Software Requirements Specification

Taxi Tap by Git It Done

Table of Contents

Introduction

Taxi Tap is a mobile platform designed to revolutionize South Africa's minibus taxi industry by digitizing route information, eliminating the need for constant hooting, and creating a semi-structured booking system while preserving the flexibility that makes taxis an essential mode of transport. The system connects passengers and taxi operators through a location-aware mobile application that facilitates taxi requests, communicates passenger locations, manages payments, and provides real-time vehicle tracking – all without fundamentally changing the existing system's multi-passenger, flexible route nature.

User Characteristics

The users of the Taxi Tap system are expected to fit into the following groups:

Driver User Characteristics

Attribute	Description
Familiarity with Mobile Technology	Varies widely: - Some drivers may be tech-comfortable while others may struggle with apps.
Access to Reliable Internet and Data	Often limited or inconsistent, drivers operate in areas with poor signal or data is expensive.
Preferred Language and Communication Style	They may prefer local languages (e.g. Zulu, Xhosa, Sesotho).
Attention Capacity While Driving	Must be able to use the app **while operating a vehicle,** requiring minimal taps and distractions.
Trust and Skepticism Toward New Technology	May be skeptical of new digital systems due to fear of replacement, surveillance, or fare manipulation.
Goals and Incentives for Using the App	Wants more passengers, quicker pickups, and less idle time without changing their daily routine.

Passenger User Characteristics

Attribute	Description
Digital Literacy	Ranges from students and workers (tech-savvy) to commuters with limited app experience.
Access to Reliable Internet and Data	Frequently encounters **low or no connectivity,** especially in transit
Reasons for Using the Platform	It seeks reliable transport, less waiting, and a safer way to locate and use taxis.
Preferred Language and Communication Style	They may prefer local languages (e.g. Zulu, Xhosa, Sesotho).

App Usage Context (Where & When)	Often uses the app in crowded, noisy, or busy settings like taxi ranks.
Concerns Around Trust and Safety	Wants to be sure drivers are legitimate and that their location and personal data are protected.
Platform Interaction Needs	Need to discover taxis, request rides, track driver arrival, and receive ride notifications.

User Stories

Passenger User stories

User Story	Acceptance Criteria	Definition of Done
Account Registration & Login As a passenger, I want to sign up and log in to my account, so that I can securely access and use the Taxi Tap app.	Given that I am on the app's welcome screen, When I choose "Sign Up" or "Log In" and enter valid details, Then I should be authenticated and taken to the home screen.	Based on my input criteria, I am taken to the home page of Taxi Tap
View Available Taxis and Routes As a passenger, I want to view available taxis and their routes on a map, so that I can choose one that matches my travel needs.	Given I am logged in, When I open the home screen, Then I should see nearby taxis on a map with route or destination labels.	The map displays icons of nearby taxis, including route or destination tags, when available.
Set Pickup and Destination As a passenger, I want to share my location and set a destination, so that drivers can find and pick me up efficiently.	Given that I have granted location access, When I enter or select a pickup and destination point, Then the app should confirm my trip details and show nearby taxis.	Pickup and destination are confirmed and displayed; nearby taxis are suggested based on the selected route.
Book a Seat and Get Confirmation As a passenger, I want to book a seat on a taxi and receive confirmation, so that I'm guaranteed a spot before the taxi arrives.	**Given** I've selected a taxi, **When** I tap "Book Seat" and confirm, **Then** I should receive a booking confirmation and a ride status update.	A booking confirmation message appears with the selected taxi details and current ride status.
Track Assigned Taxi in Real-Time As a passenger, I want to track my assigned taxi in real time, so that I know when and where to expect pickup.	Given my booking is confirmed, When I open the tracking screen, Then I should see the taxi's live location and estimated time of arrival.	The assigned taxi is visible on the map with a live location marker and updated ETA.
Receiving Alerts When Taxi is Nearby	Given the assigned taxi is approaching,	A push alert is triggered and received once the taxi

As a passenger, I want to receive alerts when the taxi is nearby, so that I can be ready at the pickup location.	When it is within 500 meters, Then I should receive a push notification that it's nearby.	enters the defined proximity radius.
See Available Seats As a passenger, I want to see how many seats are available, so that I can decide whether to book a seat or wait.	Given I view a taxi on the map or booking screen, When I open its details, Then I should see the number of available seats.	The number of available seats is clearly shown for each listed or selected taxi.
Rate Completed Trip As a passenger, I want to rate my trip after completion, so that I can provide feedback to help improve the service.	**Given** my ride has ended, **When** I open the app, **Then** I should be prompted to leave a 1–5 star rating and optional comments.	The rating form appears automatically after the ride ends, and feedback is successfully submitted to the system.
Use App Offline or on Low Bandwidth As a passenger, I want to use the app offline or on low bandwidth, so that I can still interact with core features in areas with poor connectivity.	**Given** I have limited internet access, **When** I open the app, **Then** I should still be able to view saved routes, taxis, and queue a ride request that sends once reconnected.	The app functions with cached map data and stores ride requests locally, syncing once connectivity is restored.

