COMP3322 Modern Technologies on World Wide Web

Assignment 2 (12%)

[Learning Outcomes 2, 3]

Due by: 23:55, Wednesday April 25 2018

Overview

In this assignment, we are going to develop a simple single-page online bookshop application **FunBooks** using the MEAN stack (MongoDB, Express.JS, AngularJS, and Node.js). The main work flow of **FunBooks** is as follows.

- Upon loading, the sketch of the page is shown in Fig. 1. The user can click a category on the left to view books in different categories. He can click "<Previous Page" or "Next Page>" or change the page number in the select list at the bottom, to view books on different pages in that category.

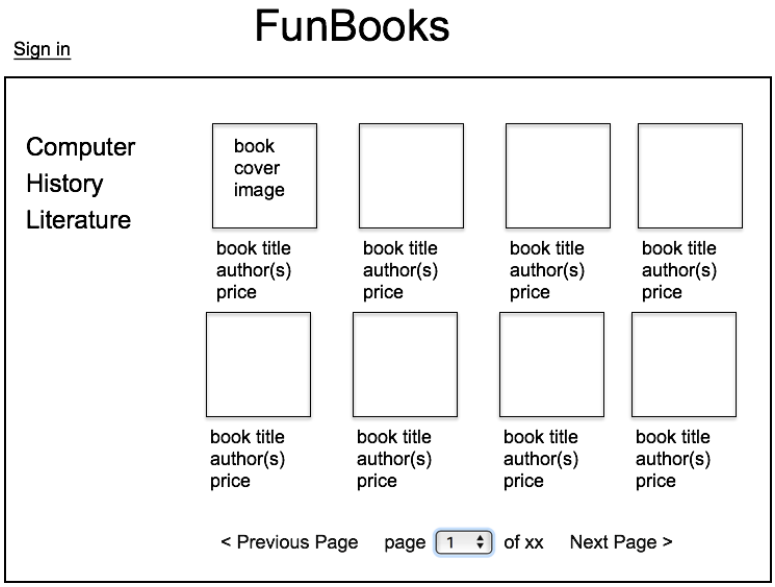

Fig. 1

- When the user clicks the block area containing a book's cover image, title, author(s) and price, the page view becomes one as shown in Fig. 2, where the detailed information of that book is displayed. In addition, there is a side panel where the user can decide the number of copies to purchase for this book, together with an "Add to Cart" button. There is also a "<go back" link on the page.

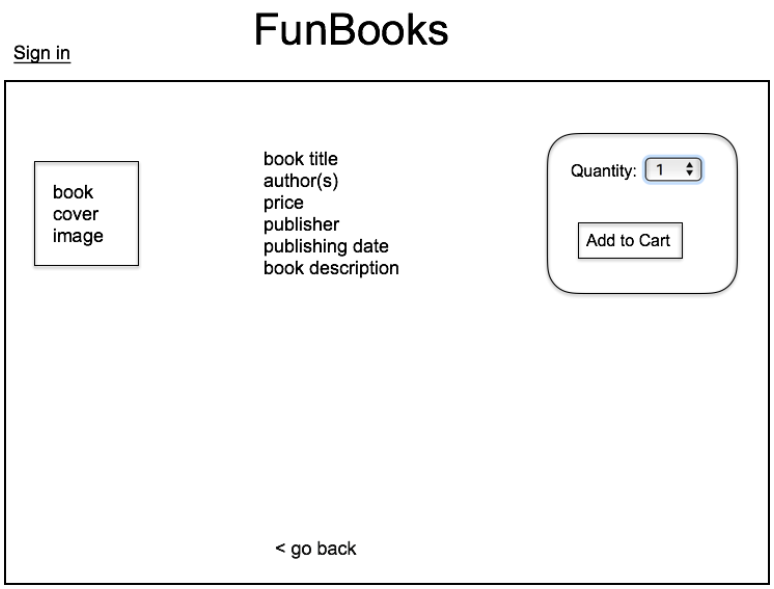

Fig. 2

- When the user clicks the "Sign in" link in one of the page views as shown in Fig. 1 or Fig. 2, Fig.3 is shown.

