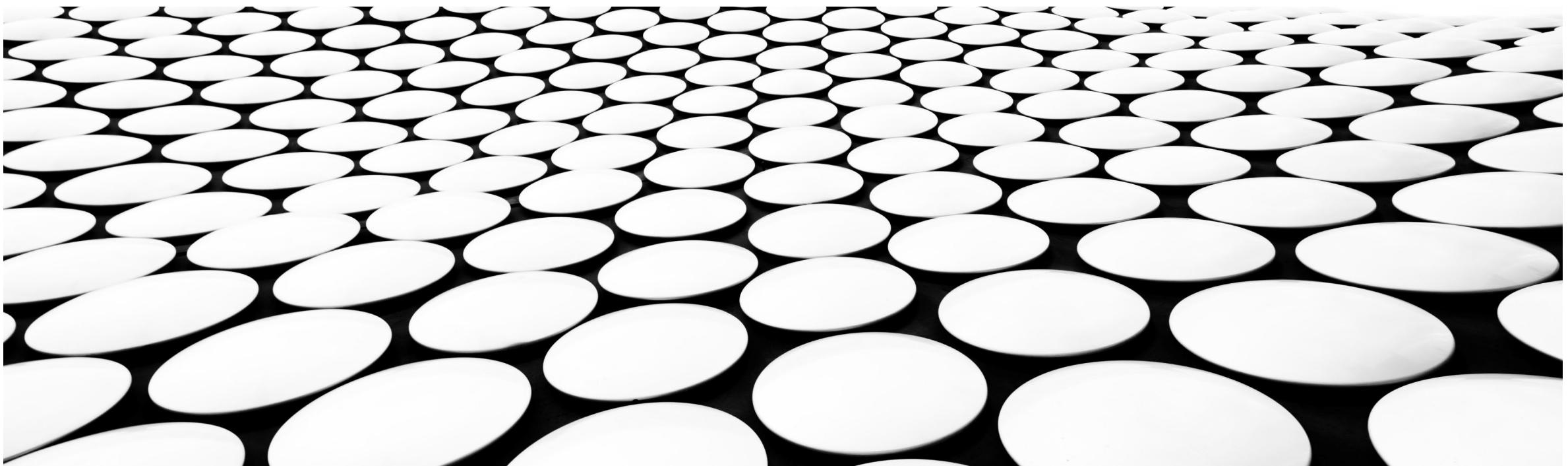
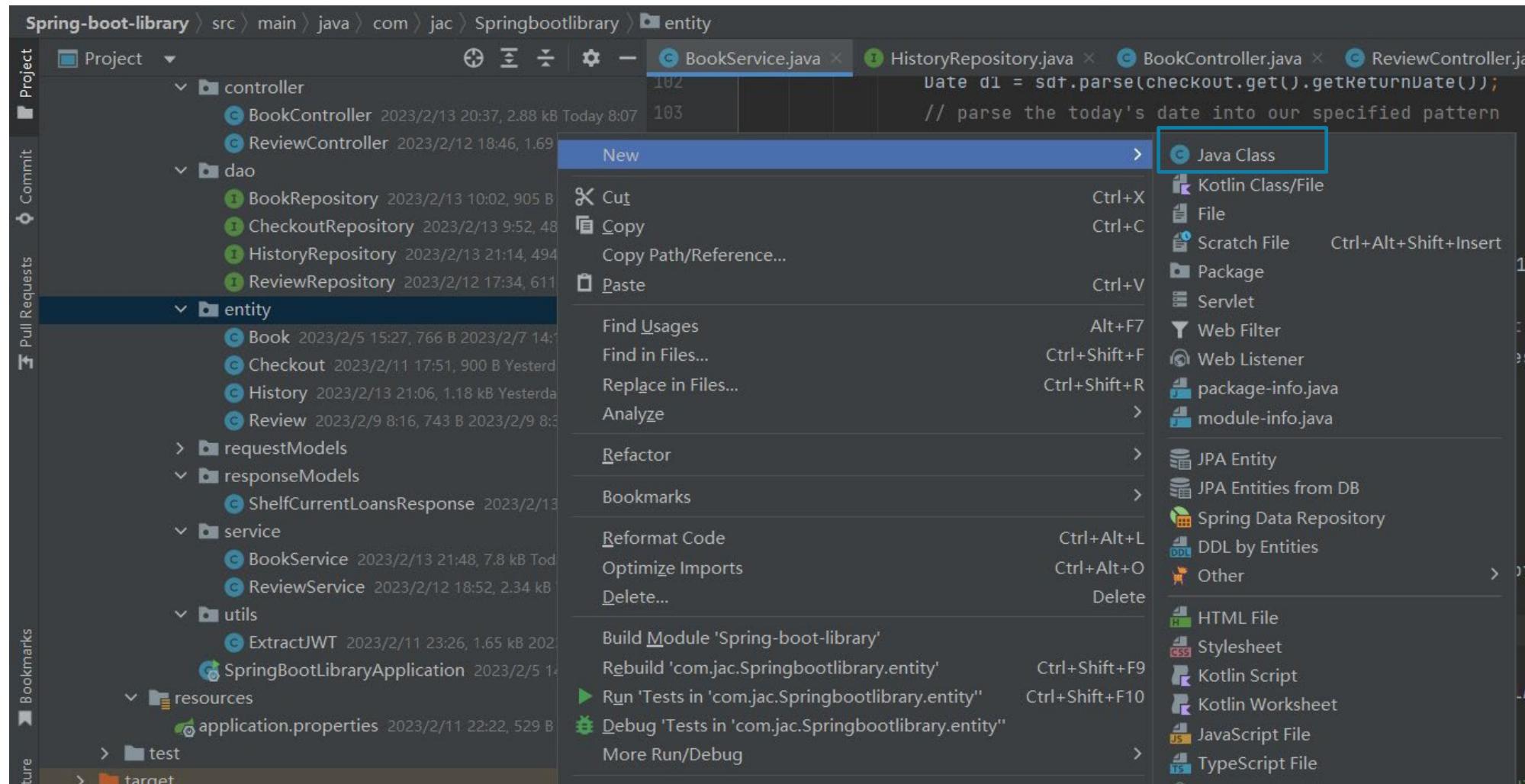

S18 LIBRARY SERVICE



1.SPRINGBOOT MESSAGE ENTITY

Step 1, create Message class file in entity package.
Right click on our entity package, choose new and java class.



Choose “Class”, and input the class name “Message”.

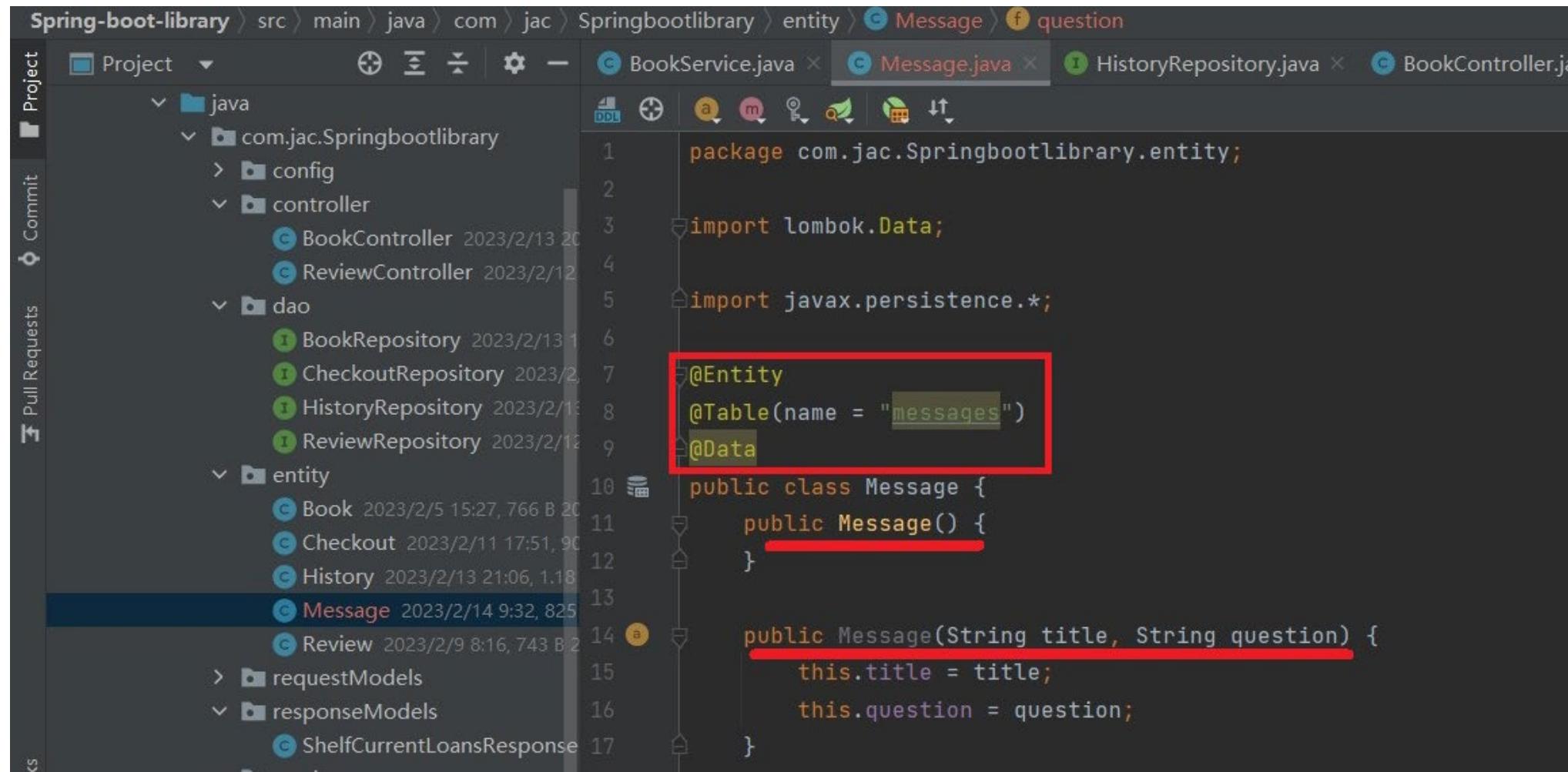
The screenshot shows a Java code editor with a context menu open. The menu is titled "New Java Class" and contains options: "Message" (selected and highlighted with a red box), "Class", "Interface", "Enum", and "@Annotation". The code editor displays a portion of a Java file with imports and a main method. The main method contains logic for finding a book by ID, validating a checkout, and updating book availability.

```
TimeUnit time = TimeUnit.DAYS;
// calculate the difference and convert it to days
long difference_In_Time = time.convert( sourceDuration: e
    TimeUnit.MILLISECONDS);
// passing the book and the difference of time(converted)
shelfCurrentLoansResponses.add(new ShelfCurrentLoansR
}

New Java Class
Message
Class
Interface
Enum
@Annotation

Optional<Book> book = bookRepository.findById(bookId);
// grab the checkout by user email and book id
Checkout validateCheckout = checkoutRepository.findByUserEmail(bookId) throws Exce
// if not exists:
if (!book.isPresent() || validateCheckout == null) {
    throw new Exception("Book does not exist or not checked out");
}
// make the number of copies available increase by 1
book.get().setCopiesAvailable(book.get().getCopiesAvailable()
// save the book into our book repository
bookRepository.save(book.get());
```

Step2, add annotations and create the constructors for this class.
for the annotation `@Table(name="messages")`, "messages" is the name of our corresponding table.
`@Data` will import Lombok, then we don't need do write getter and setter manually.



```
package com.jac.Springbootlibrary.entity;
import lombok.Data;
import javax.persistence.*;
@Entity
@Table(name = "messages")
@Data
public class Message {
    public Message() {
    }
    public Message(String title, String question) {
        this.title = title;
        this.question = question;
    }
}
```

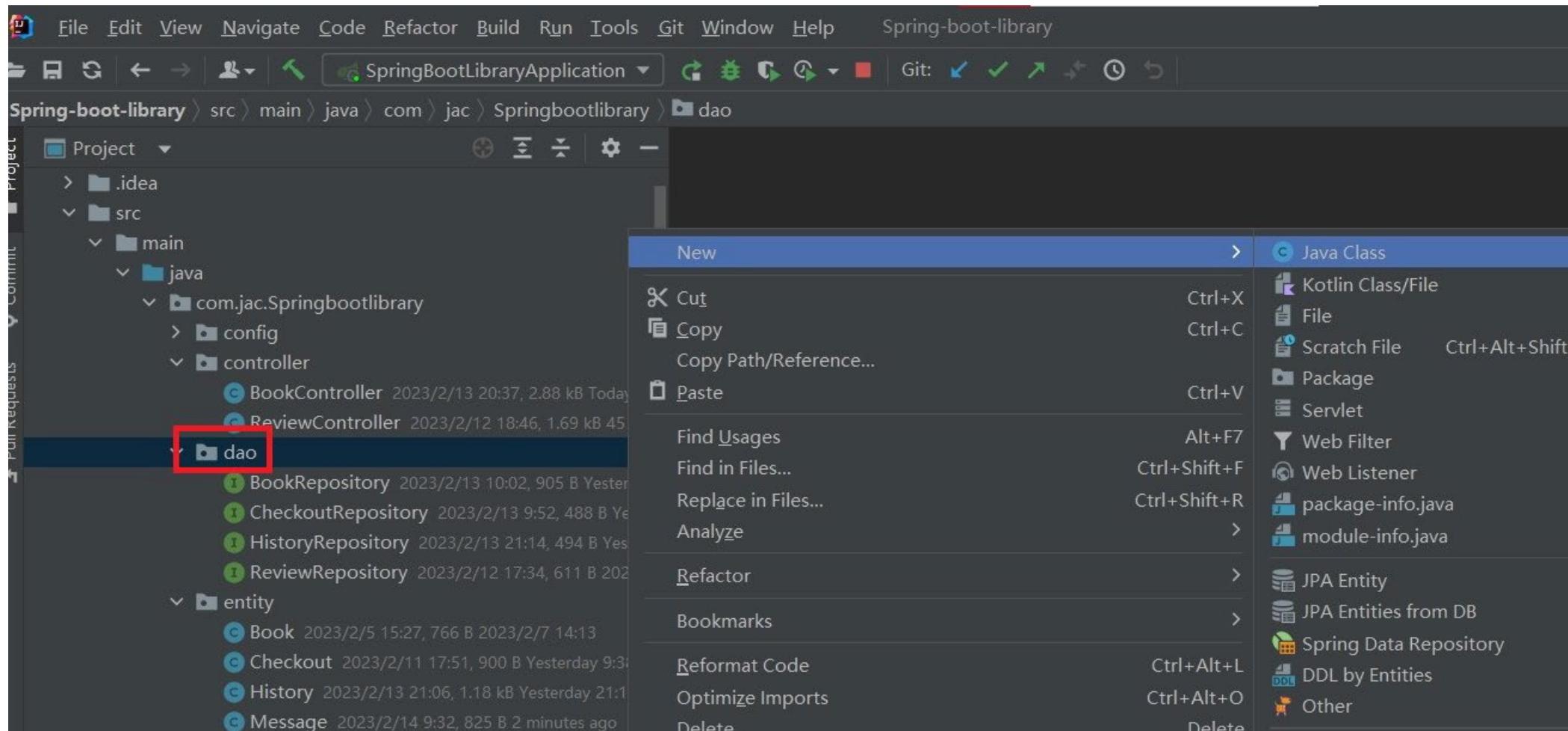
```
16     this.question = question;
17 }
18
19 @Id
20 @GeneratedValue(strategy = GenerationType.IDENTITY)
21 @Column(name = "id")
22 private Long id;
23
24 @Column(name = "user_email")
25 private String userEmail;
26
27 1 usage
28 @Column(name = "title")
29 private String title;
30
31 1 usage
32 @Column(name = "question")
33 private String question;
34
35 @Column(name = "admin_email")
36 private String adminEmail;
37
38 @Column(name = "response")
39 private String response;
40
41 @Column(name = "closed")
42 private boolean closed;
```

Step 4, create properties which are mapping to columns in our table "messages".

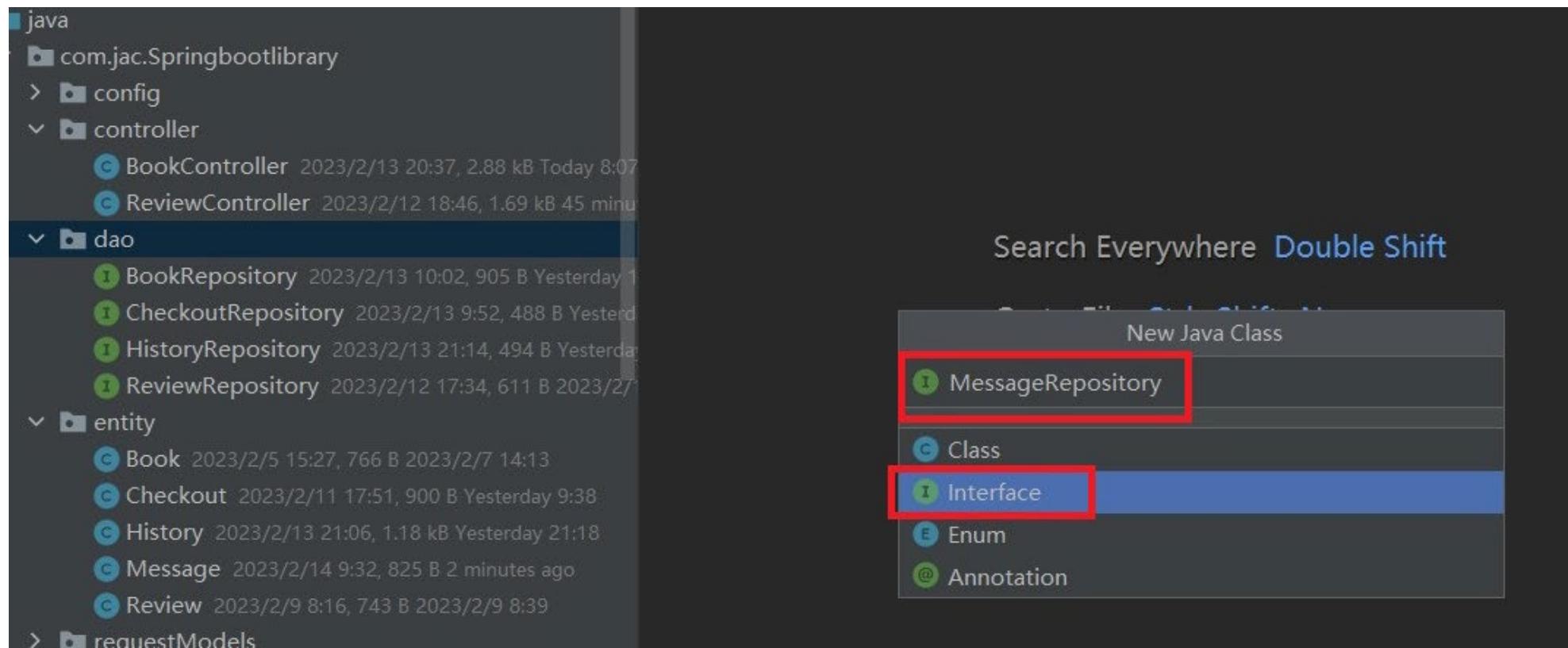
2. SPRINGBOOT MESSAGE REPOSITORY

Step1, Create a new interface MessageRepository.

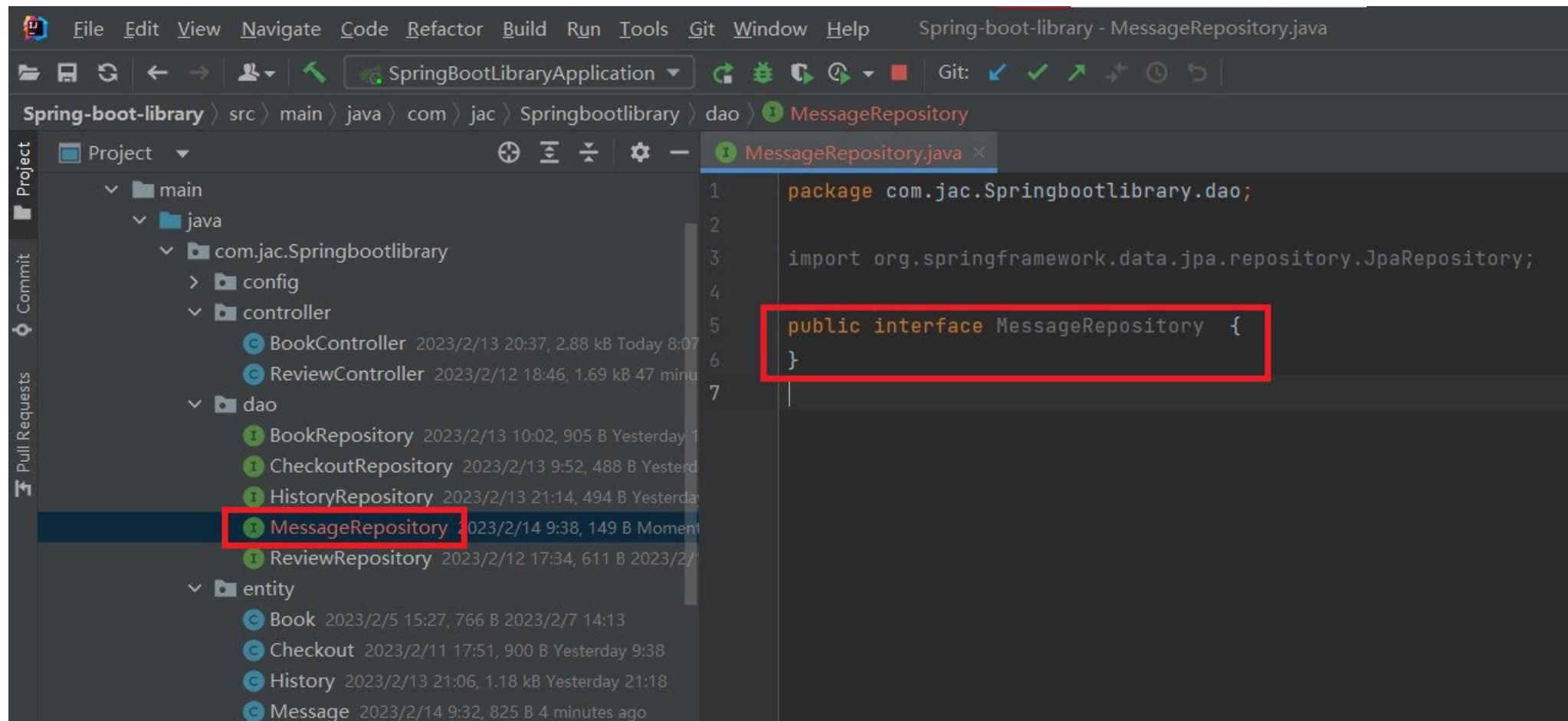
Right click dao directory, choose new ,then Java class.



