

Wastewater Modeling

Ayaan Sunesara

Quinlon Horndasch

Matthew Delorenzo

Grace Salau

CONCEPT OF OPERATIONS

REVISION – 2

4 December 2022

CONCEPT OF OPERATIONS FOR Wastewater Modeling

TEAM <15>

APPROVED BY:

<u>Quinlon Horndasch</u>	<u>12/04/22</u>
Project Leader	Date

<u>Prof. Kalafatis</u>	<u> </u>
	Date

<u>Dalton Cyr</u>	<u>12/04/22</u>
T/A	Date

Change Record

Rev	Date	Originator	Approvals	Description
0	09/15/2022	Team 15	Quinlon Horndasch	Original Release
1	10/03/2022	Team 15	Quinlon Horndasch	Midterm Report
2	12/04/2022	Team 15	Quinlon Horndasch	Final Report

Table of Contents

Table of Contents	III
List of Tables	IV
List of Figures	V
1. Executive Summary	1
2. Introduction	2
2.1. Background	2
2.2. Overview	2
2.3. Referenced Documents and Standards	3
3. Operating Concept	4
3.1. Scope	4
3.2. Operational Description and Constraints	4
3.3. System Description	4
3.4.1. Backend	4
3.4.1.1. Data Collection and Processing	4
3.4.1.2. Data Analysis	5
3.4.1.3. Data Projection (Trend Algorithm)	5
3.4.2. Front End	5
3.4. Modes of Operation	5
3.5. Users	6
3.6. Support	6
4. Scenario(s)	7
4.1. LANL Application	7
4.2. Public Application	7
4.3. Expansion of Tracked items	7
5. Analysis	7
5.1. Summary of Proposed Improvements	7
5.2. Disadvantages and Limitations	7
5.3. Alternatives	8
5.3.1. Constraints	8
5.3.2. Benefits	8
5.4. Impact	8

List of Tables

Table 1. Reference documents.....	3
-----------------------------------	---

List of Figures

Figure 1. System Overview.....	3
--------------------------------	---

1. Executive Summary

Los Alamos National Laboratory (LANL) has requested a browser-based application to track COVID-19 cases through wastewater testing. Currently, LANL (along with most other major government agencies) uses reported COVID-19 cases as an indicator of which areas are at risk or currently hotspots for the virus. Unfortunately, this form of tracking COVID-19 has limitations due to inconsistency of reporting, asymptomatic cases, and the long time between testing and results. However, with new strategies like wastewater research, COVID-19 data can be gathered earlier and more accurately. To take advantage of this strategy, the wastewater modeling application must allow for the input, analysis, and storage of data in the cloud, while also organizing the data for easy viewing. This will allow LANL management to make more informed decisions with regards to dealing with and preventing further spread of the virus. The application will satisfy the growing need for a more accurate testing option while creating an opportunity to counter the virus in the early stages rather than react to the aftermath.

2. Introduction

This document is an introduction to wastewater modeling, using the virus levels in a county's wastewater as a predictive measure to the correlating reported cases of the virus. In this instance, the virus in focus is SARS-CoV-2 (COVID-19) but this can be outreached to deal with other viruses, such as the flu. COVID-19 wastewater modeling will serve as a measure for LANL management to take precautionary measures for their company along with their families if the trend between COVID-19 wastewater levels correlates with the number of reported cases.

2.1. Background

COVID-19 is a virus that stopped the world in its tracks by onsetting a worldwide pandemic that had millions of people on lockdown in the United States alone until a vaccine was developed. During this period of panic researchers and scientists developed and implemented various tools to help control as well as to understand the virus at hand - wastewater modeling was one of these measures. Individuals that are infected with COVID-19 shed the virus through their feces before symptoms are present even if they are fully asymptomatic. As a result, the levels of COVID-19 can be tested through wastewater, enabling the data to be a potential predictive model as to whether cases could increase or decrease within a certain span of time.

Similar wastewater modeling websites have been created for other cities, but the application will be oriented specifically to Los Alamos, New Mexico. Through wastewater modeling, the project could eventually be able to predict the hotspots of COVID-19 and potentially apply this process, algorithm and data analysis to other viruses and diseases, such as the flu or monkeypox.

2.2. Overview

The Web Development app will act as a predictive model for the rise of COVID-19 cases. Once the data is analyzed, a predictive algorithm and model will be created to represent the relation of reported COVID-19 cases and the level of COVID-19 in Los Alamos's wastewater data. After, then use the recorded data to predict what the trend of COVID-19 could be for about a week into the future and use it as a preventive measure to inform individuals and Los Alamos management to take preventative measures like wearing masks or having employees work from home for an allotted amount of time. Below is a system overview that helps visualize how the system will interact and communicate, the two major systems are the frontend and backend which all have their supplemental subsystems that are necessary for the functionality of the system. Essentially data that details, Covid -19 wastewater levels as well as clinical cases, will be held in the AWS server which will communicate with data processing to parse through data and organize it, which will then be fed into the algorithm to create a prediction which will then be shown on the frontend GUI.

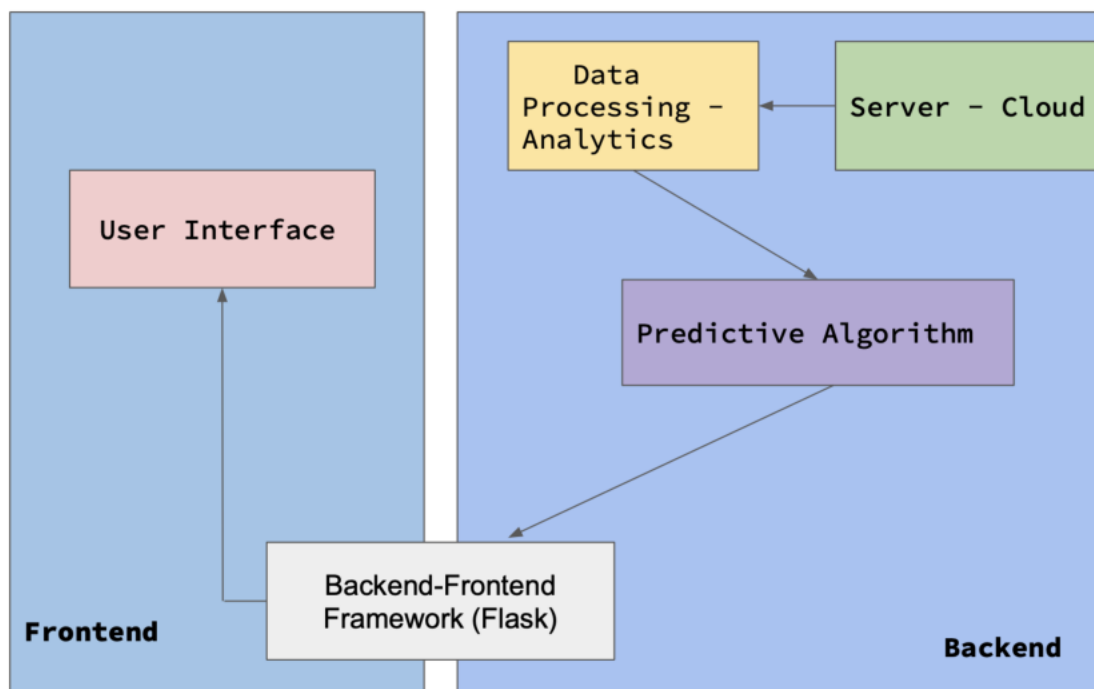


Figure 1. System Overview

2.3. Referenced Documents and Standards

Document Name	Revision/Release Date	Publisher
Heroku Manual	7.0	Salesforce Inc.
Python2.6	2.6	Python Software Foundation
AWS EC2	4.9.458	Amazon Inc.
Flask	2.2.2	Python Software Foundation
Python Library Reference	3.10.7	Python Software Foundation

Table 1. Reference documents

3. Operating Concept

3.1. Scope

The proposed project requires creating a web application. The web application will contain predicted COVID-19 cases about a week out from the current date which will be presented in a graphical format. The algorithm to calculate the number of predicted cases will be developed in the backend of the application through the utilization of wastewater data provided by the representative of a local county. The output of the algorithm will be presented in the graphical representation on the web application for LANL management to make conclusions from.

3.2. Operational Description and Constraints

This project will allow Los Alamos National Laboratory (LANL) to track the cases of COVID-19 via testing the local wastewater. This project requires developing an algorithm that creates a prediction of what the cases will look like about a week in advance. Having a week-long warning can allow the laboratory to take precautions based on the resulting trend. This allows LANL to tailor their work environment, if test cases are rising or falling, to the current threat level of COVID-19. Additionally, this project can be used on a larger scale such as tracking the COVID-19 spread between multiple counties, by testing their wastewater sources and creating a representative model for it. This project has the potential to be expanded to other viruses or diseases, such as the flu or monkeypox, through additional wastewater data testing and modeling.

3.3. System Description

The web application consists of two primary subsystems - backend and frontend development. The back end of the application consists of data storage, hosted through a cloud storage medium, such as Amazon Web Services (AWS). The backend will consist of the subsystems that handle the data, such as initial collection/processing, analysis, and projections that utilizes python 3.6 and various libraries.

The frontend will then take these results and present them through a user interface, enabling users to effectively view correlations and projections between the recent wastewater trends to the COVID-19 cases within their region. Wastewater data will also be able to be uploaded by certain users (managers or data analysts) if needed.

3.3.1. Backend

3.3.1.1. Data Collection and Processing

This stage is responsible for the collection and processing of wastewater data (primarily COVID-19 levels) and the corresponding COVID-19 cases from that location. Although the ultimate source of data will come from Los Alamos Labs, the initial collections will come from local wastewater systems within the state, such as Brazos County (depending on availability). The COVID-19 positive cases reported within the location will then be collected from official government sources (such as [texas.gov](https://www.texas.gov) and [cdc.gov](https://www.cdc.gov) if available). The data will likely come in various forms (Excel files, tables, or string/dictionary formats), a data processing program (Python or MySQL) will be utilized to standardize it all to a single format.

3.3.1.2. Data Analysis

With all data processed uniformly, it can then be analyzed to determine potential trends between the COVID-19 levels in the wastewater and the number of reported cases. This includes plotting the COVID-19 wastewater levels over a predetermined span of time, cleaning the data by removing extreme outliers from the sample, and determining a trendline that best fits the data. The same is then done for the corresponding reported number of COVID-19 cases.

The correlation between the wastewater and the number of COVID-19 cases can be modeled and compared through determining the variances between the trendlines. Depending on the strength of the correlation, the results in the model can be used to draw conclusions based on various emergent qualities. This can include determining the accuracy of the model to the location through deviance of the wastewater data from reported cases and finding the amount of time ahead. The wastewater model can also determine trends by how left translated it appears as compared to the reported cases.

3.3.1.3. Data Projection (Trend Algorithm)

With the wastewater data trends modeled for a specific location, and then project them further into the future. This can initially be done purely by extending the determined trendline forward, such as in slopes depicting the beginning of an upward trend. This has the limitations of how accurate the trendline model is to the location (from data analysis).

However, to make these predictions more advanced than trendline extension, machine learning may be utilized to make more informed predictions. This component is reliant upon the amount of data available to be utilized for supervised learning and testing, such as previous wastewater data and the reported COVID-19 cases. This could be done through a Python/MATLAB training algorithm.

3.3.2. Front End

The front end consists of the user interface in which the user will be able to effectively view and interact with the contents of the web application. This consists of taking the resulting data from the backend and converting it to an easy to view model. This requires the web app to be hosted publicly for users to have access to the web app, which can be done through deployment tools such as Heroku. Then, a link between the backend-data (Flask which uses Python) will need to be established to the HTML of the application, which can be accomplished through a web framework like Flask. Finally, the data on the webpage will then be formatted such that the webpage can be viewed and interacted with easily, which is done through CSS and HTML styling.

3.4. Modes of Operations

With the requirements presented to us by the Los Alamos Laboratory, there is only one main mode of operation. This is where the developer of the COVID-19 wastewater data can upload such data to the web application and present that community with their number of predicted cases of COVID-19 by about a week into the future. If there are any requirements that are presented to us in the future relating to the modes of operation, this section will be updated accordingly.

3.5. Users

The primary use of the COVID-19 wastewater web application will be for management at the Los Alamos Laboratory to allow them to have access to a system to make informed decisions on company protocols (COVID-19 related). Individuals who are permitted to upload additional wastewater data, will be able to do so through the web-application. Since the predicted trends will be shown using a web application, there is no training that is required for installing or using the web application. Workers at the laboratory will benefit from it because they can start taking precautionary measures if cases seem to be rising from the predicted trend to keep themselves and others safe at the laboratory and in the county.

3.6. Support

To support the user's interaction with the web application, a sort of tech support would be given by providing contact information to the user on the web application to report any issues or questions with the application. A user manual is not necessary for the user as the UI of the web application will be sufficient for most of its use cases. Any involved tasks (such as uploading additional data) will have clear directions in the application.

4. Scenario(s)

4.1. LANL Application

The primary use of the wastewater modeling application will be to provide LANL managers a tool to be knowledgeable about and prevent the spread of COVID-19 in the LANL facility. Currently, LANL (among other major institutions) must wait for long periods of time to get accurate test results of COVID-19. This prevents them from stopping the virus from spreading within the facility. However, by using, analyzing, and presenting the wastewater data, in an easy to upload and read format, managers can stay ahead of the curve rather than react after the fact. Wastewater will be continuously tested and uploaded for an accurate, fast, and steady stream of COVID-19 results.

