Taxi Tap
Architectural Requirements Document

1 3.6 Architectural Requirements Document

1.1 3.6.1 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

This approach accelerates development while ensuring traceability, maintainability, and scalability.

1.2 3.6.2 Architectural Strategies

Chosen Style: *Microkernel Architecture (Plug-in Style)*

Each feature system in Taxi Tap functions as a plug-in module extending a core backend. This microkernel architecture suits our use case because:

- Features can evolve and scale independently
- Systems are decoupled but unified by shared infrastructure
- Testing and deployment can occur at the system level
- Perfect fit for the Convex + Expo stack

1.3 3.6.3 Architectural Quality Requirements

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

1. **Scalability:** System must handle at least **100 concurrent ride requests** with a backend response time of under **100ms**.
2. **Security:** All backend functions must enforce **role-based access control** (driver/passenger/adm
3. **Testability:** Each feature system must achieve **90%+ test coverage** across unit and integration tests.
4. **Availability:** Maintain at least **99.5% uptime** under expected usage, with graceful degradation.
5. **Maintainability:** All features must follow the FDD folder structure and be independently swappable.

1.4 3.6.4 Architectural Design and Pattern

Overview: Taxi Tap is structured using a feature-driven, microkernel-inspired architecture. The diagram below illustrates this architecture.

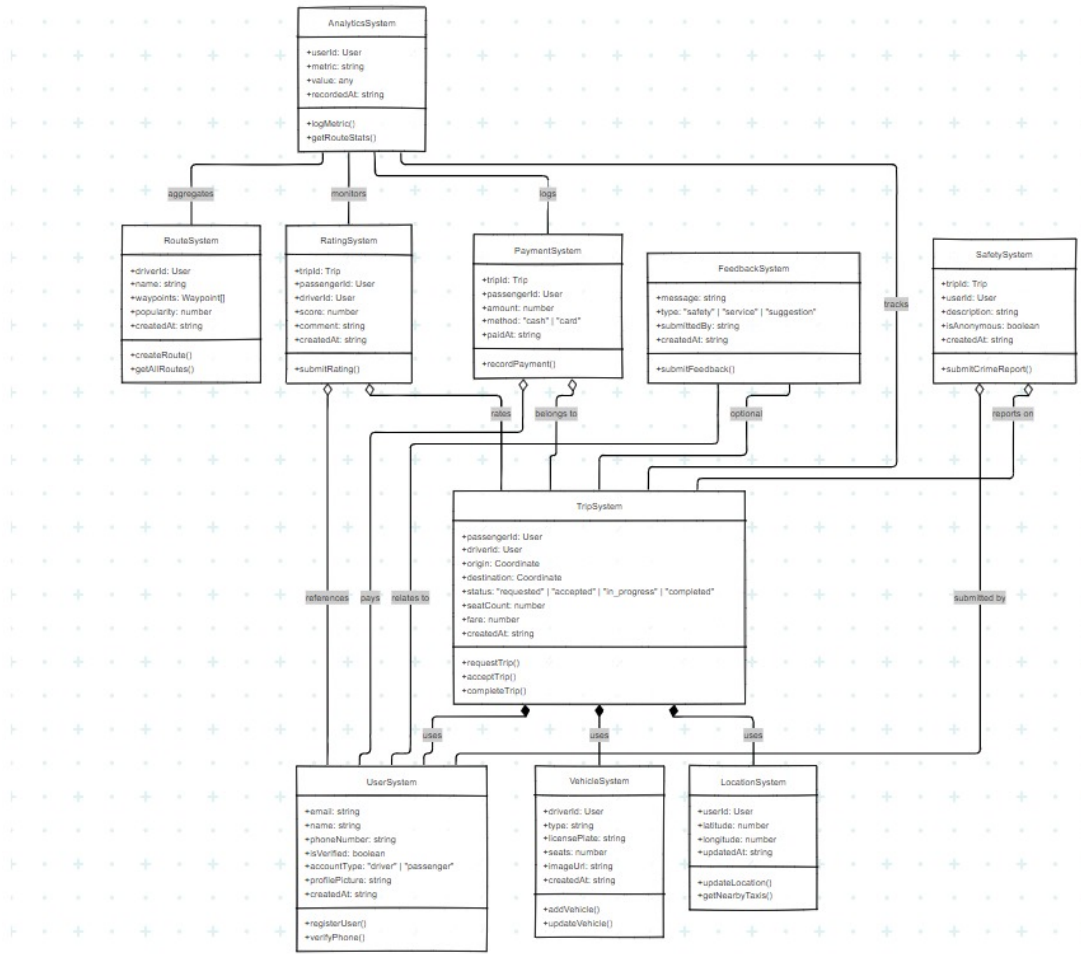

Figure 1: Taxi Tap Feature-Driven Architecture Diagram

Components:

- **Expo Frontend:** Mobile-first interface using React Native
- **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.

1.5 3.6.5 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.

1.6 3.6.6 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.	