**Solution for Flight reservation system assignment**

Below is a solution to implement the flight reservation system using Groovy REST client and JSON Server as a mock backend:

1. Set up the JSON Server with the provided instructions in the previous responses. Make sure it's running and accessible at **http://localhost:3000**.
2. Install Groovy, if you haven't already, by following the instructions on the official website: <https://groovy-lang.org/install.html>.
3. Create a new Groovy script file, **FlightReservationClient.groovy**, and import the required classes:

import groovy.json.JsonOutput

import groovy.json.JsonSlurper

import groovyx.net.http.ContentType

import groovyx.net.http.HTTPBuilder

import groovyx.net.http.Method

1. Configure the REST client to use the JSON Server:

def baseURL = 'http://localhost:3000'

def restClient = new HTTPBuilder(baseURL)

1. Implement the helper methods for CRUD operations:

// Get all flights

def getAllFlights() {

restClient.request(Method.GET, ContentType.JSON) {

uri.path = '/flights'

response.success = { resp, json ->

return json

}

}

}

// Get all reservations

def getAllReservations() {

restClient.request(Method.GET, ContentType.JSON) {

uri.path = '/reservations'

response.success = { resp, json ->

return json

}

}

}

// Create a new reservation

def createReservation(Map reservation) {

restClient.request(Method.POST, ContentType.JSON) {

uri.path = '/reservations'

body = reservation

response.success = { resp, json ->

return json

}

}

}

// Get a specific reservation by ID

def getReservationById(String reservationId) {

restClient.request(Method.GET, ContentType.JSON) {

uri.path = "/reservations/${reservationId}"

response.success = { resp, json ->

return json

}

}

}

// Update an existing reservation by ID

def updateReservation(String reservationId, Map updates) {

restClient.request(Method.PUT, ContentType.JSON) {

uri.path = "/reservations/${reservationId}"

body = updates

response.success = { resp, json ->

return json

}

}

}

// Delete a reservation by ID

def deleteReservation(String reservationId) {

restClient.request(Method.DELETE, ContentType.JSON) {

uri.path = "/reservations/${reservationId}"

response.success = { resp ->

return true

}

}

}

1. Test the helper methods with sample data:

// Get all flights

println "All flights:"

println JsonOutput.prettyPrint(JsonOutput.toJson(getAllFlights()))

// Get all reservations

println "All reservations:"

println JsonOutput.prettyPrint(JsonOutput.toJson(getAllReservations()))

// Create a new reservation

def newReservation = [

flightId: "2",

passenger: [

firstName: "Emily",

lastName: "Johnson",

email: "emily.johnson@example.com"

]

]

println "New reservation:"

println JsonOutput.prettyPrint(JsonOutput.toJson(createReservation(newReservation)))

// Get a specific reservation by ID

println "Reservation R1:"

println JsonOutput.prettyPrint(JsonOutput.toJson(getReservationById("R1")))

// Update an existing reservation by ID

def updates = [status: "CANCELLED"]

println "Updated reservation R1:"

println JsonOutput.prettyPrint(JsonOutput.toJson(updateReservation("R1", updates)))

// Delete a reservation by ID

println "Deleted reservation R1: ${deleteReservation("R1")}"

1. Run the Groovy script by executing the following command in the terminal or command prompt:

**groovy FlightReservationClient.groovy**

After executing the script, you should see the output corresponding to the sample data and operations performed using the helper methods. The output will show results for retrieving flights and reservations, creating a new reservation, fetching a specific reservation by ID, updating a reservation, and deleting a reservation.

This solution demonstrates how to use the Groovy REST client to interact with a mock backend (JSON Server) for a flight reservation system. You can further extend this solution to include more advanced operations and error handling as needed.

Solution using **RESTClient** instead of **HTTPBuilder**. Note that you need to have the **groovy-wslite** library in your classpath to use **RESTClient**. You can add it as a dependency using Gradle or Maven.