Choose an interface, input name MessageRepository.



We will get the class as below:



The screenshot shows a Java IDE interface with the following details:

- File Bar:** File Edit View Navigate Code Refactor Build Run Tools Git Window Help
- Title Bar:** Spring-boot-library - MessageRepository.java
- Toolbar:** Standard icons for file operations.
- Project Bar:** SpringBootLibraryApplication
- Code Editor:** The file `MessageRepository.java` is open. The code is as follows:

```
package com.jac.Springbootlibrary.dao;

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

public interface MessageRepository { }
```
- Project Explorer:** Shows the project structure under `Spring-boot-library`:
 - `main` folder
 - `java` folder
 - `com.jac.Springbootlibrary` package
 - `config` folder
 - `controller` folder
 - `BookController`
 - `ReviewController`
 - `dao` folder
 - `BookRepository`
 - `CheckoutRepository`
 - `HistoryRepository`
 - `MessageRepository` (highlighted with a red box)
 - `ReviewRepository`
 - `entity` folder
 - `Book`
 - `Checkout`
 - `History`
 - `Message`
 - Git Status:** Shows green checkmarks indicating successful operations.

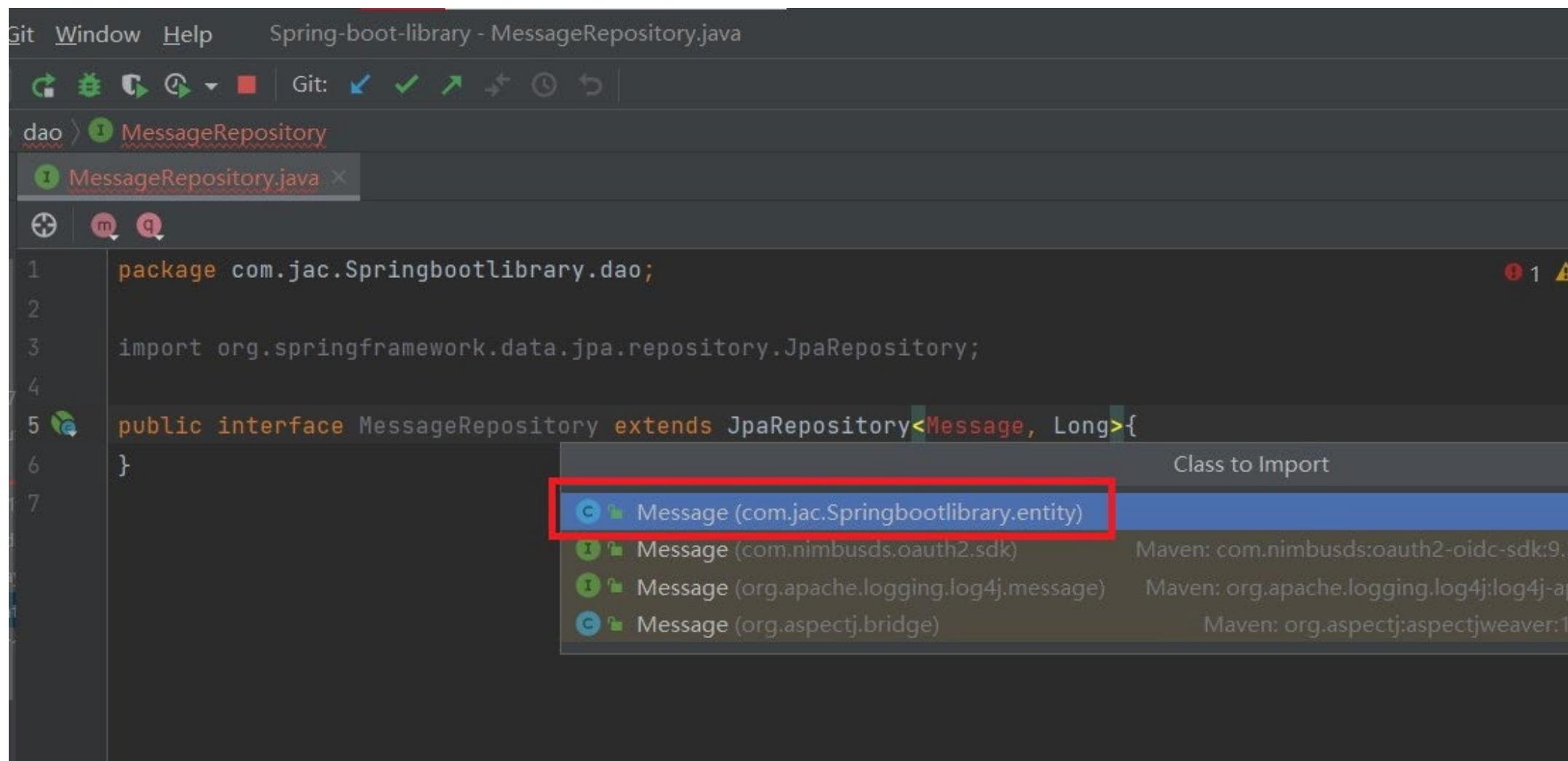
Step 2, add “extends JpaRepository<Message, Long>”.
Click on the “import class” to import Message class.

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

```
1 package com.jac.Springbootlibrary.dao;
2
3 import org.springframework.d①com.jac.Springbootlibrary.entity.Message? (multiple choices...) Alt+Enter
4
5 public interface MessageRepository extends JpaRepository<Message, Long>
6 }
```

A code completion tooltip is displayed at the end of the line 'MessageRepository'. The tooltip contains the text '@com.jac.Springbootlibrary.entity.Message?' followed by '(multiple choices...) Alt+Enter'. Below the tooltip, a message says 'Cannot resolve symbol 'Message''. At the bottom of the tooltip, there are two buttons: 'Import class' (highlighted with a red box) and 'More actions... Alt+Enter'.

Make sure that we choose the class which we have created in our entity package.



The screenshot shows an IDE interface with a dark theme. The menu bar at the top includes 'Git', 'Window', and 'Help'. The current file is 'Spring-boot-library - MessageRepository.java' located in the 'dao' package. The code editor shows the following Java code:

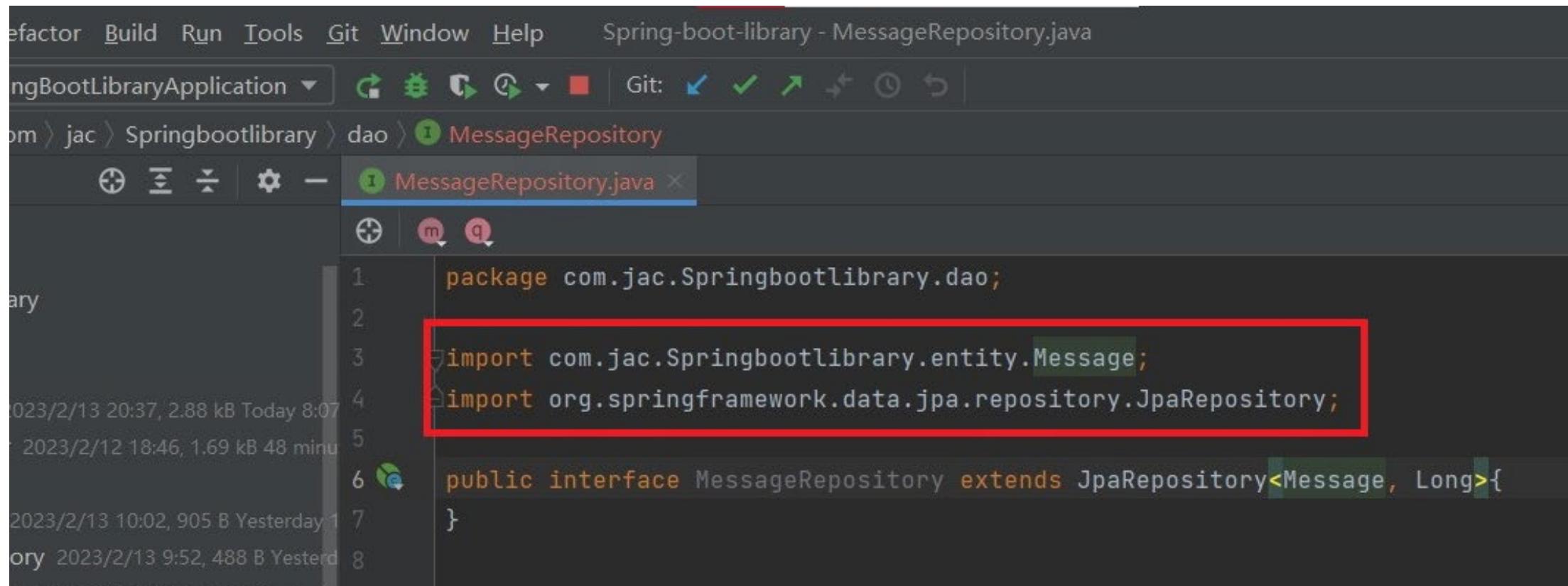
```
1 package com.jac.Springbootlibrary.dao;  
2  
3 import org.springframework.data.jpa.repository.JpaRepository;  
4  
5 public interface MessageRepository extends JpaRepository<Message, Long>{  
6 }  
7
```

A code completion dropdown is open at the end of the line 'extends JpaRepository<Message, Long>'. The dropdown is titled 'Class to Import' and contains four entries, each with a small icon and a tooltip showing the class name and its source Maven dependency:

- Message (com.jac.Springbootlibrary.entity) (highlighted with a red box)
- Message (com.nimbusds.oauth2.sdk)
- Message (org.apache.logging.log4j.message)
- Message (org.aspectj.bridge)

The first entry, 'Message (com.jac.Springbootlibrary.entity)', is highlighted with a red box, indicating it is the correct choice to match the entity class defined in the 'entity' package.

Check the import part is exactly as below:



```
package com.jac.Springbootlibrary.dao;

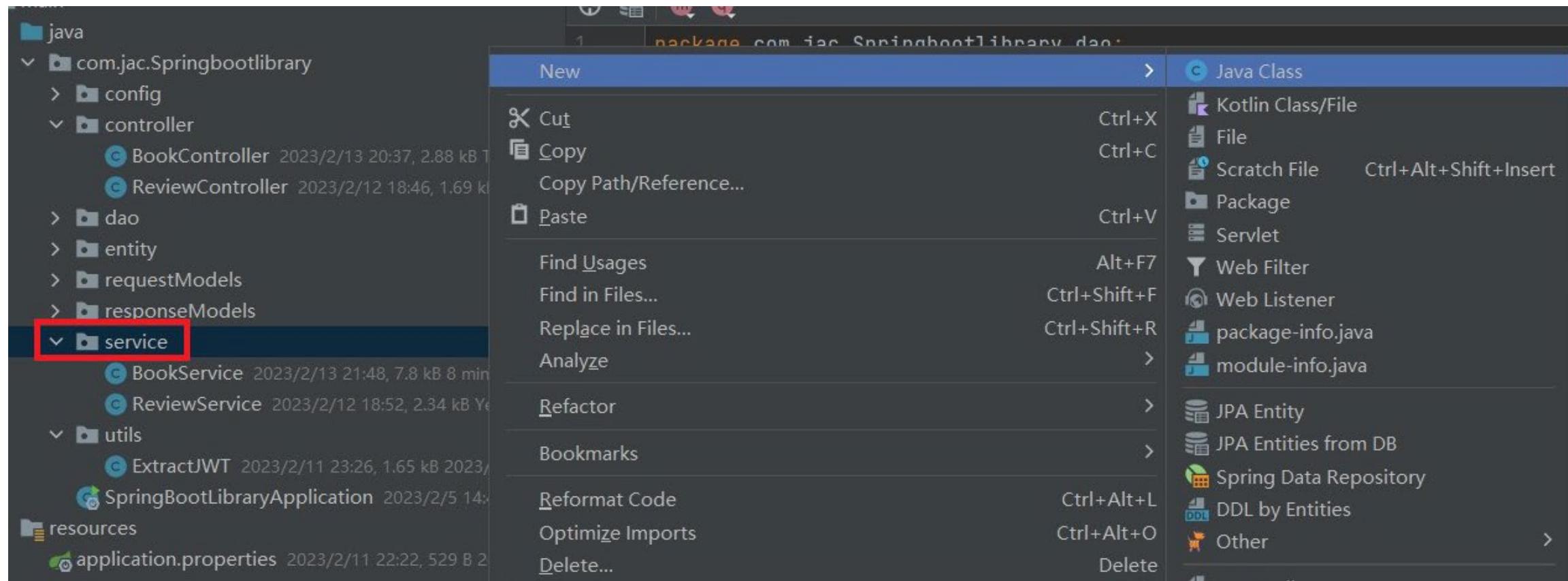
import com.jac.Springbootlibrary.entity.Message;
import org.springframework.data.jpa.repository.JpaRepository;

public interface MessageRepository extends JpaRepository<Message, Long>{}
```

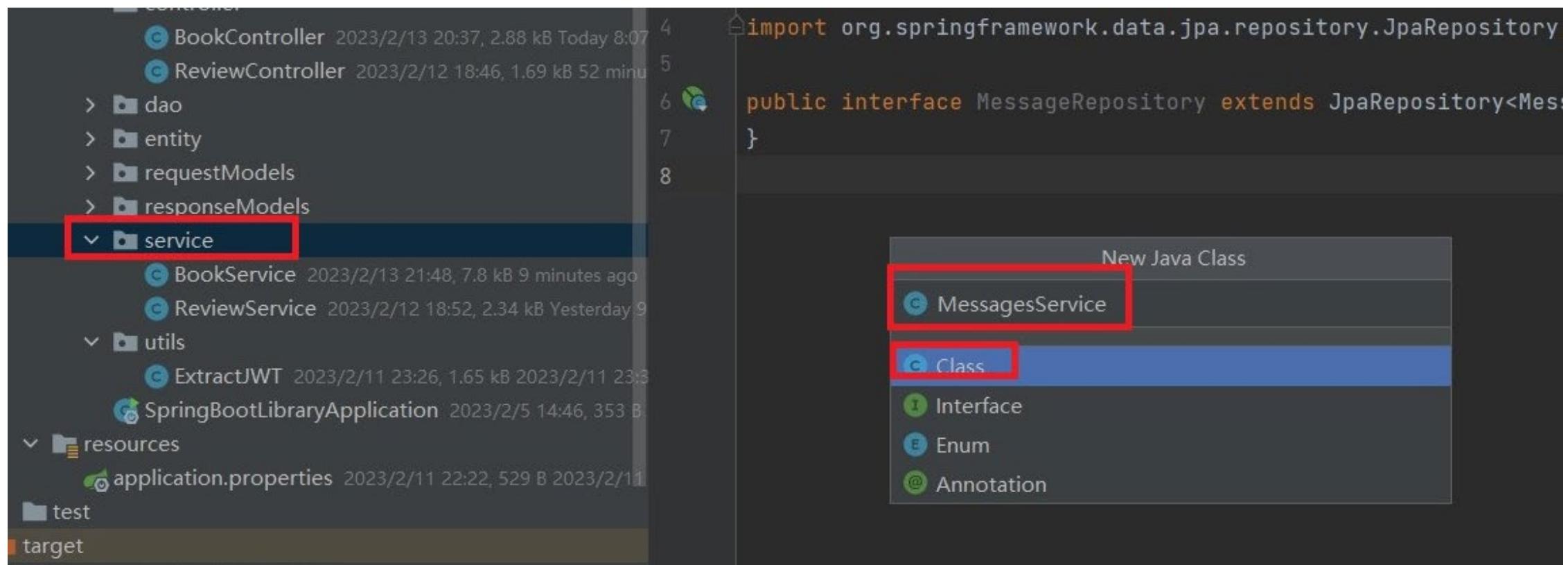
3.SPRINGBOOT MESSAGE SERVICE

Step 1, create MessagesService class.

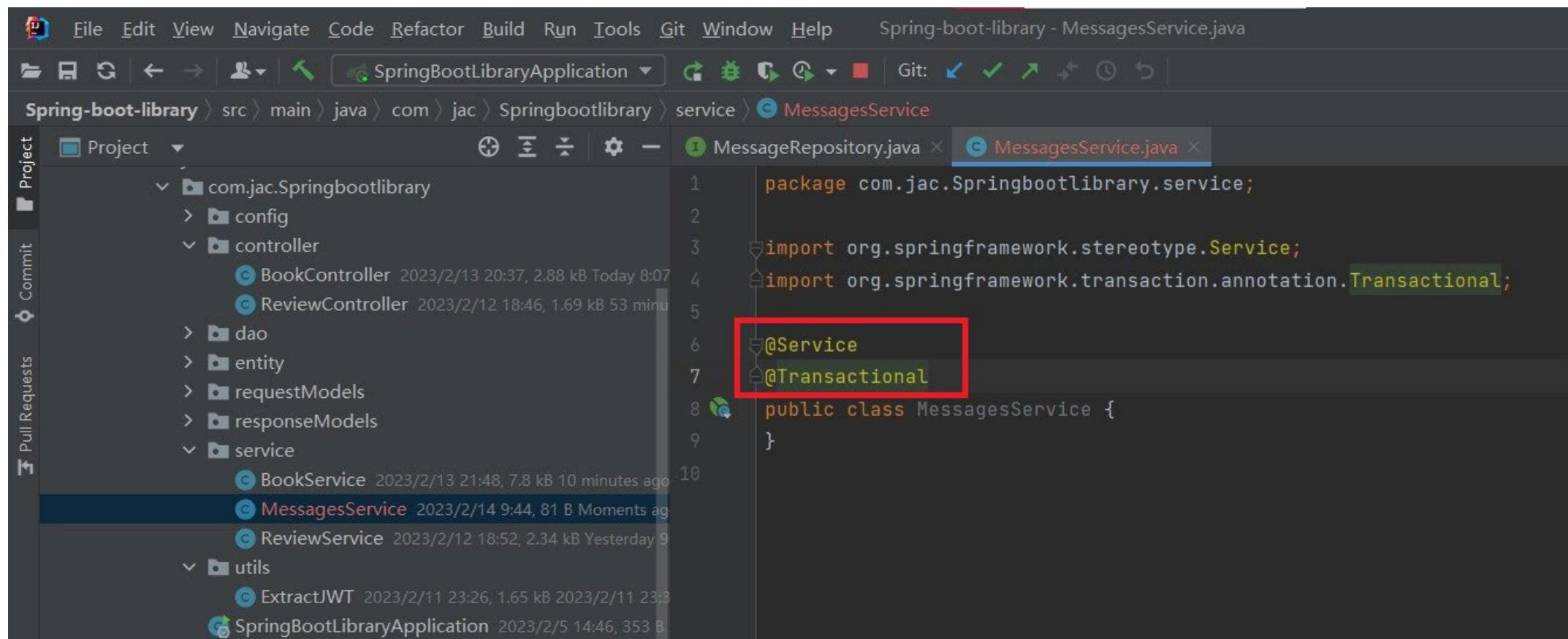
Right click service directory, and choose new, Java class.



Choose “Class”, input name “MessagesService”.

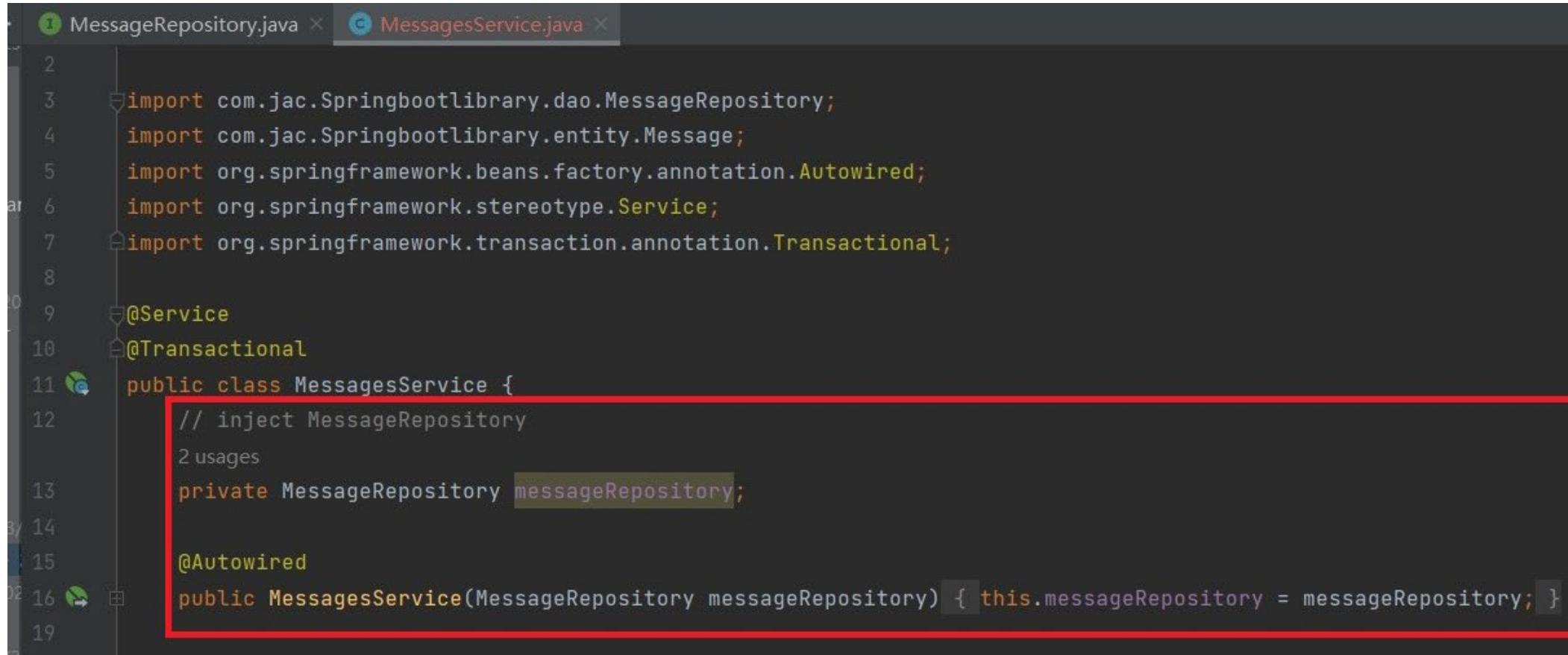


Step 2, add annotation @Service ,@Transactional



```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help Spring-boot-library - MessagesService.java
SpringBootLibraryApplication SpringBootLibraryApplication
Spring-boot-library > src > main > java > com > jac > Springbootlibrary > service > MessagesService.java
Project Commit Pull Requests
com.jac.Springbootlibrary
  > config
  < controller
    BookController 2023/2/13 20:37, 2.88 kB Today 8:07
    ReviewController 2023/2/12 18:46, 1.69 kB 53 minu
  > dao
  > entity
  > requestModels
  > responseModels
  < service
    BookService 2023/2/13 21:48, 7.8 kB 10 minutes ago
    MessagesService 2023/2/14 9:44, 81 B Moments ag
    ReviewService 2023/2/12 18:52, 2.34 kB Yesterday 9
  < utils
    ExtractJWT 2023/2/11 23:26, 1.65 kB 2023/2/11 23:3
SpringBootLibraryApplication 2023/2/5 14:46, 353 B
MessageRepository.java MessagesService.java
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 MessagesService {
9 }
```

Step3 inject MessageRepository and @Autowire it to our constructor.



