# **Okada Transportation Solution**

# **Technical Requirements Document**

*Version 1.0 - March 13, 2025*

## **Table of Contents**

1. [Introduction](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#1-introduction)
2. [System Architecture](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#2-system-architecture)
3. [Database Design](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#3-database-design)
4. [API Specifications](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#4-api-specifications)
5. [Mobile Applications](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#5-mobile-applications)
6. [Web Applications](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#6-web-applications)
7. [Security Requirements](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#7-security-requirements)
8. [Third-Party Integrations](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#8-third-party-integrations)
9. [Offline Functionality](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#9-offline-functionality)
10. [Development Guidelines](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#10-development-guidelines)
11. [Testing Requirements](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#11-testing-requirements)
12. [Deployment Strategy](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#12-deployment-strategy)
13. [Monitoring and Maintenance](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#13-monitoring-and-maintenance)
14. [Appendices](https://claude.ai/chat/1d4a7de0-2ae2-4a36-80f4-1d5950cdf265#14-appendices)

## **1. Introduction**

### **1.1 Purpose**

This Technical Requirements Document (TRD) provides comprehensive specifications for the development of the Okada Transportation Solution. It serves as the primary reference for the technical team to implement all components of the system according to the approved project scope.

### **1.2 Document Conventions**

* **SHALL**: Indicates a mandatory requirement
* **SHOULD**: Indicates a recommendation
* **MAY**: Indicates an optional feature
* **TBD**: To be determined during development

### **1.3 Intended Audience**

* Software Developers
* Quality Assurance Engineers
* DevOps Engineers
* System Architects
* Project Managers
* Technical Stakeholders

### **1.4 Project References**

* Project Scope Document v1.0
* Market Research Report
* Regulatory Compliance Requirements
* User Research Findings

### **1.5 Glossary of Terms**

* **Okada**: Commercial motorcycle taxi
* **Rider**: Motorcycle driver/operator
* **Passenger**: Customer using the motorcycle taxi service
* **FRSC**: Federal Road Safety Commission (Nigeria)
* **DVLA**: Driver and Vehicle Licensing Authority (Ghana)
* **USSD**: Unstructured Supplementary Service Data

## **2. System Architecture**

### **2.1 Architecture Overview**

The Okada Transportation Solution follows a hybrid architecture combining microservices for critical components with a monolithic core for shared functionality, optimized for performance in low-connectivity environments.

#### **2.1.1 High-Level Architecture Diagram**

┌────────────────────────────┐ ┌───────────────────────────┐

│ Client Applications │ │ Third-Party Services │

│ │ │ │

│ ┌─────────┐ ┌─────────┐ │ │ ┌─────────┐ ┌─────────┐ │

│ │ Rider │ │Passenger│ │ │ │ Payment │ │ Mapping │ │

│ │ App │ │ App │ │←────→│ │ Gateway │ │ Services│ │

│ └─────────┘ └─────────┘ │ │ └─────────┘ └─────────┘ │

│ │ │ │

│ ┌─────────────────────┐ │ │ ┌─────────┐ ┌─────────┐ │

│ │ Web Dashboards │ │ │ │ SMS │ │Emergency│ │

│ └─────────────────────┘ │ │ │ Gateway │ │Services │ │

└────────────────────────────┘ │ └─────────┘ └─────────┘ │

↑ └───────────────────────────┘

│ ↑

↓ │

┌─────────────────────────────────────────────────┴───────────────┐

│ API Gateway Layer │

└─────────────────────────────────────────────────────────────────┘

↑

│

↓

┌────────────────────────────────────────────────────────────────┐

│ Microservices Layer │

│ │

│ ┌─────────┐ ┌─────────┐ ┌─────────┐ ┌─────────┐ ┌────────┐ │

│ │ User │ │ Ride │ │ Payment │ │ Safety │ │Analytics│ │

│ │ Service │ │ Service │ │ Service │ │ Service │ │ Service │ │

│ └─────────┘ └─────────┘ └─────────┘ └─────────┘ └────────┘ │

│ │

└────────────────────────────────────────────────────────────────┘

↑

│

↓

┌────────────────────────────────────────────────────────────────┐

│ Data Layer │

│ │

│ ┌─────────────┐ ┌─────────────┐ ┌─────────────┐ │

│ │ PostgreSQL │ │ MongoDB │ │ Redis │ │

│ │ (Relational)│ │ (NoSQL) │ │ (Cache) │ │

│ └─────────────┘ └─────────────┘ └─────────────┘ │

│ │

└────────────────────────────────────────────────────────────────┘

### **2.2 Component Specifications**

#### **2.2.1 Core Services**

| **Service** | **Primary Responsibility** | **Technology Stack** | **Scaling Strategy** | **Fault Tolerance** |
| --- | --- | --- | --- | --- |
| User Management | Authentication, profile management, authorization | Node.js, Express, PostgreSQL | Horizontal | Multiple instances, database replication |
| Ride Matching | GPS tracking, rider-passenger matching, route optimization | Python, FastAPI, Redis | Horizontal | Multiple instances, data partitioning |
| Payment Gateway | Transaction processing, wallet management | Java, Spring Boot | Vertical | Active-passive failover |
| Safety & SOS | Emergency alerts, crash detection | Golang, WebSockets | Horizontal | Multiple instances across regions |
| Analytics | Reporting, business intelligence | Python, Elasticsearch | Vertical | Data redundancy |

