**WEEK 3 ADDITIONAL SOLUTIONS:**

***Exercise 5: Configuring the Spring IoC Container***

***Scenario:***

***The library management application requires a central configuration for beans and dependencies.***

***Steps:***

1. ***Create Spring Configuration File:***
   * ***Create an XML configuration file named applicationContext.xml in the src/main/resources directory.***
   * ***Define beans for BookService and BookRepository in the XML file.***
2. ***Update the BookService Class:***
   * ***Ensure that the BookService class has a setter method for BookRepository.***
3. ***Run the Application:***
   * ***Create a main class to load the Spring context and test the configuration.***

**CODE:**

***applicationContext.xml***

<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans.xsd">

<!-- Define repository bean -->

<bean id="bookRepository" class="com.library.repository.BookRepository"/>

<!-- Define service bean and inject repository -->

<bean id="bookService" class="com.library.service.BookService">

<property name="bookRepository" ref="bookRepository"/>

</bean>

</beans>

*BookRepository.java*

package com.library.repository;

public class BookRepository {

public String getBookTitle() {

return "Clean Code by Robert C. Martin";

}

}

*BookService.java*

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private BookRepository bookRepository;

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

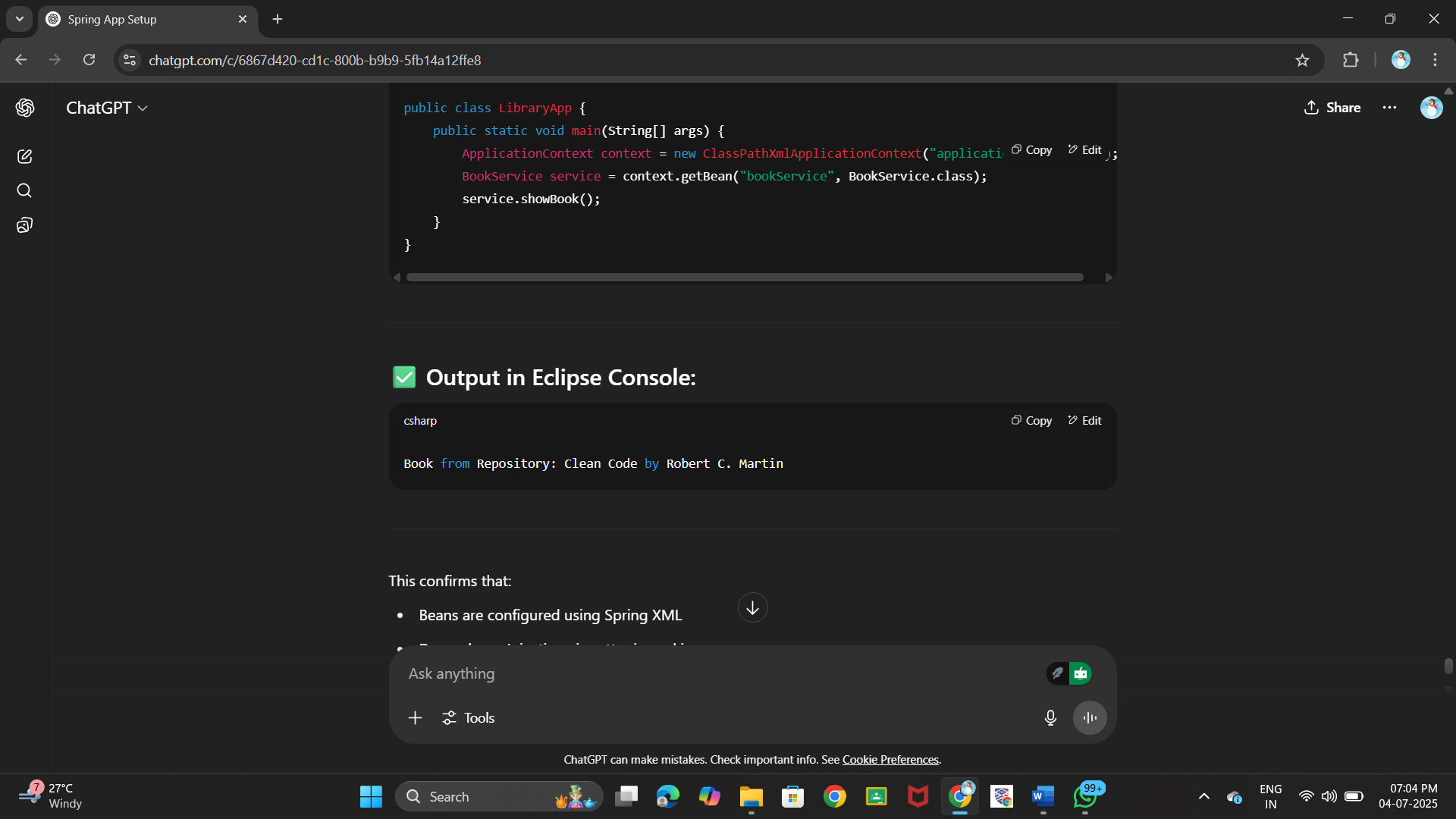
public void showBook() {

System.out.println("Book from Repository: " + bookRepository.getBookTitle());

}

}

**OUTPUT:**



***Exercise 7: Implementing Constructor and Setter Injection***

***Scenario:***

*The library management application requires both constructor and setter injection for better control over bean initialization.*

