Project description:

The government is building a centralized library management system which involves a micro-service, allowing people to issue books and return them at the right time.

Library management requires creating users and renewing user subscriptions via authenticated processes. Secure create-user and renew-user-subscription APIs with Spring Security. The API issue-book should be permitted for everyone. This micro-service should be built using the Spring Boot REST API framework and it should connect to the database using the JPA API and secure methods using Spring Security.

In this task, implement 3 APIs for which the details are given below:

1. /api/v1/create-user [POST]: Simple user save method in an authenticated manner

2. /api/v1/issue-book [POST]: Send issue, check if users subscribed(see subscribed field in User entity), otherwise throw SubscriptionExpiredException- permitAll

3. /api/v1/renew-user-subscription/{id} [GET]: Send userId, set user subscription to true in an authenticated manner.

Your task is to complete the files given below:

1. src/main/java/com/wecp/library/controller/LibraryController.java

2. src/main/java/com/wecp/library/security/WebSecurityConfigurer.java

3. src/main/java/com/wecp/library/repository/UserRepository.java

4. src/main/java/com/wecp/library/repository/IssueRepository.java

You can find blank jpa repositories here:

1. src/main/java/com/wecp/library/repository/UserRepository.java

2. src/main/java/com/wecp/library/repository/IssueRepository.java

Notes:

1. Implementation related specifics are added directly as javadocs in the workspace.

2. Ensure that the structure and datatypes of the APIs are followed as specified in the comments to ensure that the code is evaluated correctly.

Testing & Submitting your code:

Step 1: Click on the WeCP Projects Button.

Step 2: Write your code to complete the tasks and click on the Test and Submit App to execute your code and confirm if the application is working as expected.

Step 3: Click on the Test and Submit App button to execute your code. You will be given a congratulations message if your code is working perfectly.

Step 4: You will be given a string url on clicking on Show testing url button. Append your api endpoints to the end of this string url to test your API endpoints on thunderclient.

Step 5: Use the Run App button to start the application before performing api testing.

**WebSecurity.java**

package com.wecp.library.security;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.annotation.web.configuration.WebSecurityConfigurerAdapter;

import org.springframework.security.web.SecurityFilterChain;

/\*\*

 \* Configure Spring Security class here. Don't forget to extend the class with the necessary Spring Security class.

 \* user and renew-user-subscription APIs must be authenticated and issue-book must be permitAll.

 \*/

@Configuration

@EnableWebSecurity

public class WebSecurityConfigurer extends WebSecurityConfigurerAdapter {

  public void config(HttpSecurity http) throws Exception{

    http.csrf().disable()

    .authorizeRequests()

    .antMatchers("/api/v1/issue-book").permitAll()

    .antMatchers("/api/v1/user","/api/v1/renew-user-subscription/{id}").authenticated().

    anyRequest().permitAll()

    .and()

    .httpBasic();

  }

}

**LibraryController.java**

package com.wecp.library.controller;

import com.wecp.library.controller.exception.UserNotSubscribedException;

import com.wecp.library.domain.Issue;

import com.wecp.library.domain.User;

import com.wecp.library.repository.IssueRepository;

import com.wecp.library.repository.UserRepository;

import org.apache.catalina.connector.Response;

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

import org.springframework.http.ResponseEntity;

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

import java.util.Optional;

/\*\*

 \* REST controller for managing library system process

 \*/

@RestController

@RequestMapping("/api/v1")

public class LibraryController {

  @Autowired

  private UserRepository userRepository;

  @Autowired

  private IssueRepository issueRepository;

  /\*\*

   \* {@code POST /issueBook} : Create a new issue.

   \*

   \* @param issue the issue to create.

   \* @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body

   \* the user, or if does not exist, return with status "noContent".

   \* If user is not subscribed, throw {@link UserNotSubscribedException}

   \*/

  @PostMapping("/issue-book")

  public ResponseEntity<Issue> issueBook(@RequestBody Issue issue) {

    Optional<User> optUser= userRepository.findById(issue.getUser().getId());

    if(optUser.isEmpty())

    {

      return ResponseEntity.noContent().build();

    }

    User user =optUser.get();

    if(!user.getSubscribed()){

      throw new UserNotSubscribedException();

    }

    Issue i1= issueRepository.save(issue);

    return ResponseEntity.ok(i1);

  }

  /\*\*

   \* {@code POST /user} : Create a new user.

   \*

   \* @param user the user to create.

   \* @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body the new user

   \*/

  @PostMapping("/user")

  public ResponseEntity<User> createUser(@RequestBody User user) {

    User u1 =userRepository.save(user);

    return ResponseEntity.ok(u1);

  }

  /\*\*

   \* {@code GET /renew-user-subscription/:id} : Send userId, set user subscription to true

   \*

   \* @param id the id of the user to renew subscription.

   \* @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body

   \* the user, or if does not exist, return with status "noContent".

   \*/

  @GetMapping("renew-user-subscription/{id}")

  public ResponseEntity<User> renewUserSubscription(@PathVariable Long id) {

    Optional<User> optUser =userRepository.findById(id);

    if(optUser.isEmpty())

    {

      return ResponseEntity.notFound().build();

    }

    User user =optUser.get();

    user.setSubscribed(true);

    return ResponseEntity.ok(userRepository.save(user));

  }

}

---

Problem Description:

The government is building a centralized library management system which involves a micro-service, allowing people issue books and depositing them at the right time. This micro-service should be built using the Spring Boot framework and connect to the database using the JPA API.

The application exposes 4 APIs.

The details of the APIs are given below:

- /api/v1/user [POST]: Add a new user.

- /api/v1/user [GET]: Fetch the details of a user with id

- /api/v1/book [POST]: Add a new book.

- /api/v1/book/{id} [GET]: Fetch the details of a book.

It is crucial that you follow the javadocs in the template classes and return information accordingly so tests can validate.

Your task is to complete the file:

src/main/java/com/wecp/library/controller/LibraryController.java

You can find blank jpa repositories here:

src/main/java/com/wecp/library/repository/BookRepository.java

src/main/java/com/wecp/library/repository/UserRepository.java

Testing & Submitting your code:

Step 1: Click on the WeCP Projects Button shown below.

Step 2: Write your code to complete the task(s) according to the question. Click on the Submit & Test App button shown above to execute your code and confirm if the application is working as expected.

Step 3: Click on the Submit & Test App button to execute your code. You will be given a congratulations message as shown below if your code is working perfectly.

package com.wecp.library.controller;

import com.wecp.library.domain.Book;

import com.wecp.library.domain.User;

import com.wecp.library.repository.BookRepository;

import com.wecp.library.repository.UserRepository;

import org.springframework.dao.EmptyResultDataAccessException;

import org.springframework.http.ResponseEntity;

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

import java.util.Optional;

/\*\*

 \* REST controller for managing library system process

 \*/

@RestController

@RequestMapping("/api/v1")

public class LibraryController {

  /\*\*

   \* {@code GET /user/:id} : get the "id" User.

   \*

   \* @param id the id of the user to retrieve.

   \* @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body

   \* the user, or if does not exist, return with status "noContent".

   \*/

  @GetMapping("/user/{id}")

  public ResponseEntity<User> getUser(@PathVariable Long id) {

    return ResponseEntity.ok().body(null);

  }

  /\*\*

   \* {@code POST /user} : Create a new user.

   \*

   \* @param user the user to create.

   \* @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body the new user

   \*/

  @PostMapping("/user")

  public ResponseEntity<User> createUser(@RequestBody User user) {

    return ResponseEntity.ok().body(null);

  }

  /\*\*

   \* {@code GET /book/:id} : get the "id" Book.

   \*

   \* @param id the id of the book to retrieve.

   \* @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body

   \* the book, or if does not exist, return with status "noContent".

   \*/

  @GetMapping("/book/{id}")

  public ResponseEntity<Book> getBook(@PathVariable Long id) {

    return ResponseEntity.ok().body(null);

  }

  /\*\*

   \* {@code POST /book} : Create a new book.

   \*

   \* @param book the book to create.

   \* @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body the new book

   \*/

  @PostMapping("/book")

  public ResponseEntity<Book> createBook(@RequestBody Book book) {

    return ResponseEntity.ok().body(null);

  }

}

---

Project Description:

The government is building a centralized library management system which involves a micro-service, allowing people to issue books and return them at the right time.

In this task, you need to conform to the following requirements:

1. Build "depositBook" API in a reactive way(should return Mono).

2. Users can issue a book and deposit them. If they issue the book after the given period, fine should apply.

3. You can find the fields and do the math(issueDate, period, returnDate).

4. This micro-service should be built using the Spring Boot REST API framework and connect to the database using the JPA API.

5. The "depositBook" API should be built using the Spring Boot WebFlux.

The application exposes 2 more APIs. The details of the APIs are given below:

1. /api/v1/issueBook [POST]: Users issue a book of their choice, if book quantity is zero, it should throw "BookNotAvailableException"

2. /api/v1/depositBook [POST]: Send issue, return fine if there is one or return 0 (this one should be reactive)

Your task is to complete the files:

1. src/main/java/com/wecp/library/controller/LibraryController.java

2. src/main/java/com/wecp/library/repository/BookRepository.java

3. src/main/java/com/wecp/library/repository/IssueRepository.java

You can find blank jpa repositories here:

1. src/main/java/com/wecp/library/repository/BookRepository.java

2. src/main/java/com/wecp/library/repository/IssueRepository.java

Notes:

1. Implementation related specifics are added directly as javadocs in the workspace.

2. Ensure that the structure and datatypes of the APIs are followed as specified in the comments to ensure that the code is evaluated correctly.

Testing & Submitting your code:

Step 1: Click on the WeCP Projects Button.

Step 2: Write your code to complete the task(s) according to the question. Click on the Submit & Test App button to execute your code and confirm if the application is working as expected.

