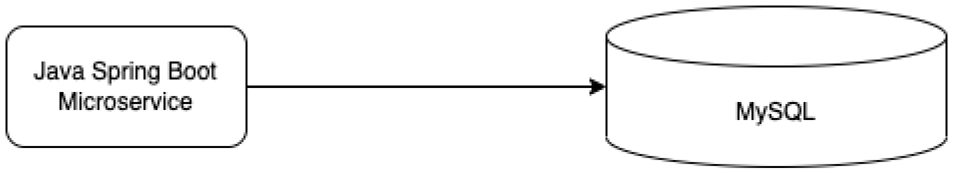
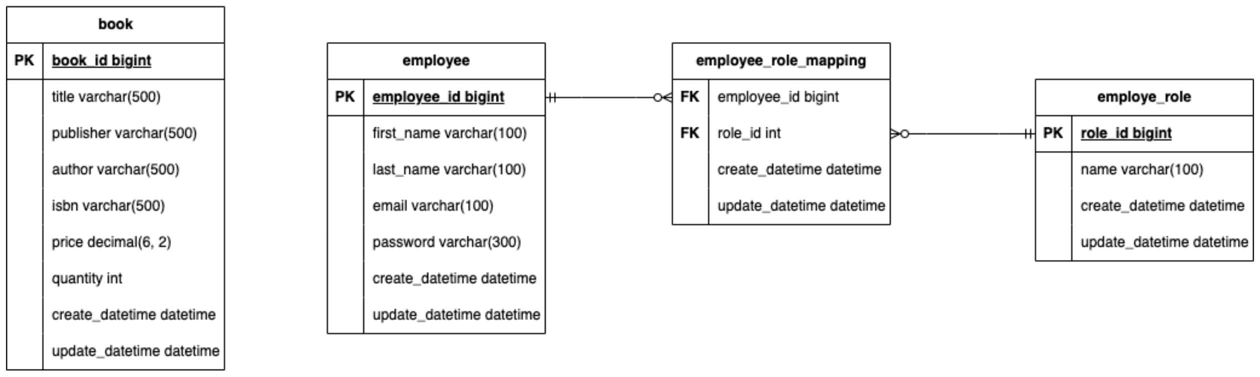
Architecture Diagram



The bookstore inventory system comprises a Java Spring Boot microservice and a MySQL database. The details of setting up the system is described in the README of the code repository.

Tech stack used:
- Java Spring 3.1.2
- MySQL 8.0.33
- Java 17

Entity Relational Model



Implemented Features

Overview

- All mandatory features in the requirement are implemented and documented in the API Endpoints section.
- Apart from the mandatory features, all features in the additional requirements are also implemented. The features are documented in the API Endpoints, Authentication, and Performance Optimizations section.

API Endpoints

Index	Feature & HTTP Endpoint Description

1	• **Feature:** Register an employee into the 'Employee' table.
	• **EndPoint**: /employees/auth/signup
	• **HTTP Method**: POST
	• **Required Payload format**: JSON
	• **Payload Description:** An object.
	{
	  "firstName": "string",
	  "lastName": "string",
	  "email": "string",
	  "password": "password"
	}
	
	• **Payload Example:**
	{
	  "firstName": "John",
	  "lastName": "Doe",
	  "email": "johndoe@gmail.com",
	  "password": "12345678"
	}

2	• **Feature:** Sign in as an employee into the book inventory system.
	• **EndPoint**: /employees/auth/signin
	• **HTTP Method**: POST
	• **Required Payload format**: JSON
	• **Payload Description:** An object.
	{
	  "email": "string",
	  "password": "string"
	}
	
	• **Payload Example:**
	{
	  "email": "johndoe@gmail.com",
	  "password": "12345678"
	}

3	• **Feature:** Add a book to the inventory. • **EndPoint**: /books • **HTTP Method**: POST • **Required Payload format**: JSON • **Payload Description:** An array of objects. The fields in each object correspond to the columns with the same name in the 'Book' table in the database. The column constraints of the 'Book' table apply to the fields in

	each object.
	
