**Web Application Development Project Submission 2024**

Name: Anastasia Kucerovska

Student ID: G00439362

Date: 19 May 2024

Home page/index page/start page (e.g., page user should open first): index.ejs

Using the site - information:

This site is built on the idea of 'Groupon' and 'Pigsback'. On the index.ejs/home page, you start off by seeing the logo and navigation panel which shows the main categories offered. You have a carousel of some of the potential options, some information explaining the website, an option to subscribe to a newsletter, customer reviews, and the footer. The idea of the website is to offer customer an option to try something new, whether dining, services, or experiences at a fraction of the price, and to offer businesses a chance to list their own dining, service, and experience option with a discount, in order to gain new customers.

The content pages, 'Things To Do', 'Food & Drink', 'Beauty & Spas', and 'Hotels & Travel' offer information of the products and services from the G00439362 product database. They let the customers explore the local/national/internation offerings, provide information on those offerings, the pricing, and the option to add those items to cart.

The Sign In/Log In page helps redirect user to the Summary/Cart and is accessible through user and pass log in details as per the project brief. If the information is incorrect, the user is notified of that. Here the customer is able to look through the added products and the total and proceed to checkout.

**Project Requirements Implementation**

|  |  |
| --- | --- |
| **ITEM 1** | **Reference** |
| *Allow the customer to enter their login details:* | See page login.html and the associated JavaScript function from auth.js and index.js. |
| *Login details validated (via a login screen) before receiving a summary of the order:* | Validation is performed using JavaScript function and Node.js on auth.js and tested/routed on index.js  Validation of the login form is processed by the route handle in index.js. This validates the username and password through the POST method, using the authenticateUser function in auth.js. If validated, the user is redirected to the order summary page. If details are incorrect, the user is notified that the details are incorrect. |
| *Username set to “user”* | user |
| *Password set to “pass”* | pass |
| *Brief description of implementation details:* | This implementation involves three components:  1. login.html – the html form allows user input of username and password.  2. auth.js – module provides JavaScript function to create users and authenticate their credentials. Maintains in-memory user store and checks that the credentials match the stored ones.  3. index.js - the Express server handles form submission, validates the credentials from the auth.js, and manages the user sessions. Authorized logins are redirected to order summary, while unauthorized produce an error. |

|  |  |
| --- | --- |
| **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:* | The Newsletter form in index.ejs is implemented with both HTML and JavaScript. The HTML has attributes, such as required and type to implement basic client-side validation, such as the email field not being empty and being formatted in the correct style. Then JavaScript handles the form submission and performs further validation before sending the data via an AJAX request. |
| *Brief description of implementation details:* | This involves creating a Newsletter form in index.ejs that contains HTML5 validation attributes. This is intercepted by JavaScript to extract the email address and send it via fetch API request. On the server side, index.js handles the POST request to check if the email provided is valid and inserting it into the database. This then responds with the appropriate status codes and messages. |

|  |  |
| --- | --- |
| **ITEM 3** | **Reference** |
| *Include a slideshow or carousel which displays a different image each time the page is loaded;* | Carousel can be found on index.ejs, both the item and the associated JavaScript function. This Carousel displays a different image every time the page is loaded and is linked to the product pages. |
| *Brief description of implementation details:* | The Carousel structure is defined in HTML with images and uses as well as modifies the code from Get Bootstrap. Internal CSS is used to enhance the visual of the carousel, and JavaScript is responsible for the logic that randomly selects a starting image when the page loads, as well as the intervals to cycle through the images. |

|  |  |
| --- | --- |
| **ITEM 4** | **Reference** |
| *Allow the user to ‘purchase’ items from the site;* | This can be seen on thingstodo.ejs, foodndrink.ejs, beautynspas.ejs, hotelsntravel.ejs, cart.html, cart.js and checkout.ejs. I wanted the website to have a more realistic feel to it, so I created a JavaScript function that the product and quantity from each individual category page and can be added to a cart. The checkout can only be accessed when the user is logged in. |
| *Brief description of implementation details:* | Users have the ability to add items to cart. The cart is similar to the user authentication. This means that in memory we keep the product id and the quantity the user has added / removed from their cart. I created cart.js in static/js folder to support the ability to add item to cart.  Once the user has added their items to cart, they can then review their cart which is completely dynamically generated. If they are logged in, they can purchase the order in checkout.ejs. If not, the user is redirected to the login page. |

|  |  |
| --- | --- |
| **ITEM 5** | **Reference** |
| *Use an object or an array in JavaScript;* | For example, see cart.js and associated files |
| *Brief description of implementation details:* | This is used in multiple places, the way the cart items are handled is done by a map data structure. Other arrays are used when we render the templates. So, for example, when the user hits the thingstodo endpoint, a call to the database is made and then an array of products is returned to a template called thingstodo.ejs.  The above shows usage of objects and arrays and additionally a map data structure for the cart. |

|  |  |
| --- | --- |
| **ITEM 6** | **Reference** |
| *Use at least one custom module in node;* | I used the auth.js module to handle user authentication. It's responsible for creating users and authentication management. It includes the function such as createUser, and authenticateUser. This is utilized in my index.js file to validate the login details. |
| *Brief description of implementation details:* | Like the authentication section above, in the index.js file, the auth.js module is imported. This custom module provides functions to create and authenticate users. This is tested with 'user' and 'pass'. The login route in index.js functions from auth.js to verify the credentials entered by the user. If they are correct, the user is redirected to the cart page, otherwise the user gets an error message. |

|  |  |
| --- | --- |
| **ITEM 7** | **Reference** |
| *Include capability for handling post and get requests;* | The website handles both POST and GET requests to facilitate user interactions and data management.  GET requests are used to retrieve information and render pages, this is done to make sure that users can access different sections of the website, such as product details and categories. This can be seen in index.js.  Similarly, POST requests handle actions such as the Newsletter submission forms and user login, allowing for interactive features that require server-side data processing. This is also seen in index.js. |
| *Brief description of implementation details:* | This implementation relies on the Express framework in the index.js file to define routes for handling GET and POST requests.  Static files and specific pages are rendered using GET routes, which include database interactions to fetch product data based on categories or specific IDs.  The POST routes are used to manage form submissions, such as the sign up for the Newsletter and user authentication, which involves data validation and database operations. |

|  |  |
| --- | --- |
| **ITEM 8** | **Reference** |
| *Include both static and dynamic content:* | Static content is the images that are used for the various products, and the dynamic content in my website is the EJS files, for example, cart, checkout. |
| *Brief description of implementation details:* | This means that static resources have their own folder, in my project this is called 'static'.  Dynamic content is completely generated by ejs. So, all cart, checkout interactions are done by dynamic templates. |

|  |  |
| --- | --- |
| **ITEM 9** | **Reference** |
| *Include the use of templates in Node:* | I've used the EJS (Embedded JavaScript template) through the index.ejs file, and all the category content files, for example, thingstodo.ejs. |
| *Brief description of implementation details:* | The EJS template is used to dynamically generate HTML content based on the server-side data. By having the category content done with this template, it has proven to be a powerful tool to be able to connect the webpages and the database to provide a great user experience. |

|  |  |
| --- | --- |
| **ITEM 10** | **Reference** |
| *Include error messages to provide feedback to user sin case of issues or errors;* | Error messages are integrated in most dynamic aspects of the website to communicate with the user during those operational scenarios, for example, during form submissions and database operations. |
| *Brief description of implementation details:* | For example, the Newsletter subscription functionality - if a user submits an empty email field, the server-side code in Node.js promptly checks for this condition and responds with an HTTP 400 status code and the message "Email is required." Furthermore, if there's a server-side error during the database insertion (like a duplicate email), the code catches this error and returns an HTTP 500 status along with the message "Error subscribing to newsletter."  On the client side, the JavaScript handling the form submission uses **fetch** to send data to the server and includes **.catch()** to handle any errors that might not have been caught by server validations, displaying an alert like "Sorry, your email exists or it could not be added." |

|  |  |
| --- | --- |
| **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);* | The database is used with the student ID name, so 'G00439362', and it contains both the 'productData' table and the 'subscribers' table. |
| *Brief description of implementation details:* | The implementation follows the lecture videos in Week 11 and is later expanded to hold an additional table to handle the Newsletter subscription form user emails. |

|  |  |
| --- | --- |
| **ITEM 12** | **Reference** |
| *Use Bootstrap version 5 via CDN* | This can be seen in all pages |
| *Brief description of implementation details:* | As shown in the lecture videos, the CDN information has been copied and pasted from the Bootstrap website directly into the website pages. |

Additional information:

1. I've spread out the option to 'Add to Cart' on multiple pages to make the website feel more dynamic and user friendly. This still calculates the total in the cart and allows a logged in user to access the checkout.

2. The Login Form is only accessible when the user clicks the Checkout button in the Cart, which automatically brings you to a Login Page.

3. Dynamic price calculation is applied to the cart, for line item, and for cart summary.

4. Dynamically render the Log Out button depending on if the user is logged in or not.

5. Significant amount of custom cart code written.