**Assignment: Developing a Flight Reservation System with Groovy REST Client or HttpBuilder**

**Objective**: The purpose of this assignment is to build a flight reservation system using the Groovy REST Client to interact with a RESTful API. The system should allow users to search for flights, view available options, and make reservations.

Requirements:

1. Design and implement a simple RESTful API for a flight reservation system. The API should support the following endpoints:

**Flights**:

* Create a new flight: POST /flights
* Get all flights: GET /flights
* Get a specific flight by ID: GET /flights/:id
* Update an existing flight by ID: PUT /flights/:id
* Delete a flight by ID: DELETE /flights/:id

**Reservations**:

* Create a new reservation: POST /reservations
* Get all reservations: GET /reservations
* Get a specific reservation by ID: GET /reservations/:id
* Update an existing reservation by ID: PUT /reservations/:id
* Delete a reservation by ID: DELETE /reservations/:id

1. Develop a Groovy REST Client to interact with the RESTful API. The client should support the following operations:
   * Search for flights based on origin, destination, and date.
   * View the details of a specific flight.
   * Make a flight reservation.
   * View the details of a specific reservation.
   * Cancel an existing reservation.
2. Implement error handling and input validation to ensure a robust system.
3. Write unit tests for the Groovy REST Client.
4. Document the API and the Groovy REST Client, including how to set up and run the application, as well as examples of how to use the client.

**Guidelines**:

1. Use Groovy and the Groovy REST Client library to interact with the RESTful API.
2. Ensure that the RESTful API follows best practices for API design, including proper use of HTTP methods and status codes.
3. Use a database to store flight and reservation data. Choose an appropriate database system based on your preference and experience.
4. Implement proper error handling and input validation to ensure a robust system. This includes checking for invalid input, handling API errors, and returning appropriate error messages.
5. Write unit tests for the Groovy REST Client to ensure proper functionality and reliability.
6. Provide clear documentation on how to set up and run the application, as well as examples of how to use the Groovy REST Client.

**Submission**: Submit the following items for evaluation:

1. A zip file containing the source code for the RESTful API and the Groovy REST Client.
2. A README.md file with setup instructions, usage examples, and any additional information needed to understand and run the application.
3. A report detailing the design and implementation decisions made during the development of the flight reservation system, as well as any challenges encountered and lessons learned.

**Evaluation Criteria:**

1. Functionality: The flight reservation system should be fully functional and meet all the requirements specified.
2. Code Quality: The code should be well-structured, follow best practices, and include proper error handling and input validation.
3. Documentation: The API and Groovy REST Client should be well-documented, including setup instructions and usage examples.

**JSON Server instructions:**

1. **Create a JSON file with your data:** Create a new file named **db.json** and add the sample JSON data for flights and reservations. Your JSON data should look like this:

{

"flights": [

{

"id": "1",

"origin": "Los Angeles",

"destination": "New York",

"departureDate": "2023-04-10",

"departureTime": "08:00",

"arrivalDate": "2023-04-10",

"arrivalTime": "16:30",

"price": 450,

"currency": "USD",

"availableSeats": 25

},

... // means you can add another object

],

"reservations": [

{

"id": "R1",

"flightId": "1",

"passenger": {

"firstName": "John",

"lastName": "Doe",

"email": "john.doe@example.com"

},

"reservationDate": "2023-03-29",

"status": "CONFIRMED"

},

... // means you can add another object

]

}

1. **Start JSON Server:** In the terminal or command prompt, navigate to the folder containing your **db.json** file, and run the following command to start JSON Server:

**json-server --watch db.json**

1. **Sample API URLs:** With your JSON Server up and running, you can access the following API endpoints:
   * Get all flights: **GET http://localhost:3000/flights**
   * Get a specific flight by ID: **GET http://localhost:3000/flights/1**
   * Get all reservations: **GET http://localhost:3000/reservations**
   * Get a specific reservation by ID: **GET http://localhost:3000/reservations/R1**
   * Create a new reservation: **POST http://localhost:3000/reservations**
   * Update an existing reservation by ID: **PUT http://localhost:3000/reservations/R1**
   * Delete a reservation by ID: **DELETE http://localhost:3000/reservations/R1**

You can also filter, sort, and paginate the results using query parameters. For example:

* + Get flights with a specific origin: **GET http://localhost:3000/flights?origin=Los%20Angeles**
  + Get flights with a specific destination: **GET http://localhost:3000/flights?destination=New%20York**
  + Get flights with a specific departure date: **GET http://localhost:3000/flights?departureDate=2023-04-10**

1. Sample JSON data for the flight reservation system:
2. List of available flights:

[

{

"id": "1",

"origin": "Los Angeles",

"destination": "New York",

"departureDate": "2023-04-10",

"departureTime": "08:00",

"arrivalDate": "2023-04-10",

"arrivalTime": "16:30",

"price": 450,

"currency": "USD",

"availableSeats": 25

},

{

"id": "2",

"origin": "New York",

"destination": "Los Angeles",

"departureDate": "2023-04-11",

"departureTime": "14:00",

"arrivalDate": "2023-04-11",

"arrivalTime": "17:30",

"price": 500,

"currency": "USD",

"availableSeats": 30

},

{

"id": "3",

"origin": "London",

"destination": "Paris",

"departureDate": "2023-04-12",

"departureTime": "10:30",

"arrivalDate": "2023-04-12",

"arrivalTime": "12:00",

"price": 200,

"currency": "EUR",

"availableSeats": 50

}

]

