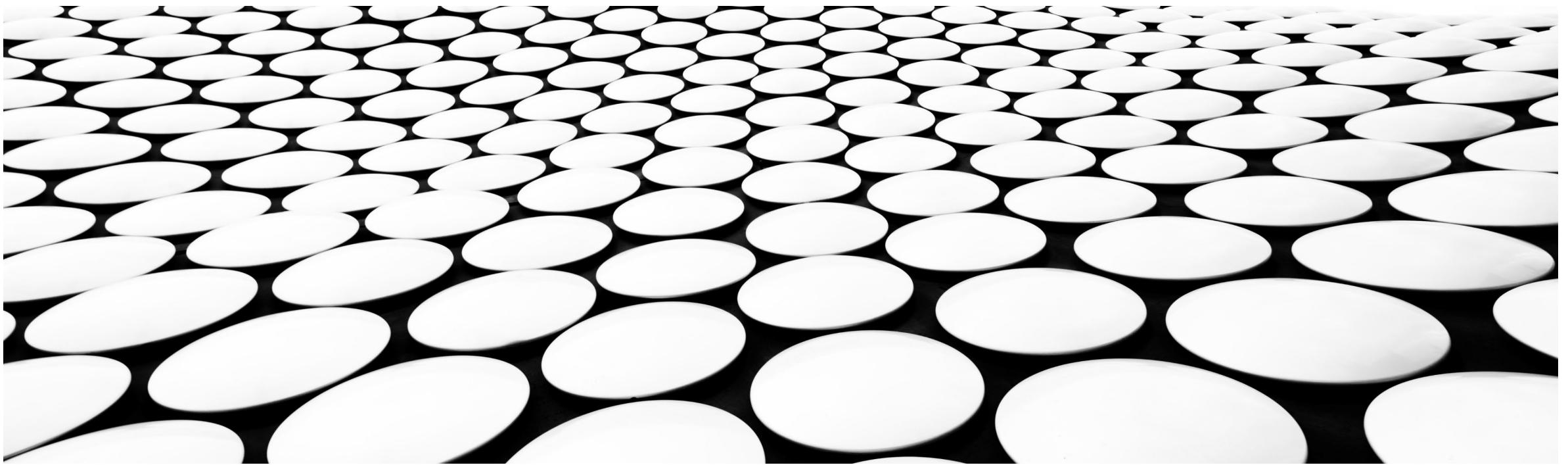
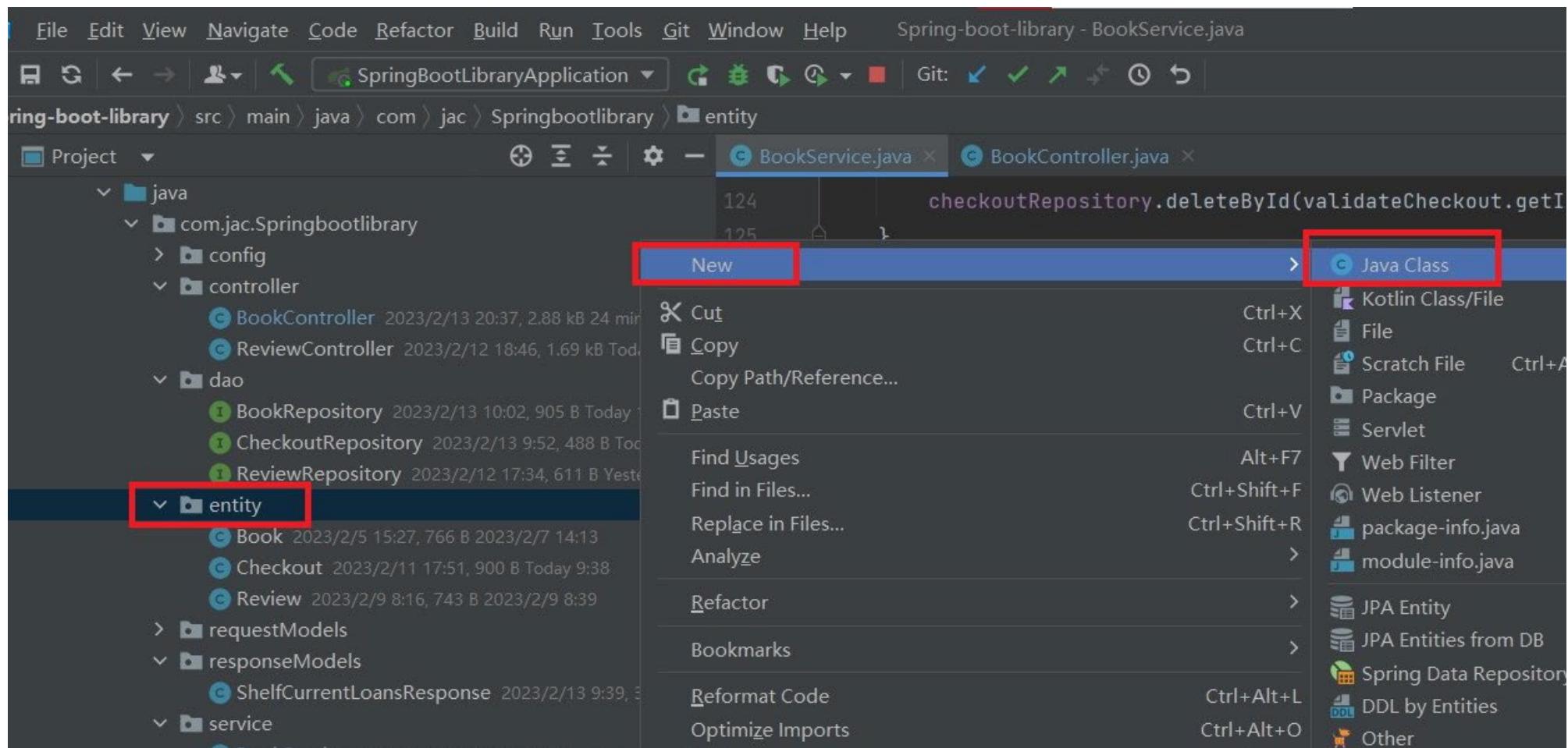

S17 HISTORY PAGE



1. SPRINGBOOT HISTORY ENTITY

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



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

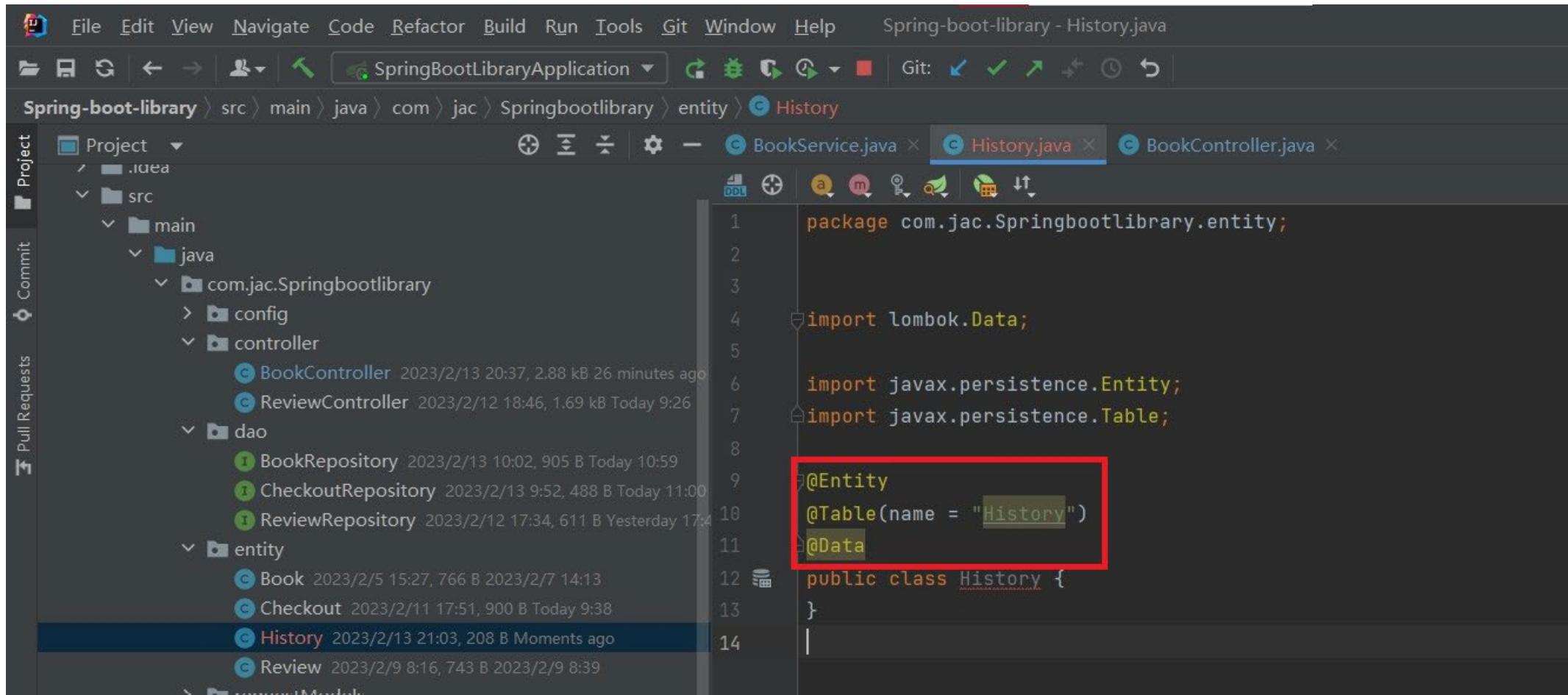
The screenshot shows a Java code editor with a dark theme. On the left, there is a file tree with the following structure:

- config
- controller
 - BookController
 - ReviewController
- dao
 - BookRepository
 - CheckoutRepository
 - ReviewRepository
- entity
 - Book
 - Checkout
 - Review
- requestModels
- responseModels
 - ShelfCurrentLoansResponse
- service
 - BookService
 - ReviewService
- utils
 - ExtractJWT
 - SpringBootLibraryApplication
- resources

The code editor has a scroll bar on the right. Lines 126 through 143 are visible, showing a method for renewing a loan. A context menu is open at the bottom of the screen, titled "New Java Class". The menu items are: History (highlighted with a red box), Class (highlighted with a blue selection bar), Interface, Enum, and Annotation. The "Class" item is currently selected.

```
// renew book
1 usage  suyu06 *
public void renewLoan(String userEmail, Long bookId) throws Ex
    //find the book by user email and id and save it into vali
    Checkout validateCheckout = checkoutRepository.findByUserEma
    // if not found
    if (validateCheckout == null) {
        // exist or not checked
        Date d1 = sdfFormat.parse(validateCheckout.getDueDate())
        Date d2 = sdfFormat.parse(LocalDate.now().toString());
        // if not exceed the due date, renew a new 7 days
        if (d1.compareTo(d2) > 0 || d1.compareTo(d2) == 0) {
            validateCheckout.setReturnDate(LocalDate.now().plusDays(7));
        }
    }
}
```

