**PART -A**

1. The path /employees/{id} provides access to an Employee resource specified by an ID. Four HTTP methods can be invoked on this resource. Your task is to draw a similar table to the following on your paper and fill all corresponding fields which allow performing those four HTTP operations. For each operation, you need to write three types of annotations. [20 (5+5+5+5) marks]

Solution:

|  |  |  |  |
| --- | --- | --- | --- |
| HTTP Operation | Annotation for Corresponding HTTP Operation | Annotation to Describe the Data Type if a Method Produces or Consumes | Annotation to Describe Parameter(s) |
| GET | @GET | @Produces(MediaType.APPLICATION\_XML) | @PathParam("id") |
| POST | @POST | @Consumes(MediaType.APPLICATION\_XML) | @PathParam("id") |
| PUT | @PUT | @Consumes(MediaType.APPLICATION\_XML) | @PathParam("id") |
| DELETE | @DELETE | *(No data type annotation needed)* | @PathParam("id") |

**Because we know each operation and they are:**

1. **GET**: Retrieves the resource representation in XML format.
2. **POST**: Creates or updates the resource by consuming an XML payload.
3. **PUT**: Updates the resource, consuming an XML payload.
4. **DELETE**: Deletes the resource, doesn't produce or consume any content, so no data type annotation is required.

2. In the given following scenarios, identify the type of web services that can be designed [15 marks]

1) Developer only wants to use HTTP as the transfer protocol and NOT any other protocol [3 marks]

A. REST, B. SOAP C. Both

Answer: A. REST

Because REST is coupled with HTTP and is designed to work directly with it meanwhile SOAP, can use other protocols like SMTP or TCP in addition to HTTP.

2) Developer only wants to represent messages using XML and NOT any other representation [3 marks]

A. REST, B. SOAP C. Both

Answer: B. SOAP

Because SOAP strictly uses XML for message format and REST can use various formats like JSON, XML, or plain text.

3) Developer wants to utilize his expertise in JAX- RS [3 marks]

A. REST, B. SOAP C. Both

Answer: A. REST

Because JAX-RS (Java API for RESTful Web Services) is specifically designed for implementing RESTful web services.

4) Company providing web services wants to enforce a formal contract [3 marks]

A. REST, B. SOAP C. Both

Answer: B. SOAP

Because SOAP uses WSDL (Web Services Description Language) to enforce a formal, strongly typed contract. REST does not inherently support formal contracts.

5) Client wants to access an object on a server, but the client does not always use the web [3 marks]

A. REST, B. SOAP C. Both

Answer: B. SOAP

Because SOAP can operate over protocols other than HTTP, making it suitable for scenarios where the client does not always rely on web-based communication. I can say REST, being web-centric, requires HTTP.

3. Suppose you are developing a REST-Full API for managing a payroll system for employees working in a company. In that company, employees are being paid on different scales. For such a payroll system, associate HTTP verbs to perform the following 5 scenarios [15 marks]

Since we know that the operations are:

* **GET** is used for read-only operations.
* **POST** is used for creating new records.
* **DELETE** removes resources.
* **PUT** modifies or updates existing resources.

1) Get a list of all employees [3 marks]

Solution:

**HTTP Verb:** **GET**

* Used to retrieve data without modifying the server state.
* Endpoint should be: /employees

2) Register a particular employee for a pay scale [3 marks]

Solution:

**HTTP Verb:** **POST**

* Used to create or add new data on the server.
* Endpoint: /employees/{id}/pay-scale

3) Get all the employees who have registered for any particular pay scale [3 marks]

Solution:

**HTTP Verb:** **GET**

* Used to retrieve data for specific criteria.
* Endpoint: /pay-scales/{scale\_id}/employees

4) Delete the record of a pay-scale details [3 marks]

Solution:

**HTTP Verb:** **DELETE**

* Used to remove data from the server.
* Endpoint: /pay-scales/{scale\_id}

5) Update the pay scale of an employee [3 marks]

Solution:

**HTTP Verb:** **PUT**

* Used to update or modify existing data on the server.
* Endpoint: /employees/{id}/pay-scale

**PART -B**

**Management API Project Documentation**

# **Overview**

The end number of my student number is ending with 7, so I have done the management API in a Java-based RESTful service that allows users to manage projects and tasks.