***Steps:***

1. ***Configure Constructor Injection:***
   * *Update applicationContext.****xml*** *to configure constructor injection for* ***BookService****.*
2. ***Configure Setter Injection:***
   * *Ensure that the* ***BookService*** *class has a setter method for* ***BookRepository*** *and configure it in* ***applicationContext.xml****.*
3. ***Test the Injection:***
   * *Run the* ***LibraryManagementApplication*** *main class to verify both constructor and setter injection.*

**CODE:**

*BookRepository.java*

package com.library.repository;

public class BookRepository {

public String getBookTitle() {

return "Effective Java by Joshua Bloch";

}

}

*BookService.java*

package com.library.service;

import com.library.repository.BookRepository;

public class BookService {

private String libraryName;

private BookRepository bookRepository;

// Constructor for constructor injection

public BookService(String libraryName) {

this.libraryName = libraryName;

}

// Setter for setter injection

public void setBookRepository(BookRepository bookRepository) {

this.bookRepository = bookRepository;

}

public void displayDetails() {

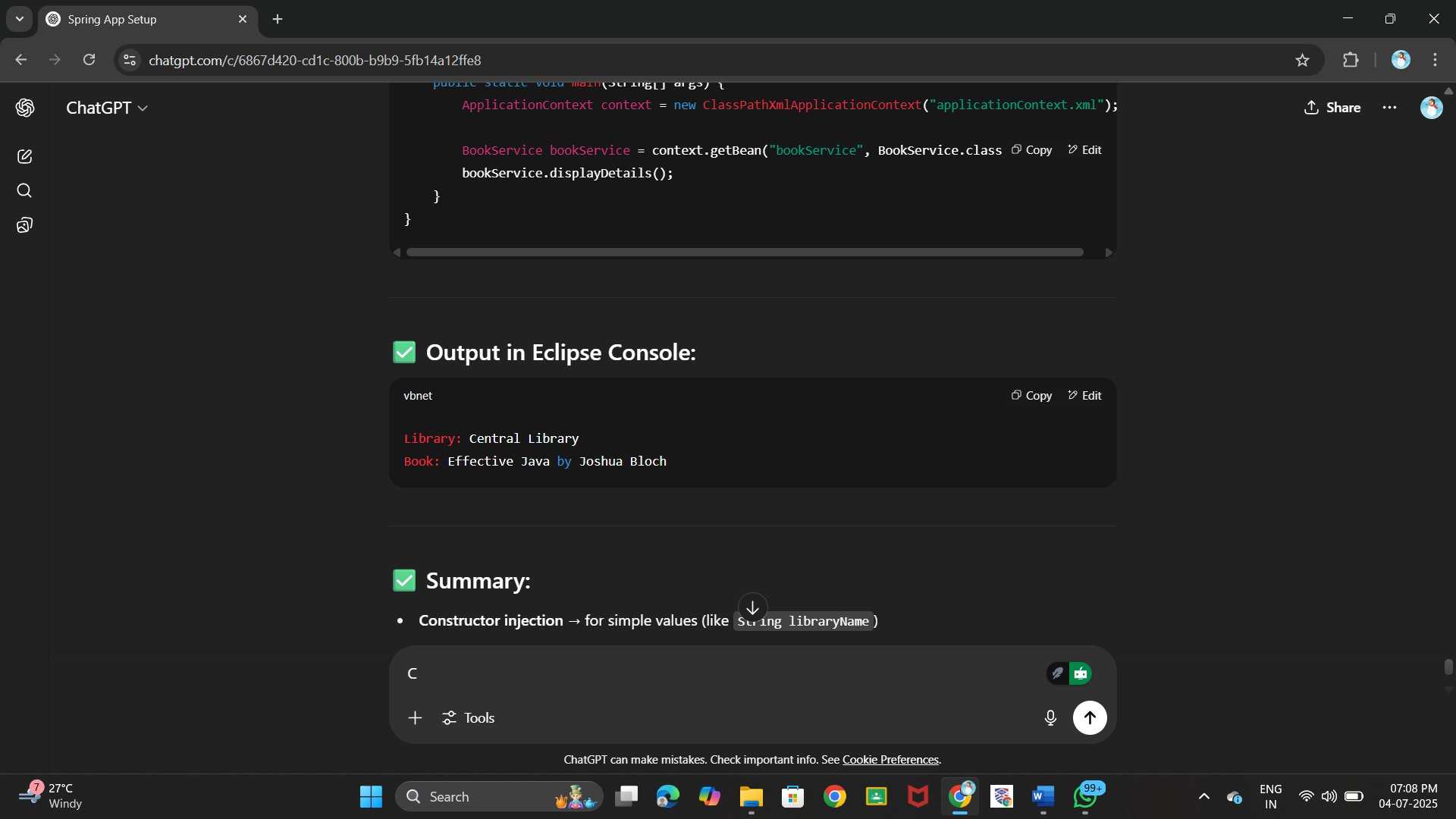
System.out.println("Library: " + libraryName);

System.out.println("Book: " + bookRepository.getBookTitle());

}

}

**OUTPUT:**



***Exercise 9: Creating a Spring Boot Application***

***Scenario:***

*You need to create a Spring Boot application for the library management system to simplify configuration and deployment.*

