Software Design Document

for

Climate Trace
(Central America)

Version 1.1

Prepared by Vitaliy Stepanov
Naoki Lucas
Andre Weertman
Max Ayala

WattTime

December 1, 2020

Table of Contents

1. Introduction **3**
1.1 Purpose 3
1.2 Product Scope 3
1.3 Definitions, Acronyms and Abbreviations 3
1.4 References 3
2. System Overview **4**
3. System Components **5**
3.1 Decomposition Description 5
3.2 Dependency Description 5
3.3 Interface Description 7
3.3.1 Module Interfaces 7
3.3.2 User Interfaces (GUI) 7
4. Detailed Design **8**
4.1 Module Detailed Design 8
4.2 Data Detailed Design 9
4.3 RTM 10

Revision History

Name	Date	Reason For Changes	Version
Max Ayala	12/1/2020	MongoDB turned out to be more cost effective	1.1
Max Ayala	12/1/2020	The overall architecture was too confusing	1.1

1. Introduction

1.1 Purpose

The purpose of the software design document (SDD) is to describe the architecture and system design of the website scrapers, forecasting model, and GUI for visualizing that data. It shows diagrams to engineers, designers, and product managers on how all the components should be created and interact with each other.

1.2 Product Scope

This document describes the implementation of this project and its three main components: the web scrapers and the system that continuously runs them on a timed interval, the forecasting model that uses data along with tensorflow and other machine learning libraries to produce forecasted data for the future, and the main GUI/App with which the user can see the data visualized in a map or chart form.

1.3 Definitions, Acronyms and Abbreviations

- MongoDM - free non-relational (NoSQL) database
- DB - database
- Document DB - a document based non-relational database (NoSQL)
- GUI - Graphical User Interface
- Use Case Diagram - Shows potential use cases on an actor by actor basis
- Naive Bayes Classifier - a probability-based machine learning algorithm

1.4 References

"Adapter Pattern - GeeksforGeeks." GeeksforGeeks, 3 May 2016, www.geeksforgeeks.org/adapter-pattern/. Accessed 12 Nov. 2020.

Layered Architecture. "Software Architecture Patterns." O'Reilly Online Learning, 2020, www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch01.html. Accessed 12 Nov. 2020.