Driver User stories

User Story	Acceptance Criteria	Definition of Done
Account Registration & Login As a driver, I want to sign up and log in to my account, so that I can securely access and use the Taxi Tap app.	Given that I am on the app's welcome screen, When I choose "Sign Up" or "Log In" and enter valid details, Then I should be authenticated and taken to the home screen.	Based on my input criteria, I am taken to the home page of Taxi Tap
Announce Route & Destination As a driver, I want to input the route I will be taking and the destination, so that passengers can see if I'm heading in their direction.	**Given** that I'm logged in, **When** I set my starting point and destination, **Then** the route is visible to nearby passengers.	The route is stored and displayed to the eligible passenger's interface.
(Go Online/Offline) As a driver, I want to go online or offline as needed, so that I can control when I am available to receive ride requests.	**Given** that I'm on the driver dashboard, **When** I tap "Go Online" or "Go Offline", **Then** my status is updated accordingly and affects request visibility.	The driver's online/offline status is reflected, and the passenger can no longer see the taxi on the map.

Receive Ride Requests As a driver, I want to receive ride requests from nearby passengers, so that I can choose which pickups to accept.	**Given** that I am online and have an active route, **When** a passenger requests a ride, **Then** I receive a notification with request details.	Ride requests from matching passengers are delivered in real-time to the driver's interface.
Accept or Decline Requests As a driver, I want to accept or decline a ride request, so that I can manage my route and taxi capacity efficiently.	**Given** I have received a ride request, **When** I tap "Accept" or "Decline", **Then** the system updates the request status and notifies the passenger.	Accepted rides appear on the active list; declined requests are logged and cleared.
View Passenger Pickup Details As a driver, I want to see the passenger's pickup point and basic information, so that I know where to stop and who I'm picking up.	**Given** that I've accepted a booking, **When** I view the trip summary, **Then** I should see the passenger's location and name or contact info.	Pickup details are accurately displayed on the driver's map and trip screen.
View Map & Navigation As a driver, I want to see a map with passenger pickup and route directions, so that I can navigate efficiently.	**Given** that I have one or more assigned pickups, **When** I open the map view, **Then** I should see my location and passenger's location.	Live maps with GPS and routing is functional and accurate within the app.
Update Seat Availability As a driver, I want to update how many seats are available in my taxi, so that passengers can decide whether to book or wait.	**Given** that I've started a trip or gone online, **When** I adjust seat count manually, **Then,** passengers see the updated availability.	Seat count updates in real time and is reflected in the passenger's booking screen.
Receive Alerts for New Requests or Updates As a driver, I want to receive real-time notifications, so that I don't miss ride requests or updates while driving.	**Given that** I am online, **When** a new request or important event occurs, **Then** I receive a push notification with the relevant details.	Push and in-app alerts trigger correctly and lead to actionable pages.
Work Offline (Partial Functionality) As a driver, I want to continue using key features even when I'm offline, so that I can operate in areas with poor connectivity.	**Given** that I am offline or have poor signal, **When** I open the app, **Then** I should be able to see cached routes and queue ride requests.	The app stores critical data locally and syncs changes once reconnected.

Service Contracts

<u>User Service Contract</u>

1. createUser
    a. Request:

        {
        name: string,
        phoneNumber: string,
        password: string
        }

    b. Response

        {
        userID: string,
        createdAt: timestamp
        }

    c. Effect
        i. Saves a new user to the database and returns the generated user ID and creation timestamp.
2. LoginUser
    a. Request:

        {
        phoneNumber: string,
        password: string
        }

    b. Response

        {
        token: string,
        userId: string
        }

    c. Effect
        i. Authenticates the user and provides a session token.

Ride Booking Service Contract

1. ViewAvailableTaxis
    a. Request

        {
        userLocation: { latitude: float, longitude: float }
        }

    b. Response

        [{
        taxiId: string,
        driverName: string,
        availableSeats: int,
        route: string,
        location: {latitude: float, longitude: float}
        }]

    c. Effect
        i. Returns a list of nearby taxis with route, price and seat information.
2. bookSeat

a. Request
    {
    userId: string,
    taxiId: string,
    pickupLocation: {latitude: float, longitude: float },
    destination: string
    }
b. Response
    {
    bookingId: string,
    status: confirmed,
    estimatedArrivalTime: timestamp
    price: double
    }
c. Effect
    i. Reserves a seat in the taxi and returns confirmation.

