Web Application Report

COCKTAIL CAPTURE WEB APPLICATION

Gavin Blue
UNIVERSITY OF THE WEST OF SCOTLAND | B00420549

TABLE OF CONTENTS

INTRODUCTION

This report documents the development of a cocktail capture application that will allow users to create, read, update, and delete (CRUD) operations on a variable set of data. The application is built using React and is open-sourced allowing contributions from the wider community. The purpose of this application is to provide both professional and amateur bartenders to store their favourite cocktail recipes.

Access to the working application can also be found through the free domain name server account:

www.cocktail-dwt.duckdns.org

BACKGROUND

The cocktail capture application is designed to record the cocktail recipes of both novice and experienced bartenders. With the rise of home cocktail making during the pandemic, many people have discovered a passion for all things mixology and are eager to show of tier skills. (Mason, 2021) This app encourages conversation and creativity by providing an application that users can add their personalised recipes and showcase their mixology skills. As people adjust to life outside again and on premises, the app provides a convenient way to continue exploring the world of cocktails and sharing their creations with others.

FUNCTIONAL REQUIREMENTS

1. Allow users to add cocktail recipes to the local storage IndexedDb.
2. Add Persistent storage for cocktail data.
3. Allow users to optionally add location data for the cocktail recipe.
4. Provide an option to add an image of the cocktail.
5. Allow users to view and search for cocktail recipes.
6. Allow users to edit and delete their own cocktails.

NON-FUNCTIONAL REQUIREMENTS

1. The application should be responsive and work on different devices and screens.
2. The application should be accessible to users with disabilities.
3. The application should be easy to use and have clear navigation and instructions for the user.

IMPLEMENTATION

DuckDNS

DuckDNS was chosen as the domain name server for the application as it is a free and open source and allows users to assign a domain name to their public IP address. A developer signs up through email and then has the option of choosing their sub-domain. The developer then becomes the owner of [sub-domain].duckdns.org. For this project the sub-domain chosen is 'cocktail-dwt'. After signing up the developer is prompted to update the ip and ipv6 addresses of their sever. (DuckDNS, 2023)

Amazon Web Services

There are many open-source options for hosting the application, as the university provides access to Amazon Web Services, it will be utilised in this report, considerations were given to other hosting providers including Microsoft Azure, Digital Ocean, and Google Cloud.

These are the steps to set up the AWS instance and configure it to use the DuckDNS domain name server:

1. Sign up to Amazon Web Services.
2. Click the 'Launch Instance' button.
3. Select Ubuntu Amazon Machine Image.
4. Choose EC2 instance type.
5. Configure the security for the instance to allow http, https and SSH.
6. Launch instance and download 'pem' key-pair file.
7. Log into the instance locally using the key pair for SSH access through Windows PowerShell.
8. Update the environment using 'sudo apt-get update'.
9. Add the web server software Apache to AWS using the 'sudo apt install apache2' command.
10. Follow the instructions on the DuckDNS website that connects the instance to the domain name by adding the ip and ipv6 from the AWS instance.
11. Test that Apache loads when connecting to the duckDNS sub-domain.
12. Finally, after the development stage, follow the instructions on the apache homepage to the application is added to the var/www/html file before running the build.
13. The build folder is then moved one folder back, and hen the sub-domain is reloaded the application should load instead of the Apache homepage.

(AWS, 2023)

React

React is an open-source JavaScript library created by Facebook for building user interfaces for web applications. React is a component-based architecture that is easy to learn and use, declarative and unobtrusive. It is supported by an extensive and well-supported component library and NPM packages, which makes it easier for developers to build robust UI's. As the software is open source it has a rich, diverse, and active community, which has produced many useful libraries. React uses hooks which are functions that facilitate functional components to access the 'state', 'lifecycle', and 'context'. They were introduced to make it easier to reuse stateful logic between pages and components. There are many hooks, but this application utilises three. (React, 2023)

USESTATE

'useState' facilitates adding state to functional components, it returns an array with 2 elements, the current state value, and a function to update the state. (W3Schools, 2023) This hook is used several times throughout the application to updata the state of inputs and the checkboxes, without reloading the page or component.

The input boxes for the edit and add a cocktail page will be changed by the user, so the useState hook will be used to manage the state of these input boxes:

```
16  // state for the form elements
17  const [name, setName] = useState("");
18  const [method, setMethod] = useState("");
19  const [ice, setIce] = useState("");
20  const [garnish, setGarnish] = useState("");
21  const [glassware, setGlassware] = useState("");
22  // state for the ingredients array
23  const [ingredients, setIngredients] = useState([]);
24  // state for the imageSrc
25  const [imageSrc, setImageSrc] = useState(null);
26  // state for the cocktail object
27  const [cocktail, setCocktail] = useState(null);
```

The form elements are set with an empty string, and array is created for the ingredients, the image and cocktail object are set with null values.

```
79
80  // functions to handle the form elements
81  const handleNameChange = (event) => {
82  | setName(event.target.value);
83  };
84
85  const handleMethodChange = (event) => {
86  | setMethod(event.target.value);
87  };
88
89  const handleIceChange = (event) => {
90  | setIce(event.target.value);
91  };
92
93  const handleGarnishChange = (event) => {
94  | setGarnish(event.target.value);
95  };
96
97  const handleGlasswareChange = (event) => {
98  | setGlassware(event.target.value);
99  };
100
```

The form elements are then handled in a function so that any changes made by the user to the input fields are recorded.

```
102 const handleIngredientNameChange = (event, index) => {
103 | const newIngredients = [...ingredients];
104 | newIngredients[index].name = event.target.value;
105 | setIngredients(newIngredients);
106 };
107
108 const handleIngredientQuantityChange = (event, index) => {
109 | const newIngredients = [...ingredients];
110 | newIngredients[index].quantity = event.target.value;
111 | setIngredients(newIngredients);
112 };
113
114 const handleAddIngredient = () => {
115 | setIngredients([...ingredients, { name: "", quantity: "" }]);
116 };
117
118 const handleDeleteIngredient = (index) => {
119 | const newIngredients = [...ingredients];
120 | newIngredients.splice(index, 1);
121 | setIngredients(newIngredients);
```

Each cocktail can have many ingredients, so logic was created that allows the user to add or remove ingredients from the cocktail recipe, each ingredient has a quantity and name associated with each entry in the array of ingredients.

The ‘CocktailDashboard’ page also uses the UseState to set the cocktail to the cocktail recipe card.

USEEFFECT

‘UseEffect’ facilitates side effects in the functional components. A side effect is an action that affects something outside of the scope of the function that is being executed. It is used when the application needs to do something other than simply rendering such as fetching data from an API. (W3Schools, 2023)

In the cocktail capture application ‘UseEffect’ is used to fetch the cocktail data from the IndexedDB in both the EditACocktail and the CocktailDashboard page.

```
14    // useEffect hook that runs when the component is mounted
15    useEffect(() => {
16      // get the cocktail from the database with the key from the cocktail state
17      get(key).then((cocktail) => {
18        // set the cocktail state to the cocktail from the database
19        setCocktail(cocktail);
20      });
21    }, [key]);
```

In the ‘PersonalCocktailTable’ the ‘useEffect’ hook is used to get all the cocktails stored in the database:

```
19    // useEffect hook that runs a function when the component is mounted
20    useEffect(() => {
21      // get all cocktails from the database
22      getAll().then((cocktails) => {
23        // set the cocktails state to the cocktails from the database
24        setCocktails(cocktails);
25      });
26    }, []);
```

USEREF

There were issues trying to connect to the users camera directly, an investigation into suitable libraries found that ‘react-webcam’ could be used but it would been to use the useRef hook, which allows you to persist values between renders. (W3Schools, 2022)

Create-react-app

This project utilised ‘create-react-app’ CLI tool, which provides a quick and easy way to set up a new React project with a pre-configured development environment. The technologies used include Babel, Webpack, ESLint amongst others. Developers are not restricted to any of these settings as they can modify React through the config files. (Create React App, 2023)

It uses these technologies to build a fully functional React project that is optimised for productivity and ease of use. The generated project includes files and directories that are essential for building a react application:

- ‘index.html’ – The main HTML file that serves as the entry point for the application.

