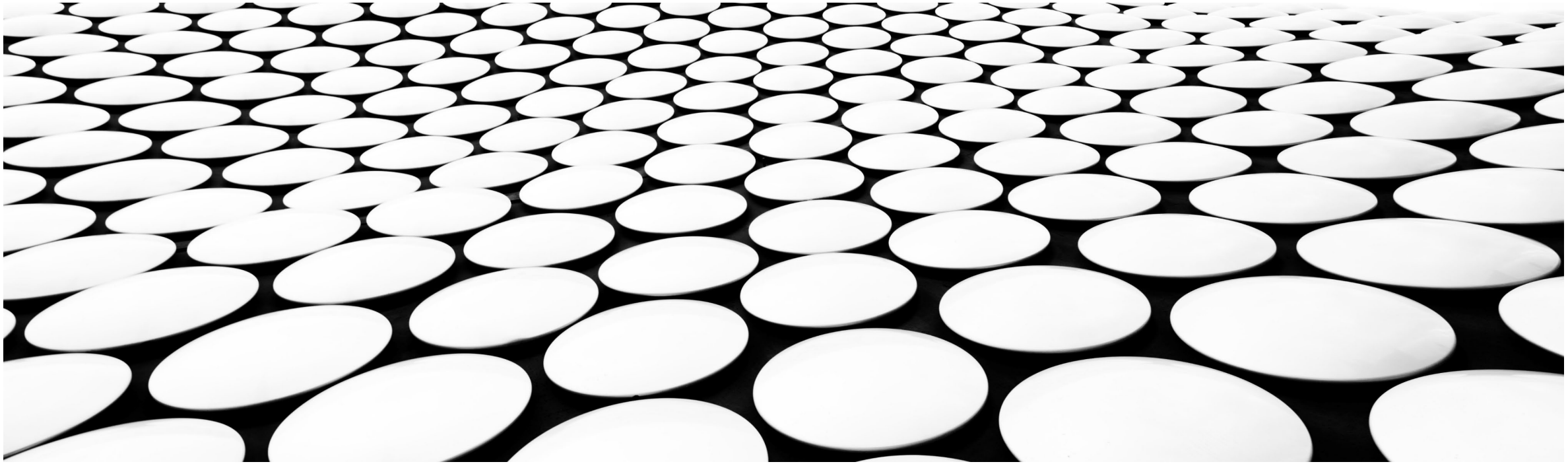
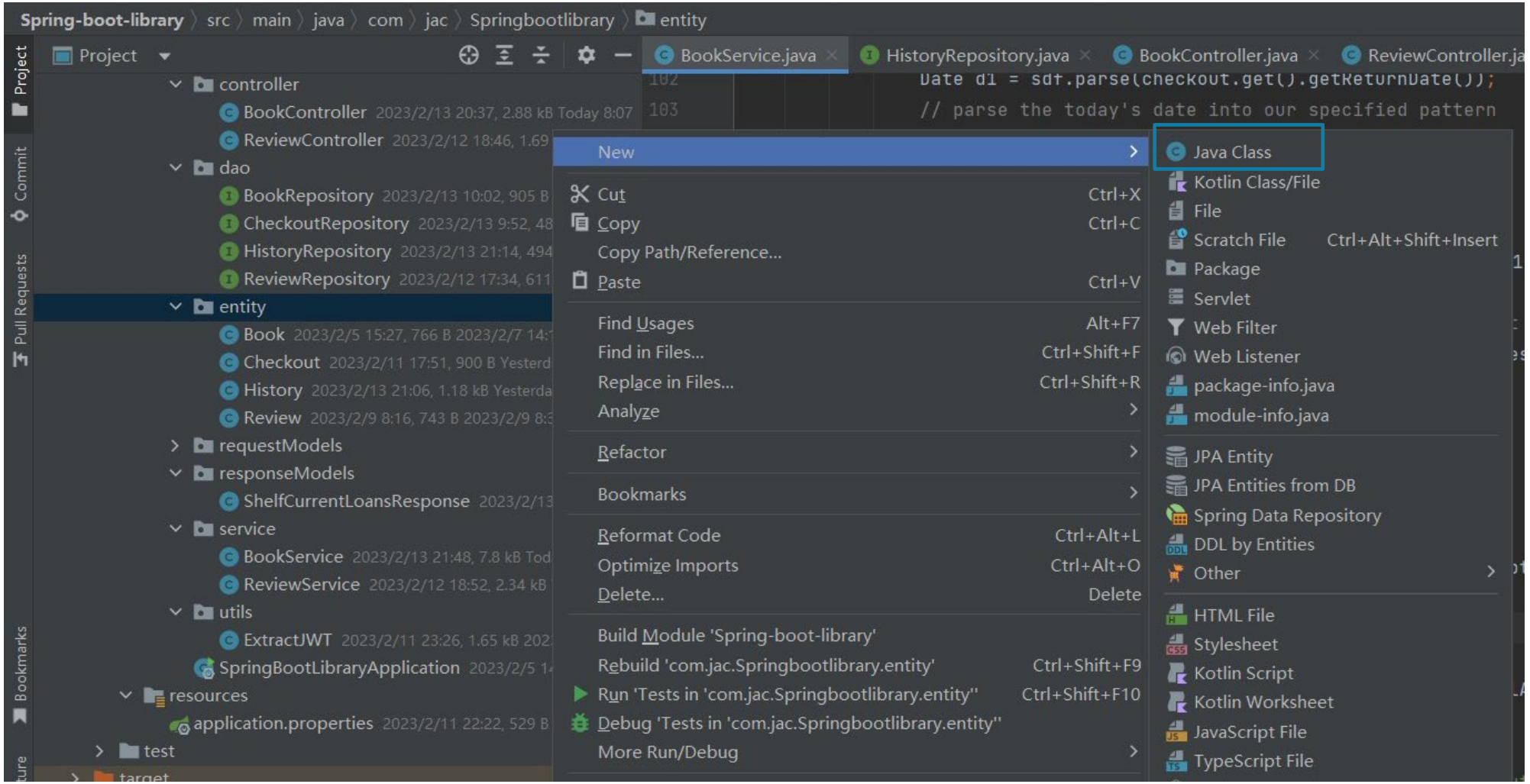
S18 LIBRARY SERVICE

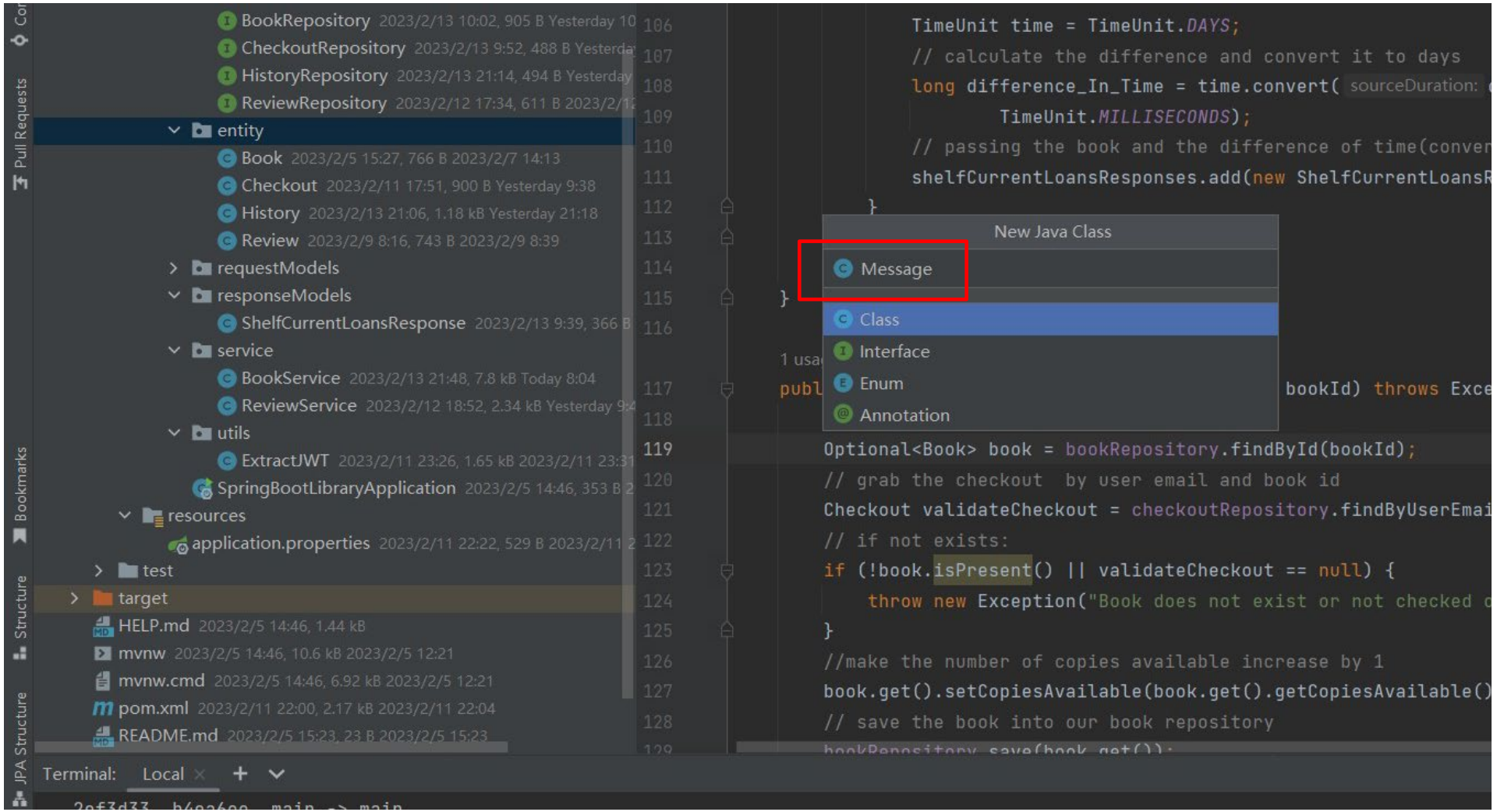


1.SPRINGBOOT MESSAGE ENTITY

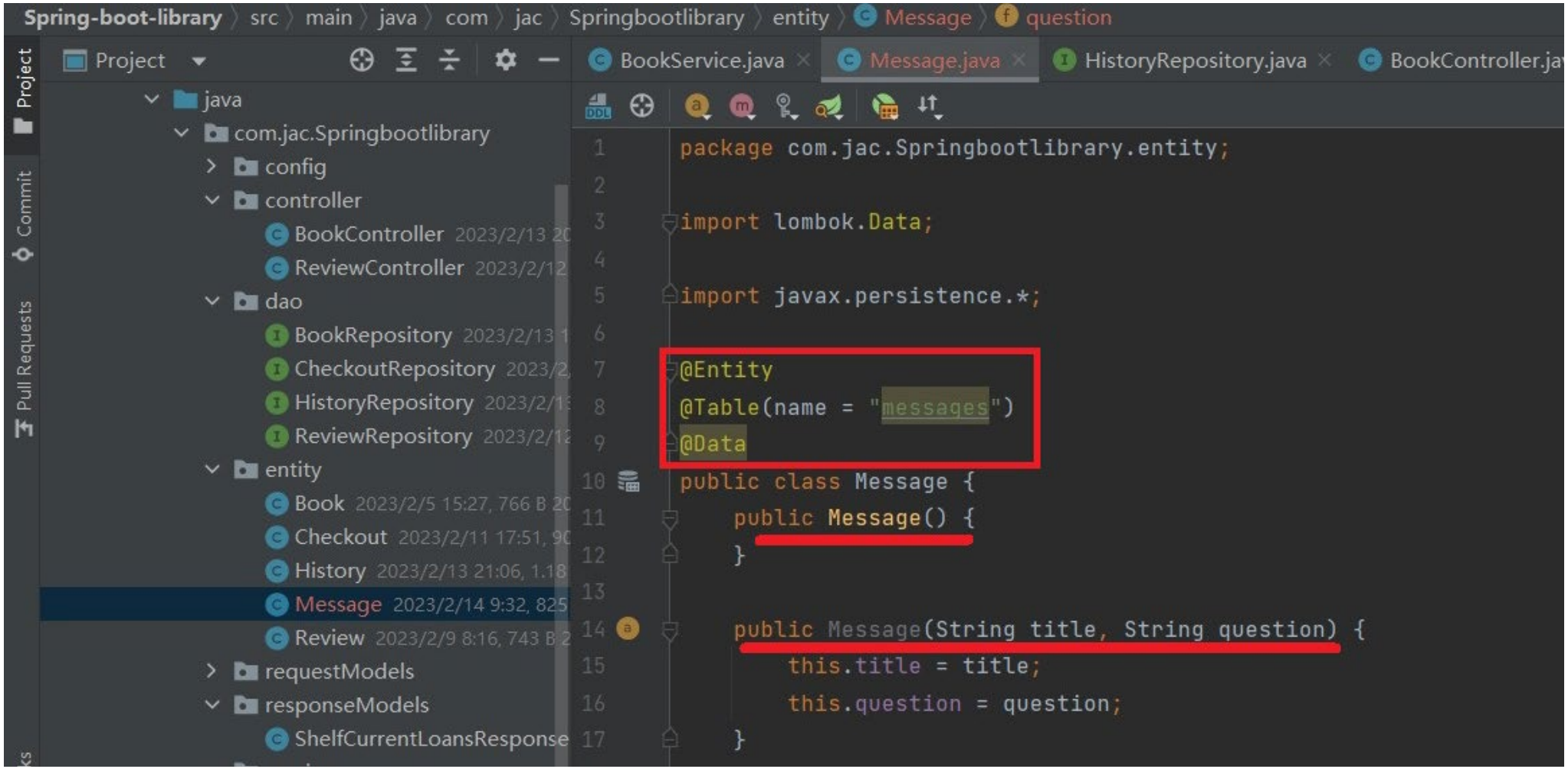
Step 1, create Message class file in entity package.
Right click on our entity package, choose new and java class.

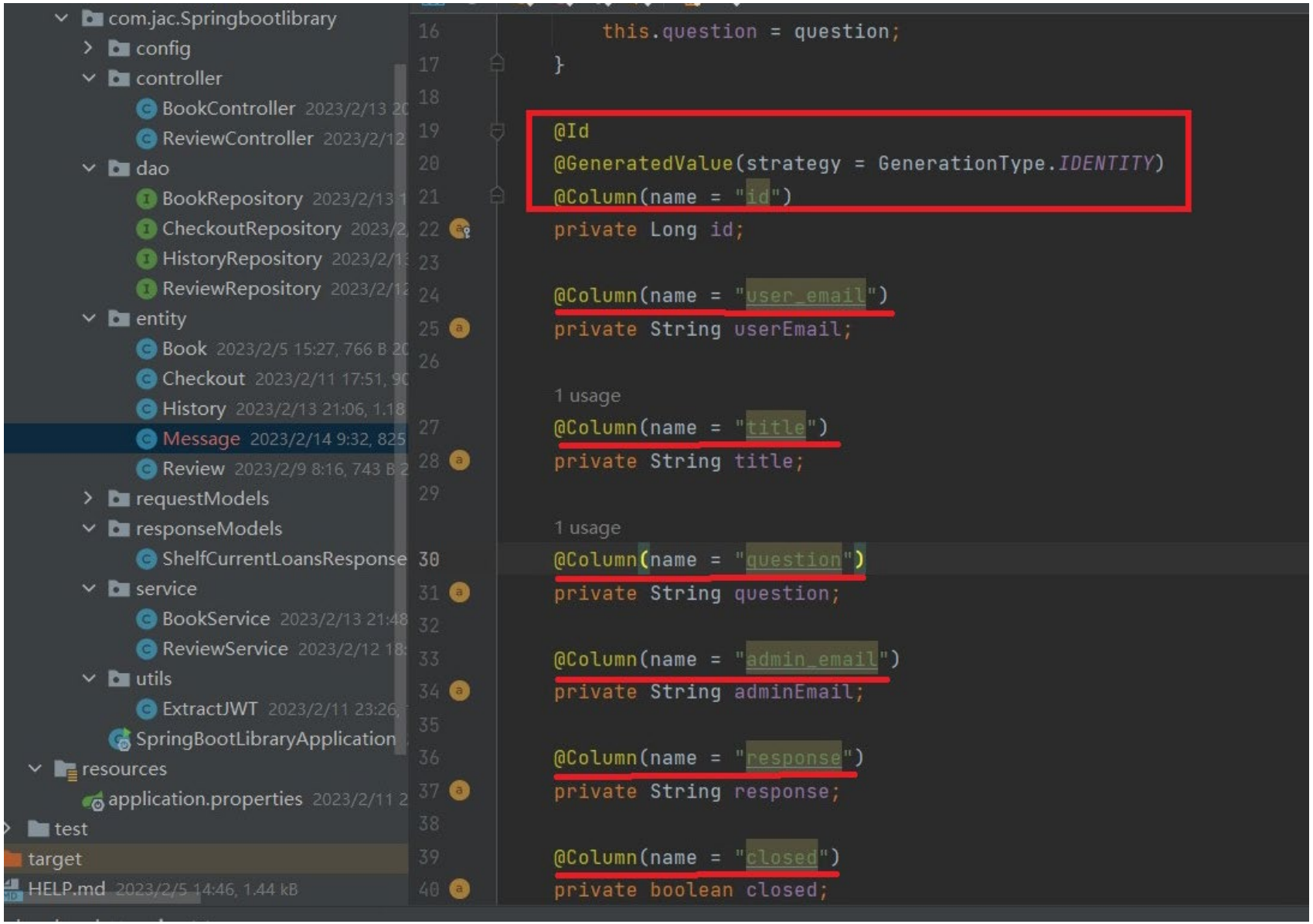


Choose “Class”, and input the class name “Message”.



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.

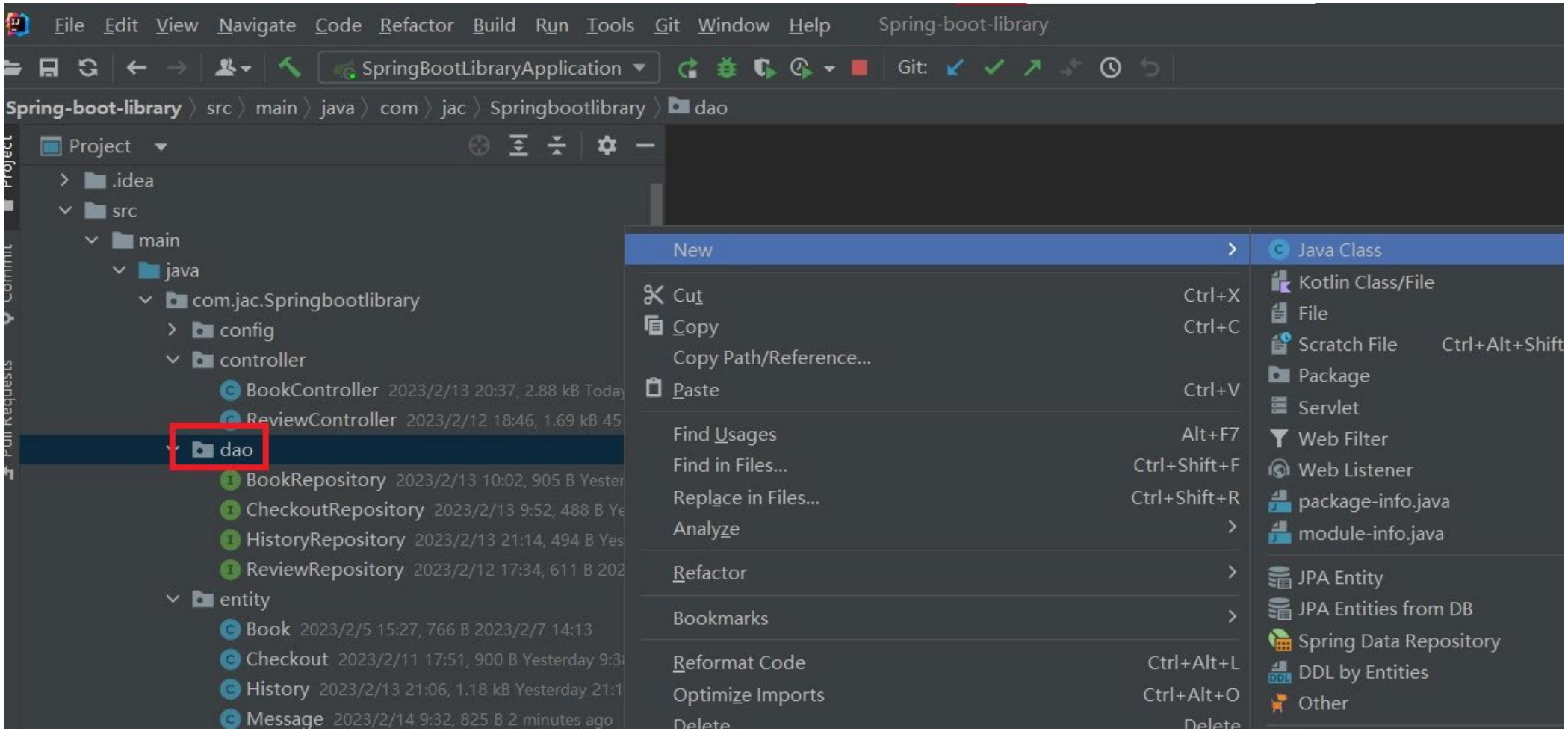




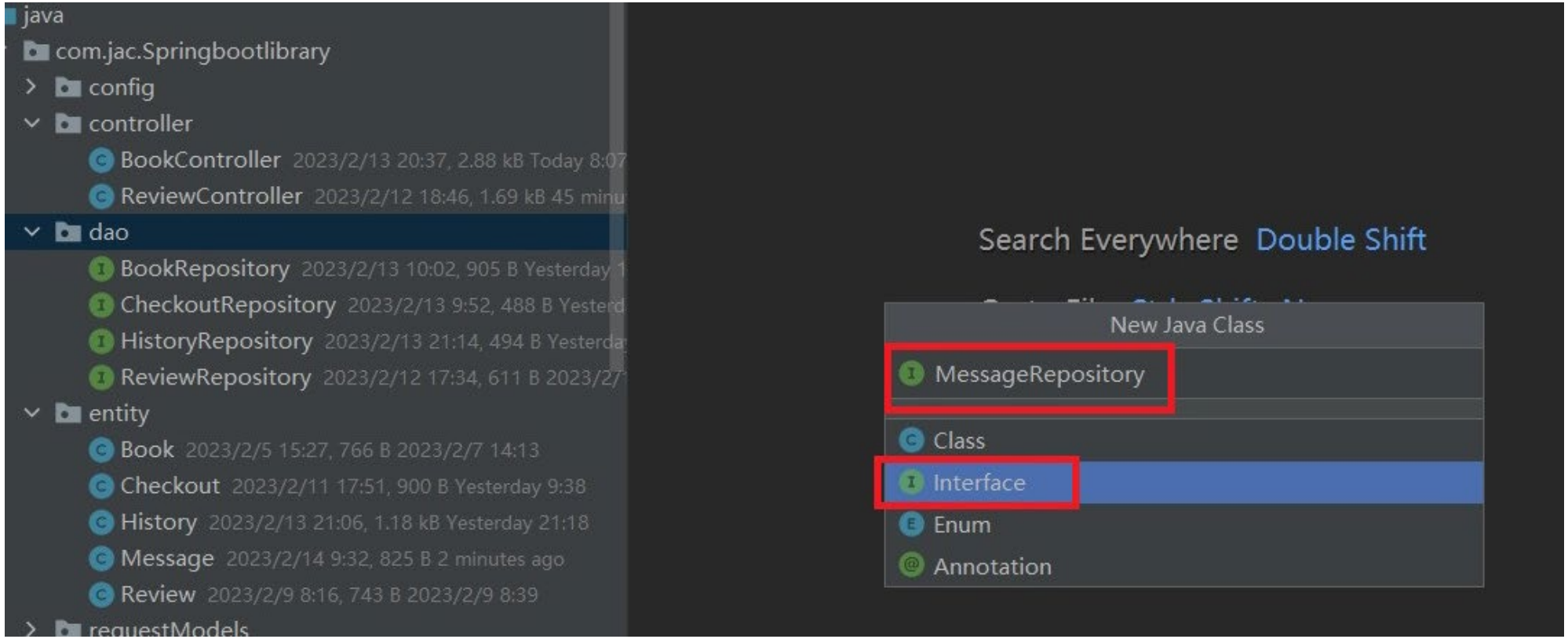
Step 4, create properties which are mapping to columns in our table "messages".

2. SPRINGBOOT MESSAGE REPOSITORY

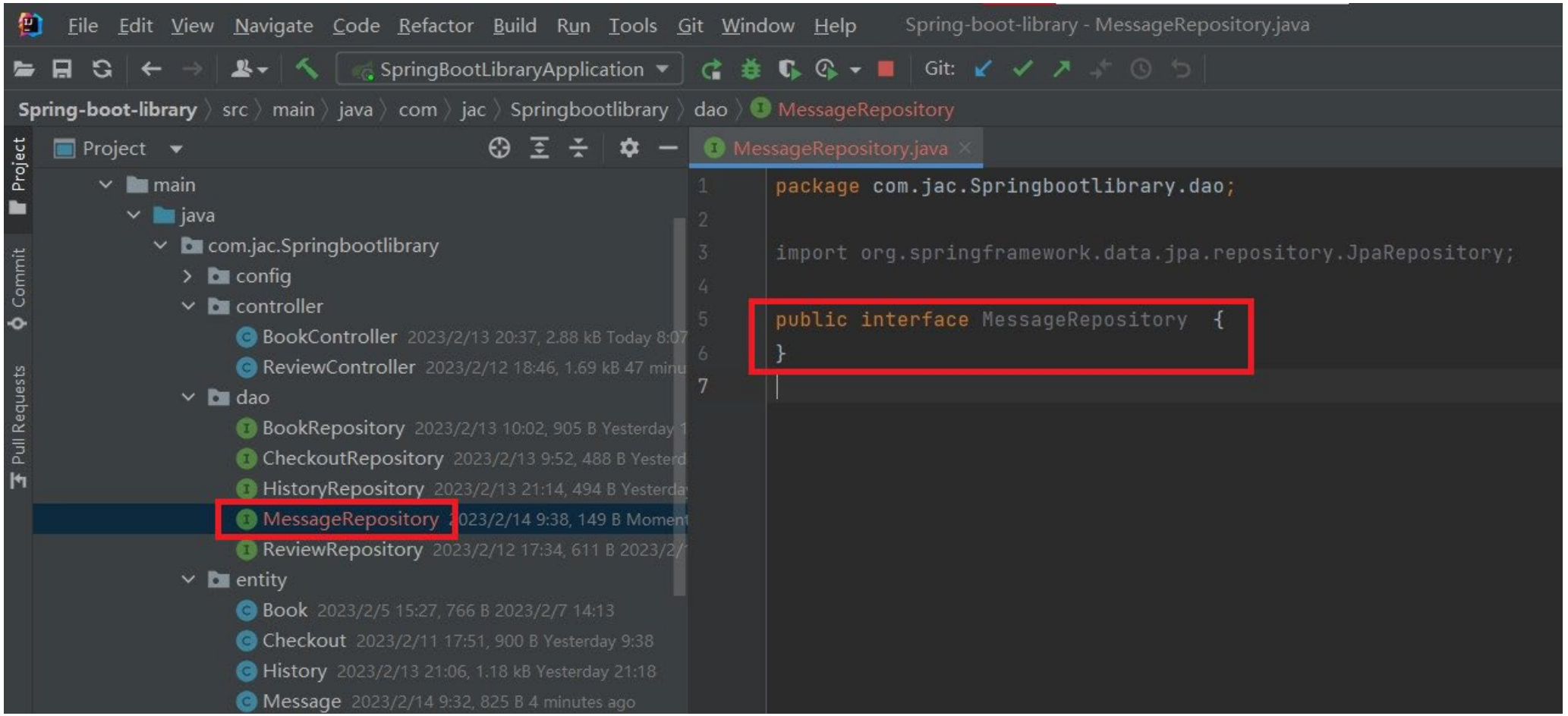
Step1, Create a new interface MessageRepository.
Right click dao directory, choose new ,then Java class.



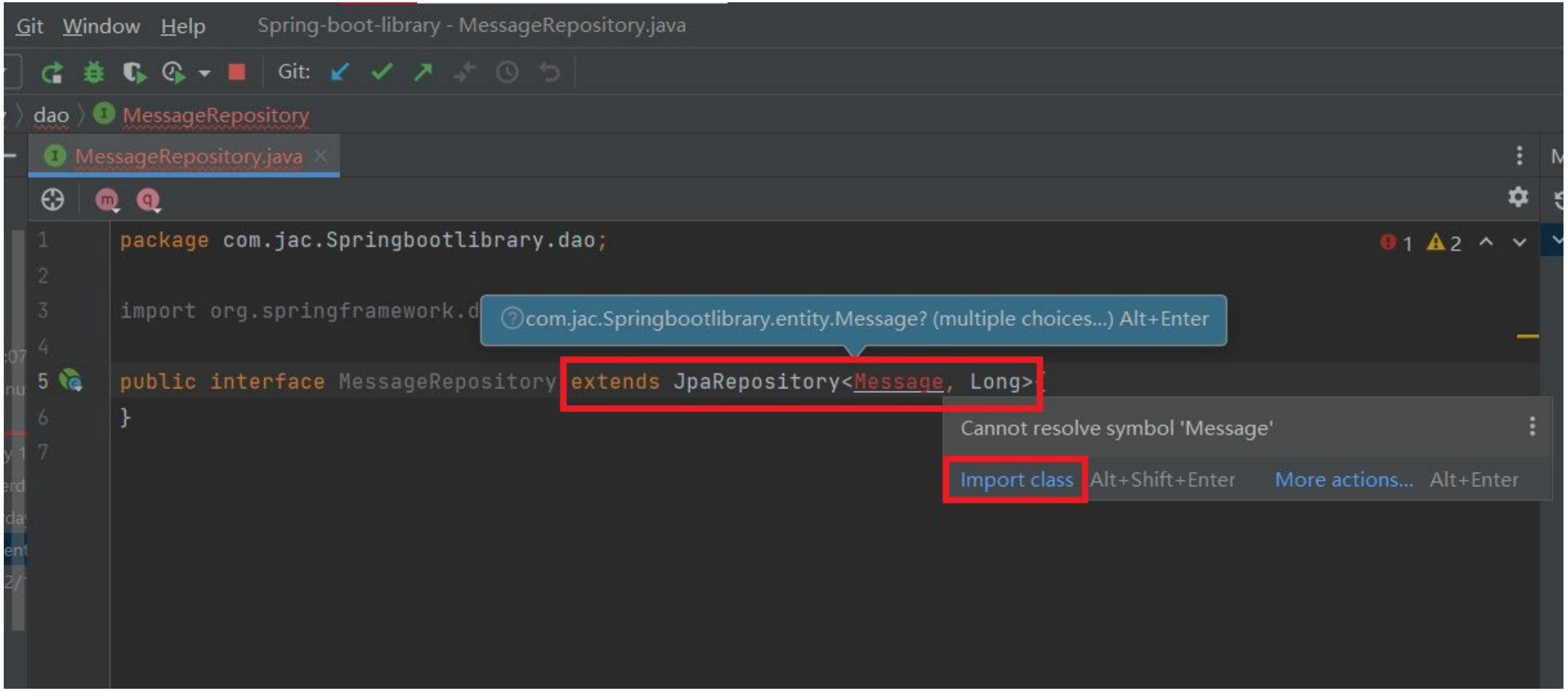
Choose an interface, input name MessageRepository.