The screenshot shows a code editor with two tabs: 'MessageRepository.java' and 'MessagesService.java'. The 'MessagesService.java' tab is active, displaying the following Java code:

```
1 import com.jac.Springbootlibrary.dao.MessageRepository;
2 import com.jac.Springbootlibrary.entity.Message;
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Service;
5 import org.springframework.transaction.annotation.Transactional;
6
7 @Service
8 @Transactional
9 public class MessagesService {
10     // inject MessageRepository
11     2 usages
12     private MessageRepository messageRepository;
13
14     @Autowired
15     public MessagesService(MessageRepository messageRepository) { this.messageRepository = messageRepository; }
16 }
```

A red rectangular box highlights the constructor line at the bottom of the code, specifically the part: `public MessagesService(MessageRepository messageRepository) { this.messageRepository = messageRepository; }`. This indicates that the code is being annotated or checked by an IDE for autowiring.

Step 4, create a postMessage(), passing messageRequest and userEmail to it.

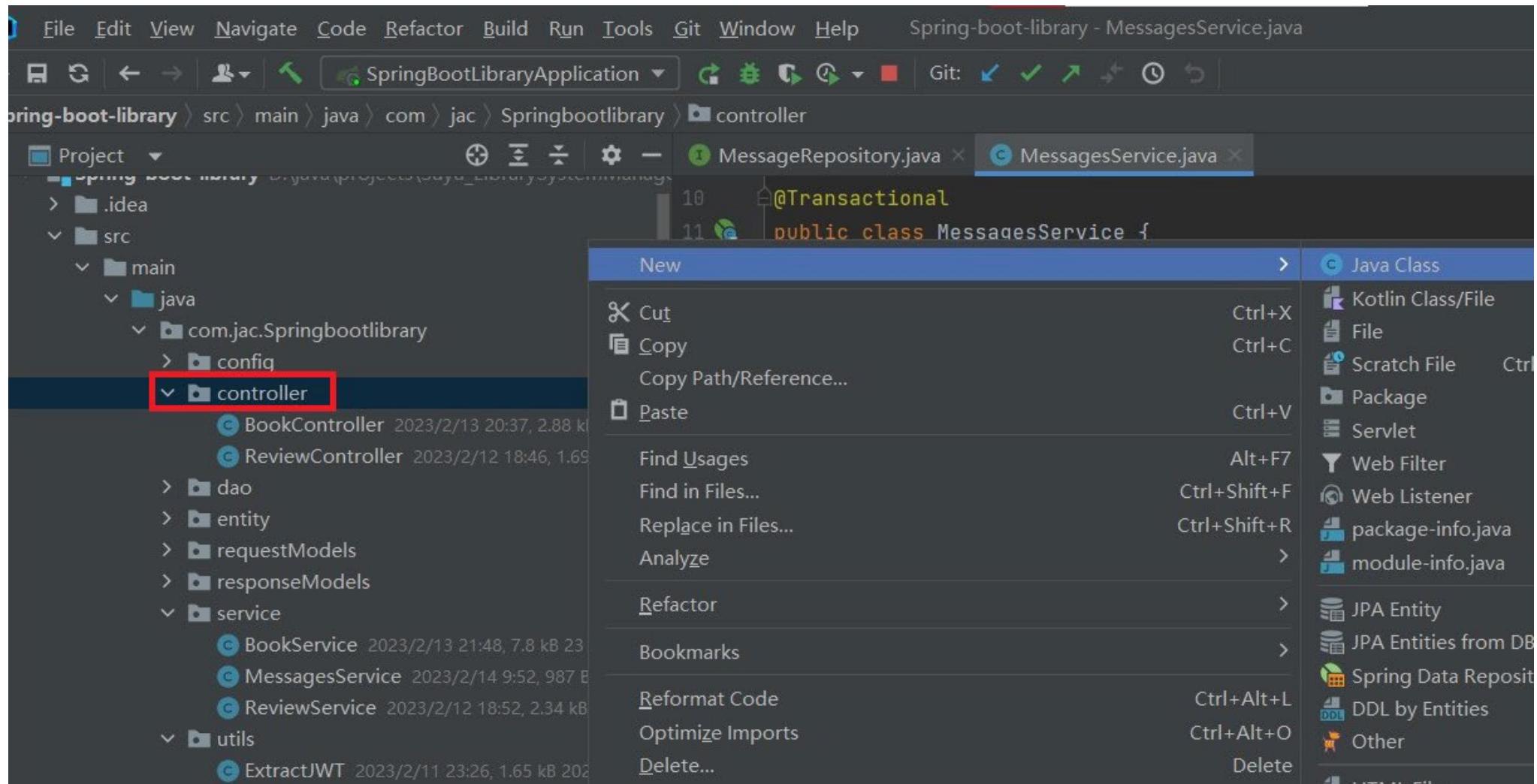
```
er 20 9
ller 10
    @Service
    @Transactional
    public class MessagesService {
        // inject MessageRepository
        2 usages
        private MessageRepository messageRepository;
2023/14
rice 15
@Autowired
e 202 16
    public MessagesService(MessageRepository messageRepository) { this.messageRepository = messageRepository; }
19
23/20
yAp 21
    @
es 22
    public void postMessage(Message messageRequest, String userEmail) {
23
        // get the message from the request and save it into messageRepository
        Message message = new Message(messageRequest.getTitle(), messageRequest.getQuestion());
24
        message.setUserEmail(userEmail);
        messageRepository.save(message);
25
    }
023/26
2 kB 27
kB 28
3 kB 29
    }
```

4.SPRINGBOOT MESSAGE CONTROLLER



Step 1, Create Message Controller class.

Right click controller directory and choose new , Java class.



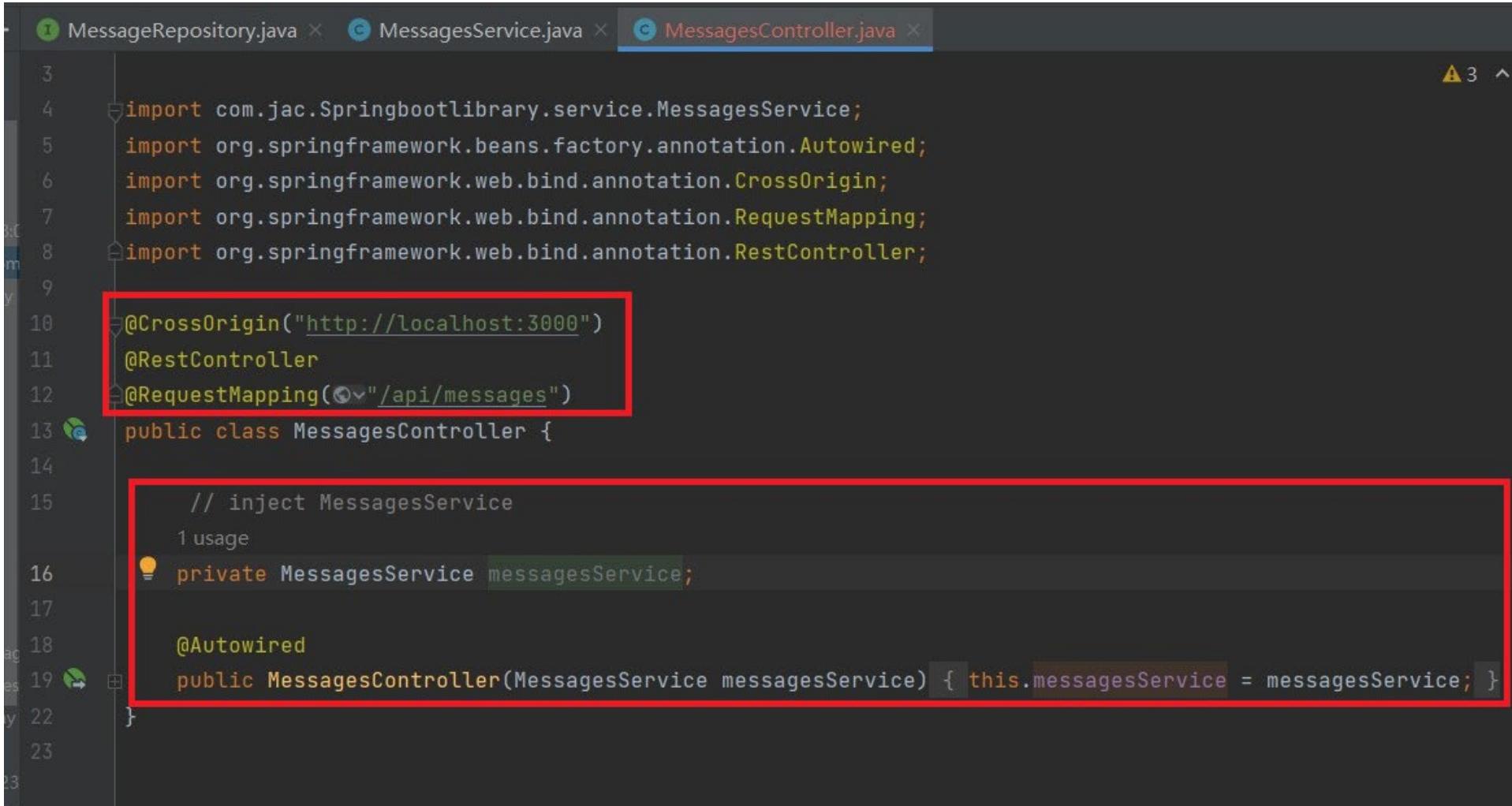
Choose “Class”, input name “MessageController”.

The screenshot shows a code editor with Java code. On the left, there's a file tree with packages like controller, dao, entity, requestModels, responseModels, service, and utils. The service package contains classes BookService, MessagesService, and ReviewService. The utils package contains ExtractJWT. The resources folder contains application.properties. The main code area has a line starting with 'public'. A code completion dropdown is open, showing options: New Java Class, MessagesController (highlighted with a red box), Class (selected and highlighted with a blue box), Interface, Enum, and Annotation. The 'MessagesController' option is also highlighted with a red box.

```
15 @Autowired
16 public MessagesService(MessageRepository messageRepos
17
18 //post message
19
20 public
21     @
22     // Mes
23     Me
24     mes
25     mes
26 }
27
28 }
29 }
```

New Java Class
MessagesController
Class
Interface
Enum
Annotation

Step 2, add annotation to the class, then inject MessagesService and Autowire the constructor to the class.



The screenshot shows a Java IDE interface with three tabs at the top: MessageRepository.java, MessagesService.java, and MessagesController.java. The MessagesController.java tab is active. The code is as follows:

```
import com.jac.Springbootlibrary.service.MessagesService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RestController;

@CrossOrigin("http://localhost:3000")
@RestController
@RequestMapping("/api/messages")
public class MessagesController {

    // inject MessagesService
    private MessagesService messagesService;

    @Autowired
    public MessagesController(MessagesService messagesService) { this.messagesService = messagesService; }

}
```

A red box highlights the annotations on the class definition (Lines 10-12) and the constructor injection (Line 19). A yellow warning icon is visible in the top right corner of the editor area.

Step 4, create a post message API with post method.

```
  @RequestMapping("/api/messages")
  public class MessagesController {

    // inject MessagesService
    2 usages
    private MessagesService messagesService;

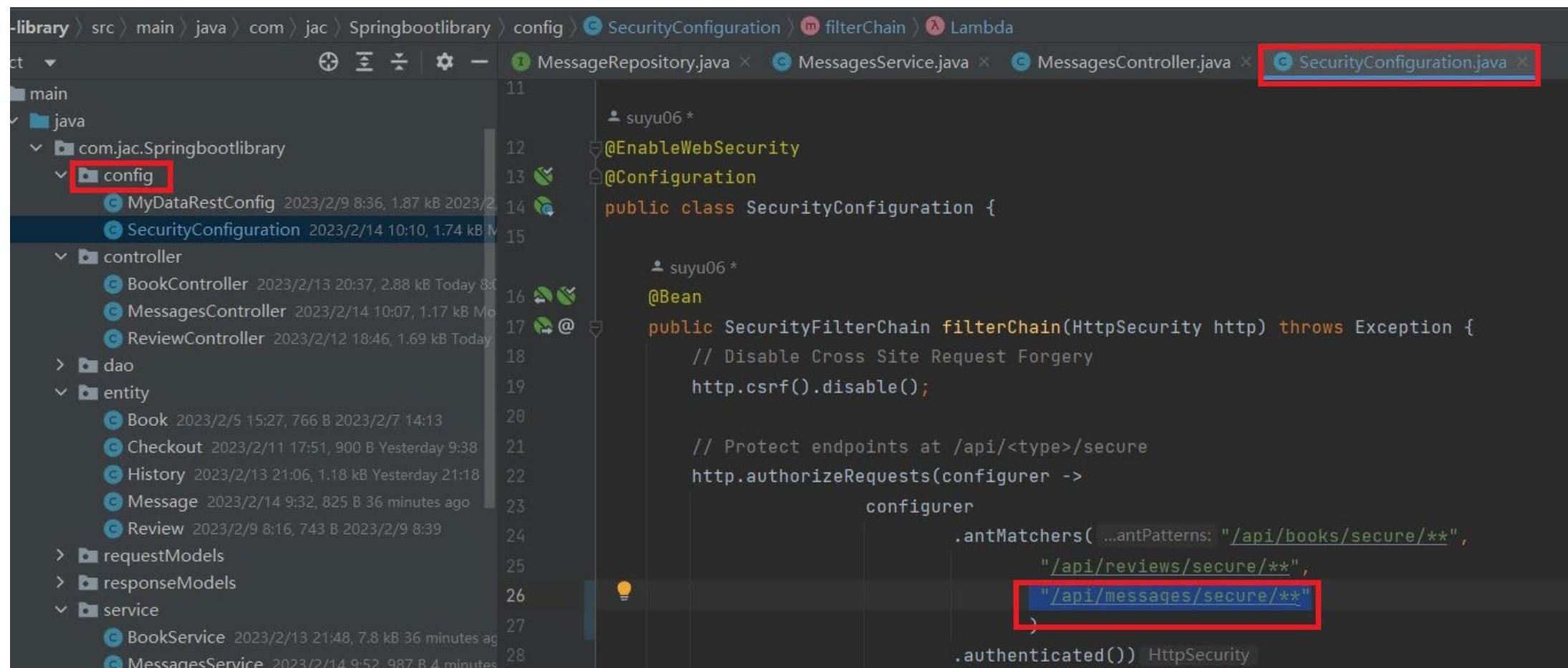
    @Autowired
    public MessagesController(MessagesService messagesService) { this.messagesService = mess

    // post a message
    @PostMapping("/secure/add/message")
    public void postMessage(@RequestHeader(value="Authorization") String token,
                           @RequestBody Message messageRequest) {
      // get user email from token
      String userEmail = ExtractJWT.payloadJWTExtraction(token, extraction: "\"sub\"");
      //call postMessage() in messagesService to save the message to our database
      messagesService.postMessage(messageRequest, userEmail);
    }
  }
```

5.SPRINGBOOT MESSAGE SECURITY AND TEST



Under “config” package, SecurityConfiguration class, In filterChain(), add “ /api/messages/secure/** ” . Which means all the API begin with this string , only can be visited by authenticated user.



```
library > src > main > java > com > jac > Springbootlibrary > config > SecurityConfiguration > filterChain > Lambda
ct - MessageRepository.java < MessagesService.java < MessagesController.java < SecurityConfiguration.java
main
java
com.jac.Springbootlibrary
config
MyDataRestConfig 2023/2/9 8:36, 1.87 kB 2023/2/9 8:36, 1.87 kB
SecurityConfiguration 2023/2/14 10:10, 1.74 kB 2023/2/14 10:10, 1.74 kB
controller
BookController 2023/2/13 20:37, 2.88 kB Today 8:36, 2.88 kB
MessagesController 2023/2/14 10:07, 1.17 kB Today 8:36, 1.17 kB
ReviewController 2023/2/12 18:46, 1.69 kB Today 8:36, 1.69 kB
dao
entity
Book 2023/2/5 15:27, 766 B 2023/2/7 14:13
Checkout 2023/2/11 17:51, 900 B Yesterday 9:38
History 2023/2/13 21:06, 1.18 kB Yesterday 21:18
Message 2023/2/14 9:32, 825 B 36 minutes ago
Review 2023/2/9 8:16, 743 B 2023/2/9 8:39
requestModels
responseModels
service
BookService 2023/2/13 21:48, 7.8 kB 36 minutes ago
MessagesService 2023/2/14 9:52, 987 B 4 minutes ago
```

```
11     * suyu06 *
12     * @EnableWebSecurity
13     * @Configuration
14     public class SecurityConfiguration {
15
16         * suyu06 *
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                     )
29                     .authenticated() HttpSecurity
```

Let's test in Postman.

We input `http://localhost:8080/api/messages/secure/add/message`, choose "Post".

Click on "body", choose "raw", "JSON", input the content of title and question, then click on "Send" button. We will get a "401 unauthorized" status. Because the user is not authenticated. We need to have an authorization.

The screenshot shows the Postman interface for a POST request to `http://localhost:8080/api/messages/secure/add/message`. The 'Body' tab is selected, with 'raw' and 'JSON' also highlighted with red boxes. The JSON payload is:

```
1 {"title": "Example title for message",  
2 "question": "what's your question?"}
```

The response details show a **401 Unauthorized** status with the following description:

Similar to 403 Forbidden, but specifically for use when authentication is possible but has failed or not yet been provided. The response must include a WWW-Authenticate header field containing a challenge applicable to the requested resource.

Use email and password log into our website, then press F12, click on the “accessToken”, copy the token string.

The screenshot shows a web browser window with developer tools open. The main content area displays a loan application interface titled "JAC Read". It has tabs for "Loans" and "Your History", with "Loans" selected. Below the tabs, it says "Current Loans:" and shows a small image of a set of concrete steps. The developer tools' "Console" tab is active, showing the following output:

```
react-dom.development.js:29840
Download the React DevTools for a better development experience: https://reactjs.org/link/react-devtools
Navbar.tsx:14
{accessToken: {...}, idToken: {...}, refreshToken: undefined, isAuthenticated: true}
  ▼ accessToken:
    accessToken: "eyJraWQiOiJFeW1GaFZESVk4TWF0cVkJ4TmthWTREQnJ1ZFdYQWM5dFV1UXpF
      authorizeUrl: "https://dev-65756343.okta.com/oauth2/default/v1/authorize"
      claims: {ver: 1, jti: 'AT.fhan-2cOTwuSbdsC2TmveaD1ZDtBuh00_n7ow7vGDfg', isExpiresAt: 1676390266
      scopes: (3) ['profile', 'email', 'openid']
      tokenType: "Bearer"
      userinfoUrl: "https://dev-65756343.okta.com/oauth2/default/v1/userinfo"
      [[Prototype]]: Object
    }
    idToken: {idToken: 'eyJraWQiOiJFeW1GaFZESVk4TWF0cVkJ4TmthWTREQnJ1ZFdYQWM5dFV1UXpF..badaf
      isAuthenticated: true
      refreshToken: undefined
      [[Prototype]]: Object
    }
  
```

Click on “Authorization”, choose “Bearer Token”, then paste the accessToken string to the token box. Keep the body part the same. Click on “Send” button again.

The screenshot shows the Postman application interface. At the top, there's a search bar labeled "Search Postman" and various navigation and settings icons. Below the header, a list of requests is visible, with one POST request highlighted. The main area shows a POST request to "http://localhost:8080/api/messages/secure/add/message". The "Authorization" tab is selected and highlighted with a red box. Under the "Authorization" tab, the "Type" dropdown is set to "Bearer Token" and is also highlighted with a red box. To the right of the dropdown is a "Token" input field containing a long string of characters: "eyJRaWQiOjFeW1GaFZESVk4TWFOcVkJTmt hWTREQnJlZFdYQWM5dFVIUXpRXzhjUHFzli wiYWxnljoiUlMyNTYifQ.eyJ2ZXliOjEslmp0aSI 6IkFULmZoYW4tMmNPVHd1U2Jkc0MyVG12 ZWFEMVpEdEJ1aDBPX243b3c3dkdEZmcilC Jpc3MiOiJodHRwczovL2Rldi02NTc1NjM0My 5va3RhLmNvbS9vYXV0aDlvZGVmYXVsdcIsI mF1ZCI6ImFwaTovL2RlZmF1bHQiLCJpYXQiO jE2NzYzODY2NjUsImV4cCl6MTY3NjM5MDI2 NSwiY2lkjoiMG9hOGF2aG8xMVE2S3IO TVo1 ZDciLCJ1aWQiOjlwMHU4YXY0YXRiYWhFTU NISjVknYlsInNjcCl6WyJwcm9maWxliwiZW1h". A large red box surrounds this entire section. A blue arrow points from the text "Click on ‘Authorization’, choose ‘Bearer Token’, then paste the accessToken string to the token box." down to the "Token" input field.

This time, we can see a status of “200 ok”.

The screenshot shows the Postman application interface. At the top, there is a header bar with a dark grey background. Below it, the main interface has a light grey header with the method "POST" and the URL "http://localhost:8080/api/messages/secure/add/message". To the right of the URL is a blue "Send" button with a dropdown arrow. The "Body" tab is selected, indicated by an orange underline. Below the tabs, there are several radio buttons for selecting the data type: "none", "form-data", "x-www-form-urlencoded", "raw" (which is selected and highlighted in orange), "binary", "GraphQL", and "JSON". The "JSON" tab has a dropdown arrow. To the right of the tabs, there are buttons for "Cookies" and "Beautify". The main body area contains a code editor with the following JSON payload:

```
1 {  
2   "title": "Example title for message",  
3   "question": "what's your question?"  
4 }  
5
```

The JSON object is highlighted with a red box. At the bottom of the interface, there are tabs for "Body", "Cookies (1)", "Headers (14)", and "Test Results". The "Body" tab is selected, indicated by an orange underline. On the right side, there is a summary section with a globe icon, the status "Status: 200 OK", the time "Time: 81 ms", and the size "Size: 469 B". There is also a "Save Response" button with a dropdown arrow. Below the summary, there are buttons for "Pretty", "Raw", "Preview", "Visualize", and "Text" with a dropdown arrow. To the right of these buttons are icons for copy, search, and refresh. The number "1" is located at the bottom left of the main body area.

Let's check in our database. Right click on the table "messages", choose "select row", then it will show the whole data of the table.

We can see the message that we have input in the postman has added into this table successfully.

