CSA -CW

20220616

Hashir

Health System API

This project consists of 7 classes which all of them are included with CRUD operations related to each.

The project file hierarchy follows 4 separate packages to ensure proper system architecture layers.

The data layer and logic layers are the only layers included in the package.

The data layer is in model package where all the object and relevant methods are included.

The logic layer is in 2 separate packages where the operation layers are in DAO and the web requests resources are in resources package.

DAO package includes all the methods relevant to the request done by the resource package.

Each class follows this structure under relevant packages.

* Person.model.java
* Person.DAO.java
* Person.resources.java

In order to control errors an exception package is created where all the relevant methods required for the application running when encountered errors are available.

Each resource class has the main 4 crud operation path methods to execute requests.

@GET – retrieves and displays all data available.

@POST – add the relevant data posted.

@PUT – update the relevant data requested.

@DELETE – removes the data requested.

To test them , the url which is <http://localhost:8080/cw/rest/> is used.

In here the “/rest” points to where the requests should be done.

To check it in postman this is how it should be done.

Consider the example Appointment class methods.

- GET /Appointment/: Retrieves all appointments.

- GET /Appointment/{appointmentID}: Retrieves a specific appointment by its ID.

- POST /Appointment: Creates a new appointment.

- PUT /Appointment/{appointmentID}: Updates an existing appointment.

- DELETE /Appointment/{appointmentID}: Deletes an appointment by its ID.

Usage Instructions:

1. GET /Appointment:

- Retrieve all appointments.

- No request body required.

- Returns a list of all appointments in JSON format.

- If no appointments are found, a `404 Not Found` error is returned.

2. GET /Appointment/{appointmentID}:

- Retrieve a specific appointment by its ID.

- Provide the appointment ID in the URL path.

- Returns the appointment details in JSON format.

- If the appointment with the specified ID is not found, a `404 Not Found` error is returned.

3. POST /Appointment:

- Create a new appointment.

- Provide the appointment details in the request body in JSON format.

- Returns a `200 OK` response if the appointment is successfully created.

- If the appointment ID is missing in the request body, a `400 Bad Request` error is returned.

4. PUT /Appointment/{appointmentID}:

- Update an existing appointment.

- Provide the appointment ID in the URL path.

- Provide the updated appointment details in the request body in JSON format.

- Returns a `200 OK` response if the appointment is successfully updated.

- If the appointment with the specified ID is not found, a `404 Not Found` error is returned.

5. DELETE /Appointment/{appointmentID}:

- Delete an appointment by its ID.

- Provide the appointment ID in the URL path.

- Returns a `200 OK` response if the appointment is successfully deleted.

- If the appointment with the specified ID is not found, a `404 Not Found` error is returned.

Error Handling:

- If an appointment is not found (e.g., when fetching by ID), a `404 Not Found` error with the message No appointments found with ID in the body is returned.

- If an appointment cannot be created due to missing fields, a `400 Bad Request` error is returned.

- If an appointment cannot be updated or deleted because it does not exist, a `404 Not Found` error is returned.