UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Informatics Institute of Technology

Department of Computing

BSc (Hons) Computer Science

Client-Server Architectures

5COSC022C.2

Student Name: Lakmindee Siyathna Jayamanne

UoW ID: w2053264

IIT ID: 20222100

Table of Contents

1. Introduction

This report presents a clear overview of the testing process for the Bookstore RESTful API, developed using the JAX-RS framework. The API models the core operations of a typical e-commerce bookstore by managing books, authors, customers, carts, and orders through dedicated resource classes.

Following REST principles, the API supports standard HTTP methods (GET, POST, PUT, DELETE) and communicates exclusively through JSON. Backend data is handled in memory using structures like HashMap and Array List, providing fast, temporary storage without external databases.

Functional and negative test cases were executed in Postman to verify the correctness, resilience, and error handling of the API. These tests cover successful transactions as well as invalid input scenarios, missing resources, and stock-related restrictions.

Exception handling is implemented using custom exceptions and ExceptionMapper classes to ensure consistent JSON formatted error responses and appropriate HTTP status codes such as 400, 404, and 409.

The following sections document the full set of test cases across all API endpoints, showing input data, expected outcomes, actual results, and validations to demonstrate full requirement coverage.

2. Test Cases Table

2.1. Book Resource class Testing

Endpoint	Description	HTTP Method	Request Body (JSON)	Expected HTTP Status Code	Expected Response Body (JSON)	Actual Result (Pass/Fail)
POST / books	Create a book (valid data)	POST	{ "title": "The Alchemist", "isbn": "1234567890", "authorId": 1, "publicationYear": 1988, "price": 15.99, "stock": 10 }	201 Created	{ "authorId": 1, "id": 1, "isbn": "1234567890", "price": 15.99, "publicationYear": 1988, "stock": 10, "title": "The Alchemist" }	Pass
POST / books	Create a book Missing title.	POST	{ "isbn": "1234567890", "authorId": 1, "publicationYear": 1988, "price": 15.99, "stock": 10 }	400 Bad Request	{ "error": "Title is required" }	Pass
POST / books	Create a book With missing ISBN.	POST	{ "title": "The Alchemist", "authorId": 1, "publicationYear": 1988, "price": 15.99, "stock": 10 }	400 Bad Request	{ "error": "ISBN is required" }	Pass
POST / books	Create a book	POST	{ "title": "The Alchemist",	400	{	Pass

	With negative author ID		"isbn": "1234567890", "authorId": -9, "publicationYear": 1988, "price": 15.99, "stock": 10 }	Bad Request	"error": "Valid author ID is required" }	
POST / books	Create a book Without the publication year	POST	{ "title": "The Alchemist", "isbn": "1234567890", "authorId": 1, "price": 15.99, "stock": 10 }	400 Bad Request	{ "error": "Valid publication year is required" }	Pass
GET /books	Get all books (without creating any book)	GET		200 OK	[]	Pass
GET /books	Get all books (with creating a book)	GET		200 OK	{ "authorId": 1, "id": 1, "isbn": "1234567890", "price": 15.99, "publicationYear": 1988, "stock": 10, "title": "The Alchemist" }	Pass
GET/books/{id}	Get books by valid ID (id = 1)	GET		200 OK	{ "authorId": 1, "id": 1, "isbn": "1234567890", "price": 15.99, "publicationYear": 1988, "stock": 10,	Pass

					"title": "The Alchemist" }	

GET/books/{id}	Get books by invalid ID (id = 67)	GET		404 Not Found	{
					"error": "Book with ID67not found."
					}
						Pass

PUT/books/{id}	Update a book using the ID
	
	(id = 1)
		PUT	{ "title": "The Alchemist - Updated", "isbn": "1234567890", "authorId": 1, "publicationYear": 1988, "price": 18.99, "stock": 12 }	200 OK	{
					"authorId": 1,
					"id": 1,
					"isbn": "1234567890",
					"price": 18.99,
					"publicationYear": 1988,
					"stock": 12,
					"title": "The Alchemist - Updated"
					}
						Pass