	[
	{
	"title": "string”,
	"publisher": "string",
	"author": "string",
	"isbn": "string",
	"price": 0 or a positive decimal with 2 decimal places,
	"quantity": 0 or a positive integer
	},
	{...}
	]
	
	● **Payload Example:**
	[
	{
	"bookId": 7,
	"title": "abc",
	"publisher": "somePublisher",
	"author": "John Doe 1",
	"isbn": "xyz",
	"price": 12.98,
	"quantity": 10
	}
	]

4	● **Feature:** Remove the given books from the inventory. ● **EndPoint**: /books ● **HTTP Method:** DELETE ● **Required Query Param**: id ● **Example:** /books?id=1,2,3

5	● **Feature:** Update the quantity in stock for the given books.
	● **EndPoint**: /books/quantity
	● **HTTP Method:** POST
	● **Required Payload format**: JSON
	● **Payload Description:** An object.
	{
	“book_id”: {
	“quantity”: 0 or a positive integer
	},
	…
	}
	
	● **Payload Example:**
	{

	```
	    "1": {
	      "quantity": 10
	    },
	    "2": {
	      "quantity": 20
	    }
	}
	```
6	● **Feature:** Retrieve the quantity in stock for the given books. ● **EndPoint**: /books/quantity ● **HTTP Method:** GET ● **Required Query Param**: id ● **Example:** /books?id=1,2,3
7	● **Feature:** List all books in the inventory or search for books that meet certain criteria. The criteria are specified through the query param. ● **EndPoint**: /books ● **HTTP Method:** GET ● **Optional Query Param**: title, author, isbn, isAvailable, minPrice, maxPrice, quantity ○ minPrice, maxPrice: 0 or positive decimal number ○ quantity: 0 or positive integer ● **Example:** /books?title=SomeBook&isAvailable=true&minPrice=5
8	● **Feature:** Generate a CSV report that contains all the books in the 'Book' table. ● **EndPoint**: /books/report/csv ● **HTTP Method:** GET
9	● **Feature:** Generate an XML report that contains all the books in the 'Book' table. ● **EndPoint**: /books/report/xml ● **HTTP Method:** GET

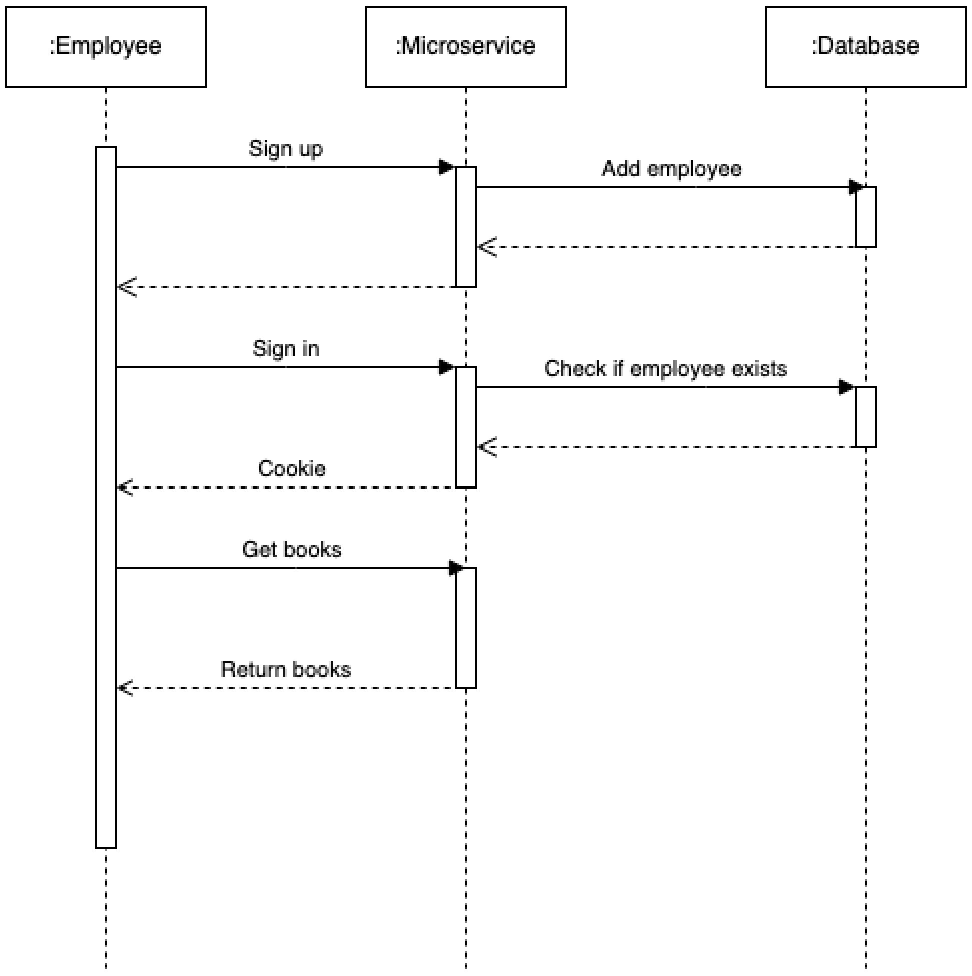
Authentication and Authorization

The authentication and authorization of the microservice is implemented using a username-and-password mechanism supported by Spring Security. It uses a session cookie based mechanism.

All endpoints except the sign up and sign in endpoint require the employee to authenticate himself/herself first. The general flow of the authentication and authorization is as follows:

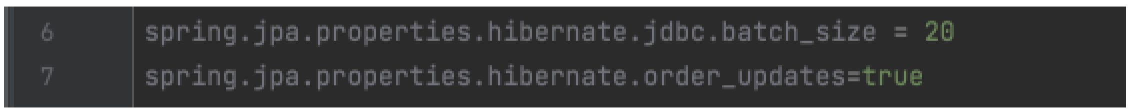
1. An employee signs up an account using the microservice.
2. The employee logs into the microservice.
3. The employee starts to use the other end points.

The flow is illustrated in the sequence diagram below:



Performance Optimization

An implemented database query optimization is enabling batch update in Hibernate. This speeds up the process of updating the quantities of multiple books.



```
spring.jpa.properties.hibernate.jdbc.batch_size = 20
spring.jpa.properties.hibernate.order_updates=true
```

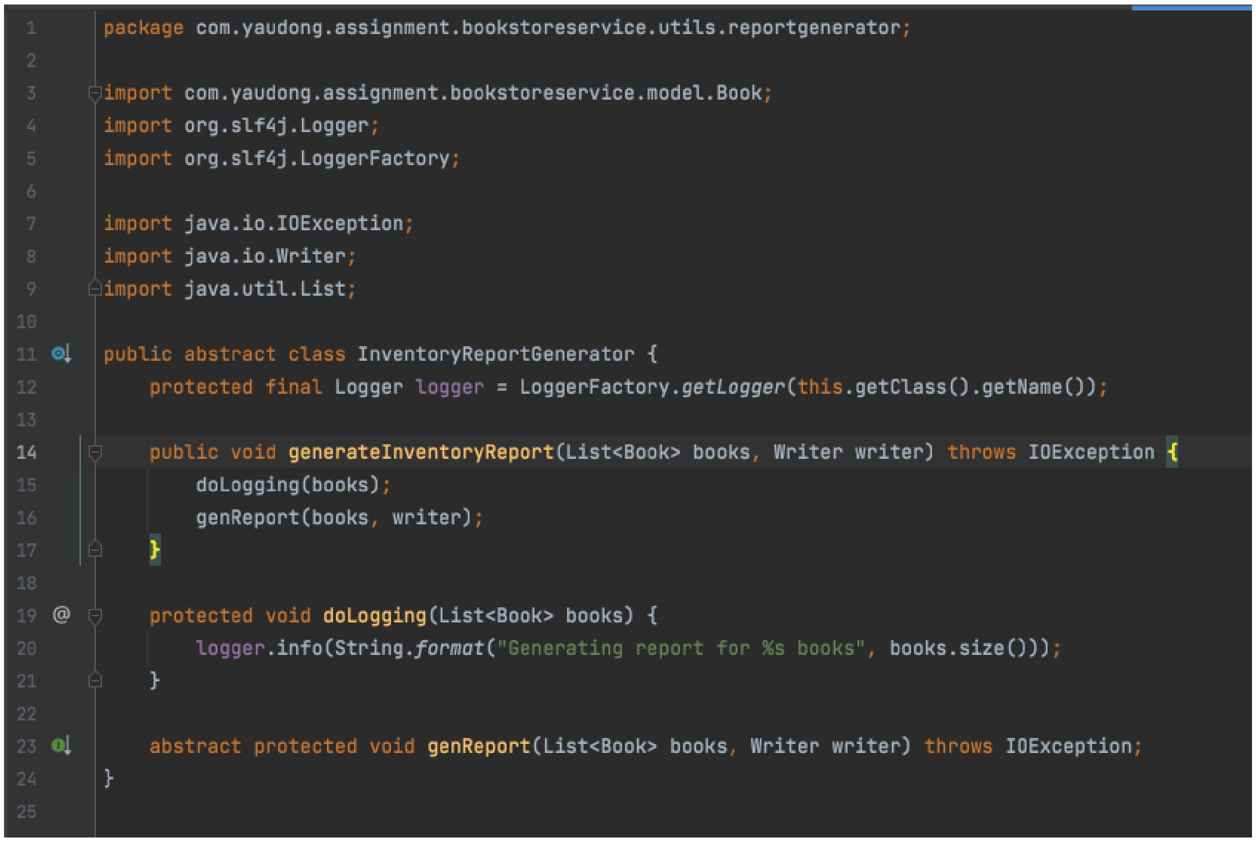
Error Handling and Logging

Error handling is done at applicable points in the application. The log output of the microservice is directed to both stdout and a log file called 'book_store_inventory.log' where the jar is run.

