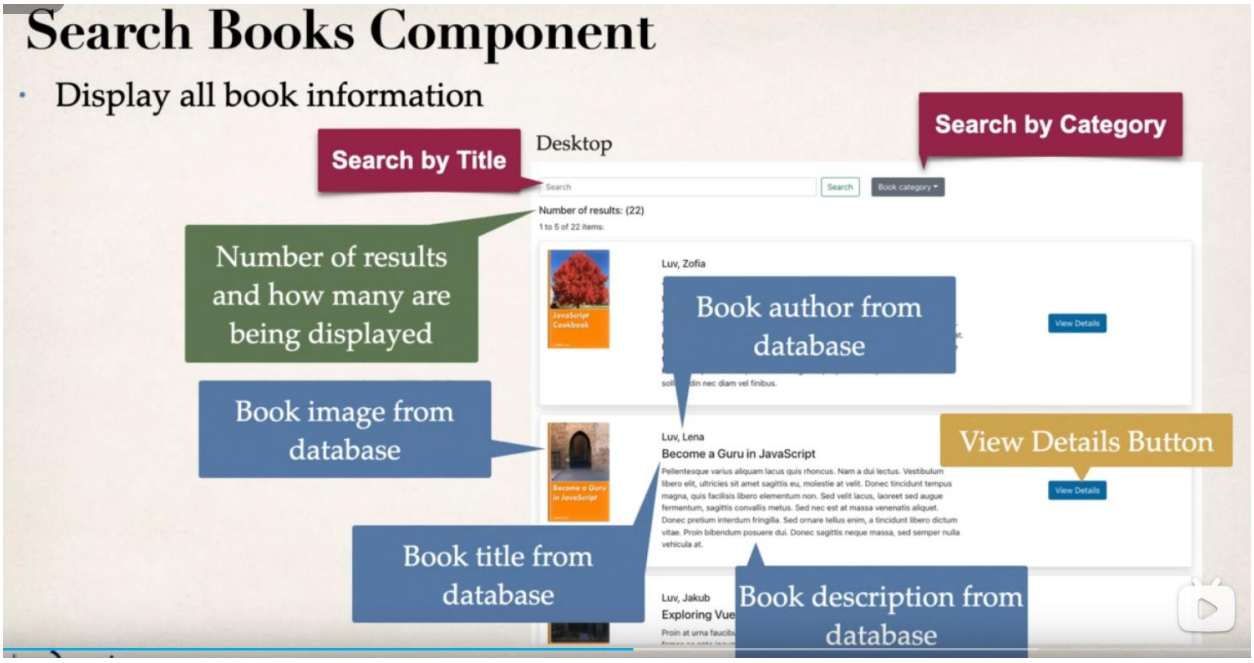
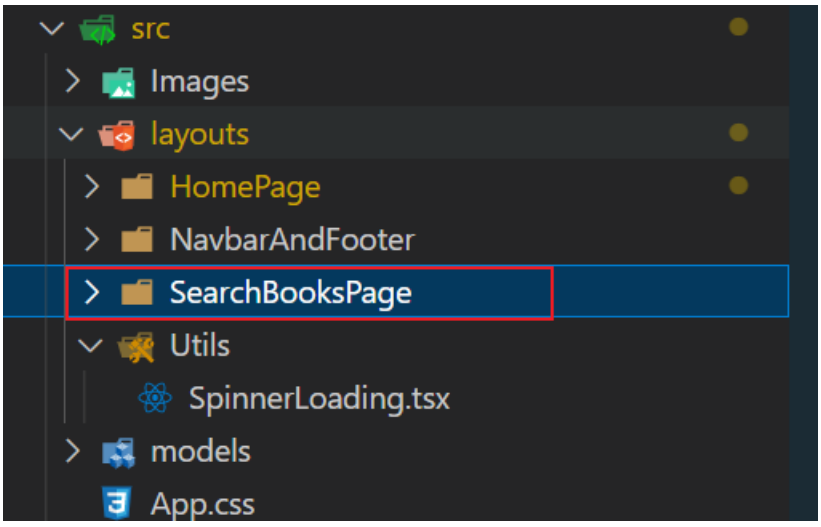
S04 SEARCH BOOKS COMPONENT

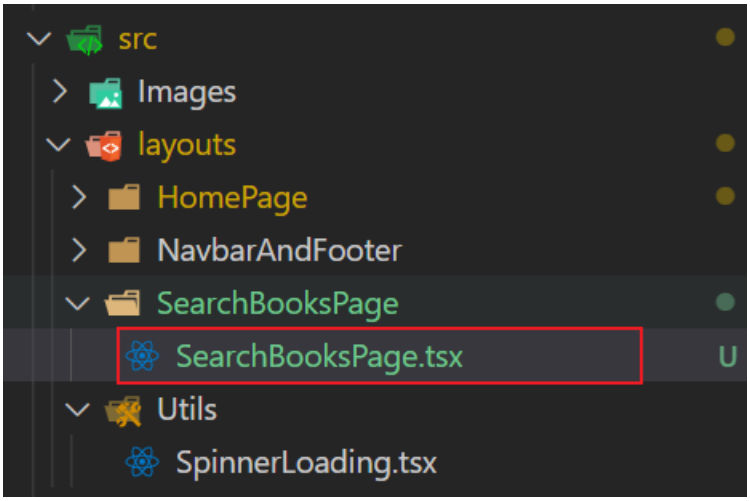


1. REACT-SEARCH BOOKS PAGE

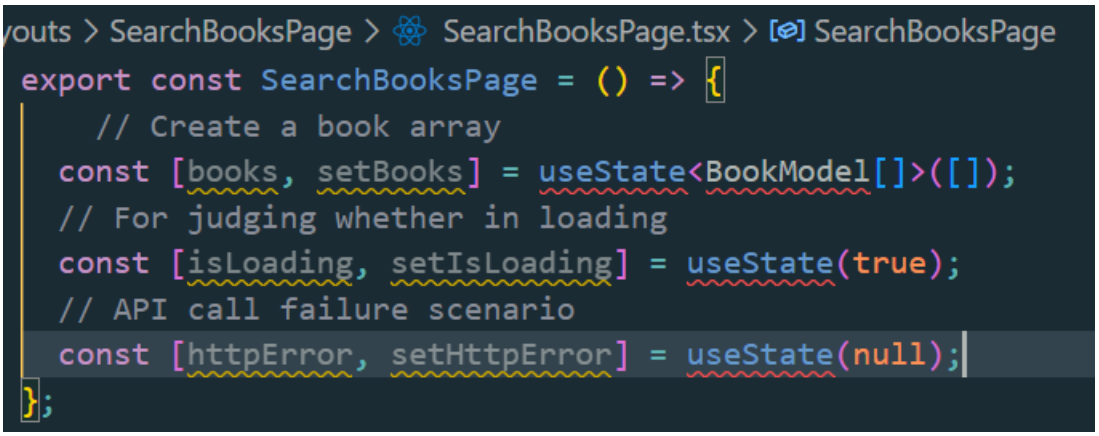
Create a new folder “SearchBooksPage” under the layouts folder.



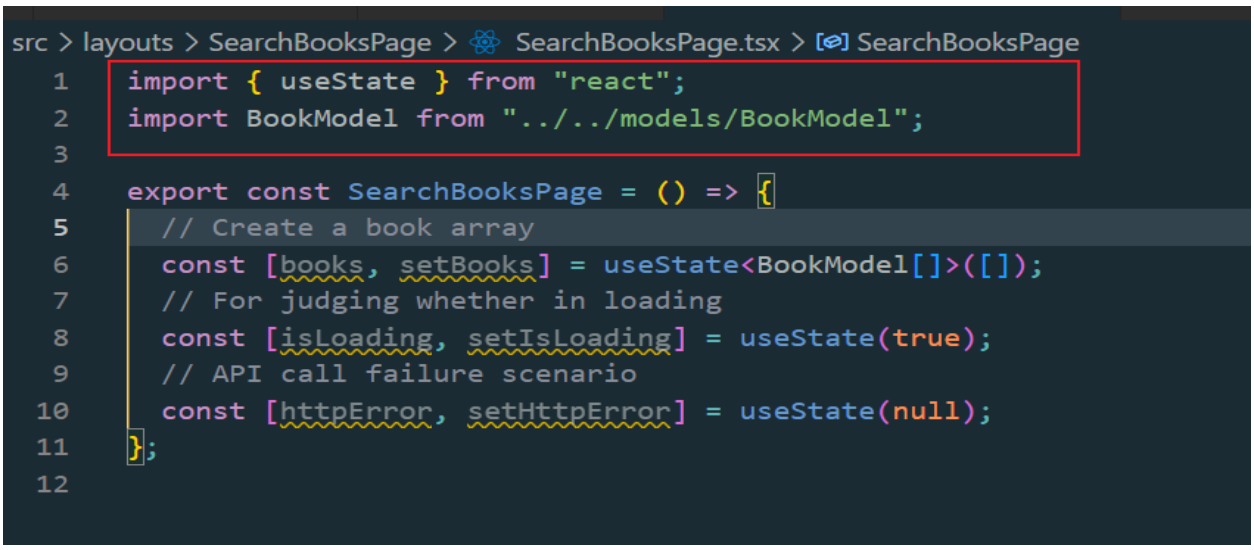
Create a new file “SearchBooksPage.tsx”



create export const SearchBookPage () structure. Create useState for books, isLoading and httpError. Go to Carousel.tsx file, copy these three useStates.



Fix the issues by importing some packages.



Create useEffect in SearchBooksPage.tsx. Similar to the useState, we go to Carousel.tsx, copy the useEffect code. Paste them into SearchBookPage.tsx file.

