Motorcycle Store Management

Analysis and Design Document: Assignment 1

Student: Pop Alexandra

Group: 30435

1.Requirements Analysis

1.1 Assignment Specification

The Motorcycle Shop Management application is a full-stack system that allows users to manage motorcycle inventory, customers (persons), and purchase orders. The app enables users to add, update, and delete motorcycles and persons, and to place and manage orders with multiple motorcycles per order, each with specific quantities. The frontend is built using React and communicates with a RESTful backend developed in Spring Boot.

1.2 Functional Requirements

The application supports the following main functionalities:

1. Allow users to create, update, and delete motorcycle records including details like brand, model, price, engine capacity, and available quantity.
2. Allow users to manage persons by adding, updating, and removing their information (name, age, and email).
3. Enable placing orders that include one or more motorcycles, each with its own quantity and price.
4. Calculate and display the total cost for each order automatically.

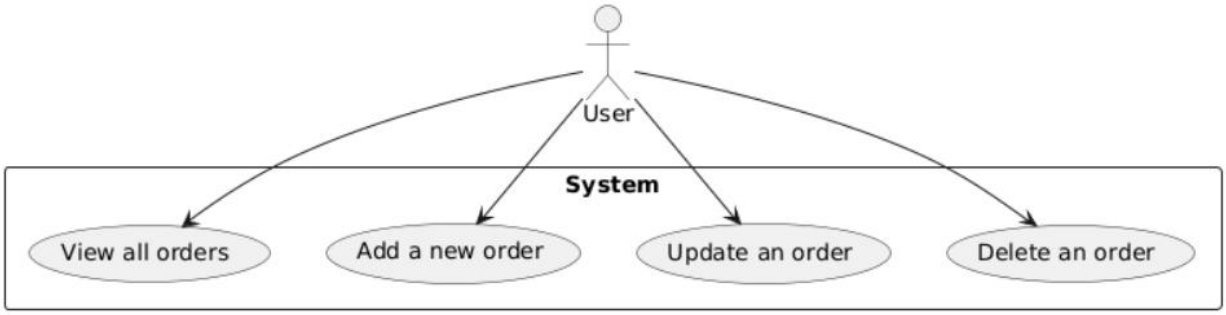
5. Ensure data integrity by validating available quantities and preventing invalid or duplicate data.
6. Automatically update motorcycle availability when orders are placed or modified.

1.3 Non-functional Requirements

The application meets the following non-functional requirements:

1. Uses a relational database (e.g., PostgreSQL) to persist all motorcycle, person, and order data.
2. Follows a layered architecture (Controller-Service-Repository) to promote separation of concerns and maintainability.
3. Uses as the ORM framework to interact with the database.
4. Implements input validation both on the frontend (React) and backend (Spring Boot with Jakarta Validation) to ensure data consistency.

2. Use-Case Model



1. View Orders
Goal: Display all orders placed in the system.
Actor: Application user.
Main Success Scenario: Orders load correctly with motorcycle details and person info.
Failure: Backend unreachable or empty result.

2. Add Order
Goal: Place a new order with multiple motorcycles.
Actor: Application user.
Main Success Scenario: Valid data is submitted, motorcycles are updated, and a new order is stored.
Failure: Invalid input, out-of-stock motorcycles, or backend issues.

3. Update Order
Goal: Modify a previously placed order.

Actor: Application user.
Main Success Scenario: Order is updated and total recalculated.
Failure: Backend error or no longer valid order/motorcycle.

4. Delete Order
Goal: Remove an order from the system.
Actor: Application user.
Main Success Scenario: Order is deleted successfully.
Failure: Backend fails or ID not found.