4.2. Public Application

Much like LANL, the public and businesses also have the same issue of getting reliable and current COVID-19 data. This can create a false sense of security for many because an area can be inaccurate to the current state. With the use of the wastewater application, this issue can be addressed through an early and representative depiction of Covid-19. With the use of the application in certain locations, the public will be able to make informed decisions on when to wear masks, stay out of highly condensed areas and practice proper social distancing.

4.3. Expansion of Tracked Items

Wastewater allows for faster COVID-19 test results but is not the only thing trackable through wastewater. Much like with COVID-19, the flu, monkeypox, and even marijuana can be tested through wastewater data. These different tests could create expansion of the application or create different versions pertaining to each individual test. Much like with COVID-19 cases this can create opportunity for both awareness and prevention of each case. Not only would this be a good library for viruses' historic spread in certain areas but also creates an opportunity to influence real-time decisions.

5. Analysis

5.1. Summary of Proposed Improvements

- Depicts an accurate model of COVID-19 trends up about a week before reported cases data is updated.
- Easy to use interface to view previous data, correlations, and projections.
- Decreased reliance on individual data from official case number reports.

5.2. Disadvantages and Limitations

- Locations where wastewater data can be collected on a regular basis is limited.
- The amount of data that can be received regularly is limited.
- The precision/accuracy of the wastewater to detect COVID-19 is still being tested.
- The strength of the correlations between COVID-19 levels in the wastewater and the subsequent reported COVID-19 cases.

5.3. Alternatives

As is the case with most cities, Los Alamos is utilizing the official COVID-19 cases reported from New Mexico to analyze trends. This system results in constraints that can be addressed through the utilization of wastewater modeling:

5.3.1. Constraints

- The unreliable availability and access to data from the state, as current government policies regarding COVID-19 data may be influential factors.
- The need for access to individual data (COVID-19 case results).
- Accuracy of COVID-19 positive cases (false positives and/or unreported positive cases).
- Timing of the positive COVID-19 reported data is post-infection, making preventative measures to limit the spread less effective.

5.2.2. Benefits

- The wastewater data coming from cities rather than individual reports, resulting in a macro-view of COVID-19 spread without the reliance on data sourced directly reported from the state/government.
- The ability to obtain COVID-19 information without the need for any individual data.
- Unreported cases and other variations in reports would be better accounted for through a single wastewater measurement.
- The potential to depict COVID-19 levels before individual tests are reported, causing preventative measures to be more effective in minimizing risk.

5.4. Impact

Through enabling trends in COVID-19 cases to be effectively tracked and projected up to about a week in advance, preventative measures can be taken earlier to limit the spread of the virus, thereby benefiting the overall health of the population. The macro nature of the wastewater data also reduces the need to gather individual data, thereby providing increased privacy of individual information.

Wastewater Modeling

Matthew Delorenzo

Quinlon Horndasch

Grace Salau

Ayaan Sunesara

Functional System Requirements

FUNCTIONAL SYSTEM REQUIREMENTS FOR Wastewater Modeling

PREPARED BY:

Quinlon Horndasch 12/04/22
Author Date

APPROVED BY:

Quinlon Horndasch 12/04/22
Project Leader Date

John Lusher, P.E. Date

Dalton Cyr 12/04/22
T/A Date

Change Record

Rev	Date	Originator	Approvals	Description
0	10/03/2022	Team 15	Quinlon Horndasch	Original Release (Midterm Report)
1	12/04/2022	Team 15	Quinlon Horndasch	Final Report

Table of Contents

Table of Contents	III
List of Tables	IV
List of Figures	V
1. Introduction	1
1.1. Purpose and Scope	1
1.2. Responsibility and Change Authority	2
2. Applicable and Reference Documents	3
2.1. Applicable Documents	3
2.2. Reference Documents	3
2.3. Order of Precedence	4
3. Requirements	5
3.1. System Definition	5
3.2. Characteristics	6
3.2.1. Functional/Performance Requirements	6
3.2.1.1. Predictive Algorithm	6
3.2.1.2. Data Processing Execution	6
3.2.1.3. File Upload	7
3.2.1.4. Data Storage	7
3.2.2. Software Requirements	7
3.2.2.1. Software – Cloud Server	7
3.2.2.2. Programming Language – Predictive Algorithm	7
3.2.2.3. Programming Language – Frontend User Interface	7
3.2.2.4. Software – Backend Framework	8
3.2.3. Interface Requirements	8
3.2.3.1. File Inputs	8
3.2.3.1.1. Operating System	8
4. Support Requirements	9
Appendix A: Acronyms and Abbreviations	9
Appendix B: Definition of Terms	9

List of Tables

Table 1. Subsystem Leads.....	7
Table 2. Applicable Documents.....	8
Table 3. Reference Documents.....	8

List of Figures

Figure 1. Your Project Conceptual Image.....	1
Figure 2. Block Diagram of System.....	4

1. Introduction

1.1. Purpose and Scope

This document provides information on the subsystem requirements regarding the performance, functionality, software, and communication requirements for the system to be fully functional. Wastewater modeling is broken into two major system groups which are the front end and the backend. The front-end system involves a graphic user interface that communicates with the backend to show the predictive trends stemming from the acquired wastewater and clinical cases in the LANL area. This shows positivity rates (predicted covid trends) for the upcoming week in a graph as well as other graphics. The backend system consists of a predictive algorithm subsystem, data processing and analytics, as well as a server. The predictive algorithm is machine learning created to take in the data that's been processed and analyzed and predict the trends for the following week. The server supplied by AWS is where the data will be held for it to communicate with the algorithm as well as the frontend.

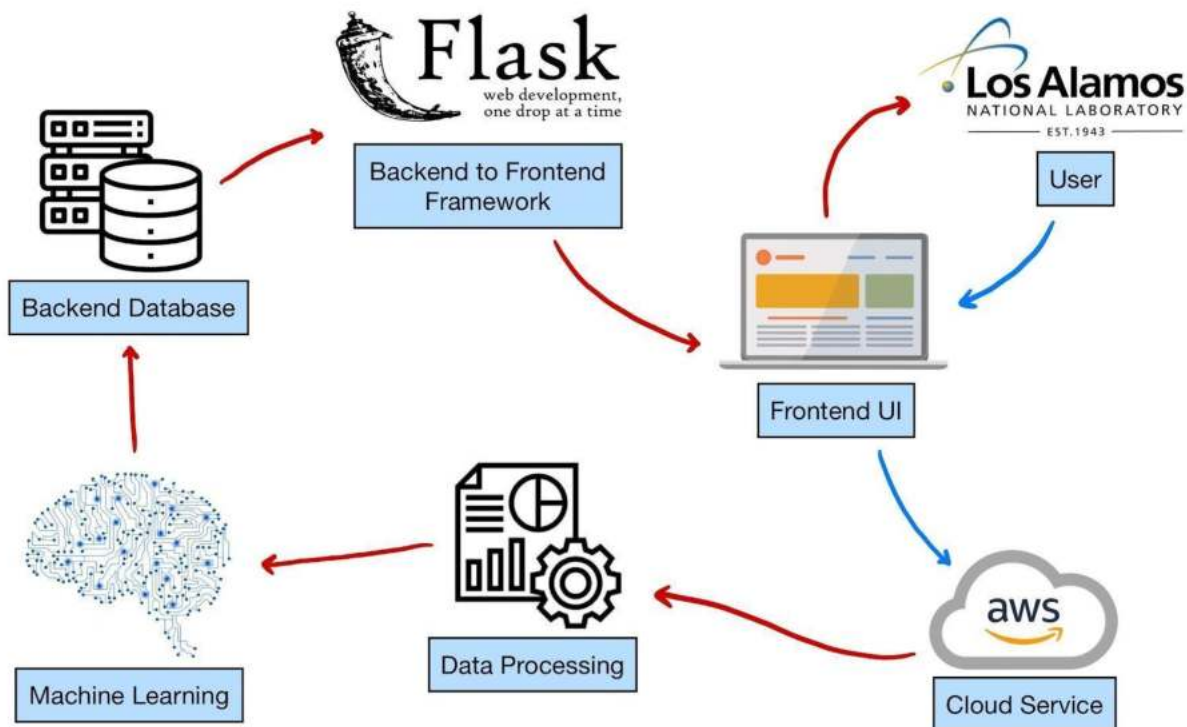


Figure 1. Functional System Diagram of Wastewater Modeling Web Application

1.2. Responsibility and Change Authority

The team leader, Quinlon Horndasch, is responsible for approving and verifying requirements of the project are met. Said requirements can only be passed with the approval of the team leader and Professor Stavros Kalafatis and prior discussion with the team and LANL sponsors.

Subsystem	Responsibility
Frontend User Interface	Quinlon Horndasch
Predictive Algorithm (Machine Learning)	Grace Salau
Backend Data Connection to Frontend (Flask)	Matthew DeLorenzo
Data Processing and Analytics	Ayaan Sunesara
AWS Server	Matthew DeLorenzo and Ayaan Sunesara

Table 1. Subsystem Leads

2. Applicable and Reference Documents

2.1. Applicable Documents

The following documents, of the exact issue and revision shown, form a part of this specification to the extent specified herein. This includes the versions of all installed major packages utilized within the application when publicly hosted. (For a more detailed breakdown of each individual software package component currently utilized, reference the requirements.txt file within the repository).

Document/Module Name	Revision/Release Date	Publisher
boto/boto3	2.49.0/1.24.9	Amazon AWS
Flask	2.0.3	Python Software Foundation
google-auth	1.35	Google Inc.
gunicorn	20.1.0	Gunicorn
Jinja2	3.0.3	Jinja
matplotlib	3.3.4	matplotlib
numpy	1.19.5	numpy
pandas	1.1.5	pandas
Python Library Reference	3.6.15	Python Software Foundation
scikit-learn	0.24.0	Scikit-learn
scipy	1.5.4	SciPy
Tensorflow/Tensorboard/Keras	2.6.0	Tensorflow/Keras
torch	1.5.0	PyTorch
s3transfer	0.5.2	PyPI

Table 2. Applicable Documents

2.2. Reference Documents

The following documents are reference documents utilized in the development of this specification. These documents do not form a part of this specification and are not controlled by their reference herein.

Module Name	Revision/Release Date	Publisher
Heroku Manual	7.0	Salesforce Inc.
Python	3.6.15	Python Software Foundation
AWS EC2	4.9.458	Amazon Inc.

Table 3. Reference Documents

2.3. *Order of Precedence*

In the event of a conflict between the text of this specification and an applicable document cited herein, the text of this specification takes precedence without any exceptions.

All specifications, standards, exhibits, drawings, or other documents that are invoked as “applicable” in this specification are incorporated as cited. All documents that are referred to within an applicable report are for guidance and information only, except ICDs that have their relevant documents considered to be incorporated as cited.

3. Requirements

3.1. System Definition

Wastewater Modeling is composed of two major core subsystems, the backend system, and the frontend system. The backend consists of three subsystems that control the data, beginning with the collection and organization of data in a storage server supplied by AWS. This is then cleaned by the data processing subsystem which enables the predictive algorithm to synthesize what the upcoming positivity rate of COVID-19 will be based on the processed and analyzed data. The resulting data is then passed to the frontend through the Flask framework. The frontend side consists of the user interface web development subsystem, which uses a GUI to present the data and predictive trends (from backend system) of COVID-19 based on the levels/copies of SARS-CoV-2 in the wastewater as well as the clinical cases in the area. Figure 2 presents a block diagram of the overview and flow of the system, to show how each subsystem communicates and flows to present the finished product.

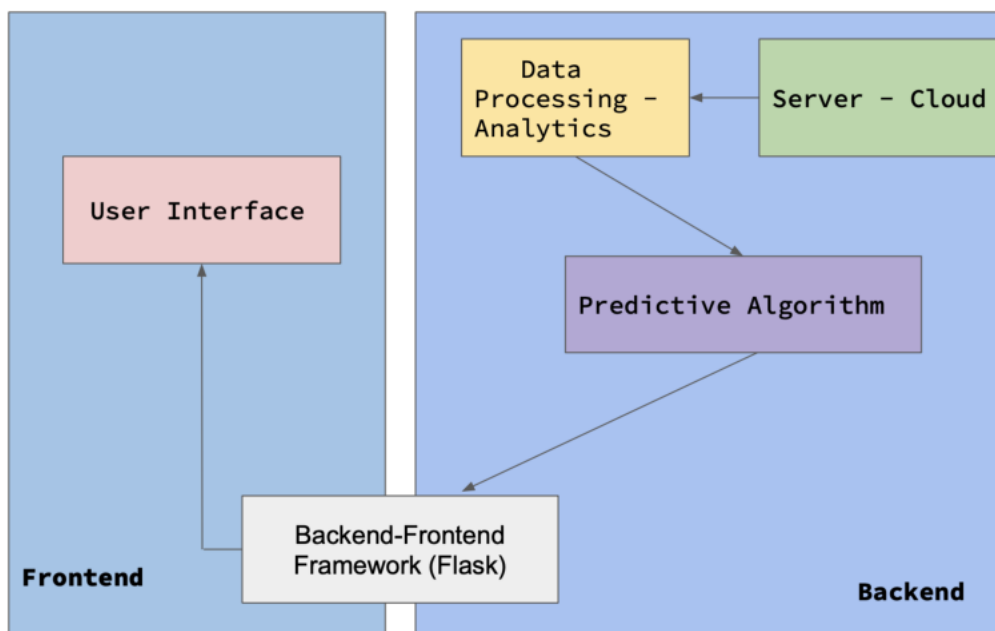


Figure 2. Block Diagram of System

AWS Server: The AWS server will be used to store the excel data automatically when it is uploaded onto the website. There are frameworks in AWS which processes the excel data that can then be used in Python for calculations. Once the data is processed, Flask can be utilized to grab any data or graphs stored in the AWS server to be posted as an output in real time.

