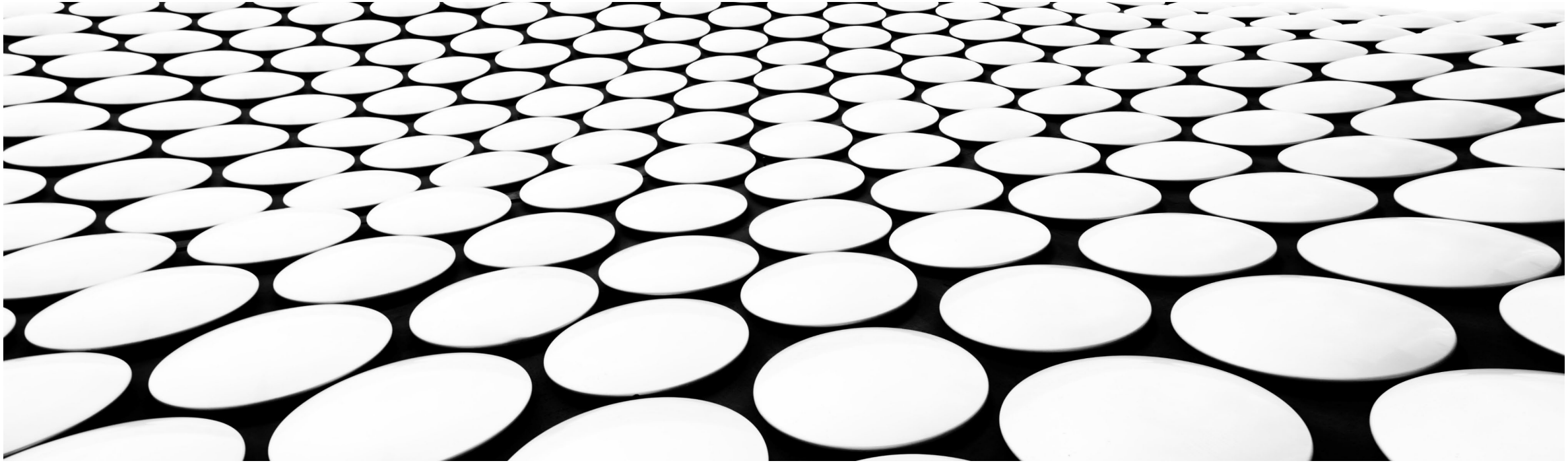
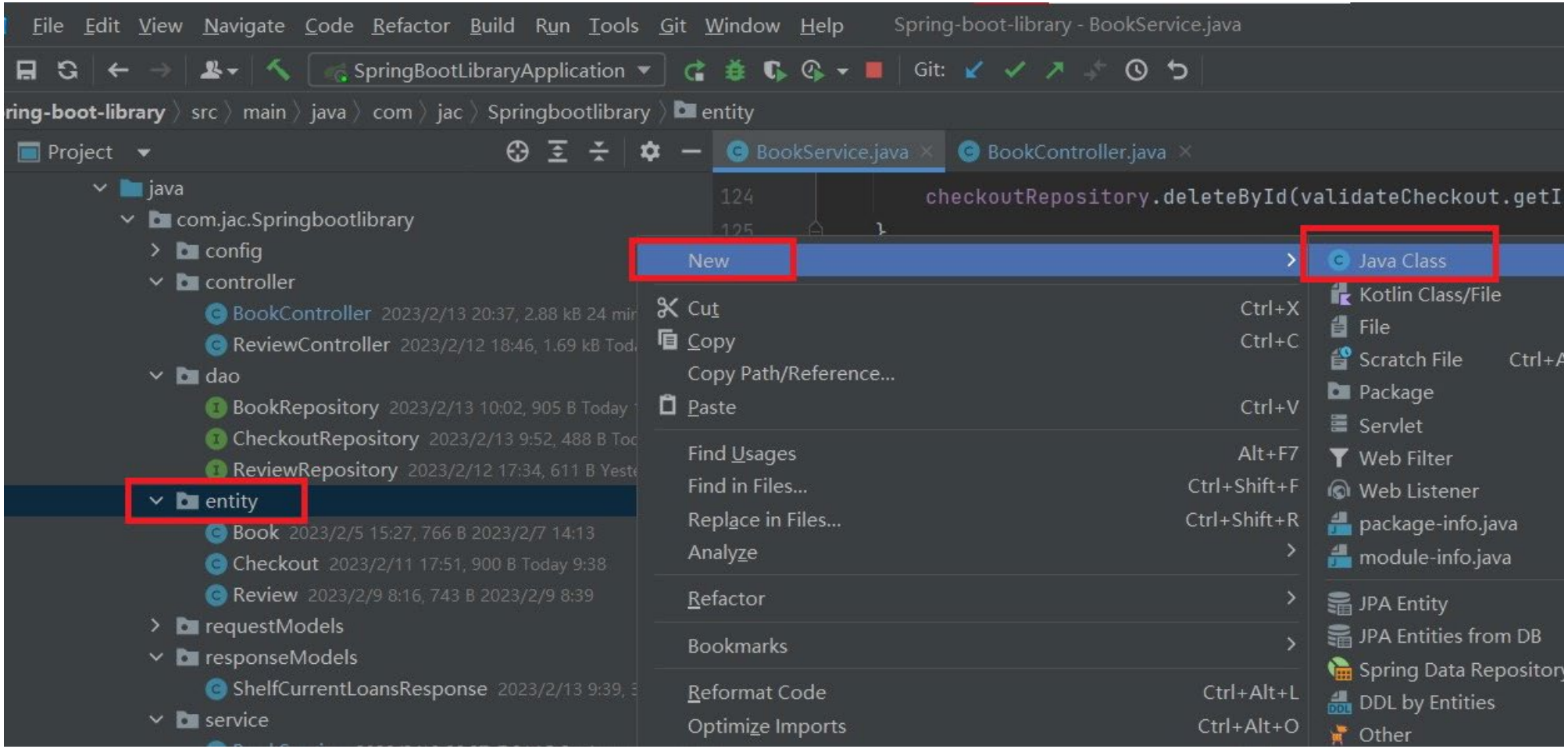
S17 HISTORY PAGE

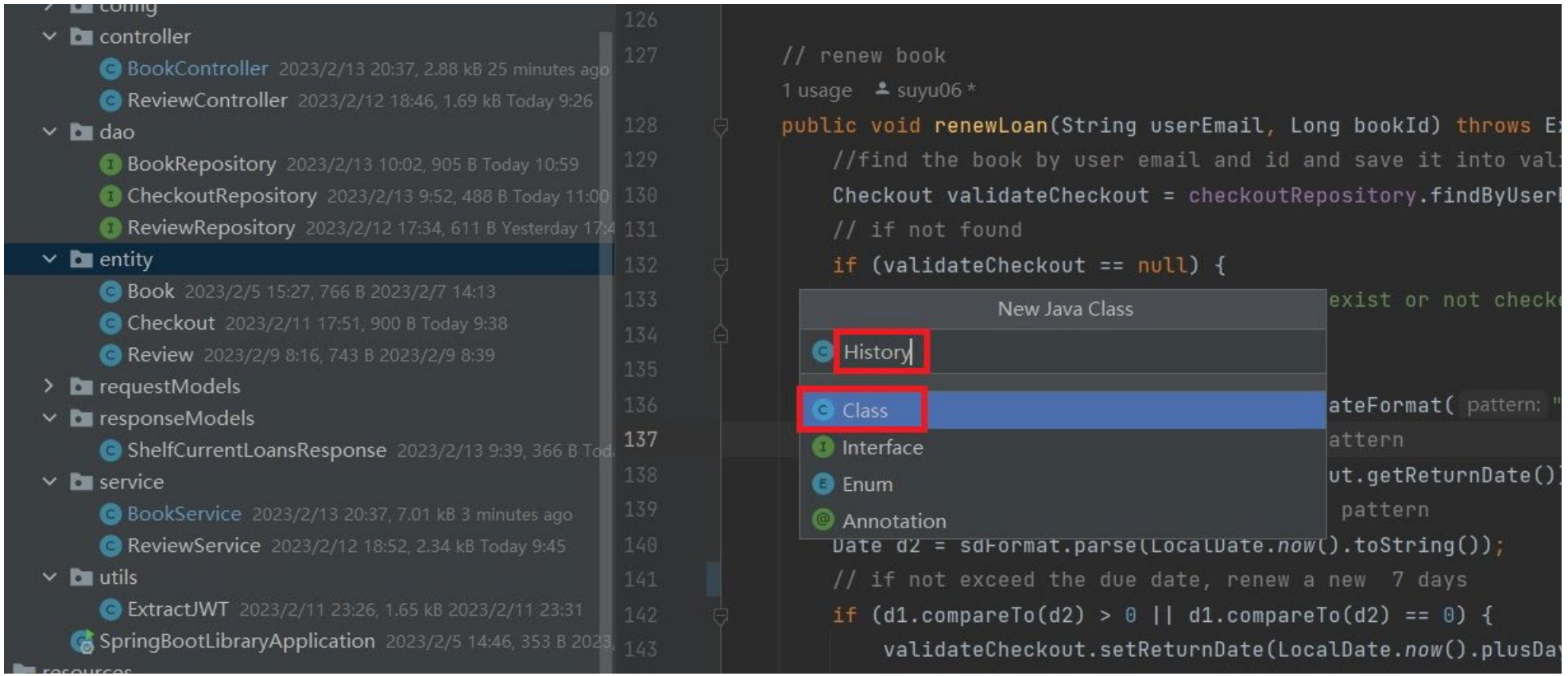


1. SPRINGBOOT HISTORY ENTITY

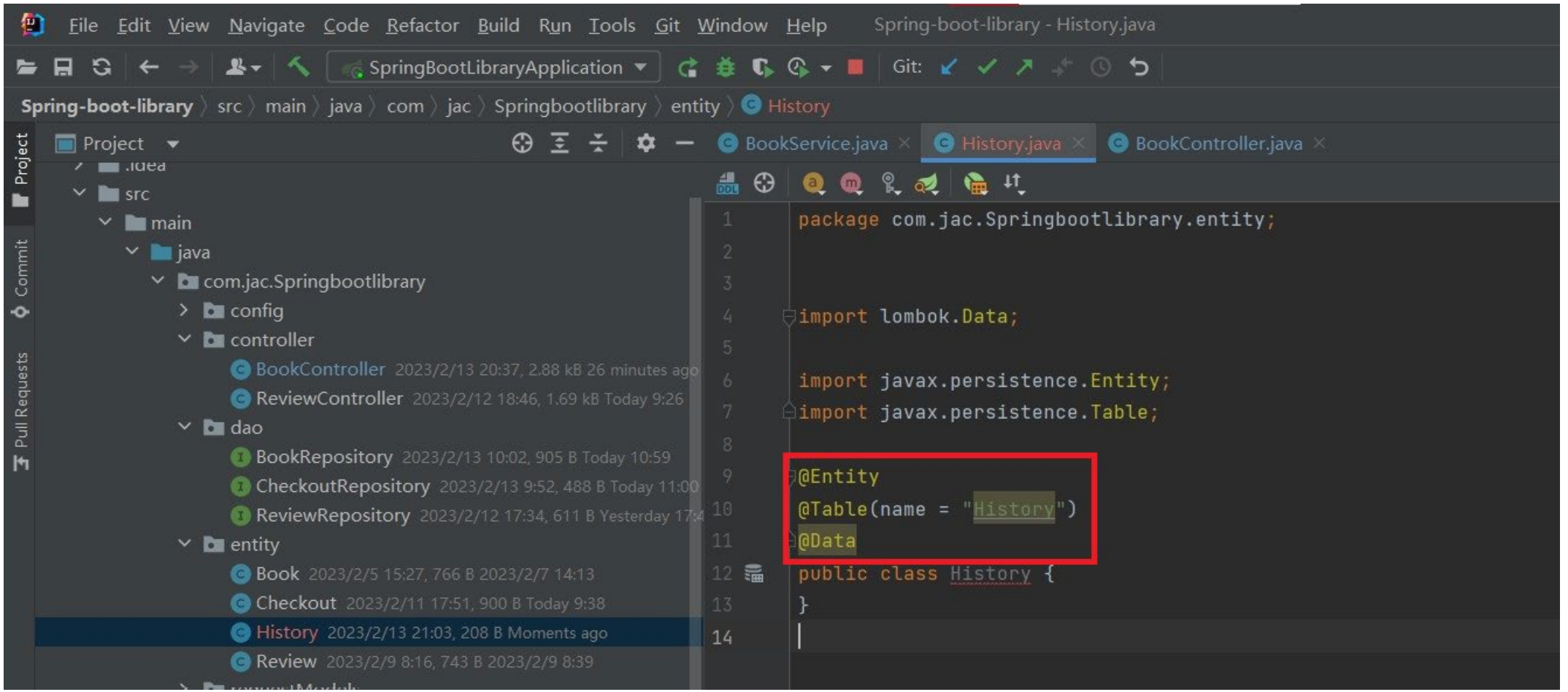
Step 1, right click on “entity” packge, choose New->Java Class



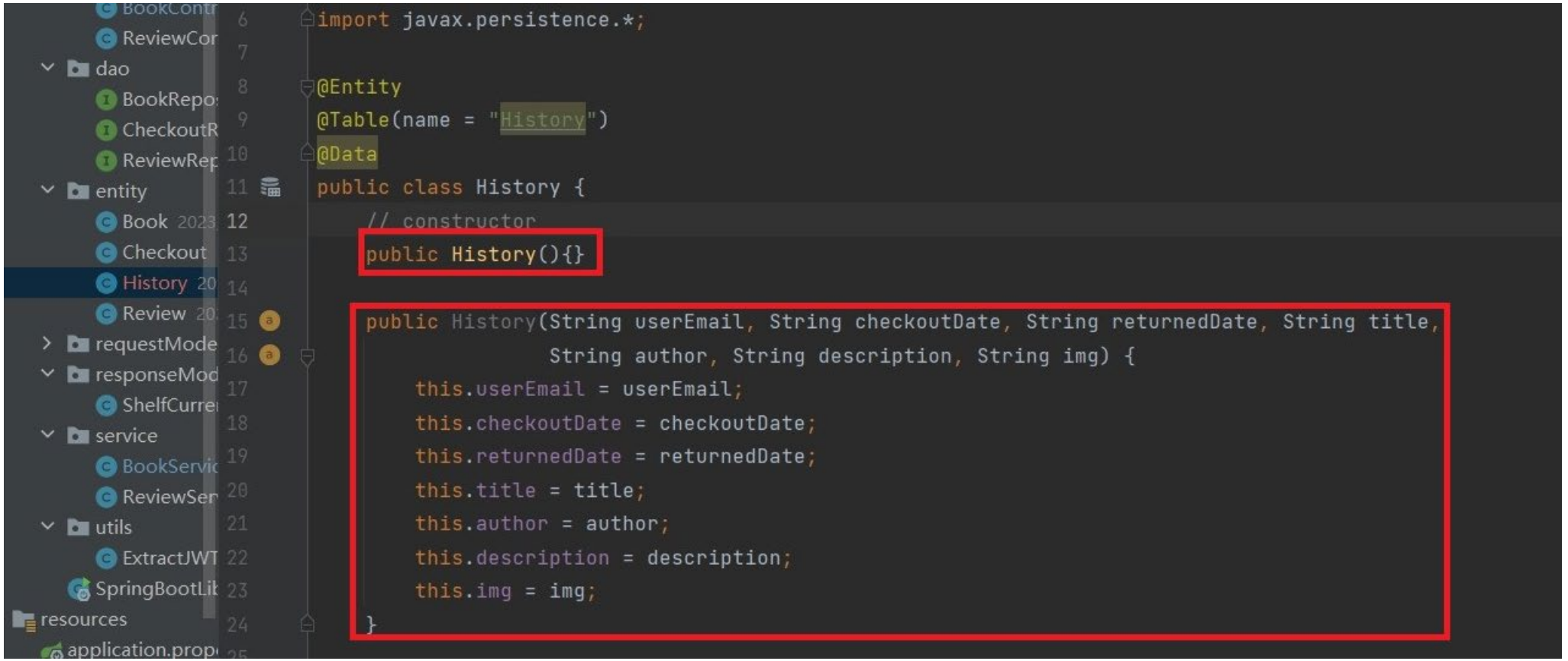
Choose “Class” , input the name “History”.