PUT/books/{id}	Update a book using an not created ID (id = 3)	PUT	{ "title": "The Alchemist - Updated", "isbn": "1234567890", "authorId":3 , "publicationYear": 1988, "price": 18.99, "stock": 12 }	404 Not Found	{
					"error": "Book with ID 3 not found"
					}
						Pass

PUT/books/{id}	Update a book using an invalid ID (id = 0)	PUT	{ "title": "The Alchemist - Updated", "isbn": "1234567890", "authorId":3 , "publicationYear": 1988, "price": 18.99, "stock": 12 }	400 Bad Request	{
					"error": "Book ID must be positive"
					}
						Pass

PUT/books/{id}	Update a book with missing title
	
	(id = 1)
		PUT	{ "isbn": "1234567890", "authorId":3 , "publicationYear" : 1988, "price": 18.99, "stock": 12 }	400 Bad Request	{
					"error": "Title is required"
					}
						Pass

DELETE/books/{id}	Deleting a book with id
	
	(id = 1)
		DELETE		200 OK	{
					"status": "success",
					"message": "Book with ID 1 was deleted"
					}
						Pass

DELETE/books/{id}	Deleting a book with an ID that hasn't been created yet
	
	(id = 9)
		DELETE		404 Not Found	{
					"error": "Book with ID 9 not found."
					}
						Pass

DELETE/books/{id}	Deleting a book with negative ID
	(id = 0)
		DELETE		400 Bad Request	{
					"error": "Book ID must be positive"
					}
						Pass

2.2. Author Resource class Testing

Endpoint	Description	HTTP Method	Request Body (JSON)	Expected HTTP Status Code	Expected Response Body (JSON)	Actual Result (Pass/Fail)
POST / authors	Create an author (valid data)	POST	{ "firstName": "Paulo", "lastName": "Coelho", "biography": "Brazilian lyricist and novelist." }	201 Created	{ "biography": "Brazilian lyricist and novelist.", "firstName": "Paulo", "id": 1, "lastName": "Coelho" }	Pass
POST / authors	Create an author Missing first name.	POST	{ "lastName": "Coelho", "biography": "Brazilian lyricist and novelist." }	400 Bad Request	{ "error": "First name is required" }	Pass
POST / authors	Create an author With missing last name.	POST	{ "firstName": "Paulo", "biography": "Brazilian lyricist and novelist." }	400 Bad Request	{ "error": "Last name is required" }	Pass
GET /authors	Get all authors (without creating any authors)	GET		200 OK	[]	Pass
GET /authors	Get all authors with creating	GET		200 OK	[{ "biography": "Brazilian	Pass

	an author				lyricist and novelist.", "firstName" : "Paulo", "id": 1, "lastName": "Coelho" }, { "firstName" : "Paulo", "id": 2, "lastName": "Coelho" }]	
GET/authors/{id}	Get author by valid ID (id = 1)	GET		200 OK	{ "biography": "Brazilian lyricist and novelist.", "firstName": "Paulo", "id": 1, "lastName": "Coelho" }	Pass
GET/authors/{id}	Get author by ID which haven't been created (id = 9)	GET		404 Not Found	{ "message": "Author with ID 9 not found.", "error": "Author Not Found" }	Pass
PUT/authors/{id}	Update a author using the ID (id = 1)	PUT	{ "firstName": "Paulo", "lastName": "Coelho Updated", "biography": "Updated biography." }	200 OK	{ "biography": "Updated biography.", "firstName": "Paulo", "id": 1,	Pass

					"lastName": "Coelho Updated" }	
PUT/authors/{id}	Update an author using a not created ID (id = 5)	PUT	{ "firstName": "Paulo", "lastName": "Coelho Updated", "biography": "Updated biography." }	404 Not Found	{ "message": "Author with ID 5 not found.", "error": "Author Not Found" }	Pass
PUT/authors/{id}	Update an author using an invalid ID (id = 0)	PUT	{ "firstName": "Paulo", "lastName": "Coelho Updated", "biography": "Updated biography." }	400 Bad Request	{ "error": "ID must be positive" }	Pass
PUT/authors/{id}	Update an author with missing first name	PUT	{ "lastName": "Coelho Updated", "biography": "Updated biography." }	400 Bad Request	{ "error": "First name is required" }	Pass
DELETE/authors/{id}	Deleting an author with id (id = 1)	DELETE		200 OK	{ "status": "success", "message": "Author with ID 1 was deleted" }	Pass
DELETE/authors/{id}	Deleting an author with an	DELETE		404 Not Found	{ "message": "Author with ID 9 not found.",	Pass

