EE 461L Design Report

Team : Breakfast Book Club
Project : Booklopedia
Canvas Group : morning-2
Github Repository : https://github.com/VishruthiR/EE461L_IDB
Website Link: http://booklopedia.appspot.com/

Information Hiding :

Information hiding is implemented through the separation of our front and back end code with a middleware section. MongoDB is used to store our thousands of model instances, and queries are made from the front end, handled by middleware, and passed on to the backend where the requested data is pulled and returned through the same steps in reverse. In this example, the database is the secret. We anticipated that the formatting of our database might change as we add new features and the middle end encapsulates these changes in its own module.
The middleware acts as a buffer between the front and back end, as the front end does not need any knowledge of where it is receiving its data. Another database could be substituted in place of MongoDB, and no changes would be needed on the front end side. The following pseudocode demonstrates this process:

```
React component action
send data request to middleware
query database
if data exists
        data = getData
else data = empty
return data to middleware
send data back to client
update React component to display received data
```

As shown above, the front end never directly interacts with the backend, thanks to the middleware.

Our original and current choice of database is MongoDB, but due to query size constraints and pricing, future revisions may require a different database. This will be a simple fix since the front end will not be affected by any database changes.

Information hiding is also inherent to React. Data always flows down from parent to child in React components, and state, one of the primary forms of model data in React, is localized to its respective component. That is, neither a component's parent nor its children have any knowledge of whether it is stateful or stateless. Thus, state acts as a secret to store the data that the component will keep track of or display.

The modular design of React components also allows us to easily redesign our web pages. Features such as the sidebar, carousels, buttons, and search results along with their behaviors are encapsulated in their respective component files. When design changes need to be made, components can simply be replaced or removed without affecting other parts of code.

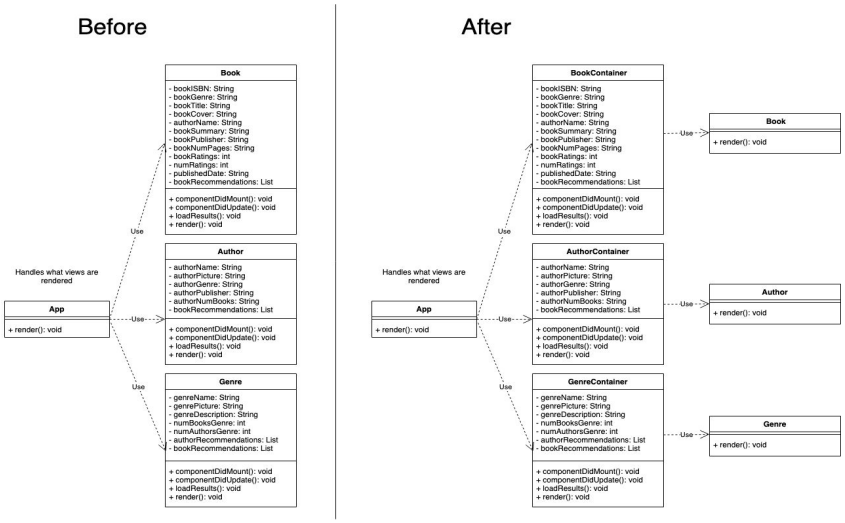
One possible disadvantage to our design is the dependence on the specific data we're receiving throughout the front end and middleware. Each of these portions of the project expect our data to be of a specific format, so if we decide to change our data representations, we'll have to make significant changes to both the front end, middleware, and, of course, the back end in replacing the existing data.