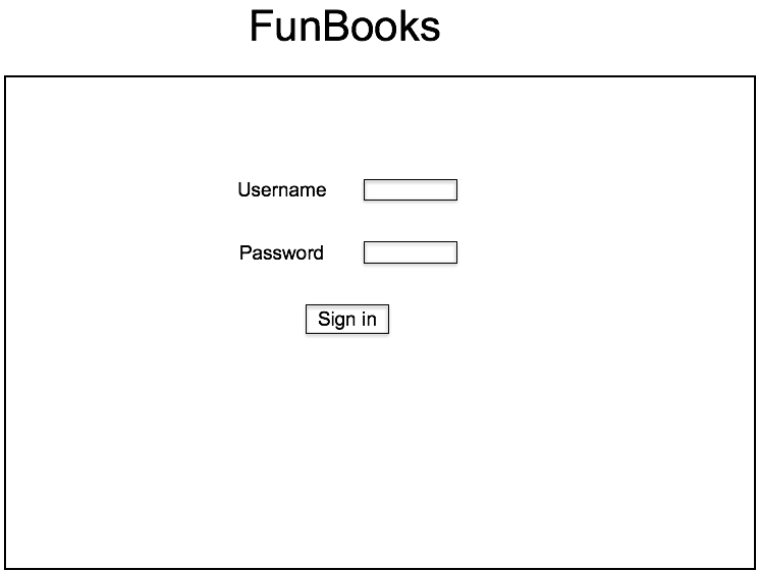

Fig. 3

- For a signed-in user, the page views of Fig. 1 and Fig. 2 become the ones as shown in Fig. 4 and Fig. 5, respectively (suppose the user's name is Henry). The number of items in the shopping cart of the user is displayed on the top-right corner of the page.

