Tropical Cyclone Tracker

WWLLN

Revision 0.6

ECE270, Fall, 2021

Table of Contents

1 Introduction **2**
1.1 About This Document 2
1.1.1 Document Version History 2
1.2 Using This Document 2

2 Style Guidelines **3**
2.1 Naming Conventions 3
2.1.1 Functions 3
2.1.2 Classes 3
2.1.3 Variables 3
2.2 Dependencies 4
2.3 GIT Workflow 4
2.4 Security 4

3 Design Details **5**
3.1 Data Flow 5
3.2 Directory Structure 5
3.2.1 TCDataCollection 5
3.2.2 TCDataProcessing 5
3.2.3 TCFrontEnd 6
3.2.4 wwlln 6
3.3 Security 6

4 API Details **6**
4.1 Functions 6
4.2 Classes/Structures 6

1 Introduction

1.1 About This Document

This document provide an overview of the code base as well as a style guide to be followed during the development of this project. This document is intended for those working on the WWLLN Storm Tracking project.

1.1.1 Document Version History

Document Version	CatOS Version	Changes
v1.0	v0.6	Initial Release

1.2 Using This Document

For style guidelines please see: Style Guidelines This document is meant both to help new developers on the project get up to speed quickly, and to offer in-depth documentation for the modules present within the code. This document should be updated as new features and revisions occur.

2 Style Guidelines

This section describes the style guidelines for this project. New developers should be sure to review this section thoroughly. These style guidelines are used in order to ensure clean and easy to read code. This allows all developers on the project to understand the code base without having to dedicate extensive amounts of time.

Any changes to these guidelines should be discussed with all project members involved before being implemented.

2.1 Naming Conventions

The naming conventions of the project are intended to provide readability. The important part when deciding on names is that they properly describe the use of the object/function being named. Functionality and use should be easily understandable with minimal comments required using the name.

2.1.1 Functions

Functions should use the snake case naming convention where all words are lower case and separated by underscores.

Example: descriptive_function_name(params){...}

2.1.2 Classes

Classes should use the pascal case naming convention where all words are denoted by an uppercase as their first letter.

Example: DescriptiveClassName

2.1.3 Variables

Variables should use the camel case naming convention where all words except the first are denoted by an uppercase as their first letter.

Example: descriptiveVariableName

2.2 Dependencies

One of the most important aspects of a code base that can greatly alter readability of the code is its dependencies. When learning a new code base it can be quite confusing to have to dig multiple files deep in order to understand the functionality of a single function or object. For this reason chains of dependencies should be avoided where possible.

Good comments, descriptive names, and document strings can help limit how deep a developer has to go in order to better understand the code base.

Dependencies should be limited to at least 3 layers deep where possible. As layers progress the functionality should simplify allowing for more complex modules to be built from simple ones.

2.3 GIT Workflow

In order to facilitate isolation of functional changes a proper GIT workflow should be followed. This will allow for versions to be tracked and reverted to if needed.

The repository structure consists of a main branch, a development branch, and various feature branches. Most development should be done on feature branches. The development branch should be used to test integration of completed features. The main branch should only be merged into to update the live version.

2.4 Security

In order to keep the various credentials private, all credentials should be placed in a single file and that file should be added to the .gitignore file. Credentials should be kept private.

For further security specifications see Design Details: Security

3 Design Details

This section goes over the design choices made for this project. Very little code will be explained here. This section serves as the philosophy and design behind the structure for the server.

3.1 Data Flow

The flow of data is the main focus of many design choices made in this project. The project consists of three steps focused around the flow of this data: Data collection, data processing and displaying the data as a front end product.

3.2 Directory Structure

The directory structure is modeled directly after the data flow. Objects and scripts should be placed in the directory corresponding to the step in the data flow that it correlates with.

Django related files can be placed in the root of these directories (Models, Views, Urls, etc.). A subdirectory for scripts should be used to store related python or matlab scripts.

3.2.1 TCDataCollection

The TCDataCollection directory should store any files related to the collection of data. This can include collection from remote or local sources. Processing of the data should be left to the later stage of the data flow process. Modules in this step should only be concerned with retrieving data and placing it in its correct location

3.2.2 TCDataProcessing

The TCDataCollection directory should store any files related to the processing of data. The data processed should be considered to have been correctly collected in the previous step(s) of the data flow process. This step should process the data in whatever ways are necessary to produce the products presented in the front end.

3.2.3 TCFrontEnd

The TCFrontEnd directory should store any files related to the presentation of the processed data to the users. This data should be considered to have been correctly collected and processed by previous step(s) of the data flow process. This step should handle interaction with the user, both in user input and in presentation of data.

3.2.4 wwlln

The wwlln directory should store minor utilities and minor functionalities that multiple steps in the data flow process use or utilities. These functionalities should be minor. If functionalities within this directory become too large, consideration should be given to a new directory for the functionality.

3.3 Security

The data flow process and design choices are both made to simplify the collection and processing of the data as well as simplify the security of any sensitive materials within the project.

Any interaction with data inputs should be handled by the data collection step. This step should only make requests and not take input from potential non-data sources.

Any interaction with user inputs should be handled by the front end step. Special care should be taken to ensure only specific data and products of the processing should be output to the users. Any user input should be filtered or abstracted to ensure no malicious users can gain access to sensitive data.

4 API Details

4.1 Functions

4.2 Classes/Structures