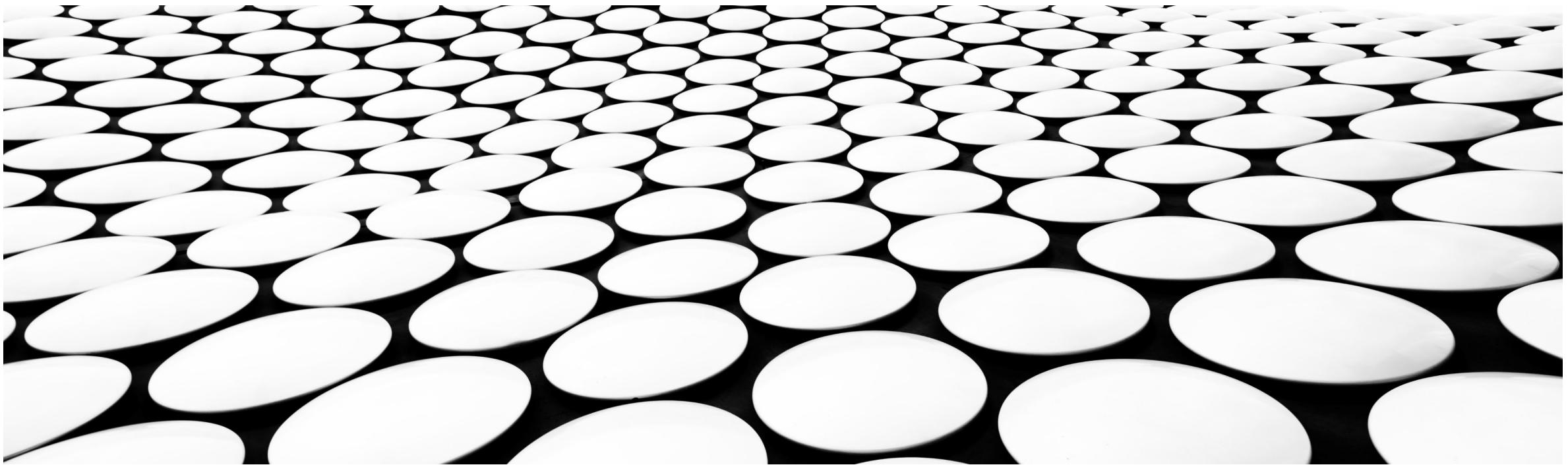


---

---

# S19 ADMIN SERVICES



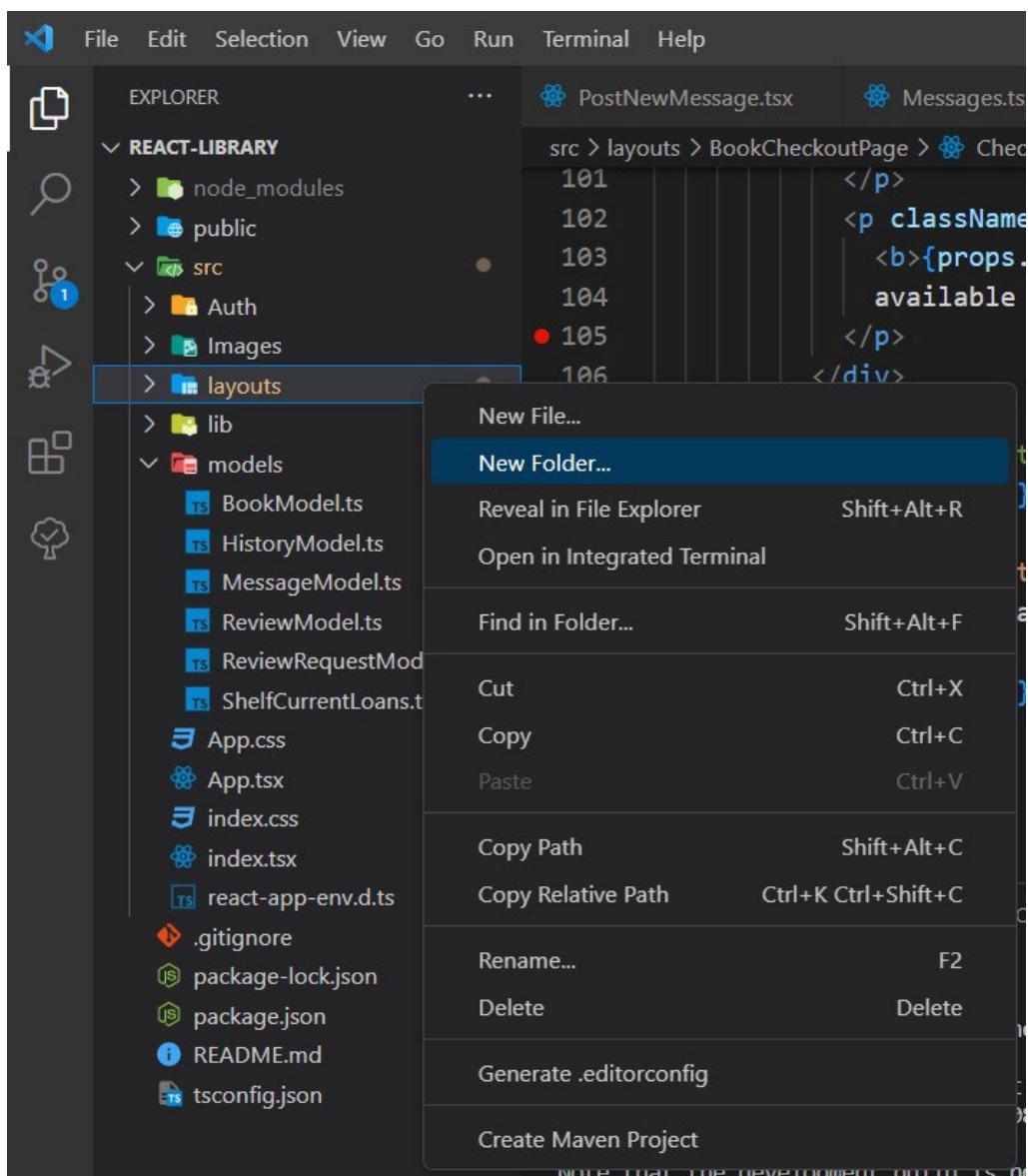
---

---

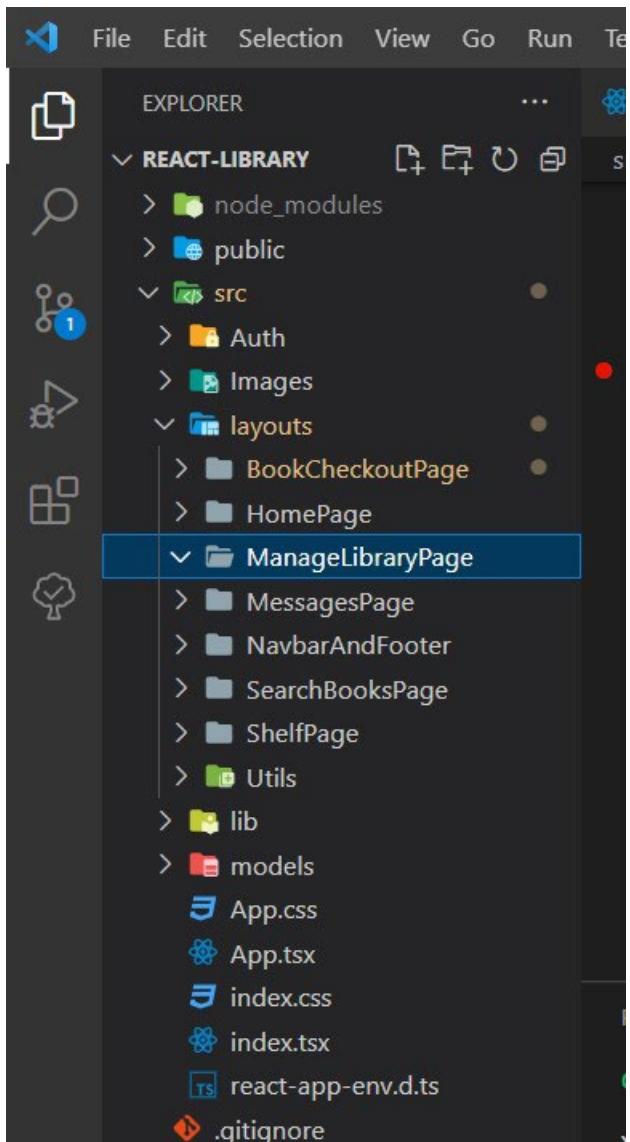
---

# **1.REACT MANAGELIBRARYPAGE**

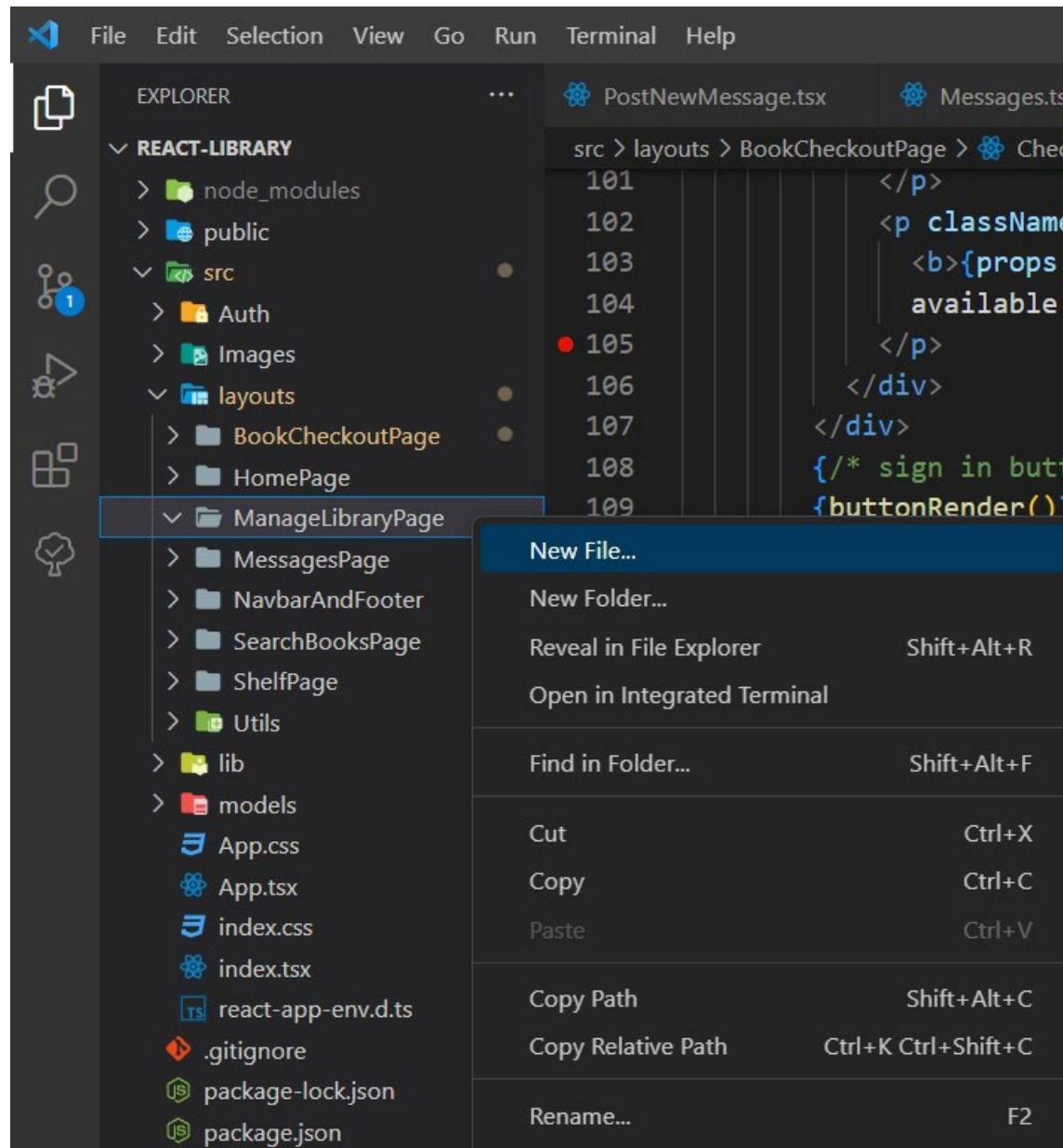




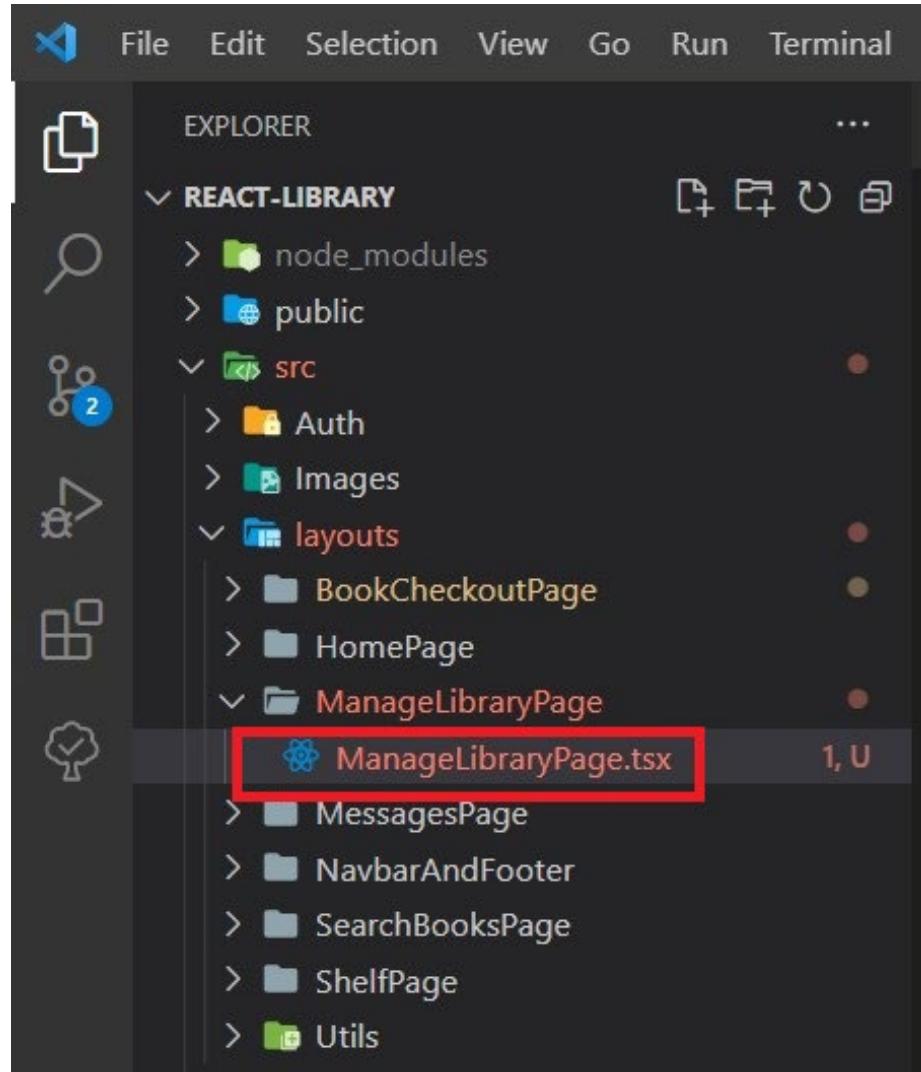
Step1, Jump into our layouts, create a new directory. Right click on the “layouts”, choose “new folder”, and we're going to call this ManageLibraryPage.



ManageLibraryPage folder created.

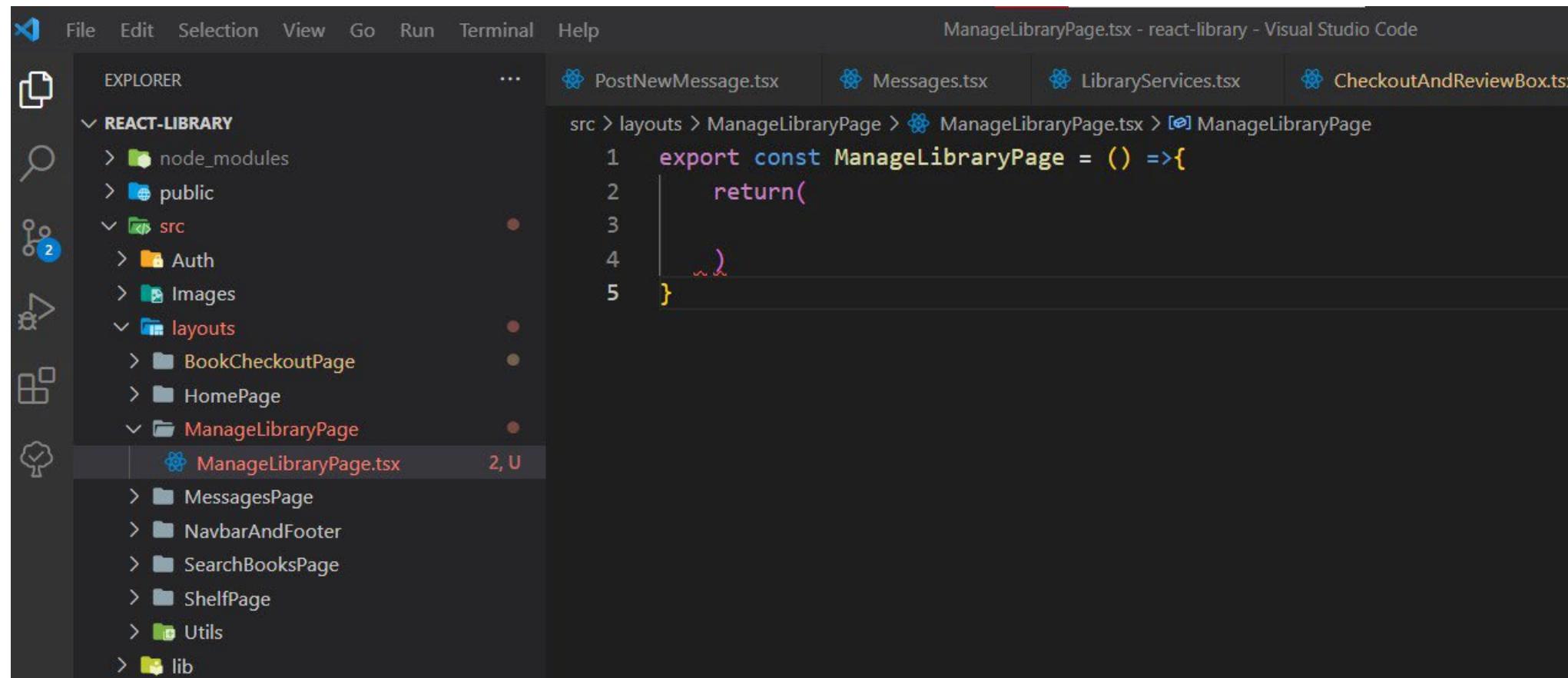


Step2, Right click this ManageLibraryPage, Choose “new file” , create a file named ManageLibraryPage.tsx.



ManageLibraryPage.tsx. created.

Step 3, now the very first thing we want to do, as we ‘ve done with every single new component, is to create export constant function structure.



The screenshot shows a Visual Studio Code interface with a dark theme. The left sidebar is the Explorer, displaying the project structure:

- REACT-LIBRARY
  - node\_modules
  - public
  - src
    - Auth
    - Images
    - layouts
      - BookCheckoutPage
      - HomePage
      - ManageLibraryPage
        - ManageLibraryPage.tsx
        - MessagesPage
        - NavbarAndFooter
        - SearchBooksPage
        - ShelfPage
        - Utils
  - lib

The code editor window shows the file `ManageLibraryPage.tsx` with the following content:

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

## Step 4 , add authState and import {useOkataAuth}.

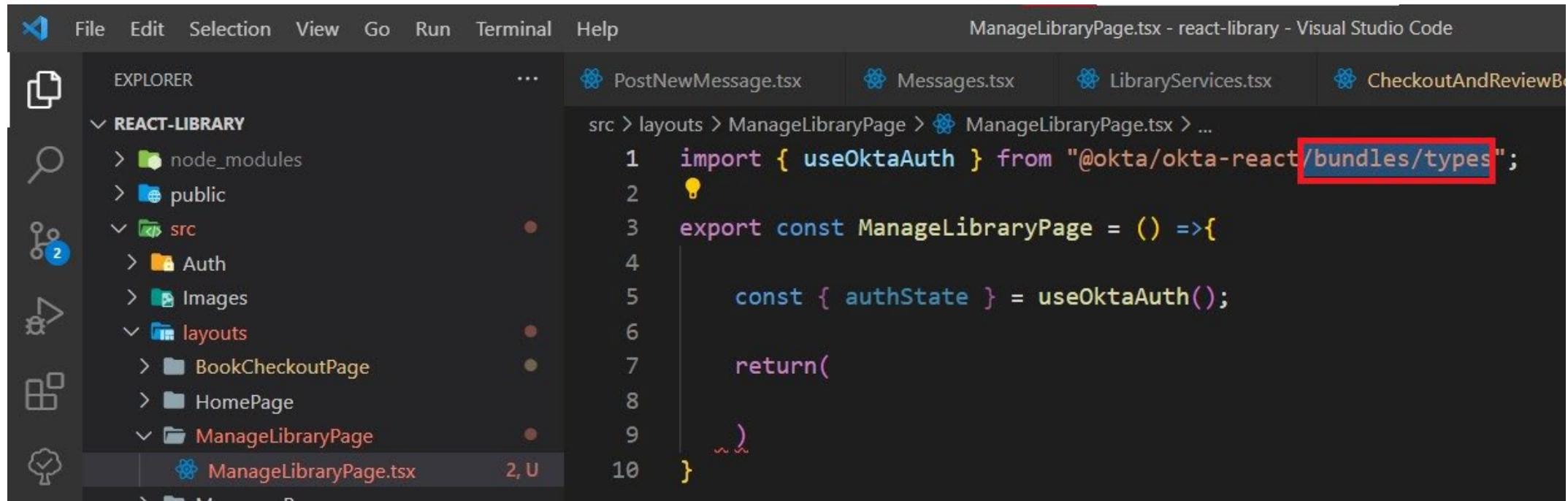
The screenshot shows a Visual Studio Code interface with the following details:

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Explorer:** Shows the project structure under "REACT-LIBRARY". The "ManageLibraryPage" folder is selected, containing "ManageLibraryPage.tsx". Other files like "PostNewMessage.tsx", "Messages.tsx", "LibraryServices.tsx", and "CheckoutAndReviewBox.tsx" are also listed.
- Code Editor:** The file "ManageLibraryPage.tsx" is open. The code is as follows:

```
1 export const ManageLibraryPage = () =>[  
2  
3     const { authState } = useOkataAuth();  
4  
5     return(  
6         ...  
7     )  
8 ]
```

A yellow lightbulb icon is shown over the line "const { authState } = useOkataAuth();", indicating a potential issue or suggestion.
- Quick Fix...:** A tooltip appears with three suggestions:
  - Add import from "@okta(okta-react/bundles/types"
  - Add import from "@okta(okta-react/bundles/types/OktaContext"
  - Add missing function declaration 'useOkataAuth'
- Terminal:** Not visible in the screenshot.

Remove the “/bundle/types” in the import directory for the useOktaAuth. Or else, we will get an compile error when our app starts.



File Edit Selection View Go Run Terminal Help ManageLibraryPage.tsx - react-library - Visual Studio Code

EXPLORER

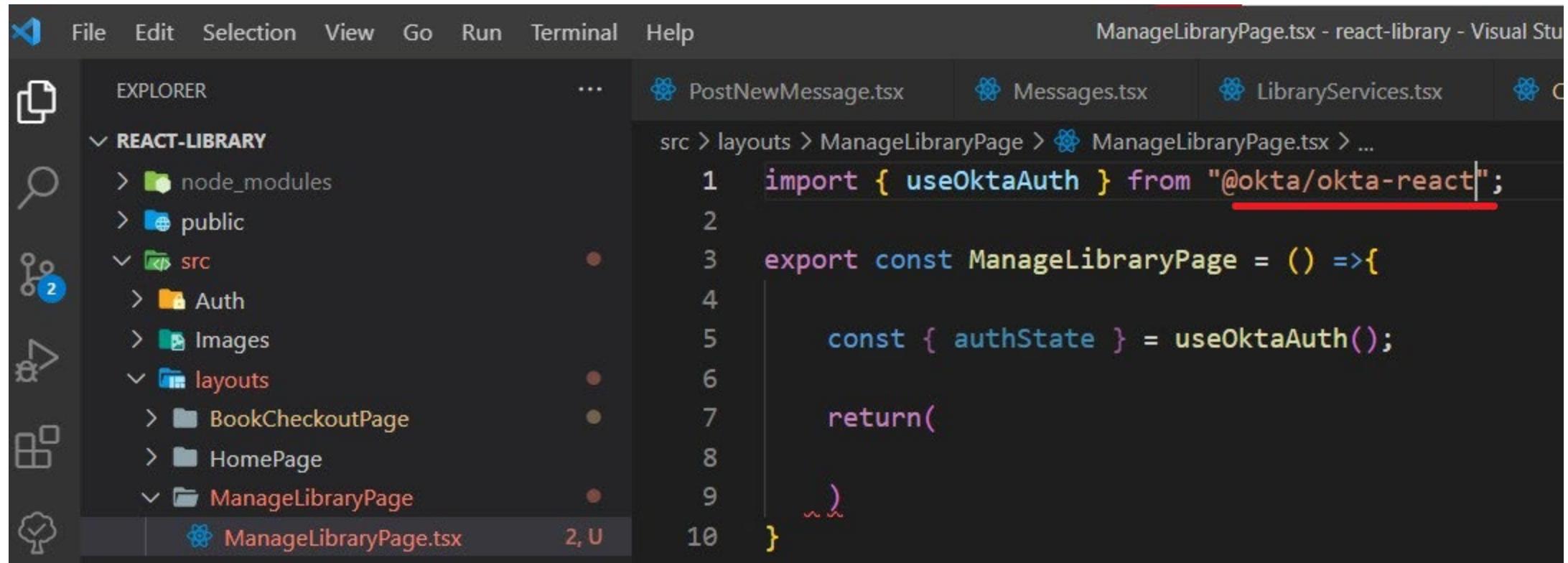
REACT-LIBRARY

- > node\_modules
- > public
- > src
  - > Auth
  - > Images
  - > layouts
    - > BookCheckoutPage
    - > HomePage
    - > ManageLibraryPage
      - ManageLibraryPage.tsx

src > layouts > ManageLibraryPage > ManageLibraryPage.tsx > ...

```
1 import { useOktaAuth } from "@okta/okta-react/bundles/types";
2 
3 export const ManageLibraryPage = () =>{
4 
5     const { authState } = useOktaAuth();
6 
7     return(
8 
9         ...
10    )
```

useOktaAuth import directory should be like this:



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

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** ManageLibraryPage.tsx - react-library - Visual Studio Code.
- Explorer:** Shows a tree view of the project structure:
  - REACT-LIBRARY
    - node\_modules
    - public
    - src
      - Auth
      - Images
      - layouts
        - BookCheckoutPage
        - HomePage
        - ManageLibraryPage
          - ManageLibraryPage.tsx
- Code Editor:** The current file is ManageLibraryPage.tsx, showing the following code:

```
1 import { useOktaAuth } from "@okta(okta-react";
2
3 export const ManageLibraryPage = () =>{
4
5   const { authState } = useOktaAuth();
6
7   return(
8
9     ...
10   )}
```
- Status Bar:** Shows "2, U" indicating two unstaged changes.