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

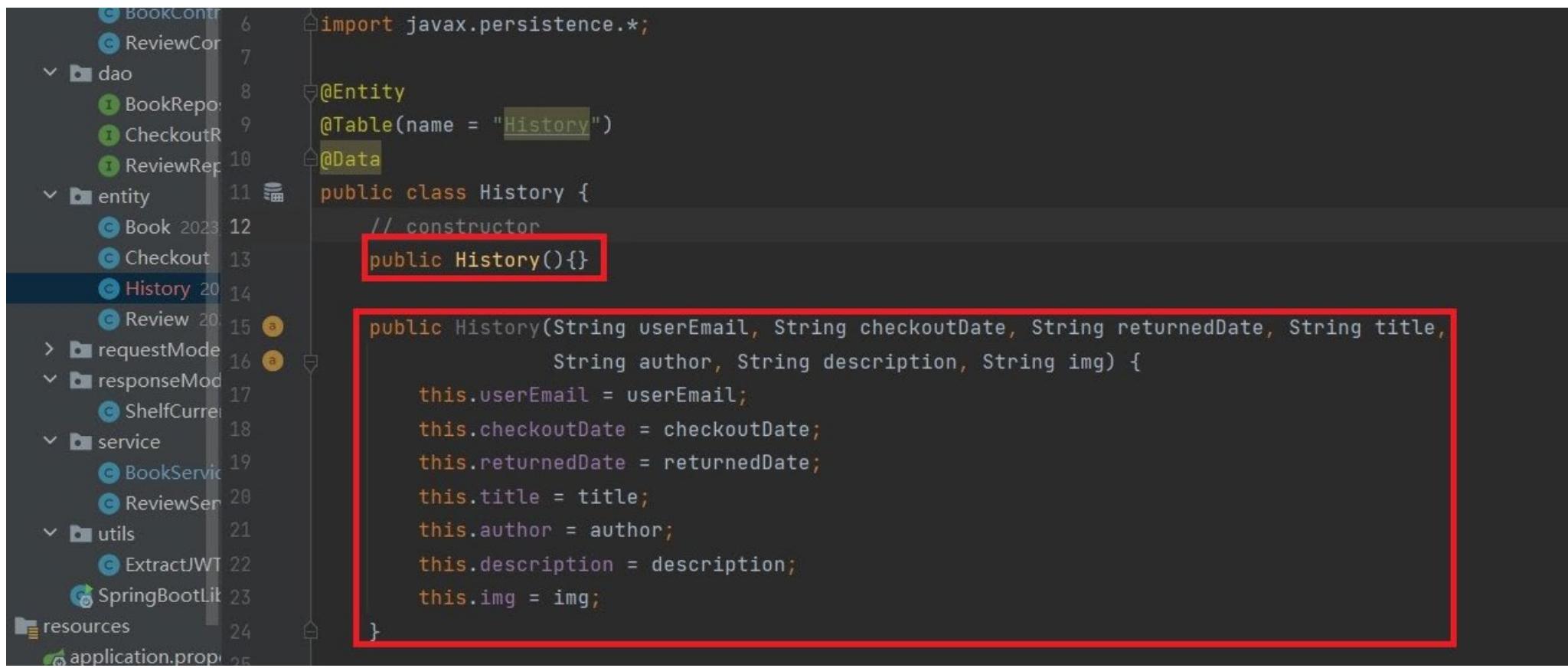


The screenshot shows a Java IDE interface with the following details:

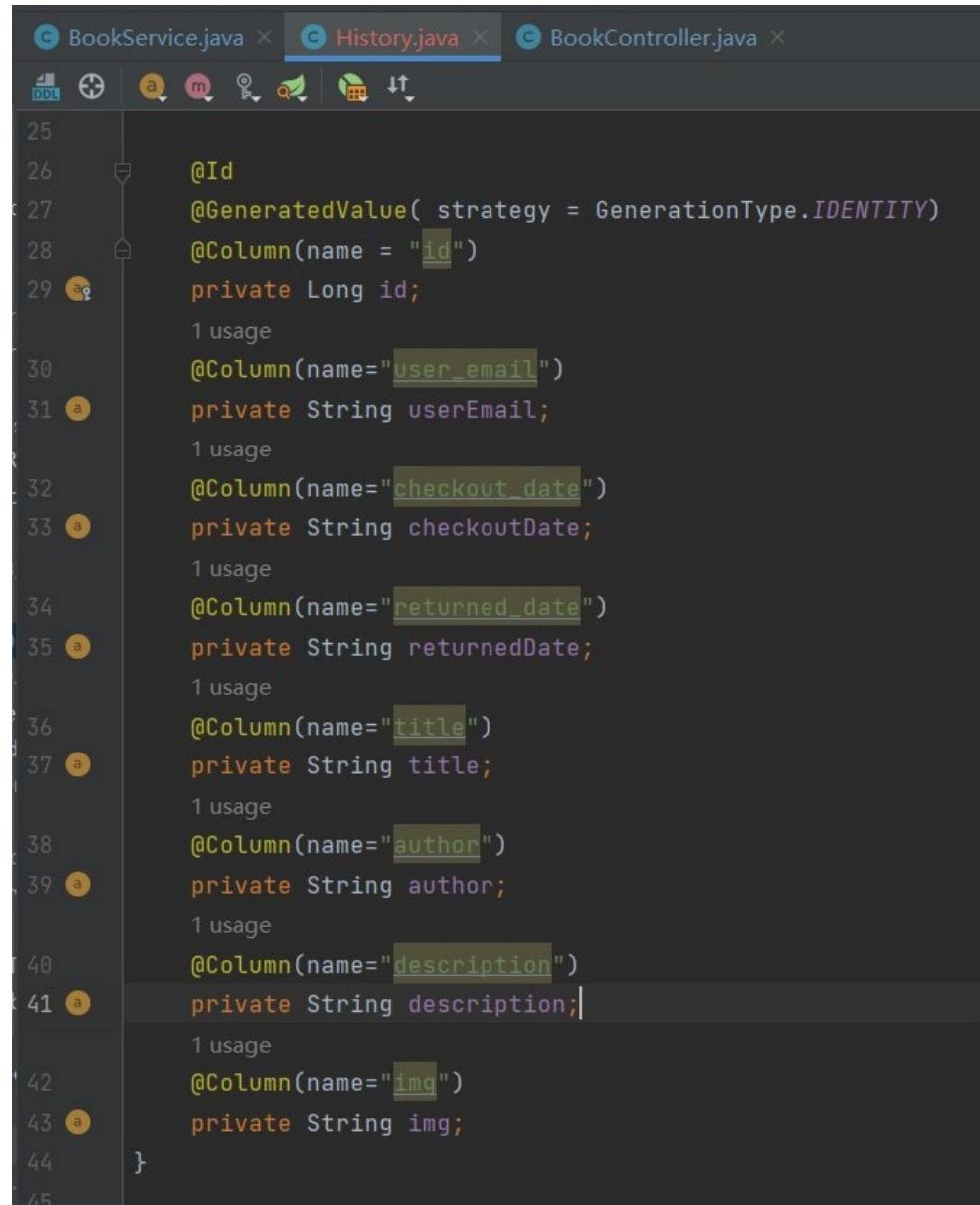
- Project Structure:** The left sidebar shows a project named "Spring-boot-library" with a .idea folder and a src directory containing main, java, and resources. The java directory has subfolders com.jac.Springbootlibrary, config, controller, dao, entity, and interfaces.
- File:** The main editor window displays the History.java file under the com.jac.Springbootlibrary.entity package.
- Annotations:** The code includes the following annotations:

```
1 package com.jac.Springbootlibrary.entity;
2
3 import lombok.Data;
4
5 import javax.persistence.Entity;
6 import javax.persistence.Table;
7
8
9 @Entity
10 @Table(name = "History")
11 @Data
12
13 public class History {
14 }
```
- Toolbars and Status:** The top bar includes standard file operations (File, Edit, View, Navigate, Code, Refactor, Build, Run, Tools, Git, Window, Help) and a status bar indicating the current file is "Spring-boot-library - History.java".

Step 3 create constructor.



```
import javax.persistence.*;
import java.util.*;  
  
@Entity  
@Table(name = "History")  
@Data  
public class History {  
    // constructor  
    public History(){  
  
        public History(String userEmail, String checkoutDate, String returnedDate, String title,  
                      String author, String description, String img) {  
            this.userEmail = userEmail;  
            this.checkoutDate = checkoutDate;  
            this.returnedDate = returnedDate;  
            this.title = title;  
            this.author = author;  
            this.description = description;  
            this.img = img;  
        }  
    }  
}
```



The screenshot shows a Java IDE interface with three tabs at the top: BookService.java, History.java (which is active), and BookController.java. The History.java tab contains Java code for a domain object. The code includes annotations for mapping properties to database columns: @Id, @GeneratedValue(strategy = GenerationType.IDENTITY), and @Column(name = "id"). Other properties like userEmail, checkoutDate, returnedDate, title, author, description, and img are also defined with their respective column names.

