# **Taxi Tap: Architectural Strategy and Patterns**

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 etc.), 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, 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.**

1. **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.