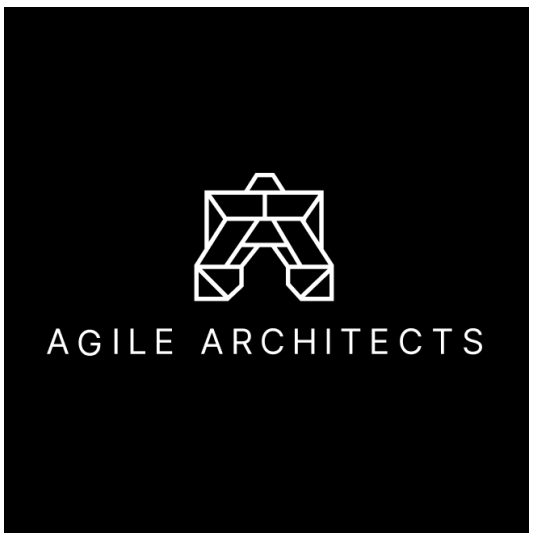
PRONTO

Code Quality Standards

Name	Student Number
Shashin Gounden	u21458686
Tyrone Sutherland-MacLeod	u21578878
Bandisa Masilela	u19028182
Andile Ngwenya	u20612894
Jonel Albuquerque	u21598267



Contents

Code Quality Standards

The development of our app has adhered to the following code quality standards:

1. Readability:

We coded using meaningful variable and function names, writing clear comments, and organising code into logical sections.

2. Formatting:

We followed consistent use of indentation, line length, spacing, and the placement of braces. We used GitHub Super Linter to ensure that the code is formatted uniformly.

3. Error Handling:

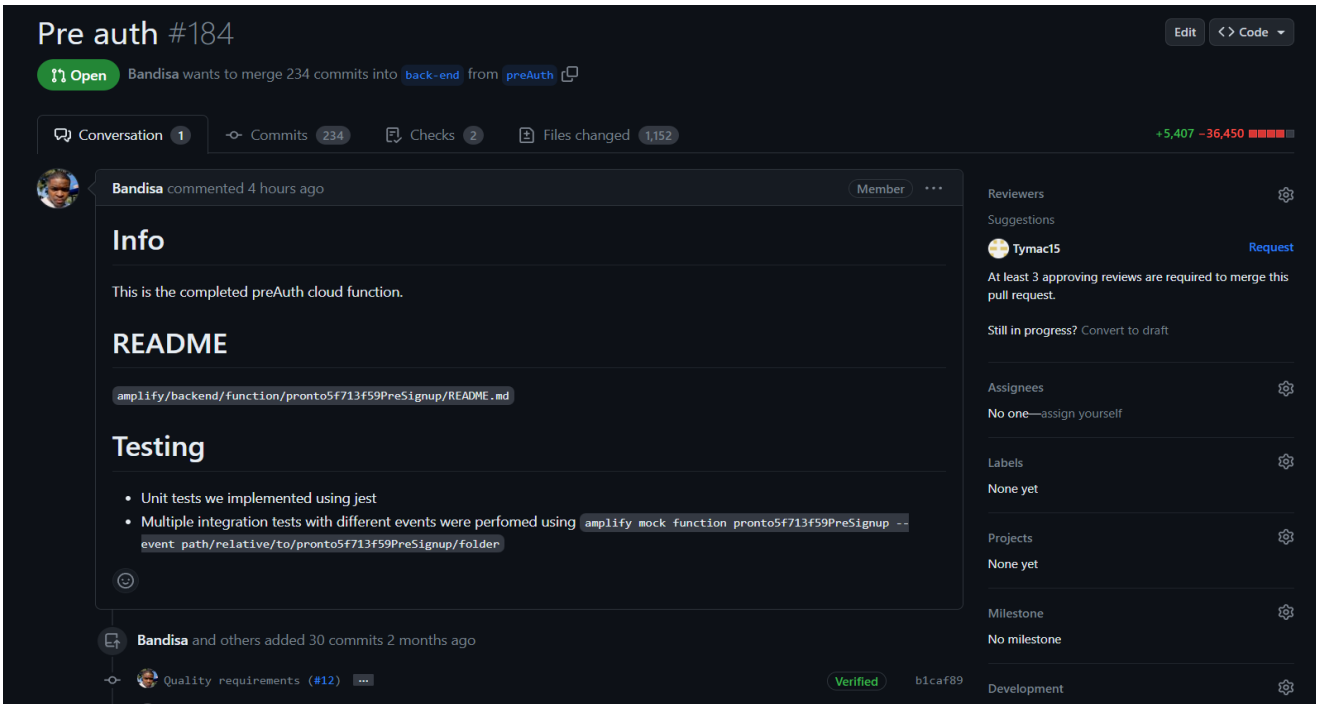
We use appropriate error messages in the UI, and use try catch statements to handle exceptions as needed. The error messages can be shown to users in the form of alerts or error boxes.

An example of a file that represents these standards can be seen below:

```
import React, { useState, useEffect } from "react";
import { Amplify, Storage } from "aws-amplify";

// Folder name for S3 bucket
let folderNameS3 = "UniversityOfPretoria";

function DropzoneComponent() {
  const [selectedFile, setSelectedFile] = useState(null);
  const [uploadProgress, setUploadProgress] = useState(0);
  const [message, setMessage] = useState("");

  const createFolder = async (folderName) => {
    try {
      const folderKey = `${folderName}/`; // Include the trailing slash ("/") to indicate it's a folder
      await Storage.put(folderKey, "", {
        contentType: "application/octet-stream", // Set the content type to a generic value
      });
    } catch (error) {
      setMessage("Error creating folder: " + error);
    }
  };

  useEffect(() => {
    Amplify.configure({
      Auth: {
        identityPoolId: "",
        region: "us-east-1",
      },
      Storage: {
        AWSS3: {
          bucket: "",
          region: "us-east-1",
          keyPrefix: `${folderNameS3}/`,
        },
      },
    });
  }, []);
```

4. Documentation:

We documented new additions of code in pull requests, explained its purpose and any important considerations. In conjunction with comments within the code, we included separate documentation files in our pull requests.



