CSC8104 - Enterprise Software and Services

Coursework Introduction

Today's Objectives:

- **Introduce this year’s coursework assignment: goals, structure, and assessment.**
- **Explain the technologies and workflow you’ll use: arkus, OpenShift, Git, Maven, REST Assured.**
- **Outline the three project parts (build → extend → integrate).**
- **Clarify testing, version control, and submission expectations.**
- **Prepare you to start the CSC8104 Quickstart Tutorial, which must be completed before coding.**

Coursework Overview

Goal:

Develop, test, and deploy a real-world cloud-based enterprise application using Quarkus.

01

Outcome:

A working Travel Agency REST API integrating multiple services (taxi, flight, hotel).

Each student builds one service and integrates with two classmates' services to complete the system.

Structure

Part 1 – Build the Base REST Service

Outcome:

You'll design and implement a core RESTful API for your assigned service type (Taxi, Hotel, or Flight).
This part focuses on:

- Understanding JAX-RS (for REST endpoints) and JPA (for persistence).
- Designing three key entities:
  - Customer – represents users of the system.
  - Commodity – represents the service you provide (Taxi, Flight, or Hotel).
  - Booking – links a Customer to a Commodity.
- Implementing endpoints to Create, List, and Cancel bookings.
- Returning and consuming JSON data.
- Using Bean Validation to ensure data integrity.
- Documenting endpoints via Swagger / OpenAPI.
- Writing REST Assured tests to verify your API behaviour.

Structure

Part 1 – Build the Base REST Service

Key Technical Terms Explained

Term	Meaning in Simple Words	Why It Matters Here
JAX-RS	Java API for RESTful Services, a standard way to create and handle HTTP requests (GET, POST, etc.) in Java.	You'll use it to define your API endpoints (URLs that clients can call).
JPA	Java Persistence API, connects Java objects to database tables automatically (no manual SQL).	Used to store and retrieve Customers, Bookings, and Commodities.
Bean Validation	A built-in system to check that data meets certain rules before saving or sending it (e.g. "email must not be empty").	Ensures your API only accepts valid input.
Swagger / OpenAPI	Tools that read your Java annotations and generate a live, clickable documentation page for your API.	Makes testing and integration easier, others can use your endpoints correctly.
JSON	JavaScript Object Notation, a lightweight text format for exchanging data between systems.	Your REST API will send and receive all data in JSON.
REST Assured	A testing library that sends real HTTP requests to your API and checks the responses automatically.	You'll use it to prove that your endpoints work correctly.

Structure

Part 2 – Add Advanced Features

02

Outcome:

You'll now extend your base API to include real-world enterprise logic.
This part introduces data consistency, transactions, and entity relationships.
Key additions:

- **Implement delete endpoints for Customers and Commodities.**
- **Enable cascading deletions — when a Customer or Commodity is removed, all related bookings are deleted automatically via JPA relationships (@OneToMany, @ManyToOne, etc.).**
- **Create a GuestBooking transactional endpoint that handles:**
  - **A POST request combining Customer + Booking data.**
  - **Both are persisted in a single JTA transaction.**
  - **If one operation fails → rollback everything and return an error response.**