The screenshot shows the SQL Server Management Studio interface. The left pane displays the database schema with the 'reactlibrarydatabase' database selected, specifically the 'messages' table. The right pane shows the results of a query:

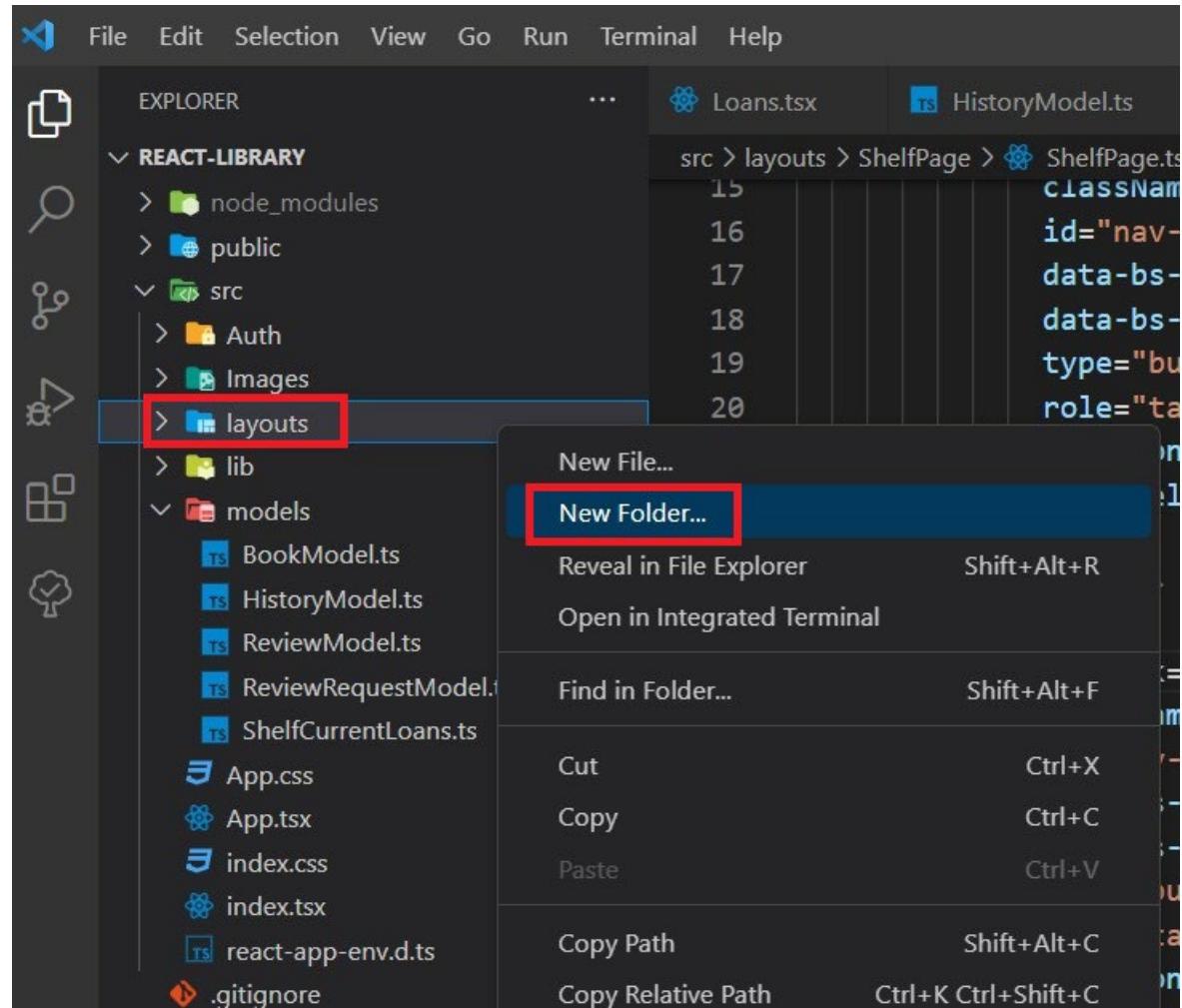
```
1 •   SELECT * FROM reactlibrarydatabase.messages;
```

	id	user_email	title	question	admin_email	response	closed
*	1	testuser2@email.com	Example title for message	what's your question?	NULL	NULL	0
*	NULL	NULL	NULL	NULL	NULL	NULL	NULL

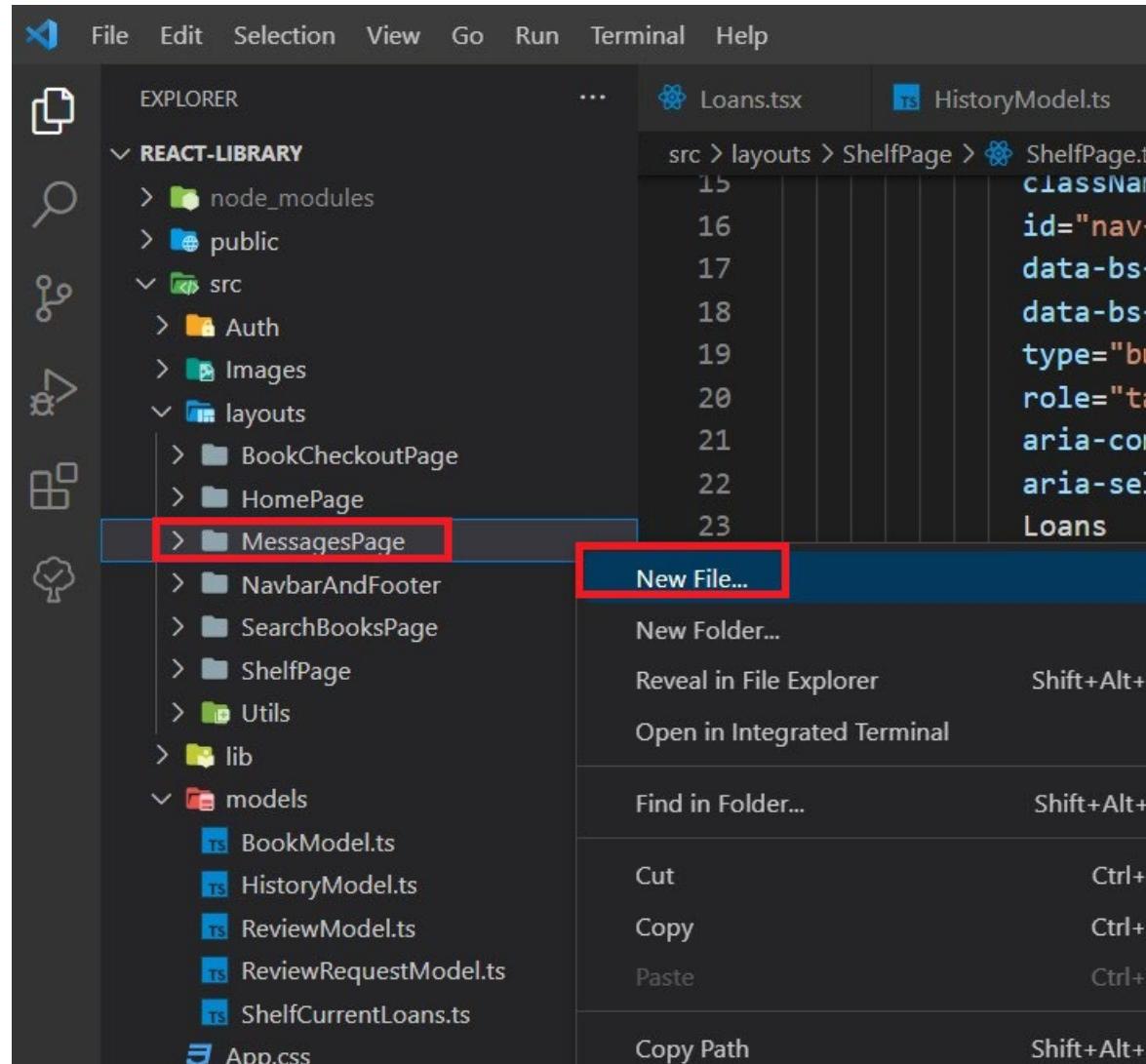
The first row of the result grid is highlighted with a red box, indicating the selected row.

6. REACT MESSAGE COMPONENT





Step 1 , Right click layouts and choose “new folder” , name it “MessagesPage”.



Step 2, In this folder, create a new file, MessagesPage.tsx.

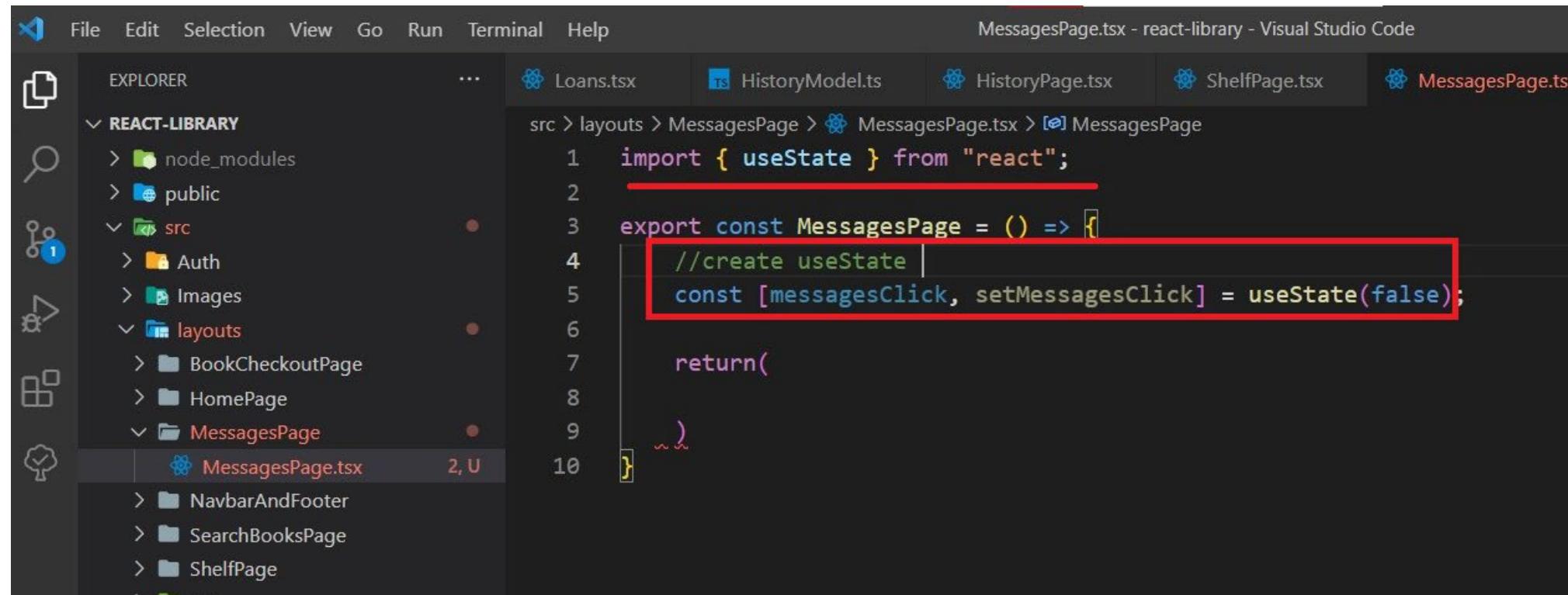
Step 3 create export const MessagesPage () structure.

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

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** MessagesPage.tsx - react-library - Visual Studio Code
- Explorer:** Shows the project structure under REACT-LIBRARY:
 - node_modules
 - public
 - src
 - Auth
 - Images
 - layouts
 - BookCheckoutPage
 - HomePage
 - MessagesPage
 - MessagesPage.tsx (highlighted with a red border)
 - NavbarAndFooter
 - SearchBooksPage
 - ShelfPage
 - Utils
- Editor:** The file MessagesPage.tsx is open, showing the following code:

```
src > layouts > MessagesPage > MessagesPage.tsx > MessagesPage
1  export const MessagesPage = () => {
2    return(
3      ...
4    )
5 }
```

Step 4 , create a useState array messageClick, setMesssageClick. If you see a red tilde under the useState, import it from “React”.



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

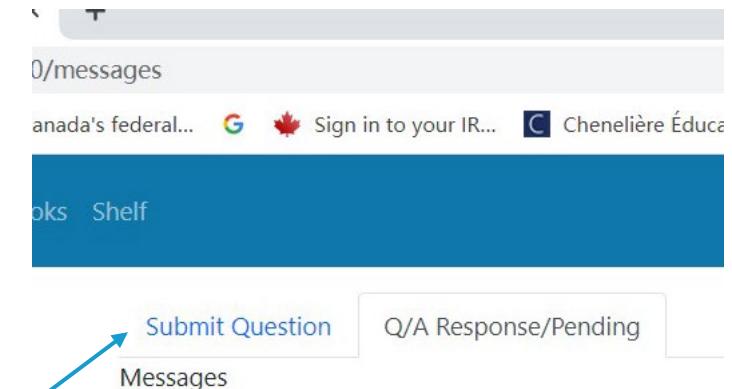
- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** MessagesPage.tsx - react-library - Visual Studio Code
- Explorer:** Shows the project structure under REACT-LIBRARY:
 - node_modules
 - public
 - src
 - Auth
 - Images
 - layouts
 - BookCheckoutPage
 - HomePage
 - MessagesPage
 - MessagesPage.tsx (selected)
 - NavbarAndFooter
 - SearchBooksPage
 - ShelfPage
- Code Editor:** The file MessagesPage.tsx is open. The code is as follows:

```
1 import { useState } from "react";
2
3 export const MessagesPage = () => {
4     //create useState
5     const [messagesClick, setMessagesClick] = useState(false);
6
7     return(
8         ...
9     )
10}
```

The line `const [messagesClick, setMessagesClick] = useState(false);` is highlighted with a red box.

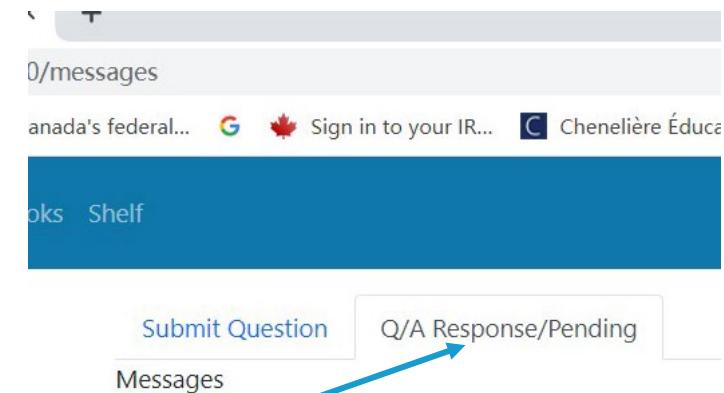
Step 5, fill in return part with Html and CSS code.
First , create a submit question tab.

```
// html and css part
return (
  <div className="container">
    <div className="mt-3 mb-2">
      <nav>
        <div className="nav nav-tabs" id="nav-tab" role="tablist">
          {/* submit btn */}
          <button
            onClick={() => setMessagesClick(false)}
            className="nav-link active"
            id="nav-send-message-tab"
            data-bs-toggle="tab"
            data-bs-target="#nav-send-message"
            type="button"
            role="tab"
            aria-controls="nav-send-message"
            aria-selected="true">
            Submit Question
          </button>
```



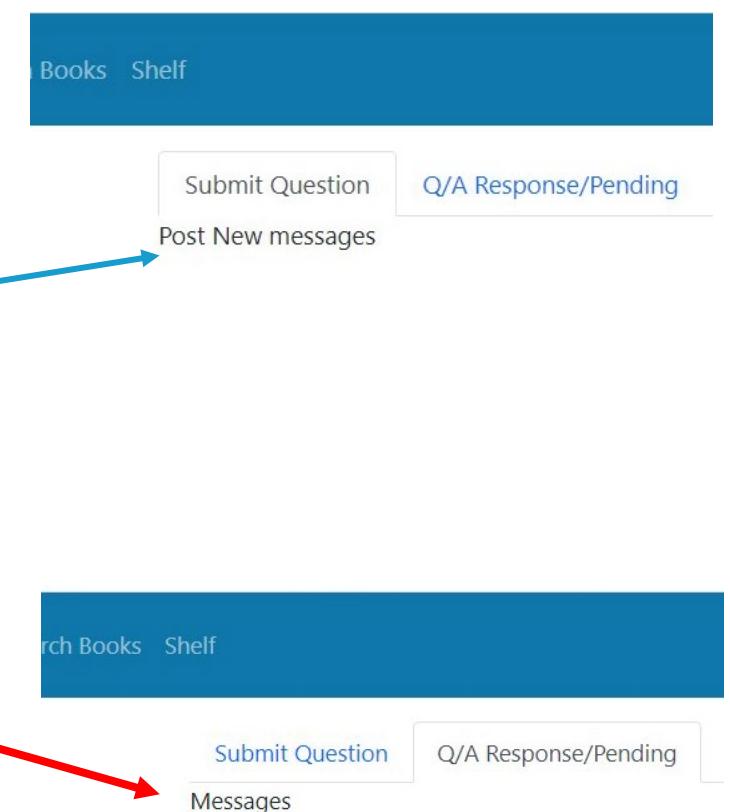
Second, response tab.

```
        aria-selected="true">
    Submit Question
</button>
/* response btn */
<button
    onClick={() => setMessagesClick(true)}
    className="nav-link"
    id="nav-message-tab"
    data-bs-toggle="tab"
    data-bs-target="#nav-message"
    type="button"
    role="tab"
    aria-controls="nav-message"
    aria-selected="false">
    Q/A Response/Pending
</button>
</div>
```



Then the words under the tab

```
        Q/A Response/Pending
    </button>
</div>
</nav>
<div className="tab-content" id="nav-tabContent">
    <div
        className="tab-pane fade show active"
        id="nav-send-message"
        role="tabpanel"
        aria-labelledby="nav-send-message-tab">
        <p>Post New messages</p>
    </div>
    <div className="tab-pane fade" id="nav-message" role="tabpanel" aria-labelledby="nav-message-tab">
        /* if it's clicked, show p tag or show nothing */
        {messagesClick ? <p>Messages</p>:<></>}
    </div>
</div>
```



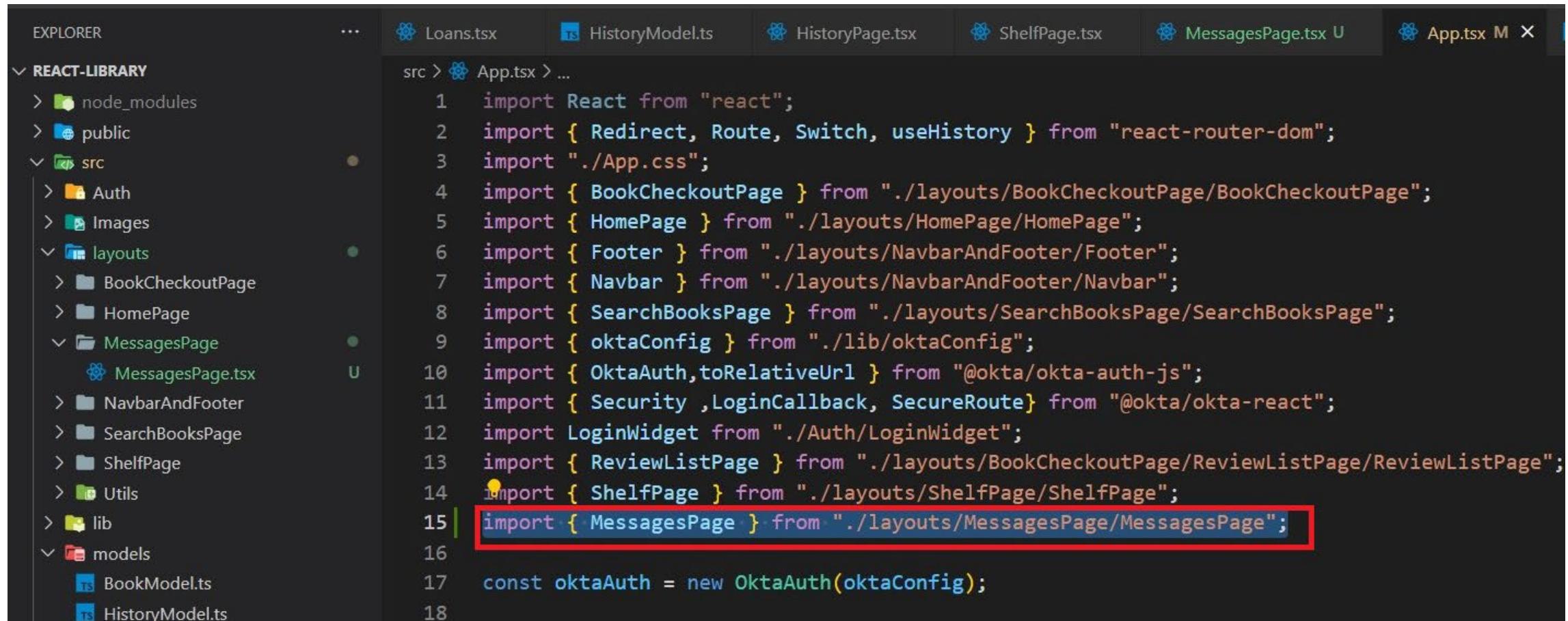
Step 6, Go to App.tsx, add Secure Route for “MessagesPage”, and import MessagesPage.

The screenshot shows a code editor with a sidebar on the left displaying a file tree. The tree includes: SearchBooksPage, ShelfPage, Utils, lib, models (with files BookModel.ts, HistoryModel.ts, ReviewModel.ts, ReviewRequestModel.ts, ShelfCurrentLoans.ts), App.css, App.tsx (which is highlighted with a red box), index.css, and index.tsx. The main pane shows the code for App.tsx:

```
43     </Route>
44     <Route path='/reviewlist/:bookId'>
45       <ReviewListPage/>
46     </Route>
47     <Route path="/checkout/:bookId">
48       <BookCheckoutPage></BookCheckoutPage>
49     </Route>
50     <Route path='/login' render={
51       () => <LoginWidget config={oktaConfig} />
52     }
53   >
54     <Route path='/login/callback' component={LoginCallback} />
55     <SecureRoute path='/shelf'> <ShelfPage/> </SecureRoute>
56     <SecureRoute path='/messages'> <MessagesPage/> </SecureRoute>
57   </Switch>
```

The line `<SecureRoute path='/messages'> <MessagesPage/> </SecureRoute>` is highlighted with a red box.

Check the top of the page, if the MessagePage is imported.