3. System Architectural Design

3.1 Architectural Pattern Description

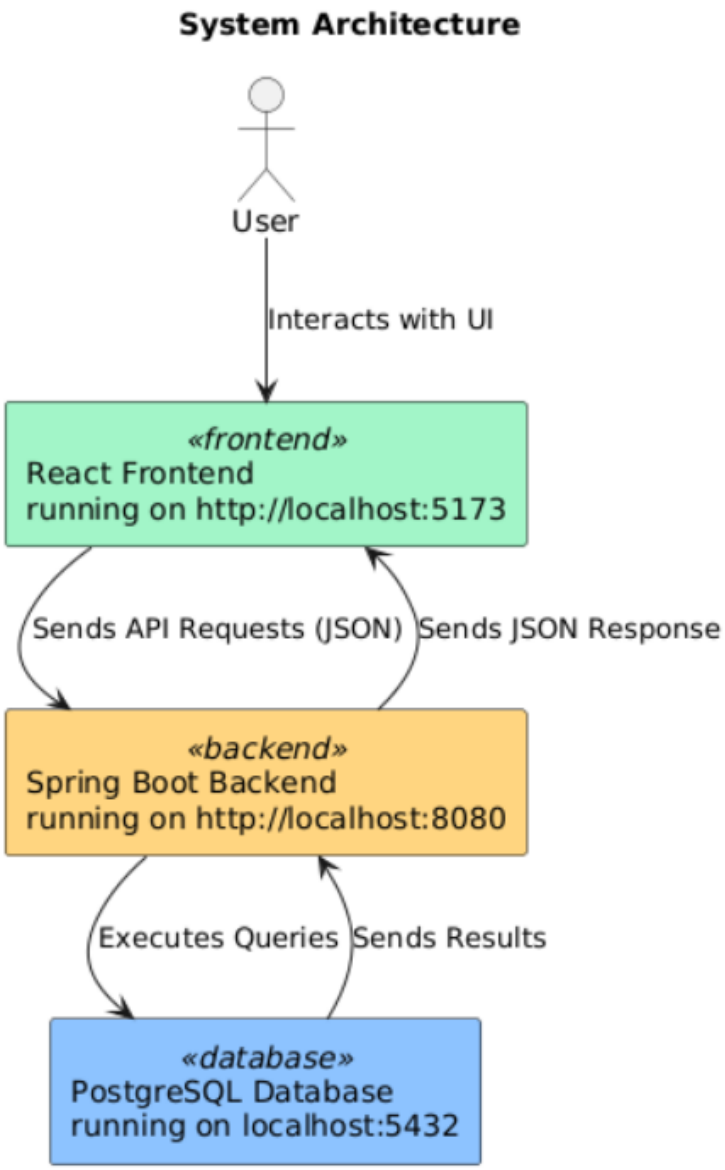
This application follows a layered architecture composed of three main layers: Presentation, Business Logic, and Data Access.

- **Presentation Layer**
  - **React Frontend**
    - Runs in the browser at `http://localhost:5173`
    - Renders the UI components and handles user interactions
    - Sends HTTP requests (GET, POST, PUT, DELETE) to the backend
    - Displays the backend responses in a clean interface

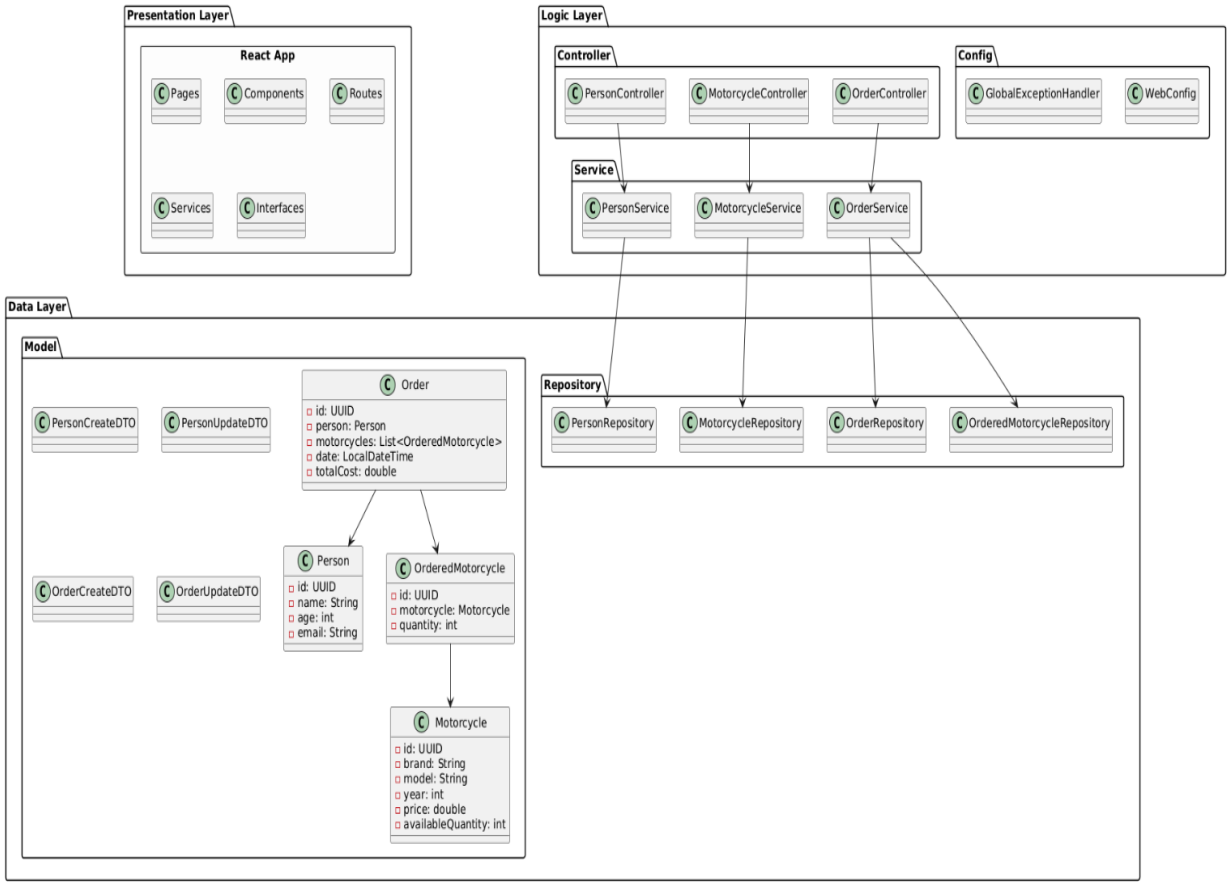
- **Business Logic Layer**
  - **Spring Boot Controllers**
    - Expose REST endpoints on `http://localhost:8080`
    - Receive and handle requests from the frontend
    - Return JSON responses
  - **Spring Boot Services**
    - Implement core logic for `Person`, `Motorcycle`, and `Order` management
    - Validate input data and enforce business rules (e.g., available quantity checks)
    - Update motorcycle availability after orders

