Documentation for the

“Hotel Management System Project”

This documentation outlines the creation process and the important parts of the Hotel Management System project, which is a full-stack web application built using React, Node.js, Express, and Oracle Database.

Project Overview

The Hotel Management System is a web application that allows users to interact with various aspects of a hotel, including managing hotels, rooms, guests, bookings, payments, and reviews. It includes functionalities such as:

- Viewing, adding, editing, and deleting hotels, rooms, and guests.
- Creating and managing bookings.
- Managing payments for bookings.
- Posting reviews for hotels and destinations.

The application consists of:

1. **Frontend**: React-based UI with Bootstrap for responsive design.
2. **Backend**: Node.js with Express to handle API requests, and OracleDB for data storage.
3. **Database**: Oracle database with tables for hotels, rooms, guests, bookings, payments, and reviews.

Technologies Used

Frontend:

- **React.js:** JavaScript library for building user interfaces.
- **React Router:** For routing and navigation.
- **React-Bootstrap:** For responsive UI components.

- **Axios:** For making HTTP requests to the backend API.

Backend:

- **Node.js:** JavaScript runtime environment for executing server-side code.
- **Express.js:** Web framework for handling HTTP requests and middleware.
- **OracleDB:** Database used to store hotel data, including rooms, guests, bookings, and more.

Other Tools:

- **dotenv:** For managing environment variables.
- **body-parser:** For parsing incoming request bodies in middleware.
- **nodemon:** For automatic server restarting during development.

Project Structure

Frontend (/frontend/src)

- **/components/**: Contains reusable UI components.
- **CRUDModal.js:** Modal component for adding, editing, and deleting records.
- **DynamicTable.js:** Table component for displaying data dynamically from the API.
- **/pages/**: Contains page components for different parts of the application.
- **HotelPage.js:** Page to manage and display hotels.
- **RoomPage.js:** Page to manage and display rooms.
- **GuestPage.js:** Page to manage and display guests.
- **BookingPage.js:** Page to manage and display bookings.
- **/api.js**: Contains functions to interact with the backend API for performing CRUD operations.
- **App.js**: Main React component that contains routing and links to different pages.

Backend (/backend)

- **server.js**: The entry point of the backend application, setting up the Express server and routes.

- **api.js**: Helper functions for interacting with the OracleDB database, performing CRUD operations.

- **/routes/**: Contains route handlers for API endpoints, including the routes for managing hotels, rooms, guests, bookings, and payments.

Step-by-Step Project Creation

Step 1: Initialize Backend

1. **Initialize the Node.js project and install dependencies:**

```
npm init -y
npm install express oracledb body-parser cors dotenv
npm install --save-dev nodemon
```

2. **Create the server.js file to configure the Express server:**

```
const express = require('express');
const oracledb = require('oracledb');
const bodyParser = require('body-parser');
const cors = require('cors');
const dotenv = require('dotenv');

dotenv.config();

const app = express();
app.use(cors());
app.use(bodyParser.json());

// Connect to OracleDB and perform queries using helper functions

app.listen(process.env.PORT || 5000, () => {
  console.log('Server running on port 5000');
});
```

3. **Create an api.js file for the OracleDB helper functions:**

```
const oracledb = require('oracledb');
```

```
async function executeQuery(query, params = []) {
  const connection = await oracledb.getConnection({
    user: process.env.DB_USER,
    password: process.env.DB_PASSWORD,
    connectString: process.env.DB_CONNECT_STRING,
  });

  const result = await connection.execute(query, params, {
    outFormat: oracledb.OUT_FORMAT_OBJECT,
    autoCommit: true,
  });

  await connection.close();
  return result.rows;
}

module.exports = { executeQuery };
```

4. **Create API routes for managing hotels, rooms, guests, etc.**

Step 2: Initialize Frontend

1. **Initialize the React project:**

```
npx create-react-app frontend
cd frontend
npm install react-router-dom react-bootstrap bootstrap axios
```

2. **Add Bootstrap to your project by importing it in index.js:**

```
import 'bootstrap/dist/css/bootstrap.min.css';
```

3. **Create reusable components for displaying data:**

- **DynamicTable.js**: For rendering tables dynamically based on API data.

- **CRUDModal.js**: For handling modal forms for adding/editing data.

3. Create pages for displaying hotel data and managing bookings, rooms, and guests:

- **HotelPage.js**: Displays hotels and provides functionality for CRUD operations.

- **RoomPage.js**: Displays rooms and allows CRUD operations.

- **GuestPage.js**: Displays guest data and allows CRUD operations.

Step 3: Set Up API Interaction

1. **Create an api.js file in the frontend/src folder to handle API calls:**

```
import axios from 'axios';

const API_URL = "http://localhost:8000/api"; // backend API URL

export const getData = async (tableName) => {
  const response = await axios.get(`${API_URL}/${tableName}`);
  return response.data;
};

export const postData = async (tableName, data) => {
  await axios.post(`${API_URL}/${tableName}`, data);
};

export const putData = async (tableName, id, data) => {
  await axios.put(`${API_URL}/${tableName}/${id}`, data);
};

export const deleteData = async (tableName, id) => {
  await axios.delete(`${API_URL}/${tableName}/${id}`);
};
```

2. **Use the functions from api.js in the components to fetch, add, update, and delete data from the backend.**

Step 4: Add Routing

1. **Use React Router for navigation between pages (HotelPage, RoomPage, etc.):**

```
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import HotelPage from './pages/HotelPage';
import RoomPage from './pages/RoomPage';
import GuestPage from './pages/GuestPage';

const App = () => {
  return (
    <Router>
      <Routes>
        <Route path="/hotels" element={<HotelPage />} />
        <Route path="/rooms" element={<RoomPage />} />
        <Route path="/guests" element={<GuestPage />} />
      </Routes>
    </Router>
  );
};
```

Important Parts of the Project

1. CRUD Operations

The application implements basic **CRUD (Create, Read, Update, Delete)** functionality for tables such as **Hotel**, **Room**, **Guest**, **Booking**, and **Payment**.

- **Create**: The user can add new records to the database.
- **Read**: The user can view records in a table.
- **Update**: The user can edit existing records.
- **Delete**: The user can remove records from the database.

2. Dynamic Tables

The DynamicTable component dynamically fetches and displays data from the backend. The table is designed to handle any table name passed to it, making it reusable across different pages.

3. Modal Form for CRUD Operations

The CRUDModal component is used to perform CRUD operations in a modal form. This modal adapts to the table name and fields dynamically.

Running the Application

Backend:

1. **Install the required dependencies in the backend folder:**

npm install

2. **Start the backend server:**

npm run start

Frontend:

1. **Install the required dependencies in the frontend folder:**

npm install

2. **Start the React development server:**

npm start

Conclusion

This project provides a simple yet functional Hotel Management System with a full-stack implementation using **React**, **Express**, and **OracleDB**. It allows users to view, add, update, and delete records for hotels, rooms, guests, and bookings. The frontend is built using **React** and **React-Bootstrap**, while the backend uses **Node.js**, **Express**, and **OracleDB** for database interaction.