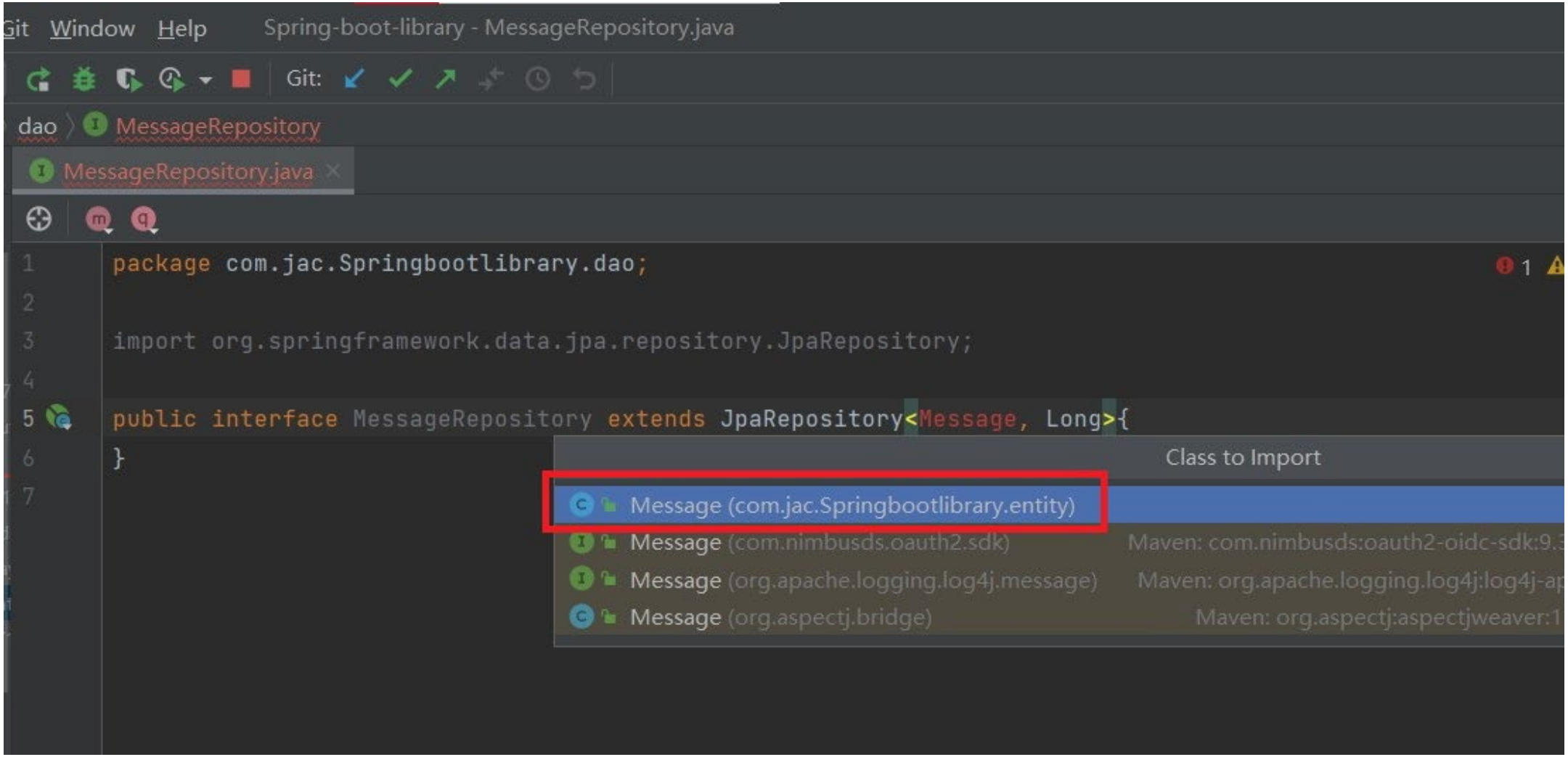
We will get the class as below:



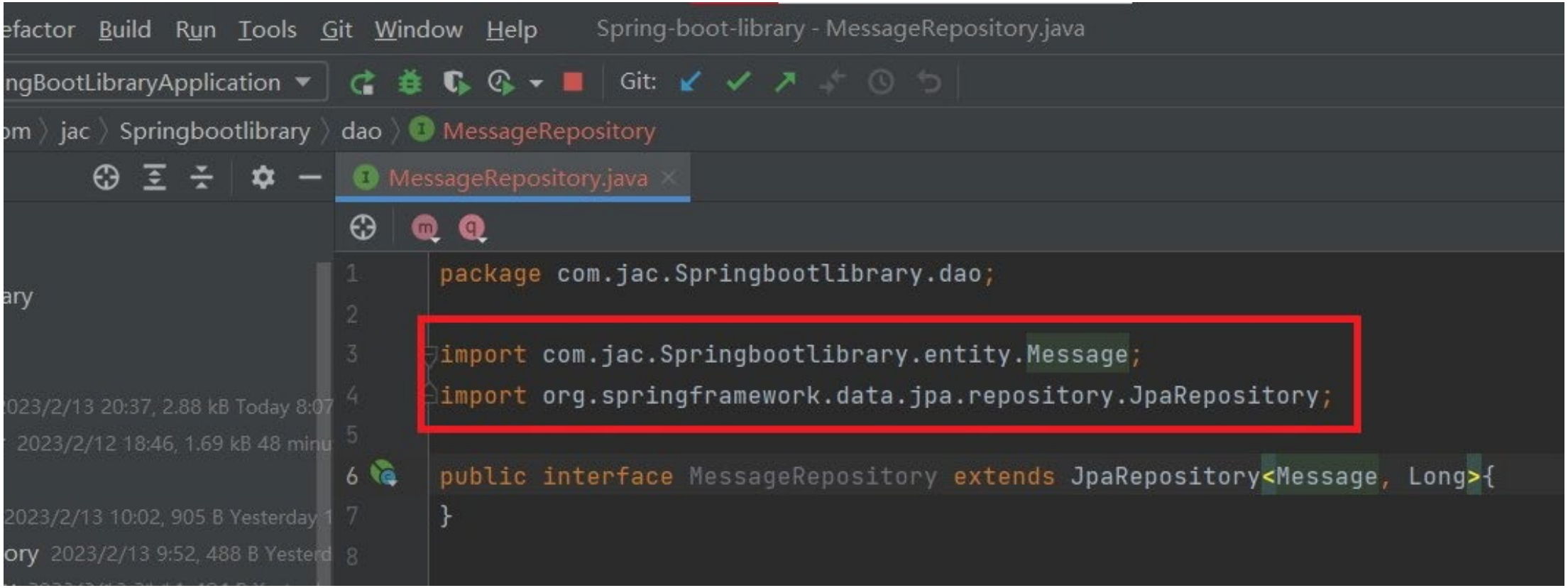
Step 2, add “extends JpaRepository<Message, Long>”.
Click on the “import class” to import Message class.



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

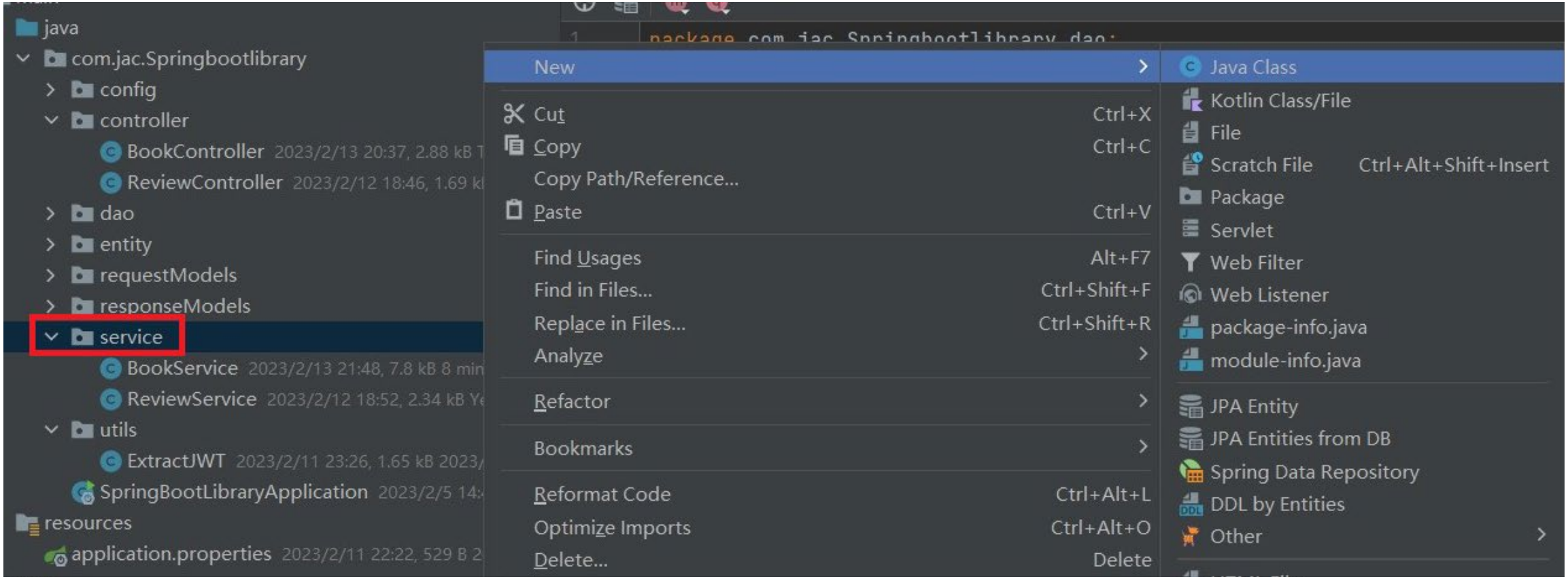


Check the import part is exactly as below:

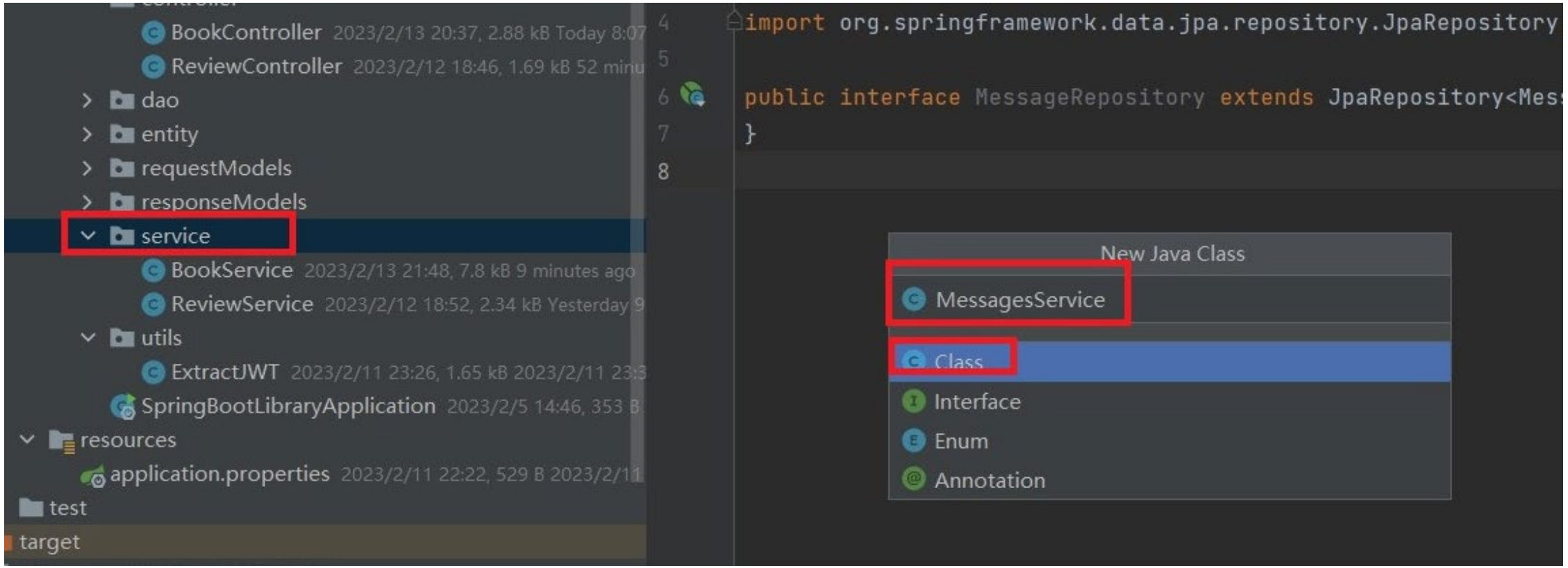


3.SPRINGBOOT MESSAGE SERVICE

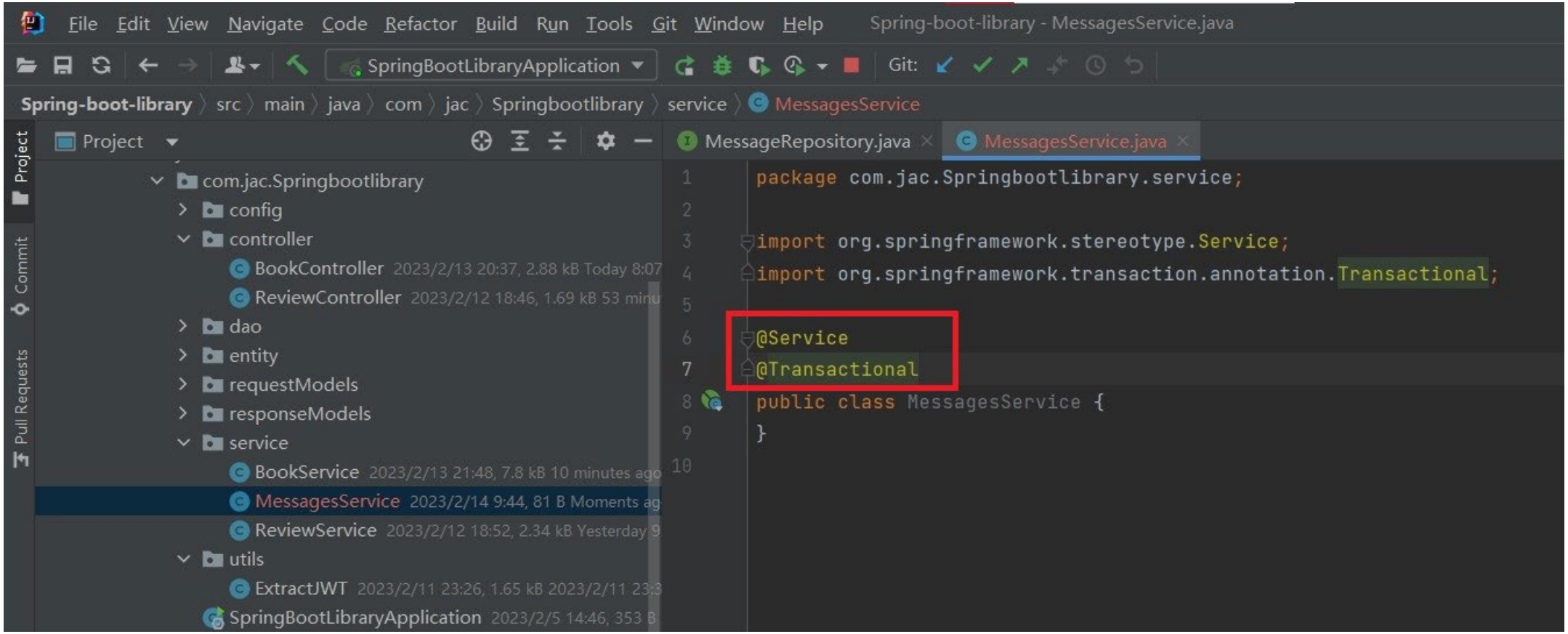
Step 1, create MessagesService class.
Right click service directory, and choose new, Java class.



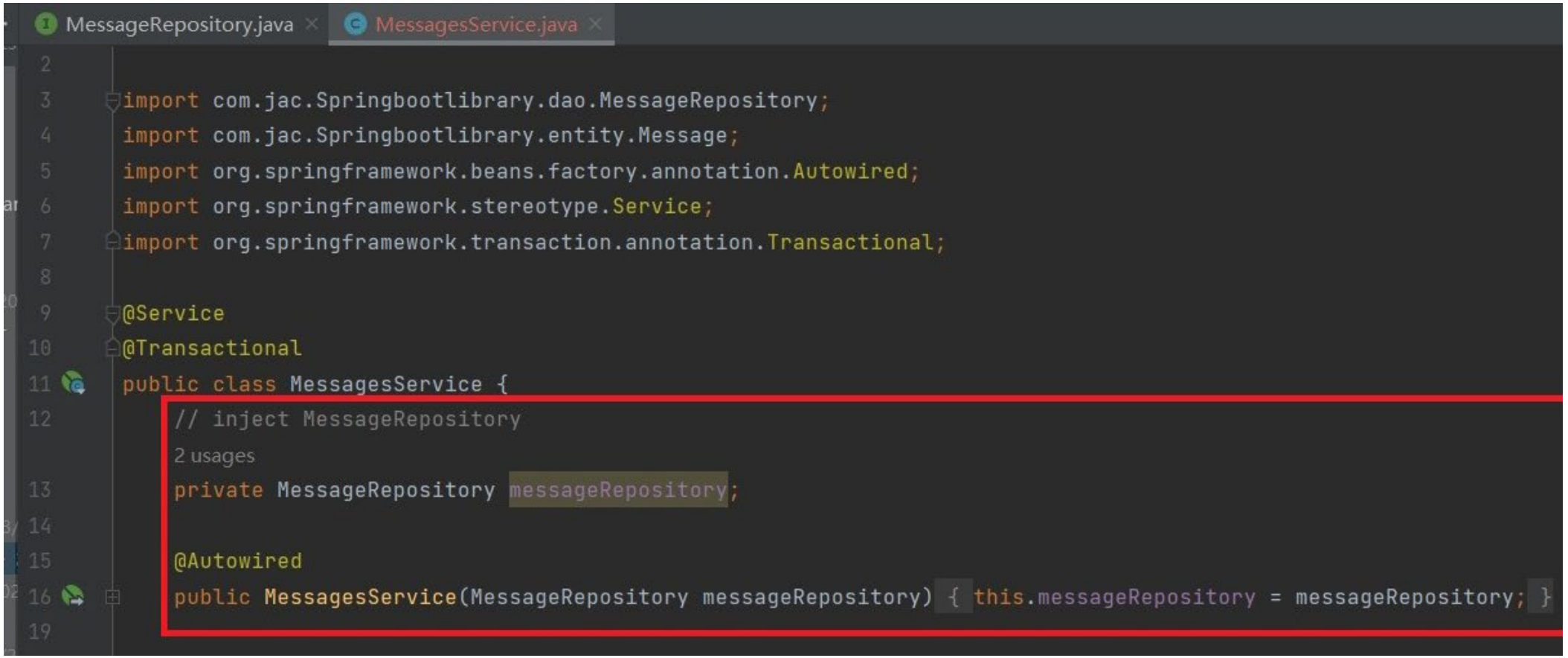
Choose “Class”, input name “MessagesService”.



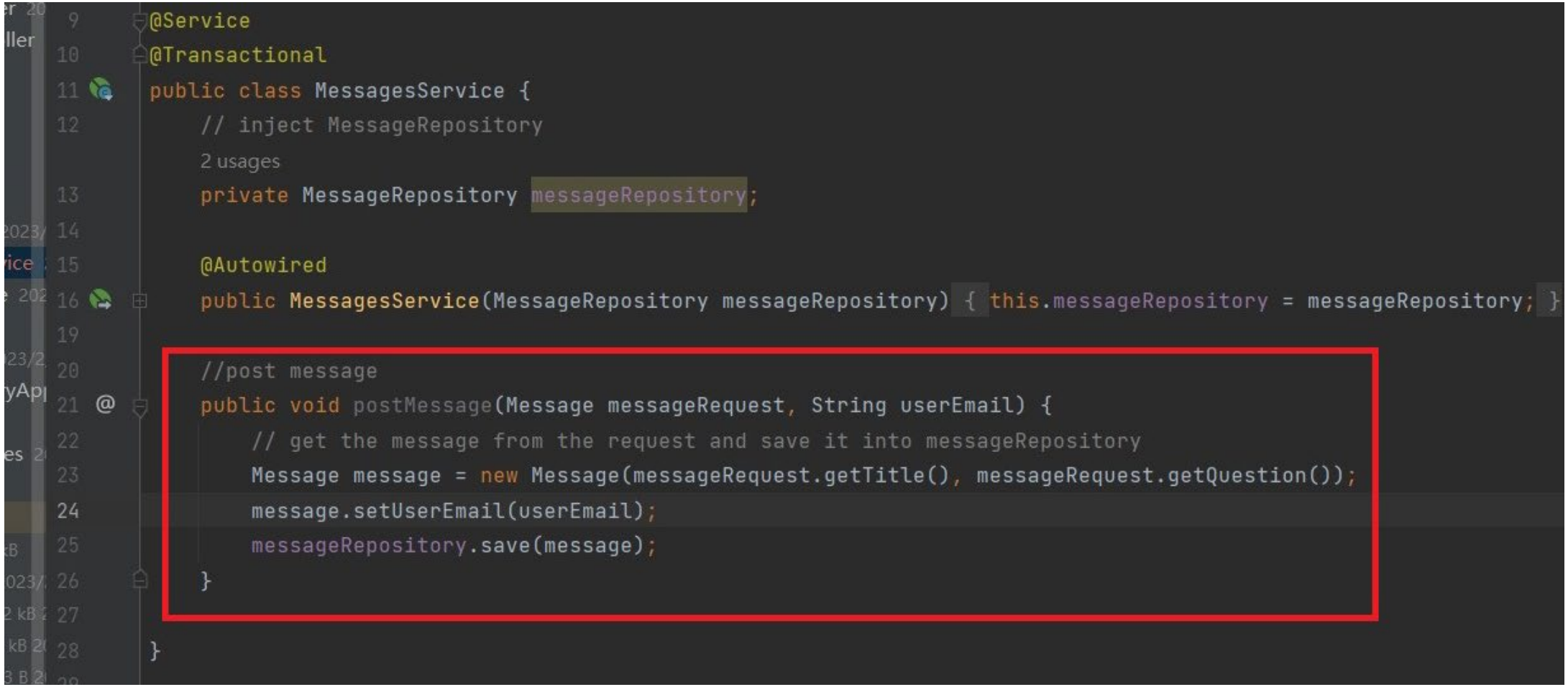
Step 2, add annotation @Service ,@Transactional



Step3　inject MessageRepository and @Autowire it to our constructor.

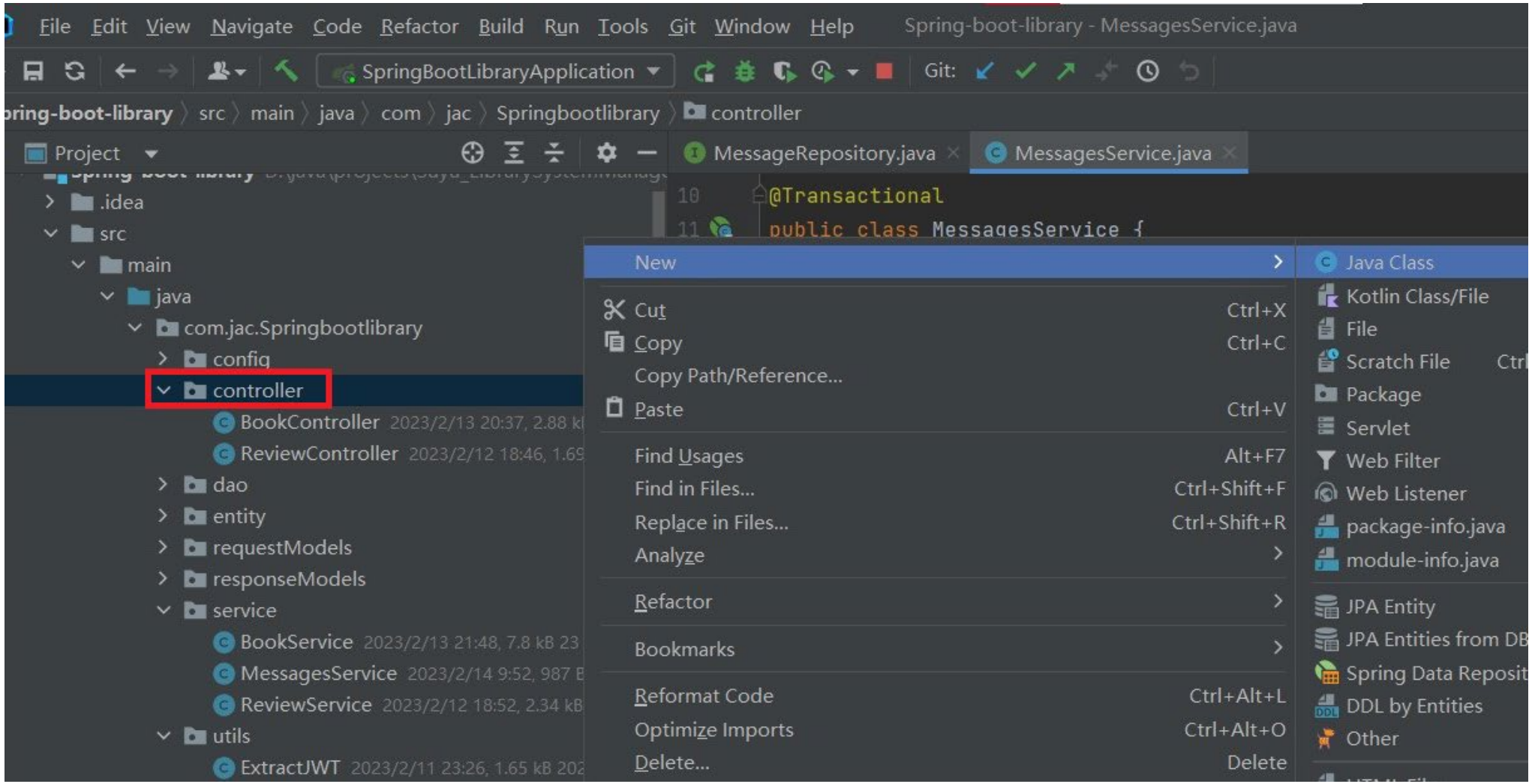


Step 4, create a postMessage(), passing messageRequest and userEmail to it.

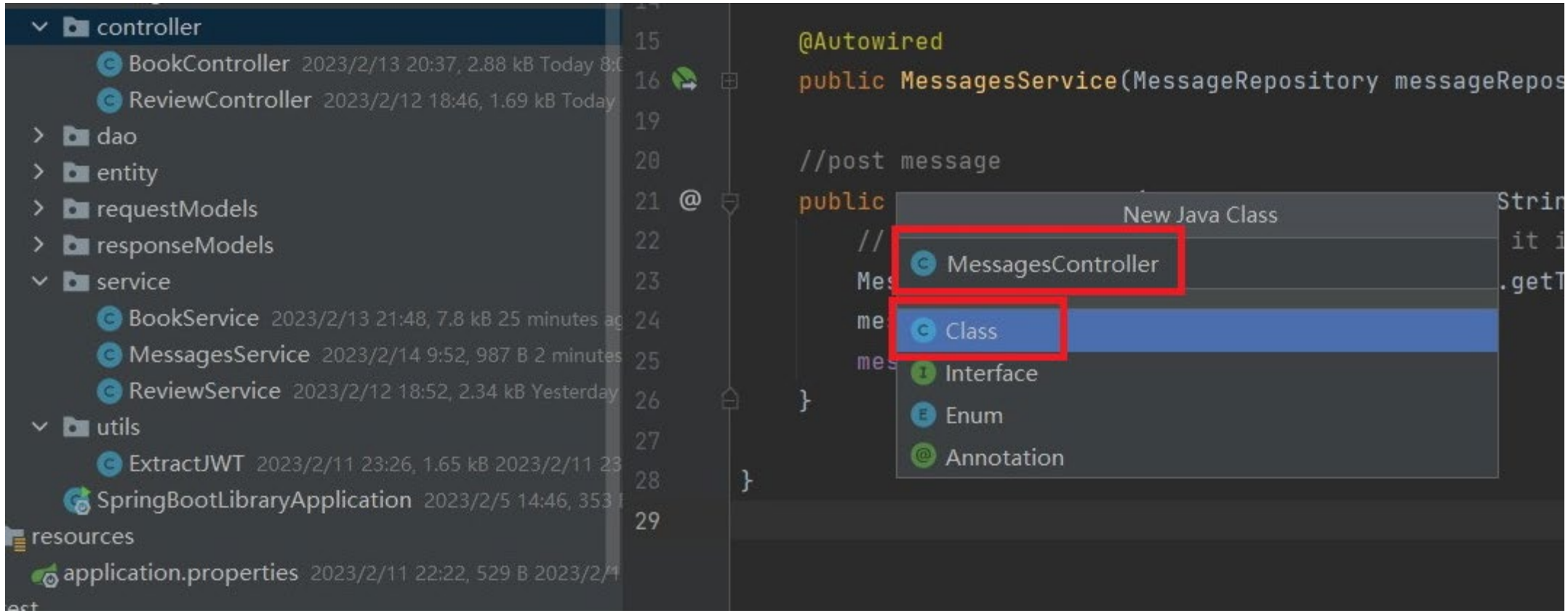


4.SPRINGBOOT MESSAGE CONTROLLER

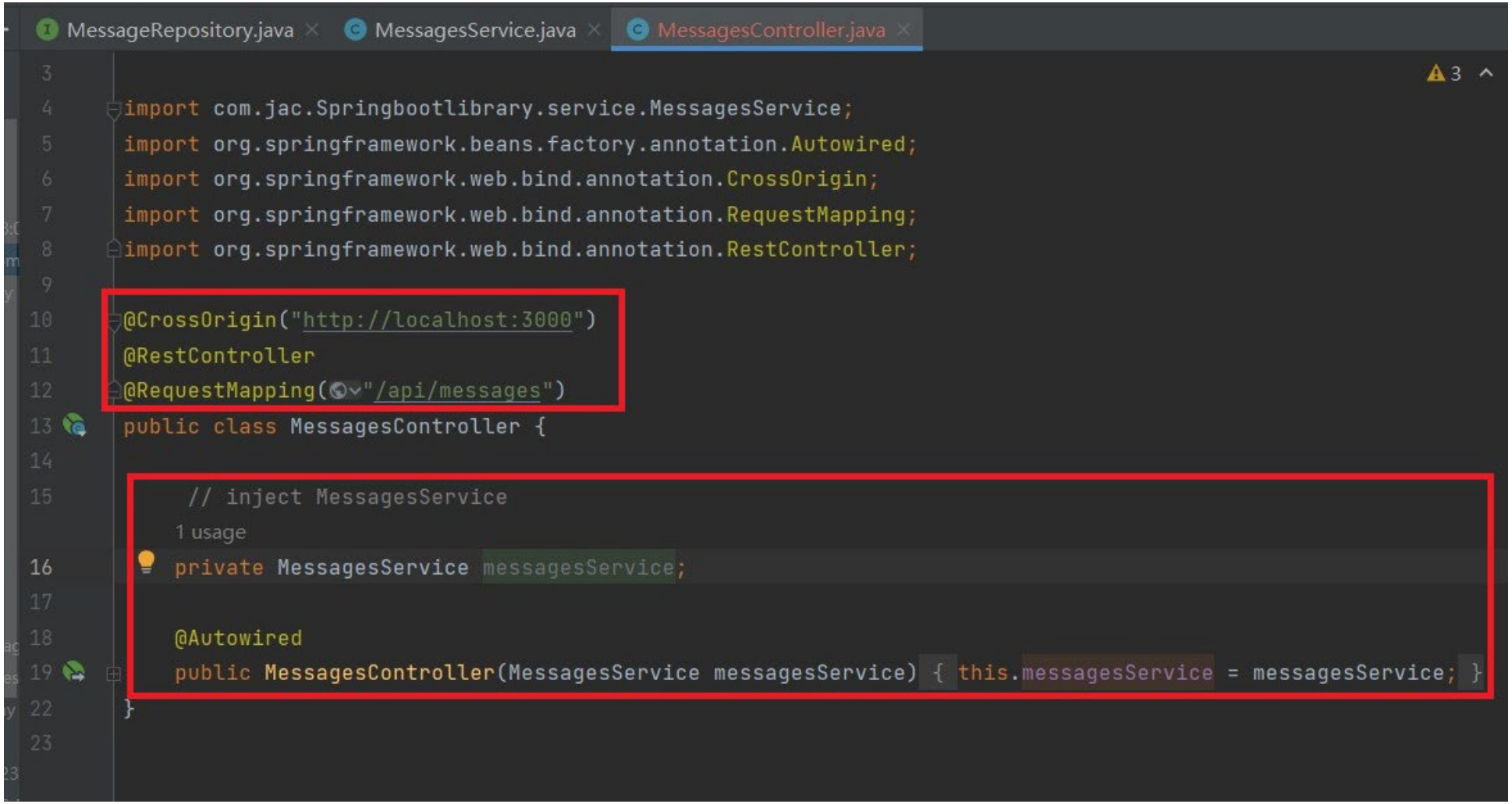
Step 1, Create Message Controller class.
Right click controller directory and choose new , Java class.