- ‘index.js’ – the main javascrit file that initialises the React application and renders the top-level component.
- ‘App.js’ – A sample React component that serves as the root for the application.
- ‘app.css’ – Styling for the application.
- ‘src/’ directory – Where most of the application is developed, for many projects this will include pages, components, stylesheets, images and any local storage logic.

(Create React App, 2023)

The project structure for our application will also include the backend server code, this would usually be in a folder separate from the client-side code but has been included here as part of the client-side ‘src’ folder for consideration.

The folder structure employed allows for readability and scalability, the applications main page are kept in the pages folder, components rendered by pages are stored in the components folder and styling for the components are stored in the stylesheets. The project folder is shown in figure 1 and the structure is as follows:

- Src
    - App.js
    - App.css
    - index.js
    - indexDB.js
    - Pages
        - Home.jsx
        - ViewCocktail.jsx
        - EditCocktail.jsx
        - AddCocktail.jsx
        - CocktailList.jsx
    - Components
        - Header.jsx
        - AddACocktail.jsx
        - CocktailDashboard.jsx
        - EditACocktail.jsx
        - PersonalCocktailTable.jsx
        - GlobalCocktailTable.jsx
    - Stylesheets
        - header.css
        - home.css
        - addACocktail.css
        - cocktailDashboard.css
        - editACocktail.css
        - personalCocktailTable.css
        - globalCocktailTable.css

APPLICATION PROGRAMMABLE INTERFACES (API)

IndexedDB

IndexedDb is a low-level JavaScript API that allows client-side storage of structured data including files/blobs and provides high performance searches through indexes. This makes it suitable for this application as there could be many cocktails saved with image data attached, by keeping them all within the one object there can be no instances where images are stored on the system that is not associated with a cocktail id.

To configure the IndexedDB, the MDN docs recommends that the application creates a Promise that resolves with the database instance when it opens and uses the Promise to interact with the database in a consistent and predictable way.

The window.indexDb.open() method is used to create or open a database with the name ‘cocktails’ and version number. The ‘onerror’ and ‘onsuccess’ event handlers are used hadne erros and successful openings of the database, respectively.

When the database is opened the ‘db’ variable is assigned to the database instance, and the Promise is resolved with that instance.

(Mozilla, 2023)

```
// client side Database for storing cocktails
let db;

//  promise for opening the indexDB
let dbPromise = new Promise((resolve, reject) => {
  // request indexDB for storing cocktails locally
  const request = window.indexedDB.open("cocktails", 1);
  // check if the indexDB is already open
  request.onerror = function(event) {
    console.error(`Database error: ${event.target.errorCode}`);
    reject(event.target.errorCode);
  };
  // check if the indexDB is already open
  request.onsuccess = function(event) {
    console.log("indexDB opened successfully");
    // Save the IDBDatabase interface
    db = event.target.result;
    // resolve the promise
    resolve(db);
  };
});
```

CRUD Operations

ADD A COCKTAIL

IndexDB.js

In the indexDB.js file a function called ‘add’ handles the adding of cocktails to the database, it checks if the database is already open with a Promise, which is created during the loading of the database. If the database is not yet open, it will wait until it is before proceeding.

```
37  // add a cocktail to the indexDB
38  function add(cocktail) {
39    // check if the indexDB is already open
40    return dbPromise.then(() => {
41      // return a promise
42      return new Promise((resolve, reject) => {
43        // open a transaction
44        const transaction = db.transaction(["cocktails"], "readwrite");
45        // open the object store
46        const objectStore = transaction.objectStore("cocktails");
47        // add the cocktail to the object store
48        const request = objectStore.add(cocktail);
49        // check if the cocktail was added successfully
50        request.onsuccess = () => {
51          console.log("cocktail added in indexDB");
52          // resolve the promise
53          resolve(request.result);
54        };
55        // check if the cocktail was added successfully
56        request.onerror = () => {
57          // reject the promise
58          reject(request.error);
59        };
60      });
61    });
62  }
```

A new Promise is then created that is resolved when the cocktail is added to the object store. This promise is returned by the function so it can be used to handle operations elsewhere in the application.

Inside the promise the code opens a transaction on the ‘cocktails’ object store with read-write access so that the transaction can make changes to the object store. (Mozilla, 2023)

The cocktail is then added to the store using the objectStore.add() method. If the request is successful then the promise is resolved, if there are any errs the Promise is rejected with the error.

AddACocktail.jsx

The component AddACocktail uses the add function from indexDb.js file:

```
4   // import the add function from the indexDB.js file
5   import {add} from "../indexDB.js";
```

The add function is then used in the handleSubmit function, that is used to handle the from submission:

```
// handle the form submission
const handleSubmit = async (event) => {
  // prevent the default form submission
  // stops other buttons from submitting the form
  event.preventDefault();
  // create a cocktail object
  const cocktail = {name, method, ice,garnish,glassware, ingredients};
  // add the location if the checkbox is checked
  if (includelocation) {
    // add the location to the cocktail object
    cocktail.location = { latitude: location.latitude, longitude: location.longitude };
  };
  // create a unique key for the cocktail
  cocktail.myKey = uuidv4() ;
  // add the image to the cocktail object
  cocktail.imageSrc = imageSrc;
  // try to add the cocktail to the database
  try {
    // add the cocktail to the database
    add(cocktail);
    // navigate to the CocktailDashboard page with the cocktail key
    navigate(`/cocktail/${cocktail.myKey}`);
  }
  // catch any errors
  catch (error) {
    console.error(error);
  }
};
```

The first line is added as there are other buttons within the form that can cause the form to submit prematurely, by disabling the default behaviour and adding type=button to other buttons within the form, the DOM will ignore them as calls to submit the form.

The cocktail object is then created using the various properties that are needed to successfully create a cocktail.

The additional information gathered here is explained elsewhere in this report.

After this data has been gathered the ‘handleSubmit’ function tries to add the cocktail to the ‘cocktials’ database and navigates to the cocktail dashboard with the newly created cocktail object id ’myKey’.