Design Patterns :

Instance Pages

- We implemented the Decorator design pattern, a pattern that allows us to attach new behaviors to components by wrapping them inside another component which will contain that behavior. In the terminology of React, this is commonly referred to as higher-order components, or HOC.
- An issue we had with our instance page components (Book, Author, Genre) was that data fetching and rendering were coupled within the component. We did not have the ability to reuse any instance page component anywhere else because it was tightly coupled with exactly how the data was received. If we wanted to later use additional databases then it would be difficult to add this functionality while retaining the use of the same instage page component. What the Decorator pattern allows us to do is separate out the concerns of rendering and have the fetch be an additional "feature" added to the rendering. Different databases could require different methods of fetching but the core rendering should be the same. In this way, we wrapped our instance page components in another fetching component and returned the correctly rendered component by passing it the appropriate data it requires based on how the higher-order components decided to fetch.
- Advantages:
    - We can reuse our components dedicated for rendering. For example, if we wanted to fetch in a different manner elsewhere in our site but reuse this UI of our book instance page then we can do so by simply creating a new wrapper component around the base Book UI component.
- Disadvantages:
    - Requires maintenance of more classes. If the base class is changed so that it needs more input parameters, then all of the wrapper classes need to be adjusted to provide this. This can make future refactors tedious if the base class has to be changed.

UML Diagrams:



BookContainer & Book Code Snippets:
Before:

```
import React from "react";
import Recommendations from "./Recommendations";
import Description from "./Description";
import BookHeader from "./BookHeader";

class Book extends React.Component {
  constructor(props) {…
  }
  componentDidMount() {…
  }
  componentDidUpdate() {…
  }

  loadResults() {…
  }
  render() {
    return (
      <React.Fragment>
        <BookHeader title={this.state.bookTitle} image={this.state.bookCover} author={this.state.authorName}
            genre={this.state.bookGenre} publisher={this.state.bookPublisher}
            numPages={this.state.bookNumPages} rating={this.state.bookRating}
            numRatings={this.state.numRatings} publishedDate={this.state.publishedDate}
            />
        <br />
        <Description
          typeOfDescription="Book Description"
          description={this.state.bookSummary}
        />
        <Recommendations
          recommendations={this.state.bookRecommendations}
          typeOfRecommendation={"Recommended Books"}
        />
        {/*<Reviews />*/}
      </React.Fragment>
    );
  }
}
```

After:

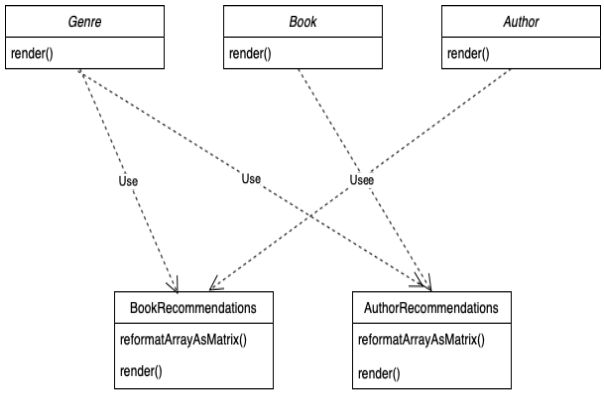
```
import React from "react";
import Book from "./Book";

class BookContainer extends React.Component {
    constructor(props) {…
    }

    componentDidMount() {…
    }

    componentDidUpdate() {…
    }

    loadResults() {…
    }

    render() {
        return (
            <Book bookTitle={this.state.bookTitle}
                bookCover={this.state.bookCover}
                authorName={this.state.authorName}
                bookGenre={this.state.bookGenre}
                bookPublisher={this.state.bookPublisher}
                bookNumPages={this.state.bookNumPages}
                bookRating={this.state.bookRating}
                numRatings={this.state.numRatings}
                bookSummary={this.state.bookSummary}
                bookRecommendations={this.state.bookRecommendations}
            />
        );
    }
}

export default BookContainer;
```

```
import React from "react";
import Description from "./Description";
import BookHeader from "./BookHeader";
import RecommendationCarousel from "./RecommendationCarousel";

class Book extends React.Component {
  render() {
    return (
      <React.Fragment>
        <BookHeader title={this.props.bookTitle}
          image={this.props.bookCover}
          author={this.props.authorName}
          genre={this.props.bookGenre}
          publisher={this.props.bookPublisher}
          numPages={this.props.bookNumPages}
          rating={this.props.bookRating}
          numRatings={this.props.num any js}
          publishedDate={this.props.publishedDate}
        />
        <br />
        <Description
          typeOfDescription="Book Description"
          description={this.props.bookSummary}
        />
        <RecommendationCarousel
          recommendations={this.props.bookRecommendations}
          typeOfRecommendation={"Recommended Books"}
          type={"Book"}
        />
      </React.Fragment>
    );
  }
}

export default Book;
```

