

**Diagram 1:** The diagram above shows a basic overview of the Buzz My Day application. The application is a MERN stack application that uses MongoDB, Express.js, React.js, and Node.js. In a nutshell, the application features a client side (front-end) and server side (backend). The front-end interacts with the backend through HTTP requests and responses. On the server side, Mongoose is used within the Express.js framework to interact with the MongoDB database through data requests.

**Main Components:**

**Front-End:**

**Codes used:** React.js, HTML, CSS, JavaScript

**React.js:** React.js is a JavaScript library responsible for the user interface. It is the view part of Model View Controller and renders the user interface based on information from the server. As the user interface is the part the client sees, this is referred to as the client-side part of the application. React.js also allows markup, CSS, and JavaScript to be combined into custom components

**HTML:** Hyper Text Markup Language (HTML) is responsible for the layout of the web application pages, used in React.js components.

**CSS:** Cascading Style Sheet is used to style the pages in the web application.

**JavaScript:** Used for testing in the front-end.

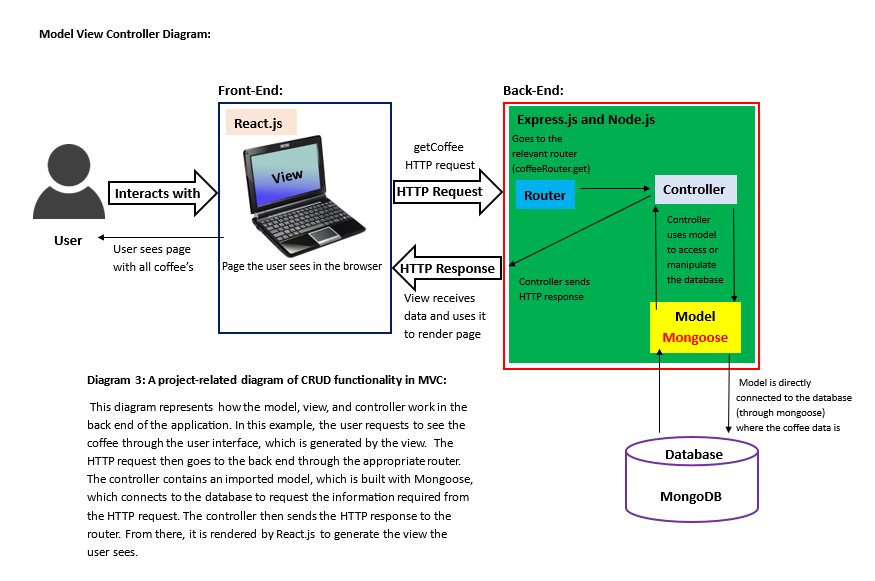
**Back End:**

**Express.js** is a Node.js framework used on top of Node.js to build a quick and robust API.

**Node.js** is a JavaScript run-time environment that allows JavaScript to run outside the browser, making it ideal to build scalable web applications, including this app.

**Mongoose:** A libraryused in the Express.js framework to connect to the MongoDB database for data requests and responses

**MongoDB:** A NoSQL database that is used to store data for the application, which is accessed through Mongoose

****

**MERN Stack in-depth look**

**Back End**

**Express Restful API**

**Languages used:**

* Node.js
* Express.js

**Components:**

* Model View Controller
* Routers
* Security
  + Bcrypt library
  + jsonwebtoken library

**Libraries:**

* mongoose
* cors (for cors middleware)
* dotenv
* express-validator
* save
* jest
* supertest
* express

**CRUD Operation**

Get, Post, Patch, Delete

**Routes:**

api/account, api/coffee, api/comment, api/favourite, api/post, api/review, api/user

**Front End**

]

**Languages used:**

* React.js
* HTML
* CSS
* JavaScript

**Components:**

* Routes
* JSX Files

**Libraries:**

* axios
* prettier
* eslint
* vite
* react
* react-dom
* react-router-dom
* classname\_corrector\_react

**HTTP Request**

dsx

**Axios is used to connect the Backend to the frontend**

**Data Request**

**Cors Middleware**

**HTTP Response**

**Data Response**

**Mongo DB**

**Node.js**

**Diagram 3:** The diagram above shows a more in-depth version of diagram 1, showing the components, middleware, libraries, and CRUD operations used.

**Front End:**

**JSX Files:** JSX files are a syntax extension of JavaScript that allows HTML to be written. It has been used in this web application to define the structure and content of the user interface.

**CSS:** Files imported into JSX files to apply styling to the web application.

**React.js:** useState and useEffect imported where needed throughout the JSX files. The useState hook was used to manage the states of components, and the useEffect hook was used to fetch data.

The web application consists of the following JSX files:

**JSX files** that are responsible for different pages, navigation bar, and footer on the web application (extra tab means files were imported into the file above)

* **Account.jsx: Account.css**
* **Coffee.jsx: Coffee.css**
  + **edit.jsx: edit.css**
* **Contact.jsx: Contact.css**
* **Favourite.jsx: Favourite.css**
* **Footer.jsx: Footer.css**
* **Home.jsx: Home.css**
* **LoginRegister.jsx: LoginRegister.css**
* **Reviews.jsx: Review.css**
  + **StarRate.jsx: StarRate.css**

**Other JSX files:**

* **App.jsx:** Contains the mounted routes for the application, including Home, Contact, Nav, Footer, Coffee, Favourites, LoginRegister, Reviews, Account, and AdminCoffee routes.
* **Api.jsx:** This JSX file connects the backend server to the front-end using Axios, an imported library that allows API calls to be made through React.js in the form of HTTP requests.