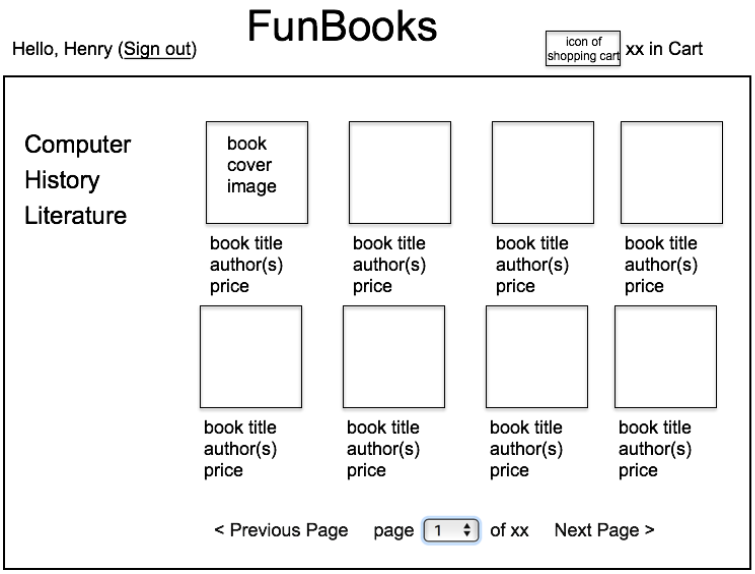

Fig. 4

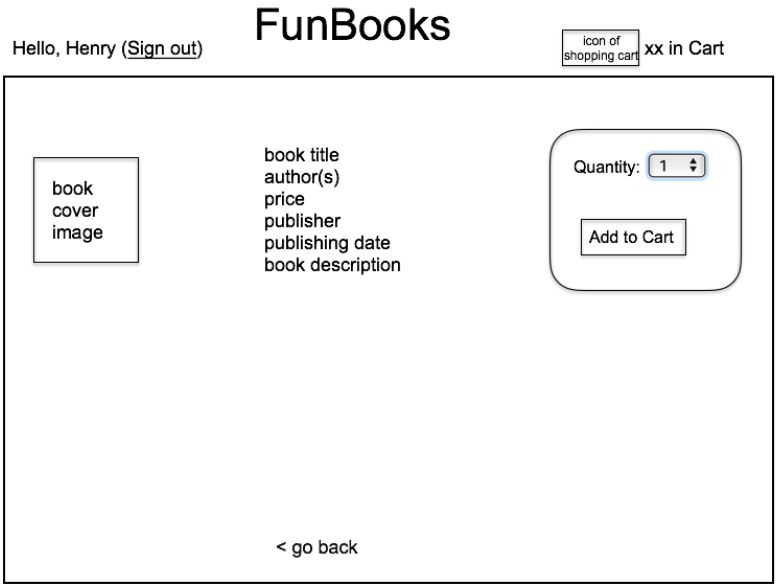

Fig. 5

- When the "<go back" link in Fig. 2 or Fig. 5 is clicked, the page goes back to the previous page view (i.e., the book catalog page containing this book, as shown in Fig. 1 or Fig. 4, respectively).

- When the user clicks "Add to Cart" on the page view in Fig. 5, Fig. 6 is shown. The number in "xx in Cart" at the top-right corner of the page should be updated according to the quantity just added. The total number of items in the cart and total price should be displayed as "Cart subtotal (xx item(s)): $xx". Especially, if there is 1 item, it should be "Cart subtotal (1 item): $xx"; if there are more than one item, it should be "Cart subtotal (xx items): $xx".

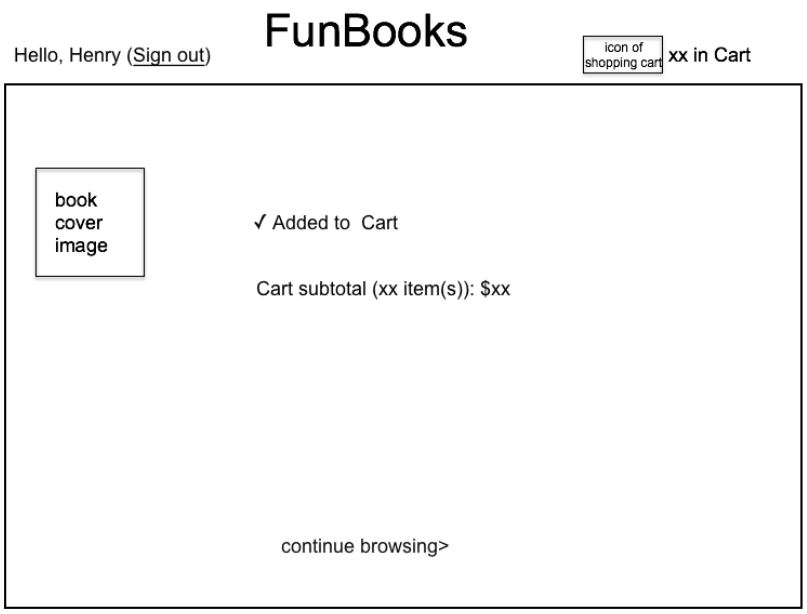

Fig. 6

If the user clicks "Add to Cart" on the page view as in Fig. 2, Fig. 3 is shown, asking the user to sign in first. After successful sign-in, Fig. 6 is then displayed.

- When the user clicks the block area containing the icon of shopping cart and "xx in Cart" on the top-right corner of a page, the shopping cart content is displayed, as shown in Fig. 7. The selected quantity of each book is displayed, and the user can change the quantity. When a quantity is changed, "Cart subtotal (xx item(s)): $xx" should change accordingly.