Choose “Class”, input name “MessageController”.



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

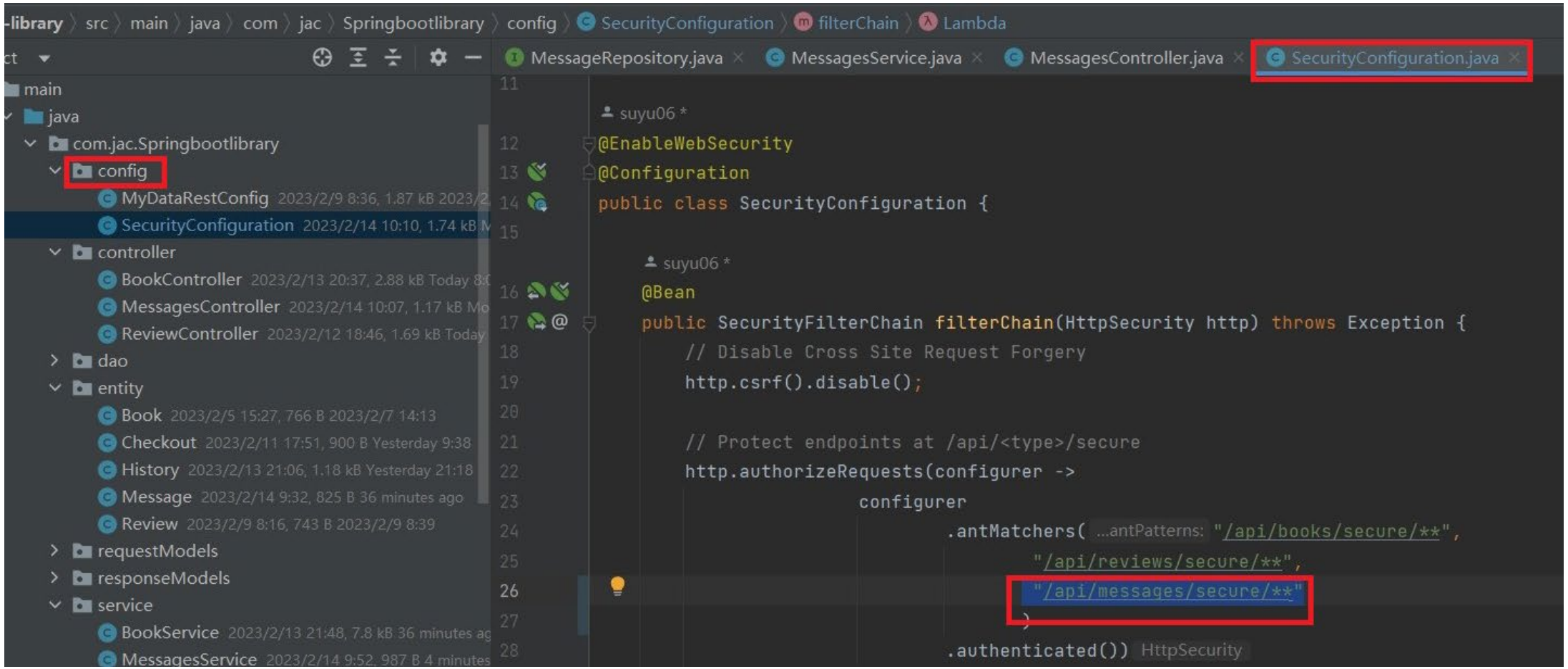


Step 4, create a post message API with post method.

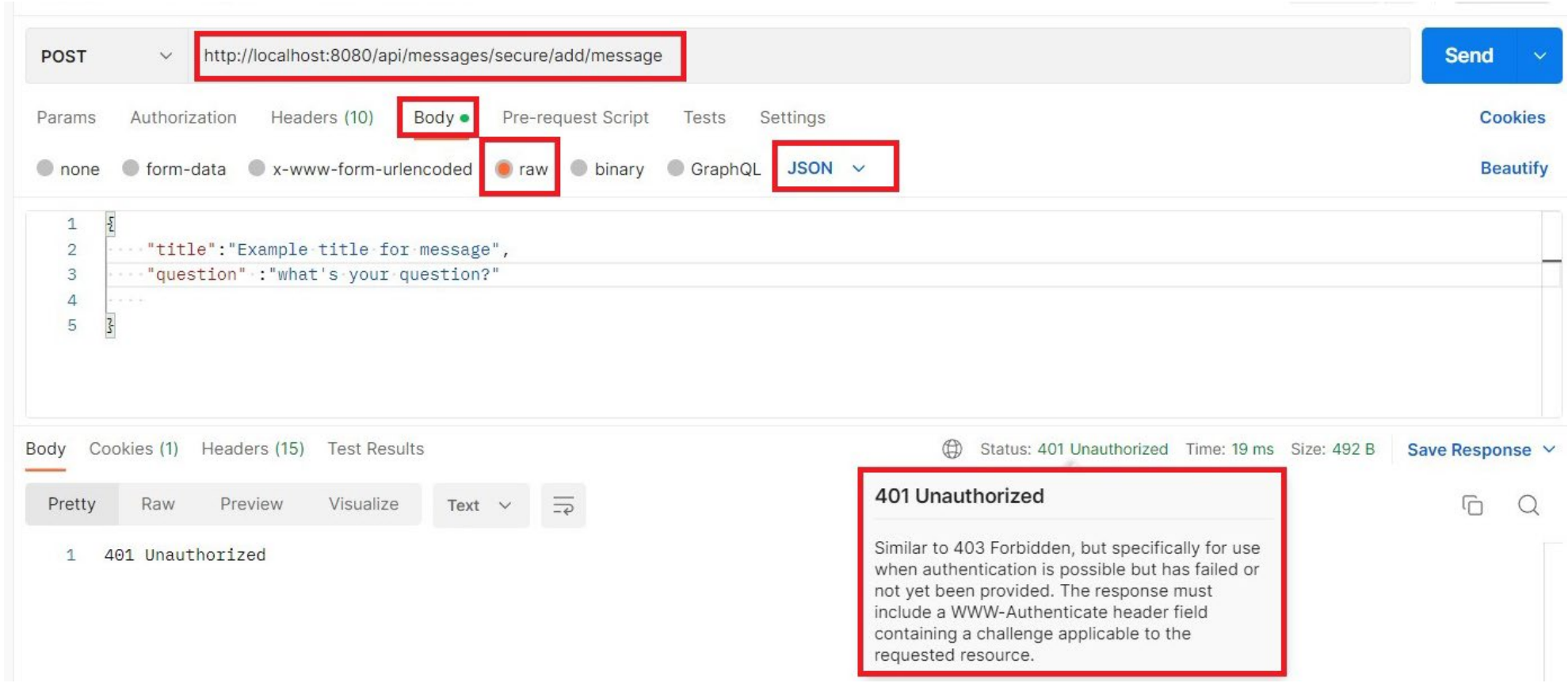
```java
@RequestMapping("/api/messages")
public class MessagesController {

    // 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

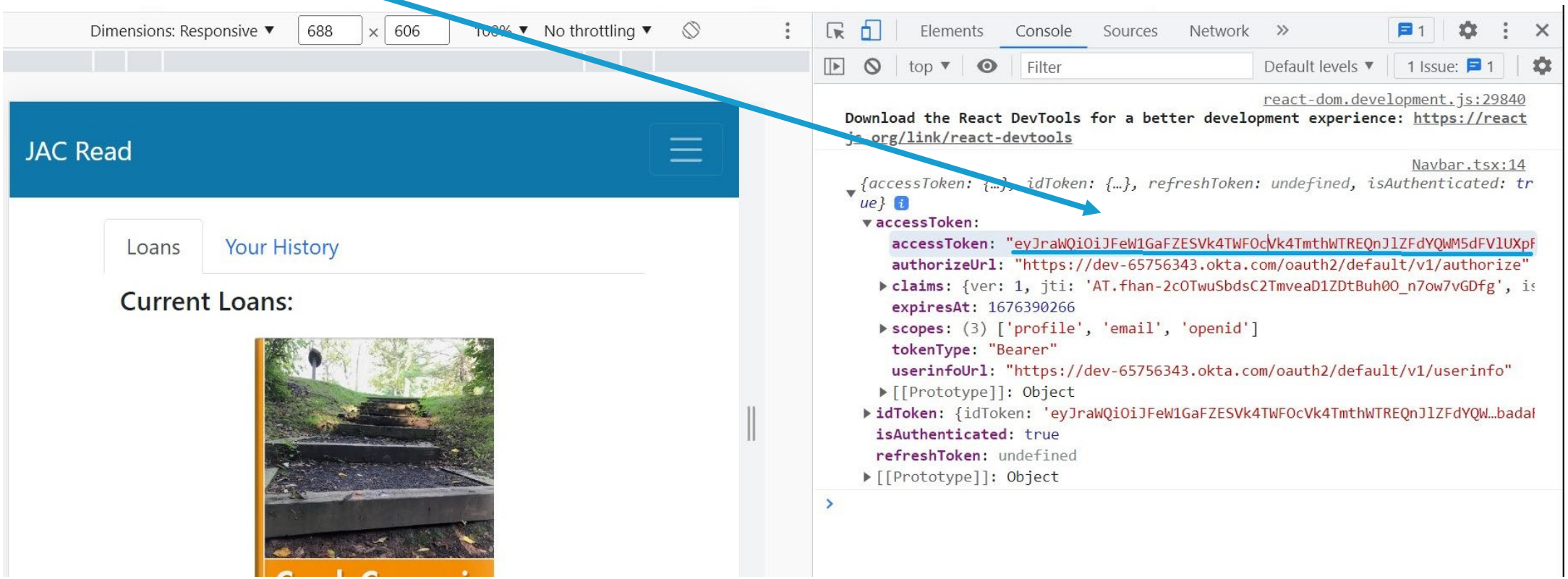
Under “config” package, SecurityConfiguration class, In filterChain(), add “ /api/messages/secure/** ” .
Which means all the API begin with this string , only can be visited by authenticated user.



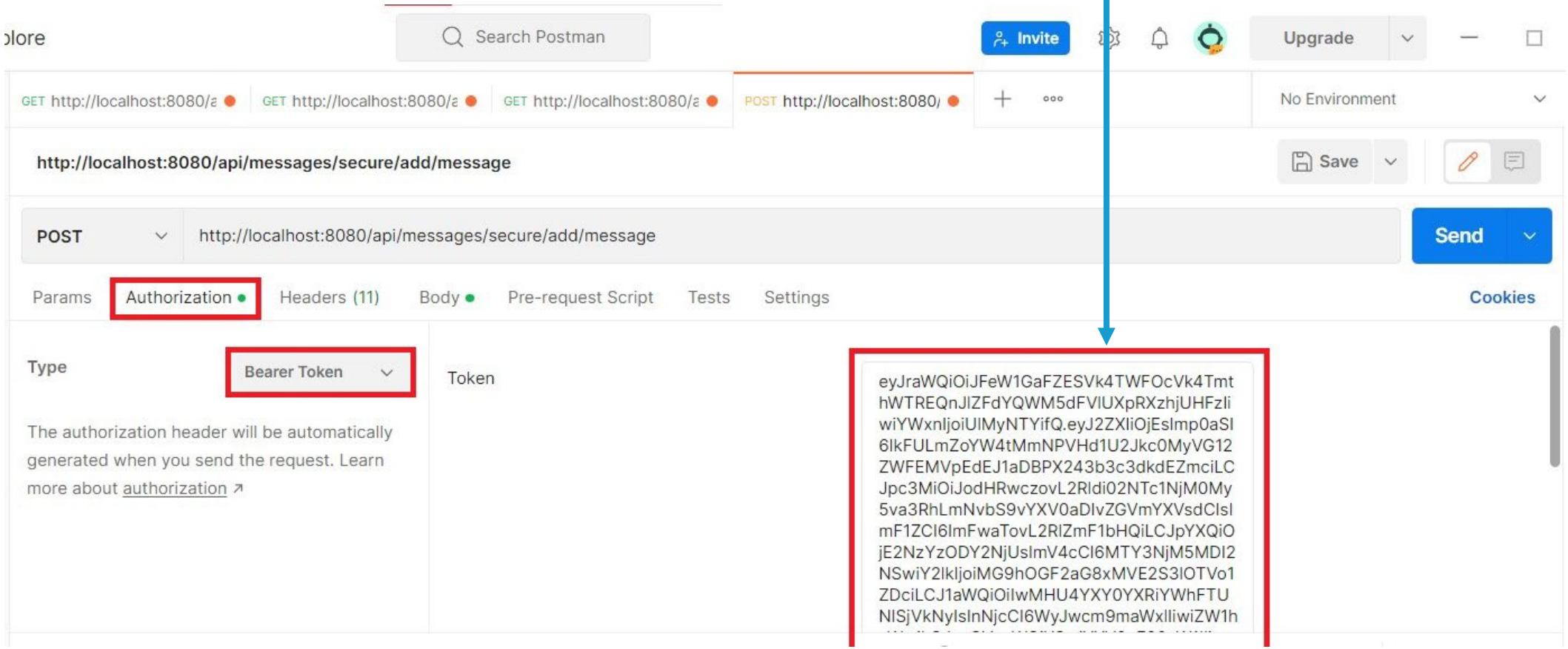
Let’s test in Postman.
We input http://localhost:8080/api/messages/secure/add/message, choose “Post”. Click on “body”, choose “raw”, “JSON”, input the content of title and question, then click on “Send” button. We will get a “401 unauthorized” status. Because the user is not authenticated. We need to have an authorization.



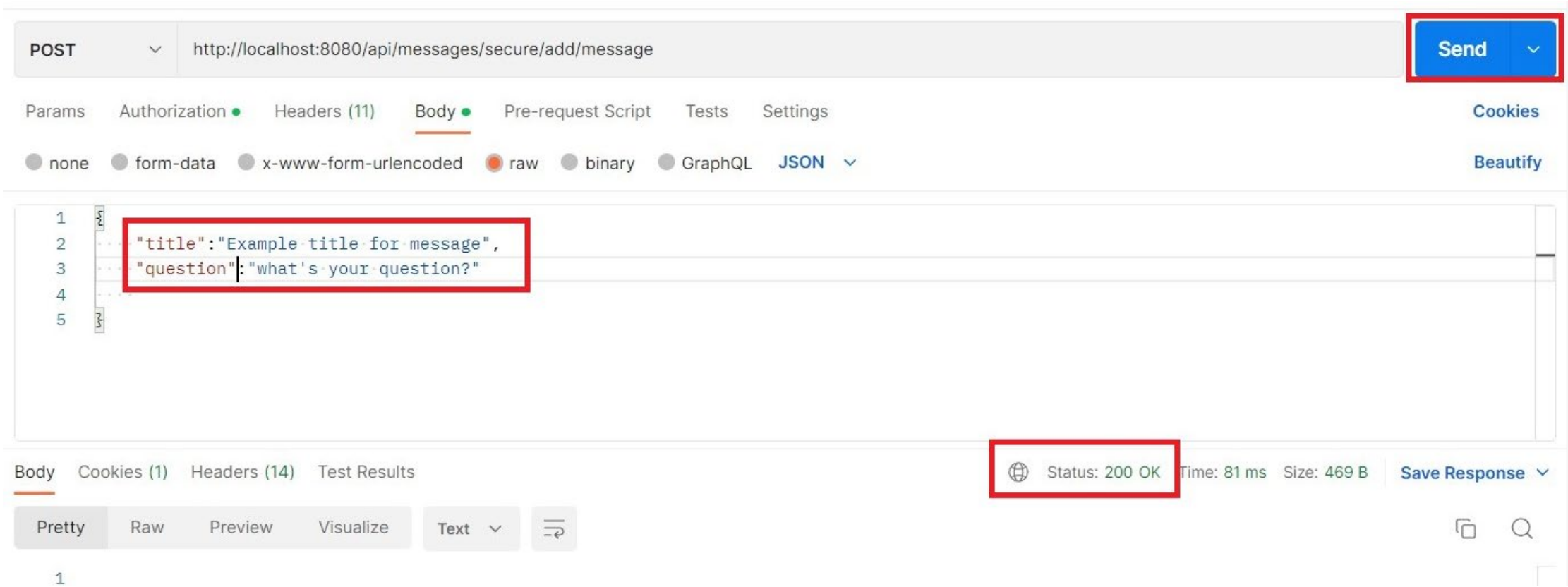
Use email and password log into our website, then press F12, click on the “accessToken”, copy the token string.



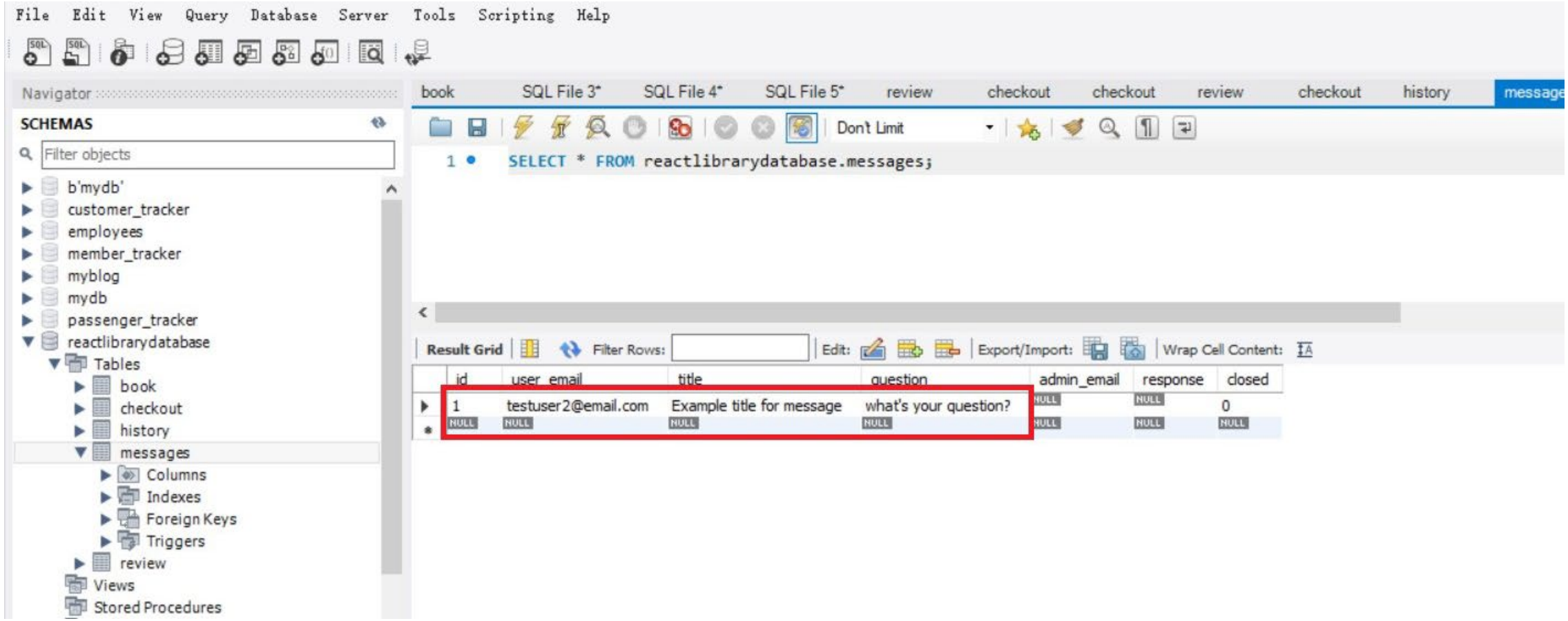
Click on “Authorization”, choose “Bearer Token”, then paste the accessToken string to the token box. Keep the body part the same. Click on “Send” button again.



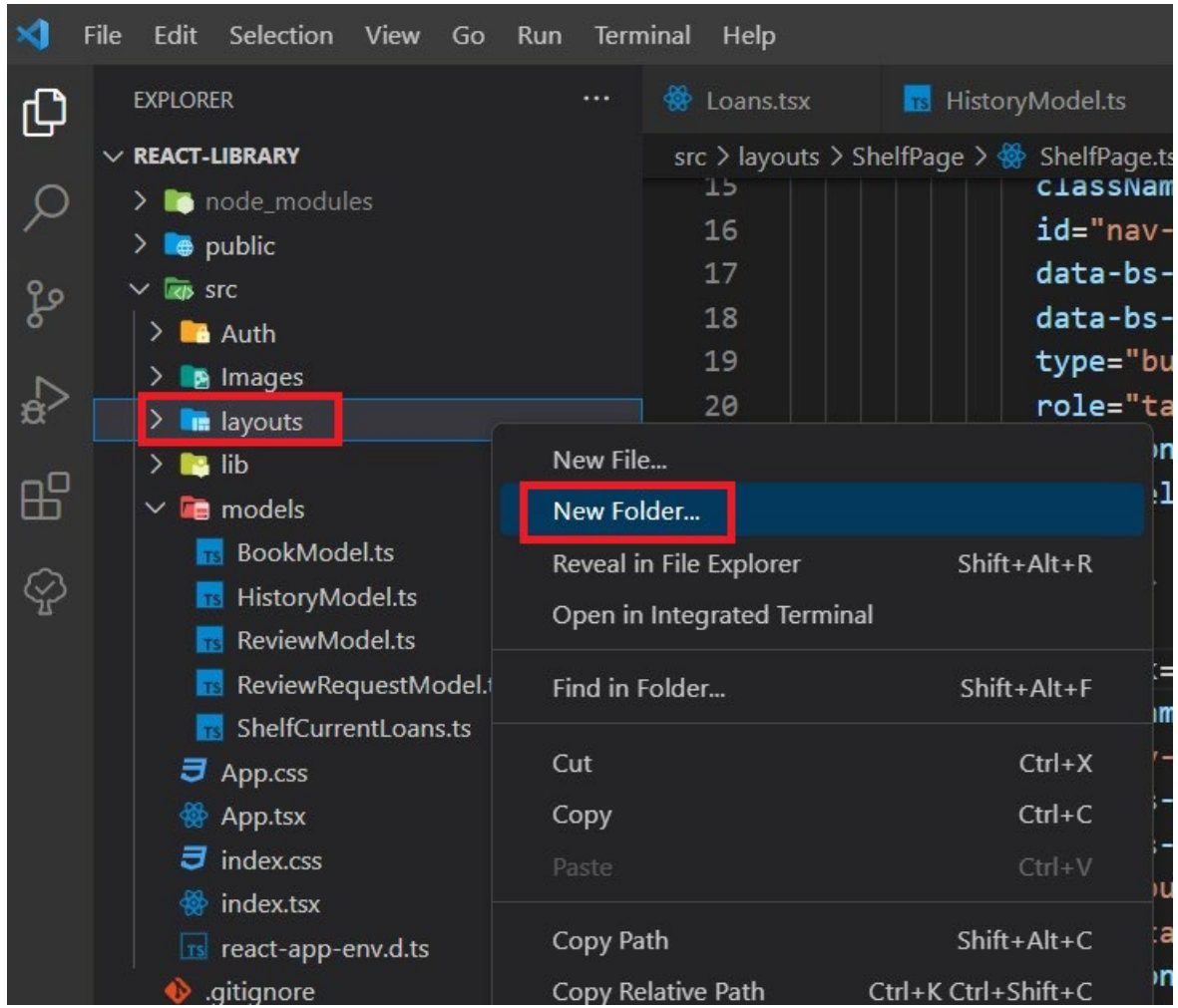
This time, we can see a status of “200 ok”.



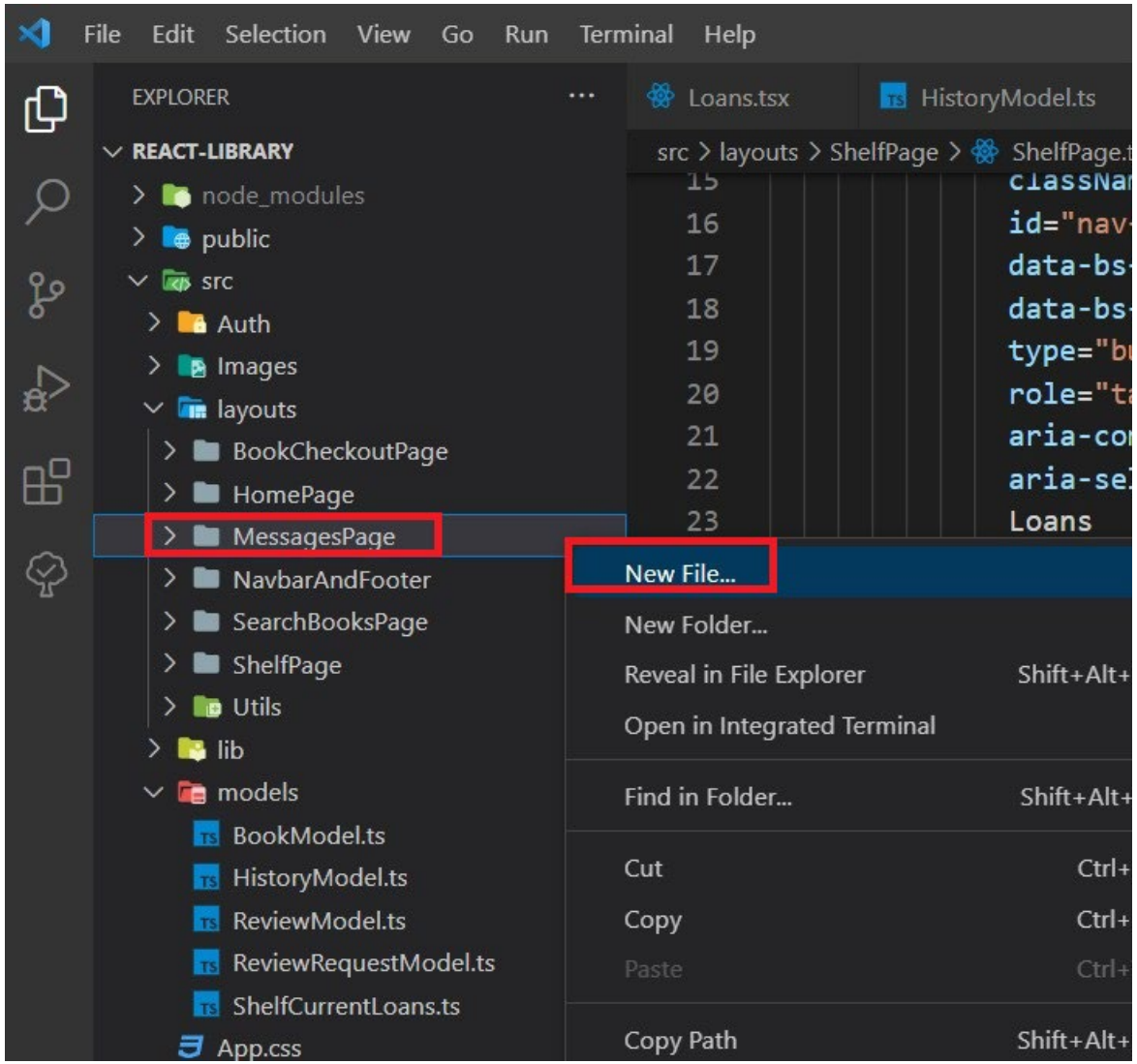
Let’s check in our database. Right click on the table “messages”, choose “select row”, then it will show the whole data of the table.
We can see the message that we have input in the postman has added into this table successfully.



6. REACT MESSAGE COMPONENT

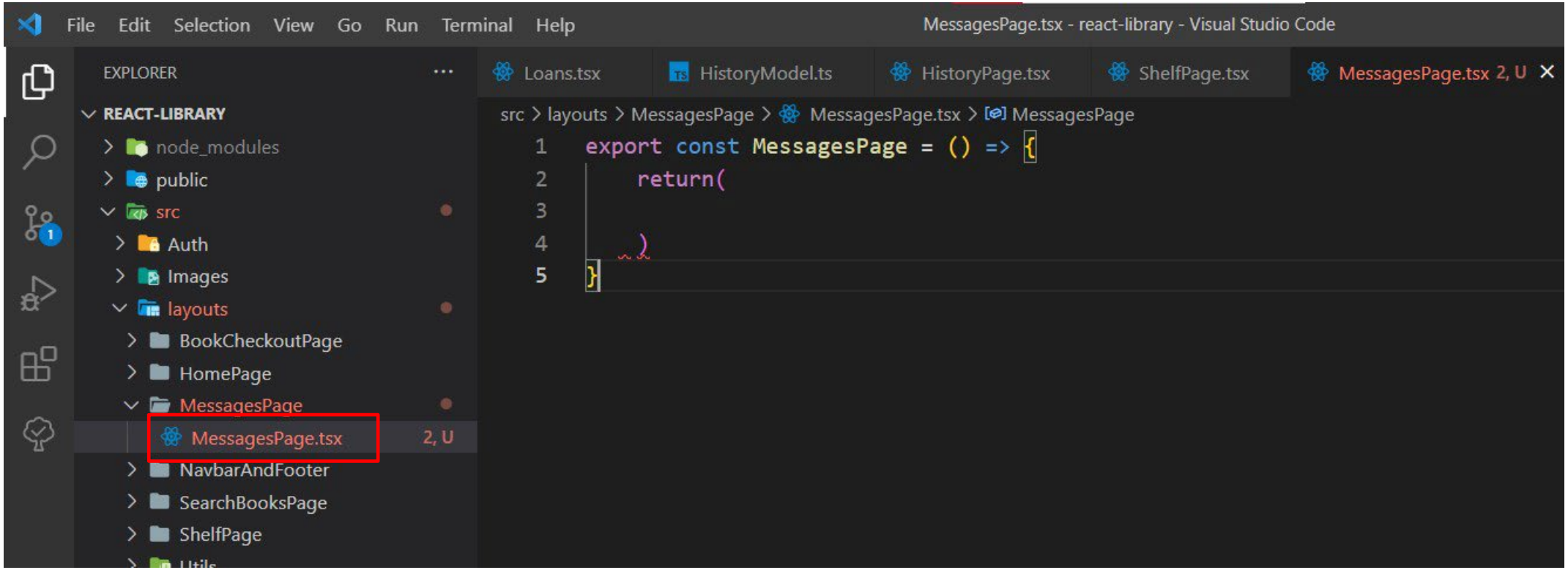


Step 1 , Right click layouts and choose “new folder” , name it “MessagesPage”.

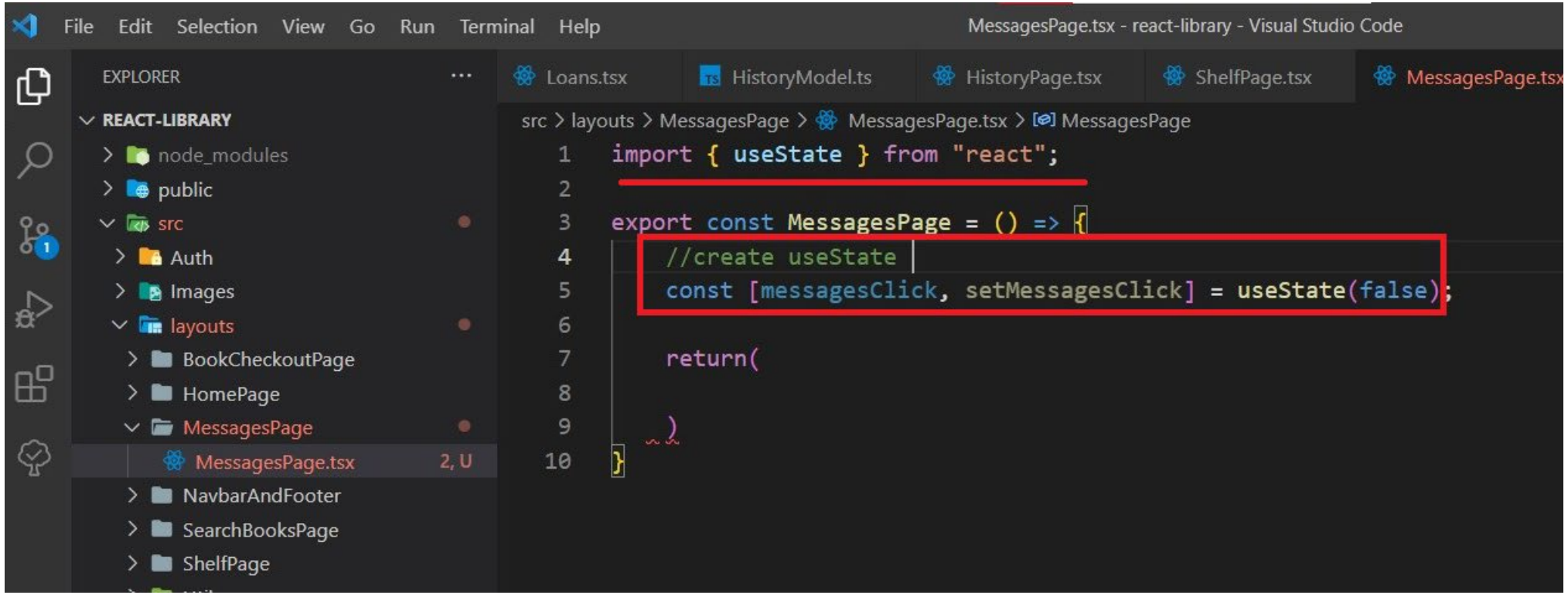


Step 2, In this folder, create a new file, MessagesPage.tsx.

Step 3 create export const MessagesPage () structure.

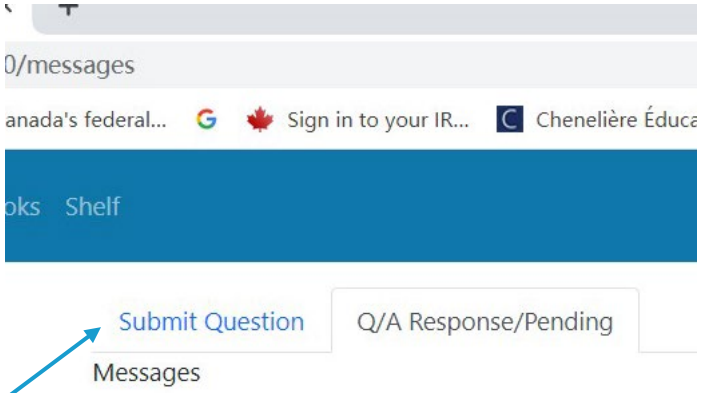


Step 4 , create a useState array messageClick, setMesssageClick. If you see a red tilde under the useState, import it from “React”.



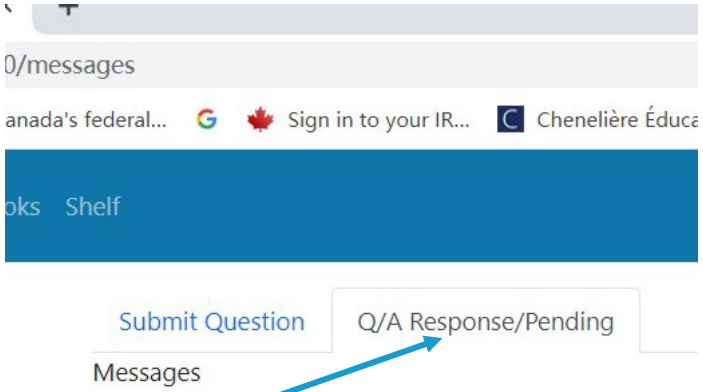
Step 5, fill in return part with Html and CSS code.
First , create a submit question tab.

