Vehicle Rental System Documentation

Table of Contents

1. System Overview

The Vehicle Rental System is a modern FastAPI-based backend service that enables comprehensive vehicle rental management with real-time updates. The system implements a microservices architecture using Apache Kafka for event handling and MongoDB for data persistence.

A comprehensive `readme.md` file is included in the project to provide detailed instructions for running and testing all functionalities. Additionally, a concise 2-minute video demo.mov demonstration has been provided to showcase the implementation of authentication, Kafka, and WebSocket functionalities.

2. Architecture Components

2.1 Database Layer

MongoDB serves as the primary database. Collections include: users, vehicles, rentals, and branches. Database configuration is managed in `database.py`

2.2 Event-Driven Architecture

Apache Kafka handles event messaging. Events are managed through:

- `kafka_config/producer.py`: Handles event production
- `kafka_config/consumer.py`: Manages event consumption
- `kafka_config/init_kafka.py`: Initializes Kafka topics
- `kafka_config/event_types.py`: Defines all system events

2.3 Real-time Communication

WebSocket implementation for real-time updates. Managed through:

- `routes/routes_websocket.py`: Handles WebSocket endpoints
- `websocket/connection_manager.py`: Manages WebSocket connections

2.4 Authentication & Authorization

JWT-based authentication system. Role-based access control (CUSTOMER/EMPLOYEE)
Implemented in:

- `middleware/auth.py`: Handles authentication logic
- `middleware/password.py`: Manages password hashing
- `routes/routes_auth.py`: Authentication endpoints

3. Core Modules

3.1 User Management

Python file: `routes/routes_user.py`
Handles user management with CRUD operations (create, read, update, delete), includes password hashing for security, role-based access control (employee/customer), and emits Kafka events for user-related actions.

3.2 Vehicle Management

Python file: `routes/routes_vehicle.py`
Manages rental operations including creating rental requests, updating rental status (employee only), tracking rental history, and handling vehicle availability status changes. includes vehicle filtering by type/price/location/availability. Features WebSocket notifications for real-time updates and Kafka event emission for rental-related actions.

3.3 Rental Management

Python file: `routes/routes_rental.py`
Provides branch location management with CRUD operations, requires employee access for modifications while allowing authenticated users to view branch information. Includes validation for branch data updates and emits Kafka events for branch-related actions.

3.4 Branch Management

Python file: `routes/routes_branch.py`
Handles vehicle management with CRUD operations, , tracks rental history, manages vehicle status updates, and provides real-time WebSocket notifications. Features employee-only access for modifications while allowing authenticated users to view vehicle information.

4. Event Processing System

4.1 Kafka Architecture and Configuration

The system utilizes Apache Kafka for event streaming and system-wide logging. The event-driven architecture ensures reliable tracking and real-time updates across all components. The Kafka broker manages four primary topics: vehicle_events, rental_events, user_events, and branch_events. The infrastructure features automatic topic creation, single partition configuration, and asynchronous event processing with persistent storage.

4.2 Event Management System

Producer System

The producer component handles asynchronous event publishing with guaranteed message delivery. Each event is timestamped and includes detailed payload information. The system implements automatic message serialization and comprehensive error handling with delivery confirmations.

Consumer System

The consumer implementation manages group-based consumption patterns with earliest offset configuration. It features automatic partition assignment and robust error handling mechanisms. The system processes messages through continuous polling and implements graceful shutdown procedures.

Event Categories

- **Vehicle Event Tracking:** Monitors all vehicle-related operations including creation, modifications, deletions, and status changes. Each event captures the complete state of the vehicle at the time of the operation.
- **Branch Management Events:** Tracks branch operations, resource management, and location updates. The system maintains a comprehensive log of all branch-related activities and status changes.
- **Rental Process Events:** Handles the complete rental lifecycle including request handling, approval workflow, completion processing, and status updates. Each stage of the rental process is captured with relevant metadata.
- **User Activity Events:** Manages user-related events including account management, profile modifications, authentication events, and session tracking. The system maintains a complete audit trail of user activities.

5. Real-time Communication System

5.1 WebSocket Implementation

The system employs WebSocket connections to deliver instant notifications for critical system events. When vehicle status changes occur (such as availability updates or maintenance status) or rental request statuses are modified (approved, rejected, or completed), the WebSocket server automatically broadcasts these updates to relevant parties. Customers receive immediate notifications about their rental request status changes, while employees are notified about new rental applications and vehicle status updates in real-time. This integration with the Kafka event system ensures that all stakeholders stay informed about relevant changes without requiring manual refresh of their applications.

6. Rule-based Access Control

The Vehicle Rental System implements a comprehensive security architecture that combines JWT (JSON Web Token) authentication with role-based access control. This system ensures secure access management while maintaining flexibility and scalability.

6.1 Token Generation and Authentication

The system implements JWT (JSON Web Token) authentication with the following features:

- Tokens are generated during user login with a 30-minute expiration time
- Token payload includes user email and role (CUSTOMER/EMPLOYEE)
- Token verification is handled through middleware functions
- Uses OAuth2PasswordBearer scheme for token handling
- Implementation can be found in: `middleware/auth.py`

6.2 Password Security

The system uses bcrypt for password hashing and verification:

- New user passwords are automatically hashed before storage
- Password verification during login compares hashed passwords
- Salt is automatically generated for each password hash
- Implementation can be found in: `middleware/password.py`