Structure

Part 2 – Add Advanced Features

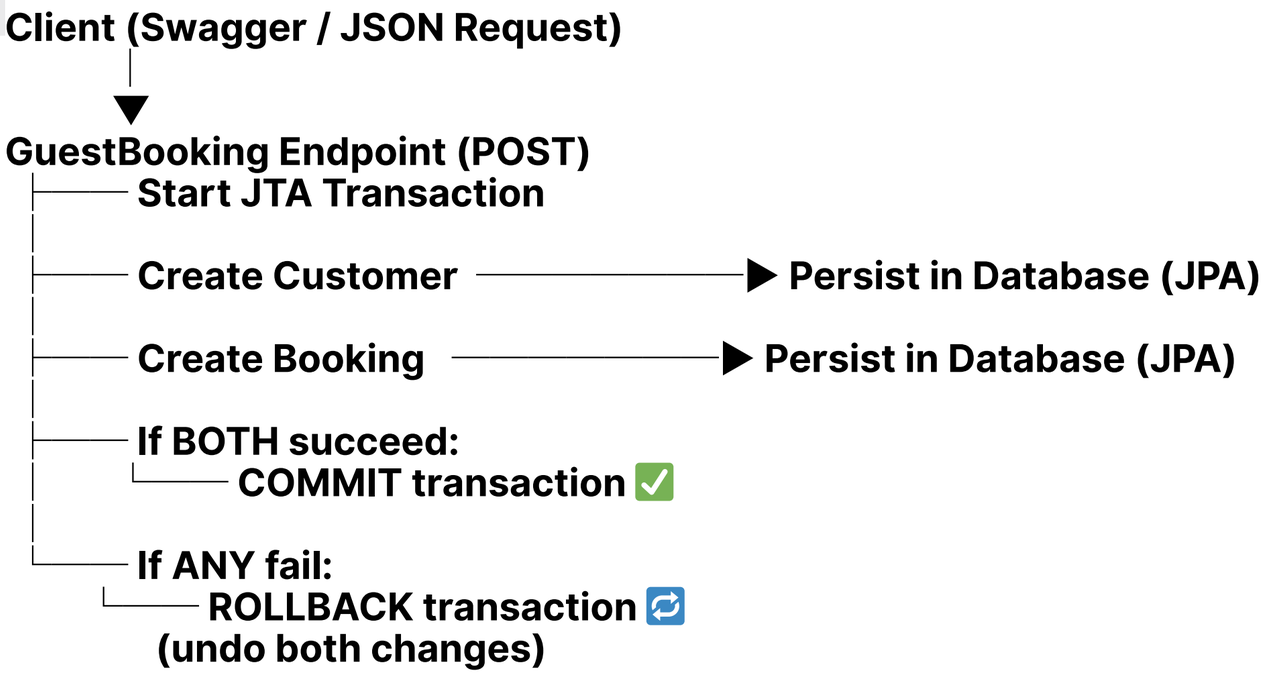
Key Technical Terms Explained

Term	Meaning in Simple Words	Why It Matters in Part 2
Data Consistency	Keeping all parts of the database accurate and in sync, so no “orphan” or mismatched records remain.	Ensures that bookings, customers, and commodities always match correctly after updates or deletions.
Entity Relationship	A link between different data objects (for example, one Customer can have many Bookings). Defined in Java using annotations like @OneToMany or @ManyToOne.	Lets your API understand how objects depend on each other, and how to cascade changes automatically.
Cascading Deletion	When one entity is deleted, any related data is also removed automatically.	If a Customer is deleted, their Bookings are also deleted, prevents leftover data.
Transaction	A group of operations that must all succeed or all fail ,“all-or-nothing”.	Guarantees that partial or broken bookings don’t exist.
JTA (Java Transaction API)	Java’s system for controlling transactions manually (start, commit, or rollback).	Used in the GuestBooking endpoint to wrap Customer + Booking creation in one safe operation.
Rollback	Reversing changes if something goes wrong partway through a transaction.	If saving the Booking fails, the Customer creation is undone too.