3. trackTaxi
    a. Request
        {
        bookingId: string
        }
    b. Response
        {
        taxiLocation: {latitude: float, longitude: float},
        estimatedArrivalTime: timestamp
        }
    c. Displays live taxi location and ETA.

<u>Ride Receiving Service Contract</u>

1. UpdateTaxiLocation
    a. Request
        {
        driverId: string,
        location: {latitude: float, longitude: float}
        }
    b. Response
        {
        status: updated
        }
    c. Effect
        i. Updates the taxi's real-time location in the system.
2. AcceptBooking
    a. Request
        {
        driverId: string,
        bookingId: string
        }

   b. Response
      {
       status: accepted
      }
   c. Effect
      i. Confirms the driver’s acceptance of a user’s booking.
3. CompleteRide
   a. Request
      {
       bookingId: string
      }
   b. Response
      {
       status: completed,
       fare: double
      }
   c. Effect
      i. Marks the ride as completed.

Notification Service Contract

1. sendNotification
   a. Request
      {
       userId: string,
       type: string,   e.g., rideStatus
       message: string
      }
   b. Response
      {
       status: sent,
       timestamp: timestamp
      }
   c. Effect
      i. Sends a notification to the user.

Review And Rating Service Contract

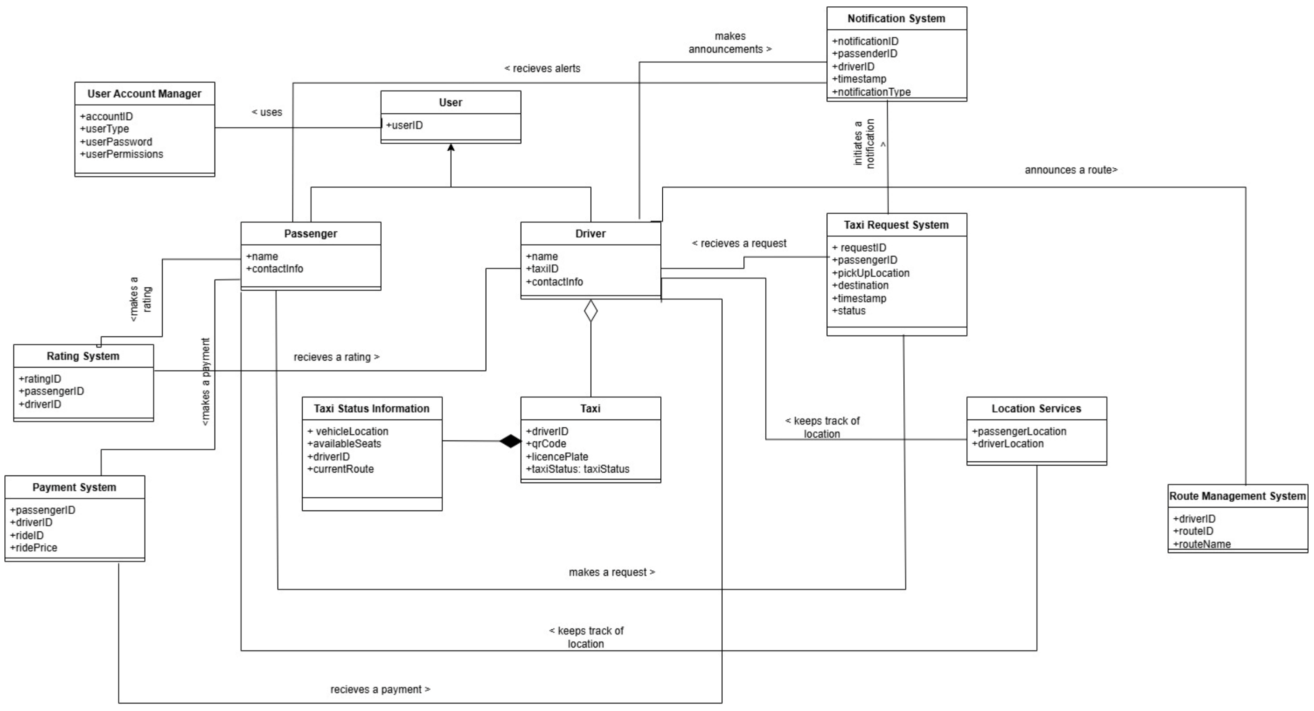
1. SubmitReview
   a. Request
      {
       bookingId: string,
       userId: string,
       driverId: string,
       rating: int, // 1 to 5
       reviewText: string
      }
   b. Response
      {
       reviewId: string,

status: submitted

}

c. Submits and stores a review for the ride.