#### **2.2.2 Infrastructure Components**

| **Component** | **Technology** | **Purpose** | **Configuration** |
| --- | --- | --- | --- |
| API Gateway | Kong Gateway | Traffic routing, rate limiting | Auto-scaling, multi-region |
| Message Queue | RabbitMQ | Asynchronous communication | Clustered, persistent storage |
| Cache | Redis | Session management, real-time data | Multi-node cluster |
| CDN | CloudFront/Cloudflare | Static asset delivery | Edge caching enabled |
| Load Balancer | AWS ELB/GCP LB | Traffic distribution | Health checks, SSL termination |

### **2.3 Technology Stack Summary**

#### **2.3.1 Backend Technologies**

* **Primary Languages**: JavaScript/TypeScript (Node.js), Python, Golang, Java
* **Frameworks**: Express.js, FastAPI, Spring Boot, Gin
* **Databases**: PostgreSQL, MongoDB, Redis
* **Cloud Providers**: AWS (primary), GCP (secondary)
* **Container Orchestration**: Kubernetes

#### **2.3.2 Frontend Technologies**

* **Mobile**: React Native (primary), Flutter (optional)
* **Web**: React.js, Material UI, Tailwind CSS
* **State Management**: Redux, Context API
* **Mapping**: MapboxGL, Google Maps API, OpenStreetMap

#### **2.3.3 DevOps Technologies**

* **CI/CD**: GitHub Actions, Jenkins
* **Infrastructure as Code**: Terraform
* **Monitoring**: Prometheus, Grafana, ELK Stack
* **Logging**: Fluentd, Elasticsearch, Kibana

## **3. Database Design**

### **3.1 Database Strategy**

The solution will implement a polyglot persistence strategy with:

* **PostgreSQL**: For structured, relational data requiring ACID compliance
* **MongoDB**: For unstructured or semi-structured data with flexible schema
* **Redis**: For caching, session management, and real-time features

### **3.2 Relational Database Schema (PostgreSQL)**

#### **3.2.1 Core Tables**

| **Table Name** | **Primary Purpose** | **Key Fields** | **Relationships** |
| --- | --- | --- | --- |
| users | Store user accounts | id, email, phone, password\_hash, user\_type, status | One-to-many with riders, passengers |
| riders | Store rider-specific information | id, user\_id, license\_number, vehicle\_registration, safety\_rating | Many-to-one with users |
| passengers | Store passenger-specific information | id, user\_id, home\_locations, work\_locations, preferred\_payment | Many-to-one with users |
| rides | Store ride information | id, rider\_id, passenger\_id, start\_location, end\_location, status, fare | Many-to-one with riders, passengers |
| payments | Store payment transactions | id, ride\_id, amount, payment\_method, status, timestamp | Many-to-one with rides |
| vehicles | Store vehicle information | id, rider\_id, make, model, year, color, plate\_number | Many-to-one with riders |
| licenses | Store license information | id, rider\_id, license\_type, issued\_date, expiry\_date, authority | Many-to-one with riders |
| emergency\_contacts | Store emergency contact information | id, user\_id, name, relationship, phone\_number | Many-to-one with users |
| ride\_reviews | Store ride ratings/reviews | id, ride\_id, rating, review\_text, timestamp | Many-to-one with rides |

#### **3.2.2 ER Diagram**

users 1───┐

│

├──\* riders ─1───┐

│ │

│ ├──\* vehicles

│ │

├──\* passengers │

│ ├──\* licenses

│ │

└──\* emergency\_contacts

│

riders 1────┐ │

│ │

└──\* rides ───┘

│

│

passengers 1─────┘

│

│

├──\* payments

│

└──\* ride\_reviews

### **3.3 NoSQL Database Schema (MongoDB)**

#### **3.3.1 Collections**

| **Collection** | **Purpose** | **Sample Document Structure** |
| --- | --- | --- |
| locationLogs | Track real-time GPS positions | { rider\_id, timestamp, coordinates: {lat, lng}, speed, heading, battery\_level, accuracy } |
| rideTelemetry | Store detailed ride data | { ride\_id, waypoints: [{timestamp, coordinates, speed}], events: [{type, timestamp, details}] } |
| safetyIncidents | Record accidents, near-misses | { ride\_id, timestamp, incident\_type, severity, coordinates, description, media\_urls } |
| appAnalytics | Store app usage statistics | { user\_id, session\_id, app\_version, device\_info, events: [{type, timestamp, details}] } |

### **3.4 Cache Strategy (Redis)**

| **Key Pattern** | **Purpose** | **TTL** | **Update Strategy** |
| --- | --- | --- | --- |
| user:{id}:session | User session data | 24 hours | Refresh on activity |
| rider:{id}:location | Current rider location | 5 minutes | Update on location change |
| area:{geohash}:available\_riders | Available riders in area | 30 seconds | Update on status change |
| ride:{id}:status | Current ride status | Duration of ride | Update on status change |
| user:{id}:auth\_attempts | Failed login attempts | 1 hour | Reset on successful login |

### **3.5 Data Migration Strategy**

* **Initial Data Loading**: Scripts for importing existing rider associations data
* **Schema Versioning**: Database migrations managed through Flyway (PostgreSQL)
* **Data Archiving**: Automated archiving of ride data older than 6 months to cold storage

## **4. API Specifications**

### **4.1 API Design Principles**

* RESTful design for standard operations
* WebSockets for real-time features
* JWT-based authentication
* Rate limiting to prevent abuse
* Comprehensive error handling with appropriate HTTP status codes
* Versioned endpoints (e.g., /api/v1/resource)
* Consistent response format