Fix the issue by importing the useEffect.

fix the url. Change the “size=9” to “size=5”. we just want to each page show 5 books

Continue to copy and paste this part

```
if (isLoading) {
  return <SpinnerLoading />;
}

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

Fix the issue by importing the SpinnerLoding component.

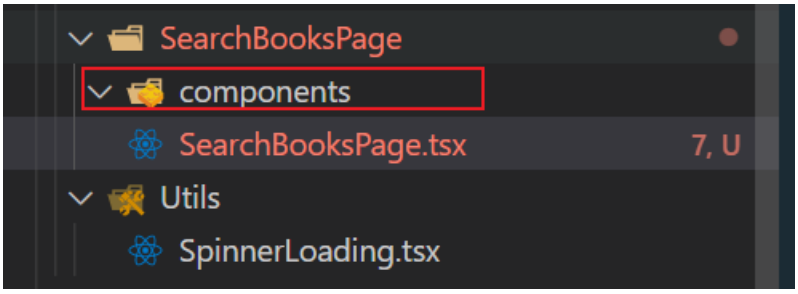
```
import { SpinnerLoading } from "../Utils/SpinnerLoading";
```

2. REACT-SEARCH BOOKS PAGE

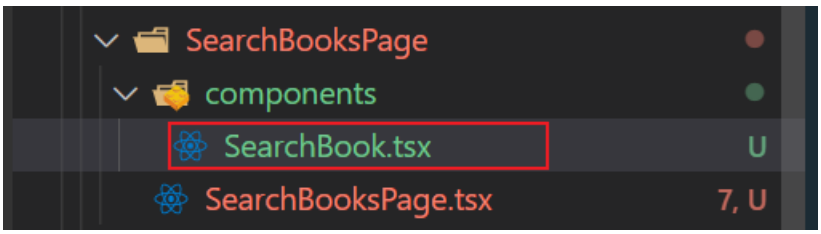
Write the code in the SearchBooksPage.tsx

```tsx
return (
  <div>
    <div className="container">
      <div>
        <div className="row mt-5">
          <div className="col-6">
            <div className="d-flex">
              {/* Search input box */}
              <input
                className="form-control me-2"
                type="search"
                placeholder="Search"
                aria-labelledby="Search"
              />
              <button className="btn btn-outline-success">Search</button>
            </div>
          </div>
          <div className="col-4">
            <div className="dropdown">
              <button
                className="btn btn-secondary dropdown-toggle"
                type="button"
                id="dropdownMenuButton1"
                data-bs-toggle="dropdown"
                aria-expanded="false"
              >
                Category
              </button>
              <ul
                className="dropdown-menu"
                aria-labelledby="dropdownMenuButton1"
              >
                <li>
                  <a className="dropdown-item" href="#">
                    All
                  </a>
                </li>
                <li>
                  <a className="dropdown-item" href="#">
                    Front End
                  </a>
                </li>
                <li>
                  <a className="dropdown-item" href="#">
                    Back End
                  </a>
                </li>
                <li>
                  <a className="dropdown-item" href="#">
                    Data
                  </a>
                </li>
                <li>
                  <a className="dropdown-item" href="#">
                    DevOps
                  </a>
                </li>
              </ul>
            </div>
          </div>
        </div>
        <div className="mt-3">
          <h5>Number of results: (22)</h5>
        </div>
        <p>1 to 5 of 22 items:</p>
        {books.map((book) => (
          <SearchBook book={book} key={book.id} />
        ))}
      </div>
    </div>
  </div>
);
};
```

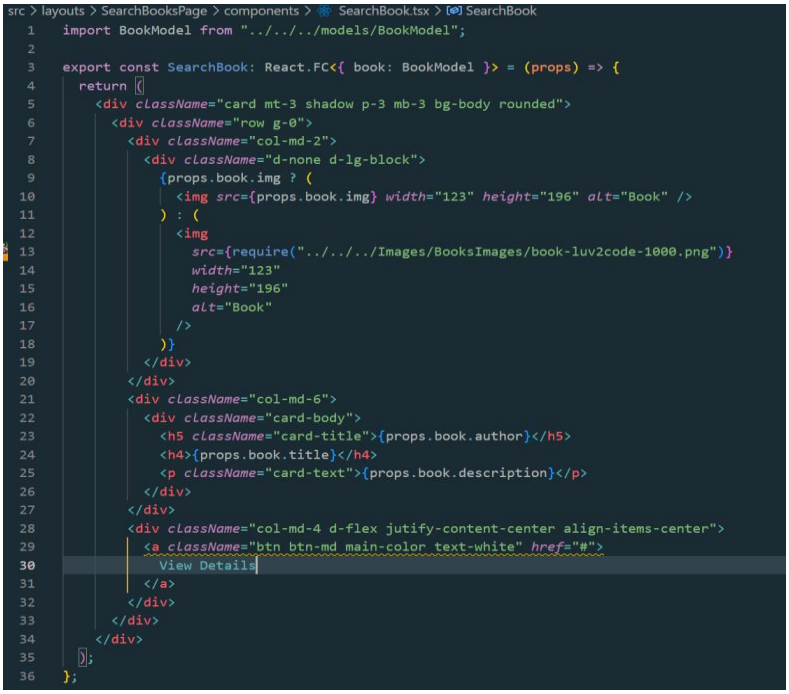
Create a new folder “components” in the SearchBooksPage folder.



Then create a new file in it. “SearchBook.tsx”

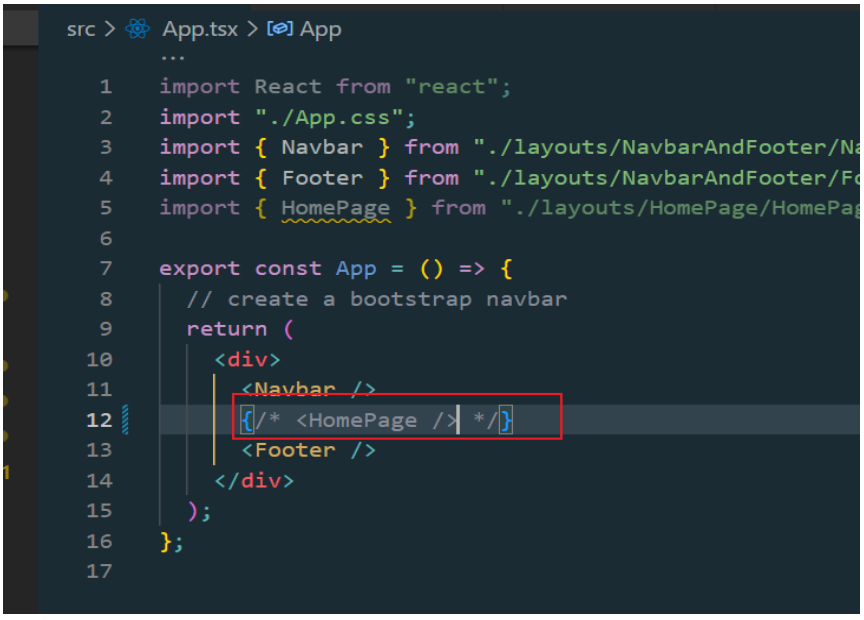


Write the code of SearchBook component.



Go back to the SearchBooksPage.tsx,fix the issue by importing the SearchBook component.

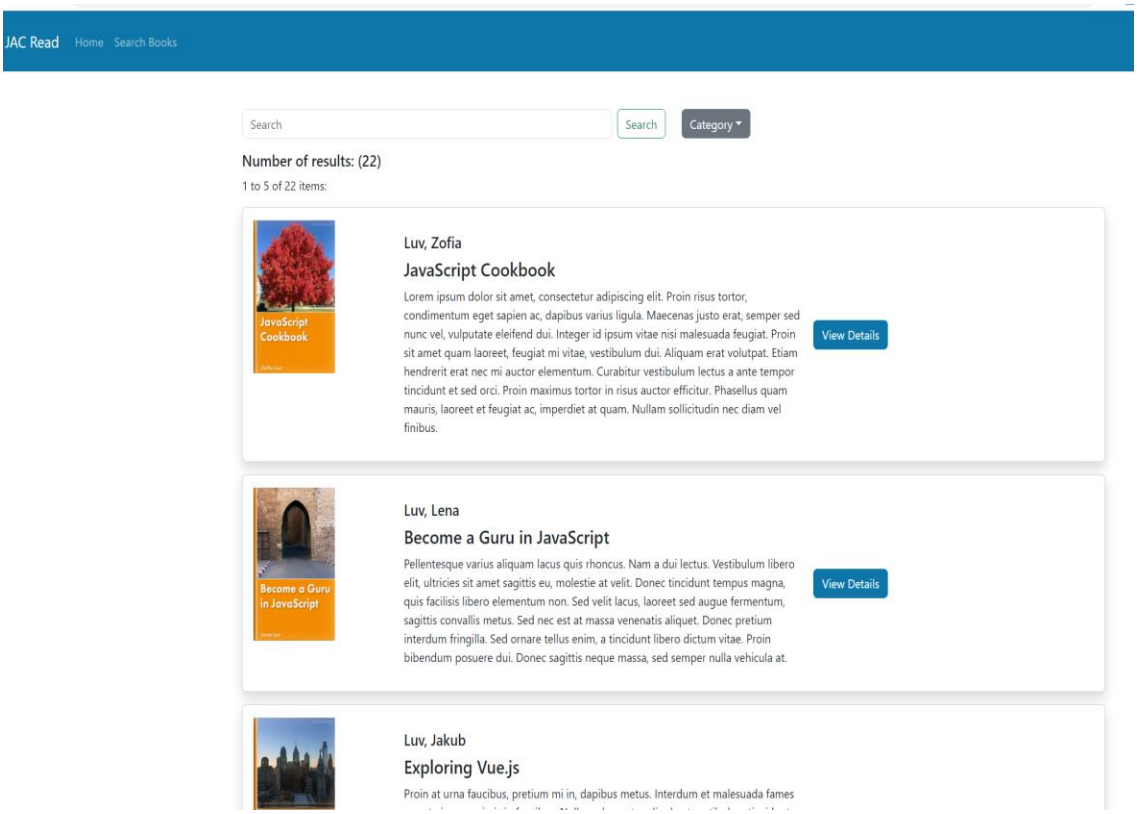
Open the App.tsx, comment out the HomePage component



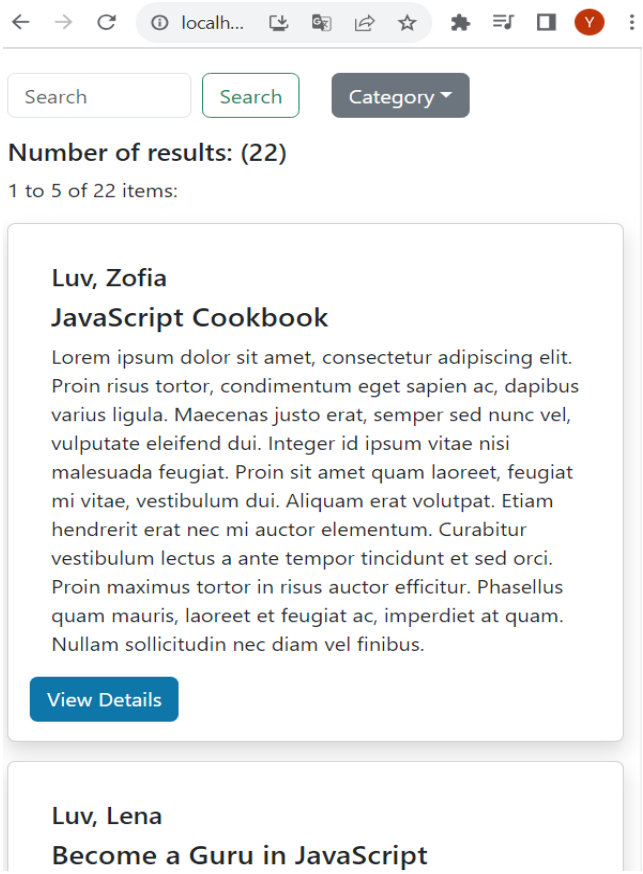
Add the SearchBooksPage component.