DELETE A COCKTAIL

IndexDB.js

A function to handle the deletion of a cocktail was also created called ‘remove’, which deletes the cocktail object using the ‘myKey’ uuid which is sent in the request.

```
64  // delete a cocktail from the indexDB
65  function remove(key) {
66    // check if the indexDB is already open
67    return dbPromise.then(() => {
68      // return a promise
69      return new Promise((resolve, reject) => {
70        // open a transaction
71        const transaction = db.transaction(["cocktails"], "readwrite");
72        // open the object store
73        const objectStore = transaction.objectStore("cocktails");
74        // delete the cocktail from the object store
75        const request = objectStore.delete(key);
76        // check if the cocktail was deleted successfully
77        request.onsuccess = () => {
78          // resolve the promise
79          resolve(request.result);
80        };
81        request.onerror = () => {
82          // reject the promise
83          reject(request.error);
84        };
85      });
86    });
87  }
```

The function firs checks if the indexDb is already open using the dbPromise, when the connection s established the function returns a new promise that resolves or rejects based on the result of the delete operation. The delete function then follows the same design pattern as the add function, except that it deletes the cocktail instead of adding it.

EditACocktail.jsx

The cocktail dashboard page allows a user to edit or delete a cocktail, the ‘remove’ function is added and then used in the handleDelete function:

```
4   import { get, editCocktail, remove } from "../indexDB.js";
5
```

```
113   const handleDelete = () => {
114     remove(cocktail.myKey);
115     navigate("/View");
116   };
```

The ‘remove’ function is called with the ‘cocktail.myKey’ uuid and then the user is redirected to the list of cocktails in the ViewCocktail page.

UPDATE A COCKTAIL

IndexDB.js

The function to edit a cocktail follows the same design pattern as the ‘add’ and ‘delete’ functions, with the function accepting a ‘cocktail’ object like the ‘add’ function, where ‘delete’ only needs the ‘myKey’ uuid.

```
89  // edit a cocktail in the indexDB
90  function editCocktail(cocktail) {
91    // check if the indexDB is already open
92    return dbPromise.then(() => {
93      // return a promise
94      return new Promise((resolve, reject) => {
95        // open a transaction
96        const transaction = db.transaction(["cocktails"], "readwrite");
97        // open the object store
98        const objectStore = transaction.objectStore("cocktails");
99        //  edit the cocktail in the object store
100       const request = objectStore.put(cocktail);
101       // check if the cocktail was edited successfully
102       request.onsuccess = () => {
103         // resolve the promise
104         resolve(request.result);
105       };
106       request.onerror = () => {
107         // reject the promise
108         reject(request.error);
109       };
110     });
111   });
112 }
```

EditAcocktail.jsx

The import used is the same import expression as the remove function from before, the 'handleSubmit' function is written as in the add a cocktail 'handleSubmit' except instead of calling the 'add' function, the cocktail is sent to the 'editCocktail' function:

```
103   // handle the form submission
104   const handleSubmit = (event) => {
105     // prevent the default form submission behavior
106     // stops other buttons from sending the form
107     event.preventDefault();
108     // create a new cocktail object
109     const editedCocktail = {
110       ...cocktail,
111       name,
112       method,
113       ice,
114       garnish,
115       glassware,
116       ingredients,
117       imageSrc
118     };
119     // call the editCocktail function
120     editCocktail(editedCocktail);
121     // navigate to the View page
122     navigate(`/cocktail/${cocktail.myKey}`);
123   };
```

READ A COCKTAIL

There are two reasons why a read operation would be used for the cocktail capture application, when the 'PersonalCocktailTable' component needs the full list of the stored cocktails, and when either the 'EditACocktail' or the 'CocktailDashboard' need a single cocktail object.

indexDB.js

The two ways that the user needs to read the cocktails from the database are implemented using the 'get' and 'getAll' functions. The 'get' function takes a 'key' from the request and uses it to retrieve the cocktail object, the 'getAll' function returns all cocktail objects.

```
114  // get all cocktails from the indexDB
115  function getAll() {
116    // check if the indexDB is already open
117    return dbPromise.then(() => {
118      // return a promise
119      return new Promise((resolve, reject) => {
120        // open a transaction
121        const transaction = db.transaction(["cocktails"], "readonly");
122        // open the object store
123        const objectStore = transaction.objectStore("cocktails");
124        // get all cocktails from the object store
125        const request = objectStore.getAll();
126        // check if the cocktails were retrieved successfully
127        request.onsuccess = () => {
128          // resolve the promise
129          resolve(request.result);
130        };
131        request.onerror = () => {
132          // reject the promise
133          reject(request.error);
134        };
135      });
136    });
137  }
```

```
139  // get a cocktail from the indexDB
140  function get(key) {
141    // check if the indexDB is already open
142    return dbPromise.then(() => {
143      // return a promise
144      return new Promise((resolve, reject) => {
145        // open a transaction
146        const transaction = db.transaction(["cocktails"], "readonly");
147        // open the object store
148        const objectStore = transaction.objectStore("cocktails");
149        // get the cocktail from the object store
150        const request = objectStore.get(key);
151        // check if the cocktail was retrieved successfully
152        request.onsuccess = () => {
153          // resolve the promise
154          resolve(request.result);
155        };
156        request.onerror = () => {
157          // reject the promise
158          reject(request.error);
159        };
160      });
161    });
162  }
```

PersonalCocktailTable.jsx

The locally saved cocktails are loaded through the ‘PersonalCocktailTable using the useEffect hook, this hook loads when the component is first mounted, it uses the ‘getAll’ finction in the indexDD.js file to get all the cocktails and adds the cocktail objects to the ‘setCocktails’ array.

```
12      // cocktails state that is initialized to an empty array
13      const [cocktails, setCocktails] = React.useState([]);
14
15
16      // useEffect hook that runs a function when the component is mounted
17      useEffect(() => {
18          // get all cocktails from the database
19          getAll().then((cocktails) => {
20              // set the cocktails state to the cocktails from the database
21              setCocktails(cocktails);
22          });
23      }, []);
24
```

The application then uses the ‘setCocktails’ array to populate the cocktails table if there is nothing in the search term, if there is something in the search bar the list of cocktails is filtered by the search term.

```
return (
  <div className="cocktails-container">
    <h2 className='cocktail-title'>Shared Cocktails</h2>
    <input type="text" placeholder="Search cocktails" value={searchTerm} onChange={(e) => setSearchTerm(e.target.value)} className="search-input"/>
    <div className="cocktail-list">
      {filteredCocktails.map((cocktail) => (
        <div key={cocktail.myKey} className="cocktail-window">
          <div className="cocktail-image-table">
            {cocktail.imageSrc ? (
              <img src={cocktail.imageSrc} className="image-icon" alt={cocktail.name} />
            ) : (
              <FontAwesomeIcon icon={faImage} className="image-icon" />
            )}
          </div>
          <div className="cocktail-content">
            <h3>{cocktail.name}</h3>
            {expandedCocktail === cocktail ? (
              <div>
                <p>To make {cocktail.name}, you need: {`${cocktail.ingredients.slice(0, -1).map(ingredient => ingredient.name).join(", ")} and ${cocktail.ingredients.slice(-1)[0].name}`}</p>
                <button onClick={handleCollapse}>Hide Description</button>
                <button>
                  <Link to={`/cocktail/${cocktail.myKey}`}>View Recipe</Link>
                </button>
                <button>
                  <Link to={`/edit/${cocktail.myKey}`}>Edit Recipe</Link>
                </button>
              </div>
```

If the capitalisation is not followed then cocktails can be missed from the search, so an arrow function is used to make the cocktails names case insensitive.

```
32      // arrow function that returns a filtered array of cocktails
33      const filteredCocktails = cocktails.filter((cocktail) =>
34      // check if the cocktail name includes the search term (case insensitive)
35      cocktail.name.toLowerCase().includes(searchTerm.toLowerCase())
36      );
```