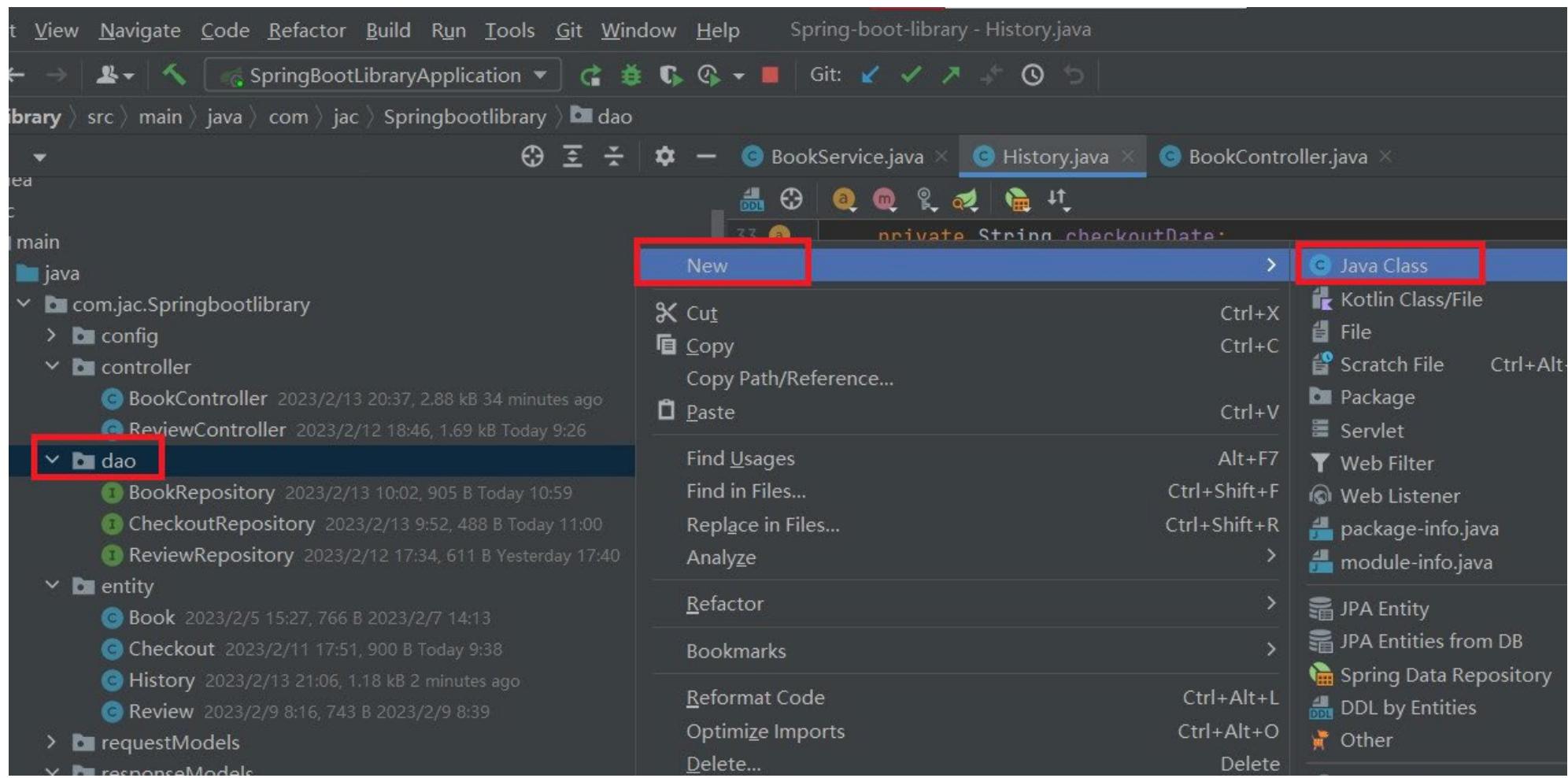
```
25  
26     @Id  
27     @GeneratedValue(strategy = GenerationType.IDENTITY)  
28     @Column(name = "id")  
29     private Long id;  
30     1 usage  
31     @Column(name="user_email")  
32     private String userEmail;  
33     1 usage  
34     @Column(name="checkout_date")  
35     private String checkoutDate;  
36     1 usage  
37     @Column(name="returned_date")  
38     private String returnedDate;  
39     1 usage  
40     @Column(name="title")  
41     private String title;  
42     1 usage  
43     @Column(name="author")  
44     private String author;  
45     1 usage  
46     @Column(name="description")  
47     private String description;  
48     1 usage  
49     @Column(name="img")  
50     private String img;  
51 }
```

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.

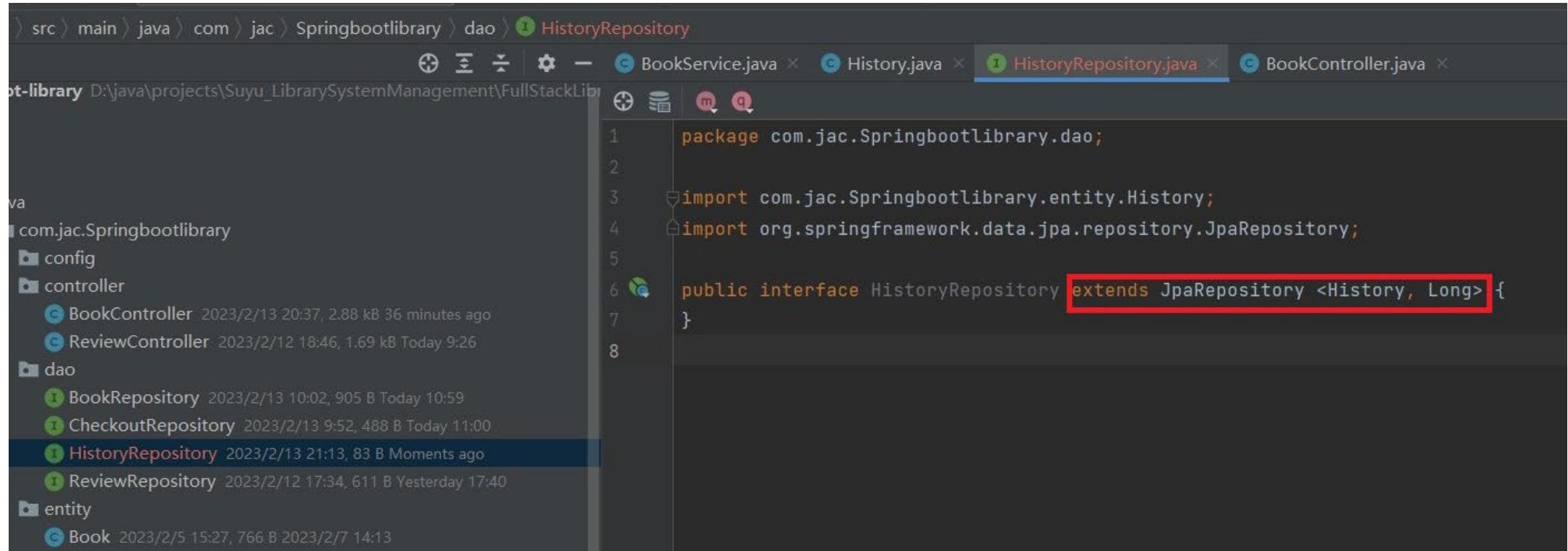


Choose “Interface”, Input the name “HistoryRepository”.

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

- Project Structure:** The left sidebar shows the project structure under "Spring-boot-library". The "src/main/java/com/jac/Springbootlibrary/dao" package is selected, containing sub-directories like "entity", "requestModels", and "responseModels".
- Code Editor:** The main editor area displays a portion of a Java file with annotations for columns: `@Column(name="returned_date") private String returnedDate;`, `@Column(name="title") private String title;`, and `@Column(name="author")`.
- Code Completion:** A completion dropdown is open at the bottom of the editor, listing options: "New Java Class", "HistoryRepository" (highlighted with a red box), "Class", "Interface" (highlighted with a blue box), "Enum", and "Annotation".
- Status Bar:** The bottom status bar shows the text "1s" and "16:54".

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

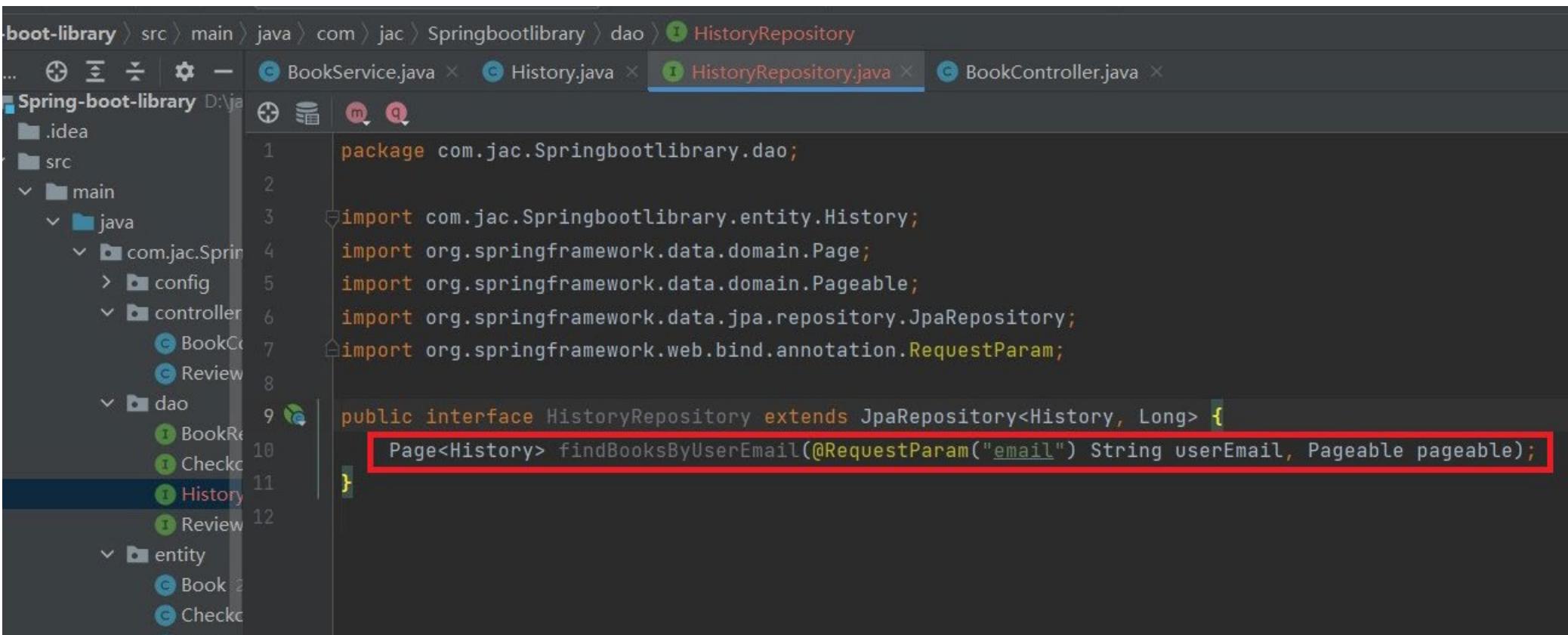


The screenshot shows a Java IDE interface with the following details:

- Project Structure:** The left sidebar shows a project tree with packages like `com.jac.Springbootlibrary` containing `config`, `controller` (with `BookController` and `ReviewController`), `dao` (with `BookRepository`, `CheckoutRepository`, `HistoryRepository` (selected), and `ReviewRepository`), and `entity` (with `Book`).
- File List:** The top navigation bar lists files: `BookService.java`, `History.java`, `HistoryRepository.java` (highlighted in green), and `BookController.java`.
- Code Editor:** The main editor area contains the following code for `HistoryRepository.java`:

```
package com.jac.Springbootlibrary.dao;  
import com.jac.Springbootlibrary.entity.History;  
import org.springframework.data.jpa.repository.JpaRepository;  
  
public interface HistoryRepository extends JpaRepository<History, Long> {  
}
```

Step 3 create an interface function findBooksByUserEmail().