```
export const App = () => {
  // create a bootstrap navbar
  return (
    <div>
      <Navbar />
      {/* <HomePage /> */}
      <SearchBooksPage />        You, 1
      <Footer />
    </div>
  );
};
```

Let's run the application.



But for the mobile mode, we noticed that the images can’t be displayed.

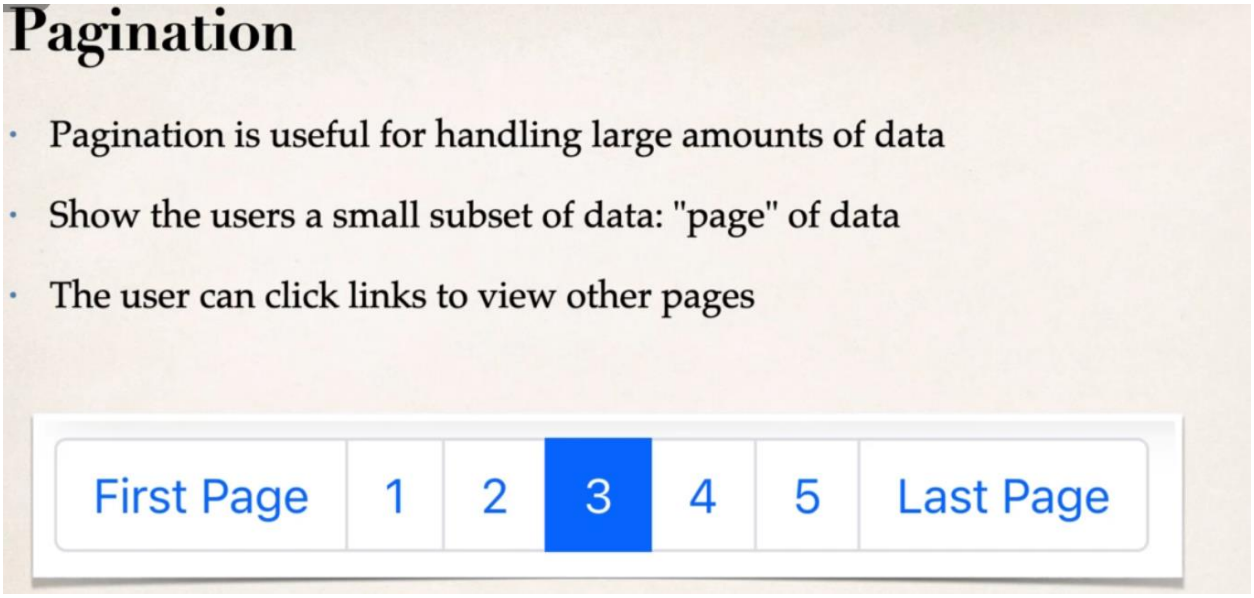


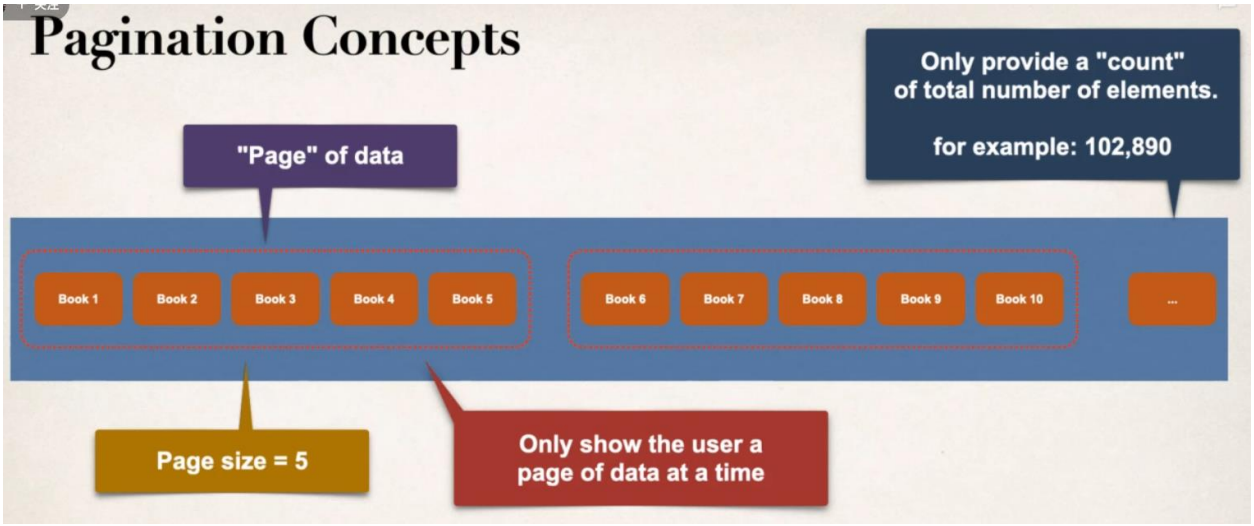
Let me back to the SearchBook.tsx, we need to create a div for mobile mode.

```tsx
export const SearchBook: React.FC<{ book: BookModel }> = (props) => {
  return (
    <div className="card mt-3 shadow p-3 mb-3 bg-body rounded">
      <div className="row g-0">
        <div className="col-md-2">
          <div className="d-none d-lg-block">
            {props.book.img ? (
              <img src={props.book.img} width="123" height="196" alt="Book" />
            ) : (
              <img
                src={require("../../../Images/BooksImages/book-luv2code-1000.png")}
                width="123"
                height="196"
                alt="Book"
              />
            )}
          </div>
          <div className="d-lg-none d-flex justify-content-center align-items-center">
            {props.book.img ? (
              <img src={props.book.img} width="123" height="196" alt="Book" />
            ) : (
              <img
                src={require("../../../Images/BooksImages/book-luv2code-1000.png")}
                width="123"
                height="196"
                alt="Book"
              />
            )}
          </div>
        </div>
        <div className="col-md-6">
          <div className="card-body">
            <h5 className="card-title">{props.book.author}</h5>
            <h4>{props.book.title}</h4>
            <p className="card-text">{props.book.description}</p>
```

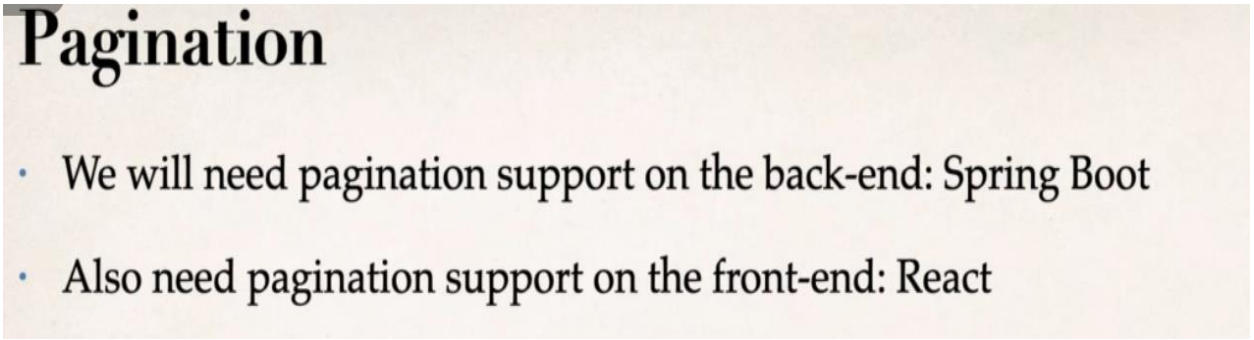
Let’s rerun the application. Now we can see the images on the mobile mode.

3. REACT-PAGINATION





We need pagination support both on the font-end and back-end.



Spring Data REST - Parameters

- By default, Spring Data REST returns: 20 elements
- We can customize this by passing in parameters

Parameter	Purpose
`page`	The page number to access. **0-based** ... defaults to 0.
`size`	The size of the page to return (items per page). Defaults to 20

Spring Data REST - Response Meta Data

- The response meta data has valuable information