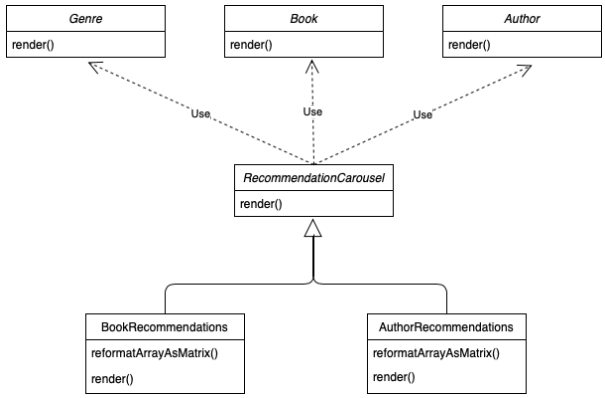
Recommendation Carousels

- *Original* 180 lines (total), 2 classes → *Improved* 200 lines (total), 1 superclass 2 subclasses.
- Implemented Factory Method Design Pattern.
- This pattern was relevant to our "Recommendation Carousel" component, because there were many different types of carousels that could be implemented when it came to what model page. But the base carousel item should have some general properties that are shared between the many variations. This is why implementing a super class of "Recommended Carousel" and creating subclasses "AuthorRecommendations" and "BookRecommendations" allowed us to avoid a lot of conditional statements that were put in place to check what kind of restrictions/changes needed to be made to each carousel. This fits the Factory Method design pattern as we can allow the factory to create the appropriate recommendation carousel based on the context of the current webpage.
    - This pattern was also applied to the Header components (AuthorHeader, BookHeader, GenreHeader), because it was in a similar situation where all headers were somewhat similar but with some difference between each of them.
- Advantages
    - For the future, if another type of Recommendation Carousel were to be created - it would be easily integrated into our current design. It is useful to have a base factor class of recommendation carousels and then further adding in specific attributes/restrictions for the new list of recommendations. This also makes it easier to alter, since adding a new type would just be an independent subclass of "Recommendation Carousel" instead of affecting the preexisting subclasses.
- Disadvantages
    - More code and more classes to write which makes it a little more complex and tedious.

UML Diagram (Original):

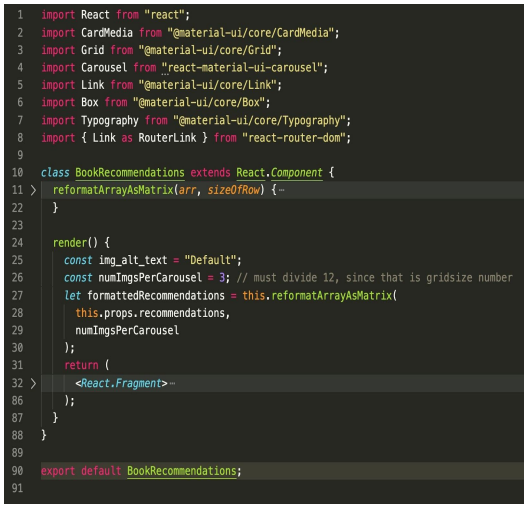


UML Diagram (Improved):