The screenshot shows the VS Code interface with the following details:

- EXPLORER** sidebar on the left showing the project structure under **REACT-LIBRARY**, including **src**, **node_modules**, **public**, **Auth**, **Images**, **layouts** (with **BookCheckoutPage**, **HomePage**, **MessagesPage**), **NavbarAndFooter**, **SearchBooksPage**, **ShelfPage**, **Utils**, **lib**, and **models** (with **BookModel.ts** and **HistoryModel.ts**).
- App.tsx** file is open in the main editor area.
- The code in **App.tsx** is as follows:

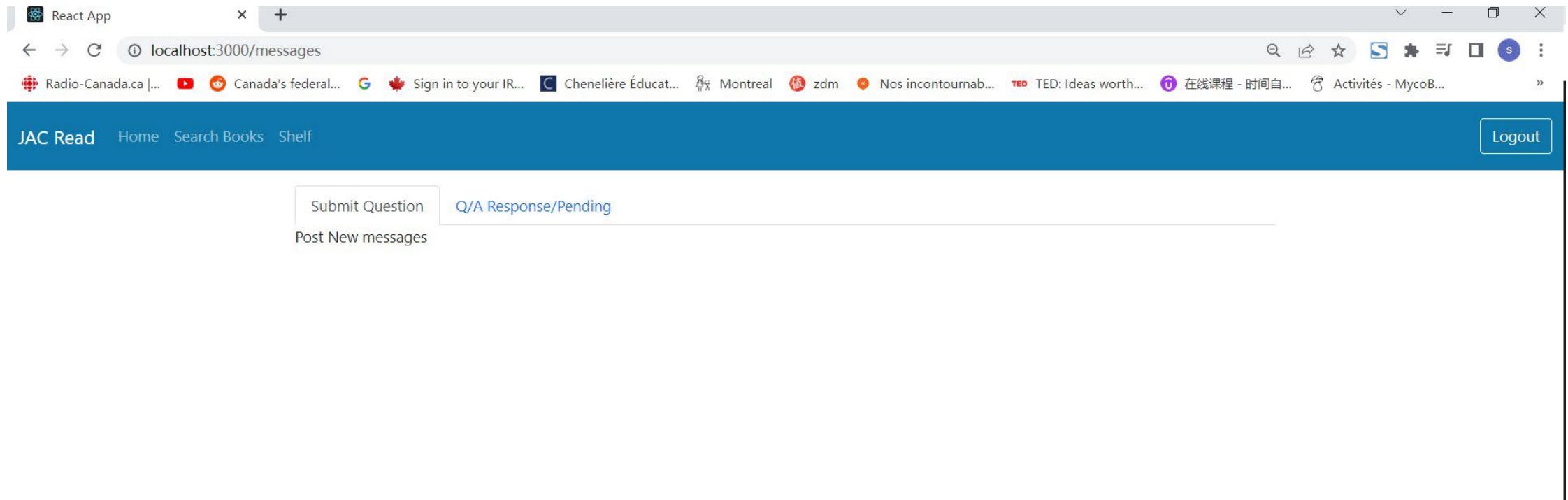
```
src > App.tsx > ...
1  import React from "react";
2  import { Redirect, Route, Switch, useHistory } from "react-router-dom";
3  import "./App.css";
4  import { BookCheckoutPage } from "./layouts/BookCheckoutPage/BookCheckoutPage";
5  import { HomePage } from "./layouts/HomePage/HomePage";
6  import { Footer } from "./layouts/NavbarAndFooter/Footer";
7  import { Navbar } from "./layouts/NavbarAndFooter/Navbar";
8  import { SearchBooksPage } from "./layouts/SearchBooksPage/SearchBooksPage";
9  import { oktaConfig } from "./lib/oktaConfig";
10 import { OktaAuth,toRelativeUrl } from "@okta/okta-auth-js";
11 import { Security ,LoginCallback, SecureRoute} from "@okta/okta-react";
12 import LoginWidget from "./Auth/LoginWidget";
13 import { ReviewListPage } from "./layouts/BookCheckoutPage/ReviewListPage/ReviewListPage";
14 import { ShelfPage } from "./layouts/ShelfPage/ShelfPage";
15 import { MessagesPage } from "./layouts/MessagesPage/MessagesPage";
16
17 const oktaAuth = new OktaAuth(oktaConfig);
```

The line `import { MessagesPage } from "./layouts/MessagesPage/MessagesPage";` is highlighted with a red box.

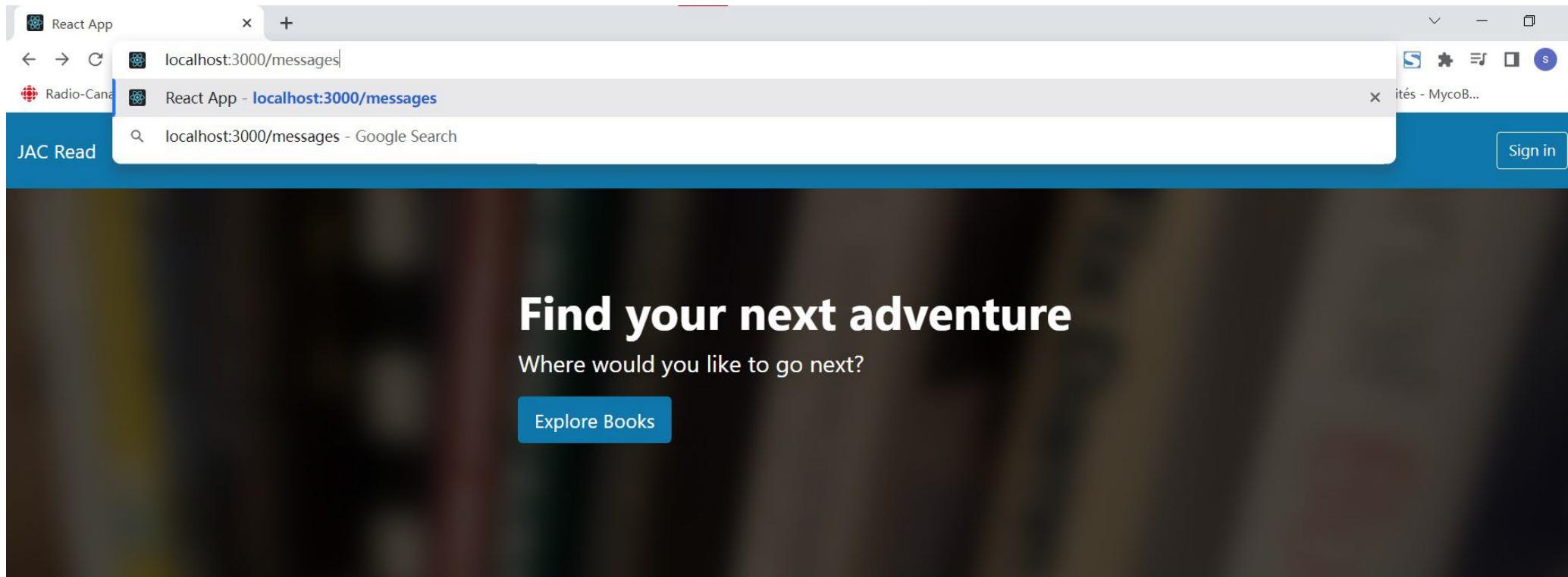
Now, log in the account with the email and password.
if successfully ,the sign in button will change into logout.
then visit <http://localhost:3000/messages>, click on “Q/A Response/Pending” , it will show:

A screenshot of a web browser window titled "React App". The address bar shows "localhost:3000/messages". The page content includes a navigation bar with "JAC Read", "Home", "Search Books", "Shelf", and "Logout". Below the navigation bar are three buttons: "Submit Question" (disabled), "Q/A Response/Pending" (highlighted with a red border), and "Messages". The browser's toolbar at the top includes standard icons for back, forward, search, and refresh.

Click on “Submit Question” , it will show:



Then we log out , type <http://localhost:3000/messages>



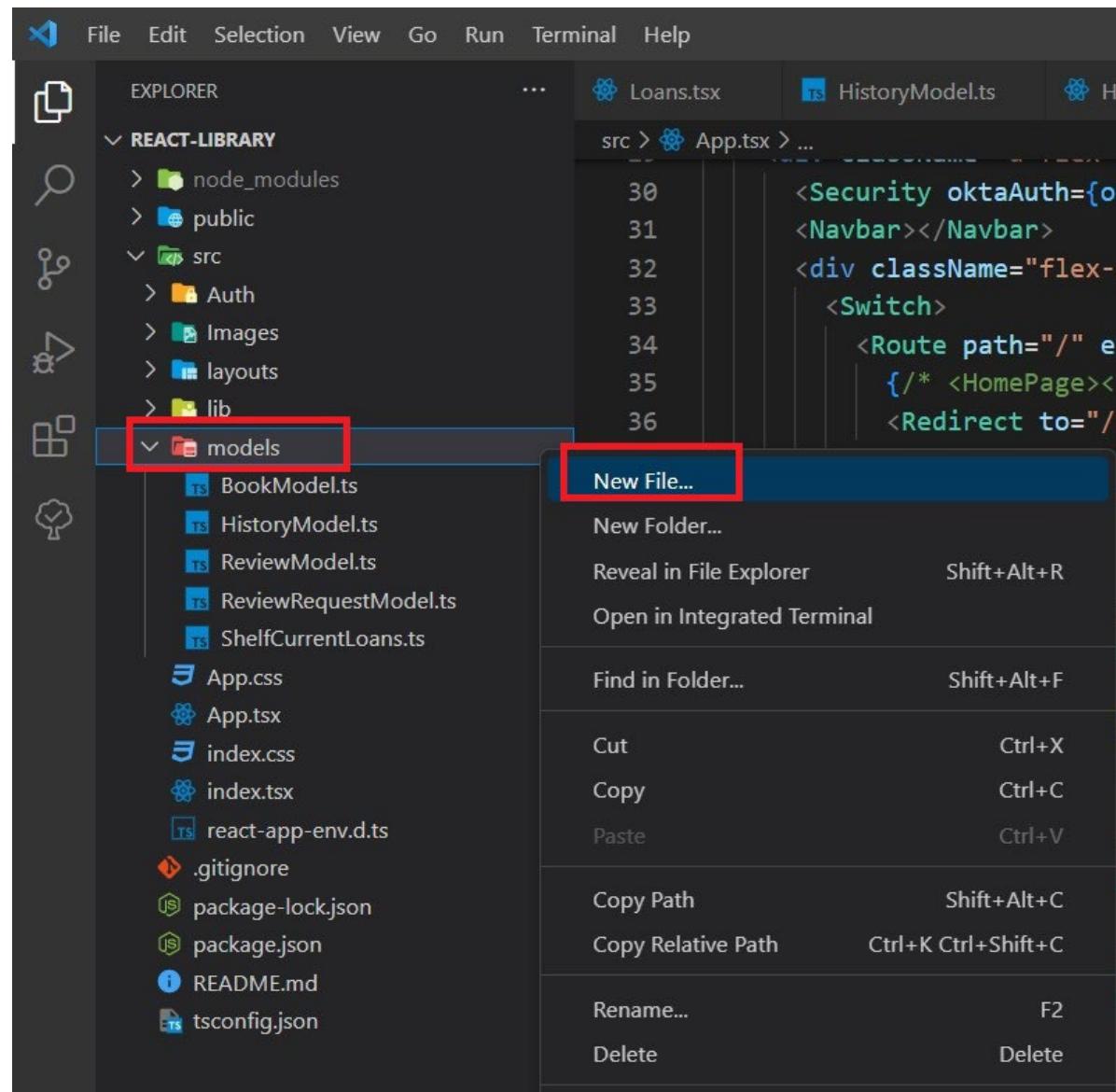
It will redirect to sign in page, so only authenticated user can use message page, which means our security configuration works.

A screenshot of a web browser window titled "React App". The address bar shows "localhost:3000/login". The page content is a login form for "JAC Read". The form has fields for "Username" (containing "testuser2@email.com") and "Password". A "Remember me" checkbox is checked. A "Sign In" button is at the bottom. The browser toolbar includes various icons for navigation and search.

The login form is contained within a white rectangular box. At the top center is the text "Sign In". Below it is a "Username" field containing "testuser2@email.com". Below that is a "Password" field, which is currently empty and highlighted with a light blue border. To the left of the password field is a "Remember me" checkbox, which is checked and has a small checkmark icon. At the bottom of the form is a large, solid blue "Sign In" button. Below the button, there is a small, faint link that says "Need help signing in?".

7.REACT MESSAGE MODEL





Step 1, right click models and choose new file, name it MessageModel.ts.

Step 2 , In MessageModel.ts, create a class with the properties that is mapping with Message entity in our springboot.

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

- File Explorer (Left):** Shows the project structure under "REACT-LIBRARY". The "models" folder contains files: BookModel.ts, HistoryModel.ts, ReviewModel.ts, ReviewRequestModel.ts, ShelfCurrentLoans.ts, App.css, App.tsx, index.css, index.tsx, react-app-env.d.ts, .gitignore, and package-lock.json. The "MessageModel.ts" file is highlighted with a red rectangle.
- Code Editor (Right):** The current file is "MessageModel.ts". The code defines a class "MessageModel" with properties: title (string), question (string), id? (number), userEmail? (string), adminEmail? (string), response? (string), and closed? (boolean). It includes a constructor to initialize these properties and an export statement at the bottom.
- Top Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Tab Bar:** Loans.tsx, HistoryModel.ts, HistoryPage.tsx, ShelfPage.tsx, MessagesPage.tsx, App.tsx.
- Title Bar:** MessageModel.ts - react-library - Visual Studio Code.

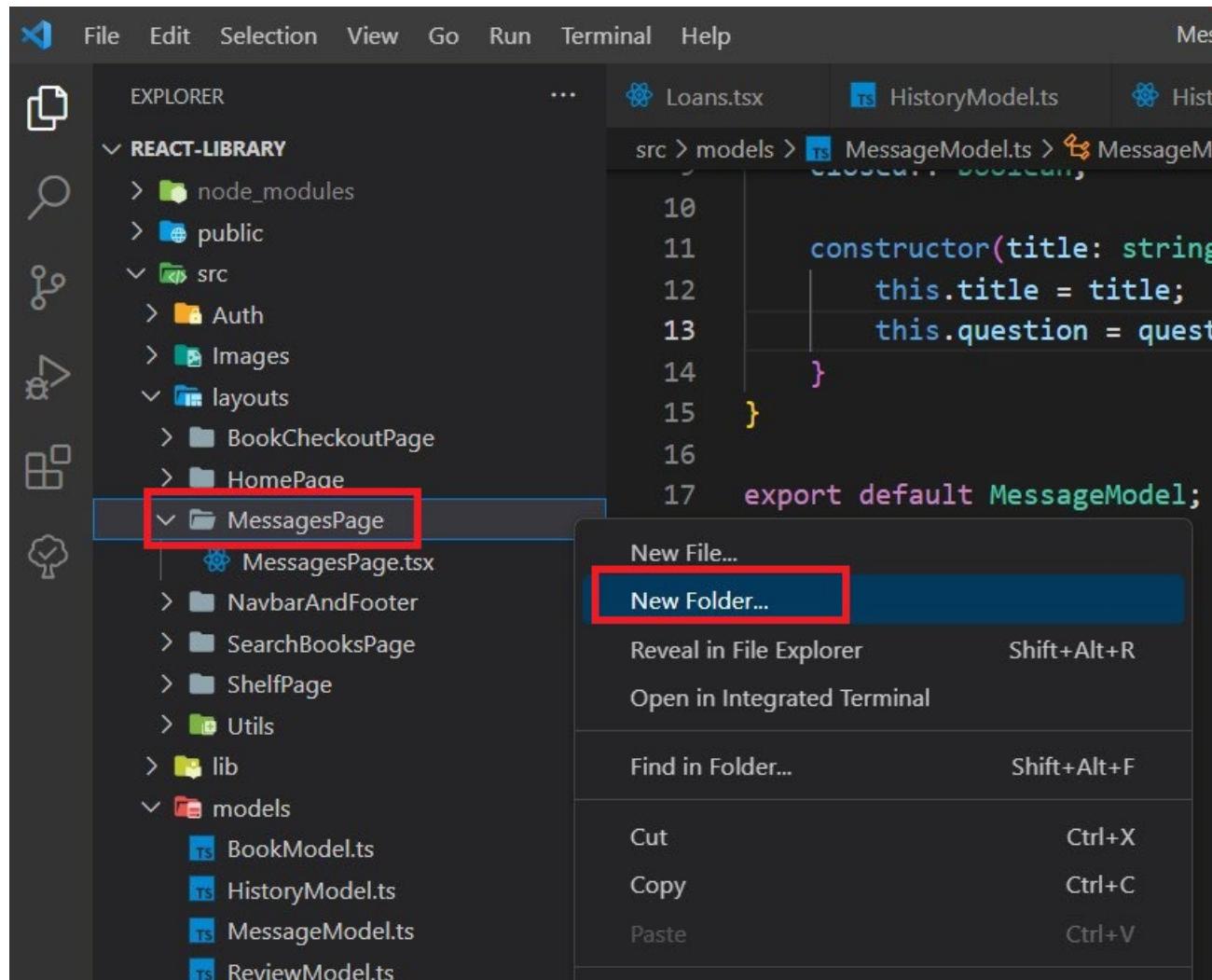
```
// mapping with Message entity in our springboot
class MessageModel {
    title: string;
    question: string;
    id?: number;
    userEmail?: string;
    adminEmail?: string;
    response?: string;
    closed?: boolean;

    constructor(title: string, question: string) {
        this.title = title;
        this.question = question;
    }
}

export default MessageModel;
```

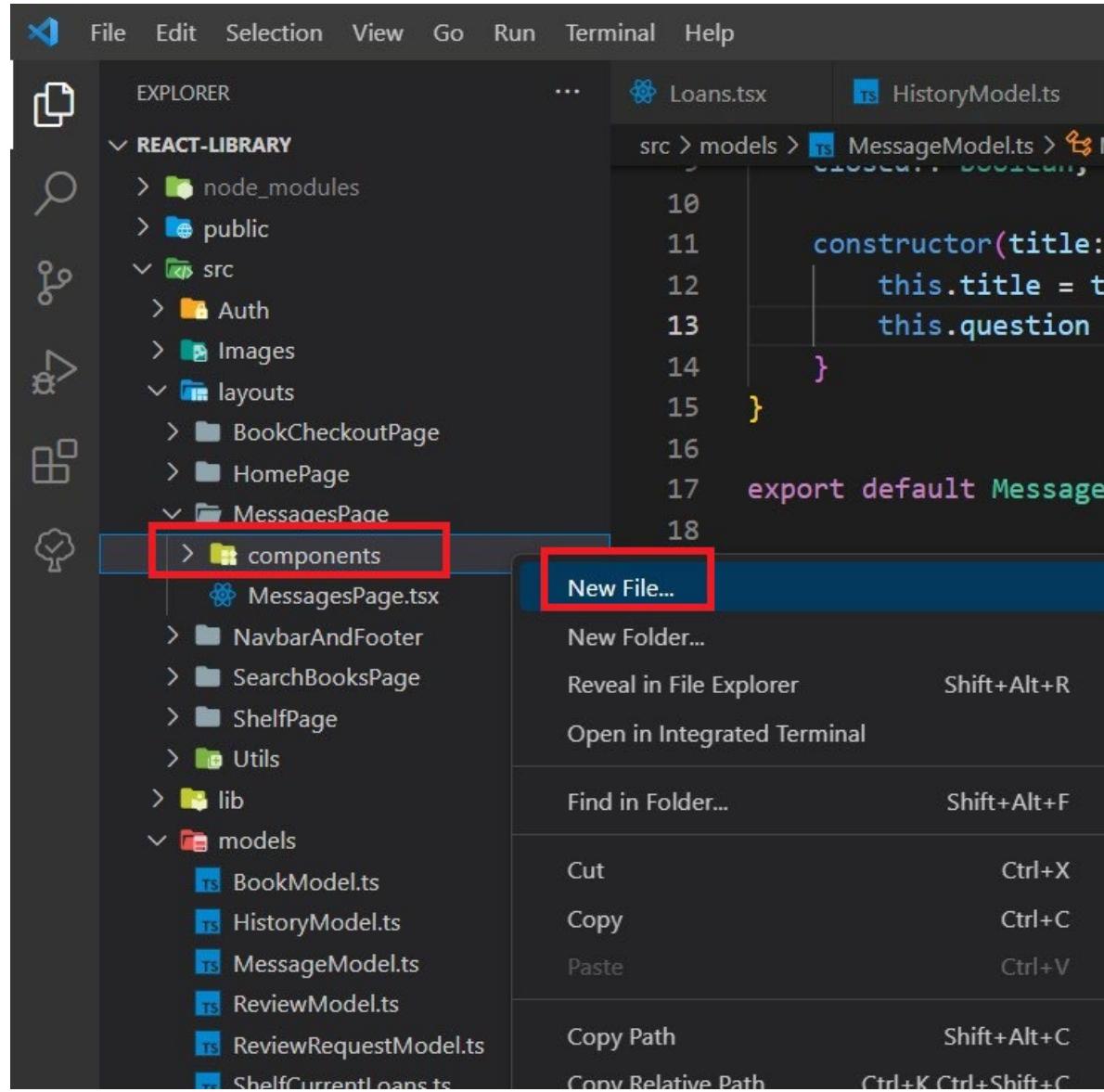
8. REACT POST A QUESTION COMPONENT



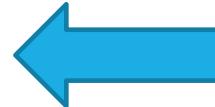
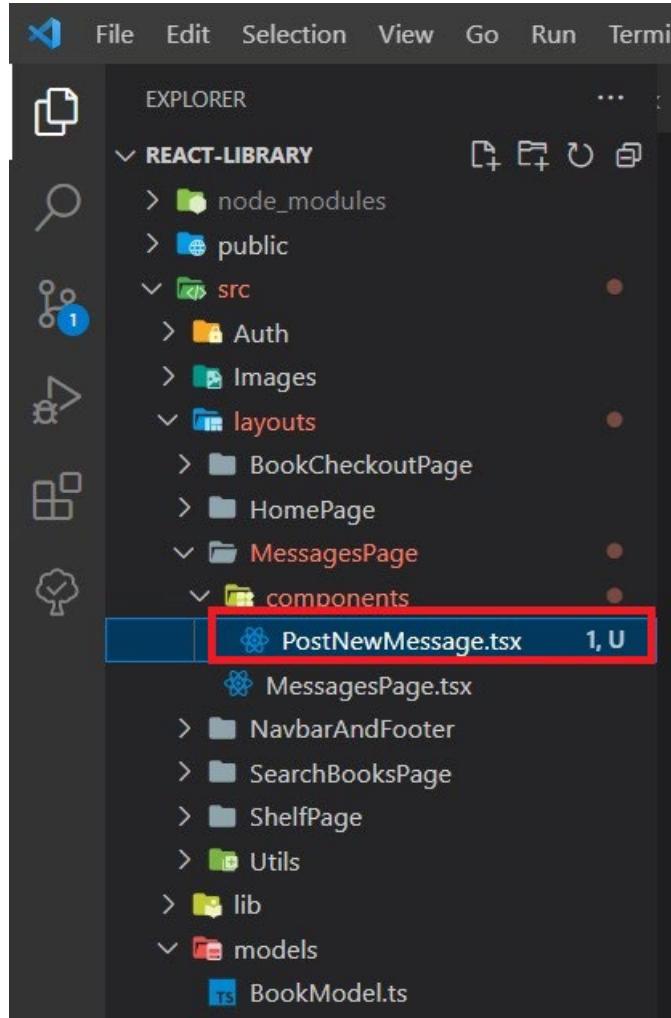


Here, we will create new form where we can post new messages to our admins into our database.