My project provides endpoints to create, retrieve, and delete projects and tasks. The API requires an API key for certain operations to interact with the application throughout POSTMAN api = (“API-Key”), value = 1234.

# **Features**

1. Project Management
   * Create a new project.
   * Retrieve all projects.
   * Delete an existing project.
2. Task Management
   * Create a task for a specific project.
   * Retrieve all tasks across all projects.
3. Security
   * API key validation for protected endpoints, api = (“API-Key”), value = 1234.

# **Getting Started**

You can download the project from this link:

<https://github.com/EskandarAtrakchi/Management-RESTful-API-/raw/refs/heads/main/TaskManager-TABA.zip>

Then open the project on NetBeans, once it is open you can clean and build just in case then run the project

A screenshot of a computer program

Description automatically generated

After clean and build is successful run the project

A screenshot of a computer

Description automatically generated

The browser should pop up and this message should show with URL <http://localhost:8080/TaskManager/>

A screenshot of a computer

Description automatically generated

If we change the local host URL to this [localhost:8080/TaskManager/api/management/projects](http://localhost:8080/TaskManager/api/management/projects) the empty array will show, that means we do not have projects or tasks

A screenshot of a computer

Description automatically generated

To post on POSTMAN we make sure that the API-Key is correct, and the value is correct.  
The we check the URL.

A screenshot of a computer

Description automatically generated

Then we add the body with raw JSON body

A screenshot of a computer

Description automatically generated

After clicking on send the message should show (Project created successfully) with 201 created

And when we go to the browser, we should see the message after refreshing the browser

A screenshot of a computer

Description automatically generated

We can also get the message using POSTMAN by adding the API-Key and value with the correct URL the click on send

A screenshot of a computer

Description automatically generated

# **APIs**

The base URL for the API is:

[*http://localhost:8080/TaskManager/api/management*](http://localhost:8080/TaskManager/api/management)

**API Endpoints**

1. Get All Projects

* Endpoint: /projects
* Method: GET
* Description: Retrieve all available projects.
* Example: GET <http://localhost:8080/TaskManager/api/management/projects>

1. Create a Project

* Endpoint: /projects
* Method: POST
* Description: Create a new project.
* Headers:
  + API-Key: 1234
* Body:  
  {

"name": "Project Alpha",

"description": "Initial Test Project",

"tasks": []

}

* Example:  
  POST http://localhost:8080/TaskManager/api/management/projects \

-H "API-Key: 1234" \

-H "Content-Type: application/json" \

-d '{

"name": "Project Alpha",

"description": "Initial Test Project",

"tasks": []

}'

1. Delete a Project

* Endpoint: /projects/{id}
* Method: DELETE
* Description: Delete a project by its ID.
* Headers:
  + API-Key: 1234
* Example:  
  DELETE http://localhost:8080/TaskManager/api/management/projects/1 \  
  -H "API-Key: 1234"

1. Create a Task

* Endpoint: /projects/{projectId}/tasks
* Method: POST
* Description: Add a task to a specific project.
* Headers:
  + API-Key: 1234
* Body:  
  {

"name": "Task 1",

"description": "Sample task for Project Alpha",

"status": "Pending"

}

* Example:  
  POST http://localhost:8080/TaskManager/api/management/projects/1/tasks\

-H "API-Key: 1234" \

-H "Content-Type: application/json" \

-d '{

"name": "Task 1",

"description": "Sample task for Project Alpha",

"status": "Pending"

}'

1. Get All Tasks

* Endpoint: /tasks
* Method: GET
* Description: Retrieve all tasks across all projects.
* Example: GET <http://localhost:8080/TaskManager/api/management/tasks>

# **Conclusion**

This Management API provides a very important foundation for managing projects and tasks. I have done all the steps by following what the brief required me to deliver. Anyone can set up and run the service, and extend it as needed for their specific requirements. The project has been committed to GitHub can be accessed here <https://github.com/EskandarAtrakchi/Management-RESTful-API->