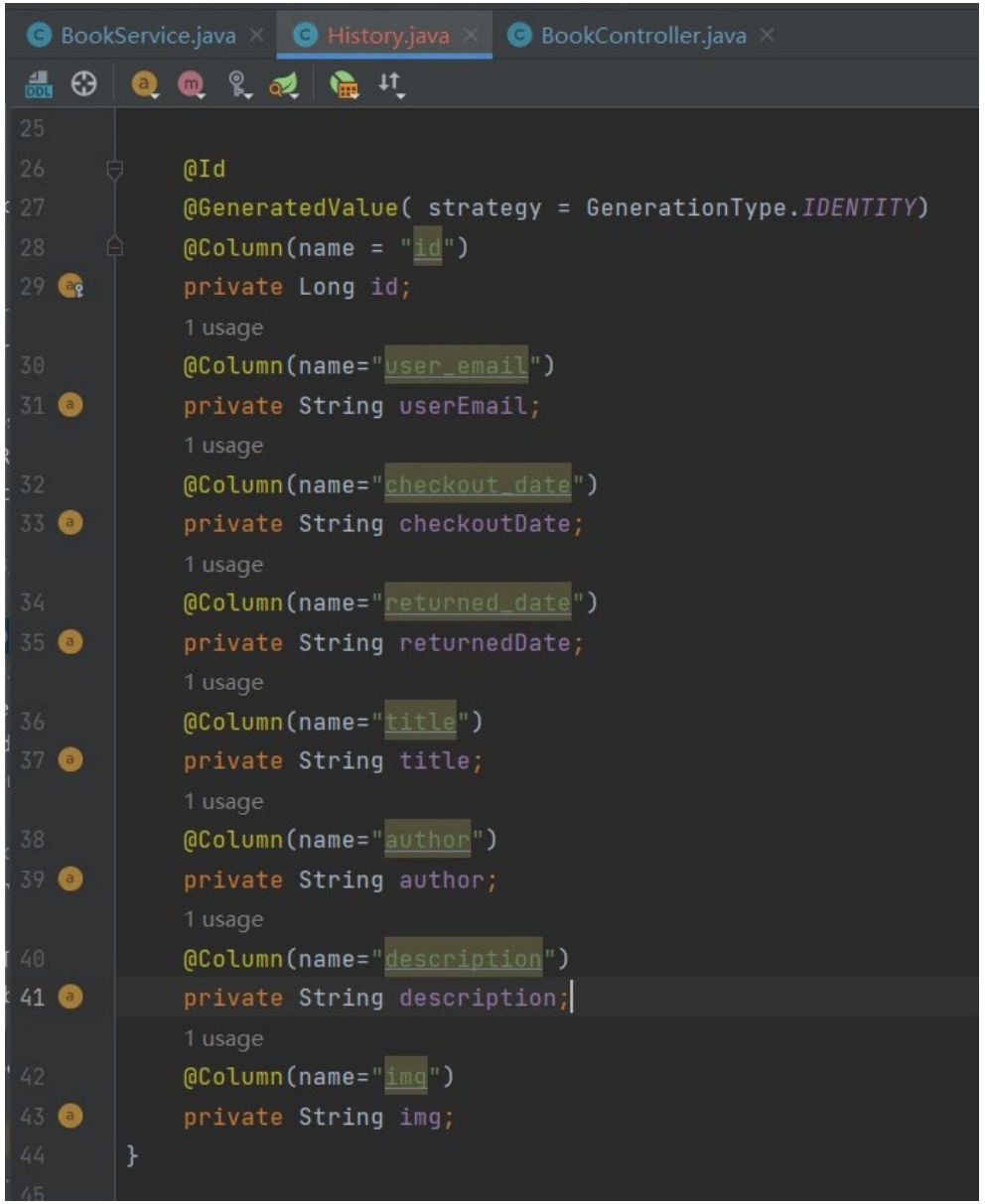


Step 2 add annotation: @Entity, @Table, @Data.



Step 3 create constructor.

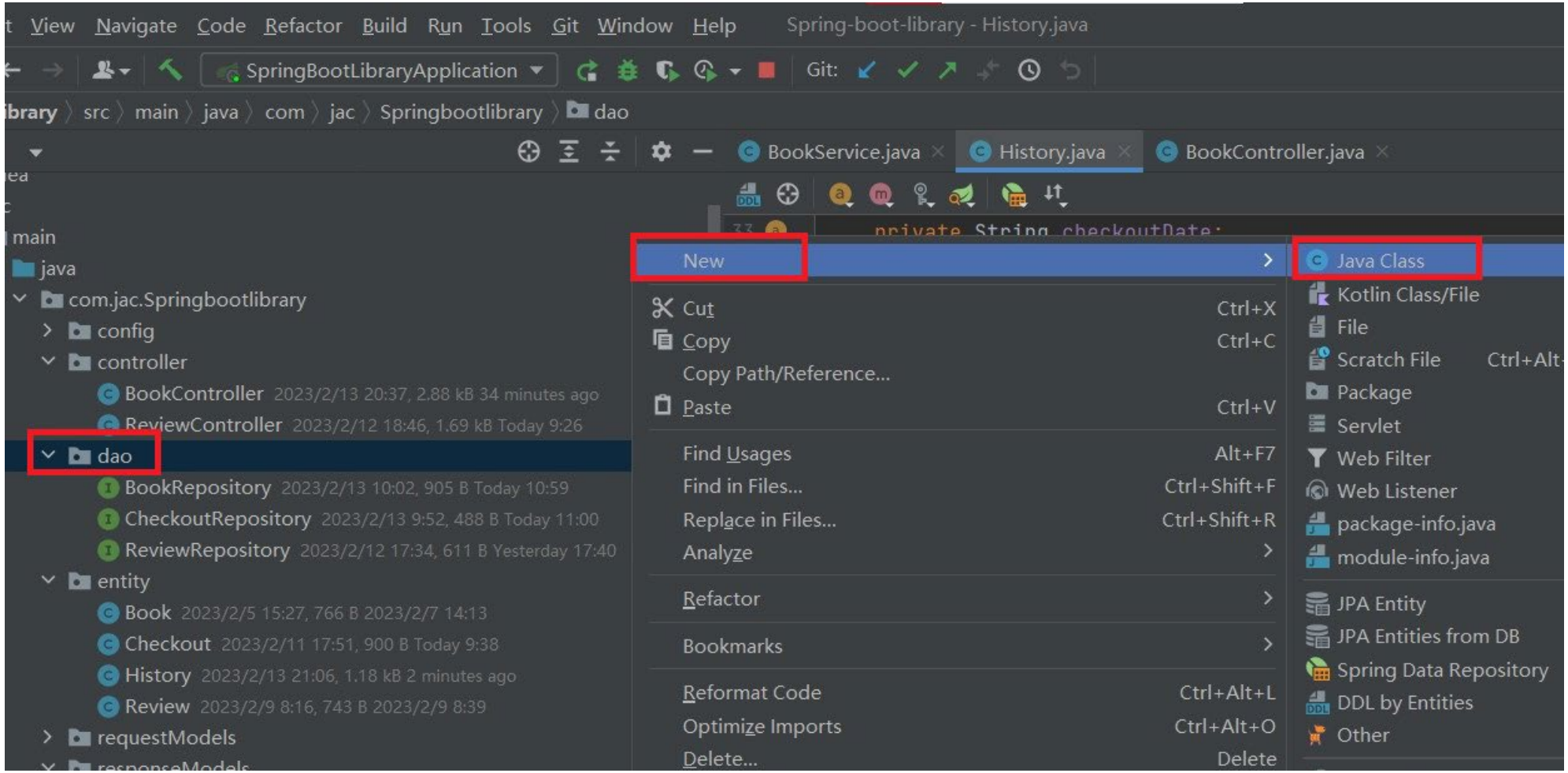




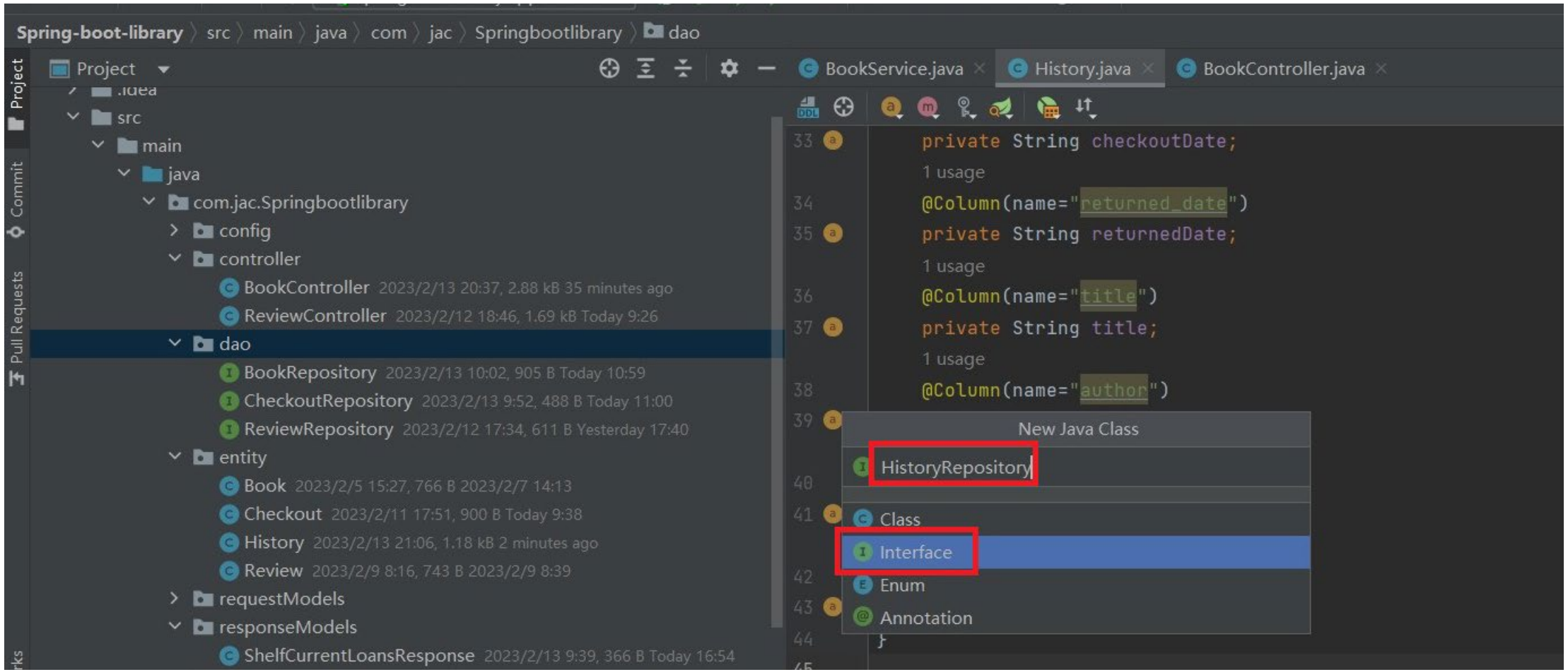
Step 4 Create properties which are mapping to the column name of table “History”.