Data Processing/Analytics: Processing of the data is composed of reading the wastewater numbers from the excel sheet that is uploaded onto the frontend of the web

application to calculate the amount of positive COVID-19 cases. Once the wastewater data is parsed and the calculations are stored to be used for the predictive algorithm, a trendline graph of COVID-19 cases will be generated.

Predictive Algorithm: The predictive algorithm consists of using machine learning to get accurate predictions of the positivity rates and COVID-19 trends based on a plethora of parameters. Some of these parameters include but are not limited to population size and density of the region, levels/copies of the SARS-CoV-2 virus in the wastewater and the number of clinical cases in the area. The combination of these parameters gives a solid basis for creating a neural network or combination of deep machine learning algorithms to give an accurate prediction. The creation of this algorithm will be created using python and accessing libraries such as TensorFlow, NumPy and matplotlib.

Backend Framework to UI: The resulting prediction data structure generated from the algorithm can then be accessed and passed to the frontend of the web application. This is done through utilizing a Flask microframework. This provides tools such as the Jinja template library that enables the web-app to render specified backend data from a Python program into the HTML, which is then displayed within the user's browser. Flask can also be utilized in retrieving user input data from the UI if needed, such as HTML forms, to communicate back to the backend subsystems.

User Interface: The user interface consists of the GUI that interacts with the user to see the graphed trends/predictions of the wastewater's COVID-19 levels. It will also include interactive tabs, consisting of a function where the user (primarily LANL management) can upload data which then communicates with the backend system and displays the predictive trend for the data uploaded.

3.2. Characteristics

3.2.1. Functional/Performance Requirements

3.2.1.1. Predictive Algorithm

The predictive algorithm predicts COVID-19 positivity rate about a week into the future with 85% accuracy to prove the created algorithm is valid.

Rationale: Having a predictive algorithm within the range of 85% depicts a realistic outcome as compared to around >70% accurate where the algorithm is considered ideal rather than realistic.

3.2.1.2. Data Processing Execution

The total time necessary to parse and clean the data introduced by the server should not exceed 60 seconds.

Rationale: Giving 60 seconds as the time frame to parse and clean the data is reasonable due to the runtime for looping through data which can take some time depending on the size of the

dataset. The code needs to traverse the data set and remove unnecessary data which can be optimized in the future.

3.2.1.3. File Upload

The frontend user interface will have a functional tab which provides the user with a file upload option. This system must work 100% of the time and allow for at least 1000 data points.

Rationale: The file upload function should work 100% of the time because it is the primary way the web application will receive information. The file that is being uploaded must also take in at least 1000 data points without failure so that LANL can properly populate the web application with wastewater data.

3.2.1.4. Data Storage

The AWS Server will be able to hold all data necessary for the predictive algorithm to generate a prediction. This system will allow 16 GB of persistent storage that can be accessed at any point by the algorithm.

Rationale: As Heroku web applications have limited persistent storage, there must be a separate area where the raw data is stored necessary for the predictive algorithm to run correctly. 16 GB is chosen as an initial estimate for the upper limits of storage needed to train a machine learning program. Heroku is also compatible with AWS storage systems, allowing easier connections to be made.

3.2.2. Software Requirements

3.2.2.1. Software - Cloud Server

The AWS server will be utilized with Python as the primary programming language to ensure compatibility with backend subsystems.

Rationale: Python is chosen as the server interaction language due to it being easily readable, a simple application development tool, and to minimize the interfacing complications between subsystems.

3.2.2.2. Programming Language - Predictive Algorithm

The programming language used to create the predictive algorithm will be python 3.6 as well as TensorFlow for any machine learning requirements.

Rationale: Due to the flexibility and frameworks that Python has; it makes it easier to develop code. Additionally, there are large amounts of documentation related to machine learning in Python which will be a great source of information.

3.2.2.3. Programming Language - Frontend User Interface

The programming language used to create the website and build the frontend graphical user interface will be HTML and will be hosted through Heroku.

Rationale: Heroku allows deployment of websites via GitHub without any expenses. The website will be programmed in HTML as it is the standard language to build a website and it works well with CSS (HTML styling language).

3.2.2.4. Software - Backend Framework

The web microframework chosen to integrate the backend data into the HTML of the UI will be Flask. Flask uses its tools such as Jinja2 templating system to complete the backend to frontend connection.

Rationale: Flask was chosen to be the web app backend framework due to the data processing and predictive algorithm subsystems utilizing Python programs - Flask is ideal for lightweight and extensible Python based web applications.

3.2.3. Interface Requirements

3.2.3.1. File Inputs

The input of the data file must be in a .csv file format. There will also be a file size limitation which will be decided in the future once there is proof of what file size the data processing takes is too long to execute.

Rationale: Having the file input be restricted to a .csv format allows for a singular method of accessing the data. Additionally, a file size being too big will cause the code in the backend to run for an extended period, making the web application inefficient.

3.2.3.1.1 Operating System

The web application can be run through devices on Linux, Windows, or MacOS. However, the application is optimized for desktop/laptop devices, so mobile devices are not recommended to interact with the website.

Rationale: This web application is not limited to a specific operating system, as it is publicly hosted on a Heroku cloud platform. However, the website is not designed for mobile devices, which makes the functionality less effective for mobile users.

4. Support Requirements

Appendix A: Acronyms and Abbreviations

Below is a list of common acronyms and abbreviations used in this project.

AWS	Amazon Web Services
ANN	Artificial Neural Network
CSS	Cascading Style Sheets
CSV	Comma Delimited Value
DL	Deep Learning
GUI	Graphic User Interface
LANL	Los Alamos National Laboratory
OS	Operating System

Appendix B: Definition of Terms

Frontend: The part of the system that interacts with the user, usually contains a graphical user interface.

Backend: The part of the system that is not accessed by the user, usually a database that stores and manipulates data.

Neural Network: A subset of machine learning that models the human brain by being composed of layers, to give accurate models and analysis.

Flask: Backend framework used in web development to interact with the user interface (Connects backend to frontend).

Jinja: Library within Flask framework enabling templates to implement Python data into HTML.

TensorFlow: Machine learning library that helps coders to create dataflow graphs by using numerical computation and large-scale machine learning.

NumPy: Python library used for working with data in arrays.

Matplotlib: A visualization library in python used to create graphs and charts.

HTML: Hypertext Markup Language which is responsible for frontend development.

Wastewater Modeling

Matthew Delorenzo

Quinlon Horndasch

Grace Salau

Ayaan Sunesara

Interface Control Document

REVISION – 1
4 December 2022

INTERFACE CONTROL DOCUMENT FOR Wastewater Modeling

PREPARED BY:

<u>Quinlon Horndasch</u>	<u>12/04/22</u>
Author	Date

APPROVED BY:

<u>Quinlon Horndasch</u>	<u>12/04/22</u>
Project Leader	Date

<u>John Lusher II, P.E.</u>	<u> </u>
	Date

<u>Dalton Cyr</u>	<u>12/04/22</u>
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
0	10/03/2022	Team 15	Quinlon Horndasch	Original Release (Midterm Report)
1	12/04/2022	Team 15	Quinlon Horndasch	Final Report

Table of Contents

Table of Contents	III
List of Tables	IV
List of Figures	V
1. Overview	1
2. References and Definitions	2
2.1. References	2
2.2. Definitions	2
3. Server ↔ Data Processing Interfacing	3
4. Data Processing ↔ Predictive Algorithm Interfacing	4
5. Predictive Algorithm ↔ Flask Interfacing	5
6. Flask ↔ UI Interfacing	6
7. User Interface	6
7.1. Title Block	6
7.2. Functional Tabs	6
7.2.1. Home	6
7.2.2. File Upload	6
7.2.3. Graphs & Data	6
7.2.4. History	7
7.2.5. Github	7
7.2.6. TAMU	7
8. Validation Plan	8
9. Execution Plan	9

List of Tables

No table of figures.

List of Figure

No table of figures.

1. Overview

The Interface Control Document will provide an overview of the communications and interfaces between each software subsystem. The overall web application is structured around data analysis in the backend. First the AWS server must first collect and clean the data which is then accessed by the predictive algorithm (machine learning) for analysis and then projects its results to the frontend Heroku web application through a Flask integration. The user can then interact with the system through the GUI.

2. References and Definitions

2.1. References

Refer to section 2.2 of the Functional System Requirements (FSR) document.

2.2. Definitions

Below is a list of common acronyms and abbreviations used in this project.

AWS	Amazon Web Services
ANN	Artificial Neural Network
CSS	Cascading Style Sheets
CSV	Comma Delimited Value
DL	Deep Learning
GUI/UI	Graphic User Interface
LANL	Los Alamos National Laboratory
OS	Operating System

Frontend: The part of the system that interacts with the user, usually contains a graphical user interface.

Backend: The part of the system that is not accessed by the user, usually a database that stores and manipulates data.

Neural Network: A subset of machine learning that models the human brain by being composed of layers, to give accurate models and analysis.

Flask: Backend framework used in web development to interact with the user interface (Connects backend to frontend).

TensorFlow: Machine learning library that helps coders to create dataflow graphs by using numerical computation and large-scale machine learning.

NumPy: Python library used for working with data in arrays.

Matplotlib: A visualization library in python used to create graphs and charts.

HTML: Hypertext Markup Language which is responsible for frontend development.

3. Server ↔ Data Processing Interfacing

With the uploaded data uploaded as a csv file in the AWS S3 server, its data can then be accessed through a Python connection. This connection can be established through a Python client file, which utilizes AWS S3 tools (such as boto) and access keys. With a connected python client, the data within the uploaded files can be accessed. The data processing (including parsing the file into arrays of valid data points) can then occur through Python tools (such as pandas or numpy) and be saved to local variables. These tools enable the user-uploaded data to be processed quickly and effectively into utilizable data for other subsystems. Once the data is processed, it can be utilized in subsystems such as the predictive algorithm or the frontend-backend integration system for further use in the application.

4. Data Processing ↔ Predictive Algorithm Interfacing

With the data properly parsed, cleaned, organized, and separated into training and learning sets and the necessary parameters for the algorithm to read and separate the data is set, the data is delegated into its respective layers. The data will then be imported into the algorithm to use as a CSV file. Within the written code of the algorithm a matrix of variables for the independent data will be created by pulling data supplied from the server, as well as generating the dependent variable and then encoding categorical data. The main code for the predictive algorithm will then use the data and the created datasets are output as an accurate model of the forecasted predicted data points.

5. Predictive Algorithm ↔ Flask Interfacing

With the predictive algorithm finished, a resulting data structure (such as a pair of arrays) consisting of the date and the COVID-19 levels (known and predicted), will be generated. As this result is generated within a Python file, the Flask microframework will be integrated into it. This framework contains tools such as Jinja's "render_template()", which will be utilized to pass the data in as a parameter. Flask also involves the utilization of the "app.route()" to specify the URL of the website that handles the associated logic. With these aspects of the microframework integrated, the data can then be utilized in HTML to provide the user with an interface containing the data that was just processed.

6. Flask ↔ UI Interfacing

The HTML code of the UI now needs to display the resulting data generated by the predictive algorithm. This process is done through Flask templating. With the data structure containing the prediction results passed in through a “render_template()”, the HTML code of the UI can then utilize this template to extract the data. This can be done through Jinja’s templating engine, which enables Python code to be written and passes information from the backend to the HTML file. This is often done through double curly brackets “{{}}” to return the passed-in variable’s value. Jinja also enables a loop to be created, allowing for multiple values of a single variable (like an array/dictionary) to be an output. With this, the raw data can be extracted for the UI to depict or graph as needed.

7. User Interface

7.1. Title Block

The Title Block will include the title of the web application (Wastewater Modeling), team number (Team #15), names of the members, and tabs (described in section 7.2.). This title block will be uniform throughout the entirety of the web application. That means that regardless of what tab the user is currently in, the title block will remain. This allows the user to navigate throughout the web application without having to go back to the homepage every time a new tab is needed.

7.2. Functional Tabs

The user interface will include functional tabs to create an organized environment for the user. The tabs included are: Home, File Upload, Graphs & Data, History, GitHub, and TAMU. These tabs will allow the user to easily navigate through the web application.

7.2.1. Home

The user interface will include a home tab which will direct the user back to the homepage of the web application. The homepage will include the basic title block (discussed in section 7.1.), project description and images related to the project.

7.2.2. File Upload

The user interface will include a File Upload tab which will allow the user to upload wastewater data (in CSV format) to simultaneously train the predictive algorithm while also providing further analysis of uploaded data. This tab will include a file upload button, preview of uploaded data, explanation on how to use the function and explanation on what happens next (what tab to check for completed data analysis).

7.2.3. Graphs & Data

The user interface will include a Graphs and Data tab which will provide the user with the graphs associated with the uploaded data post calculations and predictive algorithm. The tab will include the current (most recently uploaded data) COVID-19 levels in a graphical format along with the associated prediction from the machine learning element, uploaded raw data and recommendation of action (from CDC) based on the prediction.

