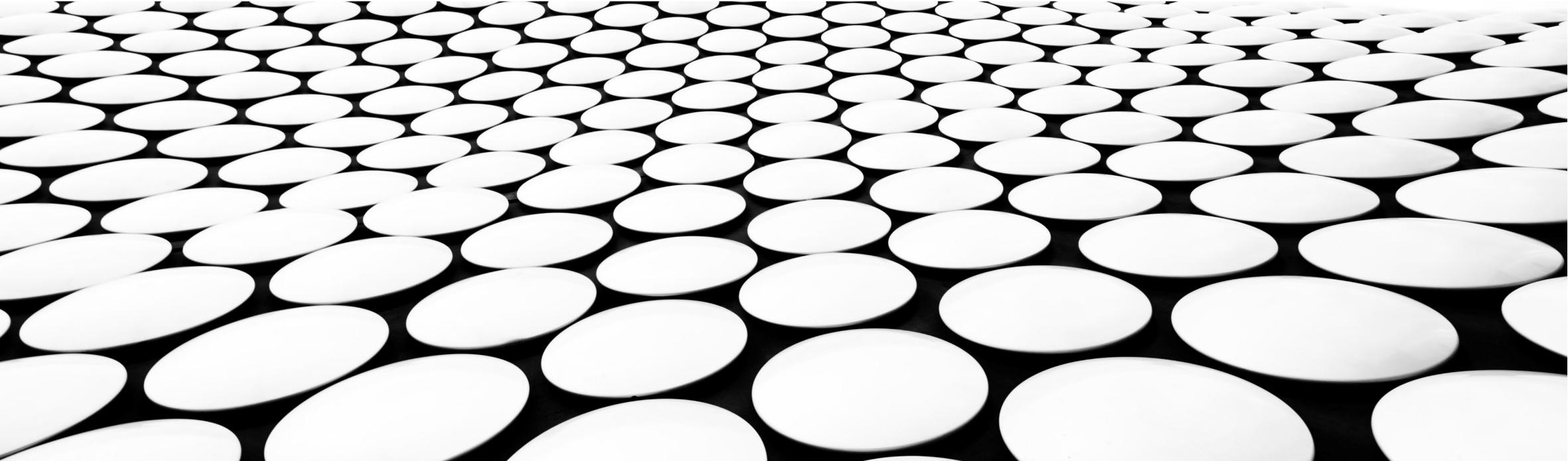


---

---

# S20 ADD NEW BOOK ADMIN



---

---

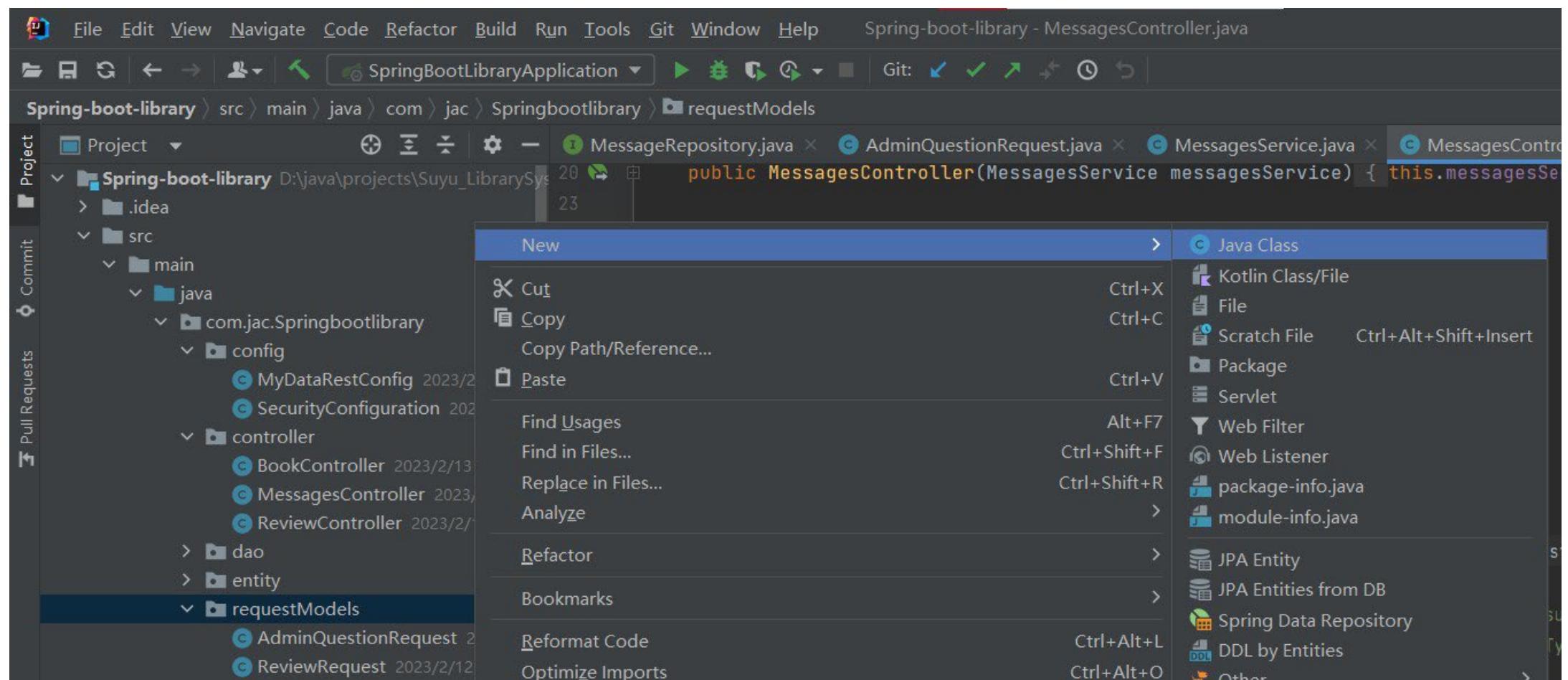
---

# 1. SPRINGBOOT ADD BOOK REQUEST



To add a new book , the very first thing we need to do is to create a object, for the request that's coming from the client to the spring boot back end.

Step1 Right click “requestModels” package, choose “New”, “Java Class”.



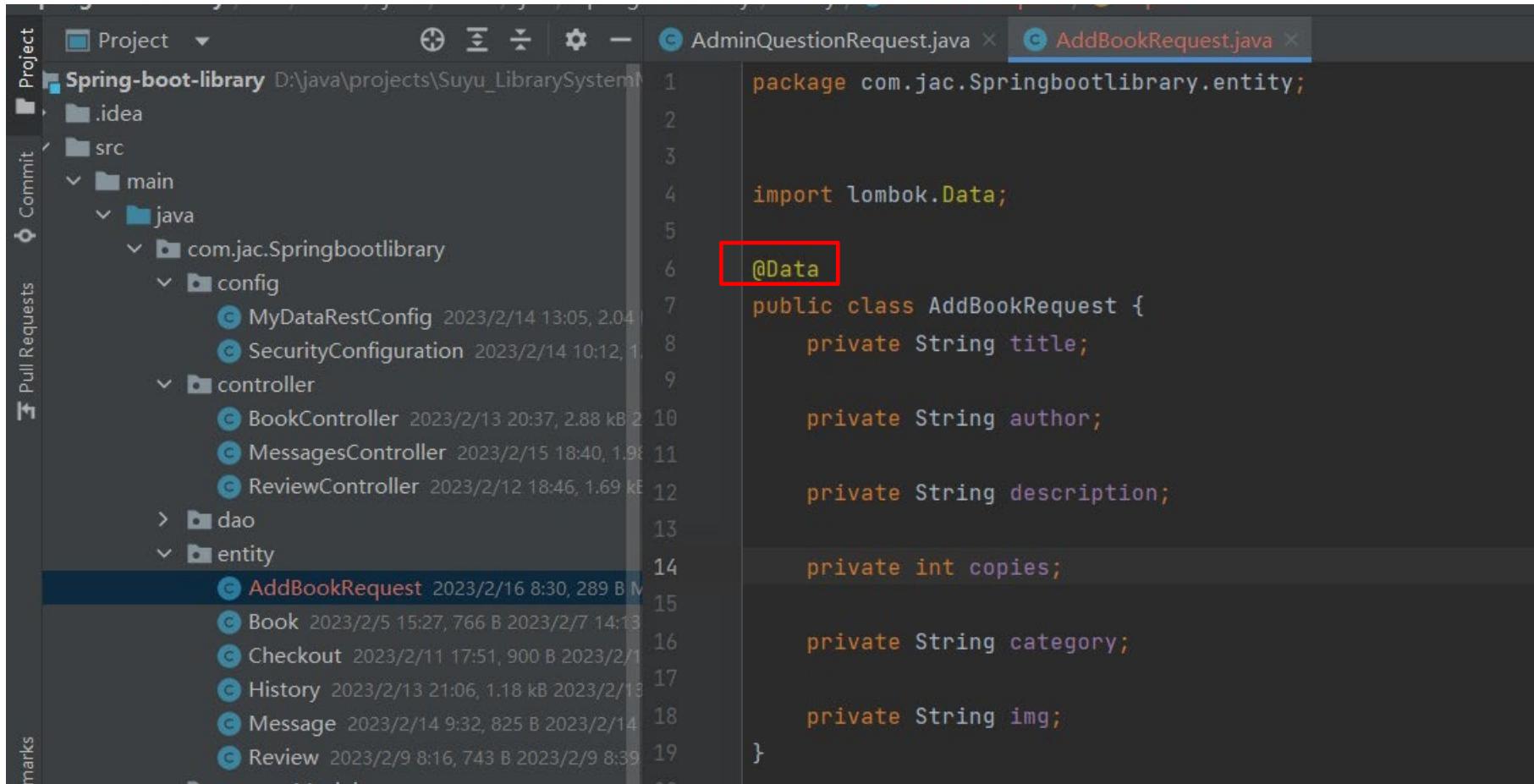
Choose “Class”, input name “AddBookRequest”.

The screenshot shows a Java code editor with a dark theme. On the left is a file tree for a Spring Boot project named 'Suyu\_LibrarySys'. The 'entity' folder is selected. In the main editor area, there is a Java class definition:

```
public MessagesController(MessagesService messagesService) { this.messagesSe  
23  
24 // post a message  
25 @PostMapping("/secure/add/message")  
26 public void postMessage(@RequestHeader(value="Authorization") String token,  
27 @RequestBody Message messageRequest) {...}  
28  
29 //update a message with admin's response  
30 @PutMapping("/secure/admin/message")  
31 public void putMessage(@RequestHeader(value="Authorization") String token,  
32 @RequestBody AdminQuestionRequest adminQuestionRequest) {...}  
33  
34 //extract user information from token  
35 String userEmail = token.substring(7);  
36 String adminToken = token.substring(0, 7);  
37 if (adminToken.equals(adminToken)) {  
38     //call putMessage() in messageService class  
39 }  
40  
41 }  
42  
43 }  
44
```

A code completion dropdown is open at the bottom of the editor, listing options for 'putMessage()' parameters. The 'Class' option is highlighted, and the input field contains 'AddBookRequest'. Other options include 'Interface', 'Enum', and 'Annotation'.

## Step 2 add annotation @Data and class properties.



```
package com.jac.Springbootlibrary.entity;

import lombok.Data;

@Data
public class AddBookRequest {
    private String title;

    private String author;

    private String description;

    private int copies;

    private String category;

    private String img;
}
```

The screenshot shows the IntelliJ IDEA interface with the project 'Spring-boot-library' open. The file 'AddBookRequest.java' is selected in the tabs at the top. The code editor displays the following Java class:

```
package com.jac.Springbootlibrary.entity;

import lombok.Data;

@Data
public class AddBookRequest {
    private String title;

    private String author;

    private String description;

    private int copies;

    private String category;

    private String img;
}
```

The `@Data` annotation is highlighted with a red rectangular box. The code editor's status bar at the bottom shows '14 / 19 lines'.

