ADVANCED PROGRAMMING

National College of Ireland

BSc. in Computing – Year 3 (BSHC3)

BSc. in Computing, Evening – Year 3 (BSHCE3)

Dr Abdul Razzaq

TABA Release date: 2nd December (week 12) 2024

Deadline for Online Moodle Submission: Tuesday, 9th January 2025 (23:55 Irish Time)

Weight: The weight is 50% and the assignment will be marked out of 100.

LEARNING OBJECTIVE:

This assessment examines the following learning objectives for the module:

LO1	Explain the theory, concepts and principles of distributed systems operation and design
LO2	Compare and contrast the methods, theories, and concepts of Web Services.
LO3	Demonstrate conceptual, technical and practical skills in the analysis, design and test of distributed systems
LO4	Demonstrate conceptual, technical and practical skills in the implementation of advanced communication systems.
LO5	Demonstrate conceptual and technical skills in the analysis, design and implementation, management of systems and applications based on web services and REST web services.

SUBMISSION DETAILS:

- Answer all of the questions.
- For Part 1, you need to submit your solution in word format.
- For part 2, one NetBeans application must be submitted online on Moodle as a 1 zip file with the entire application.

This is a terminal assessment. NO late submission or extension will be allowed.

Turnitin tool will be used to check for plagiarism and similarity for all submitted documents. Submissions with a high level of plagiarism will be referred to Academic Honesty & Integrity committee.

PART A – Theory [50 marks]

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]

HTTP Operations	Annotation for Corresponding HTTP Operation Type	Annotation to Describe the Data Type **IF** a Method Produces or Consumes	Annotation to Describe Parameter(s)
Operation 1: GET	@GET	@Produces(MediaType.XML)	@path-param({id})
Operation 2: POST	@POST	@Consumes(MediaType.XML)	@path-param({id})
Operation 3: PUT	@PUT	@Consumes(MediaType.XML)	@path-param({id})
Operation 4: DELETE	@DELETE		@path-param({id})

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

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

A. REST, B. SOAP C. Both

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

A. REST, B. SOAP C. Both

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

A. REST, B. SOAP C. Both

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

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]

1) Get a list of all employees [3 marks]
2) Register a particular employee for a pay scale [3 marks]
3) Get all the employees who have registered for any particular pay scale [3 marks]
4) Delete the record of a pay-scale details [3 marks]
5) Update the pay scale of an employee [3 marks]

PART B – Practical [50 marks]

STUDENTS whose IDs end with digits 0-4 will complete the following project.

Consider an Online Quiz Management RESTful API. You will design and implement a RESTful web service in Java that manages an online quiz system without using a database or Swagger. The API should allow users (teachers) to create quizzes, add questions, retrieve quiz details, and allow students to take the quiz and submit their answers. All data will be stored in-memory using Java collections such as HashMap or ArrayList. The system will handle various functionalities like quiz creation, question management, student submissions, and score calculation. Users should be able to perform the following actions using HTTP methods (GET, POST, PUT, DELETE):

1. Create a Quiz: Teachers can create a quiz with a unique quiz ID, quiz title, and a set of questions.
2. Add Questions to Quiz: Teachers can add multiple-choice questions (question text, choices, correct answer) to a quiz.
3. View Quiz Details: Teachers can view the details of a quiz, including the questions and correct answers.
4. Delete Quiz: Teachers can delete a quiz by its ID, removing all associated questions.
5. Retrieve Available Quizzes: Students can retrieve a list of available quizzes to take.
6. Submit Answers: Students can submit answers for a specific quiz. The system should calculate and return the score immediately.
7. Error Handling: Implement robust error handling to respond with appropriate status codes (404 Not Found, 400 Bad Request) for invalid quiz IDs, malformed requests, etc.

The web application must provide RESTful web services for the above services. The RESTful web services must use the JAX-RS/Jersey Annotations and must accept the input data as path parameter format.

a) Develop a web application in NetBeans that implements the RESTful web services that support the above-mentioned services. Submit one zip file consisting of the implemented NetBeans project on the Moodle page.

b) Test the functionality of the implemented RESTful web services by providing screenshots of the web browser page or POSTMAN window output. The screenshots must show the URL for the resource call and the response produced by the REST service.