Domain Model



Functional Requirements

R1: User Account Management

- R1.1: Users should be able to register as either a driver or a passenger
- R1.2: Users should be able to update their Profile information
- R1.3: The system should support role-based access control for passenger and driver interfaces
- R1.4: Users should be able to reset or change their passwords

R2: Location Services

- R2.1: The system should track driver locations in real-time using GPS
- R2.2: The system should determine passenger locations for pickup requests
- R2.3: The system should calculate proximity between taxis and passengers
- R2.4: The system should send proximity alerts to notify passengers when their requested taxi is approaching
- R2.5: The system should display estimated time of arrival for approaching taxis

R3: Taxi Request System

- R3.1: Passengers should be able to request taxi pickups based on their location
- R3.2: Passengers should be able to see nearby available taxis
- R3.3: Drivers should be notified of nearby passenger pickup requests
- R3.4: Drivers should be able to accept or decline pickup requests
- R3.5: Passengers should be able to specify their destinations

R4: Route Management

- R4.1: The system should allow drivers to announce their routes
- R4.2: The system should display taxi routes to passengers
- R4.3: The system should allow drivers to indicate their destinations
- R4.4: The system should support flexible drop-off points along routes
- R4.5: The system should optimize routes based on multiple passenger pickup/drop-off points
- R4.6: The system should display route information in a visual format suitable for quick comprehension

R5: Taxi Status Information

- R5.1: The system should display real-time taxi tracking showing vehicle location
- R5.2: The system should show available seats in approaching taxis
- R5.3: The system should allow drivers to update their seat availability status
- R5.4: The system should indicate taxi status (en route, picking up, full, etc.)
- R5.5: The system should notify waiting passengers when taxis reach capacity

R6: Notifications

- R6.1: The system should send push notifications for taxi proximity alerts
- R6.2: The system should notify passengers when their requested taxi accepts or declines the pickup
- R6.3: The system should notify drivers of new nearby passenger requests
- R6.4: The system should provide ETA updates to waiting passengers
- R6.5: The system should send notifications even with limited connectivity
- R6.6: The system should allow users to customize notification preferences

R7: Passenger Destination Management

- R7.1: The system should allow passengers to specify their drop-off locations
- R7.2: The system should support grouping passengers by similar destinations
- R7.4: The system should suggest optimal drop-off order to drivers

R8: User Interface

- R8.1: The system should provide separate interfaces for passengers and drivers
- R8.2: The system should offer a clean, easy-to-use interface with visual elements
- R8.3: The system should support multiple South African languages

R9: Rating and Feedback

- R9.1: Passengers should be able to rate drivers/taxis
- R9.2: The system should collect feedback on routes and service
- R9.3: Users should be able to anonymously report safety incidents

R10: Fare Management

- R10.1: The system should calculate fare estimates based on route and distance
- R10.2: The system should support both digital and cash payment options
- R10.3: The system should provide payment confirmation receipts
- R10.4: The system should track payment status for trips

R11: Taxi Identification

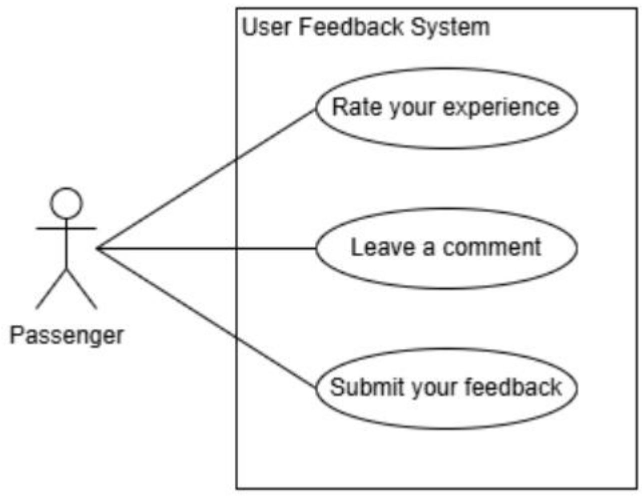
- R11.1: The system should provide unique identifiers for each taxi
- R11.2: The system should support QR code-based taxi identification and verification
- R11.3: The system should display taxi information (registration, operator) to passengers
- R11.4: The system should verify taxi authenticity through the system