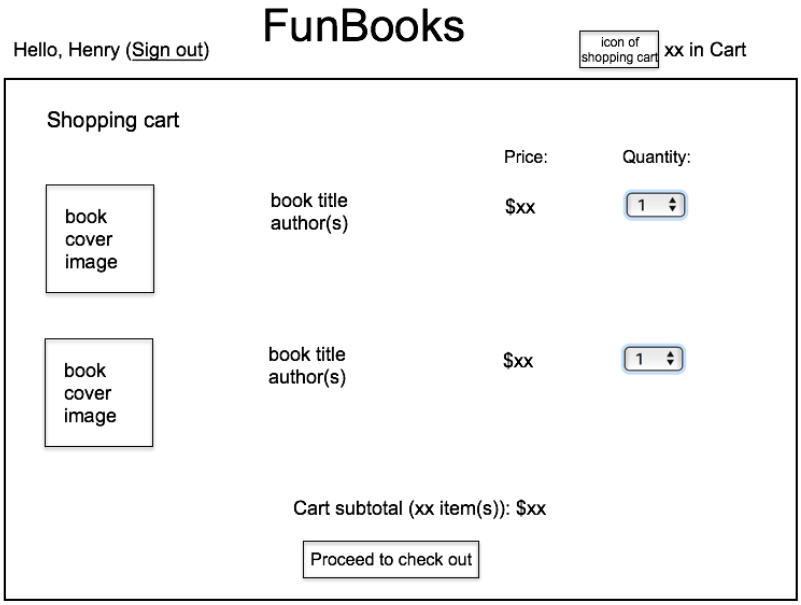

Fig. 7

- When the user clicks the "Proceed to check out" button, the page is as shown in Fig. 8. In our simplified shopping app, we omit the payment page, but directly display this order success page. Note the items in the shopping cart should be cleared such

that “0 in Cart” is displayed on the top-right corner.

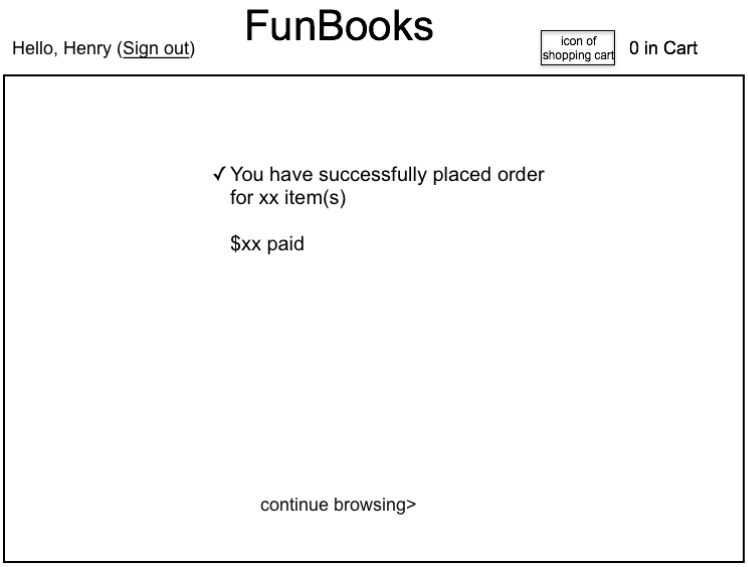

Fig. 8

- When the “continue browsing>” link on Fig. 6 or Fig. 8 is clicked, the page goes back to Fig. 4, with the first page of books in the first category displayed.

- When the user clicks the “Sign out” link in the page views as shown in Figures 4-8, Fig. 9 is displayed. If there is 0 item in the cart, you do not need to display the sentence “You still have xx item(s) in your cart”. If the “Cancel sign-out” button is clicked, the page goes back to the previous page view; if the “Confirm sign-out” button is clicked, the page goes back to: (i) Fig. 1 (to display the same page of book being viewed as where the user is on before clicking the “Sign out” link), if the previous page view is Fig. 4; (ii) Fig. 2, if the previous page view is Fig. 5 or Fig. 6; (iii) Fig. 1 (to display page 1 of the first category), if the previous page view is Fig 7 or Fig. 8.

Fig. 9

We are going to implement this application by implementing code in the following files in an Express app:

app.js
package.json
./routes/myroutes.js
./public/index.html
./public/javascripts/myscripts.js
./public/stylesheets/mystyles.css

Preparations

1. Following steps in Lab 6, install the Node.js environment and the Express framework, create an Express project named **FunBooks**, add dependencies for MongoDB in **package.json**, and install dependencies.

