**[Build a Library Management Application in Spring Boot (Beginner Level)**

The government is building a centralized library management system that involves a micro-service, allowing people to issue books and deposit them. This micro-service is built using the Spring Boot framework and connects to the database using the Java Persistence API (JPA).

The application exposes the following four APIs:

1. /api/v1/user [POST]: An API to Add a new user.
2. /api/v1/user [GET]: An API to Fetch the details of a user with id.
3. /api/v1/book [POST]: An API to Add a new user.
4. /api/v1/book/{id} [GET]: An API to Fetch the details of a book.

**Your task is to complete the files:**

* src/main/java/com/wecp/library/controller/LibraryController.java
* src/main/java/com/wecp/library/repository/BookRepository.java
* src/main/java/com/wecp/library/repository/UserRepository.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

**Notes:**

* Implementation related specifics are added directly as javadocs in the workspace.
* Ensure that the structure and datatypes of the APIs are followed as specified in the comments to ensure that the code is evaluated correctly.
* package com.wecp.library.repository;
* import com.wecp.library.domain.Book;
* import org.springframework.data.jpa.repository.JpaRepository;
* public interface BookRepository extends JpaRepository<Book,Long> {
* }
* package com.wecp.library.repository;
* import com.wecp.library.domain.User;
* import org.springframework.data.jpa.repository.JpaRepository;
* public interface UserRepository extends JpaRepository<User,Long>{
* }
* 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.beans.factory.annotation.Autowired;
* 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 {
* @Autowired
* private BookRepository br;
* @Autowired
* private UserRepository ur;
* /\*\*
* \* {@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) {
* Optional<User> user = ur.findById(id);
* if(user.isPresent()){
* return ResponseEntity.ok().body(ur.findById(id).orElse(null));
* }
* 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) {
* ur.save(user);
* return ResponseEntity.ok().body(user);
* }
* /\*\*
* \* {@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) {
* Optional<Book> book = br.findById(id);
* if(book.isPresent()){
* return ResponseEntity.ok().body(br.findById(id).orElse(null));
* }
* 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) {
* br.save(book);
* return ResponseEntity.ok().body(book);
* }
* }

**Build a Library Management Application in Spring Boot (Intermediate Level)**

**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). This micro-service should be built using the Spring Boot REST API framework and connect to the database using the JPA API.
4. "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 the exception **"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 file:**

* src/main/java/com/wecp/library/controller/LibraryController.java
* src/main/java/com/wecp/library/repository/BookRepository.java
* src/main/java/com/wecp/library/repository/IssueRepository.java

**You can find blank jpa repositories here:**

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

package com.wecp.library.repository;

import com.wecp.library.domain.Book;

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

import org.springframework.stereotype.Repository;

/\*\*

\* Spring Data repository for the Book entity. Use correct annotation and extend the class with necessary Spring Data class

 \*/

public interface BookRepository extends JpaRepository<Book,Long>{

}

package com.wecp.library.repository;

import com.wecp.library.domain.Issue;

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

import org.springframework.stereotype.Repository;

/\*\*

 \* Spring Data repository for the Issue entity. Use correct annotation and extend the class with necessary Spring Data class

 \*/

public interface IssueRepository extends JpaRepository<Issue,Long> {

}

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.temporal.ChronoUnit;

import java.util.Optional;

/\*\*

 \* REST controller for managing library system process

 \*/

@RestController

@RequestMapping("/api/v1")

public class LibraryController {

    //private final BookRepository bookRepository;

    @Autowired

    private BookRepository bookRepository;

    //private final IssueRepository issueRepository;

    @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) {

        Optional<Book> book=bookRepository.findById(issue.getBook().getId());

        if(book.get().getQuantity()==0){

            throw new BookNotAvailableException();

        }

        return ResponseEntity.ok().body(issueRepository.save(issue));

    }

    /\*\*

     \* {@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) {

        Long day = Math.abs(ChronoUnit.DAYS.between(issue.getIssueDate(),issue.getReturnDate()));

        if(day>issue.getPeriod()){

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

        }

        return Mono.just(0);

    }

}

**Build a Library Management Application in Spring Boot (Advanced Level)**

**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 three APIs for which the details are given below:

* /api/v1/create-user [POST]: Simple user save method in an authenticated manner
* /api/v1/issue-book [POST]: Send issue, check if users subscribed(see subscribed field in User entity), otherwise throw SubscriptionExpiredException- permitAll
* /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:**

* Implementation related specifics are added directly as javadocs in the workspace.
* Ensure that the structure and datatypes of the APIs are followed as specified in the comments to ensure that the code is evaluated correctly.
* package com.wecp.library.controller.exception;
* import org.springframework.http.HttpStatus;
* import org.springframework.web.bind.annotation.ResponseStatus;
* @ResponseStatus(HttpStatus.FORBIDDEN)
* public class UserNotSubscribedException extends RuntimeException {
* public UserNotSubscribedException(String msg){
* super(msg);
* }
* }

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.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> user=userRepository.findById((issue.getUser().getId()));

        if(user.isEmpty())

        {

            return ResponseEntity.noContent().build();

        }

        if(user.get().getSubscribed()==false)

        {

            throw new UserNotSubscribedException("user not subscribed");

        }

        return ResponseEntity.ok().body(issueRepository.save(issue));

    }

    /\*\*

     \* {@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(userRepository.save(user));

    }

    /\*\*

     \* {@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> user = userRepository.findById(id);

        if(user.isEmpty())

        {

            return ResponseEntity.noContent().build();

        }

        user.get().setSubscribed(true);

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

    }

}

package com.wecp.library.repository;

import com.wecp.library.domain.Issue;

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

import org.springframework.stereotype.Repository;

public interface IssueRepository extends JpaRepository<Issue,Long>{

}

package com.wecp.library.repository;

import com.wecp.library.domain.User;

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

import org.springframework.stereotype.Repository;

// public interface UserRepository

public interface UserRepository extends JpaRepository<User,Long>{

}

//////// package com.wecp.library.security;

import org.springframework.context.annotation.Configuration;

import org.springframework.http.HttpStatus;

import org.springframework.security.authentication.dao.DaoAuthenticationProvider;

import org.springframework.security.config.annotation.authentication.builders.AuthenticationManagerBuilder;

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.authentication.HttpStatusEntryPoint;

/\*\*

 \* 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 {

    @Override

    public void configure(AuthenticationManagerBuilder auth) throws Exception{

        auth.authenticationProvider(new DaoAuthenticationProvider());

    }

    @Override

    public void configure(HttpSecurity http) throws Exception{

        http.authorizeRequests().antMatchers("api/v1/renew-user-subscription/\*\*").permitAll().anyRequest().authenticated()

        .and().exceptionHandling().authenticationEntryPoint(new HttpStatusEntryPoint(HttpStatus.UNAUTHORIZED));

    }

}