# Architectural Requirements
# Taxi Tap by Git It Done

# Contents

# 1 Architectural Design Strategy

**Strategy Chosen:** *Decomposition via Feature-Driven Development (FDD)*

Taxi Tap is built using a modular, feature-based decomposition strategy. Each functional system (e.g., User System, Vehicle System, Trip System) is designed, tested, and deployed independently. This strategy allows for:

- Clear modularity and separation of concerns.
- Parallel development and testing per feature.
- Easy onboarding and maintainability.
- Reduced risk when scaling or introducing new features.

# 2 Architectural Strategies

**Chosen Style:** *Event-Driven Architecture*

Event-driven architecture is centered around asynchronous communication between components. Components emit and react to events, allowing for real-time responsiveness, loose coupling and scalability. Why this is the best fit for our system:

- Real-time Interaction: Location updates, ride requests, driver availability, and notifications all benefit from real-time triggers and updates. Convex supports reactive data + background functions, making it a natural fit for an event-driven model.
- Asynchronous Processing: Tasks like sending push notifications, updating seat availability, or logging analytics should not block the main user flow. EDA allows these to run in the background, improving app responsiveness.
- Loose Coupling: With EDA, components (like driver matching and notifications) can be developed and deployed independently. This aligns well with Convex's function-based model, which is modular and reactive.
- Scalability and Maintainability: New features can easily be added by listening to events without modifying core components.

# 3 Architectural Quality Requirements

## 3.1 Quality attributes

The quality attributes for Taxi Tap are prioritized and defined as follows:

- **Availability:**
  **Why:** Ensures continuous service. If the app is down, users cannot book or accept rides, which impacts revenue and reputation.
  **Solution:** Maintain at least **99.5% uptime** under normal usage, with support for graceful degradation during failure.

- **Scalability:**
  **Why:** Supports many users using the app simultaneously and allows the system to handle growth in demand.
  **Solution:** The system must support at least **100 concurrent ride requests** with a backend response time of under **100ms**.

- **Usability:**
  **Why:** Users must be able to complete essential tasks easily, regardless of their technical skill level. Poor usability leads to frustration and abandonment.
  **Solution:** Use simple language, clear layout, and high-contrast color schemes. Employ interface metaphors such as intuitive icons to make navigation easy for users, including those who are technologically inexperienced.

- **Security:**
  **Why:** The system manages sensitive user data, including locations and contact details. Breaches can result in legal issues and loss of trust.
  **Solution:** Enforce **role-based access control (RBAC)** in all backend functions. Convex encrypts data both in transit (via HTTPS) and at rest, ensuring robust platform security.

- **Performance:**
  **Why:** Real-time features must respond quickly to meet user expectations. Delays in booking, navigation, or messaging reduce usability during peak hours.
  **Solution:** Ensure that the average response time of backend functions remains under **20ms**.

## 3.2 Architectural Strategies

- **Availability:**
  **Why:** If the system becomes unavailable, users cannot book or accept rides, damaging both reliability and trust.
  **Solution:**
  - **Replication:** Convex provides high availability by replicating data and functions across multiple geographic zones, ensuring continuity during failures.

- **Scalability:**
  **Why:** To support growing demand and simultaneous ride requests, the system must dynamically handle increased load.
  **Solution:**
  - **Horizontal scale-out:** Convex automatically scales infrastructure horizontally to meet demand.
  - **Data sharding:** Data is partitioned across shards by Convex without manual configuration, improving scalability.
  - **Asynchronous processing:** Non-critical tasks (e.g., sending notifications) are offloaded to background jobs using Convex async functions, freeing up system resources.

- **Usability:**
  **Why:** An intuitive and responsive interface is crucial to support users with varying levels of digital literacy.
  **Solution:**
  - **Real-time UI:** Keeps location and notification data live using subscriptions or efficient polling mechanisms.
  - **Responsiveness:** Improves user experience by minimizing UI latency and limiting the use of loading spinners to essential operations.
- **Security:**
  **Why:** The system manages sensitive user data, including real-time locations and account details. Breaches can result in legal and reputational harm.
  **Solution:**
  - **Secure communication:** All data in transit is encrypted using TLS by default.
  - **Role-based access control (RBAC):** Server functions enforce strict access controls based on user roles (e.g., driver vs passenger).
- **Performance:**
  **Why:** Real-time systems require fast responses. Delays in booking, tracking, or communication reduce system usability.
  **Solution:**
  - **Database indexing:** Convex uses optimized queries with `withIndex(...)` for fast access to common fields like driver ID or ride ID.
  - **Asynchronous tasks:** Operations like sending notifications are processed asynchronously to reduce response time for users.