Step 5 , add another two useState and import the {useState}.

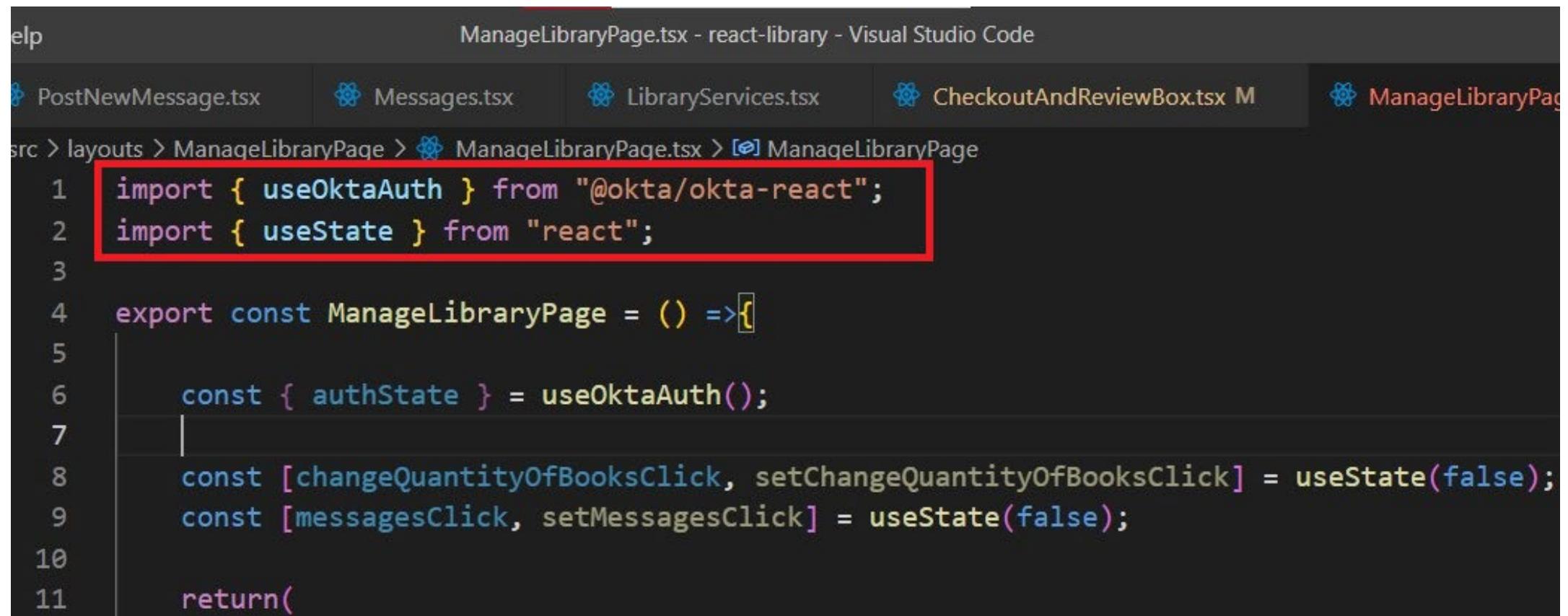
The screenshot shows a Visual Studio Code interface with the title bar "ManageLibraryPage.tsx - react-library - Visual Studio Code". The file tab bar includes "PostNewMessage.tsx", "Messages.tsx", "LibraryServices.tsx", "CheckoutAndReviewBox.tsx", "ManageLibraryPage.tsx 4, U X", and "MessagesPage.tsx". The current file is "ManageLibraryPage.tsx". The code editor displays the following TypeScript code:

```
1 import { useOktaAuth } from "@okta(okta-react)";
2
3 export const ManageLibraryPage = () =>{
4
5     const { authState } = useOktaAuth();
6
7     const [changeQuantityOfBooksClick, setChangeQuantityOfBooksClick] = useState(false);
8     const [messagesClick, setMessagesClick] = useState(false);
9
10    return(
11
12        ...
13    )
}
```

A red box highlights the import statement "import { useOktaAuth } from "@okta(okta-react)";". A red box also highlights the "useState" call in the code. A "Quick Fix..." context menu is open at the bottom right, with the "Add import from "react"" option highlighted.

- Quick Fix...
- Add import from "react"
- Add all missing imports
- Add missing function declarations
- Add all missing function declarations

Until now, the import directory in the top should be as below:



The screenshot shows a Visual Studio Code interface with the title bar "ManageLibraryPage.tsx - react-library - Visual Studio Code". Below the title bar, there is a tab bar with several files: PostNewMessage.tsx, Messages.tsx, LibraryServices.tsx, CheckoutAndReviewBox.tsx, and ManageLibraryPage.tsx. The ManageLibraryPage.tsx tab is active. The code editor displays the following TypeScript code:

```
1 import { useOktaAuth } from "@okta(okta-react";
2 import { useState } from "react";
3
4 export const ManageLibraryPage = () => {
5
6     const { authState } = useOktaAuth();
7
8     const [changeQuantityOfBooksClick, setChangeQuantityOfBooksClick] = useState(false);
9     const [messagesClick, setMessagesClick] = useState(false);
10
11     return(

```

The first two lines of the code, which are the imports, are highlighted with a red rectangular box.

Step 6, fill in Return part with html and CSS code.

layouts > ManageLibraryPage > ManageLibraryPage.tsx > ManageLibraryPage

```
return(  
  <div className='container'>  
    <div className='mt-5'>  
      <h3>Manage Library</h3>  
      <nav>  
        {/* three tabs */}  
        <div className='nav nav-tabs' id='nav-tab' role='tablist'>  
          <button className='nav-link active' id='nav-add-book-tab' data-bs-toggle='tab'  
            data-bs-target='#nav-add-book' type='button' role='tab' aria-controls='nav-add-book'  
            aria-selected='false'>  
            > Add new book  
          </button>  
          <button className='nav-link' id='nav-quantity-tab' data-bs-toggle='tab'  
            data-bs-target='#nav-quantity' type='button' role='tab' aria-controls='nav-quantity'  
            aria-selected='true'>  
            > Change quantity  
          </button>  
          <button className='nav-link' id='nav-messages-tab' data-bs-toggle='tab'  
            data-bs-target='#nav-messages' type='button' role='tab' aria-controls='nav-messages'  
            aria-selected='false'>  
            > Messages  
          </button>  
        </div>  
      </nav>  
    </div>  
  </div>
```



The image displays three tabs from a web application interface, each with a red box highlighting a specific element, and a code editor below showing the corresponding HTML/JSX code.

**Manage Library**

- Add new book
- Change quantity
- Add New Book**

**Manage Library**

- Add new book
- Change quantity
- Change Quantity Of Books**

**Manage Library**

- Add new book
- Change quantity
- Admin Messages**

**Code Editor (ManageLibraryPage.js)**

```
return(  
  <div className='container'>  
    <div className='mt-5'>  
      <h3>Manage Library</h3>  
      <nav>...</nav>  
      /* content showed when the tab is clicked */  
      <div className='tab-content' id='nav-tabContent'>  
        <div className='tab-pane fade show active' id='nav-add-book' role='tabpanel'  
          aria-labelledby='nav-add-book-tab'>  
            <b>Add New Book</b>  
          </div>  
        <div className='tab-pane fade' id='nav-quantity' role='tabpanel' aria-labelledby='nav-quantity-tab'>  
          <b>Change Quantity Of Books</b>  
        </div>  
        <div className='tab-pane fade' id='nav-messages' role='tabpanel' aria-labelledby='nav-messages-tab'>  
          <b>Admin Messages</b>  
        </div>  
      </div>  
    </div>  
  </div>
```

---

---

## **2. REACT SECURE ROUTES FOR MANAGELIBRARY**



let's hop into our App.tsx.

And let's create a new secure route for our new ManageLibraryPage.

Right click red tilde, click on "Quick Fix".

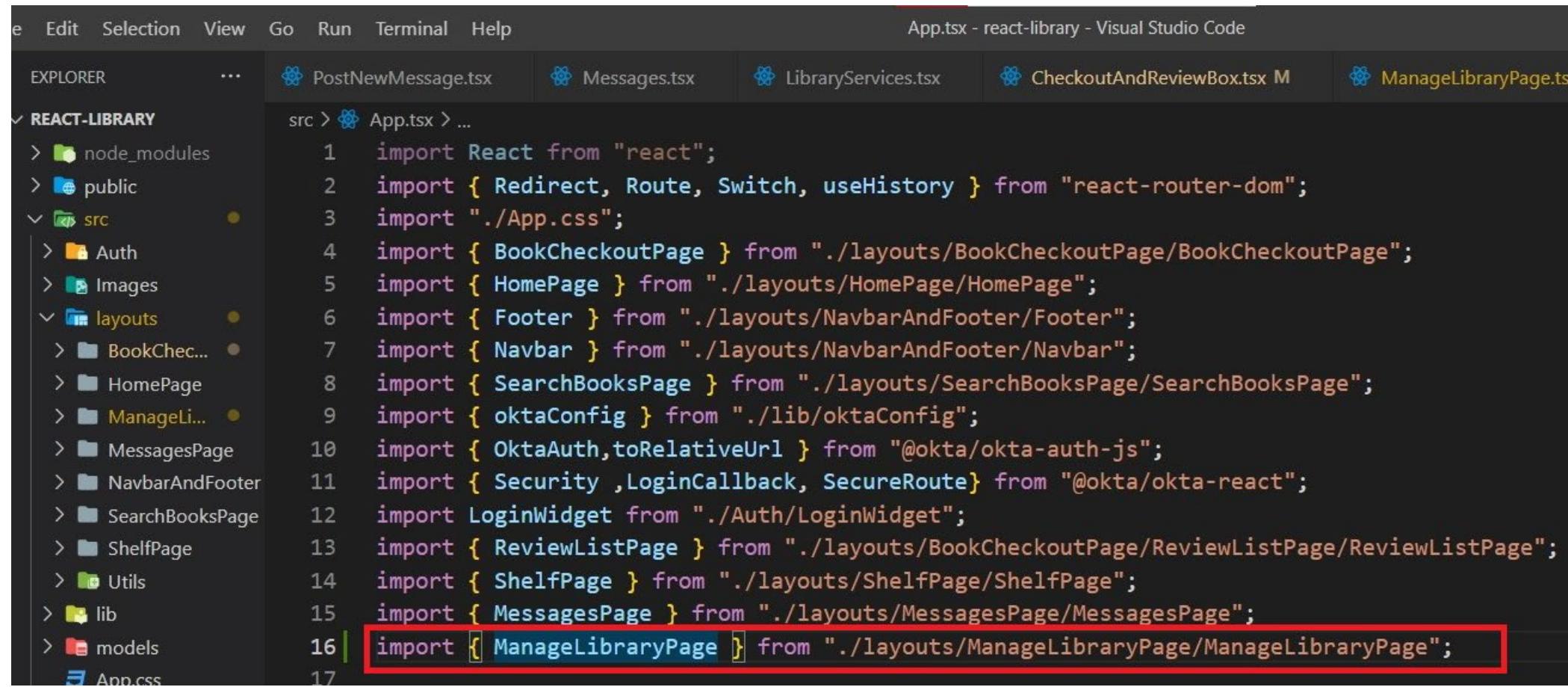
Choose "add import".

The screenshot shows a code editor with the file `App.tsx` open. The sidebar on the left lists various files: HomePage, ManageLi..., MessagesPage, NavbarAndFooter, SearchBooksPage, ShelfPage, Utils, lib, models, App.css, App.tsx (which is the active file), index.css, index.tsx, react-app-env.d.ts, .gitignore, and package-lock.json. In the main editor area, there is a red squiggle under the line `<SecureRoute path='/admin'> <ManageLibraryPage/> </SecureRoute>`. A context menu is open at the end of this line, with the title "Quick Fix..". The menu items are: "Add import from './layouts/ManageLibraryPage/ManageLibraryPage'", "Disable react/jsx-no-undef for this line", "Disable react/jsx-no-undef for the entire file", and "Show documentation for react/jsx-no-undef".

```
> HomePage
> ManageLi... ●
> MessagesPage
> NavbarAndFooter
> SearchBooksPage
> ShelfPage
> Utils
> lib
> models
> App.css
> App.tsx 2, M
> index.css
> index.tsx
> react-app-env.d.ts
> .gitignore
> package-lock.json

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  <SecureRoute path='/admin'> <ManageLibraryPage/> </SecureRoute>
58  </Switch>
59  </div>
60  <Footer></Footer>
61  </Security>
62  </div>
63  ;
```

ManageLibraryPage is imported.

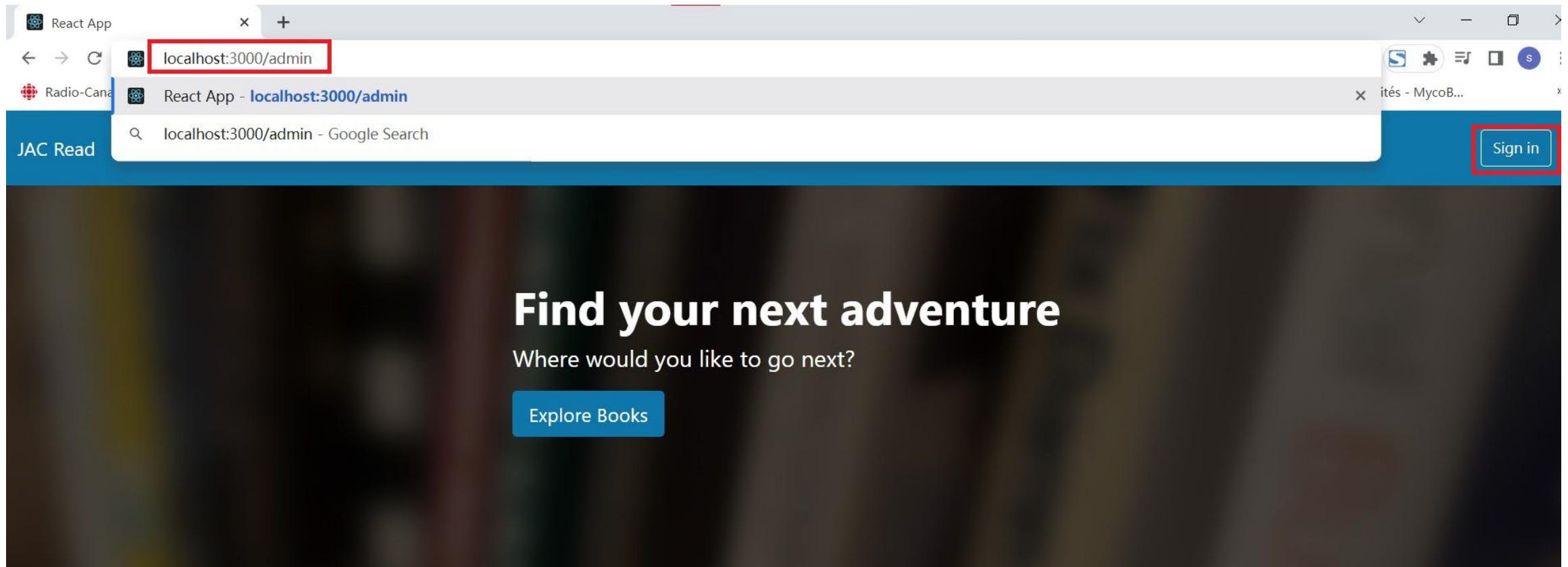


The screenshot shows the Visual Studio Code interface with the following details:

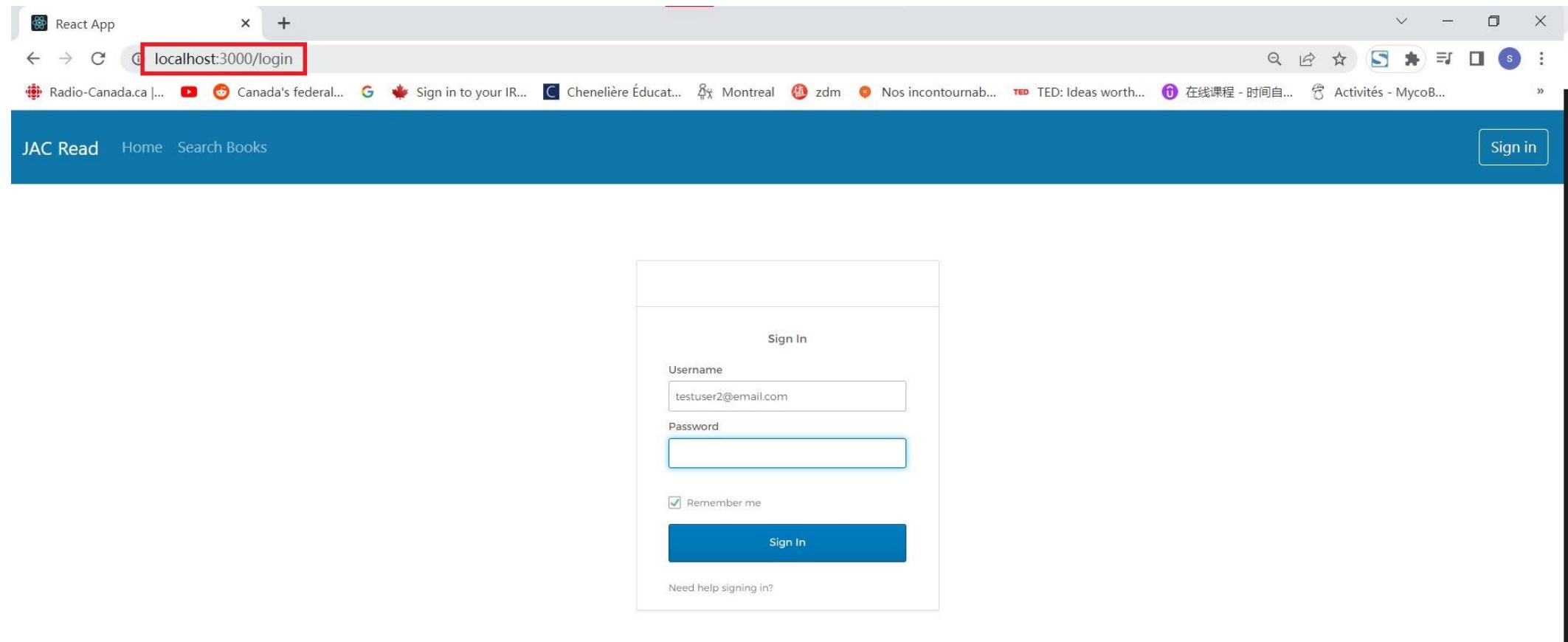
- Menu Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** App.tsx - react-library - Visual Studio Code
- Explorer Bar (Left):** Shows the project structure under 'REACT-LIBRARY'. The 'src' folder is expanded, showing subfolders like 'Auth', 'Images', 'layouts', 'BookChec...', 'HomePage', 'ManageLi...', 'MessagesPage', 'NavbarAndFooter', 'SearchBooksPage', 'ShelfPage', 'Utils', 'lib', 'models', and 'App.css'.
- Code Editor (Right):** Displays the content of 'App.tsx'. The code includes imports for various components and pages. The line 'import { ManageLibraryPage } from './layouts/ManageLibraryPage/ManageLibraryPage';' is highlighted with a red rectangular selection.

```
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 import { ManageLibraryPage } from './layouts/ManageLibraryPage/ManageLibraryPage';
17
```

With the log out status, Visit <http://localhost:3000/admin> :



It will redirect us to the sign in page, because we add a secure route to this page.



once logged in, it will show the ManageLibraryPage.

The screenshot shows a web browser window titled "React App" with the URL "localhost:3000/admin" highlighted by a red box. The browser's address bar also displays "localhost:3000/admin". The page content is titled "Manage Library" and includes tabs for "Add new book", "Change quantity", and "Messages", with "Messages" being the active tab. Below the tabs, there is a link "Admin Messages". On the right side of the header, there is a "Logout" button also highlighted by a red box. The footer of the page contains the copyright notice "© JAC Library App, Inc" and links for "Home" and "Search Books".

---

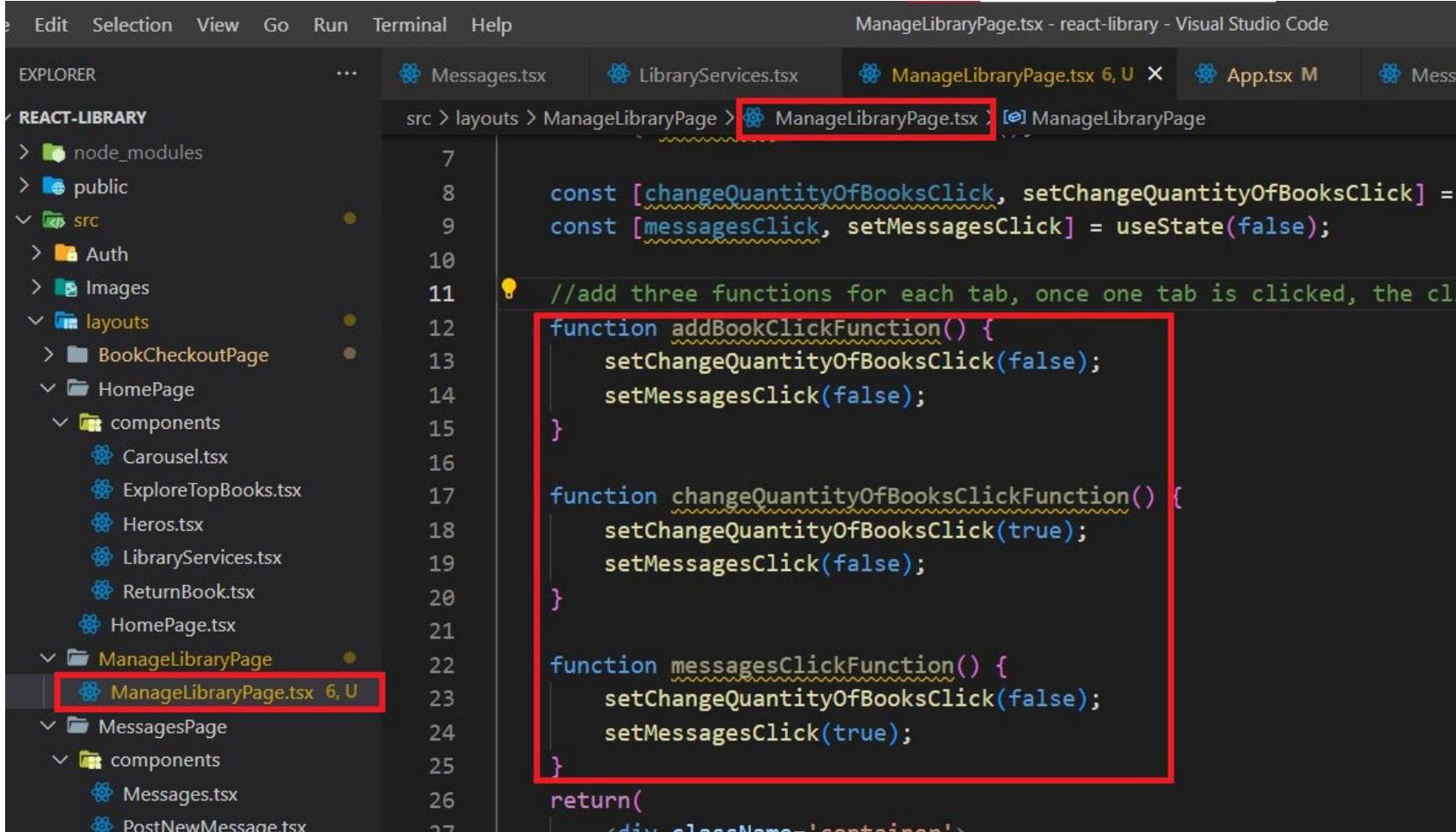
---

---

### **3. REACT STATE FOR MANAGELIBRARY**



Step1, add three functions for each tab, once one tab is clicked, the click state of the other two tabs should be false.