	ID that hasn't been created yet (id = 9)				"error": "Author Not Found" }	

DELETE/authors/{id}	Deleting an author with negative ID
	(id = 0)
		DELETE		400 Bad Request	{
					"error": "Author ID must be positive"
					}
						Pass

GET /authors/{id}/books	Books by author (valid)
	(id = 2)
		GET		200 OK	{
					"authorId": 2,
					"id": 4,
					"isbn": "1234567890",
					"price": 15.99,
					"publicationYear": 1988,
					"stock": 10,
					"title": "The Alchemist"
					}
						Pass

GET /authors/{id}/books	Books by author (invalid)
	(id = 98)
		GET		404 Not Found	{
					"message": "Author with ID 98 not found.",
					"error": "Author Not Found"
					}
						Pass

2.3. Customer Resource class Testing

Endpoint	Description	HTTP Method	Request Body (JSON)	Expected HTTP Status Code	Expected Response Body (JSON)	Actual Result (Pass/ Fail)
POST / customers	Create a customer (valid data)	POST	{ "firstName": "John", "lastName": "Doe", "email": "john@example.com", "password": "secret123" }	201 Created	{ "email": "john@example.com", "firstName": "John", "id": 1, "lastName": "Doe", "password": "secret123" }	Pass
POST / customers	Using already used email .	POST	{ "firstName": "John", "email": "john@example.com", "password": "secret123" }	400 Bad Request	{ "error": "Email already registered" }	Pass
POST / customers	Create a customer Missing first name.	POST	{ "lastName": "Doe", "email": "john@example.com", "password": "secret123" }	400 Bad Request	{ "error": "First name is required" }	Pass
POST / customers	Create a customer With missing last name.	POST	{ "firstName": "John", "email": "john@example.com", "password": "secret123" }	400 Bad Request	{ "error": "Last name is required" }	Pass
GET /customers	Get all customers (without creating	GET		200 OK	[]	Pass

	any customers)					
GET customers	Get all customers (with creating an customer	GET		200 OK	{ "email": "john@example .com", "firstName ": "John", "id": 1, "lastName" : "Doe", "password" : "secret123" },	Pass
GET/customers/ {id}	Get customer by valid ID (id = 1)	GET		200 OK	{ "email": "john@example .com", "firstName": "John", "id": 1, "lastName": "Doe", "password": "secret123" }	Pass
GET/customers/ {id}	Get customer by ID which haven't been created (id = 9)	GET		404 Not Found	{ "error": "Customer with ID 9 not found." }	Pass
PUT/customers/ {id}	Update a customer using the ID (id = 1)	PUT	{ "firstName": "Johnny", "lastName": "Doe", "email": "johnny@exam ple.com", "password": "newpass456" }	200 OK	{ "email": "johnny@exam ple.com", "firstName": "Johnny", "id": 1, "lastName": "Doe",	Pass

					"password": "newpass456" }	
PUT/customers/{id}	Update a customer using an not created ID (id = 5)	PUT	{ "firstName": "Johnny", "lastName": "Doe", "email": "johnny@example.com", "password": "newpass456" }	404 Not Found	{ "error": "Customer with ID5not found." }	Pass
PUT/customers/{id}	Update a customer using an invalid ID (id = 0)	PUT	{ "firstName": "Johnny", "lastName": "Doe", "email": "johnny@example.com", "password": "newpass456" }	400 Bad Request	{ "error": "Customer ID must be positive" }	Pass
PUT/customers/{id}	Update a customer with missing first name	PUT	{ "lastName": "Doe", "email": "johnny@example.com", "password": "newpass456" }	400 Bad Request	{ "error": "First name is required" }	Pass
DELETEcustomers/{id}	Deleting a customer with id (id = 1)	DELETE		200 OK	{ "status": "success", "message": "Customer with ID 1 was deleted" }	Pass
DELETE/customers/{id}	Deleting a customer with an	DELETE		404 Not Found	{ "error": "Customer with ID 9 not found."	Pass

