# **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, 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. Layered Architecture with Clean Architecture Principles

Taxi Tap applies a layered architecture to its backend, separating responsibilities across presentation, application, domain, and infrastructure layers. This structure supports Clean Architecture principles by isolating business logic (use cases and domain models) from external dependencies like databases and frameworks. It allows each layer to evolve independently, improving testability and long-term maintainability.

* **Layers**:
  + Presentation Layer (API controllers)
  + Application Layer (use cases)
  + Domain Layer (business logic)
  + Infrastructure Layer (external integrations

### 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 mobile apps and backend services is encrypted using HTTPS with TLS (Transport Layer Security).**
* **Data at Rest:** 
  + **All user data stored in backend services (e.g., DynamoDB, S3) is encrypted using AES-256 encryption through AWS-managed services.**
* **Local Device Storage:** 
  + **Sensitive user data stored temporarily on mobile devices (e.g., offline ride requests) is encrypted using platform-specific secure storage APIs.**
* **Access Control:** 
  + **Role-based access policies and authentication mechanisms (e.g., JWTs, OAuth2) 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.

The two architectural patterns integrate to form a cohesive system:

* Event-driven architecture provides the communication backbone between components
* Layered Architecture organises the internal structure of each component