



# 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](mailto:fragile.cos301@gmail.com)

---

# Table Of Contents

<b>Introduction</b>	<b>2</b>
<b>Repository</b>	<b>3</b>
2.1 Description	3
2.2 File Structure	3
2.2.1 Backend	3
2.2.2 Frontend	4
2.3 File Structure Diagram	4
<b>Coding Standards</b>	<b>6</b>
3.1 Naming Conventions	6
3.1.1 Directories and Folders	6
3.1.2 Files	6
3.1.3 Database Tables	6
3.1.4 Database Columns	6
3.1.5 Functions	6
3.1.6 URL Mapping and Routing	7
3.1.7 Variables	7
3.1.8 Constants	7
3.1.9 Globals	7
3.1.10 References, Imports, Classes and Finals	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	9

---

## 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, system documents 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 two directories namely [delete](#) and [primary](#).

[delete](#) contains the following folders:

- [deletemodel](#) - contains the models for the entities to be created in the secondary “delete” database
- [deleterepository](#) - contains repositories for entities that are deleted and added to the secondary database

[primary](#) contains following folders:

- [auth](#) - Directory for system authentication
- [config](#) - Directory for system configuration
- [controller](#) - The model controllers
- [exceptions](#) - The classes where exceptions are exclusively handled
- [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.

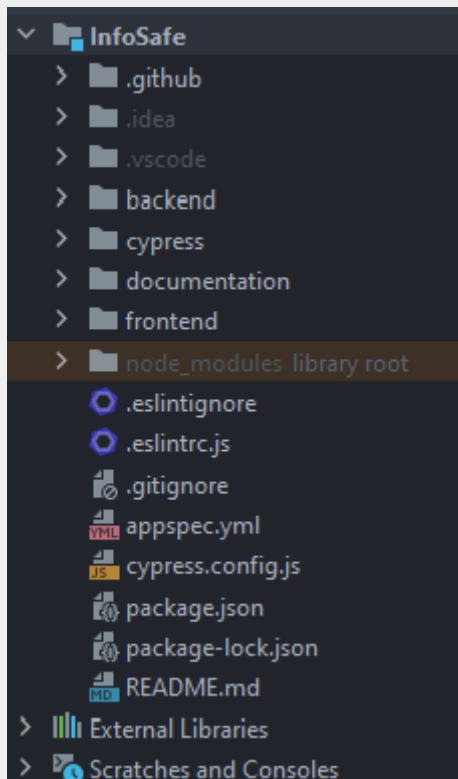
The directory [infosafe\\_frontend/src](#) contains the following folders:

- [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

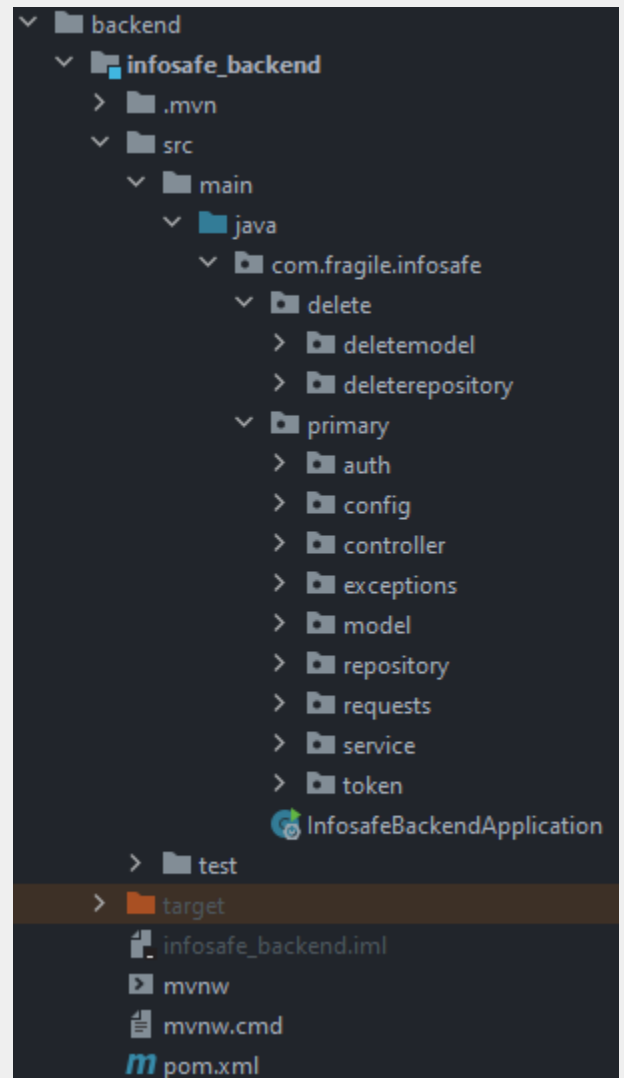
The index page and styling are also located in this directory