The screenshot shows the IntelliJ IDEA interface with the file `HistoryRepository.java` selected in the tab bar. The code editor displays the following Java code:

```
1 package com.jac.Springbootlibrary.dao;
2
3 import com.jac.Springbootlibrary.entity.History;
4 import org.springframework.data.domain.Page;
5 import org.springframework.data.domain.Pageable;
6 import org.springframework.data.jpa.repository.JpaRepository;
7 import org.springframework.web.bind.annotation.RequestParam;
8
9 public interface HistoryRepository extends JpaRepository<History, Long> {
10     Page<History> findBooksByUserEmail(@RequestParam("email") String userEmail, Pageable pageable);
11 }
```

The line `Page<History> findBooksByUserEmail(@RequestParam("email") String userEmail, Pageable pageable);` is highlighted with a red rectangular selection.

3. SPRINGBOOT ENHANCE BOOKSERVICE FOR HISTORY



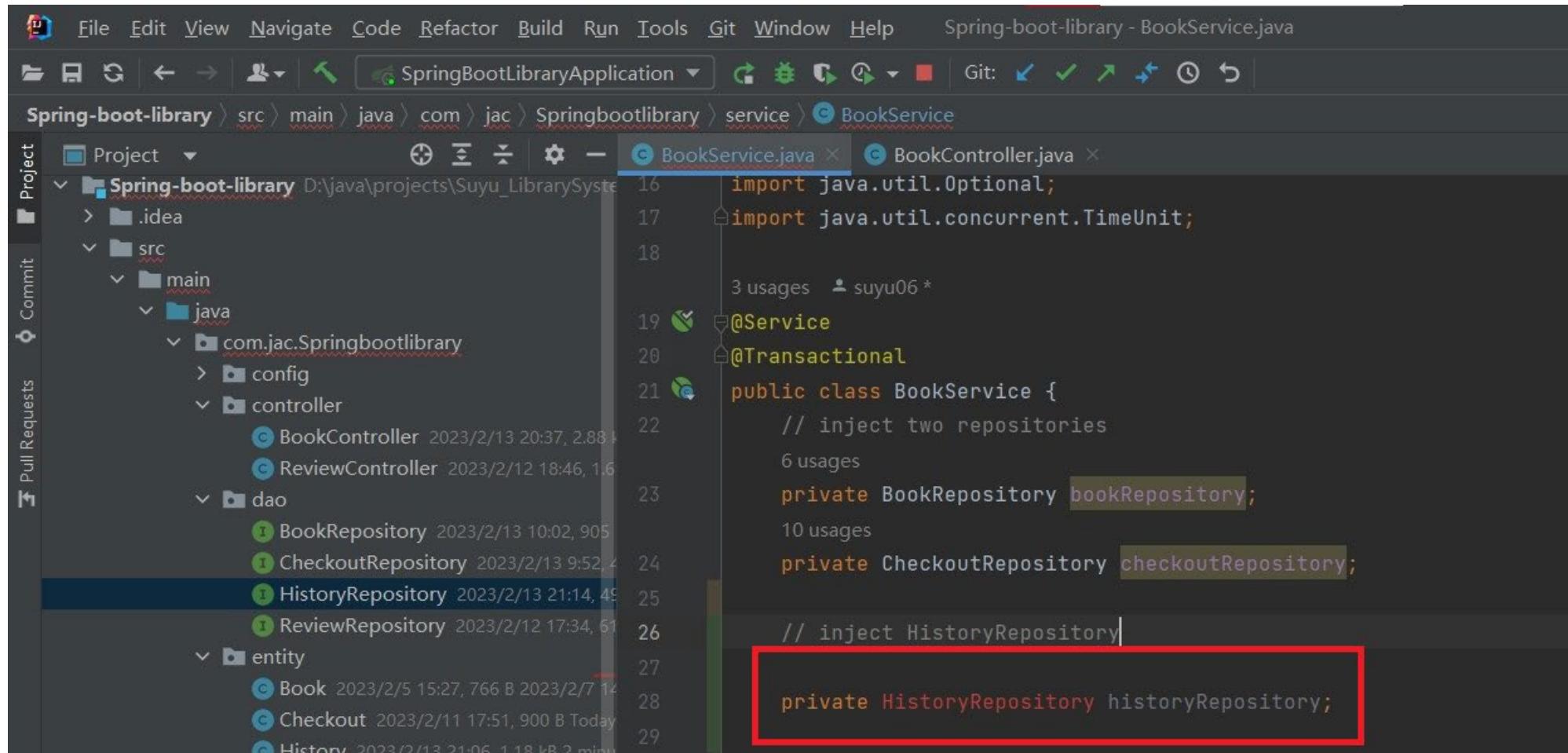
We have our history entity created and we have our history repository created.

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.



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

- Project Structure:** The project is named "Spring-boot-library". The "src/main/java/com/jac/Springbootlibrary/service" package contains the "BookService.java" file.
- Code Editor:** The "BookService.java" file is open. It contains Java code for a service class annotated with `@Service` and `@Transactional`. The code includes imports for `java.util.Optional` and `java.util.concurrent.TimeUnit`, and defines two private fields: `bookRepository` and `checkoutRepository`.
- Annotations:** The `@Service` annotation is highlighted in green, and the `bookRepository` field is also highlighted in green.
- Callout:** A red rectangular callout highlights the line of code: `private HistoryRepository historyRepository;`

```
File Edit View Navigate Code Refactor Build Run Tools Git Window Help Spring-boot-library - BookService.java
SpringBootLibraryApplication Project Commit Pull Requests
Spring-boot-library > src > main > java > com > jac > Springbootlibrary > service > BookService.java
import java.util.Optional;
import java.util.concurrent.TimeUnit;
3 usages suyu06 *
@Service
@Transactional
public class BookService {
    // inject two repositories
    6 usages
    private BookRepository bookRepository;
    10 usages
    private CheckoutRepository checkoutRepository;
    // inject HistoryRepository
    private HistoryRepository historyRepository;
}
```

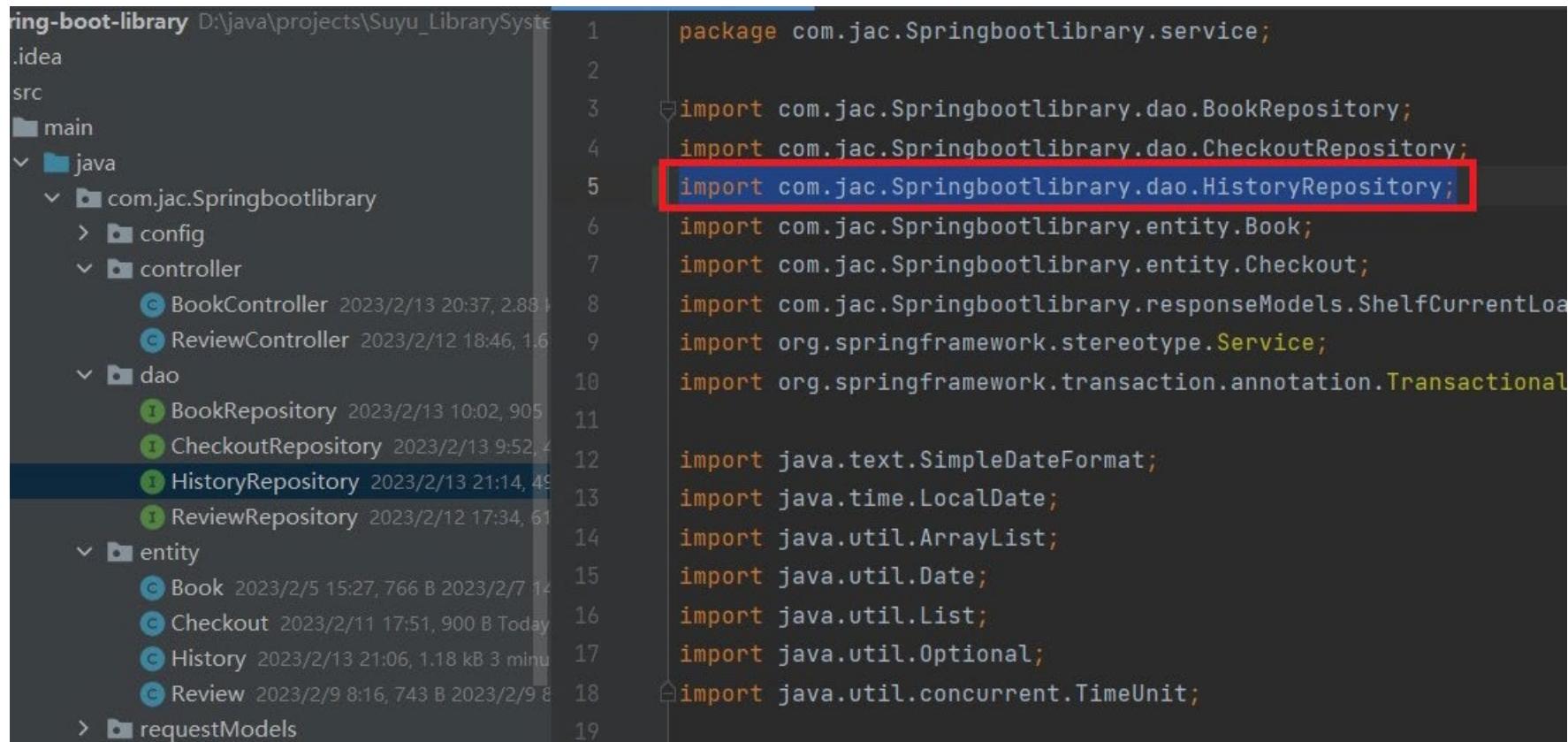
1.1 After initialize a new historyRepostiroy object. We will get a red error.
Hover on the HistoryRepository, click on “import class”.

The screenshot shows a Java project structure on the left and a code editor on the right. The code editor displays the following Java code:

```
16 import java.util.Optional;
17 import java.util.concurrent.TimeUnit;
18
19 3 usages suyu06 *
20
21 @Service
22 @Transactional
23 public class BookService {
24     // inject two repositories
25     6 usages
26     private BookRepository bookRepository;
27     10 usages
28     private CheckoutRepository checkoutRepository;
29
30     // inject HistoryRepository
31
32     private HistoryRepository historyRepository;
33
34     // use constructor dependency
35     suyu06
36
37     public BookService(BookRepository bookRepository, CheckoutRepository checkoutRepository) {
38         this.bookRepository = bookRepository;
39         this.checkoutRepository = checkoutRepository;
40         this.historyRepository = historyRepository;
41     }
42 }
```

A red box highlights the line `private HistoryRepository historyRepository;`. A tooltip appears above the line, stating `Cannot resolve symbol 'HistoryRepository'`. Below the tooltip, there is a button labeled `Import class`, which is also highlighted with a red box. A blue arrow points from the `Import class` button to the `historyRepository` field in the code.