---

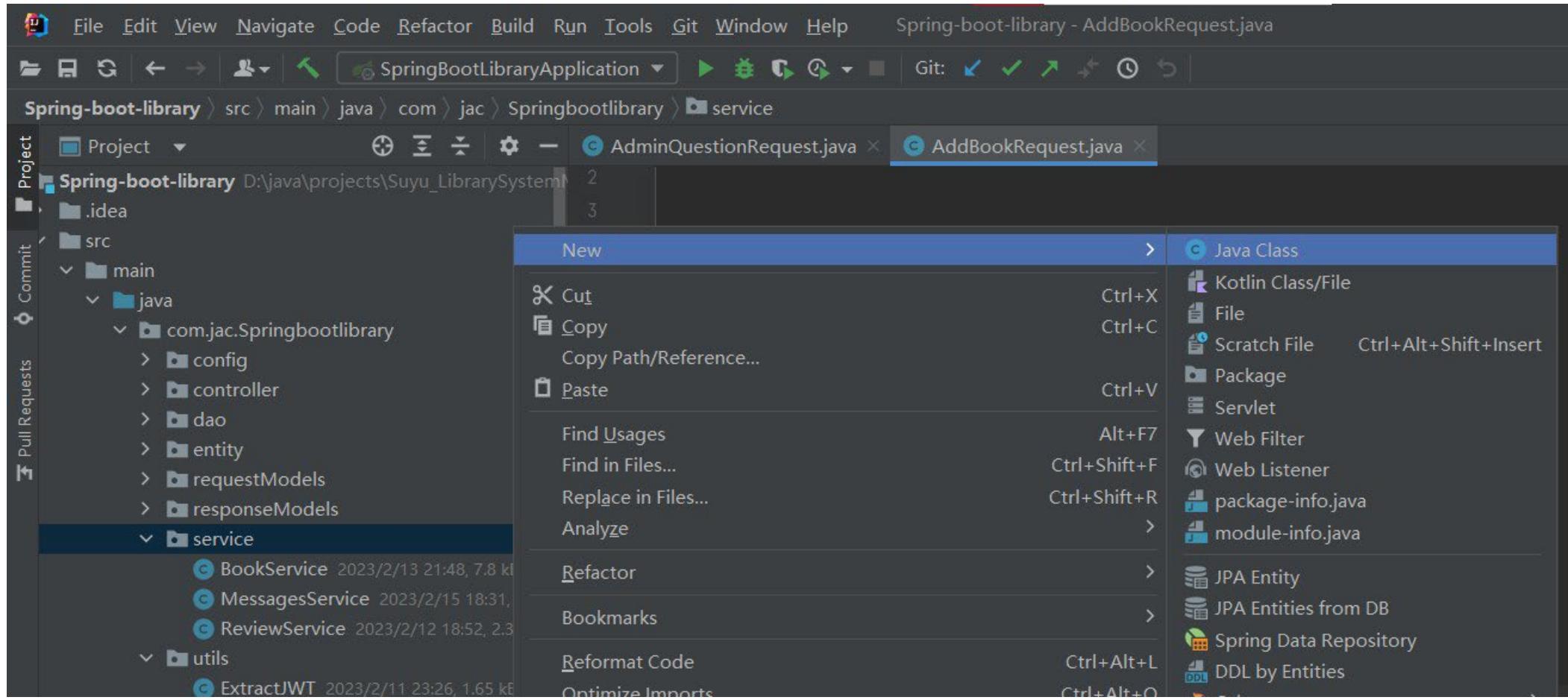
---

---

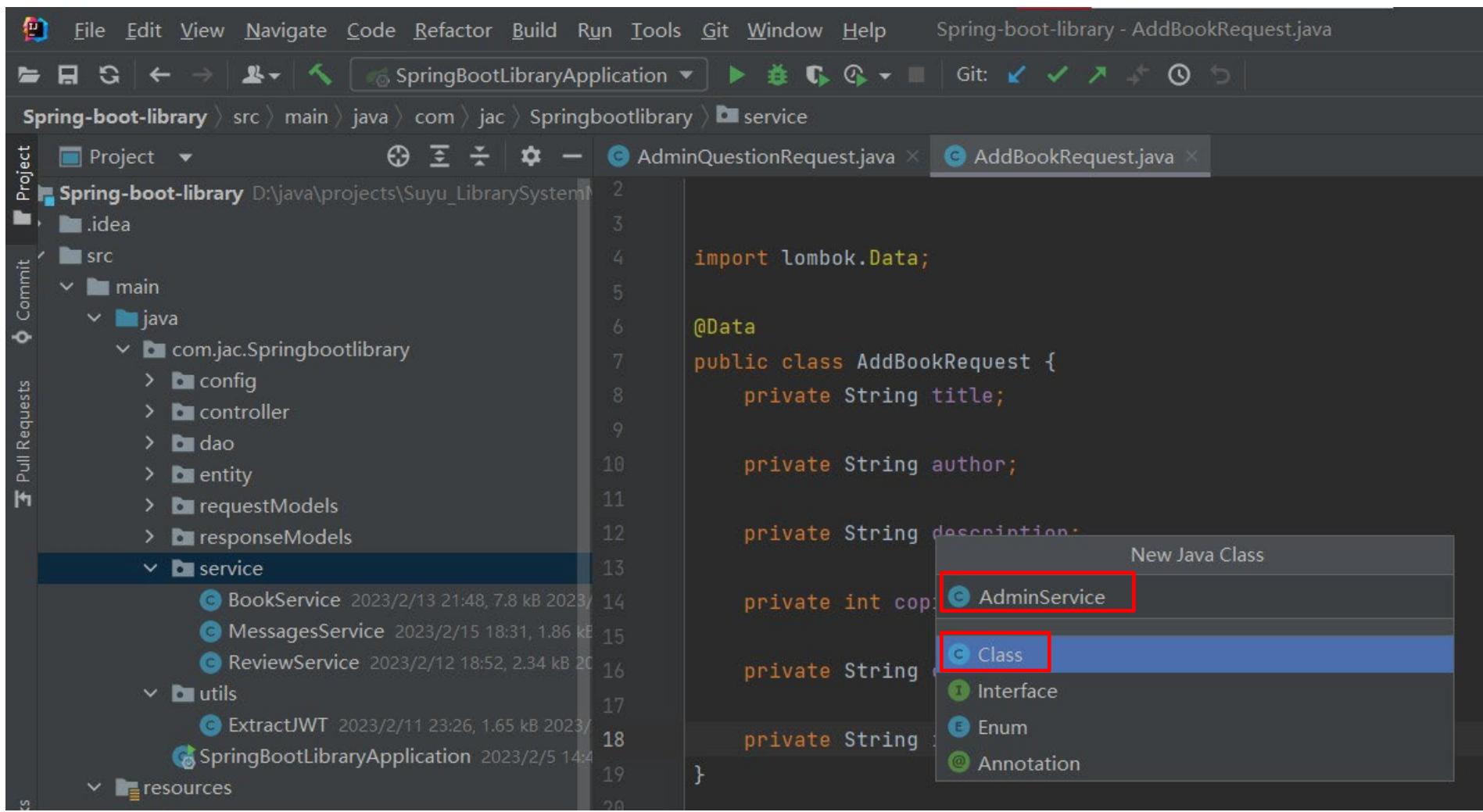
## **2. SPRINGBOOT ADMIN SERVICE LAYER**



Step 1, Right click “service” package, choose “New”, “Java class”.



Choose “Class”, input the name “AdminService”.

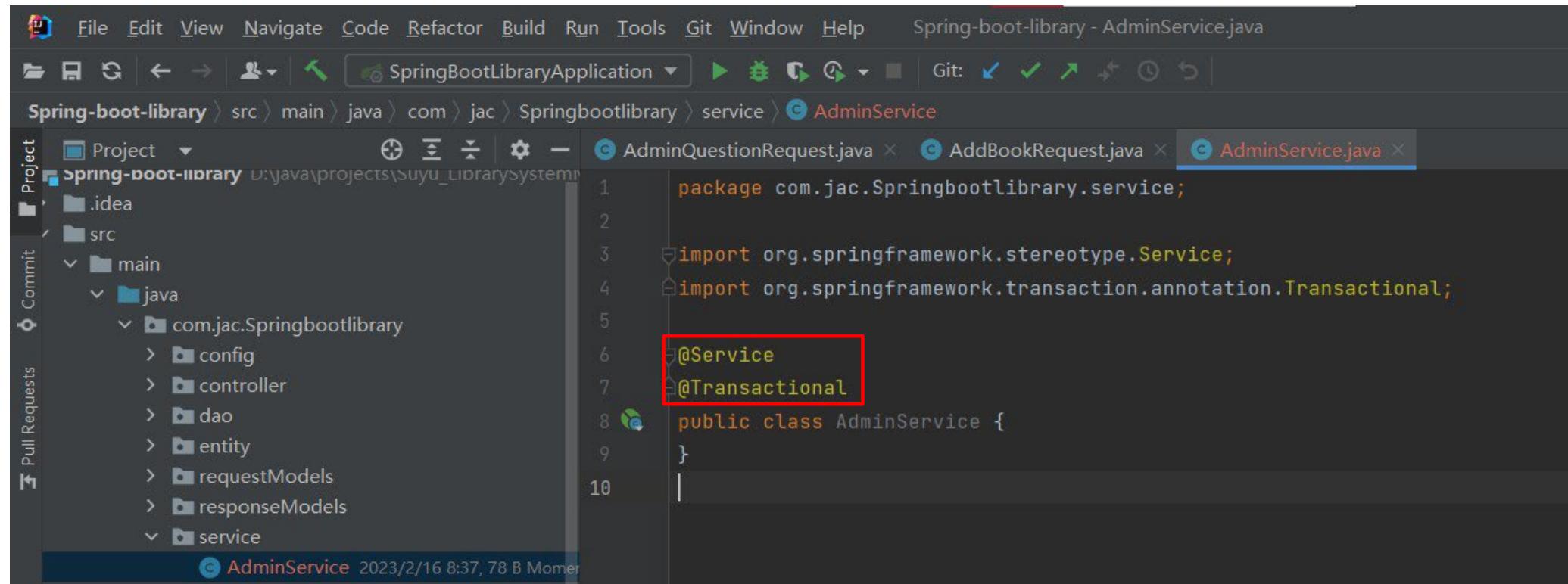


The screenshot shows the IntelliJ IDEA interface with the following details:

- File Menu:** File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, Help.
- Title Bar:** Spring-boot-library - AddBookRequest.java
- Toolbar:** Includes icons for file operations like Open, Save, Undo, Redo, and navigation.
- Project Structure:** Shows the project structure under "Spring-boot-library". The "service" package is selected.
- Code Editor:** Displays the `AddBookRequest.java` file with the following code:

```
import lombok.Data;  
  
@Data  
public class AddBookRequest {  
    private String title;  
  
    private String author;  
  
    private String description;  
}  
}
```
- Code Completion:** A "New Java Class" dialog is open at the bottom right, listing options: AdminService (highlighted with a red box), Class (highlighted with a blue box), Interface, Enum, and Annotation.

Step 2, add @Service ans @Transactional annotation.



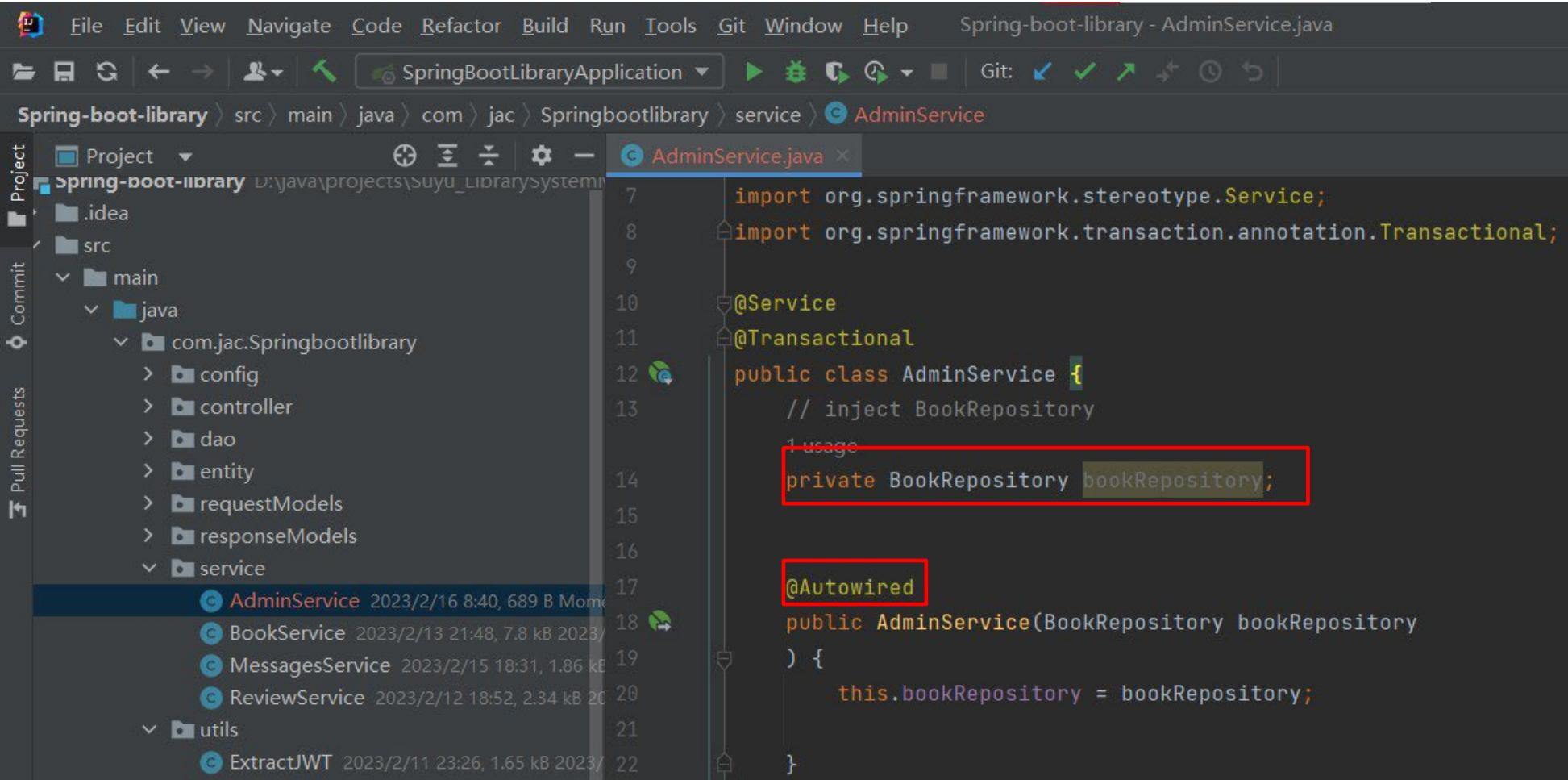
The screenshot shows the IntelliJ IDEA interface with the following details:

- File Bar:** File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, Help.
- Toolbar:** Standard icons for file operations like Open, Save, Cut, Copy, Paste, Find, Replace, and Git.
- Project Bar:** Spring-boot-library > src > main > java > com > jac > Springbootlibrary > service > AdminService.
- Toolbars:** Project, Commit, Pull Requests.
- Code Editor:** The file AdminService.java is open. The code is as follows:

```
1 package com.jac.Springbootlibrary.service;
2
3 import org.springframework.stereotype.Service;
4 import org.springframework.transaction.annotation.Transactional;
5
6 @Service
7 @Transactional
8 public class AdminService {
9 }
10
```

The annotations `@Service` and `@Transactional` are highlighted with a red rectangular box.
- Status Bar:** AdminService 2023/2/16 8:37, 78 B Mome.