### **4.2 Authentication & Authorization**

#### **4.2.1 Authentication Endpoints**

| **Endpoint** | **Method** | **Purpose** | **Request Parameters** | **Response** |
| --- | --- | --- | --- | --- |
| /api/v1/auth/register | POST | User registration | {phone, email, password, user\_type} | {user\_id, token} |
| /api/v1/auth/login | POST | User login | {phone, password} | {user\_id, token, refresh\_token} |
| /api/v1/auth/refresh | POST | Refresh access token | {refresh\_token} | {token, refresh\_token} |
| /api/v1/auth/verify-phone | POST | Verify phone via OTP | {phone, otp\_code} | {verification\_status} |

#### **4.2.2 Authorization Framework**

* **Role-Based Access Control (RBAC)** with following roles:  
  + passenger: Basic user capabilities
  + rider: Rider-specific capabilities
  + admin: System administration
  + regulator: Government/regulatory access

**JWT Claims Structure**:  
  
 {

"sub": "[user\_id]",

"roles": ["passenger"],

"iat": 1615910970,

"exp": 1615997370

}

### **4.3 Core API Endpoints**

#### **4.3.1 User Management API**

| **Endpoint** | **Method** | **Purpose** | **Authentication** | **Request** | **Response** |
| --- | --- | --- | --- | --- | --- |
| /api/v1/users/profile | GET | Get user profile | Required | - | {user} |
| /api/v1/users/profile | PUT | Update user profile | Required | {profile\_data} | {updated\_user} |
| /api/v1/users/documents | POST | Upload user documents | Required | {document\_type, file} | {document\_id, status} |
| /api/v1/users/{id}/verification | GET | Get verification status | Admin/Regulator | - | {verification\_status} |

#### **4.3.2 Ride Management API**

| **Endpoint** | **Method** | **Purpose** | **Authentication** | **Request** | **Response** |
| --- | --- | --- | --- | --- | --- |
| /api/v1/rides/request | POST | Request a ride | Passenger | {pickup, destination, payment\_method} | {ride\_id, status, fare\_estimate} |
| /api/v1/rides/nearby | GET | Get nearby available riders | Passenger | {latitude, longitude, radius} | {riders: [{id, distance, eta}]} |
| /api/v1/rides/{id}/accept | POST | Accept a ride request | Rider | - | {ride\_details} |
| /api/v1/rides/{id}/start | POST | Start ride | Rider | - | {ride\_status} |
| /api/v1/rides/{id}/complete | POST | Complete ride | Rider | {end\_location} | {ride\_summary} |
| /api/v1/rides/{id}/cancel | POST | Cancel ride | Either | {reason} | {cancellation\_details} |
| /api/v1/rides/history | GET | Get ride history | Either | {page, limit, filters} | {rides: [], pagination} |

#### **4.3.3 Payment API**

| **Endpoint** | **Method** | **Purpose** | **Authentication** | **Request** | **Response** |
| --- | --- | --- | --- | --- | --- |
| /api/v1/payments/methods | GET | List payment methods | Required | - | {payment\_methods: []} |
| /api/v1/payments/methods | POST | Add payment method | Required | {provider, details} | {method\_id, status} |
| /api/v1/payments/wallet/balance | GET | Get wallet balance | Required | - | {balance, currency} |
| /api/v1/payments/wallet/topup | POST | Add funds to wallet | Required | {amount, payment\_method} | {transaction\_id, status} |
| /api/v1/payments/{ride\_id}/process | POST | Process ride payment | Required | {payment\_method} | {payment\_id, status} |

#### **4.3.4 Safety & SOS API**

| **Endpoint** | **Method** | **Purpose** | **Authentication** | **Request** | **Response** |
| --- | --- | --- | --- | --- | --- |
| /api/v1/safety/sos | POST | Trigger SOS alert | Required | {location, situation} | {sos\_id, status} |
| /api/v1/safety/incidents | POST | Report safety incident | Required | {type, description, location} | {incident\_id, status} |
| /api/v1/safety/zones | GET | Get safety zone information | Required | {latitude, longitude} | {zone\_data, restrictions} |

### **4.4 WebSocket Endpoints**

| **Endpoint** | **Purpose** | **Authentication** | **Events** |
| --- | --- | --- | --- |
| /ws/rider-location | Real-time rider location updates | Required | location\_update, connection\_status |
| /ws/ride-status | Real-time ride status updates | Required | status\_change, eta\_update, route\_change |
| /ws/emergency | Emergency alerts and responses | Required | sos\_triggered, emergency\_response |

### **4.5 API Documentation**

* **Format**: OpenAPI (Swagger) 3.0
* **Documentation URL**: /api/docs
* **Automatic Generation**: API documentation generated from code annotations
* **Authentication for Docs**: Required in production environment

## **5. Mobile Applications**

### **5.1 Common Requirements (Both Apps)**

#### **5.1.1 Platform Support**

* **Android**: API Level 19 (KitKat) and above, targeting API 33+
* **iOS**: iOS 12 and above (considered Phase 2)

#### **5.1.2 Framework & Languages**

* **Primary Framework**: React Native
* **Languages**: JavaScript/TypeScript
* **Navigation**: React Navigation
* **State Management**: Redux + Redux Toolkit

#### **5.1.3 Performance Requirements**

* **Cold Start Time**: < 3 seconds on mid-range devices
* **Response Time**: < 1 second for common operations
* **Memory Usage**: < 150MB in normal operation
* **Battery Impact**: < 5% per hour of active use
* **Storage Size**: < 50MB for initial installation

#### **5.1.4 Offline Capabilities**

* **Data Persistence**: Async Storage / SQLite
* **Sync Strategy**: Queue-based operations when reconnected
* **Offline Maps**: Downloadable map regions (10km radius)

