Cyber Justitia: Coding Standards Document

Introduction

This document outlines the coding standards used to develop Cyber Justitia by Team 25 for the module CSC2033. Adhering to these standards ensures code quality, readability, maintainability, and consistency across the project.

General Guidelines

- Clear, readable, and maintainable code must be written at all times.
- Use of the DRY (Don't Repeat Yourself) principle avoids code duplication and discrepancies.
- Use of meaningful and descriptive names for variables, functions, classes, and other entities.
- Code is documented with in-line comments and docstrings.
- File purpose and authors are defined at the top of each Python file.
- Docstrings are provided for every class and function/method definition.
- All function and variable names are defined in snake case.
- All class names are defined in Pascal case.
- All constants are defined with capitalized letters.
- All Python files follow the PEP 8 style guide for Python code.
- Use of consistent indentation (4 spaces for Python, 4 spaces for HTML/CSS/JS).
- No hardcoded values, all environment variables are stored in a .env file.
- No unnecessary imports in any of the Python files.
- All imports are at the top of the file, under the file documentation.
- No commented-out code or debug prints left in production code.

Backend (Django + PostgreSQL)

- Functionality for each part of the project is contained in its own app (users, forum, chatbot)
- Each app contains at least the following files and directories:
    - __init__.py
    - admin.py
    - apps.py
    - urls.py
    - models.py
    - views.py
    - migrations/
        - __init__.py
    - templates/
    - tests/

- The project contains a directory (justitia) with at least the following configuration files:
    - __init__.py
    - settings.py
    - asgi.py
    - wsgi.py

Models

- Special __str__ method is defined to return a readable string representation of the class instance.
- Meta class is used for model-specific options (e.g., ordering, verbose_name).
- Use of meaningful field names and appropriate field types.
- Use of comments and docstrings to explain database fields and class methods.
- Database schema changes are defined using Django migrations.

Templates

- Use of the Jinja templating engine for Django.
- Templates are organised into the templates/ folder in each app, with the only exception of project-wide templates defined in the root templates/ folder.
- Use of inheritance to avoid code duplication.

Forms

- Use of Django's forms framework for handling form validation and rendering.
- Forms are defined in forms.py within each app.
- Forms also perform validation to add a third layer between frontend and backend validation.

URLs

- Use of the path() Django function for defining URLs.
- URL patterns for each app are defined in the app's urls.py file and linked in the project urls.py file using the includes() Django function.
- URLs are grouped and ordered based on relevant functionality.

Tests

- Unit tests must exist for every app's models, views, and forms.
- Tests are placed in the tests/ directory of each app and contain files that begin with the "test_" prefix.
- Use of Django's test framework and coverage tools.

Frontend Standards (HTML, CSS, JS, Jinja)

HTML

- Styling is kept separate from the structure of the code where possible in CSS stylesheets.
- Sections are wrapped in elements to group together relevant functionality.
- Forms perform validation of user input on the frontend for an additional level of security.

CSS

- Follow BEM (Block, Element, Modifier) methodology for class naming.
- Styles are stored in the static/ folder found in the root of the project.
- Each app stores its static files in its subdirectory of static/.
- External stylesheets are linked to Jinja templates in the <head> of each template using <link> elements within the {% block styles %} tag.

JavaScript

- Use of ES6+ syntax.
- Modular, reusable code written wherever possible.
- Use of a linter like ESLint to maintain code quality.
- Minimize direct DOM manipulation, frameworks and libraries should be used where needed.

Jinja

- Use of Jinja for dynamic content rendering.
- Templates are kept DRY by using includes and extends tags.
- Context variables are passed from the appropriate view functions and populate the template.