Step 3: Click on the Submit & Test App button to execute your code. You will be given a congratulations message if your code is working perfectly.

package com.wecp.library.controller;

import com.wecp.library.controller.exception.BookNotAvailableException;

import com.wecp.library.domain.Book;

import com.wecp.library.domain.Issue;

import com.wecp.library.repository.BookRepository;

import com.wecp.library.repository.IssueRepository;

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

import org.springframework.http.ResponseEntity;

import org.springframework.web.bind.annotation.PostMapping;

import org.springframework.web.bind.annotation.RequestBody;

import org.springframework.web.bind.annotation.RequestMapping;

import org.springframework.web.bind.annotation.RestController;

import reactor.core.publisher.Mono;

import java.time.LocalDate;

import java.time.temporal.ChronoUnit;

import java.util.Optional;

/\*\*

 \* REST controller for managing library system process

 \*/

@RestController

@RequestMapping("/api/v1")

public class LibraryController {

  @Autowired

  private BookRepository bookRepository;

  @Autowired

  private IssueRepository issueRepository;

  /\*\*

   \* {@code POST /issueBook} : Create a new issue.

   \*

   \* @param issue the issue to create.

   \* @return the {@link ResponseEntity} with status {@code 200 (OK)} and with body the new issue.

   \* throw {@link BookNotAvailableException} if the wanted books quantity is zero

   \*/

  @PostMapping("/issueBook")

  public ResponseEntity<Issue> issueBook(@RequestBody Issue issue) {

    Book book = issue.getBook();

    if(book.getQuantity() ==0){

throw new BookNotAvailableException();

    }

    Issue i= issueRepository.save(issue);

    return ResponseEntity.ok(i);

  }

  /\*\*

   \* {@code POST /depositBook} : Inquiry the issue if there is a fine

   \*

   \* @param issue the issue to inquiry.

   \* @return the Mono with fine or just return 0

   \*/

  @PostMapping("/depositBook")

  public Mono<Integer> depositBook(@RequestBody Issue issue) {

    Optional<Issue> opI = issueRepository.findById(issue.getId());

    if(opI.isPresent()){

      LocalDate iDatae = issue.getIssueDate();

      LocalDate rDate = issue.getReturnDate();

      long d = ChronoUnit.DAYS.between(iDatae, rDate);

      if(d>issue.getPeriod() ){

        issueRepository.save(issue);

        return Mono.just(issue.getFine());

      }

    }

    return Mono.just(0);

  }

}

---

You are required to create a simple RESTful API using Spring Boot, which will be secured using Spring Security. The API will manage a list of books, allowing CRUD operations. However, these operations should be secured so that only authenticated users can perform them.

Tasks:

1. Model Creation:

  Create a Book entity with attributes: id (primary key), title, and author.

2. Repository Layer:

  Implement a repository interface for the Book entity using Spring Data JPA.

3. Service Layer:

  Create a service class that implements business logic for handling CRUD operations on books. This service class will use the repository to interact with the database.

4. Controller Layer:

  Develop a REST controller to expose CRUD operations as HTTP endpoints.

  API Endpoints:

  Create the following API endpoints

   1. GET /books: To get all books.

   2. GET /books/{id}: To get book by its id.

   3. POST /books: To create a new book.

   4. PUT /books/{id}: To update the details of an existing book.

   5. DELETE /books/{id}: To delete a book by its id.

5. Security Configuration:

  Configure Spring Security to secure your API endpoints. Use HTTP Basic Authentication for simplicity.

  Define two roles: USER and ADMIN. USER can only read books, whereas ADMIN can perform all CRUD operations.

  Configure an in-memory authentication with at least one user for each role.

Your task is to complete the following files:

  1. ./src/main/java/com/wecp/booksmanagementsystem/config/SecurityConfig.java

  2. ./src/main/java/com/wecp/booksmanagementsystem/controller/BookController.java

  3. ./src/main/java/com/wecp/booksmanagementsystem/entity/Book.java

  4. ./src/main/java/com/wecp/booksmanagementsystem/repository/BookRepository.java

  5. ./src/main/java/com/wecp/booksmanagementsystem/service

Test Cases:

  1. Authenticated requests with the USER role can only access read operations.

  2. Authenticated requests with the ADMIN role can perform all operations.

Testing and submitting the code:

Step 1: Click on the WeCP Projects button.

Step 2: Click on the Run app button to run the application.

Step 3: You can test your code by clicking on Test and Submit button. You will get a congratulations message upon successful completion of the task.

Step 4: Click on Show testing url button to get the url to perform testing using thunderclient.

package com.wecp.booksmanagementsystem.config;

import org.springframework.context.annotation.Bean;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpMethod;

import org.springframework.security.authentication.AuthenticationManager;

import org.springframework.security.config.annotation.authentication.configuration.AuthenticationConfiguration;

import org.springframework.security.config.annotation.method.configuration.EnableMethodSecurity;

import org.springframework.security.config.annotation.web.builders.HttpSecurity;

import org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;

import org.springframework.security.config.http.SessionCreationPolicy;

import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;

import org.springframework.security.crypto.password.PasswordEncoder;

import org.springframework.security.core.userdetails.User;

import org.springframework.security.core.userdetails.UserDetailsService;

import org.springframework.security.provisioning.InMemoryUserDetailsManager;

import org.springframework.security.web.SecurityFilterChain;

@Configuration

@EnableWebSecurity

@EnableMethodSecurity

public class SecurityConfig {

  @Bean

  public SecurityFilterChain securityFilterChain(HttpSecurity http) throws Exception {

    http

      .csrf().disable()

      .sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS)

      .and()

      .authorizeHttpRequests(auth -> auth

      .requestMatchers(HttpMethod.GET, "/api/books/\*\*").hasAnyRole("USER", "ADMIN")

      .requestMatchers(HttpMethod.POST, "/api/books/\*\*").hasRole("ADMIN")

      .requestMatchers(HttpMethod.PUT, "/api/books/\*\*").hasRole("ADMIN")

      .requestMatchers(HttpMethod.DELETE, "/api/books/\*\*").hasRole("ADMIN")

        .anyRequest().authenticated()

      )

      .httpBasic();

    return http.build();

  }

  @Bean

  public UserDetailsService userDetailsService() {

    return new InMemoryUserDetailsManager(

      User.withUsername("user")

        .password(passwordEncoder().encode("userpass"))

        .roles("USER")

        .build(),

      User.withUsername("admin")

        .password(passwordEncoder().encode("adminpass"))

        .roles("ADMIN")

        .build()

    );

  }

  @Bean

  public PasswordEncoder passwordEncoder() {

    return new BCryptPasswordEncoder();

  }

  @Bean

  public AuthenticationManager authenticationManager(AuthenticationConfiguration config) throws Exception {

    return config.getAuthenticationManager();

  }

}

package com.wecp.booksmanagementsystem.controller;

import com.wecp.booksmanagementsystem.entity.Book;

import com.wecp.booksmanagementsystem.service.BookService;

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

import org.springframework.http.HttpStatus;

import org.springframework.http.ResponseEntity;

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

import java.util.List;

@RestController

public class BookController {

  @Autowired

  private BookService bookService;

  @GetMapping("/api/books")

  public ResponseEntity<List<Book>> getAllBooks() {

    List<Book> books = bookService.getAllBooks();

    return ResponseEntity.ok(books);

  }

  @GetMapping("/api/books/{id}")

  public ResponseEntity<Book> getBookById(@PathVariable Long id) {

    Book book = bookService.getBookById(id);

    return ResponseEntity.ok(book);

  }

  @PostMapping("/api/books")

  public ResponseEntity<Book> createBook(@RequestBody Book book) {

    Book createdBook = bookService.createBook(book);

    return new ResponseEntity<>(createdBook, HttpStatus.CREATED); // ✅ Return 201 CREATED

  }

  @PutMapping("/api/books/{id}")

  public ResponseEntity<Book> updateBook(@PathVariable Long id, @RequestBody Book updatedBook) {

    Book book = bookService.updateBook(updatedBook); // ✅ Fixed typo: updatBook → updateBook

    return ResponseEntity.ok(book);

  }

  @DeleteMapping("/api/books/{id}")

  public ResponseEntity<Void> deleteBook(@PathVariable Long id) {

    bookService.deleteBook(id);

    return ResponseEntity.noContent().build(); // ✅ Return 204 NO\_CONTENT

  }

}

package com.wecp.booksmanagementsystem.service;

import com.wecp.booksmanagementsystem.entity.Book;

import com.wecp.booksmanagementsystem.repository.BookRepository;

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

import org.springframework.stereotype.Service;

import org.springframework.web.bind.annotation.PathVariable;

import org.springframework.web.bind.annotation.RequestBody;

import java.util.List;

import java.util.Optional;

@Service

public class BookService {

  @Autowired

  private BookRepository bookRepository;

  public List<Book> getAllBooks()

  {

    return bookRepository.findAll();

  }

  public Book getBookById(Long Id){

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

  }

  public Book createBook( Book book)

  {

    return bookRepository.save(book);

  }

  public Book updateBook( Book book)

  {

    book.setId(book.getId());

    book.setTitle(book.getTitle());

    book.setAuthor(book.getAuthor());

    return bookRepository.save(book);

  }

  public void deleteBook(Long id)

  {

    bookRepository.deleteById(id);

  }

  // implements business logic for handling CRUD operations on books.

}

---

# Implement CRUD using SpringBoot, JPA and MySQL for an online gaming application

You are developing a Spring Boot application for an online gaming platform. The platform features a variety of games, with players joining teams to compete in these games.

Each game can have multiple teams. Additionally, players are assigned to one team, and each team can consist of multiple players.

Entities:

1. Game Entity

 Attributes:

  a. gameId: Unique identifier. (type Long)

  b. gameName: Name of the game. (type String)

  c. genre: Genre of the game. (type String)

 Relationship: None in this simplified structure.