CocktailDashboard.jsx

When the ‘ViewCocktail’ page mounts the ‘CocktailDashboard’ component it uses the useEffect hook to get the cocktail object form the cocktail state, it then sets the retrieved cocktail for the component to use.

```
// useEffect hook that runs when the component is mounted
useEffect(() => {
  // get the cocktail from the database with the key from the cocktail state
    get(key).then((cocktail) => {
      // set the cocktail state to the cocktail from the database
        setCocktail(cocktail);
    });
}, [key]);
```

The cocktail is then displayed as a recipe card that gives the full description of the cocktail.

```
return (
  <div className="recipe-card">
    <div className="cocktail-recipe-header">
      {cocktail.imageSrc ? (
        <img className="cocktail-image" src={cocktail.imageSrc} alt="A capture of a delicious cocktail" />
      ) : (
        <FontAwesomeIcon icon={faImage} style={{ fontSize: "200px" }} />
      )}
      <h2 className="cocktail-header">{cocktail.name}</h2>
    </div>
    <hr />
    <p>Method: {cocktail.method}</p>
    <hr />
    <p>Ice: {cocktail.ice}</p>
    <hr />
    <p>Garnish: {cocktail.garnish}</p>
    <hr />
    <p>Glassware: {cocktail.glassware}</p>
    <hr />
    <p>
      Ingredients:{" "}
      {cocktail.ingredients.map((ingredient) => (
        <li>
          {ingredient.quantity} {ingredient.name}
        </li>
      ))}
    </p>
    <hr/>
    <div className="action-buttons">
      <Link to="/view" className="button">Back</Link>
      <Link to={`/edit/${key}`} className="button">Edit</Link>
    </div>
  </div>
);
    };
export default CocktailDashboard;
```

Storage Manager API

Although the IndexedDB is a powerful client-side API that facilitates the storage of a wide range of data, without some form of permanent storage the user's data will be lost whenever the browser is closed or if the session expires. To address this the MDN docs have several recommendations including Storage Manager API which works with IndexedDB to add permanent storage on browsers that permit it and the user gives permission for.

(Mozilla, 2023)

```
// check if the browser supports persisting data
if (navigator.storage && navigator.storage.persist) {
  // request permission to persist data
  navigator.storage.persist().then((persistent) => {
    // check if the permission was granted
    if (persistent) {
      console.log("Storage will not be cleared except by explicit user action");
    } else {
      console.log("Storage may be cleared by the UA under storage pressure.");
    }
  });
}
```

The file checks if the browser supports the API if it does then, it then returns a promise that resolves to a Boolean value indicating whether the permission to persist data was granted. It then sends a message to the console informing the user if it was successful or not.

React-Webcam

As cocktails are known to be visually pleasing, users of the application may want to take pictures of the cocktail, to achieve this the popular React library 'react-webcam' was utilised. Although this is a library on its own, it uses the WebRTC 's API which is provided by the browser to access the users webcam/ front facing camera. The WebRTC API proved to be difficult to work with directly and so 'react-webcam' was installed using 'npm install react-webcam'. (NPMJS, 2022)

The functionality is used two times in the project, 'EditACocktail' and the 'AddACocktail' components, the functionality is the same in each so only one is described:

```
31    // reference to the webcam
32    const webcamRef = useRef(null);
33    // state for the camera
34    const [showCamera, setShowCamera] = useState(false);
35    // state for the camera capture to be open
36    const [cameraOpen, setCameraOpen] = useState(false);
37
38  // arrow function to handle the camera capture
39    const handleCapture = () => {
40      // get the image from the webcam
41      const imageSrc = webcamRef.current.getScreenshot();
42      if (imageSrc) {
43        console.log('Captured Image: ', imageSrc);
44        // set the imageSrc state
45        setImageSrc(imageSrc);
46      }
47    };
48    // arrow function to toggle the camera on and off
49    const toggleCamera = () => {
50      // set the showCamera state to the opposite of what it is
51      setShowCamera(!showCamera);
52      // set the cameraOpen state to the opposite of what it is
53      setCameraOpen(!cameraOpen);
54    };
```

Following the npmjs documentation for the 'react-webcam' capture the handle capture function takes a screenshot of the webcam or front facing camera using the 'getScreenshot()' function (Anon., 2022) The

toggle camera function allows the user to open or close the webcam and the image capture window that is displayed.

```
<>
<img className = "cocktail-image" src={imageSrc} alt="Cocktail image"/>
<button type="button" onClick={() => setShowCamera(true)}>Take Photo</button>
{showCamera && (
  <div>
    <Webcam
      className="webcam"
      audio={false}
      ref={webcamRef}
      screenshotFormat="image/jpeg"
      videoConstraints={{
        width: 200,
        height: 200,
        facingMode: "user",
      }}
    />
    <button type="button" onClick={handleCapture}>Capture</button>
  </div>
)}
</>
```

Navigator API

The Navigator API allows scripts to query it and to register themselves to carry out activities. It can be accessed using the window.navigator property. the APi provides access to many features, in the application it is used to gather the geolocation data for the device at the time the cocktail is added.

(Mozilla, 2023)

The user may be in a cocktail bar and would like to document the location that they enjoyed the cocktail, on the other hand they may be at home and do not want to record the location of where the recipe and photo were taken.

```
24  // use state to store the location and the checkbox value
25  const [location, setLocation] = useState({latitude:null, longitude:null});
26  const [includelocation, setIncludelocation] = useState(false);
```

To handle both scenarios the location and the include location were set up using ‘useState’ to handle these changes.

```
43  // get the current location
44  useEffect(() => {
45    // check if the browser supports the geolocation API
46    navigator.geolocation.getCurrentPosition((position) => {
47      // set the location state
48      setLocation({
49        // set the latitude and longitude
50        latitude: position.coords.latitude,
51        longitude: position.coords.longitude,
52      });
53    });
54  }, []);
```

The ‘useEffect’ hook was used to check if the browser supports the navigator API, if it does then it fetches the latitude and longitude. The user is given the choice when first opening the browser if they want to share the location. The user still has the option of adding the location data through a checkbox.

```
<form onSubmit={handleSubmit}>
<div className= "geolocation">
    {location.latitude && location.longitude ? (
      <h3>  Your location is: {location.latitude}, {location.longitude} </h3>
    ) : (
      <p>Getting your location...</p>
    )}
    <label htmlFor="includelocation">Include Location:</label>
    <input type="checkbox" id="includelocation" name="includelocation" checked={includelocation} onChange={handleLocationChange} />
    <br />
```

If the checkbox is empty, it is not added to the cocktail.

```
// handle the form submission
const handleSubmit = async (event) => {
  // prevent the default form submission
  // stops other buttons from submitting the form
  event.preventDefault();
  // create a cocktail object
  const cocktail = {name, method, ice,garnish,glassware, ingredients};
  // add the location if the checkbox is checked
  if (includelocation) {
    // add the location to the cocktail object
    cocktail.location = { latitude: location.latitude, longitude: location.longitude };
  };
  // create a unique key for the cocktail
  cocktail.myKey = uuidv4() ;
  // add the image to the cocktail object
  cocktail.imageSrc = imageSrc;
  // try to add the cocktail to the database
  try {
    // add the cocktail to the database
    add(cocktail);
    // navigate to the CocktailDashboard page with the cocktail key
    navigate(`/cocktail/${cocktail.myKey}`);
  }
  // catch any errors
  catch (error) {
    console.error(error);
  }
};
```