5. Modularity:

We structured our code into reusable functions and components that perform specific tasks. This helps improve maintainability, reusability, and testability of the code.

6. Testing:

We have frontend and backend unit tests, integration tests to validate the code's functionality and ensure that code is reliable.

```
            height={"155px"}
            data-testid={'UniversityImage'}
          />

          {/*display lecturers name*/}
          <div className="lecturer-name">Stefan Gruner</div>
        </div>

        <ul className="navbar-nav">

          {/*Buttons on the nav bar menu*/}
          <li className="nav-item text-center"  data-testid={'EditModuleInfo'}>
            <a href="/lecture-homepage" className="nav-link"  data-testid={'EditModuleInfoLink'}>
              <b>Edit Module Information</b>
            </a>
          </li>
          <li className="nav-item text-center"  data-testid={'RecentAnnouncements'}>
            <a href="recent-announcement" className="nav-link"  data-testid={'RecentAnnouncementsLink'}>
              <b>Recent Announcements</b>
            </a>
          </li>
          <li className="nav-item text-center"  data-testid={'EditPersonalInfo'}>
            <a href="personal-info" className="nav-link"  data-testid={'EditPersonalInfoLink'}>
              <b>Edit Personal Information</b>
            </a>
          </li>
        </ul>
      </nav>

      {/*button to log out*/}
      <div className="logoutbtn fixed-bottom col-2 p-4 ml-4">
        <button className={"btn btn-danger btn-lg btn-block"} style={{borderRadius:"25px"}} data-testid={'LogoutButton'} onClick={onSignOut
      </div>
    </div>
  );
}
```

Note that in the screenshot on the left each button has a “data-test-id” which is used to test that each of them is fireable as part of our unit testing for the frontend. This file is also a component, the nav-bar, which can conveniently be reused in multiple pages of the lecture view, allowing us to reap the benefits of modular code.

7. Security:

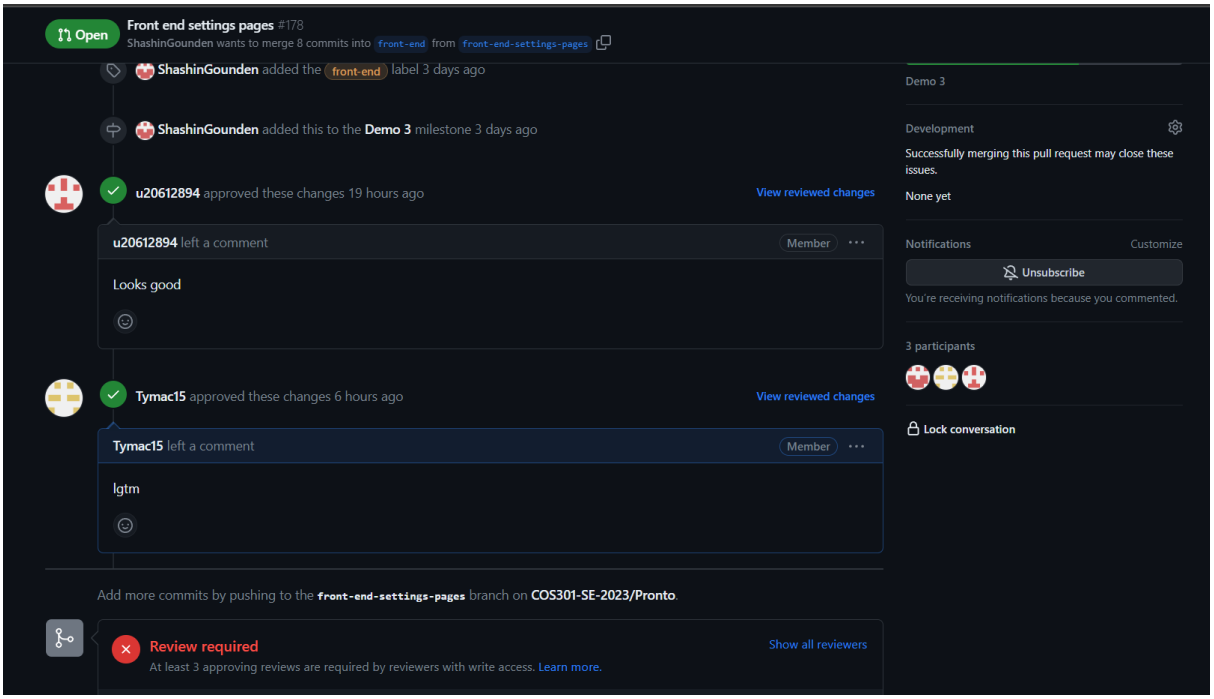
Care has been taken and common secure coding practices such as input validation, have been used to ensure that areas of vulnerability (e.g input from a user) are secured against common attacks.

8. Version Control:

We used GitHub for version control of our code, with clear commit messages, a GitHub flow branching strategy, and creating pull requests which must be reviewed by 3 members before being merged.

9. Code Reviews:

The team participates in code reviews, providing constructive feedback in the comments of pull requests and addressing any identified issues. At least three members need to approve a pull request for it to be merged.



As you can see in the screenshot above, two people have reviewed a pull request and made comments that all is well, however the branch protection rules are blocking a merge as 3 reviews are required to proceed. This ensures that at least 3 other people have reviewed the code and made sure that it follows the coding standards and can be given the go ahead to merge.