Introduction

This coursework is designed to provide you with practical experience in designing and implementing RESTful web services using the Java API for RESTful Web Services (JAX-RS). You will build a comprehensive "Bookstore" application API, gaining hands-on experience with core RESTful principles, JAX-RS features, and common API design patterns.

This project simulates a real-world scenario where you need to create a robust and scalable backend API to manage various entities like books, authors, customers, shopping carts, and orders. You'll learn how to structure your application using resource classes, handle different HTTP methods, process requests and responses, and implement proper error handling.

Coursework Objectives

Upon successful completion of this coursework, you will be able to:

- Understand and apply the fundamental principles of RESTful architecture.
- Design and implement RESTful APIs using the JAX-RS specification.
- Structure a JAX-RS application using resource classes and sub-resources.
- Handle various HTTP methods (GET, POST, PUT, DELETE) appropriately.
- Work with different data formats, specifically JSON, for request and response bodies.
- Implement data validation and exception handling using ExceptionMapper.
- Understand and implement basic API testing using Postman.
- Model basic e-commerce functionalities such as shopping cart and order management.

Coursework Specifications

1. Application Domain:

You will be building a RESTful API for a "Bookstore" application. This API will allow clients to interact with the following entities:

- **Books:** Products with attributes like title, author, ISBN, publication year, price, and stock quantity.
- **Authors:** Creators of books, with attributes like name and biography.
- **Customers:** Users of the bookstore, with attributes like name, email, and a simple password for this coursework.
- **Carts:** Shopping carts associated with customers, allowing them to add, remove, and update the quantity of books.
- **Orders:** Represent completed purchases, created from a customer's cart.

2. Technology Stack:

- **JAX-RS:** You **must** use the JAX-RS API for implementing your RESTful services. You can choose a specific JAX-RS implementation like Jersey or RESTEasy.
- **JSON:** You **must** use JSON as the data format for all request and response bodies.
- **Postman:** You **must** use Postman for testing and demonstrating your API.
- **In-Memory Data Storage:** You **must** use simple in-memory data structures (e.g., ArrayList, HashMap) to store your application data. **Do not use any external databases or persistence frameworks.**

3. Resource Classes and Endpoints:

Your API must include the following resource classes and endpoints:

- **BookResource (/books)**
  - POST /books
  - GET /books
  - GET /books/{id}
  - PUT /books/{id}
  - DELETE /books/{id}
- **AuthorResource (/authors)**
  - POST /authors
  - GET /authors
  - GET /authors/{id}
  - PUT /authors/{id}
  - DELETE /authors/{id}
  - GET /authors/{id}/books
- **CustomerResource (/customers)**
  - POST /customers
  - GET /customers
  - GET /customers/{id}
  - PUT /customers/{id}
  - DELETE /customers/{id}
- **CartResource (/customers/{customerId}/cart)**
  - POST /customers/{customerId}/cart/items
  - GET /customers/{customerId}/cart
  - PUT /customers/{customerId}/cart/items/{bookId}
  - DELETE /customers/{customerId}/cart/items/{bookId}
- **OrderResource (/customers/{customerId}/orders)**
  - POST /customers/{customerId}/orders

- GET /customers/{customerId}/orders
- GET /customers/{customerId}/orders/{orderId}

4. Data Models:

You need to create corresponding Java classes to represent the Book, Author, Customer, cardItem and order entities. These classes should have appropriate attributes, data types, and constructors.

5. Exception Handling:

You must implement custom exception classes and use ExceptionMapper to handle the following scenarios gracefully:

- BookNotFoundException
- AuthorNotFoundException
- CustomerNotFoundException
- InvalidInputException
- OutOfStockException
- CartNotFoundException

Map these exceptions to appropriate HTTP status codes (e.g., 404 Not Found, 400 Bad Request) and provide informative error messages in JSON format.

6. Code Structure:

- Organize your code into the specified resource classes.
- Follow best practices for JAX-RS development.
- Use meaningful names for variables, methods, and classes.

7. Report Structure:

This report requires you to document the test cases you have designed and executed for your RESTful Bookstore API. The report will consist primarily of tables that clearly outline each endpoint, its expected behavior, and the results of your tests.

Your report should have the following structure:

1. Introduction (Brief - Approximately 1/4 page)

- Briefly introduce the purpose of the report, which is to document the test cases for your Bookstore API.
- Mention the technologies used to build the API (JAX-RS, JSON) and the tool used for testing (Postman).
- State that the report focuses on presenting test cases in a tabular format.

2. API Endpoint Test Case Tables (Main Content of the Report)

- For **each endpoint** in your API, create a separate table to document its test cases.
- Organize the tables by resource class (e.g., a section for BookResource test cases, a section for AuthorResource test cases, etc.).
- Use the following table structure for each endpoint:

Sample Test Case Table: POST /books Endpoint:

Endpoint	Description	HTTP Method	Request Body (JSON)	Expected HTTP Status Code	Expected Response Body (JSON)	Actual Result (Pass/Fail)
POST /books	Create a book with valid data.	POST	{ "title": "The Lord of the Rings", "authorId": 1, "isbn": "978-0-618-05326-7", "publicationYear": 1954, "price": 20.99, "stock": 100 }	201 Created	{ "id": <generated_id>, "title": "The Lord of the Rings", "authorId": 1, "isbn": "978-0-618-05326-7", "publicationYear": 1954, "price": 20.99, "stock": 100 }	
POST /books	Create a book with invalid author ID (not found).	POST	{ "title": "The Lord of the Rings", "authorId": 999, "isbn": "978-0-618-05326-7", "publicationYear": 1954, "price": 20.99, "stock": 100 }	404 Not Found	{ "error": "Author Not Found", "message": "Author with ID 999 does not exist." }	
POST /books	Create a book with invalid publication year (future).	POST	{ "title": "The Lord of the Rings", "authorId": 1, "isbn": "978-0-618-05326-7", "publicationYear": 2025, "price": 20.99, "stock": 100 }	400 Bad Request	{ "error": "Invalid Input", "message": "Publication year cannot be in the future." }	
GET /books	Get all books.	GET		200 OK	[{ "id": 1, "title": "Book 1", ... }, { "id": 2, "title": "Book 2", ... }, ...]	
GET /books/{id}	Get book by valid ID.	GET		200 OK	{ "id": 1, "title": "Book 1", ... }	
GET /books/{id}	Get book by invalid ID.	GET		404 Not Found	{ "error": "Book Not Found", "message": "Book with ID 999 does not exist." }	