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

- File Explorer (Left):** Shows the project structure under "REACT-LIBRARY". The "ManageLibraryPage" folder is expanded, showing "ManageLibraryPage.tsx" (highlighted with a red box).
- Code Editor (Right):** The file "ManageLibraryPage.tsx" is open. The code includes:

```
const [changeQuantityOfBooksClick, setChangeQuantityOfBooksClick] = useState(false);
const [messagesClick, setMessagesClick] = useState(false);

//add three functions for each tab, once one tab is clicked, the cli
function addBookClickFunction() {
    setChangeQuantityOfBooksClick(false);
    setMessagesClick(false);
}

function changeQuantityOfBooksClickFunction() {
    setChangeQuantityOfBooksClick(true);
    setMessagesClick(false);
}

function messagesClickFunction() {
    setChangeQuantityOfBooksClick(false);
    setMessagesClick(true);
}

return(
    <div className='container'>
```
- Top Bar:** Shows "Edit Selection View Go Run Terminal Help" and the current file "ManageLibraryPage.tsx - react-library - Visual Studio Code".
- Status Bar:** Shows the path "src > layouts > ManageLibraryPage > ManageLibraryPage.tsx" and the status "[ManageLibraryPage]".

```
function messagesClickFunction() {
  setChangeQuantityOfBooksClick(false);
  setMessagesClick(true);
}

return(
  <div className='container'>
    <div className='mt-5'>
      <h3>Manage Library</h3>
      <nav>
        {/* three tabs */}
        <div className='nav nav-tabs' id='nav-tab' role='tablist'>
          <button onClick={addBookClickFunction} className='nav-link active' id='nav-add-book-tab' data-bs-target='#nav-add-book' type='button' role='tab' aria-controls='nav-add-book' aria-selected='false'>
            Add new book
          </button>
          <button onClick={changeQuantityOfBooksClickFunction} className='nav-link' id='nav-quantity-tab' data-bs-target='#nav-quantity' type='button' role='tab' aria-controls='nav-quantity' aria-selected='true'>
            Change quantity
          </button>
          <button onClick={messagesClickFunction} className='nav-link' id='nav-messages-tab' data-bs-target='#nav-messages' type='button' role='tab' aria-controls='nav-messages' aria-selected='false'>
            Messages
          </button>
        </div>
      </nav>
    </div>
  </div>
)
```

Step 2, add onClick listener to each tab.

ManageLibraryPage.tsx - react-library - Visual Studio Code

Services.tsx ManageLibraryPage.tsx 1, U X App.tsx M MessagesPage.tsx

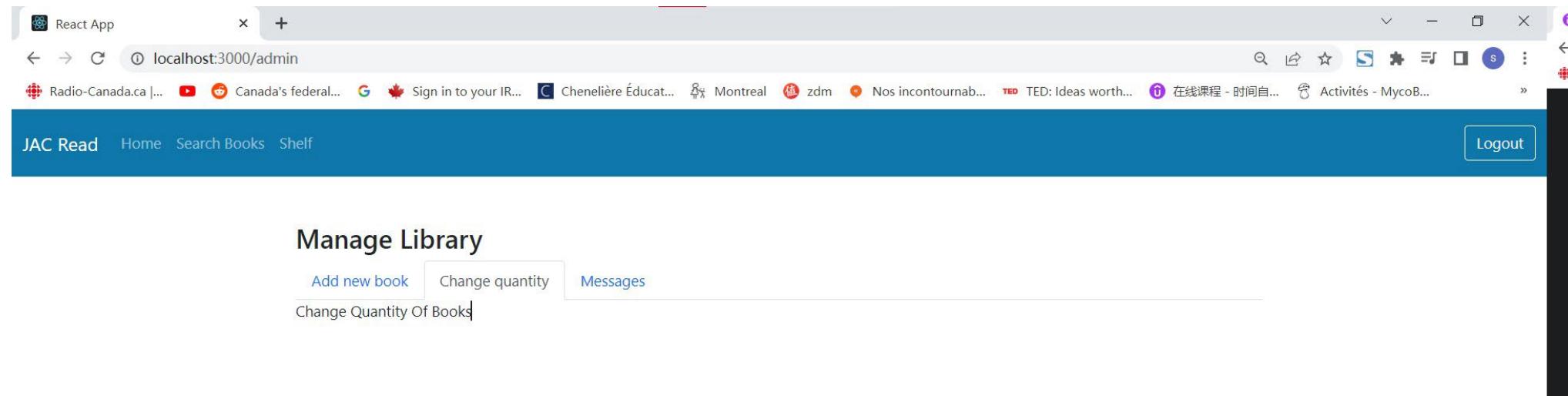
ge > ManageLibraryPage.tsx > [o] ManageLibraryPage

```
</button>
<button onClick={messagesClickFunction} className='nav-link' id='nav-messages' data-bs-target='#nav-messages' type='button' role='tab' aria-controls='nav-messages' aria-selected='false'>
    >
        Messages
    </button>
</div>
</nav>
/* content showed when the tab is clicked */
<div className='tab-content' id='nav-tabContent'>
    <div className='tab-pane fade show active' id='nav-add-book' role='tabpanel' aria-labelledby='nav-add-book-tab'>
        Add New Book
    </div>
    <div className='tab-pane fade' id='nav-quantity' role='tabpanel' aria-labelledby='nav-quantity-tab'>
        {changeQuantityOfBooksClick ? <>Change Quantity Of Books</> : <></>}
    </div>
    <div className='tab-pane fade' id='nav-messages' role='tabpanel' aria-labelledby='nav-messages-tab'>
        {messagesClick ? <>Admin Messages </> : <></>}
    </div>
</div>
```

Step 3, add click state judge to the tab content.

Open up our React app, refresh the page, and we can see that the tab still swaps between the two.

The only difference is we're calling different pieces of state now, and state is changing, which means when we start adding elements to these tabs, it'll call the `useEffect` each time.



---

---

## **4. REACT REDIRECT IMPLEMENTATION FOR NON-AMIN**



# Role-based authentication

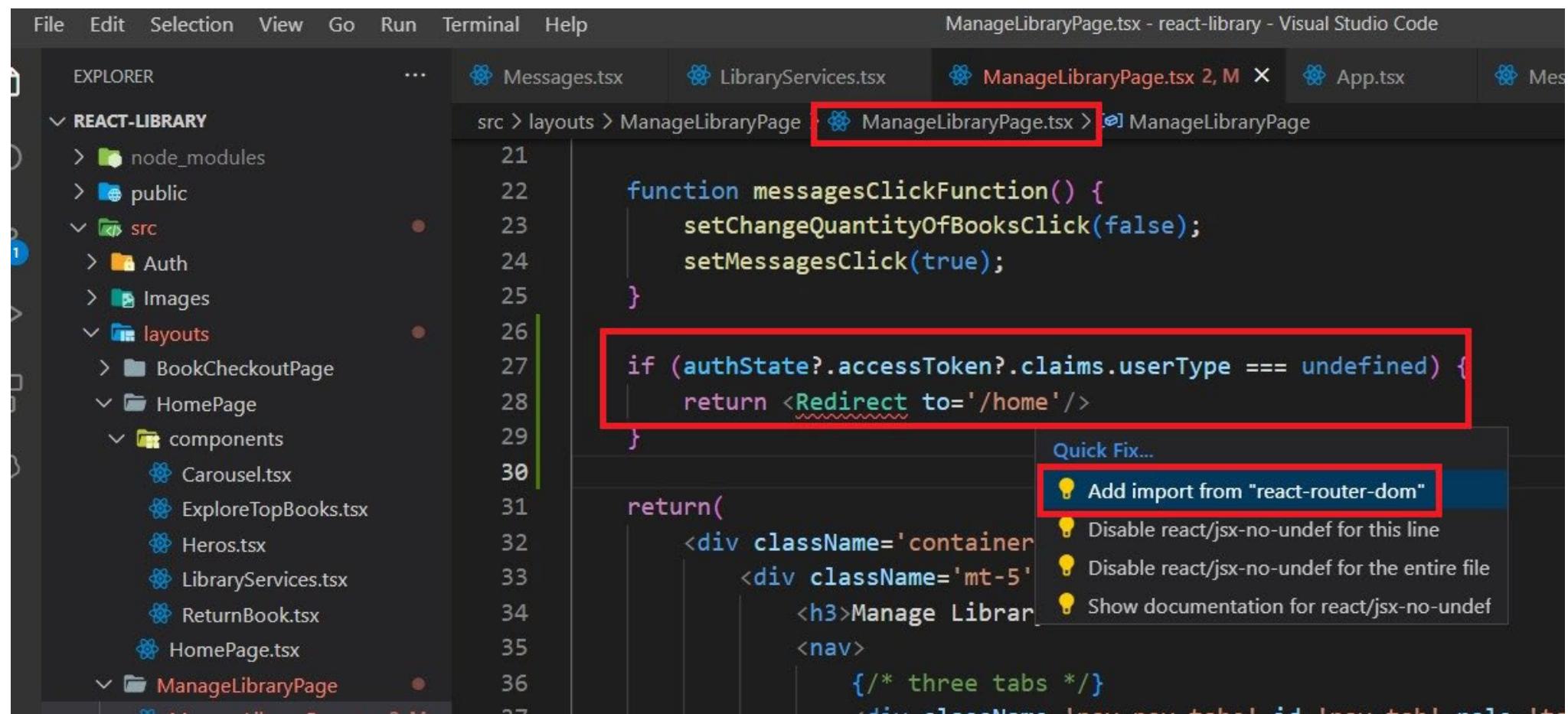
- We don't want anybody who's currently authenticated to be able to go to this Manage library page.
- And we currently have that set in our secure route where, hey, a user needs to be authenticated before they come to our component.
- But besides this authentication, we want to make sure that we're checking the user type or the role . That's to say only the role of admin can visit this page.

## Solution:

- add a new conditional statement on our manage library page that would check the user type of the user to see if they're an admin or not.
- Use okta to define the user type

Step1, Go to ManageLibraryPage.tsx, add a piece of code to judge if the user type is undefined.  
if so, redirect to home page.

Click on the “quick fix” to import “react-router-dom” to fix the error when we input “Redirect”.



File Edit Selection View Go Run Terminal Help ManageLibraryPage.tsx - react-library - Visual Studio Code

EXPLORER ... Messages.tsx LibraryServices.tsx ManageLibraryPage.tsx 2, M App.tsx Messages

src > layouts > ManageLibraryPage > ManageLibraryPage.tsx > ManageLibraryPage

```
function messagesClickFunction() {
    setChangeQuantityOfBooksClick(false);
    setMessagesClick(true);
}

if (authState?.accessToken?.claims.userType === undefined) {
    return <Redirect to='/home' />
}

return(
    <div className='container'>
        <div className='mt-5'>
            <h3>Manage Library</h3>
            <nav>
                /* three tabs */
            </nav>
        </div>
    </div>
)
```

Quick Fix... Add import from "react-router-dom"

- Disable react/jsx-no-undef for this line
- Disable react/jsx-no-undef for the entire file
- Show documentation for react/jsx-no-undef

---

---

---

## 5. REACT CREATE ADMIN USER



Step1, log into our okta account, choose “Directory”, “ People”, then click on “ add person”.

The screenshot shows the Okta web interface. On the left, a sidebar menu includes 'Dashboard', 'Directory' (which is selected and highlighted with a red box), 'People' (also highlighted with a red box), 'Groups', 'Profile Editor', 'Directory Integrations', 'Self-Service Registration', 'Profile Sources', and 'Customizations'. The main content area is titled 'People' and features a search bar at the top. Below the search bar are four buttons: '+ Add person' (highlighted with a red box), 'Reset passwords', 'Reset multifactor', and 'More actions'. A table below lists three users: 'Test User' with primary email 'testuserjac@email.com' and status 'Active'; and 'Test User2' with primary email 'testuser2@email.com' and status 'Active'. A search bar at the bottom of the table allows users to search by first name, primary email, or username. The status filter dropdown shows 'All'.

Person & username	Primary email	Status
Test User testuserjac@email.com	testuserjac@email.com	Active
Test User2 testuser2@email.com	testuser2@email.com	Active

okta

Dashboard

Directory

People

Groups

Profile Editor

Directory Integrations

Self-Service Registration

Profile Sources

Customizations

Applications

Security

Workflow

Reports

## Add Person

User type

First name

Last name

Username

Primary email

Groups (optional)  
You haven't added any groups

Password

\*\*\*\*\*

User must change password on first login

**Save** **Save and Add Another** **Cancel**

Showing 3 of 3

Status

Active

Active

Password reset

The screenshot shows the 'Add Person' form in the Okta interface. The 'User type' dropdown is set to 'User'. The 'Password' dropdown is set to 'Set by admin'. A checkbox for 'User must change password on first login' is checked. The 'Save' button at the bottom left is highlighted with a blue box.

Make sure you input the same as the picture. Then click on “save” button.

Our admin user will appear in the table.

The screenshot shows the Okta People page. The left sidebar has 'People' selected under 'Directory'. The main area is titled 'People' with buttons for 'Add person', 'Reset passwords', 'Reset multifactor', and 'More actions'. A search bar at the top right contains the placeholder 'Search...'. Below it is an 'Advanced search' section with a status dropdown set to 'All'. A message indicates 'Showing 4 of 4'. The table lists four users:

Person & username	Primary email	Status
Admin User adminuser@email.com	adminuser@email.com	Active
Test User testuserjac@email.com	testuserjac@email.com	Active
Test User2 testuser2@email.com	testuser2@email.com	Active

The first row ('Admin User') is highlighted with a red box.

Step 2, add attribute.  
let's go into directory, profile editor. Click on our app.

The screenshot shows the Okta Directory Profile Editor interface. On the left, a sidebar menu lists various sections: Dashboard, Directory (expanded), People, Groups, Profile Editor (highlighted with a red box), Directory Integrations, Self-Service Registration, Profile Sources, Customizations, Applications, and Security. The main content area has a search bar at the top. A central panel displays a message about profiles and provisioning support, with a 'Go to Documentation' link. Below this is a 'Users' tab selected, with a 'Groups' tab also present. The 'Users' list table has columns for Filters, Profile, and Type. It shows three users: 'okta' (User (default) user, Okta), 'Developer Registration SSO User' (Developer Registration SSO User, Identity Provider, Mappings), and 'My library App User' (My library App User, Application, Mappings). The 'My library App User' row is highlighted with a red box.

Filters	Profile	Type
All	okta User (default) user	Okta
Okta	Developer Registration SSO User oidc_idp	Identity Provider <a href="#">Mappings</a>
Apps	My library App User oidc_client	Application <a href="#">Mappings</a>
Directories		
Identity Providers		

Click on “Add Attribute”.

The screenshot shows the Profile Editor interface for a user named "My library App User". The left sidebar is collapsed, and the main area displays the user's profile information and attributes.

**User Profile:**

- Display name: My library App User
- Description: (empty)
- Variable name: oidc\_client

**Attributes:**

A table lists attributes with the following columns: FILTERS, Display Name, Variable Name, Data type, and Attribute Type.

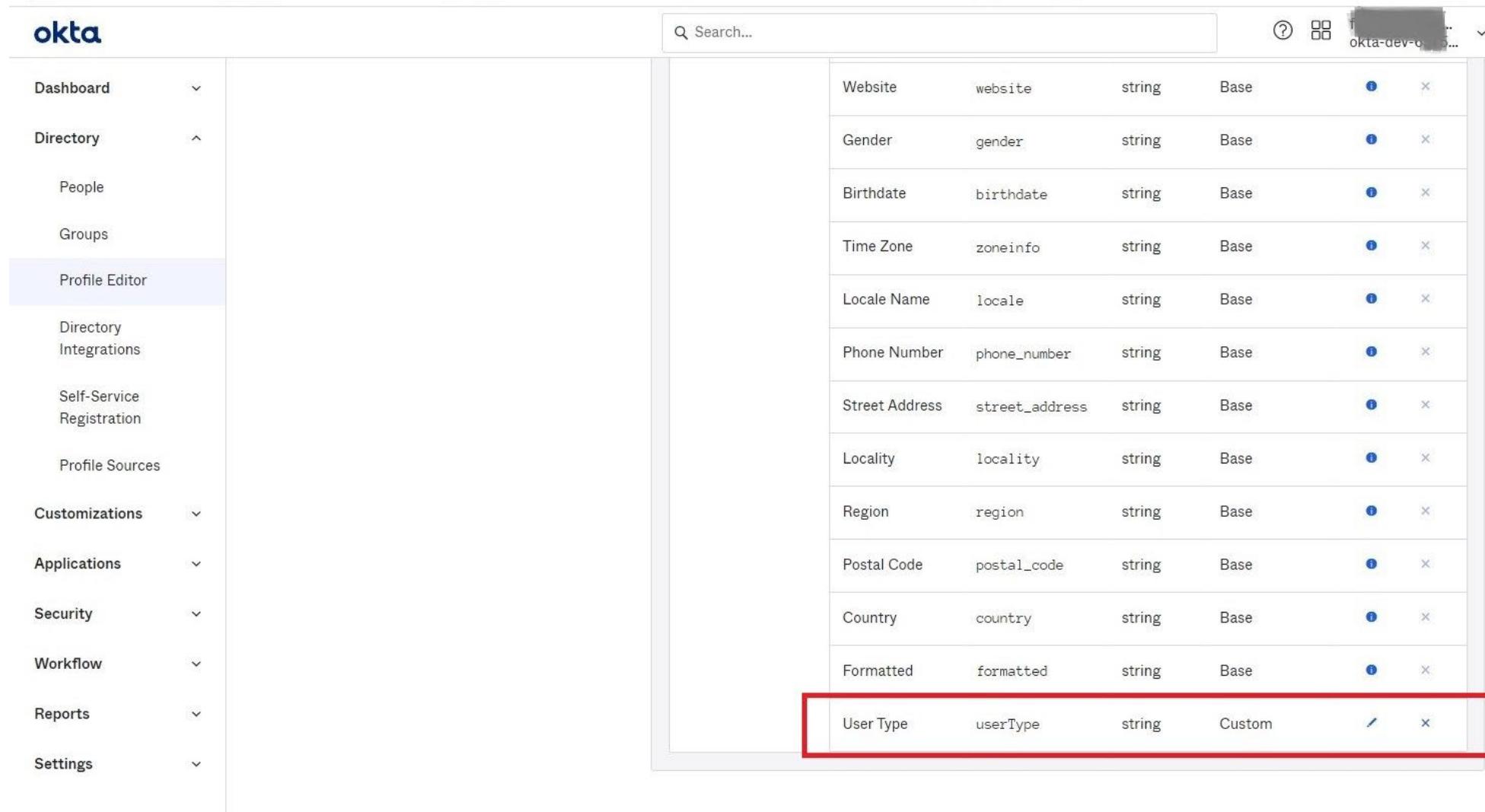
FILTERS	Display Name	Variable Name	Data type	Attribute Type
All	Username	userName	string	Base
Base	Name	name	string	Base
Custom				

**Action:** A red box highlights the "+ Add Attribute" button in the Attributes section.

The screenshot shows the Okta interface with a dark theme. On the left is a sidebar with various navigation items like Dashboard, Directory, Profile Editor, and Applications. The main area is titled "Add Attribute". It contains fields for Data type (set to "string"), Display name (set to "User Type"), Variable name (set to "userType"), Description (empty), and Enum (checkbox unchecked). Under Attribute length, it says "Between" with "min" and "max" fields. At the bottom, there are checkboxes for "Attribute required" (unchecked) and "Scope" (unchecked). Below these are three buttons: "Save" (highlighted with a red box), "Save and Add Another", and "Cancel".

Make sure you input the same as the picture.  
Then click on “save” button.

Then, scroll down the page, you will see our custom user Type is there.



The screenshot shows the Okta Profile Editor interface. On the left, a sidebar lists various sections: Dashboard, Directory (expanded), People, Groups, Profile Editor (selected and highlighted in blue), Directory Integrations, Self-Service Registration, Profile Sources, Customizations (expanded), Applications (expanded), Security (expanded), Workflow (expanded), Reports (expanded), and Settings (expanded). The main area displays a table of custom user attributes. A red box highlights the last row of the table, which contains the custom attribute 'User Type'.

	Name	Type	Format	Base		
Website	website	string	Base			
Gender	gender	string	Base			
Birthdate	birthdate	string	Base			
Time Zone	zoneinfo	string	Base			
Locale Name	locale	string	Base			
Phone Number	phone_number	string	Base			
Street Address	street_address	string	Base			
Locality	locality	string	Base			
Region	region	string	Base			
Postal Code	postal_code	string	Base			
Country	country	string	Base			
Formatted	formatted	string	Base			
User Type	userType	string	Custom			

Step 3, mappings.

Now let's go back to our profile editor.

In my library app User Let's go ahead and click mappings.

The screenshot shows the Okta Profile Editor interface. On the left, there is a sidebar with the following navigation items:

- Dashboard
- Directory
- People
- Groups
- Profile Editor** (selected)
- Directory Integrations
- Self-Service Registration
- Profile Sources

The main area is titled "Users" and contains a table of users. The table has columns for "Filters", "Profile", and "Type". There are four rows in the table:

Filters	Profile	Type
All	okta User (default)	Okta
Okta	user	
Apps	Developer Registration SSO User oidc_idp	Identity Provider
Directories		
Identity Providers	My library App User oidc_client	Application

A blue button labeled "+ Create Okta User Type" is located at the top right of the user list. To the right of the last two rows, there are "Mappings" buttons. The bottom row's "Mappings" button is highlighted with a red box.

We will see this page.

okta

Dashboard

Directory

People

Groups

Profile Editor

Directory Integrations

Self-Service Registration

Profile Sources:

Customizations

Applications

## My library App User Profile Mappings

My library App to Okta U... Okta User to My library ...

Okta User User Profile user	My library App User Profile appuser
Username is set by My library App · Override with mapping	userName string
user.displayName	name string
user.nickName	nickname string
user.firstName	given_name string

The screenshot shows the Okta Profile Editor interface. On the left is a dark sidebar with navigation links: Dashboard, Directory (expanded), People, Groups, Profile Editor, Directory Integrations, Self-Service Registration, Profile Sources, Customizations (expanded), and Applications. The main area is a configuration dialog for creating a new attribute.

The dialog has two input fields at the top:

- The first field contains the placeholder "Choose an attribute or enter an expression..."
- The second field contains the value "formatted" with a data type of "string".

Below these fields is another input field with the placeholder "Choose an attribute or enter an expression...". This field contains the value "userType" highlighted with a red box. To the right of this field is another "string" data type indicator.

Further down, there is a list of attributes with their types and descriptions:

Name	Type	Description
zipcode	string	Zip code
countryCode	country code	Country code
postalAddress	string	Postal Address
preferredLanguage	language code	Preferred language
locale	locale	Locale
timezone	timezone	Time zone
<b>userType</b>	<b>string</b>	<b>User type</b>
employeeNumber	string	Employee number
costCenter	string	Cost center
organization	string	Organization

At the bottom of the list, there is a link to "Expression Language Reference".

So let's go all the way to the bottom where we can see our new user type. click the little arrow and scroll down until you can see user type. type.

Click on the button beside the yellow arrow:

The screenshot shows the Okta Profile Editor interface. On the left is a dark sidebar with navigation links: Dashboard, Directory (expanded), People, Groups, Profile Editor, Directory, and Integrations. The main area displays a mapping configuration for the attribute `user.userType`. The attribute is mapped to the output `userType`, which is of type string. The mapping is labeled as `formatted`. A yellow arrow icon is visible next to the mapping rule, and it is highlighted with a red box. At the bottom of the mapping section, there are buttons for `Save Mappings` and `Cancel`. Below the mapping section, there is a preview section with a `Preview` button and a text input field containing the placeholder `Enter an Okta user to preview th...`.

Choose this “Apply mapping on user create and update”.

The screenshot shows the Okta Admin Console with the sidebar menu open. The 'People' section is selected. On the right, the 'Attribute Mappings' tab is active, showing a configuration for the 'userType' attribute.

**Attribute Mapping Configuration:**

- Source:** Choose an attribute or enter an expression... (dropdown)
- Mapping:** user.userType (text input)
- Target:** userType (dropdown)
- Options:**
  - Apply mapping on user create and update** (selected, highlighted with a red border)
  - Apply mapping on user create only**
  - Do not map**

**Buttons:**

- Preview (button)
- Enter an Okta user to preview the mapping (text input)
- Mappings (blue button)
- Cancel (button)

Make sure everything is the same as the red box, then click on “Save Mapping”.

The screenshot shows the Okta Profile Editor interface. On the left, there's a sidebar with navigation links: Dashboard, Directory (expanded), People, Groups, Profile Editor, Directory, and Integrations. The main area displays a mapping configuration. At the top, there's a dropdown menu with a placeholder "Choose an attribute or enter an expression...". Below it is a mapping row with a red border around the source attribute "user.userType". This row consists of three columns: the source attribute "user.userType", a green arrow pointing right, and the target attribute "userType". To the right of the target attribute are the data types "string" and "formatted". At the bottom of the mapping section, there are two buttons: "Save Mappings" (highlighted with a green box) and "Cancel".

Then, click on the “Apply updates now”.

The screenshot shows the Okta Profile Editor interface. On the left is a dark sidebar with navigation links: Dashboard, Directory (expanded), People, Groups, Profile Editor, Directory Integrations, Self-Service Registration, Profile Sources, Customizations (expanded), and Applications. The main area displays two attribute mappings:

From	To	Type
Choose an attribute or enter an expression...	-/→	formatted string
user.userType	→	userType string

Below the mappings, a green banner says "Mappings saved!". A message asks, "Do you want to apply these mappings to all users with this profile?". Two buttons are present: a red-bordered "Apply updates now" button and a "Don't apply updates" button.

Click on “mapping” again.

The screenshot shows the Okta user interface. The left sidebar is collapsed, showing the following navigation items:

- Dashboard
- Directory ▾
  - People
  - Groups
- Profile Editor
- Directory Integrations
- Self-Service Registration
- Profile Sources

The main area is titled "Users" and displays a table of users. The table has three columns: "Filters", "Profile", and "Type". The "Filters" column contains buttons for "All", "Okta", "Apps", "Directories", and "Identity Providers". The "Profile" column lists user profiles with icons: "okta" (User default), "OpenID" (Developer Registration SSO User), and a gear icon (My library App User). The "Type" column indicates the source: "Okta" for the first two, and "Identity Provider" for the third. To the right of the "Type" column, there are "Mappings" buttons for each row. A blue button at the top right of the table area says "+ Create Okta User Type".

This time choose “My library App to Okta User Profile”.