```
<< array of books >>
...
"page" : {
  "size" : 10,
  "totalElements" : 200,
  "totalPages" : 20,
  "number" : 0
}
```

Size of this page

Grand total of ALL elements in the database.
But we are NOT returning all of the elements.
Just the "count" for informational purposes only.

Total pages available

Current page number

Pagination with React:

Pagination with React

- Basic example of pagination component
- Create a new Pagination component that we can reuse within our application

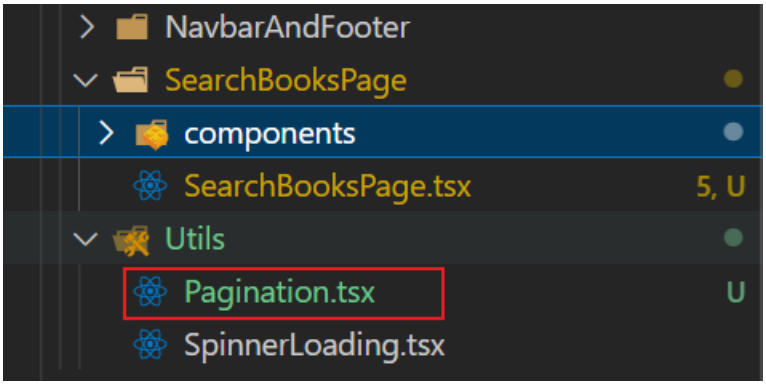
Pagination.tsx

```
export const Pagination: React.FC<{}> = (props) = {

  < **Pagination Logic** />

  return (
   <nav>
    ..... * More Code * .....
   </nav
  );
}
```

SearchBooksPage.tsx

```
<Pagination
   currentPage={currentPage}
   totalPages={totalPages}
   paginate={paginate}
/>
```

Create a new file “Pagination.tsx” in the Utils folder.



create export const Pagination() structure, passing currentPage, totalPages and paginate as its props. and add page number judgement.

src > layouts > Utils > Pagination.tsx > Pagination

```tsx
export const Pagination: React.FC<{
    currentPage: number;
    totalPages: number;
    paginate: any;
}> = (props) => {
    const pageNumbers = [];
    if (props.currentPage === 1) {
      pageNumbers.push(props.currentPage);
      if (props.totalPages >= props.currentPage + 1) {
        pageNumbers.push(props.currentPage + 1);
      }
      if (props.totalPages >= props.currentPage + 2) {
        pageNumbers.push(props.currentPage + 2);
      }
    } else if (props.currentPage > 1) {
      if (props.currentPage >= 3) {
        pageNumbers.push(props.currentPage - 2);
        pageNumbers.push(props.currentPage - 1);
      } else {
        pageNumbers.push(props.currentPage - 1);
      }
      pageNumbers.push(props.currentPage);
      if (props.totalPages >= props.currentPage + 1) {
        pageNumbers.push(props.currentPage + 1);
      }
      if (props.totalPages >= props.currentPage + 2) {
        pageNumbers.push(props.currentPage + 2);
      }
    }
```

Write the code in return part.

```
  return (
    <nav aria-label="...">
      <ul className="pagination">
        <li className="page-item" onClick={() => props.paginate(1)}>
          <button className="page-link">First Page</button>
        </li>
        {pageNumbers.map((number) => (
          <li
            key={number}
            onClick={() => props.paginate(number)}
            className={
              "page-item" + (props.currentPage === number ? "active" : "")
            }
          >
            <button className="page-link">{number}</button>
          </li>
        ))}
        <li
          className="page-item"
          onClick={() => props.paginate(props.totalPages)}
        >
          <button className="page-link">Last Page</button>
        </li>
      </ul>
    </nav>
  );
};
```

4. REACT-ADD PAGINATION TO SEARCH BOOKS PAGE COMPONENT

Go to SearchBooksPage.tsx, add useState of pagination.

```tsx
export const SearchBooksPage = () => {
  // Create a book array
  const [books, setBooks] = useState<BookModel[]>([]);
  // For judging whether in loading
  const [isLoading, setIsLoading] = useState(true);
  // API call failure scenario
  const [httpError, setHttpError] = useState(null);

  // pagination array
  const [currentPage, setCurrentPage] = useState(1);
  // each page show 5 books
  const [booksPerPage] = useState(5);
  // total amounts of books
  const [totalAmountOfBooks, setTotalAmountOfBooks] = useState(0);
  // total pages
  const [totalPages, setTotalPages] = useState(0);
```

In the url, change the page number and books per page number to the dynamic value.

```tsx
const fetchBooks = async () => {
  const baseUrl: string = "http://localhost:8080/api/books";
  const url: string = `${baseUrl}?page=${currentPage - 1}&size=${booksPerPage}`;
```

And add these two statements.

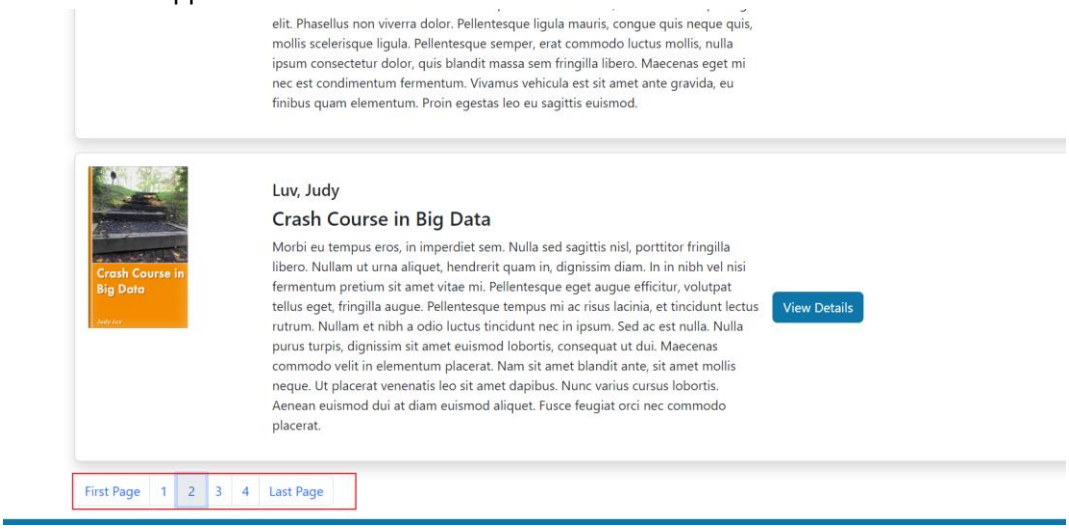
```tsx
useEffect(() => {
  const fetchBooks = async () => {
    const baseUrl: string = "http://localhost:8080/api/books";
    const url: string = `${baseUrl}?page=${
      currentPage - 1
    }&size=${booksPerPage}`;
    // fetching the url data
    const response = await fetch(url);
    //failure scenario
    if (!response.ok) {
      throw new Error("Something went wrong!");
    }
    // transfer the url data into json;
    const responseJson = await response.json();
    // get the data which is the object of embedded books
    const responseData = responseJson._embedded.books;

    setTotalAmountOfBooks(responseJson.page.totalElements);
    setTotalPages(responseJson.page.totalPages1);
```

And add these statement.

```
if (isLoading) {
  return <SpinnerLoading />;
}

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

const indexOfLastBook: number = currentPage * booksPerPage;
const indexOfFirstBook: number = indexOfLastBook - booksPerPage;
let lastItem =
  booksPerPage * currentPage <= totalAmountOfBooks
    ? booksPerPage * currentPage
    : totalAmountOfBooks;

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

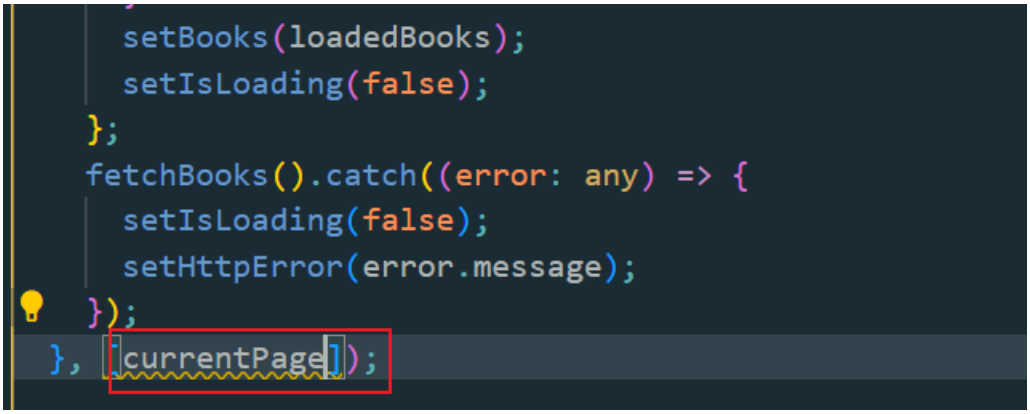
```
      <div className="mt-3">
        <h5>Number of results: (22)</h5>
      </div>
      <p>1 to 5 of 22 items:</p>
      {books.map((book) => (
        <SearchBook book={book} key={book.id} />
      ))}
      {totalPages > 1 && (
        <Pagination
          currentPage={currentPage}
          totalPages={totalPages}
          paginate={paginate}
        />
      )}
    </div>
```

Let’s run the application.