2. SPRINGBOOT HISTORY REPOSITORY

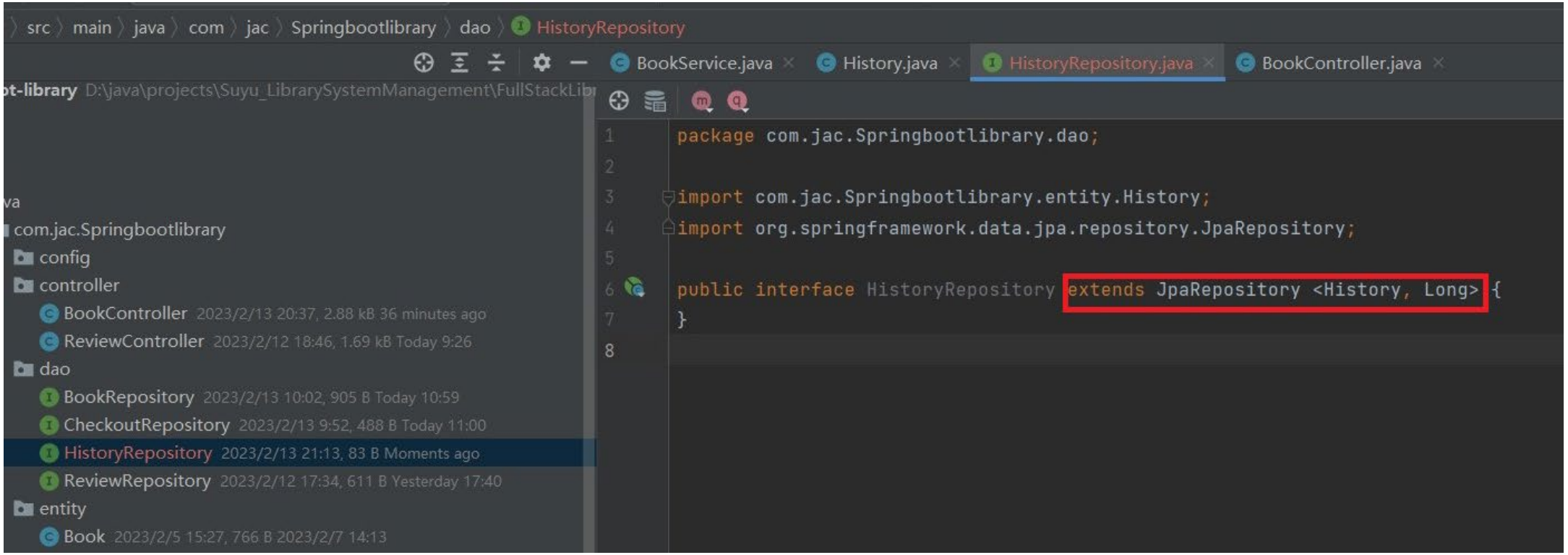
Step 1, right click on “entity” packge, choose New->Java Class.



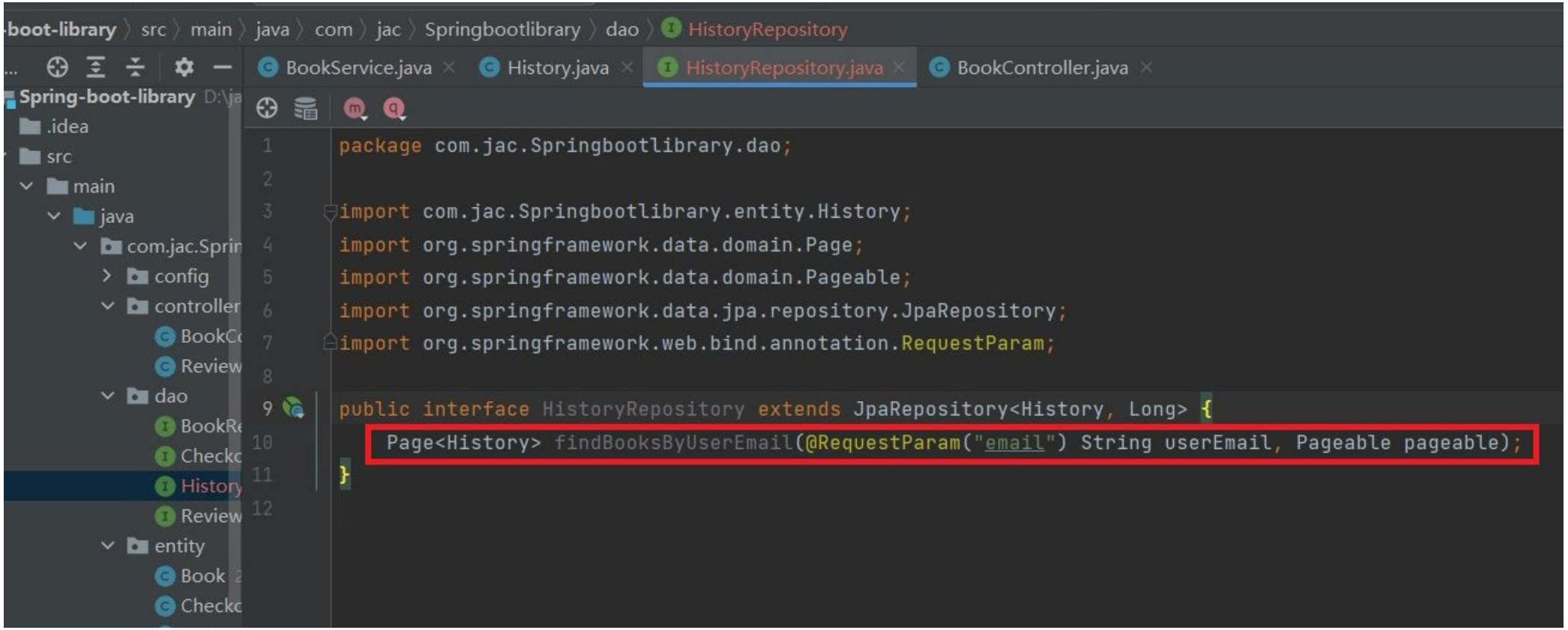
Choose “Interface”, Input the name “HistoryRepository”.



Step 2 add “extends JpaRepository <History, Long>” to inheritance Jpa repository.



Step 3 create an interface function findBooksByUserEmail().



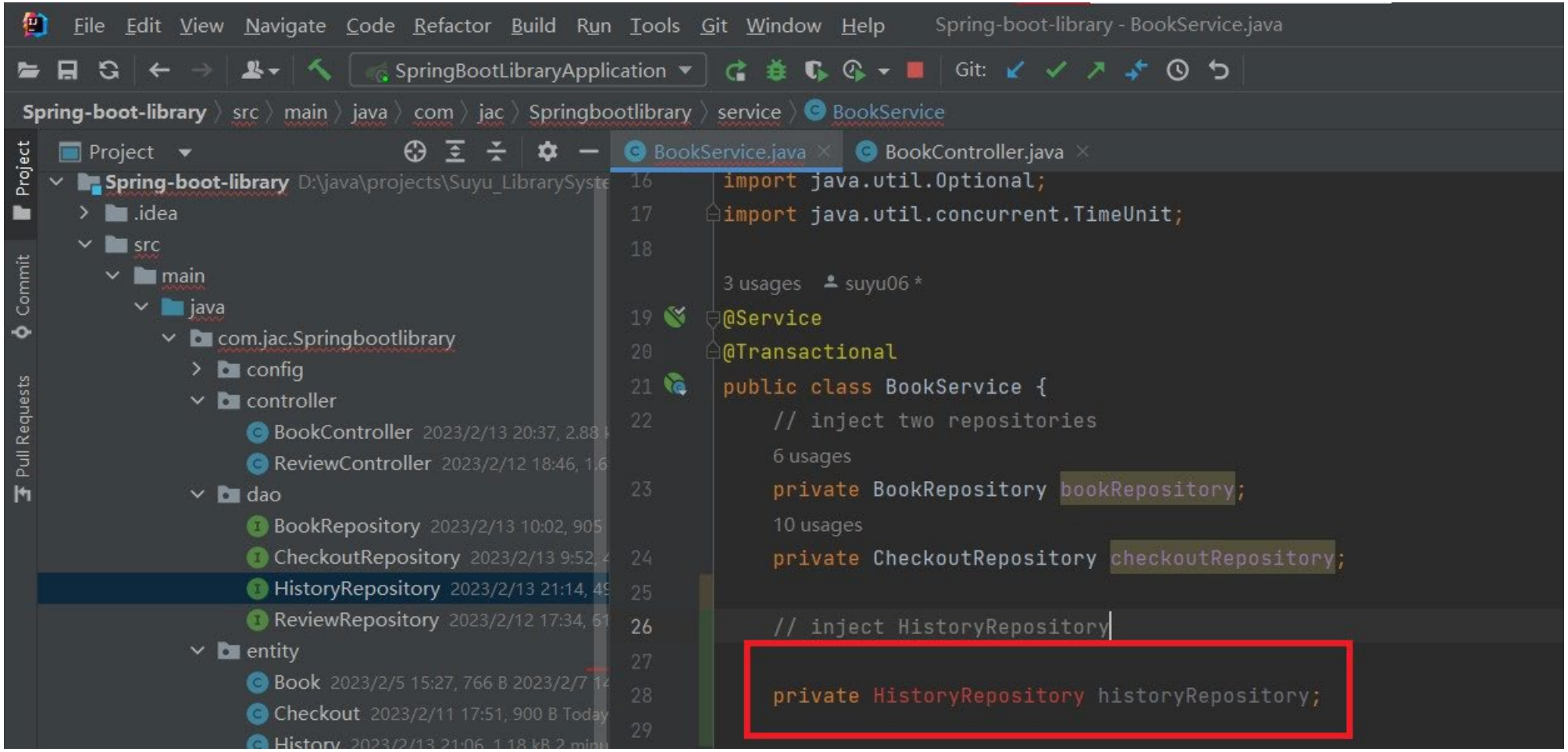
3. SPRINGBOOT ENHANCE BOOKSERVICE FOR HISTORY

We have our history entity created and we have our history repository created.

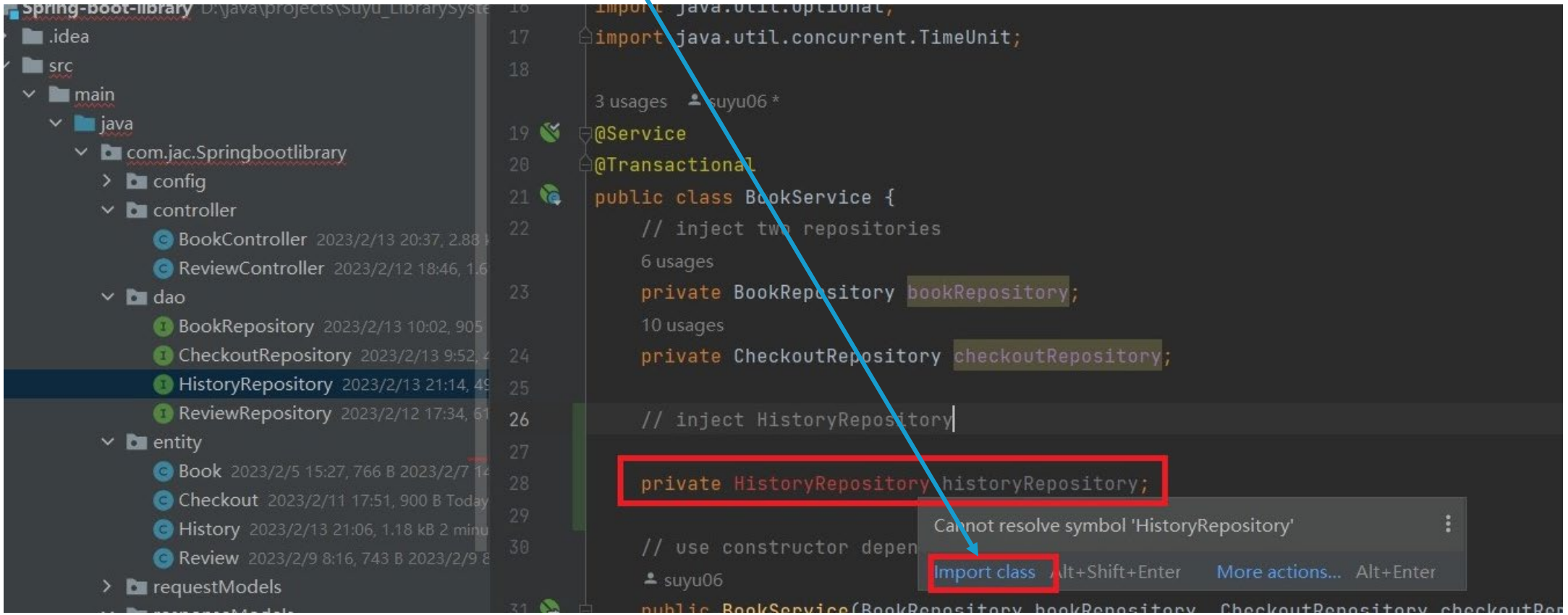
Now we want to realize this functionality:

- when a user returns a book, we want to make sure we capture all that information of the book that they are returning.
- And then save it to the history repository.

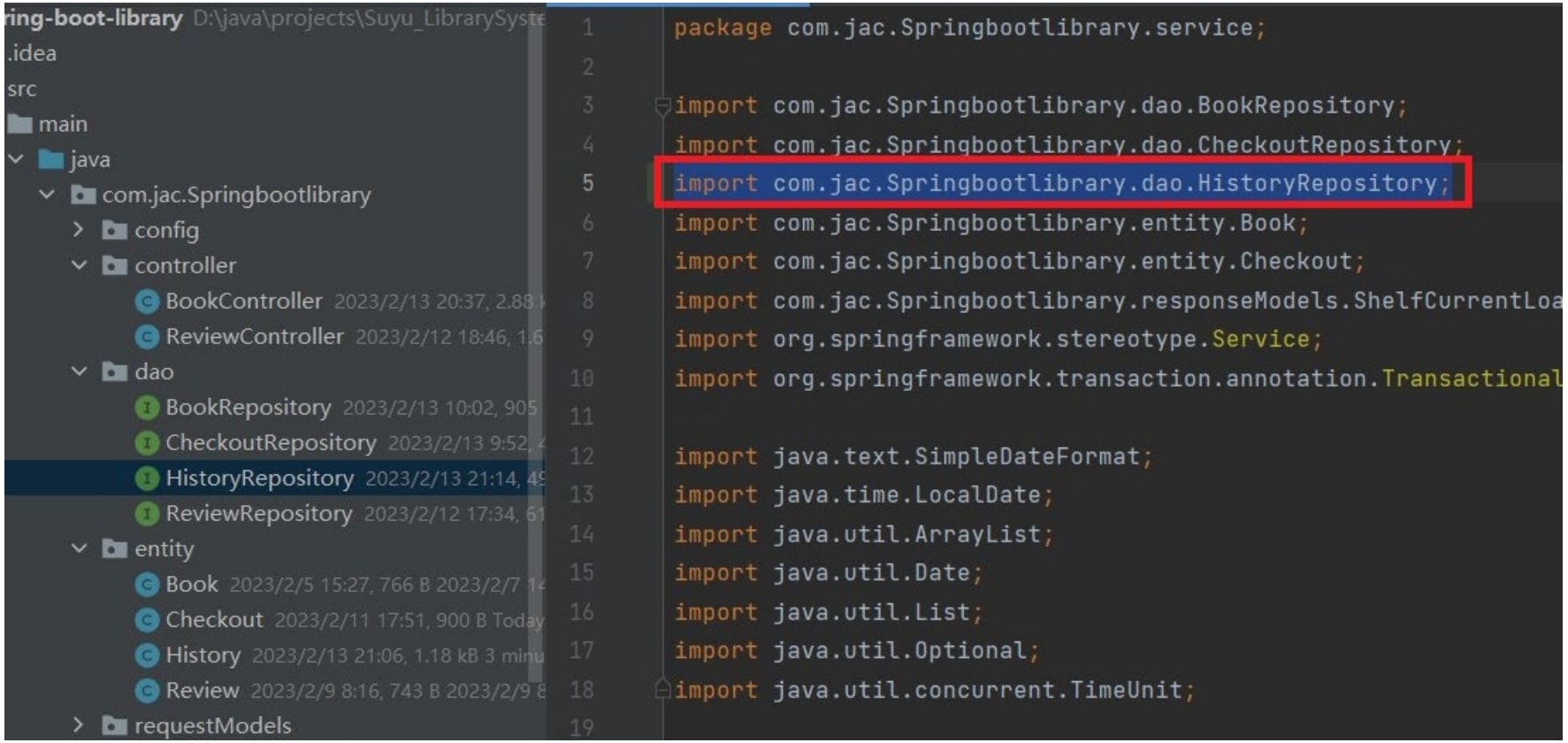
Go to “services package”, jump into our “BookService”.
Step1 inject HistoryRepository and pass it into constructor.