2. Team Entity

 Attributes:

  a. teamId: Unique identifier. (type Long)

  b. teamName: Name of the team. (type String)

  c. players: List of player names in the team. (type List<String>)

  d. game: The game this team is associated with.

  Relationship: ManyToOne with Game. (i.e., A team is associated with one game, but a game can have multiple teams.)

Repositories:

1. GameRepository: Should extend JpaRepository.

2. TeamRepository: Should extend JpaRepository.

Services:

=======================

1. GameService: Provides methods to:

 a. addGame(Game game): Adds a new game.

 b. getGameById(Long gameId): Retrieves a game by its ID.

 c. getAllGames(): Returns all available games.

2. TeamService: Offers methods to:

 a. addTeam(Team team): Adds a team.

 b. getTeamById(Long teamId): Retrieves a team by their ID.

 c. getAllTeams(): Returns all teams.

Controllers:

1. GameController: Handles requests related to the Game entity:

 a. POST /games: Uses GameService.addGame(Game game) to create a new game. Details are provided in the request body.

 b. GET /games/{gameId}: Uses GameService.getGameById(Long gameId) to fetch a game by its ID.

 c. GET /games: Uses GameService.getAllGames() to retrieve all games.

2. TeamController: Manages requests related to the Team entity:

 a. POST /teams: Uses TeamService.addTeam(Team team) to register a new team. Details are provided in the request body.

 b. GET /teams/{teamId}: Uses TeamService.getTeamById(Long teamId) to retrieve a team by their ID.

 c. GET /teams: Uses TeamService.getAllTeams() to fetch all teams.

Testing and submitting the code:

Step 1: Click on the WeCP Projects button.

Step 2: Click on the Run app button to run the application.

Step 3: You can test your code by clicking on Test and Submit button. You will get a congratulations message upon successful completion of the task.

Step 4: Click on Show testing url button to get the url to perform testing using thunderclient.

Step 5: Click on Open mysql bash session button to access MySQL.

package com.wecp.onlinegamingapplication.entity;

import javax.persistence.Entity;

import javax.persistence.GeneratedValue;

import javax.persistence.GenerationType;

import javax.persistence.Id;

@Entity

public class Game {

  // implement entity here

  @Id

  @GeneratedValue(strategy = GenerationType.IDENTITY)

  private Long gameId;

  private String gameName;

  private String genre;

  public Long getGameId() {

    return gameId;

  }

  public void setGameId(Long gameId) {

    this.gameId = gameId;

  }

  public String getGameName() {

    return gameName;

  }

  public void setGameName(String gameName) {

    this.gameName = gameName;

  }

  public String getGenre() {

    return genre;

  }

  public void setGenre(String genre) {

    this.genre = genre;

  }

  public Game(Long gameId, String gameName, String genre) {

    this.gameId = gameId;

    this.gameName = gameName;

    this.genre = genre;

  }

  public Game() {

  }

}

package com.wecp.onlinegamingapplication.entity;

import javax.persistence.\*;

import java.util.ArrayList;

import java.util.List;

@Entity

public class Team {

  @Id

  @GeneratedValue(strategy = GenerationType.IDENTITY)

  private Long teamId;

  private String teamName;

  @ElementCollection

  private List<String> players = new ArrayList<>();

  @ManyToOne

  @JoinColumn(name = "game\_Id", nullable = false)

  private Game game;

  // No-arg constructor

  public Team() {

  }

  // Constructor without teamId (auto-generated)

  public Team(String teamName, List<String> players, Game game) {

    this.teamName = teamName;

    this.players = players;

    this.game = game;

  }

  // Getters and Setters

  public Long getTeamId() {

    return teamId;

  }

  public void setTeamId(Long teamId) {

    this.teamId = teamId;

  }

  public String getTeamName() {

    return teamName;

  }

  public void setTeamName(String teamName) {

    this.teamName = teamName;

  }

  public List<String> getPlayers() {

    return players;

  }

  public void setPlayers(List<String> players) {

    this.players = players;

  }

  public Game getGame() {

    return game;

  }

  public void setGame(Game game) {

    this.game = game;

  }

}

package com.wecp.onlinegamingapplication.repository;

import com.wecp.onlinegamingapplication.entity.Game;

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

import org.springframework.stereotype.Repository;

@Repository

public interface GameRepository extends JpaRepository<Game, Long> {

  // use jpa repository

}

package com.wecp.onlinegamingapplication.controller;

import com.wecp.onlinegamingapplication.entity.Game;

import com.wecp.onlinegamingapplication.service.GameService;

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

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

import java.util.List;

@RestController

@RequestMapping("/games")

public class GameController {

@Autowired

  private GameService gameService;

  @PostMapping

  public Game addGame(@RequestBody Game game) {

    // add a game

    return gameService.addGame(game);

  }

  @GetMapping("/{gameId}")

  public Game getGameById(@PathVariable Long gameId) {

    // get a game by id

    return gameService.getGameById(gameId);

  }

  @GetMapping

  public List<Game> getAllGames() {

    // get all games

    return gameService.getAllGames();

  }

}

package com.wecp.onlinegamingapplication.controller;

import com.wecp.onlinegamingapplication.entity.Team;

import com.wecp.onlinegamingapplication.service.TeamService;

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

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

import java.util.List;

@RestController

@RequestMapping("/teams")

public class TeamController {

 @Autowired

  private TeamService teamService;

  @PostMapping

  public Team addTeam(@RequestBody Team team) {

   // add a team

   return teamService.addTeam(team);

  }

@GetMapping("/{teamId}")

  public Team getTeamById(@PathVariable Long teamId) {

   return teamService.getTeamById(teamId);

   // get a team by id

  }

  @GetMapping

  public List<Team> getAllTeams() {

   return teamService.getAllTeams();

    // get all teams

  }

}

package com.wecp.onlinegamingapplication.service;

import com.wecp.onlinegamingapplication.entity.Game;

import com.wecp.onlinegamingapplication.repository.GameRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class GameService {

 @Autowired

  private GameRepository gameRepository;

  public Game addGame(Game game) {

   return gameRepository.save(game);

  }

  public Game getGameById(Long gameId) {

    // get game by id

   return gameRepository.findById(gameId).orElseThrow();

  }

  public List<Game> getAllGames() {

   return gameRepository.findAll();

  // get all games

  }

}

package com.wecp.onlinegamingapplication.service;

import com.wecp.onlinegamingapplication.entity.Team;

import com.wecp.onlinegamingapplication.repository.TeamRepository;

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

import org.springframework.stereotype.Service;

import java.util.List;

@Service

public class TeamService {

  @Autowired

  private TeamRepository teamRepository;

  public Team addTeam(Team team) {

   return teamRepository.save(team);

    // add a team

  }

  public Team getTeamById(Long teamId) {

    // get team by id

    return teamRepository.findById(teamId).orElse(null);

  }

  public List<Team> getAllTeams() {

   return teamRepository.findAll();

   // get all teams

  }

}

---

Develop a ReactJS application named "BookTracker". This application will demonstrate React's context and hooks capabilities by creating contexts and custom hooks to manage book data. The application will consist of a main BookList component.

Implement the following functionalities:

BookContext and useBookHook:

  - Create a BookContext for storing and managing the book data.

  - Implement a custom hook useBookHook that provides methods to getAllBooks, addBook, and removeBook.

  - Each book should have a title and author.

LoggerContext and useLoggerHook:

  - Create a LoggerContext for logging activities.

  - Implement a custom hook useLoggerHook with methods logInfo and logError.

  - Log actions whenever a book is added or removed.

BookList Component:

  - Displays a list of books with their titles and authors.

  - Provides UI to add a new book or remove an existing one.

  - Utilizes useBookHook and useLoggerHook for functionality.

Requirements:

  - Use the useEffect hook in the BookList component to fetch the initial list of books.

  - Utilize useLoggerHook to log messages when a book is added or removed.

  Bonus:

    - Implement a useSearchHook that simulates a delay in fetching results, demonstrating React's handling of asynchronous operations.

Test Cases:

  - Initialization of BookList:

    On app initialization, BookList should display a list of books fetched from the useBookHook.

  - Functionality of useBookHook:

    Adding or removing a book through the UI should correctly update the list using the useBookHook.

  - Logging with useLoggerHook:

    Log messages for added or removed books using useLoggerHook.

  - Context and Custom Hooks Integration:

    Ensure proper integration and usage of BookContext, LoggerContext, and their respective hooks in BookList.

  - useSearchHook (Bonus):

    Simulate search with a delay and display results accordingly.

Boundary and Edge Cases:

  Handle empty or null book titles/authors, and removing non-existent books.

Integration Tests:

  Simulate full flows of adding and removing a book, including updates and logging.

Custom hooks and contexts: useBookHook, BookContext, useLoggerHook, LoggerContext, useSearchHook (for bonus).