***Steps:***

1. ***Create a Spring Boot Project:***
   * *Use* ***Spring Initializr*** *to create a new Spring Boot project named* ***LibraryManagement****.*
2. ***Add Dependencies:***
   * *Include dependencies for* ***Spring Web, Spring Data JPA, and H2 Database****.*
3. ***Create Application Properties:***
   * *Configure database connection properties in* ***application.properties****.*
4. ***Define Entities and Repositories:***
   * *Create* ***Book*** *entity and* ***BookRepository*** *interface.*
5. ***Create a REST Controller:***
   * *Create a* ***BookController*** *class to handle CRUD operations.*
6. ***Run the Application:***
   * *Run the Spring Boot application and test the REST endpoints.*

**CODE:**

*Book.java*

package com.library.entity;

import jakarta.persistence.Entity;

import jakarta.persistence.Id;

@Entity

public class Book {

@Id

private int id;

private String title;

private String author;

public Book() {}

public Book(int id, String title, String author) {

this.id = id;

this.title = title;

this.author = author;

}

public int getId() { return id; }

public void setId(int id) { this.id = id; }

public String getTitle() { return title; }

public void setTitle(String title) { this.title = title; }

public String getAuthor() { return author; }

public void setAuthor(String author) { this.author = author; }

}

*BookRepository.java*

package com.library.repository;

import com.library.entity.Book;

import org.springframework.data.jpa.repository.JpaRepository;

public interface BookRepository extends JpaRepository<Book, Integer> {}

*BookController.java*

package com.library.controller;

import com.library.entity.Book;

import com.library.repository.BookRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/books")

public class BookController {

@Autowired

private BookRepository bookRepository;

@GetMapping

public List<Book> getAllBooks() {

return bookRepository.findAll();

}

@GetMapping("/{id}")

public Book getBookById(@PathVariable int id) {

return bookRepository.findById(id).orElse(null);

}

@PostMapping

public Book addBook(@RequestBody Book book) {

return bookRepository.save(book);

}

@PutMapping("/{id}")

public Book updateBook(@PathVariable int id, @RequestBody Book updatedBook) {

Book book = bookRepository.findById(id).orElse(null);

if (book != null) {

book.setTitle(updatedBook.getTitle());

book.setAuthor(updatedBook.getAuthor());

return bookRepository.save(book);

}

return null;

}

@DeleteMapping("/{id}")

public void deleteBook(@PathVariable int id) {

bookRepository.deleteById(id);

}

}

***Hands on 5***

***Implement services for managing Country   
  
An application requires for features to be implemented with regards to country. These features needs to be supported by implementing them as service using Spring Data JPA.***

* ***Find a country based on country code***
* ***Add new country***
* ***Update country***
* ***Delete country***
* ***Find list of countries matching a partial country name***

***Before starting the implementation of the above features, there are few configuration and data population that needs to be incorporated. Please refer each topic below and implement the same.   
  
Explanation for Hibernate table creation configuration***

* ***Moreover the ddl-auto defines how hibernate behaves if a specific table or column is not present in the database.***
  + ***create - drops existing tables data and structure, then creates new tables***
  + ***validate - check if the table and columns exist or not, throws an exception if a matching table or column is not found***
  + ***update - if a table does not exists, it creates a new table; if a column does not exists, it creates a new column***
  + ***create-drop - creates the table, once all operations are completed, the table is dropped***

***# Hibernate ddl auto (create, create-drop, update, validate)***

***spring.jpa.hibernate.ddl-auto=validate***

***Populate country table***

* ***Delete all the records in Country table and then use the below script to create the actual list of all countries in our world.***

***Refer subsequent hands on exercises to implement the features related to country***

**CODE:**

**package com.country.entity;**

**import jakarta.persistence.Entity;**

**import jakarta.persistence.Id;**

**@Entity**

**public class Country {**

**@Id**

**private String coCode;**

**private String coName;**

**public Country() {}**

**public Country(String coCode, String coName) {**

**this.coCode = coCode;**

**this.coName = coName;**

**}**

**public String getCoCode() {**

**return coCode;**

**}**

**public void setCoCode(String coCode) {**

**this.coCode = coCode;**

**}**

**public String getCoName() {**

**return coName;**

**}**

**public void setCoName(String coName) {**

**this.coName = coName;**

**}**

**}**

*CountryRepository.java*

package com.country.repository;

import com.country.entity.Country;

import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByCoNameContainingIgnoreCase(String name);

}

*CountryService.java*

package com.country.service;

import com.country.entity.Country;

import java.util.List;

public interface CountryService {

Country getCountry(String code);

Country addCountry(Country country);

Country updateCountry(String code, Country country);

void deleteCountry(String code);

List<Country> searchByName(String name);

}

*CountryServiceImpl.java*

package com.country.service;

import com.country.entity.Country;

import com.country.repository.CountryRepository;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class CountryServiceImpl implements CountryService {

@Autowired

private CountryRepository repository;

@Override

public Country getCountry(String code) {

return repository.findById(code).orElse(null);

}

@Override

public Country addCountry(Country country) {

return repository.save(country);

} @Override

public Country updateCountry(String code, Country country) {

Country existing = repository.findById(code).orElse(null);

if (existing != null) {

existing.setCoName(country.getCoName());

return repository.save(existing);

}

return null;

}

@Override

public void deleteCountry(String code) {

repository.deleteById(code);

}

@Override

public List<Country> searchByName(String name) {

return repository.findByCoNameContainingIgnoreCase(name);

}

}

*CountryController.java*

package com.country.controller;

import com.country.entity.Country;

import com.country.service.CountryService;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.web.bind.annotation.\*;

import java.util.List;

