Code Quality Report

URXperience - Connecting UR World

ENSE 400/477 Winter 2023

Abraham Murgerwa

Dhruv Modi

Dinesh Dalip

Favor Fasunwon

Table of Content

Table of Figures

Design and Folder Architecture

Model view controller(MVC) design pattern is used for this project. For the model node.Js and mongoDB is used, the view is constructed with react.JS components and finally the controller is coded with node.JS. The advantage of this pattern was to separate the model and controller from the view, which further helps in easy modification of the code without having conflict with other segments. This made debugging and navigating far simpler. Github was utilized for version control where features for the project were implemented on different branches for concurrent work.

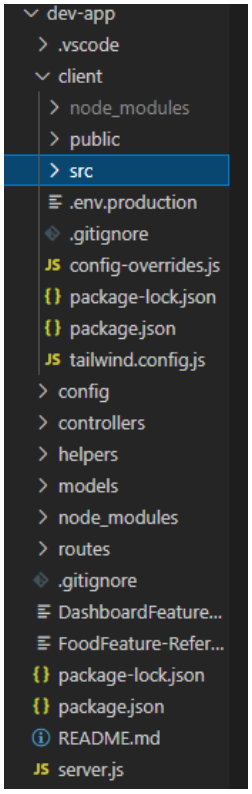

Figure 1: Folder Architecture
Path: URXperience\dev-app

MVC pattern helped in the folder architecture of the project. The folder structure is separated into frontend and backend. All components for the frontend are in the screens folder for easy, clean code. Main features have been divided into subunits of code in a component folder delegated to itself for ease of navigating and serves the purpose of not relying on other units on failure, which gives an advantage in the scalability of the project. As new features were implemented the same structure was used with no conflict with other features. Backend folder structure is mainly based on routes being in different files than controller and server. The controller folder holds all the functionalities with the models and frontend react components by the routes names. The effect of this is that all the routes are in one place and controller functionality in other for easy navigating and clean code.

Readability:

The work standard is clear and concise regarding readability of the code with usage of quality variable names which are self explanatory. Inline comments are used wherever the logic would be difficult to understand.

```
      })
      .then((res) => {
        setFormData({
          ...formData,
          title: "",
          description: "",
          date: "",
          from: "",
          to: "",
        });

        toast.success(res.data.message);
        getAllEvents();
      })
      .catch((err) => {
        toast.error(err.response.data.error);
      });
  } else {
    toast.error("Please fill all the fields");
  }
};

// delete event
const deleteEvent = (_id) => {
  if (_id) {
    axios
      .post(`${process.env.REACT_APP_SERVER}api/eventdelete`, { _id })
      .then((res) => {
        toast.success("Successfully deleted Events");
        getAllEvents();
```

Figure 2: Quality variables name
Path: URXperience\dev-app\client\src\screens\EventComponents\PostEvent.jsx

All functions and variables follow a camel case format as seen in the figure above. All routes follow quality named endpoints. Furthermore DRY principle is used whenever possible, which refers to not repeating yourself and removing duplicates. One of the places where this principle is utilized is in the valid.js file, which is used to validate various conditionals of a user. The same functions supply multiple routes/pages with validation of the logged in user. Moreover, the feature recipes with different dietitians use DRY principles. Where the same code is used to render different dietitians by simply changing the dietitian variabel.

Portability

The project was designed by keeping in mind different platforms. The initial purpose of the site was to be used on a desktop but by receiving user feedback the project was structured so no matter which device is used, the system must be flawless and consistent.

```
<Navbar />

<div className="mt-16 p-10 grid grid-cols-1 sm:grid-cols-1 md:grid-cols-2 lg:grid-cols-2 xl:grid-cols-3 gap-5">
  <RecipeOfTheDay />
  <TopCategory />
```

Figure 3: Screen size Portability
Path: URXperience\dev-app\client\src\screens\FoodScreen.jsx

As seen in figure 3 a grid has been used to render the webpage. Depending on the screen size of the output, the code will optimize with specific layout. This type of portability is used throughout the whole project to give the user a flawless experience.

```
<meta name="viewport" content="width=device-width, maximum-scale=1" />
```

Figure 4: Hardware compatibility
Path: URXperience\dev-app\client\public\index.html

One of the non-functional requirements for the project was compatibility with different hardware. Above in figure 4, is one of the examples where specific code is used for the site to stay consistent on any platform such as Desktop, IPhone, Android without changing any scales, demissions, etc. Therefore, utilizing different devices of similar dimensions keeps the functionality the same and does not require users to change their way of interacting with the site.

Security

The user data is protected at many levels in this project. User password is hashed and stored in the database. Furthermore, the future in the site for finding nearby restaurantes doesn't require user location, it is possible to manually search restaurants by location.

Usability

Practices such as KISS, signifiers are used to make the site user friendly. Many of the functionalities and signifiers are used with respect to users previous knowledge. The site doesn't require the users to learn new ways to interact as the placement of buttons and functionalities micmick other platforms such as clicking a gramabe icon deletes a specific data, clicking a Plus icon with respect to the chat page under group chat creates a group new chat. On the mobile version the hamburger menu is placed on the right side as most websites. A major rule followed to make the site usable and reliable was to always give meaningful feedback upon user interactions with the platform. All fields have a successful message upon success and as well as error message if such condition occurs during exciting unit of codes.

```
            <div className="font-bold text-xl mb-2 text-center underline">
              Foods
            </div>
            <div className="font-regular italic font-serif text-md mb-2 text-center">···
            </div>
            <RecipeOfTheDay />
          </div>
        </div>
        <div className="rounded overflow-hidden shadow-lg bg-green-400 h-fit">
          <div className="px-12 py-4">
            <div className="font-bold text-xl mb-2 text-center underline">
              Events
            </div>
            <div className="font-regular italic font-serif text-md mb-2 text-center">···
            </div>
            {allEvents.length !== 0 ? (
              allEvents.map((event) => (···
              ))
            ) : (
              <h1 className="text-center font-bold">No Current Events</h1>
            )}
          </div>
        </div>
        <div className="rounded overflow-hidden shadow-lg overflow-y-scroll container bg-blu
          <div className="px-6 py-4">
            <div className="font-bold text-xl mb-2 text-center underline">
              Online Members
            </div>
            <div className="font-regular italic font-serif text-md mb-2 text-center">···
            </div>
            {users.length !== 0 ? (
              users?.map((usr) => (···
              ))
            ) : (
              <h1 className="text-center font-bold">···
              </h1>
```

Figure 5: Dashboard- least resistance path
Path: URXperience\dev-app\client\client\src\screens\Dashboard.jsx

Additionally, the project was implemented taking into account the problem of least resistance path. In the above figure this problem is solved, for example, when the user logs on to the site a list of recently posted events are displayed, as a least resistance path rather than checking for newly posted events in the events page. Similarly, instead of users going to the chat page to find online users, a list of online users is displayed on the dashboard. This technique is

used at numerous places for various functionalities such as the logout button being available on most screens for quickly logging out instead of locating it in the menu.

Maintainability:

The Law of Demeter was utilized throughout the project where each unit has limited information of other units. Each major feature in the project serves a different purpose and has its own container. Modifying/ changing any unit of code does not break any additional unit and requires none to limited knowledge of other units. Our top features for this project, Complain Box, Chat, Events will not break and serve its sole purpose if any of the other features are down. During user testing it was extremely easy to navigate bugs, change, and add additional functionalities. These techniques helped in improving the flexibility, code quality, and maintainability of our code. As the project moved towards future MVPs the maintainability of the code was fairly easy due to loose coupling.