EPI-USE LABS
value through innovation

Coding Standards

Contents

1. Steps taken to ensure Compliance 2
2. Coding Conventions Used 2
2.1. Ionic React 2
2.2. Heroku 2
3. File Structure 3
3.1. Front End 3
3.2. Back End 4
3.3. Branching 5
4. Comments 6
4.1. Comments used for Code Blocks 6
4.2. Comments used for Single Lines 6
4.3. Doxygen 6

1. Steps taken to ensure Compliance

The team began the project with a set of standards that were agreed upon at the beginning and were based on the standards that had been taught during the COS 301 Graduate Portal Year 1 project. Additional coding conventions from languages and technologies were also integrated. These standards ensure a seamless connection between the various components and technologies used, whilst promoting good coding practices.

2. Coding Conventions Used

The basic coding conventions, such as indentation within functions and classes, are applied to all code. This includes consistent naming of branches, filenames and variable names.

2.1. Ionic React

The basic coding conventions of Ionic React are used, such as indenting each label as they open and close, to ensure all code is easy to read and comprehend.

Ionic React works well with the decomposition architectural design strategy, and helps enable the reusing of components, thus each component is created separately in the front-end component file system and imported into the front-end pages of the project.

Each component and application page is situated within its own separate folder under the correct parent folders, this ensures that all parts that make up a page or component, e.g. typescript file and CSS file, are all stored together.

There is a main CSS file that is used as a global variable file for all CSS components, with only small changes to the CSS components for individual pages stored separately. This main CSS file co-ordinates the colour scheme, font specifications and margins and size guides for the entire application.

2.2. Heroku

The backend of the application uses Heroku and thus complies with the Heroku coding standards.

The database of the application uses PostgreSQL, and its coding conventions.

The backend of the application also complies with the JavaScript API coding conventions.

The backend of the application uses the ORM (Object Relational Mapping) design pattern that specifies that each table is treated as its own entity, and the team follows this coding convention.

Any errors that occur within the backend is treated as an error warning in the front-end of the application to ensure that the data within the database and the structure of the backend is not compromised or altered in an unauthorized manner.

Each component that comprises the backend is subject to both unit testing and integration testing, to ensure that all components work properly as an individual unit and as a part of the larger whole.

Each API call is properly documented within the SRS document, and follows the coding conventions of the database, backend architecture, and JavaScript language.

3. File Structure

The file structure was divided into two parts – the front-end and the back-end. The front-end and the back-end operate independently of each other. The front-end and the back-end contain a folder called *src* which contains all the source code the respective part.