Structure

Part 2 – Add Advanced Features

End-to-End Flow of a GuestBooking Transaction (JTA Example)



Structure

Part 3 – Integrate and Deploy the TravelAgent Service

03

Outcome:

Finally, you'll move from a single service to a distributed system.
This part simulates real enterprise integration and service coordination.
You will:

- **Develop a new TravelAgent REST resource that integrates your service with two classmates' APIs.**
- **Implement logic to:**
  - **Create combined bookings across all three services.**
  - **Cancel any partially successful bookings if one service fails (no inconsistent states).**
  - **Retrieve and list aggregate bookings by customer.**
- **Use the Quarkus REST Client for cross-service communication.**
- **Handle and test multiple error scenarios (network failures, invalid data, unavailable services).**
- **Deploy your integrated system to OpenShift Cloud.**

Structure

Part 3 – Integrate and Deploy the TravelAgent Service

Key Technical Terms Explained

Term	Meaning in Simple Words	Why It Matters in Part 3
Distributed System	A collection of independent services (running on different machines or servers) that communicate over a network to achieve a shared goal.	You'll combine three separate APIs, Taxi, Flight, and Hotel, to act together as one TravelAgent service.
Integration	Connecting multiple systems or APIs so they can exchange data and work as one.	Your TravelAgent resource integrates your own service with your classmates' REST APIs.
REST Client	A tool within Quarkus that lets one service send HTTP requests to another (instead of using a web browser).	Enables your TravelAgent service to call the other APIs programmatically to create or cancel bookings.
Partial Booking / Inconsistent State	When some parts of a multi-step process succeed, but others fail, e.g., a flight booked successfully but the hotel fails.	You must detect this and roll back previous bookings to keep data consistent.
Error Handling	Strategies for dealing with failed API calls, invalid inputs, or network issues.	You'll need to catch and respond to errors gracefully so users get clear feedback and no data corruption occurs.
OpenShift Cloud Deployment	Hosting your distributed system on Red Hat's cloud platform, built on Kubernetes.	Makes your service accessible to others online for integration and testing.

Technologies Used

You'll gain hands-on experience with the following stack:

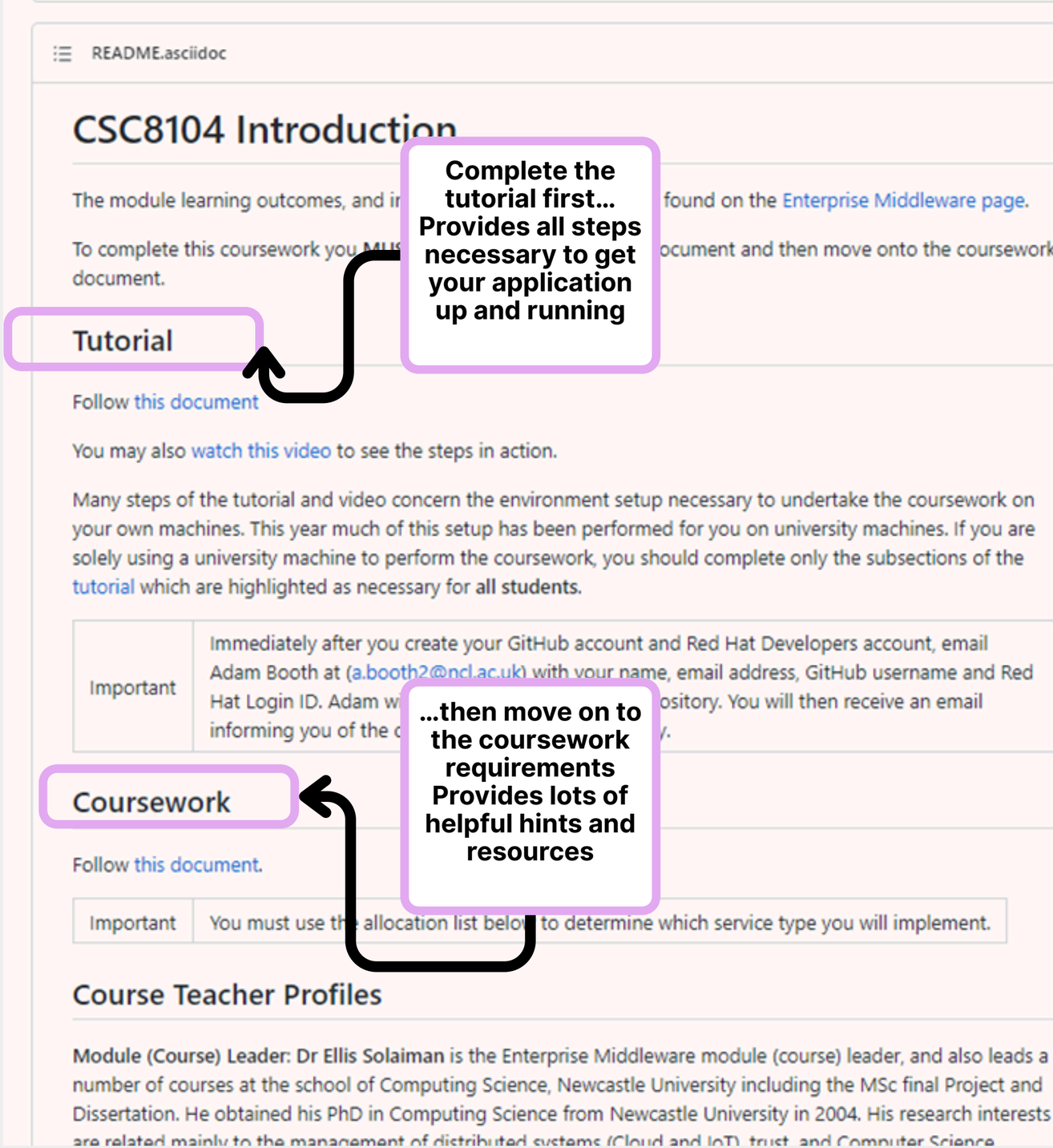
- **Quarkus – Full-stack cloud-native Java framework**
- **JAX-RS – Build RESTful services**
- **JPA & JTA – Data persistence and transaction management**
- **Bean Validation (Hibernate Validator) – Data constraints**
- **JSON & Swagger/OpenAPI – API documentation**
- **JUnit + REST Assured – Automated API testing**
- **Maven – Project build and dependency management**
- **Git & GitHub – Version control (assessed)**
- **OpenShift (PaaS) – Cloud deployment platform**