The screenshot shows the Okta User Profile Mappings configuration screen. On the left, there's a sidebar with navigation links: Dashboard, Directory, People, Groups, Profile Editor, Directory Integrations, Self-Service Registration, Profile Sources, Customizations, and Applications. The 'Customizations' section is expanded, showing 'My library App to Okta User Profile' and 'Okta User to My library ...'. The main area is titled 'My library App User Profile Mappings'. It shows two columns: 'Okta User User Profile' and 'My library App User Profile'. The 'Okta User User Profile' column contains fields: 'user.displayName', 'user.nickName', and 'user.firstName'. The 'My library App User Profile' column contains fields: 'userName', 'name', 'nickname', and 'given\_name'. Arrows indicate the mapping from Okta fields to My library App fields. A note says 'Username is set by My library App · Override with mapping'. The 'My library App to Okta User Profile' link in the sidebar is highlighted with a red box.

Okta User User Profile	My library App User Profile
user	appuser
user.displayName	name
user.nickName	nickname
user.firstName	given_name

You will see the tab we have chosen become blue. Scroll down until we find user Type.

The screenshot shows the Okta Developer Platform interface. The left sidebar has a dark theme with the Okta logo at the top. The main content area is titled "My library App User Profile Mappings". A red box highlights the "My library App to Okta ..." tab. Below it is another tab labeled "Okta User to My library ...".

The mapping table lists three fields:

My library App User Profile appuser	Okta User User Profile user	Type
appuser.userName	login	string
appuser.given_name	firstName	string
appuser.family_name	lastName	string

Below the table, there is a note: "Use default username setting for Okta user".

The same as the last step, choose as below:

The screenshot shows the Okta Profile Editor interface. On the left is a sidebar with navigation links: Dashboard, Directory (expanded), People, Groups, Profile Editor, Directory Integrations, Self-Service Registration, Profile Sources, Customizations (expanded), and Applications. The main area is titled "Choose an attribute or enter an expression..." and contains three dropdown menus. The first dropdown has two options: "timezone" and "userType". The second dropdown also has two options: "timezone" and "userType". The third dropdown has several options: "street\_address", "locality", "region", "postal\_code", "country", "formatted", and "userType". The "userType" option in the third dropdown is highlighted with a red border. The "userType" entry in the third dropdown is also highlighted with a red border.

Attribute	Type	Description
timezone	string	Timezone · Time zone identifier.
userType	string	User Type · User type identifier.
street_address	string	Street Address · Full street address component.
locality	string	Locality · City or locality component.
region	string	Region · State, province, prefecture, or region component.
postal_code	string	Postal Code · Zip code or postal code component.
country	string	Country · Country name component.
formatted	string	Formatted · Full mailing address, formatted for display.
<b>userType</b>	<b>string</b>	<b>User Type</b>

Here it's okay to leave it a yellow arrow.

The screenshot shows the Okta Profile Editor interface. On the left, there is a sidebar with navigation links: Dashboard, Directory (expanded), People, Groups, Profile Editor, Directory, Integrations, Self-Service, and Registration. The main area contains four mapping fields:

- preferredLanguage**: Type: language code. Expression: Choose an attribute or enter an expression... (empty)
- locale**: Type: locale. Expression: Choose an attribute or enter an expression... (empty)
- timezone**: Type: timezone. Expression: Choose an attribute or enter an expression... (empty)
- userType**: Type: string. Expression: appuser.userType. The mapping icon for this field is highlighted with a yellow box and a yellow arrow pointing right.

Click “Save Mapping”.

The screenshot shows the Okta Profile Editor interface for mapping attributes from an external source to Okta user profiles. The left sidebar lists various Okta services: Dashboard, Directory, People, Groups, Profile Editor, Directory Integrations, Self-Service Registration, Profile Sources, and Customizations. The main area displays four mapping rows:

Attribute	Type
division	string
department	string
managerId	string
manager	string

Each row contains a dropdown menu labeled "Choose an attribute or enter an expression..." and a mapping icon (a dashed arrow). At the bottom of the screen, there is a "Preview" button with the placeholder "Enter an Okta user to preview th" and two buttons: "Save Mappings" (highlighted in blue) and "Cancel".

Then click on “Apply updates now”.

The screenshot shows the Okta Profile Editor interface. On the left is a dark sidebar with navigation links: Dashboard, Directory (expanded), People, Groups, Profile Editor, Directory Integrations, Self-Service Registration, Profile Sources, Customizations (expanded), and Applications. The main area displays four mapping rows for attributes: division, department, managerId, and manager. Each row has a dropdown labeled "Choose an attribute or enter an expression..." and a mapping icon (a dashed arrow). To the right of each attribute name and type (string) are up and down arrows for sorting. Below the mappings, a green banner displays the message "Mappings saved!". A question follows: "Do you want to apply these mappings to all users with this profile?". Two buttons are present: a blue button with white text labeled "Apply updates now" and a smaller blue link labeled "Don't apply updates".

division string

department string

managerId string

manager string

Mappings saved!

Do you want to apply these mappings to all users with this profile?

**Apply updates now**

Don't apply updates

Step 4, configure claim in our API.  
Go to security tab.

The screenshot shows the Okta user interface. On the left, there is a navigation sidebar with the following items:

- Dashboard
- Directory
- Customizations
- Applications
- Security** (highlighted with a red box)
- Workflow
- Reports
- Settings

The main content area is titled "Users". It features a search bar and a blue button labeled "+ Create Okta User Type". Below these are two tabs: "Filters" and "Profile". The "Profile" tab is selected. The table displays three user entries:

Profile	Type	Mappings
okta User (default) user	Okta	
OpenID Developer Registration SSO User oidc_idp	Identity Provider	<a href="#">Mappings</a>
My library App User oidc_client	Application	<a href="#">Mappings</a>

Scroll down until you can see API.

The screenshot shows the Okta Admin Console interface. On the left, a vertical sidebar lists various administrative sections: Authentication, Multifactor, Identity Providers, Delegated Authentication, Networks, Behavior Detection, Administrators, and API. The 'API' section is highlighted with a red box. The main content area is titled 'API' and contains three tabs: 'Authorization Servers' (selected), 'Tokens', and 'Trusted Origins'. Below the tabs is a button labeled '+ Add Authorization Server' and a search bar. A table lists one authorization server entry:

Name	Audience	Issuer URI	Status	Action
default	api://default	https://dev-65756343.okta.com/oauth2/default	Active	<a href="#">Edit</a>

Click on “Edit”.

The screenshot shows the Okta API interface. On the left, a sidebar lists navigation options: Authentication, Multifactor, Identity Providers, Delegated Authentication, Networks, Behavior Detection, and Administrators. The main content area is titled "API" and has tabs for "Authorization Servers", "Tokens", and "Trusted Origins". The "Authorization Servers" tab is selected. A button labeled "+ Add Authorization Server" is visible. Below it, a table lists one entry: "default" with Audience "api://default" and Issuer URI "https://dev-65756343.okta.com/oauth2/default". To the right of the table, there is a status indicator "Active" followed by a dropdown arrow and a red-bordered edit icon (pencil symbol).

Name	Audience	Issuer URI	Status	Action
default	api://default	https://dev-65756343.okta.com/oauth2/default	Active	

Click on the tab “Claims”.

The screenshot shows the Okta web interface for managing an authorization server. The left sidebar contains navigation links like Dashboard, Directory, Customizations, Applications, Security, General, HealthInsight, Authentication, Multifactor, Identity Providers, Delegated Authentication, Networks, and Behavior Detection. The main content area has a search bar at the top. Below it, there are two tabs: "Active" (selected) and "Default". Underneath these are four sub-tabs: Settings, Scopes, Claims (which is highlighted with a red box), and Access Policies. The main panel displays the "Settings" for the default authorization server. It includes fields for Name (default), Audience (api://default), Description (Default Authorization Server for your Applications), Issuer (Okta URL (https://dev-65756343.okta.com/oauth2/default)), Metadata URI (https://dev-65756343.okta.com/oauth2/default/.well-known/oauth-authorization-server), Signing Key Rotation (Automatic), and Last Rotation (11 Feb 2023). To the right of the settings, there is a sidebar titled "Authorization Servers" with descriptive text about what an authorization server does and a link to the help page.

Settings		Edit
Name	default	
Audience	api://default	
Description	Default Authorization Server for your Applications	
Issuer	Okta URL (https://dev-65756343.okta.com/oauth2/default)	
Metadata URI	https://dev-65756343.okta.com/oauth2/default/.well-known/oauth-authorization-server	
Signing Key Rotation	Automatic	
Last Rotation	11 Feb 2023	

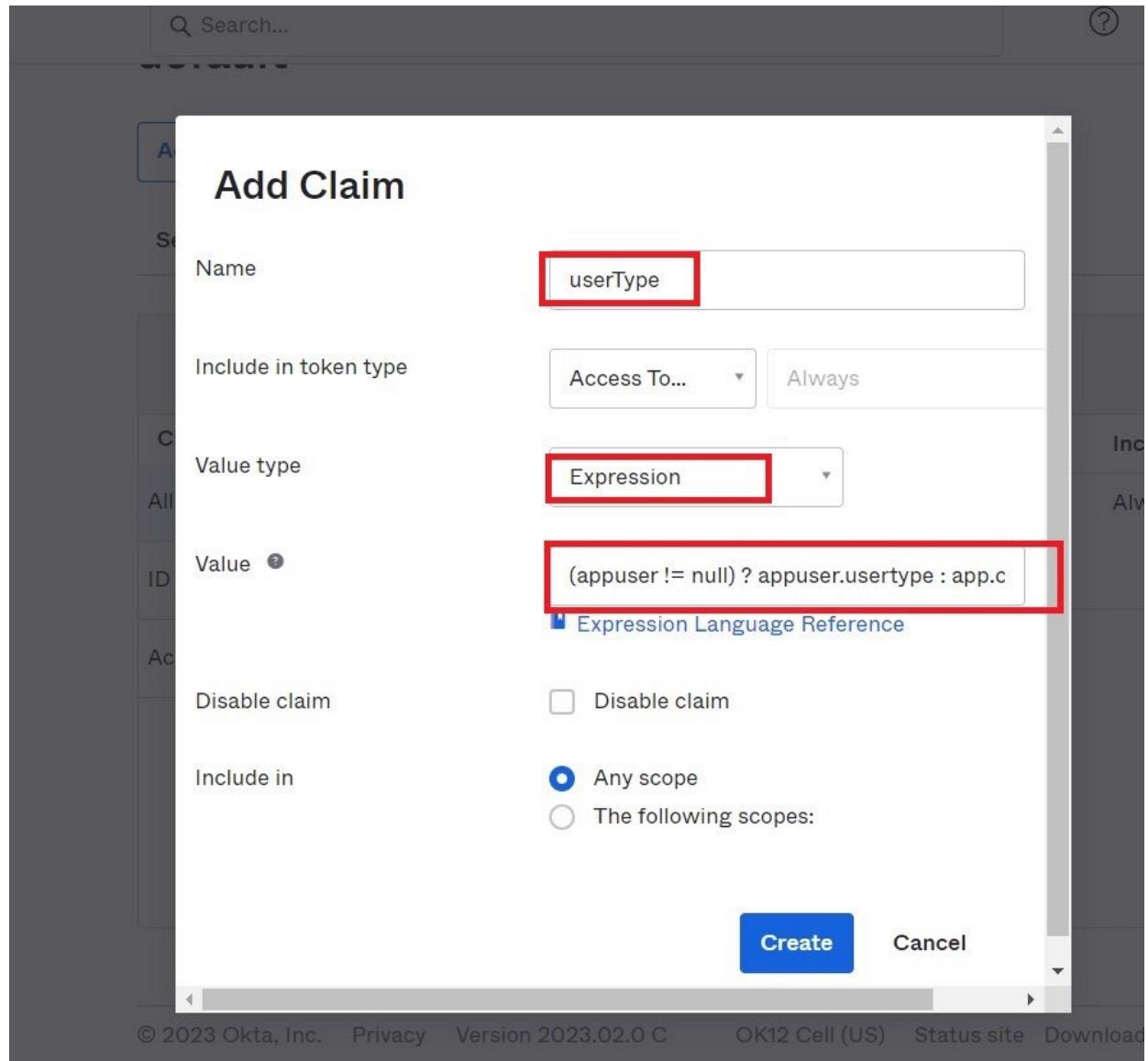
**Authorization Servers**

An authorization server defines your security boundary, and is used to mint access and identity tokens for use with OIDC clients and OAuth 2.0 service accounts when accessing your resources via API. Within each authorization server you can define your own OAuth scopes, claims, and access policies. Read more at [help page](#)

Click on “Add Claim”.

The screenshot shows the Okta interface for managing claims. On the left, there's a navigation sidebar with categories like Dashboard, Directory, Customizations, Applications, and Security (expanded to show General, HealthInsight, Authentication, Multifactor, Identity Providers, and Delegated Authentication). The main area has tabs for Active (selected), Default, Settings, Scopes, Claims (selected), Access Policies, and Token Preview. Below these tabs is a table with a header: Claim type, Name, Value, Scopes, Type, and Included. A red box highlights the '+ Add Claim' button in the top-left corner of the table area. The table contains three rows: 1) All, sub, (appuser != null) ? appuser.userName : app.clientId, Any, acces, Always; 2) ID; 3) Access.

Claim type	Name	Value	Scopes	Type	Included
All	sub	(appuser != null) ? appuser.userName : app.clientId	Any	acces	Always
ID					
Access					



Make sure you do the same as the picture,  
Value should be:  
`(appuser != null) ? appuser.usertype : app.c`  
Then , click on “create”.

Here, we can get this page and our new created userType appeared.

The screenshot shows the Azure Active Directory (Azure AD) application configuration interface. On the left, there is a navigation sidebar with the following items:

- Dashboard
- Directory
- Customizations
- Applications
- Security ▾
  - General
  - HealthInsight
  - Authentication
  - Multifactor
  - Identity Providers
  - Delegated Authentication

The main area of the screen is titled "Active" and shows the "Claims" tab selected. The "Claims" section has a header with "Add Claim" and tabs for "Settings", "Scopes", "Claims", "Access Policies", and "Token Preview". Below this is a table with the following data:

Claim type	Name	Value	Scopes	Type	Included
All	sub	(appuser != null) ? appuser.userName : app.clientId	Any	access	Always
ID					
Access	userType	(appuser != null) ? appuser.userType : app.clientId	Any	access	Always

A red box highlights the last row of the table, which contains the newly created claim for "userType".

Now let's go back to our directory and choose "people".  
Click on "admin user".

The screenshot shows the Okta Admin Console interface. On the left, there is a sidebar with the following navigation items:

- Dashboard
- Directory
- People** (highlighted with a red box)
- Groups
- Profile Editor
- Directory Integrations
- Self-Service Registration
- Profile Sources
- Customizations
- Applications
- Security

The main content area is titled "People". It features several buttons at the top: "Add person", "Reset passwords", "Reset multifactor", and "More actions". Below these are search and filter options: a search bar with placeholder "Search for users by first name, primary email or username", a "Advanced search" dropdown, and a "Status" dropdown set to "All". A message indicates "Showing 4 of 4".

A table lists four users:

Person & username	Primary email	Status
Admin User adminuser@email.com	adminuser@email.com	Active
Test User testuserjac@email.com	testuserjac@email.com	Active
Test User2 testuser2@email.com	testuser2@email.com	Active
suzy zhang	forsuzy.zhang@gmail.com	Password reset

Let's go to profile.

The screenshot shows the Okta Admin Console interface. On the left, there is a sidebar with the following menu items:

- Dashboard
- Directory
- People** (selected)
- Groups
- Profile Editor
- Directory Integrations
- Self-Service Registration
- Profile Sources
- Customizations
- Applications
- Security

The main content area displays the details for a user named "Admin User" (adminuser@email.com). The user is marked as "Active". Below the user info, there are four buttons: "Reset Password", "More Actions", "User", and "View Logs".

At the top of the main content area, there is a navigation bar with tabs: Applications, Groups, **Profile**, and Admin roles. The "Profile" tab is highlighted with a red box.

The "Attributes" section shows the following user information:

Attribute	Value
Username	adminuser@email.com
login	
First name	Admin
firstName	
Last name	User
lastName	

To the right of the attributes, there is a "Profile" section with the following text:

**Profile**  
A profile is a collection of attributes that describe a user in Okta. Some apps and directories can sync attributes with Okta.

Click on “Edit”.

The screenshot shows the Okta User Profile interface. On the left, a sidebar menu includes options like Dashboard, Directory, People (which is selected and highlighted in light blue), Groups, Profile Editor, Directory Integrations, Self-Service Registration, Profile Sources, Customizations (with a dropdown arrow), and Applications (with a dropdown arrow). The main content area displays a user profile for "Admin User" (adminuser@email.com). The profile card includes a placeholder user icon, the user's name, email, and two buttons: "Reset Password" and "More Actions". Below the card are status filters: "User", "Active", and "View Logs". A navigation bar at the top of the content area has tabs for "Applications", "Groups", "Profile" (which is underlined in blue), and "Admin roles". The "Profile" tab is active. Under the "Profile" tab, there is a section titled "Attributes" containing user information: "Username" (adminuser@email.com) and "login"; "First name" (Admin) and "firstName". To the right of the "Attributes" table is a "Profile" sidebar with the following text:

**Profile**  
A profile is a collection of attributes that describe a user in Okta. Some apps and directories can sync attributes with Okta.

The "Edit" button in the "Attributes" table is highlighted with a red box.

Input “admin” in User type box.

The screenshot shows the Okta Directory interface. On the left, there's a sidebar with navigation links: Dashboard, Directory (expanded), People (selected), Groups, Profile Editor, Directory Integrations, Self-Service Registration, Profile Sources, Customizations, and Applications. The main area displays a user profile form with the following fields:

Field	Value
Time zone timezone	(empty input field)
User type userType	admin
Employee number employeeNumber	(empty input field)
Cost center costCenter	(empty input field)
Organization organization	(empty input field)
Division division	(empty input field)
Department department	(empty input field)

A red box highlights the "User type" field, which contains the value "admin". A blue box highlights the "userType" label. The top right of the screen has a search bar, a help icon, and a grid icon.

okta

Search... ?

Dashboard ▾

Directory ^

People

Groups

Profile Editor

Directory Integrations

Self-Service Registration

Profile Sources

Customizations ▾

Applications ▾

Security ▾

Preferred language  
preferredLanguage

Locale  
locale

Time zone  
timezone

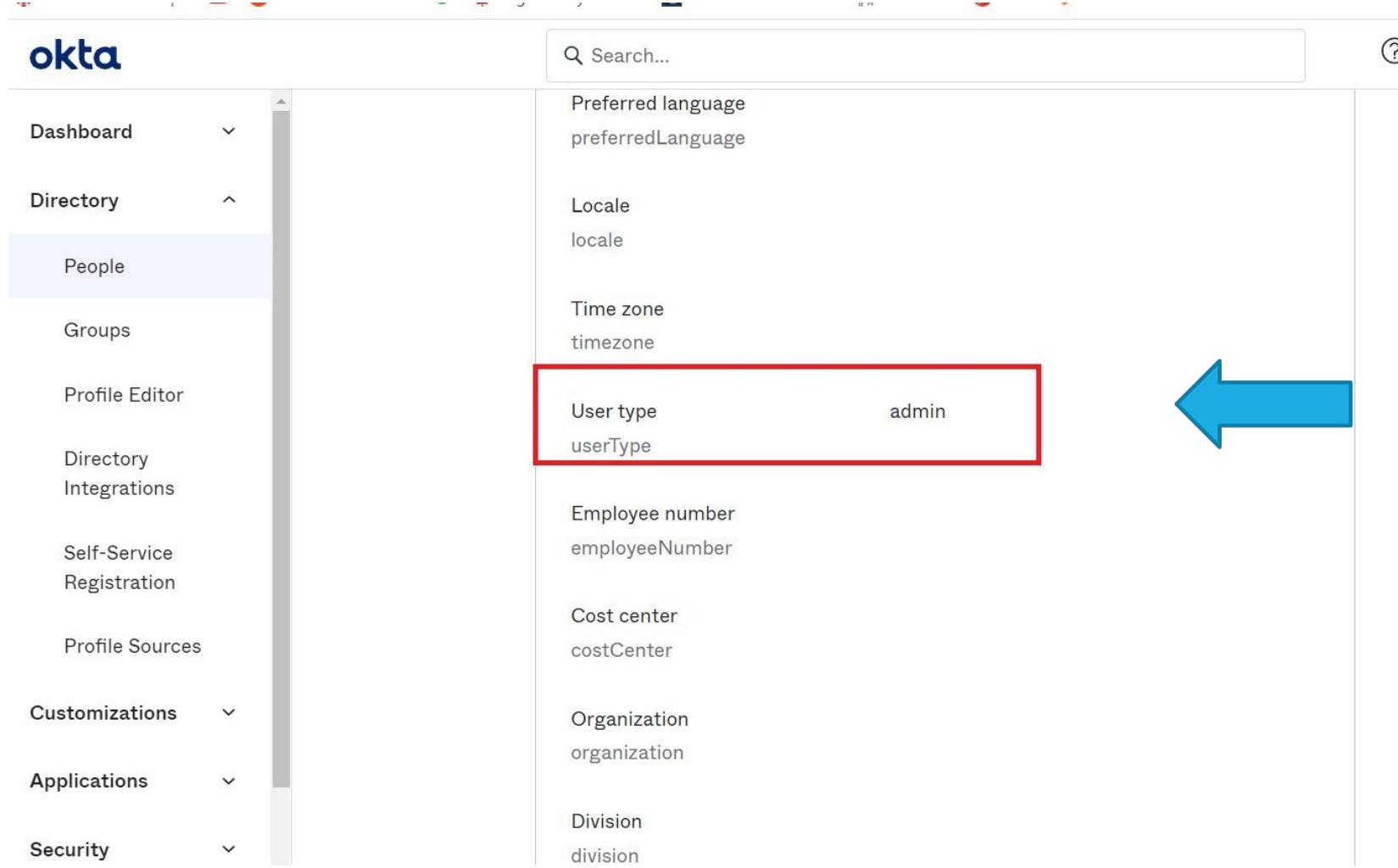
User type admin  
userType

Employee number  
employeeNumber

Cost center  
costCenter

Organization  
organization

Division  
division



So if we go back to “people”, our admin user, profile. We can see a user type of admin.

Go to our Test User2, “profile”,

The screenshot shows the Okta user interface for managing a user profile. On the left, a sidebar menu is visible with options like Dashboard, Directory, People (which is selected and highlighted in blue), Groups, Profile Editor, Directory Integrations, Self-Service Registration, Profile Sources, Customizations, and Applications. The main content area displays a user card for "Test User2" with the email "testuser2@email.com". Below the card are buttons for "Reset Password" and "More Actions". Underneath the card, there are filters for "User", "Active", and "View Logs". A navigation bar below these filters includes links for "Applications", "Groups", "Profile" (which is highlighted with a red box), and "Admin roles". The "Profile" section is expanded, showing a table of attributes. The table has two columns: "Attribute" and "Value". The first row shows "Username" as "testuser2@email.com" and "login". The second row shows "First name" as "Test" and "firstName". An "Edit" link is located at the top right of this table. To the right of the table, a "Profile" section is described with the text: "A profile is a collection of attributes that describe a user in Okta. Some apps and directories can sync attributes with Okta."

Attribute	Value
Username	testuser2@email.com
login	
First name	Test
firstName	

**Profile**  
A profile is a collection of attributes that describe a user in Okta. Some apps and directories can sync attributes with Okta.

This user has nothing in admin.

The screenshot shows the Okta Admin Console interface. On the left, there is a navigation sidebar with the following items:

- Dashboard
- Directory ▾
  - People
  - Groups
- Profile Editor
- Directory Integrations
- Self-Service Registration
- Profile Sources
- Customizations ▾

The "People" item under "Directory" is highlighted with a light blue background. The main content area on the right displays a search bar at the top with the placeholder "Search...". Below the search bar is a list of attributes grouped into sections:

- Postal address:
  - postalAddress
- Preferred language:
  - Preferred language
  - preferredLanguage
- Locale:
  - Locale
  - locale
- Time zone:
  - Time zone
  - timezone
- User type:
  - User type
  - userType
- Employee number:
  - Employee number
  - employeeNumber
- Cost center:
  - Cost center
  - costCenter

So now if we go to our application, sign in as our new admin user's email.

The screenshot shows a web browser window with the following details:

- Tab Bar:** React App, Home | Okta Developer, okta-dev-65756343 - Person, +
- Address Bar:** localhost:3000/login
- Toolbar:** Back, Forward, Refresh, Home, Search, Favorites, etc.
- Page Header:** JAC Read, Home, Search Books, Sign in
- Content Area:** A "Sign In" form with fields for Username and Password, a Remember me checkbox, and a Sign In button. The Username field contains "adminuser@email.com" and is highlighted with a red border. The Password field contains "\*\*\*\*\*".
- Page Footer:** © JAC Library App, Inc., Home, Search Books

Right click, choose “Inspect”. We have our object for our admin user. And now if we look at claims, we can see “userType:admin”.

The screenshot shows a browser window with the URL `localhost:3000/home`. The page content is a dark-themed landing page for "JAC Read" with text "Find your next adventure" and "Where would you like to go next?". A blue button at the bottom says "Explore Books". Overlaid on the page is the browser's developer tools, specifically the "Console" tab. The console displays an "Object" inspect pane. One of the properties, "userType", is highlighted with a red box. The full object structure shown in the console is:

```
▶ Object
  ▶ accessToken:
    accessToken: "eyJraWQiOijFeW1GaFZESV4TWF0cV4TmthWTR...QW...37PU...wvLB_wnf7sx17qkNXuWedPw...T2NGlyog"
    authorizeUrl: "https://dev-65756343.okta.com/oauth2/default/v1/authorize"
  ▶ claims:
    aud: "api://default"
    auth_time: 1676478104
    cid: "0oa8avho11Q6KyNMZ5d7"
    exp: 1676481705
    iat: 1676478105
    iss: "https://dev-65756343.okta.com/oauth2/default"
    jti: "AT.x66QoW2guZ-6i23QGg97uos53DHUsT0VzZ9cLEJeyXo"
    ▶ scp: (3) ['profile', 'openid', 'email']
    sub: "adminuser@email.com"
    uid: "00u8ceoigaD7hyKTb5d7"
    userType: "admin" userType: "admin"
    ver: 1
  ▶ [[Prototype]]: Object
  expiresAt: 1676481706
  ▶ scopes: (3) ['profile', 'openid', 'email']
  tokenType: "Bearer"
  userinfoUrl: "https://dev-65756343.okta.com/oauth2/default/v1/userinfo"
  ▶ [[Prototype]]: Object
  ▶ idToken: {idToken: 'eyJraWQiOijFeW1GaFZESV4TWF0cV4TmthWTR...QW...37PU...wvLB_wnf7sx17qkNXuWedPw...T2NGlyog', claims: ...}
  isAuthenticated: true
  refreshToken: undefined
  ▶ [[Prototype]]: Object
```

React App x Home | Okta Developer x okta-dev-65756343 - Person x +

localhost:3000/admin

Radio-Canada.ca |... Canada's federal... G Sign in to your IR... C Chenelière Éducat... B\* Montreal zdm Nos incontournab... TED TED: Ideas worth... 在线课...