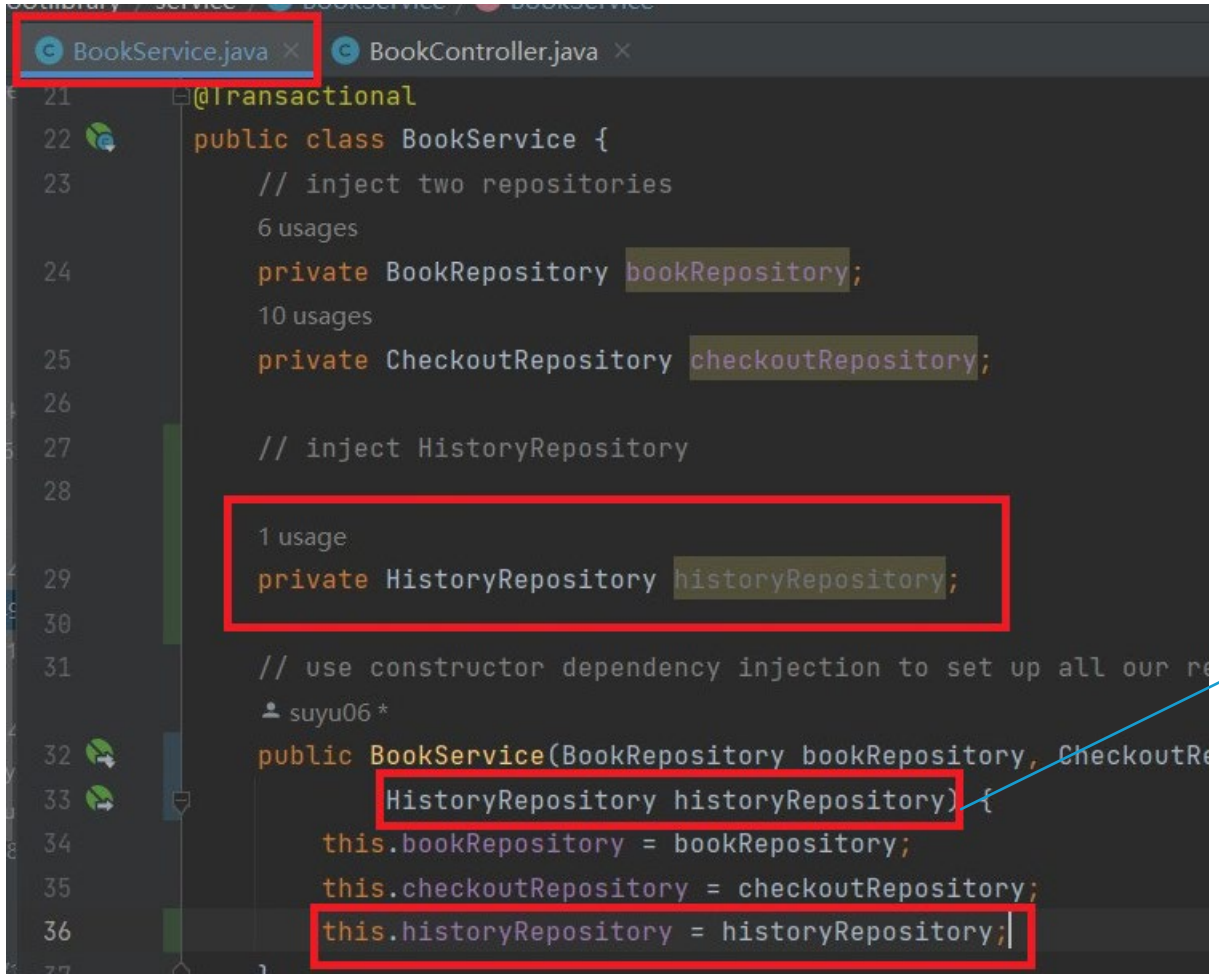


1.1 After initialize a new historyRepostiroy object. We will get a red error. Hover on the HistoryRepository, click on “import class”.



HistoryRepository imported.





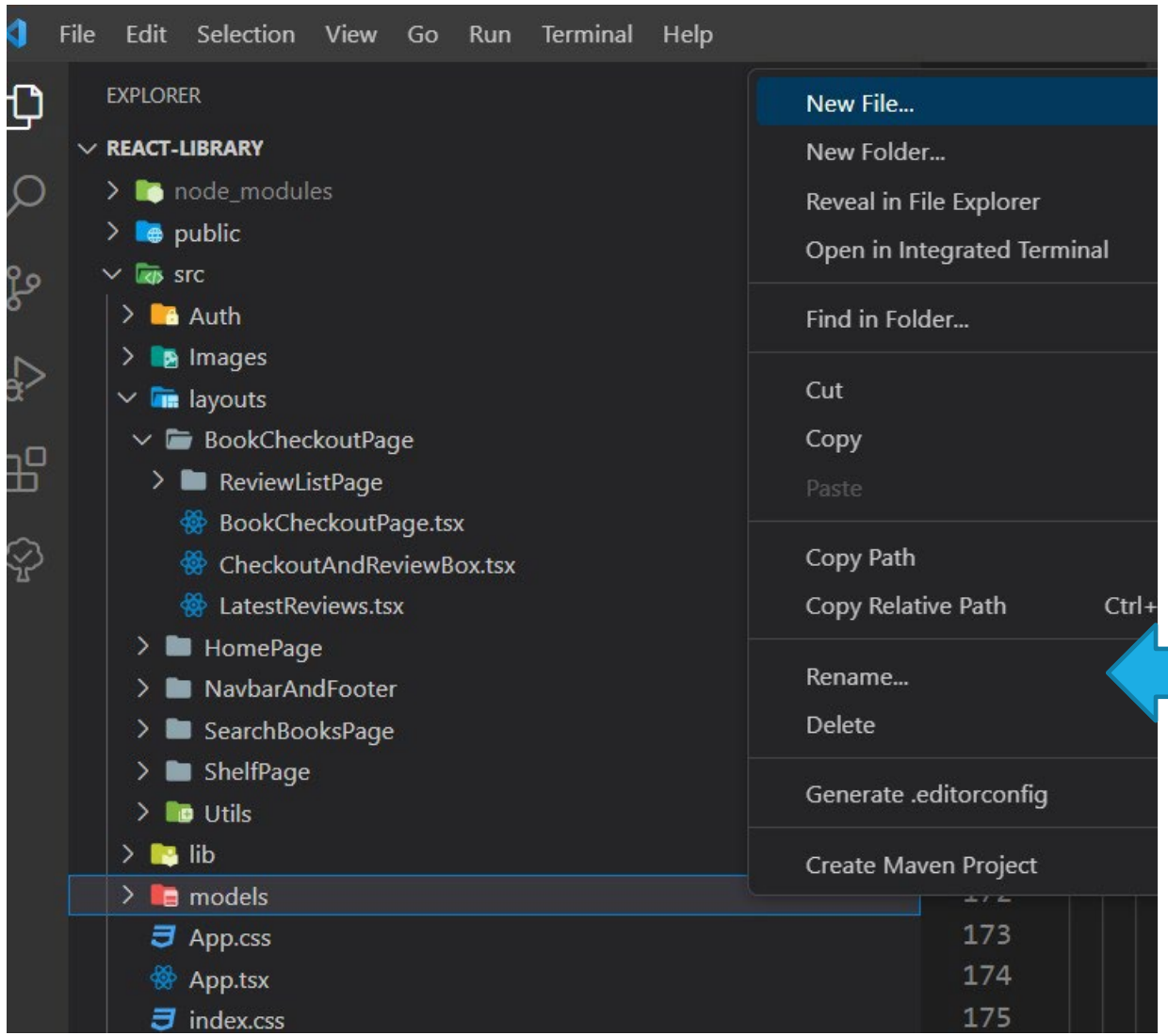
1.2 Add the historyRepository to the constructor.

BookService.java | BookController.java

```
1 usage   suyu06 *
public void returnBook(String userEmail, Long bookId) throws Exception
    // grab the book by book Id
    Optional<Book> book = bookRepository.findById(bookId);
    // grab the checkout  by user email and book id
    Checkout validateCheckout = checkoutRepository.findByUserEmailAndBo
    // if not exists:
    if (!book.isPresent() || validateCheckout == null) {
        throw new Exception("Book does not exist or not checked out by
    }
    //make the number of copies available increase by 1
    book.get().setCopiesAvailable(book.get().getCopiesAvailable() + 1);
    // save the book into our book repository
    bookRepository.save(book.get());
    //delete the book from our checkout database
    checkoutRepository.deleteById(validateCheckout.getId());
    // create a new history object with this returned book
    History history = new History(
            userEmail,
            validateCheckout.getCheckoutDate(),
            LocalDate.now().toString(),
            book.get().getTitle(),
            book.get().getAuthor(),
            book.get().getDescription(),
            book.get().getImg()
    );
    // save it into the repository
    historyRepository.save(history);
}
```

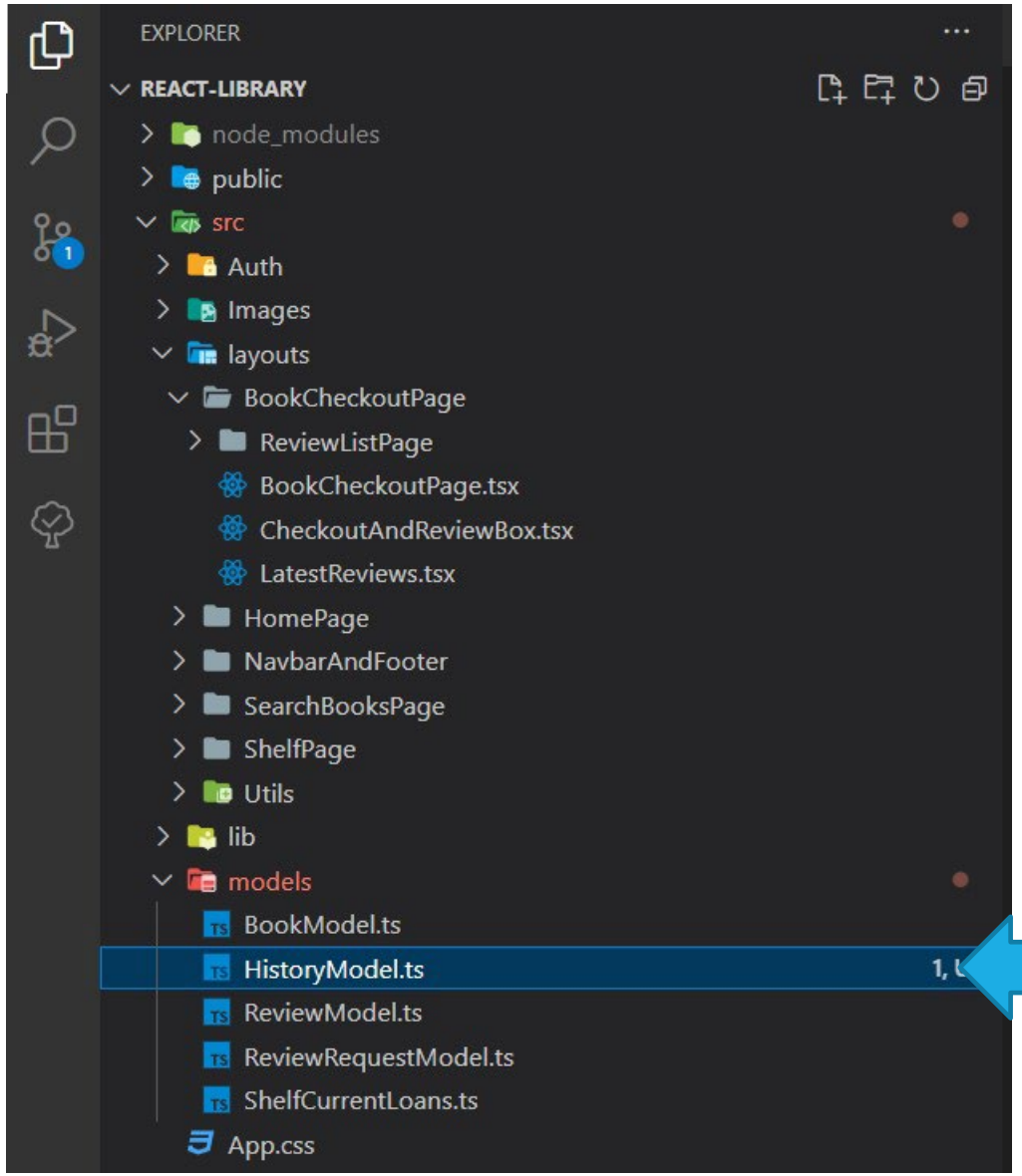
Step 2 In the function returnBook(), save this returned book's all info into our historyRepository.