#### **5.1.5 Shared UI Components Library**

| **Component** | **Purpose** | **Variants** |
| --- | --- | --- |
| AppButton | Standard buttons | Primary, Secondary, Danger |
| AppInput | Input fields | Text, Number, Phone, Password |
| LocationSelector | Location selection | Pickup, Destination |
| RideCard | Ride information display | Active, History, Request |
| StatusBadge | Status indicators | Success, Warning, Error, Info |
| MapView | Display interactive map | Full, Mini, Route |
| LoadingIndicator | Loading states | Spinner, Skeleton |
| ErrorDisplay | Error messages | Toast, Modal, Inline |
| RatingStars | Rating input/display | Input, ReadOnly |

### **5.2 Rider App Specifications**

#### **5.2.1 Key Features**

* Ride request management
* Navigation and routing
* Earnings tracking and history
* Documentation and compliance management
* Safety features and SOS

#### **5.2.2 Screen Specifications**

| **Screen** | **Primary Purpose** | **Key Components** | **Navigation Paths** |
| --- | --- | --- | --- |
| Login/Registration | Account access | Phone verification, Identity verification | → Dashboard |
| Dashboard | Overview of activity | Earnings summary, Available rides, Online toggle | → Ride Details, → Profile, → Earnings |
| Ride Request | View incoming requests | Request details, Accept/Reject actions, Map preview | → Active Ride, → Dashboard |
| Active Ride | Manage ongoing ride | Navigation, Passenger details, Ride actions | → Ride Completion, → SOS |
| Navigation | Guide to destination | Map, Turn-by-turn directions, ETA | → Active Ride |
| Earnings | Financial management | Daily/weekly/monthly summaries, Transaction history | → Cashout, → Dashboard |
| Compliance Hub | Document management | License status, Vehicle registration, Safety certifications | → Document Upload |
| Profile | Account management | Personal details, Vehicle details, Rating | → Settings, → Dashboard |
| SOS | Emergency assistance | One-touch emergency alert, Location sharing | → Emergency Contacts |

#### **5.2.3 Offline Mode Functionality**

* Cache of active ride information
* Offline navigation using downloaded map data
* Queued ride completion for processing when online
* Locally stored earnings records synced when connection restored

#### **5.2.4 Performance Optimizations**

* Background location updates optimized for battery (variable frequency)
* Map pre-loading for common areas
* Lightweight ride acceptance flow for quick response in low-connectivity
* Progressive image loading for documents

### **5.3 Passenger App Specifications**

#### **5.3.1 Key Features**

* Ride booking
* Rider selection
* Ride tracking
* Payment processing
* Safety features and SOS

#### **5.3.2 Screen Specifications**

| **Screen** | **Primary Purpose** | **Key Components** | **Navigation Paths** |
| --- | --- | --- | --- |
| Login/Registration | Account access | Phone verification, Account creation | → Home |
| Home | Main interaction point | Destination input, Saved locations, Recent rides | → Ride Booking |
| Ride Booking | Configure ride details | Map, Location refinement, Payment selection | → Rider Selection |
| Rider Selection | Choose available rider | Rider list, Ratings, ETA, Fare estimate | → Ride Tracking |
| Ride Tracking | Monitor active ride | Real-time map, Rider details, Share ride | → Ride Completion, → SOS |
| Ride Completion | Finalize ride | Rating, Payment completion, Receipt | → Home |
| Payment | Manage payment | Payment methods, Wallet, Transaction history | → Add Payment Method |
| Profile | Account management | Personal details, Saved locations, Preferences | → Settings, → Home |
| Safety Center | Safety features | Emergency contacts, Ride sharing, SOS button | → SOS |
| Ride History | View past rides | Ride list, Details, Receipts | → Ride Detail, → Home |

#### **5.3.3 Offline Mode Functionality**

* Cached home locations and frequent destinations
* Basic ride request queuing for low-connectivity areas
* Offline access to ride receipts and history
* Local storage of safety contacts

#### **5.3.4 Performance Optimizations**

* On-demand location tracking (only during active rides)
* Deferred loading of ride history
* Compressed map assets for faster loading
* Lazy loading of rider profiles

### **5.4 SMS/USSD Fallback System**

#### **5.4.1 SMS Commands**

| **Command** | **Format** | **Purpose** | **Response** |
| --- | --- | --- | --- |
| RIDE | RIDE [pickup location] TO [destination] | Request a ride | Confirmation with rider details |
| STATUS | STATUS [ride\_id] | Check ride status | Current status and ETA |
| CANCEL | CANCEL [ride\_id] | Cancel a ride | Cancellation confirmation |
| SOS | SOS [ride\_id] | Emergency alert | Confirmation and support contact |
| RATE | RATE [ride\_id] [1-5] | Rate completed ride | Rating confirmation |

#### **5.4.2 USSD Menu Structure**

Main Menu (\*123#)

├── 1. Request Ride

│ ├── Enter pickup location

│ ├── Enter destination

│ └── Confirm request

├── 2. Active Ride

│ ├── 1. Get status

│ ├── 2. Contact rider

│ ├── 3. Share ride details

│ └── 4. Cancel ride

├── 3. Payment

│ ├── 1. Check wallet balance

│ ├── 2. Top up wallet

│ └── 3. Payment history

└── 4. Account

├── 1. View profile

├── 2. Ride history

└── 3. Help

## **6. Web Applications**

### **6.1 Admin Dashboard**

#### **6.1.1 Core Features**

* User management and verification
* Ride monitoring and intervention
* Payment oversight and reconciliation
* Safety incident management
* Analytics and reporting
* System configuration