JAC Read Home Search Books Shelf Logout

## Manage Library

Add new book Change quantity Messages

Add New Book

Now, with admin's log in,  
Visit  
<http://localhost:3000/admin>,  
it will show the manage  
library page.

Now log in as “ testuser2 ”:

← → ⌂ ⓘ localhost:3000/login

Radio-Canada.ca |... YouTube Canada's federal... G Sign in to your IR... C Chenelière Éducat... 蒙特利爾 Montreal zdm Nos incontournab... TED TED: Ideas worth... 在线

JAC Read Home Search Books

Sign in

Sign In

Username

Password

Remember me

Need help signing in?

Open “Inspect”, and click on “claims”, you will see there is no user type.

The screenshot shows a web browser window with multiple tabs. The active tab is 'okta-dev-65756343 - Person' at 'localhost:3000/search'. The page displays search results for books, with one book titled 'JavaScript Cookbook' by 'Luv, Zofia' shown in detail. The developer tools console is open on the right, showing a JSON object with a red box highlighting the 'claims' field. The 'claims' field contains various JWT claims including 'aud', 'auth\_time', 'cid', 'exp', 'iat', 'iss', 'jti', 'scp', 'sub', 'uid', 'ver', and 'expiresAt'. The 'tokenType' field is set to 'Bearer'.

```
1 Issue: 1
Connected the React DevTools for a better development experience: https://reactjs.org/link/react-devtools
Navbar.tsx:14
{
  accessToken: {...},
  idToken: {...},
  refreshToken: undefined,
  isAuthenticated: true
}
  accessToken: "eyJraWQiOiJFeW1GaFZESVk4TWF0cVk4TmthWTREQnJ
  authorizeUrl: "https://dev-65756343.okta.com/oauth2/default"
  claims: {
    aud: "api://default"
    auth_time: 1676478272
    cid: "0oa8avho11Q6KyNMZ5d7"
    exp: 1676481874
    iat: 1676478274
    iss: "https://dev-65756343.okta.com/oauth2/default"
    jti: "AT.hUQTOuBV6dPt0G-QWPCPRfWxweAs04oVFZl2Zn1Izxc"
    scp: (3) ['email', 'profile', 'openid']
    sub: "testuser2@email.com"
    uid: "00u8av4atbahEMCHJ5d7"
    ver: 1
  }
  [[Prototype]]: Object
  expiresAt: 1676481874
  scopes: (3) ['email', 'profile', 'openid']
  tokenType: "Bearer"
  userinfoUrl: "https://dev-65756343.okta.com/oauth2/default"
  [[Prototype]]: Object
  idToken: {idToken: "eyJraWQiOiJFeW1GaFZESVk4TWF0cVk4TmthWTREQnJ
  isAuthenticated: true
  refreshToken: undefined
  [[Prototype]]: Object
```

Then input <http://localhost:3000/admin>,

The screenshot shows a browser window with three tabs:

- React App
- Home | Okta Developer
- okta-dev-65756343 - Person

The URL bar shows [localhost:3000/admin](http://localhost:3000/admin). The main content area is a book catalog with the following details:

Search  Search Book category ▾

Number of results: (22)  
1 to 5 of 22 items:

Book Image	Author	Title	Description	Action
	Luv, Zofia	JavaScript Cookbook	... Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin risus tortor, condimentum eget sapien ac, dapibus varius ligula. Maecenas justo erat, semper sed nunc vel, vulputate eleifend dui. Integer id ipsum vitae nisi malesuada feugiat. Proin sit amet quam laoreet, feugiat mi vitae, vestibulum dui. Aliquam erat volutpat. Etiam hendrerit erat nec mi auctor elementum. Curabitur vestibulum lectus a ante tempor tincidunt et sed orci. Proin maximus tortor in risus auctor efficitur. Phasellus quam mauris, laoreet et feugiat ac, imperdiet at quam. Nullam sollicitudin nec diam vel finibus.	<a href="#">View details</a>
	Luv, Lena	Become a Guru in JavaScript	...	

A developer tools panel is open on the right, showing the following token object:

```
1 Issue: 1
Download the REACT DEV TOOLS for a better development experience: https://reactjs.org/link/react-devtools
Navbar.tsx:14
{
  accessToken: {...}, idToken: {...}, refreshToken: undefined, isAuthenticated: true
}
  accessToken:
    accessToken: "eyJraWQiOiJFeW1GaFZESVk4TWF0cVk4TmthWTREQnC"
    authorizeUrl: "https://dev-65756343.okta.com/oauth2/default"
  claims:
    aud: "api://default"
    auth_time: 1676478272
    cid: "0oa8avho11Q6KyNMZ5d7"
    exp: 1676481874
    iat: 1676478274
    iss: "https://dev-65756343.okta.com/oauth2/default"
    jti: "AT.hUQTOuBV6dPt0G-QWPcPrFxweAs04oVFZl2Zn1Izxc"
  scp: (3) ['email', 'profile', 'openid']
  sub: "testuser2@email.com"
  uid: "00u8av4atbahEMCHJ5d7"
  ver: 1
  [[Prototype]]: Object
  expiresAt: 1676481874
  scopes: (3) ['email', 'profile', 'openid']
  tokenType: "Bearer"
  userinfoUrl: "https://dev-65756343.okta.com/oauth2/default"
  [[Prototype]]: Object
}
idToken: {idToken: "eyJraWQiOiJFeW1GaFZESVk4TWF0cVk4TmthWTREQnC", isAuthenticated: true, refreshToken: undefined}
[[Prototype]]: Object
```

It will redirect us to the home page.

The screenshot shows a web browser window with three tabs:

- React App
- Home | Okta Developer
- okta-dev-65756343 - Person

The address bar shows `localhost:3000/home`, which is highlighted with a red box.

The main content area displays the "JAC Read" application. It features a large banner with the text "Find your next adventure" and "Where would you like to go next?". Below the banner is a blue button labeled "Explore Books".

At the bottom, there is a section titled "Find your next 'I stayed up too late reading' book" with three cards:

- JavaScript
- Become a Guru
- Exploring

The developer tools console on the right side shows a single issue related to the React DevTools. It also displays the contents of the `accessToken` object from the user's token, which includes details like `accessToken`, `claims`, and `idToken`.

```
1 Issue: 1
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: "eyJraWQiOijFeW1GaFZESVk4TWF0cVk4TmthWTREQnJ...[redacted]"
  authorizeUrl: "https://dev-65756343.okta.com/oauth2/default"
  claims: {
    aud: "api://default"
    auth_time: 1676478272
    cid: "Ooa8avho11Q6KyNMZ5d7"
    exp: 1676481874
    iat: 1676478274
    iss: "https://dev-65756343.okta.com/oauth2/default"
    jti: "AT.hUQTouBV6dPt0G-QWPCPRfWxweAs04oVFZl2Zn1Izxc"
    scp: (3) ['email', 'profile', 'openid']
    sub: "testuser2@email.com"
    uid: "00u8av4atbahEMCHJ5d7"
    ver: 1
  }
  [[Prototype]]: Object
  expiresAt: 1676481874
}
  scopes: (3) ['email', 'profile', 'openid']
  tokenType: "Bearer"
  userinfoUrl: "https://dev-65756343.okta.com/oauth2/default"
}
  [[Prototype]]: Object
  idToken: {idToken: 'eyJraWQiOijFeW1GaFZESVk4TWF0cVk4TmthWTREQnJ...[redacted]', isAuthenticated: true, refreshToken: undefined}
  [[Prototype]]: Object
```

---

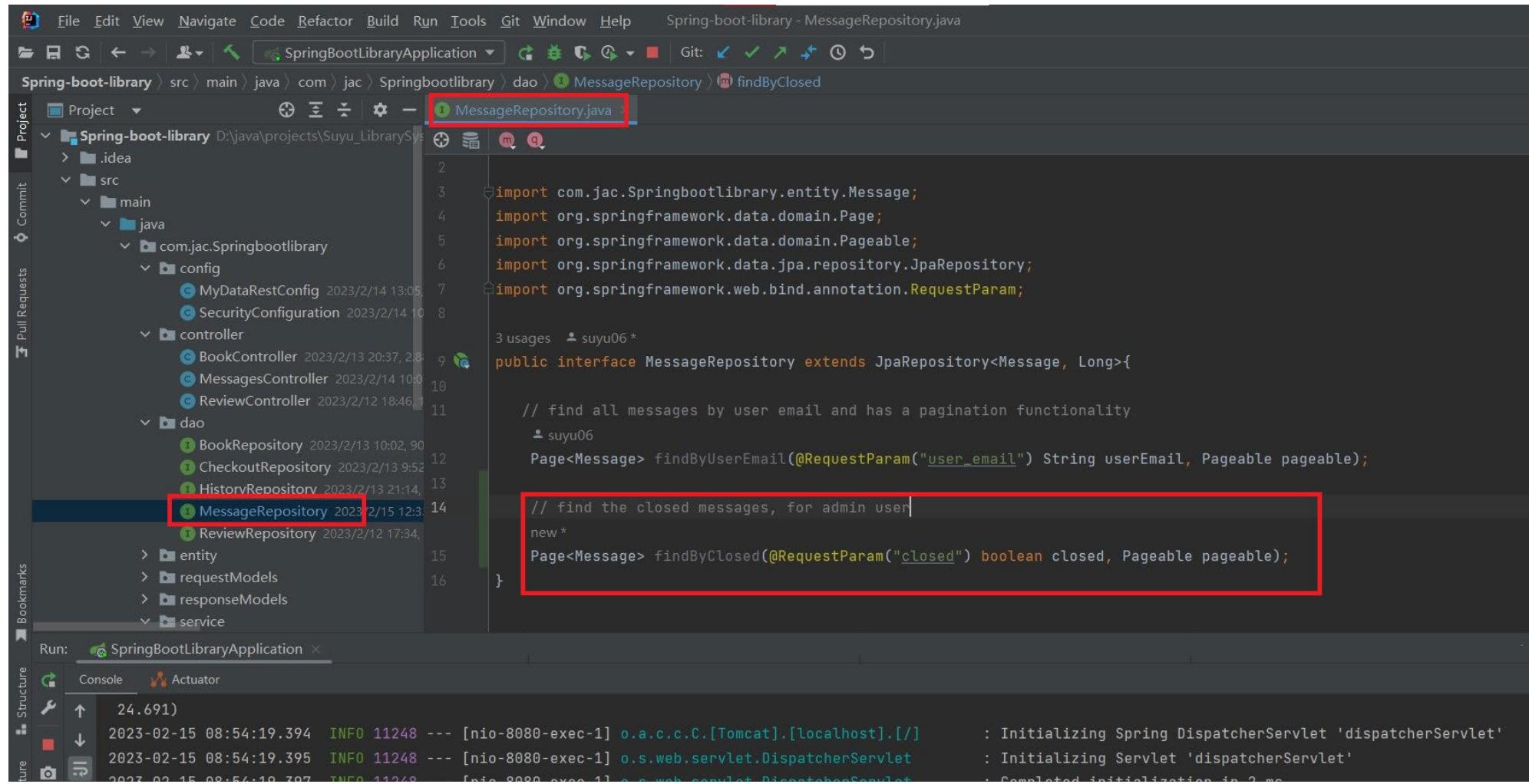
---

---

## **6. SPRINGBOOT OPEN TICKET ENDPOINT**



Go to dao package , in MessageRepository class ,add a findByClosed() interface function.



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

- File Path:** Spring-boot-library > src > main > java > com > jac > Springbootlibrary > dao > MessageRepository
- Code Editor Content:**

```
import com.jac.Springbootlibrary.entity.Message;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.web.bind.annotation.RequestParam;

public interface MessageRepository extends JpaRepository<Message, Long>{

    // find all messages by user email and has a pagination functionality
    Page<Message> findByUserEmail(@RequestParam("user_email") String userEmail, Pageable pageable);

    // find the closed messages, for admin user
    new *

    Page<Message> findByClosed(@RequestParam("closed") boolean closed, Pageable pageable);
}
```
- Toolbars and Menus:** Standard IntelliJ IDEA menu items like File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, Help are visible at the top.
- Sidebar:** Project, Commit, Pull Requests, Bookmarks, and Structure tabs are visible on the left.
- Bottom:** Run tab (SpringBootLibraryApplication), Console tab (output: 24.691, log entries for INFO messages), and Actuator tab.

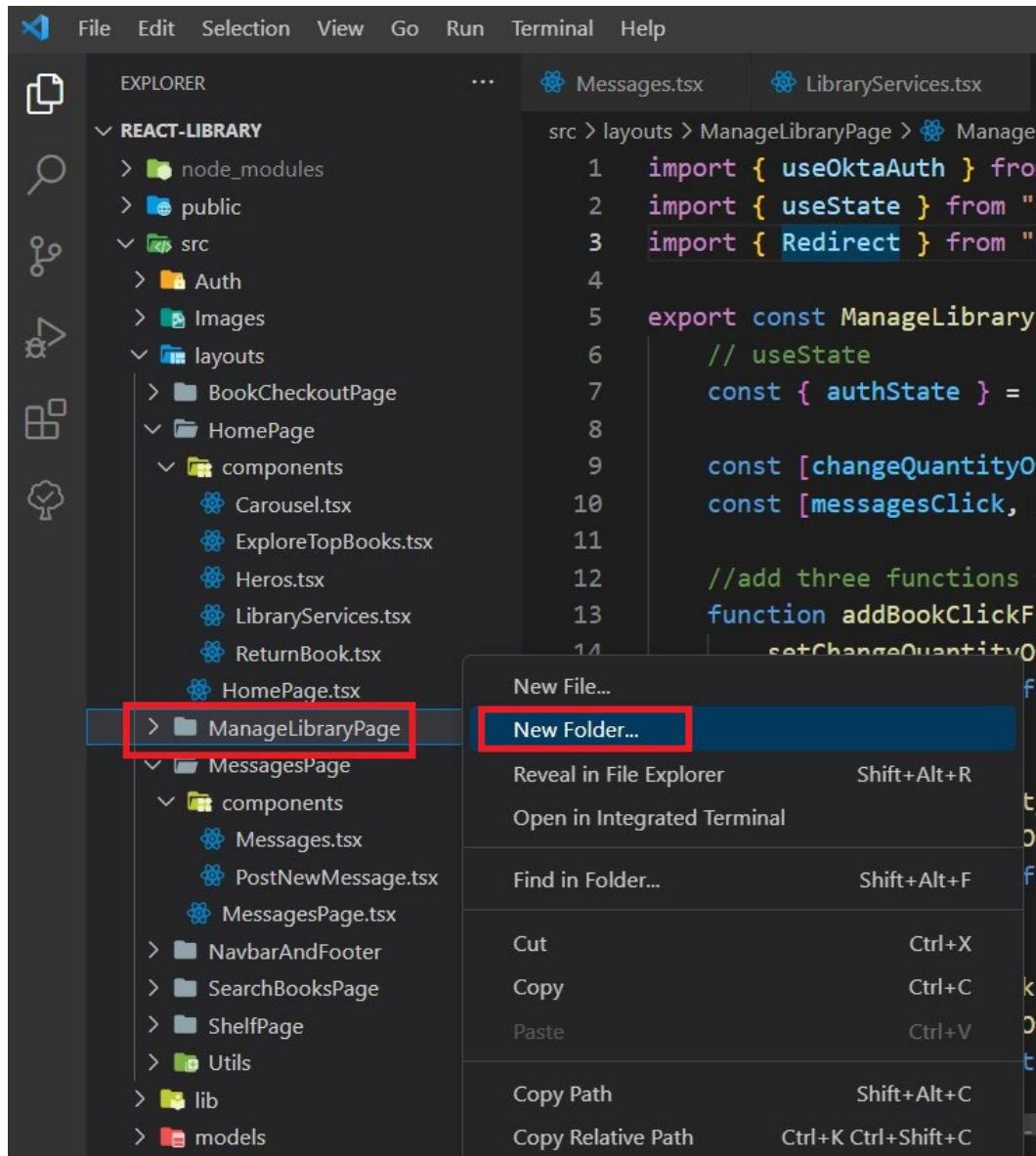
---

---

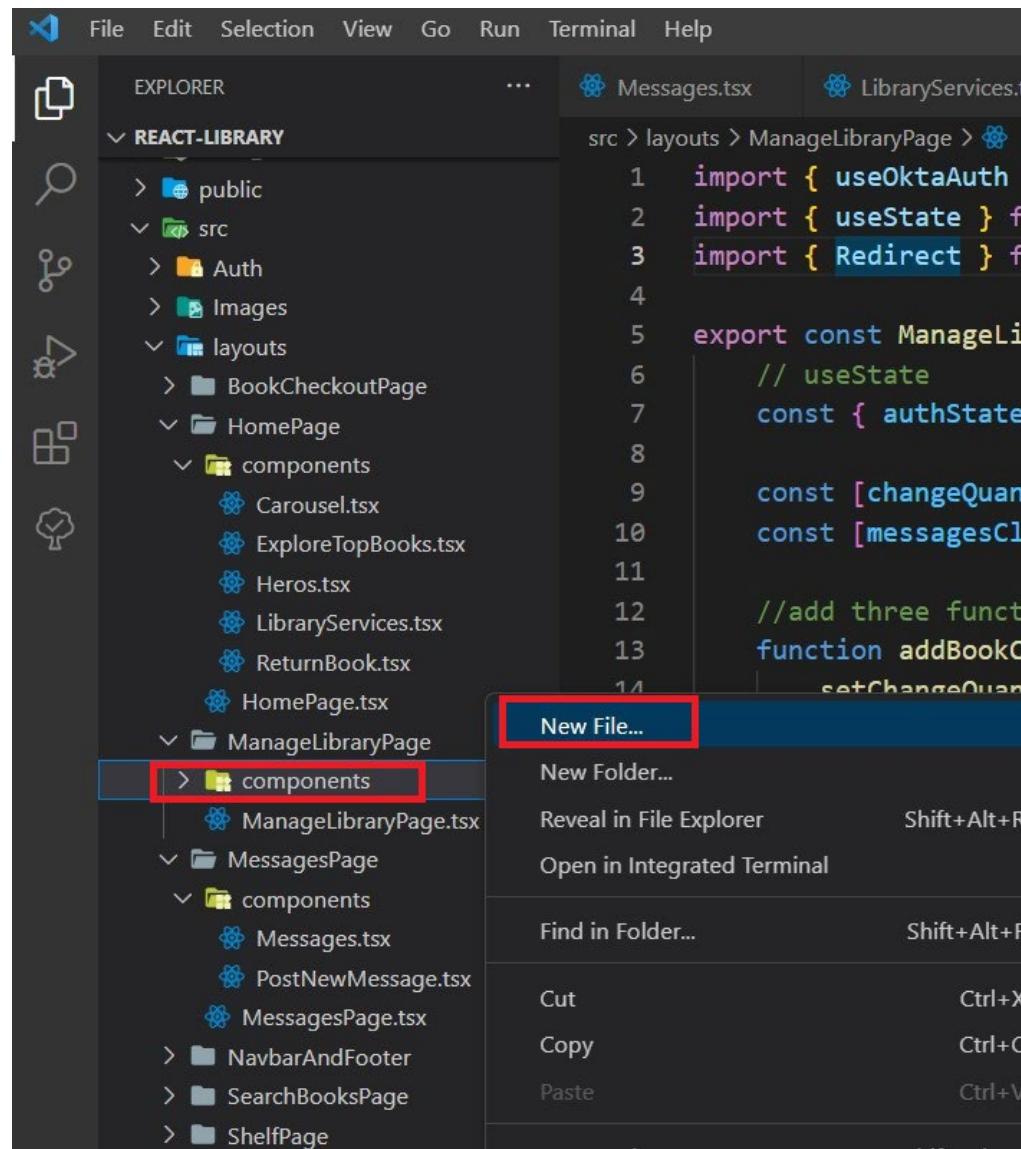
---

## 7. REACT ADMIN MESSAGES COMPONENT





Step1, Let's go ahead and jump into our layouts, ManageLibraryPage,right click to create a new folder called Components.



Step 2, And inside the components, reate a new file AdminMessages.tsx.

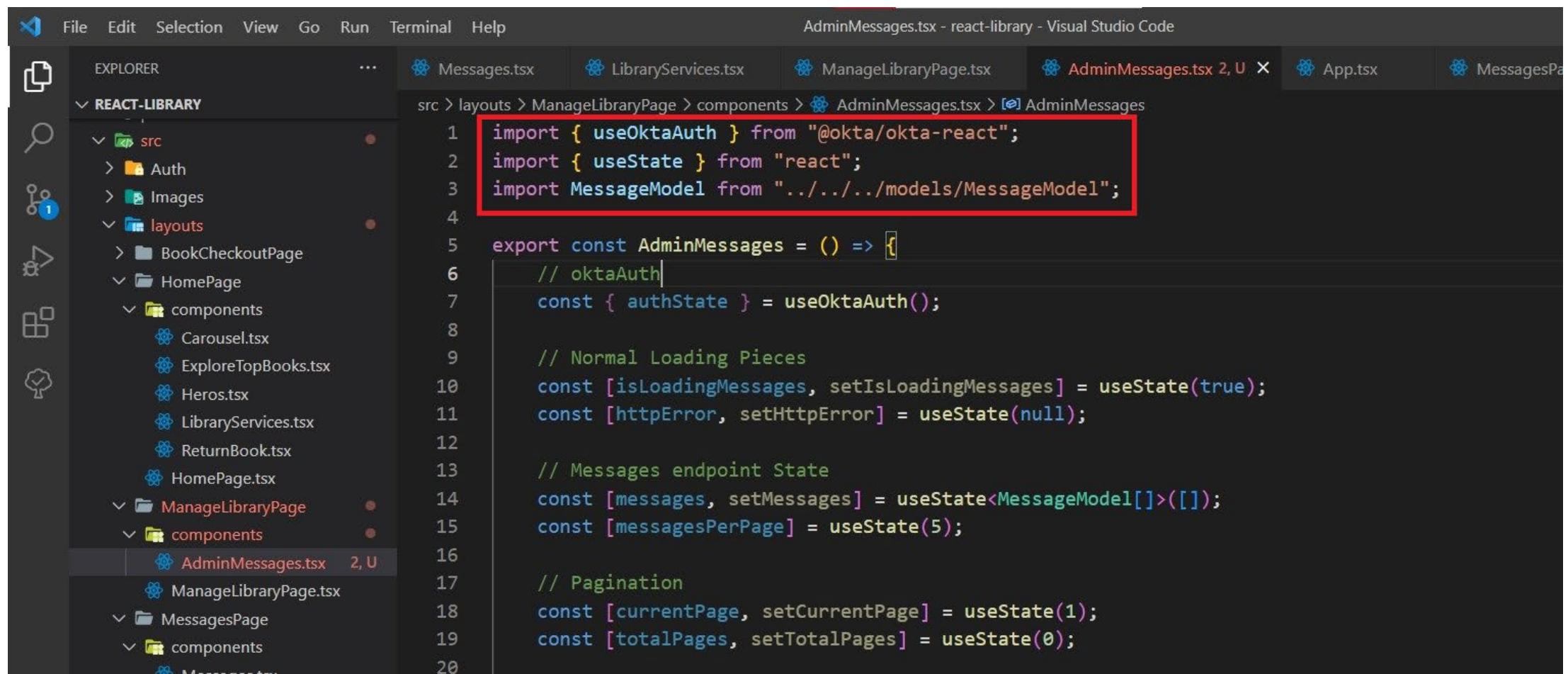
Step3, as always, create export constant AdminMessages() structure;

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

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Title Bar:** AdminMessages.tsx - react-library - Visual Studio Code.
- Explorer:** Shows the project structure under "REACT-LIBRARY".
  - src
    - Auth
    - Images
  - layouts
    - BookCheckoutPage
    - HomePage
      - components
        - Carousel.tsx
        - ExploreTopBooks.tsx
        - Heros.tsx
        - LibraryServices.tsx
        - ReturnBook.tsx
        - HomePage.tsx
    - ManageLibraryPage
      - components
        - AdminMessages.tsx 2, U
        - ManageLibraryPage.tsx
    - MessagesPage
    - components
      - Messages.tsx
      - PostNewMessage.tsx
- Code Editor:** The current file is AdminMessages.tsx, showing the following code:

```
1  export const AdminMessages = () => [
2    return(
3      ...
4    );
5  ]
```

## Step4, add useState and import useOktaAuth, useState and MessageModel.



The screenshot shows the Visual Studio Code interface with the AdminMessages.tsx file open. The code editor displays the following imports:

```
import { useOktaAuth } from "@okta(okta-react";
import { useState } from "react";
import MessageModel from ".../.../models/MessageModel";
```

The first three lines of code, which are the imports, are highlighted with a red rectangular box. The rest of the code is standard React/TypeScript syntax for managing state and fetching data from a messages endpoint.

