Requirements

Function Requirements

# Core Ride-Sharing Functions

1. User Registration and Digital ID verification
   * Allow students/faculty/staff to sign up using their campus credentials and verify identity via government‐ID/ID card selfie matching in real timeUser Profile Management
   * Support multi-factor authentication for additional security.
2. Geolocation and Route Optimization
   * Use GPS to select pick-up/drop-off points and calculate optimal routes based on real-time traffic
3. Real-time Vehicle Tracking
   * Show live map tracking of assigned driver’s approach to pick-up, with ETA
4. Fare Estimation and Cost Splitting
   * Calculate shared-ride costs dynamically; split payment among riders automatically
5. Payment Processing and Multiple Payment Options
   * Integrate credit/debit, campus account billing, mobile wallets, and (if needed) cash toggles
6. Payment History
   * Shows the history and the details of the payment
   * Such as, drivers name, vehicle type, vehicle plate number, payment time and date, payment method and the location.
7. In-App Communication and Notifications
   * Offer chat/call features between riders and drivers
   * push notifications for ride status updates and campus alerts
8. Ride Scheduling and Reservations
   * Allow advance booking for carpool, with easy cancellation/reschedule options
9. User Profile Management
   * Enable users to create and manage profiles including name, contact information, vehicle details (for drivers), preferences and rating

# Parking System Integration Functions

1. Real-Time Parking Availability
   * Integrate with campus sensors or gate systems to display live occupancy levels of each lot/garage
2. Parking Reservation and Prebook
   * Let users reserve specific parking spots when they book a ride or need individual parking
3. Automated Entry / Exit Gate Control
   * Use RFID or license-plate recognition at campus gates to automate barrier lifts based on reservation/permit status
4. Dynamic Pricing and Incentives
   * Adjust parking fees by demand (peak vs. off-peak) and offer ride-sharing discounts when lots are near capacity
5. Digital Parking Permit and Enforcement
   * Issue and validate digital permits; integrate citation management and enforcement workflows

# Safety and Security Functions

1. In-App Safety tools
   * SOS/emergency button, “Trusted Contacts” trip sharing, speed-alert notifications
   * Emergency contact number can be set in account settings
2. Driver Background Checks and Ratings
   * Display driver profiles (ratings, trip history) and enforce regular background screening
3. Vehicle and License Plate Verification

* Cross-check vehicle registration and plate numbers against approved campus list

# Administrative and Analytics Functions (Admin Account)

1. Admin Dashboard
   * Oversee ride-matching activity, parking utilization, permit issuance, and system health
2. Reporting and Usage Analytics
   * Generate reports on ride-share adoption, parking lot occupancy trends, peak demand times, and revenue
3. Policy and Permission Management
   * Configure user roles (student, staff, visitor), set approval workflows for permits, and adjust pricing rules
4. Alerts and Maintenance
   * Notify campus operators of low-battery sensors, gate malfunctions, or parking lot emergencies

# Feedback and Continuous Function

1. Ratings and Reviews
   * Collect rider and driver feedback after every trip to identify issues and drivers of satisfaction
2. Suggestion Function
   * Allow the user to give suggestions if there is anything can make improvements

# Multi-Platform Access

1. Mobile app for iOS and Android
   * Able to run on android and iOS platform without any bug or issues about the application and display interface
2. Web Platform

* Web portal for desktop access and administrative tasks.

Performance Requirements

# Response Time

* Target: ≤ 2 seconds for any UI-driven operation; ≤ 500 ms for internal API call.
* Worst-case: Under peak load, 95th-percentile response times must stay under 5 seconds to avoid user frustration.

# Throughput

Target:

* Initial launch: ≥ 200 requests/second sustained.
* Two-year horizon: Scale to ≥ 1,000 requests/second without degradation in response times.
* Measurement: Monitor both peak and average throughput during business-hour load tests.

# Scalability

Target:

* Horizontal scaling: Support adding application instances behind a load balancer to maintain performance linearly up to anticipated peaks (e.g., 5,000 concurrent sessions).
* Predictive scaling: Auto-scale when CPU > 60 % or request queue length exceeds threshold.

# Availability & Reliability

* Target: ≥ 99.9 % (“three nines”) per month (≤ 43 minutes downtime) Wikipedia.
* Target: MTTR ≤ 15 minutes; MTBF ≥ 30 days.