- **Data Access Layer**
  - **JPA Entities (Models)**
    - Annotated Java classes that represent database tables like `Person`, `Motorcycle`, `Order`, and `OrderedMotorcycle`
  - **Repositories**
    - Interfaces extending Spring Data JPA's `JpaRepository`
    - Provide abstract methods like `findById`, `save`, `deleteById`, and custom queries
  - **PostgreSQL Database**

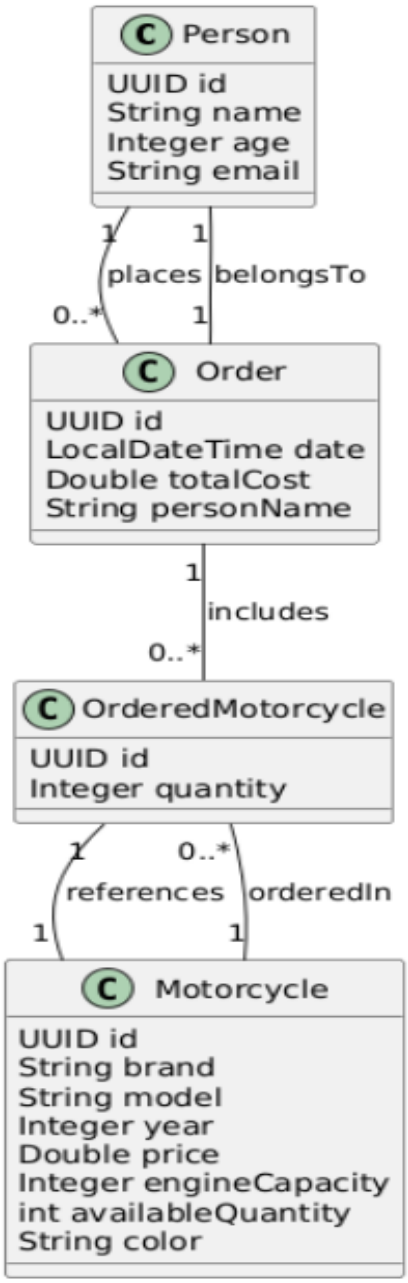
- Runs on `localhost:5432`
- Stores persistent data for all application entities
- Spring Boot communicates with it using JPA/Hibernate



4. Class Diagram



5. Data Model


Person
UUID id
String name
Integer age
String email
1
1
places
belongsTo
0..*
1
Order
UUID id
LocalDateTime date
Double totalCost
String personName
1
includes
0..*
OrderedMotorcycle
UUID id
Integer quantity
1
0..*
references
orderedIn
1
1
Motorcycle
UUID id
String brand
String model
Integer year
Double price
Integer engineCapacity
int availableQuantity
String color