Web Application Development Project Submission 2023

Name: William (Luke) Blunden .....

Student ID: G00411261.....

Date: 23/05/23.....

Home page/index page/start page (eg., page user should open first): http://localhost:3000/ (index.html).....

Using the site - information:

Default user credentials: username: user, password: pass

Index.html at the root of the website acts as a landing page for the rest of the store. It contains some site info, a carousel which links straight to the shop, or a pair of login/ signup links. In all locations the logo/ site name also link to the shop

Assuming you click through to the shop, you're presented with a page of movie posters. A row at the top contains poster collections under featured products, and several rows underneath contain the rest of the posters that are for sale. At the top of the page there is navbar containing a link to the shop, a dropdown to filter the posters by genre, a dropdown to either login/signup if you are not logged in, or a logout link if you are, and a link to the checkout.

Clicking on a poster will bring you to a page featuring that poster. On the left side there is a navigable carousel showing some images of the poster. Underneath the carousel are reviews for this and other products, with reviews for this product located at the top and highlighted in blue. On the right there is some information about the movie such as title, director, genre. The genre of the film also acts as a link to see other movies with that genre.

The dimensions drop down provides a list of sizes of poster available with their associated cost. For a collection these costs are 2.5x the amount compared to individual posters. If you select one of these sizes and click add to cart, the item will be saved to your local storage, and you will be redirected back to the shop.

Underneath the description you can fill out a review for the item that you're currently viewing. You do this by submitting your name, a star rating and your review, and clicking submit review. This sends your review to the database and reloads the item page where you should now see your review at the top of the list.

Clicking a genre under the shop by genre dropdown or by clicking the genre link on an item page, will bring you to a tiled display of other movies that have this genre, with the selected genre displayed at the top of the page, like the shop page. Clicking on one of these posters will similarly bring you to that item page.

Clicking on login or signup while logged out will bring you to their respective pages, where you have a form to fill in with your username and password. You are required to fill in both fields, and if you provide incorrect credentials while trying to login or use an existing username while trying to sign up, you will be shown a warning message and prompted to try again. Once logged in/ signed up you will be forwarded on to the shop page.

Once logged in the dropdown only shows the option to logout, which if selected logs you out and sends you to the shop page.

When clicking on the checkout, if you are not logged in you will be redirected to the login page, with the same functionality as before, however once you are logged in instead of being redirected to the shop you will be redirected to the checkout.

On the left side of the checkout page, you are given a summary of your order showing the movie poster image, name, quantity, dimensions and price. Underneath all the chosen items is a total cost. On the right side of the checkout is a form to fill in with your shipping details. Once the details are provided and the order is placed, your order is sent to the database, your local storage is cleared, and you are redirected back to the shop.

Project Requirements Implementation

ITEM 1	**Reference**
Allow the customer to enter their login details:	login.js handles users logging in, and forwards them to the entry.ejs file which handles both logging in and signing up
Login details validated (via a login screen) before receiving a summary of the order:	When clicking on the checkout link, if not logged in you are redirected to the login page, and cannot access the checkout until you have successfully logged in. Validation is done using a combination of HTML5 in the entry.ejs file and some backend authentication in the login.js file
Username set to "user"	user
Password set to "pass"	pass
Brief description of implementation details:	Username and password are validated using the appropriate types on the HTML form inputs and using the required tags. Once the form is submitted to login.js, the username and password are checked against the database using the authenticateUser function in the auth module. If authenticated, login.js then saves the user details to an express session which saves the authenticated access as a cookie, keeping you signed in until you sign out.

ITEM 2	Reference
Perform form validation through JavaScript or HTML to ensure that text fields are not empty, and a valid email address is entered	Forms are validated through HTML using the correct types for inputs tags and setting all fields to be required.
Brief description of implementation details:	Forms are located in the checkout, item, login and signup pages, and all use these input types and required tags, except for the text area under item review, as this is not required to submit a review.

ITEM 3	Reference
Include a slideshow or carousel which displays a different image each time the page is loaded;	Carousels are included on the specific item pages and on the landing page
Brief description of implementation details:	The carousels are both based on the one shown on the bootstrap website. The landing page carousel is not navigable and automatically slides, it has a shop caption positioned over the front of it to signify it's a link and is wrapped in a link tag to the shop page. The functionality to have it show a random of the 4 images is contained in the script tags at the bottom of the page. The carousel on an item page is navigable and randomly starts on either the first or second of the 3 images. Images are determined randomly by getting a random number between 0 and 1 using Math.random() and then checking whether the number falls within a range, and adding the active class to an item based on which range the number is in.

ITEM 4	Reference
Allow the user to 'purchase' items from the site;	At checkout, all previously selected items will be shown with their sizes, quantities and prices. The user must then enter their shipping details before placing their order. Once the order is placed, the shipping details and order info are sent to the database through checkout.js
Brief description of implementation details:	While shopping the order details are saved in local storage, using the product ID as the key and a map for the value. The map contains a mapping of size ids to quantities. At checkout once the shipping details are validated using the form, a post request is made to /checkout, which is handled by checkout.js. This included a hidden input which contains the local storage values stringifyed. The shipping details are inserted into the orders table of the database, and checkout.js receives back the orderid primary key from this insert. Inserts are then made to the orderitems table, using the previous orderid, as well as the productid, sizeid and quantity which have been parsed from the localStorage data. A purchase date is also included.

ITEM 5	Reference
Use an object or an array in JavaScript;	Arrays and objects have been used throughout the project in the backend js files and within HTML script tags.
Brief description of implementation details:	In the get method in item.js, 2 arrays of data are returned from queries to the database requesting reviews with a particular product ID and reviews specifically not containing that product ID. These arrays are then concatenated together and sliced to only contain the first 10 elements, so the reviews section of a page doesn't get too long. Objects are used when rendering the error page throughout the routing files, to pass a status code and an error message. Maps are used to map the poster size to the quantity ordered in local storage

ITEM 6	Reference
Use at least one custom module in node;	I have split the routes into separate modules using express.Router, moved the database connection into it's own module, and made some extra modules for common queries to the DB
Brief description of implementation details:	I made the database connection into its own module by moving the code for creating the connection and connecting to the database into a db.js file. I then promisifyed the connection.query so it can return promises, which allows us to await data from asynchronous calls to the database, an issue that came up when I wanted to make several queries within 1 get request method. I then exported this query method. With the database connection in a module I then moved the different routes into their own files for each route using express.Router, and exporting the router. App.js then imported these route modules and used them when requests are made to URL paths.

ITEM 7	Reference
Include capability for handling post and get requests;	A few different routes handle get and post requests, for example signup.js
Brief description of implementation details:	For the get request, signup.js tries to render the entry.ejs file, sending with it an object containing task and route information. This is to visually indicate and set up the correct routes in the post form for the entry.ejs file which handles both the signup and login pages. If there is an error the get request renders the error page with a status code and error message For the post request, which is routed to from a method='post' from on the signup page, signup.js retrieves the username and password input to the form inputs. It checks that it has received both and renders the entry page again with a warning message if not. If both have been provided it then checks whether the username already exists using the auth module. If they do it renders the entry page with a warning message. If not it creates the user with the auth module which inserts them into the database, and sets the user into the session. It then redirects to the shop. If any errors occur during this is renders an error page with a status code and an error message.

ITEM 8	Reference
Include both static and dynamic content;	Examples of static and dynamic content can be found in navbar.ejs, shop.ejs and index.html
Brief description of implementation details:	Navbar.ejs includes some static items such as the logo and shop title, the shop link and the checkout link. The shop by genre dropdown is dynamically generated by using a for loop to loop through the genres passed to it. The login/signup/logout dropdown is also dynamically generated by checking with the user is logged in or is a guest. Nearly all the shop.ejs content is dynamic, as the navbar is included as another template, and the featured movies and the rest of the movies are dynamically generated by looping through the arrays provided and generating card elements. Index.html is completely static (excluding the carousel that moves...)

ITEM 9	Reference
Include the use of templates in Node;	I've used a few ejs templates in node for example shop.ejs with navbar.ejs and meta.ejs, as well as entry.ejs
Brief description of implementation details:	Navbar.ejs and meta.ejs are templates that are inserted into other templates as they were common items across several pages, e.g., shop.ejs. Shop.ejs includes the meta.ejs template in the head tags and the navbar.ejs template at the top of the container div. It then dynamically generates the shop content based on the featured movie collections and other movies provided to it. Entry.ejs serves both the login and signup pages, by having data passed to it which it injects into the page heading and the action property of the form, so the form points to the correct route.

ITEM 10	Reference
Include error messages to provide feedback to users in case of issues or errors;	entry.ejs displays warnings when users enter incorrect credentials, and error.ejs displays error messages when errors occur in node or the database.
Brief description of implementation details:	The warnings in error.ejs were implemented by including a warning div at the bottom of the page that checks whether an message has been supplied by the server, and displays it if so. Error.ejs displays the status code and error message supplied by the back end when an error occurs. This typically occurs in the catch block of the try catch blocks in the route methods, which instead of rendering the correct page renders the error page with the status and error info. The error page is also rendered when some result would result in the website not functioning properly, for example in shop.js, if no products are retrieved from the database then the user is given a 404 error page.

ITEM 11	Reference
Connect to a database that contains relevant site information (eg., product info, prices) using NODE (your database name should be your ATU ID);	I've included 7 tables within my G00411261 database; genres, orderitems, orders, proddata, reviews, sizes, users
Brief description of implementation details:	The initial table was proddata which contained the product data including title, image locations. I then needed to create a sizes table, which contained dimensions and prices for those sizes when users are making orders. I wanted to implement a way to filter the movies by genre so I created a genres table, which contains the genre and an ID used by proddata as a foreign key To be able for users to make orders I then needed an orders tables to store their shipping information. Because of the nature of the orders being different sizes and quantities, I couldn't store this with the orders, so I needed an orderitems table which stores an entry for each size, and references the product ID, size ID, and order ID as foreign keys. I then wanted to implement the functionality to display and add reviews to the site so I created the reviews table, which references the product ID's as a foreign key Finally I wanted user credentials to be stored on the database instead of the server so I created a users table.

ITEM 12	Reference
Use Bootstrap version 5 via CDN	Bootstrap has been used throughout the project, from overarching layout elements such as the containers, rows and columns, but also for specific elements like cards and carousels.
Brief description of implementation details:	Layout wise, containers rows and columns were used to consistently display content across the site, and column classes were used to ensure the content is responsive. Classes were also used to alter margins/ padding, remove text-decoration, align content, etc. The poster images are generally either displayed in cards on the shop and genre pages or in carousels at the index and item pages. These were based on the bootstrap examples, but then used additional bootstrap or custom classes to modify their appearance, or introduced additional elements such as the text over the index page carousel.

Additional information:

Wish List/ if I had more time:

- Better user security/ salting passwords
- Adding items to a wish list
- Being able to remove or change quantity of items at checkout
- A search bar
- Recommendations based on past purchases (using cookies I imagine)

Used Bootstrap 5.3: https://getbootstrap.com/docs/5.3/

Images and some text taken from: https://www.etsy.com/ie/shop/PartStudioShop

References

Node.js. Available at: https://nodejs.org/en (Accessed: 23 May 2023).

W3Schools free online web tutorials (no date) W3Schools Online Web Tutorials. Available at: https://www.w3schools.com/ (Accessed: 23 May 2023).

MozDevNet (no date) Async function - javascript: MDN, JavaScript | MDN. Available at: https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function (Accessed: 23 May 2023).

ExpressJS - authentication (no date) Tutorials Point. Available at: https://www.tutorialspoint.com/expressjs/expressjs_authentication.htm (Accessed: 23 May 2023).

Routing (no date) Express Routing. Available at: https://expressjs.com/en/guide/routing.html (Accessed: 23 May 2023).