#### **6.1.2 Technology Stack**

* **Framework**: React.js
* **UI Library**: Material UI
* **State Management**: Redux
* **Data Visualization**: Chart.js, D3.js
* **Maps**: MapboxGL

#### **6.1.3 Key Screens**

| **Screen** | **Primary Purpose** | **Key Components** | **User Roles** |
| --- | --- | --- | --- |
| Dashboard | System overview | KPI summaries, Alert feeds, Activity maps | All Admin |
| User Management | Manage user accounts | User search, Verification workflow, Document review | User Admin |
| Ride Monitoring | Track ongoing rides | Real-time map, Ride list, Intervention tools | Operations Admin |
| Payment Management | Oversee finances | Transaction list, Settlement status, Dispute resolution | Finance Admin |
| Safety Center | Handle safety issues | Incident reports, SOS alerts, Safety statistics | Safety Admin |
| Reporting | Generate insights | Report builder, Scheduled reports, Export tools | Analytics Admin |
| System Settings | Configure platform | Feature toggles, Pricing rules, Geofence management | System Admin |

### **6.2 Regulatory Dashboard**

#### **6.2.1 Core Features**

* Compliance monitoring
* Safety statistics
* Rider verification
* Incident reporting
* Geographic restrictions management
* Export capabilities for official records

#### **6.2.2 Technology Stack**

* Same as Admin Dashboard with additional security features

#### **6.2.3 Key Screens**

| **Screen** | **Primary Purpose** | **Key Components** | **User Roles** |
| --- | --- | --- | --- |
| Compliance Overview | Regulatory status | Compliance rates, Document validity, Non-compliance alerts | All Regulator |
| Safety Statistics | Monitor safety metrics | Accident rates, Speed violations, Safety comparison tools | Safety Regulator |
| Rider Registry | Verify rider information | License validation, Vehicle documentation, Background verification | Verification Regulator |
| Incident Management | Track reported issues | Incident map, Severity classification, Resolution tracking | Operations Regulator |
| Geofence Management | Control restricted areas | Map editor, Restriction scheduler, Notification system | Policy Regulator |
| Reporting | Generate official reports | Standardized reports, Data export, Audit logs | All Regulator |

### **6.3 Progressive Web App (PWA) Features**

* **Offline Support**: Service workers for basic functionality without connection
* **Installation**: Installable on desktop and mobile devices
* **Push Notifications**: For administrators and regulators
* **Responsive Design**: Optimized for desktop, tablet, and mobile views

## **7. Security Requirements**

### **7.1 Data Protection**

#### **7.1.1 Data Classification**

| **Data Category** | **Sensitivity** | **Storage Requirements** | **Encryption** | **Retention** |
| --- | --- | --- | --- | --- |
| User authentication | High | Encrypted database | Hash + Salt (Argon2) | Until account deletion |
| Personal identifiable information | High | Encrypted database | AES-256 at rest | Until account deletion + 30 days |
| Payment information | Critical | Tokenized storage | PCI DSS compliant | Token only, no raw data |
| Location data | Medium | Encrypted database | AES-256 at rest | 90 days for detailed, aggregated thereafter |
| Ride history | Medium | Encrypted database | AES-256 at rest | 3 years |
| System logs | Low | Secure storage | Optional | 1 year |

#### **7.1.2 Encryption Standards**

* **Data in Transit**: TLS 1.3
* **Data at Rest**: AES-256
* **Key Management**: AWS KMS or equivalent
* **Password Storage**: Argon2id with salt and pepper

### **7.2 Authentication Security**

#### **7.2.1 Password Policy**

* Minimum 8 characters
* Require combination of letters, numbers, and special characters
* Password history (prevent reuse of last 5 passwords)
* Maximum age of 180 days
* Account lockout after 5 failed attempts (temporary, escalating duration)

#### **7.2.2 Multi-Factor Authentication**

* Required for administrative and regulatory users
* Optional for riders and passengers
* Methods: SMS OTP, TOTP (Google Authenticator), email verification

#### **7.2.3 Session Management**

* JWT expiration: 1 hour
* Refresh token expiration: 7 days
* Invalidation on password change or suspicious activity
* Concurrent session limitations for admin accounts

### **7.3 API Security**

* Rate limiting: Tiered approach based on endpoint sensitivity
* Input validation: Strict validation against schemas for all endpoints
* Output encoding: Prevention of XSS and injection attacks
* API versioning: Clear deprecation policies
* CORS: Strict origin controls

### **7.4 Mobile Security**

* Certificate pinning for API communication
* App transport security enforcement
* Secure local storage with encryption
* Jailbreak/root detection
* Runtime application self-protection (RASP)
* Obfuscation of sensitive code sections

### **7.5 Security Monitoring**

* Real-time alerts for suspicious activities
* Automated vulnerability scanning
* Regular penetration testing (quarterly)
* Audit logging for sensitive operations
* Fraud detection system for payment and ride patterns

### **7.6 Compliance Requirements**

* **GDPR**: For future expansion to European markets
* **NDPR**: Nigerian Data Protection Regulation
* **DPA**: Ghana Data Protection Act
* **PCI DSS**: For payment handling
* **ISO 27001**: Information security management alignment

## **8. Third-Party Integrations**

### **8.1 Mapping & Navigation Services**

#### **8.1.1 Primary Mapping Providers**

| **Provider** | **Purpose** | **API Features Used** | **Implementation Notes** |
| --- | --- | --- | --- |
| Google Maps | Primary mapping on client | Maps SDK, Directions API, Places API, Distance Matrix API | Implement custom caching for offline use |
| OpenStreetMap | Alternative/fallback mapping | Tile API, Nominatim API | Used in areas with better OSM coverage |
| Mapbox | Custom styling and offline maps | Navigation SDK, Offline Pack API | Preferred for offline navigation |