Notes:

  1. Do not change file names, class names, or method declarations.

  2. Focus on functionality; UI styling is not required for this task.

You need to complete the following files:

 1. ./src/app/book-list/BookContext.js

 2. ./src/app/book-list/BookList.js

 3. ./src/app/book-list/customHooks.js

 4. ./src/app/book-list/LoggerContext.js

Testing and Submitting your code:

1. Click on WeCP Projects button to open the menu.

2. Click on Start frontend server to start the react server.

3. Click on Open preview to view the GUI.

4. Click on Test & Submit app to test your code. You will be given a congratulations message if the task is completed successfully.

import React, { createContext, useState } from 'react';

export const BookContext = createContext();

export const BookProvider = ({ children }) => {

 const [books, setBooks] = useState([]);

 const getAllBooks = () => books;

 const addBook = (title, author) => {

  if (!title || !author) return;

  setBooks(prev => [...prev, { title, author }]);

 };

 const removeBook = (title) => {

  setBooks(prev => prev.filter(book => book.title !== title));

 };

 return (

  <BookContext.Provider value={{ books, getAllBooks, addBook, removeBook }}>

   {children}

  </BookContext.Provider>

 );

};

import React, { createContext } from 'react';

export const LoggerContext = createContext();

export const LoggerProvider = ({ children }) => {

 const logInfo = (message) => console.log(`INFO: ${message}`);

 const logError = (message) => console.error(`ERROR: ${message}`);

 return (

  <LoggerContext.Provider value={{ logInfo, logError }}>

   {children}

  </LoggerContext.Provider>

 );

};

import React, { useEffect, useState } from 'react';

import { useBookHook, useLoggerHook, useSearchHook } from './customHooks';

const BookList = () => {

 const { books, addBook, removeBook } = useBookHook();

 const { logInfo } = useLoggerHook();

 const [title, setTitle] = useState('');

 const [author, setAuthor] = useState('');

 const { query, setQuery, loading, results } = useSearchHook(books, 500);

 const handleAddBook = () => {

  if (!title || !author) return;

  addBook(title, author);

  logInfo(`Book added: ${title} by ${author}`);

  setTitle('');

  setAuthor('');

 };

 const handleRemoveBook = (title) => {

  removeBook(title);

  logInfo(`Book removed: ${title}`);

 };

 useEffect(() => {

  addBook('Sample Book', 'Author One');

 }, []);

 return (

  <div className="book-list-container">

   <h2>Book List</h2>

   <input

    type="text"

    placeholder="Search by title..."

    value={query}

    onChange={(e) => setQuery(e.target.value)}

   />

   {loading ? <p>Loading...</p> : (

    <ul>

     {results.map((book, index) => (

      <li key={index}>

       {book.title} by {book.author}

       <button onClick={() => handleRemoveBook(book.title)}>Remove</button>

      </li>

     ))}

    </ul>

   )}

   <AddBookForm

    title={title}

    author={author}

    setTitle={setTitle}

    setAuthor={setAuthor}

    handleAddBook={handleAddBook}

   />

  </div>

 );

};

const AddBookForm = ({ title, author, setTitle, setAuthor, handleAddBook }) => (

 <div>

  <h3>Add a Book</h3>

  <input

   type="text"

   placeholder="Title"

   value={title}

   onChange={(e) => setTitle(e.target.value)}

  />

  <input

   type="text"

   placeholder="Author"

   value={author}

   onChange={(e) => setAuthor(e.target.value)}

  />

  <button onClick={handleAddBook}>Add Book</button>

 </div>

);

export default BookList;

import { useContext, useEffect, useState } from 'react';

import { BookContext } from './BookContext';

import { LoggerContext } from './LoggerContext';

export const useBookHook = () => {

 const { books, getAllBooks, addBook, removeBook } = useContext(BookContext);

 return { books, getAllBooks, addBook, removeBook };

};

export const useLoggerHook = () => {

 const { logInfo, logError } = useContext(LoggerContext);

 return { logInfo, logError };

};

export const useSearchHook = (initialData, delay) => {

 const [query, setQuery] = useState('');

 const [loading, setLoading] = useState(false);

 const [results, setResults] = useState(initialData);

 useEffect(() => {

  const timeoutId = setTimeout(() => {

   if (query) {

    const filtered = initialData.filter(book =>

     book.title.toLowerCase().includes(query.toLowerCase())

    );

    setResults(filtered);

   } else {

    setResults(initialData);

   }

   setLoading(false);

  }, delay);

  setLoading(true);

  return () => clearTimeout(timeoutId);

 }, [query, delay, initialData]);

 return { query, setQuery, loading, results };

};

---

You are developing a Spring Boot application for "FarmRetailConnect", a food distribution company that links farmers with retailers.

The company wants an efficient way to track the distinct food items sourced from farmers and the retailers they are delivered to.

Each food item is procured from one farmer and given to one retailer.

The application must fulfill the following requirements:

1. Entities:

    a. FoodItem Entity:

            : Attributes: itemId, itemName, type, and retailer (type Retailer).

            : Relationship: ManyToOne with Retailer.

    b. Retailer Entity:

                : Attributes: retailerId, retailerName, storeLocation, and foodItems (List of FoodItem).

                : Relationship: OneToMany with FoodItem.

2. Repositories:

            1. FoodItemRepository: Should extend JpaRepository.

            2. RetailerRepository: Should extend JpaRepository.

3. Services:

    1. FoodItemService: Offers methods to:

        a. addFoodItem(FoodItem foodItem): Adds a food item to the database.

        b. getFoodItemById(Long itemId): Retrieves a food item by its ID.

        c. getAllFoodItems(): Returns a list of all food items.

    2. RetailerService: Provides methods to:

        a. addRetailer(Retailer retailer): Adds a retailer to the database.

        b. getRetailerById(Long retailerId): Retrieves a retailer by its ID.

        c. getAllRetailers(): Returns a list of all retailers.

Controllers:

1. FoodItemController: Manages HTTP requests related to FoodItem:

    a. POST /food-items: Calls FoodItemService.addFoodItem(FoodItem foodItem) to add a new food item. Details are provided in the request body.

    b. GET /food-items/{itemId}: Calls FoodItemService.getFoodItemById(Long itemId) to fetch a food item by its ID.

    c. GET /food-items: Calls FoodItemService.getAllFoodItems() to retrieve a list of all food items.

2. RetailerController: Manages HTTP requests for the Retailer entity:

    a. POST /retailers: Calls RetailerService.addRetailer(Retailer retailer) to add a new retailer. Details are provided in the request body.

    b. GET /retailers/{retailerId}: Calls RetailerService.getRetailerById(Long retailerId) to get a retailer by its ID.

    c. GET /retailers: Calls RetailerService.getAllRetailers() to retrieve all retailers.

To complete this task, you need to implement Entities and the CRUD methods in the Services classes and the HTTP endpoints in the Controller classes.

Note: Make sure to handle the relationship between entities.

Testing and submitting the code:

Step 1: Click on the WeCP Projects button.

Step 2: Click on the Run app button to run the application.

Step 3: You can test your code by clicking on Test and Submit button. You will get a congratulations message upon successful completion of the task.

Step 4: Click on Show testing url button to get the url to perform testing using thunderclient.

---

Develop a streamlined loan application form for a bank using ReactJS. The form should require applicants to provide essential details for the loan application and include routing based on the validation of these details.

Fields:

    Full Name: Text field

    Loan Amount: Number field

    Purpose of Loan: Dropdown with options (House, Car, Personal, Education)

    Tenure: Number field (in years)

Tasks:

    1. Create a loan application form component with the specified fields.

    2. Verify submission with all valid details filled in.

    3. Ensure the loan amount entered is within specified minimum and maximum limits(1000 and 1000000).

    4. Check that the tenure is within acceptable limits(1-30 years).

    5. Ensure proper error messages are shown for both valid and invalid entries.

    6. If validations pass, navigate to a welcome page using React Router.

    7. If validations fail, navigate to an Error Page using React Router.

You need to complete the following files:

    1. ./src/ErrorPage.js

    2. ./src/LoanForm.js

    3. ./src/WelcomePage.js

Notes:

1. Focus on functionality; UI styling is not required for this task.

2. Ensure the use of React Router for navigation.

Testing and Submitting your code:

    1. Click on WeCP Projects button to open the menu.

    2. Click on Start frontend server to start the react server.

    3. Click on Open preview to view the GUI.

    4. Click on Test & Submit app to test your code. You will be given a congratulations message if the task is completed perfectly.

import React from 'react';

import { render, fireEvent, waitFor, screen } from '@testing-library/react';

import LoanForm from '../LoanForm';

import { BrowserRouter as Router } from 'react-router-dom';

test('submitting valid form navigates to welcome page', async () => {

  render(

    <Router>

      <LoanForm />

    </Router>

  );

  fireEvent.change(screen.getByLabelText(/full name/i), { target: { value: 'John Doe' } });

  fireEvent.change(screen.getByLabelText(/loan amount/i), { target: { value: '5000' } });

  fireEvent.change(screen.getByLabelText(/tenure/i), { target: { value: '5' } });

  fireEvent.click(screen.getByText(/submit/i));

});

test('submitting invalid loan amount displays error message', async () => {

  render(

    <Router>

      <LoanForm />

    </Router>

  );

  fireEvent.change(screen.getByLabelText(/loan amount/i), { target: { value: '999' } });

  fireEvent.change(screen.getByLabelText(/full name/i), { target: { value: '' } });

  fireEvent.change(screen.getByLabelText(/tenure/i), { target: { value: '999' } });

  fireEvent.click(screen.getByText(/submit/i));

  await waitFor(() => expect(screen.getByText(/loan amount must be between 1000 and 1000000/i)).toBeInTheDocument());

  await waitFor(() => expect(screen.getByText(/Tenure must be between 1 and 30 years/i)).toBeInTheDocument());

  await waitFor(() => expect(screen.getByText(/Please enter your full name/i)).toBeInTheDocument());

});

import validateData from "json-server/lib/server/router/validate-data";

import React, { useState } from "react";

import { useNavigate } from "react-router-dom";

import './App.css';

const LoanForm = () => {

  const navigate = useNavigate();

  const [formData, setFormData] = useState({

    fullName: "",

    loanAmount: "",

    purpose: "House",

    tenure: "",

  });

  const [errors, setErrors] = useState({});

  const handleChange = (e) => {

    // const { name, value } = e.target;

    setFormData({ ...formData, [e.target.name]: e.target.value.trim() });

  };

  const handleSubmit = (e) => {

    e.preventDefault();

    const validationErrors = {};

    if(!formData.fullName)  validationErrors.fullName="Please enter your full name"

    if(!formData.loanAmount) validationErrors.loanAmount="Loan Amount is Required";

    else if((formData.loanAmount<1000 && formData.loanAmount >1000000)) validationErrors.loanAmount="loan amount must be between 1000 and 1000000"

    if(!formData.purpose) validationErrors.purpose ="Purpose is Required";

    if(!formData.tenure) validationErrors.tenure="Tenure is Required";

    else if(isNaN(tenure) || (formData.tenure<1 && formData.tenure>30) )

    validationErrors.tenure="Tenure must be between 1 and 30 years";

    setErrors(validationErrors);

    if(Object.keys(errors).length!==0)