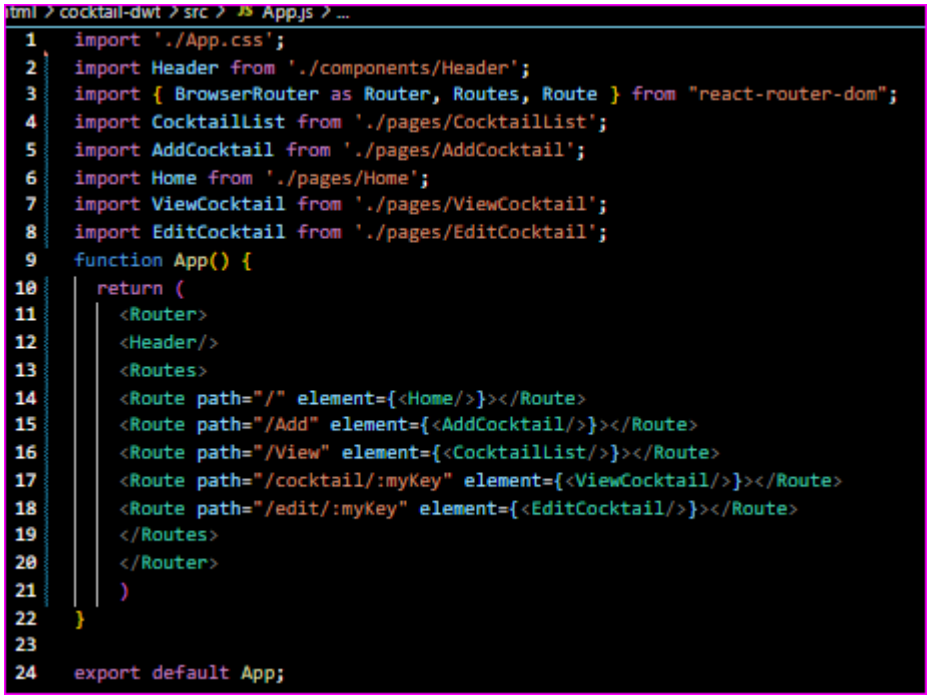
OTHER JAVASCRIPT LIBRARIES

React-Router-Dom

‘react-router-dom’ is an npm package that allows a developer to implement dynamic routing in a web application. It is installed using the ‘npm instal react-router-dom’ command. In the cocktail capture application BrowserRouter and useNavigate were implemented. (React Router, 2023)

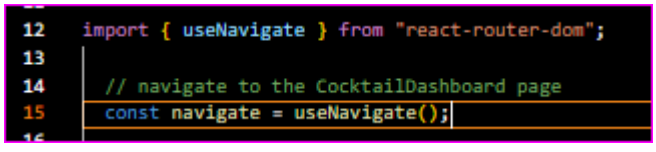
BROWSERROUTER

'BrowserRouter' is a component that creates a browser history object and provides it all the child components. In the cocktail capture application BrowserRouter is imported as Router, Routes, and Route. The routes component is used to wrap all the route components, which each define a different route for the application. Each Route then has a path that specifies the URL path for that route and an element that specifies the url path for that route. (Ail, 2020)

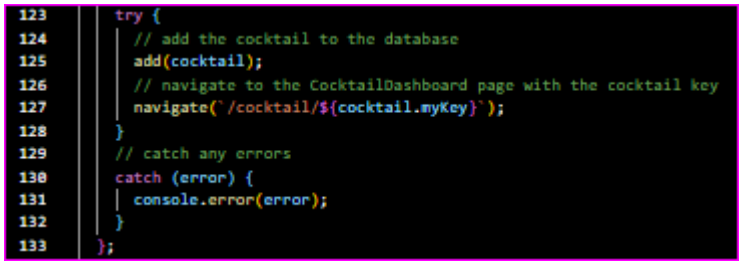


USENAVIGATE

'useNavigate' is a hook that returns a function that can be used to navigate to different routes in the application.



It is used in the edit and add a cocktail page to automatically navigate to a different page after the cocktail has been added or updated.



```
    try {
      // add the cocktail to the database
      add(cocktail);
      // navigate to the CocktailDashboard page with the cocktail key
      navigate(`/cocktail/${cocktail.myKey}`);
    }
    // catch any errors
    catch (error) {
      console.error(error);
    }
  };
```

UUID

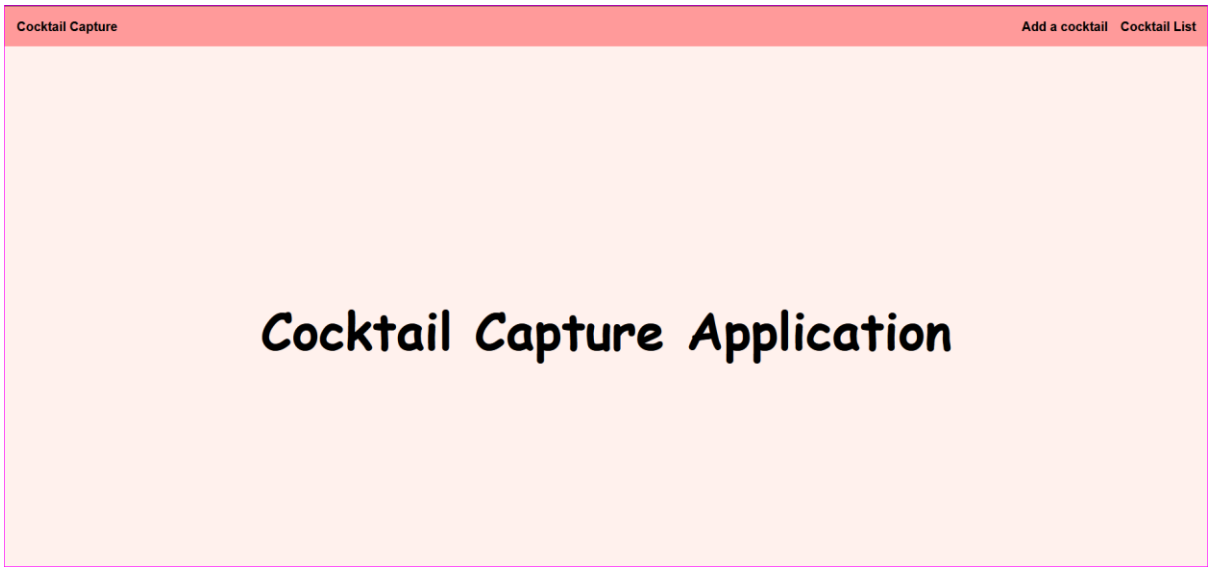
As there are many recipes for the same cocktail, using the cocktail name as the id would not be suitable, at first a simple integer id was considered, but the implementation was deemed unsuitable. Instead, a unique id for each cocktail would be created and used to call the individual cocktail from the cocktails database. The ‘uuid’ package was installed using the ‘npm install uuid’ command, then version 4 was used as this creates a random UUID. This ensures that no cocktails have the same id and stops any errors happening during retrieval of data from the database. (npmjs, 2022)