@RestController

@RequestMapping("/countries")

public class CountryController {

@Autowired

private CountryService service;

@GetMapping("/{code}")

public Country getCountry(@PathVariable String code) {

return service.getCountry(code);

}

@PostMapping

public Country addCountry(@RequestBody Country country) {

return service.addCountry(country);

}

@PutMapping("/{code}")

public Country updateCountry(@PathVariable String code, @RequestBody Country country) {

return service.updateCountry(code, country);

}

@DeleteMapping("/{code}")

public void deleteCountry(@PathVariable String code) {

service.deleteCountry(code);

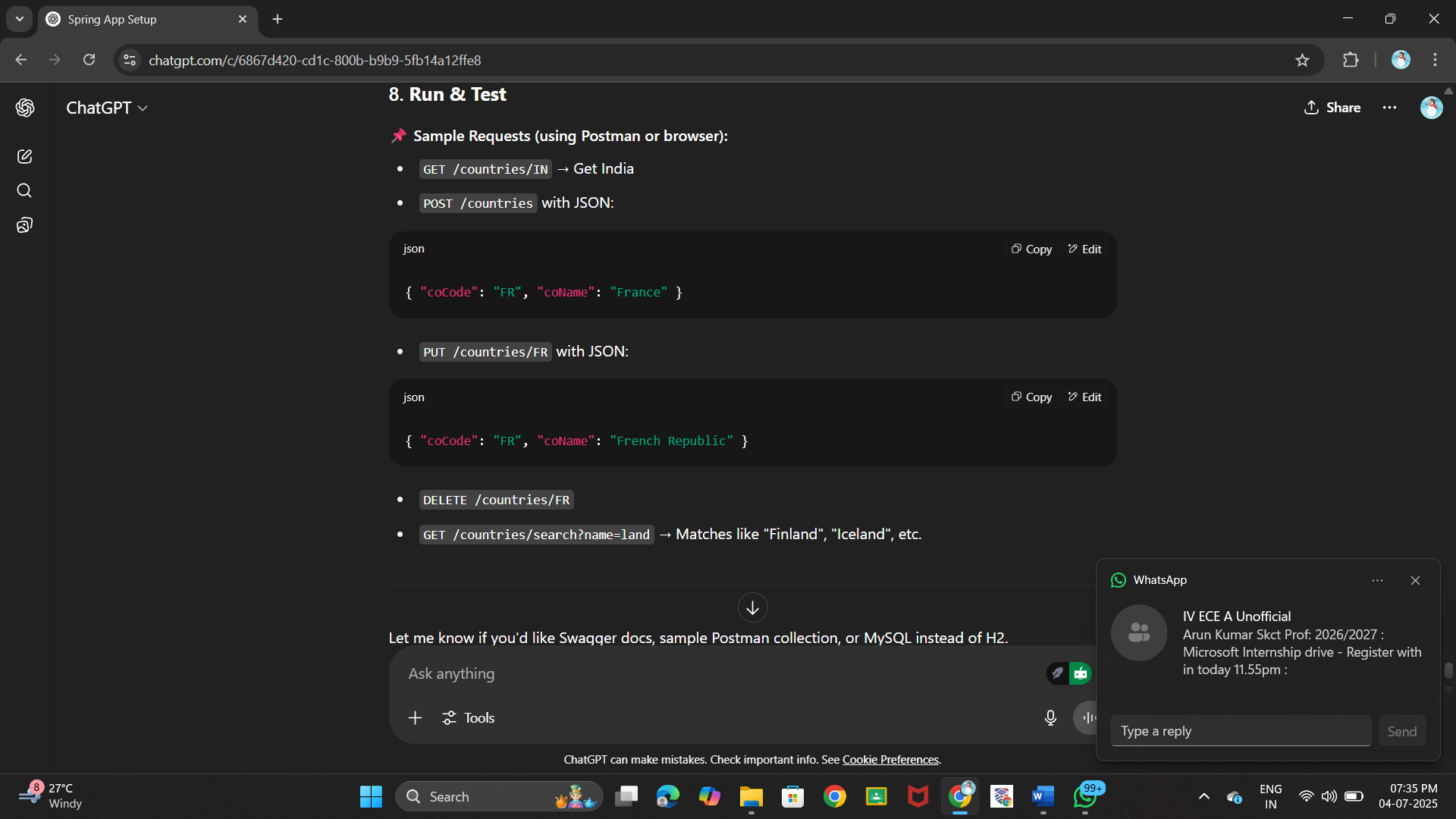
}

public List<Country> searchByName(@RequestParam String name) {

return service.searchByName(name);

}

}

**OUTPUT:**

***Hands on 6***

***Find a country based on country code   
 Create new exception class CountryNotFoundException in com.cognizant.spring-learn.service.exception***

* ***Create new method findCountryByCode() in CountryService with @Transactional annotation***
* ***In findCountryByCode() method, perform the following steps:***
  + ***Method signature***

***@Transactional***

***public Country findCountryByCode(String countryCode) throws CountryNotFoundException***

* ***Get the country based on findById() built in method***

***Optional<Country> result = countryRepository.findById(countryCode);***

* ***From the result, check if a country is found. If not found, throw CountryNotFoundException***

***if (!result.isPresent())***

* ***Use get() method to return the country fetched.***

***Country country = result.get();***

* ***Include new test method in OrmLearnApplication to find a country based on country code and compare the country name to check if it is valid.***