Step 3, inject BookRepository and AutoWire our constructor of public AdminService constructor.



The screenshot shows the IntelliJ IDEA interface with the following details:

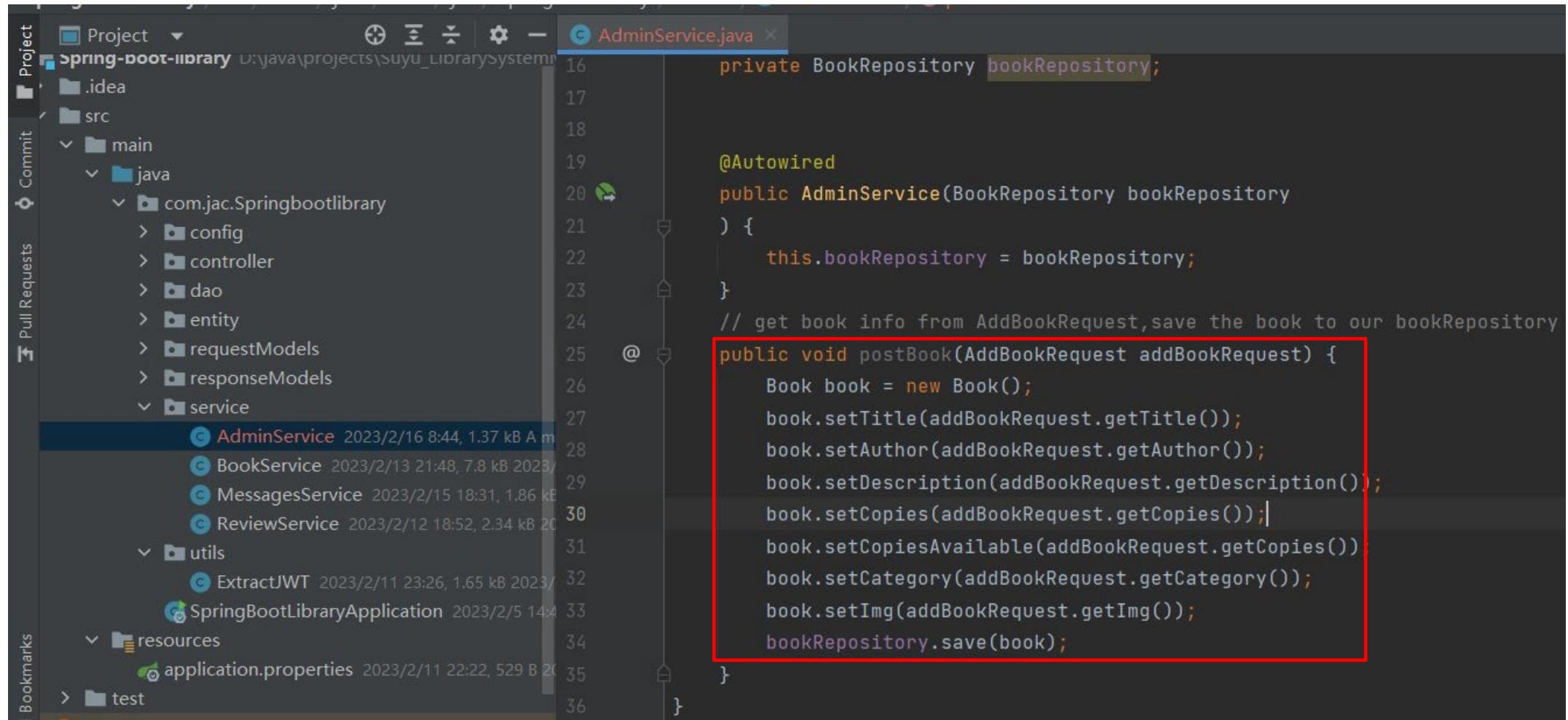
- Project Bar:** Shows "Spring-boot-library - AdminService.java".
- Toolbars:** Standard Java development tools like File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, Help.
- Toolbar Buttons:** Includes icons for file operations, navigation, and Git integration.
- Code Editor:** Displays the `AdminService.java` file under the package `com.jac.Springbootlibrary.service`.
- Code Content:**

```
import org.springframework.stereotype.Service;
import org.springframework.transaction.annotation.Transactional;

@Service
@Transactional
public class AdminService {
    // inject BookRepository
    private BookRepository bookRepository;

    @Autowired
    public AdminService(BookRepository bookRepository) {
        this.bookRepository = bookRepository;
    }
}
```
- Project Explorer:** Shows the project structure with packages like `.idea`, `src`, `main`, `java`, `com.jac.Springbootlibrary` containing `config`, `controller`, `dao`, `entity`, `requestModels`, `responseModels`, `service` (containing `AdminService`), and `utils`.
- File Status:** Shows files like `AdminService`, `BookService`, `MessagesService`, `ReviewService`, and `ExtractJWT` with their last modified dates and sizes.

Step 4, create a `postBook()`, which allow us to get book info from `AddBookRequest`, save the book to our `BookRepository`.



The screenshot shows the IntelliJ IDEA interface with the project structure on the left and the code editor on the right. The code editor displays the `AdminService.java` file. A red box highlights the `postBook()` method, which is annotated with `@Transactional`. The method takes an `AddBookRequest` object as a parameter and creates a new `Book` object, setting its properties from the request and saving it to the `bookRepository`.

```
private BookRepository bookRepository;

@Autowired
public AdminService(BookRepository bookRepository) {
    this.bookRepository = bookRepository;
}

// get book info from AddBookRequest, save the book to our bookRepository
@Transactional
public void postBook(AddBookRequest addBookRequest) {
    Book book = new Book();
    book.setTitle(addBookRequest.getTitle());
    book.setAuthor(addBookRequest.getAuthor());
    book.setDescription(addBookRequest.getDescription());
    book.setCopies(addBookRequest.getCopies());
    book.setCopiesAvailable(addBookRequest.getCopies());
    book.setCategory(addBookRequest getCategory());
    book.setImg(addBookRequest.getImg());
    bookRepository.save(book);
}
```

---

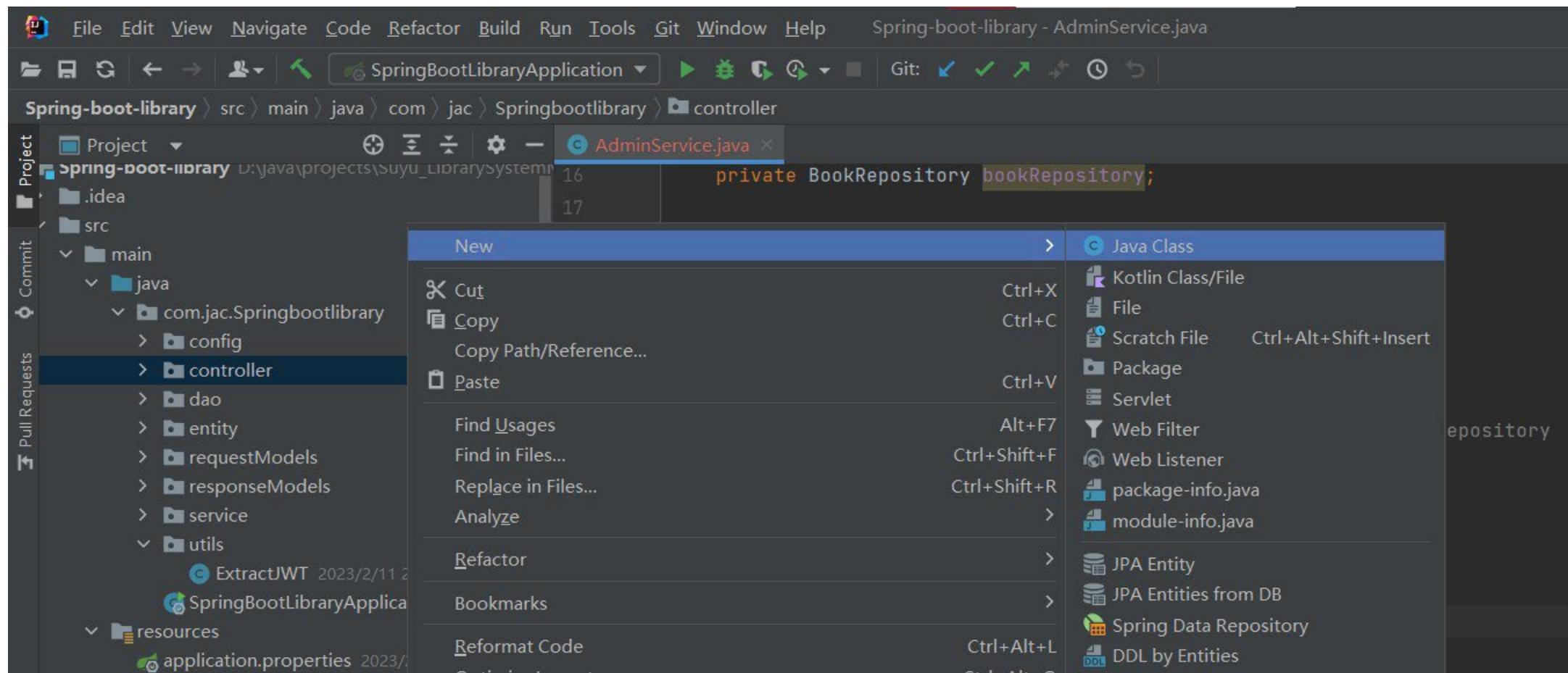
---

## **3. SPRINGBOOT ADMIN CONTROLLER AND SECURITY**



Now we will create a new admin controller with the endpoint to be able to create a new book.

Step 1, Right click our “controller” package and choose “new”, “Java class”.



Choose “Class”, input the name “AdminController”.

