Vehicle Rental Booking System

1. Project Overview

The vehicle rental booking system allows users to select a vehicle type (car or bike), specific model, and rental dates through a form interface. The backend handles vehicle availability checks, booking processing, and data persistence. The frontend presents a multi-step form for a smooth user experience.

---

2. Database Design

The database schema consists of three main tables to store information about vehicle types, models, and bookings. The design follows a **relational database model** suitable for SQL databases.

2.1 Tables and Relationships

1. **VehicleType Table**
    - Stores information about vehicle types (e.g., Car, Bike).
    - **Fields**:
        - `id`: Primary key (Integer, auto-incremented)
        - `name`: Type name (String, e.g., Car, Bike)
2. **VehicleModel Table**
    - Stores individual vehicle models for each type (e.g., SUV, Cruiser).
    - **Fields**:
        - `id`: Primary key (Integer, auto-incremented)
        - `name`: Model name (String, e.g., SUV, Cruiser)
        - `type_id`: Foreign key referencing `VehicleType` (Integer)
        - `wheels`: Number of wheels (Integer, 2 or 4)
3. **Booking Table**
    - Stores booking records, tracking which model was rented by which user, and for what dates.
    - **Fields**:
        - `id`: Primary key (Integer, auto-incremented)
        - `user_name`: Name of the user (String)
        - `vehicle_model_id`: Foreign key referencing `VehicleModel` (Integer)
        - `start_date`: Start date of the booking (Date)
        - `end_date`: End date of the booking (Date)

2.2 Database Diagram

Here's a simplified database diagram:

```
VehicleType       VehicleModel       Booking
-----------       ------------       -------
id (PK)           id (PK)            id (PK)
name              name               user_name
                  type_id (FK)       vehicle_model_id (FK)
                  wheels             start_date
                                      end_date
```

2.3 Seed Data

- Initial vehicle types include Car and Bike.
- Example models include `SUV`, `Hatchback`, `Cruiser`, `Sports` (linked to vehicle types).
- Seed script populates the database with this initial data for quick testing and development.

---

3. Backend Design

The backend is built with Node.js and Express, utilising Sequelize ORM for database interaction. It provides a RESTful API for frontend communication.

3.1 Tech Stack

- **Node.js**: Runtime environment.
- **Express**: Web framework for building the API.
- **Sequelize**: ORM for SQL database operations.
- **MySQL**: Database for storing vehicle and booking data.
- **dotenv**: Environment variable management.

3.2 API Endpoints

1. **Vehicle Data Endpoints**
    - **GET `/api/vehicle-types`**: Fetches all vehicle types (e.g., Car, Bike).
    - **GET `/api/models?type=<vehicle_type>`**: Fetches vehicle models for a given type (e.g., SUVs for Car).
2. **Booking Endpoint**
    - **POST `/api/bookings`**:
        - Accepts booking data: `{ user_name, vehicle_model_id, start_date, end_date }`
        - Checks for overlapping dates to prevent double-booking for the same model.
        - Responds with booking confirmation or error if dates conflict.

3.3 Controllers

1. **Vehicle Controller** (`vehicleController.js`)
    - Handles fetching vehicle types and models.
    - Retrieves data from the database and sends responses.
2. **Booking Controller** (`bookingController.js`)
    - Manages booking requests, checks date availability, and stores valid bookings.
    - Returns error messages for conflicts or incomplete data.

3.4 Middleware and Error Handling

- Middleware validates incoming data and provides meaningful error messages if required fields are missing or invalid.
- Sequelize validation and constraints are also used to ensure data integrity.

---

4. Frontend Design

The frontend is built with React, utilising a multi-step form to capture user data in stages. Material UI is used for consistent styling, and Axios is used to communicate with the backend API.

4.1 Tech Stack

- **React**: Library for building the user interface.
- **Material UI**: Component library for UI styling.
- **Tailwind CSS**: (Optional) Utility classes for custom styles.
- **Axios**: HTTP client for making API requests to the backend.

4.2 UI Components

The frontend consists of several components, each responsible for one step in the booking form:

1. **Step1 Name**:
    - Input for user's first and last names.
    - Validates that both fields are filled before proceeding.
2. **Step2 Wheels**:
    - Radio button input for selecting the number of wheels (2 or 4).
    - Determines whether Car or Bike options are displayed.
3. **Step3 Type**:
    - Radio button for selecting a vehicle type (Car or Bike).
    - Fetches and displays options dynamically from the backend.
4. **Step4 Model**:
    - Radio button to select the specific model of the vehicle (e.g., SUV, Cruiser).
    - Options are dynamically fetched based on the previous selection.
5. **Step5 DateRange**:
    - Date picker component to select start and end dates for booking.
    - Validates that dates are logical (e.g., end date is after start date).
6. **Form Navigator**:
    - Next and Previous buttons for navigation between form steps.
    - Only enables Next after each step's validation passes.

4.3 API Service (`apiService.js`)

- Manages all backend API calls using Axios.
- Functions include:
    - **getVehicleTypes()**: Fetches vehicle types.
    - **getModelsByType(typeId)**: Fetches models based on selected type.
    - **submitBooking(bookingData)**: Submits the final booking to the backend.

4.4 User Flow

1. **Step 1**: User enters their name.
2. **Step 2**: User selects the number of wheels.
3. **Step 3**: User selects the type of vehicle (filtered by wheels).
4. **Step 4**: User selects the specific model (filtered by type).
5. **Step 5**: User selects the booking date range.
6. **Submission**: Form data is validated and sent to the backend, which confirms the booking.

---

5. Validation and Error Handling

1. **Frontend Validation**:

    - Each form step includes validation for required fields, valid dates, and logical choices.
    - User feedback (e.g., error messages) is provided if input is incomplete or invalid.
2. **Backend Validation**:
    - The backend verifies that required fields are present and that booking dates do not overlap for the same vehicle model.
    - Returns descriptive error responses if data is invalid or dates are unavailable.

---

6. Summary

This document covers the structural design for the **database**, **backend**, and **frontend** of the vehicle rental booking system. It includes a relational database schema, RESTful API design, a multi-step form for a streamlined user experience, and validation at both the frontend and backend levels.