***private static void getAllCountriesTest() {***

***LOGGER.info("Start");***

***Country country = countryService.findCountryByCode("IN");***

***LOGGER.debug("Country:{}", country);***

***LOGGER.info("End");***

***}***

* ***Invoke the above method in main() method and test it.***

***NOTE: SME to explain the importance of @Transactional annotation. Spring takes care of creating the Hibernate session and manages the transactionality when executing the service method.***

**CODE:**

package com.cognizant.springlearn;

import com.cognizant.springlearn.entity.Country;

import com.cognizant.springlearn.service.CountryService;

import com.cognizant.springlearn.service.exception.CountryNotFoundException;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

@Autowired

private CountryService countryService;

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

@Override

public void run(String... args) throws Exception {

getAllCountriesTest();

}

private void getAllCountriesTest() {

LOGGER.info("Start");

try {

Country country = countryService.findCountryByCode("IN");

LOGGER.debug("Country: {}", country);

} catch (CountryNotFoundException e) {

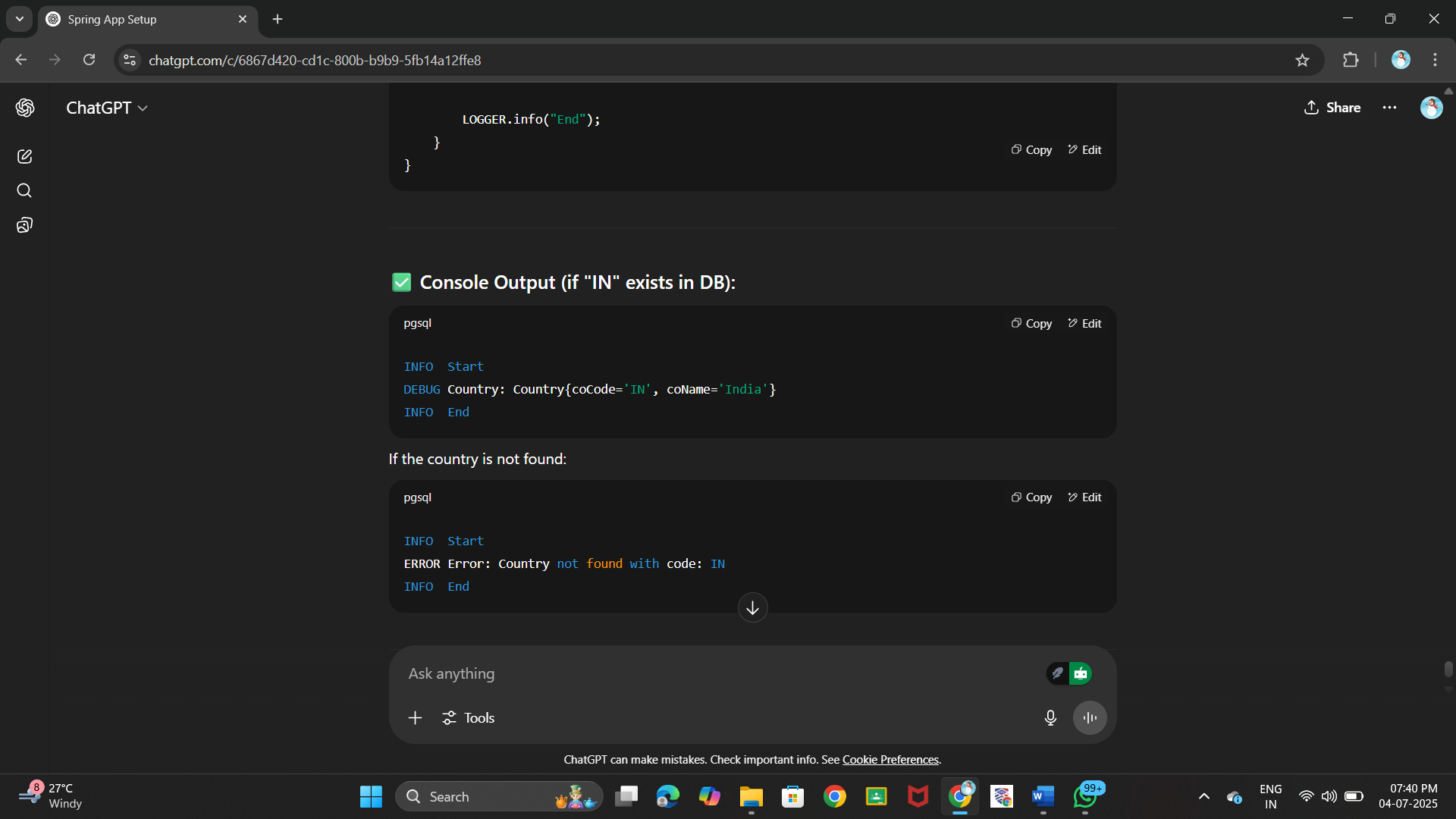
LOGGER.error("Error: {}", e.getMessage());