Author/Book Recommendation (Before):

```
import React from "react";
import CardMedia from "@material-ui/core/CardMedia";
import Grid from "@material-ui/core/Grid";
import Carousel from "react-material-ui-carousel";
import Link from "@material-ui/core/Link";
import Box from "@material-ui/core/Box";
import Typography from "@material-ui/core/Typography";
import { Link as RouterLink } from "react-router-dom";

class AuthorRecommendations extends React.Component {
  reformatArrayAsMatrix(arr, sizeOfRow) {…
  }

  fixGenreName(genre) {…
  }

  render() {
    const img_alt_text = "Default";
    const numImgsPerCarousel = 3; // must divide 12, since that is gridsize number
    let formattedRecommendations = this.reformatArrayAsMatrix(
      this.props.recommendations,
      numImgsPerCarousel
    );
    return (
      <React.Fragment>…
    );
  }
}

export default AuthorRecommendations;
```



```
import React from "react";
import CardMedia from "@material-ui/core/CardMedia";
import Grid from "@material-ui/core/Grid";
import Carousel from "react-material-ui-carousel";
import Link from "@material-ui/core/Link";
import Box from "@material-ui/core/Box";
import Typography from "@material-ui/core/Typography";
import { Link as RouterLink } from "react-router-dom";

class BookRecommendations extends React.Component {
  reformatArrayAsMatrix(arr, sizeOfRow) {…
  }

  render() {
    const img_alt_text = "Default";
    const numImgsPerCarousel = 3; // must divide 12, since that is gridsize number
    let formattedRecommendations = this.reformatArrayAsMatrix(
      this.props.recommendations,
      numImgsPerCarousel
    );
    return (
      <React.Fragment>…
    );
  }
}

export default BookRecommendations;
```

Recommendation Carousel (After):

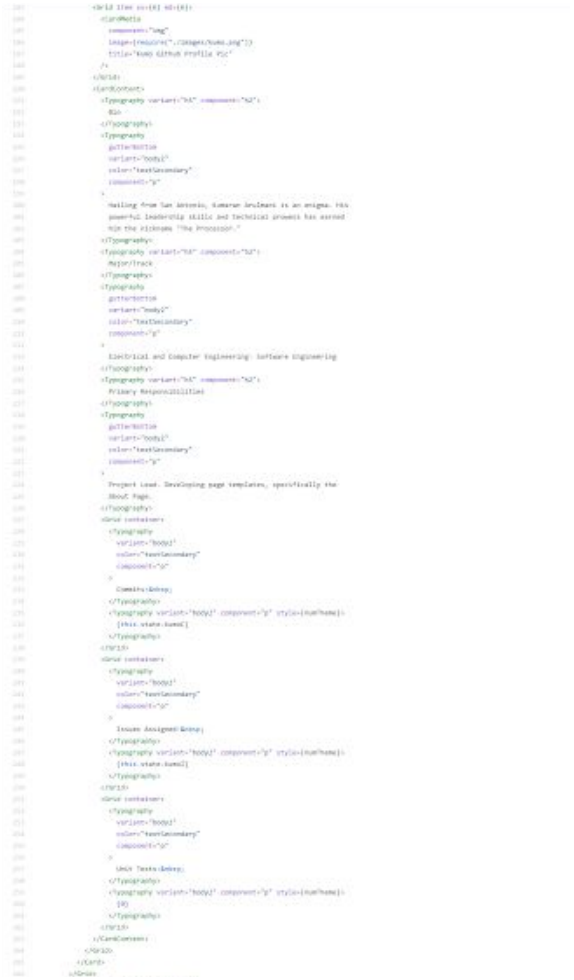
```
import React from "react";
import BookRecommendations from "./BookRecommendations";
import AuthorRecommendations from "./AuthorRecommendations";

class RecommendationCarousel extends React.Component{

    render() {
        switch (this.props.type) {
            case "Author":
                return (
                    <AuthorRecommendations
                        recommendations={this.props.recommendations}
                        typeOfRecommendation={"Recommended Authors"}
                        type={"Author"} />
                );
            case "Book":
                return (
                    <BookRecommendations
                        recommendations={this.props.recommendations}
                        typeOfRecommendation={"Books by this author"}
                        type={"Book"} />
                );
        }
    }
}

export default RecommendationCarousel;
```

Refactorings :

About Us Page

- Originally 862 lines.
- Performed extract class refactoring, reducing code to 166 lines.
- Refactoring was relevant for this class of code, because the class was heavy with many methods. Instead, dividing the methods for this class would be a neater solution than compiling them all in one class.
- Advantages
    - Made the code a lot easier to read & and more organized, the code is now more segmented (with the creation & division of the code into new classes) which allows for easier changes to specific methods/attributes for the About Page.

