University of Westminster

School of Computer Science and Engineering

5COSC022W	**Client-Server Architectures - Coursework (2025/26)**
Module leader	Hamed Hamzeh
Unit	Coursework
Weighting:	60%
Qualifying mark	30%
Description	REST API design, development and implementation.
This assignment contributes towards the following Learning Outcomes (LOS):	
Learning Outcomes Covered in this Assignment:	LO1 Gain a thorough understanding of RESTful principles and their application in API design. LO2 Acquire familiarity with the JAX-RS framework as a tool for building RESTful APIs in Java.
Handed Out:	February 2026
Due Date	24th April 2026, 13:00
Expected deliverables	A public GitHub repo link contains all parts of the project
Method of Submission:	Video demonstration Report included in README.md file in Github Electronic submission on Blackboard via a provided link close to the submission time.
Type of Feedback and Due Date:	Written feedback within 15 working days.
BCS CRITERIA MEETING IN THIS ASSIGNMENT	2.1.1 Knowledge and understanding of facts, concepts, principles & theories 2.1.2 Use of such knowledge in modelling and design 2.1.3 Problem solving strategies 2.2.1 Specify, design or construct computer-based systems 2.2.4 Deploy tools effectively 2.3.2 Development of general transferable skills 3.1.1 Deploy systems to meet business goals 4.1.1 Knowledge and understanding of scientific and engineering principles 4.1.3 Knowledge and understanding of computational modelling

Assessment regulations

Refer to section 4 of the "How you study" guide for undergraduate students for a clarification of how you are assessed, penalties and late submissions, what constitutes plagiarism etc.

Penalty for Late Submission

If you submit your coursework late but within 24 hours or one working day of the specified deadline, 10 marks will be deducted from the final mark, as a penalty for late submission, except for work which obtains a mark in the range 40-49%, in which case the mark will be capped at the pass mark (40%). If you submit your coursework more than 24 hours or more than one working day after the specified deadline you will be given a mark of zero for the work in question unless a claim of Mitigating Circumstances has been submitted and accepted as valid.
It is recognised that on occasion, illness or a personal crisis can mean that you fail to submit a piece of work on time. In such cases you must inform the Campus Office in writing on a mitigating circumstances form, giving the reason for your late or non-submission. You must provide relevant documentary evidence with the form. This information will be reported to the relevant Assessment Board that will decide whether the mark of zero shall stand. For more detailed information regarding University Assessment Regulations, please refer to the following website:
http://www.westminster.ac.uk/study/current-students/resources/academic-regulations

Learning Goals:

- **Define** key principles of REST architecture.
- **Differentiate** between RESTful and non-RESTful APIs.
- **Recognize** the importance of resource-based interactions.
- **Understand** the role of JAX-RS in Java-based API development.
- **Explore** JAX-RS annotations for resource mapping and HTTP method handling.
- **Implement** basic resource classes using JAX-RS.

Coursework: Developing a JAX-RS RESTful Service (Total 100 Marks)

Module: Client-Server Architectures
Title: "Smart Campus" Sensor & Room Management API
Weight: 60% of final grade

1. Introduction & Scenario

Scenario: You have been appointed as the Lead Backend Architect for the university's "Smart Campus" initiative. What began as a pilot project for tracking individual temperature sensors has evolved into a comprehensive campus-wide infrastructure project. The university now requires a robust, scalable, and highly available RESTful API to manage

thousands of Rooms and the diverse array of Sensors (e.g., CO2 monitors, occupancy trackers, smart lighting controllers) contained within them.
This system will be built as a high-performance web service using JAX-RS (Jakarta RESTful Web Services). Your goal is to provide a seamless interface for campus facilities managers and automated building systems to interact with the campus data.
Objective: This coursework is designed to simulate a real-world development task. It assesses your proficiency in RESTful architectural patterns, the implementation of resource nesting, and the design of a resilient error-handling strategy using JAX-RS. You are expected to demonstrate industry-standard practices, including the use of appropriate HTTP status codes, meaningful JSON responses, and a logical resource hierarchy that reflects the physical structure of the campus.

Core "Resource" Data Models:

You will model three primary entities that represent the core pillars of the Smart Campus. Use these POJOs as your foundation, ensuring they are properly encapsulated with getters and setters:

```
        public class Room {
            private String id;              // Unique identifier, e.g., "LIB-301"
            private String name;            // Human-readable name, e.g., "Library
                Quiet Study"
            private int capacity;           // Maximum occupancy for safety
                regulations
            private List<String> sensorIds = new ArrayList<>(); // Collection of
                 IDs of sensors deployed in this room

            // Constructors, getters, setters...
        }

        public class Sensor {
            private String id;              // Unique identifier, e.g., "TEMP-001"
            private String type;            // Category, e.g., "Temperature", "
                Occupancy", "CO2"
            private String status;          // Current state: "ACTIVE", "
                MAINTENANCE", or "OFFLINE"
            private double currentValue; // The most recent measurement recorded
            private String roomId;          // Foreign key linking to the Room
                where the sensor is located.

            // Constructors, getters, setters...
        }

        public class SensorReading {
            private String id;              // Unique reading event ID (UUID
                recommended)
            private long timestamp;         // Epoch time (ms) when the reading was
                 captured
            private double value;           // The actual metric value recorded by
                the hardware

            // Constructors, getters, setters...
        }
```