## Step5, as usual , create useEffect structure.

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

- File Explorer (Left):** Shows the project structure under "REACT-LIBRARY". Key components include "src" (Auth, Images, layouts, HomePage, ManageLibraryPage, MessagesPage), "utils", and "utils". The "AdminMessages.tsx" file is currently selected and has a status bar indicating "2, U".
- Code Editor (Right):** Displays the code for "AdminMessages.tsx".

```
useEffect(() => {
  const fetchUserMessages = async () => {
    try {
      await fetchUserMessages();
    } catch (error) {
      setIsLoadingMessages(false);
      setHttpError(error.message);
    }
    window.scrollTo(0, 0);
  }, [authState, currentPage])

  if (isLoadingMessages) {
    return (
      <SpinnerLoading/>
    );
  }

  if (httpError) {
    return (
      <div className='container m-5'>
        <p>{httpError}</p>
      </div>
    );
  }

  const paginate = (pageNumber: number) => setCurrentPage(pageNumber);
  return(

```
- Status Bar:** Shows the file name "AdminMessages.tsx - react-library - Visual Studio Code" and the status "2, U".

```
useEffect(() => {
  const fetchUserMessages = async () => {
    // only if user is authenticated
    if (authState && authState.isAuthenticated) {
      const url = `http://localhost:8080/api/messages/search/findByClosed
      /?closed=false&page=${currentPage - 1}&size=${messagesPerPage}`;
      const requestOptions = {
        method: 'GET',
        headers: {
          Authorization: `Bearer ${authState.accessToken?.accessToken}`,
          'Content-Type': 'application/json'
        }
      };
      //fetch() to get the data from the url
      const messagesResponse = await fetch(url, requestOptions);
      //meets error
      if (!messagesResponse.ok) {
        throw new Error('Something went wrong!');
      }
      // convert to JSON
      const messagesResponseJson = await messagesResponse.json();
      // save messages and total page number
      setMessages(messagesResponseJson._embedded.messages);
      setTotalPages(messagesResponseJson.page.totalPages);
    }
    //finish loadin process
    setIsLoadingMessages(false);
  }
})
```

Step 6, fill in useEffect with  
async function.

Step 7, fill in return part with Html/CSS code.

```
return(
  <div className='mt-3'>
    {/* if there is messages: */}
    {messages.length > 0 ?
      <>
        <h5>Pending Q/A: </h5>
        {messages.map(message => (
          <p>Questions that need a response</p>
        )));
      </>
      :
      //no messages
      <h5>No pending Q/A</h5>
    }
    {totalPages > 1 && <Pagination currentPage={currentPage} totalPages={totalPages} paginate={paginate}/>}
  </div>
);
```

Step 8, Open up our ManageLibraryPage, Scroll to the bottom.  
And right here in our third tab, we can see “messages”.  
Let's remove these words with new AdminMessages component.

The screenshot shows a code editor with a dark theme. On the left is a file tree:

- > BookCheckoutPage
- > HomePage
- < ManageLibraryPage
  - < components
    - AdminMessages.tsx
  - ManageLibraryPage.tsx
- < MessagesPage
  - < components
    - Messages.tsx
    - PostNewMessage.tsx
    - MessagesPage.tsx
- > NavbarAndFooter
- < SearchBooksPage
  - < components
    - SearchBooksPage.tsx
- > ShelfPage

The file `ManageLibraryPage.tsx` is selected and highlighted with a red box. The code itself is also highlighted with a red box around the word `Admin Messages`:

```
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
```

Messages

```
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
```

```
        </button>
    </div>
</nav>
/* content showed when the tab is clicked */
<div className='tab-content' id='nav-tabContent'>
    <div className='tab-pane fade show active' id='nav-add-book' aria-labelledby='nav-add-book-tab'>
        Add New Book
    </div>
    <div className='tab-pane fade' id='nav-quantity' role='tabpanel' {changeQuantityOfBooksClick ? <>Change Quantity Of Books</> : <></>}>
    </div>
    <div className='tab-pane fade' id='nav-messages' role='tabpanel' {messagesClick ? <>Admin Messages </> : <></>}>
    </div>
</div>
```

Like this below:

The image shows a code editor interface with a sidebar on the left displaying a file tree. The tree includes 'node\_modules', 'public', and 'src' folders. Under 'src', there are 'Auth', 'Images', 'layouts' (which contains 'BookCheckoutPage', 'HomePage', and 'ManageLibraryPage' components), 'MessagesPage' (with 'Messages.tsx' and 'PostNewMessage.tsx'), 'NavbarAndFooter', 'SearchBooksPage' (with 'components' and 'SearchBooksPage.tsx'), and 'ShelfPage'. The 'ManageLibraryPage' component is currently selected in the tree.

The main pane displays the 'ManageLibraryPage.tsx' file content:

```
src > layouts > ManageLibraryPage > ManageLibraryPage.tsx > ManageLibraryPage
```

```
  50   |     Change quantity
  51   |     </button>
  52   |     <button onClick={messagesClickFunction} className='nav-link'
  53   |       data-bs-target='#nav-messages' type='button'
  54   |       aria-selected='false'
  55   |     >
  56   |       Messages
  57   |     </button>
  58   |   </div>
  59   | </nav>
  60   | /* content showed when the tab is clicked */
  61   <div className='tab-content' id='nav-tabContent'>
  62     <div className='tab-pane fade show active' id='nav-add-book' aria-labelledby='nav-add-book-tab'>
  63       Add New Book
  64     </div>
  65     <div className='tab-pane fade' id='nav-quantity' role='tabpanel' aria-labelledby='nav-quantity-tab'>
  66       {changeQuantityOfBooksClick ? <>Change Quantity Of Books</> : null}
  67     </div>
  68     <div className='tab-pane fade' id='nav-messages' role='tabpanel' aria-labelledby='nav-messages-tab'>
  69       {messagesClick ? <AdminMessages/> : null}
  70     </div>
  71   </div>
```

A red rectangular box highlights the line containing the component import: '{messagesClick ? <AdminMessages/> : null}'.

## Manage Library

Add new book

Change quantity

Messages

### Pending Q/A:

Questions that need a response

Questions that need a response

Questions that need a response

With admin user account logged in, click on messages tab, we can see there are three “questions needs to a response”.

Check our table “messages”, you can see there are three questions without response.

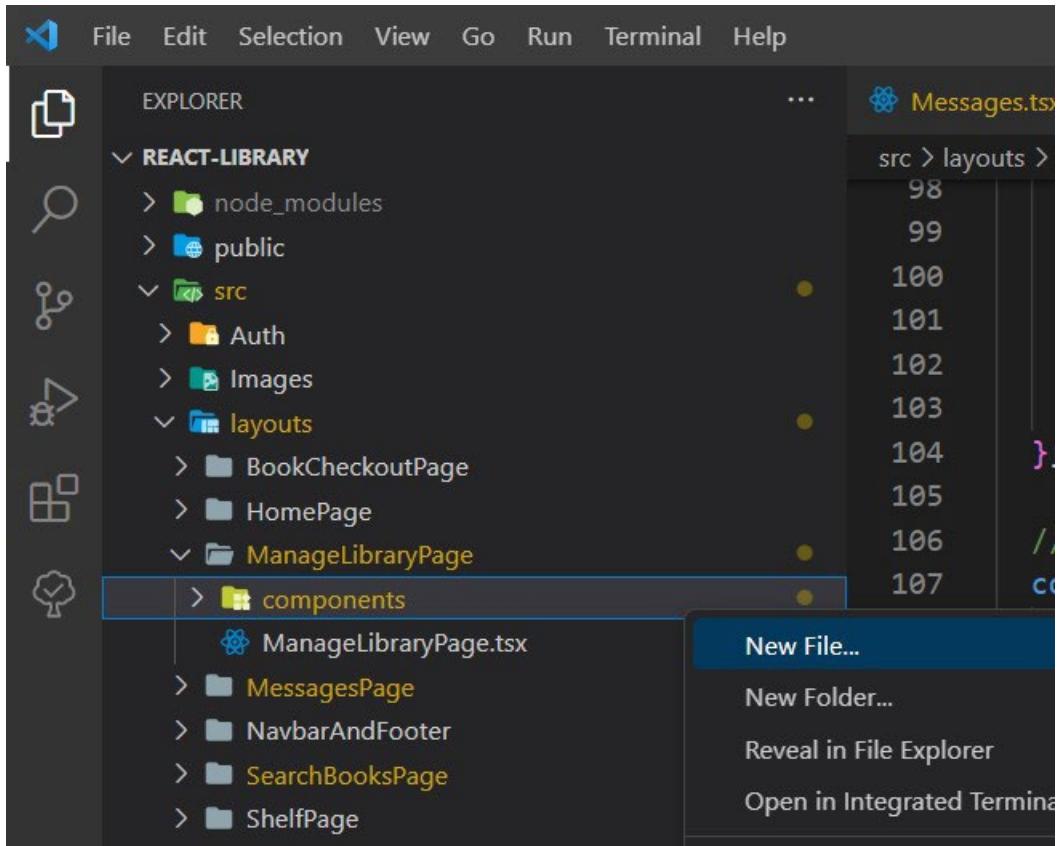
---

---

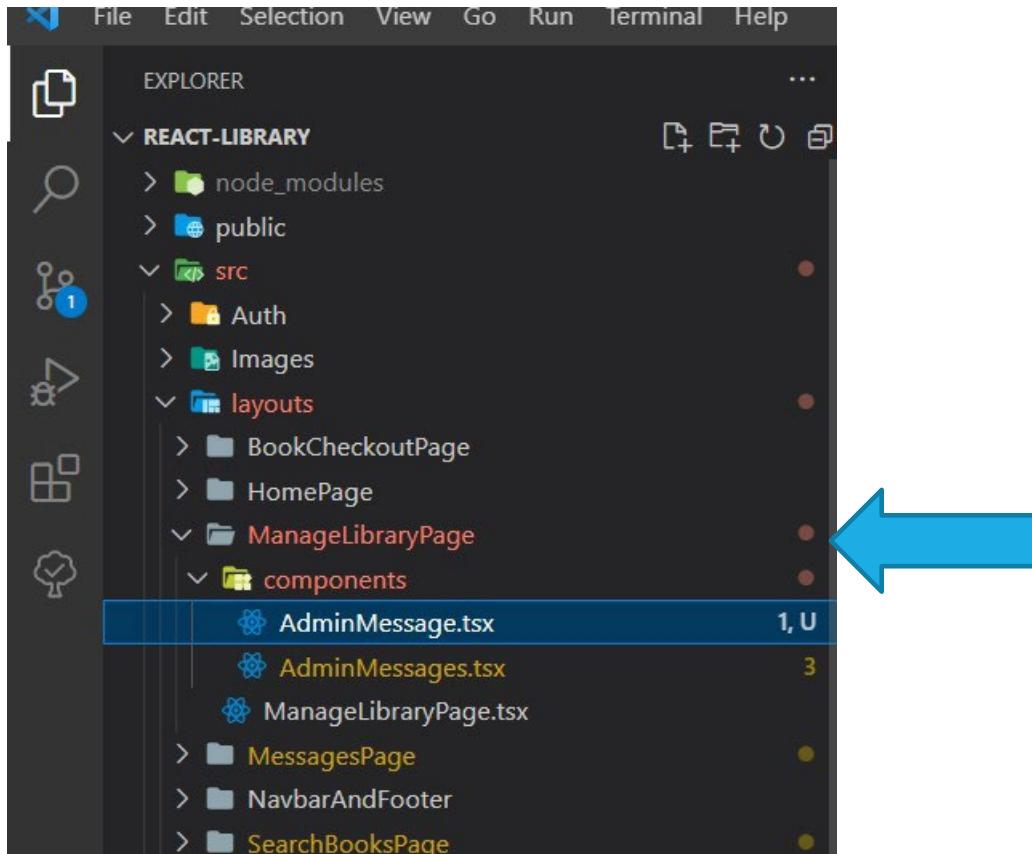
---

## **8.REACT ADMIN MESSAGES COMPONENT**



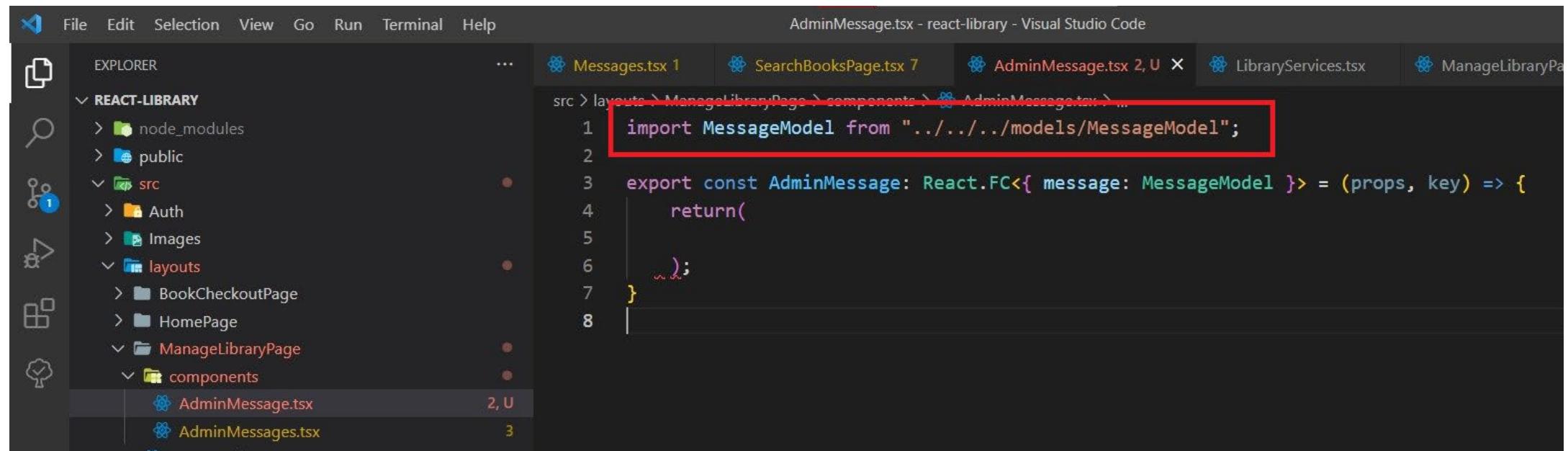


Step1 Let's go ahead and right click on our components within our ManageLibraryPage and say new file.



create our component `AdminMessage.tsx`.

Step 2, create export constant function ,take in a message of type message model as a props.



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

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** AdminMessage.tsx - react-library - Visual Studio Code
- Explorer:** Shows the project structure under REACT-LIBRARY:
  - node\_modules
  - public
  - src
    - Auth
    - Images
    - layouts
      - BookCheckoutPage
      - HomePage
    - ManageLibraryPage
      - components
- Editor:** The AdminMessage.tsx file is open, showing the following code:

```
import MessageModel from "../../../../../models/MessageModel";
export const AdminMessage: React.FC<{ message: MessageModel }> = (props, key) => {
  return(
    ...
  );
}
```
- Status Bar:** Shows AdminMessage.tsx 2, U and AdminMessages.tsx 3

Step3, create useState.

AdminMessage.tsx - react-library - Visual Studio Code

Messages.tsx 1    SearchBooksPage.tsx 7    AdminMessage.tsx 2, U X    LibraryServices.tsx    ManageLibraryPage.tsx

```
src > layouts > ManageLibraryPage > components > AdminMessage.tsx > AdminMessage
1 import { useState } from "react";
2 import MessageModel from "../../../../../models/MessageModel";
3
4 export const AdminMessage: React.FC<{ message: MessageModel }> = (props, key) => {
5     //useState
6     const [displayWarning, setDisplayWarning] = useState(false);
7     const [response, setResponse] = useState('');
8
9     return(
10
11         );
12     }
13 }
```

## Step 4 return part : html/CSS part

```
// html/CSS part
return(
  // go through each message by its id
  <div key={props.message.id}>
    <div className='card mt-2 shadow p-3 bg-body rounded'>
      <h5>Case #{props.message.id}: {props.message.title}</h5>
      <h6>{props.message.userEmail}</h6>
      <p>{props.message.question}</p>
      <hr/>
```

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

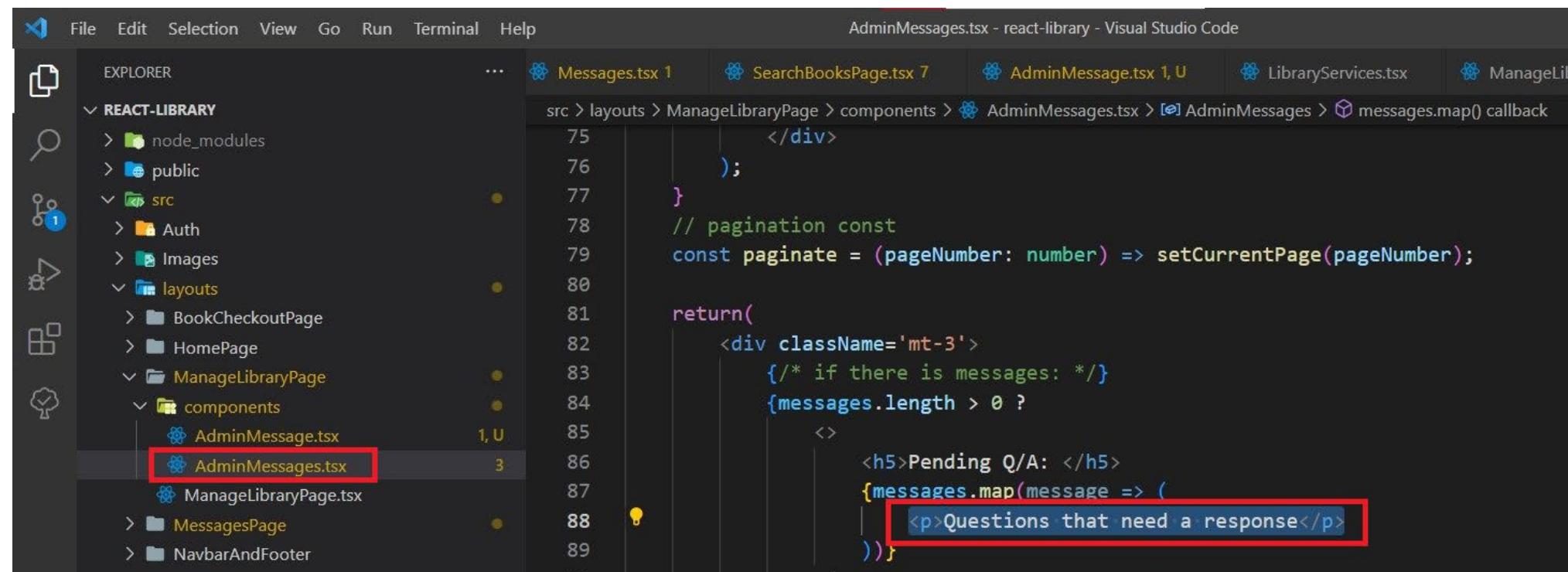
```
<div>
  <h5>Response: </h5>
  <form action="PUT">
    {displayWarning &&
      <div className='alert alert-danger' role='alert'>
        |   All fields must be filled out.
      </div>
    }
    <div className='col-md-12 mb-3'>
      <label className='form-label'> Description </label>
      <textarea className='form-control' id='exampleFormControlTextarea1' rows={3}>
        onChange={e => setResponse(e.target.value)} value={response}</textarea>
    </div>
    <div>
      <button type='button' className='btn btn-primary mt-3'>
        |   Submit Response
      </button>
    </div>
  </form>
</div>
```

Response:

Description

Submit Response

Step 5 Open up our AdminMessages.tsx, change AdminMessage p tag into our component.



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

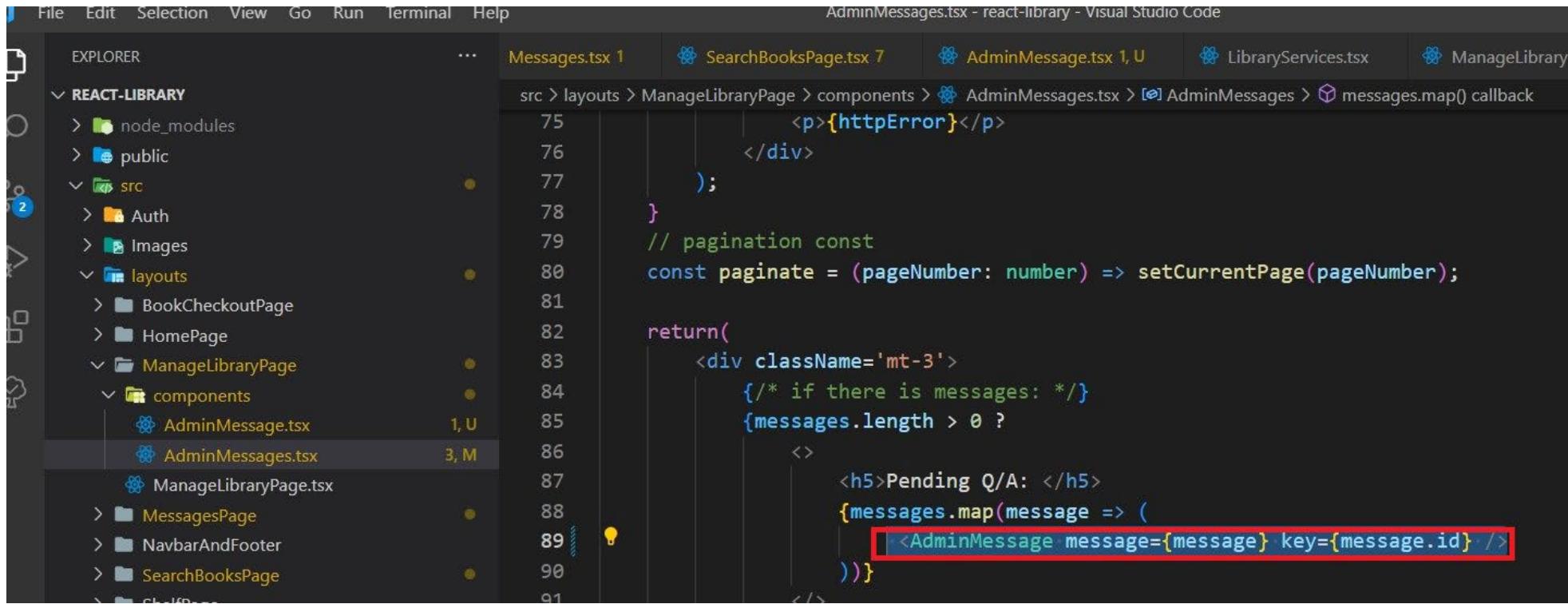
- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** AdminMessages.tsx - react-library - Visual Studio Code
- Explorer:** Shows the project structure under "REACT-LIBRARY".
  - node\_modules
  - public
  - src
    - Auth
    - Images
    - layouts
      - BookCheckoutPage
      - HomePage
    - ManageLibraryPage
      - components
        - AdminMessage.tsx
        - AdminMessages.tsx
    - MessagesPage
    - NavbarAndFooter
- Editor:** The code editor displays the AdminMessages.tsx file.

```
src > layouts > ManageLibraryPage > components > AdminMessages.tsx > AdminMessages > messages.map() callback
    75             </div>
    76         );
    77     }
    78     // pagination const
    79     const paginate = (pageNumber: number) => setCurrentPage(pageNumber);
    80
    81     return(
    82         <div className='mt-3'>
    83             /* if there is messages: */
    84             {messages.length > 0 ?
    85                 <>
    86                     <h5>Pending Q/A: </h5>
    87                     {messages.map(message => (
    88                         <p>Questions that need a response</p>
    89                     ))}
    90             :
    91                 <h5>No pending Q/A's!</h5>
    92             )
    93         </div>
    94     );
    95 }
```

Two specific lines of code are highlighted with red boxes:

- The line `<p>Questions that need a response</p>` in the `map` function body is highlighted.
- The line `AdminMessages.tsx` in the Explorer sidebar is highlighted.

Then the AdminMessage will be imported automatically.



The screenshot shows a Visual Studio Code interface. The left sidebar displays a file tree under 'REACT-LIBRARY' with files like 'node\_modules', 'public', 'src', 'Auth', 'Images', 'layouts', 'BookCheckoutPage', 'HomePage', 'ManageLibraryPage', 'components', 'AdminMessage.tsx', 'AdminMessages.tsx', 'ManageLibraryPage.tsx', 'MessagesPage', 'NavbarAndFooter', 'SearchBooksPage', and 'ShopPage'. The 'AdminMessages.tsx' file is currently selected and open in the main editor area. The code in the editor is:

```
src > layouts > ManageLibraryPage > components > AdminMessages.tsx > AdminMessages > messages.map() callback
  75   <p>{httpError}</p>
  76   </div>
  77   );
  78 }
// pagination const
const paginate = (pageNumber: number) => setCurrentPage(pageNumber);

return(
  <div className='mt-3'>
    /* if there is messages: */
    {messages.length > 0 ?
      <>
        <h5>Pending Q/A: </h5>
        {messages.map(message => (
          <AdminMessage message={message} key={message.id} />
        ))}
      </>
    }
  </div>
)
```

A red box highlights the line of code: `<AdminMessage message={message} key={message.id} />` at line 89. The status bar at the bottom of the editor shows '1, U' and '3, M'.

React App

Home | Okta Developer

okta-dev-65756343 - Sign In

localhost:3000/admin

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...

IAC Read Home Search Books Shelf

## Manage Library

Add new book Change quantity Messages

Pending Q/A:

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

Response:  
Description

Submit Response

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

With admin user account logged in, click on messages tab, we can get all the questions that need a response.