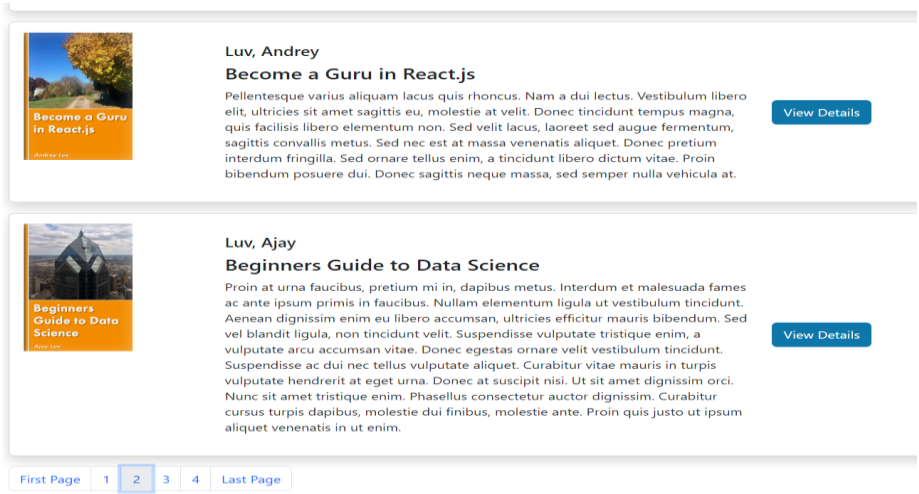


We can see the pagination, but when we click the different page number, the book won’t change.

We go back to the SeachBooksPage.tsx, add currentPage useSate in the end of our array of state changes that are recall the use effect.



Then we refresh our page, if we click the different pages, we can find the books changed.



```
    fetchBooks().catch((error: any) => {
      setIsLoading(false);
      setHttpError(error.message);
    });
    window.scrollTo(0, 0);
  }, [currentPage]);
```

Add this line of code, so that when we change the pages, it will always go to the top. And then change the static number to dynamic.

```
      <div className="mt-3">
        <h5>Number of results: ({totalAmountOfBooks})</h5>
      </div>
      <p>
        {indexOfFirstBook + 1} to {lastItem} of {totalAmountOfBooks} items:
      </p>
      {books.map((book) => (
        <SearchBook book={book} key={book.id} />
```

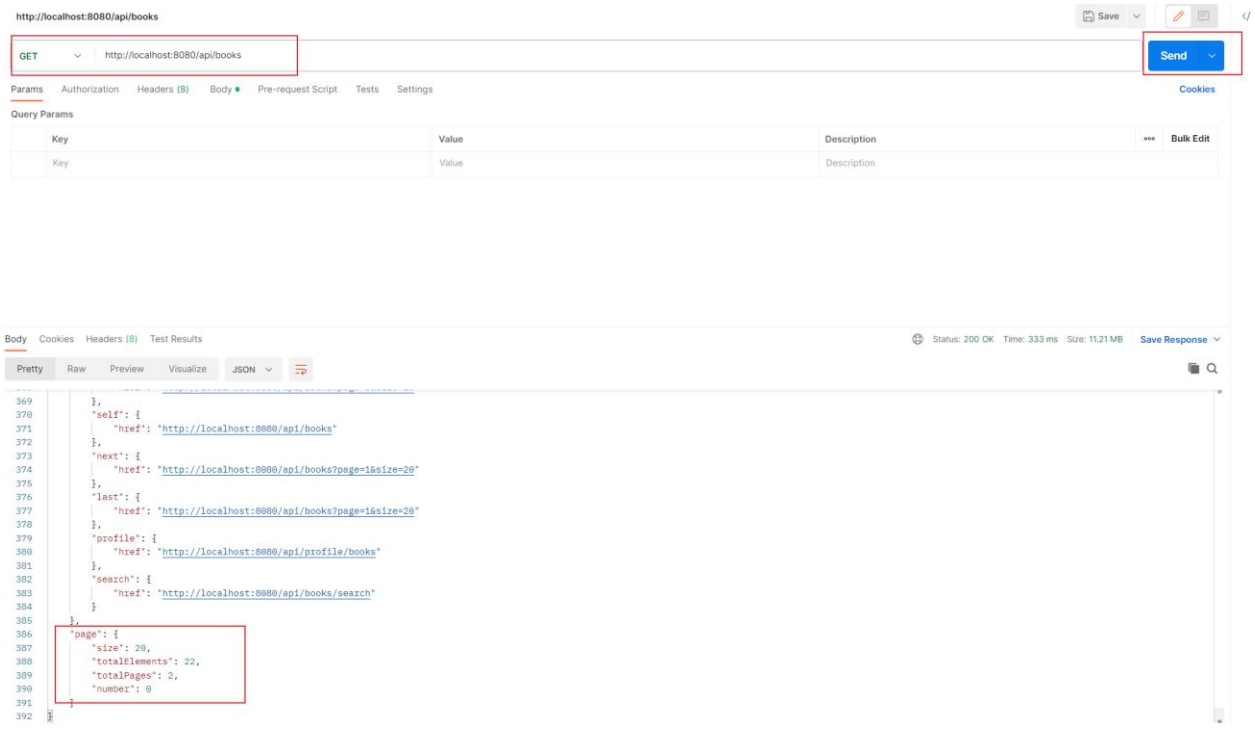
Then we refresh the page, when we change the page number, it will show the information dynamically.

5. SPRINGBOOT-FIND BY TITLE

Let's go to BookRepository class, create findByTitleContaining interface function. Then import book class.

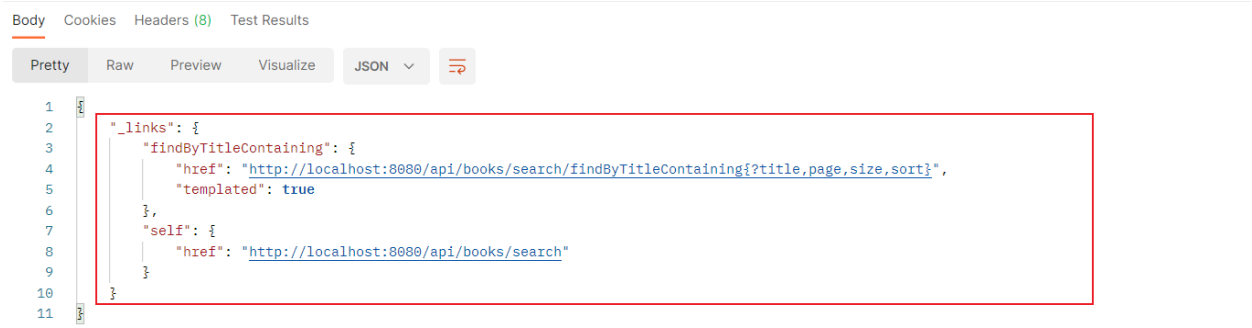
```java
import com.fsd07.springbootlibrary.entity.Book;
import org.springframework.data.domain.Page;
import org.springframework.data.domain.Pageable;
import org.springframework.data.jpa.repository.JpaRepository;
import org.springframework.web.bind.annotation.RequestParam;

/**
 * @Author: Yeming Hu
 * @Date: 9/3/2023
 * @Description: com.fsd07.springbootlibrary.dao
 * @Version: 1.0
 */
public interface BookRepository extends JpaRepository<Book, Long> {
    Page<Book> findByTitleContaining(@RequestParam("title") String title, Pageable pageable);
}
```

Rerun the application, then we open the postman.

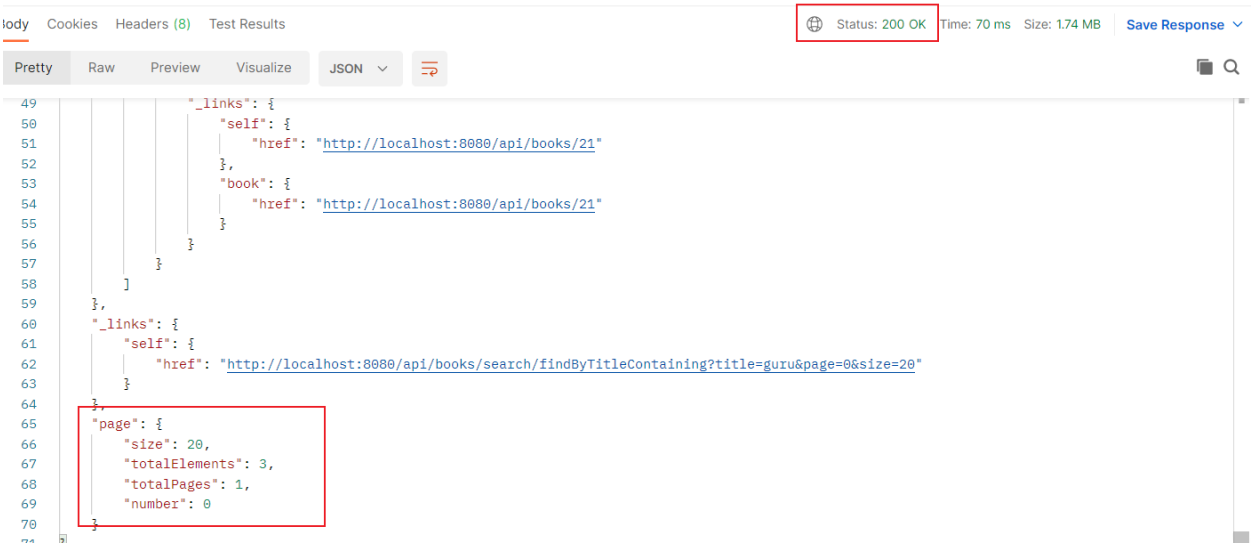


If we run the http://localhost:8080/api/books, use get method, send the request, we can get the information of 22 books.

