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 fully functional **REST API**. The API should incorporate best practices and showcase proficiency in various aspects of modern API development, including relationships between entities, data transfer mechanisms, error handling, and compliance with HATEOAS principles. The focus is on implementing a scalable, well-documented, and robust API that can handle complex real-world use cases.

The key objectives are as follows:

1. **Demonstrating a One-to-Many Relationship**

- The API should showcase a one-to-many or a many-to-many relationship between two entities.

  Example:

  Parent Entity: A Customer.

  Child Entities: Multiple Orders placed by the customer.

  The relationship must be reflected in the database schema, and the API endpoints should allow for operations that demonstrate this relationship:

  - Retrieving all orders for a specific customer.
  - Creating, updating, and deleting orders linked to a customer.
  - Implementing a cascading delete if needed (e.g., deleting a customer also deletes their associated orders).

2. **Using Date Objects**

   The API should make use of **Date objects** to handle operations related to dates. This involves:

   - Accepting and validating date inputs in the API (e.g., date of order placement or customer registration date).
   - Formatting dates consistently (e.g., ISO 8601 standard: YYYY-MM-DD).

- Providing endpoints that may filter or sort data based on dates, such as:
- Retrieving orders within a specific date range.
- Sorting customers or orders by creation date.

3. **Incorporating DTOs (Data Transfer Objects)**

   Use **Data Transfer Objects (DTOs)** to structure 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**

   Implement robust **error handling** mechanisms to provide meaningful feedback to API consumers. This includes:

   - Validating input data and returning appropriate HTTP status codes and error messages (e.g., 400 Bad Request, 404 Not Found).
   - Catching unhandled exceptions and returning structured error responses.
   - Example:
   - If a user tries to fetch a non-existent order, the API should return a 404 error with a message like "Order not found."
   - If a required field is missing in a request body, return a 400 error with details about the missing field.

5. **HATEOAS (Hypermedia as the Engine of Application State)**

   The API should be compliant with **HATEOAS principles**, enhancing discoverability for clients by including hypermedia links in API responses. For example:

   - When retrieving a customer, include links for:
     - Fetching their orders (`GET /customers/{id}/orders`).
     - Updating their details `(PUT /customers/{id}`).
     - Deleting the customer (`DELETE /customers/{id})`.
   - When retrieving orders, include links for:
     - Accessing the associated customer (`GET /customers/{customerId}`).
     - Updating the order (`PUT /orders/{orderId}`).

6. **PAGINATION**

   Pagination helps in managing large datasets efficiently by returning data in chunks which reduces server load and provides a better UI experience. For example, instead of fetching all orders, the API could return a subset of them (e.g. 10 orders per call)

d. Deadlines

Submission Date
Monday 24th February 2024 (Report, Code and Screencast)

e. Submission

	What to submit
Report	**Title Page** • Assignment Title: Development and Demonstration of a REST API • Name, Student ID, Course Title, Date of Submission
	Introduction • Overview of the task • Objectives of the REST API project

	Design and Implementation
	
	• Description of the REST API:
	  ○ **One-to-Many Relationships**: Explanation with an example (e.g., Orders and Products).
	  ○ **Using Date Objects**: Handling date inputs and outputs in the API.
	  ○ **DTO (Data Transfer Objects)**: Purpose and how they simplify API responses.
	• Database design:
	  ○ Tables and relationships (include diagrams like ERD).
	• Error Handling:
	  ○ How errors are managed (e.g., validation errors, exceptions).
	• HATEOAS (Hypermedia as the Engine of Application State):
	  ○ Example of implementing HATEOAS in API responses.

	Code Explanation
	
	• Explanation of key sections of the code:
	  ○ Setting up controllers and routes.
	  ○ Service and repository layers (if applicable).
	• Example of a one-to-many relationship implementation in code.
	• Sample API responses.

	Challenges and Solutions • Issues faced during development. • How those were resolved.
	Conclusion

<table>
<tr><td></td><td>

- Summary of the project and learnings.

</td></tr>
<tr><td></td><td>

References

- All sources of information, ideas, and data used in the assignment must be appropriately cited using the **Harvard referencing style**. This includes in-text citations and a full bibliography or reference list at the end of the report. Failure to adhere to this requirement will result in a deduction of marks or non-compliance with academic integrity policies.