## 3.3 Architectural Patterns

- **Availability:**
  - **Leader-Follower (Replication):** Ensures high availability through replication across geographic zones. Convex's architecture inherently provides replication, fitting this pattern.
- **Scalability:**
  - **Microservices:** Enables horizontal scaling and data sharding by separating features into independent services. Services can be deployed and scaled independently, supporting high scalability.
  - **Event-Driven:** Aligns with our use of Convex async functions, background jobs, and non-blocking operations. Allows components to react to events like `rideRequested` and `rideCompleted` asynchronously.
- **Usability:**

- **MVVM:** Helps maintain a clear separation of concerns in the frontend. Improves usability and supports a responsive real-time UI with clean state management.

- **Security:**
  - **API Gateway Pattern:** Centralizes access control, TLS encryption, RBAC, and routing for backend services. Prevents unauthorized access to ride-matching or location services. Verifies tokens or credentials and enforces **Role-Based Access Control (RBAC)**, while hiding internal services from the public.

- **Performance:**
  - **Microservices:** Isolation of services ensures that each feature runs independently, reducing overhead and avoiding bottlenecks that can impact the entire system.
  - **Event-Driven:**
    * *Asynchronous Processing:* Time-consuming or non-critical tasks are handled in the background, keeping the main user flow fast.
    * *Non-blocking:* The system responds quickly without waiting for every operation to complete, and processes tasks as resources become available.
    * *Decoupling:* Services emit events (e.g., `RideRequested`), and subscribers handle them independently, improving responsiveness and modularity.

# 4 Architectural Design and Pattern

**Overview:** Taxi Tap is structured using a feature-driven and event-driven architecture. The diagram below illustrates this architecture.

| *[Architecture Diagram Placeholder]* |
|---|

**Components:**

- **Expo Frontend:** Mobile-first interface using React Native. The user intefraces are built with MVVM.
- **Convex Backend:** Serverless backend with modular mutations and schema.
- **Convex Database:** Strongly-typed database used by each module.
- **Feature Modules:** Each with its own schema, adapter, hook, and UI screen.

This design provides modularity, scalability, and testability with minimal DevOps complexity.

# 5 Architectural Constraints

- **Client Constraints:** Must remain within the AWS Free Tier; performance must be maintained under low-cost infrastructure.
- **Deployment Constraints:** Fully serverless; no Docker/Kubernetes; must deploy via CI/CD with minimal setup.
- **Security Constraints:** Only verified users may access trip, payment, or GPS functionality.
- **Latency Constraints:** Real-time location updates must occur under **1 second**.
- **Scalability Constraints:** Design must accommodate scaling to 1,000+ users without architectural changes.

# 6 Technology Choices

### Backend Platform

| Option | Pros | Cons |
|---|---|---|
| Convex | Fully serverless, fast dev, native React support | New ecosystem, TypeScript only |
| Firebase | Realtime syncing, easy integration | Poor test tooling, security rule complexity |
| AWS Lambda | Highly scalable, mature | Complex CI/CD, requires DevOps setup |
| **Chosen:** Convex | Perfect fit for modular, testable architecture. Free tier-friendly. | |

### Frontend Platform

| Option | Pros | Cons |
|---|---|---|
| Expo (React Native) | Fast prototyping, hot reload, cross-platform | Slightly heavier bundles |
| Flutter | Beautiful UI, good performance | Slower iteration, Dart-only |
| Native iOS/Android | Highest performance | High dev effort, no code sharing |
| **Chosen:** Expo | Fastest mobile-first path with TypeScript and Convex integration. | |

### Database

| Option | Pros | Cons |
|---|---|---|
| Convex DB | Type-safe, built for Convex, no config | Smaller community |
| Firestore | Realtime, battle-tested | Complex security model |
| Supabase | Postgres-based, open source | Overhead for micro-systems |
| **Chosen:** Convex DB | Natively integrated with our serverless logic. | |

**Payment Processor**

| Option | Pros | Cons |
|---|---|---|
| Yoco | Local SA support, fast onboarding, easy to integrate | Limited advanced payment flows |
| Paystack | Clean APIs, good reliability | Limited card support in SA |
| Stripe | Powerful API, subscriptions | International fees, SA limitations |
| **Chosen:** Yoco | Best fit for local payments in South Africa. Simple, effective, mobile-friendly. | |