1. Create a new Groovy script file, **FlightReservationRESTClient.groovy**, and import the required classes:

import groovy.json.JsonOutput

import groovy.json.JsonSlurper

import wslite.rest.RESTClient

1. Configure the REST client to use the JSON Server:

def baseURL = 'http://localhost:3000'

def restClient = new RESTClient(baseURL)

1. Implement the helper methods for CRUD operations:

// Get all flights

def getAllFlights() {

def response = restClient.get(path: '/flights')

return response.json

}

// Get all reservations

def getAllReservations() {

def response = restClient.get(path: '/reservations')

return response.json

}

// Create a new reservation

def createReservation(Map reservation) {

def response = restClient.post(path: '/reservations', contentType: 'application/json', body: reservation)

return response.json

}

// Get a specific reservation by ID

def getReservationById(String reservationId) {

def response = restClient.get(path: "/reservations/${reservationId}")

return response.json

}

// Update an existing reservation by ID

def updateReservation(String reservationId, Map updates) {

def response = restClient.put(path: "/reservations/${reservationId}", contentType: 'application/json', body: updates)

return response.json

}

// Delete a reservation by ID

def deleteReservation(String reservationId) {

def response = restClient.delete(path: "/reservations/${reservationId}")

return response.status == 204

}

1. Test the helper methods with sample data, just as in the previous solution:

// Get all flights

println "All flights:"

println JsonOutput.prettyPrint(JsonOutput.toJson(getAllFlights()))

// Get all reservations

println "All reservations:"

println JsonOutput.prettyPrint(JsonOutput.toJson(getAllReservations()))

// Create a new reservation

def newReservation = [

flightId: "2",

passenger: [

firstName: "Emily",

lastName: "Johnson",

email: "emily.johnson@example.com"

]

]

println "New reservation:"

println JsonOutput.prettyPrint(JsonOutput.toJson(createReservation(newReservation)))

// Get a specific reservation by ID

println "Reservation R1:"

println JsonOutput.prettyPrint(JsonOutput.toJson(getReservationById("R1")))

// Update an existing reservation by ID

def updates = [status: "CANCELLED"]

println "Updated reservation R1:"

println JsonOutput.prettyPrint(JsonOutput.toJson(updateReservation("R1", updates)))

// Delete a reservation by ID

println "Deleted reservation R1: ${deleteReservation("R1")}"

1. Run the Groovy script by executing the following command in the terminal or command prompt:

**groovy FlightReservationRESTClient.groovy**

After executing the script, you should see the output corresponding to the sample data and operations performed using the helper methods. This solution demonstrates how to use the **RESTClient** in Groovy to interact with a mock backend (JSON Server) for a flight reservation system. You can further extend this solution to include more advanced operations and error handling as needed.

**RESTClient and HTTPBuilder? Which one is better?**

Both **RESTClient** and **HTTPBuilder** are popular choices for making HTTP requests in Groovy, and each has its advantages and disadvantages. Here's a comparison to help you decide which one to use:

1. RESTClient:
   * Part of the **groovy-wslite** library, which is lightweight and focused on RESTful web services.
   * Provides a simple, concise syntax for making requests.
   * Can be easier to understand for those who are new to Groovy or prefer a more "Groovy-like" syntax.
2. HTTPBuilder:
   * Part of the **http-builder-ng** library, which is more versatile and powerful.
   * Allows for more customization and advanced features like request/response interceptors, authentication, and proxy support.
   * Can be more suitable for complex scenarios where you need finer control over the request and response processing.

In summary, if you're working on a simple project or prefer a more straightforward syntax, **RESTClient** might be the better choice. On the other hand, if you need advanced features or expect your project to grow in complexity, **HTTPBuilder** would be more suitable.

Keep in mind that **http-builder-ng** is the successor to the original **HTTPBuilder** library, so it is recommended to use **http-builder-ng** if you decide to go with **HTTPBuilder**.

Finally, your choice may also depend on your personal preference and which library you find more convenient and easier to work with. Try both and see which one fits your requirements and style better.

Top of Form

Bottom of Form