Technical Design Document

Technical Design Document

1. Title Page

Project Name: RentalX — Unified Rental & Car Sales Platform

Author(s): [Team of 3 Developers]

Date: May 23, 2025

Version: 1.0

2. Overview / Introduction

Brief Description:

RentalX is a monolithic web-based application for managing car rentals, driver hiring, and car sales. It connects car owners, drivers, and customers through a verified ecosystem.

Purpose of the Document:

To outline the technical architecture, components, data structures, and deployment strategy of the RentalX application.

Scope:

The system includes car and driver registration, KYC verification, rental and booking management, sales of vehicles, and an administrative dashboard.

3. Requirements Summary

Functional Requirements:

- - User authentication and role-based access
- - Car registration by owners
- - Driver registration and verification
- - KYC process for all user roles
- - Car rental booking
- - Driver hiring
- - Car listing and sales module
- - Payment processing

- - Admin panel for managing platform activities

Non-Functional Requirements:

- - Scalability to support increasing user traffic
- - High availability
- - Secure user data handling (encryption, validation)
- - Responsive design for mobile and desktop
- - API rate limiting and error handling

4. Architecture Overview

System Architecture Diagram: Monolithic Architecture Diagram

Technologies:

- - **Frontend**: React.js, Tailwind CSS, TypeScript
- - **Backend**: Golang
- - **Database**: PostgreSQL
- - **Authentication**: JWT-based
- - **Containerization**: Docker, Docker Compose
- - **CI/CD**: GitHub Actions

5. Data Design

Database Schema:

- - Users (id, name, email, password_hash, role, verified)
- - Cars (id, owner_id, make, model, status, type, rental_price, sale_price)
- - Drivers (id, user_id, experience_years, license_info, verified)
- - Bookings (id, customer_id, car_id, driver_id, start_date, end_date, status)
- - Sales (id, car_id, buyer_id, sale_price, date)
- - KYC_Verifications (id, user_id, document_type, document_url, status)

Data Flow Diagrams:

- - Use **DFD Level 0** for overall system flow
- - Use **DFD Level 1** for booking process

6. Component Design

Modules:

- - Auth Module: Manages login/signup and JWT token
- - Car Module: CRUD operations for cars
- - Driver Module: Onboarding, bio, and license validation
- - Booking Module: Handles car/driver reservations
- - Sales Module: Car sales listing and purchase
- - KYC Module: Document upload and verification

- - Notification Module: Email alerts
- - Admin Dashboard: Platform oversight

Interactions:

- - Auth connects all user actions
- - Booking interacts with Car and Driver modules
- - Sales interacts with Car and User modules

7. API Design

Endpoints Examples:

- - `POST /api/auth/register`
- - `GET /api/cars`
- - `POST /api/drivers`
- - `POST /api/kyc/upload`
- - `POST /api/bookings`
- - `POST /api/sales`

Request/Response:

- - JSON format
- - Standard HTTP status codes

Authentication:

- - JWT in Authorization headers
- - Middleware for role-based access

8. User Interface Design

> ***Wireframes: Home, Car Listings, Booking Page, Admin Panel (mockups recommended)***

UI Flow:

- - Login/Register > KYC > Dashboard > Rent/Book/Sell > Payment > Confirmation

9. Security Considerations

- - **Authentication**: JWT
- - **Authorization**: Role-based (Admin, Owner, Driver, Customer)
- - **Data Protection**: HTTPS, input validation, password hashing, file upload filtering

10. Error Handling and Logging

- - Centralized error handler in Golang
- - Standardized API error response
- - Logging via logrus/zap with Docker log streaming

11. Testing Plan

- - **Unit Tests**: Golang testing package
- - **Integration Tests**: Postman / Go test suites
- - **System Tests**: Browser automation (e.g., Cypress for React)

12. Deployment Plan

Environment:

- - VPS or Cloud (AWS, GCP, or DigitalOcean)
- - Docker Compose for service orchestration

CI/CD:

- - GitHub Actions:
- - On push: Build, test, lint
- - On merge: Deploy to staging/production

13. Risks and Assumptions

Risks:

- - Identity verification complexity (KYC APIs)
- - Fraudulent user accounts
- - Scaling under high load

Assumptions:

- - Users have stable internet access
- - External KYC and Payment APIs are reliable

14. Appendix

- - **Glossary:** KYC – Know Your Customer, JWT – JSON Web Token, CRUD – Create Read Update Delete
- - **References:** PostgreSQL docs, Golang stdlib, React documentation
- - **Diagrams:** Use Case Diagram, System Architecture Diagram, DFDs