HistoryRepository imported.



The screenshot shows a Java code editor with a file named `Service.java` open. The code is part of a Spring Boot application. The imports section includes several repository interfaces from the `com.jac.Springbootlibrary.dao` package. One specific import, `import com.jac.Springbootlibrary.dao.HistoryRepository;`, is highlighted with a red rectangular box. The code also includes imports for `Book`, `Checkout`, `SimpleDateFormat`, `LocalDate`, `ArrayList`, `Date`, `List`, `Optional`, and `TimeUnit`. The code editor's sidebar shows the project structure with modules like `main`, `java`, and `requestModels`, and files like `BookController`, `ReviewController`, `HistoryRepository`, and `ReviewRepository`.

```
ring-boot-library D:\java\projects\Suyu_LibrarySystem
.idea
src
└── main
    └── java
        └── com.jac.Springbootlibrary
            ├── config
            ├── controller
            │   ├── BookController
            │   └── ReviewController
            ├── dao
            │   ├── BookRepository
            │   ├── CheckoutRepository
            │   ├── HistoryRepository
            │   └── ReviewRepository
            └── entity
                ├── Book
                ├── Checkout
                ├── History
                └── Review
        └── requestModels
1  package com.jac.Springbootlibrary.service;
2
3  import com.jac.Springbootlibrary.dao.BookRepository;
4  import com.jac.Springbootlibrary.dao.CheckoutRepository;
5  import com.jac.Springbootlibrary.dao.HistoryRepository;
6  import com.jac.Springbootlibrary.entity.Book;
7  import com.jac.Springbootlibrary.entity.Checkout;
8  import com.jac.Springbootlibrary.responseModels.ShelfCurrentLoa
9  import org.springframework.stereotype.Service;
10 import org.springframework.transaction.annotation.Transactional
11
12 import java.text.SimpleDateFormat;
13 import java.time.LocalDate;
14 import java.util.ArrayList;
15 import java.util.Date;
16 import java.util.List;
17 import java.util.Optional;
18 import java.util.concurrent.TimeUnit;
```

```
21  @Transactional
22  public class BookService {
23      // inject two repositories
24      private BookRepository bookRepository;
25      10 usages
26      private CheckoutRepository checkoutRepository;
27
28      // inject HistoryRepository
29
30      1 usage
31      private HistoryRepository historyRepository;
32
33      // use constructor dependency injection to set up all our re
34      public BookService(BookRepository bookRepository, CheckoutRe
35          HistoryRepository historyRepository) {
36              this.bookRepository = bookRepository;
37              this.checkoutRepository = checkoutRepository;
38              this.historyRepository = historyRepository;|
```

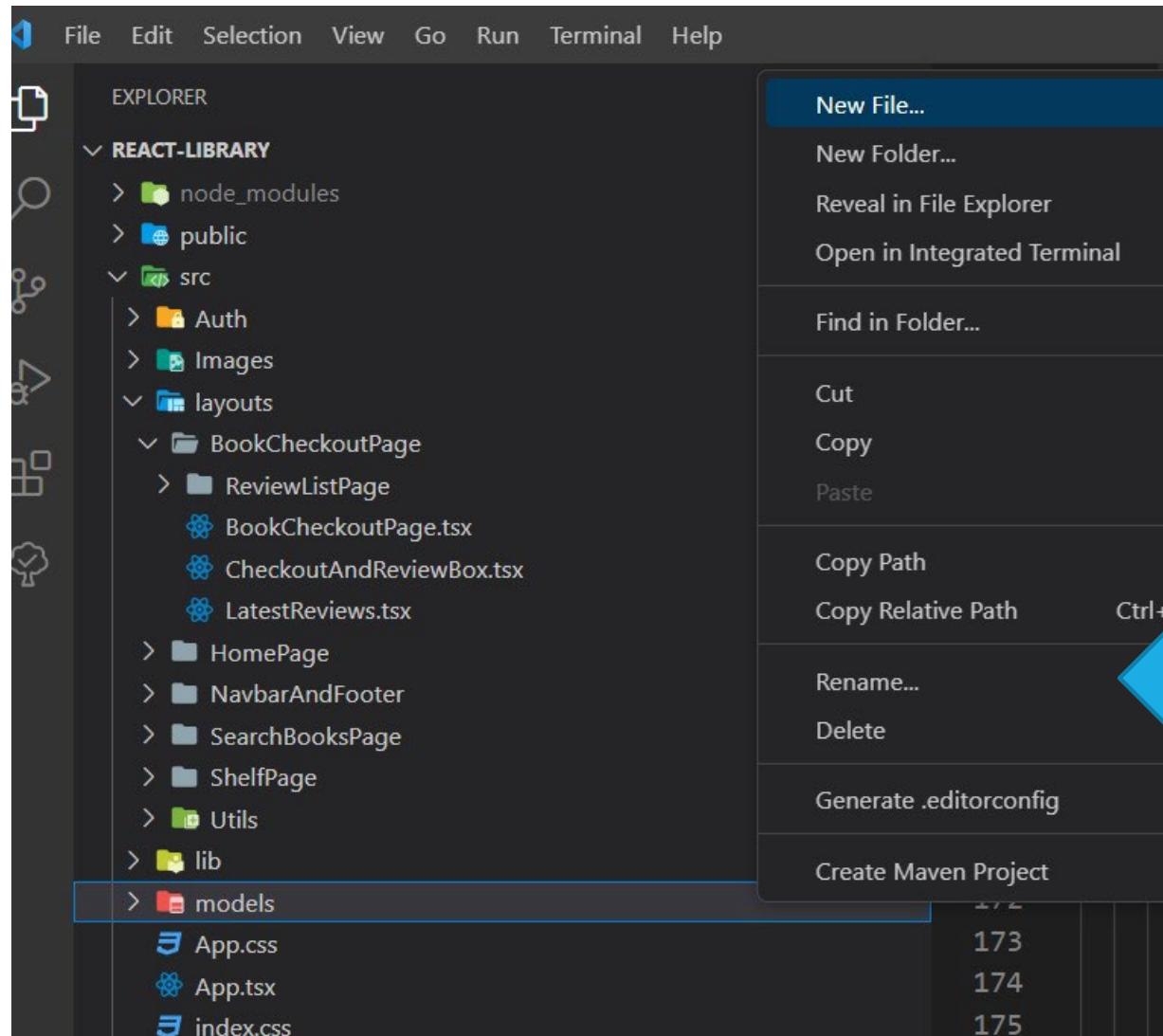
1.2 Add the historyRepository to the constructor.

```
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.findByUserEmailAndBookId(userEmail, bookId);
    // if not exists:
    if (!book.isPresent() || validateCheckout == null) {
        throw new Exception("Book does not exist or not checked out by " + userEmail);
    }
    //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`.

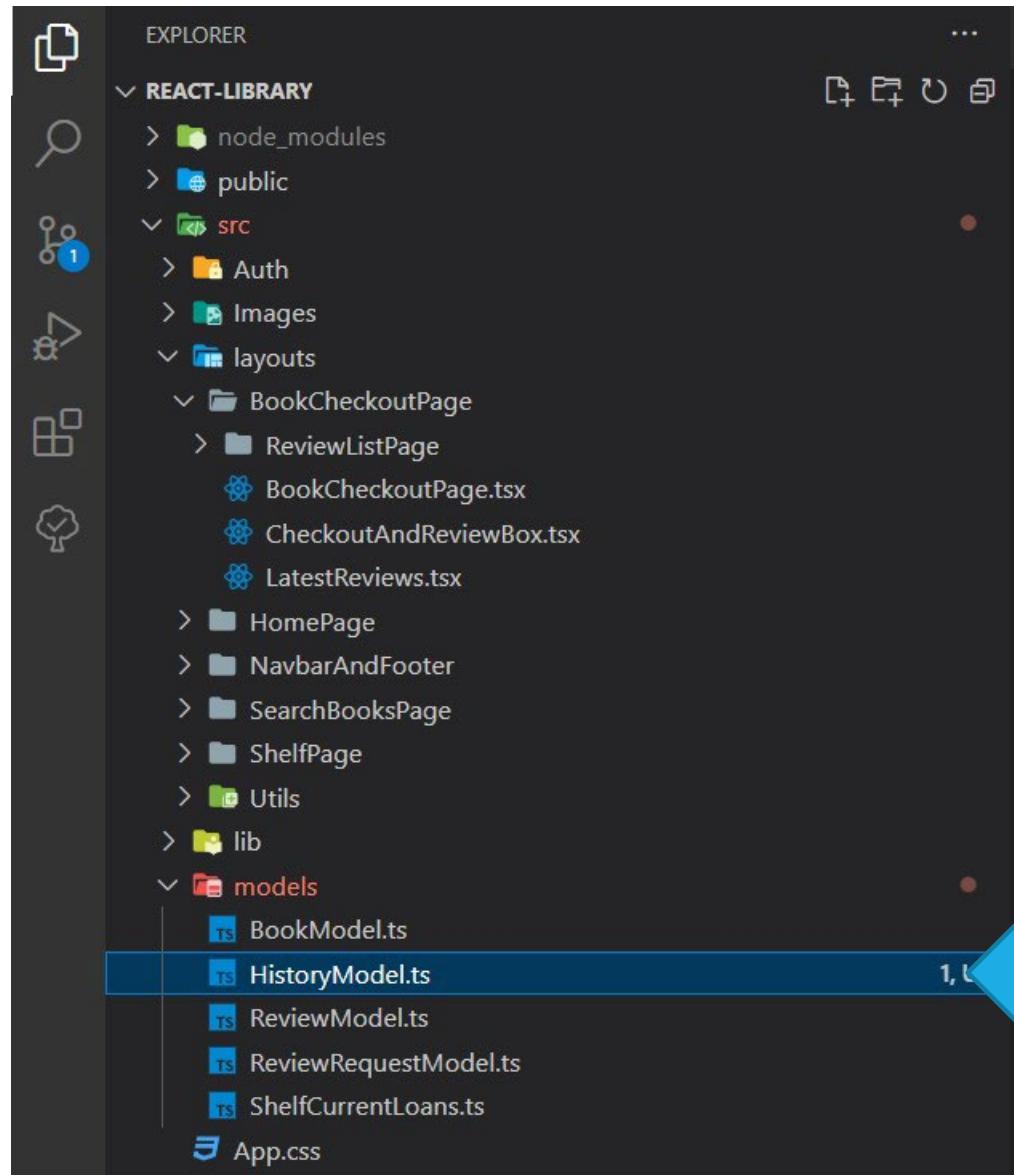
5. REACT HISTORY MODEL





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,



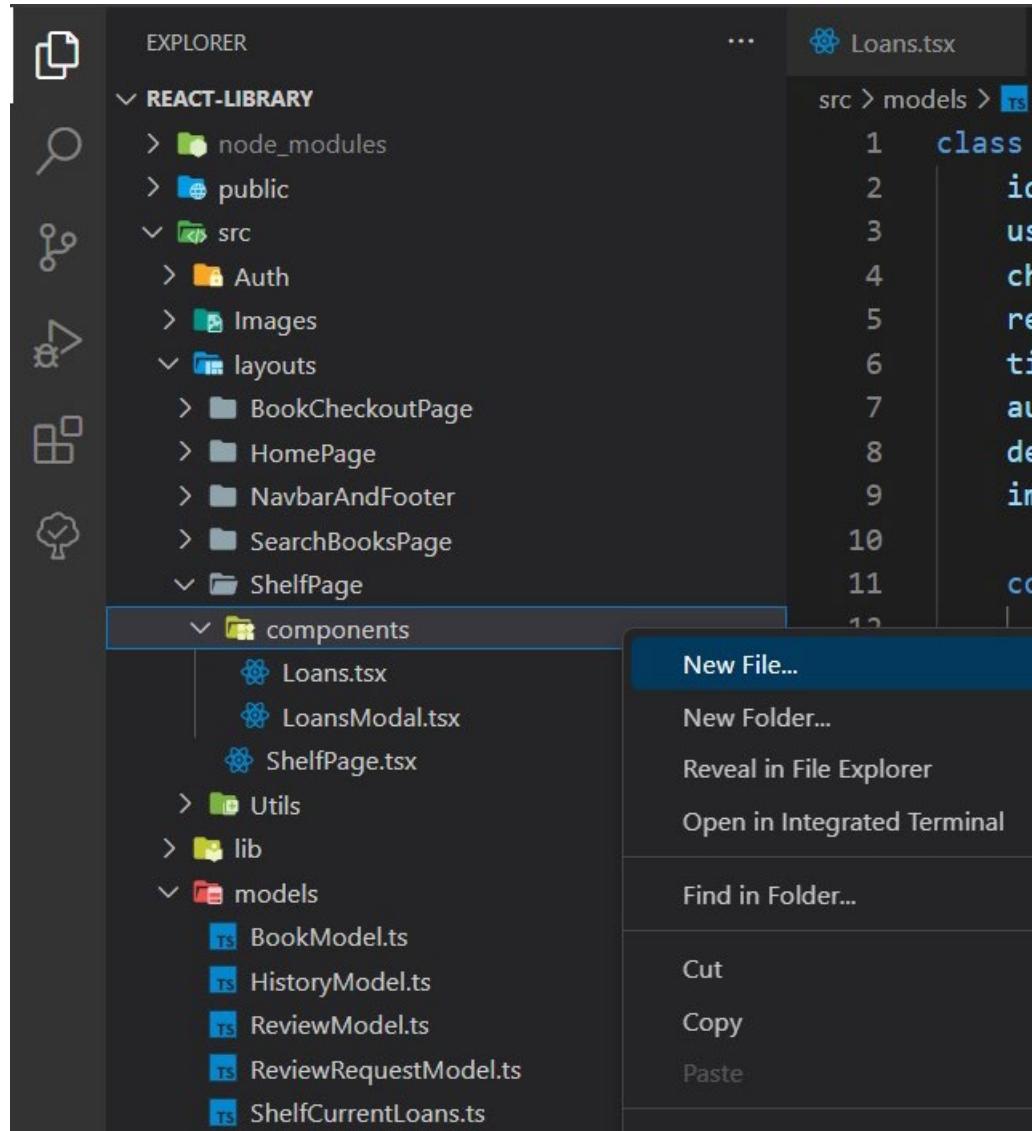
Name it HistoryModel.ts