The screenshot shows a Java code editor with the following code snippet:

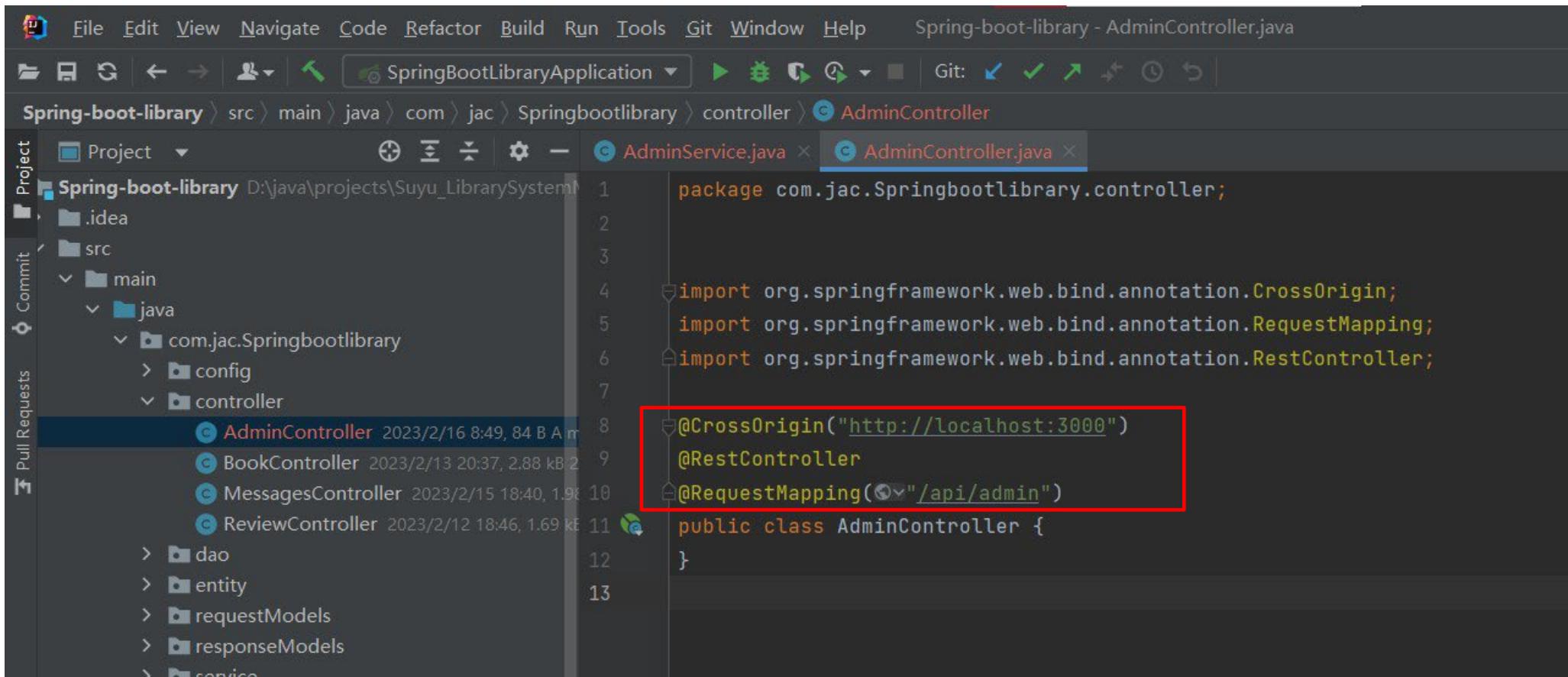
```
    } {
        this.bookRepository = bookRepository;
    }
    // get book info from AddBookRequest, save the book to our bookRepository
    @
    public void postBook(AddBookRequest addBookRequest) {
        Book book = new Book();
        book.setT: New Java Class
        book.setAu: AdminController
        book.setD: Class
        book.setC: Interface
        book.setCo: Enum
        book.setCa: Annotation
        book.setIn: ...
        bookRepository.save(book);
    }
}
```

A context menu is open at the position of the assignment operator (`=`) in the line `book.setT: New Java Class`. The menu items are:

- New Java Class
- AdminController
- Class (highlighted with a red box)
- Interface
- Enum
- Annotation

Step 2, add annotations:

```
@CrossOrigin("http://localhost:3000")
@RestController
@RequestMapping("/api/admin")
```



The screenshot shows the IntelliJ IDEA interface with the AdminController.java file open. The code editor displays the following Java code:

```
package com.jac.Springbootlibrary.controller;

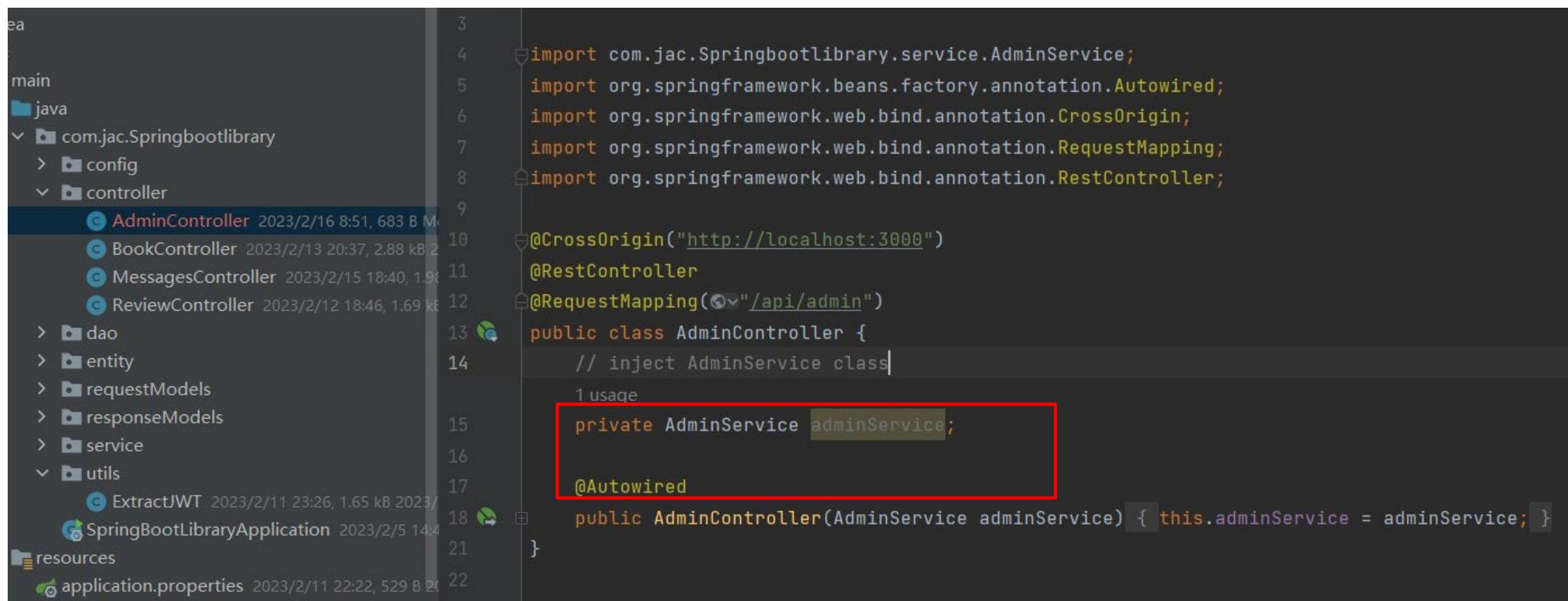
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequestMapping("/api/admin")

public class AdminController {
```

The annotations `@CrossOrigin`, `@RestController`, and `@RequestMapping` are highlighted with a red rectangular box. The code editor's navigation bar shows the current file is AdminController.java. The project structure on the left shows the package structure: Spring-boot-library > src > main > java > com > jac > Springbootlibrary > controller > AdminController.java.

Step 3, inject AdminService and then auto wire our AdminController constructor to our AdminService.

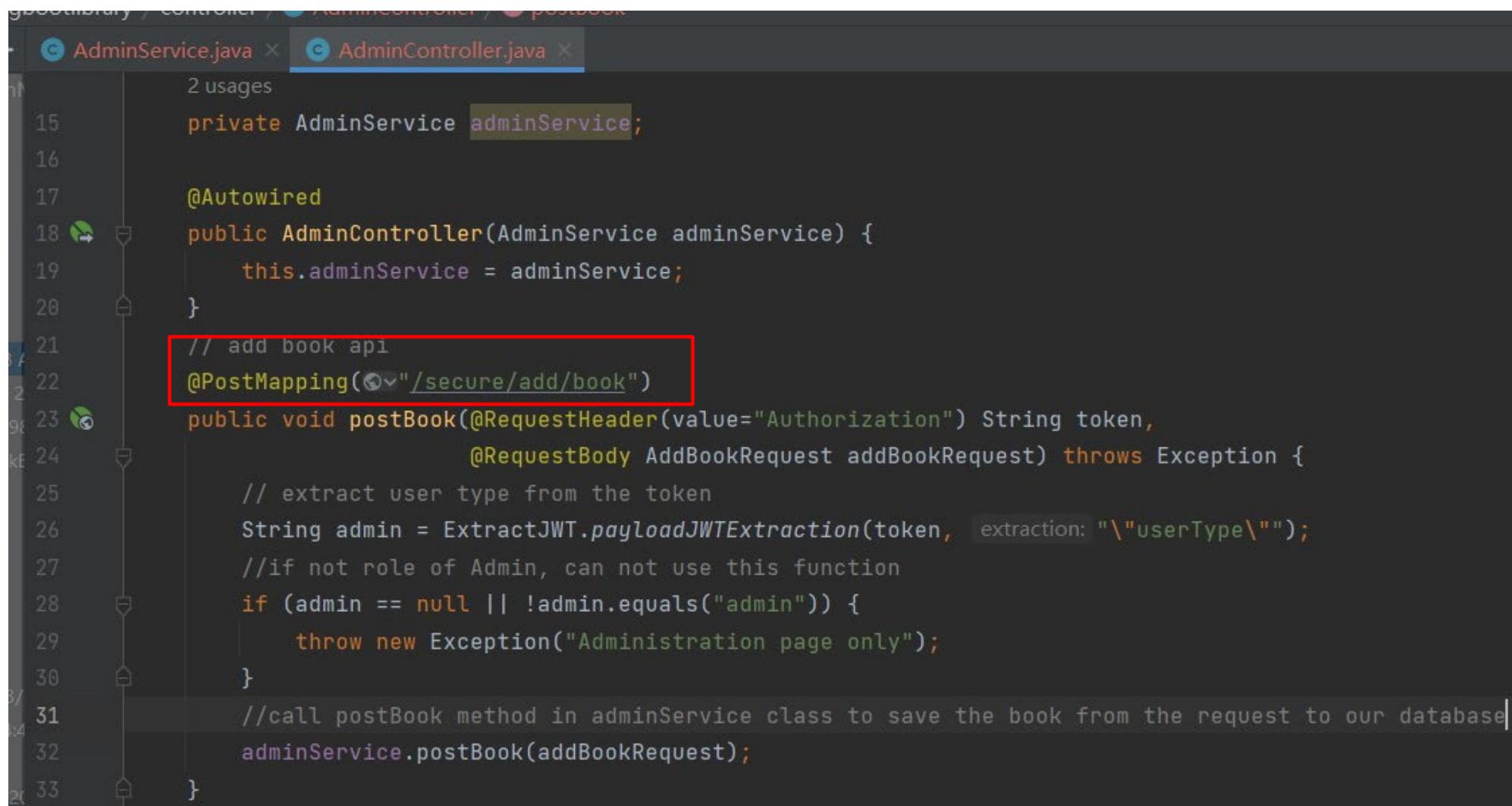


```
ea

main
  java
    com.jac.Springbootlibrary
      config
      controller
        AdminController 2023/2/16 8:51, 683 kB
        BookController 2023/2/13 20:37, 2.88 kB
        MessagesController 2023/2/15 18:40, 1.93 kB
        ReviewController 2023/2/12 18:46, 1.69 kB
      dao
      entity
      requestModels
      responseModels
      service
      utils
        ExtractJWT 2023/2/11 23:26, 1.65 kB 2023/2/11 23:26, 1.65 kB
        SpringBootLibraryApplication 2023/2/5 14:44 2023/2/5 14:44
    resources
    application.properties 2023/2/11 22:22, 529 kB 2023/2/11 22:22, 529 kB
```

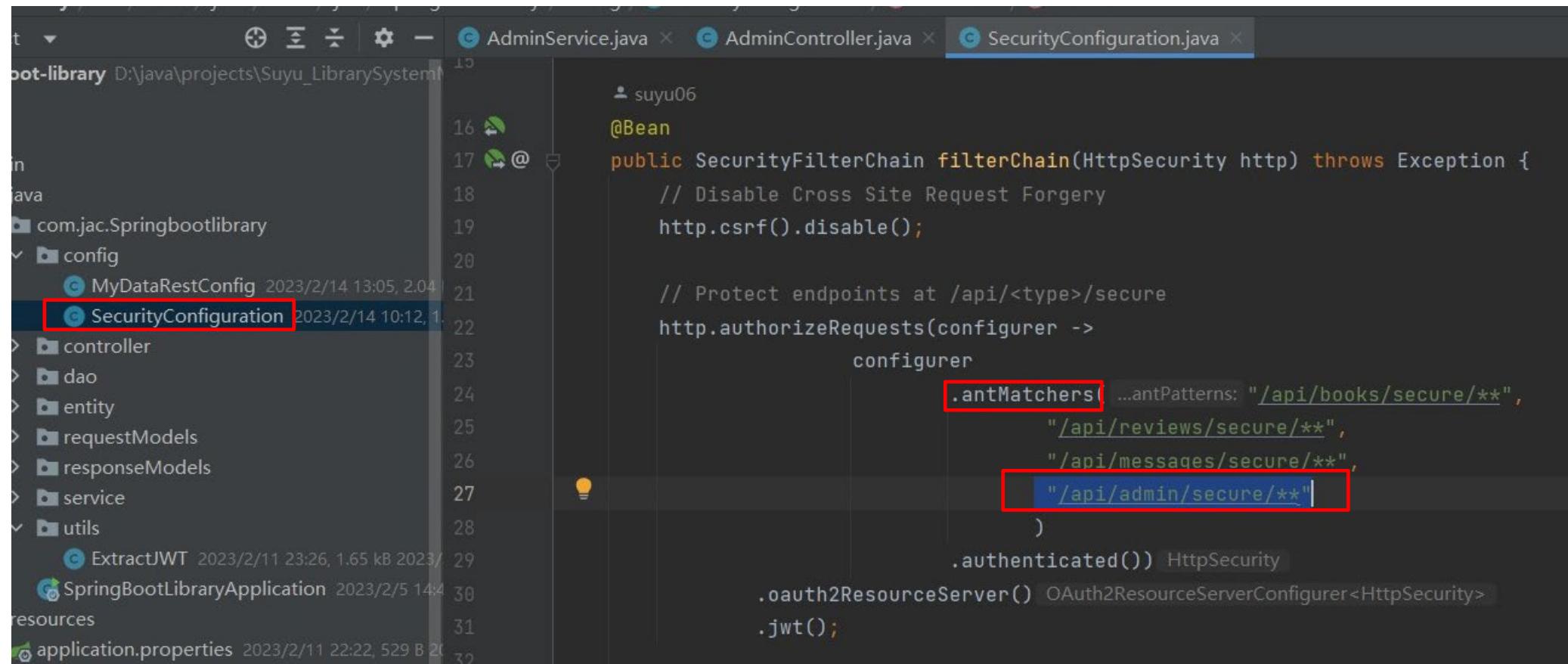
```
3
4 import com.jac.Springbootlibrary.service.AdminService;
5 import org.springframework.beans.factory.annotation.Autowired;
6 import org.springframework.web.bind.annotation.CrossOrigin;
7 import org.springframework.web.bind.annotation.RequestMapping;
8 import org.springframework.web.bind.annotation.RestController;
9
10 @CrossOrigin("http://localhost:3000")
11 @RestController
12 @RequestMapping("/api/admin")
13 public class AdminController {
14     // inject AdminService class
15     private AdminService adminService;
16
17     @Autowired
18     public AdminController(AdminService adminService) { this.adminService = adminService; }
19
20 }
```