R12: Safety Features

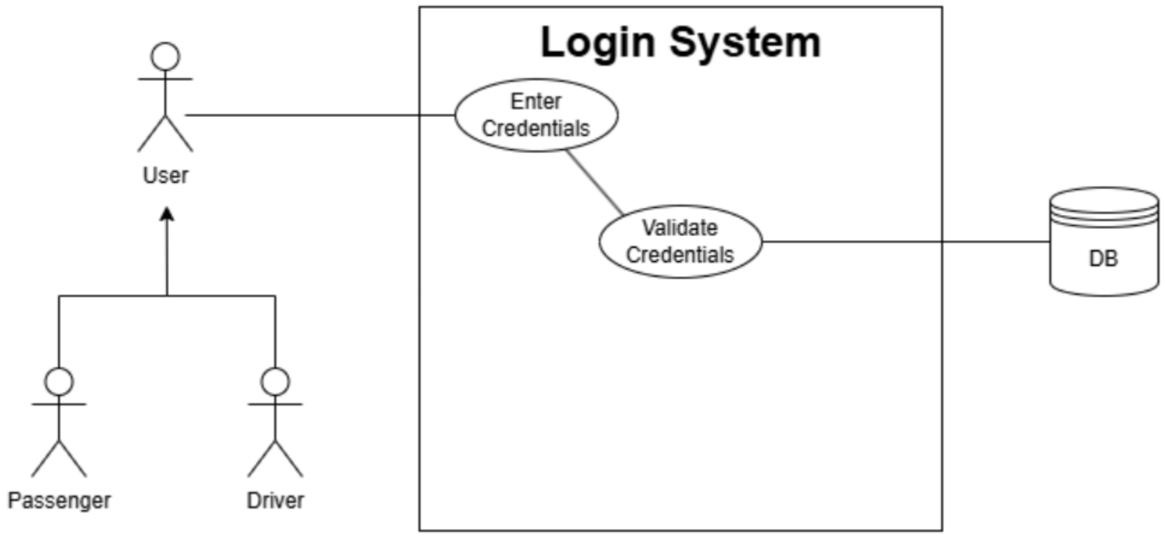
- R12.1: The system should provide an anonymous crime reporting tool
- R12.2: The system should include emergency contact features