2. System Overview

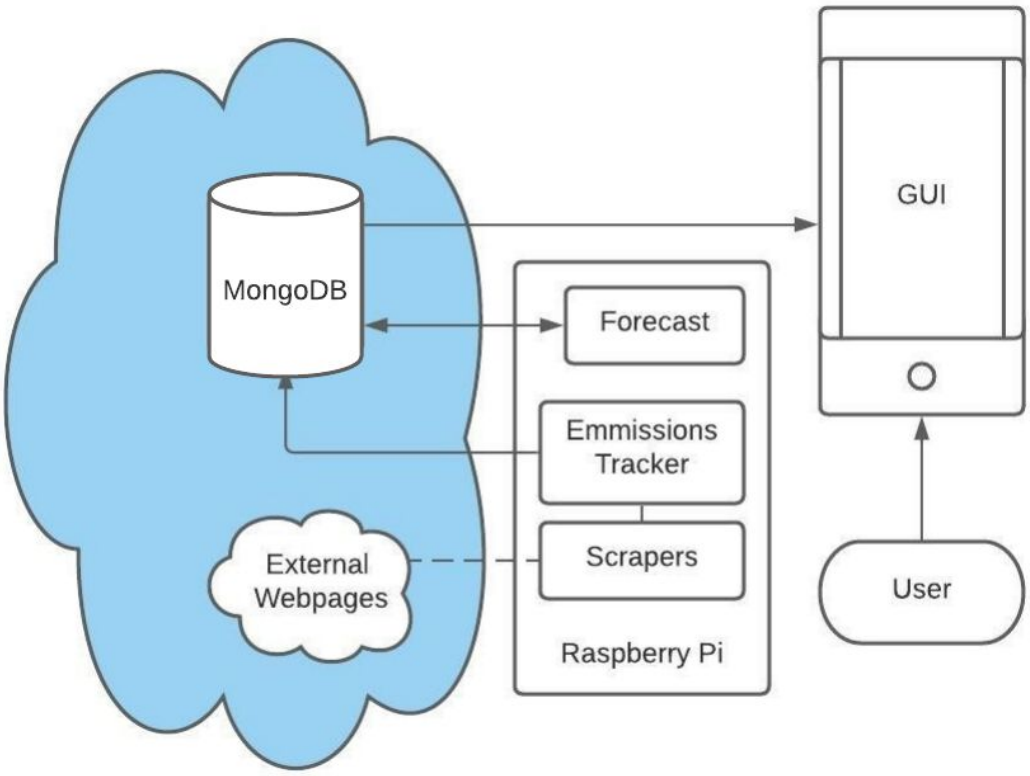

FIGURE 2.1

2.1 The Cloud

External Webpages:

- Each country present in our model has its own unique webpage from which it reports it's electrical generation data.
- These webpages are all different, they report data in different formats and with varying degrees of availability and accuracy.

MongoDB:

- The data that is scraped from external web pages will eventually return to the cloud as it will be stored in a database using MongoDB
- The data in the database will follow a certain format and will therefore be easy to access and manipulate for our visualization tool.

2.2 Raspberry Pi

The Raspberry Pi will be set up to automate scraping and forecasting of the data. It will be running two main programs on timed intervals

Emissions Tracker:

- This program is responsible for scraping the data, reformatting it, and storing it to the database.
- It will implement the scrapers using an adapter pattern as described in section 3.2.2

Forecast:

- This process use libraries such as TensorFlow to run machine learning algorithms on scraped data and build a predictive model
- It will then stored the predicted data back into the database

2.3 Application

The application will be a computer or mobile app that will be implemented with unity. The end result will be a visual app that can generate various maps and charts of the data. It should be capable of viewing real and predicted data, as well as more specific representations of data by country and in certain time frames.