Design Pattern Implementation

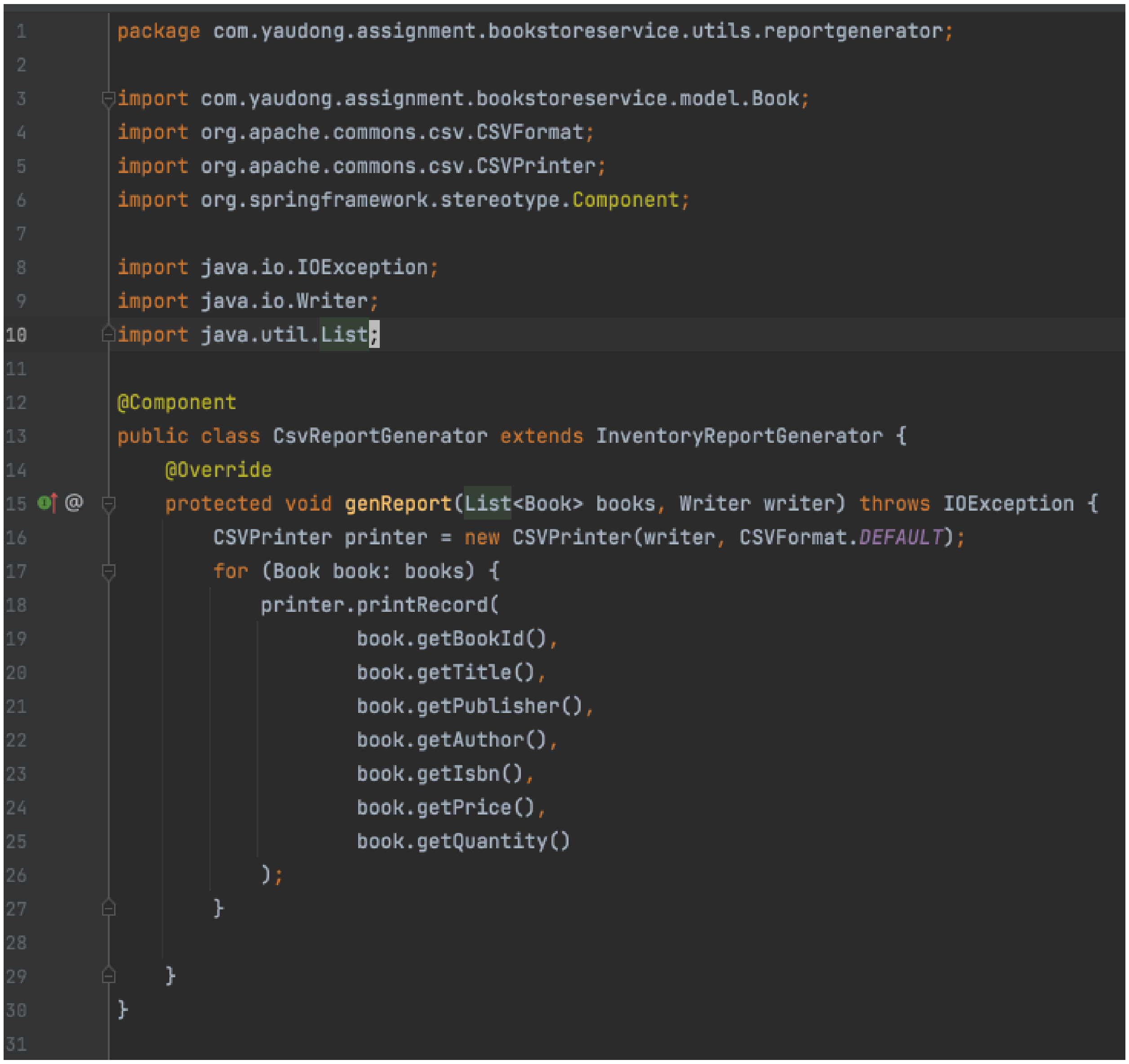
One of the implemented design patterns is the Template Method design pattern. Template Method pattern is a pattern that defines the algorithm skeleton in the superclass and lets the subclasses implement some steps in the algorithm without altering the algorithm structure. The superclass often comes with some implemented default steps in the algorithm skeleton.

Template Method pattern is implemented in the InventoryReportGenerator class which encapsulates the logic of generating a report of all the books in the inventory.