Use Case Diagrams

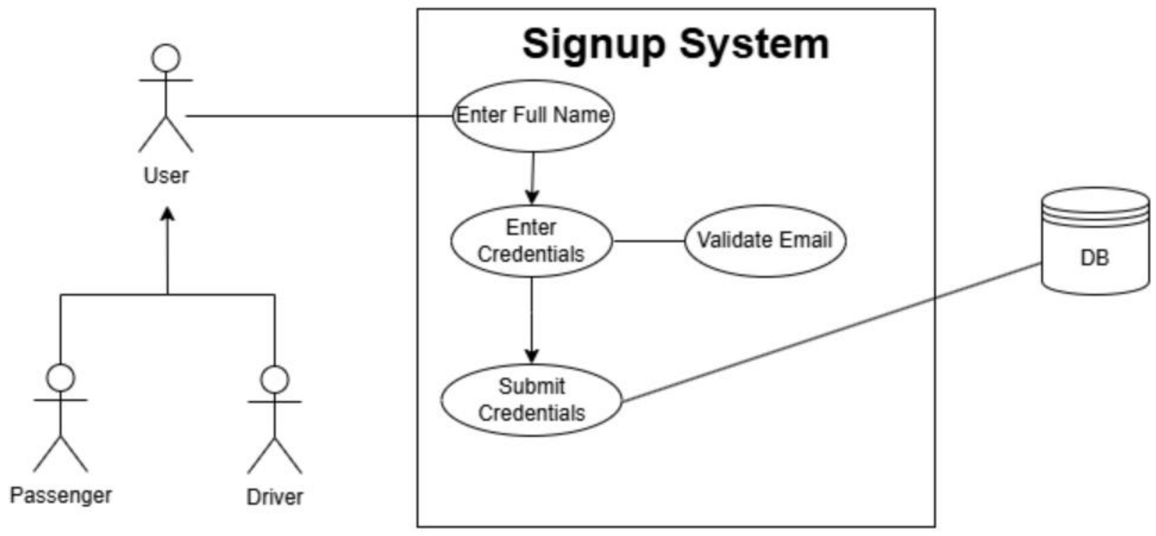
Feedback



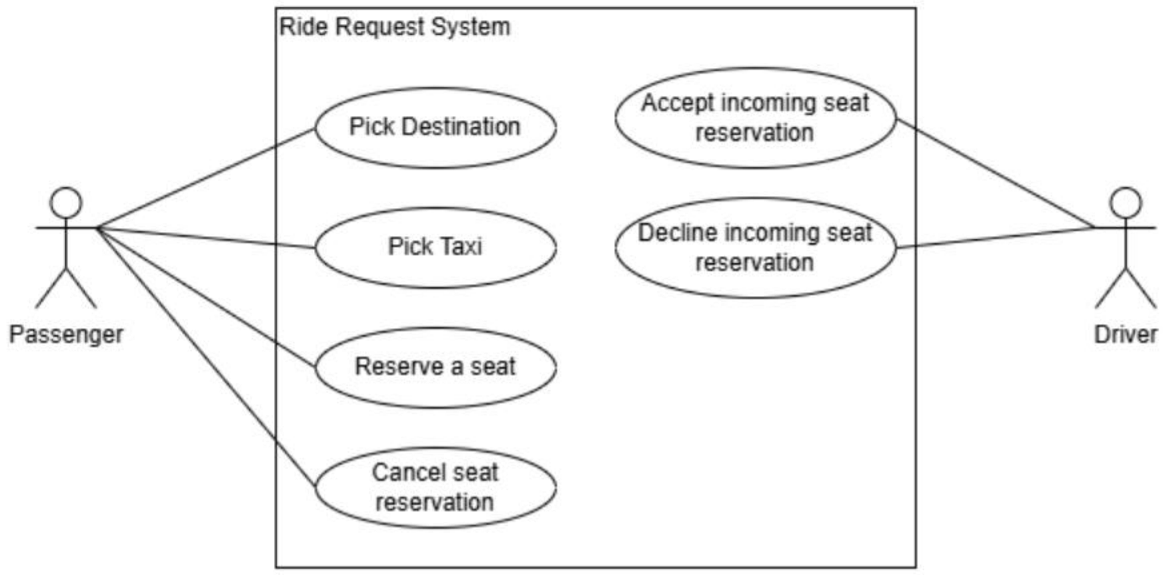
Login



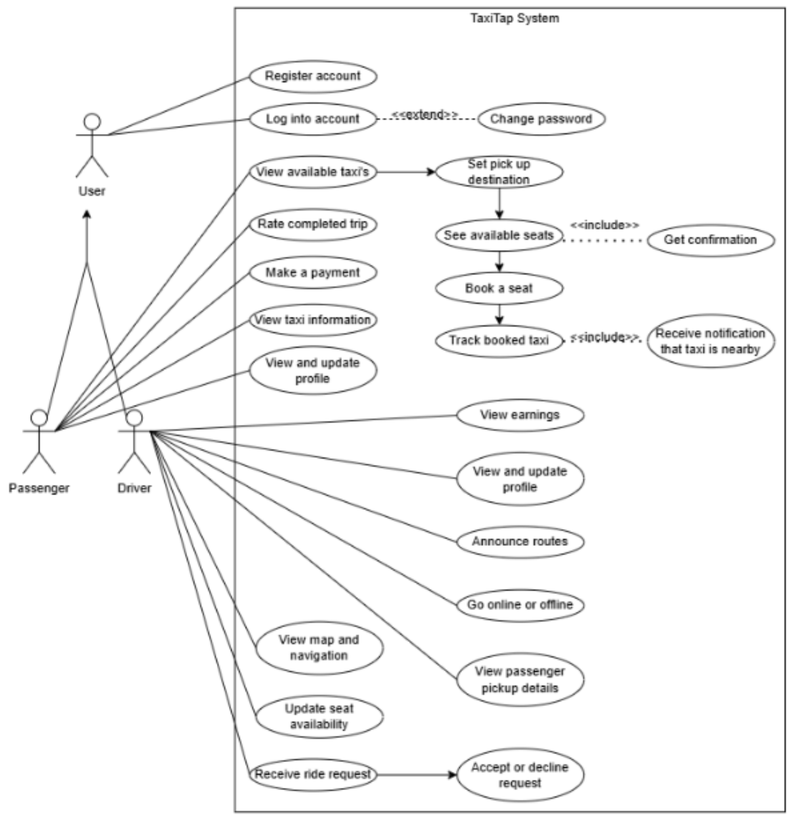
Signup



Ride Request



Overall System



Technology Requirements

Frontend

Expo (React Native with TypeScript)

Why we chose to use Expo?

- **Cross-platform Compatibility**: Code once, deploy to both Android and iOS.
- **Native Features**: Access to GPS, accelerometer, push notifications, offline storage, camera, QR scanning, etc.
- **Web Support**: Leverages Expo Web for rendering web-based dashboards and admin panels.
- **Live Reloading & Fast Iteration**: Expo Go provides hot reloading and rapid prototyping with a unified development experience.
- **Battery & Data Optimization**: React Native ecosystem provides fine-grained control over performance, reducing overhead.

Backend

Convex (TypeScript)

Why we chose to use Convex?

- **Truly Serverless**: No provisioning, no scaling headaches. Functions, database, and auth all run in one integrated environment.

- **Built-in Database**: Convex provides a powerful document-oriented database that supports relations, IDs, indexes, and real-time reactivity.

- **Type Safety**: Schema definition is in TypeScript, ensuring end-to-end type safety from backend to frontend.