MERN

Not included in this report was the use of Mongoose, express and node API’s to create a connection to a MongoDb, if the user is connected to the internet and the server is running, they can access a global set of cocktails. The functionality for the crud operations were created, but user use was out of the scope of the project. The user can view the cocktails that are already added to the mongoDb by selecting the global table option in the view cocktails page. (Media, 2023)

USER INTERFACE

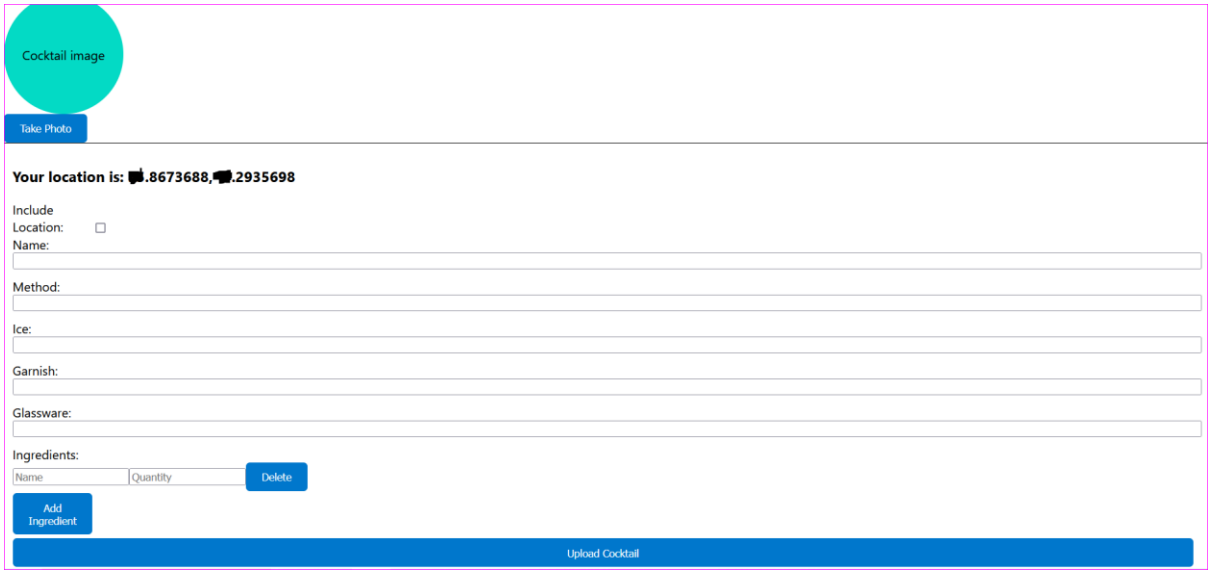
Home Page



The cocktail capture application is designed to work both on mobile and desktop, when the user first opens the application or visits the site they are welcomed by a home screen, there is no functionality added to this screen and is something that could be worked on in the future, the page is styled with css to be centred in the middle of the screen with a pink background.

```
.home-container {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 100vh;
    background-color: #FFF1ED;
}

.home-title {
    font-size: 4rem;
    font-family: cursive;
    text-align: center;
}
```

Add A Cocktail



This page has been designed in a way that makes it easy for the user to see what information is needed, the buttons were chosen to be blue to stand out from the white background of the form.

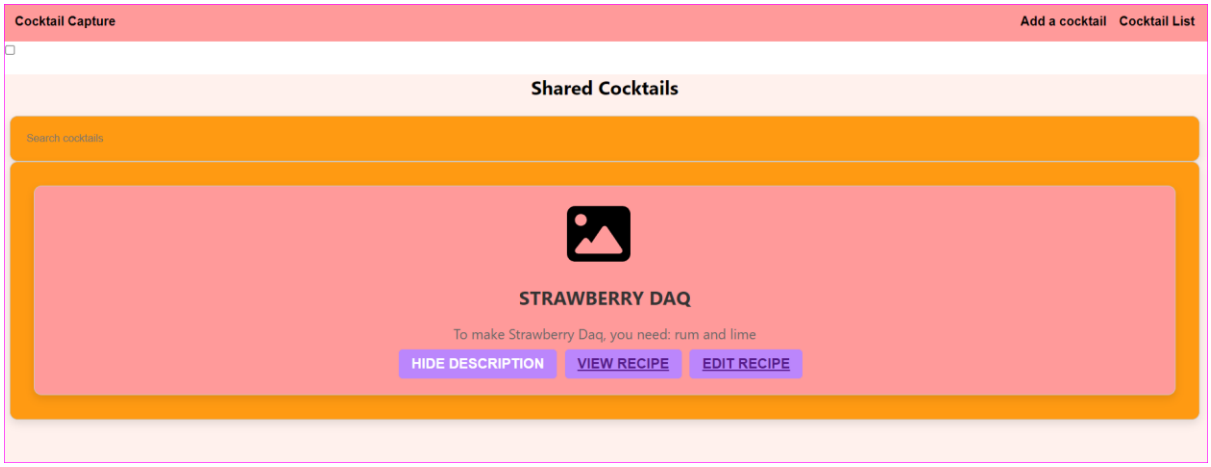
```css
/* Button CSS */
button {
  display: inline-block;
  background-color: #0077cc;
  color: #ffffff;
  border: none;
  border-radius: 5px;
  padding: 10px 20px;
  margin-right: 10px;
  cursor: pointer;
  transition: all 0.2s ease-in-out;
  text-decoration: none;
}

button:hover {
  transform: translateY(-2px);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
}

/* edit buttons */

  .camera-button-container {
    margin-bottom: 5px;
  }
  button[type="submit"] {
    margin-top: 5px;
  }
```

The cocktail image has also been styles to have a circular theme to make it stand out from the square image found on popular food recipe sites. (Begom, 2022)