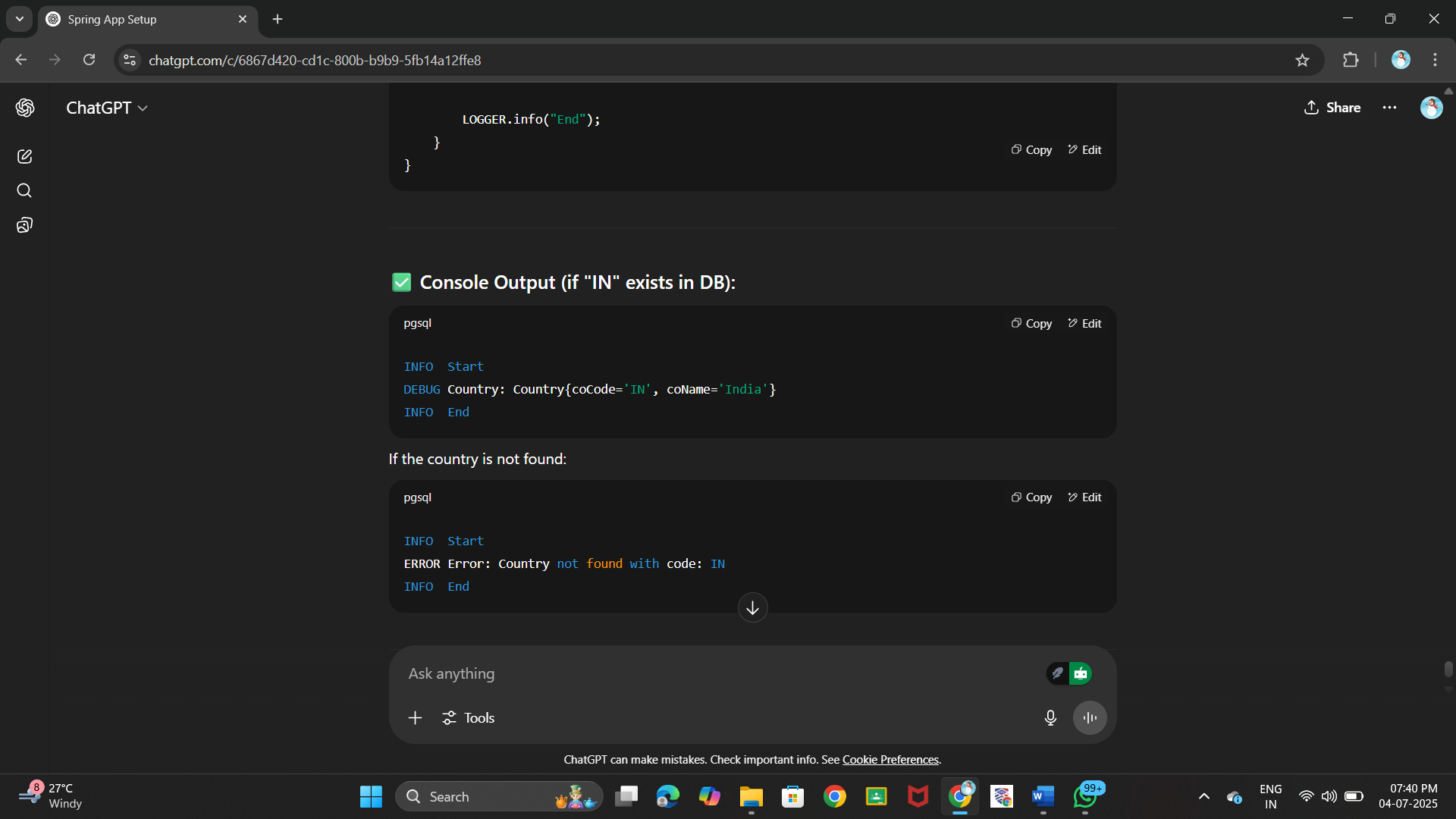
}

LOGGER.info("End");

}

}

**OUTPUT:**

if code not found:

***Hands on 7***

***Add a new country***

* ***Create new method in CountryService.***

***@Transactional***

***public void addCountry(Country country)***

* ***Invoke save() method of repository to get the country added.***

***countryRepository.save(country)***

* ***Include new testAddCountry() method in OrmLearnApplication. Perform steps below:***
  + ***Create new instance of country with a new code and name***
  + ***Call countryService.addCountry() passing the country created in the previous step.***
  + ***Invoke countryService.findCountryByCode() passing the same code used when adding a new country***
  + ***Check in the database if the country is added***

**CODE:**

private void testAddCountry() {

LOGGER.info("Start");

Country newCountry = new Country("XY", "Xyland");

countryService.addCountry(newCountry);

try {

Country result = countryService.findCountryByCode("XY");

LOGGER.debug("Added Country: {}", result);