7.2.4. History

The user interface will include a history tab which will allow the user to view past data sets, graphs, and predictions (from the predictive algorithm). This will allow the user to compare the current trend to the predictive trend and confirm data from a previous upload.

7.2.5. GitHub

The user interface will include a GitHub tab which will bring the user directly to the Team #15 Wastewater Modeling GitHub to look at the code used to create the web application.

7.2.6. TAMU

The user interface will include a TAMU tab which will bring the user directly to the Texas A&M University website to get more insight into the University that the project is being completed in.

8. Validation Plan

Validation Plan						
Subsystem	FSR Tie-In	Test Name	Success Criteria	Methodology	Status	Engineer
Predictive Algorithm	3.2.1.1 / 3.2.2.2	Testing Functionality of Predictive algorithm and having a 50% completed algorithm	Runs without error, takes in csv file data of various sizes, and outputs sample graph	Testing corner cases as well as fundamental cases in the algorithm with a data set/data that's reliable	Validated	Grace
	3.2.1.1	85% completed algorithm with intermediate testing	Outputted predictive trend has required accuracy as well as runs without error	Using a dataset that has been deemed as reliable and running the algorithm and validating the trend against what the trend was	Validated	Grace
	3.2.1.1	100% completed algorithm testing wastewater data and validation comparison	Outputted predictive trend runs with using covid related dataset of any size	Using a dataset that is deemed as reliable and running the algorithm and validating the trend against what the trend was	Validated	Grace
	3.2.1.1	Fully functional algorithm outputs past and future data with 85% accuracy	Outputted predictive forecast has an accuracy of 85% & outputs graph with necessary data	Using a dataset and inputting more training and testing data and running the algorithm and comparing the outputted trend against what happened	Validated	Grace

User Interface	3.2.1.3	Testing functionality of tabs in the web application	Each tab correctly directs user to the specified part of the website (ie. Home tab brings user to the homepage)	Open the web application and click on each of the tabs to ensure the tabs correctly direct the user	Validated	Quinlon
	3.2.1.3	File upload function uploads data to AWS server	The user uploads the data (CSV) and the data gets properly uploaded onto the AWS server	Using the file upload button, a dataset will be uploaded. Next the AWS server will be checked to ensure the data was stored correctly	Validated	Quinlon
	3.2.1.3	Graphs properly show the uploaded data's trend and predictive algorithms prediction	The uploaded data is properly shown visually through graphs along with the predictive trend within the web application	Open the web application, click on the upload files tab and upload a dataset that has a known trend then click on the graphs and data tab to ensure the correct data is presented	Validated	Quinlon
	3.2.1.3	History tab shows past uploads, graphs, and datasets	When history tab is clicked and loaded, the previously uploaded datasets and associated graphs are visible	Open the web application, click on the upload files tab and upload multiple datasets then click on the history tab to ensure that there are multiple entries, and that the data is viewable	Validated	Quinlon

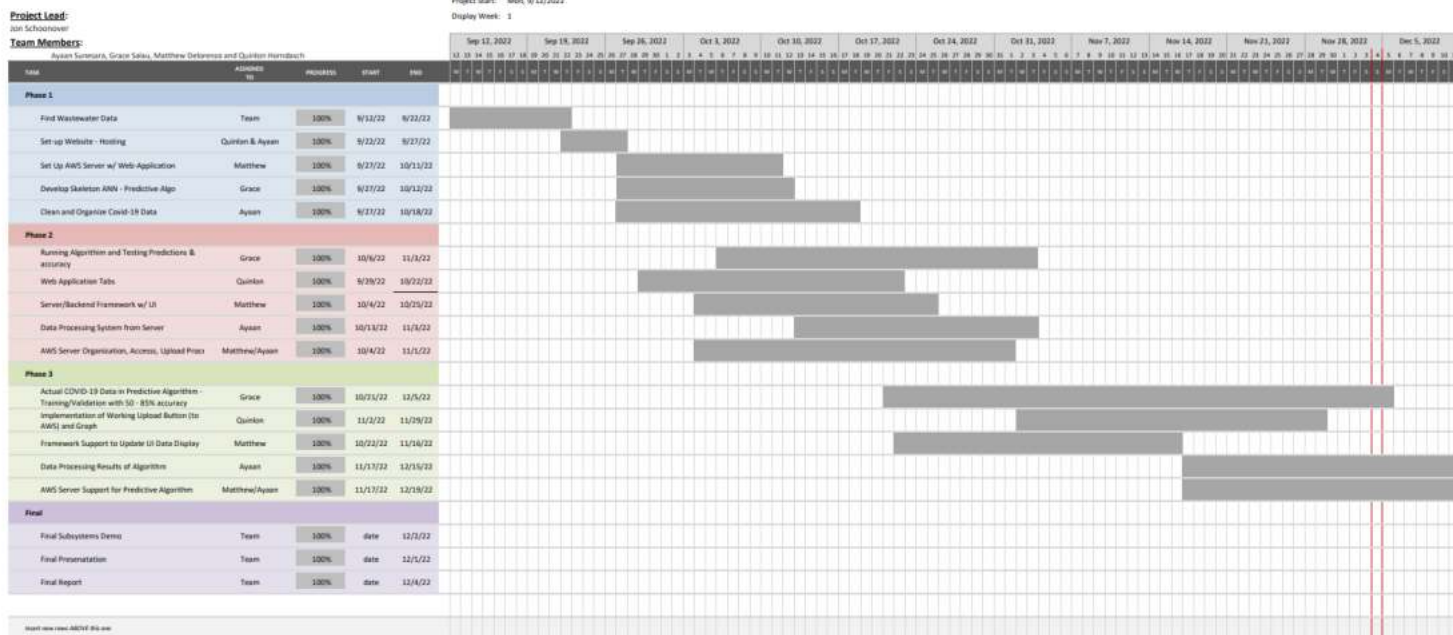
Data Processing	3.2.1.2	Downloading Data from Example Files to Analyze	Able to grab data from the excel sheets and create an array out of it from the server	Use an example data file to represent the wastewater data LANL will give to figure out if AWS Quicksight can read data values from excel sheet	Validated	Ayaan
	3.2.1.2	Parse Useful Data and Save as new variable	Remove unnecessary data from the array created to have the data and covid values only	Be able to parse the data grabbed from the excel sheets and be able to distinguish required data to save separately. (Only covid and date data can stay)	Validated	Ayaan
	3.2.1.1	Create data file and save it on a AWS Server	Plot the trend on a graph and export it back to AWS server to use	Using matplotlib library in Python to plot the x and y values, creating a graph showing a trendline	Validated	Ayaan
	3.2.3.1	Check uploaded file to be the correct format that can be parsed.	Error message is thrown in the upload file tab when an incorrect file type is uploaded and the file should not be saved on the server	Parsing the input file name and the extension of the file to check if the extension is either a csv or xlsx file. If the file doesn't match the extension, it will not be saved in the server and an error message will show up on the front end	Validated	Ayaan
Backend-Frontend Integration	3.2.2.3	Able to connect the web app backend data to the frontend.	A variable declared in a Python file can be displayed on the UI of the web app.	Utilize Flask templating to pass the variable from the Python file into a predefined portion of HTML code - test if correct value is then displayed	Validated	Matthew

	3.2.2.4	Data values from the server can be displayed.	A variable derived from the AWS Server can be displayed on the UI of the web app	Find the variable stored in AWS from the server, utilize Flask templating and routing in the file to pass the variable to a predefined HTML code portion	Validated	Matthew
	3.2.2.4	Able to push large data constructs (graphs/projections) to UI.	A larger chunk of data (such as an array variable for a graph) can be displayed to the UI	Utilize Flask to take the data structure through templating/routing, and loop through the values to display them in the HTML side	Validated	Matthew
	3.2.2.4	Can refresh presented data with an updated version when specified.	When a button is pushed on the UI, the backend can push the requested data to the frontend	When a button is pushed, triggers backend data processing to update, Flask utilized as before to push new data to the UI.	Validated	Matthew
	3.2.2.4	Can transfer user input data from the UI to the server.	Users can input data in the UI that is able to be input in the data server.	Use Flask/HTML to grab the UI data, place it in the Python code, then deposit it into the server data repository correctly.	Validated	Matthew

9. Execution Plan

Wastewater Modeling - Execution Plan

Los Alamos National Laboratory



Wastewater Modeling

Matthew Delorenzo

Quinlon Horndasch

Grace Salau

Ayaan Sunesara

Subsystem Reports

REVISION – 0
4 December 2022

SUBSYSTEM REPORTS FOR Wastewater Modeling

PREPARED BY:

<u>Quinlon Horndasch</u>	<u>12/04/22</u>
Author	Date

APPROVED BY:

<u>Quinlon Horndasch</u>	<u>12/04/22</u>
Project Leader	Date

<u>John Lusher II, P.E.</u>	<u> </u>
	Date

<u>Dalton Cyr</u>	<u>12/04/22</u>
T/A	Date

Change Record

Rev.	Date	Originator	Approvals	Description
0	12/04/2022	Team 15	Quinlon Horndasch	Original Release (Final Report)

Table of Contents

Table of Contents	III
List of Tables	V
List of Figures	VI
1. Introduction	1
2. Predictive Algorithm Subsystem Report	2
2.1. Subsystem Introduction	2
2.2. Subsystem Details	2
2.2.1. Recurrent Neural Network/Long Short Term Memory Model	2
2.2.2. Forecasted Predictions	2
2.3. Subsystem Validation	3
2.3.1. Testing Functionality of LSTM	3
2.3.2. Forecasting Functionality and Accuracy Validation	4
2.4. Subsystem Conclusion	6
3. Data Processing/Analytics Subsystem Report	7
3.1. Subsystem Introduction	7
3.2. Subsystem Details	7
3.2.1. AWS S3 Bucket Set Up	7
3.2.2. Data Parsing and Cleaning from Uploaded Data	7
3.2.3. Reshaping Predictive Algorithm	8
3.2.4. File Upload Error Checking	8
3.3. Subsystem Validation	8
3.3.1. Validation for AWS S3 Bucket Set Up	8
3.3.2. Validation for Data Parsing and Cleaning from Uploaded Data	9
3.3.3. Validation for Reshaping Predictive Algorithm	9
3.3.4. Validation for File Upload Error Checking	10
3.4. Subsystem Conclusion	11
4. Frontend/Backend Integration Subsystem Report	13
4.1. Subsystem Introduction	13
4.2. Subsystem Details	13
4.2.1. Flask Microframework Structure, Routing and Templating	13
4.2.2. Dynamic Chart Generation Algorithm	15
4.2.3. Publicly Hosted through Heroku	15
4.3. Subsystem Validation	16
4.3.1. Flask Microframework Structure, Routing and Templating	16
4.3.2. Dynamic Chart Generation Algorithm	17
4.3.3. Publicly Hosted through Heroku	18
4.4. Subsystem Conclusion	18
5. User Interface Subsystem Report	20
5.1. Subsystem Introduction	20
5.2. Subsystem Details	20
5.2.1. Title Block	20
5.2.2. Functional Tabs	20
5.2.2.1. Home	20
5.2.2.2. File Upload	21
5.2.2.3. Graphs & Data	22

5.2.2.4. History	22
5.2.2.5. GitHub	23
5.2.2.6. TAMU	23
5.2.3. File Upload Button	23
5.2.4. Graph	23
5.2.5. History Downloads	24
5.3. Subsystem Validation	24
5.3.1. Validation for Title Block	24
5.3.2. Validation for Functional Tabs	24
5.3.3. Validation for File Upload Button	25
5.3.4. Validation for Graph	26
5.3.5. Validation for History Downloads	28
5.4. Subsystem Conclusion	30
5.4.1. Progress	30
5.4.2. Difficulties and Solutions	30
5.4.3. Final Remarks	31

List of Tables

Table 1: Flask Routes and the Associated Purposes.....	14
--	----

List of Figures

Figure 1. Tested LSTM Model Results on Wastewater Data.....	3
Figure 2. Tested LSTM Model on COVID-19 Cases in Texas.....	4
Figure 3. Tested LSTM Model on Tesla Stock.....	4
Figure 4. Forecast Predictions, Actual Predictions and Percent Accuracy (11/11 to 11/17)...	5
Figure 5. Forecast Predictions, Actual predictions and Percent Accuracy (4/30 to 5/6).....	5
Figure 6. Result of Outputted Graph of Past Data and Forecasted Predictions.....	6
Figure 7. File Uploaded to the Server from the Upload Button.....	9
Figure 8. Data Set in the Excel Sheet 'testdata_2.csv'	9
Figure 9. Output of the Downloaded File from the Server to Print the Data.....	9
Figure 10. Example Output of Predictive Algorithm.....	10
Figure 11. Reshaped Output of the Predictive Algorithm.....	10
Figure 12. Showing Chosen Text File Before Upload (no error message).....	11
Figure 13. Showing Chosen Picture File (error message).....	11
Figure 14. Error Message Shown from Figures 10 and 11.....	11
Figure 15. Last Modified Files in the Server.....	11
Figure 16. Depiction of Resulting Flask Application File Structure.....	13
Figure 17. Depiction of Flask Application Connection Flow.....	14
Figure 18. Routing and Templating of Python Integer and Python Array Retrieved from a File in the Server to the HTML File to Display on User Interface.....	16
Figure 19. Successful Generation of Chart Object After a User Uploads CSV File.....	17
Figure 20. Image of Graph with Generated Prediction Trend on User Interface.....	17
Figure 21. Heroku Website Depicting Successful Deployment and Build.....	18
Figure 22. Build Log Detailing Successful Heroku Build.....	18
Figure 23. Title Block of Web Application (Includes Tab Buttons).....	20
Figure 24. Homepage of Web Application.....	21
Figure 25. File Upload Page of Web Application.....	21
Figure 26. Graphs & Data Page of Web Application ('Generate Prediction' Button and Graph Description).....	22
Figure 27. Graphs & Data Page of Web Application (Graph).....	22
Figure 28. History Page of Web Application.....	23
Figure 29. Validation of Title Block of Web Application.....	24
Figure 30. Validation of 'Choose File' Button of Web Application.....	25
Figure 31. Validation for Successful File Upload.....	26
Figure 32. Validation of Unsuccessful File Upload.....	26
Figure 33. Validation of 'Generate Prediction' Button (Before Prediction).....	27
Figure 34. Validation of 'Generate Prediction' Button (During Prediction).....	27
Figure 35. Validation of 'Generate Prediction' Button (After Prediction).....	28
Figure 36. Validation Message for Graph (Before File is Uploaded).....	28
Figure 37. Validation Message for History Page (Before File is Uploaded).....	29
Figure 38. Validation of Multiple Uploaded Files in History Page.....	29
Figure 39. Validation of History's Download.....	30

1. Introduction

Los Alamos National Laboratory (LANL) has requested a browser-based application to track COVID-19 cases through wastewater testing. Currently, LANL (along with most other major government agencies) uses reported COVID-19 cases as an indicator of which areas are at risk or currently hotspots for the virus. Unfortunately, this form of tracking COVID-19 has limitations due to inconsistency of reporting, asymptomatic cases, and the long time between testing and results. However, with new strategies like wastewater research, COVID-19 data can be gathered earlier and more accurately. To take advantage of this, the wastewater modeling application must allow for the input, analysis, and storage of data in a server, while also organizing the data for easy viewing. This will allow LANL management to make more informed decisions with regards to dealing with and preventing further spread of the virus.