#### **8.1.2 Geocoding Strategy**

* Forward geocoding: Convert address to coordinates
* Reverse geocoding: Convert coordinates to readable address
* Local caching of frequent locations
* Fallback between providers based on accuracy

### **8.2 Payment Gateways**

#### **8.2.1 Mobile Money Integration**

| **Provider** | **Markets** | **Features** | **Implementation Notes** |
| --- | --- | --- | --- |
| MTN Mobile Money | Nigeria, Ghana | Direct deposits, withdrawals, merchant payments | Direct API integration using OAuth 2.0 |
| Airtel Money | Nigeria | Direct deposits, withdrawals, merchant payments | Direct API integration using API keys |
| Vodafone Cash | Ghana | Direct deposits, merchant payments | Integration via aggregator |

#### **8.2.2 Payment Aggregators**

| **Provider** | **Purpose** | **Features** | **Implementation Notes** |
| --- | --- | --- | --- |
| Flutterwave | Multi-provider integration | Single API for multiple payment methods | Primary integration for payment processing |
| Paystack | Card processing, bank transfers | Recurring payments, payment pages | Secondary provider for certain payment types |

#### **8.2.3 Reconciliation Process**

* Daily settlement file processing
* Automated matching of transactions
* Dispute management workflow
* Financial reporting integration

### **8.3 SMS & Communication Services**

| **Provider** | **Purpose** | **Features** | **Implementation Notes** |
| --- | --- | --- | --- |
| Twilio | Primary SMS delivery | Programmable SMS, Verify API | Used for verification codes and notifications |
| Africa's Talking | Secondary SMS provider | Bulk SMS, USSD API | Used for USSD implementation and fallback SMS |
| Firebase Cloud Messaging | Push notifications | Topic messaging, token-based delivery | Used for app notifications |

### **8.4 Identity Verification**

| **Provider** | **Purpose** | **Features** | **Implementation Notes** |
| --- | --- | --- | --- |
| Smile Identity | KYC verification | Document verification, facial matching | Primary verification for Nigeria |
| Veriff | KYC verification | Document verification, liveness detection | Secondary verification option |
| Local API integrations | License verification | FRSC (Nigeria) and DVLA (Ghana) integrations | Direct verification with authorities where available |

### **8.5 Analytics & Monitoring**

| **Provider** | **Purpose** | **Features** | **Implementation Notes** |
| --- | --- | --- | --- |
| Firebase Analytics | User behavior tracking | Event tracking, conversion funnels | Implemented on all mobile apps |
| Mixpanel | Advanced user analytics | User journeys, retention analysis | Secondary analytics for deeper insights |
| Datadog | System monitoring | APM, infrastructure monitoring, logs | Primary monitoring solution |
| Sentry | Error tracking | Real-time error reporting, stack traces | Implemented across all applications |

## **9. Offline Functionality**

### **9.1 Offline-First Design Principles**

* Default to local operations with background synchronization
* Graceful degradation of features in offline mode
* Clear user feedback about connectivity status
* Prioritize critical functions for offline capabilities

### **9.2 Offline Data Management**

#### **9.2.1 Data Synchronization Strategy**

| **Data Type** | **Storage Mechanism** | **Sync Priority** | **Conflict Resolution** |
| --- | --- | --- | --- |
| User profile | SQLite/Realm | Medium | Server wins with notification |
| Active ride details | SQLite/Realm | High | Last write wins with merge |
| Location history | SQLite | Low | Append only, timestamp-based |
| Cached map data | File system | Medium | Server wins, version-based |
| Pending transactions | SQLite + Queue | Critical | Client intent preserved, state reconciliation |

#### **9.2.2 Sync Algorithms**

* Exponential backoff for retry attempts
* Delta synchronization where possible
* Background sync with batch processing
* Priority-based queue processing

### **9.3 Offline Maps Implementation**

* **Map Tile Caching**: Automatic caching of viewed areas
* **Offline Region Management**: User-controlled map downloads
* **Vector Maps**: Efficient storage of map data
* **POI Caching**: Local storage of frequently used locations

### **9.4 Offline Ride Flow**

#### **9.4.1 Passenger App Offline Capabilities**

1. View previously cached rider locations (marked as potentially outdated)
2. Request ride (queued for when connection is restored)
3. View active ride details from last known state
4. Emergency features with SMS fallback

#### **9.4.2 Rider App Offline Capabilities**

1. Complete ongoing ride (stored locally, synced later)
2. Navigate using offline maps
3. Record fare information offline
4. Emergency features with SMS fallback

### **9.5 SMS/USSD Integration**

* Fallback to SMS for critical operations
* USSD menus as alternative interface
* SMS notifications for important status changes
* Bridge between SMS commands and app functionality

## **10. Development Guidelines**

### **10.1 Code Organization**

#### **10.1.1 Backend Structure**

/src

/api

/v1

/controllers

/middlewares

/routes

/validators

/config

/models

/services

/utils

/scripts

/tests

/unit

/integration

/performance

/docs

#### **10.1.2 Mobile App Structure**

/src

/api

/assets

/components

/common

/screens

/hooks

/navigation

/redux

/slices

/thunks

/services

/styles

/utils

/tests

### **10.2 Coding Standards**

#### **10.2.1 JavaScript/TypeScript Standards**

* Airbnb JavaScript Style Guide
* TypeScript strict mode enabled
* ESLint configuration with automatic formatting
* Function-first approach with minimal class usage
* Async/await preferred over Promises