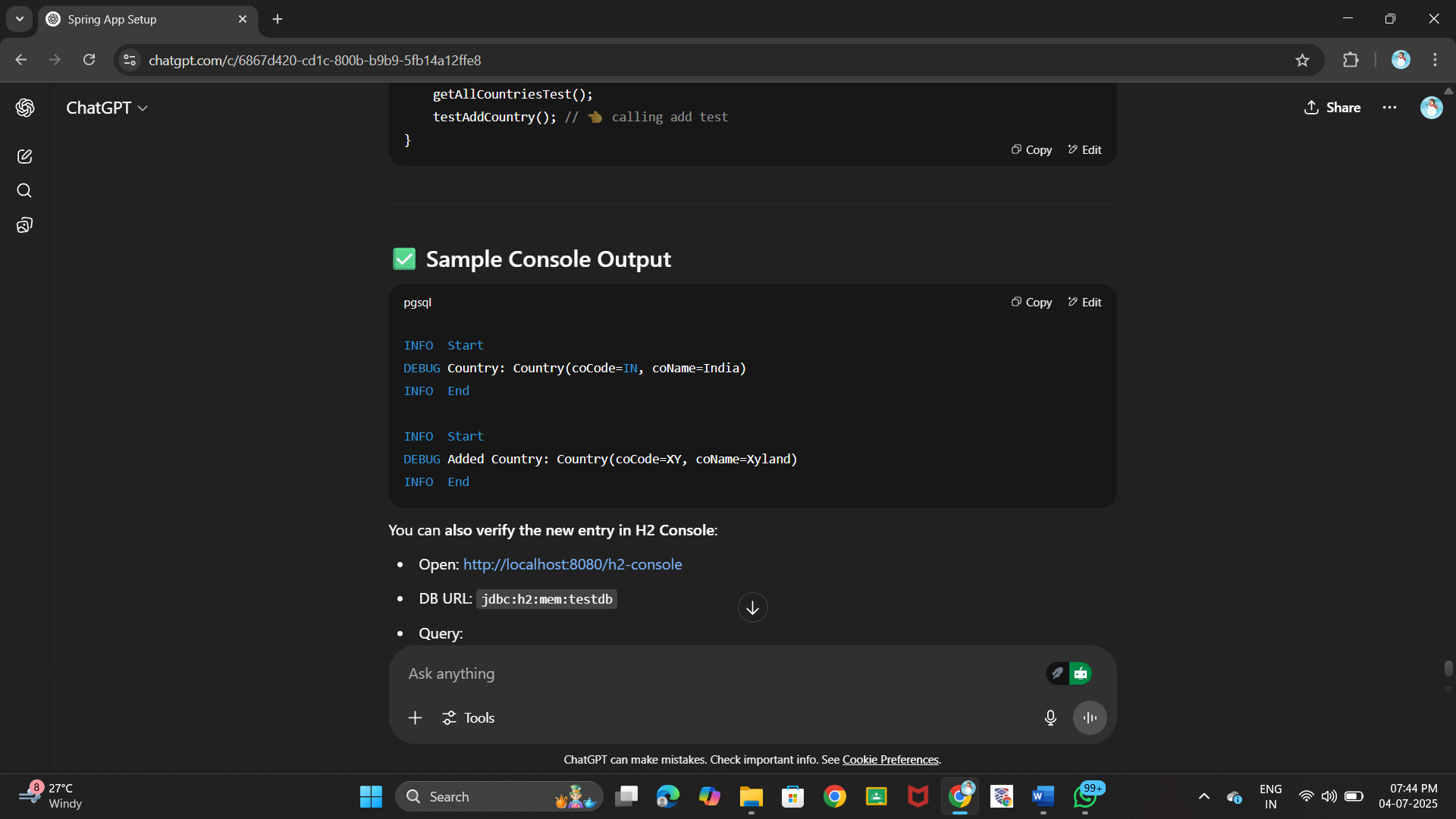
} catch (CountryNotFoundException e) {

LOGGER.error("Error: {}", e.getMessage());

}

LOGGER.info("End");

}

**OUTPUT:**

***Hands on 1***

***Write queries on country table using Query Methods   
  
Following are the list of queries that is required for an application. Implement these queries using Query Methods feature of Spring Data JPA. Click***[***here***](https://docs.spring.io/spring-data/jpa/docs/2.2.0.RELEASE/reference/html/#jpa.query-methods.query-creation)***for reference. Include appropriate methods in OrmLearnApplication and test the same.***

* ***An application has a search text box for searching by country. When typing characters on the text box, a list of all the matching countries should be displayed. For example, if 'ou' is entered in the search box the following countries should be displayed. Write a Query Method to achieve this feature. Implement this method in CountryRepository.***

***BV       Bouvet Island***

***DJ       Djibouti***

***GP       Guadeloupe***

***GS       South Georgia and the South Sandwich Islands***

***LU       Luxembourg***

***SS       South Sudan***

***TF       French Southern Territories***

***UM       United States Minor Outlying Islands***

***ZA       South Africa***

* ***Enhance the above method to return the countries in ascending order. Modify the query method name defined in the previous problem to achieve this.***

***BV       Bouvet Island***

***DJ       Djibouti***

***TF       French Southern Territories***

***GP       Guadeloupe***

***LU       Luxembourg***

***ZA       South Africa***

***GS       South Georgia and the South Sandwich Islands***

***SS       South Sudan***

***UM       United States Minor Outlying Islands***

* ***To select a country an alphabet index is displayed in a web page, when the user clicks on the alphabet, all the countries starting that alphabet needs to be displayed. For example if the alphabet choose is 'Z', then the following countries should be displayed. Write a query method to get this feature incorporated.***

***ZM       Zambia***

***ZW       Zimbabwe***

**CODE:**

*CountryRepository.java*

package com.cognizant.springlearn.repository;

import com.cognizant.springlearn.entity.Country;

import org.springframework.data.jpa.repository.JpaRepository;

import org.springframework.stereotype.Repository;

import java.util.List;

@Repository

public interface CountryRepository extends JpaRepository<Country, String> {

List<Country> findByCoNameContaining(String partial);

List<Country> findByCoNameContainingOrderByCoNameAsc(String partial);

List<Country> findByCoNameStartingWith(String alphabet);

}

*OrmLearnApplication.java*

package com.cognizant.springlearn;

import com.cognizant.springlearn.entity.Country;

import com.cognizant.springlearn.repository.CountryRepository;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.beans.factory.annotation.Autowired;

import org.springframework.boot.CommandLineRunner;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

import java.util.List;