## Step 4, use PostMapping method to create add book API.



```
gRPClibrary / controller / AdminController.java
1 AdminService.java x 2 usages
2 AdminController.java x
3
4
5
6
7
8
9
10
11
12
13
14
15     private AdminService adminService;
16
17     @Autowired
18     public AdminController(AdminService adminService) {
19         this.adminService = adminService;
20     }
21
22     // add book api
23     @PostMapping(value="/secure/add/book")
24     public void postBook(@RequestHeader(value="Authorization") String token,
25                          @RequestBody AddBookRequest addBookRequest) throws Exception {
26
27         // extract user type from the token
28         String admin = ExtractJWT.payloadJWTExtraction(token, extraction: "\"userType\"");
29
30         //if not role of Admin, can not use this function
31         if (admin == null || !admin.equals("admin")) {
32             throw new Exception("Administration page only");
33         }
34
35         //call postBook method in adminService class to save the book from the request to our database
36         adminService.postBook(addBookRequest);
37     }
38 }
```

Step 5, Hop into our “config” package , “SecurityConfiguration” class, the same thing that we ‘ve done before, add “admin/secure/\*\*” to our .antMatchers().



```
16
17 @Bean
18 public SecurityFilterChain filterChain(HttpSecurity http) throws Exception {
19     // Disable Cross Site Request Forgery
20     http.csrf().disable();
21
22     // Protect endpoints at /api/<type>/secure
23     http.authorizeRequests(configurer ->
24         configurer
25             .antMatchers(...antPatterns: "/api/books/secure/**",
26                         "/api/reviews/secure/**",
27                         "/api/messages/secure/**",
28                         "/api/admin/secure/**")
29             .authenticated()
30             .oauth2ResourceServer()
31             .jwt());
32 }
```

---

---

---

## **4. REACT ADD BOOK REQUEST MODEL**



The screenshot shows a dark-themed instance of Visual Studio Code. The Explorer sidebar on the left displays a file tree for a 'REACT-LIBRARY' project. The 'models' folder is currently selected. A context menu is open over the 'models' folder, with the 'New File...' option highlighted. Other visible options in the menu include 'New Folder...', 'Reveal in File Explorer', 'Open in Integrated Terminal', 'Find in Folder...', and keyboard shortcuts for some of these actions.

```
src > layouts > ManageLibraryPage > components
1 import { useOktaAuth } from "okta-react"
2 import { useEffect, useState } from "react"
3 import AdminMessageRequest from "./AdminMessageRequest"
4 import MessageModel from "./MessageModel"
5 import { Pagination } from "./Pagination"
6 import { SpinnerLoading } from "./SpinnerLoading"
7 import { AdminMessage } from "./AdminMessage"
8
9 export const AdminMessages = () =>
10   // oktaAuth
11   const { authState } = useOktaAuth()
12
13   // Normal Loading Piece
14   const [isloadingMessage, setisloadingMessage] = useState(false)
15   const [adminMessages, setAdminMessages] = useState([])
```

Step1 Let's go to our “models” folder, create a new typescript file that's called “AddBookRequest.ts”.

Step 2, add the same properties as our AddBookRequest class in springboot, and its constructor.

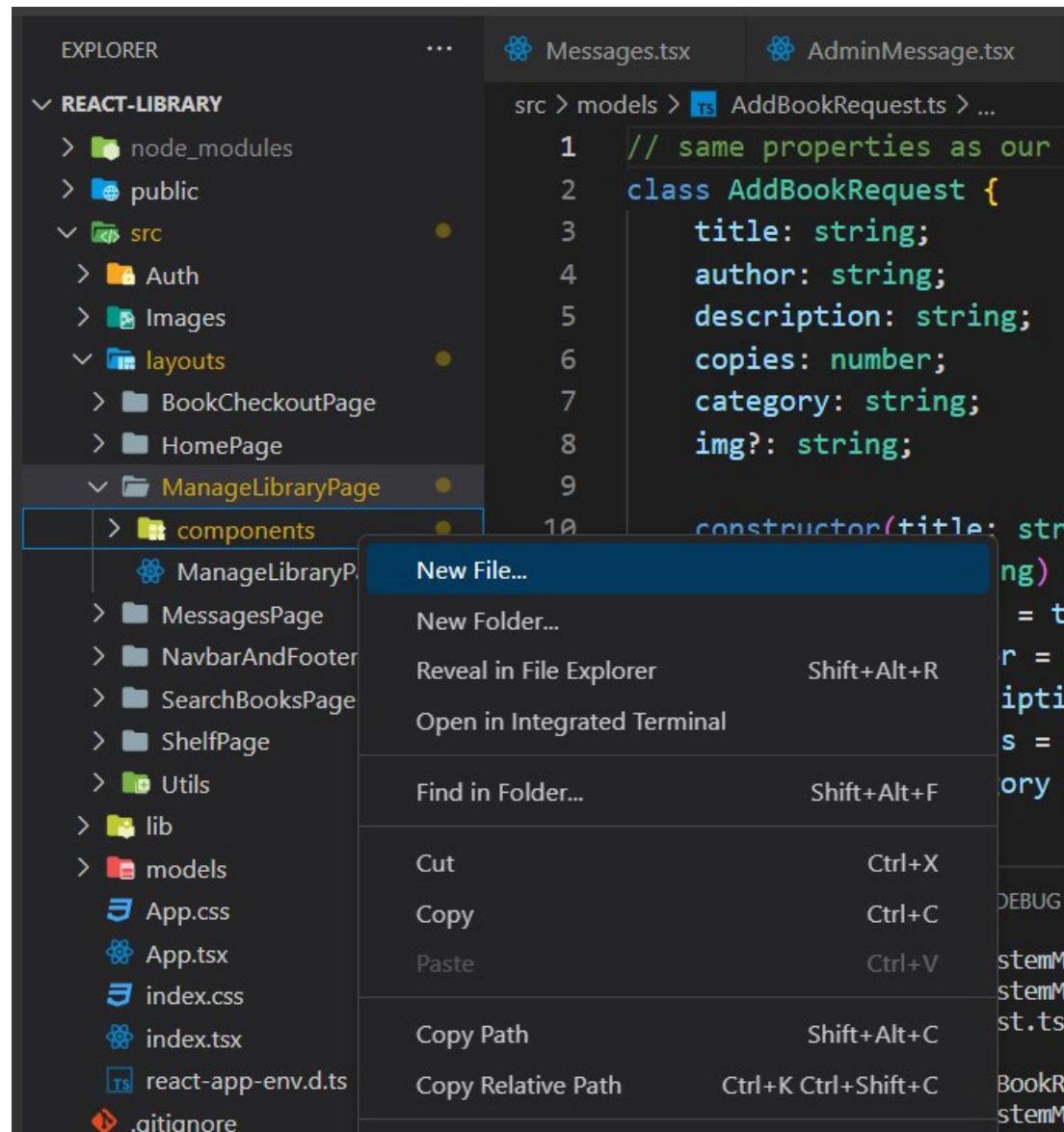
```
src > models > AddBookRequest.ts > default
1  class AddBookRequest {
2      title: string;
3      author: string;
4      description: string;
5      copies: number;
6      category: string;
7      img?: string;
8
9      constructor(title: string, author: string, description: string, copies: number,
10          category: string) {
11              this.title = title;
12              this.author = author;
13              this.description = description;
14              this.copies = copies;
15              this.category = category;
16      }
17  }
18
19  export default AddBookRequest;
```

---

---

## **5. REACT ADD BOOK COMPONENT STATE AND HTML/CSS**





Step 1, jumping into our “layouts”, “ManageLibraryPage” , go into “components”, right click , choose “new file”, name it “AddNewBook.tsx”.

Step 2, In AddNewBook.tsx, as always, create export const AddNewBook() structure.

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "REACT-LIBRARY". The "src" folder contains "Auth", "Images", "layouts" (which includes "BookCheckoutPage" and "HomePage"), and "ManageLibraryPage" (which includes "components"). "AddNewBook.tsx" is selected in the file list.
- Code Editor (Right):** The file "AddNewBook.tsx" is open. The code is as follows:

```
1  export const AddNewBook = () => {
2      return(
3          ...
4      );
5 }
```
- Status Bar:** Shows "AddNewBook.tsx - react-library - Visual Studio Code".

## Step 3, import oktaAuth and create useState for new book and alert messages.

The screenshot shows a Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "REACT-LIBRARY". Key folders include "src" (which contains "Auth", "Images", "layouts", and "ManageLibraryPage"), "layouts", and "utils". Specific files shown are "AddNewBook.tsx", "AdminMessage.tsx", "AdminMessages.tsx", and "ManageLibraryPage.tsx".
- Code Editor (Right):** The active file is "AddNewBook.tsx". The code imports "useOktaAuth" from "@okta(okta-react)" and "useState" from "react". It defines a function "AddNewBook" that uses useState to manage state for title, author, description, copies, category, and selectedImage. It also manages display states for warning and success messages.
- Terminal (Top):** Shows the current working directory as "src > layouts > ManageLibraryPage > components > AddNewBook.tsx".
- Status Bar (Bottom):** Shows the file name "AddNewBook.tsx - react-library - Visual Studio Code".

```
import { useOktaAuth } from "@okta(okta-react)";
import { useState } from "react";

export const AddNewBook = () => {
    // authState
    const { authState } = useOktaAuth();

    // New Book
    const [title, setTitle] = useState('');
    const [author, setAuthor] = useState('');
    const [description, setDescription] = useState('');
    const [copies, setCopies] = useState(0);
    const [category, setCategory] = useState('Category');
    const [selectedImage, setSelectedImage] = useState<any>(null);

    // Displays
    const [displayWarning, setDisplayWarning] = useState(false);
    const [displaySuccess, setDisplaySuccess] = useState(false);
```

Step 4, create a categoryField() to fetch the category string from the user's choice in the category dropdown menu.

The screenshot shows a code editor interface with a dark theme. On the left is a sidebar with icons for search, file navigation, and other development tools. The main area shows a file tree for a project named "REACT-LIBRARY". The "src" folder contains "layouts", "ManageLibraryPage", and "Auth" components. Under "ManageLibraryPage", there is a "components" folder containing "AddNewBook.tsx", "AdminMessage.tsx", "AdminMessages.tsx", and "ManageLibraryPage.tsx". Other files in "ManageLibraryPage" include "MessagesPage", "NavbarAndFooter", "SearchBooksPage", "ShelfPage", and "Utils". The "lib" folder is also present. The right side of the screen displays the content of "AddNewBook.tsx". A red box highlights the following code block:

```
4  export const AddNewBook = () => {
5      // authState
6      const { authState } = useOktaAuth();
7
8      // New Book
9      const [title, setTitle] = useState('');
10     const [author, setAuthor] = useState('');
11     const [description, setDescription] = useState('');
12     const [copies, setCopies] = useState(0);
13     const [category, setCategory] = useState('Category');
14     const [selectedImage, setSelectedImage] = useState<any>(null);
15
16     // Displays
17     const [displayWarning, setDisplayWarning] = useState(false);
18     const [displaySuccess, setDisplaySuccess] = useState(false);
19
20     // get the category string from the user input
21     function categoryField(value: string) {
22         setCategory(value);
23     }

```

Step 5, fill in return part with html/CSS code.