- **Zero DevOps**: No need to manage infrastructure or containers. Deploy directly from your project.

- **Realtime Sync**: Built-in support for reactive queries allows passengers to see live taxi updates, seat availability, and ETA.

Convex Database Architecture

- **Document Store**: Convex uses collections of JSON-like documents, like MongoDB, but with built-in schema validation.

- **Indexes**: Automatic indexing on IDs and custom indexing for optimized query performance.

- **Relationships**: You can use Convex `v.id()` to reference documents between tables, ensuring referential integrity.

- **Realtime Subscriptions**: Query results update automatically when the underlying data changes.

Convex Free Tier (as of 2025)

- **Compute**: Up to 1 million function calls/month.

- **Storage**: 1 GB document data storage.

- **Bandwidth**: 5 GB of egress.

- **Authentication**: Integrated with third-party auth providers (Firebase Auth, Clerk, etc.).

- **Deployment**: 1 Production Deployment and 1 Dev Deployment per project.

Perfect for COS 301: Within budget, no surprise bills, and production-grade scalability.

Key Functional Modules & Implementation Plan

User Management Subsystem

- **Authentication**: Convex Auth with Clerk or Firebase integration.
- **Registration/Login**: Role-based registration (passenger or driver) with schema enforcement.
- **Profile Updates**: Mutation to update user document with profile fields.
- **Security**: JWT-based session validation, encryption at rest and in transit.

Location Services Subsystem

- **Driver Location**: Periodic GPS updates using Expo Location API.
- **Passenger Location**: One-time or continuous tracking during trip.
- **Proximity Alerts**: Triggered from Convex using background function.
- **ETA Calculation**: Naive approach using Haversine distance + average speed (no Google Maps API due to cost).

Taxi Request Subsystem

- **Request Workflow**:
  - Passenger sends request with coordinates and optional destination.
  - Nearby drivers notified (push notification via Expo).
  - Driver accepts or rejects request.
  - Status changes handled in real time.

Route Management Subsystem

- **Driver Route Declaration**: Input form for common route + destination.
- **Passenger View**: Map view of taxis on route + destinations.
- **Optimized Routing (Optional)**: Historical route optimization using stored patterns (stretch goal).

Notification System

- **Technology**: Expo Notifications API.
- **Use Cases**:
    - Taxi is approaching.
    - Ride accepted or declined.
    - Route changes or delays.
- **Offline Support**: Caching notifications locally using AsyncStorage.

Safety and Fare Management Subsystem

- **QR Identification**: QR codes linked to taxi documents in Convex.
- **Reporting**: Anonymous incident reports saved to a secure Convex table.
- **Fare Estimate**: Static fare matrix per route (e.g., km-based fare slabs).
- **Payment**: Optional - integrate with SnapScan/Yoco for digital payments.

Testing Frameworks

- **Backend**: Jest (unit and integration tests for Convex functions).
- **Frontend**: React Native Testing Library.
- **Manual Testing**: Device tests using Expo Go and emulators.

CI/CD

- **Convex Deployment**: Triggered via GitHub Action or manual `npx convex dev` / `convex deploy`.
- **Expo Deployment**: Use `eas build` + `eas submit` for App Store/Play Store releases.
- **Linting & Tests**: Pre-commit lint checks with ESLint + Jest unit tests.

Version Control

- GitHub repo with main and dev branches.
- Feature branches for each core module.

Architectural Requirements

Quality Requirements

Quality requirements determine the overall quality of Taxi Tap by specifying criteria that define how well the system performs and behaves.

1. Security
    a. Encryption: All data must be encrypted in transit and at rest using the best security practices.
    b. Compliance: Data capturing and storing must be adhered to the POPI act, ensuring data privacy and consent handling.
    c. Secure authentication: Users must authenticate securely, and sessions must be protected.
2. Usability
    a. Simplicity: the interface should be easy to use for people with varying levels of tech literacy.
    b. Accessibility: The use of clear labels, large tap targets and minimal steps to complete key tasks.
    c. Feedback and error handling: Provide real-time feedback for user actions, loading states and clear error messages when issues occur.
3. Scalability
    a. The backend must scale horizontally and vertically to handle fluctuations in user or data load without performance degradation. This is automatically done by our chosen backend.
4. Performance
    a. Low bandwidth optimization: The system must perform reliably under low-bandwidth or intermittent connectivity.
    b. Battery efficiency: The app must minimize CPU, GPS and network usage to extend battery life.
5. Reliability and Availability
    a. Offline Support: The app must function even without a constant internet connection, using local caching or data queuing mechanisms.
    b. High uptime: The system should be available with minimal downtime to support driver operations throughout the day.
    c. Data integrity: Ensure that data is not lost or duplicated during sync offline and online states.
6. Maintainability and Extensibility
    a. Clean architecture: Backend and frontend systems should be modular and loosely coupled to allow easier updates, fixes, or feature additions in the future.
    b. Logging and monitoring: Implement centralized logging and monitoring to quickly identify and resolve issues.
    c. Configurability: Support code configurations without needing code changes.
7. Affordability

a. Low data consumption: The app must use data sparingly to remain cost-effective for users in regions with expensive or limited mobile data.
b. Resource efficiency: The system should minimize server and client-side consumption to reduce infrastructure and battery costs.

