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

Below is the structure and implementation for a **Management RESTful API** that satisfies the requirements you provided. The service uses in-memory collections (List and Map) for storing projects and tasks. Each endpoint is implemented in a RESTful manner using Jakarta RESTful Web Services (JAX-RS).

**Steps to Implement the Project**

**1. Define Models**

Create two model classes: Project and Task.

package com.mycompany.management.models;

import java.util.ArrayList;

import java.util.List;

public class Project {

private int projectId;

private String name;

private String description;

private List<Task> tasks = new ArrayList<>();

// Getters and Setters

public int getProjectId() {

return projectId;

}

public void setProjectId(int projectId) {

this.projectId = projectId;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public List<Task> getTasks() {

return tasks;

}

public void setTasks(List<Task> tasks) {

this.tasks = tasks;

}

}

package com.mycompany.management.models;

public class Task {

private int taskId;

private String name;

private String description;

private String status; // "pending" or "completed"

// Getters and Setters

public int getTaskId() {

return taskId;

}

public void setTaskId(int taskId) {

this.taskId = taskId;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getDescription() {

return description;

}

public void setDescription(String description) {

this.description = description;

}

public String getStatus() {

return status;

}

public void setStatus(String status) {

this.status = status;

}

}

**2. Service Layer**

Create a ManagementService class to handle in-memory storage and operations for Project and Task.

package com.mycompany.management.services;

import com.mycompany.management.models.Project;

import com.mycompany.management.models.Task;

import java.util.ArrayList;

import java.util.HashMap;

import java.util.List;

import java.util.Map;

public class ManagementService {

private final Map<Integer, Project> projects = new HashMap<>();

private int nextProjectId = 1;

private int nextTaskId = 1;

public List<Project> getAllProjects() {

return new ArrayList<>(projects.values());

}

public Project getProjectById(int projectId) {

return projects.get(projectId);

}

public void createProject(Project project) {

project.setProjectId(nextProjectId++);

projects.put(project.getProjectId(), project);

}

public void updateProject(int projectId, String name, String description) {

Project project = projects.get(projectId);

if (project != null) {

project.setName(name);

project.setDescription(description);

}

}

public void deleteProject(int projectId) {

projects.remove(projectId);

}

public List<Task> getTasksByProjectId(int projectId) {

Project project = projects.get(projectId);

return project != null ? project.getTasks() : new ArrayList<>();

}

public void createTask(int projectId, Task task) {

Project project = projects.get(projectId);

if (project != null) {

task.setTaskId(nextTaskId++);

project.getTasks().add(task);

}

}

public void updateTask(int taskId, String name, String description, String status) {

projects.values().forEach(project -> {

project.getTasks().stream().filter(task -> task.getTaskId() == taskId).findFirst().ifPresent(task -> {

task.setName(name);

task.setDescription(description);

task.setStatus(status);

});

});

}

public void deleteTask(int taskId) {

projects.values().forEach(project -> project.getTasks().removeIf(task -> task.getTaskId() == taskId));

}

}

**3. Resource Layer**

Create a ManagementResource class to handle HTTP requests.

package com.mycompany.management.resources;

import com.mycompany.management.models.Project;

import com.mycompany.management.models.Task;

import com.mycompany.management.services.ManagementService;

import jakarta.ws.rs.\*;

import jakarta.ws.rs.core.MediaType;

import jakarta.ws.rs.core.Response;

import java.util.List;

@Path("/management")

public class ManagementResource {

private final ManagementService service = new ManagementService();

private static final String VALID\_API\_KEY = "valid-api-key";

private boolean isValidApiKey(String apiKey) {

return VALID\_API\_KEY.equals(apiKey);

}

@GET

@Path("/projects")

@Produces(MediaType.APPLICATION\_JSON)

public Response getAllProjects() {

return Response.ok(service.getAllProjects()).build();

}

@POST

@Path("/projects")

@Consumes(MediaType.APPLICATION\_JSON)

public Response createProject(Project project, @HeaderParam("API-Key") String apiKey) {

if (!isValidApiKey(apiKey)) {

return Response.status(Response.Status.FORBIDDEN).build();

}

service.createProject(project);

return Response.status(Response.Status.CREATED).entity("Project created successfully").build();

}

@DELETE

@Path("/projects/{id}")

public Response deleteProject(@PathParam("id") int projectId, @HeaderParam("API-Key") String apiKey) {

if (!isValidApiKey(apiKey)) {

return Response.status(Response.Status.FORBIDDEN).build();

}

service.deleteProject(projectId);

return Response.ok("Project deleted successfully").build();

}

@POST

@Path("/projects/{projectId}/tasks")

@Consumes(MediaType.APPLICATION\_JSON)

public Response createTask(@PathParam("projectId") int projectId, Task task, @HeaderParam("API-Key") String apiKey) {

if (!isValidApiKey(apiKey)) {

return Response.status(Response.Status.FORBIDDEN).build();

}

service.createTask(projectId, task);

return Response.status(Response.Status.CREATED).entity("Task created successfully").build();

}

@GET

@Path("/tasks")

@Produces(MediaType.APPLICATION\_JSON)

public Response getAllTasks() {

List<Task> allTasks = service.getAllProjects().stream()

.flatMap(project -> project.getTasks().stream())

.toList();

return Response.ok(allTasks).build();

}

}

**4. Test Using Postman**

* Use API endpoints like:
  + POST /management/projects
  + GET /management/projects
  + DELETE /management/projects/{id}
  + POST /management/projects/{projectId}/tasks

**5. Deliverables**

* Zip the **NetBeans project**.
* Take screenshots of the Postman tests for CRUD operations on projects and tasks.
* Add documentation describing the API endpoints.