@SpringBootApplication

public class OrmLearnApplication implements CommandLineRunner {

private static final Logger LOGGER = LoggerFactory.getLogger(OrmLearnApplication.class);

@Autowired

private CountryRepository countryRepository;

public static void main(String[] args) {

SpringApplication.run(OrmLearnApplication.class, args);

}

public void run(String... args) throws Exception {

searchByPartialMatch();

searchByPartialMatchSorted();

searchByAlphabet();

}

private void searchByPartialMatch() {

LOGGER.info("Start - searchByPartialMatch");

List<Country> countries = countryRepository.findByCoNameContaining("ou");

for (Country c : countries) {

LOGGER.debug("{}\t{}", c.getCoCode(), c.getCoName());

}

LOGGER.info("End");

}

private void searchByPartialMatchSorted() {

LOGGER.info("Start - searchByPartialMatchSorted");

List<Country> countries = countryRepository.findByCoNameContainingOrderByCoNameAsc("ou");

for (Country c : countries) {

LOGGER.debug("{}\t{}", c.getCoCode(), c.getCoName());

}

LOGGER.info("End");

}

private void searchByAlphabet() {

LOGGER.info("Start - searchByAlphabet");

List<Country> countries = countryRepository.findByCoNameStartingWith("Z");

for (Country c : countries) {

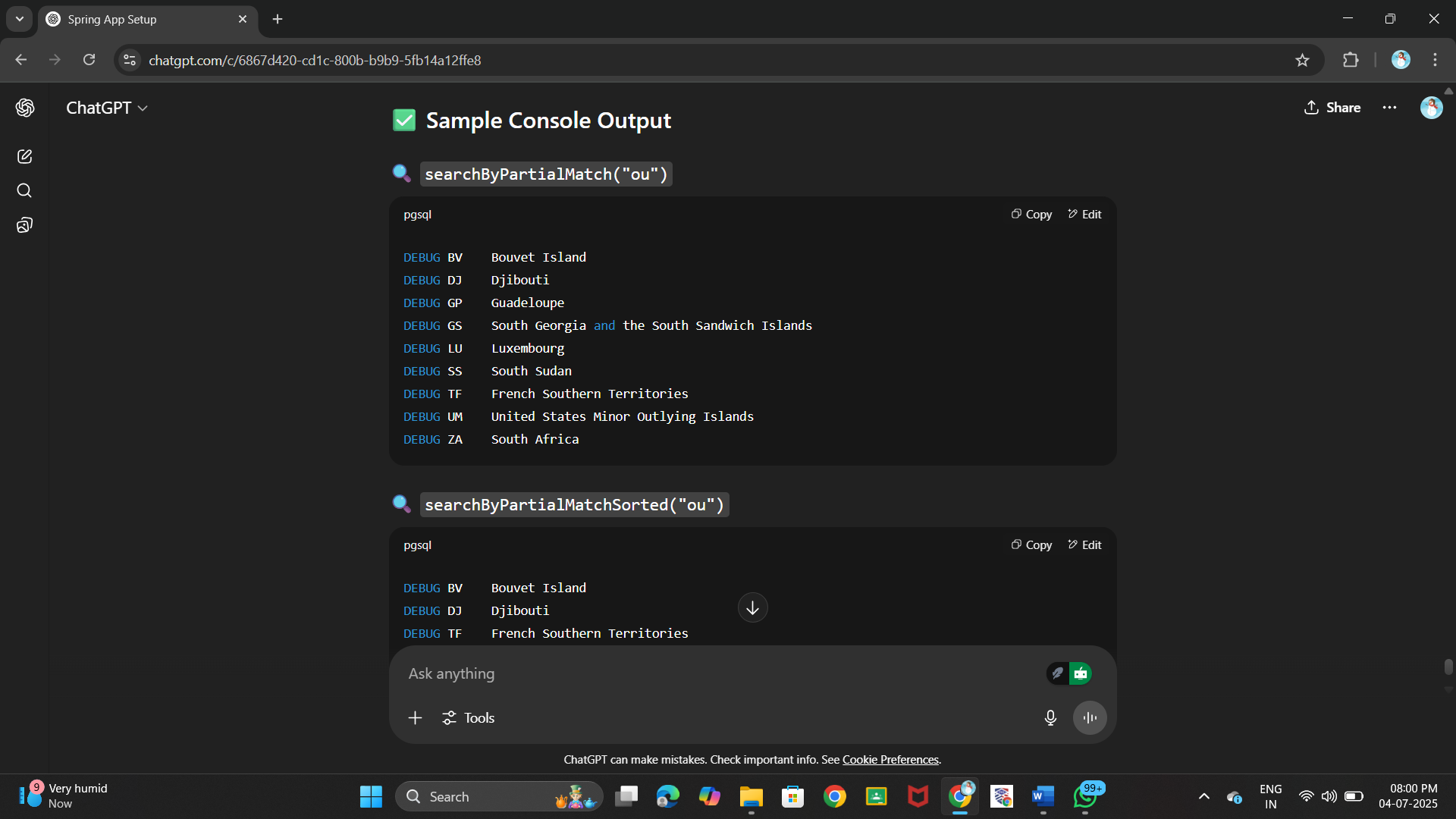
LOGGER.debug("{}\t{}", c.getCoCode(), c.getCoName());

}

LOGGER.info("End");

}

}

**OUTPUT:**