Step1, Go into “layouts” folder, right click “MessagesPage” folder, to create a new folder called “components”.



Step 2, Inside components ,
create a new file, name it PostNewMessage.tsx.



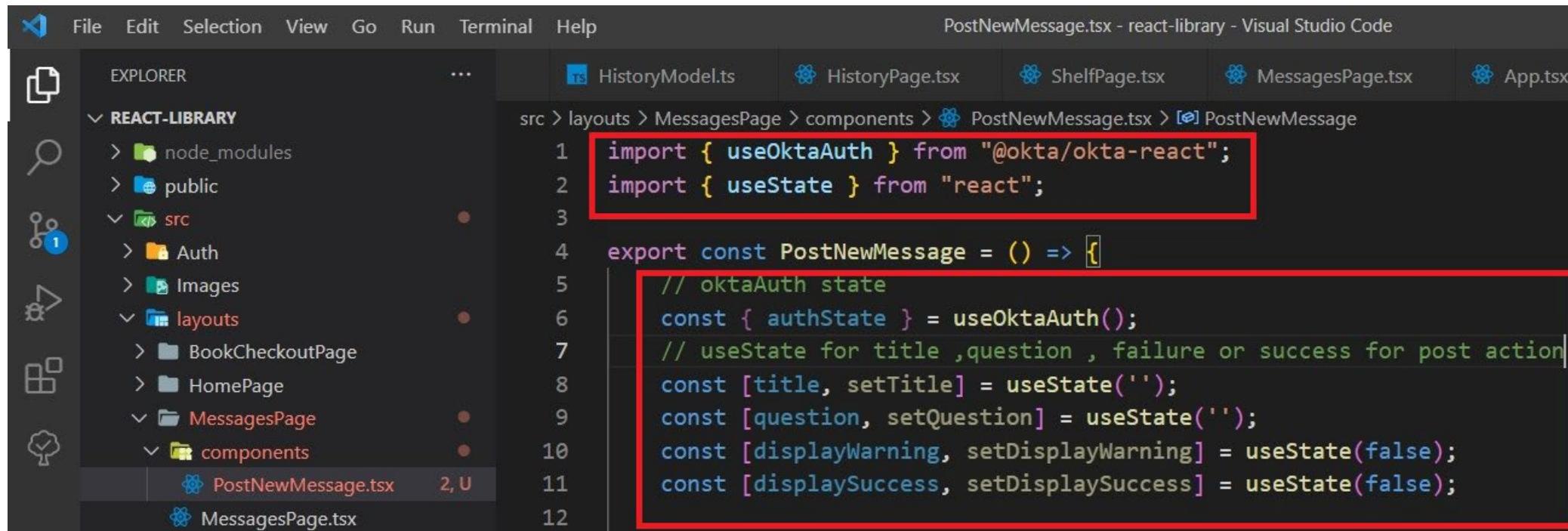
New file created.

Step 3, as usual, create export const PostNewMessage structure.

The screenshot shows the Visual Studio Code interface with a dark theme. The menu bar includes File, Edit, Selection, View, Go, Run, Terminal, and Help. The title bar indicates the current file is PostNewMessage.tsx - react-library - Visual Studio Code. The Explorer sidebar on the left shows the project structure under 'REACT-LIBRARY': node_modules, public, src (with subfolders Auth, Images, layouts), layouts (with subfolders BookCheckoutPage, HomePage, MessagesPage, components), and two files: PostNewMessage.tsx and MessagesPage.tsx. The status bar at the bottom shows 'PostNewMessage.tsx' and '2, U'. The main code editor area displays the following TypeScript code:

```
src > layouts > MessagesPage > components > PostNewMessage.tsx > PostNewMessage
1  export const PostNewMessage = () => {
2    return(
3      ...
4    )
5 }
```

Step 4, create useState and import okta and react useState.



File Edit Selection View Go Run Terminal Help

PostNewMessage.tsx - react-library - Visual Studio Code

EXPLORER

REACT-LIBRARY

- > node_modules
- > public
- > src
 - > Auth
 - > Images
 - > layouts
 - > BookCheckoutPage
 - > HomePage
 - > MessagesPage
 - > components
 - PostNewMessage.tsx

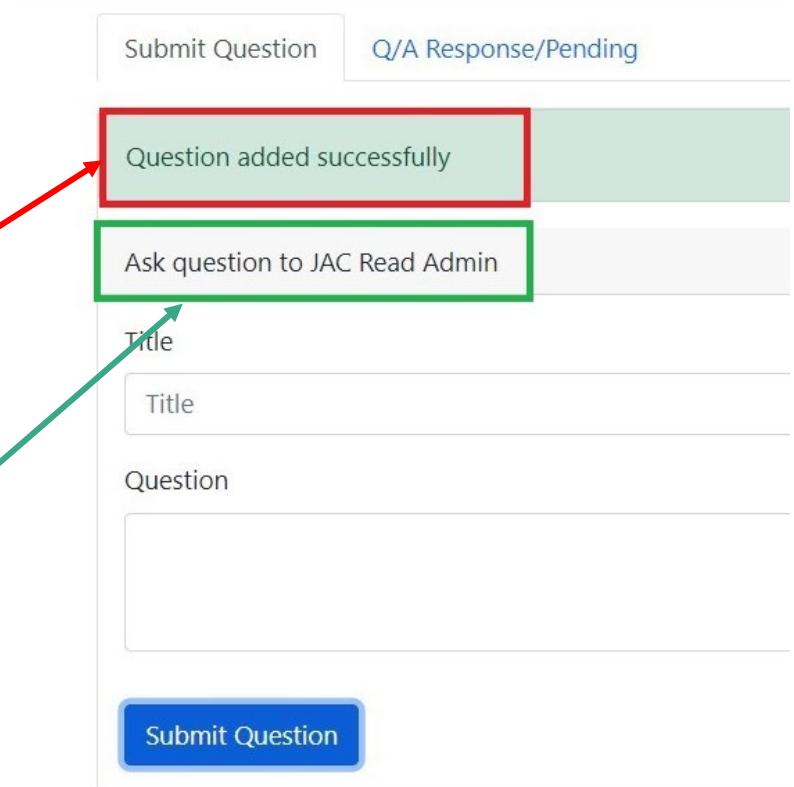
```
src > layouts > MessagesPage > components > PostNewMessage.tsx > PostNewMessage
1 import { useOktaAuth } from "@okta(okta-react";
2 import { useState } from "react";
3
4 export const PostNewMessage = () => [
5   // oktaAuth state
6   const { authState } = useOktaAuth();
7   // useState for title ,question , failure or success for post action
8   const [title, setTitle] = useState('');
9   const [question, setQuestion] = useState('');
10  const [displayWarning, setDisplayWarning] = useState(false);
11  const [displaySuccess, setDisplaySuccess] = useState(false);
```

Step 5, create Html/css in return part.

First, a success message banner and a card header.

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

return (
  <div className="card mt-3">
    /* alert message */
    {displaySuccess && (
      <div className="alert alert-success" role="alert">
        | Question added successfully
      </div>
    )}
    /* card header */
    <div className="card-header">Ask question to JAC Read Admin</div>
  </div>
)
```



Then , In card body, an alert message, a label and an input box.

PostNewMessage.tsx - react-library - Visual Studio Code

MessagesPage.tsx App.tsx MessageModel.ts PostNewMessage.tsx 3, U

MessagesPage > components > PostNewMessage.tsx > PostNewMessage

```
/* card header */


Ask question to JAC Read Admin


/* card body */


/* form */
  <form method="POST">
    /* warning text:if user do not input the fields */
    {displayWarning && (
      <div className="alert alert-danger" role="alert">
        All fields must be filled out
      </div>
    )}
    /* title label and input area */
    <div className="mb-3">
      <label className="form-label">Title</label>
      <input
        type="text"
        className="form-control"
        id="exampleFormControlInput1"
        placeholder="Title"
        onChange={(e) => setTitle(e.target.value)}
        value={title}
      />
    </div>
  </form>


```

Submit Question Q/A Response/Pending

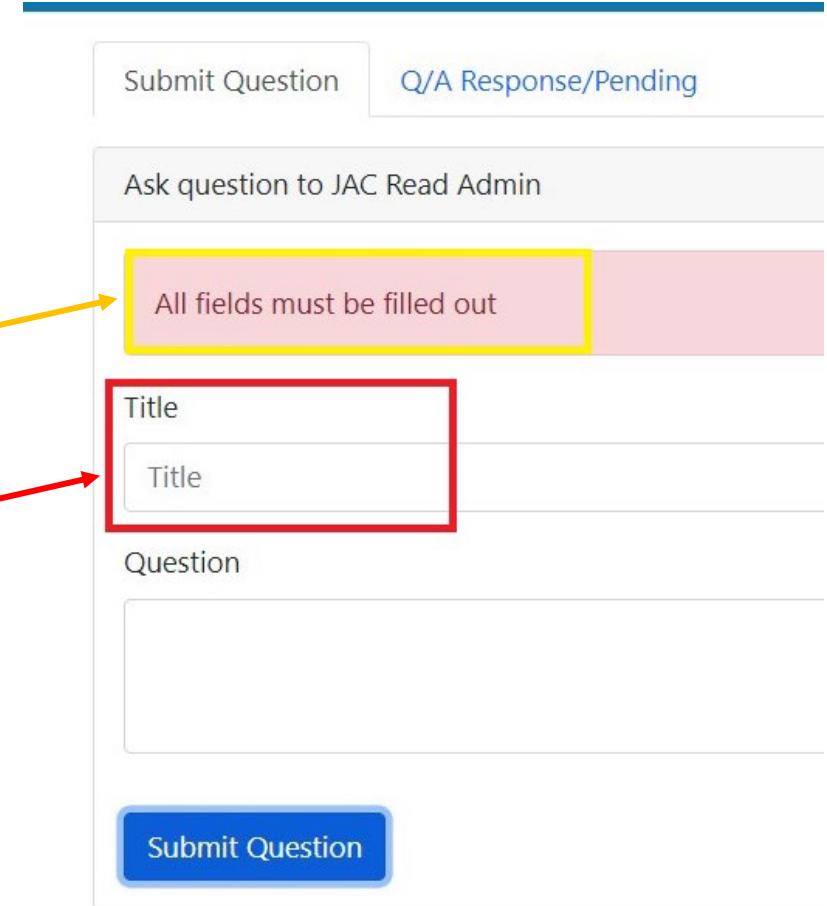
Ask question to JAC Read Admin

All fields must be filled out

Title

Question

Submit Question



Last, question label and text area, a submit button.

```
<div className="mb-3"> ...
</div>
/* question label and textarea */
<div className='mb-3'>
  <label className='form-label'>
    Question
  </label>
  <textarea className='form-control' id='exampleFormControlTextarea1'
    rows={3} onChange={e => setQuestion(e.target.value)} value={question}>
  </textarea>
</div>
/* submit btn*/
<div>
  <button type='button' className='btn btn-primary mt-3'>
    Submit Question
  </button>
</div>
```

Submit Question Q/A Response/Pending

Ask question to JAC Read Admin

All fields must be filled out

Title

Question

Submit Question

Step 6, Go to MessagesPage.tsx, replace <P>Messages</p> with PostNewMessage component and import it.

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

- File Explorer (Left):** Shows the project structure under "REACT-LIBRARY". The "MessagesPage" folder is expanded, showing "PostNewMessage.tsx" and "MessagesPage.tsx". The "MessagesPage.tsx" file is currently selected and highlighted with a red border.
- Code Editor (Right):** Displays the content of the "MessagesPage.tsx" file. The code is a tabbed navigation component. A specific line of code is highlighted with a blue selection bar:

```
    <p>Post New messages</p>
```
- Status Bar:** Shows the file name "MessagesPage.tsx - react-library - Visual Studio Code".

After the change, the code should be as below:

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", "layouts", and "components". Inside "src/layouts/MessagesPage/components", there are files: "PostNewMessage.tsx" (highlighted in green) and "MessagesPage.tsx" (highlighted in red).
- Code Editor (Right):** The file "MessagesPage.tsx" is open. The code is as follows:

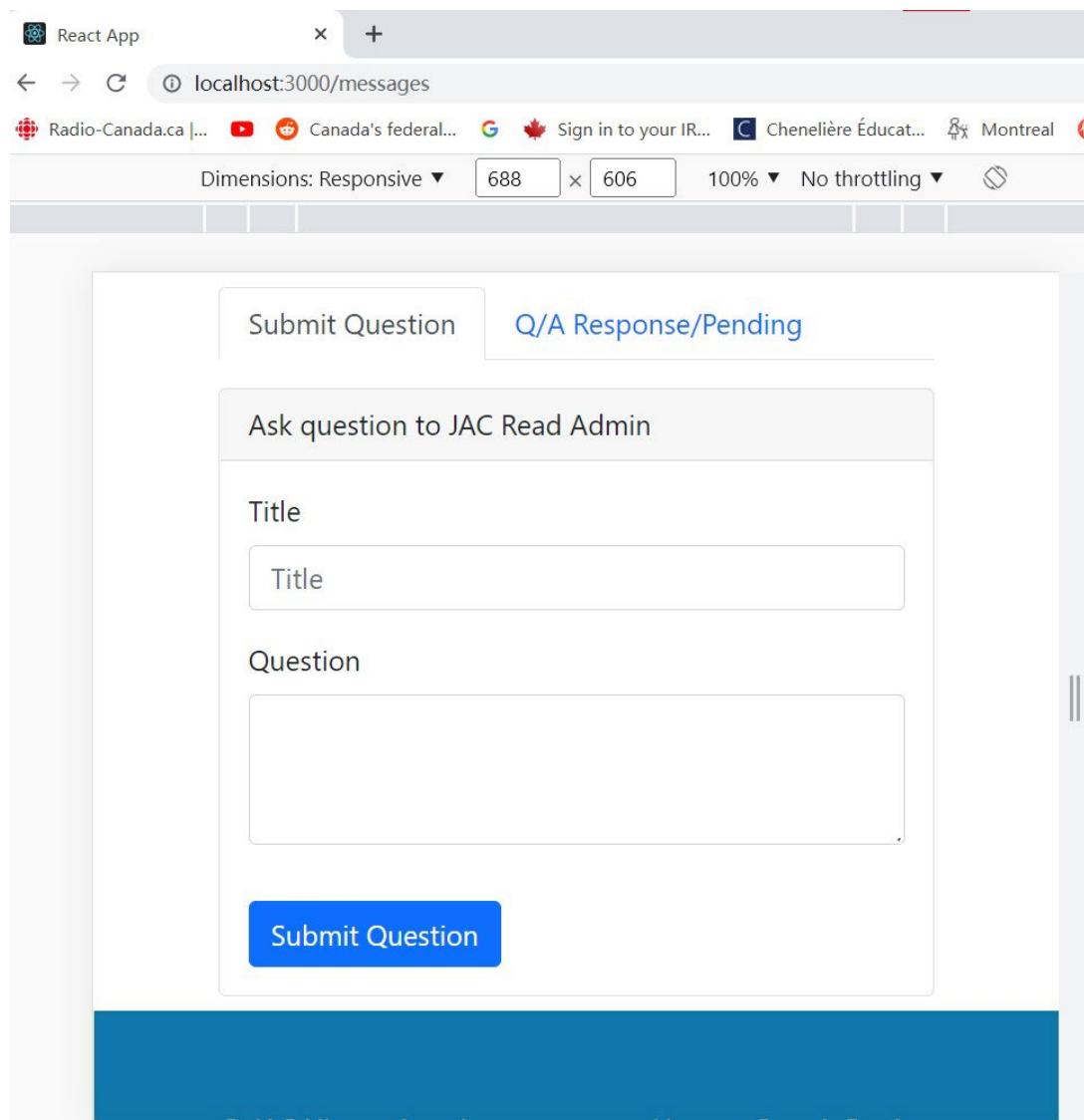
```
import {useState} from 'react'
import { PostNewMessage } from './components/PostNewMessage'

export const MessagesPage = () => {
    //create useState
    const [messagesClick, setMessagesClick] = useState(false)

    // html and css part
    return (
        <div className="container">
            <div className="mt-3 mb-2">
                <nav>...
                </nav>
                <div className="tab-content" id="nav-tabContent">
                    <div
                        className="tab-pane fade show active"
                        id="nav-send-message"
                        role="tabpanel"
                        aria-labelledby="nav-send-message-tab">
                        <PostNewMessage/>
                    </div>
                </div>
            </div>
        </div>
    )
}
```

Sign in the website , and go to <http://localhost:3000/messages>, it will show the page like this:

The screenshot shows a web browser window with the address bar containing 'localhost:3000/messages'. The page itself is titled 'Ask question to JAC Read Admin' and features a form for submitting a question. The form includes fields for 'Title' and 'Question', both represented by large text input areas. A blue 'Submit Question' button is located at the bottom left of the form. The browser's header bar shows various tabs and icons, and the overall layout is clean and modern.



Right click and choose “Inspect”, choose mobile version, the page still works!

9. REACT POST A QUESTION FUNCTIONALITY

Create a function that's going to be able to communicate with our spring boot back into endpoint so we can add and submit questions and messages.

Step1, In PostNewMessage.tsx file, jumping down from these pieces of state and creating a new async function submitNewQuestion().

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

- File Explorer (Left):** Shows the project structure under "REACT-LIBRARY". The "PostNewMessage.tsx" file is selected and highlighted with a red border.
- Code Editor (Right):** Displays the content of "PostNewMessage.tsx". A specific line of code, "async function submitNewQuestion()", is highlighted with a red underline.
- Top Bar:** Shows the menu bar (File, Edit, Selection, View, Go, Run, Terminal, Help) and the title "PostNewMessage.tsx - react-library - Visual Studio Code".
- Status Bar:** Shows the file path "src > layouts > MessagesPage > components > PostNewMessage.tsx" and the status "1, M".

```
// fetch data from the database
async function submitNewQuestion() {
    // create a const to save our url string
    const url = `http://localhost:8080/api/messages/secure/add/message`;
    // if the user is authenticated, and input both the title and question
    if (authState?.isAuthenticated && title !== '' && question !== '') {
        //create a new messageModel with the user input and assign it to messageRequestModel
        const messageRequestModel: MessageModel = new MessageModel({
            title,
            question
        });
        const requestOptions = {
            method: 'POST',
            headers: {
                'Authorization': `Bearer ${authState?.accessToken?.access_token}`,
                'Content-Type': 'application/json',
            },
            // convert string into JSON
            body: JSON.stringify(messageRequestModel),
        }
        // fetch the data with the url and request option
        const submitNewQuestionResponse = await fetch(url, requestOptions);
        //fetching failure
        if (!submitNewQuestionResponse.ok) {
            throw new Error('Something went wrong!')
        }
    }
}
```

PostNewMessage.tsx - react-library - Visual Studio Code

components > PostNewMessage.tsx > PostNewMessage

```
/* question label and textarea */


<label className="form-label">Question</label>
  <textarea
    className="form-control"
    id="exampleFormControlTextarea1"
    rows={3}
    onChange={(e) => setQuestion(e.target.value)}
    value={question}></textarea>


/* submit btn*/


<button type="button" className="btn btn-primary mt-3" onClick={submitNewQuestion}>
    Submit Question
  </button>