Architectural Patterns

Architectural patterns are compositions of architectural elements which allow a system to effectively meet its quality requirements.

The Taxi Tap system consists of three components:

- The mobile interface for passengers and drivers
- The backend service that handles ride coordination
- The real-time notification system

To meet its performance, scalability, and usability goals under strict constraints (low bandwidth, battery efficiency, AWS Free Tier), our team plans to use a combination of architectural patterns that balance simplicity with flexibility.

How the patterns will be applied in the Taxi Tap system:

1. Event-Driven Architecture (EDA) with Publisher-Subscriber

The system will use the Publisher-Subscriber pattern to implement an Event-Driven Architecture. When a passenger requests a ride, an event is published (RideRequested) that triggers subscribed modules like ride matching, GPS location updates, and notifications. This design enables loose coupling, scalability, and real-time responsiveness, which are essential for a transportation app operating under varying loads.

Key Events: RideRequested, TaxiApproaching, PassengerWaiting, etc.

2. Client-Server Architecture

The system will use the Client-Server architecture, where the mobile application acts as a client and communicates with a backend server. The application handles user input, displays route and ride information, and temporarily stores data when offline. The backend processes requests like user authentication, route announcements, ride matching, and storing data.

We will use Convex as our serverless backend. This will allow us to write backend logic without managing infrastructure. Convex automatically handles scaling and storage, which helps us focus on building features rather than maintaining servers. It also supports real-time updates and is well-suited for low-latency mobile apps like Taxi Tap.

Implementation Strategies

To complement these architectural patterns, the following strategies will be considered.

1. Offline-First Strategy:

   Users may have limited internet connectivity, therefore, the mobile application should be designed with offline-first capabilities. Actions such as ride requests or location tracking are stored locally and queued for syncing when the connection is re-established. This ensures continuity in usage and improves the app's reliability in low-bandwidth environments.

2. Security Strategy

To meet the requirement for secure handling of user data in line with POPIA and best practices, the following strategies will be used

- Data in Transit:
  - All data exchanged between the mobile application and backend services will be encrypted using HTTPS with TLS (Transport Layer Security).
- Access Control:
  - Role-based access policies and authentication mechanisms (e.g., JWTs) ensure only authorised users can access specific system resources.

These strategies work within the architectural patterns to address the specific constraints of the South African minibus taxi ecosystem.

Design Patterns

Observer Pattern

Pattern Type: Behavioural

Participants:

- Subject: Notification System
- Observer: User
- Concrete Observer: Passenger, Driver

Explanation: The Observer pattern allows an object (User) to be notified automatically of state changes in another object (Notification System). This is ideal for handling events like route updates or ride status.

Example:

- User receives alerts from the Notification System.
- Notification System initiates a notification when a route is announced.

Mediator Pattern

Pattern Type: Behavioural

Participants:

- Mediator: Taxi Request System

- Colleague: Passenger, Driver

Explanation: The Mediator pattern centralizes complex communication between objects. Instead of Passenger directly interacting with Driver, requests are handled through the Taxi Request System.

Example:

- Taxi Request System acts as an intermediary between Passenger and Driver.
- Passenger makes a request for pickup to a driver, but the Taxi request system acts as the middleman for this request.

Constraints

The client laid out the following constraints, by which Taxi Tap must abide, in their specification.

1. All data must be encrypted at transit and at rest
   All data exchanged between the mobile application and backend services will be encrypted using HTTPS with TLS (Transport Layer Security).
   Role-based access policies and authentication mechanisms (e.g., JWTs) ensure only authorised users can access specific system resources.
2. POPI act
   To ensure we abide by this, we will not collect any user data that is not necessary for the functionality of the app. With that, we will have permission set up to ensure that users are comfortable with collecting info, such as the user's location. Furthermore, we will consider providing a Terms and Conditions for the app that lays out how user data will be used.
3. The app must function with low bandwidth, low data usage and be battery-efficient
   We will accomplish this by having a UI that does not use too many resources and lightweight calls to the API.
4. Budget
   We must use AWS Free Tier platforms or any platforms that are open source or within free tier allowance.