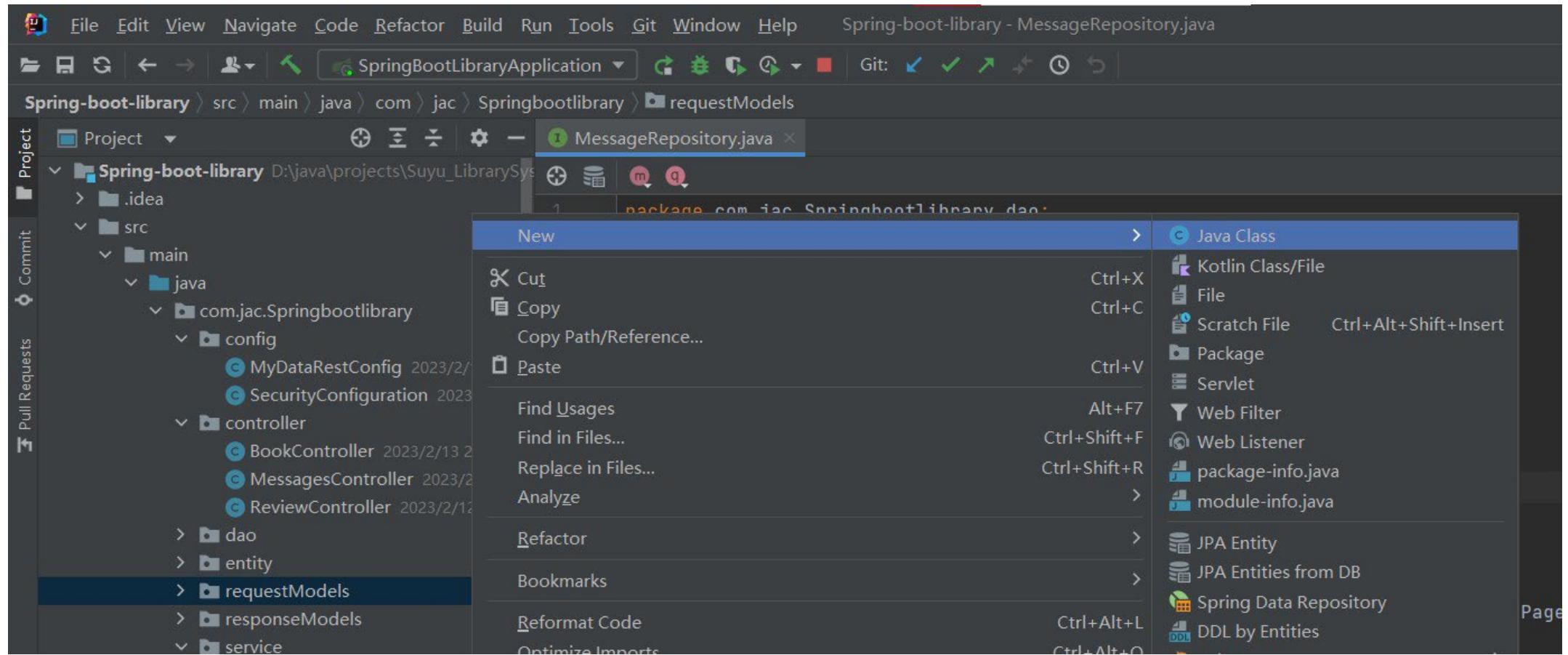
---

---

---

## **9. SPRINGBOOT ADMIN QUESTION REQUEST**

Step1 In requestModels package, create a new request model, named it AdminQuestionRequest.



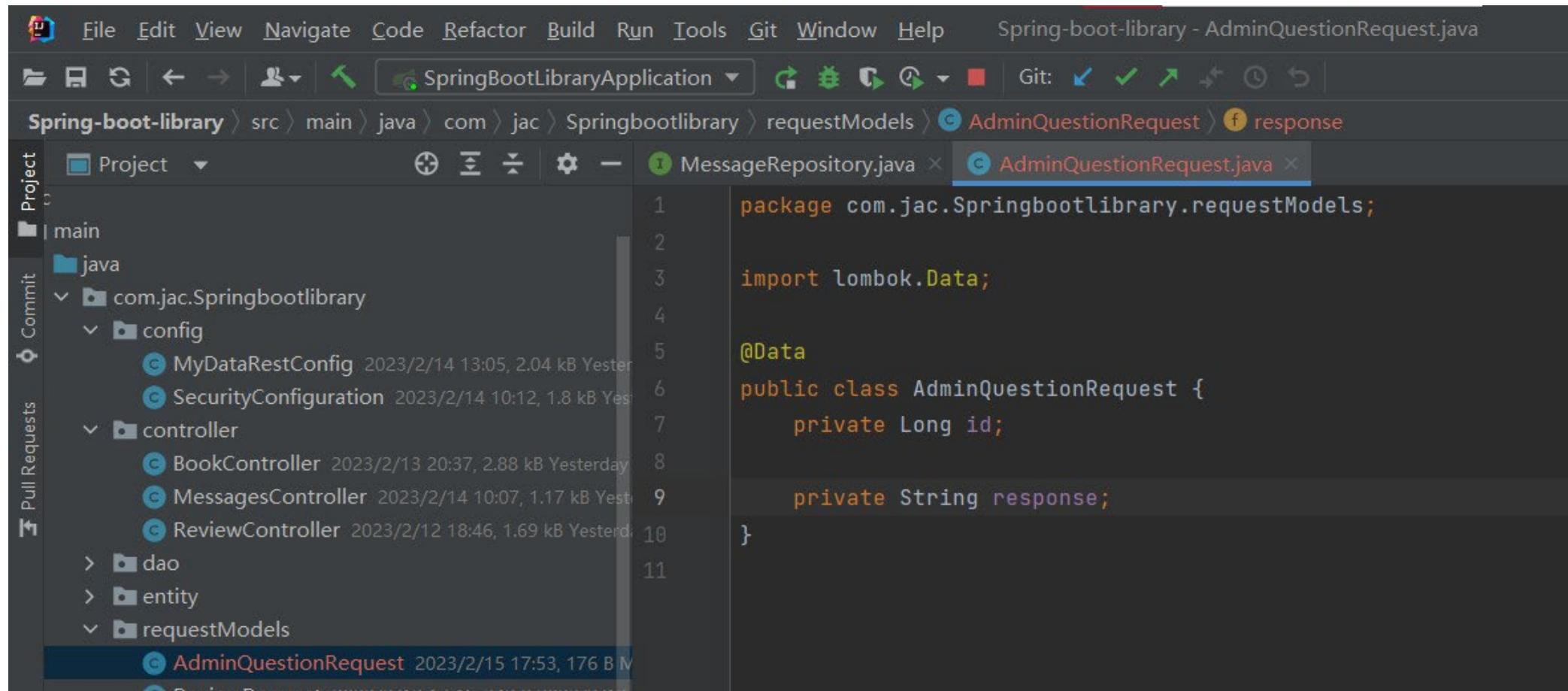
Choose “Class”, input the class name “AdminQuestionRequest”.

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

- Project Bar:** Shows "Spring-boot-library - MessageRepository.java".
- Toolbars:** Standard IntelliJ toolbars for file operations, navigation, and git status.
- File Path:** Spring-boot-library > src > main > java > com > jac > Springbootlibrary > dao > MessageRepository.java
- Project View:** Displays the project structure under "Spring-boot-library".
- Code Editor:** Shows the content of `MessageRepository.java`. The cursor is at the start of a new method definition.
- Code Completion Dialog:** A modal dialog titled "New Java Class" is open, listing options: Class (highlighted), Interface, Enum, and Annotation. The input field contains "AdminQuestionRequest".
- Code Snippet:** The generated code snippet is:

```
package com.jac.Springbootlibrary.dao;  
import com.jac.Springbootlibrary.entity.Message;  
import org.springframework.data.domain.Page;  
import org.springframework.data.domain.Pageable;  
import org.springframework.data.jpa.repository.JpaRepository;  
import org.springframework.web.bind.annotation.RequestParam;  
  
public interface Me  
    // find all mess  
    // find the clo  
    Page<Message> findByClosed(@RequestParam("closed") boolean closed, Pageable  
}
```

## Step 2, add `@Data` annotation, and add its properties.



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 - AdminQuestionRequest.java
- Toolbar:** Includes icons for file operations, navigation, and Git integration.
- Breadcrumbs:** Spring-boot-library > src > main > java > com > jac > Springbootlibrary > requestModels > AdminQuestionRequest > response
- Project Explorer:** Shows the project structure under "main":
  - java
    - com.jac.Springbootlibrary
      - config
        - MyDataRestConfig (2023/2/14 13:05, 2.04 kB)
        - SecurityConfiguration (2023/2/14 10:12, 1.8 kB)
      - controller
        - BookController (2023/2/13 20:37, 2.88 kB)
        - MessagesController (2023/2/14 10:07, 1.17 kB)
        - ReviewController (2023/2/12 18:46, 1.69 kB)
      - dao
      - entity
      - requestModels
        - AdminQuestionRequest (2023/2/15 17:53, 176 kB)
  - Code Editor:** The AdminQuestionRequest.java file is open, showing the following code:

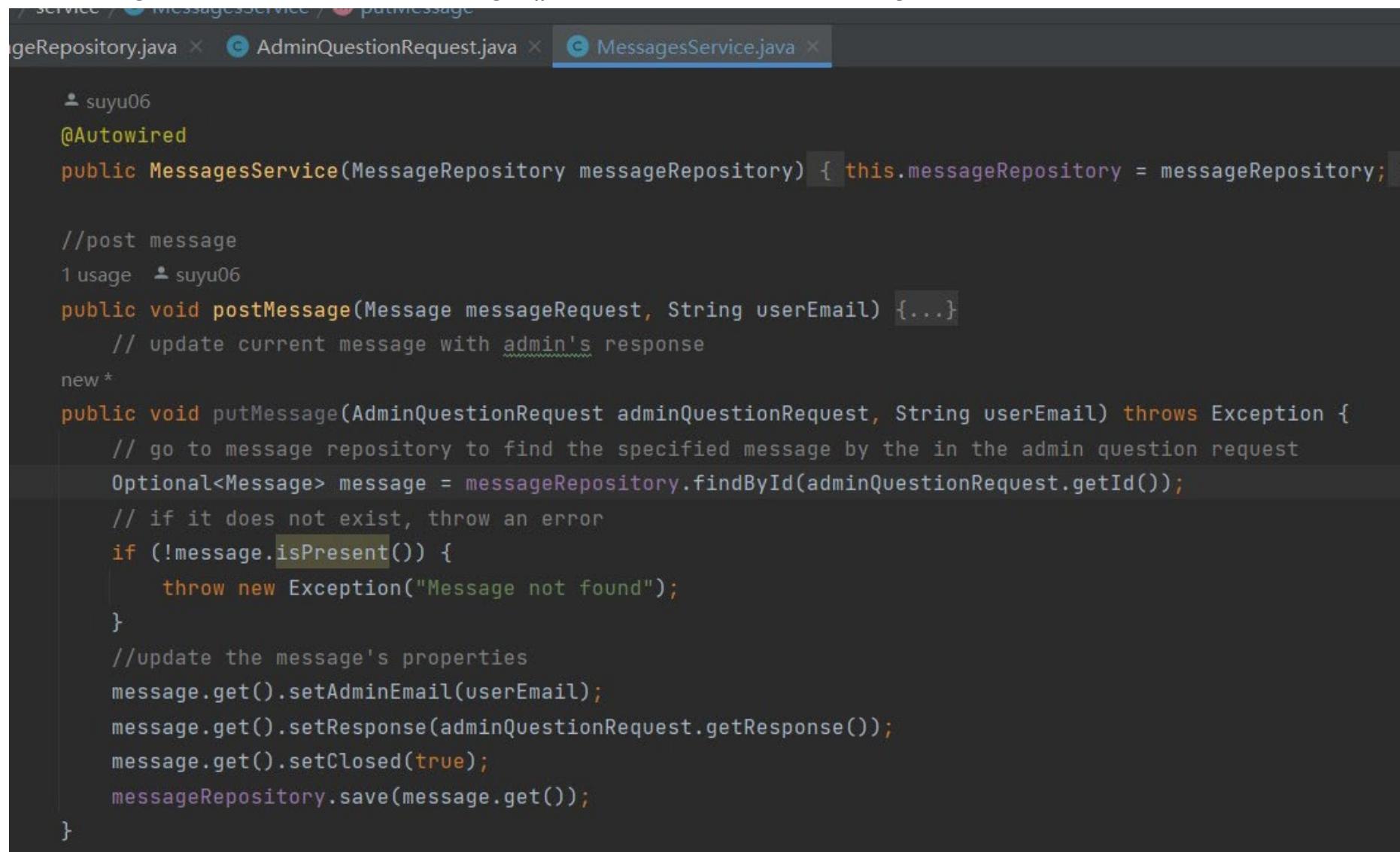
```
package com.jac.Springbootlibrary.requestModels;

import lombok.Data;

@Data
public class AdminQuestionRequest {
    private Long id;

    private String response;
}
```

Step 3, Go to MessageService class, right underneath our public void post message, create a putMessage() to update our messages with admin's response.



The screenshot shows a Java code editor with several tabs at the top: 'MessageRepository.java', 'AdminQuestionRequest.java', and 'MessagesService.java'. The 'MessagesService.java' tab is active, highlighted with a blue bar. The code is written in Java and defines a class 'MessagesService' with two methods: 'postMessage' and 'putMessage'. The 'postMessage' method takes a 'Message' object and a 'String userEmail' parameter. It contains a comment indicating it will update the current message with the admin's response. The 'putMessage' method takes an 'AdminQuestionRequest' object and a 'String userEmail' parameter, throwing an 'Exception'. It first finds a message by its ID from the repository. If the message is not present, it throws an exception. Otherwise, it updates the message's properties (admin email, response, closed status) and saves it back to the repository. The code uses annotations like '@Autowired' and imports from 'com.message.message' and 'java.util.Optional'.

```
suyu06
@Autowired
public MessagesService(MessageRepository messageRepository) { this.messageRepository = messageRepository; }

//post message
1 usage  suyu06
public void postMessage(Message messageRequest, String userEmail) {...}
    // update current message with admin's response

new *
public void putMessage(AdminQuestionRequest adminQuestionRequest, String userEmail) throws Exception {
    // go to message repository to find the specified message by the in the admin question request
    Optional<Message> message = messageRepository.findById(adminQuestionRequest.getId());
    // if it does not exist, throw an error
    if (!message.isPresent()) {
        throw new Exception("Message not found");
    }
    //update the message's properties
    message.get().setAdminEmail(userEmail);
    message.get().setResponse(adminQuestionRequest.getResponse());
    message.get().setClosed(true);
    messageRepository.save(message.get());
}
```

```
bootlibrary / controller / MessagesController.java path: message
MessageRepository.java AdminQuestionRequest.java MessagesService.java MessagesController.java
20  public MessagesController(MessagesService messagesService) { this.messagesService = messagesService; }
21
22 // post a message
23 // suyu06
24 @PostMapping("/secure/add/message")
25 public void postMessage(@RequestHeader(value="Authorization") String token,
26                         @RequestBody Message messageRequest) {...}
27
28 //update a message with admin's response
29 new *
30 @PutMapping("/secure/admin/message")
31 public void putMessage(@RequestHeader(value="Authorization") String token,
32                         @RequestBody AdminQuestionRequest adminQuestionRequest) throws Exception {
33     //extract user email and user type from token
34     String userEmail = ExtractJWT.payloadJWTExtraction(token, extraction: "\"sub\"");
35     String admin = ExtractJWT.payloadJWTExtraction(token, extraction: "\"userType\"");
36     if (admin == null || !admin.equals("admin")) {
37         throw new Exception("Administration page only.");
38     }
39     // call putMessage() in messageService class
40     messagesService.putMessage(adminQuestionRequest, userEmail);
41 }
42 }
```

Step 4, Go to  
MessagesContr  
oller class, add  
a putMapping  
API to our  
putMessage.

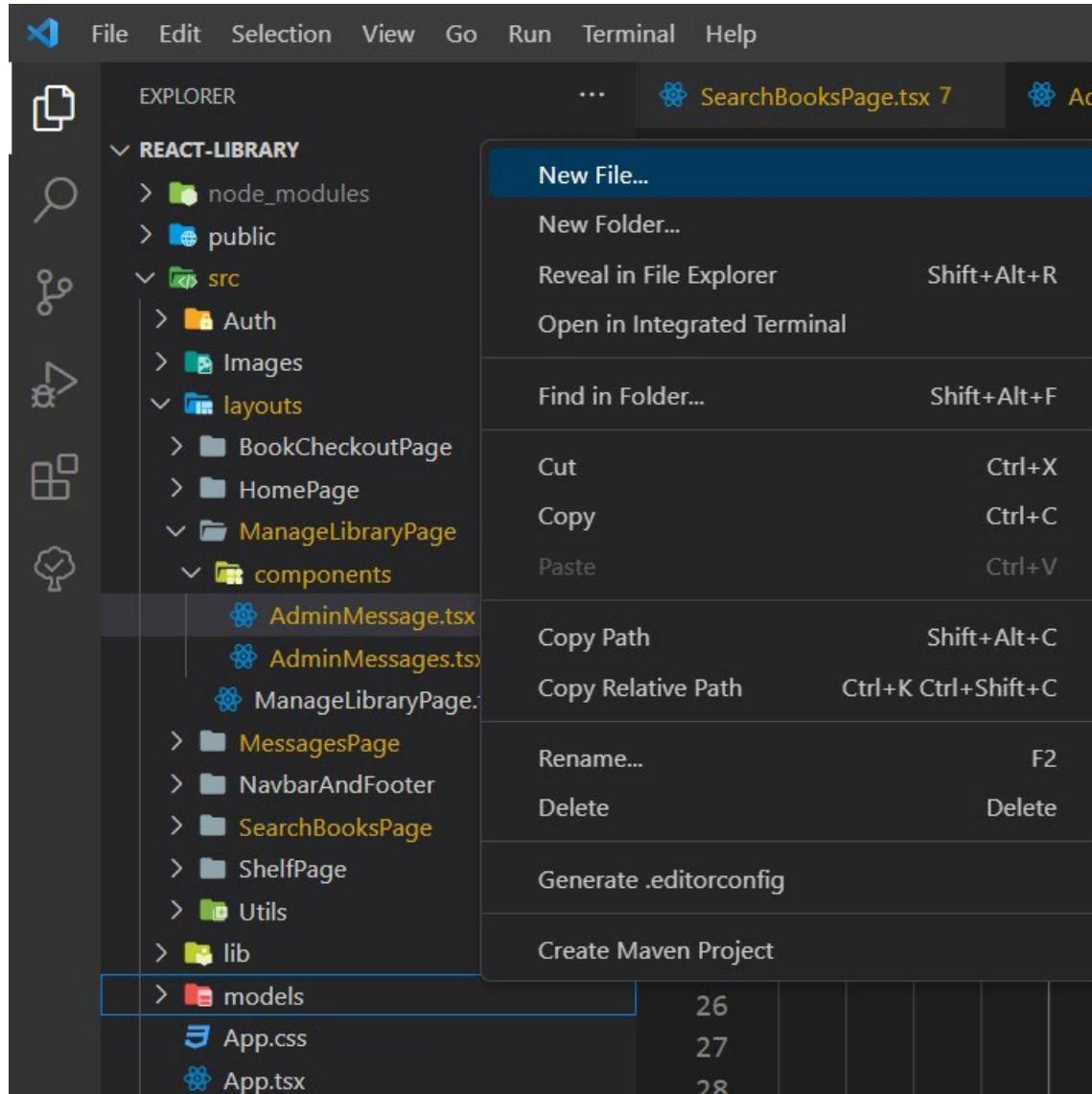
---

---

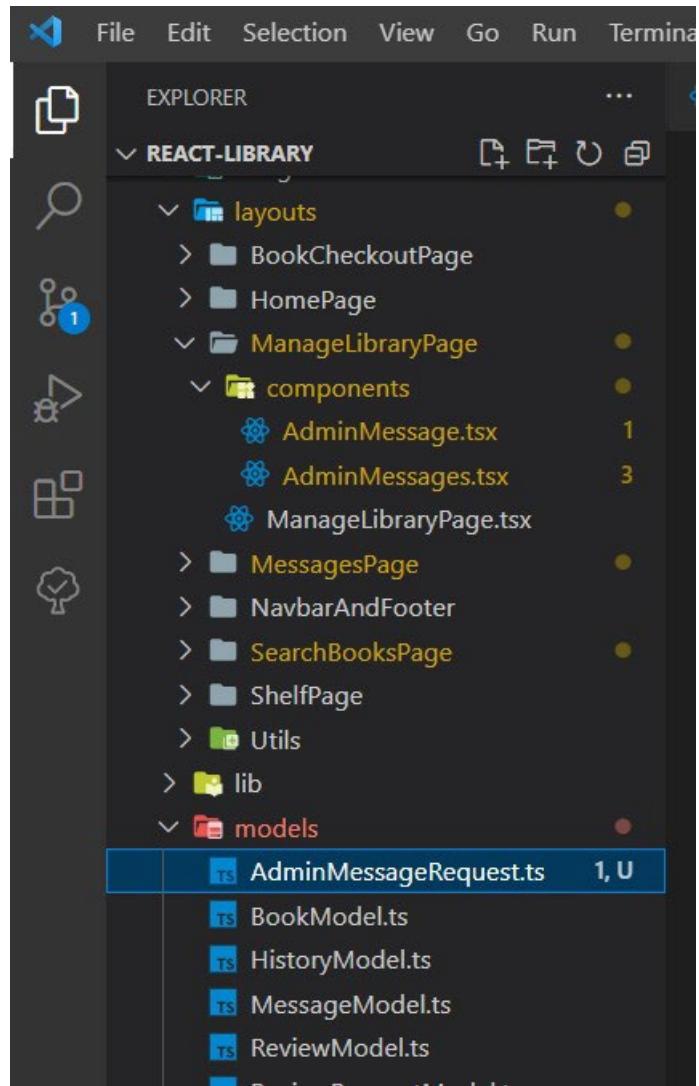
---

## 10. REACT ADMIN MESSAGES REQUEST



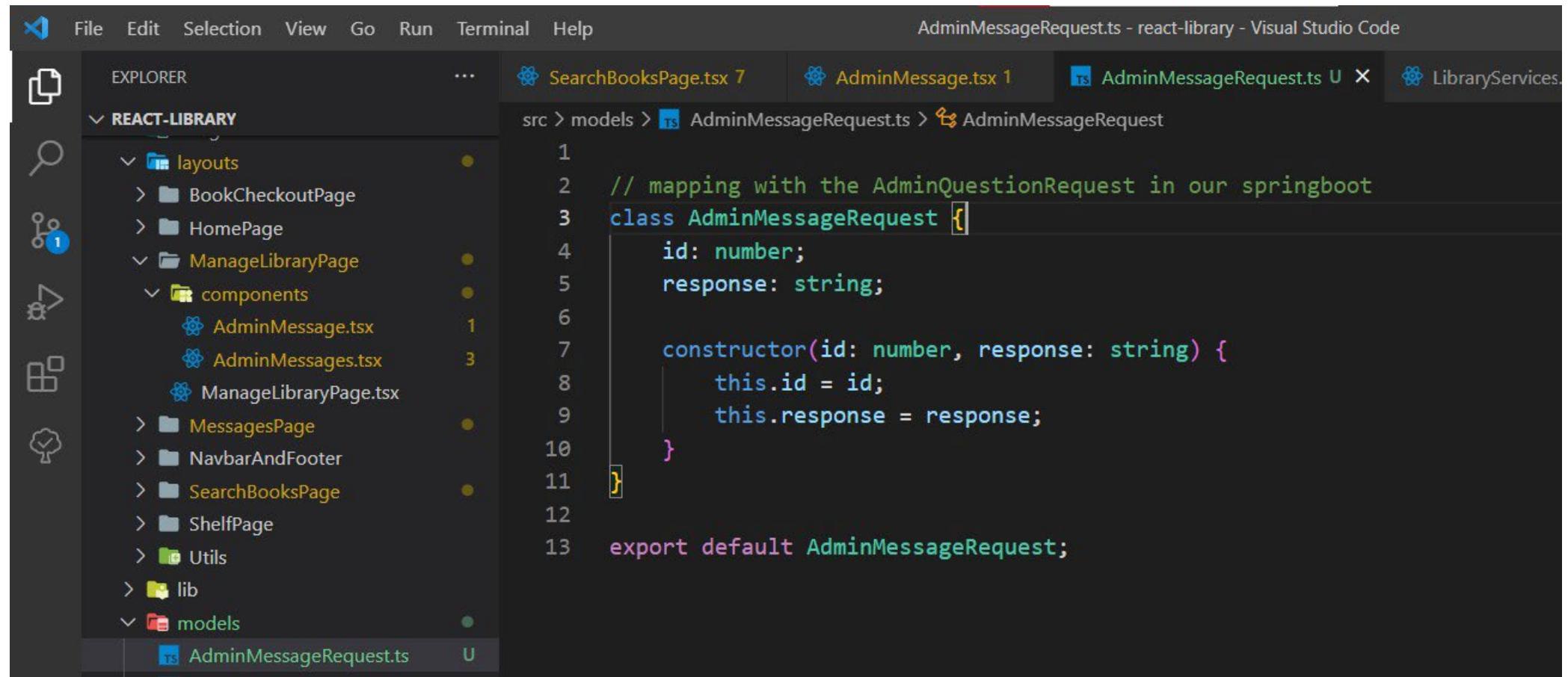


Step 1, So let's first go into our model's directory.  
And that's right click and say new file.



Name it AdminMessageRequest.ts.

Step 2, inside the class ,we will create two properties:  
ID of type number and a response of type string.



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

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** AdminMessageRequest.ts - react-library - Visual Studio Code
- Explorer:** Shows the project structure under REACT-LIBRARY, including layouts, ManageLibraryPage, components, and models. The AdminMessageRequest.ts file is selected in the models folder.
- Editor:** Displays the code for AdminMessageRequest.ts:

```
1 // mapping with the AdminQuestionRequest in our springboot
2 class AdminMessageRequest {
3     id: number;
4     response: string;
5
6     constructor(id: number, response: string) {
7         this.id = id;
8         this.response = response;
9     }
10
11 }
12
13 export default AdminMessageRequest;
```

---

---

---

# 11.REACT SUBMIT RESPONSE FUNCTION

Step1 add const [btnSubmit, setBtnSubmit] = useState(false);

The screenshot shows a dark-themed interface of the Visual Studio Code code editor. On the left is the Explorer sidebar, which lists the project structure under 'REACT-LIBRARY'. The 'src' folder contains 'layouts', 'ManageLibraryPage' (which has 'components' and 'MessagesPage'), 'SearchBooksPage', 'ShelfPage', 'Utils', 'lib', and 'models' (which contains 'AdminMessageRequest.ts' and 'BookModel.ts'). The 'AdminMessages.tsx' file is open in the main editor area. The code is a TypeScript component with several state hooks:

```
// oktaAuth
const { authState } = useOktaAuth();

// Normal Loading Pieces
const [isLoadingMessages, setIsLoadingMessages] = useState(true);
const [httpError, setHttpError] = useState(null);

// Messages endpoint State
const [messages, setMessages] = useState<MessageModel[]>([]);
const [messagesPerPage] = useState(5);

// Pagination
const [currentPage, setCurrentPage] = useState(1);
const [totalPages, setTotalPages] = useState(0);

// Recall useEffect
const [btnSubmit, setBtnSubmit] = useState(false);
```

Step 2, add btnSubmit to the useState array.