3. System Components

3.1 Decomposition Description

3.2 Dependency Description

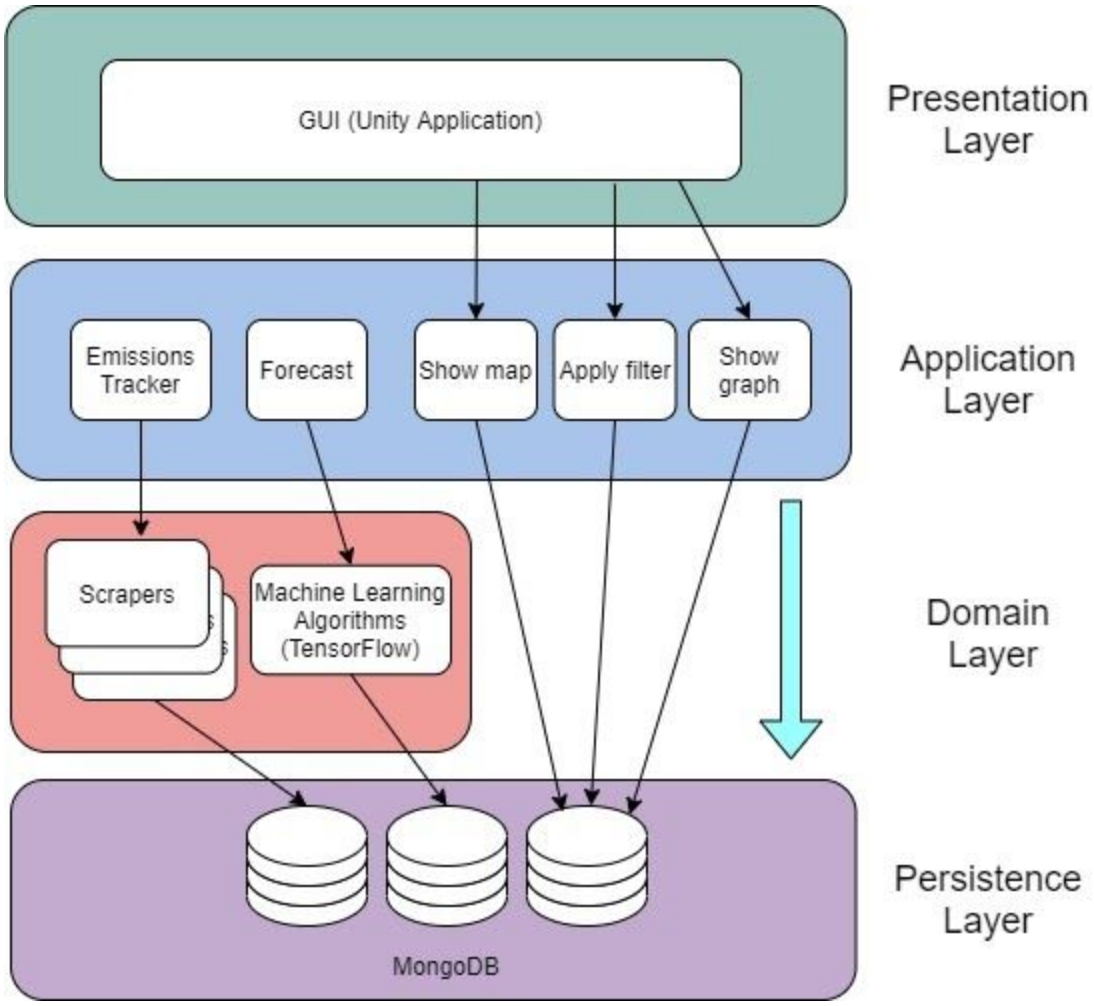

FIGURE 3.1

Figure 3.1 describes the overall architecture for this project which is based on a layered architecture, with all the dependencies flowing downward. The domain layer is open, meaning that some processes in the application layer will bypass it and communicate directly with our database. Overall each layer is independent with lower layers are unaware of higher ones and should be unaffected by any changes higher up. The main benefit to this approach is that each of these layers classes can be isolated for better testing.

3.2.1 Presentation Layer

The presentation Layer consists of the unity app that users can access to view the data. It will be capable of receiving user input which will direct it to generate various views. For these views the app will call on processes in the application layer. It will be dependent on these visualization tools as well as various C# graphics libraries for the GUI.

3.2.2 Application Layer

3.2.2.1 Emissions Tracker

This is the program responsible for automated scraping. It will be dependent on the scrapers themselves. It will use the various scrapers for each country through an adapter. This enables a lot of flexibility in creating the scrapers, and the ability to potentially use scrapers that we did not write without directly modifying them.