5. REACT HISTORY MODEL



Now that we created our history entity and our history repository on the back end, we now need to add a history model on our front end React application.

Step1
Right click on models and choose new file,



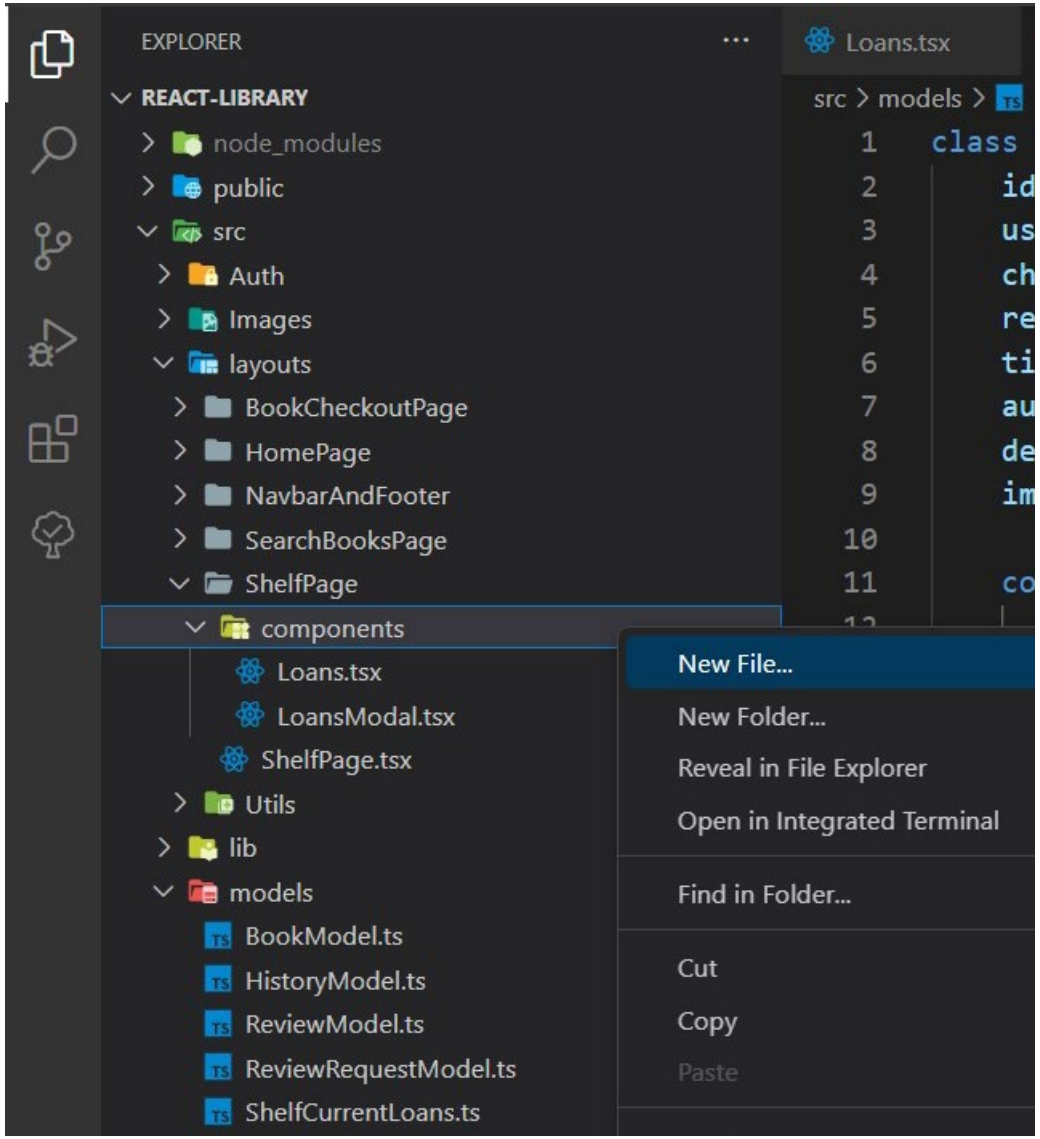
Name it HistoryModel.ts

src > models > HistoryModel.ts > HistoryModel > constructor

```
1   class HistoryModel {
2       id: number
3       userEmail: string
4       checkoutDate: string
5       returnedDate: string
6       title: string
7       author: string
8       description: string
9       img: string
10
11      constructor(id: number, userEmail: string,checkoutDate: string, returnedDate: string,
12          title: string, author: string, description: string, img: string) {
13          this.id = id
14          this.userEmail = userEmail
15          this.checkoutDate = checkoutDate
16          this.returnedDate = returnedDate
17          this.title = title
18          this.author = author
19          this.description = description
20          this.img = img
21      }
22  }
23
24  export default HistoryModel
25
```

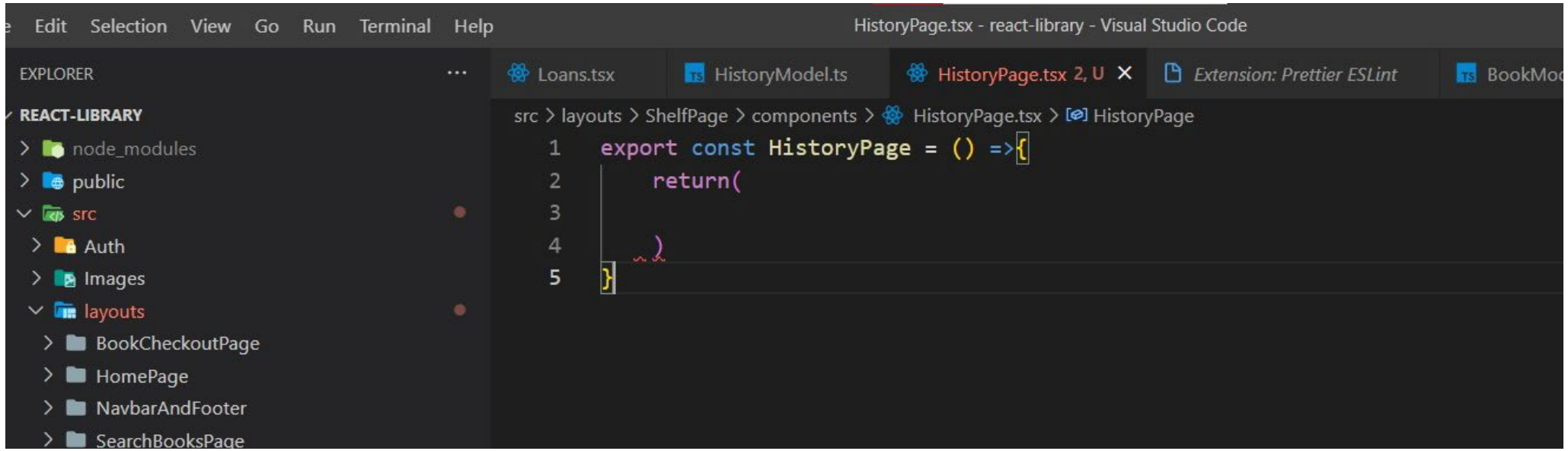
Step 2, create the class with the properties which are mapping to the backend history class.

6. REACT HISTORY PAGE



Step1
Jump into layouts folder ->ShelfPage folder, then go into components folder, right click and choose “new file”.

Step 2, as always, create export const structure.

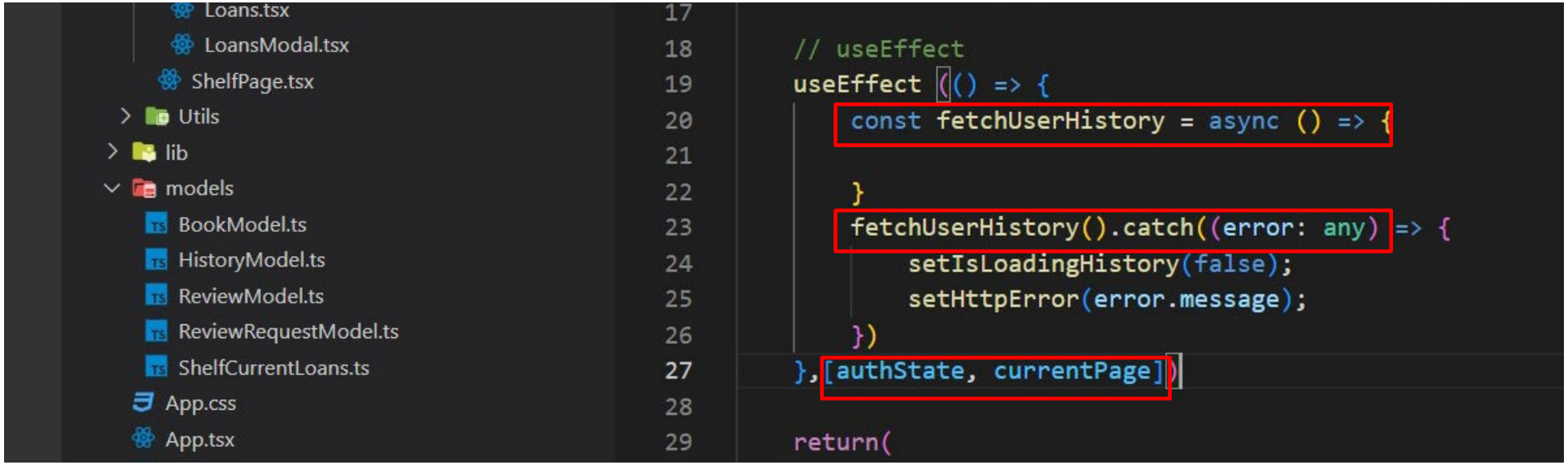


Step 3 add useState and import useOktaAuth.

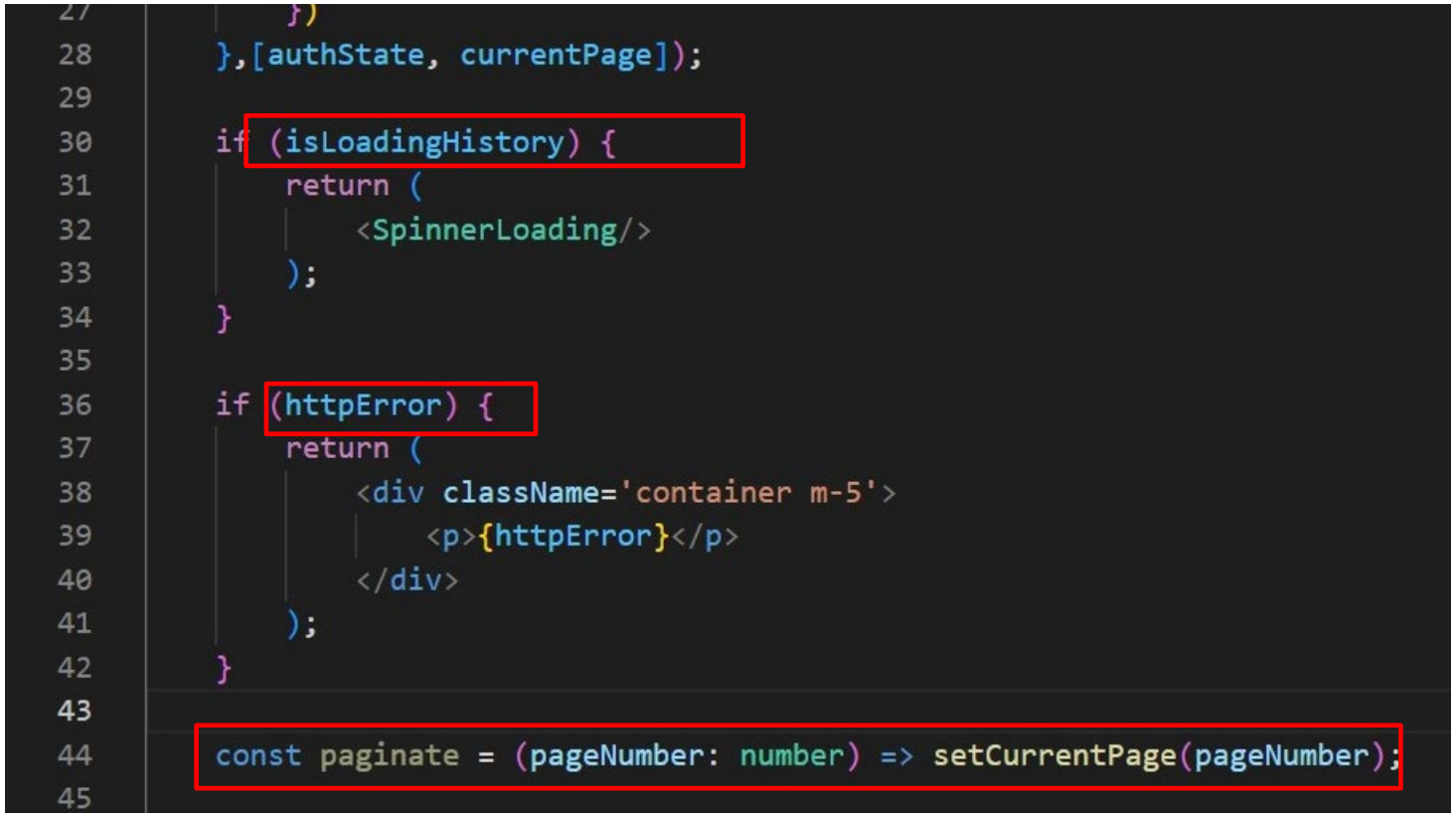
src > layouts > ShelfPage > components > HistoryPage.tsx > HistoryPage

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

export const HistoryPage = () =>{

    //useState
    const { authState } = useOktaAuth();
    const [isLoadingHistory, setIsLoadingHistory] = useState(true);
    const [httpError, setHttpError] = useState(null);
    //Histories
    const [histories, setHistories] = useState<HistoryModel[]>([]);

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

Step 4 create useEffect structure, which contains fetchUserHistory async function and error catch function.
Meanwhile, add “authState” and “currentPage” as change factor into the array at the end of the useEffect.



Step 5 add “isloading” and “httpError” and “setCurrentPage” part of code.



7. REACT HISTORY USEEFFECT

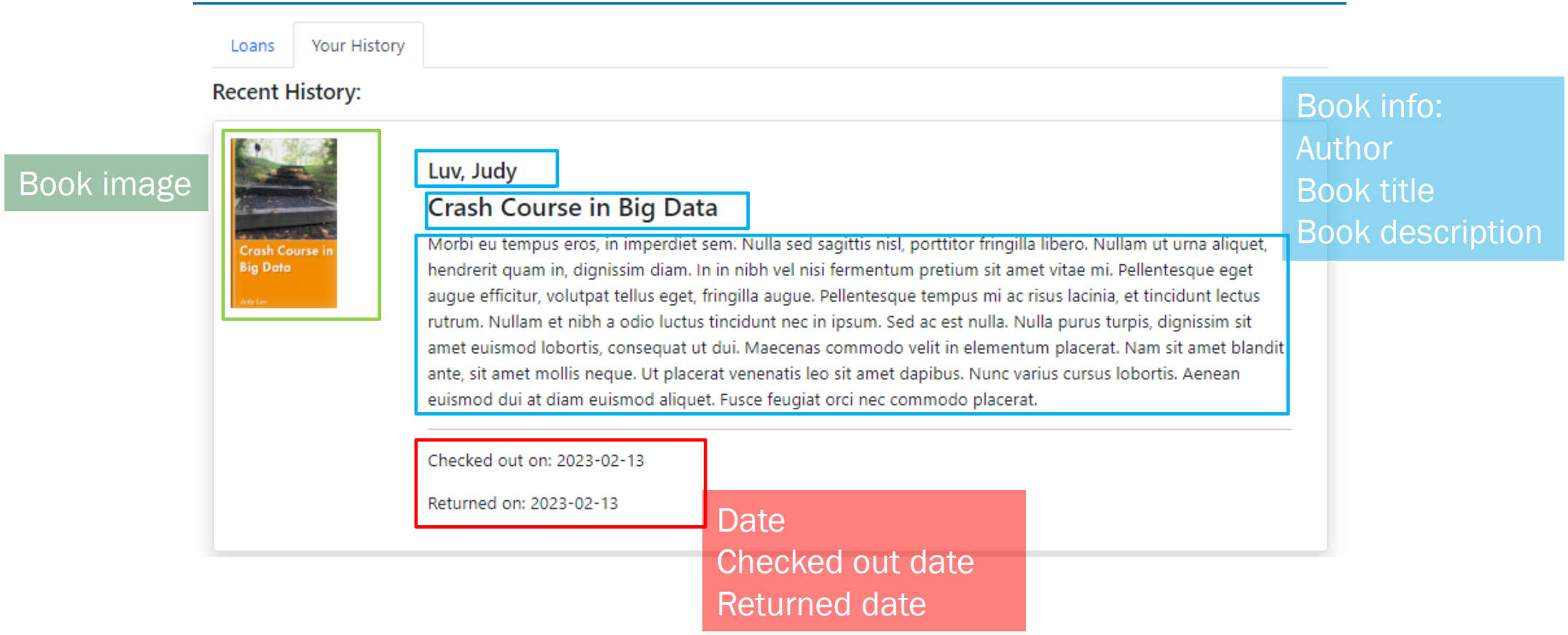
```
// useEffect
useEffect (() => {
    const fetchUserHistory = async () => {
        // only user is authenticated
        if (authState && authState.isAuthenticated) {
            const url =
            `http://localhost:8080/api/histories/search/findBooksByUserEmail/?userEmail
            =${authState.accessToken?.claims.sub}&page=${currentPage - 1}&size=5`;
            const requestOptions = {
                method: 'GET',
                headers: {
                    'Content-Type': 'application/json'
                }
            };
            // fetch data with url and request (get method and json content)
            const historyResponse = await fetch(url, requestOptions);
            // if failure
            if (!historyResponse.ok) {
                throw new Error('Something went wrong!');
            }
            // convert to json object
            const historyResponseJson = await historyResponse.json();
            // get all histories
            setHistories(historyResponseJson._embedded.histories);
            // get the total page number
            setTotalPages(historyResponseJson.page.totalPages);
        }
```

Fill in the async function.

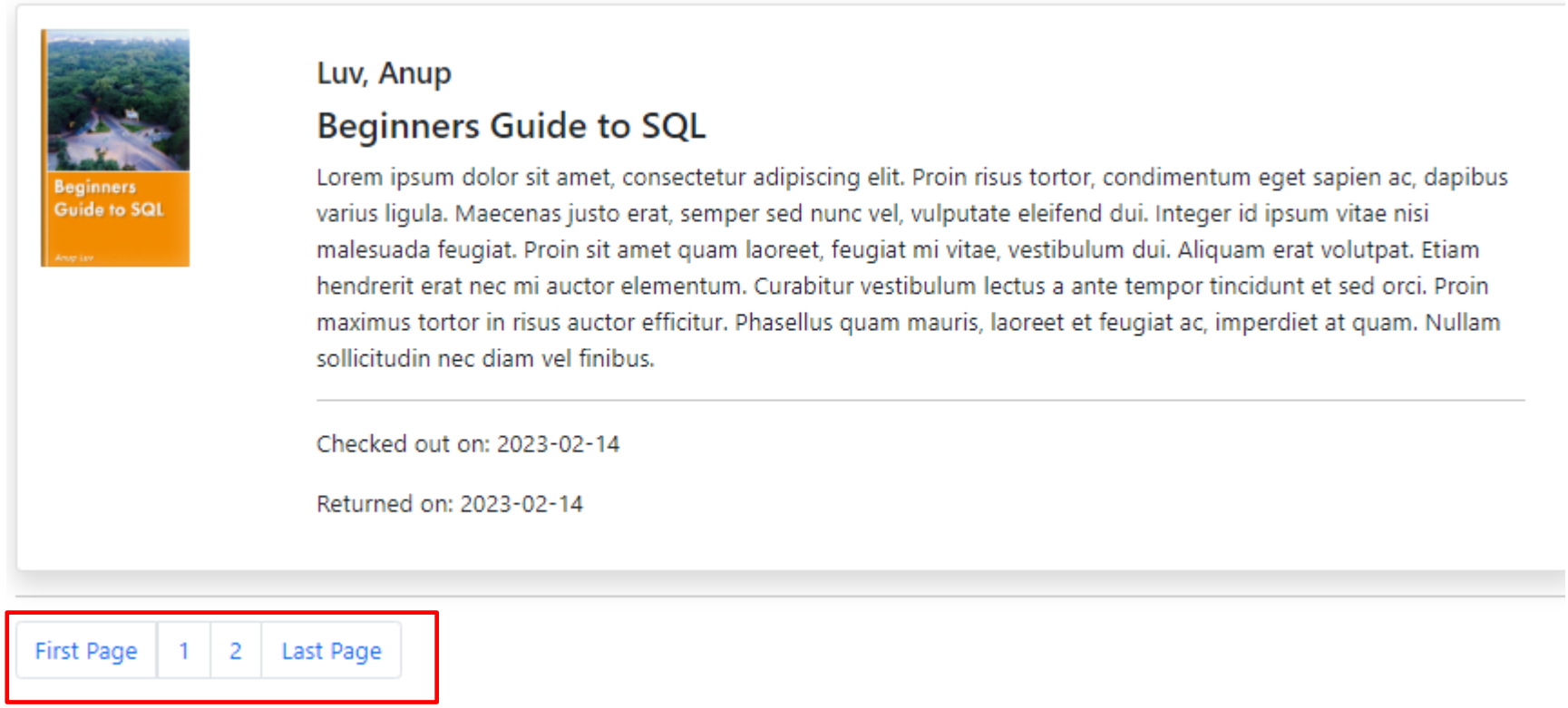
1. Call the api with the url , request method is “GET”, header’s content tyoe is JSON.
2. use “await fetch” function to fetch the data.
3. Convert the result to JSON object.
4. get the history data from “_embedded” objects and the number of total page.

8. REACT HISTORY HTML/CSS

Want to have the page as below:



In the bottom of the page, have a pagination button.



1. Book image part.

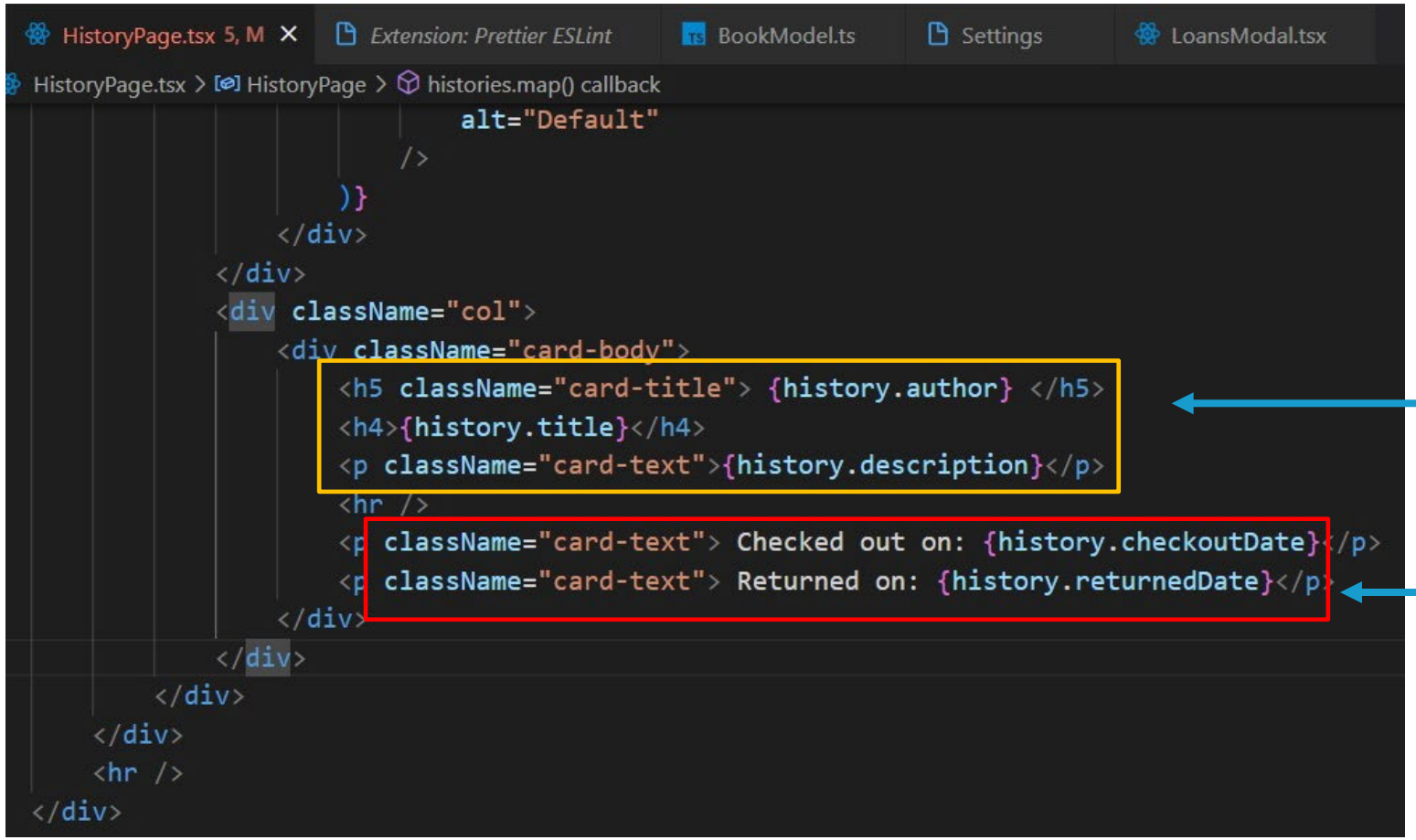
1.1 code for desktop version: If there is an image, present it; if no, present our default image.

```
return (
    <div className="mt-2">
        {histories.length > 0 ? (
            <>
                {/*if lenght is >0, which means there is book in history repository  */}
                <h5>Recent History:</h5>
                {histories.map((history) => (
                    <div key={history.id}>
                        <div className="card mt-3 shadow p-3 mb-3 bg-body rounded">
                            <div className="row g-0">
                                <div className="col-md-2">
                                    {/* image part */}
                                    <div className="d-none d-lg-block">
                                        {history.img ? (
                                            // if ther is an image
                                            <img src={history.img} width="123" height="196" alt="Book" />
                                        ) : (
                                            //if not show our default image
                                            <img
                                                src={require('./../../../Images/BooksImages/book-luv2code-1000.png')}
                                                width="123"
                                                height="196"
                                                alt="Default"
                                            />
                                        )}
                                    </div>
```

HistoryPage > histories.map() callback

```
        {history.img ? (
            // if ther is an image
            <img src={history.img} width="123" height="196" alt="Book" />
        ) : (
            //if not show our default image
            <img
                src={require('./../../../Images/BooksImages/book-luv2code-1000.png')}
                width="123"
                height="196"
                alt="Default"
            />
        )}
    </div>
    <div className="d-lg-none d-flex justify-content-center align-items-center">
        {history.img ? (
            // if ther is an image
            <img src={history.img} width="123" height="196" alt="Book" />
        ) : (
            //if not show our default image
            <img
                src={require('./../../../Images/BooksImages/book-luv2code-1000.png')}
                width="123"
                height="196"
                alt="Default"
            />
        )}
    </div>
```

1.2 Then, just copy the image code one time for the mobile version. But change the className from “d-none d-lg-block” to “d-lg-none d-flex justify-content-center align-items-center”.

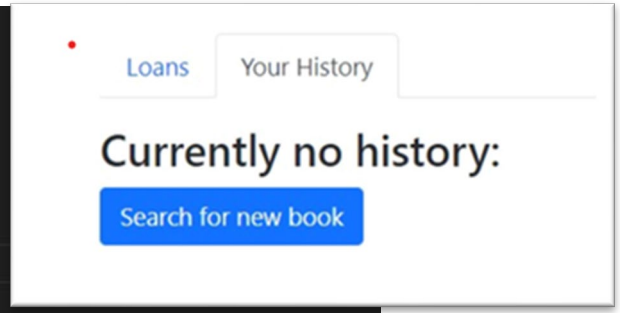


2. Book info part
Including author, book title , book description.

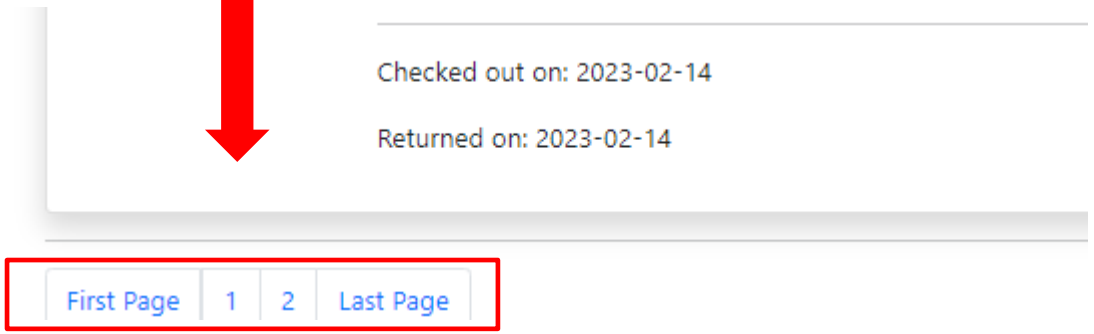
3. Date
Including checked out date, returned date.,

4. If user has no history to present:

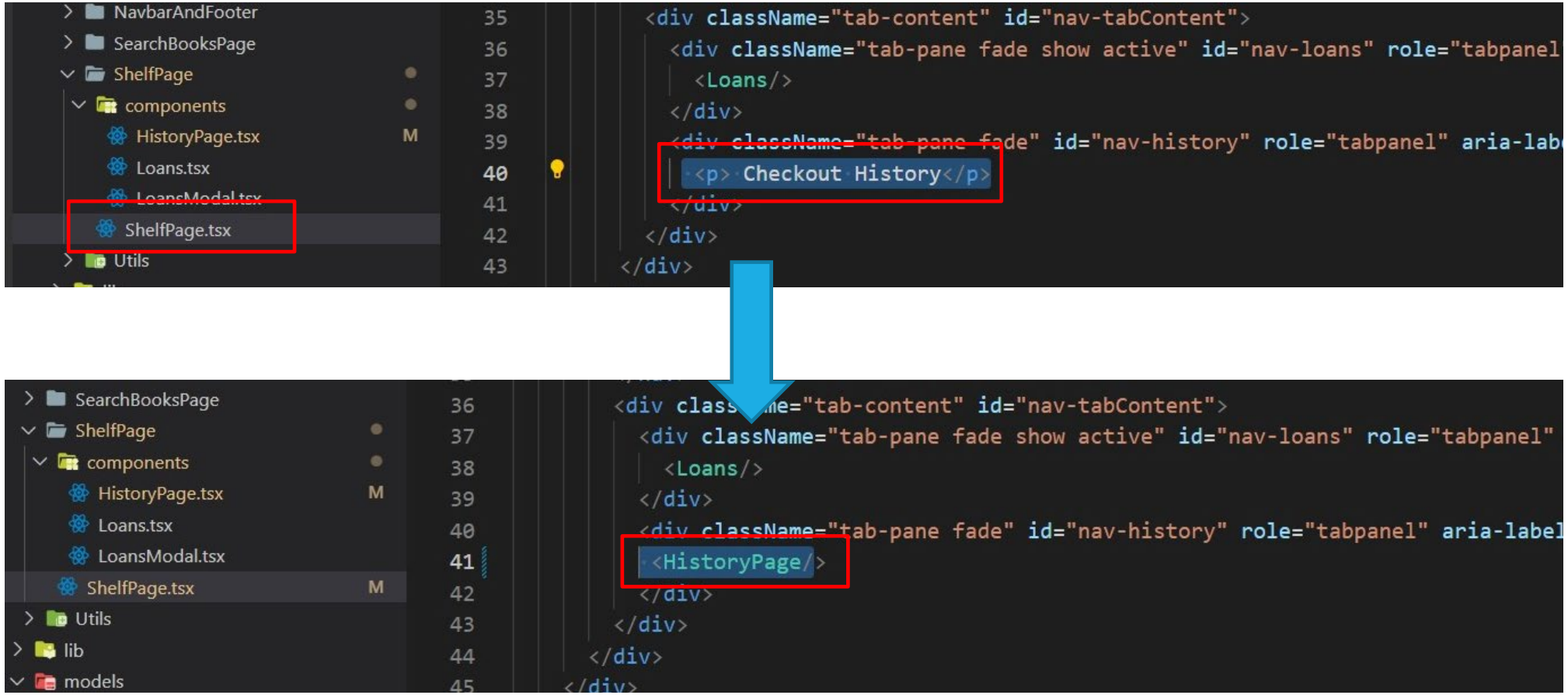
```
        </>
    ) : (
        //if no book in history repository
        <>
            <h3 className="mt-3">Currently no history: </h3>
            <Link className="btn btn-primary" to={'search'}>
                Search for new book
            </Link>
        </>
    )}
    {totalPages > 1 && <Pagination currentPage={currentPage} totalPages={totalPages} paginate={paginate} />}
</div>
```



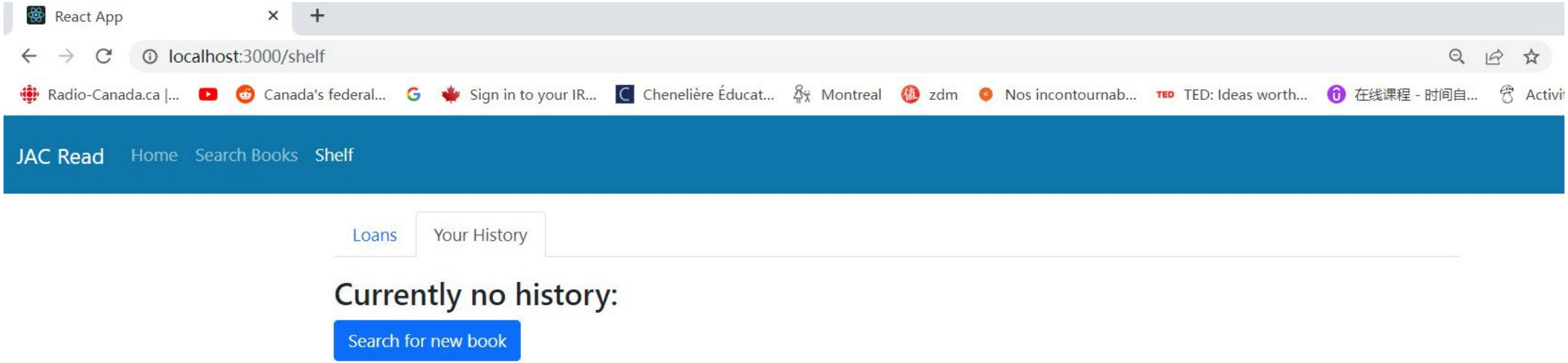
If there are more than one pages:



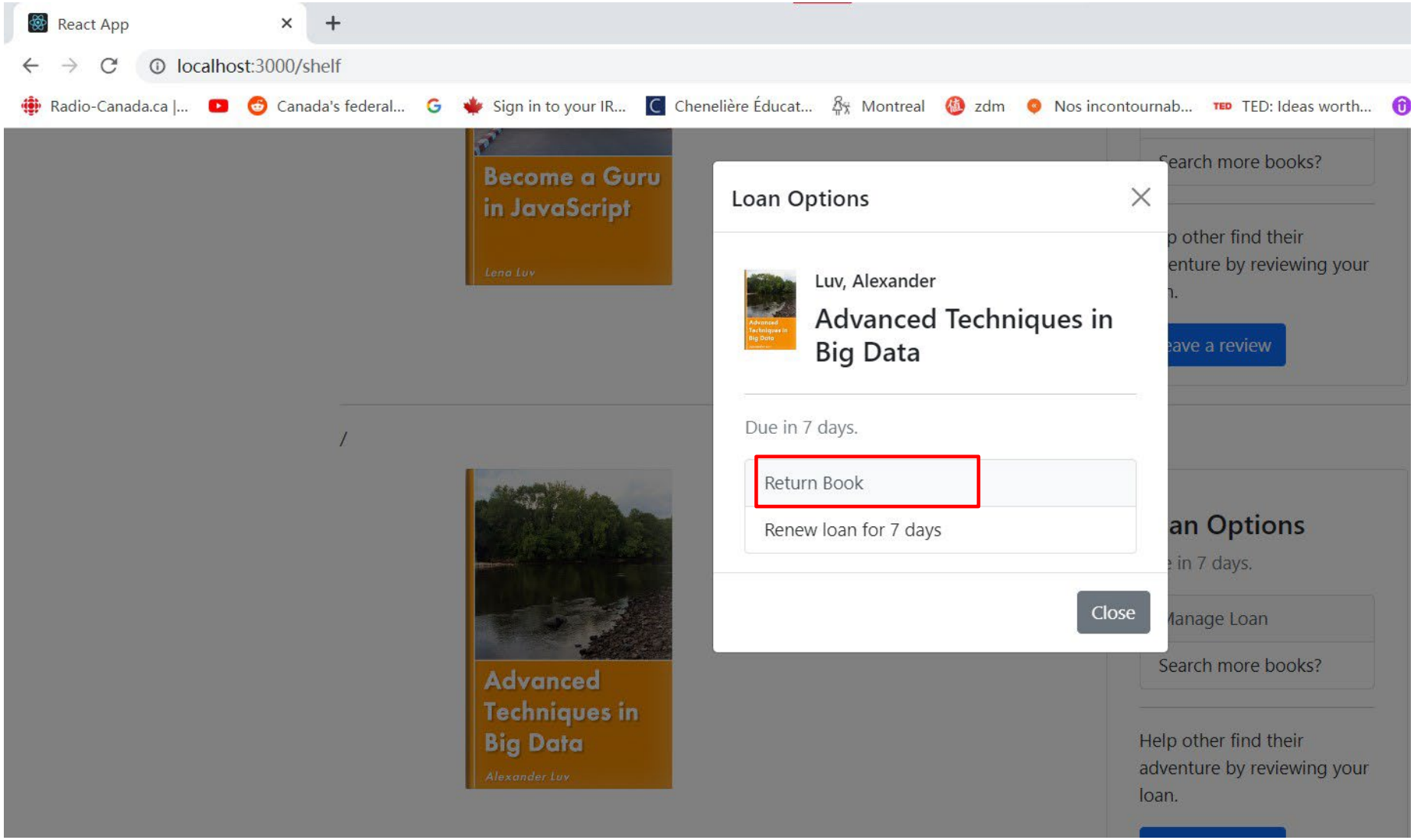
5. Go to ShelfPage.tsx , use our new created HistoryPage component to replace the static p tag.
And import the HistoryPage component.



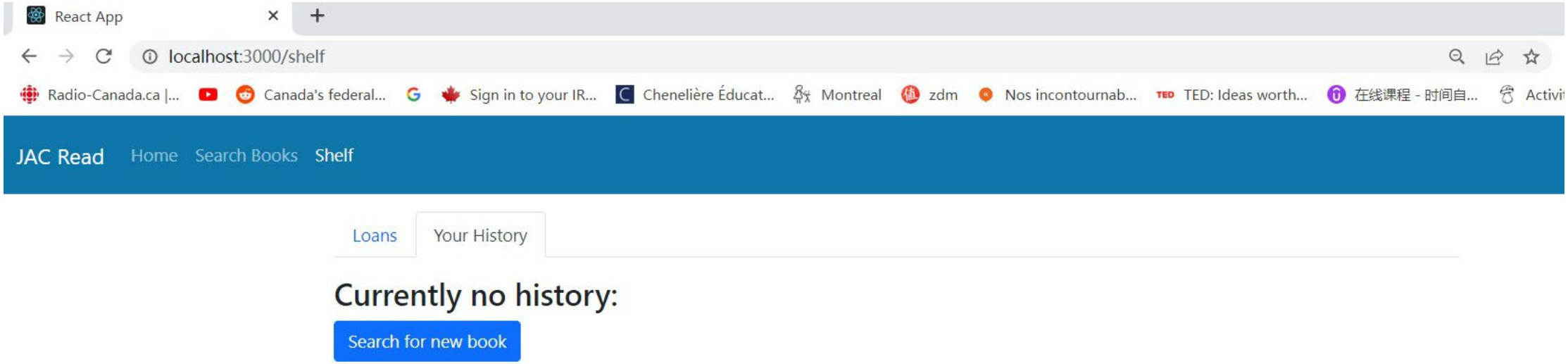
Now, with the logged in status, we click on the shelf link in the navigation bar, we can get this page:



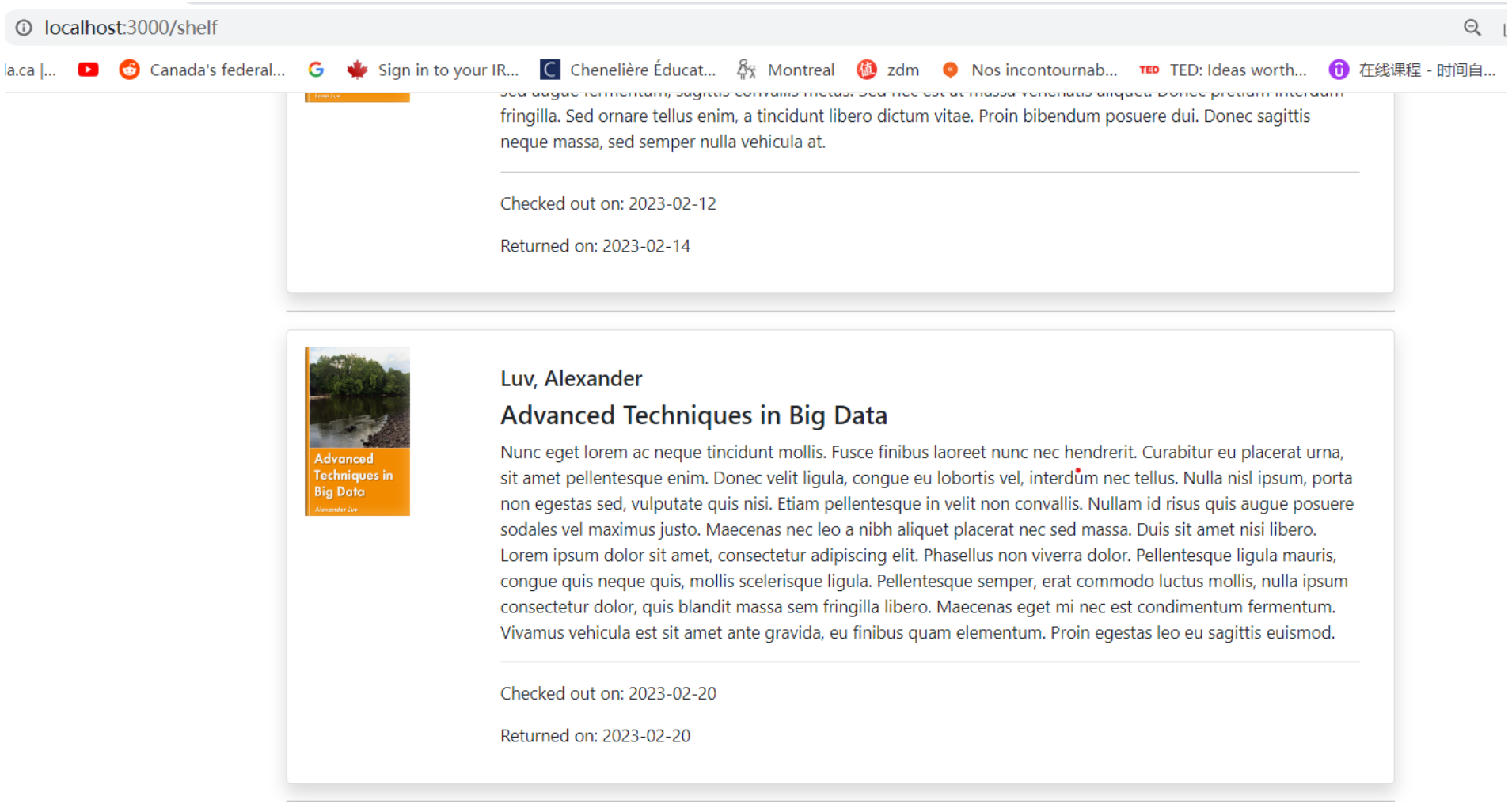
If I try to borrow a book and return it, it should be existed in my history page. Clicked on the return book button to return the book.