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

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** AdminMessages.tsx - react-library - Visual Studio Code
- Explorer:** Shows the project structure under "REACT-LIBRARY".
  - layouts
  - BookCheckoutPage
  - HomePage
  - ManageLibraryPage
    - components
      - AdminMessage.tsx
      - AdminMessages.tsx (highlighted)
      - ManageLibraryPage.tsx
    - MessagesPage
    - NavbarAndFooter
    - SearchBooksPage
    - ShelfPage
    - Utils
  - lib
  - models
- Code Editor:** AdminMessages.tsx (Line 63 is highlighted)

```
setTotalPages(messagesResponse.page.total);
}
//finish loadin process
setIsLoadingMessages(false);
}
//if fetching data meets error:
fetchUserMessages().catch((error: any) => {
    setIsLoadingMessages(false);
    setHttpError(error.message);
})
//each time turn to a new page, it will go to the
window.scrollTo(0, 0);
},[authState, currentPage, btnSubmit])
```

Step3, add an async function submitResponseToQuestion().

```
async function submitResponseToQuestion(id: number, response: string) {
  const url = `http://localhost:8080/api/messages/secure/admin/message`;
  // user is authenticated and both response request id and response exist:
  if (authState && authState?.isAuthenticated && id !== null && response !== '') {
    const messageAdminRequestModel: AdminMessageRequest = new AdminMessageRequest(id, response);
    const requestOptions = {
      method: 'PUT',
      headers: {
        Authorization: `Bearer ${authState?.accessToken?.accessToken}`,
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(messageAdminRequestModel)
    };
    // fetch the data
    const messageAdminRequestModelResponse = await fetch(url, requestOptions);
    // if fetching meets error
    if (!messageAdminRequestModelResponse.ok) {
      throw new Error('Something went wrong!');
    }
    //reset the state of the btn
    setBtnSubmit(!btnSubmit);
  }
}
```

---

---

---

## 12. REACT WRAP UP ADMIN RESOURCES



Step 1, in our AdminMessage.tsx, we need to pass in a new prop where this prop is going to be SubmitResponseToQuestion of type any.

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

- File Menu:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** AdminMessage.tsx - react-library - Visual Studio Code
- Explorer:** REACT-LIBRARY tree view showing the project structure. The AdminMessage.tsx file is selected.
- Search Bar:** SearchBooksPage.tsx 7
- Editor:** AdminMessage.tsx 1, M X (with a blue status bar icon)
- Code:** The code for AdminMessage.tsx is displayed, showing imports for useState and MessageModel, and an export for AdminMessage as a React.FC component.

```
src > layouts > ManageLibraryPage > components > AdminMessage.tsx > AdminMessage
1 import { useState } from "react";
2 import MessageModel from "../../../../../models/MessageModel";
3
4 export const AdminMessage: React.FC<{ message: MessageModel, submitResponseToQuestion: any }> = ({ message, submitResponseToQuestion }) => {
5   //useState
6   const [displayWarning, setDisplayWarning] = useState(false);
7   const [response, setResponse] = useState('');
8   // html/CSS part
9   return (
10     // go through each message by its id
11     <div>
12       <h3>{message.message}</h3>
13       <p>{message.message}</p>
14       <button onClick={() => submitResponseToQuestion(message.id)}>Submit Response</button>
15     </div>
16   );
17 }
```

## Step 2 Fix the error appeared for AdminMessages in AdminMessages.tsx,

The screenshot shows a code editor with several tabs at the top: Messages.tsx 1, AdminMessage.tsx 1, M, AdminMessages.tsx 3 X, and App.tsx. The AdminMessages.tsx 3 tab is active. The code is part of a component named AdminMessages. A tooltip is displayed over the line of code where the error occurred:

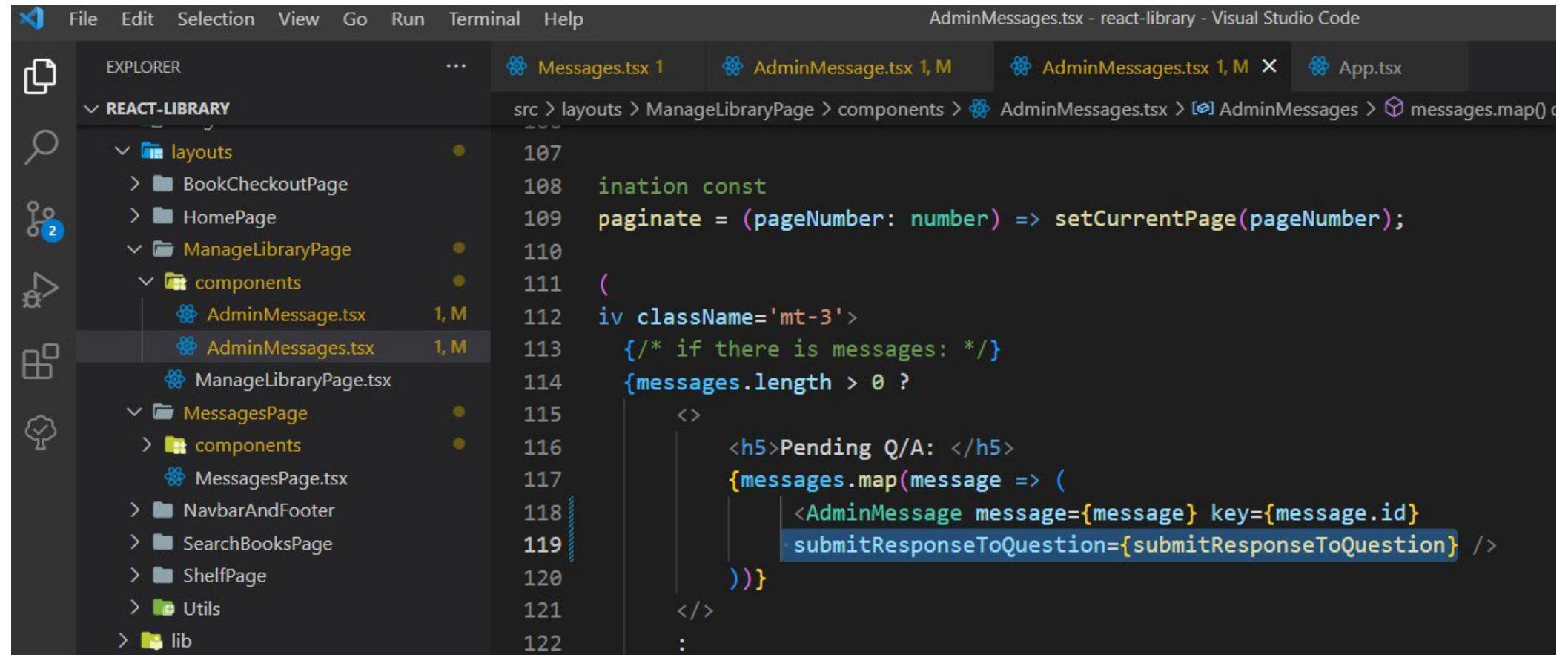
```
101     setBtnSubmit(!btnSubmit);
102 }
103 }
104
105
106
107
108 // pagination const
109 const paginate = (page: number) => {
110   return(
111     <div className='mt-4'>
112       {/* if there is no message */}
113       {messages.length === 0 ? <h5>Pending Questions</h5> : <h5>Recent Questions</h5>}
114       <>
115         <AdminMessage message={message} key={message.id} />
116       </>
117       {messages.map((message) => (
118         <AdminMessage message={message} key={message.id} />
119       ))}
120     </div>
121   :
122   //no messages
123   <h5>No pending Q/A</h5>
```

The tooltip contains the following text:

(alias) const AdminMessage: React.FC<{  
 message: MessageModel;  
 submitResponseToQuestion: any;  
}>  
import AdminMessage  
Property 'submitResponseToQuestion' is missing in type '{ message: MessageModel; key: number | undefined; }' but required in type '{ message: MessageModel; submitResponseToQuestion: any; }'. ts(2741)  
AdminMessage.tsx(4, 61): 'submitResponseToQuestion' is declared here.

Below the tooltip, there are links for "View Problem (Alt+F8)" and "Quick Fix... (Ctrl+.)".

Add "submitResponseToQuestion={submitResponseToQuestion}" as a property to AdminMessage.

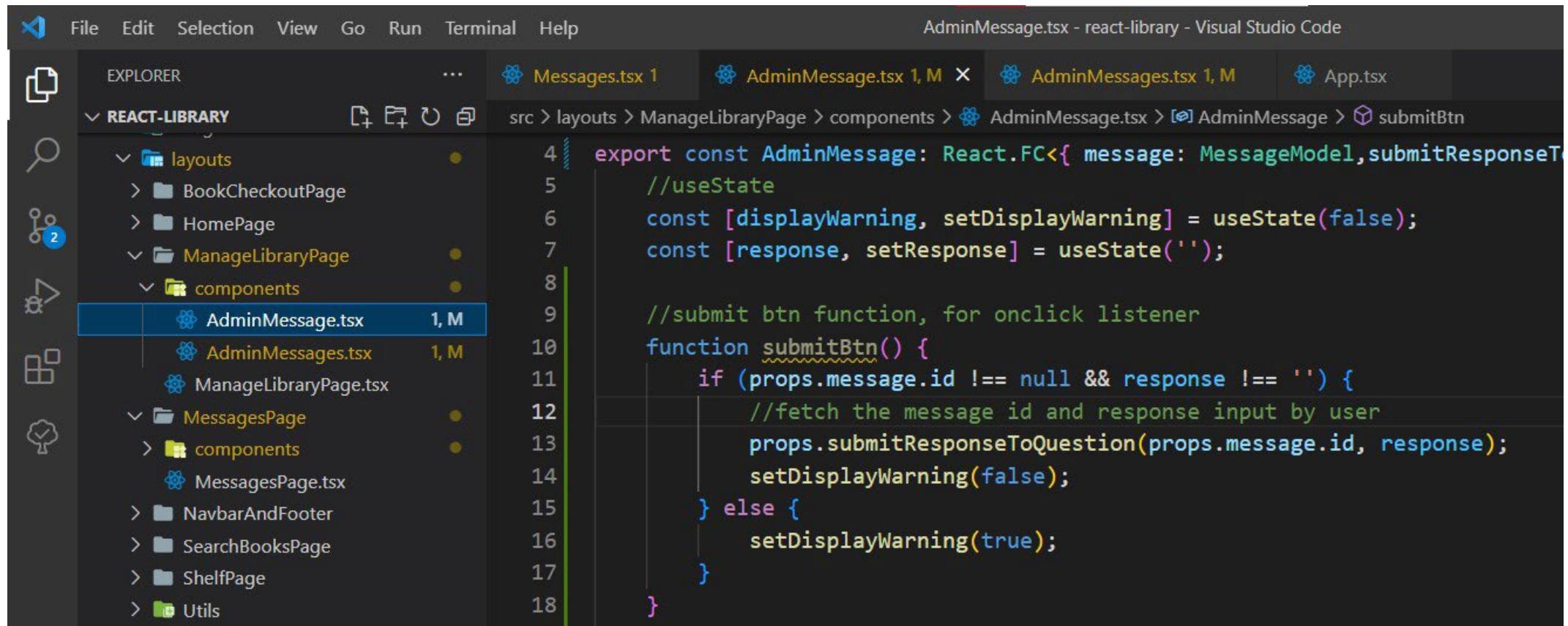


The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** AdminMessages.tsx - react-library - Visual Studio Code
- Explorer:** REACT-LIBRARY tree view showing the project structure. The AdminMessages.tsx file is selected.
- Editor:** The code for AdminMessages.tsx is displayed, showing a map operation over messages. A new property "submitResponseToQuestion" is being added to the AdminMessage component.

```
107
108   ination const
109   paginate = (pageNumber: number) => setCurrentPage(pageNumber);
110
111 (
112   iv className='mt-3'
113   /* if there is messages: */
114   {messages.length > 0 ?
115     <>
116       <h5>Pending Q/A: </h5>
117       {messages.map(message => (
118         <AdminMessage message={message} key={message.id}
119           submitResponseToQuestion={submitResponseToQuestion} />
120       ))}
121     </>
122   :
```

## Step 3, AdminMessage.tsx, create submitBtn().

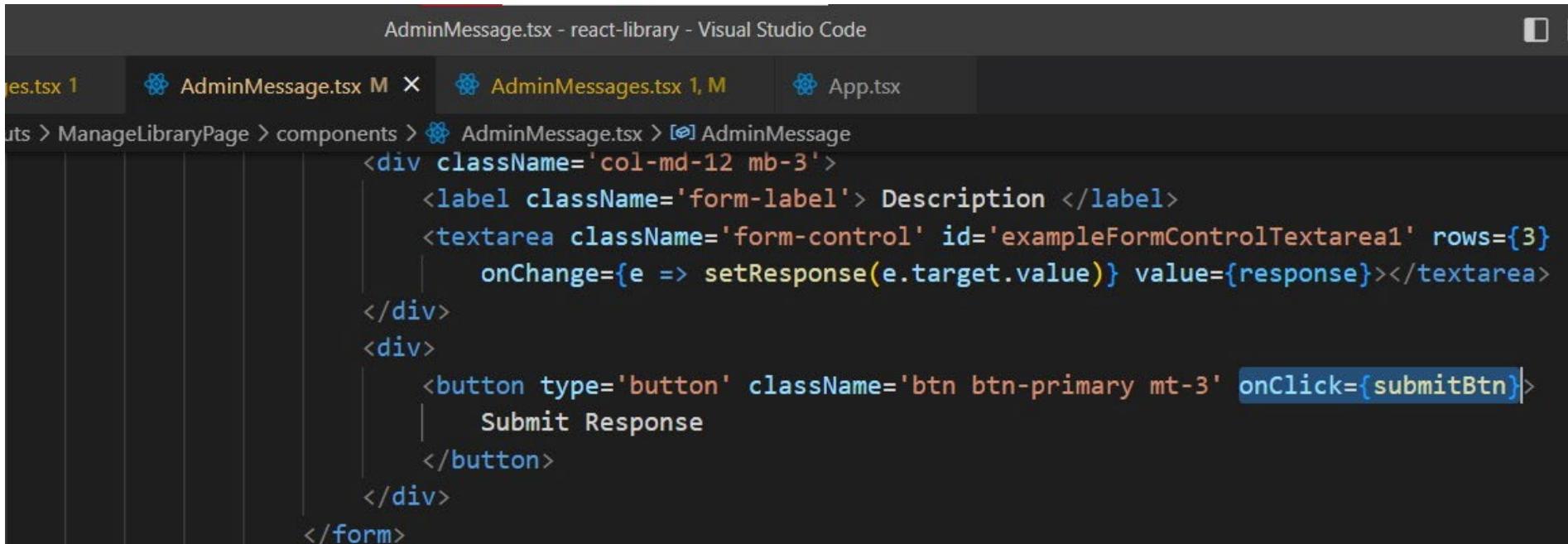


The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help.
- Tab Bar:** AdminMessage.tsx 1 M, AdminMessages.tsx 1 M, App.tsx.
- Explorer Bar (Left):** Shows the project structure under "REACT-LIBRARY".
  - layouts: BookCheckoutPage, HomePage.
  - ManageLibraryPage: component (AdminMessage.tsx 1 M, AdminMessages.tsx 1 M), MessagesPage (MessagesPage.tsx).
  - MessagesPage: component (MessagesPage.tsx).
  - NavbarAndFooter, SearchBooksPage, ShelfPage, Utils.
- Code Editor (Right):** The AdminMessage.tsx file content is displayed.

```
4 export const AdminMessage: React.FC<{ message: MessageModel, submitResponseToQuestion: (id: string, response: string) => void }> = ({ message, submitResponseToQuestion }) => {
5   //useState
6   const [displayWarning, setDisplayWarning] = useState(false);
7   const [response, setResponse] = useState('');
8
9   //submit btn function, for onclick listener
10  function submitBtn() {
11    if (props.message.id !== null && response !== '') {
12      //fetch the message id and response input by user
13      props.submitResponseToQuestion(props.message.id, response);
14      setDisplayWarning(false);
15    } else {
16      setDisplayWarning(true);
17    }
18  }
}
```

Step 4, add onClick listener to btn “submit Response”.



The screenshot shows a Visual Studio Code interface with the title bar "AdminMessage.tsx - react-library - Visual Studio Code". The tab bar includes "Index.tsx 1", "AdminMessage.tsx M X", "AdminMessages.tsx 1, M", and "App.tsx". The code editor displays the following TypeScript code:

```
AdminMessage.tsx - react-library - Visual Studio Code
Index.tsx 1 AdminMessage.tsx M X AdminMessages.tsx 1, M App.tsx

AdminMessage.tsx
  AdminMessage
    <div className='col-md-12 mb-3'>
      <label className='form-label'> Description </label>
      <textarea className='form-control' id='exampleFormControlTextarea1' rows={3}>
        onChange={e => setResponse(e.target.value)} value={response}</textarea>
    </div>
    <div>
      <button type='button' className='btn btn-primary mt-3' onClick={submitBtn}>
        Submit Response
      </button>
    </div>
  </form>
```

The `onClick` attribute for the button is highlighted with a blue selection bar.

A screenshot of a web browser window. The address bar shows the URL `localhost:3000/login`. The page itself is a login form for a service called "JAC Read". The form includes fields for "Username" (containing `adminuser@email.com`) and "Password" (containing a series of dots). There is also a "Remember me" checkbox and a "Sign In" button. A large blue arrow points from the text "Log in as admin." to the "Sign In" button.

React App

Home | Okta Developer

localhost:3000/login

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

JAC Read Home Search Books

Sign In

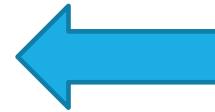
Username  
adminuser@email.com

Password  
\*\*\*\*\*

Remember me

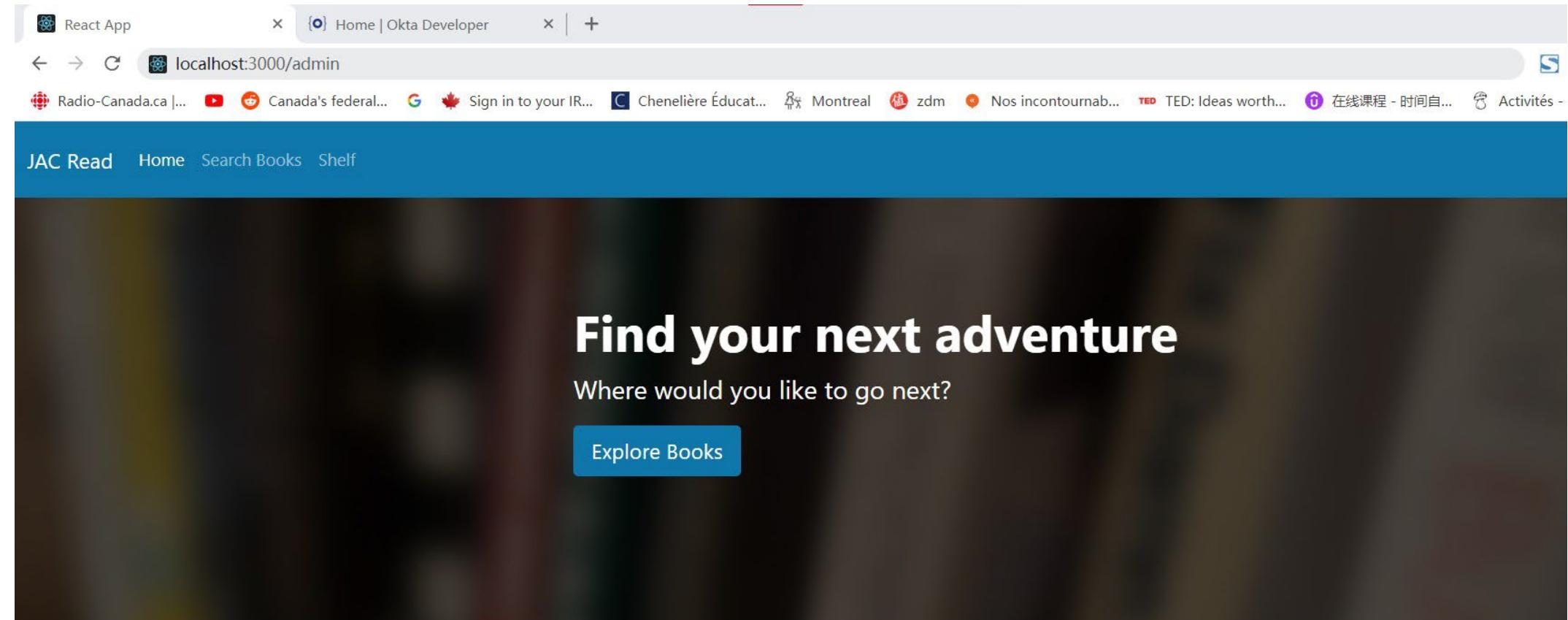
**Sign In**

Need help signing in?



Log in as  
admin.

Go to <http://localhost:3000/admin>



Find your next "I stayed up too late reading" book



Click on “Messages” tab, type nothing , click “submit response” button, **alert banner** will appear.

React App x Home | Okta Developer x | +

localhost:3000/admin

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

## Manage Library

Add new book Change quantity Messages

Pending Q/A:

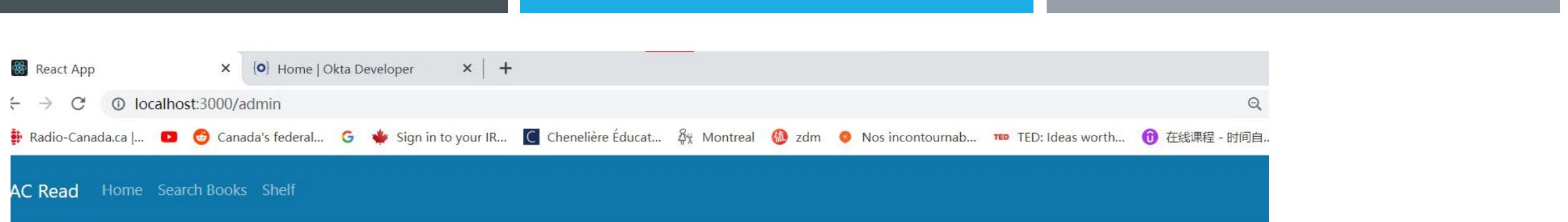
**Case #1: Example title for message**  
testuser2@email.com  
what's your question?

Response:

All fields must be filled out.

Description

Submit Response



## Manage Library

[Add new book](#)[Change quantity](#)[Messages](#)

### Pending Q/A:

#### Case #1: Example title for message

testuser2@email.com

what's your question?

#### Response:

Description

Here is the response

[Submit Response](#)

In description box for case #1, input some words, click on “submit response”.

JAC Read Home Search Books Shelf

## Manage Library

Add new book Change quantity Messages

Pending Q/A:

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

Response:

Description

Submit Response

Case #1 will disappear.  
Because it now has a response, so it is no longer a pending question, but a closed question.

Now checkout our database, the first message's data has updated, now it has a response and been in the status of “closed”.

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

- Navigator:** Shows the database schema with the `reactlibrarydatabase` database expanded, revealing tables like `book`, `checkout`, `history`, `messages`, and `review`.
- SQL Editor:** Displays the query `SELECT * FROM reactlibrarydatabase.messages;`.
- Result Grid:** Shows the data from the `messages` table in a grid format. The columns are `id`, `user_email`, `title`, `question`, `admin_email`, `response`, and `closed`. The data is as follows:

	<code>id</code>	<code>user_email</code>	<code>title</code>	<code>question</code>	<code>admin_email</code>	<code>response</code>	<code>closed</code>
1	1	testuser2@email.com	Example title for message	what's your question?	adminuser@email.com	Here is the response	1
2	2	testuser2@email.com	test title	test question	NULL	NULL	0
3	3	testuser2@email.com	test move banner	successful or not	NULL	NULL	0

A screenshot of a web browser window. The address bar shows the URL `localhost:3000/login`. The page title is "Home | Okta Developer". The browser interface includes standard controls like back, forward, and search. Below the header, there's a navigation bar with links for "JAC Read", "Home", and "Search Books". The main content area displays a "Sign In" form. The "Username" field contains `testuser2@email.com`. The "Password" field is filled with six asterisks. A "Remember me" checkbox is checked. A large blue "Sign In" button is at the bottom of the form. Below the button, a link says "Need help signing in?".

Sign In

Username  
testuser2@email.com

Password  
\*\*\*\*\*

Remember me

**Sign In**

Need help signing in?

Now, log in as  
testuser2.

React App    Home | Okta Developer    localhost:3000/messages

Radio-Canada.ca |...    Canada's federal...    Sign in to your IR...    Chenelière Éducat...    Montreal    zdm    Nos incontournab...

JAC Read    Home    Search Books    Shelf

Submit Question    Q/A Response/Pending

Current Q/A:

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

---

**Response:**  
adminuser@email.com (admin)  
Here is the response

Go to messages page, we will see, the response of Case #1 is showed there.



The other two questions still has no response.

A screenshot of a web browser window titled "React App" showing three messages from an admin user. The browser's address bar indicates the URL is "localhost:3000/messages".

**Message 1:**  
Response:  
adminuser@email.com (admin)  
Here is the response

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

**Message 3:**  
Response:  
*Pending response from administration. Please be patient.*

**Message 4:**  
Case #3: test move banner  
testuser2@email.com  
successful or not

**Message 5:**  
Response:  
*Pending response from administration. Please be patient.*