https://blog.webdevsimplified.com/2022-07/react-folder-structure/
https://profy.dev/article/react-folder-structure
https://github.com/alan2207/bulletproof-react/blob/master/docs/project-structure.md
https://medium.com/@tharshita13/creating-a-chrome-extension-with-react-a-step-by-step-guide-47fe9bab24a1
https://crxjs.dev/vite-plugin/getting-started/react/create-project

1. Our project will be a Google Chrome extension written using Javascript and React, thus we plan to follow the file architecture for this language and framework. We looked at the links above to have a good idea of how to format our folders for our program. We decided to use the following structure in the diagram to the right. For our Chrome extension will have the frontend folder and for the AWS infrastructure code, we will have the backend folder. Each directory will have a src/ and __tests__/ folders. The root directory will contain the generic github files and needed configuration files such as the manifest.json and package.json for our project. The src/ folder will contain the actual code of our project. It will have subfolders for the application code with utils and services folder, and subfolders for components and hooks for the web pages. The __tests__/ directory will contain all of the tests files with a subfolder for all of the unit tests and a subfolder for end to end and integration tests. The unit test subfolder will follow the src/ folder structure to help with organization of unit tests.

```
HashPass
- frontend/
  - src/
    - app/
    - components/
    - hooks/
    - utils/
    - services/
  - __tests__/
    - unit/
      - app/
      - hooks/
      - utils/
      - services/
    - integration/
- backend/
  - src/
    - handlers/
    - utils/
    - models/
  - __tests__/
    - unit/
  - package.json
- .gitignore
- .github/workflows
- manifest.json
- package.json
- README.md
```

2. Our repository will use a feature branching model. There will be one main branch that no collaborators should be working on. Instead, each collaborator can take up a feature/issue and in order to work on it will create a new branch to work on this feature. The naming model for each branch is as follows: #<issue-number>-<short-description>. An example of a branch name could be #123-user-authentication. To merge changes back into main, a pull request should be created. More information on the pull request is stated below.

3. Code Review Policy
    a. In order to merge changes, a team member must submit a Pull Request in Github. The PR will ask the other 2 members of the team for a code review, however, it only requires the approval of 1 other team member to approve and merge.
    b. Merge Window: Once a Pull Request is approved, merge the code as soon as possible, the merge window should not exceed 1 day at most.
    c. Integration Targets:
        i. Continuous integration and Integration Tests will run on every merge
        ii. Person who submitted the PR must merge the branch to main after approval (not the reviewer).
        iii. Person who merges should notify the rest of the team that a merge has occurred so the rest of the team can rebase and continue working on the most “up-to-date” codebase.
    d. Metadata review: Metadata for a pull request must include the following:
        i. Issue number (Generated by Github Projects)
        ii. Description of the issue
        iii. Expected output or behavior of this pull request
    e. Code review Flow:
        i. Pull Request is created → Code Review → Iterate if necessary → Get approval from 1 other dev → Merge → Run Integration Tests on merge