2. Following steps in Lab 6, install MongoDB, run MongoDB server, and create a database "assignment2" in the database server.

(1) Insert a few records to a bookCollection collection in the database in the format as follows, each corresponding to one book.

```
db.bookCollection.insert({'title': 'Convex Optimization', 'category': 'Computer', 'authorList':['Stephen Boyd', 'Lieven Vandenberghe'], 'price': 72, 'publisher': 'Cambridge University Press', 'date': 'March 8, 2004', 'coverImage': 'images/book1.jpg', 'description': 'Convex optimization problems arise frequently in many different fields. This book provides a comprehensive introduction to the subject, and shows in detail how such problems can be solved numerically with great efficiency.'})
```

For implementation simplicity, in this assignment, we do not store images in MongoDB. Instead, we store them in the hard disk under the ./public/images/ folder, and store the path of an image in the bookCollection collection only, using which we can identify a book's cover image in the images folder. Copy a few book cover images to the images folder under the public folder in your project directory.

(2) Insert a number of user records to a userCollection collection in the database in the format as follows. Here bookId should be the value of _id generated by MongoDB for the respective book record in the bookCollection collection, which you can find out using db.bookCollection.find().

```
db.userCollection.insert({'name': 'Henry', 'password': '123456', 'status': 'offline', 'cart':[{'bookId': 'xxx', 'quantity': 1}, {'bookId': 'xxx', 'quantity': 2}], 'totalnum': 3})
```

Task 1. Implement server-side logic (app.js, ./routes/myroutes.js)

app.js **(5 marks)**

In **app.js**, load the router modules implemented in ./routes/myroutes.js. Then add the middleware to specify that all HTTP requests for http://localhost:3000/ should be handled by the router in ./routes/myroutes.js.

Add necessary code for loading the MongoDB database you have created, creating an instance of the database, and passing the database instance for usage of all middlewares.

Also load any other modules and add other middlewares which you deem necessary to implement this app.

Add the middleware to serve requests for static files in the public folder.

We will let the server run on the default port 3000 and launch the Express app using command "npm start".

./routes/myroutes.js **(30 marks)**

In **myroutes.js**, implement the router module with middlewares to handle the following endpoints:

1. **HTTP GET requests for** http://localhost:3000/loadpage?category=xx&page=yy. The middleware retrieves _id, title, authorList, price and coverImage of book records in the bookCollection collection, whose category matches the category value in the URL and which belong to the yy-th display page among all the books in the category. You can specify the number of books to be displayed on each page on the page views as in Fig. 1 and Fig. 4, and display the books in alphabetical order of their titles. The total number of book pages in the category should be computed by retrieving the total number of books in the category, and dividing it by the number of books to display on each page.

If the received category value is "nil", the middleware also retrieves all category values of books in the bookCollection collection. The _id, title, authorList, price and coverImage of book records that it retrieves, and the total number of book pages, should be those of the first category (you can order all categories according to alphabetic order).

The middleware also checks if the "userId" session variable has been set for the user. If yes, it retrieves name of the current user (according to the value of the "userId" session variable), totalnum of items in the cart of the current user from the userCollection collection.

If database operations are successful, all retrieved information and the computed total number of pages in the category should be sent as a JSON string to the client; otherwise, the error message should be sent. You should decide the format of the JSON string (a consistent

format no matter which cases as in the above, e.g., userId session variable set or not, category is "nil" or not), and parse it accordingly in the client-side code to be implemented in Task 2.

Hint: Refer to https://automattic.github.io/monk/docs/collection/find.html on how to retrieve records of specified criteria in a collection.

2. **HTTP GET requests for** http://localhost:3000/loadbook/:bookid. The middleware should retrieve publisher, date and description of the book from the bookCollection collection based on the bookid in the URL. Send all retrieved information as a JSON string in the body of the response message if database operations are successful, and the error message if failure. You should decide the format of the JSON string to be included in the response body.

3. **HTTP POST requests for** http://localhost:3000/signin. The middleware should parse the body of the HTTP POST request and extract the username and password carried in request body (**Hint**: use bodyParser.json()). Then it checks whether the username and password match any record in the userCollection collection in the database. If no, send "Login failure" in the body of the response message. If yes, create a session variable "userId" and store this user's _id in the variable, update "status" of the user in userCollection collection to "Online", and retrieve totalnum of items in the cart of the user from the userCollection. Send totalnum in the body of the response message if database operations are successful and the error message if failure.

4. **HTTP GET requests for** http://localhost:3000/signout. The middleware should unset the "userId" session variable, update status of the user in userCollection collection to "Offline", and send an empty string back to the user if successful and the error message otherwise.

5. **HTTP PUT requests for** http://localhost:3000/addtocart. According to the bookId and quantity contained in the body of the request message, the middleware should update the cart array of the user (identified by the value of "userId" session variable) in the userCollection collection: If the bookId is not contained in the cart array, add an object containing the bookId and quantity in the array; otherwise, increase the quantity of the book of the bookId by the received quantity. In both cases, increase totalnum in the user's record by the received quantity. Compute total price of all items by retrieving quantities and prices from the cart array of the user's record in the userCollection and the book's record from the bookCollection. Return totalnum and the total price to the client in a JSON string if success and the error message if failure.

Hint: use collection.update method to update the record in the userCollection collection (https://automattic.github.io/monk/docs/collection/update.html).

6. **HTTP GET requests for** http://localhost:3000/loadcart. Retrieve values of bookId and quantity in the cart array, as well as totalnum of the user (identified using the value of

"userId" session variable) from the userCollection collection. Then retrieve the title, authorList, price, and coverImage of books of those bookId values from the bookCollection collection. Send all retrieved information as a JSON string in the body of the response message if database operations are successful, and the error message if failure. You should decide the format of the JSON string to be included in the response body.

7. **HTTP PUT requests for** http://localhost:3000/updatecart. The middleware should update the cart array of the user (identified by the value of "userId" session variable) in the userCollection collection, by updating the quantity of the book of the bookId carried in the request body by the quantity in the request body, and updating totalnum in the user's record accordingly. Return totalnum to the client if success and the error message if failure.

8. **HTTP DELETE requests for** http://localhost:3000/deletefromcart/:bookid. The middleware should delete the object {'bookId': 'xx', 'quantity': yy}, where xx is the bookid value in the URL, from the cart of the user's record in the userCollection collection, and update totalnum in the user's record accordingly. Return totalnum to the client if success and the error message if failure.

9. **HTTP GET requests for** http://localhost:3000/checkout. Remove all objects in the cart array of the user's record in the userCollection collection, and set totalnum to 0 in the user's record. Send an empty string to the client if success and the error message if failure.

Task 2 Implement the client-side code (./public/index.html, ./public/javascripts/myscripts.js, ./public/stylesheets/mystyles.css)

Implement **Index.html** and **myscripts.js** **as an AngularJS application**. The application (**FunBooks**) is accessed at http://localhost:3000/index.html.

index.html **(10 marks)**

Index.html should link to the external JavaScript file **myscripts.js** under directory ./public/javascripts/ and the stylesheet **mystyles.css** under directory ./public/stylesheets/.

In **index.html**, include the HTML and angularJS code for creating the page views as in Figures 1-9. **Note**: you can decide the HTML elements to use (unless specified), as long as the page views follow the sketches in Figures 1-9.

./public/javascripts/myscripts.js **(45 marks)**

In **myscripts.js**, implement the code which works together with **index.html** to achieve the following functionalities.

1. **Initial page load.** When the browser loads http://localhost:3000/index.html, an AJAX HTTP GET request is sent to the server to retrieve http://localhost:3000/loadpage?category=nil&page=1. Using data received from the server side, a page as shown in Fig. 1 or Fig. 4 should be displayed (depending on whether there is still a valid user session in the browser). _id of each book should be stored (e.g., in an attribute of the element displaying the book information) for future usage. The numbers in the select element at the bottom of the page should range from 1 to the largest page number in the category.

2. **Book page display according to category selection.** When a category in the category list on the left of the web page is clicked, an AJAX HTTP GET request is sent to the server to retrieve http://localhost:3000/loadpage?category=xx&page=1, where xx is the category value. Using data received from the server side, the first page of books in that category should be displayed, following the sketch in Fig. 1 or Fig. 4 (depending on whether there is a valid user session). _id of each book should be stored (e.g., in an attribute of the element displaying the book information) for future usage.

3. **Book page display according to page selection.** When a page number yy in the select element at the bottom of the page as in Fig. 1 or Fig. 4 is selected, an AJAX HTTP GET request is sent to the server to retrieve http://localhost:3000/loadpage?category=xx&page=yy, where xx is the category of the books currently being displayed. Using data received from the server side, the page of books should be displayed following the sketch in Fig. 1 or Fig. 4 (depending on whether there is a valid user session). _id of each book should be stored (e.g., in an attribute of the element displaying the book information) for future usage.

Suppose the page number of the books currently being displayed is yy. When the "<Previous Page" link is clicked, if yy is larger than 1, an AJAX HTTP GET request is sent to the server to retrieve http://localhost:3000/loadpage?category=xx&page=zz, where xx is the category of the books currently being displayed, and zz equals yy-1; if yy equals 1, do nothing. When the "Next Page>" link is clicked, if yy is smaller than the largest page number in this category, an AJAX HTTP GET request is sent to the server to retrieve http://localhost:3000/loadpage?category=xx&page=zz, where xx is the category of the books currently being displayed, and zz equals yy+1; if yy equals the largest page number, do nothing. When the response from the server is received, using the data it carries, display the respective page of books following the sketch in Fig. 1 or Fig. 4 (depending on whether there is a valid user session). _id of each book should be stored (e.g., in an attribute of the element displaying the book information) for future usage.

4. **Display book information.** On a page view as in Fig. 1 or Fig. 4, when the block area

containing a book's cover image, title, author(s) and price is clicked, send an AJAX HTTP GET request for http://localhost:3000/loadbook/:bookid, where bookid should be the bookId of this book. Using received data from the server side, display a page that shows the book's detailed information and other elements following the sketch in Fig. 2 (if the user has not signed in) or in Fig. 5 (if the user has signed in).

5. **Go back.** When the "<go back" link on the page view as in Fig. 2 or Fig. 5 is clicked, the page goes back to Fig. 1 or Fig. 4, respectively. The page of books displayed should be the previous page before the page as in Fig. 2 or Fig. 5 (when the "<go back" link is clicked) is displayed.

6. **Sign in**. When the "Sign in" link on a page view as in Fig. 1 or Fig. 2 is clicked, display a page as in Fig. 3. When the user clicks the "Sign in" button on the page, check if both the username and password input textboxes are non-empty: if so, send an AJAX HTTP POST request for http://localhost:3000/signin, carrying the input username and password in the request body; otherwise, show an alert popup box stating "You must enter username and password". If a "Login failure" response is received, the page view remains as in Fig. 3, except for displaying "Login failure" above the input boxes in addition; otherwise, display the page as in Fig. 4 or Fig. 5 based on information received (depending on whether the "Sign in" link was clicked on Fig. 1 or Fig. 2). If a page view as in Fig. 4 is displayed, the page of books displayed should be the same as those displayed before clicking the "Sign in" link.

7. **Sign out.** When the "Sign out" link on a page view as in Figures 4-9 is clicked, display the page as in Fig. 9. When the "Cancel sign-out" button is clicked, the page returns to the previous page view. When the "Confirm sign-out" button is clicked, send an AJAX HTTP GET request for http://localhost:3000/signout. When an empty string is received from the server side, the page goes back to: (i) Fig. 1 (to display the same page of book being viewed as where the user is on before clicking the "Sign out" link), if the previous page view is Fig. 4; (ii) Fig. 2, if the previous page view is Fig. 5 or Fig. 6; (iii) Fig. 1 (to display page 1 of the first category), if the previous page view is Fig 7 or Fig. 8. Note that more AJAX requests may need to be sent to retrieve needed information for page display.

8. **Add to cart.** When the "Add to Cart" button on a page view as in Fig. 2 is clicked, a sign-in page as in Fig. 3 is displayed. When the user clicks the "Sign in" button on the page, check if both the username and password input textboxes are non-empty: if so, send an AJAX HTTP POST request for http://localhost:3000/signin, carrying the input username and password in the request body; otherwise, show an alert popup box stating "You must enter username and password". If a "Login failure" response is received, the page view remains as in Fig. 3, except for displaying "Login failure" above the input boxes in addition; otherwise, send an AJAX HTTP PUT request for http://localhost:3000/addtocart, containing key-value pairs of bookId of the book and the quantity selected in the previous page view (as in Fig. 2, where the user clicks the "Add to Cart" button). Using data received from the server side, display a page view as in Fig. 6.

When the "Add to Cart" button on a page as in Fig. 5 is clicked, send an AJAX HTTP PUT

request for http://localhost:3000/addtocart, containing key-value pairs of bookId of the book and the quantity selected in the current page view. Using data received from the server side, display a page as in Fig. 6.

9. **Load the shopping cart**. When the user clicks the block area containing the icon of the shopping cart and "xx in Cart" on the top-right corner of the page view as in Figures 4-9, send an AJAX HTTP GET request for http://localhost:3000/loadcart. Display a page as in Fig. 7 using data received from the server side. Store _id of each book in the cart (e.g., as an attribute of the element displaying the book information) for future usage. The numbers in the select elements should range from 0 to a large number at your choice.

10. **Update the shopping cart.** When the user has changed the number in a select element on the shopping cart display page as in Fig. 7, if the number is not 0, send an AJAX HTTP PUT request for http://localhost:3000/updatecart, containing key-value pairs of bookId and the quantity selected of the corresponding book. Use the total item number received from the server side, and the total price computed with book prices and quantities in the current shopping cart, to update the current page view.

11. **Delete a book from the shopping cart**. When the user has changed the number in a select element to 0 on the shopping cart display page as in Fig. 7, send an AJAX HTTP DELETE request for http://localhost:3000/deletefromcart/:bookid. Use the total item number received from the server side, and the total price computed with prices and quantities of books remaining in the shopping cart, to update the current page view.

12. **Check out**. When the "Proceed to check out" button on the shopping cart display page as in Fig. 7 is clicked, if the shopping cart is empty, do nothing; otherwise, send an AJAX HTTP GET request for http://localhost:3000/checkout. If an empty string is received from the server side, display a page as shown in Fig. 8.

13. **Continue browsing**. When the "continue browsing>" link on a page as in Fig. 6 or Fig. 8 is clicked, send an AJAX HTTP GET request for http://localhost:3000/loadpage?category=nil&page=1. Using data received from the server side, display a page as shown in Fig. 4, i.e., display the first page of books in the first category.

./public/stylesheets/mystyles.css (**5 marks**)
Style your page views nicely using CSS rules in **mystyles.css.**

Other marking criteria:

(**5 marks**) Good programming style (avoid redundant code, easy to understand and maintain). You are encouraged to provide a **readme.txt** file to let us know more about your programs.

Submission:

You should zip the following files (in the indicated directory structure, removing any automatically generated files from those directories, which were created when you create the project) into a **yourstudentID-a2.zip** file

app.js
package.json
./routes/myroutes.js
./public/index.html
./public/javascripts/myscripts.js
./public/stylesheets/mystyles.css

and submit the zip file on Moodle:

(1) Login Moodle.
(2) Find “Assignments” in the left column and click “Assignment 2”.
(3) Click “Add submission”, browse your .zip file and save it. Done.
(4) You will receive an automatic confirmation email, if the submission was successful.
(5) You can “Edit submission” to your already submitted file, but ONLY before the deadline.