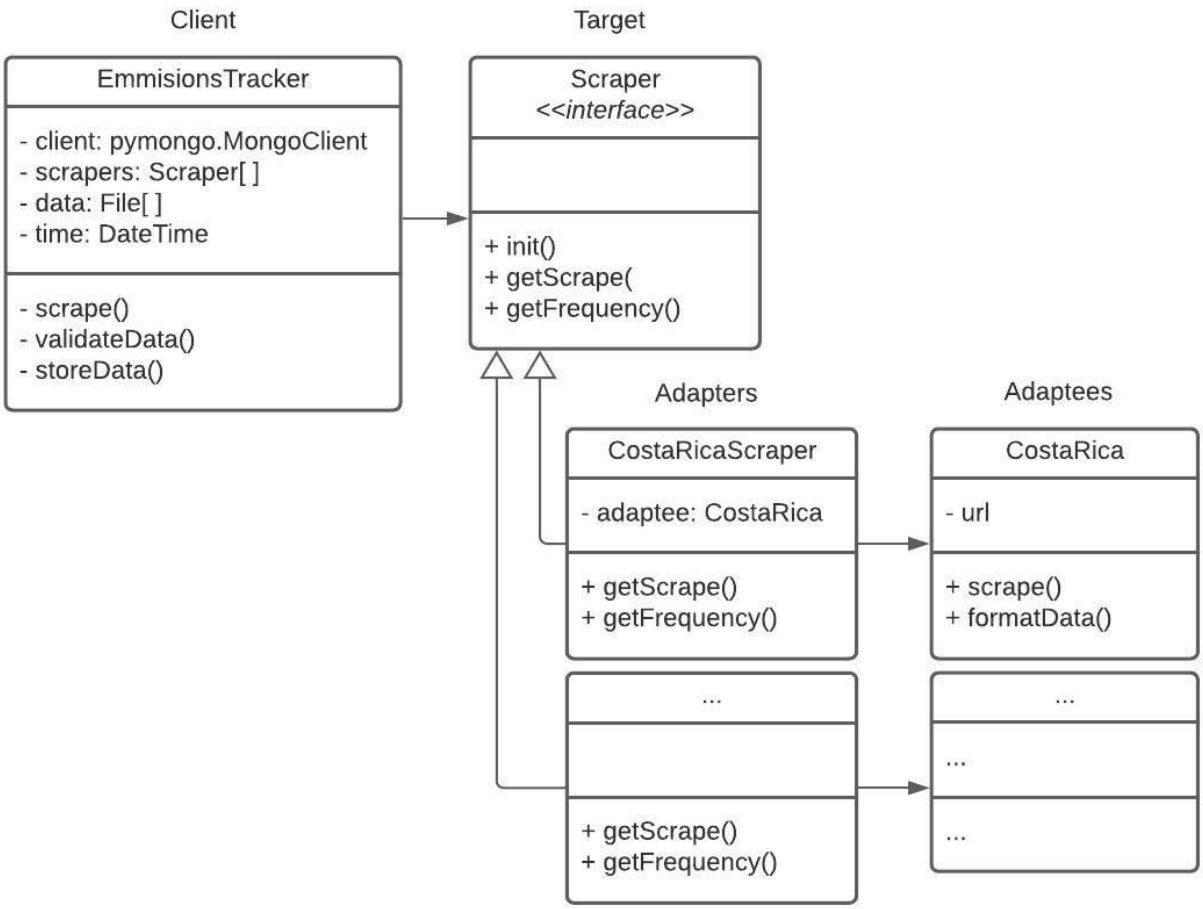

FIGURE 3.2

Figure 3.2 illustrates the design of the scrapers adapters

3.2.2.2 Forecast

The forecast tool will be dependent on various outside libraries used for machine learning.

- Tensor Flow
- PyTorch

3.2.2.3 Visualization tools

There will be a variety of visualization tools that can generate various view of the data. The filter tool will allow us to apply a filter to the map to show predicted future changes or approximate emissions from generation. The graph tool will create area charts for the data in a specific country or time frame. All of these tools will essentially depend on the accuracy of the database.

3.2.3 Domain Layer

The domain layer consists of the business logic processes, which in our project correspond to the physical scrapers and the machine learning algorithms. The machine learning algorithms will be implemented using outside libraries. The scrapers however will be written ourselves. Each scraper has a dependency on its corresponding website. These sites are prone to changes and as such the scrapers will likely require updates. Furthermore the scrapers themselves depend on a variety of outside libraries:

- BeautifulSoup
- Selenium
- Requests
- Pandas

3.2.4 Persistence Layer

The persistence layer consists of the Database. For this project we have settled on using MongoDB because it is more cost effective than the alternatives (namely Amazon Web Services). MongoDB

