Coding Standards

Coding standards help in the development of software programs that are less complex and thereby reduce the errors. Here in this project
We have used Django as a framework and HTML CSS for the front end development. Most of Django's important practices are based on Python.

Naming Convention

- Use meaningful name
- Function and variable names in snake_case
- Classname in PascalCase
- Constants snake_case capitalized

Indentation/Spacing

- Use 4 spaces for indentation(Python 3 disallows mixing the use of tabs and spaces for indentation)
- Separate method definition inside class by one line
- Maximum length of line should be less than 80 characters
- There should be no trailing white spaces
- Camel case should not be used

```
class MyFunction(arguments):
    model = Date      # variable name
    numericValue = 10 # snake_case

def my_func(var1):
      var1 = {key,value}
      return var1.attribute1ricValue = 10 # snake_case

def my_func(var1):
      var1 = {key,value}
```

```
    return var1.attribute1
```

Imports

Always import statements are placed at the top of the source file because importing a package is a direct implication of code-reusability. Here

1. Import from Python standard library
2. Import from core Django
3. Import from 3rd party vendor
4. Import from Django Apps(Current Project)
5. Avoid import *

Migrations

- Do not modify the files created by makemigration command(do not add custom sql command)
- Place custom sql command if needed in a separate file and do not mix it with the auto generated files from makemigration command
- Forward and backward migration work only on auto generated files by makemigration command
- Not all migration can be reversed
- Add — database=<dbConfigName> always in your migration

Response Status

- Response message with status codes
- 200 OK — Success — GET/PUT — return resource/status message
- 201 Created — Success — POST — provide status message or return newly created object
- 204 No Content — Success — DELETE
- 304 Unchanged — Redirect — ALL — Indicates no changes since last request
- 400 Bad Request — Failure — GET/PUT/POST — invalid request, return error messages
- 401 Unauthorized — Failure — ALL — missing credentials/Authentication required
- 403 Forbidden — Failure — ALL — restricted content
- 404 Not Found — Failure — Resource not found
- 405 Method Not Allowed Failure — Failure — ALL — An invalid HTTP method was attempted

Tools/Extra

- Code Quality — flake8
- Environment Manager — venv
- Documentation — Swagger
- Testing — pytest
- IDE — sublime,vim,pycharm
- Debug — pdb

Model-View-Template Architecture

Django is based on MVT (Model-View-Template) architecture. MVT is a software design pattern for developing a web application.

MVT Structure has the following three parts –

Model: Model is going to act as the interface of your data. It is responsible for maintaining data. It is the logical data structure behind the entire application and is represented by a database (generally relational databases such as MySql, Postgres). To check more, visit – Django Models

View: The View is the user interface — what you see in your browser when you render a website. It is represented by HTML/CSS/Javascript and Jinja files. To check more, visit – Django Views.

Template: A template consists of static parts of the desired HTML output as well as some special syntax describing how dynamic content will be inserted. To check more, visit – Django Templates