With each subsystem listed below, the current web application is able to perform these tasks. As these subsystems are tested and validated for their intended functionality, they are able to create a cohesive product as a complete web application. There is therefore a direct path to further development of functionality and features as the application becomes more expansive.

2. Predictive Algorithm Subsystem Report

2.1. Subsystem Introduction

The predictive algorithm utilizes a Long Short Term Memory model (LSTM), which is a type of recurrent neural network. This is utilized to make predictions, as well as forecast data up to 7 days into the future. It works by utilizing a pre-processed CSV file that consists of two columns: the date, and the measurement of covid cases (or wastewater measurement). Utilizing the CSV file data, as well as various libraries (such as TensorFlow, Keras, pandas, and NumPy) for data manipulation, model building, and data visualization, it outputs a graph of past data as well as the forecasted predictions. The CSV data is then made into a data frame to separate the dates from the measurements. It is then split into training and testing data to train and test the recurrent neural network. The sequential data is turned into supervised data by using a time series generator, which is then fed into the LSTM model. The outputted data from the LSTM is then reshaped and fed into two functions: a predict function that takes in the number of days to forecast data for, as well as the output of the model to then make the forecast predictions. The predicted dates also function as the number of forecast days to align the forecast predictions with the correct date.

2.2. Subsystem Details

2.2.1. Recurrent Neural Network/Long Short Term Memory Model

The predictive algorithm subsystem is built upon a recurrent neural network system called a long short term memory model. This system employs feedback connections to process and recognizes time series data without treating every data point as independent. It overcomes a long term dependency by retaining a long term context. The data being fed into the neural network is a time series of Covid-19 case data (or Covid-19 wastewater data) that has been split into testing and training sets. The LSTM model is built by first initializing the sequential model, then initializing its units, which is the dimension of the hidden state. The hidden state and input data are then activated by the sigmoid function 'relu' or rectified linear activation function unit. A dense layer, the deeply connected layer of a neural network that receives the output, is also specified as one since the model needs to output one prediction for each testing data point. The model is then compiled using the Adam Optimizer, an adaptive gradient algorithm, and utilizes mean squared error for the loss function. Also initialized for compiling are the epochs, the number of training cycles the model goes through, and verbose. A graph of the training data points, testing data points, and predictions are output for validation and visualization purposes.

2.2.2. Forecasted Predictions

The forecasting of the model utilizes two functions: a "predict" function and a "forecast dates" function. The predict function is utilized to make the forecast predictions utilizing the parameters fed into the function, including the number of days the user wants to predict (which is specified as seven in the LSTM model output). Within the function, a list of forecast predictions is initialized and a for loop is used to append every prediction made for the seven days. The prediction list is then returned to be utilized for data visualization and

calculating accuracy. The forecast dates function utilizes the number of prediction days variable to create the array of the forecasted dates that align with the forecasted predictions. The forecasted dates and predictions are then output and then added as a trace to the output graph. The predictions are then analyzed to obtain the accuracy of the prediction values to the actual values.

2.3. Subsystem Validation

2.3.1. Testing Functionality of LSTM

This task is validated by testing that the LSTM model is able to work on two different csv datasets, covid cases and covid wastewater, as well as seamlessly runs without error. The graph is outputted as expected showing the data trace, testing data trace, and prediction trace as related to their respective days. These tests are shown in Figures 1 and 2 below. To also validate that the algorithm can work on various sized csv a Tesla Stock csv was utilized for validation measures as shown in Figure 3.

Test 1: Using pre-processed CSV data points from the backend of Boston COVID-19 wastewater levels, which consisted of 1073 days of incidence data, to test the functionality of the algorithm to show that it accurately outputs past data points as well as prediction points made by LSTM.

Test 2: Using unprocessed cumulative data of COVID-19 cases in Texas, consisting of 1011 data points to test functionality.

Test 3: Due to Boston COVID-19 wastewater and COVID-19 actual cases having around the same size of data points and needing to validate the algorithm works on various different sizes of CSVs, a Tesla stock CSV of 2393 points was utilized for validation.

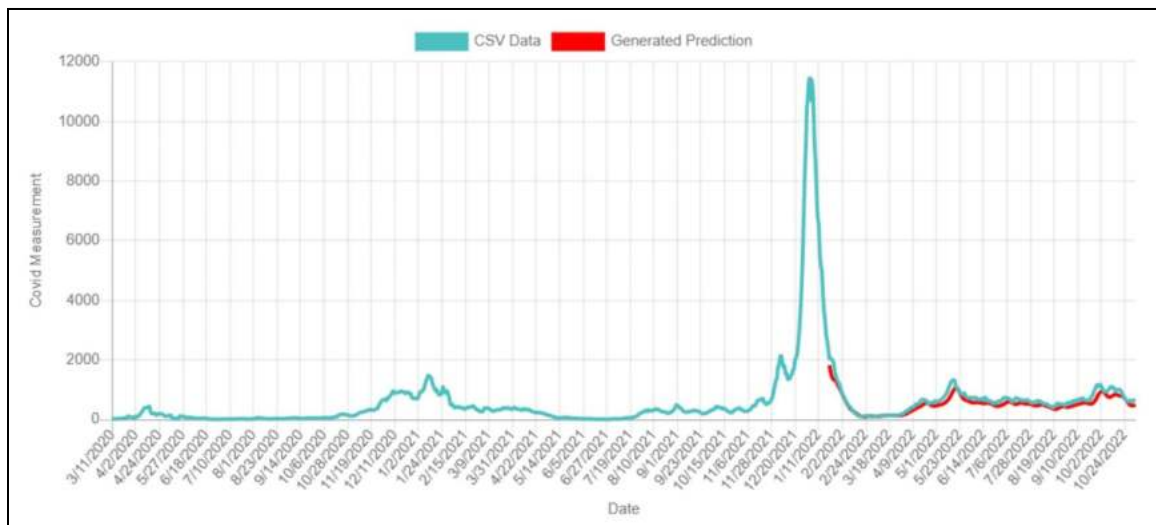


Figure 1: Tested LSTM Model Results on Wastewater Data

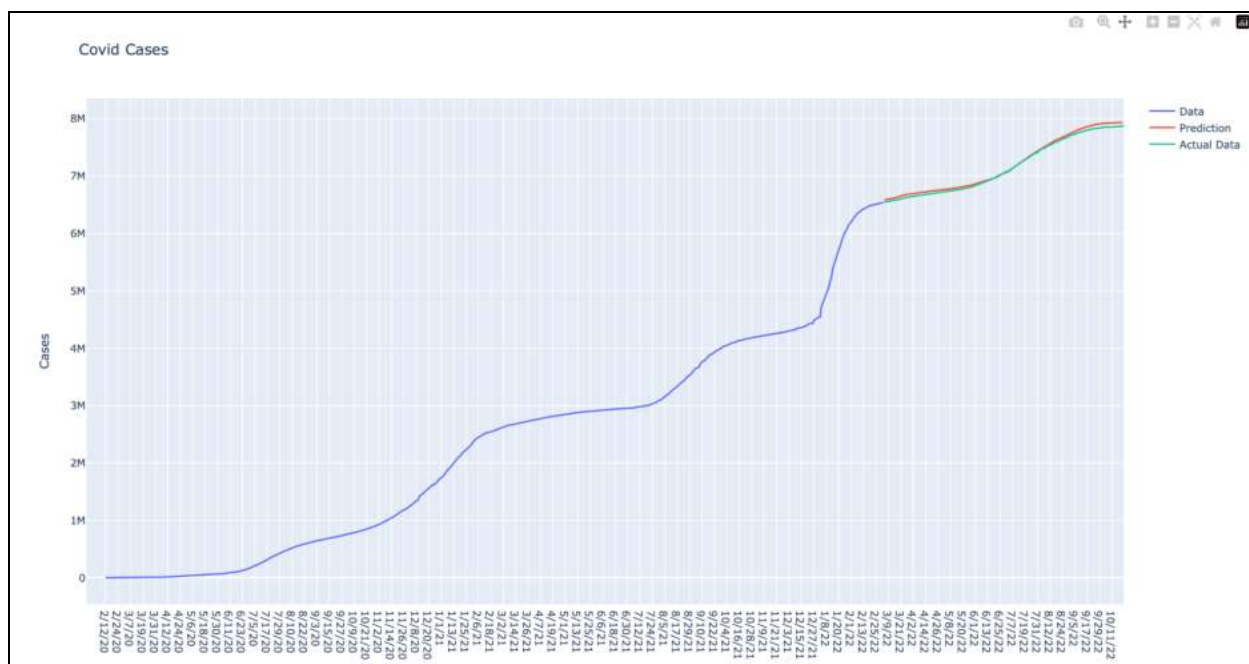


Figure 2: Tested LSTM Model on COVID-19 Cases in Texas



Figure 3: Tested LSTM Model on Tesla Stock

2.3.2. Forecasting Functionality and Accuracy Validation

Validation consisted of analyzing the accuracy of the predicted forecast. This was tested by utilizing hold-out validation of the algorithm. A week of data was withheld from the algorithm to be used as a test to see what the algorithm shows for the forecast predictions. The hold-out validation was tested in two areas - November 11th to November 17th and April 30th to May 6th, and then compared to the actual data points using the percent accuracy

formula. For the algorithm, a percent accuracy of 85% or above needed to be achieved , the accuracies in both hold out validations range from 86% to 91% showing that the forecasting prediction works as expected on the testing validation. The results of both of these holdout validations are shown in figure three and figure four respectively. The first data point of the array acts as a marker, as it is the respective amount of covid cases for the day of November 10th, which explains why it's the exact same value in the actual and predicted array. To also validate the functionality of the code still being able to output a graph that shows past data points as well as the forecasted prediction data points, an example of that is seen in figure 5.

```
actual [7902403, 7902627, 7902627, 7902627, 7910455, 7910788, 7914571, 7915047]
predicted [7902403. 7241526.5 7105170. 7048814.5 7028119. 7020858.5 7018564.
7018030. ]
91.63442106023732
89.90896318401464
89.19583956069292
88.84595133908226
88.75043168897966
88.67901999994693
88.6669403226538
```

Figure 4: Forecast Predictions, Actual Predictions and Percent Accuracy (11/11 to 11/17)

```
1/1 [=====] - 0s 15ms/step
[6712867, 6714245, 6716964, 6719204, 6721982, 6726454, 6729383, 6732109]
[6712867. 6004848. 5905599.5 5828472.5 5810871.5 5812803. 5820988.
5830899. ]
89.43444869825274
87.92066624147458
86.74349669990671
86.44580571623072
86.41704826941506
86.50106555088334
86.61325893564705
```

Figure 5: Forecast Predictions, Actual predictions and Percent Accuracy (4/30 to 5/6)