```
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

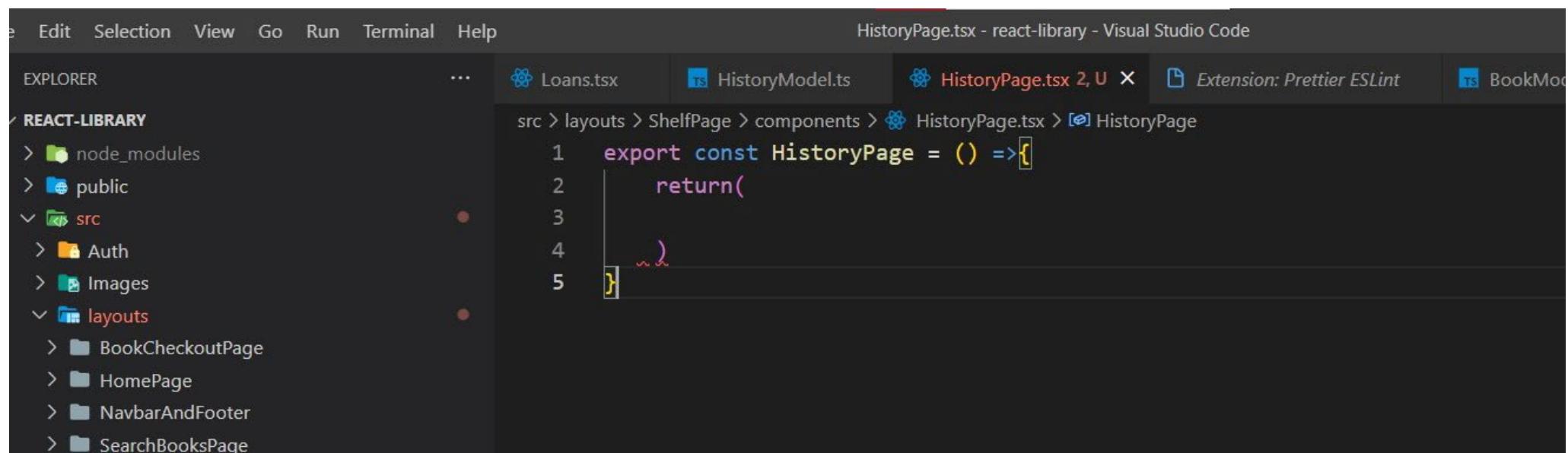




Step1

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.



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

- File Explorer (EXPLORER):** Shows the project structure under "REACT-LIBRARY". The "src" folder is expanded, revealing "Auth", "Images", and "layouts" subfolders, each containing "BookCheckoutPage", "HomePage", "NavbarAndFooter", and "SearchBooksPage" files.
- Editor:** The "HistoryPage.tsx" file is open. The code is as follows:

```
1  export const HistoryPage = () =>[  
2      return(  
3          ...  
4      )  
5  ]
```
- Status Bar:** Shows the current file path: "src > layouts > ShelfPage > components > HistoryPage.tsx".
- Top Bar:** Shows tabs for "Loans.tsx", "HistoryModel.ts", "HistoryPage.tsx 2, U X", "Extension: Prettier ESLint", and "BookMod".
- Menu Bar:** Shows "Edit", "Selection", "View", "Go", "Run", "Terminal", and "Help".

Step 3 add useState and import useOktaAuth.

```
src > layouts > ShelfPage > components > HistoryPage.tsx > HistoryPage
  1  import { useOktaAuth } from "@okta/okta-react";
  2  import { useState } from "react";
  3  import HistoryModel from "../../../../../models/HistoryModel";
  4
  5  export const HistoryPage = () =>{
  6
  7    //useState
  8    const { authState } = useOktaAuth();
  9    const [isLoadingHistory, setIsLoadingHistory] = useState(true);
 10    const [httpError, setHttpError] = useState(null);
 11    //Histories
 12    const [histories, setHistories] = useState<HistoryModel[]>([]);
 13
 14    //Pagination
 15    const [currentPage, setCurrentPage] = useState(1);
 16    const [totalPages, setTotalPages] = useState(0);
 17
```

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.

The image shows a code editor interface with a sidebar on the left displaying a file tree. The tree includes files like Loans.tsx, LoansModal.tsx, ShelfPage.tsx, Utils, lib, and models (which contains BookModel.ts, HistoryModel.ts, ReviewModel.ts, ReviewRequestModel.ts, and ShelfCurrentLoans.ts). Below the tree are App.css and App.tsx. The main editor area shows a snippet of TypeScript code:

```
17
18 // useEffect
19 useEffect [() => {
20   const fetchUserHistory = async () => {
21     }
22   fetchUserHistory().catch(error: any) => {
23     setIsLoadingHistory(false);
24     setHttpError(error.message);
25   }
26 }, [authState, currentPage]]
27
28
29 return(
```

Specific parts of the code are highlighted with red boxes: the declaration of the fetchUserHistory function, the catch block, and the dependency array [authState, currentPage].

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

```
27 |     })
28 |   ,[authState, currentPage]);
29 |
30 |   if (isLoadingHistory) {
31 |     return (
32 |       <SpinnerLoading/>
33 |     );
34 |   }
35 |
36 |   if (httpError) {
37 |     return (
38 |       <div className='container m-5'>
39 |         <p>{httpError}</p>
40 |       </div>
41 |     );
42 |   }
43 |
44 |   const paginate = (pageNumber: number) => setCurrentPage(pageNumber);
45 | }
```

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 type 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:

Loans Your History

Recent History:

Book image



Luv, Judy
Crash Course in Big Data

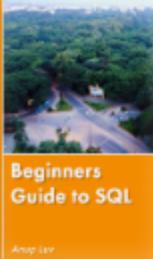
Morbi eu tempus eros, in imperdiet sem. Nulla sed sagittis nisl, porttitor fringilla libero. Nullam ut urna aliquet, hendrerit quam in, dignissim diam. In in nibh vel nisi fermentum pretium sit amet vitae mi. Pellentesque eget augue efficitur, volutpat tellus eget, fringilla augue. Pellentesque tempus mi ac risus lacinia, et tincidunt lectus rutrum. Nullam et nibh a odio luctus tincidunt nec in ipsum. Sed ac est nulla. Nulla purus turpis, dignissim sit amet euismod lobortis, consequat ut dui. Maecenas commodo velit in elementum placerat. Nam sit amet blandit ante, sit amet mollis neque. Ut placerat venenatis leo sit amet dapibus. Nunc varius cursus lobortis. Aenean euismod dui at diam euismod aliquet. Fusce feugiat orci nec commodo placerat.