If we input http://localhost:8080/api/bo oks/search, choose "GET", we will get:

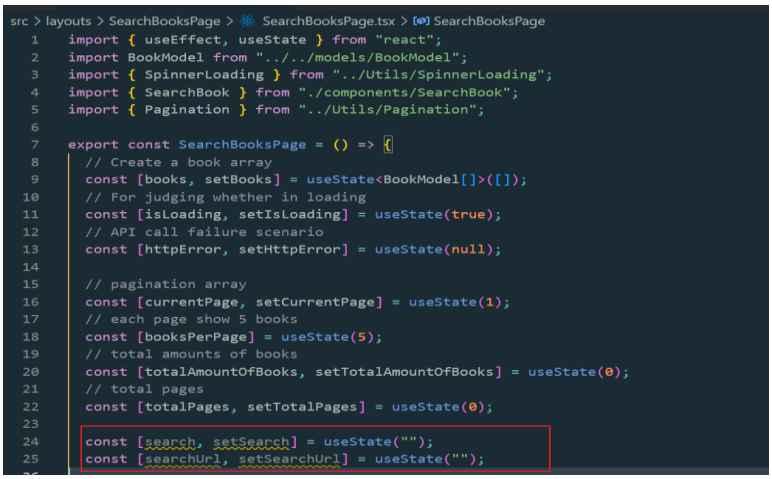


If we add “findByTitleContaining?title=guru” to our search url string, we will get a “200 ok response”. All the books whose title contain “guru” returned. Totally we have found 3 books. REACT- FIND BY TITLE

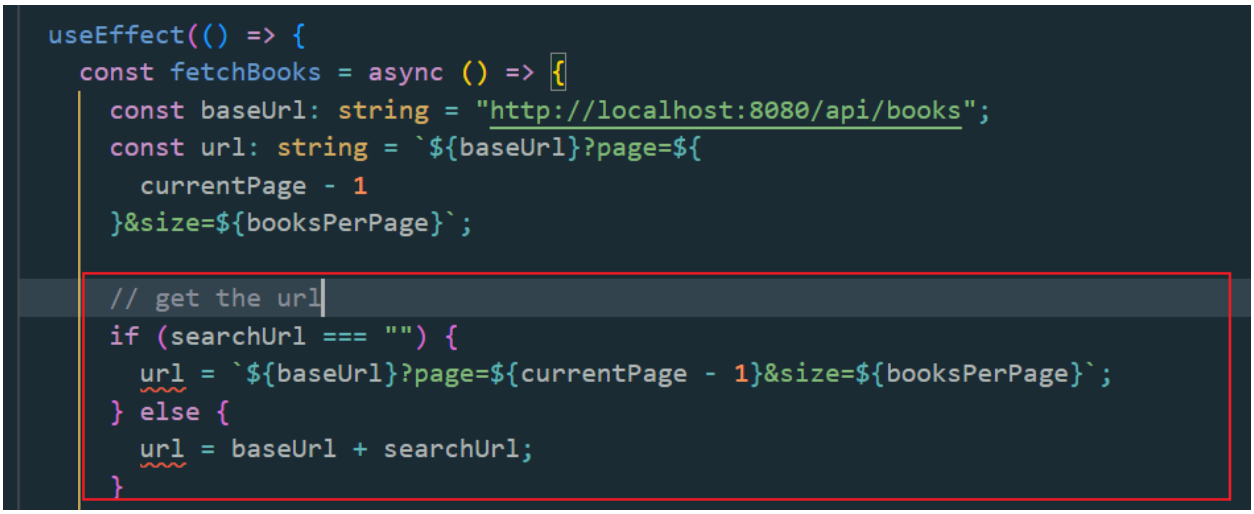


6. REACT- FIND BY TITLE

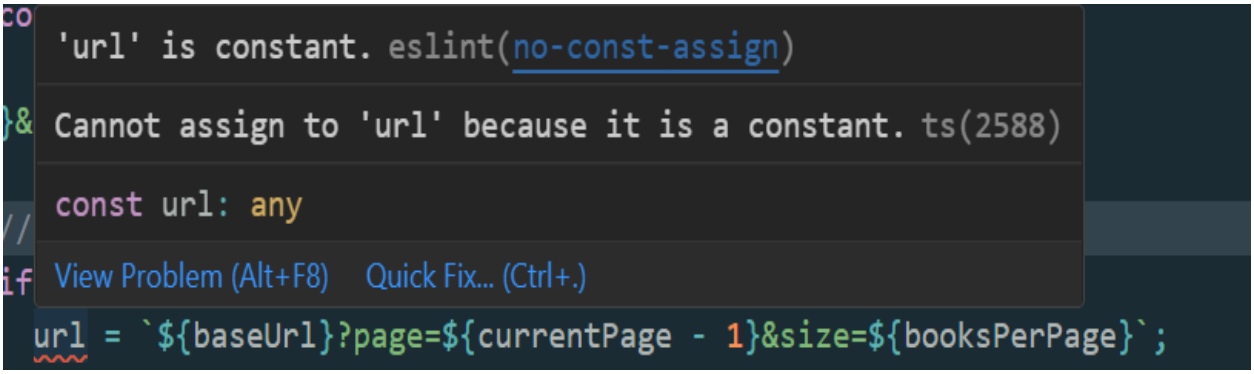
Go to SearchBooksPage.tsx. Create useState for search.



Get the url. If there is a search url string , then the final url should be the combination of base url and the search url.



We noticed that we get an error



We change the “const” into “let” , and assign a empty string as its value.

```
useEffect(() => {
  const fetchBooks = async () => {
    const baseUrl: string = "http://localhost:8080/api/books";
    let url: string = "";

    // get the url
    if (searchUrl === "") {
      url = `${baseUrl}?page=${currentPage - 1}&size=${booksPerPage}`;
    } else {
      url = baseUrl + searchUrl;
    }
```

add searchUrl useState in the end of our array of state changes that are recall the use effect.

```
    fetchBooks().catch((error: any) => {
      setIsLoading(false);
      setHttpError(error.message);
    });
    window.scrollTo(0, 0);
  }, [currentPage, searchUrl]);
```

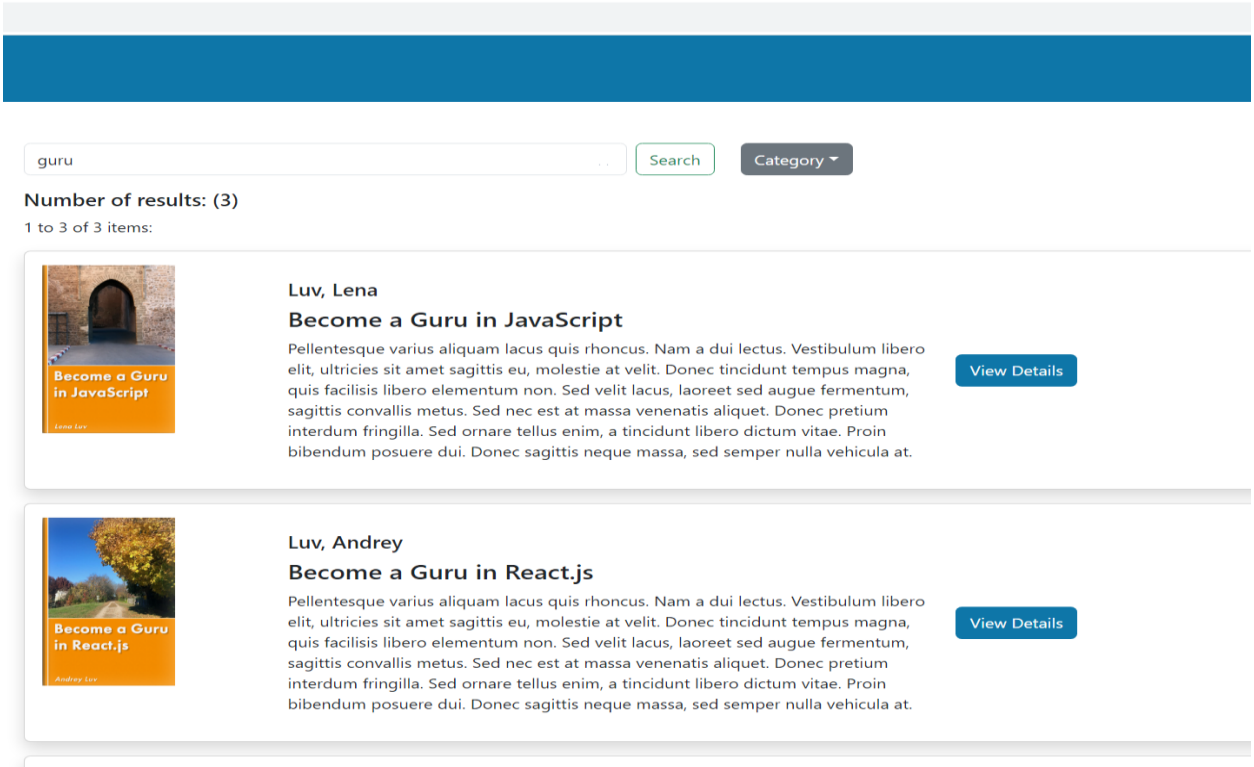
create a function which handle the searchUrl

```
82  if (httpError) {
83    return (
84      <div className="container m-5">
85        <p>{httpError}</p>
86      </div>
87    );
88  }
89
90  const searchHandleChange = () => {
91    if (search === "") {
92      setSearchUrl("");
93    } else {
94      setSearchUrl(
95        `/search/findByTitleContaining?title=${search}&page=0&size=${booksPerPage}`
96      );
97    }
98  };
```

Add a onChange listener to the value of input, so that we can get the search value, and add a onClick listener to search button, so each time the button is clicked, it will execute the search url handle funtion.

```
            {/* Search input box */}
            <input
              className="form-control me-2"
              type="search"
              placeholder="Search"
              aria-labelledby="Search"
              onChange={(e) => setSearch(e.target.value)}
            />
            <button
              className="btn btn-outline-success"
              onClick={() => searchHandleChange()}
            >
              Search
            </button>
          </div>
        </div>
```

Let's run the application. If we type guru in the search box, then we can get the search result.



