Coding Standards Documentation

Coding Standards for
Domain Pulse

Ctrl Alt Defeat

Contents

1 File Structure

Our system (Domain Pulse) has multiple different services and components that are often times independant from one another, as a result of our service oriented architecture. Furthermore, our system makes use of a Angular frontend and Django backend. As a result of these factors, the file structure of our system is designed to with modularity and simplicity as focuses, and we have achieved such. These qualities have been achieved with there being three main folder paths of note within our system: 'backend', 'frontend' and 'documentation' (of which will not be covered as it is not 'Coding'). Each folder is self-explanitory in its purpose and all code relating to said folder is contained within it, with little to no overlap.

1.1 Frontend

The frontend folder contains a folder 'src' containing the developed resources. The 'assets' folder conveniantly stores all assests (icons,backgrounds, etc...) needed for the UI development. The 'app' folder contains all developed Angular UI pages, neatly stored in their corresponding component folders (login-page, register-page, etc...)

1.2 Backend

Within the backend folder, every seperate service of the system is stored within the relevant folders. This allows for our clearly defined and seperated software oriented architecture. This ensures modularity and flexibility within our system by clearly defining individual services. Each of these services folders will contain one folder storing the needed Django settings and others containing the developed code needed for the proper execution of the service.

2 Naming Conventions

2.1 Frontend

When it comes to naming conventions for frontend development, follow these guidelines:

1. **Use meaningful and descriptive names**: Choose names that accurately describe the purpose or functionality of the component, variable, function, or class. Avoid using vague or ambiguous names.

2. **Follow camelCase**: Use camelCase notation for naming variables, functions, and object properties. Start with a lowercase letter and capitalize the first letter of each subsequent concatenated word. For example: `myVariable`, `getUserData()`, `myObject.property`.

3. **Use PascalCase for class and component names**: Class names and component names should follow PascalCase notation. Start each word with an uppercase letter. For example: `MyClass`, `MyComponent`.

4. **Avoid using single-letter names**: Use descriptive names that convey the purpose of the variable or function. Single-letter names are generally not recommended unless they are used as iterators in loops.

5. **Use lowercase for file names**: File names should be in lowercase and use hyphens to separate words. For example: `my-file.js`, `my-component.html`, `styles.css`.

6. **Be consistent with naming conventions**: Ensure consistency in naming across the entire codebase. Use the same conventions for similar types of elements (variables, functions, classes, etc.) throughout the project.

7. **Avoid reserved keywords**: Avoid using reserved keywords or language-specific identifiers as names for variables, functions, or classes, as they may cause conflicts or unexpected behavior.

8. **Use self-explanatory names for CSS classes**: Choose CSS class names that are descriptive and convey their purpose. Avoid using generic names or abbreviations. Use hyphens to separate words in class names. For example: `.main-container`, `.btn-primary`, `.error-message`.

9. **Prefixes and suffixes**: Consider using consistent prefixes or suffixes for variables or functions to indicate their purpose or type. For example, prefixing boolean variables with "is" or "has" (`isActive`, `hasError`) or suffixing functions with verbs (`calculateTotal`, `validateForm`).

10. **Avoid excessive abbreviation**: While it's important to keep names concise, avoid excessive abbreviation that may make the code less readable. Aim for a balance between brevity and clarity.

2.2 Backend

Within the backend of the project naming conventions are followed to ensure readable and understandable code. These include a well defined structure for endpoints whereby the endpoint function shall have a descriptive name. The URL shall be the same as the endpoint name. Auxiliary files are created to store the logic of the functions within the backend and these are stored in folders such as 'util' or 'preprocess' to give a clear understanding of their grouping of purpose. The functions within these auxiliary files include functions with the same name as the endpoint (indicating they are what is to be called in the endpoint) and helper functions such as '`remove_whitespace`' to create an orderly structure within the main functions. This clear definition of different main and auxiliary functions increases readability and flexibility of our code as it leads to bugs being easier to 'track down'. As well as this variables are ensured to name that clearly describes their purpose within the codebase.

3 Formatting

3.1 Frontend

3.2 Backend

The backend code of our system is developed using Django which uses Python as its programming language. Inherently, by using python, our code will follow a strict and ordered format due to pythons use of indentation. As well as this inherent structure, we also use the 'Black' formatter to ensure readability and consistency within the code developed.