```
// 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">
                    {/* subimit 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>
```



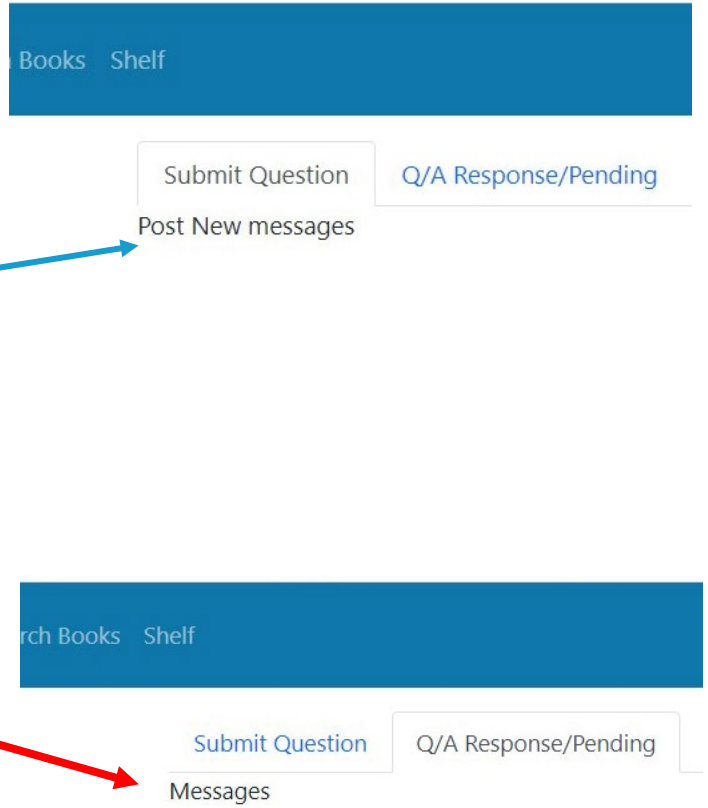
Second, response tab.

```
            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>
```

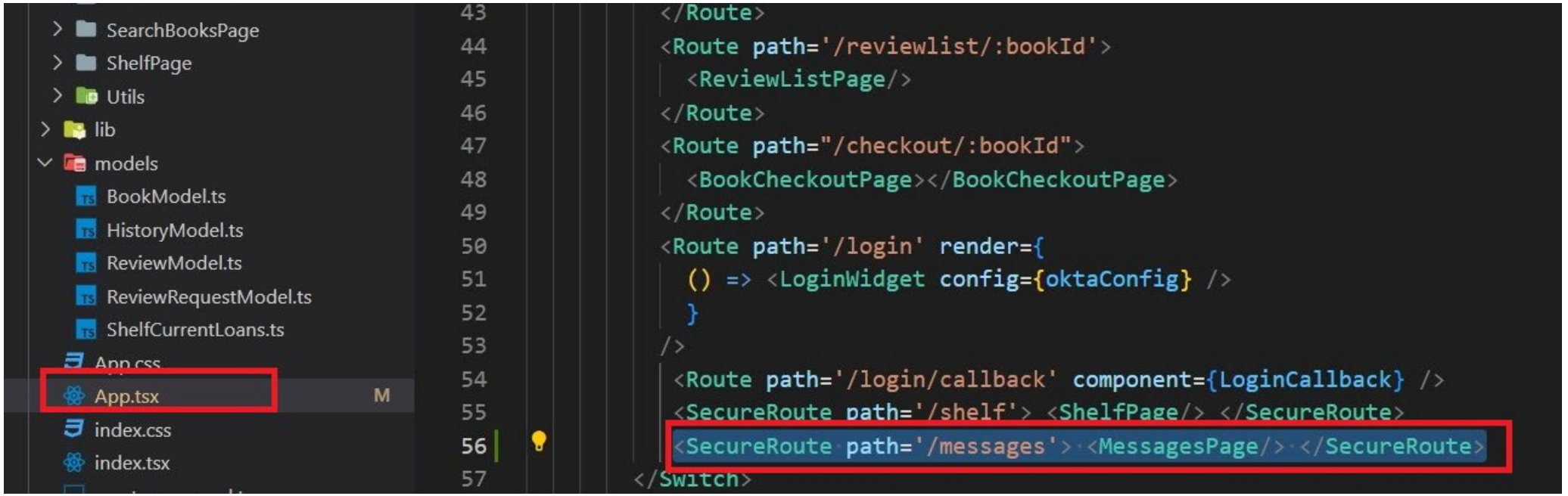


Then the words under the tab

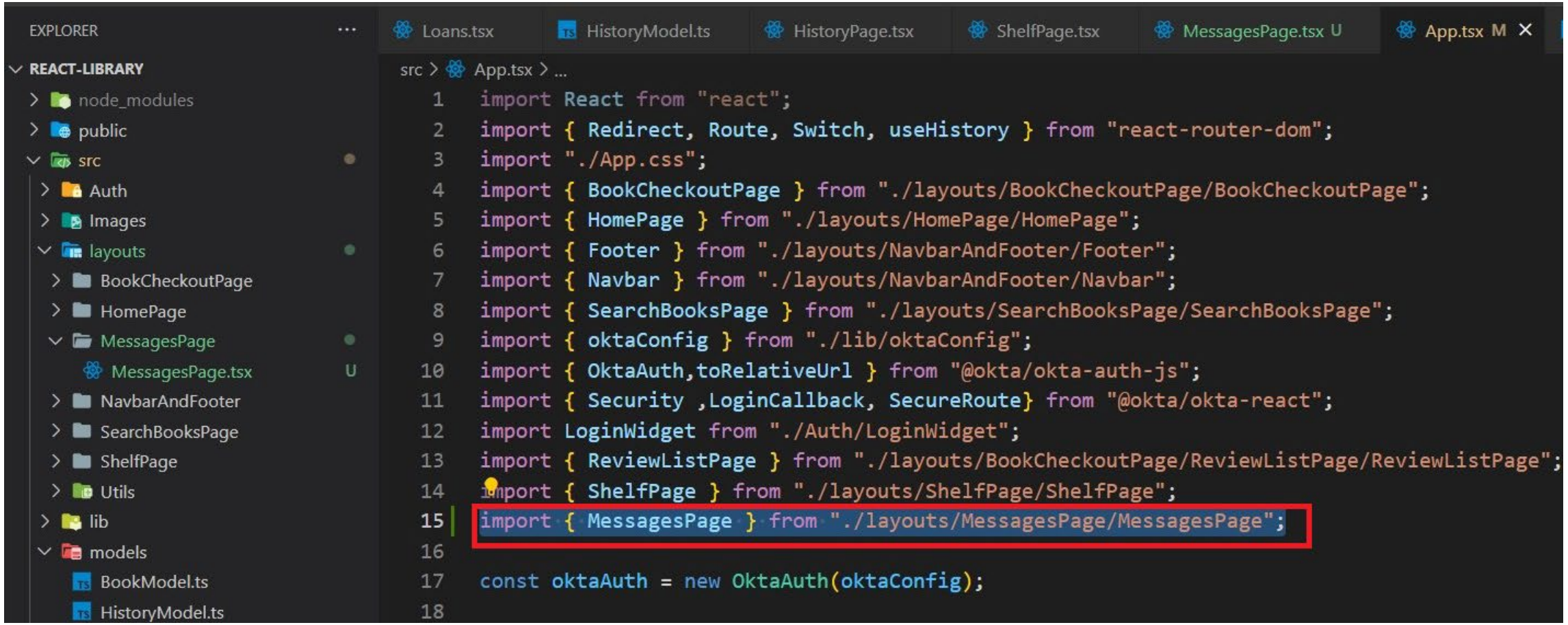
```
            Q/A Response/Pending
        </button>
    </div>
</nav>
<div className="tab-content" id="nav-tabContent">
    <div
        className="tab-pane fade show active"
        id="nav-send-message"
        role="tabpanel"
        aria-labelledby="nav-send-message-tab">
        <p>Post New messages</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>
```



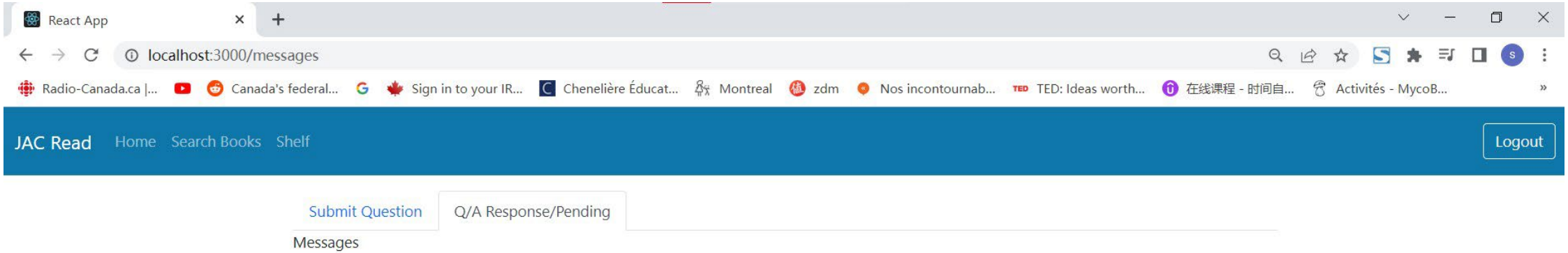
Step 6, Go to App.tsx, add Secure Route for “MessagesPage”, and import MessagesPage.



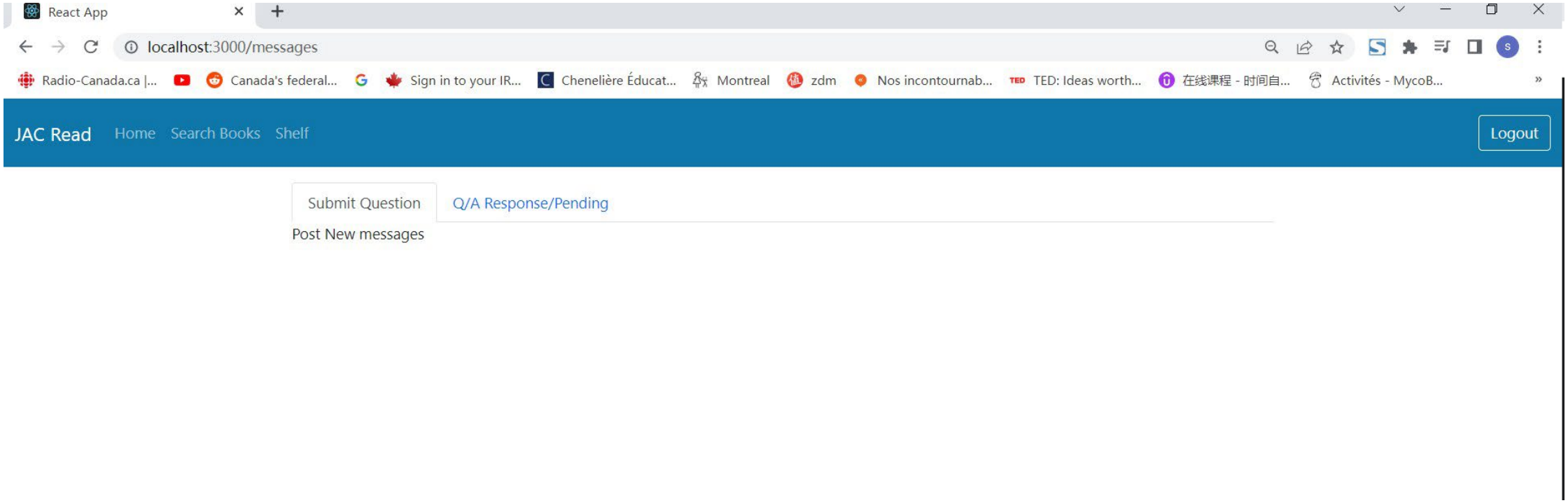
Check the top of the page, if the MessagePage is imported.



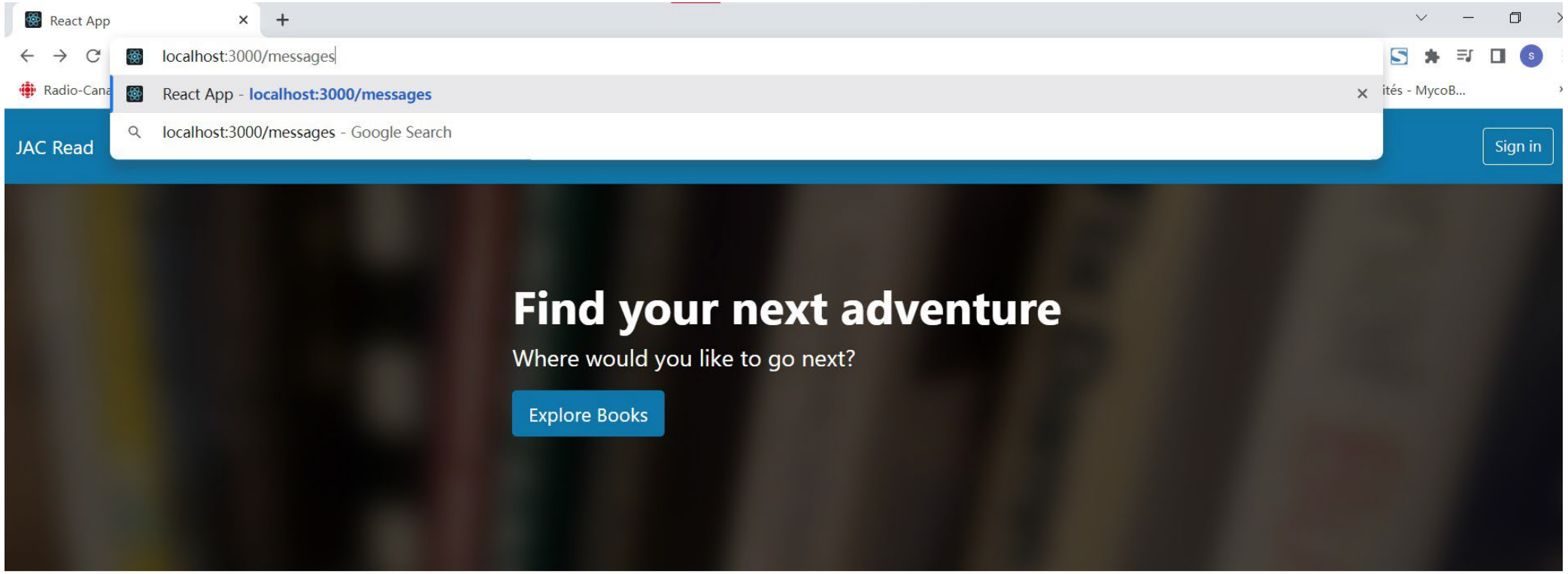
Now, log in the account with the email and password.
if successfully ,the sign in button will change into logout.
then visit http://localhost:3000/messages, click on “Q/A Response/Pending” , it will show:



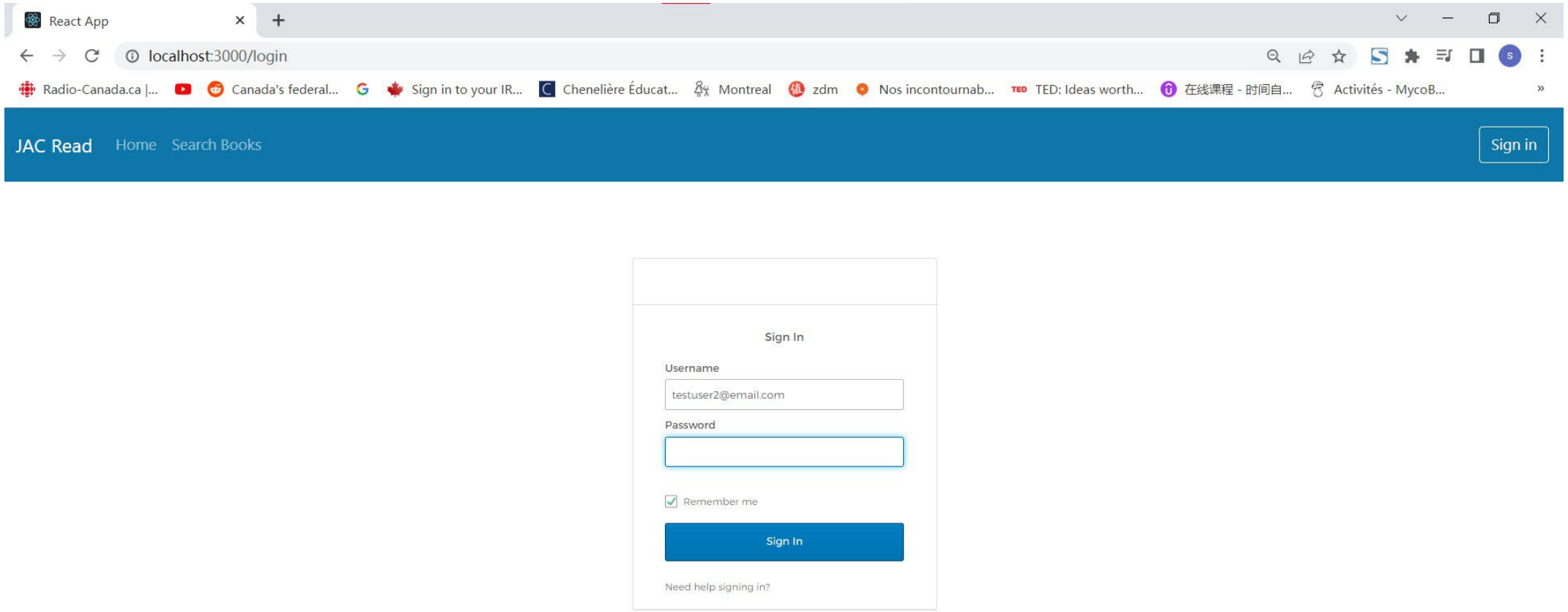
Click on “Submit Question” , it will show:



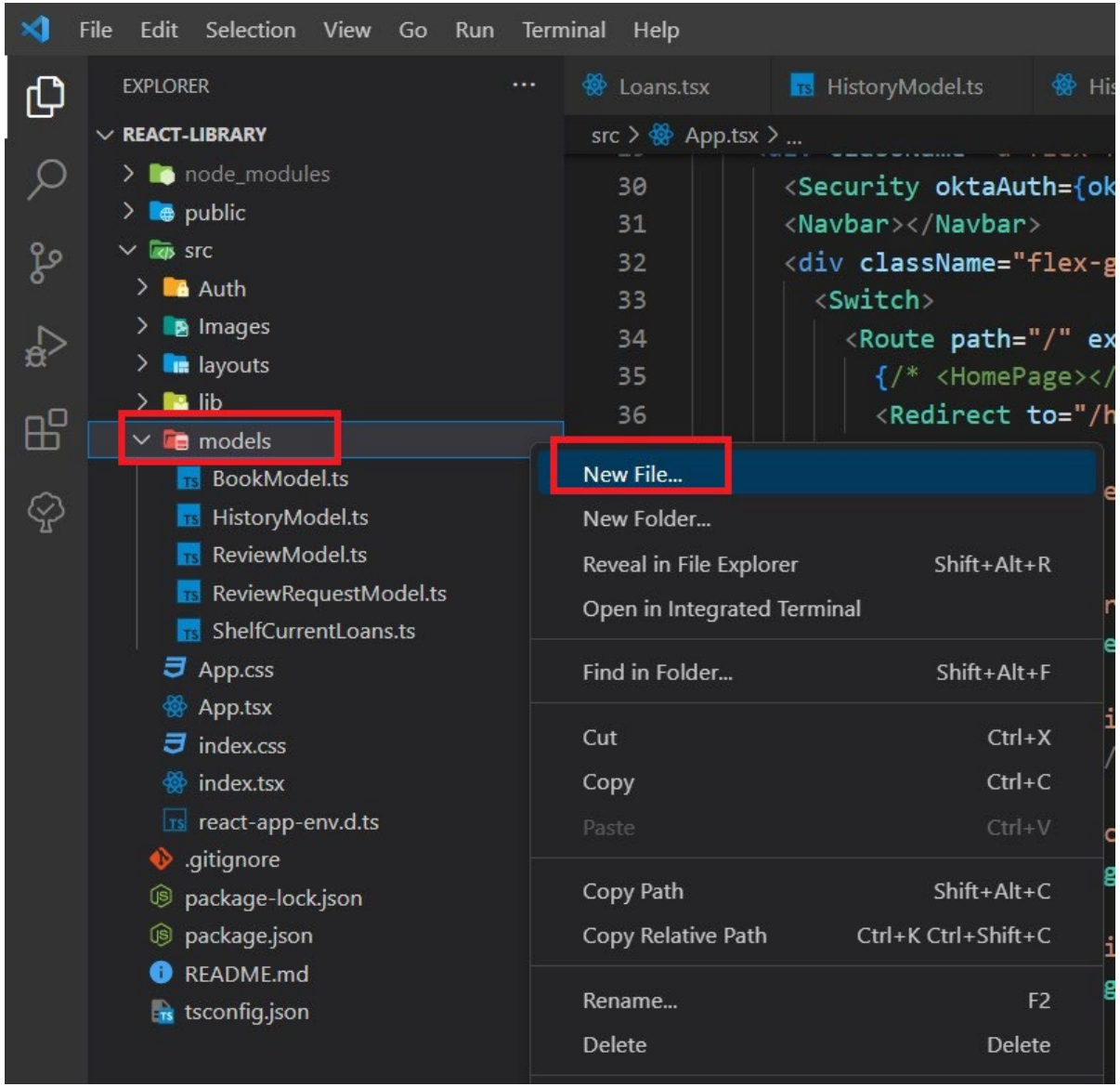
Then we log out , type http://localhost:3000/messages



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

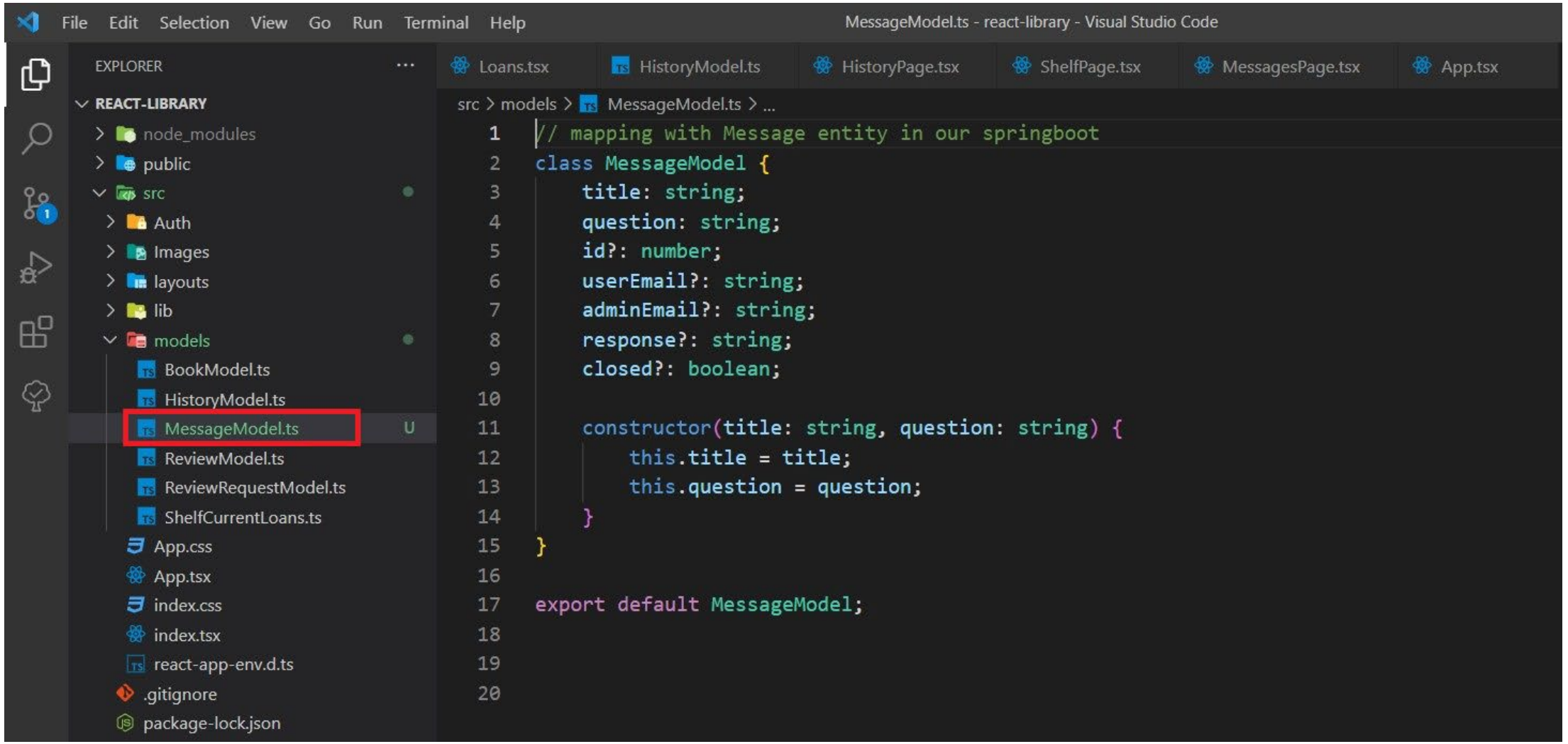


7.REACT MESSAGE MODEL

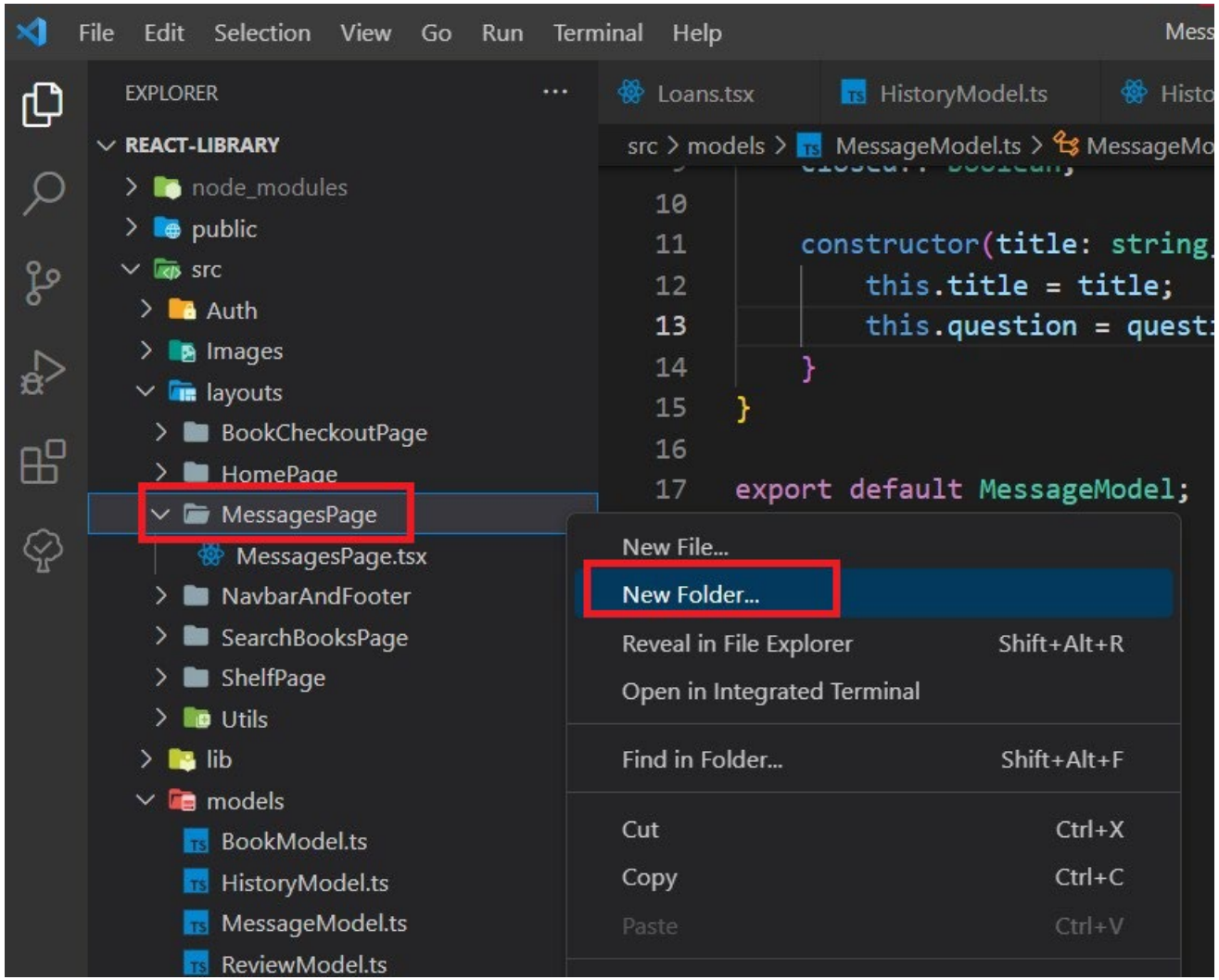


Step 1, right click models and choose new file, name it MessageModel.ts.

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

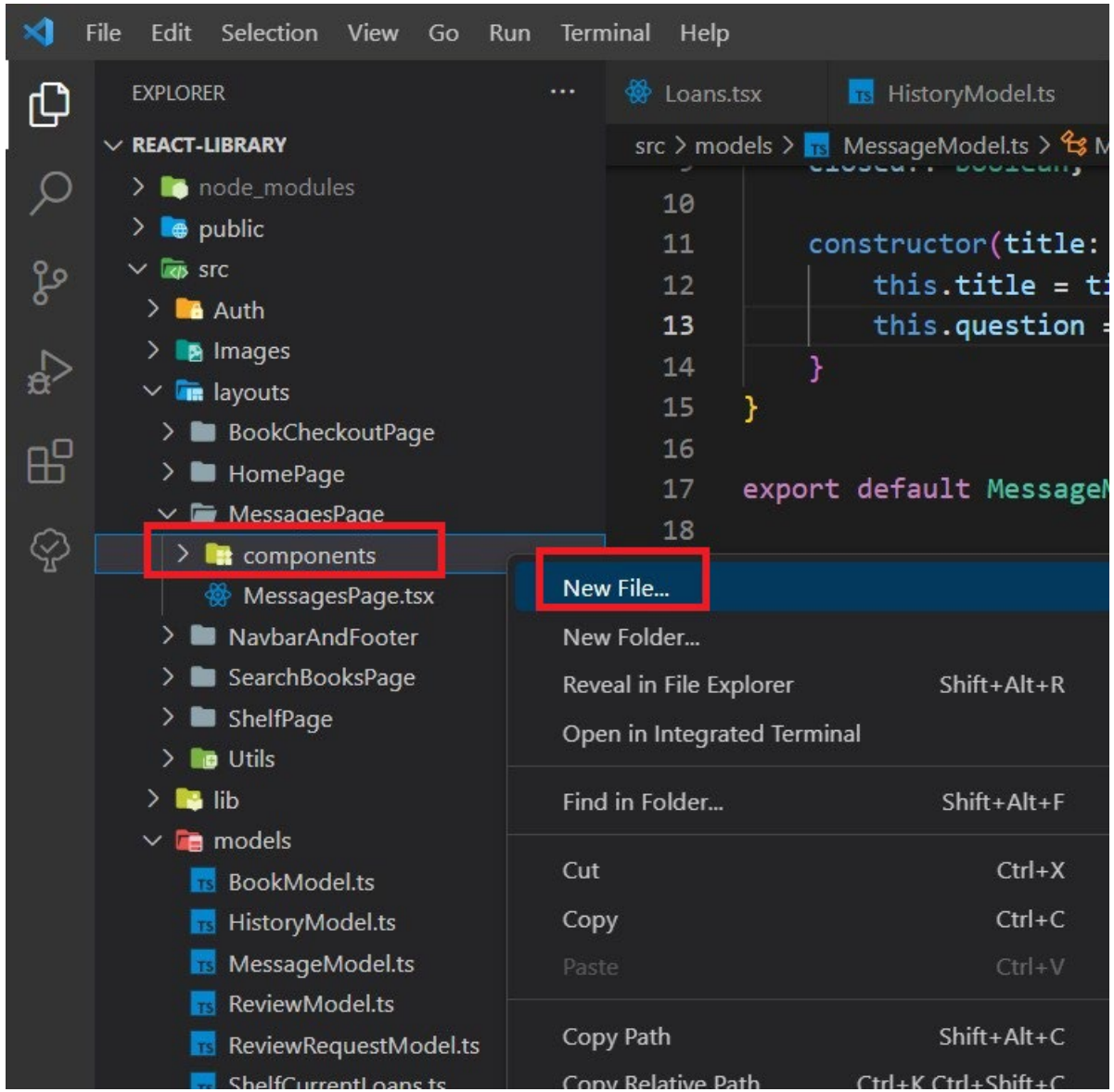


8. REACT POST A QUESTION COMPONENT

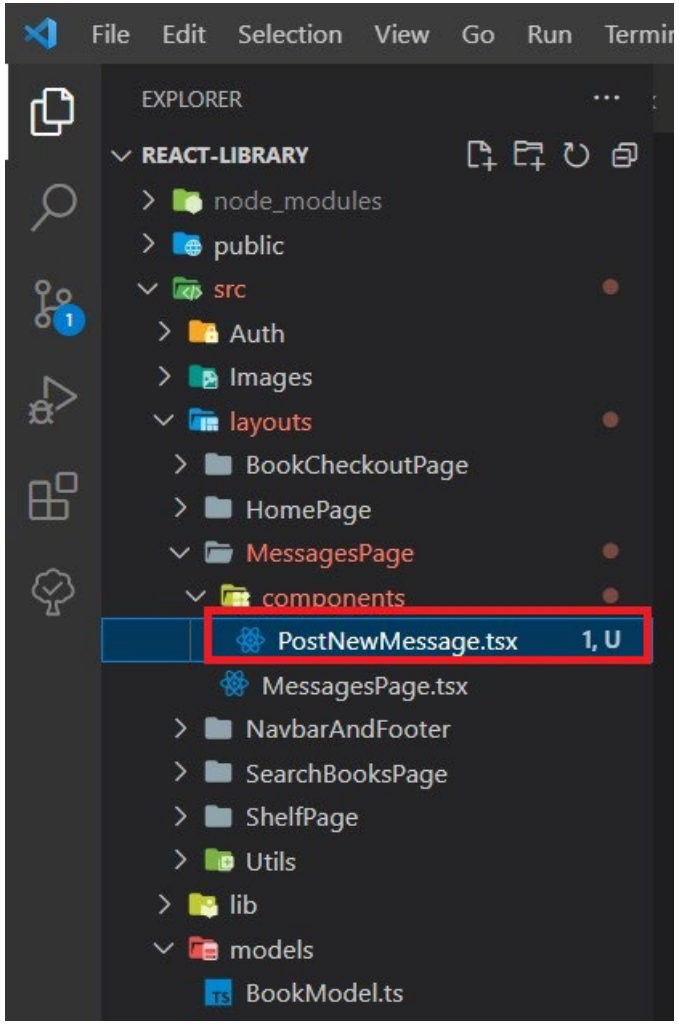


Here, we will create new form where we can post new messages to our admins into our database.

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

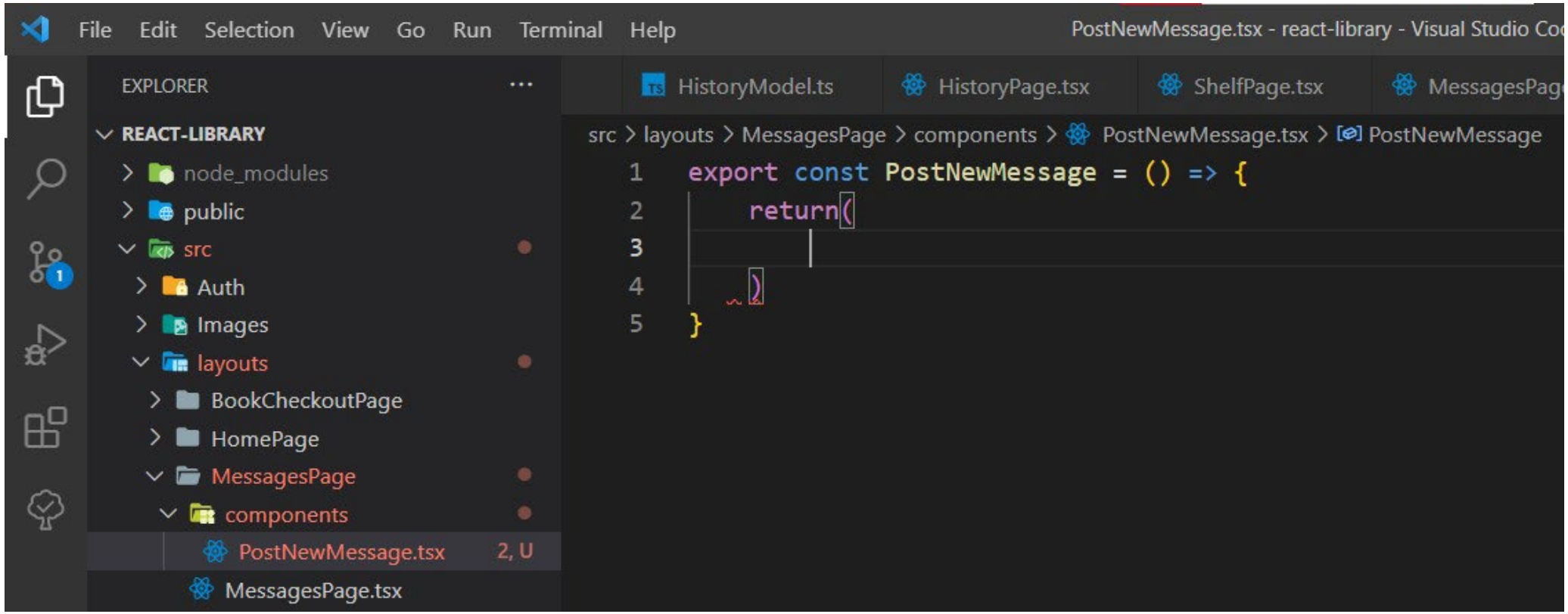


Step 2, Inside components ,
create a new file, name it PostNewMessage.tsx.

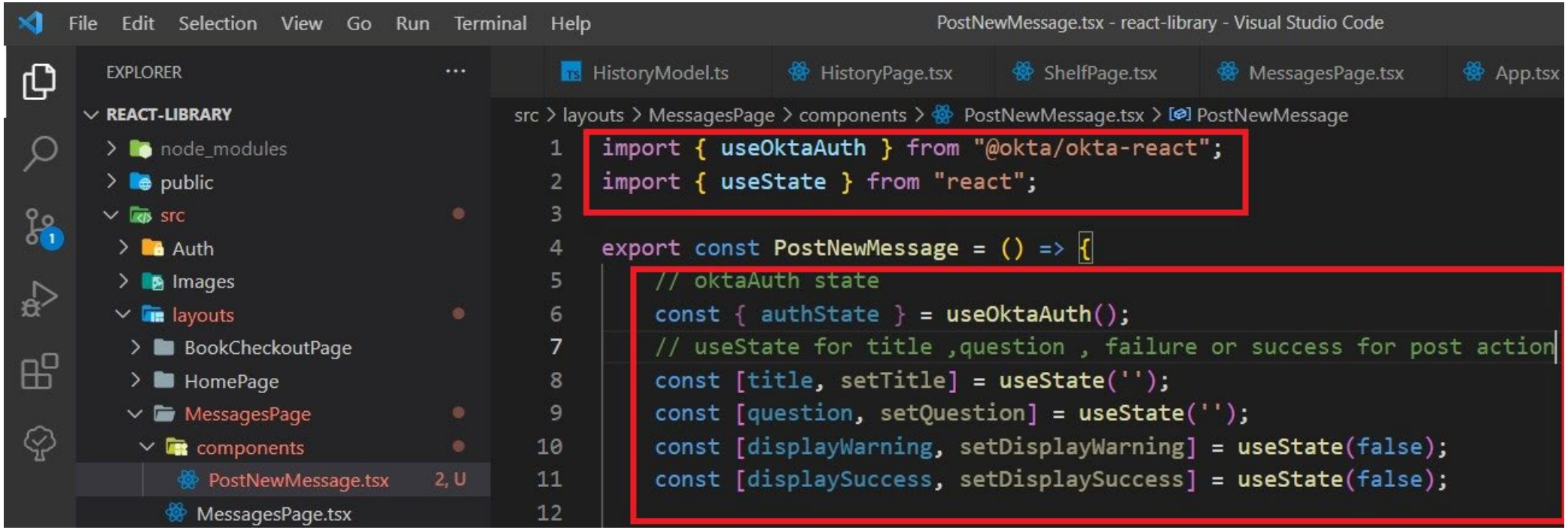


New file created.

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



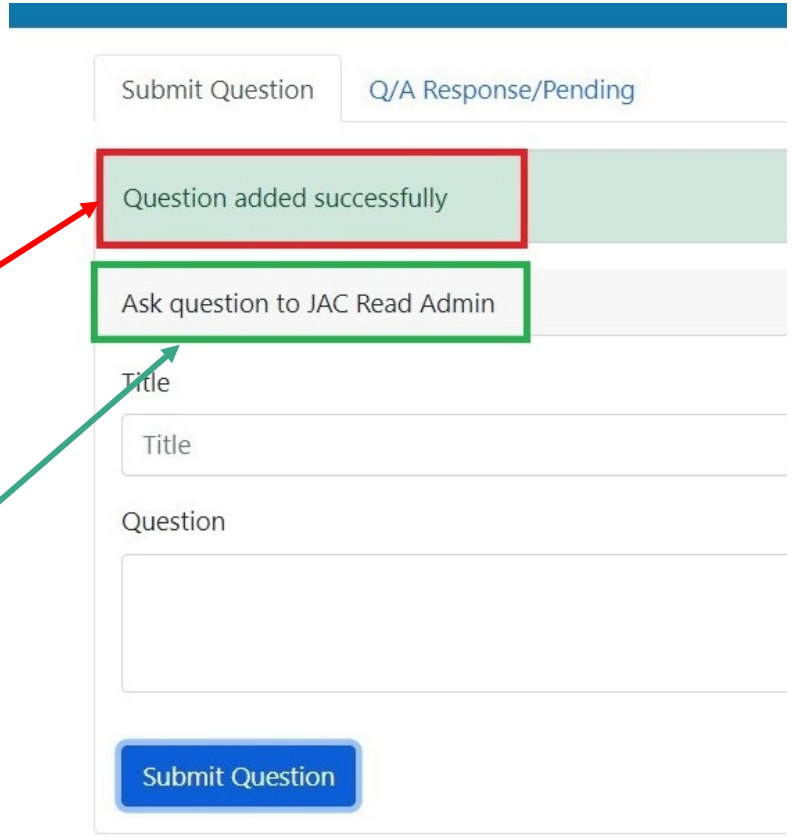
Step 4, create useState and import okta and react useState.



Stpe 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>
```