3.3 Interface Description

3.3.1 Module Interfaces

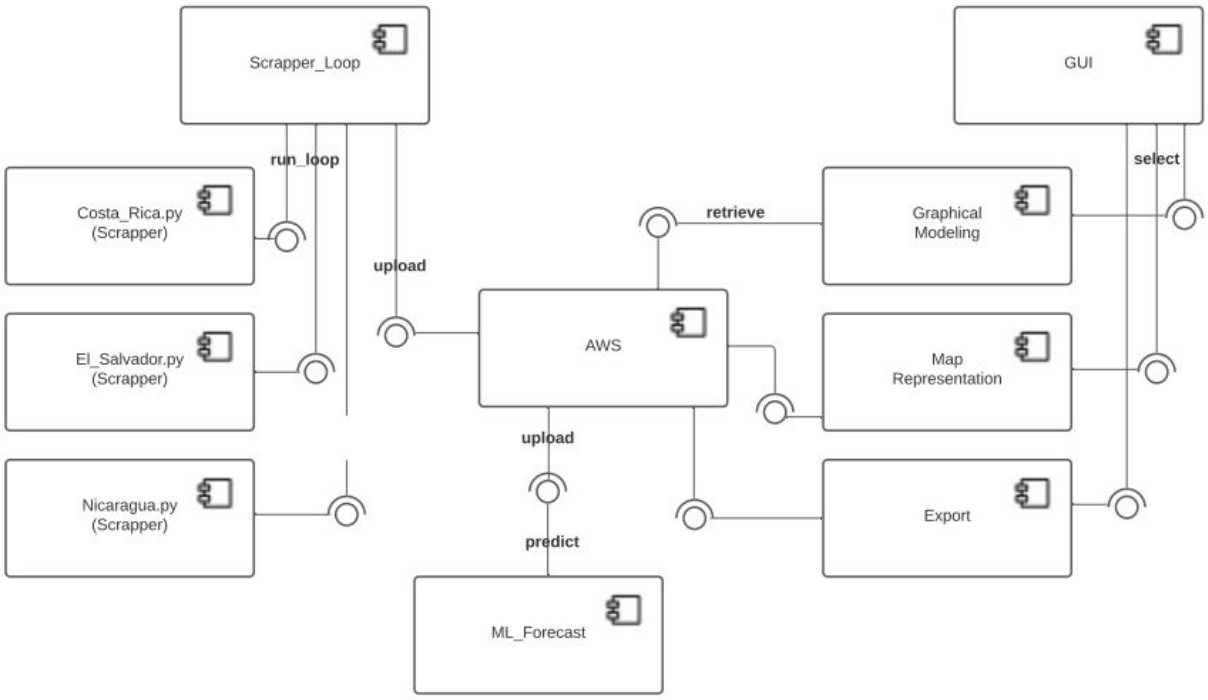

FIGURE 3.3

Figure 3.3 represents the software interface and the relationship between our software components. Everything is centralized around our MongoDB, which is constantly updated by our loop controlled scrappers. The ML_forecast software pulls data, forecasts, and stores prediction models back in our database which gets used by the different components of our GUI.