Start with the Tutorial

Start with the Tutorial

Before starting any coursework:

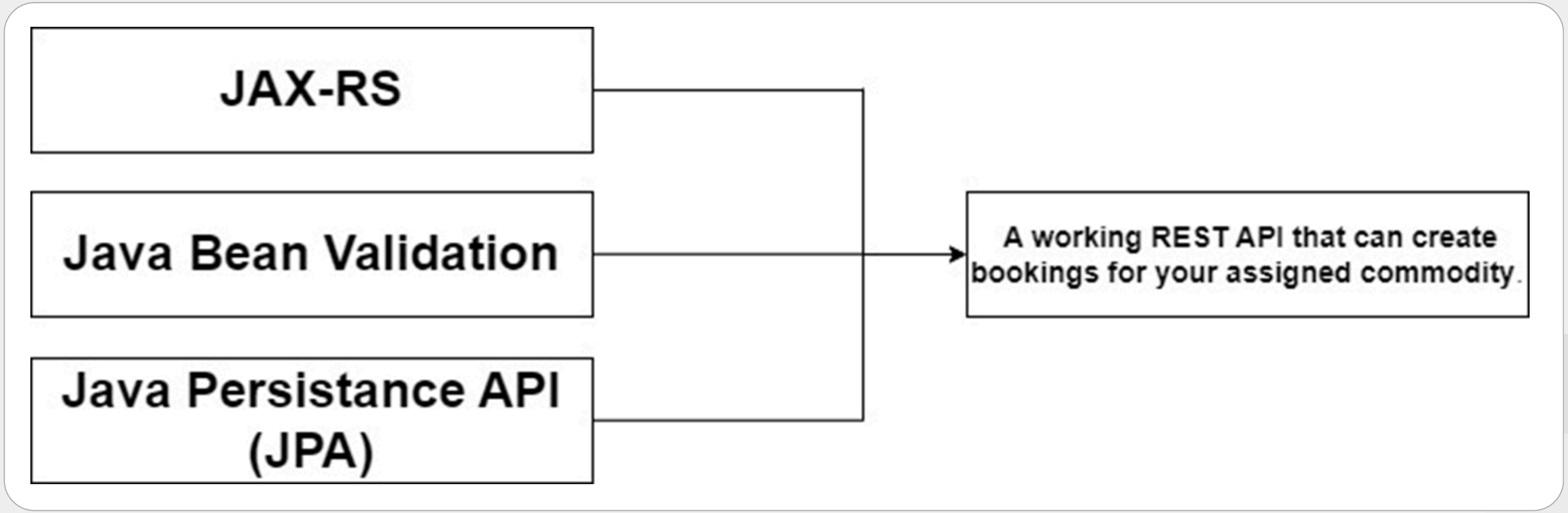
- **Complete the CSC8104 Quickstart Tutorial**
- **The tutorial sets up your project template, database, build environment, and OpenShift link.**
- **Skipping this step will cause serious build/deployment issues later.**
- **You'll also find sample REST services, tests, and Swagger annotations inside the tutorial repository.**