#### **10.2.2 Python Standards**

* PEP 8 compliance
* Type hints required
* Black for code formatting
* Docstrings required for all public methods
* Pytest for testing

#### **10.2.3 Java Standards**

* Google Java Style Guide
* Strict nullability annotations
* JUnit 5 for testing
* JavaDoc required for all public methods

### **10.3 Testing Requirements**

#### **10.3.1 Coverage Requirements**

| **Component** | **Minimum Coverage** |
| --- | --- |
| Core services | 80% |
| API controllers | 90% |
| Data models | 75% |
| Utility functions | 85% |
| UI components | 60% |

#### **10.3.2 Test Types**

* **Unit Tests**: For individual functions and components
* **Integration Tests**: For API endpoints and service interactions
* **End-to-End Tests**: For critical user flows
* **Performance Tests**: For high-traffic endpoints
* **Security Tests**: For vulnerability assessment

### **10.4 Documentation Requirements**

* OpenAPI/Swagger documentation for all APIs
* README.md for each service/component
* Architectural Decision Records (ADRs) for significant decisions
* JSDoc/PyDoc/JavaDoc for public methods
* User documentation for admin interfaces

### **10.5 CI/CD Pipeline**

* Pull request workflow with automated code review
* Build and test automation
* Staging deployment for QA
* Automated QA for critical flows
* Canary deployment strategy for production
* Automated rollback capability

## **11. Testing Requirements**

### **11.1 Testing Strategy**

* **Shift-Left Testing**: Testing early in development cycle
* **Risk-Based Testing**: Focus on high-impact areas
* **Automated Testing**: Maximum automation of repetitive tests
* **Exploratory Testing**: For edge cases and user experience
* **Accessibility Testing**: For inclusive design

### **11.2 Test Environments**

| **Environment** | **Purpose** | **Data** | **Deployment Frequency** | **Access Control** |
| --- | --- | --- | --- | --- |
| Development | Developer testing | Anonymized subset | Continuous | Development team |
| Integration | Service integration testing | Synthetic | Daily | Development + QA |
| Staging | Pre-production verification | Production-like | After successful integration | QA + Stakeholders |
| Production | Live system | Real | After staging approval | Restricted by role |

### **11.3 Test Case Examples**

#### **11.3.1 Rider Onboarding Flow**

1. User registration with valid phone number
2. Phone verification via OTP
3. Profile completion with valid information
4. Document upload and verification
5. Vehicle information registration
6. License verification
7. Safety training module completion
8. Final approval and activation

#### **11.3.2 Ride Booking Flow**

1. Passenger enters valid pickup and destination
2. System calculates accurate fare estimate
3. Available riders are displayed correctly
4. Passenger selects rider and confirms booking
5. Rider receives and accepts request
6. Rider navigates to pickup location
7. Ride starts when rider arrives
8. Navigation functions correctly
9. Ride completes at destination
10. Payment processes successfully
11. Rating system functions properly

### **11.4 Performance Testing**

#### **11.4.1 Load Test Scenarios**

| **Scenario** | **Target** | **Metrics Measured** |
| --- | --- | --- |
| Peak hour ride requests | 1000 concurrent requests | Response time, error rate, CPU usage |
| Rider location updates | 5000 concurrent riders | Message processing time, data transfer rate |
| Payment processing | 100 transactions per second | Processing time, success rate |
| Map rendering | 1000 concurrent users | Rendering time, memory usage |

#### **11.4.2 Stress Testing**

* Gradual increase in load until system failure
* Recovery testing after failure
* Component isolation testing
* Database connection pool exhaustion testing
* Network degradation simulation

### **11.5 Security Testing**

* OWASP Top 10 vulnerability assessment
* Penetration testing (quarterly)
* Secure code review
* API fuzzing
* Authentication bypass attempts
* Session management testing
* Data leakage testing

### **11.6 Usability Testing**

* Field testing with actual riders and passengers
* Low-literacy user testing
* Testing in poor connectivity areas
* Accessibility evaluation
* A/B testing of critical flows
* Feedback collection mechanism

## **12. Deployment Strategy**

### **12.1 Environments**

| **Environment** | **Purpose** | **Infrastructure** | **Data Policy** |
| --- | --- | --- | --- |
| Development | Developer testing | Lightweight containers | Synthetic data only |
| Testing | Automated tests | CI/CD pipeline | Anonymized subset |
| Staging | Pre-production | Mirror of production | Anonymized production |
| Production | Live system | Fully redundant | Real data, strict access |

### **12.2 Deployment Pipeline**

1. **Code Commit**: Developer pushes to feature branch
2. **Automated Tests**: Unit and integration tests run
3. **Code Review**: Required approvals from peers
4. **Build**: Creation of deployment artifacts
5. **Deployment to Testing**: Automated deployment for QA
6. **QA Verification**: Manual testing of key flows
7. **Deployment to Staging**: After QA approval
8. **Stakeholder Approval**: Final sign-off
9. **Canary Deployment**: 5% of production traffic
10. **Full Deployment**: Gradual rollout to all users

### **12.3 Infrastructure as Code**

* Terraform for cloud resource provisioning
* Kubernetes manifests for container orchestration
* Helm charts for application deployment
* Ansible for configuration management
* All configurations in version control

### **12.4 Deployment Architecture**

#### **12.4.1 Cloud Regions**

| **Region** | **Purpose** | **Services** | **Failover Strategy** |
| --- | --- | --- | --- |
| AWS Africa (Cape Town) | Primary - Nigeria | All services | Auto-failover to secondary |
| GCP (Ghana) | Primary - Ghana | All services | Auto-failover to primary |