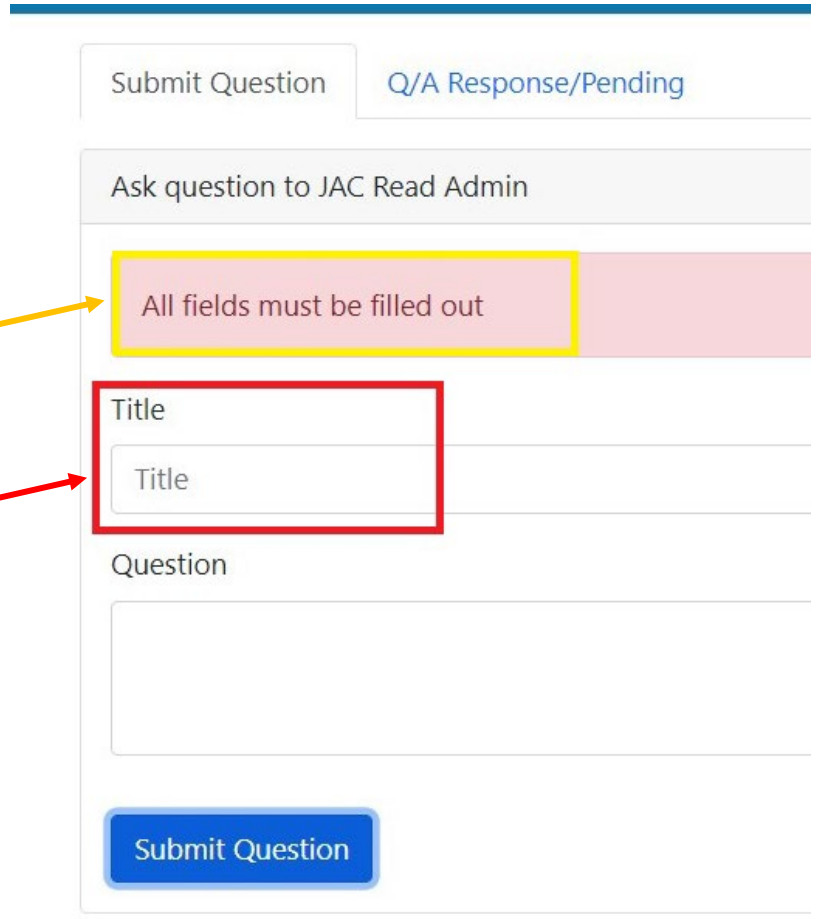
Then , In card body, an alert message, a label and an input box.

PostNewMessage.tsx - react-library - Visual Studio Code

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

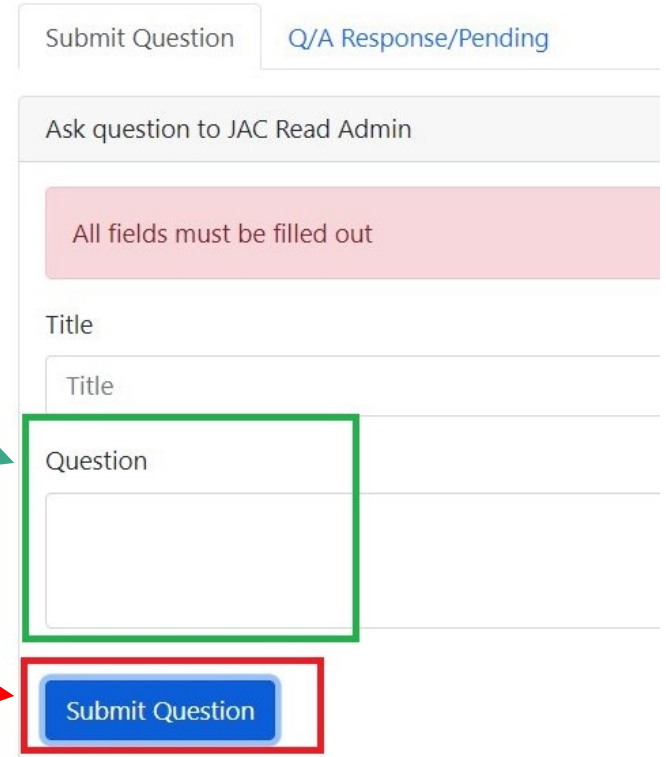
MessagesPage > components > PostNewMessage.tsx > PostNewMessage

```
{/* card header */}
<div className="card-header">Ask question to JAC Read Admin</div>
{/* card boby */}
<div className="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>
```

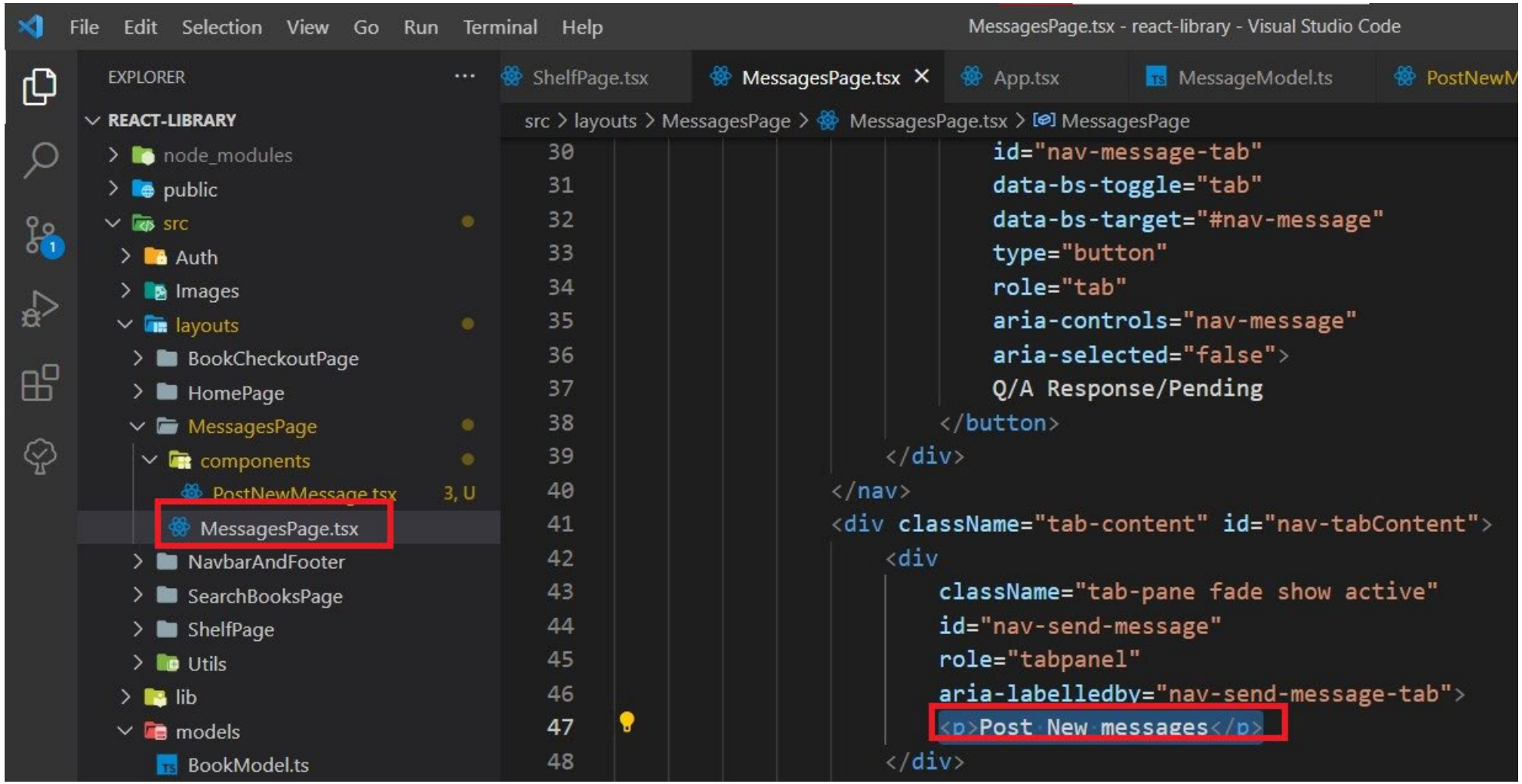


Last, question label and text area, a submit button.