Cw - Part 1

In this part, you will build a basic REST API that allows customers to book one type of service: either a taxi, a flight, or a hotel.

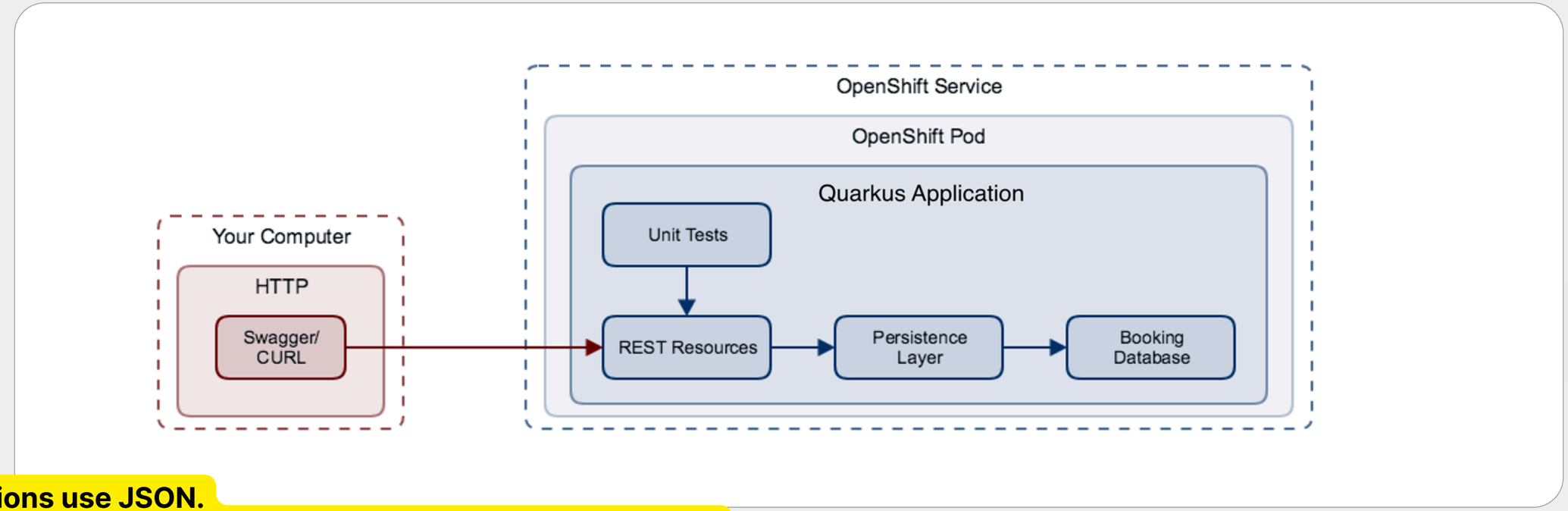
- Taxi, Flight or Hotel – you must implement the commodity you are assigned
- Each commodity requires similar implementation – one is not “better” than the other