#### **12.4.2 Kubernetes Architecture**

* **Node Pools**:
  + General purpose (2-8 nodes)
  + Memory optimized (2-4 nodes)
  + Compute optimized (2-4 nodes)
* **Namespaces**:
  + core-services
  + api-gateway
  + monitoring
  + database
  + messaging

### **12.5 Database Deployment**

* Primary-replica configuration for PostgreSQL
* Multi-node clusters for MongoDB
* Redis cluster with sentinel
* Daily automated backups
* Point-in-time recovery capability

### **12.6 Release Management**

* Semantic versioning (MAJOR.MINOR.PATCH)
* Release notes for each version
* Feature flags for gradual rollout
* Automated rollback procedures
* Change advisory board for major releases

## **13. Monitoring and Maintenance**

### **13.1 Monitoring Strategy**

#### **13.1.1 Key Metrics**

| **Category** | **Metrics** | **Thresholds** | **Alert Priority** |
| --- | --- | --- | --- |
| Application | Response time, error rate, request count | >500ms, >1%, >1000/min | High |
| System | CPU, memory, disk usage, network I/O | >80%, >85%, >90%, >1GB/s | Medium |
| Database | Query time, connection count, replication lag | >200ms, >80%, >10s | High |
| User | Active users, conversion rate, abandonment rate | <100/hour, <70%, >30% | Low |
| Business | Rides completed, payment success rate, user growth | <500/day, <95%, <5%/week | Medium |

#### **13.1.2 Logging Strategy**

* Structured logging (JSON format)
* Log aggregation with Elasticsearch
* Log retention: 30 days hot, 1 year cold
* Log levels: DEBUG, INFO, WARN, ERROR, FATAL
* Correlation IDs across services

### **13.2 Alerting and Incident Response**

#### **13.2.1 Alert Severity Levels**

| **Level** | **Response Time** | **Escalation Path** | **Example Trigger** |
| --- | --- | --- | --- |
| P1 | 15 minutes | On-call → Team → CTO | Service outage, data breach |
| P2 | 1 hour | On-call → Team | API degradation, payment failures |
| P3 | 4 hours | Team | Non-critical service issues |
| P4 | 24 hours | Regular channels | Minor bugs, UI issues |

#### **13.2.2 Incident Management Process**

1. **Detection**: Automated alert or user report
2. **Triage**: Assess impact and urgency
3. **Communication**: Notify stakeholders
4. **Investigation**: Root cause analysis
5. **Resolution**: Fix implementation
6. **Post-mortem**: Documentation and prevention

### **13.3 Backup and Recovery**

* **Database Backups**: Daily full, hourly incremental
* **Configuration Backups**: Version-controlled
* **Recovery Time Objective (RTO)**: 1 hour for critical services
* **Recovery Point Objective (RPO)**: 5 minutes for transaction data
* **Disaster Recovery Drills**: Quarterly testing

### **13.4 Maintenance Windows**

* **Regular Maintenance**: Weekly, 2-hour window during lowest usage
* **Emergency Maintenance**: As required with appropriate notification
* **Version Updates**: Monthly for minor, quarterly for major
* **Security Patches**: Within 48 hours of release

### **13.5 Performance Tuning**

* Regular performance reviews (bi-weekly)
* Automated performance regression testing
* Database query optimization
* CDN caching strategy optimization
* Mobile app performance profiling

## **14. Appendices**

### **14.1 Glossary**

| **Term** | **Definition** |
| --- | --- |
| Okada | Commercial motorcycle taxi |
| Rider | Person operating the motorcycle |
| Passenger | Customer using the motorcycle service |
| SOS | Emergency alert functionality |
| KYC | Know Your Customer (identity verification) |
| FRSC | Federal Road Safety Commission (Nigeria) |
| DVLA | Driver and Vehicle Licensing Authority (Ghana) |

### **14.2 Reference Documents**

* Project Scope Document v1.0
* System Architecture Overview
* Market Research Report
* User Research Findings
* Regulatory Requirements Analysis

### **14.3 API Response Codes and Error Messages**

| **Code** | **Message** | **Description** | **Resolution** |
| --- | --- | --- | --- |
| 200 | OK | Request successful | N/A |
| 201 | Created | Resource successfully created | N/A |
| 400 | Bad Request | Invalid input parameters | Check request format |
| 401 | Unauthorized | Authentication required | Provide valid credentials |
| 403 | Forbidden | Insufficient permissions | Request appropriate access |
| 404 | Not Found | Resource not found | Verify resource identifier |
| 409 | Conflict | Resource state conflict | Resolve conflicting state |
| 422 | Unprocessable Entity | Validation error | Fix validation issues |
| 429 | Too Many Requests | Rate limit exceeded | Reduce request frequency |
| 500 | Internal Server Error | Server-side error | Contact support |
| 503 | Service Unavailable | Temporary unavailability | Retry after delay |

### **14.4 Environment Variables**

| **Variable** | **Purpose** | **Example** | **Required** |
| --- | --- | --- | --- |
| NODE\_ENV | Environment identifier | production | Yes |
| DATABASE\_URL | Database connection string | postgres://user:pass@host:port/db | Yes |
| REDIS\_URL | Redis connection string | redis://user:pass@host:port | Yes |
| JWT\_SECRET | Secret for JWT signing | <random string> | Yes |
| AWS\_REGION | AWS region for services | af-south-1 | Yes |
| GOOGLE\_MAPS\_API\_KEY | Google Maps authentication | <api key> | Yes |
| PAYMENT\_GATEWAY\_KEY | Payment provider auth | <api key> | Yes |
| LOG\_LEVEL | Minimum log level | info | No |

*End of Technical Requirements Document*