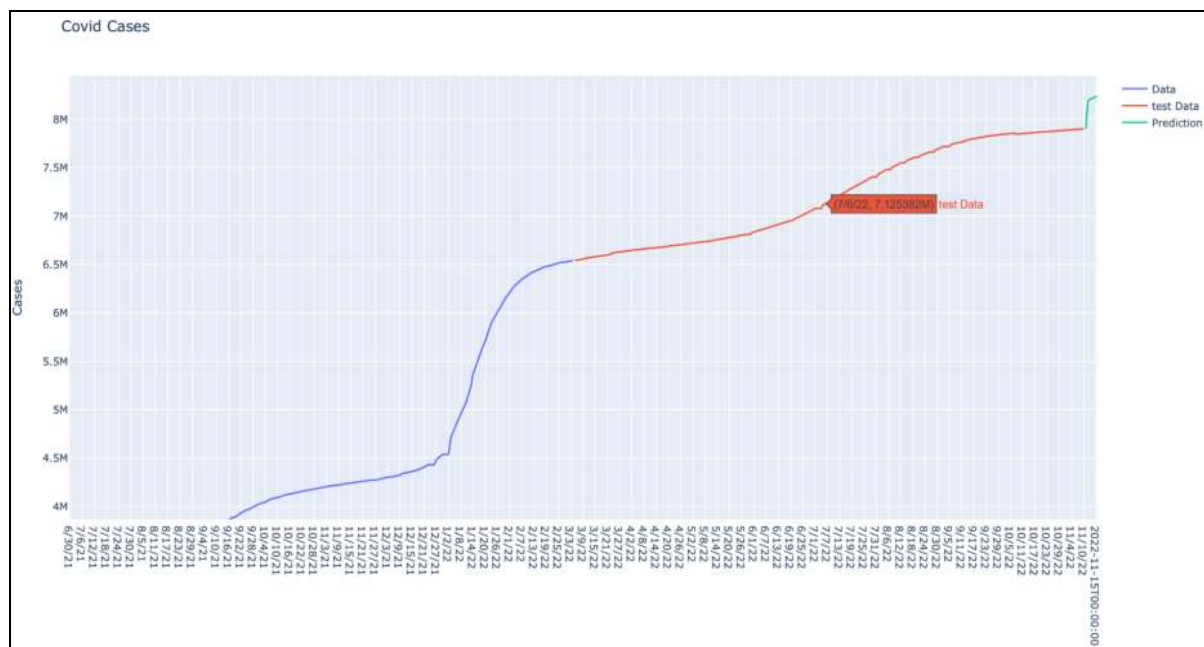


Figure 6: Result of Outputted Graph of Past Data and Forecasted Predictions

2.4. Subsystem Conclusion

The predictive algorithm operates as expected, as it can successfully take in a csv of Covid-19 case data in the state of Texas and make accurate forecasted predictions that range in accuracy from 86% to 91% when the predictive algorithm had a criteria of 85% and above accuracy. The algorithm also works as intended with actual Covid-19 wastewater data and being able to make accurate predictions.

A difficulty was not having as many data points as preferred to robustly train the model, but with the testing of different parameters and weights of the neural network, the goal of an 85% prediction accuracy was achieved. Another difficulty was ensuring that the model was not overfitting or underfitting to ensure a robust algorithm. To solve this, held-out validation was used in two areas of the data, such as in November and then in April, to ensure that the model was performing as expected on the testing data.

As more work and time is invested into this subsystem, a foreseeable plan would be adding other predictive models, such as naive prediction, as a means of validation and comparison and contrast of each model to ensure the best fit/type of model. Adding multiple LSTM networks can also be implemented to help enhance the original network. Also, implementing other validation plans to test the accuracy of the data such as showing the root mean squared error, and to test to ensure that the other models implemented are not overfitting or underfitting.

3. Data Processing/Server Subsystem Report

3.1. Subsystem Introduction

The data processing and server subsystem is responsible for creating the backend services for the website to store and retrieve data through a connection to an AWS Server, as well as cleaning uploaded data for later utilization. This process is initialized once a csv file (containing dates and covid levels) is uploaded from the UI, as the file is then saved in the AWS Server. This server utilizes an S3 bucket set up with access keys to enable a connection to be established from a Python file. The files are then able to be uploaded from Python and saved in the S3 bucket to be later accessed. Items in the csv file can then be cleaned by fixing any missing items, restructured, and stored in arrays to send to other subsystems for utilization.

Furthermore, the output of the predictive algorithm is an object (dataframe) that can't be graphed directly. This subsystem receives this, and converts the dataframe object into an array by manipulating it into a one dimensional matrix. Lastly, this subsystem contains an error-checking system (utilized on the frontend) which checks if the file uploaded is a csv or not. This prevents website crashes and gives the user an error message to upload the correct file type.

3.2. Subsystem Details

3.2.1. AWS S3 Bucket Set Up

To create a S3 bucket, an Amazon Web Services account had to be created. The bucket can then be initialized and set up, including its name and region settings, which had to be selected as "us-east-1" to be accessed through the backend program. Next ownership details were configured. With the server having multiple users, the Access Control List (ACL) was set to everyone. Once the bucket was created, properties were specified that created the ARN values, Access Key, and Secret Key required to connect the server. These keys disappear once the policies are finalized. The policies and the S3 bucket then work together to set up objects that allow permissions to the user. These policies were continuously altered to ensure the server would effectively save data uploaded to it, as well as allow access to the files.

3.2.2. Data Parsing and Cleaning from Uploaded File

This task begins with saving the user uploaded file onto the AWS Server (set up in 3.2.1). Once the file is saved, the contents can be accessed and cleaned for the graphing subsystem to use.

To save a file that is being uploaded from the front end of the website to the server, the access key of the server and the secret key of the server's bucket have to be referenced in the file. Boto and Boto.S3 Python packages were utilized to get the bucket within AWS server to be connected via Python. In specifying the bucket name and adding the AWS keys, the Python file became linked to the server. The uploaded file is then able to be saved to the bucket through this connection.

To parse data from the file, initially a function was created to loop through all the rows of the csv file and append to two new arrays. One array represented the dates while the other array represented the covid levels. The values were appended to the array, but the runtime was long, and wasn't feasible for the user - an alternative method was needed. This solution came in the form of the "pandas" Python library. A requirement that the user has in uploading csv files is that the file must consist of the first column containing dates, and the second column containing the Covid-19 level for that date. This allows the pandas library to quickly assign the columns into the corresponding arrays. There was further parsing needed which was done by omitting over any rows that had missing information (either date or measurement). With the csv file cleaned and then set to arrays, the data can then be utilized effectively by other subsystems.

3.2.3. Reshaping Predictive Algorithm

The output of the predictive algorithm is in the form of a dataframe object, similar to a series of nested arrays. This object contains all the data output from the predictive algorithm, such as the indexes, and all of the predicted values associated with the indexes. These arrays have to be separated and correctly formatted into a single array. This is done through separating every 1 dimensional array that has a single value (which is encapsulated by a large array) to combine all the values. This was accomplished by using the reshape function after several iterations of isolating the matrix into a single dimensional array. Once the reshaping is done, the data has to be parsed through again to be appended into the corresponding array variables needed for plotting.

3.2.4. File Upload Error Checking

The file upload error-checking was included within the "upload" routing portion of the frontend. When the file is uploaded using the upload button, the file name's "image" is requested and set to a variable. The image is transferred prior to the actual file being transferred to the AWS Server. If the file is not what is expected, then that file's image isn't stored in the server. Once the file name is received, the string is then split by "." with the first index being ignored (this is the assigned file name by the user), and the second index being the file type. If the second index is not equal to "csv", then the file doesn't get saved to the server, which then puts a message on the screen for the user to upload the correct file type. The error message restricts any graphs creation as well. If the correct file is uploaded, then the graph can be generated.

3.3. Subsystem Validation

3.3.1. Validation for AWS S3 Bucket Set Up

To validate this task, the tests below were performed to ensure uploading files to the AWS server works correctly, and that their data is able to be accessed locally through a connection.

Test 1: Using the frontend upload a file via the button that has been created on the file upload tab, and check the update on the AWS server.

Test 2: Outputting the contents of the file that was uploaded to the server and showing it on the terminal.

<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class
<input type="checkbox"/>	 testdata_2.csv	csv	November 29, 2022, 20:15:00 (UTC-06:00)	117.0 B	Standard

Figure 7: File Uploaded to the Server from the Upload Button

Date	Covid Level
8/2/22	0.48
8/3/22	0.3
8/4/22	0.56
8/5/22	0.53
8/6/22	0.48
8/7/22	0.5
8/8/22	0.62
8/9/22	

Figure 8: Data Set in the Excel Sheet 'testdata_2.csv'

```
(base) Ayaans-MacBook-Pro:wastewatermodeling ayaansunesara$ python download_demo_p1.py
printing dates
['8/2/22', '8/3/22', '8/4/22', '8/5/22', '8/6/22', '8/7/22', '8/8/22']
printing covid levels
[0.48, 0.3, 0.56, 0.53, 0.48, 0.5, 0.62]
(base) Ayaans-MacBook-Pro:wastewatermodeling ayaansunesara$
```

Figure 9: Output of the Downloaded File from the Server to Print the Data

3.3.2. Validation for Data Parsing and Cleaning from Uploaded File

Figures 6 and 7 represent the data parsing and cleaning of a csv file. In Figure 6, the last row contains an empty covid level cell. The parsing process is then run, with Figure 7 demonstrating the resulting dataset output with no empty values. Furthermore, the parsed data is then saved into corresponding arrays (dates and covid-levels) to be utilized by other subsystems. The arrays also remain in order, with the index of the dates corresponding to the covid level's index.

3.3.3. Validation for Reshaping Predictive Algorithm

An example of the output of the predictive algorithm (Covid-19 measurements) is shown (Figure 8) to demonstrate the reshaping of the algorithm's output into a single Python array. In Figure 8, individual values are in their own arrays which are all encapsulated in a large dataframe array. Figure 9 shows the resulting reshaped version where all the data is organized into a single array.

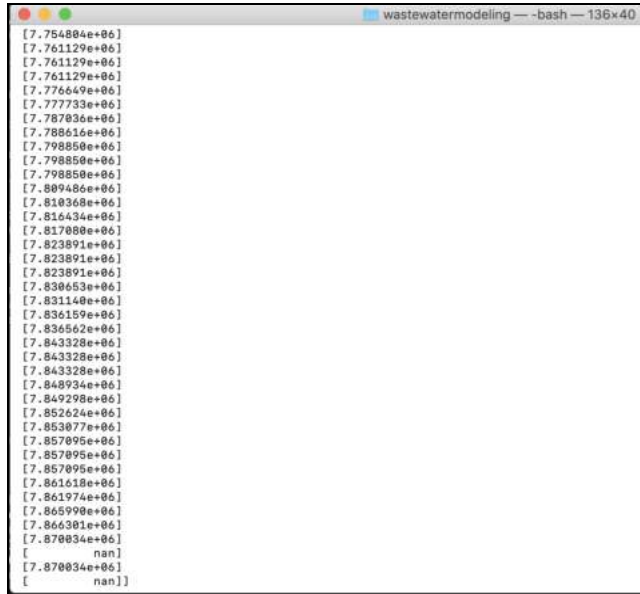


Figure 10: Example Output of Predictive Algorithm



Figure 11: Reshaped Output of the Predictive Algorithm

3.3.4. Validation for File Upload Error Checking

Multiple files are attempted to be uploaded from the UI below to demonstrate that no files other than a csv can be uploaded to the server. The files within the server are then displayed to demonstrate that the non-csv files were not saved. Figure 10-11 shows non-csv files being “uploaded,” along with the resulting error message in Figure 12. Figure 13 then shows the uploaded files in the AWS Server, filtered by last modified first. None of the files uploaded in Figures 10-11 show up in the server, validating that the error checking is successful.

File Upload

Click on the "Choose File" Button to Select a File and Click on the "Upload" Button to Send it to the Server:

requirements.txt

Figure 12: Showing Chosen Text File Before Upload (no error message)

File Upload

Click on the "Choose File" Button to Select a File and Click on the "Upload" Button to Send it to the Server:

randomFore...Outline.png
 Enter a file with correct format

Figure 13: Showing Chosen Picture File (error message)

File Upload

Click on the "Choose File" Button to Select a File and Click on the "Upload" Button to Send it to the Server:

No file chosen
 Enter a file with correct format

Figure 14: Error Message Shown from Figures 10 and 11

	Copy 53 URI	Copy URL	Download	Open	Delete	Actions ▾	Create folder
Upload							
<input type="text" value="Find objects by prefix"/>							
<	1	>					
<input type="checkbox"/>	Name	Type	Last modified	Size	Storage class		
<input type="checkbox"/>	texas_clean.csv	csv	November 30, 2022, 11:26:19 (UTC-06:00)	17.5 KB	Standard		
<input type="checkbox"/>	boston_clean.csv	csv	November 30, 2022, 11:20:59 (UTC-06:00)	14.3 KB	Standard		
<input type="checkbox"/>	testdata_2.csv	csv	November 29, 2022, 20:15:00 (UTC-06:00)	117.0 B	Standard		
<input type="checkbox"/>	testingdemo.csv	csv	November 29, 2022, 20:08:31 (UTC-06:00)	120.0 B	Standard		
<input type="checkbox"/>	Texas.csv	csv	November 29, 2022, 15:10:53	67.6 KB	Standard		

Figure 15: Last Modified Files in the Server

3.4. Subsystem Conclusion

Overall the server for the website and a function to clean the data files has been completely set up. That allows parsing of the file to be utilized effectively by other backend subsystems.

One difficulty faced included setting up the AWS server correctly. There were several instances where multiple buckets had to be created since some of the setups were not

correctly configured. Since the server policy and the bucket configurations were linked, there had to be specific settings that needed to be enabled for them to work together, and have multiple users be able to access the bucket. Furthermore, there were challenges in parsing the data where it was initially done through looping through the entire files' data, but the runtime of the program was longer than expected. With further research, the runtime was decreased significantly through the utilization of the pandas library. Lastly, reshaping the output of the predictive algorithm was also done using trial and error to figure out the correct way to reduce the array to a singular one with column data in an entire array. This was done using the reshape function instead of looping through the entire array and appending the values to a new array.