3.3.2 User Interfaces

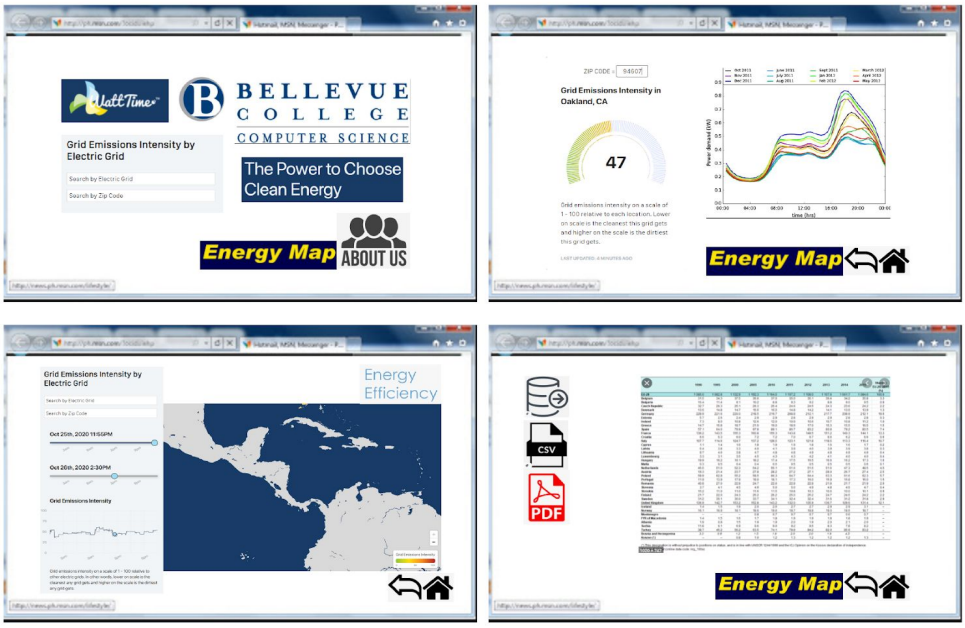

FIGURE 3.4

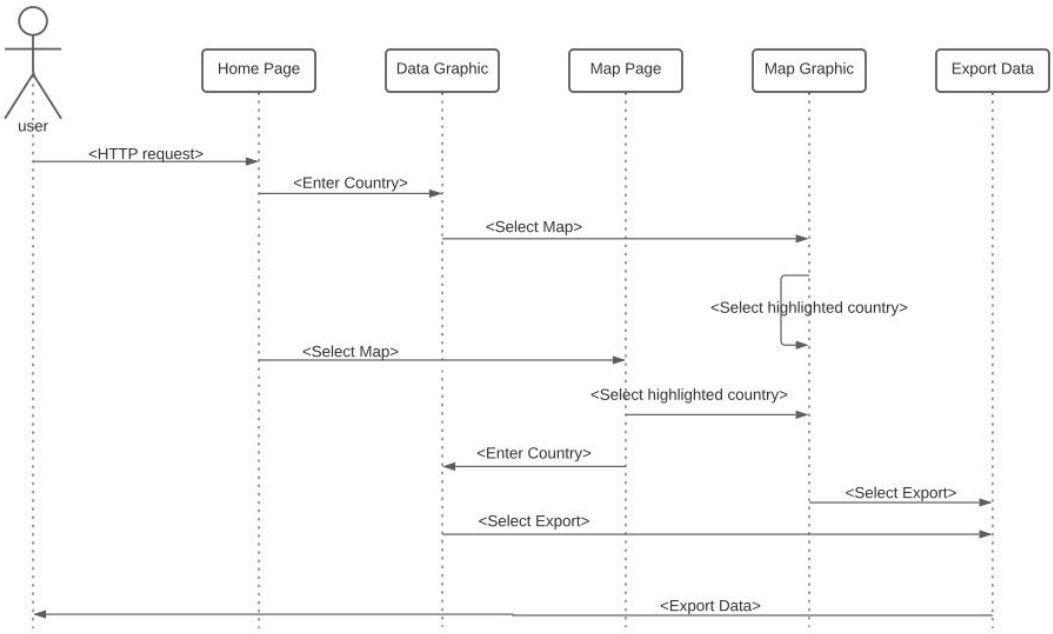

FIGURE 3.5

Figure 3.4 is the GUI representation of our website API. There are 4 main components. A home screen, data graphics, map feature, and export data. The sequence relationship between them is represented in Figure 3.5 sequence diagram.

4. Detailed Design

4.1 Module Detailed Design

```
DATA COLLECTION:
    WHILE we are continuing to collect data (this whole year)
        FOR each country on our scraping list:
            IF (country data is represented only hourly)
                Run the scraper a the top of every hour
                Save the data to the database
            ELSE IF (entire day data is represented and divided down by hour
        segments)
                Run the scraper once a day (probably midnight)
                Save the data to the database
            ELSE (special case)
                Run the special case data collection
                Save the data to the database

DATA CONFIGURATION:
    WHILE data is continuing to be collected
        FOR (each countries data set)
            IF (no forecast has been predicted)
                Run our forecast algorithm across data set
                Build an entire day forecast of energy usage presented hourly
            ELSE (we already have a forecast model)
                Run our forecast algorithm over newly collected data
                Commit changes to forecasted model based on new data updates
    RETURN the prediction modeled forecasted daily data

USER INTERFACE:
    WHILE any user is navigating our GUI interface
        IF (user selects a specific country)
            Grab the data returned from our forecast model specific to that location
            Graph the data in an area graph of hourly usage over the course of the day
            ENDIF users next selection
        ELSE IF (user selects our map feature)
            Open world map
            IF (user selects a highlighted country)
                Grab the data returned from our forecast model specific to that
            location
                Present the graphical data of the selected country
                ENDIF user selects a different country
            ENDIF user selects a different feature
        ENDIF user closes website window
```