Smart Campus POJO Models

2. Coursework Tasks (Step-by-Step)

Part 1: Service Architecture & Setup (10 Marks)

1. Project & Application Configuration (5 Marks):

- Bootstrap a Maven project integrating a JAX-RS implementation (e.g., Jersey) and a lightweight servlet container or embedded server.
- Implement a subclass of `javax.ws.rs.core.Application` and use the `@ApplicationPath("/api/v1")` annotation to establish your API's versioned entry point.
- **Question:** In your report, explain the default lifecycle of a JAX-RS Resource class. Is a new instance instantiated for every incoming request, or does the runtime treat it as a singleton? Elaborate on how this architectural decision impacts the way you manage and synchronize your in-memory data structures (maps/lists) to prevent data loss or race conditions.

2. The "Discovery" Endpoint (5 Marks):

- Implement a root "Discovery" endpoint at `GET /api/v1`. This should return a JSON object providing essential API metadata: versioning info, administrative contact details, and a map of primary resource collections (e.g., "rooms" -> "/api/v1/rooms").
- **Question:** Why is the provision of "Hypermedia" (links and navigation within responses) considered a hallmark of advanced RESTful design (HATEOAS)? How does this approach benefit client developers compared to static documentation?

Part 2: Room Management (20 Marks)

1. Room Resource Implementation (10 Marks):

- Develop a `SensorRoom` Resource class to manage the `/api/v1/rooms` path.
- `GET /`: Provide a comprehensive list of all rooms.
- `POST /`: Enable the creation of new rooms. Ensure the service returns appropriate feedback upon success.
- `GET /{roomId}`: Allow users to fetch detailed metadata for a specific room.
- **Question:** When returning a list of rooms, what are the implications of returning only IDs versus returning the full room objects? Consider network bandwidth and client side processing.

2. Room Deletion & Safety Logic (10 Marks):

- Implement `DELETE /{roomId}` to allow room decommissioning.
- **Business Logic Constraint:** To prevent data orphans, a room cannot be deleted if it still has active sensors assigned to it. If a deletion is attempted on a room with sensors, your service must block the request and return a custom error response (as detailed in Part 5).
- **Question:** Is the `DELETE` operation idempotent in your implementation? Provide a detailed justification by describing what happens if a client mistakenly sends the exact same `DELETE` request for a room multiple times.

Part 3: Sensor Operations & Linking (20 Marks)

1. Sensor Resource & Integrity (10 Marks):

- Implement `SensorResource` to manage the `/api/v1/sensors` collection.
- `POST /`: When a new sensor is registered, your logic must verify that the `roomId` specified in the request body actually exists in the system.
- **Question:** We explicitly use the `@Consumes (MediaType.APPLICATION_JSON)` annotation on the `POST` method. Explain the technical consequences if a client attempts to send data in a different format, such as `text/plain` or `application/xml`. How does JAX-RS handle this mismatch?

2. Filtered Retrieval & Search (10 Marks):

- Enhance `GET /api/v1/sensors` to support an optional query parameter named `type` (e.g., `GET /api/v1/sensors?type=CO2`). If provided, the response must filter the list to only include matching sensors.
- **Question:** You implemented this filtering using `@QueryParam`. Contrast this with an alternative design where the `type` is part of the URL path (e.g., `/api/v1/sensors/type/CO2`). Why is the query parameter approach generally considered superior for filtering and searching collections?

Part 4: Deep Nesting with Sub - Resources (20 Marks)

A key requirement is to maintain a historical log of readings for every sensor on campus.

1. The Sub-Resource Locator Pattern (10 Marks):

- In your `SensorResource` class, implement a sub-resource locator method for the path `{sensorId}/read` This method should return an instance of a dedicated `SensorReadingResource` class.
- **Question:** Discuss the architectural benefits of the Sub-Resource Locator pattern. How does delegating logic to separate classes help manage complexity in large APIs compared to defining every nested path (e.g., `sensors/{id}/readings/{rid}`) in one massive controller class?

2. Historical Data Management (10 Marks):

- Within `SensorReadingResource`, implement `GET /` to fetch history and `POST /` to append new readings for that specific sensor context.
- **Side Effect:** A successful `POST` to a reading must trigger an update to the `currentValue` field on the corresponding parent `Sensor` object to ensure data consistency across the API.

Part 5: Advanced Error Handling, Exception Mapping & Logging (30 Marks)

You must ensure the API is "leak-proof" it should never return a raw Java stack trace or a default server error page. Furthermore, the API requires observability through proper request and response logging. Implement specific Exception Mappers and Filters for these scenarios.

