Microservices Architecture #1 – Development and Demonstration of a REST API – Weighting 20%

Development and Demonstration of a REST API

The purpose of this assignment is to design, develop, and demonstrate a robust, well-structured **RESTful API**. The API should demonstrate sound architectural decisions, correct use of HTTP semantics, clean data modelling, and effective separation of concerns. The focus is on clarity, correctness and maintainability.

Core Technical Requirements

1. Entity Relationships (Required)

The API must demonstrate a **one-to-many or a many-to-many** relationship between entities.

Example:

- **Parent Entity**: Customer.
- **Child Entities**: Orders.

The relationship must be

- Reflected in the database schema
- Navigable through API endpoints

Required operations include:

- Retrieving all child entities for a given parent
- Creating, updating and deleting child entities associated with a parent.
- Implementing a cascading delete if appropriate for the domain model

2. Use of Date and Time Objects

The API must correctly handle date-related data.

Requirements:

- Accept and validate date inputs
- Use a consistent, documented format (e.g. YYY-MM-DD)
- Provide at least one endpoint that:

    - Filters by date range, or
    - Sorts results based on date fields

3. **Incorporating DTOs (Data Transfer Objects)**
   Use **Data Transfer Objects (DTOs)** to control the data exchanged between the client and the server. This ensures:

    - Separation of concerns by decoupling the internal data model from the API response format.
    - Clean and minimal responses that expose only the required data fields.
    - Examples:
    - Instead of exposing the full `Customer` entity in the API, use a `CustomerDTO` that includes only essential fields such as `name`, `email`, and `totalOrders`.
    - For the `Order` entity, use an `OrderDTO` that includes fields such as `orderId`, `orderDate`, and `amount`.

4. **Error Handling and Validation**
   The API must implement structured and meaningful error handling.
   Required: API consumers.

   - Validating client input
   - Appropriate HTTP status codes and error messages (e.g., 400 Bad Request, 404 Not Found).
   - Clear, consistent error responses

   Examples:

   - Requesting a non-existent resource, the API should return a 404 error with a descriptive message.
   - Invalid request body, return a 400 error with validation feedback.

5. Pagination

The API must support pagination for endpoints returning collections.

Requirements:

- Pagination parameters must be documented.
- Responses must clearly indicate:
  - Page size
  - Current page or offset
  - Total elements (where applicable)

Deliverables

1. Report

The report must include:

- Introduction and overview of the API
- Explanation of:
  - Entity relationships
  - Date handling
  - DTO usage
  - Pagination strategy
  - Error handling approach
- Database design (ERD or equivalent)
- Sample API responses
- Challenges encountered and solutions
- Link to the Github/Bitbucket repo (Mandatory), accessible to lecturer.

2. Code

- A zipped archive of the complete codebase
- Code should be readable, structured, and runnable
- Clear separation of layers (controller, service, persistence where applicable)

3. Screencast (Camera On)

0:00 – 0:45 | Opening Context (≈45 seconds)

State clearly and briefly:

- What the API does
- The problem domain (e.g. Customers and Orders)
- The core design goals

0:45 – 2:00 | High-Level Architecture (≈75 seconds)

Show:

- Project structure
- Main layers (controller, service, repository, DTOs)

Explain:

- Why the layers exist
- How data flows from request → controller → service → persistence → response

Avoid:

- Opening every file
- Line-by-line explanations

2:00 – 3:30 | Data Model & Relationships (≈90 seconds)

Show:

- Database schema or ERD
- One-to-many (or many-to-many) relationship

Explain:

- How entities relate
- How this relationship is enforced (foreign keys, mappings, joins)
- How the API exposes this relationship via endpoints

Example:

- Fetching all orders for a specific customer
- Creating a child entity linked to a parent

3:30 – 5:00 | DTOs and Data Flow (≈90 seconds)

Show:

- At least one DTO
- Mapping between entity and DTO

Explain:

- Why DTOs are used
- What data is intentionally *not* exposed
- Difference between internal model and API response

5:00 – 6:30 | API Demonstration: Core Endpoints (≈90 seconds)