**Libraries:** Axios helps connect the front-end to the backend through HTTP requests, prettier and classname\_corrector\_react help keep formatting consistent, eslint helps identify and fix code problems, react-dom works alongside react for rendering, and react-router-dom enables dynamic routing for the web application.

**Vite:** Vite is a JavaScript development and build environment that is being used to help build and develop web applications.

**Back End:**

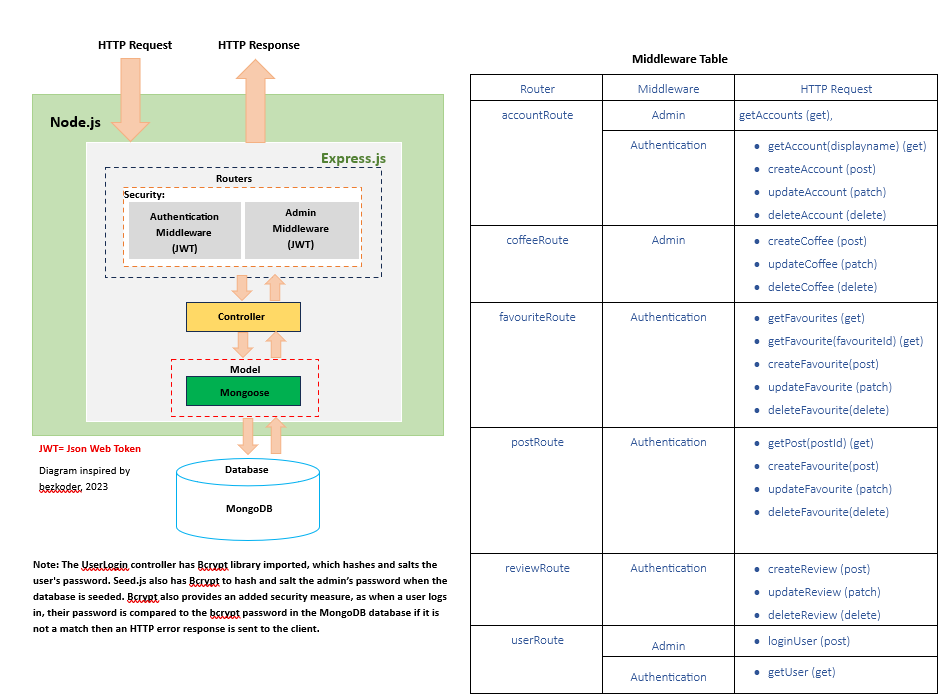
**CRUD:** This is a full-stack application with full CRUD functionality, meaning that users and admins can interact with the application by sending HTTP requests to the database to either create, read, update, or delete data through the HTTP requests get (fetch data), patch (update data), delete (delete data), and post (create data).

**CORS Middleware:** CORS stands for Cross-Origin Resource Sharing. This application uses it to handle requests between the front end and the back end by allowing the browser to make requests to different domains. This prevents the blocking of information requested from the back end.

**Library:** The library imports help with testing (supertest and jest), creating environmental files (dotenv), and creating validators (express-validator).

**Routes:** When an HTTP request is made, it goes through the appropriate route, which contains CRUD imports from the controller that deal handle the request and responses.

**Security diagram backend:**

****

**References:**

* **All library, code, and language information come from what has already been referenced and discussed in the Read.me of the Buzz My Day Application.**
* bezkoder and bezkoder (2020). Login Page with React.js, MongoDB: MERN JWT Authentication example - BezKoder. [online] BezKoder. Available at: https://www.bezkoder.com/react-node-mongodb-auth/#goog\_fullscreen\_ad [Accessed 4 May 2025].
* Buzz\_My\_Day\_App link: [ImTheDmThatsWhy/Buzz\_My\_Day\_App](https://github.com/ImTheDmThatsWhy/Buzz_My_Day_App)

**Diagram 4:** The diagram above shows how security is implemented into the Buzz My Day app. There are two different types of authentication layers: admin and authentication middleware. The admin middleware checks if a user is an admin in the MongoDB database. The authentication middleware checks that the correct user is performing actions by matching the JWT token in the MongoDB database. If the JWT tokens do not match, an error message will be sent as an HTTP response to the client. The table next to the diagram shows where middleware is used throughout the web application.

**How does it work?**

**Authentication:** When a user registers for the first time or logs in, a JSON Web Token is generated and stored within the MongoDB database. When a user goes to perform an action that requires authentication, it sends an HTTP request to the back end. The HTTP request goes through the controller, which has an imported model that uses mongoose, which connects to the MongoDB database. If the correct token is present, it is sent back to the client through the controller and the user is allowed to perform the action. If the token is unauthenticated, then an error message is sent back to the controller to the client, and the user cannot perform the action.

**Admin:** The admin role works almost identically to the authentication middleware, except it checks if a user is an admin or not. The admin role is created when the database is seeded, and when the admin logs in using the correct email and password, a JSON Web Token is created. So, when an admin task HTTP request is sent to the backend, it checks if the token of the user matches the token for admin in the MongoDB database; if the tokens match, the admin task can be performed. If the tokens fail to match, then an HTTP error message is sent to the client through the controller, and the user cannot perform the task.