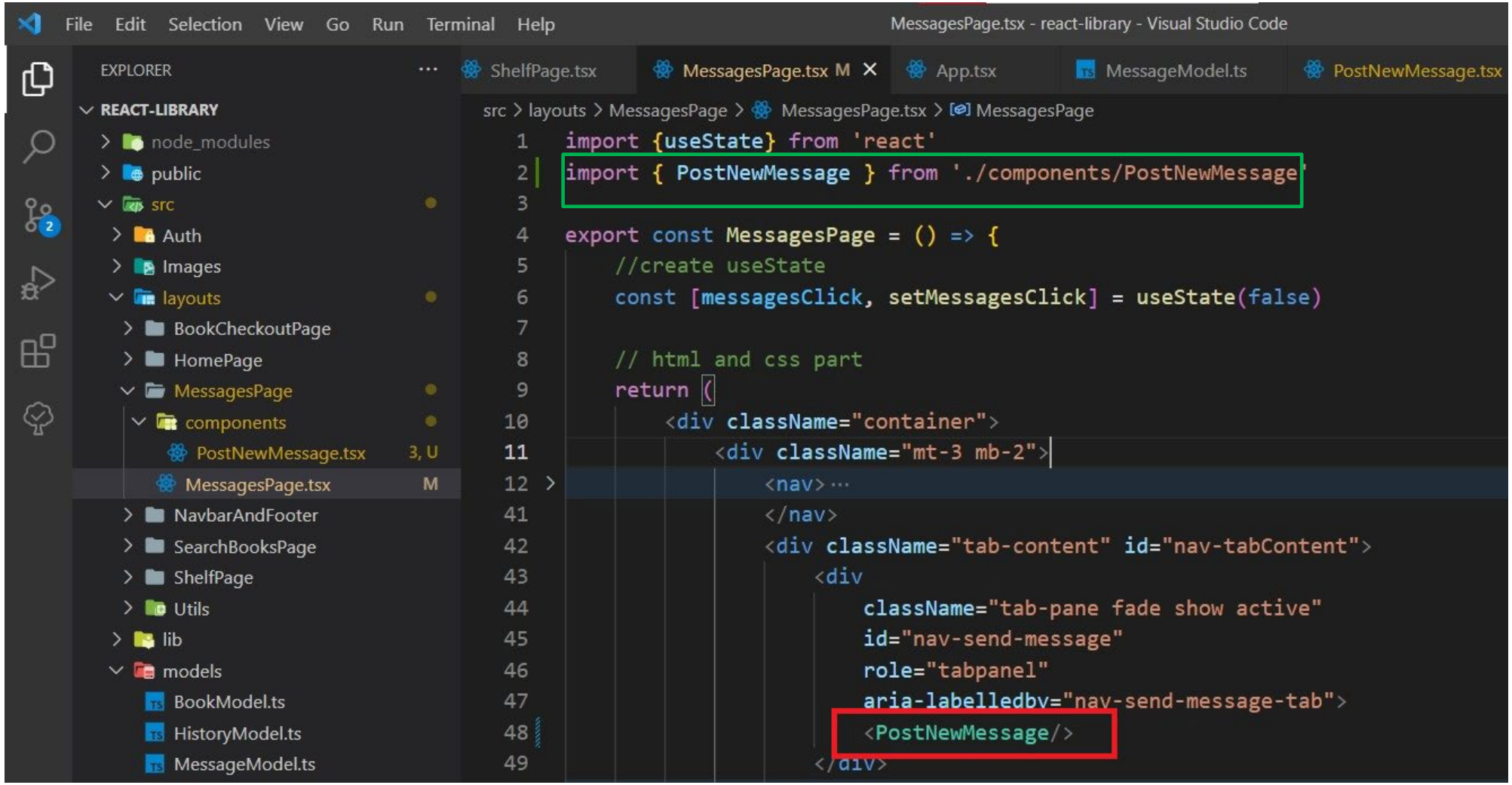
```
<div className="mb-3">…
</div>
{/* question label and textarea */}
<div className='mb-3'>
    <label className='form-label'>
        Question
    </label>
    <textarea className='form-control' id='exampleFormControlTextarea1'
        rows={3} onChange={e => setQuestion(e.target.value)} value={question}>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



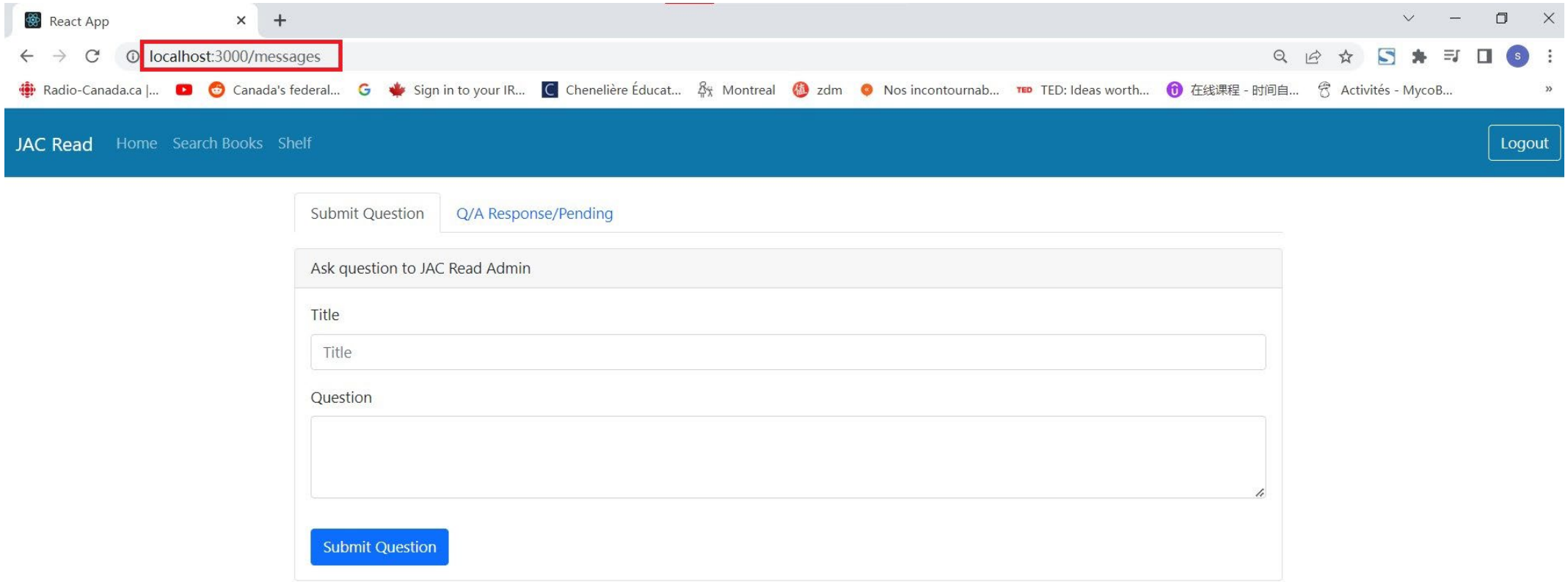
Stpe 6, Go to MessagesPage.tsx, replace <P>Messages</p> with PostNewMessage component and import it.

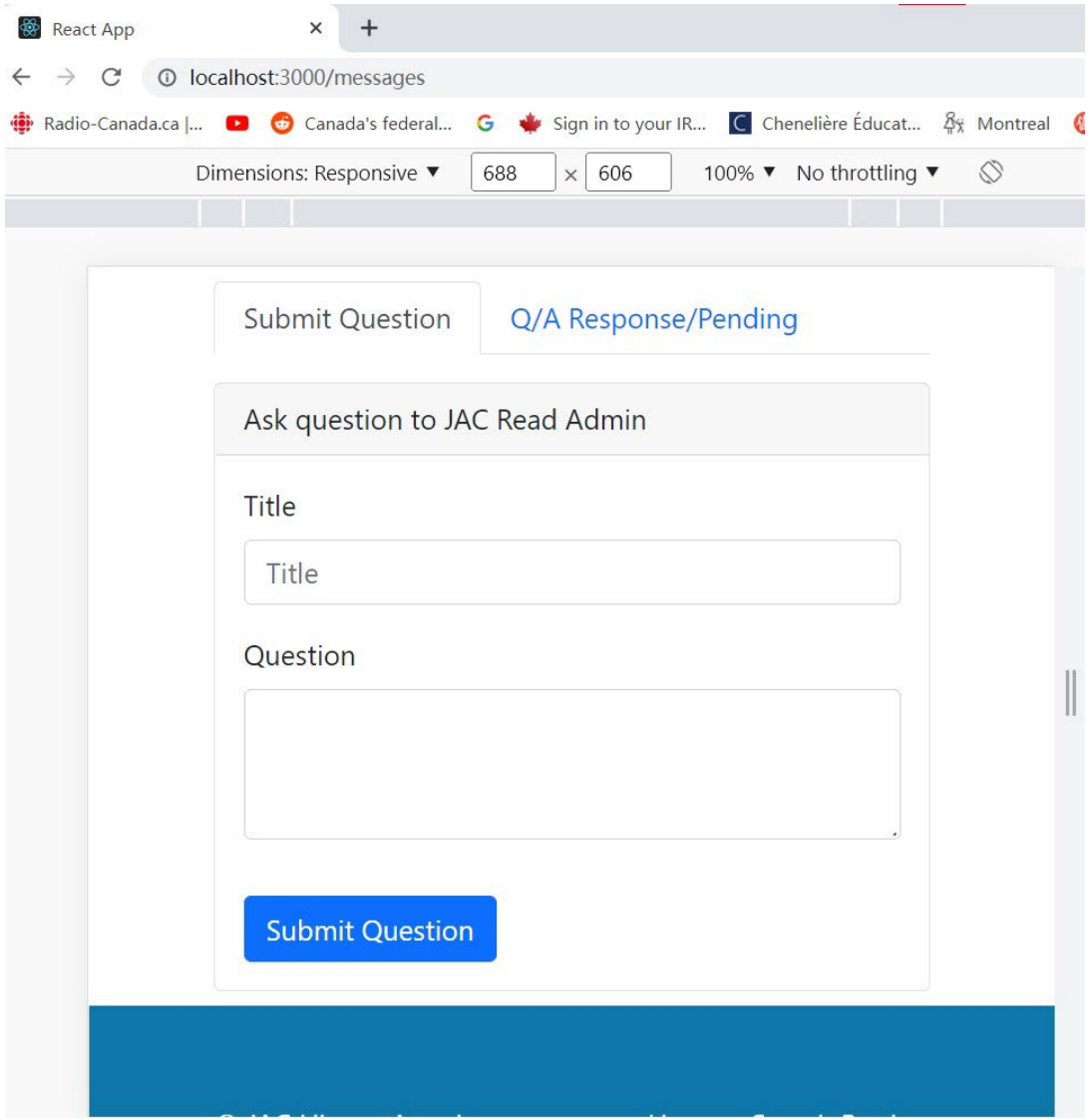


After the change, the code should be as below:



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



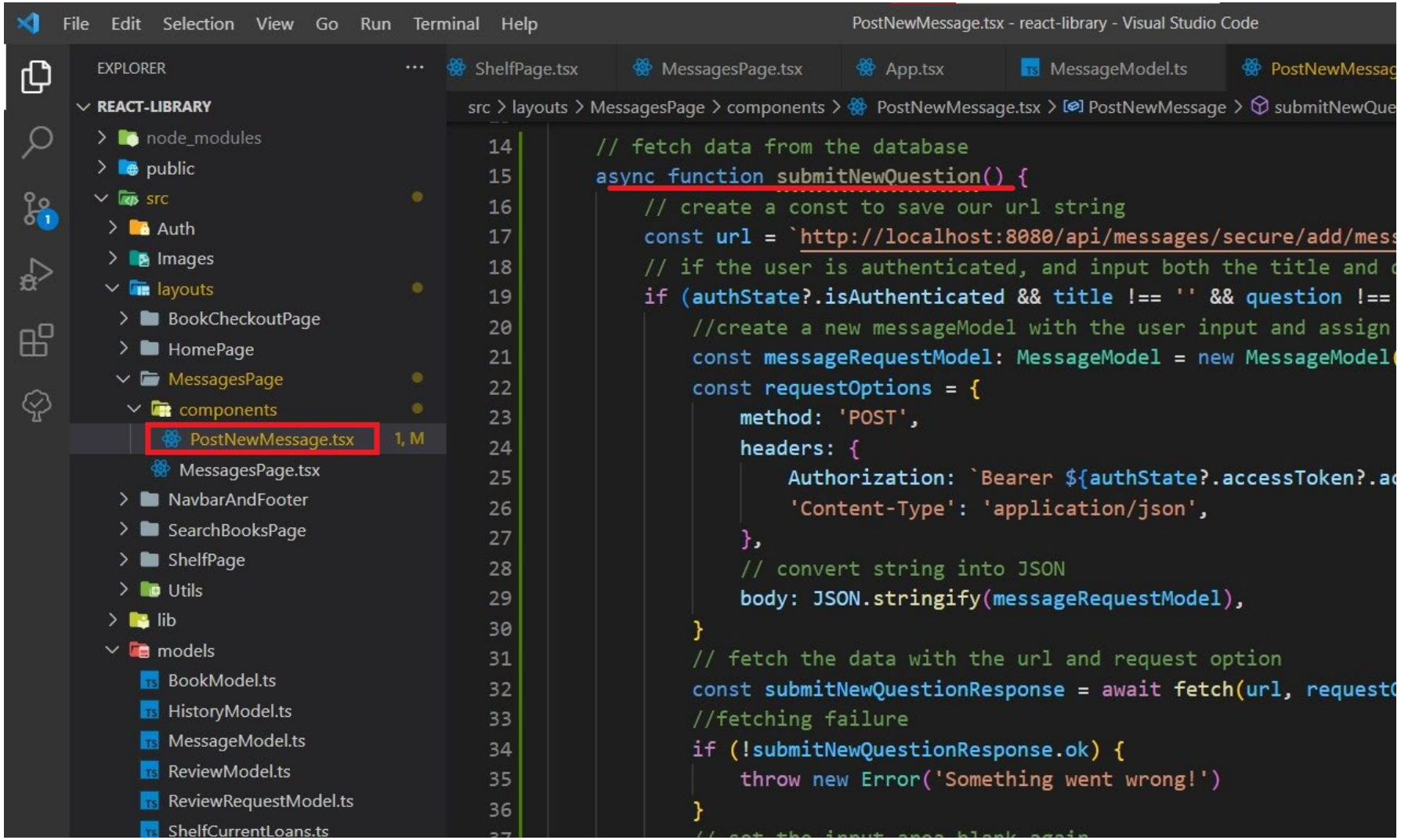


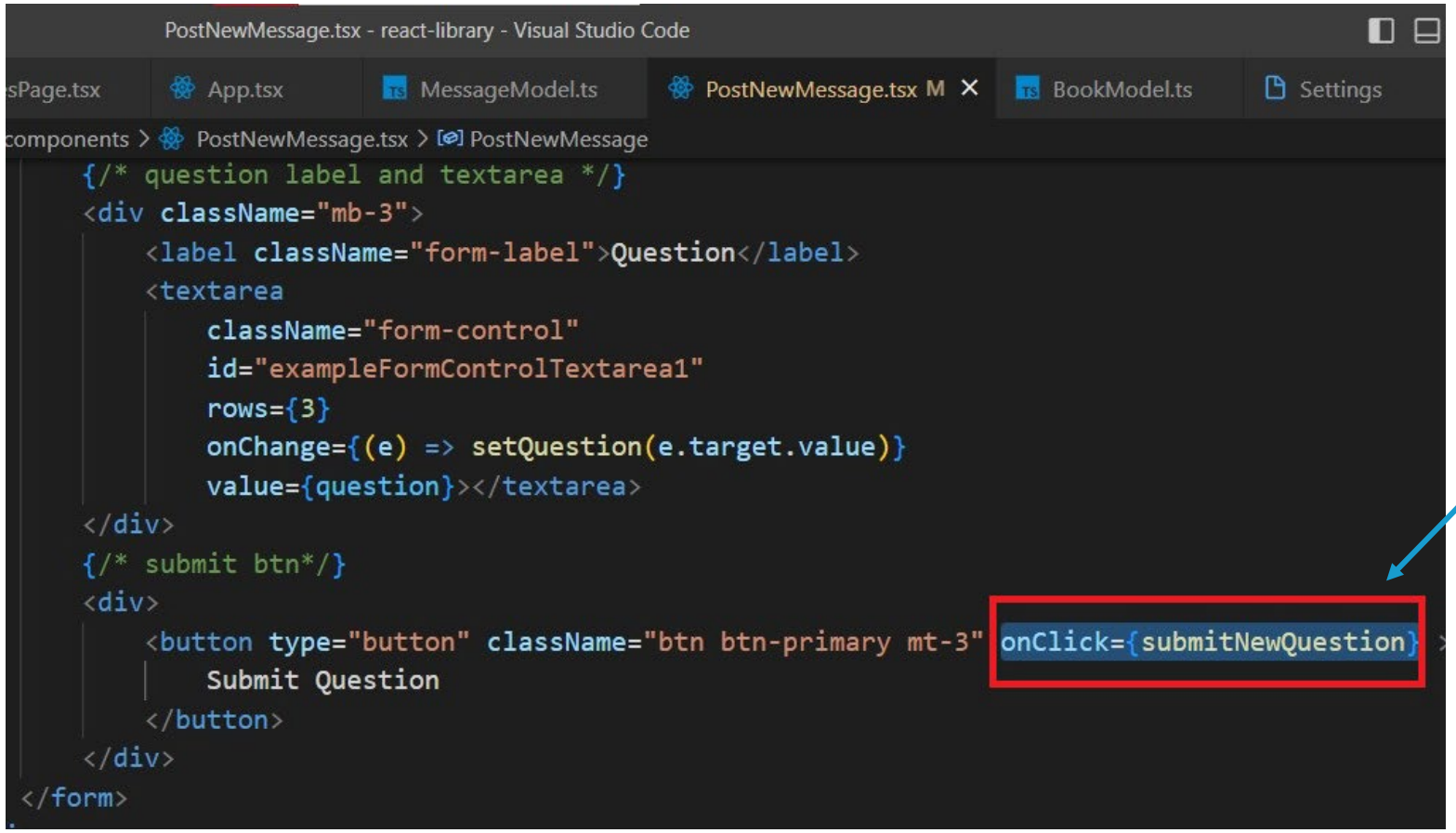
Right click and choose “Inspect”， choose mobile version, the page still works!

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.

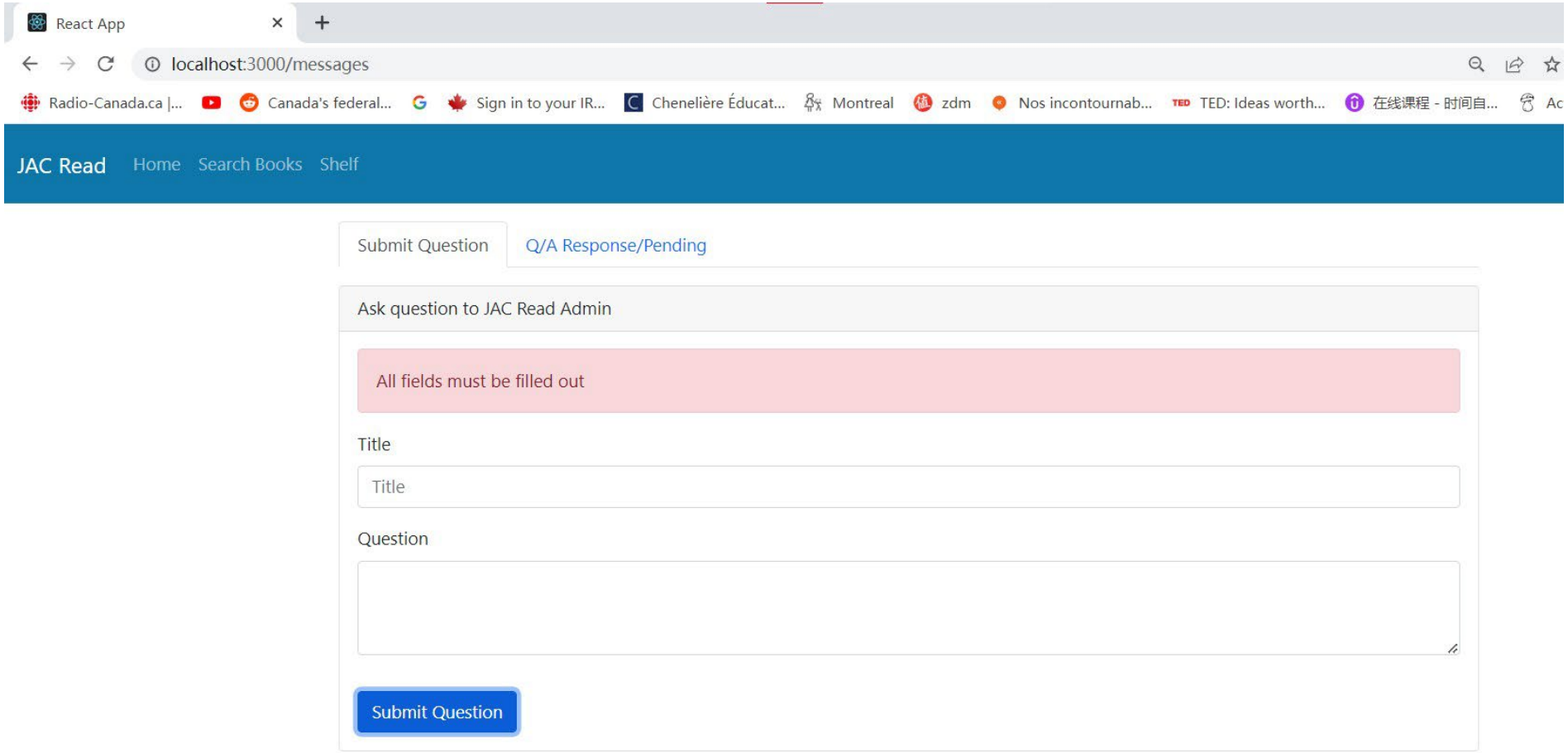
Step1, In PostNewMessage.tsx file, jumping down from these pieces of state and creating a new async function submitNewQuestion().



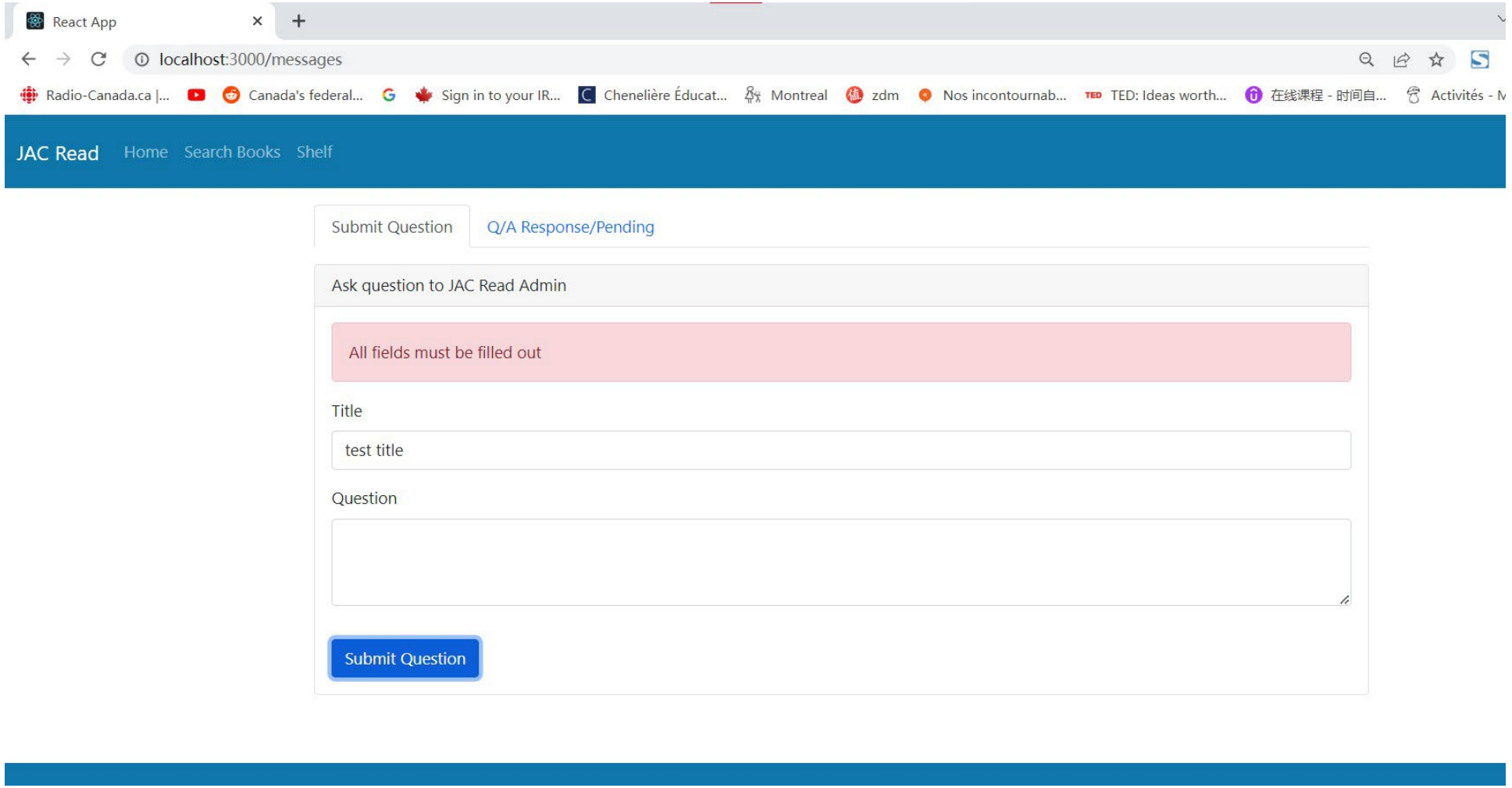


Step 2, add a onClick listener to submit button.

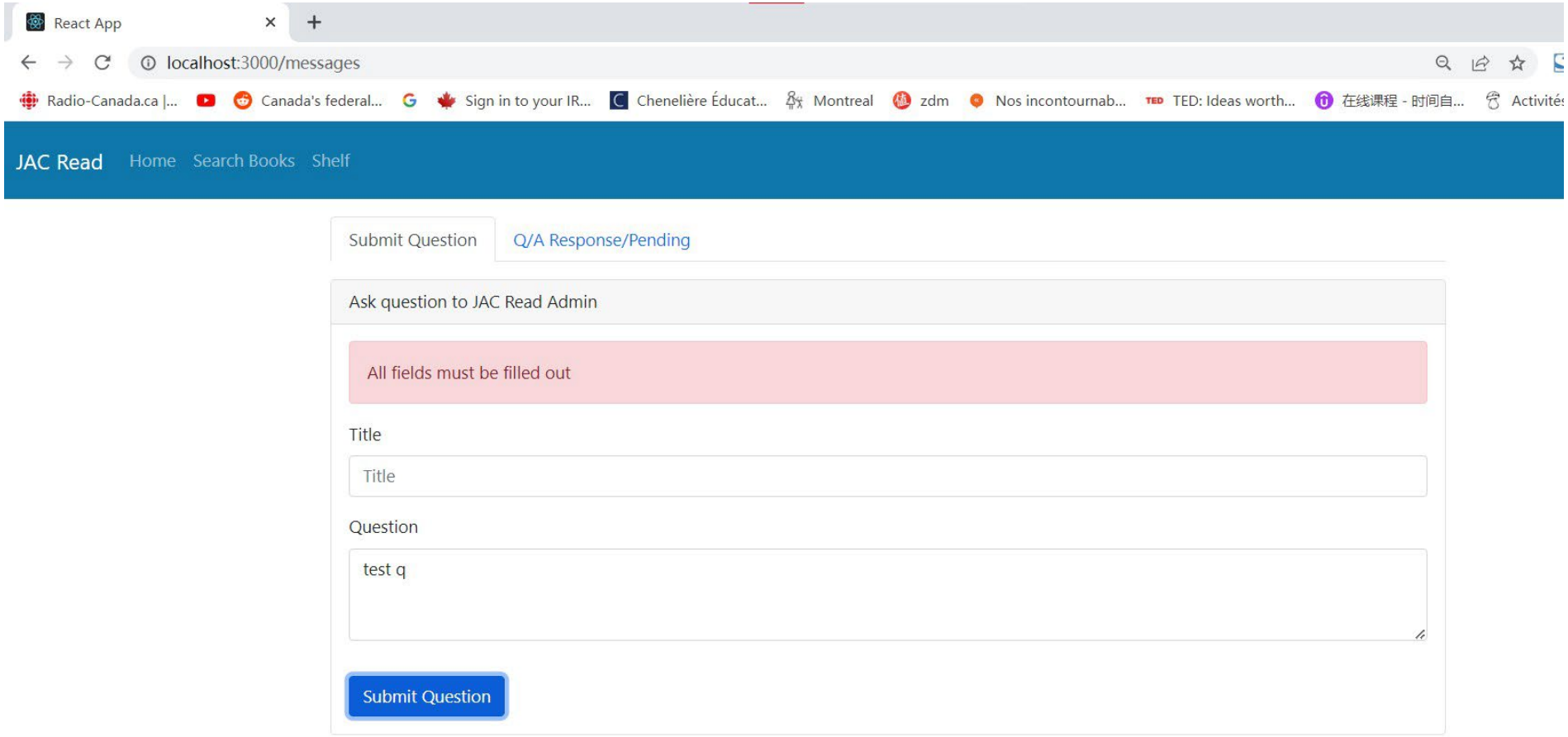
Visit http://localhost:3000/messages, try and submit a question with nothing in our title or question input area, alert message “All fields must be filled out” will show up:



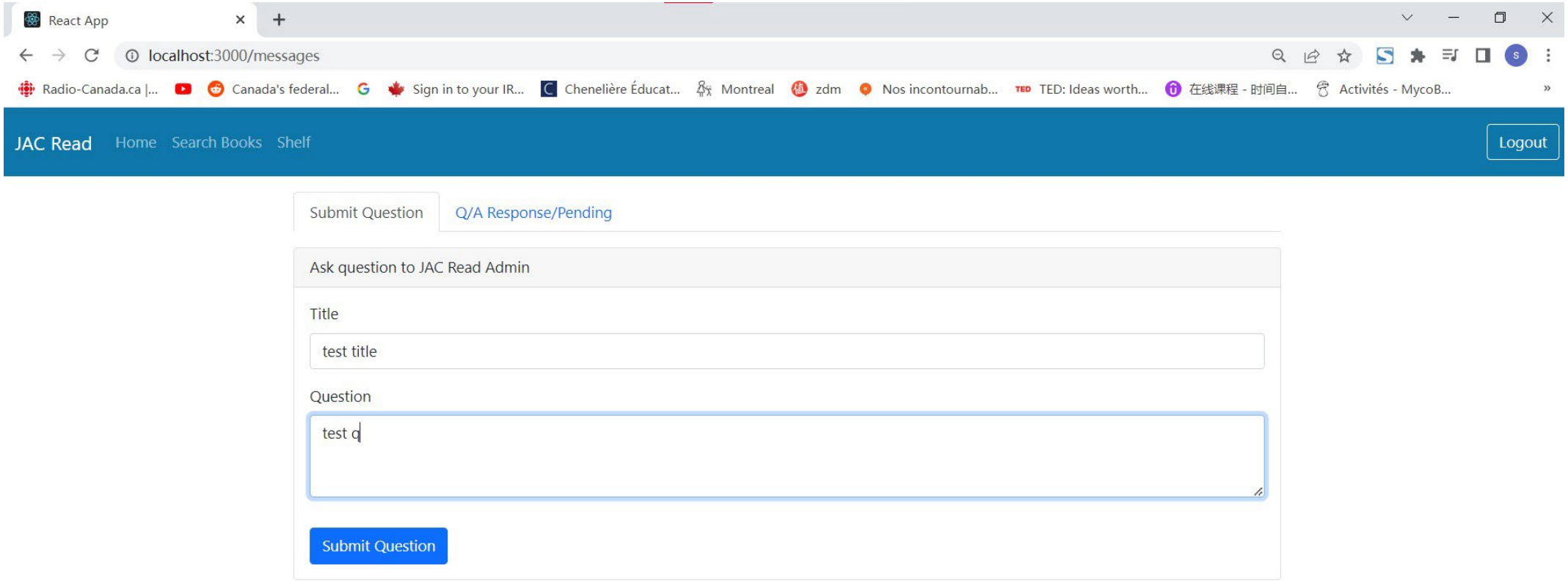
try and submit a question only with title input but nothing in question input area, alert message “All fields must be filled out” also will show up:



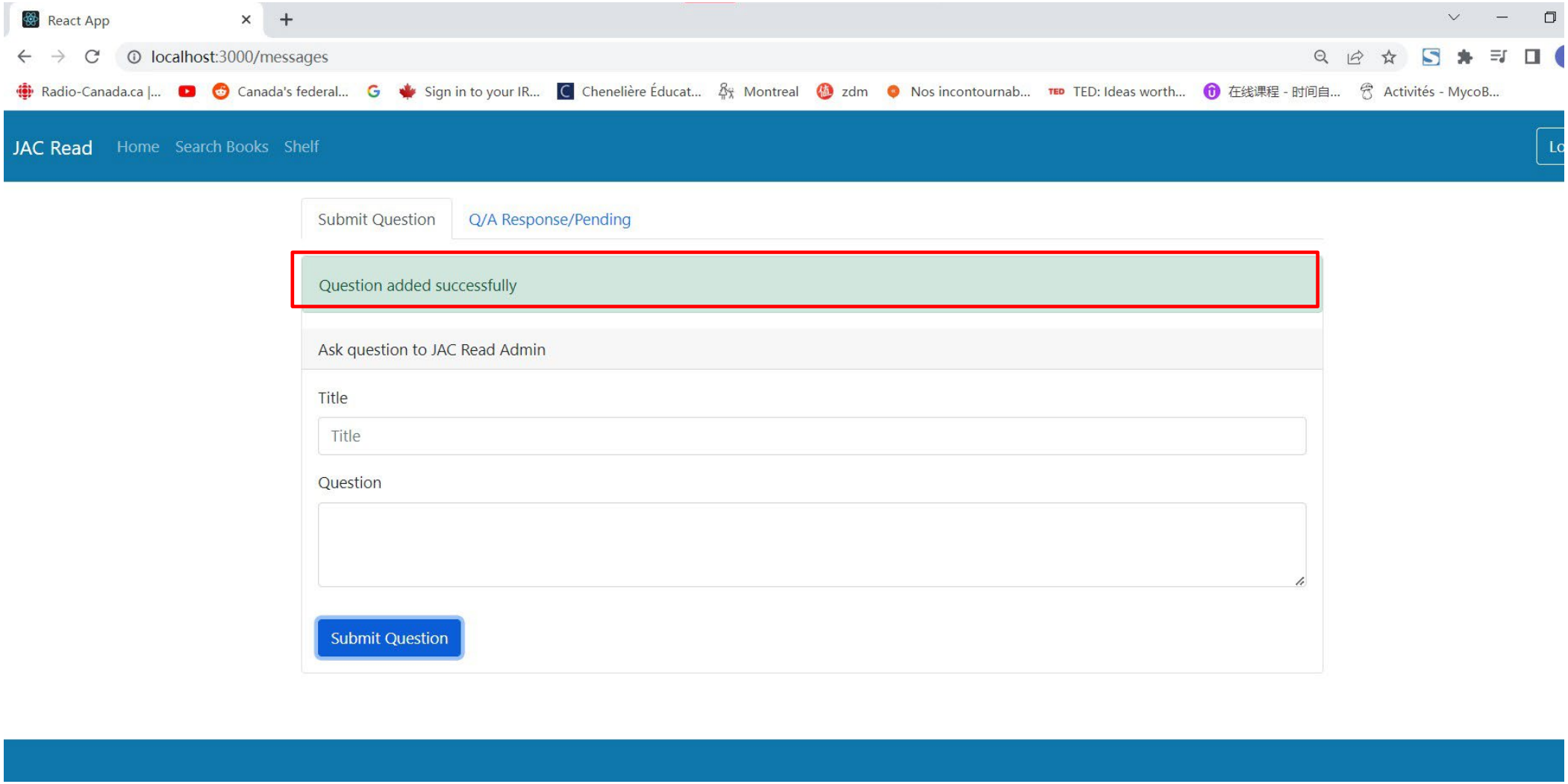
This time, with question input but nothing title input area, alert message “All fields must be filled out” also will show up:



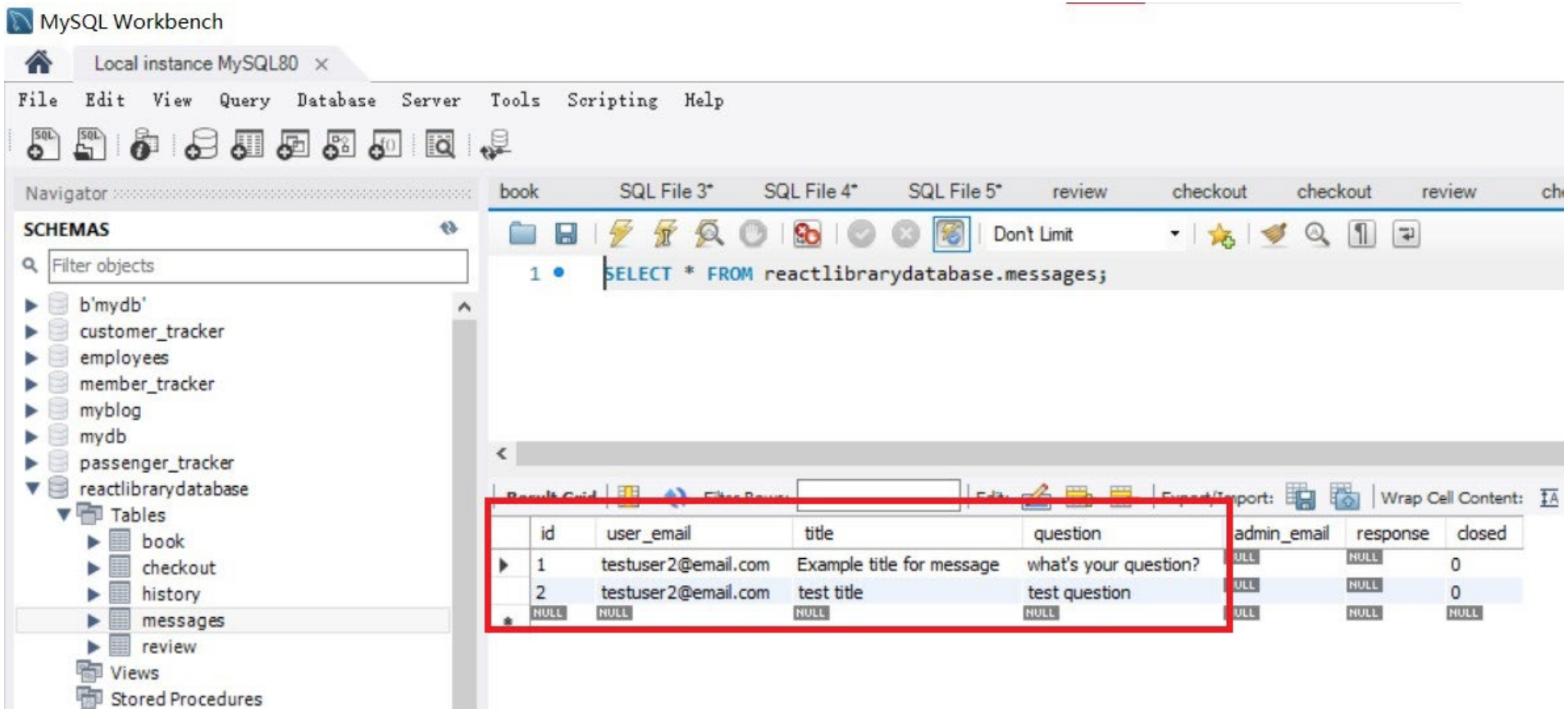
Input both two areas and click on “submit question” button,



Questions added successfully and we have the banner going above.



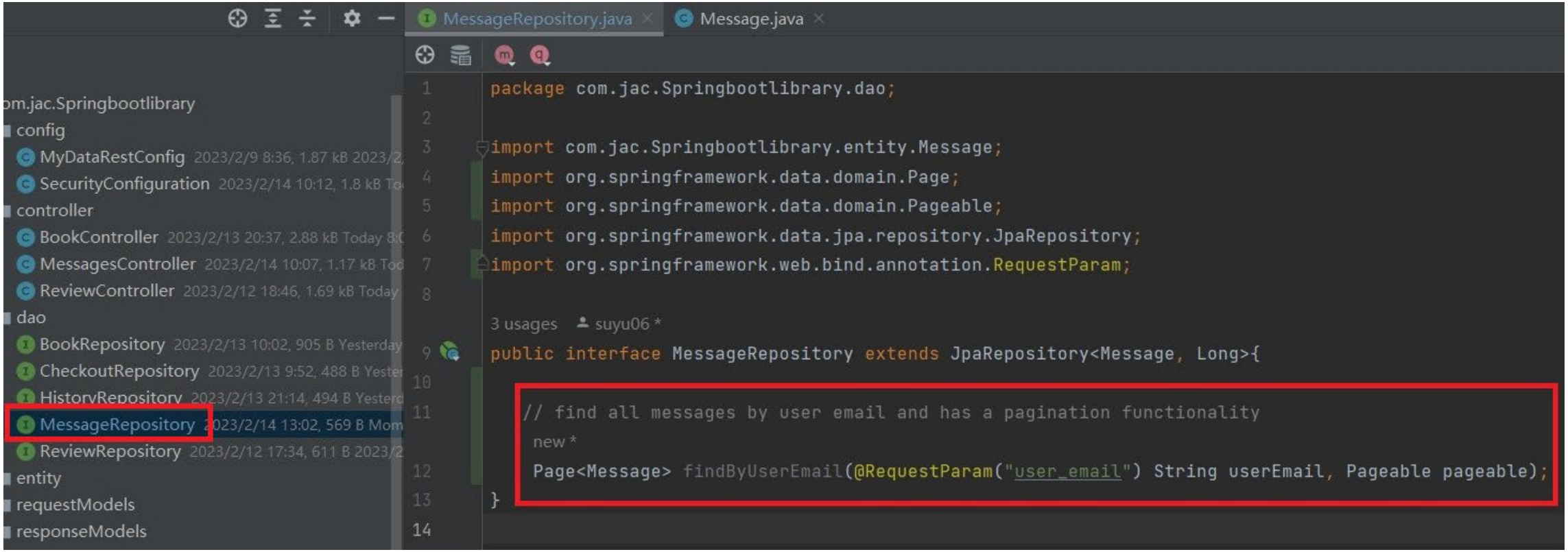
Go to our database, our question has added to our “messages” table successfully.

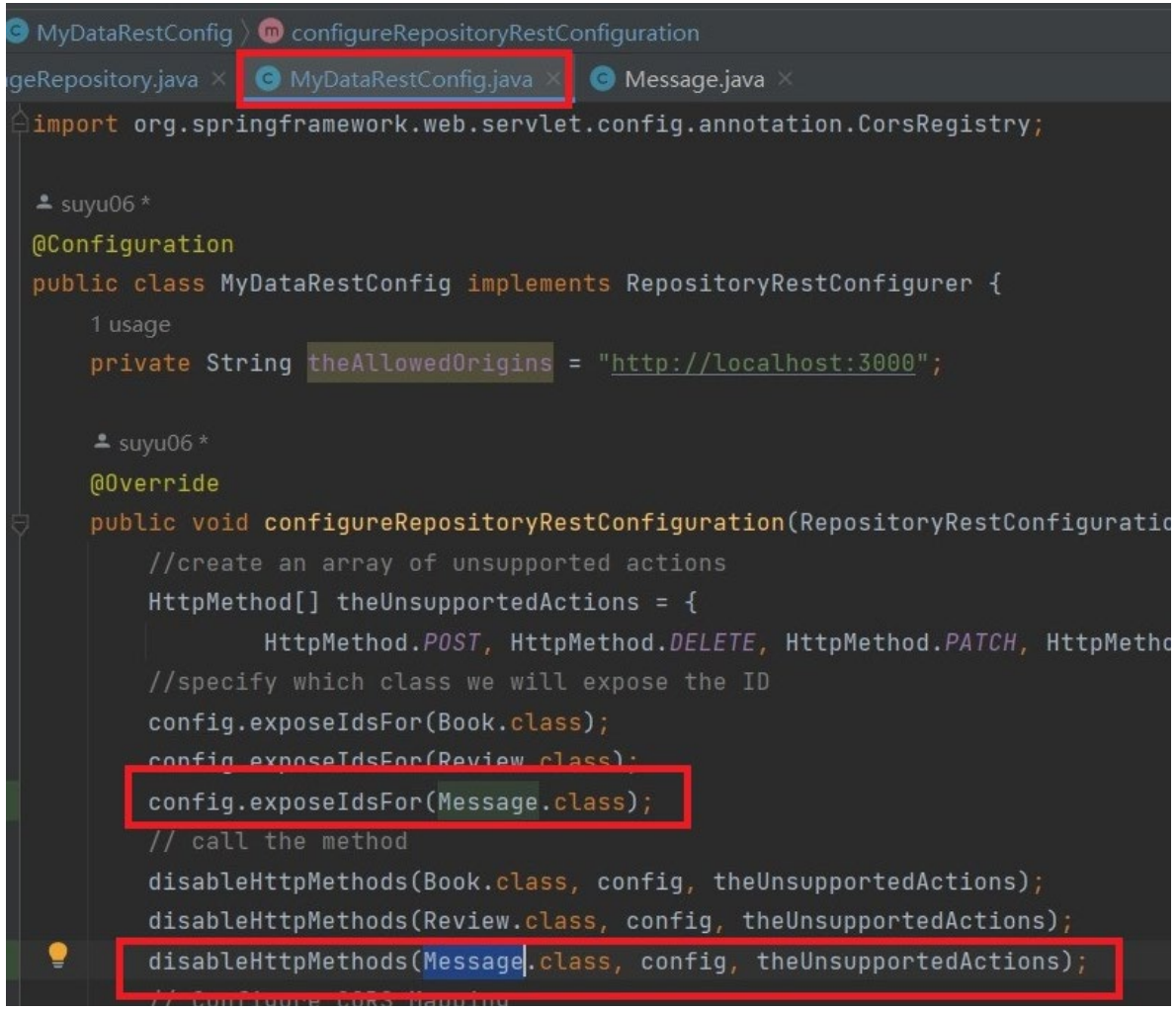


10. SPRINGBOOT MESSAGE REPOSITORY AND SECURITY

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.

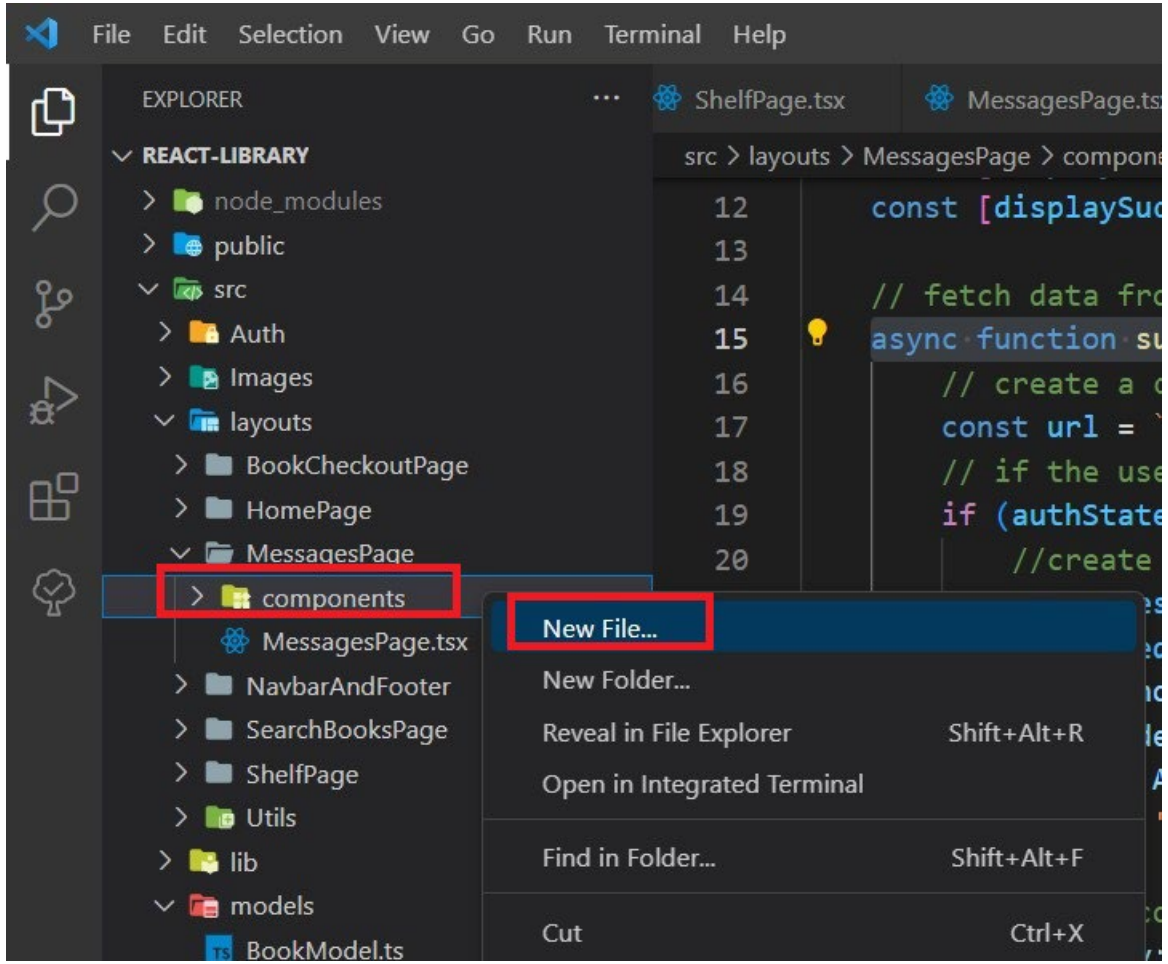




Step 2, go to MyDataRestConfig file, expose the Ids for message and we want to disable HTTPS methods for message.

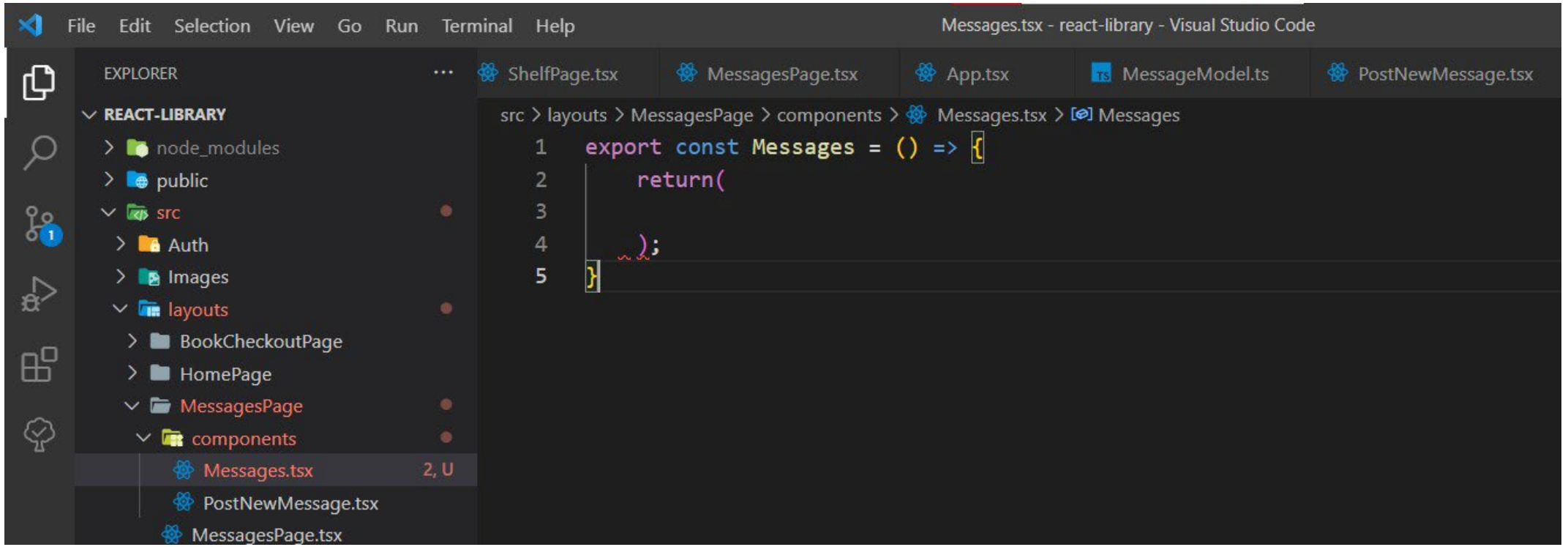
11.REACT MESSAGE COMPONENT

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.

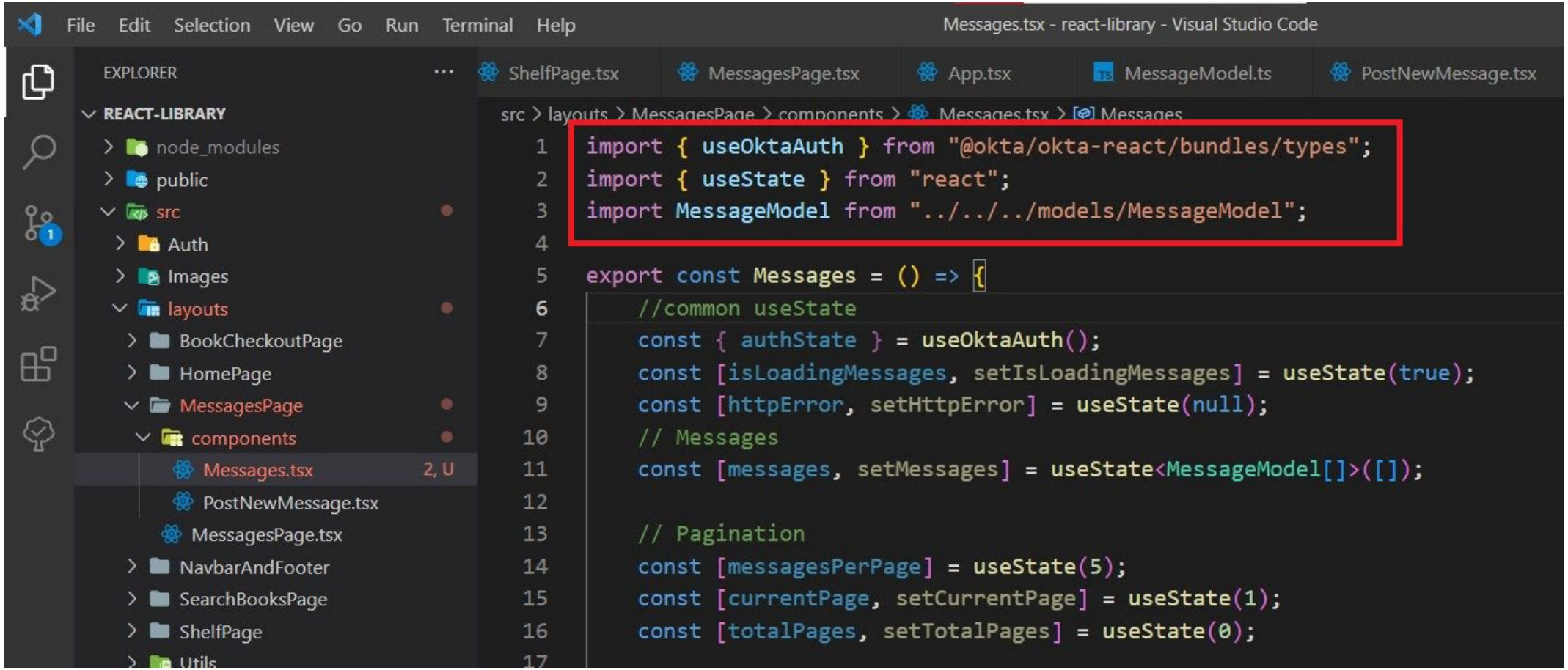


Step1, inside MessagesPage, right click components, create a new file called messages.tsx.

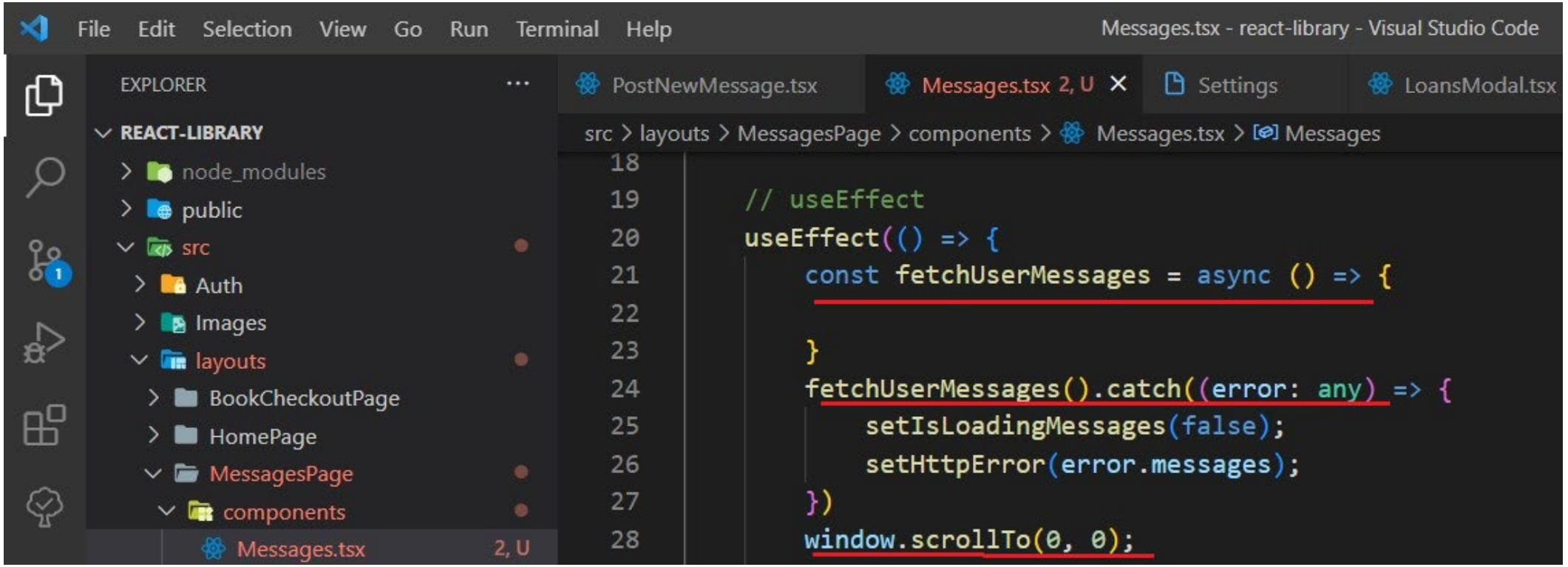
Step 2， create export const Messages stucture.



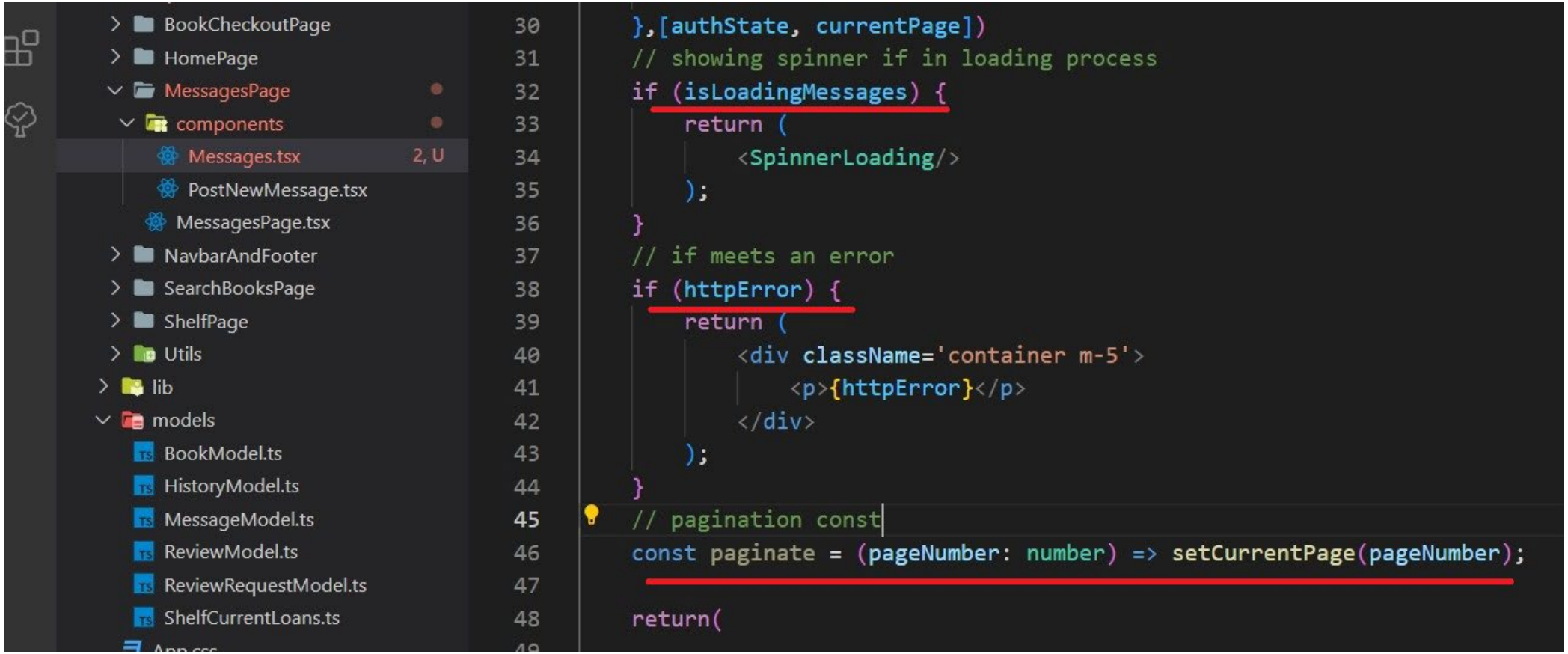
Step3, create useState and import useOkatAuth, useStae and MessageModel.



Step 4, create useEffect structure.
An async function, then a catch error function and a method to make the page show from the top.



Step 5, create loading process, httpError and paginate.

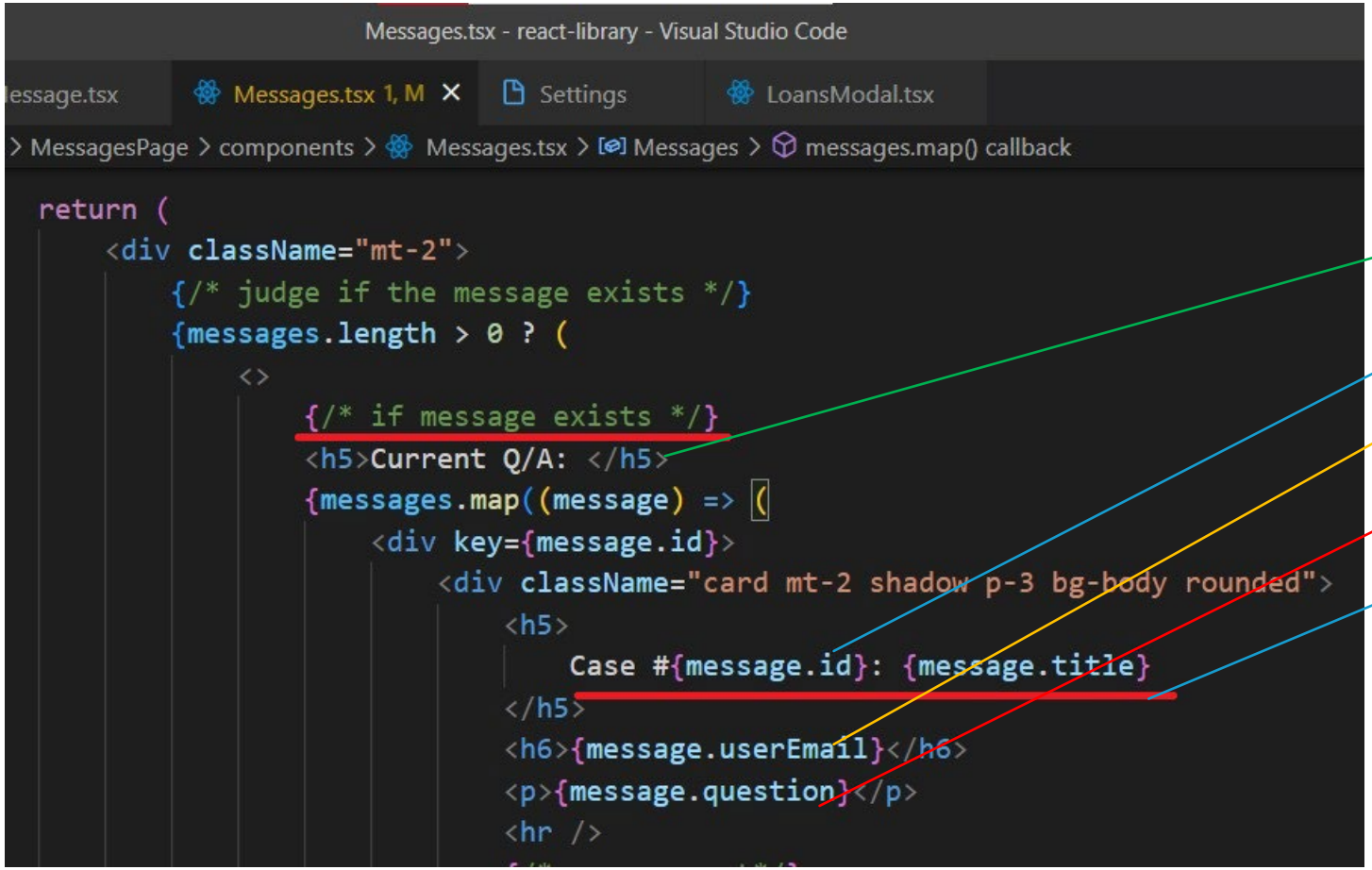


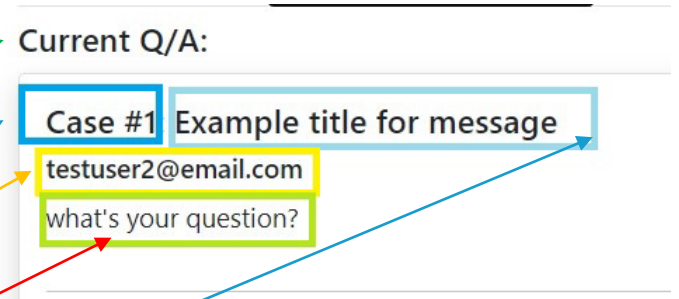
Step 6, fill in useEffect with async fetchUserMessages();