1. Flight details:

{

"id": "1",

"origin": "Los Angeles",

"destination": "New York",

"departureDate": "2023-04-10",

"departureTime": "08:00",

"arrivalDate": "2023-04-10",

"arrivalTime": "16:30",

"price": 450,

"currency": "USD",

"availableSeats": 25

}

1. Reservation details:

{

"id": "R1",

"flightId": "1",

"passenger": {

"firstName": "John",

"lastName": "Doe",

"email": "john.doe@example.com"

},

"reservationDate": "2023-03-29",

"status": "CONFIRMED"

}

1. List of reservations:

[

{

"id": "R1",

"flightId": "1",

"passenger": {

"firstName": "John",

"lastName": "Doe",

"email": "john.doe@example.com"

},

"reservationDate": "2023-03-29",

"status": "CONFIRMED"

},

{

"id": "R2",

"flightId": "2",

"passenger": {

"firstName": "Jane",

"lastName": "Smith",

"email": "jane.smith@example.com"

},

"reservationDate": "2023-03-28",

"status": "CONFIRMED"

}

]

You can use this sample JSON data as a starting point for your flight reservation system.

Sample output for each CRUD API endpoint in the flight reservation system:

1. **Flights**:

Create a new flight (**POST /flights**):

Request body:

{

"origin": "Boston",

"destination": "San Francisco",

"departureDate": "2023-04-15",

"departureTime": "09:00",

"arrivalDate": "2023-04-15",

"arrivalTime": "12:30",

"price": 600,

"currency": "USD",

"availableSeats": 20

}

Response:

{

"id": "4",

"origin": "Boston",

"destination": "San Francisco",

"departureDate": "2023-04-15",

"departureTime": "09:00",

"arrivalDate": "2023-04-15",

"arrivalTime": "12:30",

"price": 600,

"currency": "USD",

"availableSeats": 20

}

Get all flights (**GET /flights**):

Response:

[

{

"id": "1",

"origin": "Los Angeles",

"destination": "New York",

"departureDate": "2023-04-10",

"departureTime": "08:00",

"arrivalDate": "2023-04-10",

"arrivalTime": "16:30",

"price": 450,

"currency": "USD",

"availableSeats": 25

},

...

]

Get a specific flight by ID (**GET /flights/1**):

Response:

{

"id": "1",

"origin": "Los Angeles",

"destination": "New York",

"departureDate": "2023-04-10",

"departureTime": "08:00",

"arrivalDate": "2023-04-10",

"arrivalTime": "16:30",

"price": 450,

"currency": "USD",

"availableSeats": 25

}

Update an existing flight by ID (**PATCH /flights/1**):

Request body:

{

"availableSeats": 20

}

Response:

{

"id": "1",

"origin": "Los Angeles",

"destination": "New York",

"departureDate": "2023-04-10",

"departureTime": "08:00",

"arrivalDate": "2023-04-10",

"arrivalTime": "16:30",

"price": 450,

"currency": "USD",

"availableSeats": 20

}

Delete a flight by ID (**DELETE /flights/1**):

Response:

HTTP 204 No Content

1. **Reservations**:

* Create a new reservation (**POST /reservations**):

Request body:

{

"flightId": "2",

"passenger": {

"firstName": "Emily",

"lastName": "Johnson",

"email": "emily.johnson@example.com"

}

}

Response:

{

"id": "R3",

"flightId": "2",

"passenger": {

"firstName": "Emily",

"lastName": "Johnson",

"email": "emily.johnson@example.com"

},

"reservationDate": "2023-03-29",

"status": "CONFIRMED"

}

Get all reservations (**GET /reservations**):

Response:

[

{

"id": "R1",

"flightId": "1",

"passenger": {

"firstName": "John",

"lastName": "Doe",

"email": "john.doe@example.com"

},

"reservationDate": "2023-03-29",

"status": "CONFIRMED"

},

...

]

Get a specific reservation by ID (**GET /reservations/R1**):

Response:

{

"id": "R1",

"flightId": "1",

"passenger": {

"firstName": "John",

"lastName": "Doe",

"email": "john.doe@example.com"

},

"reservationDate": "2023-03-29",

"status": "CONFIRMED"

}

Update an existing reservation by ID (**PATCH /reservations/R1**):

Request body:

{

"status": "CANCELLED"

}

Response:

{

"id": "R1",

"flightId": "1",

"passenger": {

"firstName": "John",

"lastName": "Doe",

"email": "john.doe@example.com"

},

"reservationDate": "2023-03-29",

"status": "CANCELLED"

}

Delete a reservation by ID (**DELETE /reservations/R1**):

Response:

HTTP 204 No Content