

1. **API Definitives & SLAs:** Confirmed technical details & support for ERP, Library APIs?
2. **Harassment Reporting Protocol:** Finalized, approved workflow, recipient, and confidentiality measures?
3. **Utility Feature Policies:** Official guidelines for content moderation, liability, data display for L&F, B&S, Cab Share?
4. **Student Developer Capacity:** Realistic availability (RN/Backend)? Onboarding/training plan? Mentorship available?
5. **Backend Resources:** Sufficient hosting/cloud resources confirmed? Cost implications addressed?
6. **Design Resources:** Dedicated UI/UX capacity for new UIs, native component guidelines, UX audit?
7. **Google Calendar API Quotas:** Understood? Need for increase anticipated?
8. **Testing Device Availability:** Sufficient range of physical devices confirmed?
9. **Attendance Policy Details:** Exact calculation logic for percentage (how are L/C handled)? Official threshold for "Low Attendance"? ERP provides credit info?
10. **Emergency Helpline Number:** Confirmed primary number for SAFE-02?

---

19. MOMs / Appendices

- Appendix A: Complete Feedback Form Responses
- Minutes of Meeting Initial
- User Personas (Segmented)
- Competitive App Feature Analysis (InstiApp)
- Figma Link for M1 OneIITP
- All Wireframes Combined (Below)

All Wireframes Combined