Explanation of Columns:

- **Endpoint: The specific API endpoint being tested (e.g., POST /books, GET /books/{id}).**
- **Description:** A brief description of what the test case is doing.
- **HTTP Method:** The HTTP method used (e.g., POST, GET, PUT, DELETE).
- **Request Body (JSON):** The JSON payload sent in the request body (if applicable).
- **Expected HTTP Status Code:** The HTTP status code you expect the API to return.
- **Expected Response Body (JSON):** The expected JSON payload in the response body.
- **Actual Result (Pass/Fail):** Indicate whether the test case passed or failed during testing.

Important Reminders:

- **Comprehensive Test Cases:** Ensure that your test cases cover a wide range of scenarios for each endpoint, including valid inputs, invalid inputs, edge cases, and error conditions.
- **Postman for Execution:** You must use Postman to execute your test cases and record the results in the tables.
- **Focus on Documentation:** The primary goal of this report is to clearly document your test cases and their results.

Example Test Case Tables for Other Endpoints:

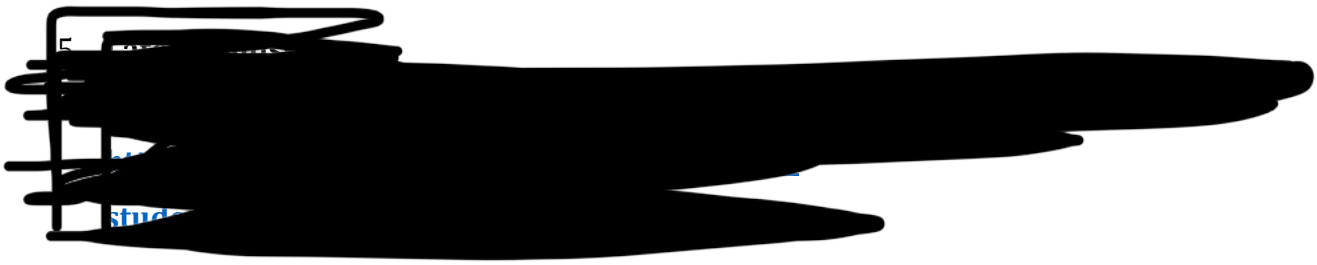
You would create similar tables for other endpoints, such as:

- **GET /books**
  - Get all books (should return 200 OK and a list of books)
- **GET /books/{id}**
  - Get book by valid ID (should return 200 OK and the book)
  - Get book by invalid ID (should return 404 Not Found)
- **PUT /books/{id}**
  - Update book with valid data (should return 200 OK and the updated book)
  - Update book with invalid data (should return 400 Bad Request)
  - Update book with non-existent ID (should return 404 Not Found)
- **DELETE /books/{id}**
  - Delete a book by valid ID (should return 204 No Content).
  - Delete a book by invalid ID (should return 404 Not Found).

- **And so on for all other endpoints in AuthorResource, CustomerResource, CartResource, and OrderResource...**

Coursework Policies and Regulations

1. **Technology Restrictions:**
    - This coursework **strictly prohibits** the use of any technologies other than **JAX-RS, JSON, and Postman** (and basic Java for data models and in-memory storage).
    - **No external databases, persistence frameworks (e.g., Hibernate), Spring, dependency injection frameworks, or external libraries are allowed.**
    - **Using any prohibited technology like Spring or Flask or any Database technology like SQL will result in a mark of zero (0) for the entire coursework.**
    - **You must only follow the concepts we covered during the tutorials. If you use any other irrelevant piece of code that is not related to the scenario or the required task, you will lose your mark for related parts/classes.**
2. **Code formatting and comments**
    - **You must ensure that your codes are well-formatted and supported by detailed comments, especially for complex coding parts. Otherwise, your mark for any related class will be reduced.**



Video Demonstration Details

A significant portion of your [redacted] will be based on a video demonstration of your API using Postman. Here's what you need to do:

1. **Postman Collection:** Create a Postman collection that includes requests for **all** the endpoints of your API.
2. **Test Cases:**
    - Include a variety of test cases for each endpoint, covering both **positive** (successful) and **negative** (error) scenarios.
    - For example, for the POST /books endpoint, you should demonstrate creating a book with valid data, as well as attempting to create a book with missing fields, incorrect data types, or a non-existent author ID.
    - Demonstrate the scenarios where your ExceptionMapper should be triggered and verify that the correct HTTP status codes and error messages are returned.
3. **Recording:**
    - Record a video of yourself using Postman to send requests to your API and show the responses. The length of the video demonstration should not exceed 10 minutes.
    - Your video should be clear, well-paced, and easy to follow.
    - **Explain each request** as you send it, describing the purpose of the request, the expected outcome, and the actual outcome.
    - **Highlight the use of different HTTP methods** (GET, POST, PUT, DELETE) and **HTTP status codes** in your demonstration.
4. **Duration:** The video should be no longer than 15 minutes.