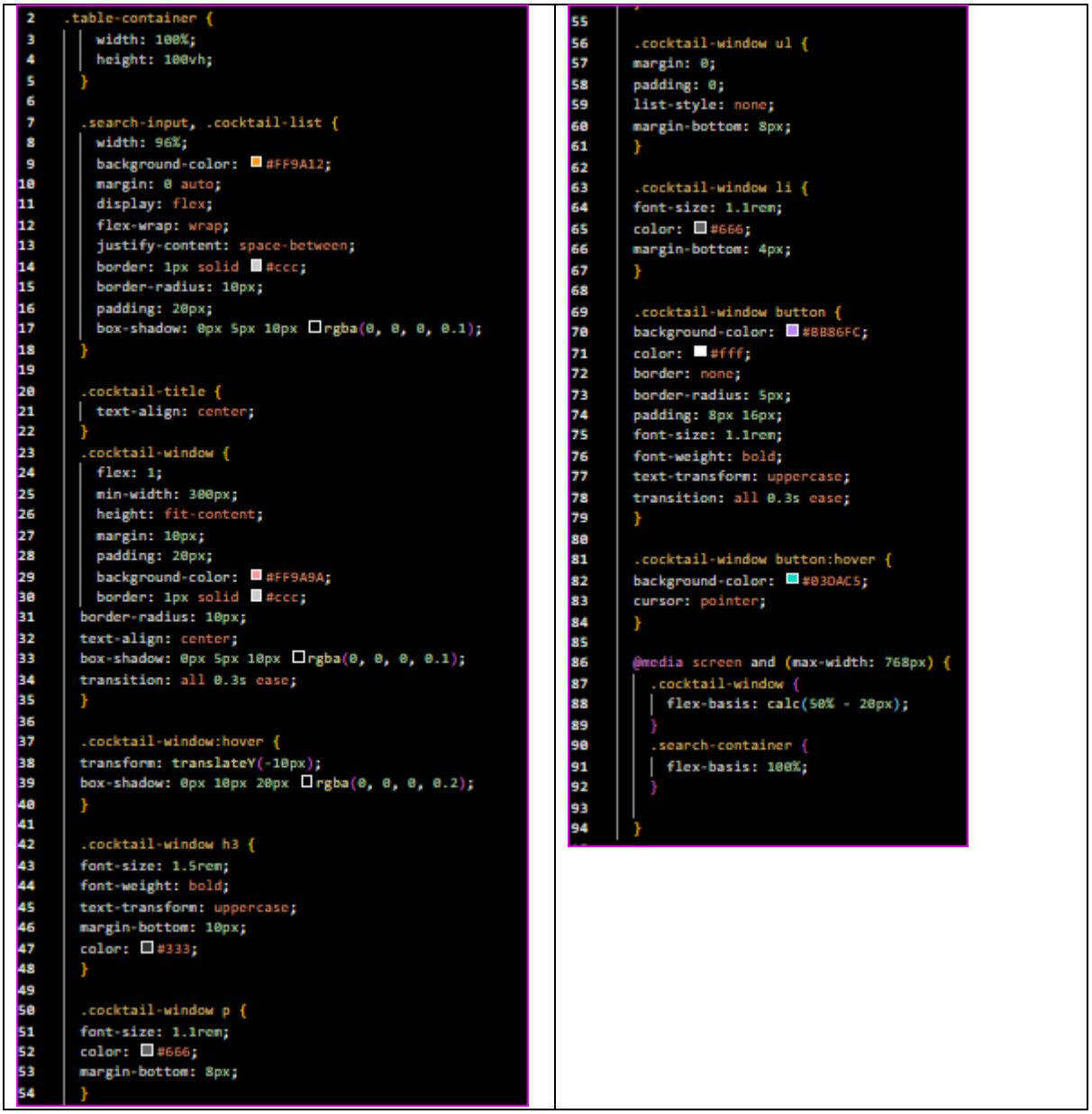
```css

@media (max-width: 600px) {
    form {
      border-width: 2px;
    }
  }
  .add-button{
    width: min-content;
  }

  .webcam {
    width: 200px;
    height: 200px;
    display: flex;
    justify-content: center;
    align-items: center;
  }

  .img{
    width: 200px;
    height: 200px;
    object-fit: contain;
  }
```

```css

  .cocktails-container {
    width: 100%;
    height: 100vh;
    background-color: #FFF1ED;
  }
  .cocktail-container {
    display: flex;
    flex-direction: column;
    align-items: center;
    justify-content: center;
    text-align: center;
  }


  .cocktail-image {
    display: flex;
    justify-content: center;
    align-items: center;
    height: 200px;
    background-color: #03DAC5;
  }

  .image-icon {
    width: 200px;
    height: 200px;
    object-fit: contain;
  }
```