4.2 Data Detailed Design

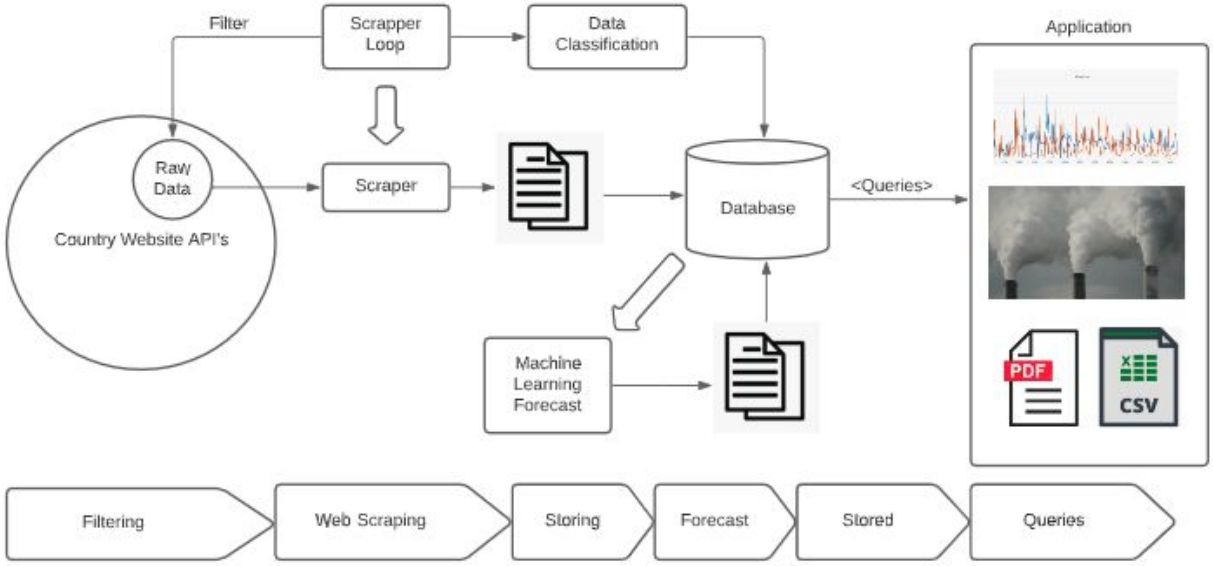

FIGURE 4.1

Figure 4.1 is a data flow diagram. Data is gathered by scrapers and organized onto a database. The way this will be organized is by country and by ‘tense’ (past, present and future). The program that organizes the data will analyze the oldest data in the ‘present’ documents (no older than 30 days for example) and move them to the ‘past’ documents if applicable. These scrapers will visit specific target websites and gather information. The data is then read from the database, analyzed, and forecasted data will be uploaded onto the database. This forecast will be created using ML/algorithms that will create data by the hour. This database can be accessed directly from users with read-only access and will also be used by our application/s.

The data is stored in MongoDB as json document files.
They will be organized by country and time and pollution type.
The diagram displays how the data will be processed.

4.3 RTM (Requirements Traceability Matrix) (after system is defined)

Req. #	Requirement	Design Specification	Program Module	Test Specification	Test Case(s) Numbers	Successful Test Verification	Modification of Requirement	Remarks
1	Scraper must retrieve data and in dictionary format.	3.2.1 Data Scrapers	3.2.1.1	All 4 scraper classes	37	Yes		Passed
2	User can filter data by pollution type	3.2.2 Visualization Tool	3.2.1.1					
3	User can view specific power plants	3.2.2 Visualization Tool	3.2.2.2					
4	User can see past, present, and future data	3.2.2 Visualization Tool	3.2.2.3					
5	Users can enter a zip code, or country, and access our cleanliness rating, meter, and hourly graph of the energy usage at that specific location	3.2.2 Visualization Tool	3.2.2.4					
6	Users can click our map feature, and scroll to any country in the world, and find our data cleanliness data mapped to the countries our scrapers pertain to.	3.2.2 Visualization Tool	3.2.2.5					
7	Map features will highlight countries by	3.2.2 Visualization Tool	3.2.2.6					

	a ‘green’, ‘yellow’, or ‘red’ indicated their overall energy cleanly gauge							
8	Users will be able to return to the home page from our map page, or by clicking on a country, go to our data page for that country	3.2.2 Visualization Tool	3.2.2.7					
9	Users can export our raw data in dictionary format or a graphical representation of the hourly rating throughout the day.	3.2.2 Visualization Tool	3.2.2.8					
10	Robust back-up data	3.3 Performance > 3.3.2 Storage	3.3.2.2					