Code Snippet Original (85 lines):



Code Snippet Refactored (5 lines):

```
<AboutUsBio name="Kumaran Arulmani" image={require("./images/kumo.png")} imagetitle="Kumo Github Profile Pic"
  bio='Hailing from San Antonio, Kumaran Arulmani is an enigma. His powerful leadership skills and technical prowess
  has earned him the nickname "The Processor."' track="Electrical and Computer Engineering: Software Engineering"
  responsibilities="Project Lead. Developing page templates, specifically the About Page."
  commits={this.state.kumoC} issues={this.state.kumoI} tests={0}/>
```

Original Code:

```
for (var i = 1; i < 4; i++) {
  url =
    "https://api.github.com/repos/VishruthiR/EE461L_IDB/commits?page=" +
    i +
    "&per_page=100";
  await fetch(url)
    .then(response => response.json())
    .then(data => {
      total_commits += data.length;
      kc += data.filter(function(e) {
        return e.commit.author.name === "kumosumo";
      }).length;
      dc += data.filter(function(e) {
        return e.commit.author.name === "davidday99";
      }).length;
      mc += data.filter(function(e) {
        return e.commit.author.name === "Matthew Jiang";
      }).length;
      vc += data.filter(function(e) {
        return e.commit.author.name === "Vishruthi Ramaswamy";
      }).length;
      sc += data.filter(function(e) {
        return e.commit.author.name === "sshetkar3858";
      }).length;
      jc += data.filter(function(e) {
        return e.commit.author.name === "Jaino Vennatt";
      }).length;
    });
}
this.setState({ totalC: total_commits });
this.setState({ kumoC: String(kc) });
this.setState({ davidC: dc });
this.setState({ matthewC: mc });
this.setState({ vishC: vc });
this.setState({ sidC: sc });
this.setState({ jainoC: jc });
```

Extracted Method:

```
getUserCommits(data, filter) {
  let commitNum = data.filter(function(e) {
    return e.commit.author.name === filter;
  }).length;
  return commitNum;
}
```

Refactored Code:

```
for (var i = 1; i < 4; i++) {
  url =
    "https://api.github.com/repos/VishruthiR/EE461L_IDB/commits?page=" + i + "&per_page=100";
  await fetch(url).then(response => response.json()).then(data => {
      total_commits += data.length;
      kc += this.getUserCommits(data, "kumosumo");
      dc += this.getUserCommits(data, "davidday99");
      mc += this.getUserCommits(data, "Matthew Jiang")
      vc += this.getUserCommits(data, "Vishruthi Ramaswamy")
      sc += this.getUserCommits(data, "sshetkar3858")
      jc += this.getUserCommits(data, "Jaino Vennatt")
    });
}
this.setState({ totalC: total_commits });
this.setState({ kumoC: kc });
this.setState({ davidC: dc });
this.setState({ matthewC: mc });
this.setState({ vishC: vc });
this.setState({ sidC: sc });
this.setState({ jainoC: jc });
```

Backend API Server: Getting recommended books

- This GET request does two main things:
    - Given the genre, find nine unique authors
    - Get recommended books of these respective authors
- Performed Extract Method to take major chunks of code and move them to separate functions.
- Refactoring this GET request was important because the resulting code at the end of Phase 3 was very hard to read/debug and occasionally resulted in asynchronous issues. By moving the 2 main items into separate functions, we were able to solve our issues and create much easier code to read and modify.

