ChatGPT

Campus Transport App – Product Requirements Document

Introduction

College students often face inconvenience when arranging off-campus transport due to limited options and long wait times. On-demand ride services tailored for campuses can alleviate this issue – for example, Uber highlights the need for "stress-free, flexible, and cost-effective transportation" for students [1]. Similarly, one study notes that "efficient and sustainable transportation is a growing priority on university campuses" and that ride-sharing solutions can reduce congestion and improve mobility [2]. Our solution is a mobile/web application ("mini-Uber") that scans for nearby auto-rickshaw or taxi drivers and displays their details to students, enabling direct contact and booking. This approach aims to reduce wait times, increase ride options, and improve overall campus mobility.

Objectives and Vision

- **Quick Access to Rides:** Enable students to find the nearest available drivers instantly.
- **Direct Communication:** Provide an in-app call/chat feature so students can negotiate and book rides without intermediaries.
- **User-Friendly Interface:** Offer a simple mobile/web UI where students can see driver names, vehicle details, and ratings at a glance.
- **Quality and Safety:** Include a driver-rating system for accountability and an emergency contact/safety feature for security [3] [4].
- **Scalable Platform:** Develop a web and smartphone app (Android/iOS) with a backend database, so new drivers can register and data is centrally managed (multi-platform compatibility as a key requirement [5]).

These goals align with key ride-sharing app features: for instance, PRDs for ride-hailing apps typically emphasize ride requests, driver ratings, and secure payments [6]. In our case, we simplify payments (cash or direct bargaining) and focus on swift matching and communication.

User Profiles and Personas

- **Student Riders:** Students who use the app to find rides. They need to see available drivers nearby, contact them quickly, and rate their service afterward.
- **Driver Partners:** Local auto/taxi drivers who register on the app, specifying their vehicle type and availability. They should have a profile (name, photo, phone) and can update their status or location.
- **Administrator:** A backend admin (or team) that manages the driver database, verifies new driver sign-ups, and monitors app usage.

Key Features (Functional Requirements)

- **User Registration & Login:** Secure sign-up for students and drivers. Students may log in via school email/SMS, while drivers register with their license and vehicle details. (User registration is a fundamental requirement in campus mobility systems [5].)
- **Location-Based Discovery:** The app uses the device's GPS or browser geolocation to find drivers within a certain radius. A map or list view shows drivers sorted by proximity. Users can select a destination or accept the default (home) campus location. *(If full real-time tracking is not yet implemented, an initial version can simply list drivers that were pre-registered around that area.)*
- **Driver Profiles & Listings:** Display each available driver's name, photo (optional), vehicle type, and average rating. Include any notes (e.g. luggage capacity). This transparency builds trust – ride-hailing research emphasizes that profiles with ratings "build trust and encourage safer interactions" [7].
- **Real-Time Status Updates:** Once a student selects a driver, the app shows the driver's status (e.g. "En route," ETA). Real-time GPS tracking of the driver's location (if implemented) allows the student to see exactly when the ride will arrive [8]. Such live updates add transparency and reduce uncertainty about wait times [8] [9].
- **Contact Driver:** A one-click call or secure messaging feature lets the student contact the driver directly to confirm pickup details. Integrating an in-app call/chat (with phone-number privacy) helps resolve pickup issues quickly [4]. For example, studies advise including chat/call functions so that drivers and passengers can coordinate without sharing personal contact info [4].
- **Rating & Feedback:** After each trip, students can rate and review the driver. Display each driver's average rating in their profile. A visible rating system helps maintain high service standards (as recommended for taxi apps [3] [7]).
- **Backend System:** A server and database to store user/driver profiles, availability status, ride requests, and ratings. The backend will handle requests quickly to ensure responses (like listing drivers) happen within a couple of seconds. The system should scale to support many concurrent users even if the user base is small (a scaled-down equivalent of ride-sharing backend constraints [10]).
- **Multi-Platform App:** The solution includes both a web application and a mobile app (Android/iOS). This ensures students can use the service on any device [5]. Web and mobile clients share the same backend APIs.
- **Security & Privacy:** Encrypt user passwords and sensitive data. Only verified drivers (approved by admin) can appear in search results. Implement an optional SOS/emergency feature so a student can quickly alert campus security if a ride feels unsafe [3].
- **Notifications:** Optional push/SMS notifications (e.g. "Your driver is 2 mins away" or "New driver matches your route"). Quick alerts improve the user experience by keeping students informed without constant checking.

Non-Functional Requirements

- **Usability:** The interface must be intuitive. Users should perform tasks (search, select, call) in as few taps/clicks as possible [11]. A clear menu and straightforward icons ensure new users can easily navigate the app.
- **Performance:** Driver search results should appear within ~2 seconds of the request. The app must handle peak campus usage (e.g. after classes) without lag. (For reference, a scaled-down Uber app targets sub-5-second response times for one million users [10].)

- **Reliability:** The system should be available 24/7, as students may need rides at all hours. Downtime must be minimal. The app should handle intermittent connectivity (e.g. cache the last known driver list if offline).
- **Scalability:** While initial deployment is small-scale, design the backend so it can add more users and drivers easily (e.g. cloud-based servers that can scale).
- **Safety & Quality:** Implement routine checks (e.g. periodically verify driver documents). Maintain a driver rating threshold: drivers below a rating may be deactivated. As one strategy report notes, offering a rating system helps "ensure high standards" [3].
- **Platform Compatibility:** The app will support major web browsers and Android/iOS OS versions. Use responsive design for the web version and native components for mobile to ensure a consistent experience [5].
- **Data Security:** Comply with data protection norms (e.g. users' phone numbers are not exposed to others; use HTTPS). Adhere to any campus policies for student data.

System Architecture

1. **Frontend (Client):** A web frontend (e.g. React) and mobile apps (e.g. Flutter or React Native) with matching UI. Both allow students to log in, view nearby drivers on a map or list, and call a selected driver.
2. **Backend (Server):** A REST API (e.g. Node.js/Python) that handles requests: authenticating users, querying driver locations, storing ratings, etc. It connects to third-party services (e.g. Google Maps API for geolocation, a telephony API like Twilio for call linking).
3. **Database:** Stores user accounts, driver profiles, current locations, ride history, and ratings. A simple relational or NoSQL database suffices (e.g. MySQL, PostgreSQL, or MongoDB).
4. **Location Service:** If implementing real-time tracking, each driver's app can periodically send GPS coordinates to the server; the server broadcasts this to the requesting student's app. This could use Google Maps or similar for mapping.
5. **Communication Module:** For calls, use the phone's dialer (via a `tel:` link) or integrate an in-app calling SDK. Messaging could be a simple chat feature backed by the server.
6. **Admin Tools:** A small web admin panel to add/approve drivers in bulk (for the first trial, as drivers are onboarded, an admin can feed their details into the system).

(In the initial MVP, manual upload of driver data is acceptable – we "gather details and feed to system" before launch. Later, we enable drivers to sign up through the app.)

User Stories

- *As a student,* I can open the app and immediately see a list of available drivers near me, so I can choose someone without delay.
- *As a student,* I can tap a driver's profile to see their name, vehicle info, and rating.
- *As a student,* I can call or message the driver directly from the app, so I can arrange the pickup quickly.
- *As a student,* after my ride, I can rate the driver on a 1–5 star scale, giving feedback for future users.
- *As a driver,* I can register on the platform, specify my service area, and update when I'm available to pick up passengers.
- *As a driver,* I can view messages from students and respond to ride requests.

- *As an admin,* I can manage the driver database (e.g. verify new drivers, remove banned ones) and monitor app usage.

These use cases cover the core flow of finding, booking, and reviewing a ride.

Development Roadmap (MVP Milestones)

1. **Phase 1 – MVP:** Build the driver directory and search interface. Populate the database with a sample list of drivers (names, contacts, locations). Implement student login, driver listing by proximity, and a tap-to-call function. Include basic rating (simple form).
2. **Phase 2 – Location & Real-Time:** Integrate GPS mapping so drivers can share live location. Show moving markers on the map or updated ETAs. Refine UI/UX based on user feedback.
3. **Phase 3 – Enhancements:** Add advanced features like push notifications, ride history, and data analytics. Consider AI-based features (e.g. predicting peak demand or recommending drivers). (For instance, data analytics could predict high-demand periods, and AI could estimate arrival times [12].)
4. **Phase 4 – Testing & Deployment:** Conduct user testing with students, gather feedback, and iterate. Prepare for small launch in one campus and scale as needed.

Constraints & Risks

- **Driver Participation:** Success depends on enough local drivers signing up. We mitigate this by initial incentives (e.g. campus posters, small perks) to drive registration.
- **Data Accuracy:** Initially, driver data may be manually entered, risking outdated info. We must regularly verify or allow drivers to update their status.
- **Network Dependence:** Real-time features need good mobile coverage. We should support offline mode to an extent (e.g. show last known list if connection drops).
- **Safety Concerns:** Any ride-sharing involves safety risks. Implement background checks for drivers and the in-app emergency button to address this [3].

Summary

This PRD outlines a campus-focused ride-sharing app to improve student transport. By combining a user-friendly interface with key features – nearby driver discovery, direct contact, real-time status, and ratings – we aim to reduce waiting time and give students more travel options. As one design guideline puts it, features like real-time GPS tracking and in-app calling “enhance transparency, safety, and user coordination” [8] [4]. With a robust backend and iterative development, this solution will offer a practical “mini-Uber” for our college community.

Sources: Authoritative guides and studies on ride-sharing and campus mobility [2] [6] [1] [3] [8] [9].

---

[1] Student transportation | Uber HigherEd
https://www.uber.com/us/en/transit/higher-education/student-transportation/

[2] [5] Designing a Secure and Efficient Campus Carpooling System: A Comprehensive Software Requirement Specification[v1] | Preprints.org
https://www.preprints.org/manuscript/202503.0991

3 12 Strategies and tactics for establishing a reliable and user-friendly taxi service
https://www.tability.io/templates/strategies/t/nfVlgTZBER5X

4 7 8 11 Essential Features for Ride-Hailing Apps
https://www.ridewyze.com/blog/essential-features-for-ride-hailing-apps

6 Mobile App Requirements Document: Template & 5-Step Guide – NIX United
https://nix-united.com/blog/how-to-write-a-proper-mobile-app-requirements-document-in-5-steps/

9 10 Software Requirements Document For Uber | PDF | Mobile App | User Interface
https://www.scribd.com/document/703571917/Software-Requirements-Document-for-Uber