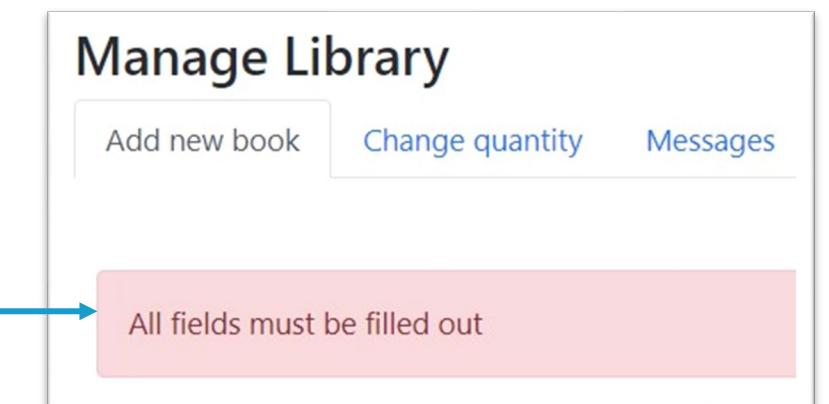
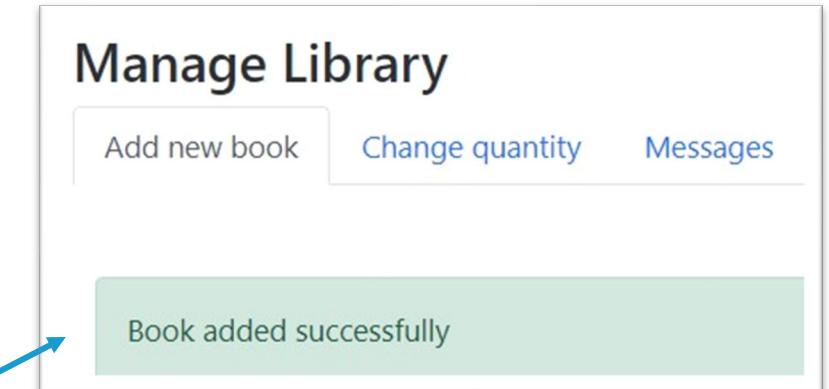
## 5.1 successful banner and alert banner

Terminal Help AddNewBook.tsx - react-library - Visual Studio Code

AdminMessage.tsx AddBookRequest.tsx AddNewBook.tsx 9+, U X

src > layouts > ManageLibraryPage > components > AddNewBook.tsx > AddNewBook

```
21     function categoryField(value: string) {
22         setCategory(value);
23     }
24
25     return(
26         <div className='container mt-5 mb-5'>
27             {/* successful banner */}
28             {displaySuccess &&
29                 <div className='alert alert-success' role='alert'>
30                     Book added successfully
31                 </div>
32             }
33             {/* alert banner */}
34             {displayWarning &&
35                 <div className='alert alert-danger' role='alert'>
36                     All fields must be filled out
37                 </div>
38             }
39         </div>
40     )
41 }
```



## 5.2 card header “Add a new book” part

The screenshot shows a web application interface and its corresponding code in a code editor.

**Web Application Interface:**

- A header bar with "Add a new book" (highlighted with a red box).
- Form fields: Title (input field), Description (input field), Author (input field), and Category (dropdown menu with options: Front End, Back End, Data, DevOps). The dropdown is currently set to "Category".

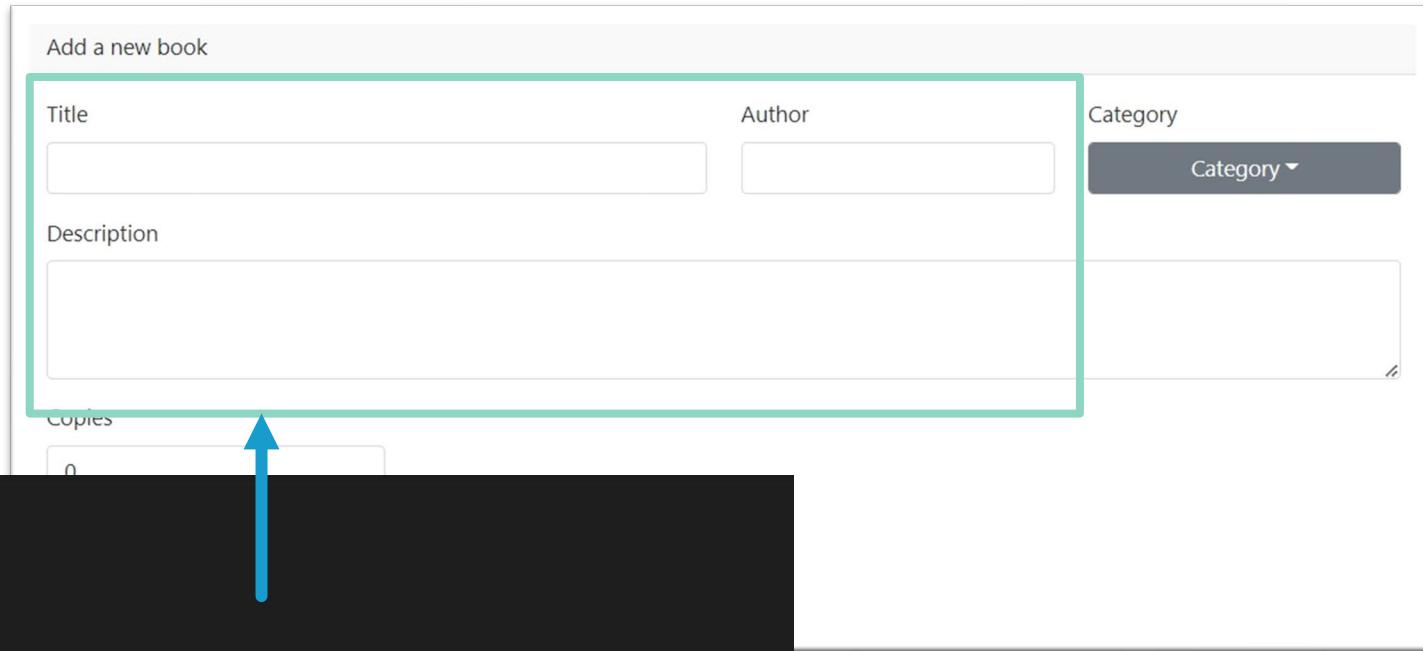
**Code Editor:**

- File: AddNewBook.tsx - react-library - Visual Studio Code
- Code snippet:

```
src > layouts > ManageLibraryPage > components > AddNewBook.tsx > AddNewBook
  ...
33     /* alert banner */
34     {displayWarning &&
35         <div className='alert alert-danger' role='alert'
36             |   All fields must be filled out
37         </div>
38     }
39     <div className='card'>
40         <div className='card-header'>
41             Add a new book
42         </div>
```

A blue arrow points from the "Add a new book" text in the code editor to the "Add a new book" text in the web application header.

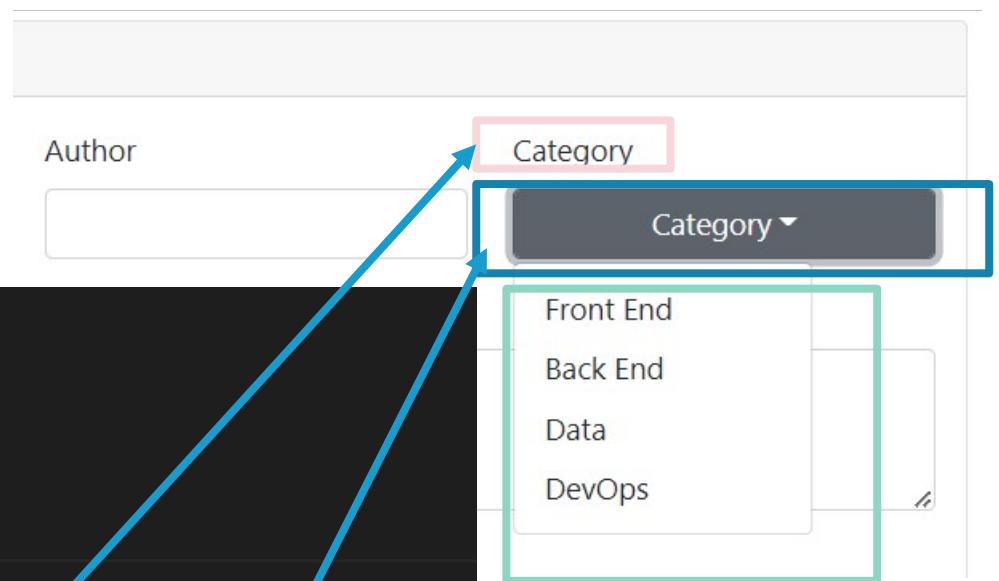
### 5.3 book info part



```
/* form in the card body */
<div className='card-body'>
  <form method='POST'>
    <div className='row'>
      /* basic book info */
      <div className='col-md-6 mb-3'>
        <label className='form-label'>Title</label>
        <input type="text" className='form-control' name='title' required
               onChange={e => setTitle(e.target.value)} value={title} />
      </div>
      <div className='col-md-3 mb-3'>
        <label className='form-label'> Author </label>
        <input type="text" className='form-control' name='author' required
               onChange={e => setAuthor(e.target.value)} value={author}/>
      </div>
    </div>
  </form>
</div>
```

## 5.4 “Category” dropdown menu part.

```
<div className='col-md-3 mb-3'>
  <label className='form-label'> Author </label>
  <input type="text" className='form-control' name='author' required
    onChange={e => setAuthor(e.target.value)} value={author}/>
</div>
{/* drop down menu */}
<div className='col-md-3 mb-3'>
  <label className='form-label'> Category</label>
  <button className='form-control btn btn-secondary dropdown-toggle' type='button'
    id='dropdownMenuButton1' data-bs-toggle='dropdown' aria-expanded='false'>
    {category}
  </button>
  {/* contene of drop down menu */}
  <ul id='addNewBookId' className='dropdown-menu' aria-labelledby='dropdownMenuButton1'>
    <li><a onClick={() => categoryField('FE')} className='dropdown-item'>Front End</a></li>
    <li><a onClick={() => categoryField('BE')} className='dropdown-item'>Back End</a></li>
    <li><a onClick={() => categoryField('Data')} className='dropdown-item'>Data</a></li>
    <li><a onClick={() => categoryField('DevOps')} className='dropdown-item'>DevOps</a></li>
  </ul>
</div>
```

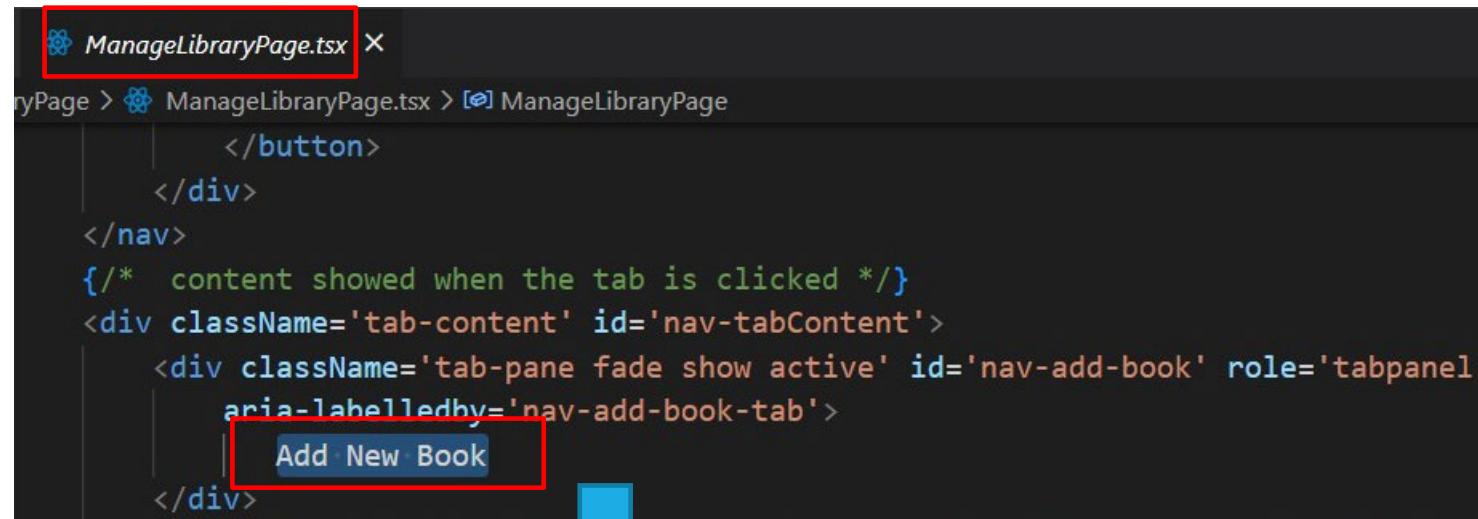


5.5 input box part for book description, number of copies, choose book image file and “add book” button.

