Coding Standards

InfoSafe

Seed Analytics

Team Name: FrAgile

Team Members: Christof Steyn
Chris Mittendorf
Karel Smit
Yané van der Westhuizen
Alistair Ross

Team Contact: fragile.cos301@gmail.com

Table Of Contents

Introduction **2**
Repository **3**
2.1 Description 3
2.2 File Structure 3
2.2.1 Backend 3
2.2.2 Frontend 3
2.3 File Structure Diagram 5
Coding Standards **7**
3.1 Naming Conventions 7
3.1.1 Directories and Folders 7
3.1.2 Files 7
3.1.3 Functions 7
3.1.4 Variables 7
3.1.5 Constants 8
3.1.6 Globals 8
3.2 Structure of Code 8
3.2.1 Functions 8
3.2.2 Documentation 8
3.2.3 Formatting 8
3.3 Editing and Styling 9
3.3.1 Components - Frontend 9
3.3.2 Linter 9
3.3.3 Prettier 9
3.4 Testing 10
3.4.1 Components - Frontend 10
3.4.2 Controllers - Backend 10
3.4.3 Services - Backend 10

Introduction

This document serves as a comprehensive documentation of the coding standards applied to the Infosafe project as well as the description of the project repository and understanding the file structure.

Infosafe is a Java Spring Boot project that uses React on the frontend and as such the frontend and backend are separated within the repository and components interact with each other via imports.

Repository

2.1 Description

The Infosafe project is a monorepo, where all the application code is located in a single repository. As stated above the fronted system and backend system are located in separate directories and our three main directories are:

- Backend - all backend features and database linking
- Frontend - all frontend pages and styling
- Documentation - files for figma designs and use case diagrams (this will not be discussed further)

2.2 File Structure

2.2.1 Backend

All backend features are located in the Backend directory.
The directory Infosafe_Backend/src/main/java/com/fragile/infosafe contains the following folders:

- auth - Directory for system authentication
- config - Directory for system configuration
- controller - The model controllers
- model - All the models for entities are located here
- repository - Repositories for entities. These are used to communicate with the database
- requests - Directory for requests and routing
- service - Systems services and general functions are located here
- token - The directory for producing session tokens

Also in this directory, the InfosafeBackendApplication.java is located and this file is used to run the backend. There is also a pom.xml file here that is used to link and store Maven dependencies for the Spring Boot project.

2.2.2 Frontend

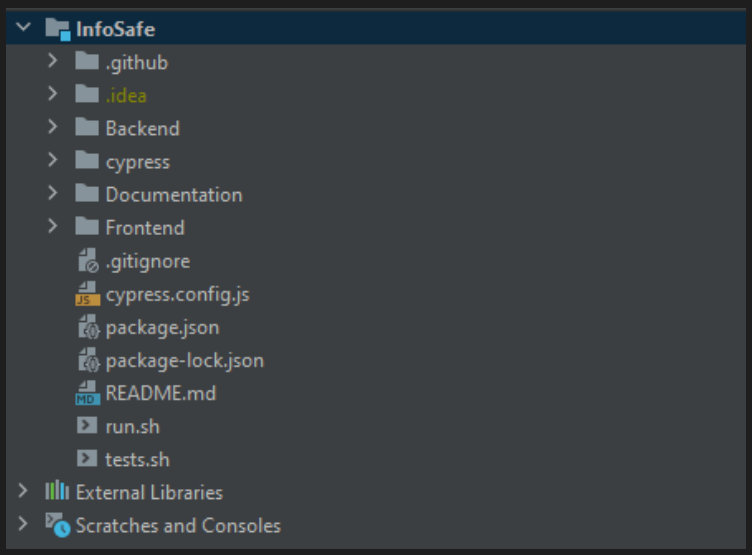
All frontend features are located in the Frontend directory.
The directory infosafe_frontend/src contains the following folders:

- Components - The React javascript files used for displaying pages for the client are here
- Images - Images for any frontend pages are stored here
- Styling - The .css files for page styling are located here

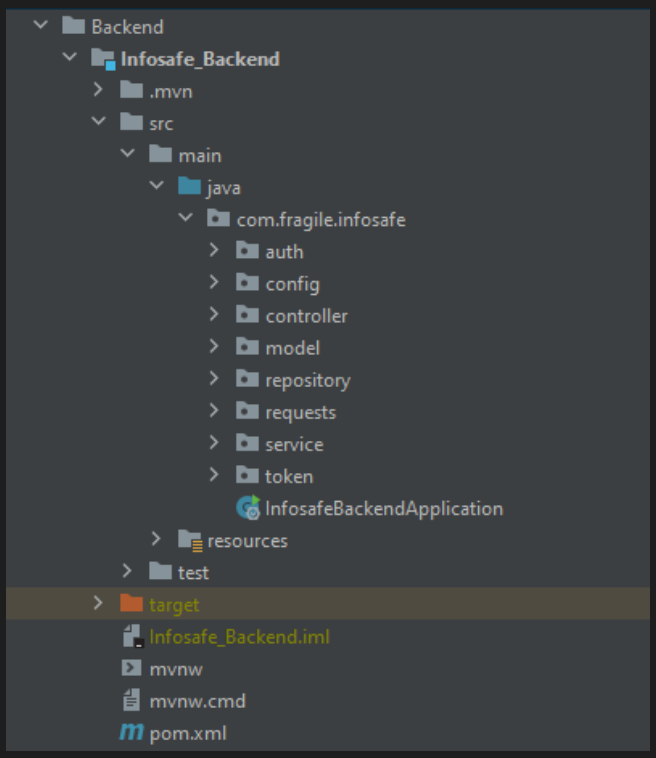
The index page and styling are also located in this directory