Checked out on: 2023-02-13
Returned on: 2023-02-13

Date
Checked out date
Returned date

Book info:
Author
Book title
Book description

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



Luv, Anup

Beginners Guide to SQL

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.

Checked out on: 2023-02-14

Returned on: 2023-02-14

[First Page](#) [1](#) [2](#) [Last Page](#)

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>
              </div>
            </div>
          </div>
        ])}
    ) : (
      <div>
        <h3>No Recent History</h3>
      </div>
    )
  </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”.

The screenshot shows a code editor with a dark theme. The active file is `HistoryPage.tsx`. The code displays a list of library histories using a map function. Each history item is rendered as a card with the following structure:

```
<div>
  <div>
    <div>
      <img alt="Default" />
    </div>
    <div>
      <div>
        <div>
          <h5> {history.author} </h5>
          <h4>{history.title}</h4>
          <p> {history.description}</p>
          <hr />
          <p> Checked out on: {history.checkoutDate}</p>
          <p> Returned on: {history.returnedDate}</p>
        </div>
      </div>
    </div>
  </div>
</div>
```

Two specific sections of the code are highlighted with boxes: the book info part (author, title, description) is highlighted with a yellow box, and the date section (checkout and return dates) is highlighted with a red box. Blue arrows point from the numbered labels on the right to these respective highlighted sections.

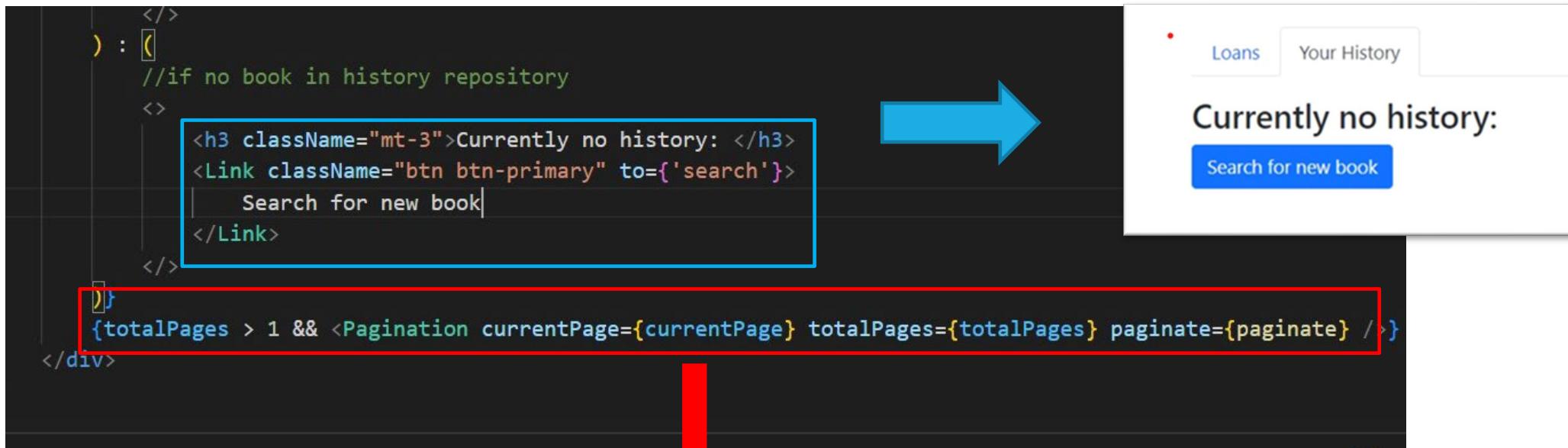
2. Book info part

Including author, book title , book description.

3. Date

Including checked out date, returned date.,

4. If user has no history to present:



The image shows a code editor on the left and a web application screenshot on the right. A blue arrow points from the code editor to the application screenshot.

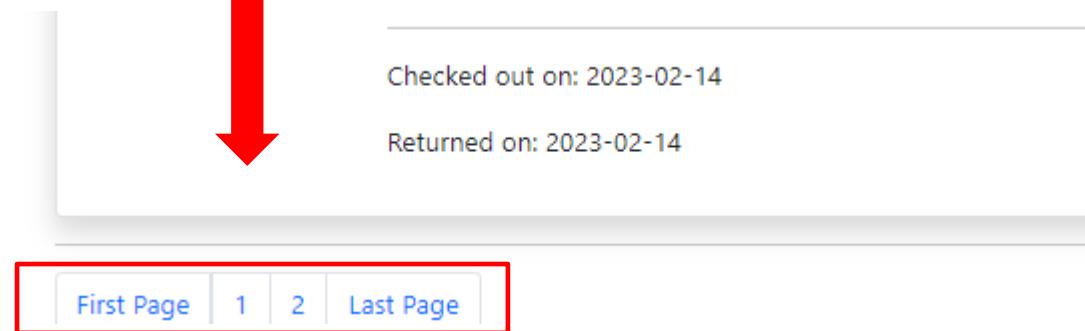
Code Editor (Left):

```
        </>
    ) : [
      //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>
      </>
    ]
  </div>
```

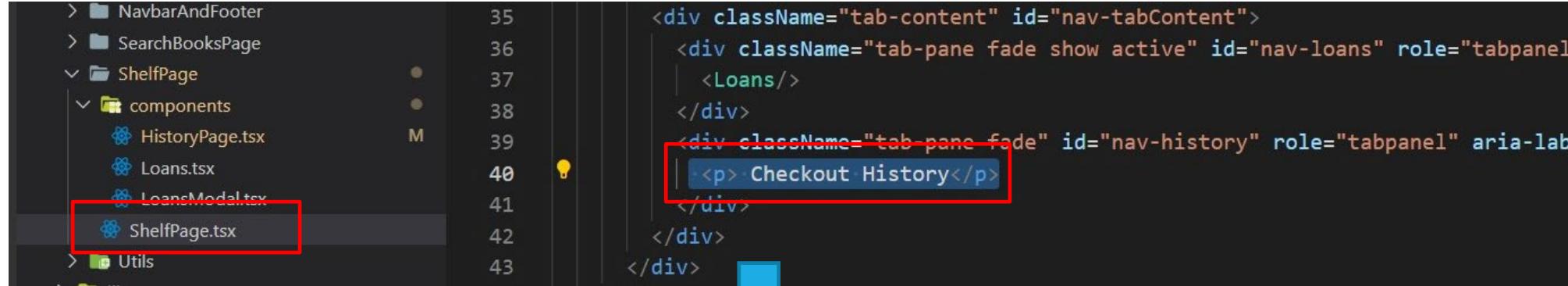
Web Application Screenshot (Right):

The screenshot shows a navigation bar with 'Loans' and 'Your History'. Below it, a message says 'Currently no history:' with a 'Search for new book' button.

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.



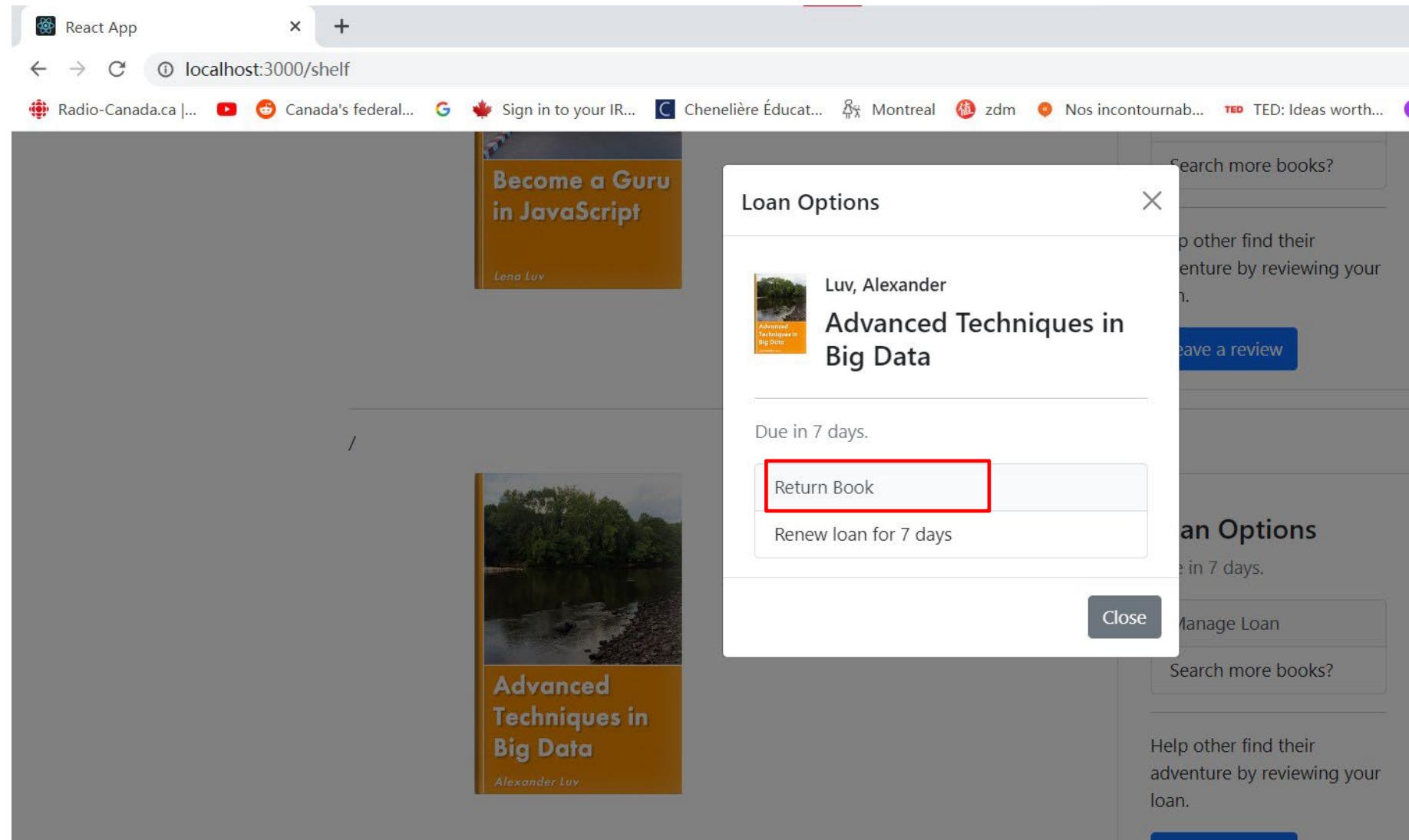
```
> └── NavbarAndFooter
> └── SearchBooksPage
└── ShelfPage
  └── components
    ├── HistoryPage.tsx
    ├── Loans.tsx
    ├── LoansModal.tsx
    └── ShelfPage.tsx
  └── Utils
  └── ...
  35   <div className="tab-content" id="nav-tabContent">
  36     <div className="tab-pane fade show active" id="nav-loans" role="tabpanel">
  37       <Loans/>
  38     </div>
  39     <div className="tab-pane fade" id="nav-history" role="tabpanel" aria-label="History">
  40       <p>Checkout History</p>
  41     </div>
  42   </div>
  43 </div>
```

```
> └── SearchBooksPage
└── ShelfPage
  └── components
    ├── HistoryPage.tsx
    ├── Loans.tsx
    ├── LoansModal.tsx
    └── ShelfPage.tsx
  └── Utils
  └── lib
  └── models
  36   <div className="tab-content" id="nav-tabContent">
  37     <div className="tab-pane fade show active" id="nav-loans" role="tabpanel">
  38       <Loans/>
  39     </div>
  40     <div className="tab-pane fade" id="nav-history" role="tabpanel" aria-label="History">
  41       <HistoryPage/>
  42     </div>
  43   </div>
  44 </div>
  45 </div>
```

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

The screenshot shows a web browser window with the title "React App" and the URL "localhost:3000/shelf". The browser's address bar also shows "localhost:3000/shelf". The page content is a React application. At the top, there is a navigation bar with links: "JAC Read", "Home", "Search Books", and "Shelf". Below the navigation bar, there are two tabs: "Loans" (which is selected, indicated by a blue border) and "Your History". A message "Currently no history:" is displayed. At the bottom, there is a blue button with the text "Search for new book".