```
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);
```

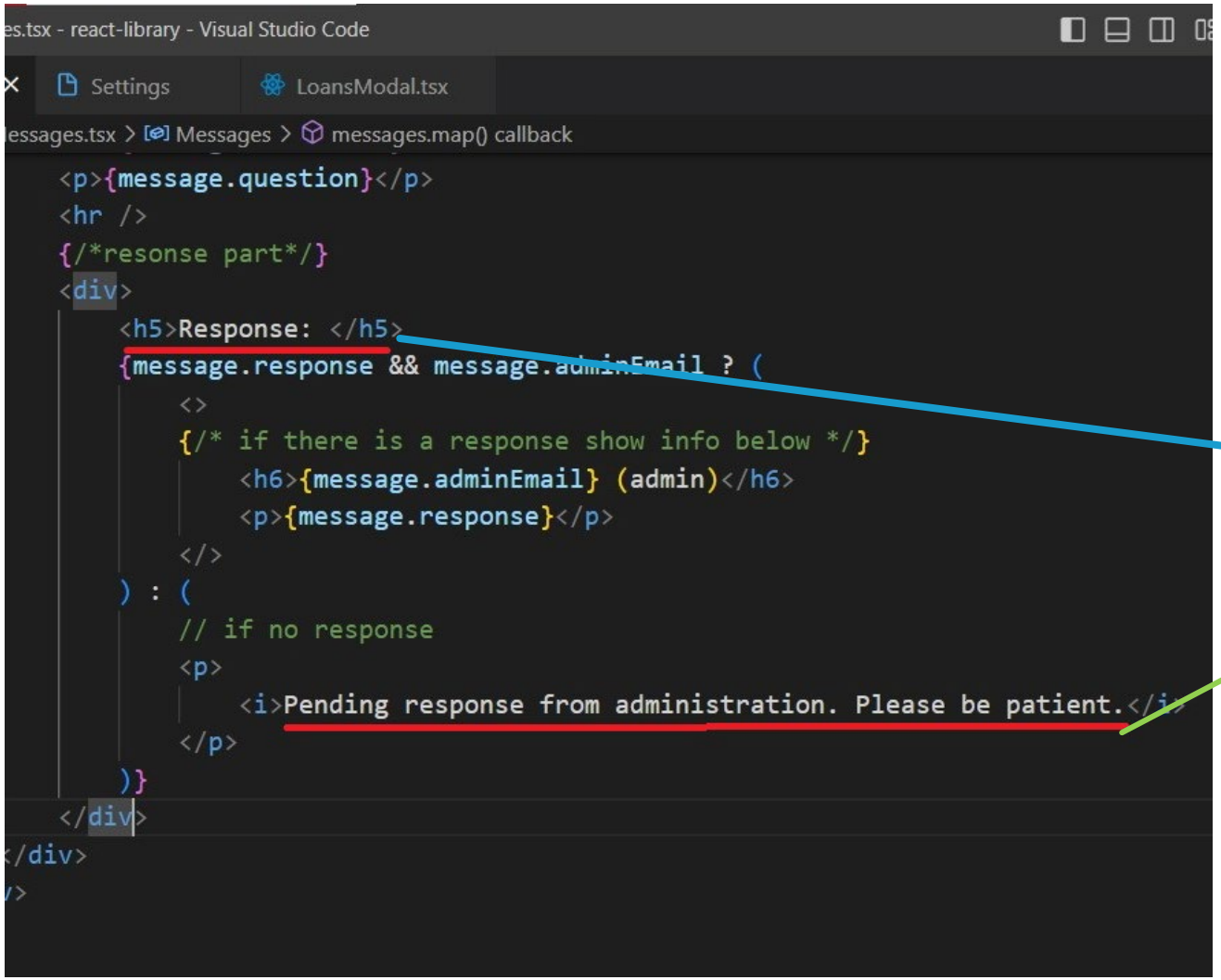
12.REACT MESSAGE COMPONENT HTML/CSS

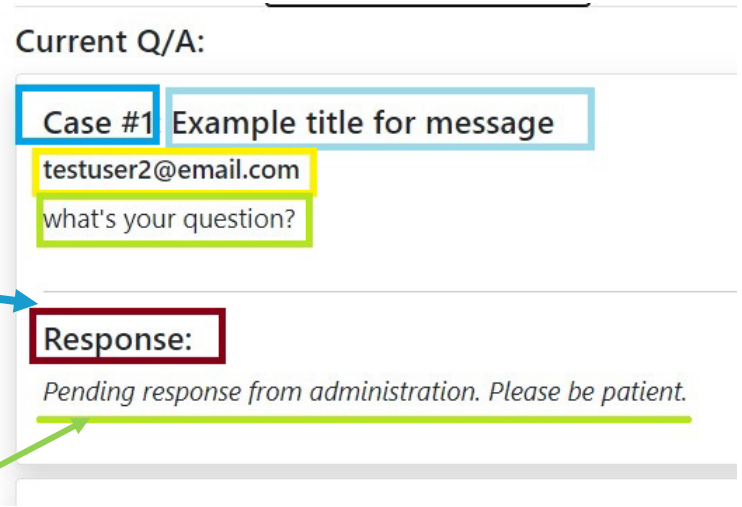
First, judge whether.message exists, if exists:



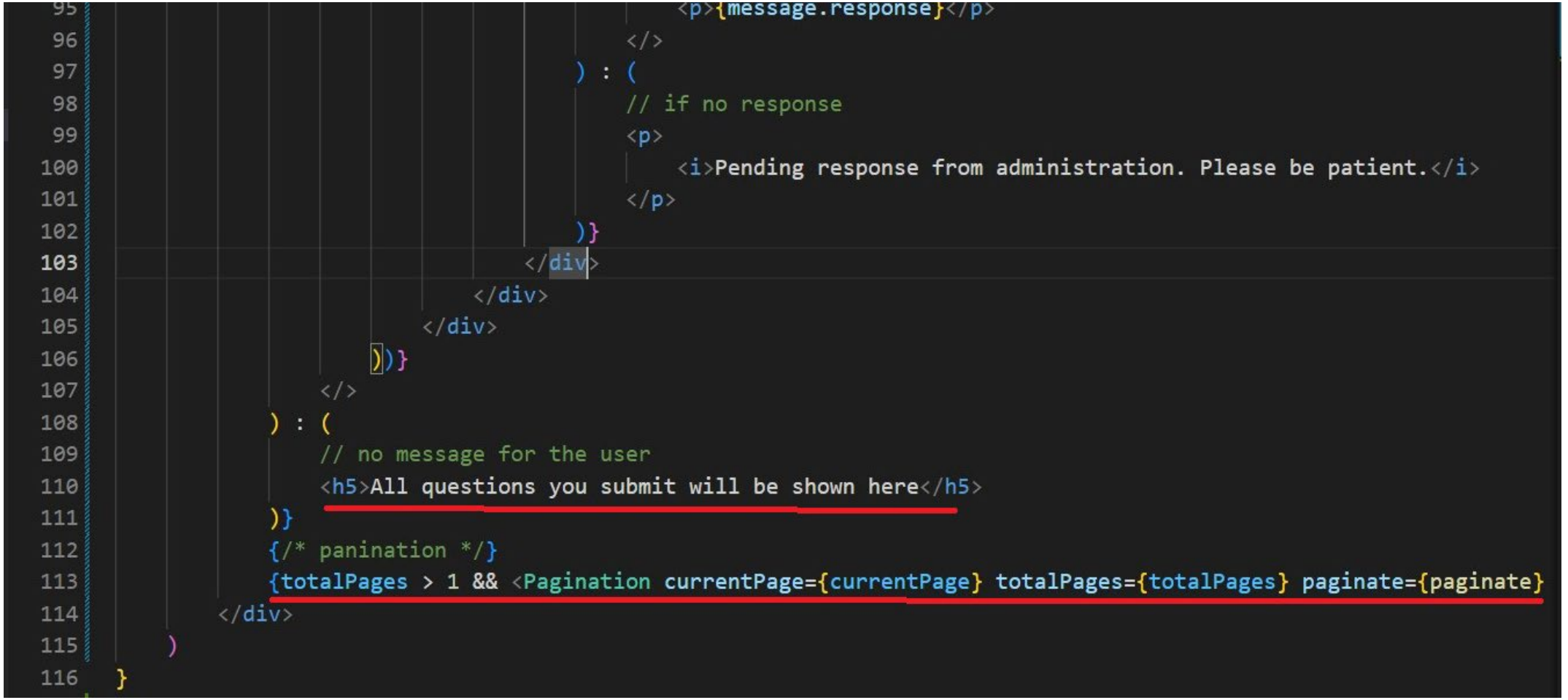


Then whether there is a Response :

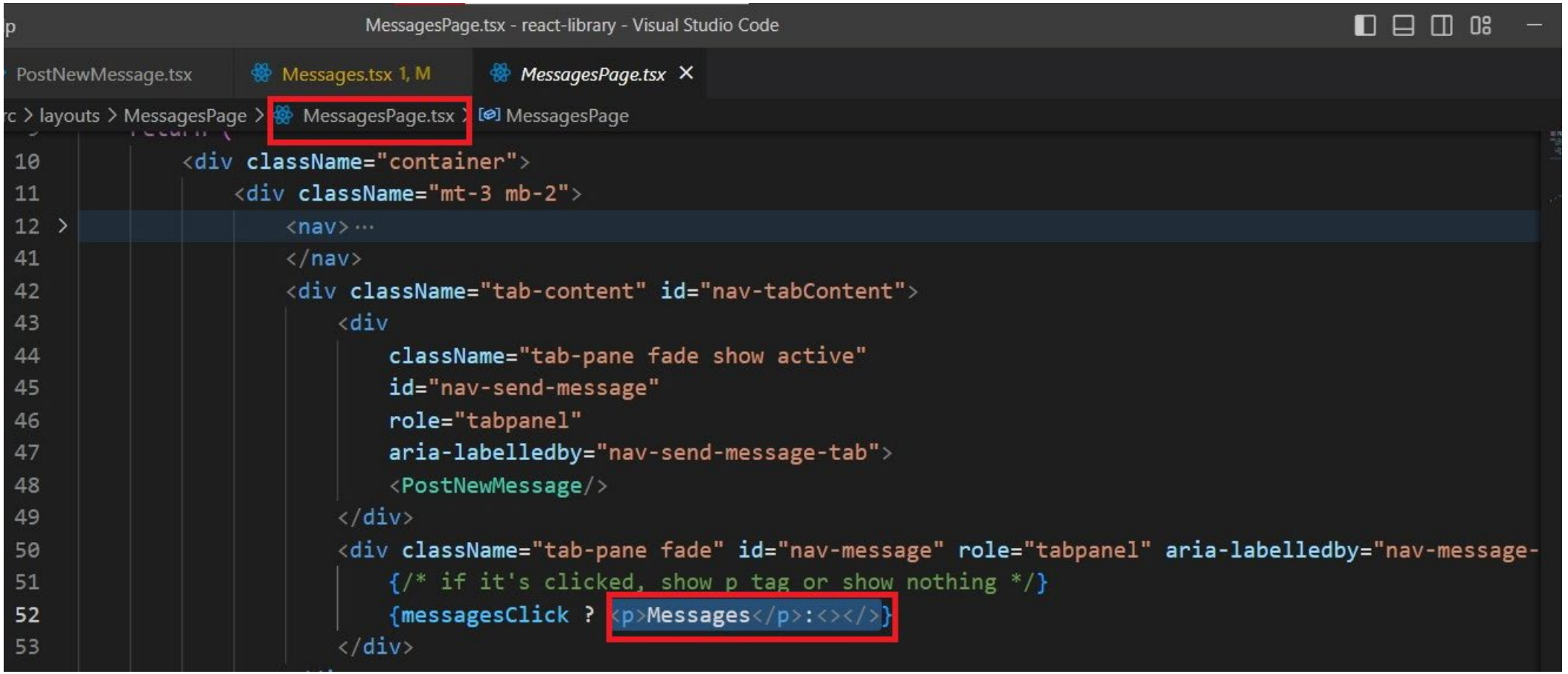




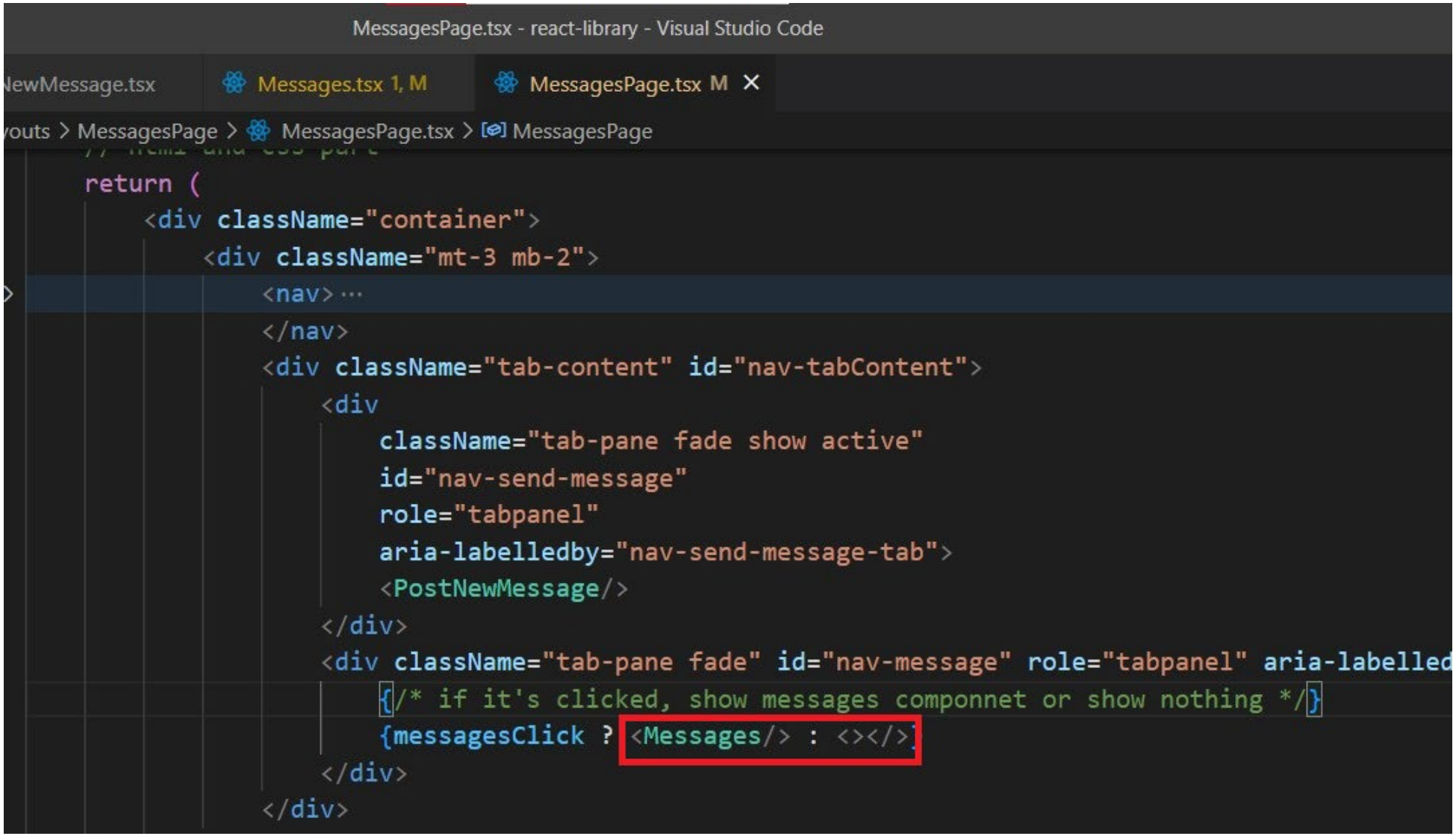
If there is no any question from the user, just show “All questions you submit will be shown here”. And then, add a pagination function.

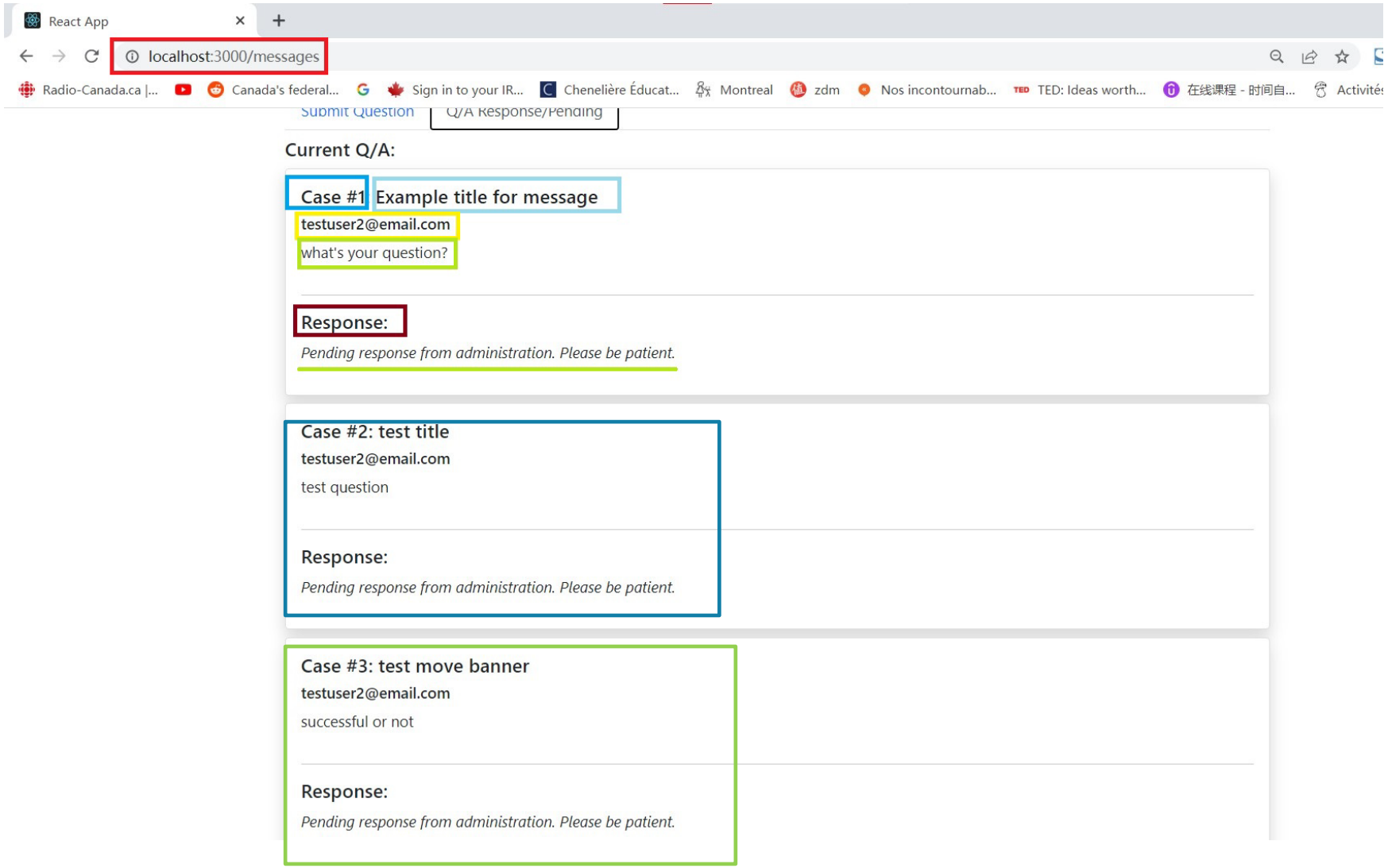


Go to MessagesPage.tsx, replace p tag messages with component.



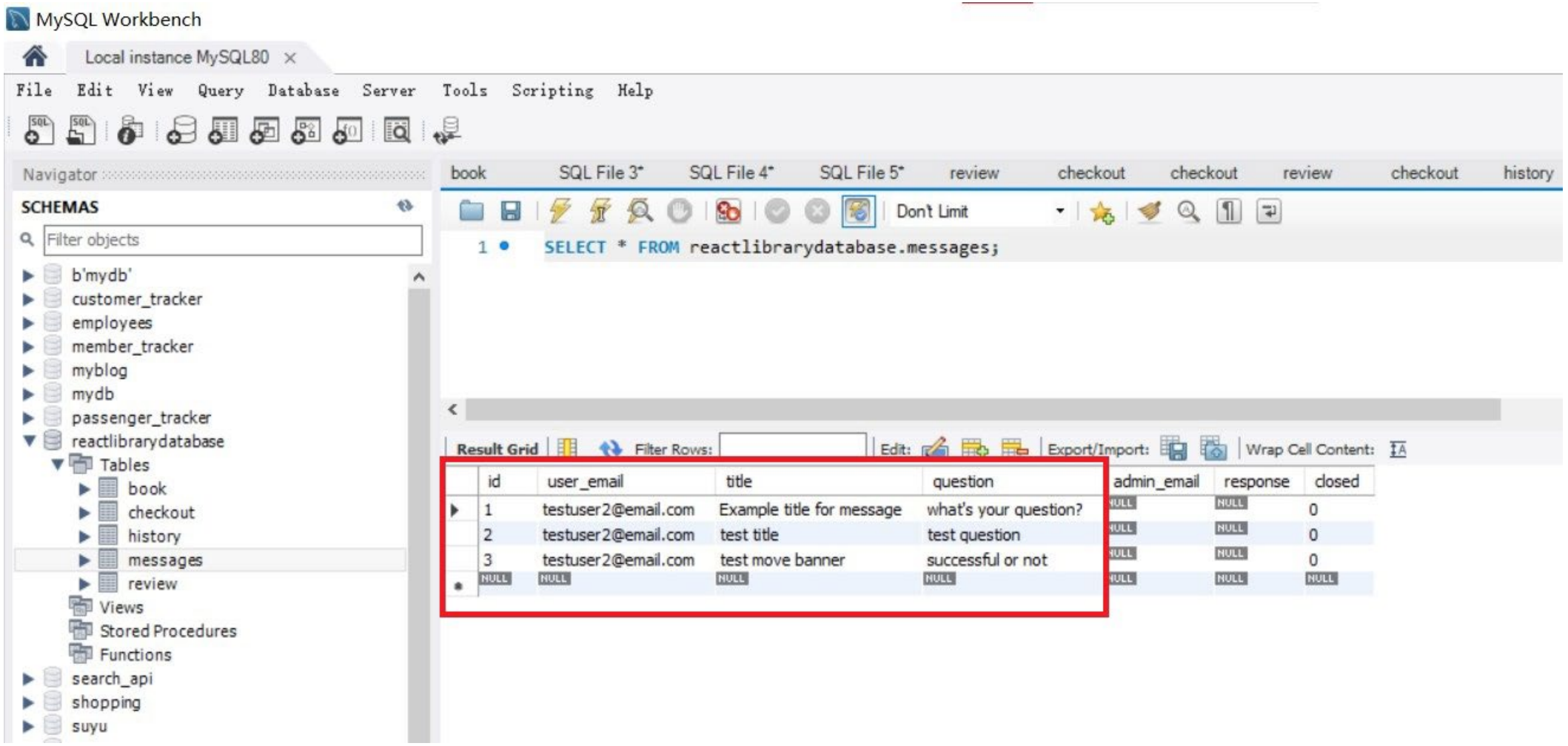
If there is any error, right click and choose “Quick Fix”, to import the component.





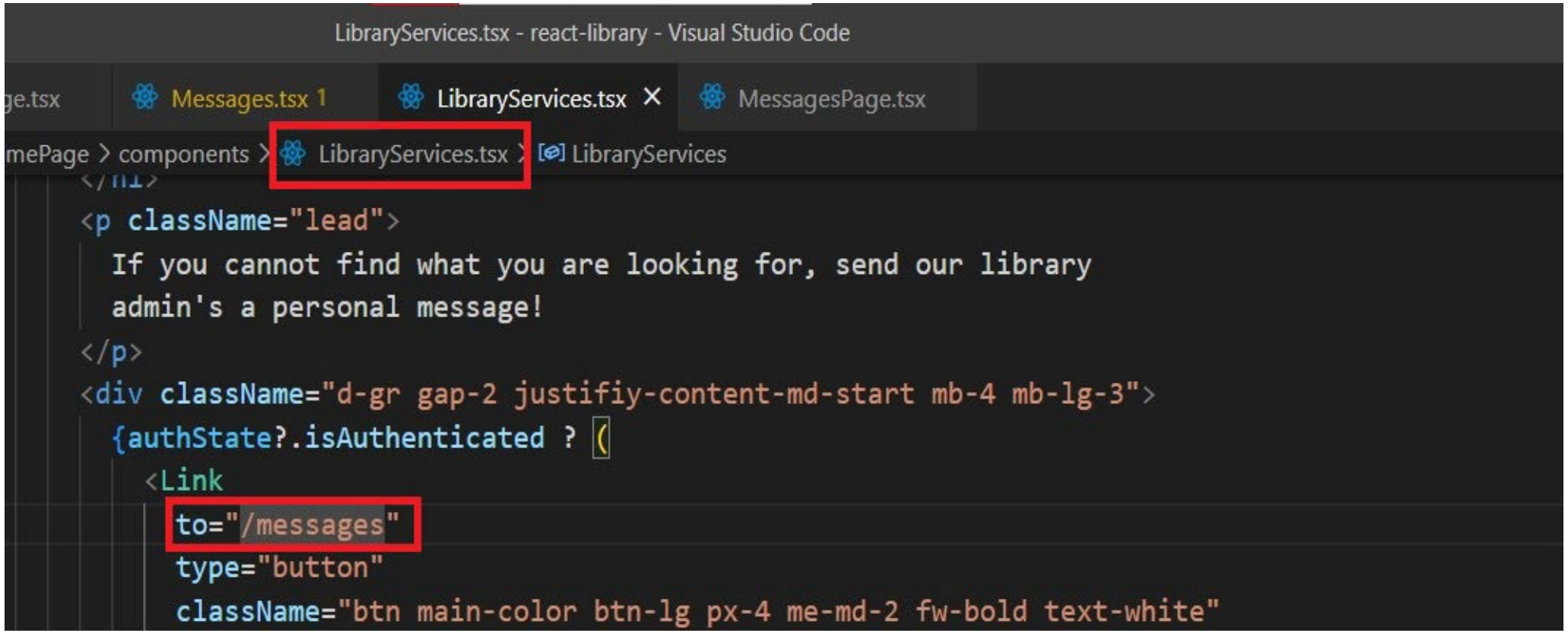
http://localhost:3000/messages, click on the "Q/A Response/Pending" tab, it will show all the messages the current user submitted.
There are 3 questions submitted by the "testuser2@email.com"

Let’s check the result in our data base.
In the table messages, the “testuser2@email.com” has submitted 3 questions.

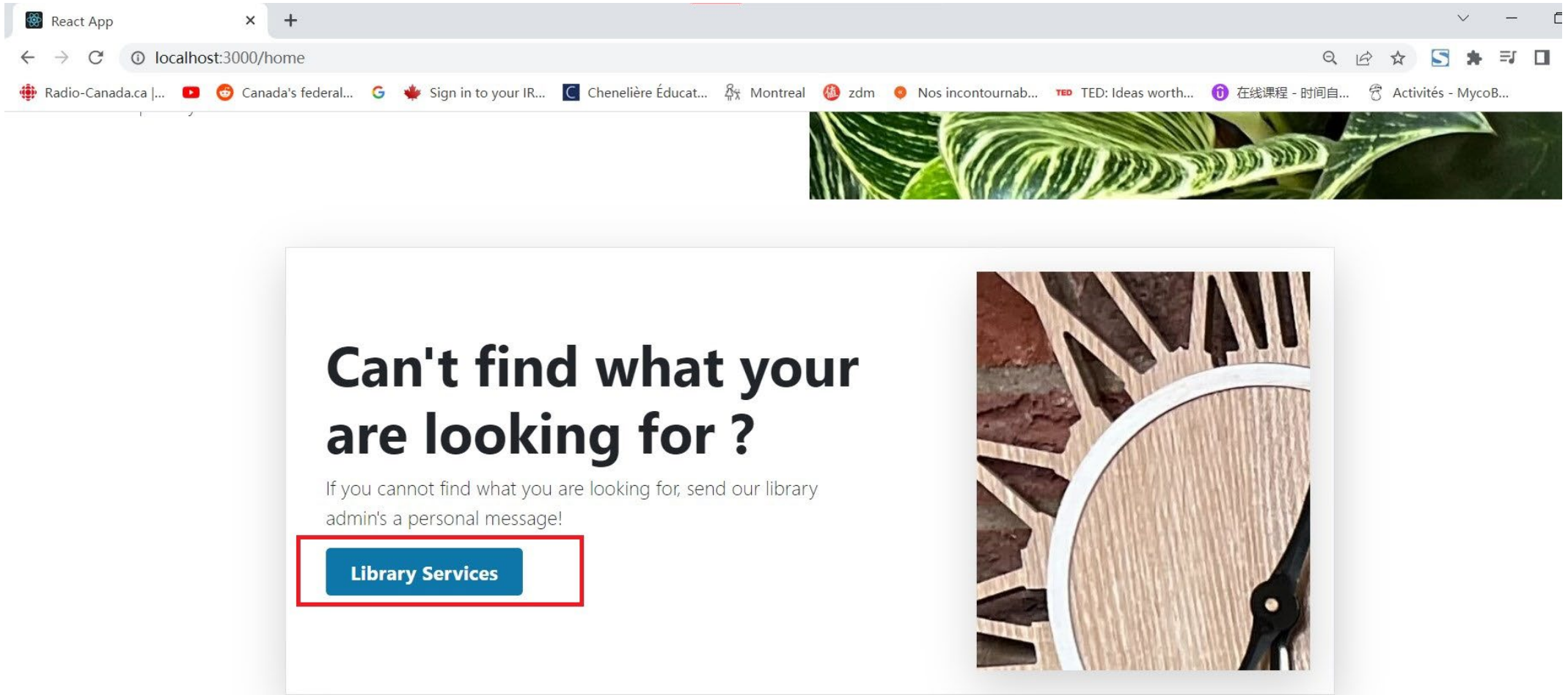


13.REACT ADD LINK TO LIBRARY SERVICE

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.



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:



It will redirect us to the messages page!