Using Postman (or similar), demonstrate:

- GET (collection + single resource)
- POST (creation)
- PUT or DELETE (update or removal)

Explain:

- Request structure
- Response structure
- HTTP status codes returned

6:30 – 7:45 | Pagination & Date Handling (≈75 seconds)

Demonstrate:

- Paginated endpoint
- Date-based filtering or sorting

Explain:

- Pagination parameters
- Why pagination is necessary
- How date validation or formatting is handled

7:45 – 9:00 | Error Handling & Validation (≈75 seconds)

Intentionally trigger errors:

- Request a non-existent resource (404)
- Send invalid input (400)

Explain:

- How validation is enforced
- How errors are structured
- Why meaningful errors matter for API consumers

9:00 – 10:00 | Wrap-Up & Reflection (≈60 seconds)

Summarise:

- What was demonstrated
- Key design decisions
- One challenge faced and how it was resolved

Total: **≤10 minutes**

4. **Assessment Rubric**

Overall Weighting

- **Screencast:** 60%
- **Report:** 40%

Screencast Rubric (60%)

1. Technical Clarity & Structure (15%)

Excellent (70+)
The screencast follows a clear, logical flow. Each section transitions naturally, and the viewer can easily understand how the system is structured and why decisions were made.

Good (55–69)
Overall structure is clear, with minor pacing or organisational issues. Most explanations are coherent and easy to follow.

Satisfactory (40–54)
The flow is uneven or occasionally confusing. Explanations exist but lack clarity or logical progression.

Fail (0–39)
The screencast is disorganised or difficult to follow. Purpose and structure are unclear.

2. API Demonstration & Functionality (45%)

Excellent (70+)
All required features are demonstrated clearly: entity relationships, CRUD operations, pagination, date handling, and error cases. Endpoints behave as expected. Error scenarios are intentionally demonstrated and clearly explained. Appropriate HTTP status codes and structured responses are used consistently.

Good (55–69)
Most required features are demonstrated, with minor omissions or limited explanation. Error handling is present and demonstrated, but explanations lack detail or consistency.

Satisfactory (40–54)
Basic functionality is shown, but key features are missing, incomplete, or poorly explained. Minimal error handling is shown, with limited explanation.

Fail (0–39)
API is incomplete, non-functional, or not meaningfully demonstrated. Error handling is absent or incorrect.

Screencast Total: 60%

Report Rubric (40%)

1. Structure & Communication (10%)

Excellent (70+)
The report is well-organised, clearly written, and professional. All required sections are present and logically ordered.

Good (55–69)
The report is mostly well-structured, with minor issues in clarity or organisation.

Satisfactory (40–54)
The report includes required sections but lacks coherence or clarity in places.

Fail (0–39)
The report is poorly structured, incomplete, or difficult to understand.

2. Technical Explanation & Design Rationale (15%)

Excellent (70+)
Provides clear, accurate explanations of design decisions, including entity relationships, DTOs, pagination, date handling, and error management. Demonstrates strong technical insight.

Good (55–69)

Explains most design decisions adequately, though some lack depth or justification.

Satisfactory (40–54)

Descriptions are mostly descriptive rather than analytical, with limited rationale.

Fail (0–39)

Little or no meaningful technical explanation is provided.

3. Database Design & Data Modelling (10%)

Excellent (70+)

Includes a clear and well-explained database design (e.g. ERD). Relationships are correctly modelled and justified.

Good (55–69)

Database design is present and mostly correct, with minor issues or limited explanation.

Satisfactory (40–54)

Database design is basic or poorly explained.

Fail (0–39)

Database design is missing, incorrect, or irrelevant.

4. Referencing & Academic Integrity (5%)

Excellent (70+)

All sources are correctly cited using Harvard referencing, with consistent in-text citations and a complete reference list.

Good (55–69)

Most sources are cited correctly, with minor formatting errors.

Satisfactory (40–54)

Some sources are cited, but referencing is inconsistent or incomplete.

Fail (0–39)

Sources are not cited or referencing conventions are not followed.

Report Total: 40%