Cw - Part 1

Requirements:

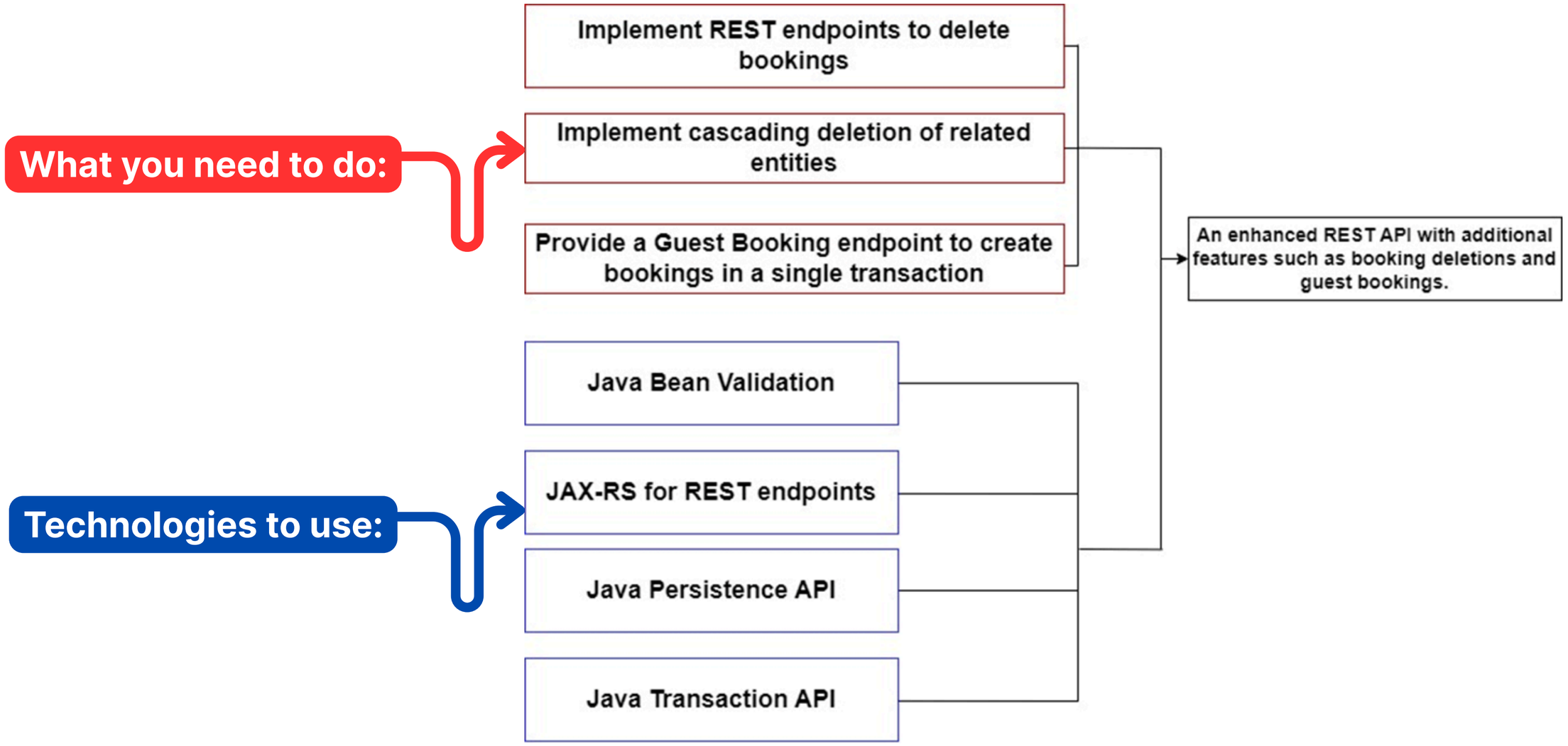
- REST endpoints for Customer, Commodity, and Booking entities.
- Implement Create, List, and Cancel operations.
- Store data using JPA entities.
- Validate data using Bean Validation annotations.
- Provide Swagger documentation for every endpoint.
- Write REST Assured tests to prove functionality.