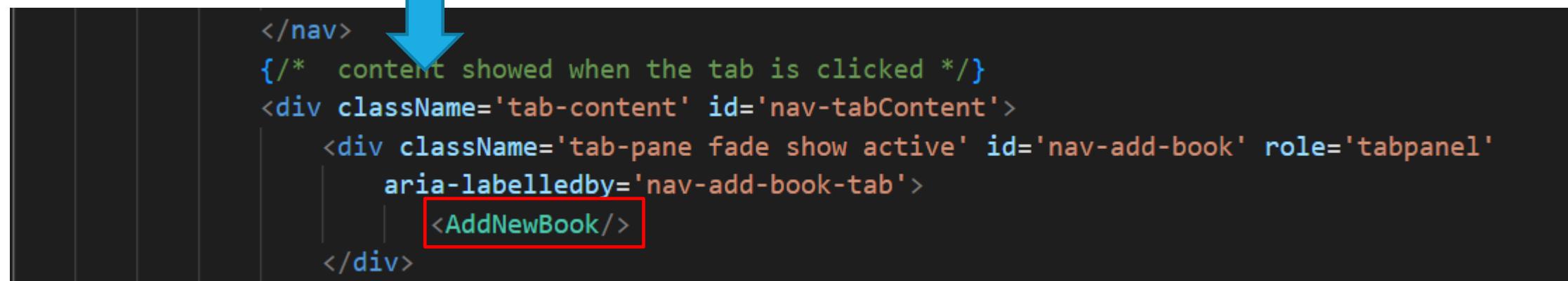
```
    </div>
  </div>
  /* input box part */
  <div className='col-md-12 mb-3'>
    <label className='form-label'>Description</label>
    <textarea className='form-control' id='exampleFormControlTextarea1' rows={3}
      onChange={e => setDescription(e.target.value)} value={description}></textarea>
  </div>
  <div className='col-md-3 mb-3'>
    <label className='form-label'>Copies</label>
    <input type='number' className='form-control' name='Copies' required
      onChange={e => setCopies(Number(e.target.value))} value={copies}>
  </div>
  <input type='file' />
  /* add book btn */
  <div>
    <button type='button' className='btn btn-primary mt-3'>
      Add Book
    </button>
  </div>
</form>
```

The screenshot shows a user interface for adding a book. It features a text area labeled "Description" with a pink border. Below it is a number input field labeled "Copies" containing the value "0". Next is a file input field with a red border, showing "Choose File No file chosen". At the bottom is a blue "Add Book" button.

Step 6, Jump back into our ManageLibraryPage, in the tab content part, replace the static word "Add new book" with our new AddNewBook component, and import it .



```
ManageLibraryPage.tsx x
ryPage > ManageLibraryPage.tsx > ManageLibraryPage
    </button>
  </div>
</nav>
/* content showed when the tab is clicked */
<div className='tab-content' id='nav-tabContent'>
  <div className='tab-pane fade show active' id='nav-add-book' role='tabpanel'
    aria-labelledby='nav-add-book-tab'>
    Add New Book
  </div>
```



```
</nav>
/* content showed when the tab is clicked */
<div className='tab-content' id='nav-tabContent'>
  <div className='tab-pane fade show active' id='nav-add-book' role='tabpanel'
    aria-labelledby='nav-add-book-tab'>
    <AddNewBook/>
  </div>
```

Sign in with our admin user's email and password.

Sign In

Username

Password

Remember me

Sign In

Need help signing in?

Visit <http://localhost:3000/admin>, we will get this add new book page.

localhost:3000/admin

JAC Read Home Search Books Shelf

## Manage Library

Add new book Change quantity Messages

Add a new book

Title	Author	Category
<input type="text"/>	<input type="text"/>	<div>Category ▾<ul style="list-style-type: none"><li>Front End</li><li>Back End</li><li>Data</li><li>DevOps</li></ul></div>

Description

Copies

0

Choose File No file chosen

Add Book

---

---

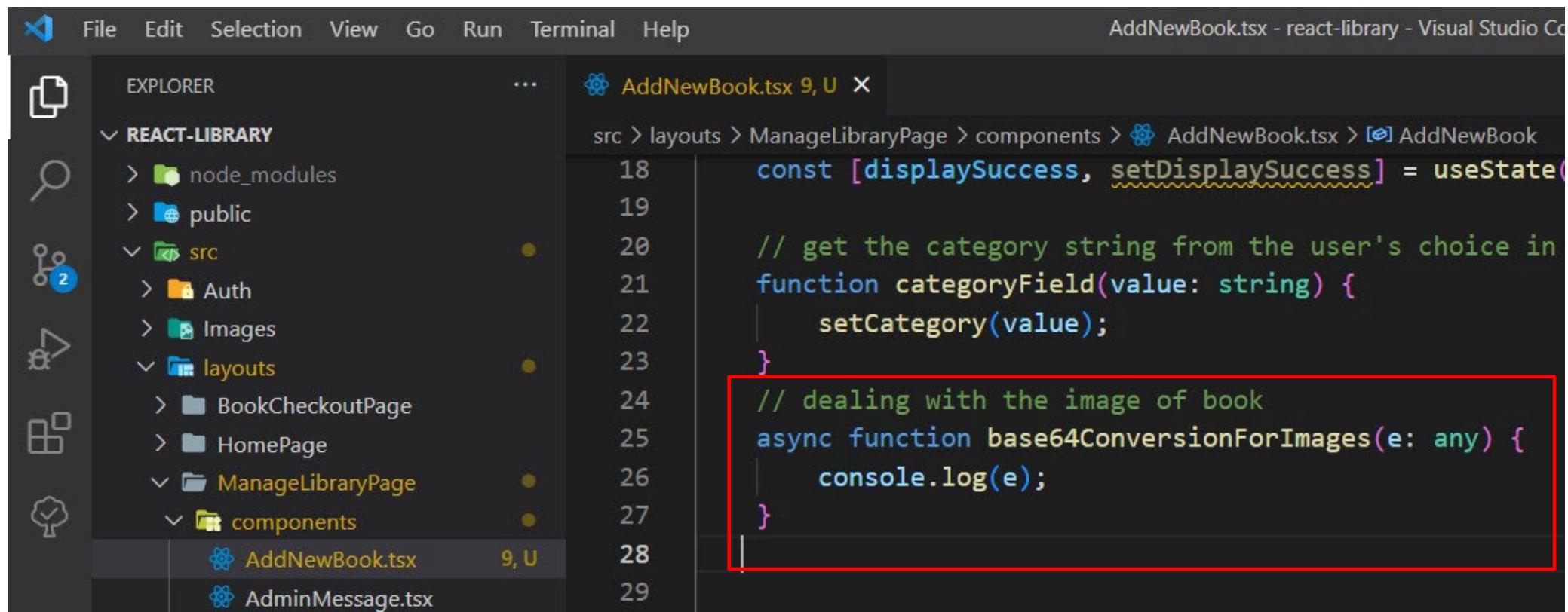
---

## 6. REACT SAVE IMAGE TO STATE



one thing that we need to do is when we select a file, allow our client side application to render this into a base 64. So we can send over this to the spring boot application.

Step 1, In AddNewBook.tsx, under the function of categoryField(), create an async function base64ConversionForImages().



The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** AddNewBook.tsx - react-library - Visual Studio Code.
- Explorer View:** Shows the project structure under "REACT-LIBRARY": node\_modules, public, src (Auth, Images), layouts (BookCheckoutPage, HomePage), ManageLibraryPage (components).
- Code Editor:** The file "AddNewBook.tsx" is open at line 18. The code is as follows:

```
const [displaySuccess, setDisplaySuccess] = useState(false);

// get the category string from the user's choice in
function categoryField(value: string) {
    setCategory(value);
}

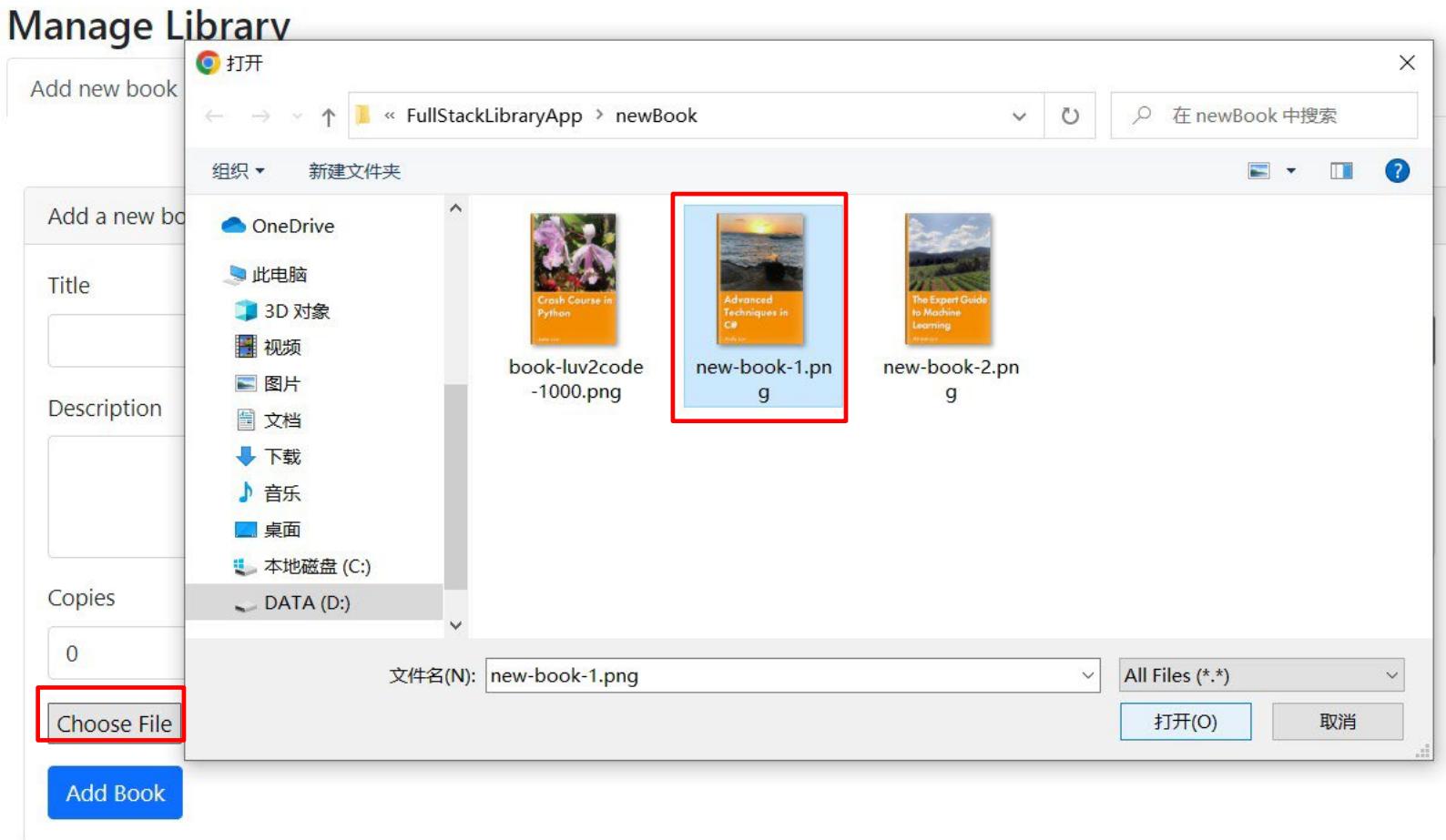
// dealing with the image of book
async function base64ConversionForImages(e: any) {
    console.log(e);
}
```

A red rectangular box highlights the entire block of code starting from the second line of the function definition (line 24) up to the final closing brace of the function (line 85).

## Step 2, add onChange listener to the file input.

```
⚙️ AddNewBook.tsx 9, U X
src > layouts > ManageLibraryPage > components > ⚙️ AddNewBook.tsx > [?] AddNewBook
78           </ul>
79           </div>
80       </div>
81   /* input box part */
82   <div className='col-md-12 mb-3'>
83       <label className='form-label'>Description</label>
84       <textarea className='form-control' id='exampleFormControlTexta
85           onChange={e => setDescription(e.target.value)} value={desc
86       </div>
87       <div className='col-md-3 mb-3'>
88           <label className='form-label'>Copies</label>
89           <input type='number' className='form-control' name='Copies' re
90               onChange={e => setCopies(Number(e.target.value))} value={c
91       </div>
92       <input type='file' onChange={e => base64ConversionForImages(e)}>
93   /* add book btn */
94   <div>
```

With admin user logged in status, go to our “add new book ” page, click on “Choose File”, open one image in our computer.  
Here, I choose “new-book-1.png”.