	ID that hasn't been created yet (id = 9)				}	

DELETE/customers/{id}	Deleting a customer with negative ID
	(id = 0)
		DELETE		400 Bad Request	{
					"error": "Customer ID must be positive"
					}
						Pass

2.4. Cart Resource class Testing

Endpoint	Description	HTTP Method	Request Body (JSON)	Expected HTTP Status Code	Expected Response Body (JSON)	Actual Result (Pass/ Fail)
POST /customers/{id}/cart/items	Add items to the cart (valid data) (customer id = 1, Book id = 2)	POST	{ "book": { "id": 2 }, "quantity": 2 }	200 OK	{ "customerId": 1, "items": [{ "book": { "authorId": 1, "id": 2, "isbn": "1234567890", "price": 15.99, "publicationYear": 1988, "stock": 8, "title": "The Alchemist" }, "quantity": 2 }], "totalPrice": 31.98 }	Pass
POST /customers/{id}/cart/items	Add items to the cart (customer id = 1,	POST	{ "book": { "id": 7 }, "quantity": 2 }	404 Not Found	{ "error": "Book not found with ID 7" }	Pass

	Book id = 7)					
POST /customers/{id}/cart/items	When no sufficient stock is available (customer id = 9, Book id = 2)	POST	{ "book": { "id": 2 }, "quantity":7 }	409 Conflict	{ "error": "Only 6 available for The Alchemist" }	Pass
GET /customers/{id}/cart	Get cart when empty or cant fond the cart (id = 7)	GET		404 Not Found	{ "error": "Cart not found or empty for customer ID 7" }	Pass
GET /customers/{id}/cart	Get cart for customer (id = 1)	GET		200 OK	{ "customerId": 1, "items": [{ "book": { "authorId": 1, "id": 2, "isbn": "1234567890", "price": 15.99, "publicationYear": 1988, "stock": 6, "title": "The Alchemist" }, "quantity": 2	Pass

					}], "totalPrice": 31.98 }	

PUT/ /customers/{id}/cart/items/2?quantity=5	Update quantity	PUT		200 OK	{
					"customerId": 1,
					"items": [
					{
					"book": {
					"authorId": 1,
					"id": 2,
					"isbn": "1234567890",
					"price": 15.99,
					"publicationYear": 1988,
					"stock": 3,
					"title": "The Alchemist"
					},
					"quantity": 5
					}
					],
					"totalPrice": 79.95}
						Pass

PUT/ /customers/{id}/cart/items/9?quantity=5	Update quantity	PUT		404 Not Found	{
					"error": "Item not found in cart"
					}
						Pass

DELETE/customers/{id}/cart	Deleting a cart with customer id
	
	(id = 1)
		DELETE		200 OK	{
					"status": "success",
					"message": "Cart cleared successfully for customer ID 1"
					}
						Pass

DELETE/customers/{id}/cart	Deleting a cart with an ID that hasn't been created yet
	
	(id = 9)
		DELETE		404 Not Found	{
					"error": "Cart not found for customer ID 98"
					}
						Pass

DELETE/customers/{id}/cart	Deleting a cart with negative ID
	(id = 0)
		DELETE		400 Bad Request	{
					"error": "Valid customer ID required"
					}
						Pass

2.5. Order Resource class Testing

Endpoint	Description	HTTP Method	Request Body (JSON)	Expected HTTP Status Code	Expected Response Body (JSON)	Actual Result (Pass/Fail)
POST /customers/{id}/orders	Create order . when books are not enough. (id = 1)	POST		409 Conflict	{ "error": "Only 3 available for The Alchemist (requested: 7)" }	Pass
POST /customers/{id}/orders	Create order when empty.(id = 5)	POST		404 Not Found	{ "error": "Cart is empty or not found for customer ID 5" }	Pass
POST /customers/{id}/orders	Create order (id = 3)	POST		201 Created	{ "customerId": 3, "items": [{ "book": { "authorId": 2, "id": 5, "isbn": "1234567890", "price": 15.99, "publicationYear": 1988, "stock": 18, "title": "Moonlight" }, "quantity": 2 }], "orderId": 1, "status": "CREATED",	Pass