## 2.3 File Structure Diagram

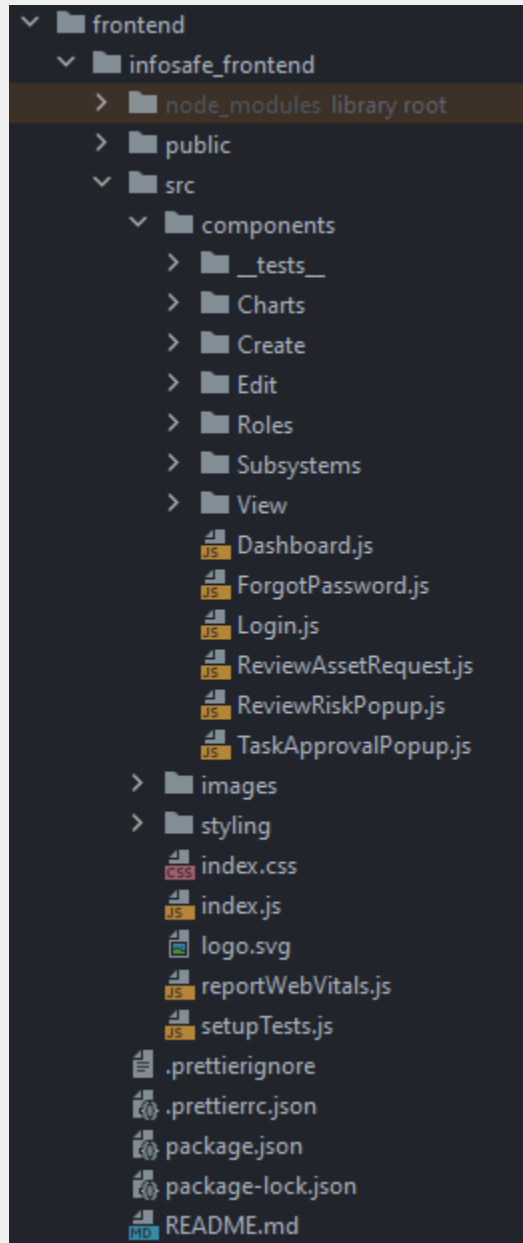
Root:



Backend:



Frontend:



---

# 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 Database Tables

- Table names should be descriptive
- Table names should all be lowercase and underscore separated when the table name is more than one word, eg: `support_requests`

### 3.1.4 Database Columns

- Column names should be descriptive
- Column names should all be lowercase and underscore separated when the columnname is more than one word, eg: `data_scope_id`

### 3.1.5 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()`

### 3.1.6 URL Mapping and Routing

- Mapping names should be descriptive and similar to the function making use of it
- Mapping names should start with a forward slash ( / ) and the first letter in lowercase and each word after starting with a capital letter, i.e. use the camelcase naming convention. Mapping names should also be a single string, eg: `/getAsset`

### 3.1.7 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`

### 3.1.8 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, if more than one word is used it should be underscore ( \_ ) separated, eg: `TIMER` or `ADMIN_RIGHTS`

### 3.1.9 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`



---

### 3.1.10 References, Imports, Classes and Finals

- These items' names should be descriptive
- These items' should only contain letters, not numbers and symbols
- These items' should be a single string and words that contain a capital letter at the start of each word, eg: `MatrixController`

## 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

- Find a link [here](#) to the testing policy document where a detailed analysis of all tests for the system can be found.