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.

A screenshot of a web browser window titled "React App". The address bar shows "localhost:3000/shelf". The page content is a dark blue header with "JAC Read" and navigation links "Home", "Search Books", and "Shelf". Below the header, there are two tabs: "Loans" (selected) and "Your History". A main message says "Currently no history:" and a blue button says "Search for new book". The browser's toolbar includes back, forward, refresh, search, and other standard icons.

React App

localhost:3000/shelf

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... Activities

JAC Read Home Search Books Shelf

Loans Your History

Currently no history:

Search for new book

When we refresh the page, it begins 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.

localhost:3000/shelf

fringilla. Sed ornare tellus enim, a tincidunt libero dictum vitae. Proin bibendum posuere dui. Donec sagittis neque massa, sed semper nulla vehicula at.

Checked out on: 2023-02-12

Returned on: 2023-02-14



Luv, Alexander

Advanced Techniques in Big Data

Nunc eget lorem ac neque tincidunt mollis. Fusce finibus laoreet nunc nec hendrerit. Curabitur eu placerat urna, sit amet pellentesque enim. Donec velit ligula, congue eu lobortis vel, interdum nec tellus. Nulla nisl ipsum, porta non egestas sed, vulputate quis nisi. Etiam pellentesque in velit non convallis. Nullam id risus quis augue posuere sodales vel maximus justo. Maecenas nec leo a nibh aliquet placerat nec sed massa. Duis sit amet nisi libero. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Phasellus non viverra dolor. Pellentesque ligula mauris, congue quis neque quis, mollis scelerisque ligula. Pellentesque semper, erat commodo luctus mollis, nulla ipsum consectetur dolor, quis blandit massa sem fringilla libero. Maecenas eget mi nec est condimentum fermentum. Vivamus vehicula est sit amet ante gravida, eu finibus quam elementum. Proin egestas leo eu sagittis euismod.

Checked out on: 2023-02-20

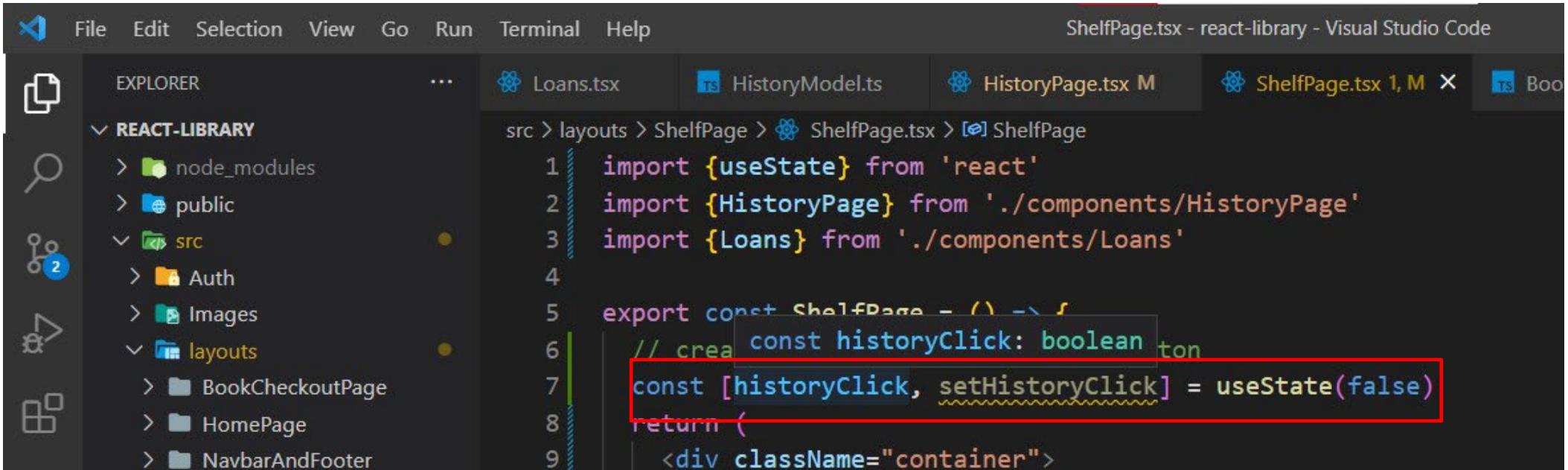
Returned on: 2023-02-20

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.



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

- File Bar:** File, Edit, Selection, View, Go, Run, Terminal, Help
- Title Bar:** ShelfPage.tsx - react-library - Visual Studio Code
- Explorer:** Shows a tree view of the project structure under "REACT-LIBRARY". It includes "node_modules", "public", "src" (which contains "Auth", "Images", and "layouts" folders), and "layouts" (which contains "BookCheckoutPage", "HomePage", and "NavbarAndFooter" files).
- Editor:** The "ShelfPage.tsx" file is open. The code is as follows:

```
src > layouts > ShelfPage > ShelfPage.tsx > ShelfPage
1 import {useState} from 'react'
2 import {HistoryPage} from './components/HistoryPage'
3 import {Loans} from './components/Loans'
4
5 export const ShelfPage = () => {
6   // crea const historyClick: boolean ton
7   const [historyClick, setHistoryClick] = useState(false)
8   return (
9     <div className="container">
```

A red box highlights the line of code where the state is being created: `const [historyClick, setHistoryClick] = useState(false)`.

Step 2 Scroll down until we see our historyPage component.

```
34         </div>
35     </nav>
36     <div className="tab-content" id="nav-tabContent">
37       <div className="tab-pane fade show active" id="nav-loans" role="tabpanel" aria-labelledby="nav-loans-tab">
38         <Loans/>
39       </div>
40       <div className="tab-pane fade" id="nav-history" role="tabpanel" aria-labelledby="nav-history-tab">
41         <HistoryPage/>|
42       </div>
43     </div>
44   </div>
45 </div>
46 );
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE node +

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

```
layouts > ShelfPage > ShelfPage.tsx > ShelfPage

const [historyClick, setHistoryClick] = useState(false)
return (
  <div className="container">
    <div className="mt-3">
      <nav>
        <div className="nav nav-tabs" id="nav-tab" role="tablist">
          <button
            onClick={() => setHistoryClick(false)}
            className="nav-link active"
            id="nav-loans-tab"
            data-bs-toggle="tab"
            data-bs-target="#nav-loans"
            type="button"
            role="tab"
            aria-controls="nav-loans"
            aria-selected="true">
            Loans
          </button>
          <button
            onClick={() => setHistoryClick(true)}
            className="nav-link"
            id="nav-history-tab"
            data-bs-toggle="tab"
            data-bs-target="#nav-history"
            type="button"
            role="tab">
            History
          </button>
        </div>
      </nav>
    </div>
  </div>
)
```

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.

The screenshot shows a web browser window with the title "React App" and the URL "localhost:3000/shelf". The browser's address bar also displays "localhost:3000/shelf". The page content is from a library application titled "JAC Read". The navigation bar includes links for "Home", "Search Books", "Shelf", and "Logout". Below the navigation bar, there are two tabs: "Loans" and "Your History", with "Your History" being the active tab. The main content area is titled "Recent History:" and lists two borrowed books:

Luv, Judy
Crash Course in Big Data
Morbi eu tempus eros, in imperdiet sem. Nulla sed sagittis nisl, porttitor fringilla libero. Nullam ut urna aliquet, hendrerit quam in, dignissim diam. In in nibh vel nisi fermentum pretium sit amet vitae mi. Pellentesque eget augue efficitur, volutpat tellus eget, fringilla augue. Pellentesque tempus mi ac risus lacinia, et tincidunt lectus rutrum. Nullam et nibh a odio luctus tincidunt nec in ipsum. Sed ac est nulla. Nulla purus turpis, dignissim sit amet euismod lobortis, consequat ut dui. Maecenas commodo velit in elementum placerat. Nam sit amet blandit ante, sit amet mollis neque. Ut placerat venenatis leo sit amet dapibus. Nunc varius cursus lobortis. Aenean euismod dui at diam euismod aliquet. Fusce feugiat orci nec commodo placerat.
Checked out on: 2023-02-13
Returned on: 2023-02-13

Luv, Alexander
Advanced Techniques in Big Data
Nunc eget lorem ac neque tincidunt mollis. Fusce finibus laoreet nunc nec hendrerit. Curabitur eu placerat urna, sit amet pellentesque enim. Donec velit ligula, congue eu lobortis vel, interdum nec tellus. Nulla nisl ipsum, porta

Check in our database.

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

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

- Navigator:** Shows the schema structure. Under the "reactlibrarydatabase" schema, the "history" table is selected.
- SQL Editor:** Displays the SQL query: `SELECT * FROM reactlibrarydatabase.history;`
- Result Grid:** Shows the data from the history table. Two rows are present, both corresponding to the book "Crash Course in Big Data".

	id	user_email	checkout_date	returned_date	title	author	description	img
1	1	testuser2@email.com	2023-02-13	2023-02-13	Crash Course in Big Data	Luv, Judy	Morbi eu tempus eros, in imperdiet sem. Nulla s...	BLOB
*	2	testuser2@email.com	2023-02-13	2023-02-13	Advanced Techniques in Big Data	Luv, Alexander	neque tincidunt mollis. Fusc...	BLOB