The tasks of the subsystem are completed, but in the future one thing that will need to be done is a way to delete files from the server after a certain period of time so that the server doesn't overload, as well as expand any data processing functionality for any other data sources that may be utilized.

4. Frontend/Backend Integration Subsystem Report

4.1. Subsystem Introduction

This subsystem is responsible for establishing the integration between the backend Python processes (including accessing the AWS S3 Server, data processing procedures, and the predictive algorithm), and the frontend of the user interface, thereby enabling the user to interact with and view the web application's functionalities and results. This is done through establishing a Flask microframework structure throughout the web application, enabling user requests to be routed to the backend for processing. The generated results of this request are then routed to the frontend HTML through Flask templating and Jinja2 tools. This includes the generation of a line-graph displaying the user's parsed input data, as well as the results of the prediction algorithm for comparison, which is accomplished through a dynamic Chart.js generation function. This Flask web app was then tested to be deployed upon the Heroku platform through version control of utilized packages.

4.2. Subsystem Details

4.2.1. Flask Microframework Structure, Routing and Templating

Structure:

The Flask microframework structure was implemented as the first step in defining the file-system requirements needed to develop each component of the web application.

As Flask is a Python based framework, the “main.py” file was first created. This contains the routing function definitions associated with each user input from GUI to render the appropriate template, and is run to generate the application locally. The majority of the code of this subsystem is located within this file. To then enable HTML pages to connect to the routing functions, the “templates” folder was created to store all HTML files. To utilize static assets (images, CSS files), an “images” and “styles” folder was created within a “static” folder.

To maintain version control of utilized Python packages, a virtual environment was created under the “venv” folder, containing all the installations of modules for the current version of Python. This enables the creation of the requirements.txt folder to track these module versions needed for deployment. To keep these installations from being uploaded to the shared GitHub, a “.gitignore” file is defined to not track these changes in commits.

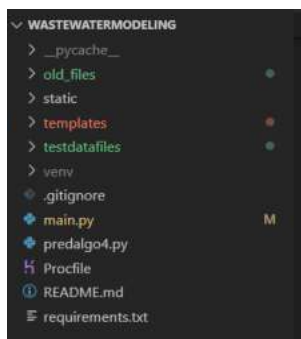


Figure 16: Depiction of Resulting Flask Application File Structure

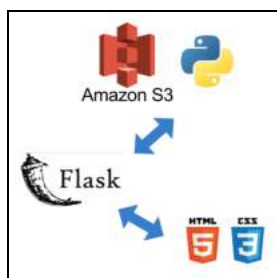


Figure 17: Depiction of Flask Application Connection Flow

Routing/Templating:

The routing functions within the “main.py” are each called upon the event of the user clicking a different element of the website, such as a tab or button. These functions have the ability to pass in variables through methods within its parameters. On this given event, the function does any processing (such as calling other functions if needed), and returns the corresponding HTML template with any result variables needing to be routed to the UI. A summary of each created routing method and its functionality is within the table below:

Route	Function	Intended Result of Route
1	@app.route('/')	The default route that is called upon the website being generated, which renders the home page, “index.html,” as its template.
2	@app.route('/index.html')	Also returns the home page on the event the user clicks the home tab.
3	@app.route('/file_upload.html')	On the event the user clicks the file upload tab, this renders the “file_upload.html” as its template.
4	@app.route('/upload', methods=['POST'])	On the event the user uploads a file through the associated button, this function checks the file for the correct type (.csv), and uploads the file to the server. The “generate” function is then called to enact the data processing to populate the variables needed to later render the graph of the uploaded data. Updates the list of uploaded files. Then re-renders the file upload page with a message variable indicating if upload was successful.
5	@app.route('/graphs_data.html')	On the event the user clicks the “Graphs” tab, this renders the “graphs_data.html” with the variables needed to generate the line-graph.
6	@app.route('/update_graph', methods=['POST'])	On the event the user wants to enact a prediction for their uploaded data, the user clicks a “Generate Prediction” button that calls this route. The “update” function is then called, which generates the prediction by calling the predictive algorithm function “generate results,” and updates the line-graph variables with prediction result data. Then re-renders the “graphs_data.html” page with the updated graph variables needed to update the line-graph with the new line.

7	@app.route('/history.html')	On the event the user clicks the “History” tab, this function renders the “history.html” page with the variables needed to generate the list of previously uploaded files.
8	@app.route('/download', methods=['POST'])	On the event the user clicks the download button for a previously uploaded file on the history tab, this function calls the file download function from the server to download the file to the user’s machine.

Table 1: Flask Routes and the Associated Purposes

4.2.2. Dynamic Chart Generation Algorithm

This function is responsible for generating the current version of the line-graph based on the current state of the line-graph variables. It is located within the graphs_data.html page, which utilizes Chart.js to create the line-graph object through Javascript. The passed in chart-data array from the backend is then utilized within the HTML through Jinja2 expressions.

Cases:

In the event the user has not uploaded a file yet, the chart function will not be run, and an instruction message will be displayed on the page instructing the user to first successfully upload a csv file. This functionality is implemented based upon the emptiness of the chart-data variable.

However if the user has successfully uploaded a data file, the chart generation occurs. This is done through creating a new chart object of type “line”, specifying the x-axis (i.e. “labels”) to be the date values from the csv, and leaving the y-axis (i.e. “datasets”) initially empty. The names for each axis are also initially set.

Population:

To then populate the chart with its lines, the “datasets” of the graph are updated. If the user has not yet generated a prediction, only the users Covid-19 wastewater measurements from the csv file (passed in within chart-data array) will be appended through the “datasets.push()” function, which also sets the qualities of the given line (label, color, size, etc.).

However if the user has also generated a prediction, the above process runs a second iteration. As the prediction line has a smaller length, the offset is determined. To ensure a 1:1 mapping of y-axis values to the x-axis dates, The array to be pushed to the graph is then populated with null values up to the index determined by the offset. The actual prediction data is then concatenated to the end of this array, enabling the line to then be pushed to the datasets as before. The chart is then updated to register changes.

4.2.3. Publicly Hosted through Heroku

A Heroku account was created to deploy the web application for public hosting. Unicorn, a dyno utilized for Python applications within Heroku, was utilized through the addition of a Procfile, specifying “main.py” as the file to run the application. The deployment of the

website was then done through connecting the GitHub repository's main branch, allowing for each deployment to "main" to automatically deploy to Heroku.

The requirements.txt file is utilized by Heroku to build the app with all needed modules installed. To ensure that all modules were able to run, the Python version for Heroku had to be set to 3.6.15, as the Tensorflow module (used in the predictive algorithm) is not supported by newer Python versions. To downgrade Python, the Heroku "stack" had to be changed from 22 to 20 (through the Heroku CLI to the web application), and a runtime.txt file was added to specify the version.

4.3. Subsystem Validation

4.3.1. Flask Microframework Structure, Routing and Templating

This task is validated through a series of tests to ensure that the basics of the routing and templating strategies are functional.

Test 1: Ability to connect web application backend data to frontend. This is validated through creating a Python variable in backend, routing it to the frontend HTML, and utilizing it with Jinja2 to display it to the UI.

Test 2: Ability to transfer large data amounts from backend to frontend. Similar to test 1, but transferring arrays of data to UI rather than a single String.

Test 3: Ability to push data retrieved from server to the frontend. This retrieves an array from a file in the AWS Server, gets an array of its data, and transfers it to the UI.

Tests 1-3 can be demonstrated as valid in the test image below, as a Python string, and an array from a file within the server are able to be displayed to the UI:



Figure 18: Routing and Templating of Python Integer and Python Array Retrieved from a File in the Server to the HTML File to Display on User Interface

4.3.2. Dynamic Chart Generation Algorithm

Test 4: Demonstrate that a graph is generated upon the successful upload of a csv file. If the graph appears as intended (correct labels, lines, settings), not only is the Chart.js generation function working properly, but this also validates that the routing processes associated with the graph (Routes 3-6 from route table) are functioning properly. This is demonstrated in the images below:

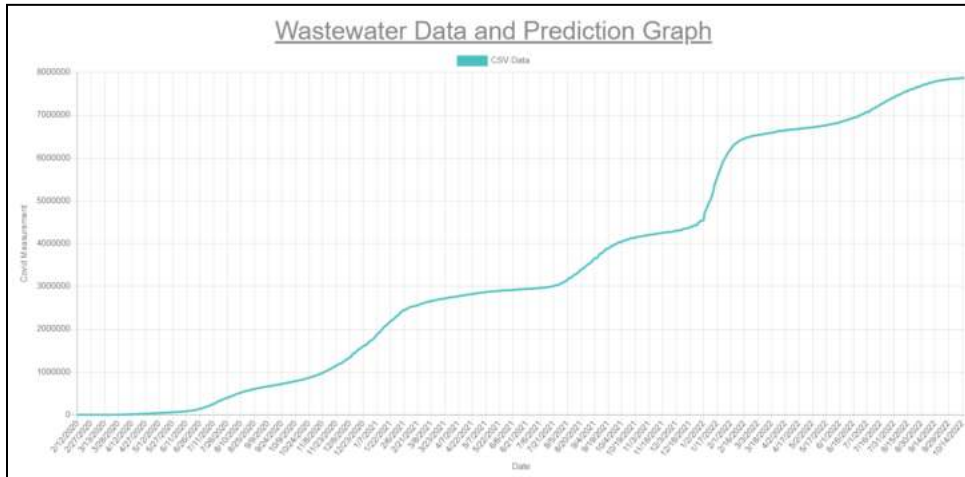


Figure 19: Successful Generation of Chart Object After a User Uploads CSV File

Test 5: Demonstrate that graph and associated objects can dynamically update upon the user generating a prediction. After clicking the button, the routing processes occur to call the predictive algorithm, generate results, and display the updated variable values onto the graph. The result of this update is shown working in the image below:

(Note: this dynamic updating of template variables is also implemented within the history page, pictured in the User Interface Validation section, 5.2.2.4, to enable users to download previously uploaded files).

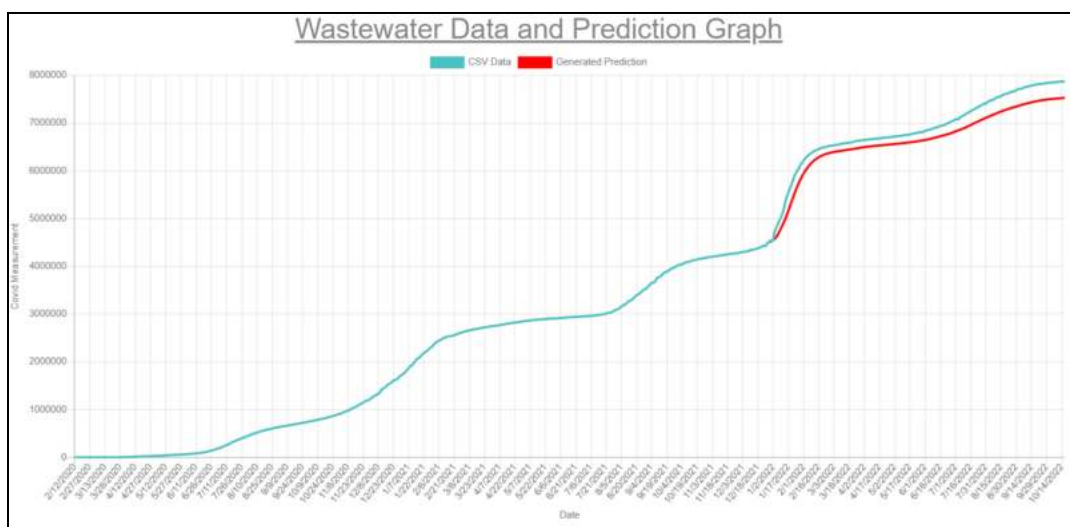


Figure 20: Image of Graph with Generated Prediction Trend on User Interface

4.3.3. Publicly Hosted through Heroku

This task is validated through determining if the web application is able to successfully deploy and fully build on Heroku's stack with all packages. This ensures that all needed modules are able to be supported, and are within Heroku's build limitations. The most recent version of the web app is shown to have been successfully built to Heroku in the images below:



Figure 21: Heroku Website Depicting Successful Deployment and Build



Figure 22: Build Log Detailing Successful Heroku Build

4.4. Subsystem Conclusion

Progress:

The subsystem was successful in supporting all intended functionality requirements for the current web application, with all its tasks validated. These include routing/templating all potential UI requests to and from the backend processes, generating a dynamic state-based graph, and hosting the web-page publicly through version control. This has enabled the complete project's functionality to be utilized by the user through only the UI.

Difficulties/Solutions:

There were unique challenges primarily in the generation of the graph. Through Chart.js, dynamically adding lines to an existing graph proved difficult, as there is no straightforward method to implement datasets at a certain x-offset (all lines are assumed to span the same x-axis). This required a dynamic solution such that the object can be regenerated for the given data values at any current state. Through first initializing the x-axis as the dates of the longest dataset (with the y-axis empty), the y-axis values for each line can then be "pushed" in a loop iteratively based upon the number of lines at a later instance. However, Chart.js still requires a 1:1 mapping for each line, even for shorter datasets. To mitigate this, the offsets of the generated lines are tracked, and utilized to set all spaces around the actual

line data to “null” within the Javascript array. This limited the Chart.js functionality to only display the intended values.