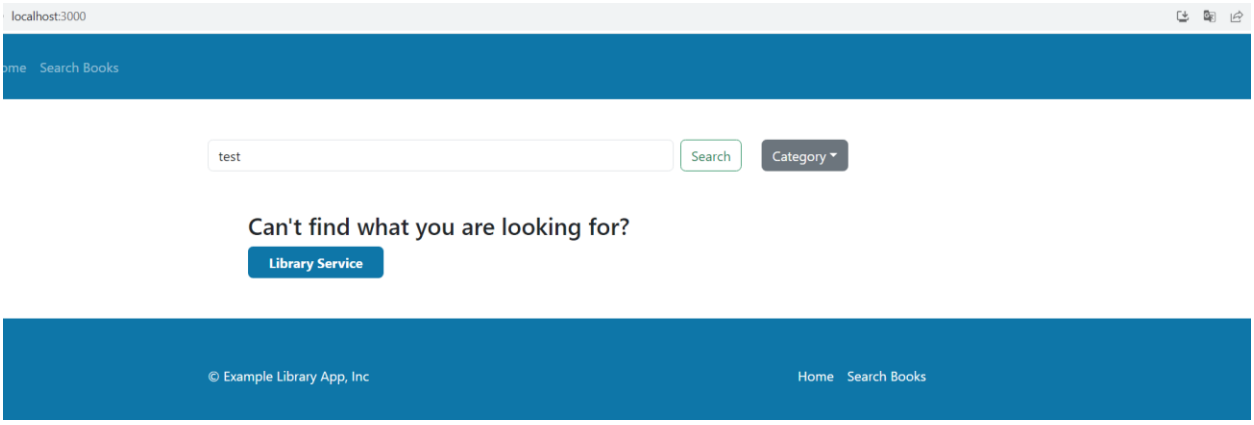
7. REACT-ALTERNATIVE TEXT FOR 0 BOOK

If the search result is 0 book found, it will show some message and a link to library service.

src > layouts > SearchBooksPage > SearchBooksPage.tsx > SearchBooksPage

```
{totalAmountOfBooks > 0 ? (
  <>
    <div className="mt-3">
      <h5>Number of results: ({totalAmountOfBooks})</h5>
    </div>
    <p>
      {indexOfFirstBook + 1} to {lastItem} of {totalAmountOfBooks}{" "}
      items:
    </p>
    {books.map((book) => (
      <SearchBook book={book} key={book.id} />
    ))}
  </>
) : (
  <div className="m-5">
    <h3>Can't find what you are looking for?</h3>
    <a
      type="button"
      className="btn main-color btn-md px-4 me-md-2 fw-bold text-white"
      href="#"
    >
      Library Service
    </a>
  </div>
)}
{totalPages > 1 && (
  <Pagination
    currentPage={currentPage}
    totalPages={totalPages}
    paginate={paginate}
  />
)}
</div>
</div>
</div>
);
};
```

We rerun the application, when we input “test” in the box, we will get 0 book returned, and get this page

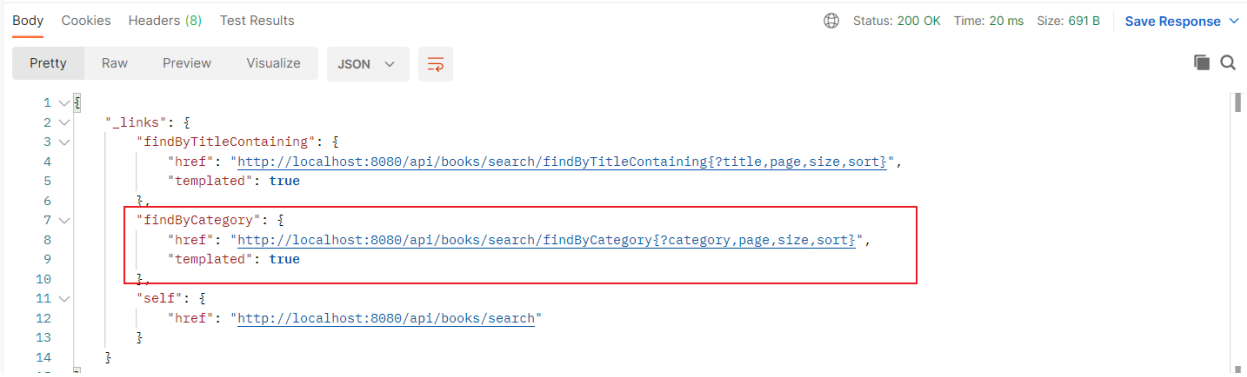


8. SPRINGBOOT-FIND BY CATEGORY

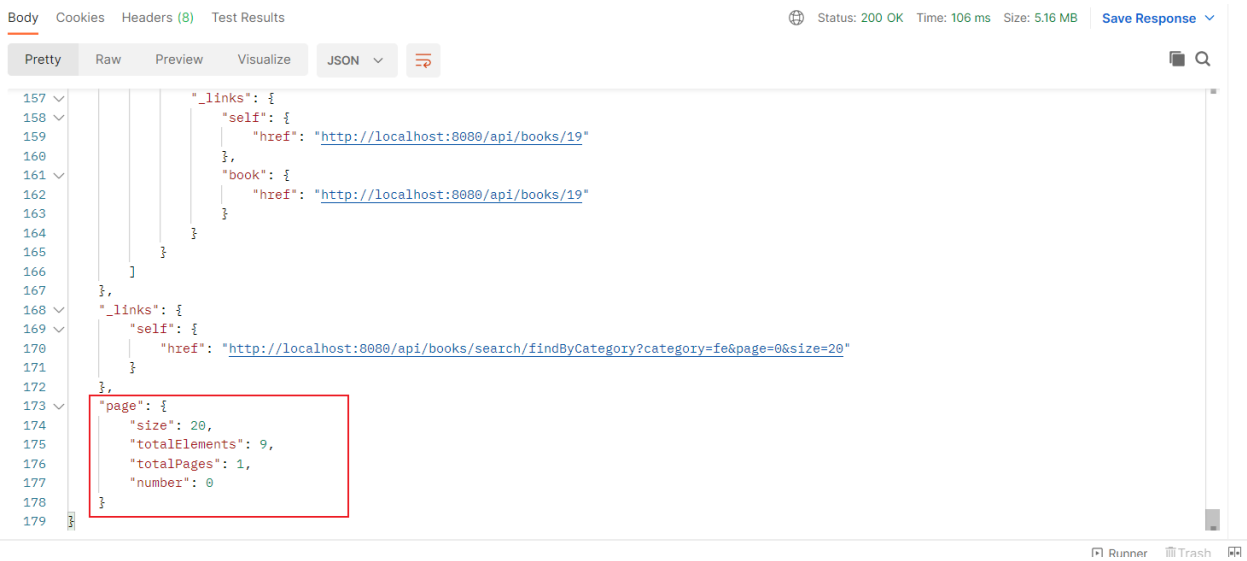
Go to BookRepository class, create findByTitleCategory interface function.

```java
public interface BookRepository extends JpaRepository<Book, Long> {
    Page<Book> findByTitleContaining(@RequestParam("title") String title, Pageable pageable);

    Page<Book> findByCategory(@RequestParam("category") String category, Pageable pageable);
}
```

Open Postman, input http://localhost:8080/api/books/search, choose “GET”, we will get:



If we add “findByCategory?category=fe” to our search url string, we will get a “200 ok response”. All the front end books returned. We can get 9 books.