All interactions use JSON.
You'll deploy the service to OpenShift, but develop locally first.

Cw - Part 2

Extend your Part 1 API with enterprise-level capabilities.



Cw - Part 3

Travel Agent Integration

Goal:

Combine your service with two colleagues' APIs to create a complete Travel Agency service.

In Part 3, you will combine your REST API with those of two of your classmates to create a full travel agency service that books all three commodities: flights, taxis, and hotels.
A webpage will be maintained containing URL's for each students service

NOTE

This is NOT a group project. Each student is responsible for their own REST service, but you will need to integrate your classmates' services into your own. Documentation will be provided on how to integrate services.

Completed Implementation

Completed Implementation

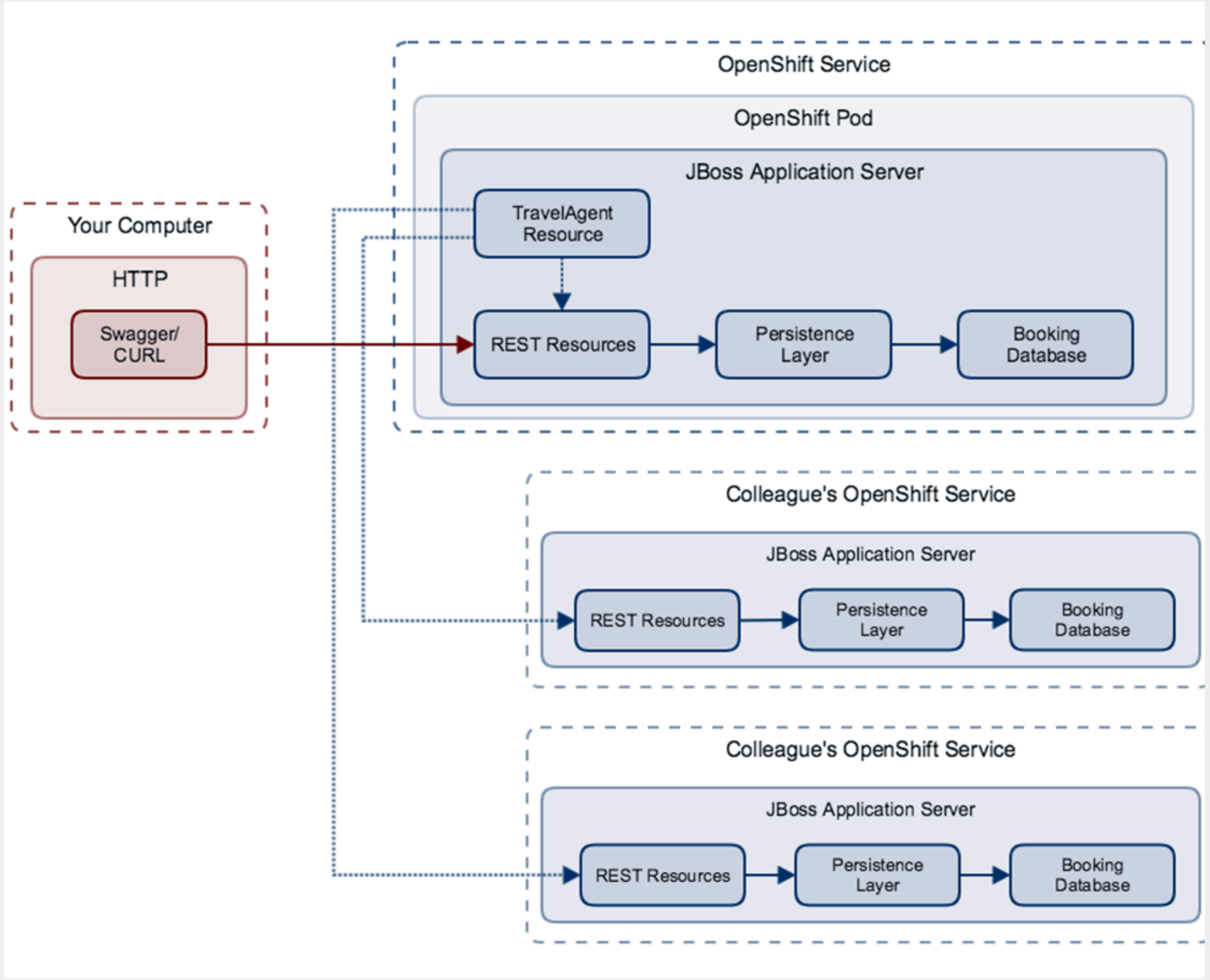
By the end of the coursework, you should have a fully functional system that:

