

Department of Computer Science
University of Pretoria

Coding Standards for GND

April Four

July 2024

Version	Version History	Author
1.0	Initial release	AprilFour

Table 1: Version History



Table of Contents

1	Introduction	3
2	Naming Conventions	4
2.1	Frontend - Angular and Electron	4
2.2	Backend - Python	4
3	Code Formatting	5
3.1	Frontend - Angular and Electron	5
3.2	Backend - Python	5
4	GitHub Strategy	6
4.1	Mono-Repo	6
4.2	Gitflow	6
4.3	Branch Protection	7
4.4	GitHub Issues	7

1 Introduction

This document serves to outline the Coding Standards for the GDPR Non-Compliance Detector (GND) application. It is a blueprint that will detail the file structure that the project uses, naming conventions to aid readability and consistency, error handling and GitHub usage.

2 Naming Conventions

The naming conventions used throughout the project aim achieve the following:

- Use names that clearly state what the function, class, variable or component does. This is remove ambiguity and allow all developers to work with existing code without further clarification or explanation.

2.1 Frontend - Angular and Electron

- **Components & Classes** - naming of components and classes should follow the **PascalCase** naming convention. In this format, all new words in a name start with a capital letter. Examples - `Inbox`, `UploadDocumentComponent`.
- **Typescript & Javascript** - within the .ts files that detail the logic of frontend components, the **camelCase** naming convention should be used for variable and function names. The first word of the variable should start in lowercase and additional words added to the variable name should start with an uppercase letter. Examples - `getReports()`, `searchComplianceStatus`.
- **HTML & CSS** - all class, tag names and variable names within HTML and CSS files should follow the **kebab-case** naming convention. Examples - `analysis-header`, `.analysis-text`, `<div class="inbox-rectangle">`.

2.2 Backend - Python

- **General** - the **snake_case** naming convention should be used for all class, variable and function names when using Python. This is that standard naming convention used across most Python projects and helps with readability and code familiarity for all developers. Examples - `file_monitor`, `backend_entry`.

3 Code Formatting

Code formatting assists in readability and a standard format of code across all files.

3.1 Frontend - Angular and Electron

- A single tab is used for indentation within and outside functions and subsequent code nesting follows the same format.
- Add comments next to a line of code to explain its purpose if it isn't clear or on the line below it.
- Do not leave excessive comments and keep files easy to read.
- Keep a standard function brace (`{}`) within a file. Either on the same line as the function definition or on the next line.
- SuperLinter is used as a linter to enforce proper code formatting across the frontend.

3.2 Backend - Python

- Due to the nature of the Python language, the indentation policy must follow the format enforced by the language. Thus a single indentation policy must be followed.
- SuperLinter and Flake8 are used as linters to enforce proper code formatting across the backend.

4 GitHub Strategy

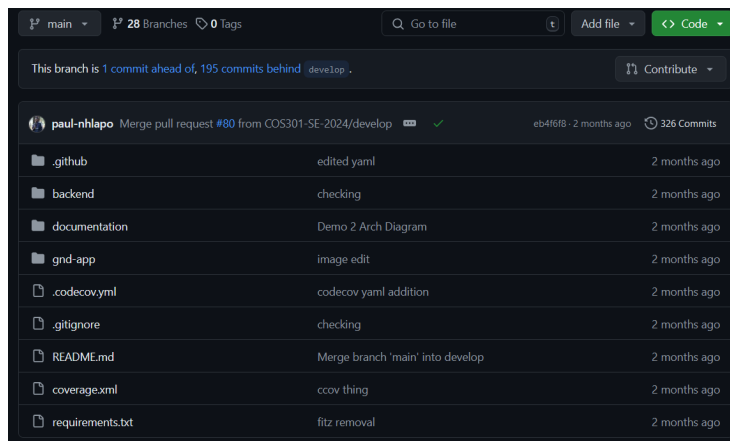
This section outlines the structure of our GitHub repository, workflow and branching protection strategies.

4.1 Mono-Repo

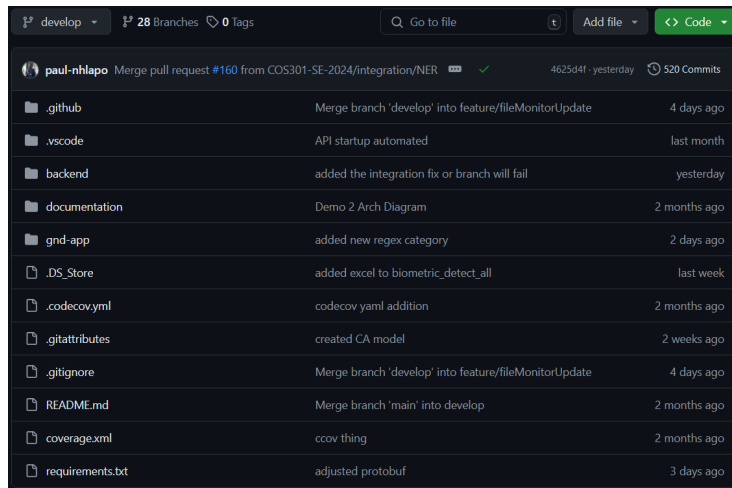
The GND team uses a mono-repo strategy for the project. All files relating to the project are stored within the "GDPR-data-noncompliance-detector" repository under the "COS 301 - 2024" repository. This includes all dependencies, components and configurations, bar the HuggingFace models that are used to detect potential GDPR violations. Due to the size of these models they were required to be hosted in a HuggingFace repository.

4.2 Gitflow

We use the Gitflow approach within the project, whereby we have a **main** branch which contains the current stable release of the application. This version is the current version of the app available for public use and has only been updated when all code has been thoroughly tested and potential exceptions handled. We update **main** before each Demo.



The **develop** branch contains integrated code from all the different feature branches that have been tested and merged. **develop** is the default branch in the repo to assist with branching. This branch contains the code that is ready to be merged into **main** once all planned features have been implemented and the team is happy to update the current app build.



The screenshot shows a GitHub repository interface for a user named 'paul-nhlapo'. At the top, it indicates '28 Branches' and '0 tags'. Below this, a table lists various files and folders in the repository, along with the commit message for the most recent change and the time since the last commit.

File/Folder	Commit Message	Time Ago
.github	Merge branch 'develop' into feature/fileMonitorUpdate	4 days ago
.vscode	API startup automated	last month
backend	added the integration fix or branch will fail	yesterday
documentation	Demo 2 Arch Diagram	2 months ago
gnd-app	added new regex category	2 days ago
.DS_Store	added excel to biometric_detect_all	last week
.codecov.yml	codecov yaml addition	2 months ago
.gitattributes	created CA model	2 weeks ago
.gitignore	Merge branch 'develop' into feature/fileMonitorUpdate	4 days ago
README.md	Merge branch 'main' into develop	2 months ago
coverage.xml	ccov thing	2 months ago
requirements.txt	adjusted protobuf	3 days ago

In order to implement new features or test code, **feature** branches are created. These new branches are typically branched off the current **develop** branch and once the feature has been implemented and tested, they are merged back into **develop** pending all integration tests passing.

4.3 Branch Protection

We prevent direct PRs to the **main** and **develop** branches by ensuring that all workflows and integration tests pass before merging. Merging into **develop** requires a review by at least one team member and merging into **main** requires at least two reviews.

4.4 GitHub Issues

When implementing a new feature, testing or fixing a bug, each team member adds this as a GitHub issue and assigns either themselves or others to the issue. Issues are given descriptive names and full description below to properly explain the problem to other team members.