3.1. Front End

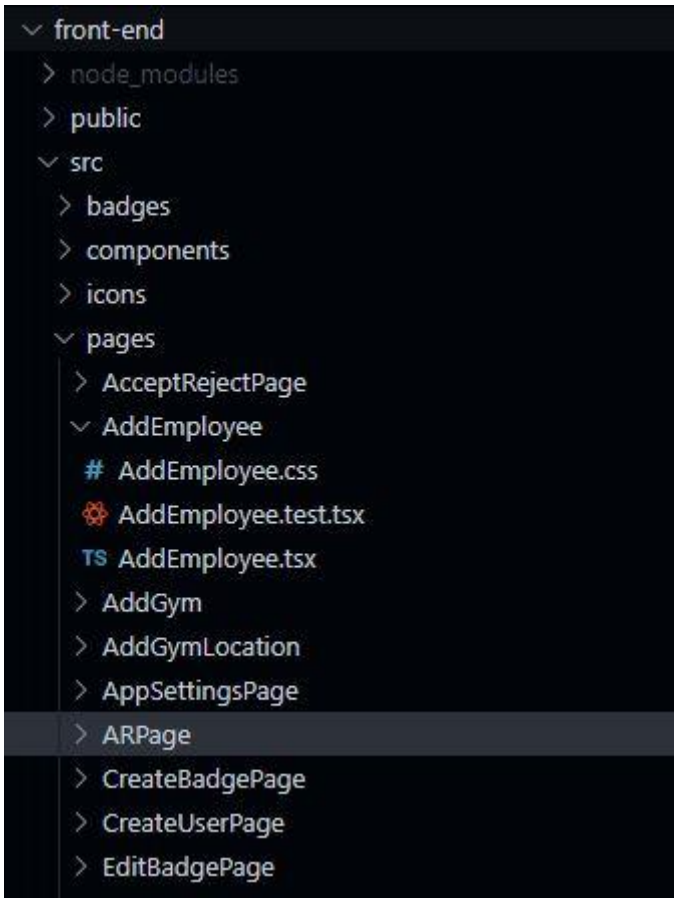

Figure 1 – Front-end File Hierarchy

3.2. Back End

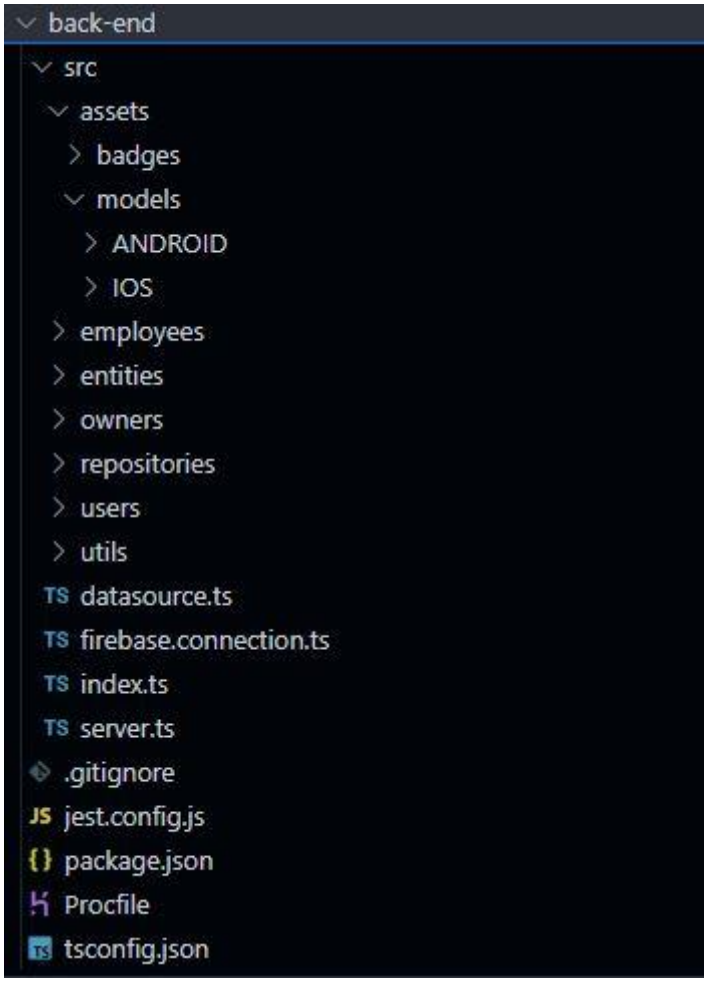

Figure 2 – Back-end File Hierarchy

3.3. Branching

Each branch follows a specific agreed naming convention with the type of branch (e.g. “Feature-” and the name of the branch. Branches are made first, and then merged with the “develop” branch, which is the main working branch of the GitHub repository.

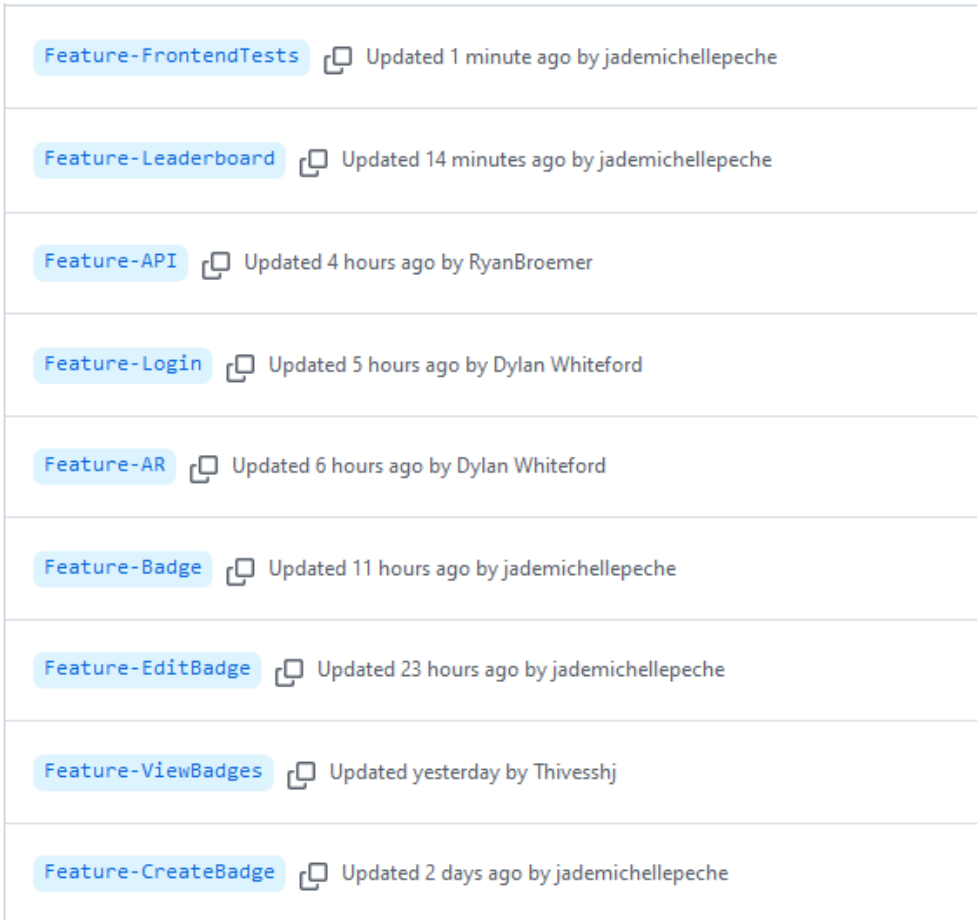

Figure 3 – Example of Branches on the GitHub repository

4. Comments

4.1. Comments used for Code Blocks

- @brief ! - Short description of the code block
- @param ? - input parameter for function with a type
- @requires ? - object needed for successful completion of the code block that is not a parameter
- @return ? - the return type of the function
- @result ? - the result of the code block's execution

4.2. Comments used for Single Lines

- // - explanation of line
- ? - optional
- ! - required

4.3. Doxygen

Doxygen is used to assist with the commenting of both front-end and back-end code.

```
/**
 * IonViewWillLeave
 * @brief on exit the modal will be closed
 */
useIonViewWillLeave(()=>{
  setOpen(false)
})
/**
 * changeCoords function
 * @brief changes coordinates of center of map aswell as gym
 * @param {number} lat latitude coordinate
 * @param {number} long  longitude coordinate
 */
function changeCoords(lat:number,long:number)
{
  setCenter([lat,long]);
  setGymCoord([lat,long]);
}
```

Figure 4 – Example of Comments within code