Personal cocktail table

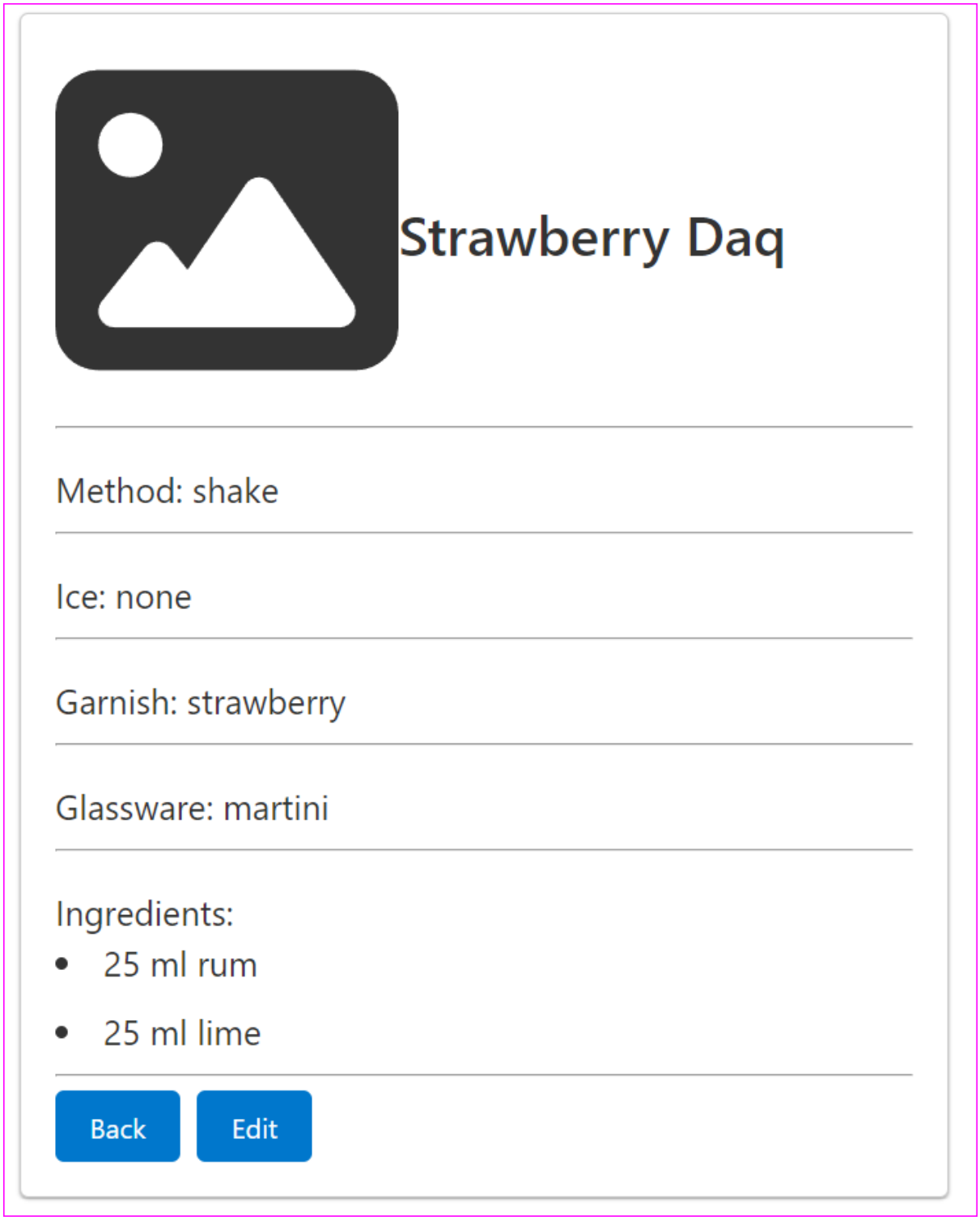


The personal cocktail table changes dynamically as each cocktail is added, the buttons on this page have been chosen as a softer colour to differentiate between the cocktail list and other parts of the application.



```css
.table-container {
  width: 100%;
  height: 100vh;
}

.search-input, .cocktail-list {
  width: 96%;
  background-color: #FF9A12;
  margin: 0 auto;
  display: flex;
  flex-wrap: wrap;
  justify-content: space-between;
  border: 1px solid #ccc;
  border-radius: 10px;
  padding: 20px;
  box-shadow: 0px 5px 10px rgba(0, 0, 0, 0.1);
}

.cocktail-title {
  text-align: center;
}
.cocktail-window {
  flex: 1;
  min-width: 300px;
  height: fit-content;
  margin: 10px;
  padding: 20px;
  background-color: #FF9A9A;
  border: 1px solid #ccc;
border-radius: 10px;
text-align: center;
box-shadow: 0px 5px 10px rgba(0, 0, 0, 0.1);
transition: all 0.3s ease;
}

.cocktail-window:hover {
transform: translateY(-10px);
box-shadow: 0px 10px 20px rgba(0, 0, 0, 0.2);
}

.cocktail-window h3 {
font-size: 1.5rem;
font-weight: bold;
text-transform: uppercase;
margin-bottom: 10px;
color: #333;
}

.cocktail-window p {
font-size: 1.1rem;
color: #666;
margin-bottom: 8px;
}

.cocktail-window ul {
margin: 0;
padding: 0;
list-style: none;
margin-bottom: 8px;
}

.cocktail-window li {
font-size: 1.1rem;
color: #666;
margin-bottom: 4px;
}

.cocktail-window button {
background-color: #BB86FC;
color: #fff;
border: none;
border-radius: 5px;
padding: 8px 16px;
font-size: 1.1rem;
font-weight: bold;
text-transform: uppercase;
transition: all 0.3s ease;
}

.cocktail-window button:hover {
background-color: #03DAC5;
cursor: pointer;
}

@media screen and (max-width: 768px) {
  .cocktail-window {
    flex-basis: calc(50% - 20px);
  }
  .search-container {
    flex-basis: 100%;
  }

}
```

Cocktail Dashboard



The cocktail dashboard presents the cocktail as a cocktail recipe card, it displays all the information gathered about the cocktail.

```css
.recipe-card {
  background-color: #ffffff;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.12), 0 1px 2px rgba(0, 0, 0, 0.24);
  border-radius: 5px;
  padding: 20px;
  margin-bottom: 20px;
}

.cocktail-recipe-header {
  display: flex;
  align-items: center;
  margin-bottom: 20px;
}

.cocktail-image {
  width: 150px;
  height: 150px;
  margin-right: 20px;
  border-radius: 50%;
  object-fit: cover;
}

.cocktail-header {
  font-size: 32px;
  font-weight: 600;
  margin-bottom: 10px;
}

.recipe-card p,
.recipe-card li {
  font-size: 20px;
  line-height: 1.5;
  margin-bottom: 10px;
}

.recipe-card button {
  background-color: #0077cc;
  color: #ffffff;
  border: none;
  border-radius: 5px;
  padding: 10px 20px;
  margin-right: 10px;
  cursor: pointer;
  transition: all 0.2s ease-in-out;
}

.recipe-card button:hover {
  transform: translateY(-2px);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
}

.recipe-card a {
  text-decoration: none;
  color: #0077cc;
}

.recipe-card a:hover {
  text-decoration: underline;
}

.recipe-card .button {
  display: inline-block;
  background-color: #0077cc;
  color: #ffffff;
  border: none;
  border-radius: 5px;
  padding: 10px 20px;
  margin-right: 10px;
  cursor: pointer;
  transition: all 0.2s ease-in-out;
  text-decoration: none;
}

.recipe-card .button:hover {
  transform: translateY(-2px);
  box-shadow: 0 2px 4px rgba(0, 0, 0, 0.2);
}
```

CONCLUSION

The cocktail capture application works as intended, the user can add a cocktail with or without location or image data. The image data is stored as a blob in the database rather than saving it elsewhere, this ensures that there are no orphan images stored in the application. The user can add a cocktail with variable length of ingredients which is stored in an array. The cocktail data can also be deleted and updated. The user can also view all the cocktails with a short description in the personal cocktail table and view a singular cocktail on the cocktail dashboard where the cocktail is displayed with all the details as a recipe card. The user also has access to view a global cocktail list from a mongoDB that currently can only be updated by admin behind the scenes. The crud operations for the mongoDB have been added to the application but there is currently no way for the user of the application to access them normally.

REFERENCES

Ail, V., 2020. *React Router Tutorial – How to Render, Redirect, Switch, Link, and More, With Code Examples.* [Online]
Available at: https://www.freecodecamp.org/news/react-router-tutorial/
[Accessed 02 01 2023].

Anon., 2022. *react-webcam.* [Online]
Available at: https://www.npmjs.com/package/react-webcam
[Accessed 01 03 2023].

AWS, 2023. *Tutorial: Get started with Amazon EC2 Linux instances.* [Online]
Available at: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EC2_GetStarted.html
[Accessed 25 01 2023].

Begom, A., 2022. *Bengali roast chicken.* [Online]
Available at: https://www.bbcgoodfood.com/recipes/bengali-roast-chicken
[Accessed 01 03 2023].

Create React App, 2023. *Create React App.* [Online]
Available at: https://create-react-app.dev/
[Accessed 02 02 2023].

Create React App, 2023. *Getting Started.* [Online]
Available at: https://create-react-app.dev/docs/getting-started/
[Accessed 09 02 2023].

DuckDNS, 2023. *About.* [Online]
Available at: https://www.duckdns.org/about.jsp?ref=upstract.com
[Accessed 21 01 2023].

Media, T., 2023. *Learn The MERN Stack - Express & MongoDB Rest API.* [Online]
Available at: https://www.youtube.com/watch?v=-0exw-9YJBo
[Accessed 01 02 2023].

Mozilla, 2023. *Add().* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Web/API/IDBObjectStore/add
[Accessed 01 03 2023].

Mozilla, 2023. *Navigator.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Web/API/Navigator
[Accessed 01 02 2023].

Mozilla, 2023. *Storage Manager.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Web/API/StorageManager
[Accessed 01 02 2023].

Mozilla, 2023. *Using IndexedDb.* [Online]
Available at: https://developer.mozilla.org/en-US/docs/Web/API/IndexedDB_API/Using_IndexedDB
[Accessed 03 02 2023].

NPMJS, 2022. *react-webcam.* [Online]
Available at: https://www.npmjs.com/package/react-webcam
[Accessed 03 02 2023].

npmjs, 2022. *uuid.* [Online]
Available at: https://www.npmjs.com/package/uuid
[Accessed 01 02 2023].

React Router, 2023. *Tutorial.* [Online]
Available at: https://reactrouter.com/en/6.10.0/start/tutorial
[Accessed 01 02 2023].

React, 2023. *react.* [Online]
Available at: https://react.dev/
[Accessed 02 02 2023].

W3Schools, 2022. *React useRef Hook.* [Online]
Available at: https://www.w3schools.com/react/react_useref.asp
[Accessed 21 03 2023].

W3Schools, 2023. *React useEffect Hooks.* [Online]
Available at: https://www.w3schools.com/react/react_useeffect.asp
[Accessed 14 02 2023].

W3Schools, 2023. *React useState.* [Online]
Available at: https://www.w3schools.com/react/react_usestate.asp
[Accessed 21 03 2023].