2.3 File Structure Diagram

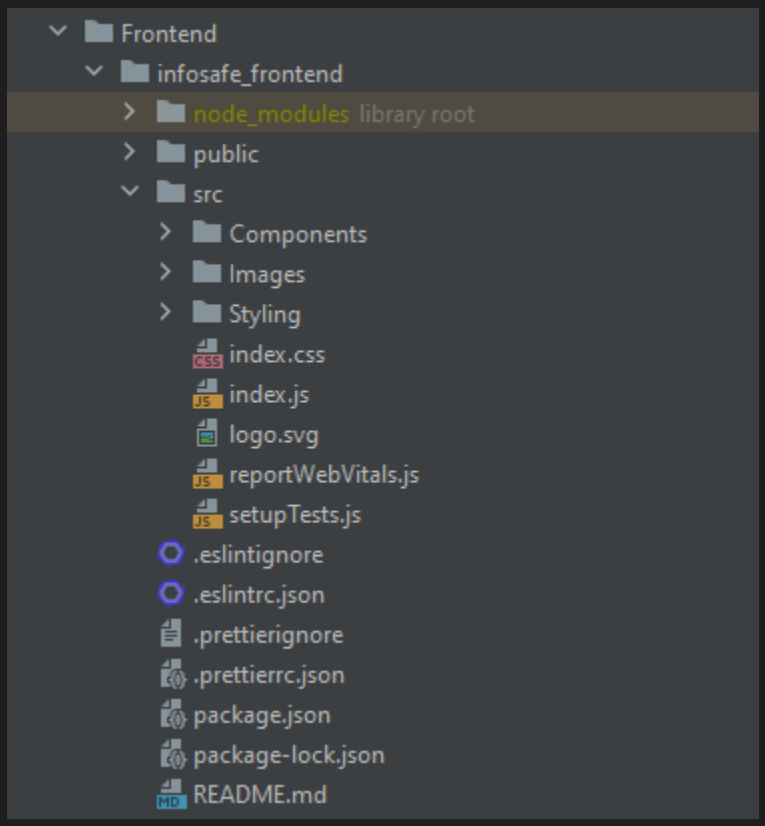
Root:



Backend:



Frontend:


Frontend
infosafe_frontend
node_modules library root
public
src
Components
Images
Styling
index.css
index.js
logo.svg
reportWebVitals.js
setupTests.js
.eslintignore
.eslintrc.json
.prettierignore
.prettierrc.json
package.json
package-lock.json
README.md

Coding Standards

3.1 Naming Conventions

3.1.1 Directories and Folders

- All folders and directories should be descriptive
- Subsystems and different types of files should be separated and located in their own directories, eg: web pages, stylesheets etc.
- Names should all be lowercase and underscore separated when the directory name is more than one word, eg: infosafe_backend

3.1.2 Files

- Files should have descriptive names
- Files should be a single string with a capital letter at each word, eg: AssetController.java

3.1.3 Functions

- Function names should be descriptive and be able to deduce the function purpose by looking at the name
- Functions should start in lowercase and each word after starting with a capital letter, i.e. use the camelcase naming convention. Function names should also be a single string, eg: encryptPassword()

3.1.4 Variables

- Variable names should be descriptive
- Variables should only contain letters, not numbers and symbols
- The name should be a single string and follow the camelcase naming convention, eg: userDocument

3.1.5 Constants

- Constant names should be descriptive
- Constants should only contain letters, not numbers and symbols
- Constants should be a single string of only capital letters and is recommended that only single words are used, eg: TIMER

3.1.6 Globals

- Global names should be descriptive
- Globals should only contain letters, not numbers and symbols
- Globals should be a single string and words that contain a capital letter at the start of each word, eg: SessionToken

3.2 Structure of Code

3.2.1 Functions

- Functions should be small and precise
- Functions should not be too large that too many parts can fail
- Functions make use of other functions to complete tasks
- A modularisation approach should be followed at all times

3.2.2 Documentation

- Comments are encouraged above functions to state what is happening at this point
- Short descriptive sentences
- If the developer feels the need to explain in depth, this should happen on the same line as the code after the semicolon(;)

3.2.3 Formatting

- Blank lines should be included in functions to improve readability. Line breaks should occur when a “block” or similar code changes

- Variables should be declared on separate lines even if they are of the same type and include the same value
- Indentations should occur in code blocks and functions
- A line break should occur between the code block declaration and the first line of code
- parenthese (braces, {}) should open on the same line as the declaration and be closed on a new line after the final line of code

3.3 Editing and Styling

3.3.1 Components - Frontend

- Each component should have a style sheet
- Each component should be styled from accepted designs and system standard should be used
- Figma designs should be referenced if any ambiguity arises

3.3.2 Linter

- A linter has been set up to check our code for the specific styles
- This will help maintain the standard throughout the system and make sure all rules are adhered to
- The linter analyzes all code being pushed to the repository
- The linter should be run locally before submitting to check for any errors in the code, this can be done with ESlint.
- Then when code is pulled into dev, all linting tests should pass

3.3.3 Prettier

- Prettier has been implemented in the system too
- It can be run on the development side to format code to meet our set rules
- This will help with the linting when pushing to the repository
- It works hand in hand with ESlint in order to fix issues.

3.4 Testing

3.4.1 Components - Frontend

- Functions in components should have integration and unit tests
- Unit tests for loading functions correctly should be tested
- Components should have end-to-end (e2e) tests

3.4.2 Controllers - Backend

- Controllers and functions in controllers should unit tests and integration tests

3.4.3 Services - Backend

- Each function in a service file should contain unit tests and tests for integration