9. REACT-FIND BY CATEGORY

Open SearchBooksPage.tsx, create useState for category selection.

src > layouts > SearchBooksPage > SearchBooksPage.tsx > SearchBooksPage

```tsx
import { useEffect, useState } from "react";
import BookModel from "../../models/BookModel";
import { SpinnerLoading } from "../Utils/SpinnerLoading";
import { SearchBook } from "./components/SearchBook";
import { Pagination } from "../Utils/Pagination";

export const SearchBooksPage = () => {
  // Create a book array
  const [books, setBooks] = useState<BookModel[]>([]);
  // For judging whether in loading
  const [isLoading, setIsLoading] = useState(true);
  // API call failure scenario
  const [httpError, setHttpError] = useState(null);

  // pagination array
  const [currentPage, setCurrentPage] = useState(1);
  // each page show 5 books
  const [booksPerPage] = useState(5);
  // total amounts of books
  const [totalAmountOfBooks, setTotalAmountOfBooks] = useState(0);
  // total pages
  const [totalPages, setTotalPages] = useState(0);

  // search by title, create two array for search
  const [search, setSearch] = useState("");
  const [searchUrl, setSearchUrl] = useState("");

  // create useState for book category
  const [categorySelection, setCategorySelection] = useState("Book Category");
```

create categoryField function to judge which category has been chose.

```tsx
const searchHandleChange = () => {
  if (search === "") {
    setSearchUrl("");
  } else {
    setSearchUrl(
      `/search/findByTitleContaining?title=${search}&page=0&size=${booksPerPage}`
    );
  }
};

const categoryField = (value: string) => {
  if (
    value.toLowerCase() === "fe" ||
    value.toLowerCase() === "be" ||
    value.toLowerCase() === "data" ||
    value.toLowerCase() === "devops"
  ) {
    setCategorySelection(value);
    setSearchUrl(
      `/search/findByCategory?category=${value}&page=0&size=${booksPerPage}`
    );
  } else {
    setCategorySelection("All");
    setSearchUrl(`?page=0&size=${booksPerPage}`);
  }
};
```

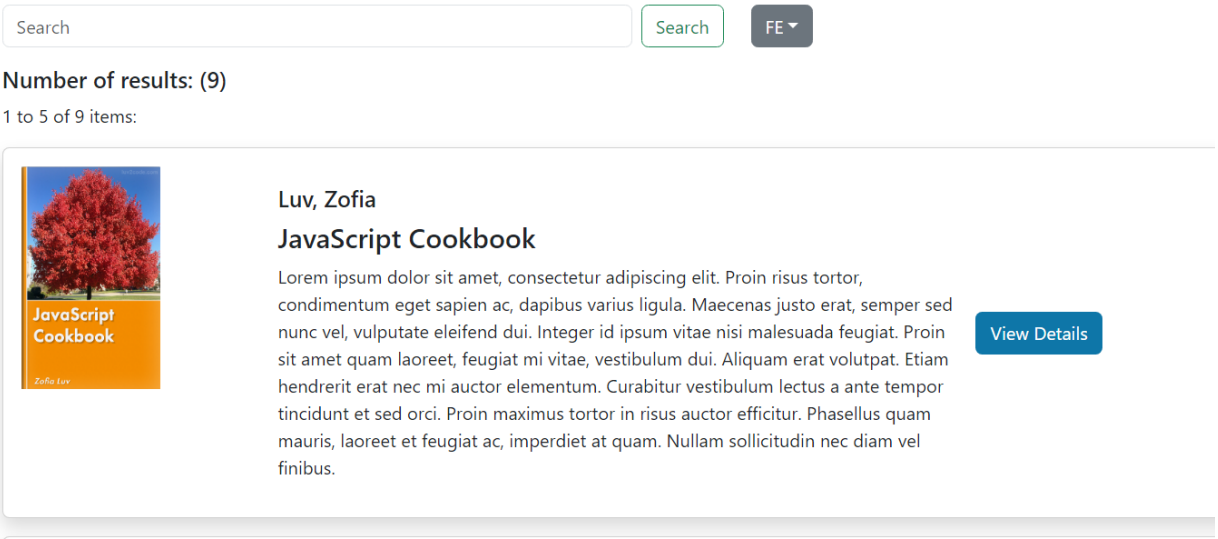
change the static word “catogory” into dynamical value.

```
152            <div className="col-4">
153              <div className="dropdown">
154                <button
155                  className="btn btn-secondary dropdown-toggle"
156                  type="button"
157                  id="dropdownMenuButton1"
158                  data-bs-toggle="dropdown"
159                  aria-expanded="false"
160                >
161                  {categorySelection}
162                </button>
163                <ul
164                  className="dropdown-menu"
165                  aria-labelledby="dropdownMenuButton1"
166                >
```

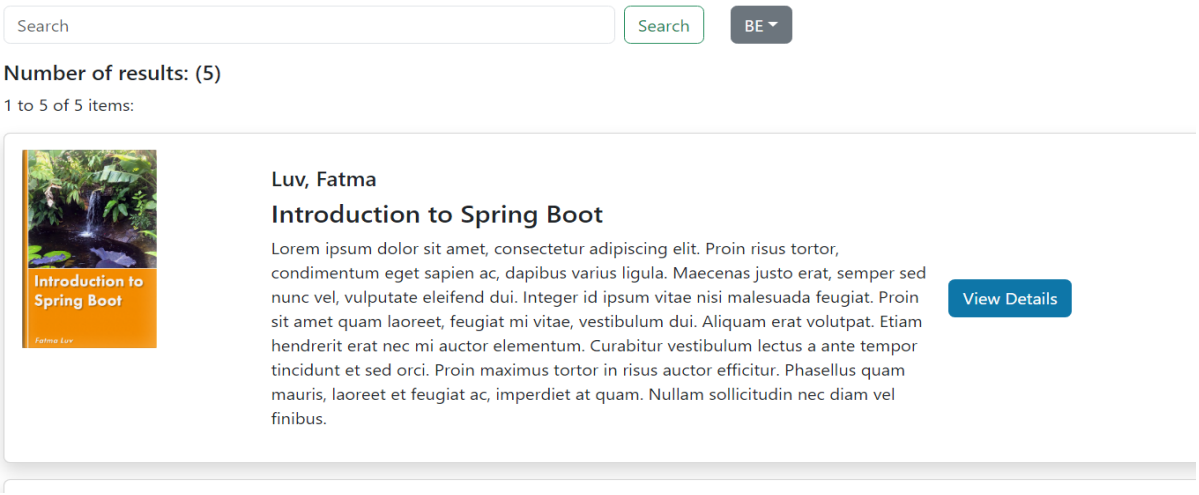
add onClick listener to each catogory dropdown item.

```
<li onClick={() => categoryField("All")}>
  <a className="dropdown-item" href="#">
    All
  </a>
</li>
<li onClick={() => categoryField("FE")}>
  <a className="dropdown-item" href="#">
    Front End
  </a>
</li>
<li onClick={() => categoryField("BE")}>
  <a className="dropdown-item" href="#">
    Back End
  </a>
</li>
<li onClick={() => categoryField("Data")}>
  <a className="dropdown-item" href="#">
    Data
  </a>
</li>
<li onClick={() => categoryField("Devops")}>
  <a className="dropdown-item" href="#">
    DevOps
  </a>
```

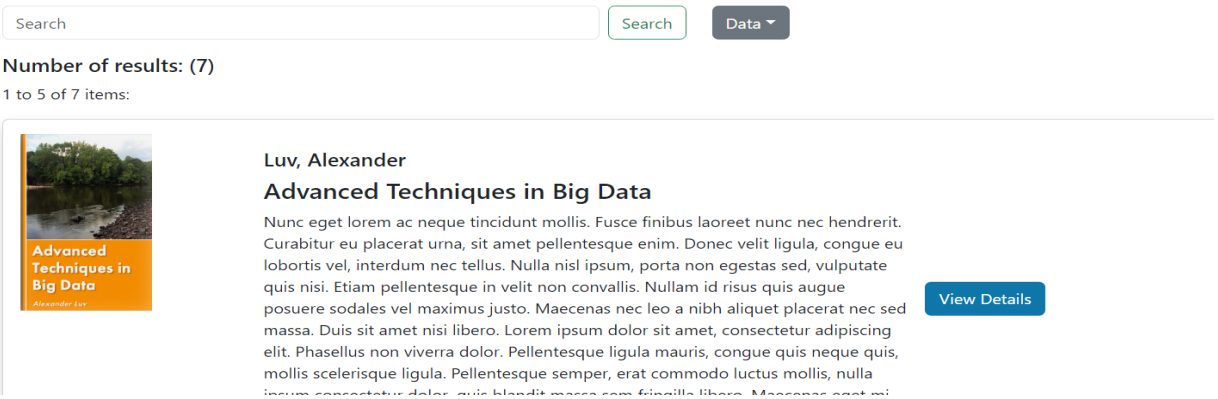
Let’s run the application, If we select “fe” in dropdown menu, it will show us all 9 frontend books in our database.



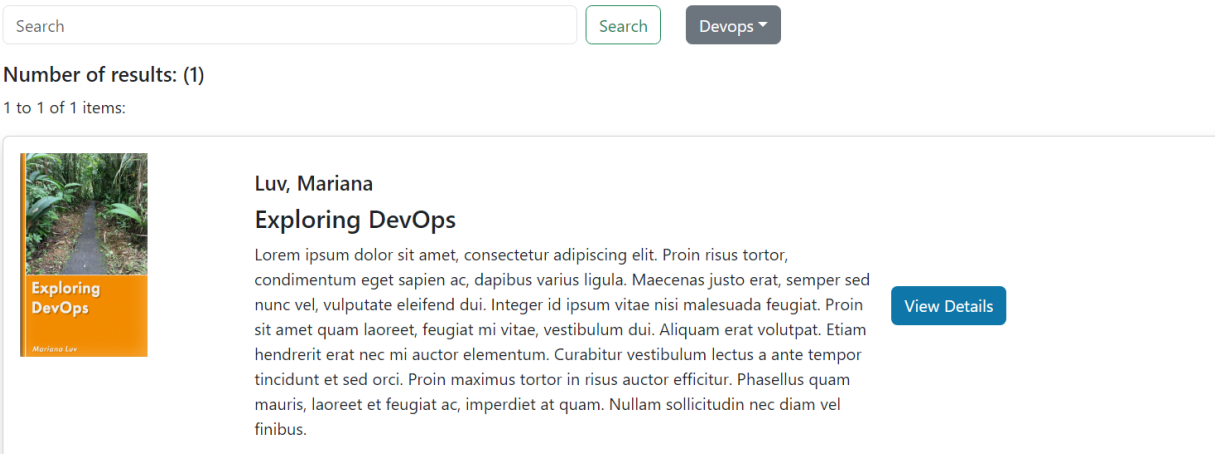
If we select “be” in dropdown menu, it will show us all 5 backend books in our database.



If we select “data” in dropdown menu, it will show us all 7 data related books in our database.



If we select “devops” in dropdown menu, it will show us 1 related books in our database.



If we select “All” in dropdown menu, it will show us all 22 books in our database.