Then, besides the “Choose File” , we can find the file name of which we opened.  
Right click and choose “Inspect” or press “F12” to open the console,

The screenshot shows a web application titled "Manage Library". On the left, there's a form for adding a new book with fields for Title, Author, Category, Description, Copies, and a "Choose File" button. A red box highlights the "Choose File" button, and a blue callout bubble points to it with the text "the file name of which we opened". The file path "new-book-1.png" is visible in the input field of the "Choose File" button. On the right, the developer tools' "Console" tab is selected, showing a detailed log of an event object. A red box highlights the "Console" tab in the top bar.

```
Download the React DevTools for act-devtools
▶ Object
▶ SyntheticBaseEvent
▶ SyntheticBaseEvent
▼ SyntheticBaseEvent {_reactName: "input", target: input, ...} ⓘ
  bubbles: true
  cancelable: false
  currentTarget: null
  defaultPrevented: false
  eventPhase: 3
  isDefaultPrevented: function()
  isPropagationStopped: function()
  isTrusted: true
  nativeEvent: Event {isTrusted: true}
  target: input
  timeStamp: 1226277.200000003
  type: "change"
  _reactName: "onChange"
  _targetInst: null
  [[Prototype]]: Object
```

Scroll down and find “files”, we can find all the information about the image file we chose, such as file name, file size ,file type etc.

We can see files are a list , and the file we chose is the element 0.

## Manage Library

Add new book    Change quantity    Messages

Add a new book

Title	Author	Category
<input type="text"/>	<input type="text"/>	Category ▾

Description

Copies

0

Choose File  new-book-1.png

The screenshot shows the browser's developer tools open to the 'Console' tab. A file object has been selected, and its properties are displayed in an expandable tree view. The 'files' property is highlighted with a red box, and the first item in the 'files' array, labeled '0', is highlighted with a green box. The properties shown for the selected file are:

- draggable: false
- elementTiming: ""
- enterKeyHint: ""
- files: FileList
- 0: File
  - lastModified: 1673911503000
  - lastModifiedDate: Mon Jan 16 2023 18:25:03 G
    - [[Prototype]]: Object
  - name: "new-book-1.png"
  - size: 441494
  - type: "image/png"
  - webkitRelativePath: ""
  - [[Prototype]]: File
- length: 1
- [[Prototype]]: FileList
- firstChild: null
- firstElementChild: null
- form: form
- formAction: "http://localhost:3000/admin"
- formEnctype: ""
- formMethod: ""
- formNoValidate: false

Step 3, create a function getBase64().

3.1 create a new fileReader.

3.2 then use readAsDataURL(). this method is used to read the contents of the specified Blob or File.

When the read operation is finished, the readyState becomes DONE, and the loadend is triggered. At that time, the result attribute contains the data as a data: URL representing the file's data as a base64 encoded string.

3.3 Set the result of the reader to a piece of state that we can send over to our backend application.

3.4 if there is an error, print it in the console.

```
function getBase64(file: any) {
    // create a file readerread the contents of file of our ccomputer
    let reader = new FileReader();
    //read the contents of file,representing the file's data as a base64 encoded string.
    reader.readAsDataURL(file);
    // set the read result to our image
    reader.onload = function () {
        setSelectedImage(reader.result);
    };
    reader.onerror = function (error) {
        console.log('Error', error);
    }
}
```

Step 4, change the `console.log` into a if statement, which says if the index 0 element exists, call the function `getBase64()` and pass the index 0 file to the function.

```
// dealing with the image of book
async function base64ConversionForImages(e: any) {
    console.log(e);
}
```



```
// dealing with the image of book
async function base64ConversionForImages(e: any) {
    if (e.target.files[0]) {
        //passing this file into getBase64();
        getBase64(e.target.files[0]);
    }
}
```

---

---

---

## **7. SUBMIT NEW BOOK**



As of right now, we have no way of being able to submit any data to our spring boot application. That's the thing we're solving in this part.

Step 1, create a new async submitNewBook().

1.1 define the url string and specify the method and headers of add book request.

```
async function submitNewBook() {
    // url string
    const url = `http://localhost:8080/api/admin/secure/add/book`;
    // user is authenticated, all fields are not null:
    if (authState?.isAuthenticated && title !== '' && author !== '' && category !== 'Category'
        && description !== '' && copies >= 0) {
        // add the info input as a new book object's properties
        const book: AddBookRequest = new AddBookRequest(title, author, description, copies, category);
        book.img = selectedImage;
        const requestOptions = {
            method: 'POST',
            headers: {
                Authorization: `Bearer ${authState?.accessToken?.accessToken}`,
                'Content-Type': 'application/json'
            },
            body: JSON.stringify(book)
        };
    }
}
```

```
};

// fetch the data
const submitNewBookResponse = await fetch(url, requestOptions);
if (!submitNewBookResponse.ok) {
  throw new Error('Something went wrong!');
}

// reset all the input area to original empty state
setTitle('');
setAuthor('');
setDescription('');
setCopies(0);
setCategory('Category');
 setSelectedImage(null);
//display success banner
setDisplayWarning(false);
setDisplaySuccess(true);
} else {
  // display warning banner
  setDisplayWarning(true);
  setDisplaySuccess(false);
}
```

Step 2, pass the url and request info to the function await fetch() to call the API and fetch the data. Once we get the data, we set them as state.

Step 2, add onClick listener to “add book” button.  
Once the button is clicked, the submitNewBook function will be called.

```
/* add book btn */


<button type='button' className='btn btn-primary mt-3' onClick={submitNewBook}>
    Add Book
  </button>
</div>


```

With admin user logged in status, visit <http://localhost:3000/admin>, let's try input nothing but click on the “Add Book” button, then an alert message “All fields must be filled out” will appear.

The screenshot shows a web browser window titled "React App" with the URL "localhost:3000/admin". The main content is titled "Manage Library" and includes tabs for "Add new book", "Change quantity", and "Messages". A pink error message box at the top contains the text "All fields must be filled out". Below it is a form for adding a new book, which includes fields for Title, Author, Category, Description, Copies, and a file upload section. The "Title" field is empty. The "Category" field has a dropdown menu open. The "Copies" field contains the value "0". The "Choose File" button has the text "No file chosen". At the bottom is a blue "Add Book" button.

React App

localhost:3000/admin

Manage Library

Add new book Change quantity Messages

All fields must be filled out

Add a new book

Title Author Category

Description

Copies

0

Choose File No file chosen

Add Book

Then, input everything , click on the button.

React App    ×    +

localhost:3000/admin

Radio-Canada.ca |...    Canada's federal...    Sign in to your IR...    Chenelière Éducat...    Montreal    zdm    Nos incontournab...    TED: Ideas worth...    在线课程 - 时间自

## Manage Library

Add new book    Change quantity    Messages

All fields must be filled out

Add a new book

Title	Author	Category
Advanced Tech in C#	New Author1	BE ▾

Description

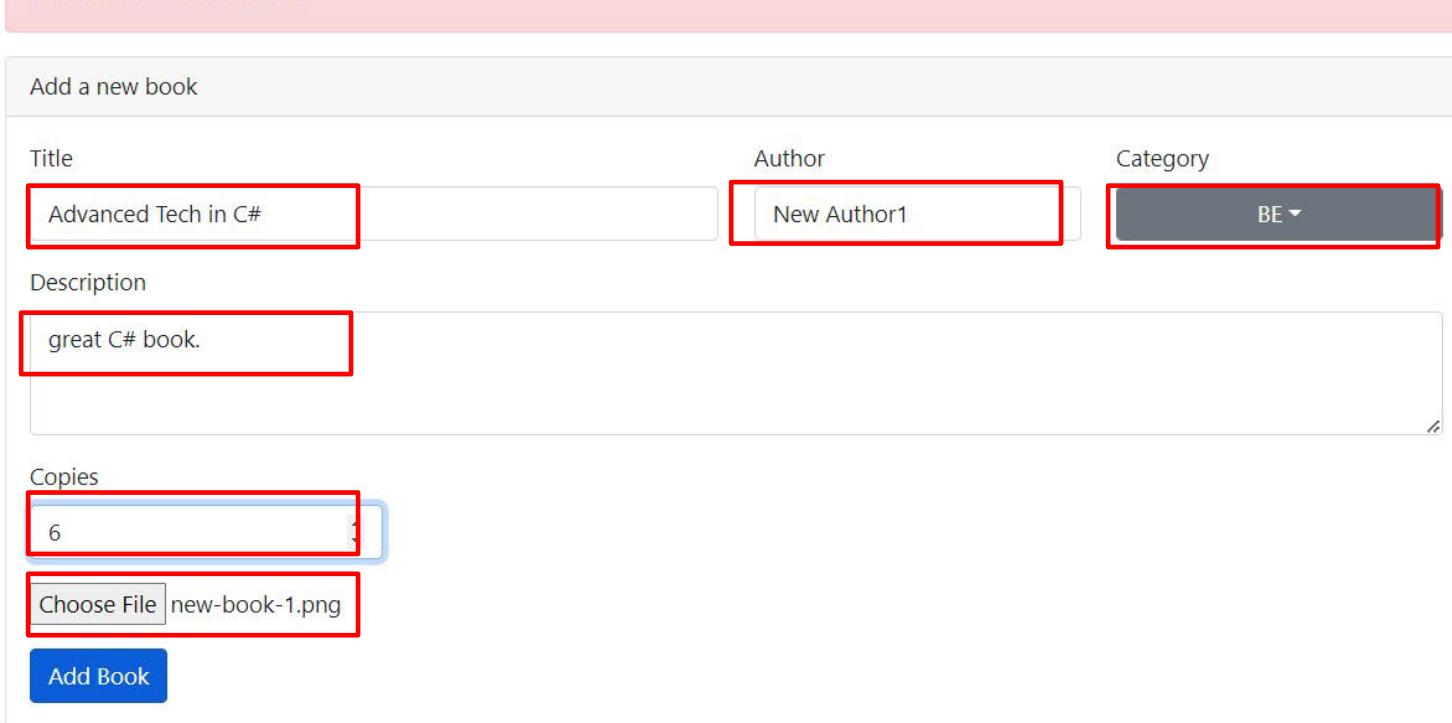
great C# book.

Copies

6

Choose File new-book-1.png

Add Book



We will get this successful page.

A screenshot of a web browser window titled "React App". The address bar shows "localhost:3000/admin". The main content area is titled "Manage Library" and contains a success message "Book added successfully" highlighted with a yellow border. Below it is a form for adding a new book, with fields for Title, Author, Category, Description, Copies, and a file upload input. A blue "Add Book" button is at the bottom.

React App

localhost:3000/admin

Manage Library

Add new book Change quantity Messages

Book added successfully

Add a new book

Title Author Category

Description

Copies

Choose File new-book-1.png

Add Book

If we go to the last page of search pages, we can see the book we added is there. Click on the “View details” button, go to the details page of the book.

React App

localhost:3000/search

Radio-Canada.ca |... Canada's federal... Sign in to your IR... Chenelière Éducat... Montreal zdm Nos incontournab... TED TED: Ideas worth... 在线课程 - 时间自...

vestibulum lectus a ante tempor tincidunt et sed orci. Proin maximus tortor in risus auctor efficitur. Phasellus quam mauris, laoreet et feugiat ac, imperdiet at quam. Nullam sollicitudin nec diam vel finibus.

Luv, Juan

**Introduction to Python**

Pellentesque varius aliquam lacus quis rhoncus. Nam a dui lectus. Vestibulum libero elit, ultricies sit amet sagittis eu, molestie at velit. Donec tincidunt tempus magna, quis facilisis libero elementum non. Sed velit lacus, laoreet sed augue fermentum, sagittis convallis metus. Sed nec est at massa venenatis aliquet. Donec pretium interdum fringilla. Sed ornare tellus enim, a tincidunt libero dictum vitae. Proin bibendum posuere dui. Donec sagittis neque massa, sed semper nulla vehicula at.

**New Author1**

**Advanced Tech in C#**

great C# book.

View details

View details

First Page 3 4 5 Last Page

The available copy number and total copies number is as exactly as we set.  
If we click on the “checkout” button, and try to leave a review.

React App

localhost:3000/checkout/23

JAC Read Home Search Books Shelf

Book image we have chosen

Advanced Techniques in C#

New Author1

great C# book.

5 stars

Available

6 copies 6 available

Checkout

This number can change until placing order has been complete.

Leave a review? ▾

5 stars

Book info we have input

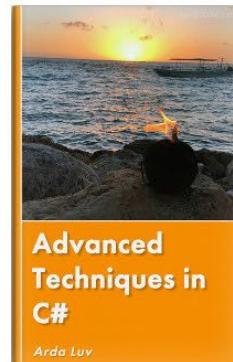
Book numbers we have set

### Latest Reviews:

Currently there are no reviews for this book

We will get this page:

A screenshot of a web browser window titled "React App" showing a book details page. The URL is "localhost:3000/checkout/23". The page header includes links to Radio-Canada.ca, Canada's federal..., Sign in to your IR..., Chenelière Éducat..., Montreal, zdm, Nos incontournab..., and TED. Below the header, there are navigation links: JAC Read, Home, Search Books, Shelf.



## Advanced Tech in C#

New Author1

great C# book.



1/5 books checked out

### Available

6 copies      5 available

Book checked out. Enjoy!

This number can change until placing order has been complete.

Thank you for your review!

Review is here

### Latest Reviews:

adminuser@email.com

February 16, 2023

great book



Read all reviews.

Number of checked out changed

Number of available changed

No any invitation of leaving a review"

If we go to our database, open the “book” table, we can find the added book too! Now, we have totally 23 books in our database.