1. Allows customers to book flights, taxis, and hotels through your Travel Agent API.
2. Can handle more advanced operations like deleting bookings and guest bookings.
3. Is deployed to the cloud using OpenShift, making it accessible from anywhere.

How it works

Before starting any coursework:

1. You send an HTTP request using Swagger or CURL from your computer.
2. This request is directed to the REST Resources on your OpenShift service, specifically the TravelAgent Resource.
3. The TravelAgent Resource communicates with your colleagues' REST services to gather or manipulate data (for booking flights, taxis, or hotels).
4. Each service has its own persistence layer and booking database, but they all work together to fulfill the requests.



Deployment on OpenShift

- Deploy final versions to OpenShift Cloud.
- Update database config:

```
quarkus.datasource.jdbc.url=jdbc:h2:mem:default;DB_CLOSE_DELAY=-1
```

- Remove try/catch from Application.java (lines 15–19) as per spec.
- Ensure your service and Swagger page are accessible for integration.
- Use Source-to-Image (S2I) builds to deploy directly from GitHub.

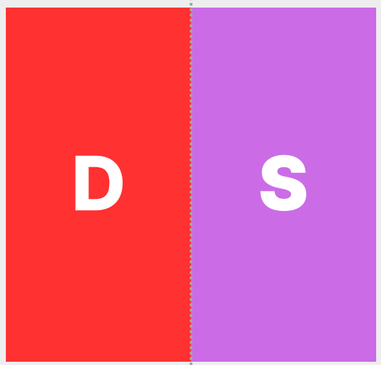
OpenShift

Once you've completed Part 3 of the coursework and have a working Travel Agent API, you should email Adam at a.booth2@newcastle.ac.uk to have the restriction removed.

Demonstration and Submission

Demonstration:

- 10–15 min live demo to a demonstrator (book via email sign-up).
- Explain your architecture, testing, and transaction management.



Submission

- Upload to NESS:
- ZIP file of full Maven project (buildable + testable).
- Short 3-page report describing:
  - Your design decisions and assumptions
  - Implementation challenges
  - Reflection on what worked / didn’t
- Missing demo → loss of marks.

Help and Support

- Practical sessions: Demonstrators available for troubleshooting.
- Discussion Boards: Canvas & Teams for questions.
- FAQ Repository:
  - https://github.com/NewcastleComputingScience/enterprise-middleware-coursework
- Peer Support: You’re encouraged to help each other and share fixes.
- Use third-party documentation actively (Swagger, Quarkus, REST Assured, JPA).

Q&A

Thank You