</td></tr>
<tr><td>Code</td><td>A .zip file of all code</td></tr>
<tr><td>Screen cast</td><td>

The screencast should cover:

1. **Introduction** (1-2 minutes)
    - Briefly introduce the project and its purpose.
2. **API Walkthrough** (5-7 minutes)
    - Show the code:
        - Controllers, DTOs, and database relationships.
        - Highlight HATEOAS implementation and error handling.
    - Explain a one-to-many relationship with an example.
3. **API Demonstration** (3-5 minutes)
    - Use Postman or a similar tool:
        - Demonstrate CRUD operations.
        - Show API responses, including HATEOAS links.
        - Handle error scenarios.
4. **Closing** (1 minute)
    - Summarise the project and highlight key features.

</td></tr>
</table>

f. Marking Rubric

Elements	Excellent (70+)	Good (55%-69%)	Satisfactory (40%-55%)	Fail (0-39%)
Presentation (10%)	The screencast is clear, engaging, and well-paced. The presenter speaks confidently and audibly throughout. High-quality audio and visuals; materials (e.g., slides, demos) are well-organised and integrated seamlessly.	The screencast is clear and audible, with some minor pacing or engagement issues. Audio and visuals are clear and appropriately used, with minor distractions.	The screencast is mostly clear, but engagement or pacing is inconsistent. Audio and visuals are present but may lack clarity or organisation.	Screencast lacks clarity or engagement, with inaudible or rushed delivery. Poor or missing visuals/audio, or materials are unorganised..
Context & Rationale (10%)	Poor or missing visuals/audio, or materials are unorganised. Rationale is directly relevant to the application's functionality and objectives. .	Clear explanation of the context and rationale, with some originality. Rationale is mostly relevant but lacks depth or detail in some areas.	Basic explanation of the context, but the rationale is limited or generic. Limited relevance to the application's functionality or objectives.	No clear explanation of context or rationale. Rationale lacks relevance or is missing entirely.
Demo (Application Completeness and Technical Understanding) (50%)	Demonstrates a fully functional API with all required features (1-to-many, pagination, HATEOS, DTOs, validation, error handling, Date).	Demonstrates a mostly complete API with most features. Displays good understanding, explaining most implementation choices adequately.	Demonstrates basic functionality, but lacks advanced features like HATEOAS or validation. Limited explanation of implementation choices or	Minimal or no API demonstrated. Many incomplete or missing features. Little or no understanding of the tech stack or implementation choices.

	Shows a strong understanding of the tech stack, explaining implementation choices and limitations clearly REST API follows design principles (ONAP). Comprehensive testing. Robust error handling implemented and explained, including server-side validation and appropriate HTTP responses.	Demonstrates sufficient testing, covering key use cases with minor gaps. Error handling is implemented with some explanation and minor gaps.	understanding of the tech stack. Demonstrates basic testing but misses important scenarios or validations. Basic error handling is implemented but lacks depth or explanation.	Minimal or no testing demonstrated. Error handling is missing or poorly implemented.

Screencast (70%)

Elements	Excellent (70+)	Good (55%-69%)	Satisfactory (40%-55%)	Fail (0-39%)
Organisation and Presentation 5%	Report is well-structured, with all required sections (intro, design, implementation, testing, etc.) logically organised. Writing is clear, concise, and professional, with no grammatical or typographical errors.	Report is organised and includes most required sections, with minor structural issues. Writing is mostly clear and professional, with a few minor errors..	Report includes required sections but lacks clarity or logical flow. Writing is understandable but contains several errors or inconsistencies	Report is incomplete or poorly structured, missing key sections. Writing is unclear or unprofessional, with frequent errors.
Technical Content, 20%	Demonstrates in-depth technical knowledge, clearly explaining the architecture, design decisions, and implementation details. Includes clear and well-annotated diagrams (e.g., ERD, workflows) and code examples that enhance understanding.	Demonstrates solid technical knowledge, explaining most design decisions and implementation adequately. Includes relevant diagrams and code examples, but they may lack annotations or clarity	Demonstrates basic technical knowledge, but explanations of design or implementation are lacking depth. Diagrams and code examples are present but poorly explained or lack relevance	Poor Demonstrates little or no technical understanding, with unclear or missing explanations. Diagrams and code examples are missing or irrelevant.
Referencing 5%	All sources are accurately cited using Harvard referencing style, with in-text citations and a complete bibliography.	Most sources are cited using Harvard style, with minor formatting errors.	Some sources are cited, but Harvard style is inconsistently applied	Sources are not cited, or referencing style is not followed.

Report and Code (30%)

Live Q&A : If I have questions or queries on your submission, a Live Demo/Q&A session may be required.

CheckList:

- Upload to moodle by Monday 24th February. 2025 17.00. .zip with your code, screencast and report.
- Live Demo/Q&A via zoom : You will be notified via student email if you need to participate in a Live Q&A.