</form>
```

Step 2, add a onClick listener to submit button.

Visit <http://localhost:3000/messages>, try and submit a question with nothing in our title or question input area, alert message “All fields must be filled out” will show up:

The screenshot shows a web browser window with the URL `localhost:3000/messages` in the address bar. The page has a blue header bar with the text "JAC Read" and other navigation links like "Home", "Search Books", and "Shelf". Below the header is a form titled "Ask question to JAC Read Admin". The form has two tabs: "Submit Question" (selected) and "Q/A Response/Pending". The "Title" field is empty, and the "Question" field is also empty. A red error message "All fields must be filled out" is displayed above the "Title" field. At the bottom of the form is a blue "Submit Question" button.

try and submit a question only with title input but nothing in question input area, alert message “All fields must be filled out” also will show up:

React App

localhost:3000/messages

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... Activités - N

JAC Read Home Search Books Shelf

Submit Question Q/A Response/Pending

Ask question to JAC Read Admin

All fields must be filled out

Title

test title

Question

Submit Question

This time, with question input but nothing title input area, alert message “All fields must be filled out” also will show up:

The screenshot shows a web browser window with the address bar displaying "localhost:3000/messages". The page title is "React App". The main content is a form titled "Ask question to JAC Read Admin". It has two input fields: "Title" and "Question". The "Title" field is empty and highlighted with a red border. The "Question" field contains the text "test q". Above the "Title" field, a red rectangular box displays the error message "All fields must be filled out". At the bottom of the form is a blue "Submit Question" button.

React App

localhost:3000/messages

JAC Read Home Search Books Shelf

Submit Question Q/A Response/Pending

All fields must be filled out

Title

Question

test q

Submit Question

Input both two areas and click on “submit question” button,

The screenshot shows a web browser window with the title "React App" and the URL "localhost:3000/messages". The page is titled "JAC Read" and includes navigation links for "Home", "Search Books", and "Shelf", along with a "Logout" button. The main content area is a form titled "Ask question to JAC Read Admin". It contains two input fields: "Title" with the value "test title" and "Question" with the value "test q". Below the form is a blue "Submit Question" button.

React App

localhost:3000/messages

JAC Read Home Search Books Shelf Logout

Submit Question Q/A Response/Pending

Ask question to JAC Read Admin

Title

test title

Question

test q

Submit Question

Questions added successfully and we have the banner going above.

A screenshot of a web browser window titled "React App" showing a successful question submission message. The URL is "localhost:3000/messages". The page has a blue header with "JAC Read" and navigation links. Below the header is a form for submitting a question, featuring fields for "Title" and "Question" and a "Submit Question" button. A green success message box with a red border contains the text "Question added successfully".

React App

localhost:3000/messages

JAC Read Home Search Books Shelf

Submit Question Q/A Response/Pending

Question added successfully

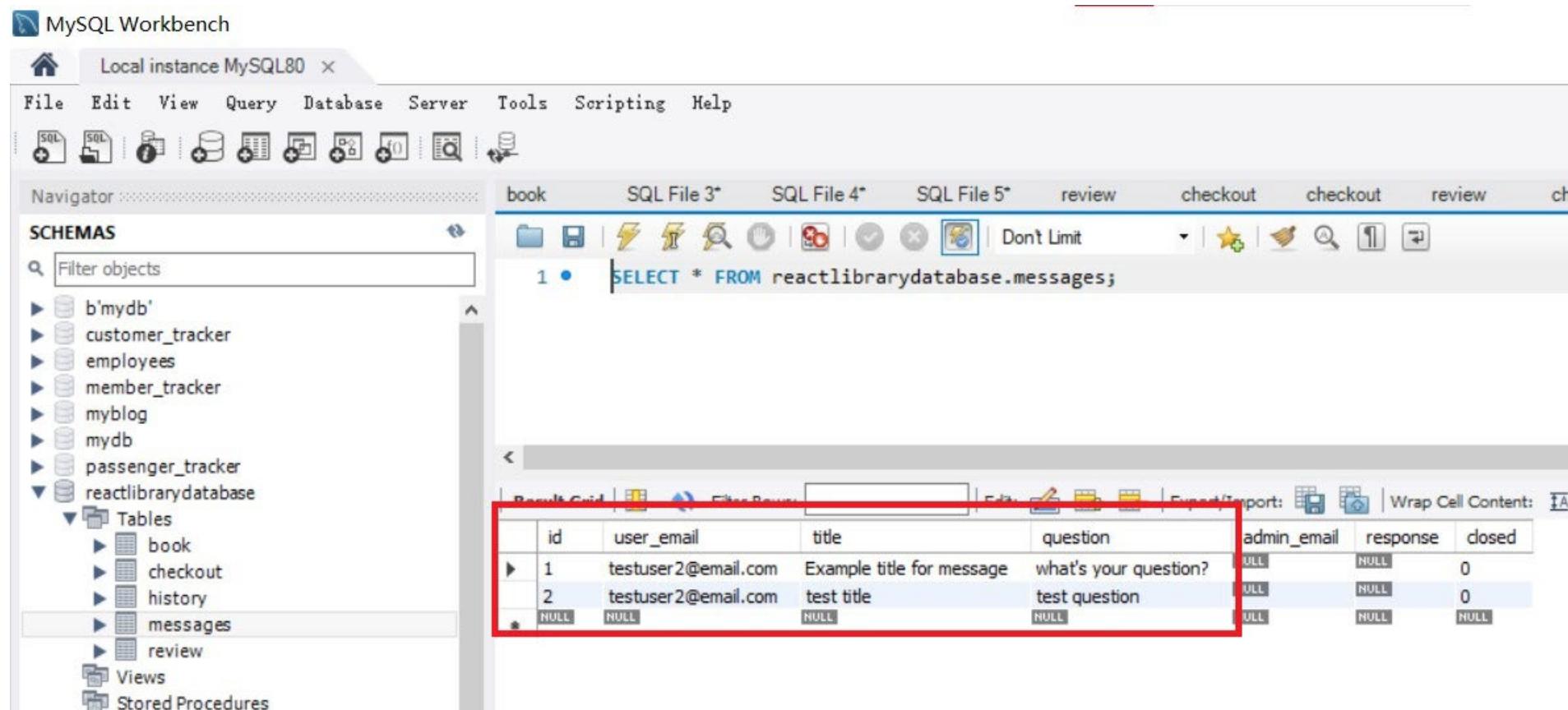
Ask question to JAC Read Admin

Title

Question

Submit Question

Go to our database, our question has added to our “messages” table successfully.



The screenshot shows the MySQL Workbench interface with the following details:

- Navigator:** Shows the database structure with the 'reactlibrarydatabase' schema expanded, revealing tables like 'book', 'checkout', 'history', 'messages', and 'review'.
- SQL Editor:** Displays the query: `SELECT * FROM reactlibrarydatabase.messages;`.
- Result Grid:** Shows the output of the query as a table with the following data:

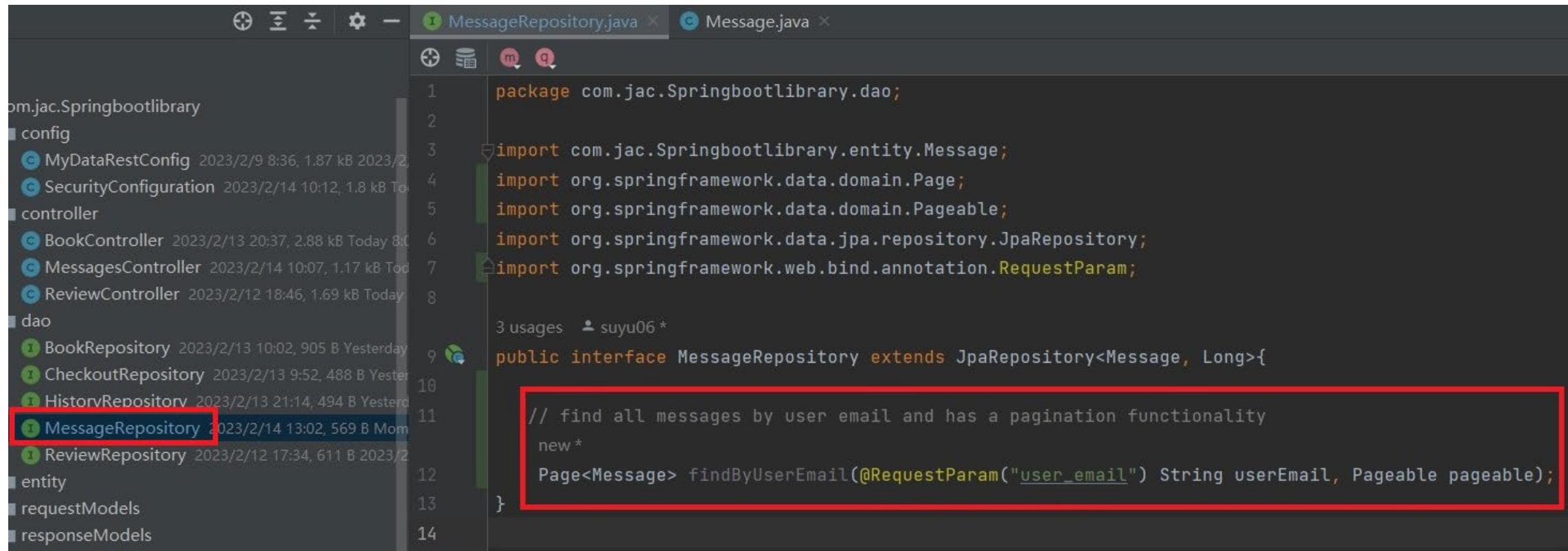
	id	user_email	title	question	admin_email	response	closed
1	1	testuser2@email.com	Example title for message	what's your question?	NULL	NULL	0
*	2	testuser2@email.com	test title	test question	NULL	NULL	0

10. SPRINGBOOT MESSAGE REPOSITORY AND SECURITY



Add to our message repository so we can find all messages by a user email.

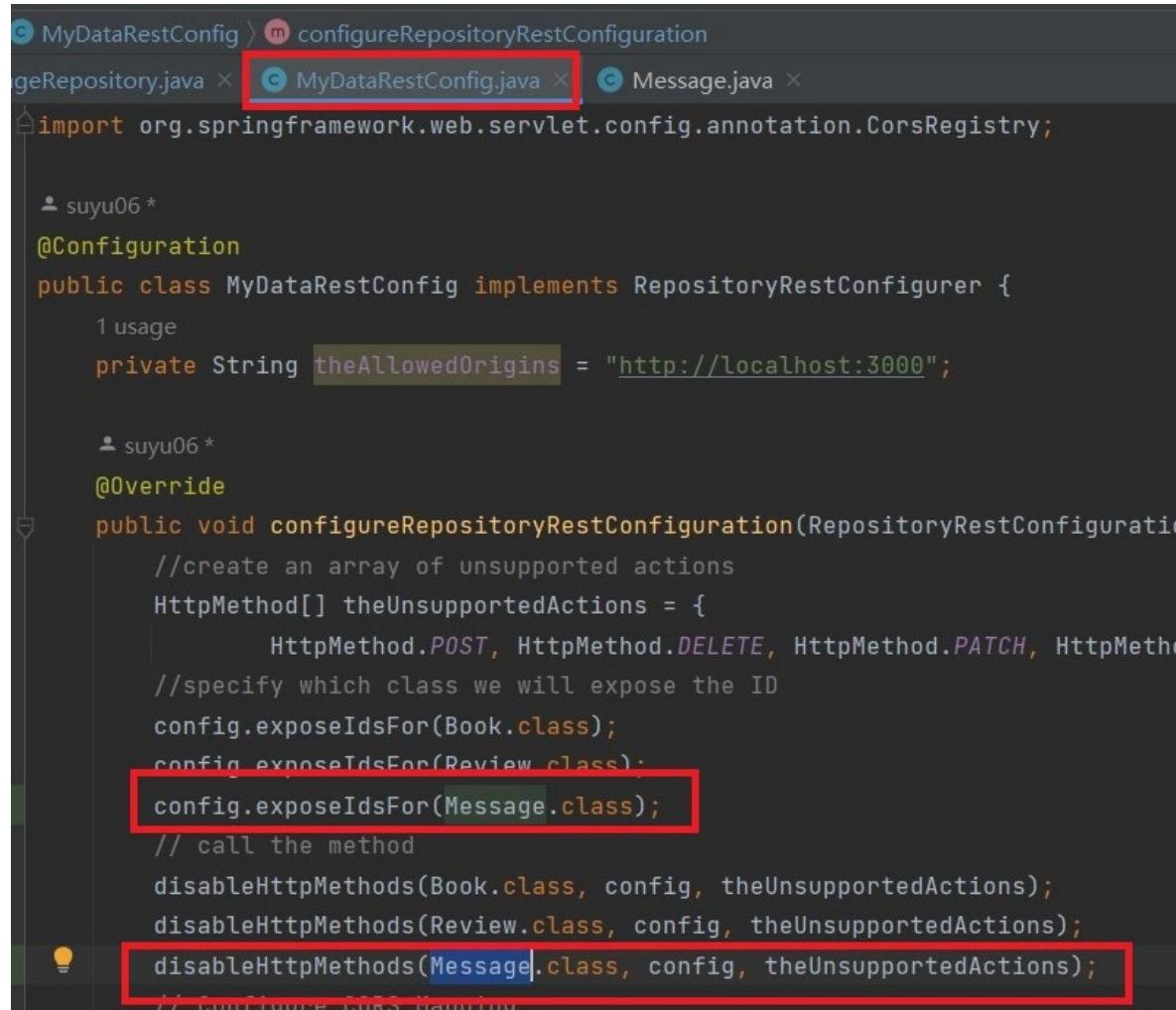
Step 1, Go to “dao” package and MessageRepository, add a fuction findByUserEmail, passing String userEmail, Pageable pageable as its parameters.



The screenshot shows a Java IDE interface with two tabs open: "MessageRepository.java" and "Message.java". The left sidebar lists various Java files in the project structure, including "BookRepository", "CheckoutRepository", "HistoryRepository", "MessageRepository" (which is highlighted with a red box), "ReviewRepository", and others. The "MessageRepository.java" tab contains the following code:

```
1 package com.jac.Springbootlibrary.dao;
2
3 import com.jac.Springbootlibrary.entity.Message;
4 import org.springframework.data.domain.Page;
5 import org.springframework.data.domain.Pageable;
6 import org.springframework.data.jpa.repository.JpaRepository;
7 import org.springframework.web.bind.annotation.RequestParam;
8
9 public interface MessageRepository extends JpaRepository<Message, Long>{
10
11     // find all messages by user email and has a pagination functionality
12     new *
13     Page<Message> findByUserEmail(@RequestParam("user_email") String userEmail, Pageable pageable);
14 }
```

A red box highlights the entire code block for the findByUserEmail method, indicating it is the new addition being discussed.



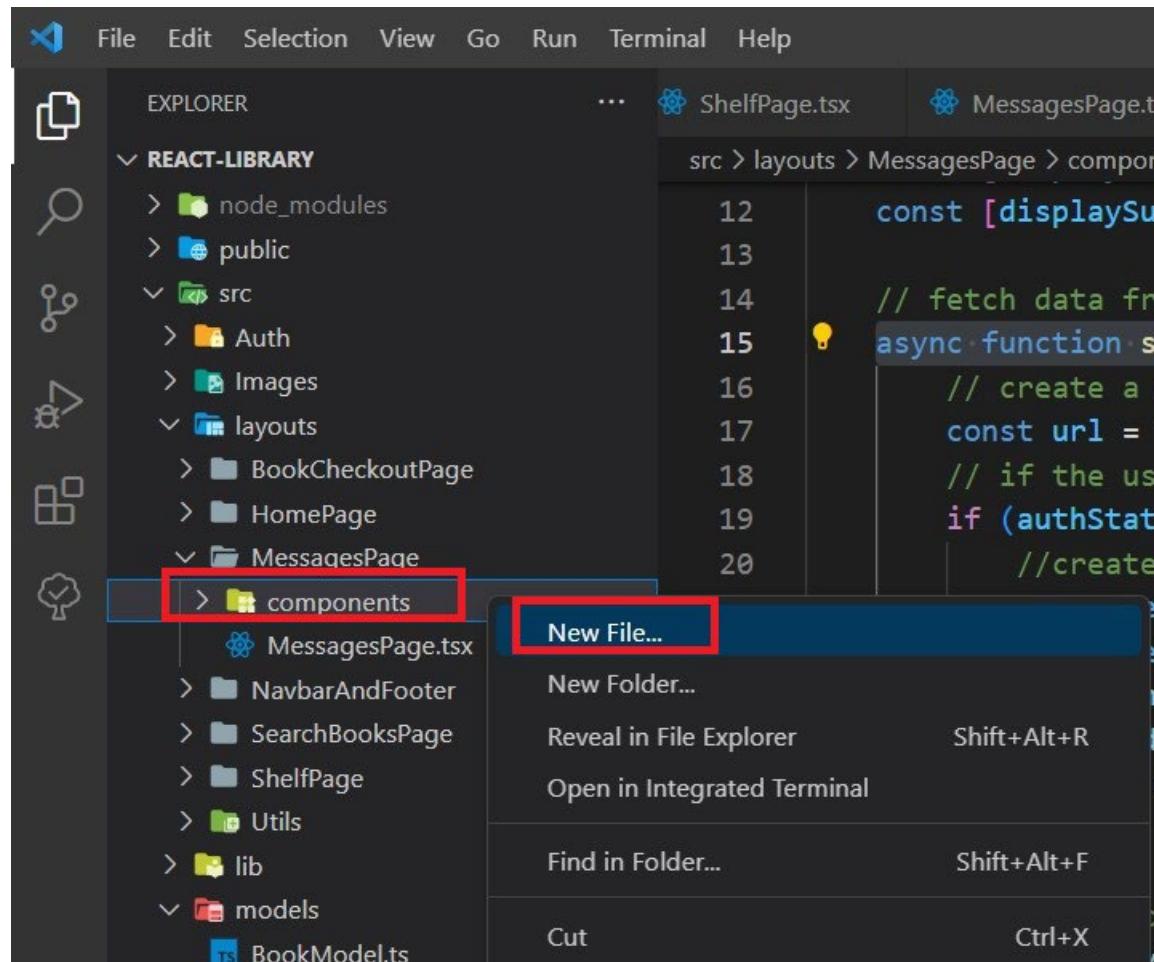
```
MyDataRestConfig > m configureRepositoryRestConfiguration
geRepository.java × MyDataRestConfig.java × Message.java ×
import org.springframework.web.servlet.config.annotation.CorsRegistry;
 
 * suyu06 *
@Configuration
public class MyDataRestConfig implements RepositoryRestConfigurer {
    1 usage
    private String theAllowedOrigins = "http://localhost:3000";
 
    * suyu06 *
@Override
public void configureRepositoryRestConfiguration(RepositoryRestConfiguration config) {
    //create an array of unsupported actions
    HttpMethod[] theUnsupportedActions = {
        HttpMethod.POST, HttpMethod.DELETE, HttpMethod.PATCH, HttpMethod.PUT
    };
    //specify which class we will expose the ID
    config.exposeIdsFor(Book.class);
    config.exposeIdsFor(Review.class);
    config.exposeIdsFor(Message.class);
    // call the method
    disableHttpMethods(Book.class, config, theUnsupportedActions);
    disableHttpMethods(Review.class, config, theUnsupportedActions);
    disableHttpMethods(Message.class, config, theUnsupportedActions);
    // Configure CORS Mapping
}
```

Step 2, go to MyDataRestConfig file, expose the Ids for message and we want to disable HTTPS methods for message.

11.REACT MESSAGE COMPONENT



Now that we're able to post new messages and we're able to call an endpoint on the backend to get messages, we want to make it so the user can see all of their messages and questions in one easy to see spot.



Step 1, inside MessagesPage, right click components, create a new file called messages.tsx.

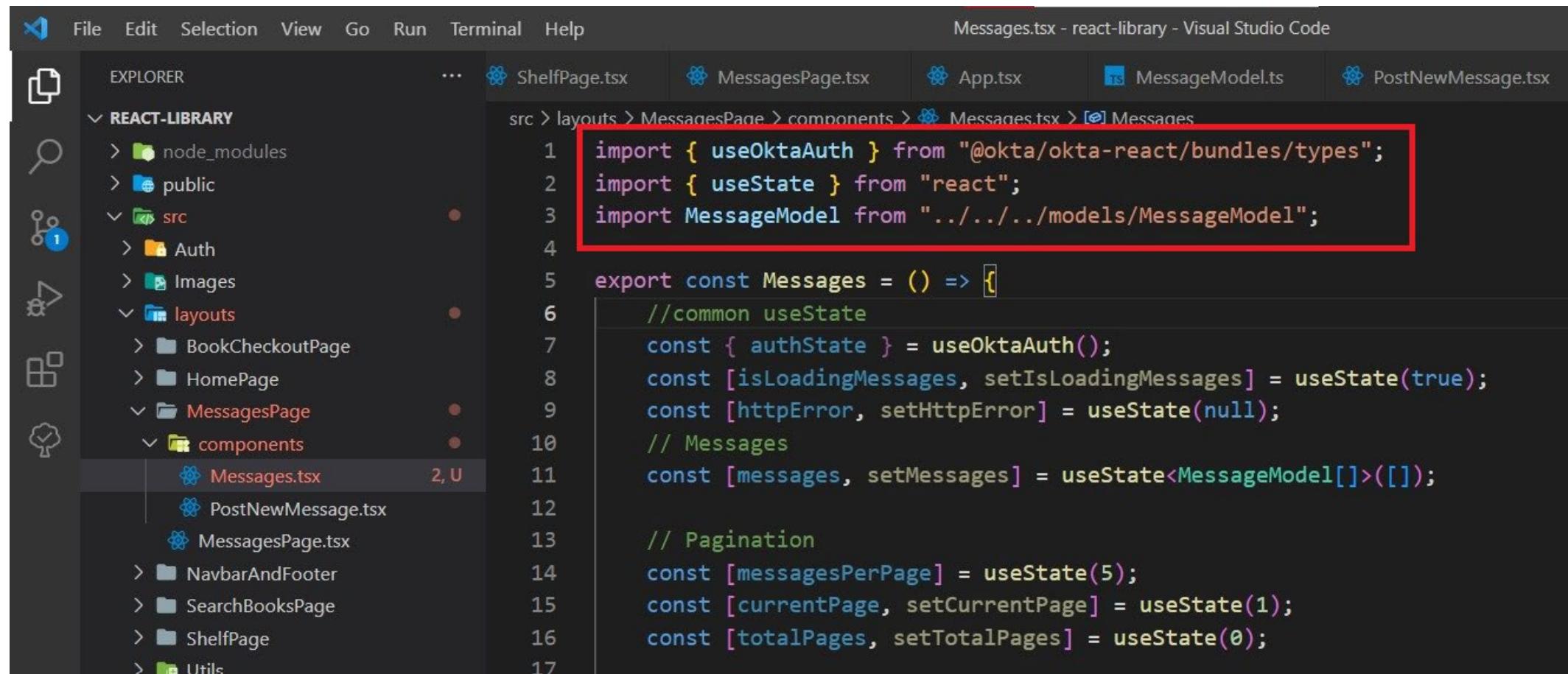
Step 2, create export const Messages stucture.

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

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** Messages.tsx - react-library - Visual Studio Code
- Code Editor:** The current file is Messages.tsx, located at `src > layouts > MessagesPage > components > Messages.tsx`. The code is as follows:

```
1  export const Messages = () => [
2    return(
3      ...
4    );
5  ]
```
- File Explorer:** Shows the project structure under "REACT-LIBRARY".
 - `node_modules`
 - `public`
 - `src` (selected):
 - `Auth`
 - `Images`
 - `layouts` (selected):
 - `BookCheckoutPage`
 - `HomePage`
 - `MessagesPage` (selected):
 - `components` (selected):
 - `Messages.tsx` (highlighted)
 - `PostNewMessage.tsx`
 - `MessagesPage.tsx`

Step3, create useState and import useOktaAuth, useState and MessageModel.



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

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** Messages.tsx - react-library - Visual Studio Code
- Explorer:** Shows the project structure under 'REACT-LIBRARY'. The 'src' folder contains 'Auth', 'Images', 'layouts', 'MessagesPage' (which has 'components' and 'Messages.tsx'), 'NavbarAndFooter', 'SearchBooksPage', 'ShelfPage', and 'Utils'.
- Code Editor:** The file 'Messages.tsx' is open. The code is as follows:

```
1 import { useOktaAuth } from "@okta(okta-react/bundles/types";
2 import { useState } from "react";
3 import MessageModel from "../../../../../models/MessageModel";
4
5 export const Messages = () => {
6   //common useState
7   const { authState } = useOktaAuth();
8   const [isLoadingMessages, setIsLoadingMessages] = useState(true);
9   const [httpError, setHttpError] = useState(null);
10  // Messages
11  const [messages, setMessages] = useState<MessageModel[]>([]);
12
13  // Pagination
14  const [messagesPerPage] = useState(5);
15  const [currentPage, setCurrentPage] = useState(1);
16  const [totalPages, setTotalPages] = useState(0);
```

A red box highlights the first three lines of the code, which are the imports for `useOktaAuth`, `useState`, and `MessageModel`.

Step 4, create useEffect structure.

An async function, then a catch error function and a method to make the page show from the top.

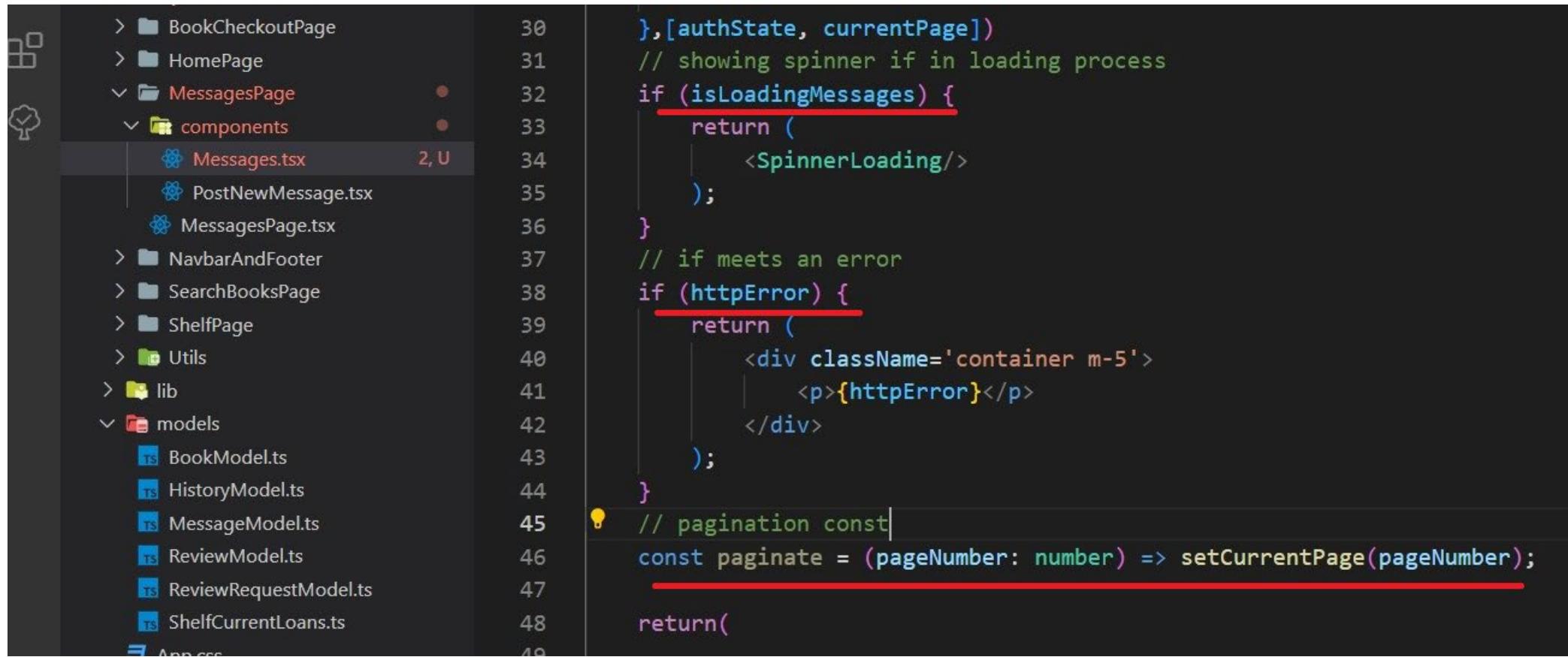
The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer (Left):** Shows the project structure under "REACT-LIBRARY". It includes "node_modules", "public", "src" (with "Auth", "Images", and "layouts" subfolders), "layouts" (with "BookCheckoutPage" and "HomePage" subfolders), "MessagesPage" (with "components" subfolder), and "Messages.tsx".
- Code Editor (Right):** The file "Messages.tsx" is open. The code is as follows:

```
// useEffect
useEffect(() => {
  const fetchUserMessages = async () => {
    ...
  }
  fetchUserMessages().catch(error: any) => {
    setIsLoadingMessages(false);
    setHttpError(error.messages);
  }
  window.scrollTo(0, 0);
}
```

The code editor shows line numbers 18 through 28. The line "const fetchUserMessages = async () => {" is highlighted with a red underline. The entire code block is enclosed in a red box.

Step 5, create loading process, httpError and paginate.



The image shows a code editor interface with a sidebar on the left displaying a file tree. The tree includes 'BookCheckoutPage', 'HomePage', 'MessagesPage' (selected), 'components' (selected), 'Messages.tsx', 'PostNewMessage.tsx', 'MessagesPage.tsx', 'NavbarAndFooter', 'SearchBooksPage', 'ShelfPage', 'Utils', 'lib', and 'models' (selected). Under 'models', files like 'BookModel.ts', 'HistoryModel.ts', 'MessageModel.ts', 'ReviewModel.ts', 'ReviewRequestModel.ts', and 'ShelfCurrentLoans.ts' are listed. At the bottom of the sidebar is an 'App.css' file. The main editor area shows a component implementation with line numbers from 30 to 49. The code handles loading processes, errors, and pagination. A yellow cursor is positioned at the start of the 'const paginate' declaration.

```
30 }, [authState, currentPage])
31 // showing spinner if in loading process
32 if (isLoadingMessages) {
33     return (
34         <SpinnerLoading/>
35     );
36 }
37 // if meets an error
38 if (httpError) {
39     return (
40         <div className='container m-5'>
41             <p>{httpError}</p>
42         </div>
43     );
44 }
45 // pagination const
46 const paginate = (pageNumber: number) => setCurrentPage(pageNumber);
47
48 return(
```

Step 6, fill in useEffect with async fetchUserMessages();

```
useEffect(() => {
  const fetchUserMessages = async () => {
    // only if user is authenticated
    if (authState && authState?.isAuthenticated) [
      // save the string url to a const
      const url = `http://localhost:8080/api/messages/search/findByUserEmail
      /?userEmail=${authState?.accessToken?.claims.sub}&page=${currentPage - 1}&size=${messagesPerPage}`;
      const requestOptions = {
        method: 'GET',
        headers: {
          Authorization: `Bearer ${authState?.accessToken?.accessToken}`,
          'Content-Type': 'application/json'
        }
      };
      // fetch the data with url and right request type and header
      const messagesResponse = await fetch(url, requestOptions);
      // if there is an fetching error
      if (!messagesResponse.ok) {
        throw new Error('Something went wrong!');
      }
      // convert to json object
      const messagesResponseJson = await messagesResponse.json();
      // save the message fetched
      setMessages(messagesResponseJson._embedded.messages);
      // save the total pages fetched
      setTotalPages(messagesResponseJson.page.totalPages);
    ]
    setIsLoadingMessages(false);
  }
}
```

12.REACT MESSAGE COMPONENT HTML/CSS



First, judge whether.message exists, if exists:

Messages.tsx - react-library - Visual Studio Code

message.tsx Messages.tsx 1, M X Settings LoansModal.tsx

> MessagesPage > components > Messages.tsx > [e] Messages > messages.map() callback

```
return (
  <div className="mt-2">
    /* judge if the message exists */
    {messages.length > 0 ? (
      <>
        /* if message exists */
        <h5>Current Q/A: </h5>
        {messages.map((message) => (
          <div key={message.id}>
            <div className="card mt-2 shadow p-3 bg-body rounded">
              <h5>
                Case #{message.id}: {message.title}
              </h5>
              <h6>{message.userEmail}</h6>
              <p>{message.question}</p>
              <hr />
```

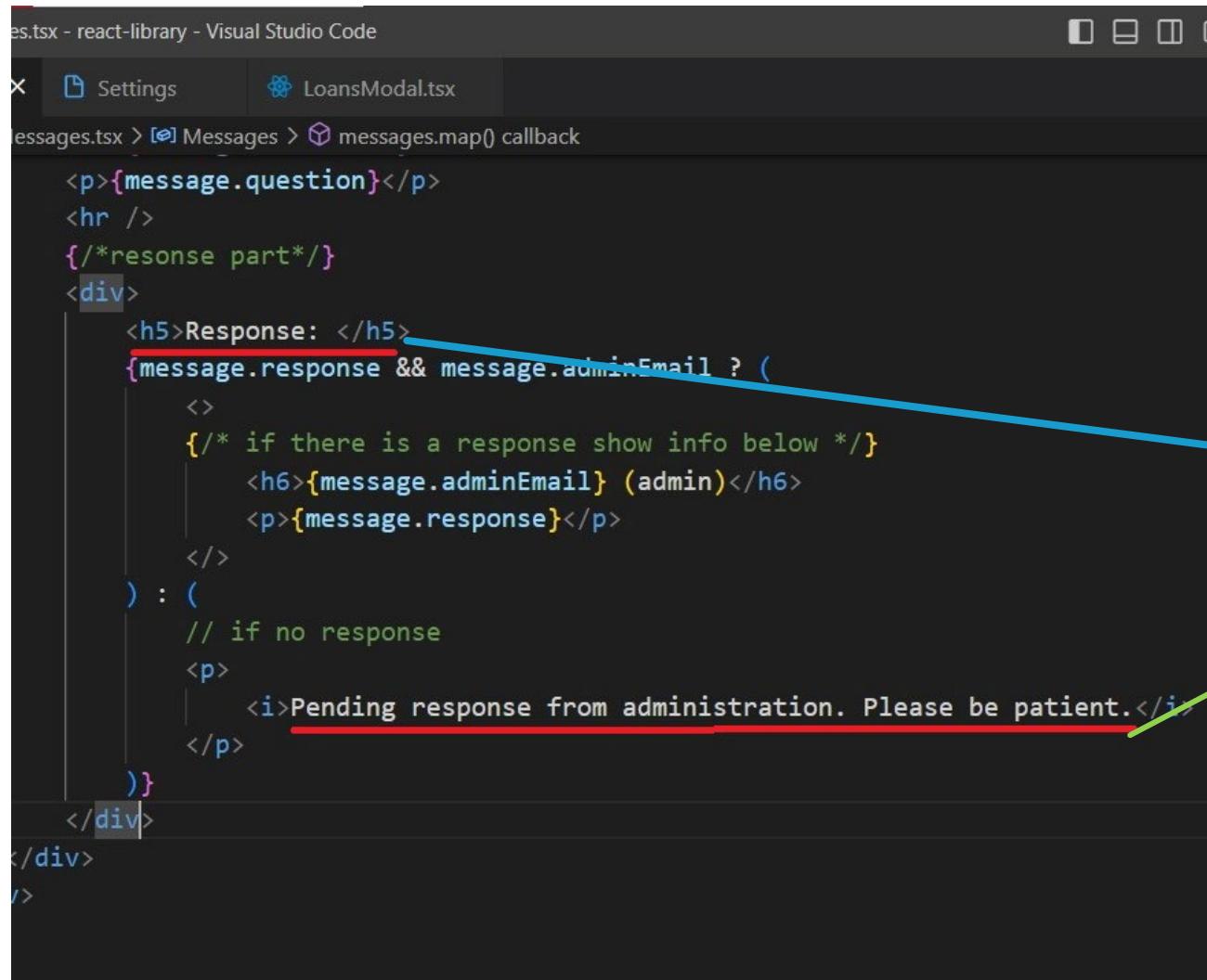
Current Q/A:

Case #1 Example title for message

testuser2@email.com

what's your question?

Then whether there is a Response :



```
es.tsx - react-library - Visual Studio Code
x Settings LoansModal.tsx
messages.tsx > [?] Messages > ↗ messages.map() callback
<p>{message.question}</p>
<hr />
{/*response part*/}
<div>
  <h5>Response: </h5>
  {message.response && message.adminEmail ? (
    <>
      /* if there is a response show info below */
      <h6>{message.adminEmail} (admin)</h6>
      <p>{message.response}</p>
    </>
  ) : (
    // if no response
    <p>
      <i>Pending response from administration. Please be patient.</i>
    </p>
  )
</div>
</div>
/>
```

Current Q/A:

Case #1 Example title for message
testuser2@email.com
what's your question?

Response:

Pending response from administration. Please be patient.

If there is no any question from the user, just show “All questions you submit will be shown here”. And then, add a pagination function.

```
95           <p>{message.response}</p>
96       </>
97     ) : (
98       // if no response
99       <p>
100      <i>Pending response from administration. Please be patient.</i>
101     </p>
102   )}
103   </div>
104
105
106  )})
107
108 ) : (
109   // no message for the user
110   <h5>All questions you submit will be shown here</h5>
111 )
112 /* panination */
113 {totalPages > 1 && <Pagination currentPage={currentPage} totalPages={totalPages} paginate={paginate}>
114   </div>
115 }
116 }
```

Go to MessagesPage.tsx, replace p tag messages with component.

The screenshot shows a Visual Studio Code interface with the title bar "MessagesPage.tsx - react-library - Visual Studio Code". The left sidebar displays a file tree with "PostNewMessage.tsx", "Messages.tsx 1, M", "MessagesPage.tsx X", and "MessagesPage". The main editor area shows the code for "MessagesPage.tsx". A red box highlights the tab for "MessagesPage.tsx". Another red box highlights the line of code at line 52, which contains a conditional statement: `{messagesClick ? <p>Messages</p>:<></>}`. The code is written in JSX, using class names like "container", "tab-content", and "tab-pane" along with IDs and roles for accessibility.

```
10     <div className="container">
11         <div className="mt-3 mb-2">
12             <nav>...
13             </nav>
14             <div className="tab-content" id="nav-tabContent">
15                 <div
16                     className="tab-pane fade show active"
17                     id="nav-send-message"
18                     role="tabpanel"
19                     aria-labelledby="nav-send-message-tab">
20                         <PostNewMessage/>
21                     </div>
22                     <div className="tab-pane fade" id="nav-message" role="tabpanel" aria-labelledby="nav-message-tab">
23                         /* if it's clicked, show p tag or show nothing */
24                         {messagesClick ? <p>Messages</p>:<></>}
25                     </div>
26             </div>
27         </div>
28     </div>
```

If there is any error, right click and choose “Quick Fix”, to import the component.

MessagesPage.tsx - react-library - Visual Studio Code

NewMessage.tsx Messages.tsx 1, M MessagesPage.tsx M X

layouts > MessagesPage > MessagesPage.tsx > [!] MessagesPage

```
return (
  <div className="container">
    <div className="mt-3 mb-2">
      <nav> ...
      </nav>
      <div className="tab-content" id="nav-tabContent">
        <div
          className="tab-pane fade show active"
          id="nav-send-message"
          role="tabpanel"
          aria-labelledby="nav-send-message-tab">
          <PostNewMessage/>
        </div>
        <div className="tab-pane fade" id="nav-message" role="tabpanel" aria-labelledby="nav-message-tab">
          /* if it's clicked, show messages component or show nothing */
          {messagesClick ? <Messages/> : <></>}
        </div>
      </div>
    </div>
  </div>
```

React App

localhost:3000/messages

Submit Question Q/A Response/Pending

Current Q/A:

Case #1: Example title for message
testuser2@email.com
what's your question?

Response:
Pending response from administration. Please be patient.

Case #2: test title
testuser2@email.com
test question

Response:
Pending response from administration. Please be patient.

Case #3: test move banner
testuser2@email.com
successful or not

Response:
Pending response from administration. Please be patient.

<http://localhost:3000/messages>, click on the “Q/A Response/Pending” tab, it will show all the messages the current user submitted. There are 3 questions submitted by the “testuser2@email.com”

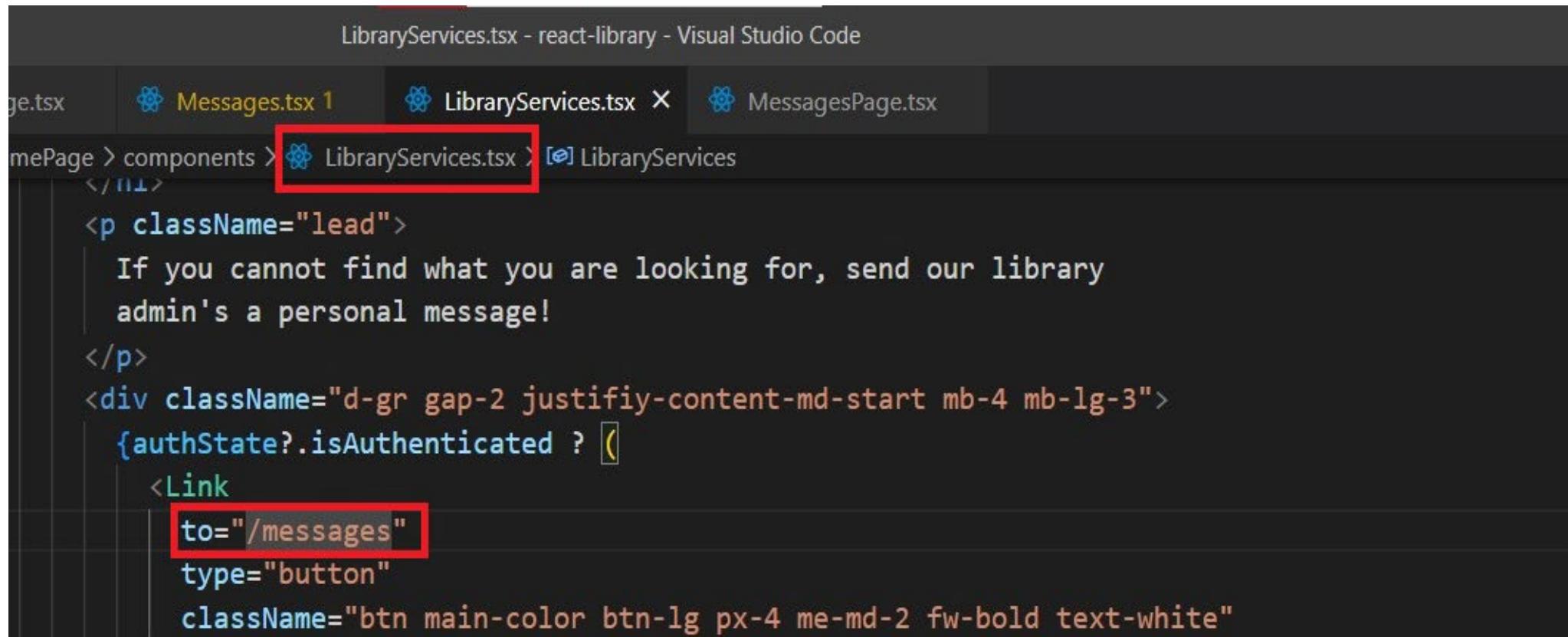
Let's check the result in our data base.

In the table messages, the “testuser2@email.com” has submitted 3 questions.

13.REACT ADD LINK TO LIBRARY SERVICE



Now that we have all of our messages working in all of the components and child components. The last thing we need to do is go into “layouts”, hop into our “HomePage”, go into “components” and let's jump into our “LibraryService.tsx”. Replace the “to= “#” to our message component.

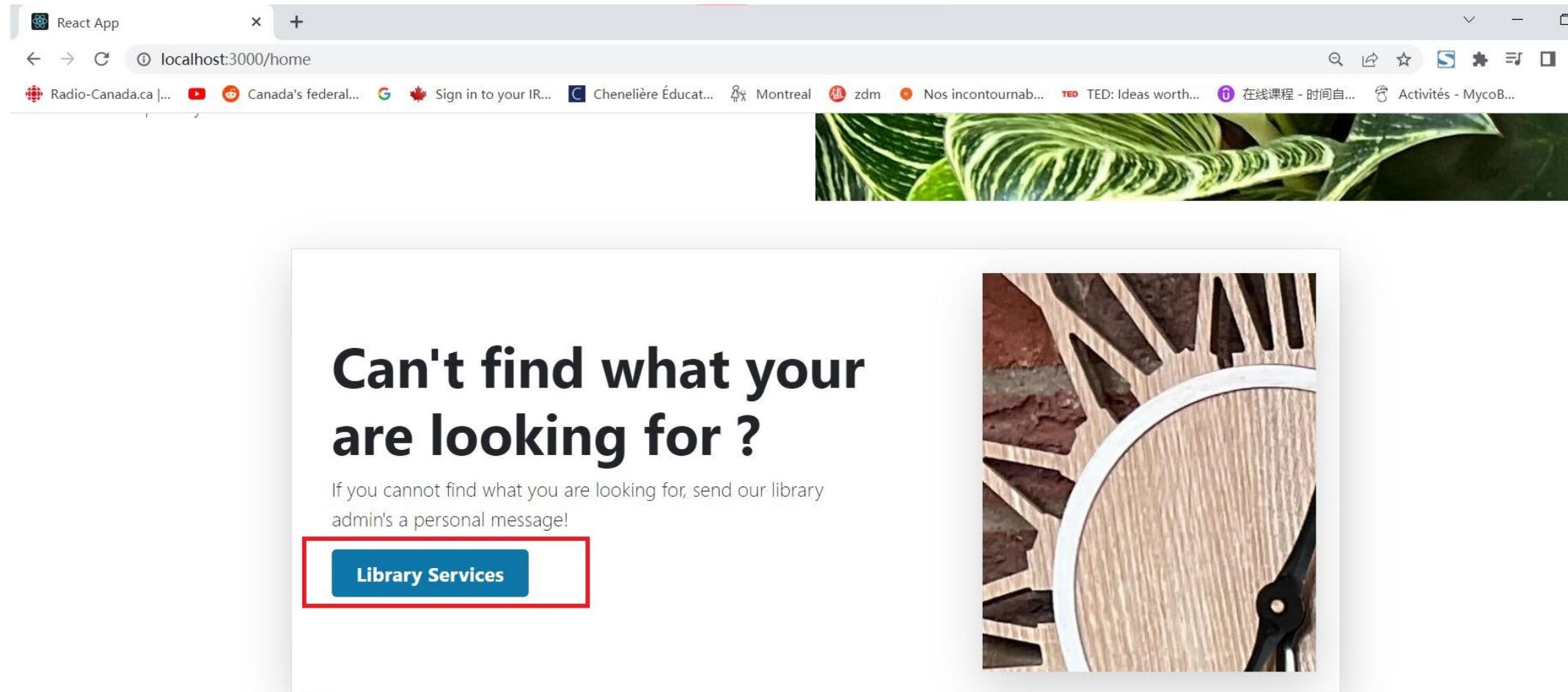


The screenshot shows a Visual Studio Code interface with the title bar "LibraryServices.tsx - react-library - Visual Studio Code". The tabs at the top include "ge.tsx", "Messages.tsx 1", "LibraryServices.tsx X", and "MessagesPage.tsx". Below the tabs, the breadcrumb navigation shows "mePage > components > LibraryServices.tsx > LibraryServices". The main code editor area contains the following TypeScript code:

```
mePage > components > LibraryServices.tsx > LibraryServices
</>
<p className="lead">
  If you cannot find what you are looking for, send our library
  admin's a personal message!
</p>
<div className="d-gr gap-2 justify-content-md-start mb-4 mb-lg-3">
  {authState?.isAuthenticated ? (
    <Link
      to="/messages"
      type="button"
      className="btn main-color btn-lg px-4 me-md-2 fw-bold text-white"
    >
      Find a message
    </Link>
  ) : (
    <div>
      <h3>No messages found</h3>
      <p>Check back later or search for something else!</p>
    </div>
  )}
</div>
```

The line "to="/messages"" is highlighted with a red box. The entire file path "LibraryServices.tsx" in the breadcrumb is also highlighted with a red box.

Now, in the bottom of our home page, for user signed in, we can see the “Library Services” button. If we click on this button:



React App

localhost:3000/home

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... Activités - MycoB...

Can't find what you are looking for ?

If you cannot find what you are looking for, send our library admin's personal message!

[Library Services](#)

It will redirect us to the messages page!