# Concurrency & Capacity

* Target: Support 5,000 concurrent authenticated sessions, rising to 10,000 within two years.

Data Volume:

* Trip records: Handle storage and retrieval of up to 1 million trips per month.
* Parking updates: Process real-time parking‐lot sensor feeds at 1 update/second per lot.

# Latency (Inter-Service)

* Target: ≤ 100 ms per RPC; ≤ 50 ms per database query under normal load.

# Resource Utilization

* CPU Usage: Keep average CPU utilization per node below 70 % during business-hour peaks
* Memory Usage: Average RAM usage < 75 % of capacity; memory leaks must not exceed 1 % of heap over 24 hours.
* Disk I/O & Network: Ensure average disk write latency < 10 ms; network throughput per node ≥ 500 Mbps.

# Peak Load & Stress Conditions

* Load-Testing Goals:
* Simulate 2× expected peak for 30 minutes with no SLA breaches.
* Ramp tests: Increase load by 10 % every 5 minutes until 200 % of target throughput.
* Error Budget: Allow no more than 0.1 % of requests to fail during sustained peak.

Usability Requirements

# Ease of Use

* Intuitive Interface: The application should have a clean and straightforward interface, allowing users to navigate effortlessly without prior training.
* Quick Onboarding: New users should be able to register and start using the app within a few minutes, with clear guidance provided throughout the process.
* Consistent Design: Maintain uniformity in design elements across all screens to reduce the learning curve and enhance user familiarity.

# Accessibility

* Multi-Platform Support: Ensure the application is accessible on various devices, including smartphones, tablets, and desktops, to cater to different user preferences.
* Language Options: Provide multiple language options to accommodate the diverse university community.
* Assistive Technologies Compatibility: Design the app to be compatible with screen readers and other assistive technologies to support users with disabilities.

# Efficiency

* Minimal Steps for Core Tasks: Users should be able to perform primary actions, such as booking a ride or checking parking availability, in as few steps as possible.
* Fast Load Times: The application should load quickly, with minimal delays, to enhance user satisfaction.
* Real-Time Updates: Provide users with real-time information on ride statuses and parking availability to facilitate timely decisions.

# Error Management

* Clear Error Messages: When errors occur, the application should display understandable messages that guide users on how to resolve the issue.
* Undo Options: Allow users to easily reverse actions, such as canceling a ride or changing a parking reservation, to prevent frustration.
* Robust Validation: Implement input validation to prevent common errors, such as entering invalid dates or times.

# Memorability

* Logical Navigation Structure: Organize the application's navigation in a logical manner, allowing users to remember how to access different features easily.
* Visual Cues: Use icons and visual indicators consistently to help users recall functions and actions.
* Frequent Actions Accessibility: Place commonly used features in easily accessible locations to facilitate quick access.

Interface Requirements

# Clear Navigation

* Persistent Menu: A simple tab bar (e.g., Rides / Parking / Profile) or hamburger menu lets users jump between main sections in one tap.
* Breadcrumbs or Back Button: Always show a clear “Back” control when drilling into sub-screens to prevent feeling lost.

# Minimalist Layouts

* Reduce Clutter: Only display core functions on each screen; hide secondary options under “More” or in settings.
* Generous White Space: Use breathing room to group related elements and guide the eye.

# Touch-Friendly Controls

* Adequate Hit Areas: Make all tappable buttons and icons at least 44×44 px to prevent mis-taps on mobile.
* Standard Components: Leverage native buttons, toggles, and dropdowns so users instantly recognize how to interact.

# Immediate Feedback

* Loading Indicators: Show spinners or skeleton screens whenever data is loading to reassure users.
* Action Confirmation: Use toasts or banners for success messages (e.g., “Ride booked!”) and clear error alerts near the relevant field.

# Basic Accessibility

* Text Alternatives: Provide labels for icons and images so screen readers can announce them.
* Contrast & Scaling: Ensure text meets a 4.5 : 1 contrast ratio and allow users to adjust font size within the app.

# Responsive Behavior

* Multi-Device Support: Design layouts that adapt at key breakpoints (phone, tablet, desktop) without hiding core features.
* Orientation Handling: Ensure content reflows gracefully in both portrait and landscape model.

# Consistent Branding

* Unified Style Guide: Use a shared component library (colors, typography, icon set) so every screen feels like part of the same system.
* Campus Identity: Incorporate your university’s logo and color palette in header or tab bar elements for instant recognition.