Original Code Snippet:

```
//--Get--Recommended Books
app.get("/recBooks", async (request, response)=>{//Ex:  http://34.71.147.72:80/recBooks?genre=scienceFiction  http://localhost:8080/recBooks?genre=scienceFiction
    var genre = String(request.query['genre']);
    var count =0;
    var cursor;
    var list = [];
    let mySet = new Set();
    var test = [];
    var result;
    for(let i =0; i<10000; i++){
        result =  await client.db("bookAppData").collection("books").findOne({"volumeInfo.genre": genre}, {skip: i});
        mySet.add(result.volumeInfo.authors);
        if(mySet.size==9)
            break;
    }
    mySet.forEach(a=>{console.log(a);test.push({"author": a});});
    console.log("first", test);
    var gaa =[];
    var result;
    for(let i =0; i<first.length;i++){
        result = await client.db("bookAppData").collection("authorImagesNew").findOne({"author": first[i].author},null,undefined);
        gaa.push(result);
    }
    for(let i =0; i<9; i++){
        if(gaa[i]==null){
            gaa[i] = { _id: 4206969,
                author: first[i].author,
                imageLink: 'https://openlibrary.org/images/icons/avatar_author-lg.png'};
        }
    }

    console.log("AUTHAS", gaa);
    console.log(`Looking for books of genre: ${genre}`);
    for(var i =0; i < 9; i++ ){
        var index = Math.floor(Math.random() * 50);
        list.push( await client.db("bookAppData").collection("books").findOne({"volumeInfo.genre": genre },{skip: index},undefined));
    }
    console.log(list.length);
    response.json({list, gaa});
})
```

Refactored Code Snippet(s):

Main function:

```
//--Get--Recommended Books
app.get("/recBooks", async (request, response)=>{                //Ex:  http://34.71.147.72:80/recBooks?genre=scienceFiction    http://localhost:8080/r
    var genre = String(request.query['genre']);
    var list = [];
    const first = await getNineAuthors(client, genre);
    const auth = await getGenreAssociatedAuthors(client, genre, first);
    for(let i =0; i<9; i++){
        if(auth[i]==null){auth[i] = { _id: 4206969,author: first[i].author,imageLink: 'https://openlibrary.org/images/icons/avatar_author-lg.png'};}
    }
    for(var i =0; i < 9; i++ ){
        var index = Math.floor(Math.random() * 50);
        list.push( await client.db("bookAppData").collection("books").findOne({"volumeInfo.genre": genre },{skip: index},undefined));
    }
    response.json({list, auth});
})
```

Extracted Methods:

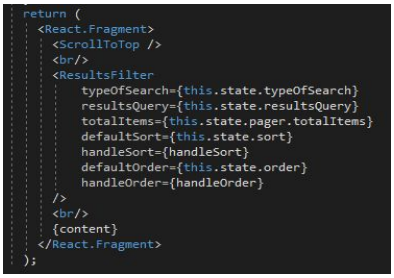
```
async function getNineAuthors(client, genre){
    let mySet = new Set();
    var test = [];
    var result;
    for(let i =0; i<10000; i++){
        result =  await client.db("bookAppData").collection("books").findOne({"volumeInfo.genre": genre}, {skip: i});
        mySet.add(result.volumeInfo.authors);
        if(mySet.size==9)
            break;
    }
    mySet.forEach(a=>{console.log(a);test.push({"author": a});});
    return new Promise(resolve=>{
        resolve(test);
    });
}

async function getGenreAssociatedAuthors(client, genre, first){
    var test =[];
    var result;
    for(let i =0; i<first.length;i++){
        result = await client.db("bookAppData").collection("authorImagesNew").findOne({"author": first[i].author},null,undefined);
        test.push(result);
    }
     return  new Promise(resolve =>{
        resolve(test);
    })
}
```

Results Filter Component

- Originally part of results page
- Performed Extract Class refactoring, removing 30 lines
- Refactoring was relevant because the results filter was relatively self-contained, and so should've been an independent React component instead of cluttering up the Results Page.
- Advantages:
    - Results Page is easier to read
    - Results Page can be modified without having to modify the Filter component
    - Filter component can be swapped with another filter component and used on different pages
- Disadvantages:
    - Results Filter is currently only used on one page, and now to edit the Results Page you must look on two separate files.

Original Code Snippet:

Refactored Code Snippet:



Extracted Class: