**Cocktail Canvas** – Documentation

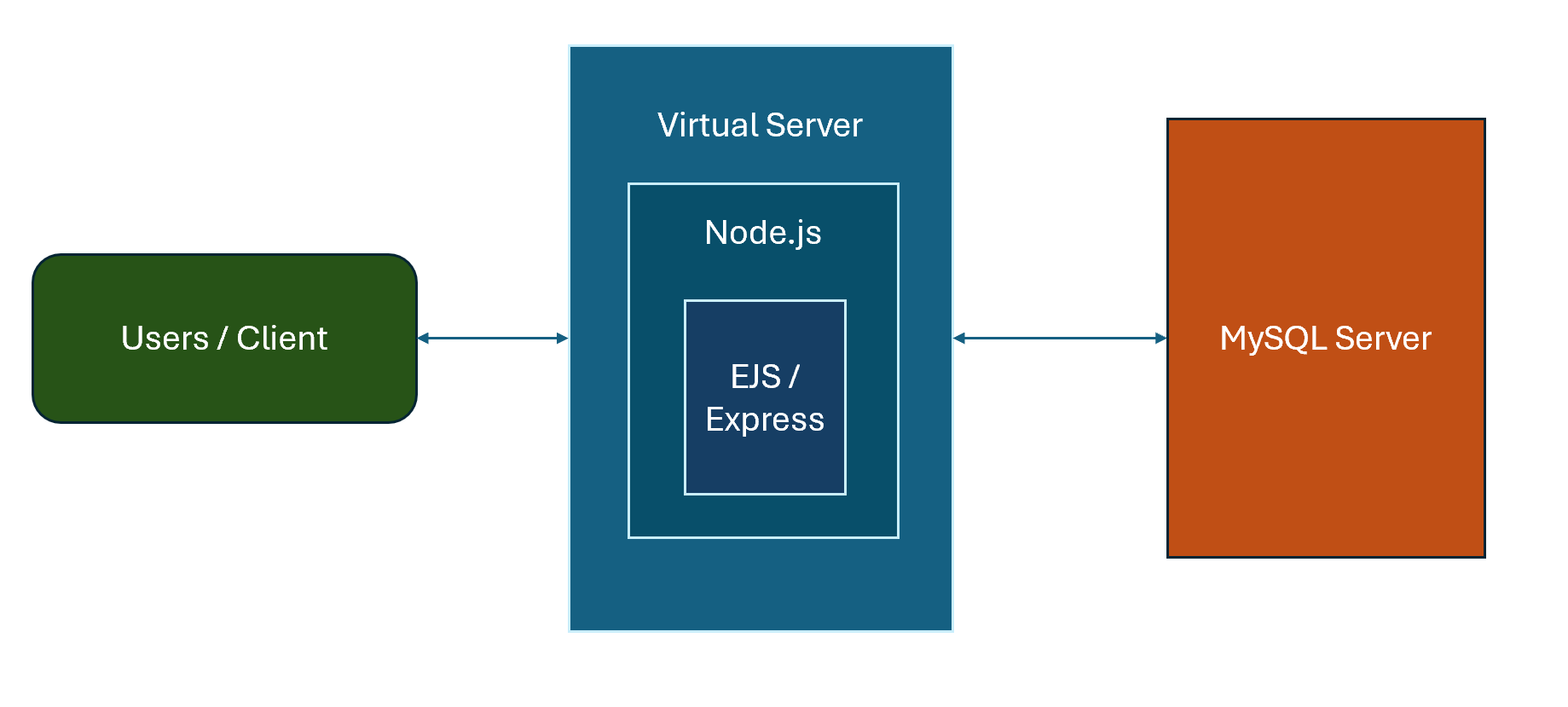
Arosd001 – Alastair Ros-Davison  
  
  
**Outline:**

Cocktail canvas is a full stack cocktail menu development website. It features a database of menus created by users, populated by drinks, all easily listable and searchable for relevant interests. Once registered, users can then begin to create their own drink menus. This can be achieved through a number of avenues.  
Once registered and logged in, one can begin to edit a menu. At this stage a dynamically refreshing page displays your menu at it’s current state, and allows the building of custom drinks, with user added properties and lists of ingredients and measurements, method, glass type and so on, each relating to a different table or join table within the relational database.  
These can be then added to the menu.   
Alternatively, one can search TheCocktailDB external API for existing drinks to add to their menu. Finally they can also search the internal Cocktail Canvas database for existing drinks to add to their menu as well.  
It also features a RESTFUL API, for GET, SEARCH and LIST requests, as well as some other routes, for all parts of the database.  
  
**Links and logins:**

* URL: <https://doc.gold.ac.uk/usr/717/>
* Github: <https://github.com/Alastairrd/Cocktail-Canvas>
* Username: MarkingUser1
* Password: haveaniceday

**Architecture:**

The application tier is built with Node.js and the Express.js framework, as well as using EJS for templating dynamic views on the frontend. It is all hosted on a virtual server managing deployment and running of the app. There are various express modules for enhancing middleware functionality.  
The data tier is handled by a MySQL Server, taking care of data storage and management. Communication comes between the node app and the MySQL Server using parameterised queries using stored procedures.



***Figure 1****: High level Architecture*

**Data Model:**

A screenshot of a computer

Description automatically generated

***Figure 2****: Data Model*

The model comprises of a table of users, which create drink menus.  
Menus then feature drinks, which is what the rest of the model seeks to solve.  
There are 4 tables: menus, drinks, glass, ingredients. Menus are linked to users via a foreign key of user\_id. The rest of the tables are linked via 3 join tables.

* Drinks and menus linked by menudrinks, this is where price is listed.
* Drinks and glasses linked by drinkglass.
* Drinks and ingredients linked by drinkingredients, this is where measurement is listed.

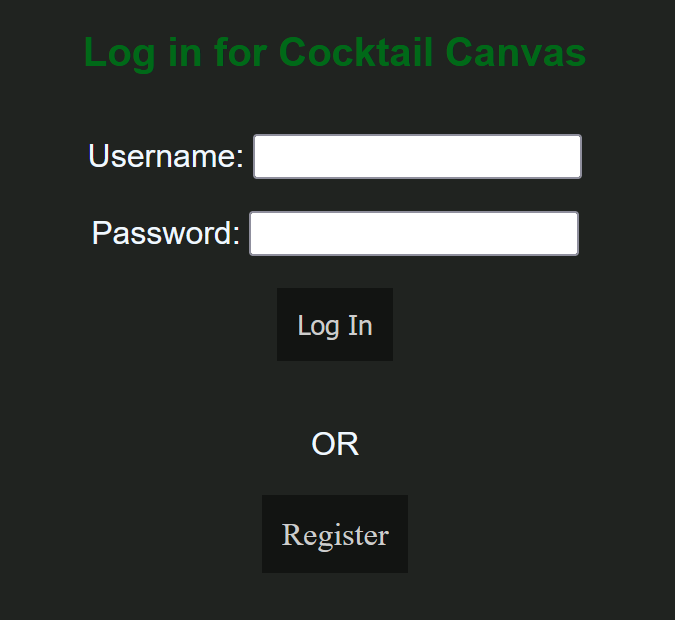
Everything comes together in 2 views, wholecocktails, and wholecocktails\_permenu.  
  
All Join tables are linked by foreign key relationships, which feature cascade delete and update rules – this allows for the removal of a menu, and subsequently the menudrinks entries linking drinks to that menu, and so on for each of the tables.

**Functionality:**

* A black and white logo

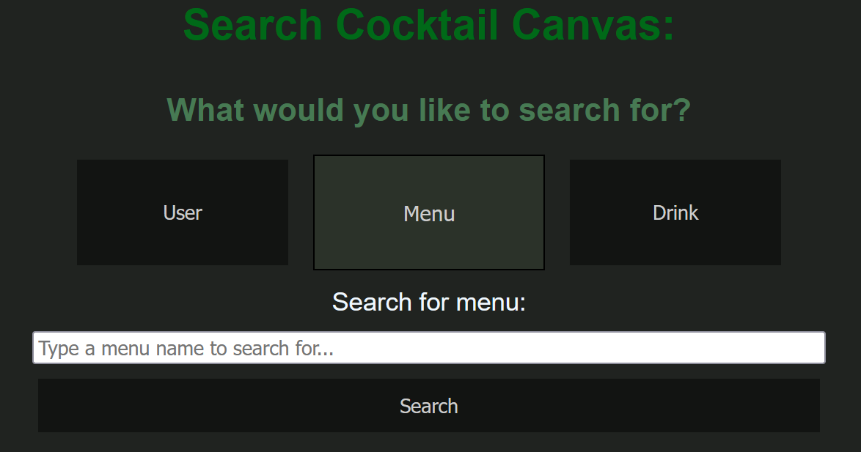
  Description automatically generatedA black and white logo

  Description automatically generatedA screenshot of a computer screen

  Description automatically generated**Login / Register**  
    
   ***Figure 3a****: Registration*  ***Figure 3c/d:*** *(login/logout nav)*  ***Figure 3b:*** *Login page*

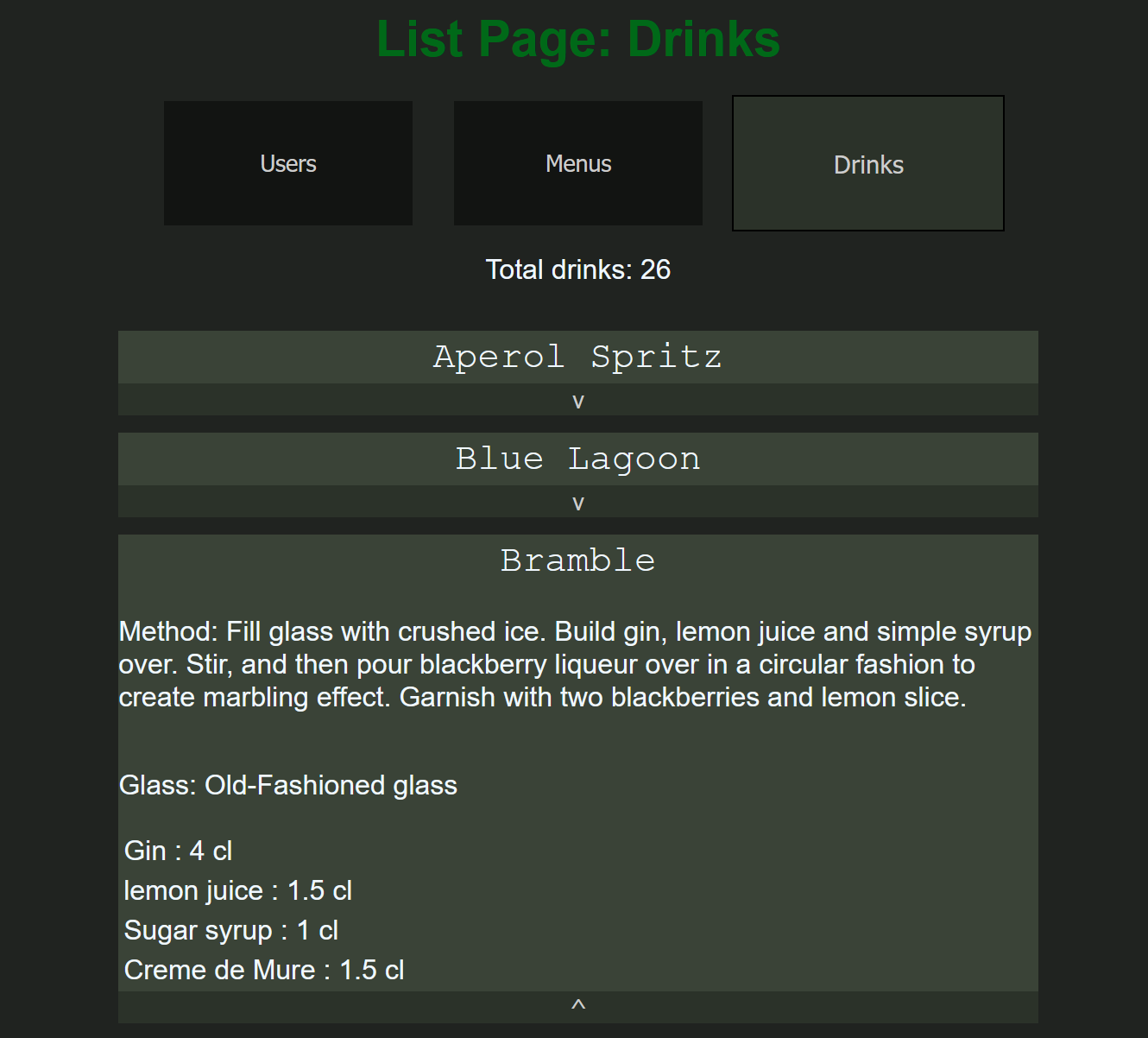
Users can login and logout via the top right corner of the navigation bar header **(Figures 3c and 3d).** The option will display Log in or Logout based on whether the user is currently logged in or not.  
**Figure 3b** shows the layout of the login page, and **Figure 3a** the registration page.  
Not all fields are mandatory for the registration, but feature data validation checks and hints for required fields.  
Some routes through the application are protected via authentication, and will redirect to the Log in page when a user tries to access them.  
A user can also only edit menus belonging to them, checked via session data.

* **Search / List / View Menus**

****

***Figure 4a****: Search page*  ***(****menu active****) Figure 4b:*** *Search results (drinks)*

The search page allows the user to swap between 3 search types (**figure 4a**), which dynamically updates the page with the relevant text and denotes which search is active.  
User and menu search will check against names for the relevant database tables, and drink search will pull all relevant rows from the ‘wholecocktails’ view, and collate them into an JS object with lists of ingredients and measurements.  
The list of drinks will be shown, and drinks can be expanded for viewing all information (**figure 4b**).

 A menu screen shot

Description automatically generated  
***Figure 5a****: List page (drinks active)* ***Figure 5b****: List page (menus active)*

The list page works much the same way as the search page, except all data is already loaded on page, and selecting one of 3 options will display the relevant list, as in **figure 5a** where drinks is selected, or **figure 5b** where menus is selected.

A screenshot of a menu

Description automatically generated

***Figure 6****: List page (menus active)*

Menus when displayed via lists or search can be clicked on to take the user to a view menu page, listing relevant drinks, expandable as always, and all associated menu information **(figure 6).**

* **Create / Edit Menus**

A screenshot of a menu

Description automatically generatedA screenshot of a menu

Description automatically generated  
***Figure 7****: Create menu page* ***Figure 8****: Edit menu list page*

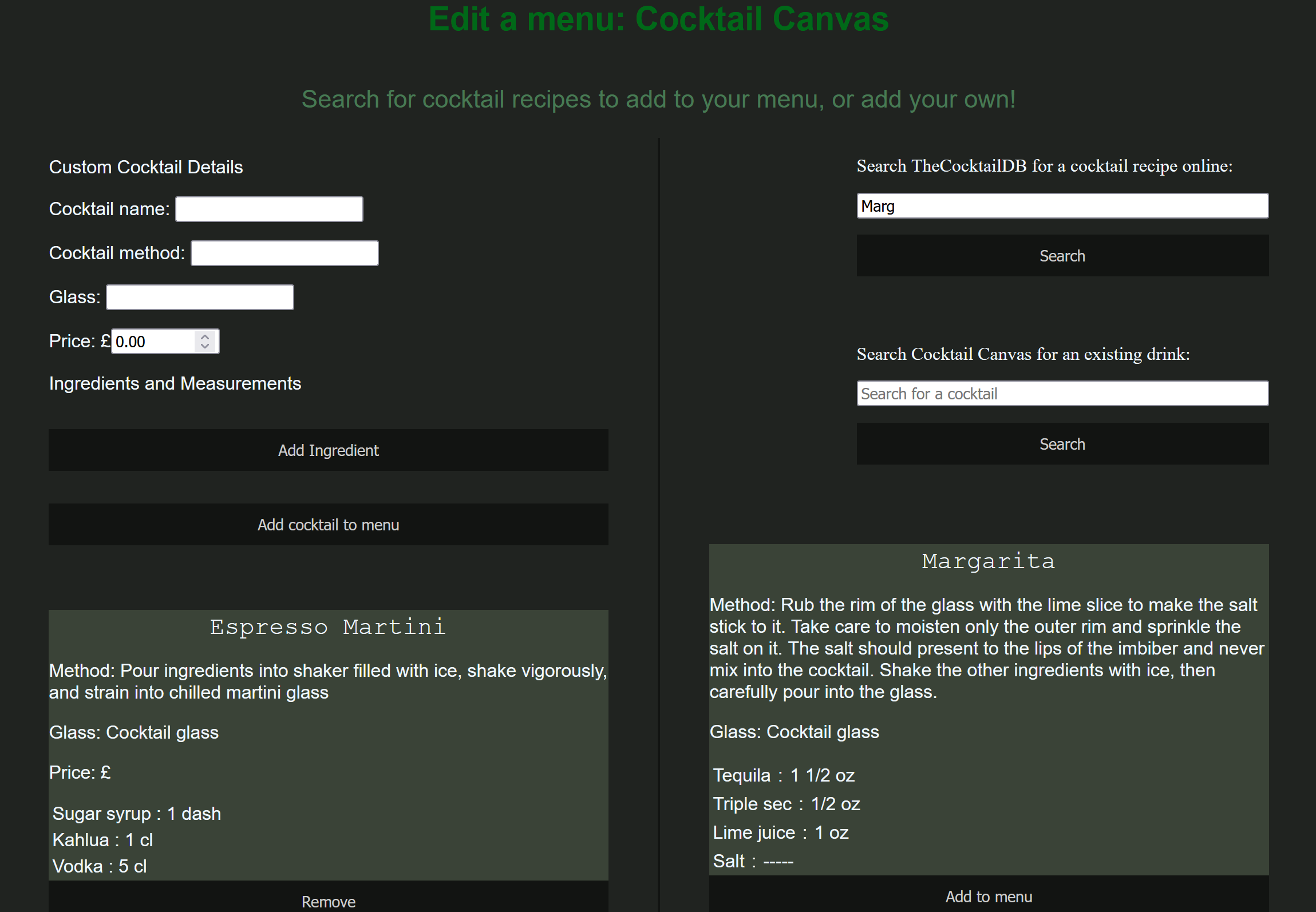
The process begins with the create menu page, which will create a menu with a name, description, and the user that created it **(figure 7)**.

The process then continues on the edit menu page, where users can choose any menus they have created to edit.

Lastly, **Figure 9** below shows the main editing menu page, most of the site centres on this page. The top left features a form for creating a custom drink and adding it to the menu.  
Below it on the bottom left features a list of drinks in the menu currently, and an option to remove each one.

The top right is a search form for TheCocktailDB’s API, and just below it, a search form to search the internal database for existing drinks added to menus.

On the bottom right is where search results are listed, with the option to add them to the current menu.

  
***Figure 9****: Editing menu page*

**Security:**

* **Bcrypt password hashing and salting**  
    
  Before a user can access and utilise the main functionality of the website, they must first register an account and log in.  
  When a user registers, before the password is stored in the database it is first encrypted using the Bcrypt node module, which is a function based on the blowfish cipher that hashes passwords. Passwords also undergo 10 rounds of salting, to eliminate the risk of rainbow tables being used to crack password hashes.
* **Authentication protected routes**  
  The main functionality of creating and editing menus or drinks is locked behind authentication, so that only users that have registered, and then authenticated against the stored password hashes in the database can input to / manipulate the database.  
  Once a user is logged in, they can create and edit menus, however through access control, they can only edit menus belonging to them.
* **Parameterised stored procedures**  
  In an effort to prevent SQL Injection – ALL queries from the node app to the database are done so using parameterised stored procedures, 34 in total **(figure 10)**. This is the case for user-facing routes and API routes.



***Figure 10****: Stored procedures*

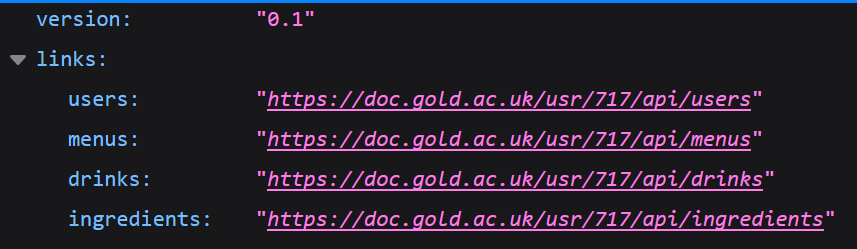
* **Data validation and sanitisation**  
    
  To both provide database integrity and prevent XSS attacks, where appropriate, route handlers will validate and sanitise inputs.  
  EJS also has inbuilt escaping, so that we can store strings in the database even if they would contain unescaped html characters, but when pulled from the database EJS will automatically escape them.  
  Input forms also have some checks for required attributes, as well as validating the format and lengths of input.

**API Usage:**

The application utilises TheCocktailDB’s API to search for recipes for users to then add to their menu, adding the drink, glass and ingredients for future usage in the Cocktail Canvas DB as well. The route used is: [www.thecocktaildb.com/api/json/v1/1/search.php?s=margarita].  
These results are displayed on the edit menu page, as users search and can add results to their own menu.

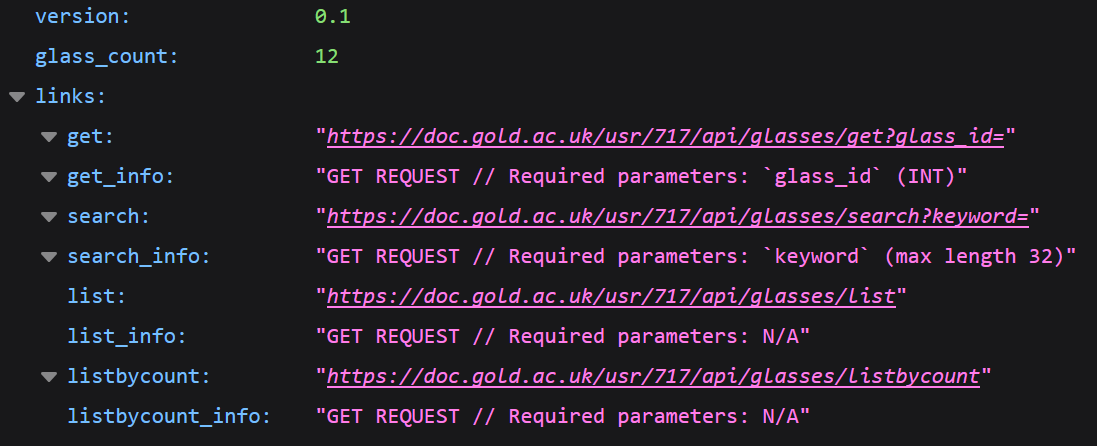
**API Provision:**

The API is served in a somewhat PURE REST style, whereby accessing <https://doc.gold.ac.uk/usr/717/api> will display the base of the api, version info, and links to the rest of the API, as seen in **figure 11.**



***Figure 11****:* [*https://doc.gold.ac.uk/usr/717/api*](https://doc.gold.ac.uk/usr/717/api)*, base API route*

Each route from this point is focused on one of the main tables of the database: users, menus, drinks, ingredients and glasses.  
Every path, for instance /api/users, is also a descriptive base page, with links to all the routes contained within, and information on how to use them.  
Each will supply 3 functions as a baseline – GET, SEARCH and LIST. Some tables provide more routes, such as api/users supplying a POST route for registering a user, and drinks providing a ‘listbycount’ route that lists drinks by how often they are used in menus. This is demonstrated in **figure 12.**



***Figure 12****:* [*https://doc.gold.ac.uk/usr/717/api/glasses*](https://doc.gold.ac.uk/usr/717/api/glasses)*, glasses API route*

**Advanced Techniques**

* **Multilayered (somewhat) PURE RESTFUL API**. Starting with a base /api route, all routes are listed with hyperlink, each leading to a section of the api for /users, /menus, /ingredients and /glasses.  
  Each section lists it’s own routes, and information on using them, featuring GET, SEARCH, LIST for all sections, and some bespoke routes for certain sections such as listing by counting or registering a user.  
  20 routes in total.  
  Opted to exclude authentication requiring routes from API for now.
* **Error handling** throughout route handling, JavaScript and EJS. Featuring a dedicated error page with dynamic error messages, and pages with dedicated dynamic error message sections.

A screen shot of a computer code

Description automatically generated A screenshot of a menu

Description automatically generated

***Figure 13a, b****: JS function dynamically displaying errors via EJS Page*

A computer screen shot of a program code

Description automatically generated

***Figure 13c****: Example of error handling within a route handler*

* **Duplicate checks**  for user registration **(figure 14a)**, and drink creation with detailed error messages and data validation. This means usernames must be unique, and when adding drinks to a menu – if it already exists in the menu, it wont be added again **(figure 14d)**, and the page will display a small error message letting the user know .  
  Lastly if the cocktail already exists in the database, it wont recreate a new identical entry, but rather pull the existing drink\_id and reference that **(figure 14b, c).**

A screen shot of a computer program

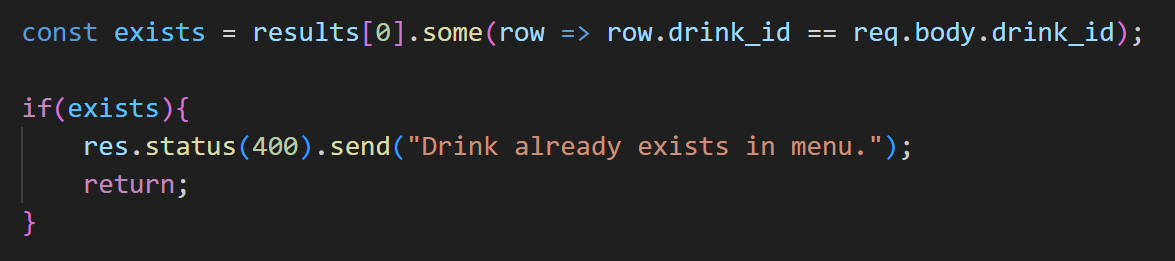
Description automatically generatedA computer screen shot of code

Description automatically generated***Figure 14a:*** *username check* ***Figure 14b:*** *using drink\_id if duplicate found*

A screen shot of a computer program

Description automatically generated

***Figure 14c****: Snippet of duplicateDrinkCheck()*



***Figure 14d****: Error handling of adding a drink already found in menu*

* **Entry suggestions** based on existing drinks within the database. As users create a custom drink, glasses and ingredients will be suggested based on typed input matching entries already in the database. This uses JavaScript functions and fetch requests to call our own API and reference the database. **(figure 15).**

A screen shot of a computer program

Description automatically generated

***Figure 15****: Dynamic input suggestion dropdown list from database (glasses)*

* **Complex SQL Query with transaction** for creating a drink, transforming object oriented structure of a drink (with lists of ingredients and measurements) into relational database structures and join tables.  
  Addition of a drink sits at 61 lines of SQL (with comments and spaces), and involves 4 tables and 3 join tables for linking them, as well as checks for existing entries to pull IDs rather than duplicate entries.   
  It will only commit if adding each part of a drink to each table and join table goes successfully, otherwise the transaction will fail. This procedure drives the engine of the application.

The full procedure is show below **(figure 16).**

A computer screen shot of a computer screen

Description automatically generated

A computer code on a white background

Description automatically generatedA screenshot of a computer code

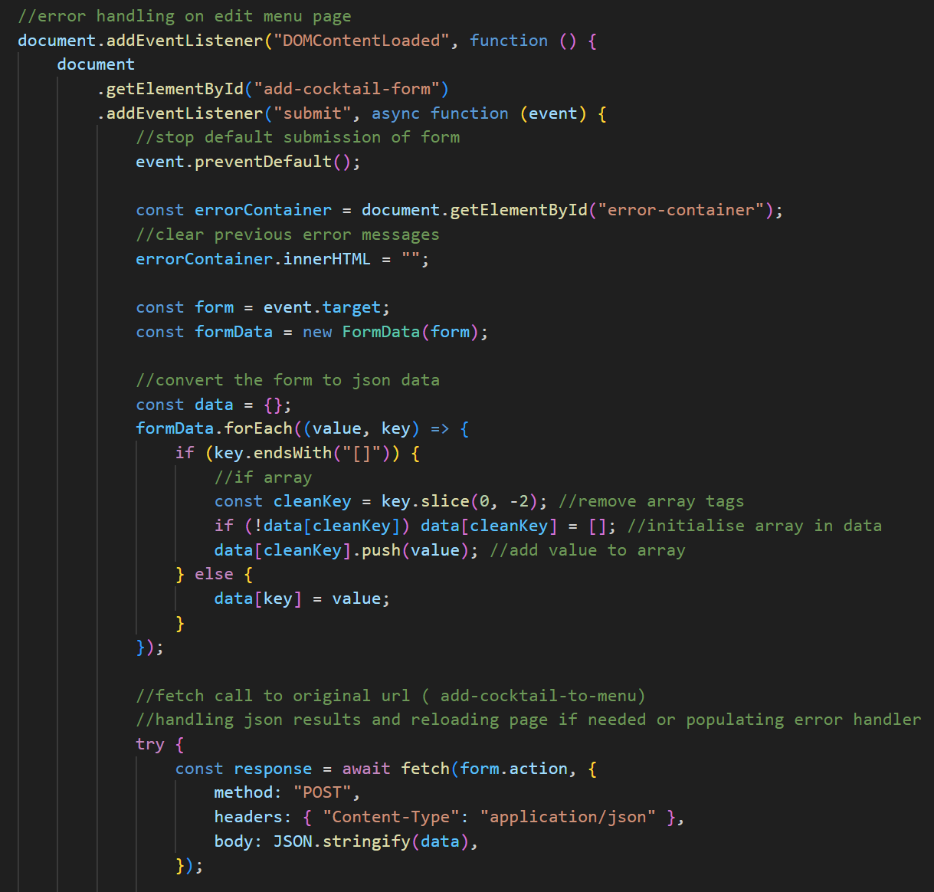
Description automatically generatedA white background with text

Description automatically generated with medium confidence

***Figure 16****: add\_drink\_to\_db MySQL procedure*

* **CSS Styling and Frontend JavaScript.**

A screen shot of a computer program

Description automatically generatedWhile not the focus of this course, there is 3 front-end java script files over 800 lines long collectively, and almost 500 lines of CSS **(figure 17a).**  
This helps to not only create a more friendly user interface, but also contributes to the core functionality of the website, as much of the error handling, and menu editing process will utilise forms called fetch functions and handling JSON responses from route handlers **(figure 17b).**

***Figure 17a****: Example CSS* ***Figure 17b:*** *Example of some JavaScript adding functionality through fetch calls to route handlers*