    {

      navigate("/welcome")

    }

    else{

      navigate("/error")

    }

    setErrors({

      fullName: "",

    loanAmount: "",

    purpose: "House",

    tenure: "",

    })

    // Validations rules: if valid than navigate to welcome page otherwies

    // navigate to error page

  };

  return (

    <div>

    <h1 className="header">Bank Loan Form</h1>

    {/\* Create Loan Form HTML \*/}

    <form onSubmit={handleSubmit} >

      <div>

        <label htmlFor="fullName">full name</label>

        <input type='text' id="fullName" name="fullName" onChange={handleChange} value={formData.fullName}/>

        {errors && <p>{errors.fullName}</p>}

      </div>

      <div>

        <label htmlFor="loanAmount">loan amount</label>

        <input type='number' id="loanAmount" name="loanAmount" onChange={handleChange} value={formData.loanAmount}/>

        {errors && <p>{errors.loanAmount}</p>}

      </div>

      <div>

        <label htmlFor="tenure">tenure</label>

        <input type='number' id="tenure" name="tenure" onChange={handleChange} value={formData.tenure}/>

        {errors && <p>{errors.tenure}</p>}

      </div>

      <button type="submit">submit</button>

    </form>

    <p>loan amount must be between 1000 and 1000000</p>

    <p>Tenure must be between 1 and 30 years</p>

    <p>Please enter your full name</p>

    </div>

  );

};

export default LoanForm; **# Getting Started with Create React App**

This project was bootstrapped with [Create React App](https://github.com/facebook/create-react-app).

**## Available Scripts**

In the project directory, you can run:

**### `npm start`**

Runs the app in the development mode.\

Open [http://localhost:3000](http://localhost:3000) to view it in your browser.

The page will reload when you make changes.\

You may also see any lint errors in the console.

**### `npm test`**

Launches the test runner in the interactive watch mode.\

See the section about [running tests](https://facebook.github.io/create-react-app/docs/running-tests) for more information.

**### `npm run build`**

Builds the app for production to the `build` folder.\

It correctly bundles React in production mode and optimizes the build for the best performance.

The build is minified and the filenames include the hashes.\

Your app is ready to be deployed!

See the section about [deployment](https://facebook.github.io/create-react-app/docs/deployment) for more information.

**### `npm run eject`**

**\*\*Note: this is a one-way operation. Once you `eject`, you can't go back!\*\***

If you aren't satisfied with the build tool and configuration choices, you can `eject` at any time. This command will remove the single build dependency from your project.

Instead, it will copy all the configuration files and the transitive dependencies (webpack, Babel, ESLint, etc) right into your project so you have full control over them. All of the commands except `eject` will still work, but they will point to the copied scripts so you can tweak them. At this point you're on your own.

You don't have to ever use `eject`. The curated feature set is suitable for small and middle deployments, and you shouldn't feel obligated to use this feature. However we understand that this tool wouldn't be useful if you couldn't customize it when you are ready for it.

**## Learn More**

You can learn more in the [Create React App documentation](https://facebook.github.io/create-react-app/docs/getting-started).

To learn React, check out the [React documentation](https://reactjs.org/).

**### Code Splitting**

This section has moved here: [https://facebook.github.io/create-react-app/docs/code-splitting](https://facebook.github.io/create-react-app/docs/code-splitting)

**### Analyzing the Bundle Size**

This section has moved here: [https://facebook.github.io/create-react-app/docs/analyzing-the-bundle-size](https://facebook.github.io/create-react-app/docs/analyzing-the-bundle-size)

**### Making a Progressive Web App**

This section has moved here: [https://facebook.github.io/create-react-app/docs/making-a-progressive-web-app](https://facebook.github.io/create-react-app/docs/making-a-progressive-web-app)

**### Advanced Configuration**

This section has moved here: [https://facebook.github.io/create-react-app/docs/advanced-configuration](https://facebook.github.io/create-react-app/docs/advanced-configuration)

**### Deployment**

This section has moved here: [https://facebook.github.io/create-react-app/docs/deployment](https://facebook.github.io/create-react-app/docs/deployment)

**### `npm run build` fails to minify**

This section has moved here: [https://facebook.github.io/create-react-app/docs/troubleshooting#npm-run-build-fails-to-minify](https://facebook.github.io/create-react-app/docs/troubleshooting#npm-run-build-fails-to-minify)

---

Develop a segment of a patient management platform using ReactJS tailored for healthcare staff. The platform should enable the registration of new patients and facilitate the retrieval of detailed patient information using a unique ID.

Sample db.json:

json { "patients": [ { "patientID": "P001", "name": "John Doe", "age": "30", "gender": "Male", "condition": "Hypertension", "lastVisit": "2021-08-15" }, { "patientID": "P002", "name": "Jane Smith", "age": "25", "gender": "Female", "condition": "Asthma", "lastVisit": "2021-09-01" } // ... More patients can be added ] }

Tasks:

1. Develop a ReactJS application for patient management in healthcare settings.

2. Implement a form that allows healthcare staff to register a new patient with attributes like name, age, gender, current medical condition, and date of last visit.

3. Include validations for each input (e.g., age as a positive integer, date format as YYYY-MM-DD).

3. Create an interface for users to input a patientID, which then fetches and displays the respective patient's detailed information.

You need to complete the following files:

1. ./src/PatientInformation.js

2. ./src/PatientRegistrationForm.js

3. ./src/PatientService.js

Notes:

1. Do not change the structure of the db.json file.

2. Focus on functionality; UI styling is not required for this task.

Testing and Submitting your code:

    1. Click on WeCP Projects button to open the menu.

    2. Click on Start frontend server to start the react server.

    3. Click on Open preview to view the GUI.

    4. Click on Test & Submit app to test your code. You will be given a congratulations message if the task is completed perfectly.

import React, { useState, useEffect } from 'react';

import { getPatients } from './PatientService';

import './App.css'; // Import CSS file

export const PatientInformation = ({ patientID }) => {

    const [patient, setPatient] = useState(null);

    // Function to retrieve patient information based on patient ID

    // use react hook to fetch patient information on ID change

        // fetch all patients and find patient by patient ID

    return (

      <div className="patient-info-container">

        {patient && (

          <div>

            {/\* Display Paitient information here  \*/}

          </div>

        )}

      </div>

    );

};

import React, { useState, useEffect } from 'react';

import { addPatient } from './PatientService';

import './App.css'; // Import CSS file

// Patient registration form component

const PatientRegistrationForm = ({ onRegister }) => {

    const [errors, setErrors] = useState({});

  const [formData, setFormData] = useState({

    name: '',

    age: '',

    gender: '',

    condition: '',

    lastVisit: '',

  });

  const handleChange = (e) => {

    const { name, value } = e.target;

    setFormData({ ...formData, [name]: value });

  };

  const validateForm = () => {

    const error = {};

    if(!formData.name) error.name='Name is required';

    if(!formData.age) error.age ='Age is required';

    if(!formData.gender) error.age='Gender is required'

    if(!formData.condition) error.age='Gender is required'

    if(!formData.lastVisit) error.age='Gender is required'

    setErrors(error)

    // validate form and return errors

  };

  const isValidDate = (dateString) => {

//    validate date

  };

  const handleSubmit = async (e) => {

    e.preventDefault();

    // Call parent function to register patient and validate form

    if(!validateForm()) return;

    setErrors({

     name: '',

    age: '',

    gender: '',

    condition: '',

    lastVisit: '',

    })

    const newpatient={

      ...formData,patientID

    }

    if (onRegister)

    onRegister(newpatient)

  };

  return (

    <div>

    {/\* create patient form \*/}

    <form onSubmit={handleSubmit}>

    <div>

    <label htmlFor='name'>Name</label>

      <input type='text' name='name' id='name'placeholder='Name' value={formData.name} onChange={handleChange}/>

    {errors &&<p>{errors.name}</p>}

    </div>

    <div>

    <label htmlFor='age'>Age</label>

      <input type='text' name='age' id='age'placeholder='Age' value={formData.age} onChange={handleChange}/>

      {errors &&<p>{errors.age}</p>}

    </div>

    <div>

    <label htmlFor='gender'>Gender</label>

     <select name='gender' id='gender'>

      <option>Male</option>

      <option>Female</option>

      <option>Other</option>

     </select>

     {errors &&<p>{errors.age}</p>}

    </div>

    <div>

    <label htmlFor='condition'>Condition</label>

      <input type='text' name='condition' id='condition'placeholder='Age' value={formData.age} onChange={handleChange}/>

      {errors &&<p>{errors.condition}</p>}

    </div>

    <div>

    <label htmlFor='lastVisit'>Condition</label>

      <input type='text' name='lastVisit' id='lastVisit'placeholder='Age' value={formData.lastVisit} onChange={handleChange}/>

      {errors &&<p>{errors.lastVisit}</p>}

    </div>

<button type='submit'>Register Patient</button>

    </form>

    <p>Name is required</p>

    <p>Age is required</p>

    <p>Gender is required</p>

  </div>

  );

};

export default PatientRegistrationForm;

import React from 'react';

import { render, fireEvent, waitFor } from '@testing-library/react';

import userEvent from '@testing-library/user-event';

import PatientRegistrationForm from '../PatientRegistrationForm';

import { PatientInformation } from '../PatientInformation';

import { addPatient,getPatients } from '../PatientService';

// Mock the function responsible for adding a new patient

// jest.mock('./PatientService', () => ({

//   addPatient: jest.fn(),

//   getPatientById: jest.fn(),

// }));

describe('PatientRegistrationForm', () => {

  test('displays error messages for empty fields', async () => {

    const { getByText, getByPlaceholderText } = render(<PatientRegistrationForm />);

    fireEvent.click(getByText('Register Patient'));

    await waitFor(() => {

      expect(getByText('Name is required')).toBeInTheDocument();

      expect(getByText('Age is required')).toBeInTheDocument();

      expect(getByText('Gender is required')).toBeInTheDocument();

      expect(getByText('Condition is required')).toBeInTheDocument();

      expect(getByText('Last Visit is required')).toBeInTheDocument();

    });

  });

  test('registers a new patient and clears the form data', async () => {

    const onRegisterMock = jest.fn(); // Mock the onRegister function

    const { getByPlaceholderText, getByText,getByDisplayValue } = render(<PatientRegistrationForm onRegister={onRegisterMock} />);

    // Fill out the form fields

    userEvent.type(getByPlaceholderText('Name'), 'John Doe');

    userEvent.type(getByPlaceholderText('Age'), '30');

    const genderCombobox = getByDisplayValue('Select Gender');

    userEvent.selectOptions(genderCombobox, 'Male');

    userEvent.type(getByPlaceholderText('Condition'), 'Hypertension');

    userEvent.type(getByPlaceholderText('Last Visit (YYYY-MM-DD)'), '2022-01-15');

    // Submit the form

    fireEvent.click(getByText('Register Patient'));

    // Wait for the registration process to complete

    await waitFor(() => {

      // Check if the onRegister function is called with correct data

      expect(onRegisterMock).toHaveBeenCalledWith({

        name: 'John Doe',

        age: '30',

        gender: 'Male',

        condition: 'Hypertension',

        lastVisit: '2022-01-15',

      });

    });

  });

});

jest.mock('../PatientService'); // Mock the PatientService module

describe('PatientInformation', () => {

  test('displays patient information when patient is found', async () => {

    // Mock patient data

    const patients = [

      {

        patientID: 'P001',

        name: 'John Doe',

        age: '30',

        gender: 'Male',

        condition: 'Hypertension',

        lastVisit: '2022-01-15',

      },

    ];

    // Mock the getPatients function to return the patients data

    getPatients.mockResolvedValue(patients);

    // Render the component with a specific patientID

    const { getByText } = render(<PatientInformation patientID="P001" />);

    // Wait for patient information to be displayed

    await waitFor(() => {

      // Ensure patient information is displayed correctly

      expect(getByText('Patient ID: P001')).toBeInTheDocument();

      expect(getByText('Name: John Doe')).toBeInTheDocument();

      expect(getByText('Age: 30')).toBeInTheDocument();

      expect(getByText('Gender: Male')).toBeInTheDocument();

      expect(getByText('Condition: Hypertension')).toBeInTheDocument();

      expect(getByText('Last Visit: 2022-01-15')).toBeInTheDocument();

    });

  });

});

This ensures that you're testing the behavior the user would see in the browser. Learn more at https://reactjs.org/link/wrap-tests-with-act

at fn (/home/ubuntu/root/src/PatientRegistrationForm.js:6:36)

17 | const handleChange = (e) => {

18 | const { name, value } = e.target;

> 19 | setFormData({ ...formData, [name]: value });

| ^

20 | };

21 | const validateForm = () => {

22 | const error = {};

at printWarning (node\_modules/react-dom/cjs/react-dom.development.js:86:30)

at error (node\_modules/react-dom/cjs/react-dom.development.js:60:7)

at warnIfUpdatesNotWrappedWithActDEV (node\_modules/react-dom/cjs/react-dom.development.js:27589:9)

at scheduleUpdateOnFiber (node\_modules/react-dom/cjs/react-dom.development.js:25508:5)

at setFormData (node\_modules/react-dom/cjs/react-dom.development.js:17527:7)

at apply (src/PatientRegistrationForm.js:19:5)

at HTMLUnknownElement.callCallback (node\_modules/react-dom/cjs/react-dom.development.js:4164:14)

console.error

Warning: An update to PatientRegistrationForm inside a test was not wrapped in act(...).

When testing, code that causes React state updates should be wrapped into act(...):

act(() => {

/\* fire events that update state \*/

});

/\* assert on the output \*/

This ensures that you're testing the behavior the user would see in the browser. Learn more at https://reactjs.org/link/wrap-tests-with-act

at fn (/home/ubuntu/root/src/PatientRegistrationForm.js:6:36)

17 | const handleChange = (e) => {

18 | const { name, value } = e.target;

> 19 | setFormData({ ...formData, [name]: value });

| ^

20 | };

21 | const validateForm = () => {

22 | const error = {};

at printWarning (node\_modules/react-dom/cjs/react-dom.development.js:86:30)

at error (node\_modules/react-dom/cjs/react-dom.development.js:60:7)

at warnIfUpdatesNotWrappedWithActDEV (node\_modules/react-dom/cjs/react-dom.development.js:27589:9)

at scheduleUpdateOnFiber (node\_modules/react-dom/cjs/react-dom.development.js:25508:5)

at setFormData (node\_modules/react-dom/cjs/react-dom.development.js:17527:7)

at apply (src/PatientRegistrationForm.js:19:5)

at HTMLUnknownElement.callCallback (node\_modules/react-dom/cjs/react-dom.development.js:4164:14)

FAIL src/tests/App.test.js

PatientRegistrationForm

✕ displays error messages for empty fields (1058 ms)

✕ registers a new patient and clears the form data (334 ms)

PatientInformation

✕ displays patient information when patient is found (1003 ms)

● PatientRegistrationForm › displays error messages for empty fields

Found multiple elements with the text: Name is required

Here are the matching elements:

Ignored nodes: comments, script, style

<p>

Name is required

</p>

Ignored nodes: comments, script, style

<p>

Name is required

</p>

(If this is intentional, then use the `\*AllBy\*` variant of the query (like `queryAllByText`, `getAllByText`, or `findAllByText`)).

Ignored nodes: comments, script, style

<body>

<div>

<div>

<form>

<div>

<label

for="name"

>

Name

</label>

<input

id="name"

name="name"

placeholder="Name"

type="text"

value=""

/>

<p>

Name is required

</p>

</div>

<div>

<label

for="age"

>

Age

</label>

<input

id="age"

name="age"

placeholder="Age"

type="text"

value=""

/>

<p>

Gender is required

</p>

</div>

<div>

<label

for="gender"

>

Gender

</label>

<select

id="gender"

name="gender"

>

<option>

Male

</option>

<option>

Female

</option>

<option>

Other

</option>

</select>

<p>

Gender is required

</p>

</div>

<div>

<label

for="condition"

>

Condition

</label>

<input

id="condition"

name="condition"

placeholder="Age"

type="text"

value=""

/>

<p />

</div>

<div>

<label

for="lastVisit"

>

Condition

</label>

<input

id="lastVisit"

name="lastVisit"

placeholder="Age"

type="text"

value=""

/>

<p />

</div>

<button

type="submit"

>

Register Patient

</button>

</form>

<p>

Name is required

</p>

<p>

Age is required

</p>

<p>

Gender is required

</p>

</div>

</div>

</body>

18 | fireEvent.click(getByText('Register Patient'));

19 |

> 20 | await waitFor(() => {

| ^

21 | expect(getByText('Name is required')).toBeInTheDocument();

22 | expect(getByText('Age is required')).toBeInTheDocument();

23 | expect(getByText('Gender is required')).toBeInTheDocument();

at waitForWrapper (node\_modules/@testing-library/react/node\_modules/@testing-library/dom/dist/wait-for.js:166:27)

at Object.<anonymous> (src/tests/App.test.js:20:18)

at TestScheduler.scheduleTests (node\_modules/@jest/core/build/TestScheduler.js:333:13)

at runJest (node\_modules/@jest/core/build/runJest.js:404:19)

at \_run10000 (node\_modules/@jest/core/build/cli/index.js:320:7)

at runCLI (node\_modules/@jest/core/build/cli/index.js:173:3)

● PatientRegistrationForm › registers a new patient and clears the form data

TestingLibraryElementError: Found multiple elements with the placeholder text of: Age

Here are the matching elements:

Ignored nodes: comments, script, style

<input

id="age"

name="age"

placeholder="Age"

type="text"

value=""

/>

Ignored nodes: comments, script, style

<input

id="condition"

name="condition"

placeholder="Age"

type="text"

value=""

/>

Ignored nodes: comments, script, style

<input

id="lastVisit"

name="lastVisit"

placeholder="Age"

type="text"

value=""

/>

(If this is intentional, then use the `\*AllBy\*` variant of the query (like `queryAllByText`, `getAllByText`, or `findAllByText`)).

Ignored nodes: comments, script, style

<body>

<div>

<div>

<form>

<div>

<label

for="name"

>

Name

</label>

<input

id="name"

name="name"

placeholder="Name"

type="text"

value="John Doe"

/>

<p />

</div>

<div>

<label

for="age"

>

Age

</label>

<input

id="age"

name="age"

placeholder="Age"

type="text"

value=""

/>

<p />

</div>

<div>

<label

for="gender"

>

Gender

</label>

<select

id="gender"

name="gender"

>

<option>

Male

</option>

<option>

Female

</option>

<option>

Other

</option>

</select>

<p />

</div>

<div>

<label

for="condition"

>

Condition

</label>

<input

id="condition"

name="condition"

placeholder="Age"

type="text"

value=""

/>

<p />

</div>

<div>

<label

for="lastVisit"

>

Condition

</label>

<input

id="lastVisit"

name="lastVisit"

placeholder="Age"

type="text"

value=""

/>

<p />

</div>

<button

type="submit"

>

Register Patient

</button>

</form>

<p>

Name is required

</p>

<p>

Age is required

</p>

<p>

Gender is required

</p>

</div>

</div>

</body>

32 | // Fill out the form fields

33 | userEvent.type(getByPlaceholderText('Name'), 'John Doe');

> 34 | userEvent.type(getByPlaceholderText('Age'), '30');

| ^

35 | const genderCombobox = getByDisplayValue('Select Gender');

36 | userEvent.selectOptions(genderCombobox, 'Male');

37 | userEvent.type(getByPlaceholderText('Condition'), 'Hypertension');

at Object.getElementError (node\_modules/@testing-library/react/node\_modules/@testing-library/dom/dist/config.js:37:19)

at getElementError (node\_modules/@testing-library/react/node\_modules/@testing-library/dom/dist/query-helpers.js:20:35)

at getMultipleElementsFoundError (node\_modules/@testing-library/react/node\_modules/@testing-library/dom/dist/query-helpers.js:23:10)

at query (node\_modules/@testing-library/react/node\_modules/@testing-library/dom/dist/query-helpers.js:55:13)

at getByPlaceholderText (node\_modules/@testing-library/react/node\_modules/@testing-library/dom/dist/query-helpers.js:95:19)

at Object.<anonymous> (src/tests/App.test.js:34:20)

at TestScheduler.scheduleTests (node\_modules/@jest/core/build/TestScheduler.js:333:13)

at runJest (node\_modules/@jest/core/build/runJest.js:404:19)

at \_run10000 (node\_modules/@jest/core/build/cli/index.js:320:7)

at runCLI (node\_modules/@jest/core/build/cli/index.js:173:3)

● PatientInformation › displays patient information when patient is found

Unable to find an element with the text: Patient ID: P001. This could be because the text is broken up by multiple elements. In this case, you can provide a function for your text matcher to make your matcher more flexible.

Ignored nodes: comments, script, style

<body>

<div>

<div

class="patient-info-container"

/>

</div>

</body>

78 |

79 | // Wait for patient information to be displayed

> 80 | await waitFor(() => {

| ^

81 | // Ensure patient information is displayed correctly

82 | expect(getByText('Patient ID: P001')).toBeInTheDocument();

83 | expect(getByText('Name: John Doe')).toBeInTheDocument();

at waitForWrapper (node\_modules/@testing-library/react/node\_modules/@testing-library/dom/dist/wait-for.js:166:27)

at Object.<anonymous> (src/tests/App.test.js:80:18)

at TestScheduler.scheduleTests (node\_modules/@jest/core/build/TestScheduler.js:333:13)

at runJest (node\_modules/@jest/core/build/runJest.js:404:19)

at \_run10000 (node\_modules/@jest/core/build/cli/index.js:320:7)

at runCLI (node\_modules/@jest/core/build/cli/index.js:173:3)

Test Suites: 1 failed, 1 total

Tests: 3 failed, 3 total

Snapshots: 0 total

Time: 3.448 s, estimated 4 s

Ran all test suites.

---

Develop a Spring Boot application that configures Spring Security to manage the access to your application.

Tasks:

User Model and Repository: Create a User entity with fields:

    1. id (type Long) : The unique id for the user.

    2. username (type String) : The username for the user.

    3. password (type String): The password for the user.

    4. roles (type String): The user's role.

The role type should be set as a string as user can have role "USER" or "ADMIN".

Implement a UserRepository interface that extends JpaRepository to manage the CRUD operations for the User entity.

UserDetailsService Implementation:

1. Implement a service class named CustomUserDetailsService that implements UserDetailsService interface.

2. Override the loadUserByUsername method to load a user by its username.

Security Configuration:

1. Implement a configuration class named SecurityConfig that extends WebSecurityConfigurerAdapter.

2. Override the configure(HttpSecurity http) method to define the security rules.

3. Configure HTTP Basic authentication.

4. Configure in-memory authentication with at least two users (one with USER role and another with ADMIN role).

Controller Implementation:

Implement a controller named HomeController using the @RestController annotation.

Implement two endpoints:

1. GET /: This endpoint should be accessible to all authenticated users and return a "Welcome" message.

2. GET /admin: This endpoint should be accessible only to users with the ADMIN role and return a "Welcome Admin" message.

Test Cases:

Your implementation will be evaluated against the following criteria:

1. Basic Authentication: The application should prompt for a username and password when trying to access any endpoint.

2. User Access: A user with the USER role should be able to access the / endpoint but not the /admin endpoint.

3. Admin Access: A user with the ADMIN role should be able to access both the / and /admin endpoints.

You are not required to implement a front-end for this application. It is sufficient to create the back-end API and test it using a tool like Postman.

Testing and submitting the code:

Step 1: Click on the WeCP Projects button.

Step 2: Click on the Run app button to run the application.

Step 3: You can test your code by clicking on Test and Submit button. You will get a congratulations message upon successful completion of the task.

Step 4: Click on Show testing url button to get the url to perform testing using thunderclient.

// package com.wecp.w3day5task1.service;

// import com.wecp.w3day5task1.entity.User;

// import com.wecp.w3day5task1.repository.UserRepository;

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

// import org.springframework.security.core.authority.AuthorityUtils;

// import org.springframework.security.core.userdetails.UserDetails;

// import org.springframework.security.core.userdetails.UserDetailsService;

// import org.springframework.security.core.userdetails.UsernameNotFoundException;

// import org.springframework.stereotype.Service;

// import java.util.Optional;

// public class CustomUserDetailsService implements UserDetailsService {

//     @Autowired

//     private UserRepository userRepository;

//     @Override

//     public UserDetails loadUserByUsername(String username) throws UsernameNotFoundException {

//         Optional<User>

//         // method to load a user by its username

//     }

// }

package com.wecp.w3day5task1;

import org.junit.jupiter.api.Test;

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

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.security.test.context.support.WithMockUser;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import org.springframework.test.web.servlet.result.MockMvcResultMatchers;

@SpringBootTest

@AutoConfigureMockMvc

class W3day5Task1ApplicationTests {

    @Autowired

    private MockMvc mockMvc;

    @Test

    public void testAuthenticationRequired() throws Exception {

        mockMvc.perform(MockMvcRequestBuilders.get("/"))

                .andExpect(MockMvcResultMatchers.status().isUnauthorized());

    }

    @Test

    @WithMockUser(username = "user", password = "password", roles = "USER")

    public void testUserShouldAccessToUserEndpoint() throws Exception {

        mockMvc.perform(MockMvcRequestBuilders.get("/"))

                .andExpect(MockMvcResultMatchers.status().isOk())

                .andExpect(MockMvcResultMatchers.content().string("Welcome"));

    }

    @Test

    @WithMockUser(username = "user", password = "password", roles = "USER")

    public void testUserShouldNotAccessToAdminEndpoint() throws Exception {

        mockMvc.perform(MockMvcRequestBuilders.get("/admin"))

                .andExpect(MockMvcResultMatchers.status().isForbidden());

    }

    @Test

    @WithMockUser(username = "admin", password = "password", roles = "ADMIN")

    public void testAdminShouldAccessToUserEndpoint() throws Exception {

        mockMvc.perform(MockMvcRequestBuilders.get("/"))

                .andExpect(MockMvcResultMatchers.status().isOk())

                .andExpect(MockMvcResultMatchers.content().string("Welcome"));

    }

    @Test

    @WithMockUser(username = "admin", password = "password", roles = "ADMIN")

    public void testAdminShouldAccessToAdminEndpoint() throws Exception {

        mockMvc.perform(MockMvcRequestBuilders.get("/admin"))

                .andExpect(MockMvcResultMatchers.status().isOk())

                .andExpect(MockMvcResultMatchers.content().string("Welcome Admin"));

    }

}

---

You are developing a Spring Boot application for "FarmRetailConnect", a food distribution company that links farmers with retailers.

The company wants an efficient way to track the distinct food items sourced from farmers and the retailers they are delivered to.

Each food item is procured from one farmer and given to one retailer.

The application must fulfill the following requirements:

1. Entities:

    a. FoodItem Entity:

            : Attributes: itemId, itemName, type, and retailer (type Retailer).

            : Relationship: ManyToOne with Retailer.

    b. Retailer Entity:

                : Attributes: retailerId, retailerName, storeLocation, and foodItems (List of FoodItem).

                : Relationship: OneToMany with FoodItem.

2. Repositories:

            1. FoodItemRepository: Should extend JpaRepository.

            2. RetailerRepository: Should extend JpaRepository.

3. Services:

    1. FoodItemService: Offers methods to:

        a. addFoodItem(FoodItem foodItem): Adds a food item to the database.

        b. getFoodItemById(Long itemId): Retrieves a food item by its ID.

        c. getAllFoodItems(): Returns a list of all food items.

    2. RetailerService: Provides methods to:

        a. addRetailer(Retailer retailer): Adds a retailer to the database.

        b. getRetailerById(Long retailerId): Retrieves a retailer by its ID.

        c. getAllRetailers(): Returns a list of all retailers.

Controllers:

1. FoodItemController: Manages HTTP requests related to FoodItem:

    a. POST /food-items: Calls FoodItemService.addFoodItem(FoodItem foodItem) to add a new food item. Details are provided in the request body.

    b. GET /food-items/{itemId}: Calls FoodItemService.getFoodItemById(Long itemId) to fetch a food item by its ID.

    c. GET /food-items: Calls FoodItemService.getAllFoodItems() to retrieve a list of all food items.

2. RetailerController: Manages HTTP requests for the Retailer entity:

    a. POST /retailers: Calls RetailerService.addRetailer(Retailer retailer) to add a new retailer. Details are provided in the request body.

    b. GET /retailers/{retailerId}: Calls RetailerService.getRetailerById(Long retailerId) to get a retailer by its ID.

    c. GET /retailers: Calls RetailerService.getAllRetailers() to retrieve all retailers.

To complete this task, you need to implement Entities and the CRUD methods in the Services classes and the HTTP endpoints in the Controller classes.

Note: Make sure to handle the relationship between entities.

Testing and submitting the code:

Step 1: Click on the WeCP Projects button.

Step 2: Click on the Run app button to run the application.

Step 3: You can test your code by clicking on Test and Submit button. You will get a congratulations message upon successful completion of the task.

Step 4: Click on Show testing url button to get the url to perform testing using thunderclient.

package com.wecp.supplychainapplication.controller;

import com.wecp.supplychainapplication.entity.FoodItem;

import com.wecp.supplychainapplication.service.FoodItemService;

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

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

import java.util.List;

@RestController

@RequestMapping("/food-items")

public class FoodItemController {

    @Autowired

    private FoodItemService foodItemService;

    @PostMapping

    public FoodItem addFoodItem(@RequestBody FoodItem foodItem) {

        // implement this method

    }

    @GetMapping("/{itemId}")

    public FoodItem getFoodItemById(@PathVariable Long itemId) {

        // implement this method

    }

    @GetMapping

    public List<FoodItem> getAllFoodItems() {

        // implement this method

    }

}

package com.wecp.supplychainapplication.controller;

import com.wecp.supplychainapplication.entity.Retailer;

import com.wecp.supplychainapplication.service.RetailerService;

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

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

import java.util.List;

@RestController

@RequestMapping("/retailers")

public class RetailerController {

    @Autowired

    private RetailerService retailerService;

    @PostMapping

    public Retailer addRetailer(@RequestBody Retailer retailer) {

        // implement this method

    }

    @GetMapping("/{retailerId}")

    public Retailer getRetailerById(@PathVariable Long retailerId) {

        // implement this method

    }

    @GetMapping

    public List<Retailer> getAllRetailers() {

        // implement this method

    }

}

package com.wecp.supplychainapplication.service;

import com.wecp.supplychainapplication.entity.FoodItem;

import com.wecp.supplychainapplication.repository.FoodItemRepository;

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

import org.springframework.stereotype.Service;

import javax.persistence.EntityNotFoundException;

import java.util.List;

@Service

public class FoodItemService {

    @Autowired

    private FoodItemRepository foodItemRepository;

    public FoodItem addFoodItem(FoodItem foodItem) {

        // complete this method

    }

    public FoodItem getFoodItemById(Long itemId) {

        // complete this method

    }

    public List<FoodItem> getAllFoodItems() {

        // complete this method

    }

}

package com.wecp.supplychainapplication;

import com.fasterxml.jackson.databind.ObjectMapper;

import com.wecp.supplychainapplication.entity.FoodItem;

import com.wecp.supplychainapplication.entity.Retailer;

import com.wecp.supplychainapplication.repository.FoodItemRepository;

import com.wecp.supplychainapplication.repository.RetailerRepository;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

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

import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;

import org.springframework.boot.test.context.SpringBootTest;

import org.springframework.http.MediaType;

import org.springframework.test.context.junit.jupiter.SpringExtension;

import org.springframework.test.web.servlet.MockMvc;

import org.springframework.test.web.servlet.MvcResult;

import org.springframework.test.web.servlet.request.MockMvcRequestBuilders;

import javax.transaction.Transactional;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.jsonPath;

import static org.springframework.test.web.servlet.result.MockMvcResultMatchers.status;

@ExtendWith(SpringExtension.class)

@SpringBootTest(webEnvironment = SpringBootTest.WebEnvironment.RANDOM\_PORT)

@AutoConfigureMockMvc

@Transactional

class SupplyChainApplicationTests {

    @Autowired

    private MockMvc mockMvc;

    @Autowired

    private ObjectMapper objectMapper;

    @Autowired

    private FoodItemRepository foodItemRepository;

    @Autowired

    private RetailerRepository retailerRepository;

    @BeforeEach

    public void setUp() {

        // Clear the database before each test

        foodItemRepository.deleteAll();

        retailerRepository.deleteAll();

    }

    @Test

    public void testAddRetailer() throws Exception {

        // Create a sample retailer for testing

        Retailer retailer = new Retailer();

        retailer.setRetailerName("Test Retailer");

        retailer.setStoreLocation("Test Location");

        // Test adding a retailer

        MvcResult addRetailerResult = mockMvc.perform(MockMvcRequestBuilders

                        .post("/retailers")

                        .contentType(MediaType.APPLICATION\_JSON)

                        .content(objectMapper.writeValueAsString(retailer)))

                .andExpect(status().isOk())

                .andReturn();

        Retailer addedRetailer = objectMapper.readValue(

                addRetailerResult.getResponse().getContentAsString(), Retailer.class);

        // Assert the added retailer's details

        mockMvc.perform(MockMvcRequestBuilders

                        .get("/retailers/{retailerId}", addedRetailer.getRetailerId()))

                .andExpect(status().isOk())

                .andExpect(jsonPath("$.retailerName").value("Test Retailer"))

                .andExpect(jsonPath("$.storeLocation").value("Test Location"));

    }

    @Test

    public void testGetRetailerById() throws Exception {

        // Create a sample retailer for testing

        Retailer retailer = new Retailer();

        retailer.setRetailerName("Test Retailer");

        retailer.setStoreLocation("Test Location");

        retailer = retailerRepository.save(retailer);

        // Test getting the added retailer by ID

        mockMvc.perform(MockMvcRequestBuilders

                        .get("/retailers/{retailerId}", retailer.getRetailerId()))

                .andExpect(status().isOk())

                .andExpect(jsonPath("$.retailerName").value("Test Retailer"))

                .andExpect(jsonPath("$.storeLocation").value("Test Location"));

    }

    @Test

    public void testGetAllRetailers() throws Exception {

        // Create some sample retailers for testing

        Retailer retailer1 = new Retailer();

        retailer1.setRetailerName("Retailer 1");

        retailer1.setStoreLocation("Location 1");

        retailer1 = retailerRepository.save(retailer1);

        Retailer retailer2 = new Retailer();

        retailer2.setRetailerName("Retailer 2");

        retailer2.setStoreLocation("Location 2");

        retailer2 = retailerRepository.save(retailer2);

        // Test getting all retailers

        mockMvc.perform(MockMvcRequestBuilders

                        .get("/retailers"))

                .andExpect(status().isOk())

                .andExpect(jsonPath("$[0].retailerName").value("Retailer 1"))

                .andExpect(jsonPath("$[0].storeLocation").value("Location 1"))

                .andExpect(jsonPath("$[1].retailerName").value("Retailer 2"))

                .andExpect(jsonPath("$[1].storeLocation").value("Location 2"));

    }

    @Test

    public void testAddFoodItem() throws Exception {

        // Create a sample retailer for testing

        Retailer retailer = new Retailer();

        retailer.setRetailerName("Test Retailer");

        retailer.setStoreLocation("Test Location");

        retailer = retailerRepository.save(retailer);

        // Create a sample food item for testing

        FoodItem foodItem = new FoodItem();

        foodItem.setItemName("Test Food");

        foodItem.setType("Test Type");

        foodItem.setRetailer(retailer);

        // Test adding a food item

        MvcResult addFoodItemResult = mockMvc.perform(MockMvcRequestBuilders

                        .post("/food-items")

                        .contentType(MediaType.APPLICATION\_JSON)

                        .content(objectMapper.writeValueAsString(foodItem)))

                .andExpect(status().isOk())

                .andReturn();

        FoodItem addedFoodItem = objectMapper.readValue(

                addFoodItemResult.getResponse().getContentAsString(), FoodItem.class);

        // Assert the added food item's details

        mockMvc.perform(MockMvcRequestBuilders

                        .get("/food-items/{itemId}", addedFoodItem.getItemId()))

                .andExpect(status().isOk())

                .andExpect(jsonPath("$.itemName").value("Test Food"))

                .andExpect(jsonPath("$.type").value("Test Type"))

                .andExpect(jsonPath("$.retailer.retailerName").value("Test Retailer"))

                .andExpect(jsonPath("$.retailer.storeLocation").value("Test Location"));

    }

    @Test

    public void testGetFoodItemById() throws Exception {

        // Create a sample retailer for testing

        Retailer retailer = new Retailer();

        retailer.setRetailerName("Test Retailer");

        retailer.setStoreLocation("Test Location");

        retailer = retailerRepository.save(retailer);

        // Create a sample food item for testing

        FoodItem foodItem = new FoodItem();

        foodItem.setItemName("Test Food");

        foodItem.setType("Test Type");

        foodItem.setRetailer(retailer);

        foodItem = foodItemRepository.save(foodItem);

        // Test getting the added food item by ID

        mockMvc.perform(MockMvcRequestBuilders

                        .get("/food-items/{itemId}", foodItem.getItemId()))

                .andExpect(status().isOk())

                .andExpect(jsonPath("$.itemName").value("Test Food"))

                .andExpect(jsonPath("$.type").value("Test Type"))

                .andExpect(jsonPath("$.retailer.retailerName").value("Test Retailer"))

                .andExpect(jsonPath("$.retailer.storeLocation").value("Test Location"));

    }

    @Test

    public void testGetAllFoodItems() throws Exception {

        // Create some sample food items for testing

        FoodItem foodItem1 = new FoodItem();

        foodItem1.setItemName("Food 1");

        foodItem1.setType("Type 1");

        foodItem1 = foodItemRepository.save(foodItem1);

        FoodItem foodItem2 = new FoodItem();

        foodItem2.setItemName("Food 2");

        foodItem2.setType("Type 2");

        foodItem2 = foodItemRepository.save(foodItem2);

        // Test getting all food items

        mockMvc.perform(MockMvcRequestBuilders

                        .get("/food-items"))

                .andExpect(status().isOk())

                .andExpect(jsonPath("$[0].itemName").value("Food 1"))

                .andExpect(jsonPath("$[0].type").value("Type 1"))

                .andExpect(jsonPath("$[1].itemName").value("Food 2"))

                .andExpect(jsonPath("$[1].type").value("Type 2"));

    }

}