But it still shows nothing.



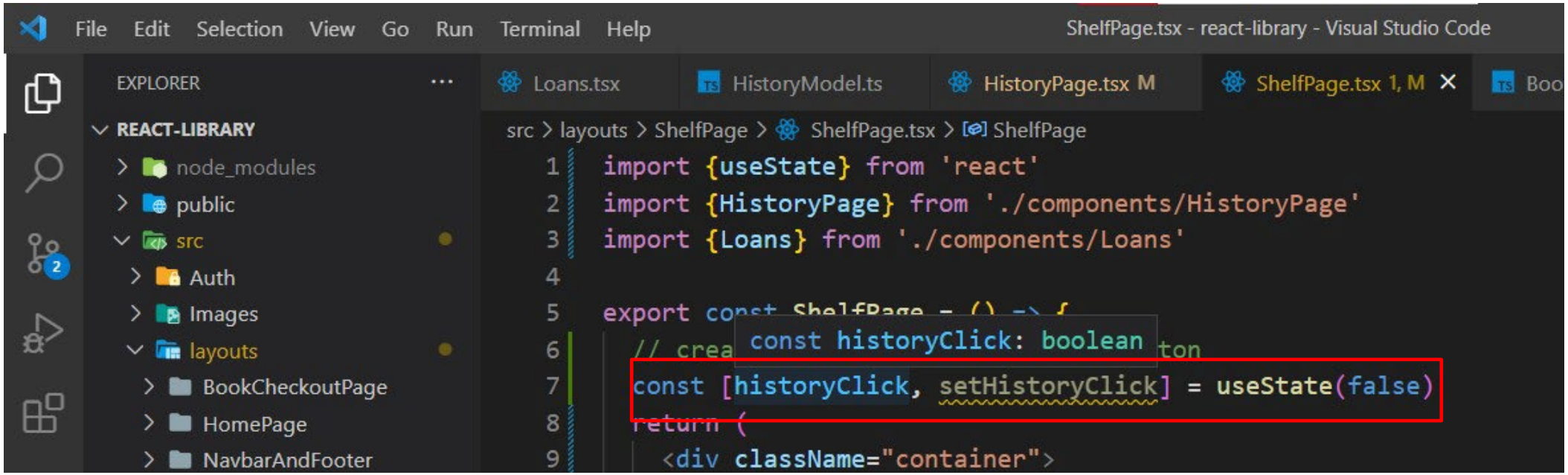
When we refresh the page, it begin to present the book we just returned. It does not populate automatically.
And that‘s because our history is loading with loans. We don’t have them connected by state.



REACT SHELF STATE

What's we wan to do is to force a new useEffect to upload each time the history page opens up.

Step 1 In ShelfPage.tsx, create a useState for historyClick.



Step 2 Scroll down until we see our historyPage component.

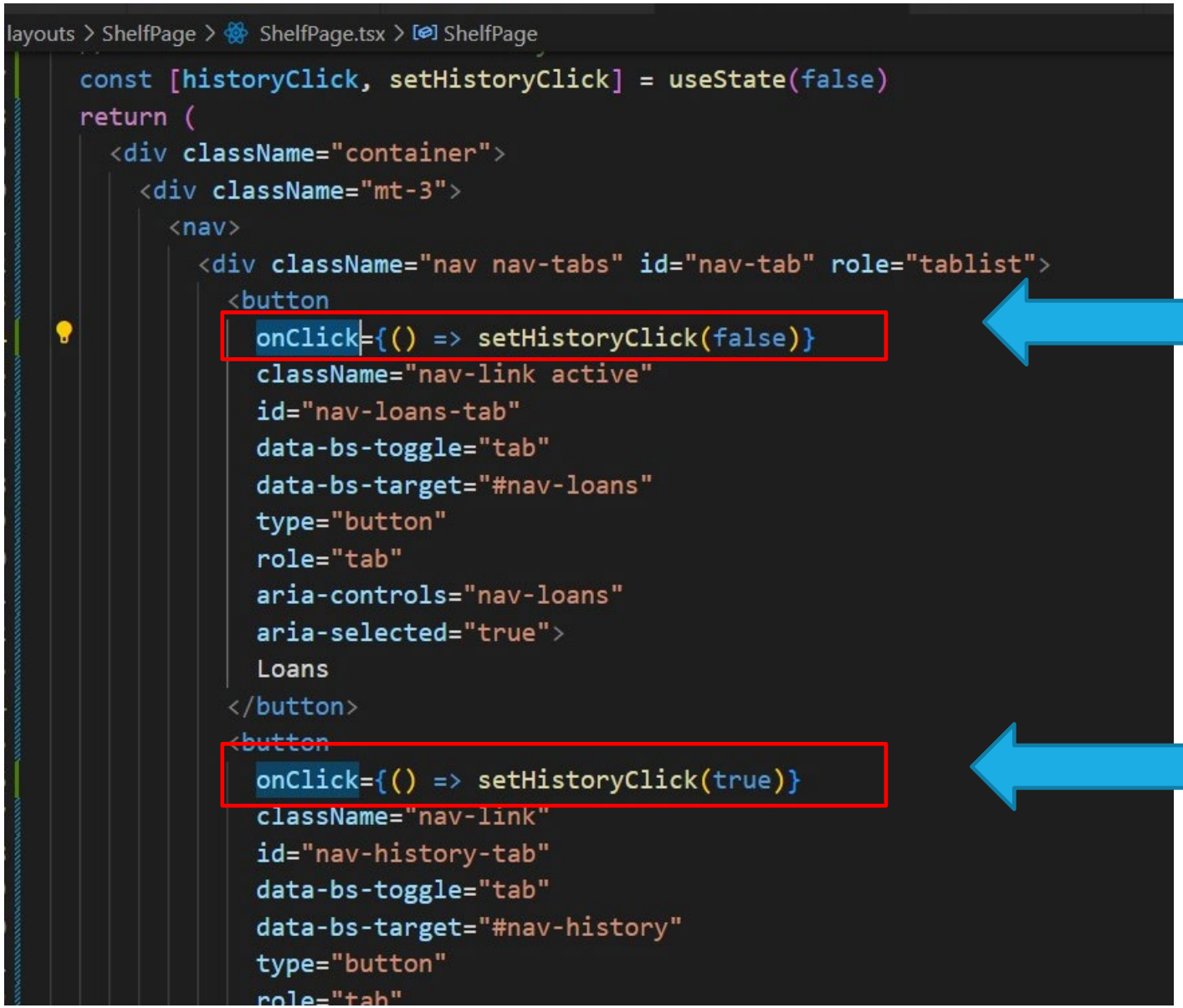
Add a judgement to decide whether the history button is clicked. when historyClick is true, we're going to render our history page.
But when it's not true, we're essentially just removing our history page.

```
    <div className="tab-content" id="nav-tabContent">
      <div className="tab-pane fade show active" id="nav-loans" role="tabpanel" aria-labelledby=
        <Loans />
      </div>
      <div className="tab-pane fade" id="nav-history" role="tabpanel" aria-labelledby="nav-histo
        {/* if history click is true, renderhistory page.if not, remove history page. */}
        {historyClick ? <HistoryPage /> : <></>}
      </div>
    </div>
  </div>
</div>
```

src > layouts > ShelfPage > ShelfPage.tsx > ShelfPage

```
7   const [historyClick, setHistoryClick] = useState(false)
8   return (
9     <div className="container">
10      <div className="mt-3">
11        <nav>
12          <div className="nav nav-tabs" id="nav-tab" role="tablist">
13            <button
14              className="nav-link active"
15              id="nav-loans-tab"
16              data-bs-toggle="tab"
17              data-bs-target="#nav-loans"
18              type="button"
19              role="tab"
20              aria-controls="nav-loans"
21              aria-selected="true">
22              Loans
23            </button>
24            <button
25              className="nav-link"
26              id="nav-history-tab"
27              data-bs-toggle="tab"
28              data-bs-target="#nav-history"
29              type="button"
30              role="tab"
31              aria-controls="nav-history"
32              aria-selected="false">
33              Your History
34            </button>
```

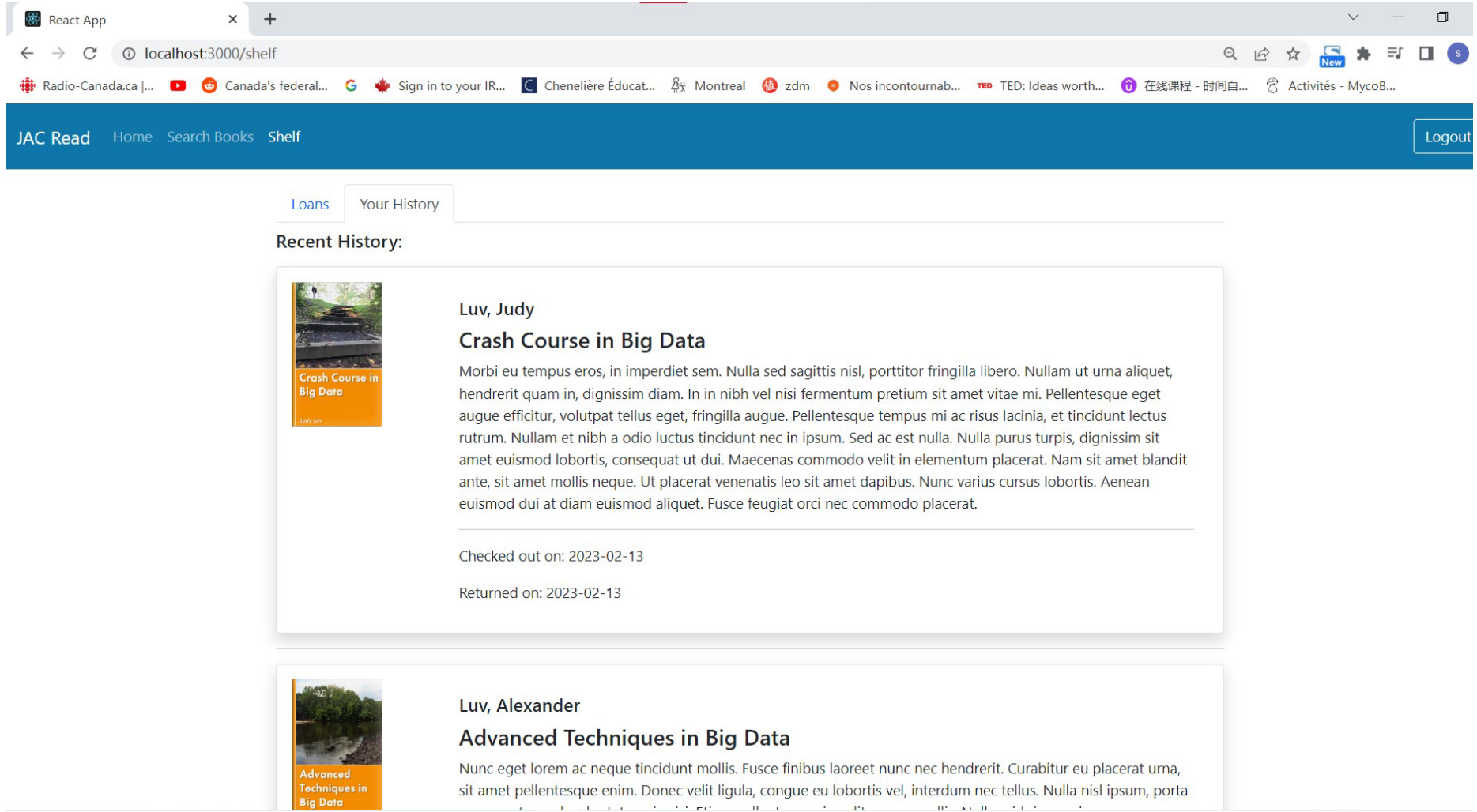
Step 3
Find the the “Loans” and “History” button.
Add onClick listener to two buttons.



If we clicked on the button “Loans”, then we set the state of historyClick as false, at this time, history page will show nothing.

If we clicked on the button “History”, then we set the state of historyClick as true, at this time, the app will render this history .

If we borrowed and returned these two books, when we click on the “Your History” button, these two books will be showed.



Check in our database.
Open history table, we will find the two books which are the same as our history page presented.