1. Resource Conflict (409) (5 Marks):

- **Scenario:** Attempting to delete a Room that still has Sensors assigned to it.

- **Task:** Create a custom `RoomNotEmptyException`. Implement an Exception Mapper that returns an HTTP 409 Conflict with a JSON body explaining that the room is currently occupied by active hardware.

2. Dependency Validation (422 Unprocessable Entity) (10 Marks):

- **Scenario:** A client attempts to POST a new Sensor with a `roomId` that does not exist.

- **Task:** Create a `LinkedResourceNotFoundException`. Use an Exception Mapper to return an HTTP 422 Unprocessable Entity (or a 400 Bad Request).

- **Question:** Why is HTTP 422 often considered more semantically accurate than a standard 404 when the issue is a missing reference inside a valid JSON payload?

3. State Constraint (403 Forbidden) (5 Marks):

- **Scenario:** A sensor currently marked with the status "MAINTENANCE" is physically disconnected and cannot accept new readings.

- **Task:** Create a `SensorUnavailableException`. Map this to an HTTP 403 Forbidden status when a `POST` reading is attempted.

4. The Global Safety Net (500) (5 Marks):

- **Task:** Implement a "catch-all" `ExceptionMapper<Throwable>`. This mapper must intercept any unexpected runtime errors (e.g., `NullPointerException`, `IndexOutOfBoundsException`) and return a generic HTTP 500 Internal Server Error.

- **Question:** From a cybersecurity standpoint, explain the risks associated with exposing internal Java stack traces to external API consumers. What specific information could an attacker gather from such a trace?

5. API Request & Response Logging Filters (5 Marks):

- **Task:** Implement API observability by creating a custom filter class that implements both `ContainerRequestFilter` and `ContainerResponseFilter`. Use `java.util.logging.Logger` to log the HTTP method and URI from the `ContainerRequestContext` for every incoming request, and log the final HTTP status code from the `ContainerResponseContext` for every outgoing response.

- **Question:** Why is it advantageous to use JAX-RS filters for cross-cutting concerns like logging, rather than manually inserting `Logger.info()` statements inside every single resource method?

3. Submission & Marking Breakdown

Professional Practice (Mandatory Prerequisites)

GitHub Hosting: The project must be hosted in a public GitHub repository. Your repository must contain a `README.md` that provides:

- An overview of your API design.
- Explicit, step-by-step instructions on how to build the project and launch the server.
- At least five sample curl commands demonstrating successful interactions with different parts of the API.

Video Demonstration: A maximum of 10-minute video demonstration of postman tests. The video must be uploaded directly to the BlackBoard submission link. There is no need to show the coding part. Please note that the video demonstration is mandatory for the assessment. In the video demonstration, you MUST be present and speak clearly.
Report: The report must be prepared in a PDF format, which must only include the answers to the questions in each part. There is no need to include test cases and an introduction.

Evaluation Criteria

- Part 1 (Setup & Discovery): 10 Marks
- Part 2 (Room Management): 20 Marks
- Part 3 (Sensors & Filtering): 20 Marks
- Part 4 (Sub - Resources): 20 Marks
- Part 5 (Error Handling & Logging): 30 Marks
- **Conceptual Report:** This document must only contain the written answers to the questions found in each section of your report. Keep in mind that your report's overall mark is influenced by your performance on each task, with 20% of the total score allocated to answering specific questions. For instance, in task 3.1, you can earn 2 mark out of 10 by including a proper response to the question posed. Be sure to address these questions thoroughly to maximise your scores! Please note that the report must be organised and written in the `README.md` file on GitHub
- **Video Demonstration:** Please keep in mind that each task of the coursework requires a recorded video demonstration to showcase your work. This video demonstration is significant, as it accounts for 30% of the total score for each task. For instance, in Task 3.1, the video demonstration will contribute 3 marks out of the total 10 marks available for that task. Make sure to put effort into your video presentations, as they are an important part of your overall assessment. Please note that you should clearly present during the video presentation. Therefore, it is critical to ensure that your camera and microphone are working properly. Failing to submit a recorded video presentation will result in a 20% loss of marks on each task that requires video recording. Please note that the Video Demonstration link/File must be included in BlackBoard submission link.
- **Coding:** Please note that the allocated mark for coding for each section is 50%, which means that if a section has 10 mark, 5 marks is allocated to coding, 3 mark is allocated to video demonstration and 2 mark is allocated to answering questions in the report.

- **Technology Stack:** You must only use JAX-RS. Using other technologies like Spring boot are not allowed, which means that you will receive immediate ZERO mark for the whole coursework.
- **NO ZIP FILE**: PLEASE NOTE THAT, SUBMITTING ANY ZIP FILE INSTEAD OF HOSTING THE APP IN THE GITHUB WILL RESULT IN ZERO MARK FOR THE WHOLE COURSEWORK.
- **Database:** You are not allowed to use any database technology like SQL Server. Otherwise, you will receive ZERO mark for the coursework. You must only use data structures like Hashmap or ArrayList.