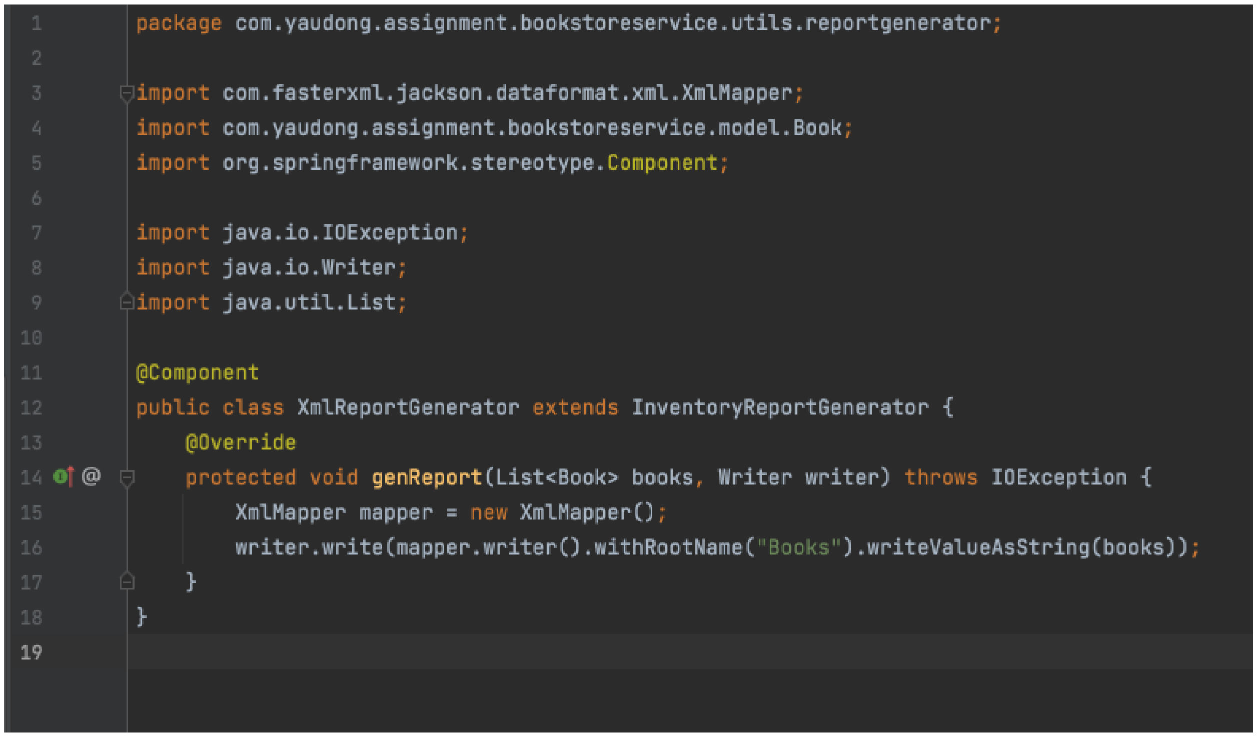


```java
package com.yaudong.assignment.bookstoreservice.utils.reportgenerator;

import com.yaudong.assignment.bookstoreservice.model.Book;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

import java.io.IOException;
import java.io.Writer;
import java.util.List;

public abstract class InventoryReportGenerator {
    protected final Logger logger = LoggerFactory.getLogger(this.getClass().getName());

    public void generateInventoryReport(List<Book> books, Writer writer) throws IOException {
        doLogging(books);
        genReport(books, writer);
    }

    protected void doLogging(List<Book> books) {
        logger.info(String.format("Generating report for %s books", books.size()));
    }

    abstract protected void genReport(List<Book> books, Writer writer) throws IOException;
}
```

The main method of the InventoryReportGenerator class is generateInventoryReport. The overall report generation steps are specified in this method. The specific logic of generating the report is delegated to the specific subclasses, which are CsvReportGenerator and XmlReportGenerator. A default implementation of the doLogging method is provided in the InventoryReportGenerator class. The subclasses can override the doLogging method if applicable.



```java
package com.yaudong.assignment.bookstoreservice.utils.reportgenerator;

import com.yaudong.assignment.bookstoreservice.model.Book;
import org.apache.commons.csv.CSVFormat;
import org.apache.commons.csv.CSVPrinter;
import org.springframework.stereotype.Component;

import java.io.IOException;
import java.io.Writer;
import java.util.List;

@Component
public class CsvReportGenerator extends InventoryReportGenerator {
    @Override
    protected void genReport(List<Book> books, Writer writer) throws IOException {
        CSVPrinter printer = new CSVPrinter(writer, CSVFormat.DEFAULT);
        for (Book book: books) {
            printer.printRecord(
                    book.getBookId(),
                    book.getTitle(),
                    book.getPublisher(),
                    book.getAuthor(),
                    book.getIsbn(),
                    book.getPrice(),
                    book.getQuantity()
            );
        }

    }
}
```



```java
package com.yaudong.assignment.bookstoreservice.utils.reportgenerator;

import com.fasterxml.jackson.dataformat.xml.XmlMapper;
import com.yaudong.assignment.bookstoreservice.model.Book;
import org.springframework.stereotype.Component;

import java.io.IOException;
import java.io.Writer;
import java.util.List;

@Component
public class XmlReportGenerator extends InventoryReportGenerator {
    @Override
    protected void genReport(List<Book> books, Writer writer) throws IOException {
        XmlMapper mapper = new XmlMapper();
        writer.write(mapper.writer().withRootName("Books").writeValueAsString(books));
    }
}
```