					"timestamp": "2025-04-27T02:58:38.6373129", "totalAmount": 31.98 }	
GET /customers/{id}/orders	Get cart when empty or cant find the cart (id = 1)	GET		404 Not Found	{ "error": "No orders found for customer ID 1" }	Pass
GET /customers/{id}/orders	Get cart for customer (id = 3)	GET		200 OK	[{ "customerId": 3, "items": [{ "book": { "authorId": 2, "id": 5, "isbn": "1234567890", "price": 15.99, "publicationYear": 1988, "stock": 18, "title": "Moonlight" }, "quantity": 2 }], "orderId": 1, "status": "CREATED", "timestamp": "2025-04-27T02:58:38.6373129",	Pass

					"totalAmount": 31.98 }]	

GET /customers/{id}/orders/{id}	Getting orders by the id of customer and order
	
	(order id = 5, customer id = 3)
		GET		404 Not Found	{
					"error": "Order with ID 5 not found for customer ID 3"
					}
						Pass

GET /customers/{id}/orders/{id}	Getting orders by the id of customer and order
	
	(order id = 5, customer id = 1)
		GET		200 OK	{
					"customerId": 3,
					"items": [
					{
					"book": {
					"authorId": 2,
					"id": 5,
					"isbn": "1234567890",
					"price": 15.99,
					"publicationYear": 1988,
					"stock": 18,
					"title": "Moonlight"
					},
					"quantity": 2
					}
					],
					"orderId": 1,
					"status": "CREATED",
					"timestamp": "2025-04-27T02:58:38.6373129",
						Pass

					"totalAmount": 31.98 }	

GET /customers/{id}/orders/{id}	Getting orders by the id of customer and order
	
	(order id = 6, customer id = 6)
		GET		404 Not Found	{
					"error": "No orders found for customer ID 6"
					}
						Pass

3.Conclusion

This project provided practical experience in implementing important REST architecture principles and creating a RESTful web service with JAX-RS. With full CRUD support for resources including books, authors, customers, carts, and orders, the Bookstore API effectively illustrates fundamental e-commerce functions.

The system reflects the behaviors found in contemporary online retail systems by placing a strong emphasis on stateless interactions, appropriate resource modeling, and appropriate HTTP method usage. The API's dependability and user-friendliness were further improved with exception handling, input validation, and structured error messaging.
All things considered, the testing and development process emphasized the value of strong backend service design and supported best practices in the creation of RESTful APIs.

4. Appendix

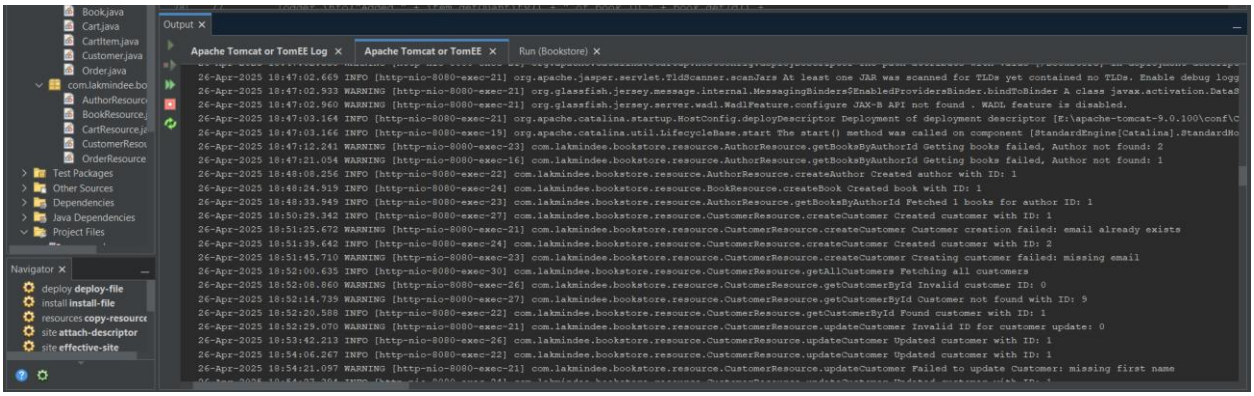

Figure 1:Example logging