Rubric for Evaluation:

CATEGORY	MARKS	CRITERIA
FUNCTIONALITY	**30**	**- 15 Points:** Full implementation of teacher functionality (quiz creation, adding questions, deleting **quizzes).** **- 15 Points:** Full implementation of student functionality (taking quizzes, submitting answers, scoring).
RESTFUL DESIGN	**20**	**- 10 Points:** Correct usage of HTTP methods (GET, POST, DELETE) according to REST principles. - 10 Points: Proper use of URL structures and path

		variables (e.g., /quizzes/{quizId}).
CODE QUALITY	**20**	**- 10 Points:** Code follows best practices, with proper structuring, naming conventions, and clear separation of concerns (e.g., services, controllers). **- 10 Points:** Efficient use of Java collections for in-memory storage.
ERROR HANDLING	**10**	**- 5 Points:** Proper HTTP status codes and meaningful error messages for bad requests or invalid quiz IDs. - 5 Points: Graceful handling of edge cases (e.g., submitting answers for a non-existent quiz).
SECURITY	**10**	**- 5 Points:** Correct implementation of API key authentication for teachers. **- 5 Points:** Handling of unauthorized access attempts with 403 Forbidden status codes.
DOCUMENTATION	**10**	**- 5 Points:** Clear and well-documented API usage (input and output formats) and endpoints. **- 5 Points:** Explanation of design decisions and justification for error handling and security choices.

STUDENTS whose IDs end with digits 5-9 will complete the following project.

Consider a Task Management RESTful API. You will design and implement a RESTful web service in Java that allows users to manage tasks and projects without using a database. Tasks and projects will be stored in-memory using Java collections such as lists and maps. The API should allow users to create, retrieve, update, and delete tasks and projects, as well as assign tasks to specific projects. Additionally, users should be able to filter tasks based on their status (e.g., pending, completed). **Users should be able to perform the following actions using HTTP methods (GET, POST, PUT, DELETE)**

1. **Create Project**: Allows the creation of a project with details like project ID, name, and description.
2. **Retrieve Projects**: Retrieve a list of all projects or a specific project by its ID.
3. **Update Project**: Update the name or description of an existing project.
4. **Delete Project**: Delete a project by its ID. Deleting a project should also remove all associated tasks.
5. **Create Task:** Allows users to create a task with details like task ID, name, description, and status (pending/completed), and assign it to a project.
6. **Retrieve Tasks:** Retrieve a list of all tasks or a specific task by its ID. Users should be able to filter tasks by project or status.
7. **Update Task:** Update the name, description, or status of an existing task.
8. **Delete Task:** Remove a task by its ID.
9. Implement basic API key authentication. Clients must include a valid API key in the headers to access the endpoints.
10. Respond with a 403 Forbidden if an invalid key is provided.

a) Develop a web application in NetBeans that implements the RESTful web services that support the above-mentioned services. Submit one zip file consisting of the implemented NetBeans project on the Moodle page.

b) Test the functionality of the implemented RESTful web services by providing screenshots of the web browser page or POSTMAN window output. The screenshots must show the URL for the resource call and the response produced by the REST service.

Rubric for Evaluation:

Category	*Marks*	*Criteria*
Functionality	**30**	- **10 Points**: All project management endpoints are fully functional. - **10 Points**: All task management endpoints are fully functional. - **10 Points**: Task filtering and association with projects

		work as expected.
RESTful Design	**20**	- **10 Points**: Proper use of HTTP methods (GET, POST, PUT, DELETE) according to REST principles. - **10 Points**: Appropriate use of URLs, query parameters, and path variables (e.g., /projects/{id} and /tasks?status=pending).
Code Quality	**20**	- **10 Points**: Code follows clean coding standards, with well-organized structure, proper naming conventions, and reusable methods. - **10 Points**: Adheres to Java best practices and efficient use of collections.
Error Handling	**10**	- **5 Points**: Meaningful error messages and correct status codes (e.g., 400 Bad Request, 404 Not Found) for invalid requests. - **5 Points**: Proper handling of edge cases (e.g., deleting non-existent tasks).
Security	**10**	- **5 Points**: API key authentication is correctly implemented and secured. - **5 Points**: Proper handling of unauthorized access (e.g., 403 Forbidden for missing or invalid API keys).
Documentation	**10**	- **5 Points**: Clear documentation with usage examples for each endpoint (input and output JSON examples). - **5 Points**: Explanation of design decisions, including error handling and security.

NOTES:

- No Database: Store objects in memory using Java Collections such as Lists or Maps.
- Use Plain Java: You are free to use libraries like JAX-RS (Jersey) for RESTful services, but no database integration.
- Error Handling: Implement meaningful error responses for bad requests or invalid operations (e.g., trying to delete a non-existent task).
- Security: Implement a basic API key authentication for accessing the endpoints.
- The client-side application is not required to be implemented for testing the REST web services.
- **Classes** with their attributes must be implemented as separate classes on the server side.
- For storing the objects of classes, a fixed-size data structure (e.g., array list) may be used.
- Your application must include exception handling.
- You should submit the project as a Maven project, including any necessary dependencies in the pom.xml.
- Ensure proper project structure, including separate classes for models, services, and controllers.
- Provide a README file with instructions on how to run the project and test the API endpoints using Postman