**System Design**

The system was deigned based off the basic principles of a CRUD application. From front to back, A React UI was used to build the interfaces. All these components were organized in an appropriately named “components” folder where App.js would reference them. App.js however was not the entry point for the front-end. Index.js is where React is initialized and where the original render function takes place. Index is where Axios (our front-end middleware) was configured and where Redux would load the user’s information. App.js was simply a parent wrapper component for the other components and defined routes with appropriate authentication.

Flow diagram for front-end.

Diagram

Description automatically generated

Loading the user as well as any other user CRUD action, each appropriate component will call the back-end using Axios. Axios would then be interpreted by Express.js (our back-end middleware), do some work, and return some data. This happens in the routes folder in our Node.js application.

Flow diagram for middleware.

Diagram

Description automatically generated

The node server was deigned from an express boilerplate application. Server/bin/www is the actual entry point of the application but any configuration we were concerned with happened in app.js. Various end points were stored in routes/api. Each endpoint was defined and when hit, whatever action was requested would access the database (MongoDB) and return/manipulate some data.

Flow diagram for back-end.

Diagram

Description automatically generated

**Implementation**

The tech stack utilized was MERN. MongoDB, Express.js, React.js, and Node.js. The MongoDB was easily set up in the Linux system and two collections were created to store data. Items and Users. Items held information about the items. Each Item had a generated ID, name, ratings array (which held all of the ratings made to that item), a url to the item in question, a category describing where the item should go, an image, and the ID of the user who added it. Users held information about the registered users. Each user had a generated ID, name, email, password, and a rated items array (an array to describe which items they’ve rated). The Node.js server utilized mongoose to communicate with MongoDB. This allowed for easy to implement queries thanks to the help of schemas. Express.js then used this data to send it to the front-end where Axios retrieved it and passed it to React.js to render it.

**Conclusions and Future Work**

…conclusion…

For the future, we plan to implement a better render system to meet our design needs. Currently, React render’s the homepage using nested loops in order to achieve the desired design. An improvement would be to let CSS handle the rendering of the design. Additionally, we would like to incorporate HTTPS with an appropriately signed certificate. Finally, some features like rendering actual stars for ratings instead of numbers, a “followers” feature, an adult content filter, and more categories are some things we would like to add.