For the routing/templating, as the UI subsystem and the backend subsystems develop more complexity, the need for increased communication between them expands, resulting in additional routes, templates, and variables. This resulted in the challenge of keeping all variable states updated such that they are utilized without error. This largely regards the dynamic variables, such as the array for the graph data or the history page data. To mitigate this, extra functions and error handling was implemented to prevent unwanted results. This includes adding a function to clear the data variables at a given instance within routes (i.e. if a user uploads a new file, the old graph array values clear), as well as limiting the “Generate Prediction” button to one click, as it disappears when the graph regenerates.

For the Heroku hosting, some difficulties included the version and sizes of certain packages. Utilizing virtual environments helped to establish version control, and to minimize sizes, the larger modules were targeted - primarily “torch,” which is used in the prediction algorithm. An older, more stable version was utilized, reducing its size from 800+ MB to ~110 MB. This will be further optimized for better hosting performance, as the modules are currently close to the upper limit (500 MB) for Heroku.

Remarks:

This subsystem is designed to ensure that scaling with additional functionality throughout the web application is easily implemented. For each new button/request created within the UI, it can be handled through an additional route to a new backend process. Any new results needing to be displayed to the UI can be appended as a new variable within its respective template. Any new models/lines/statistics needing to be displayed to the line-graph can be appended through the dynamic chart generation function. In future development, organization and error-handling will be essential components of maintaining this subsystem.

5. User Interface Subsystem Report

5.1. Subsystem Introduction

This subsystem is responsible for creating a frontend user interface design that allows the user to be able to effectively view and interact with the contents of the web application. This is done by using HTML and CSS to design and implement features that allow the user to navigate the website's features. The aforementioned features include functional tabs, file upload button, graph display (with prediction) and history feature. The functionality of these features are aided by the Integration Subsystem (in section 4.1.) to establish connections to and from the HTML and the backend.

5.2. Subsystem Details

5.2.1. Title Block

The title block of the web application is the general structure that includes the title of the website, team number, and names and the tabs. The title block will also remain consistent throughout the entirety of the web application, meaning that regardless of what tab the user is currently in, the title block will remain the same. This allows the user to navigate throughout the web application without having to go back to the homepage every time the user wants to switch to a new tab. The title block can be seen though Figure 23 below.

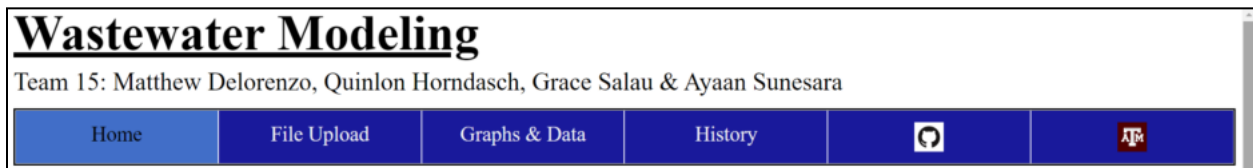


Figure 23: Title Block of Web Application (Includes Tab Buttons)

5.2.2. Functional Tabs

The functional tabs allow the web application to stay organized and provide an easy to navigate environment for the user to browse through. The tabs included are: Home, File Upload, Graphs & Data, History, GitHub, and TAMU. The tab can be seen though Figure 23 above.

5.2.2.1. Home

The Home tab directs the user back to the homepage of the web application. The homepage includes the basic title block (discussed in section 5.2.1.), the 'About Wastewater Modeling' and 'About the Web Application' description texts, and some helpful links to better describe wastewater modeling and the general goal of the web application. The Home page can be seen though Figure 24 below.

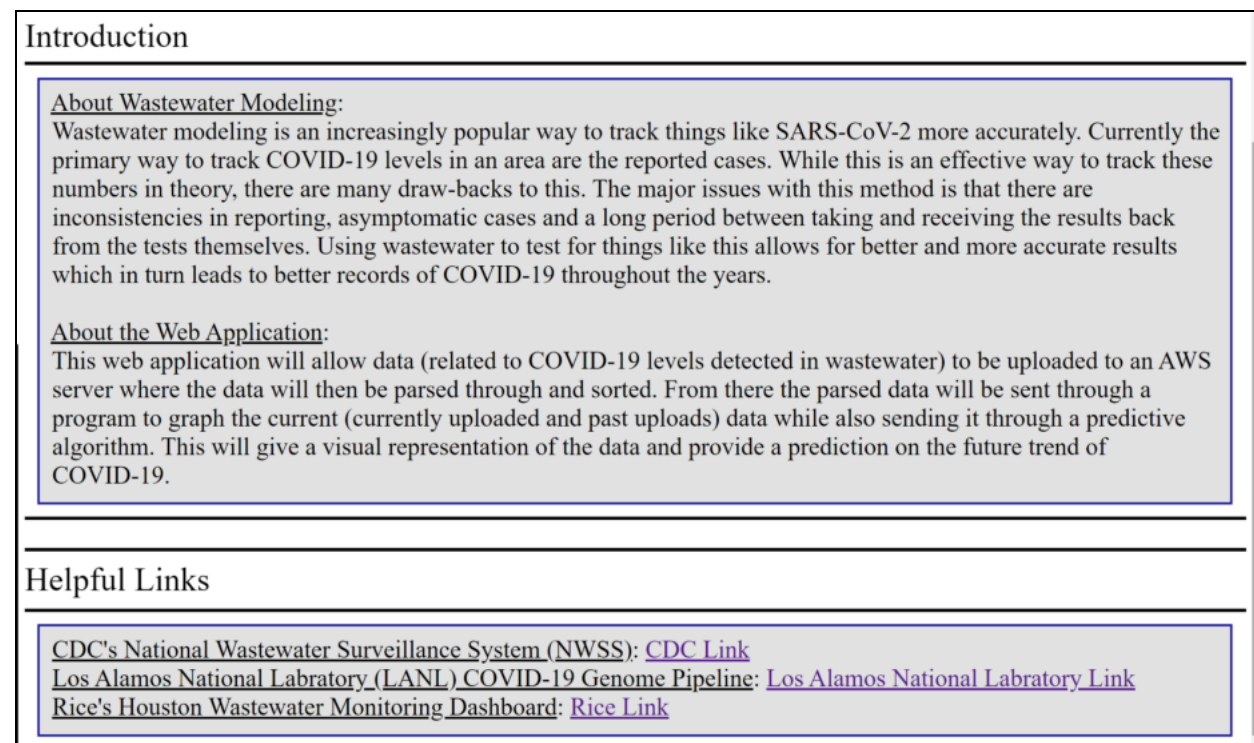


Figure 24: Homepage of Web Application

5.2.2.2. File Upload

The File Upload tab directs the user to the File Upload page of the web application. This page includes the basic title block (discussed in section 5.2.1.), the actual file upload button (discussed in section 5.2.3.), and a 'General Guidelines' description with a representative image. This page allows the user to upload wastewater data (in CSV format) through an HTML button to the AWS Server, which simultaneously creates a graph of the uploaded data (through Flask subsystem - see section 4.1). The File Upload page can be seen though Figure 25 below.

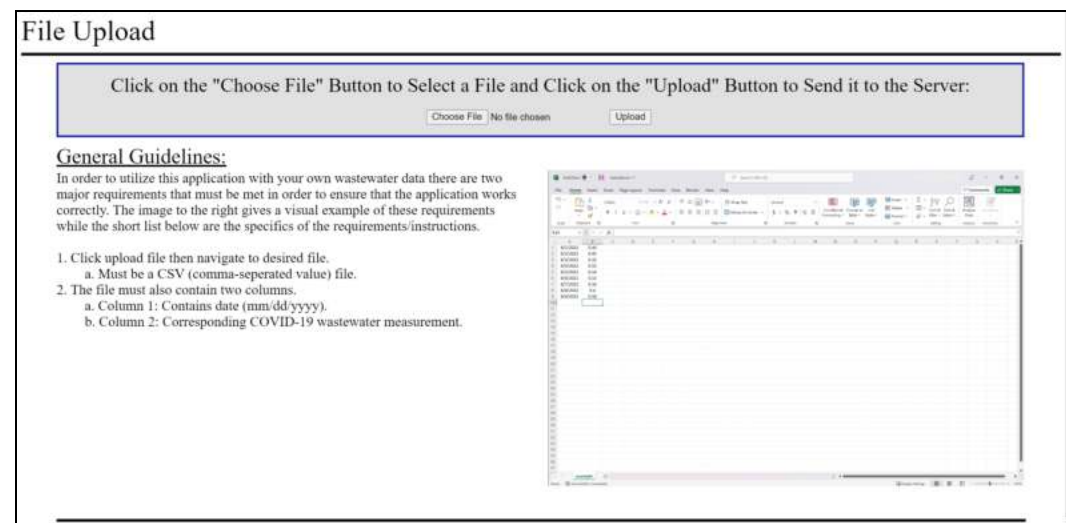


Figure 25: File Upload Page of Web Application

5.2.2.3. Graphs & Data

The Graphs & Data tab directs the user to the Graphs & Data page of the web application. This page includes the basic title block (discussed in section 5.2.1.), the 'Generate Prediction' button, an 'About the Graph' description, and the graph itself (discussed in section 5.2.4.). This page will provide the user with the graph associated with the uploaded data post calculations and predictive algorithm. The Graphs & Data page can be seen though Figures 26 and 27 below.

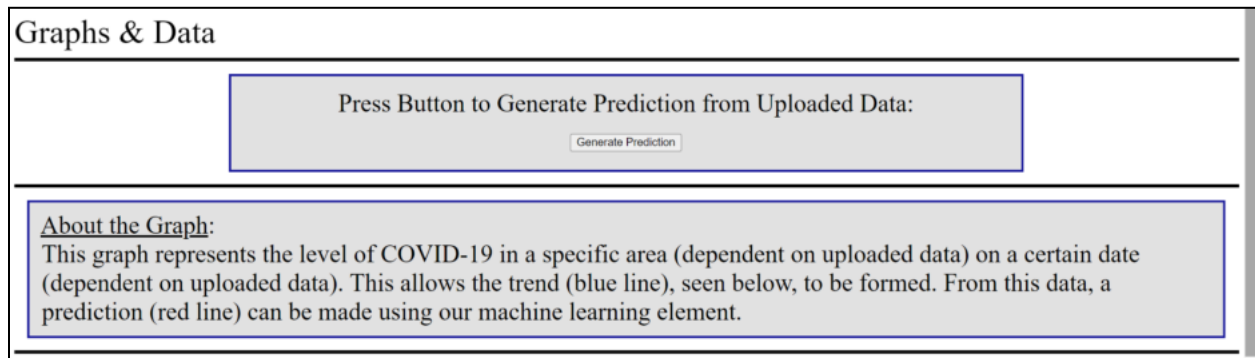


Figure 26: Graphs & Data Page of Web Application ('Generate Prediction' Button and Graph Description)

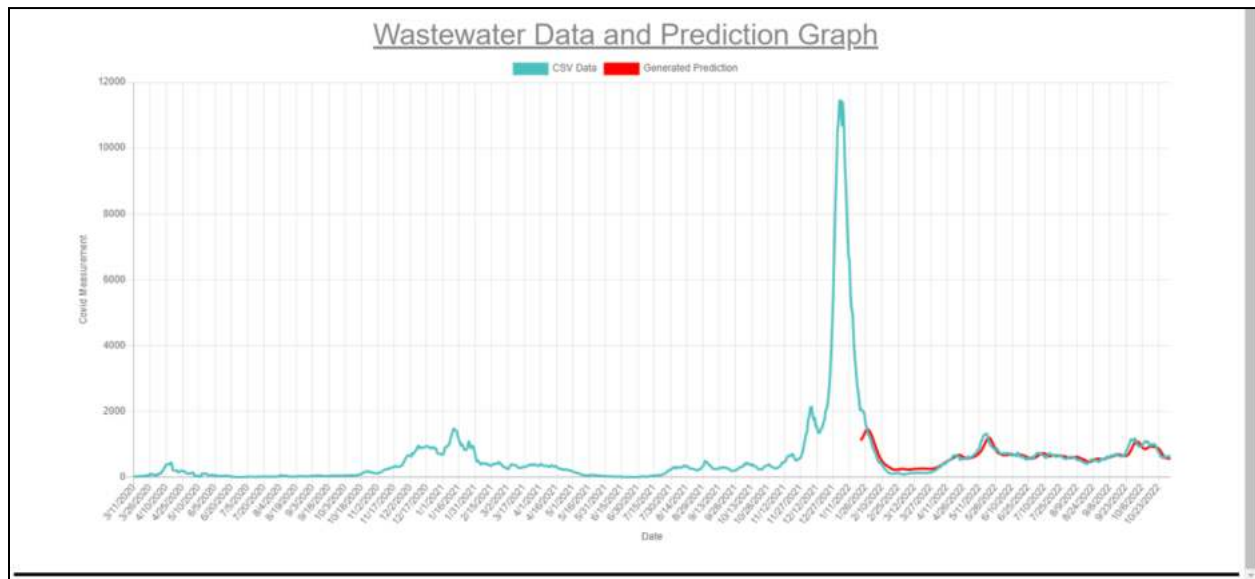


Figure 27: Graphs & Data Page of Web Application (Graph)

5.2.2.4. History

The History tab directs the user to the History page of the web application. This page includes the basic title block (discussed in section 5.2.1.), 'About the History Tab' description, and a 'Previously Uploaded Files' section. This page will provide the user with the option to view and download the past uploads. The History page can be seen though Figure 28 below.

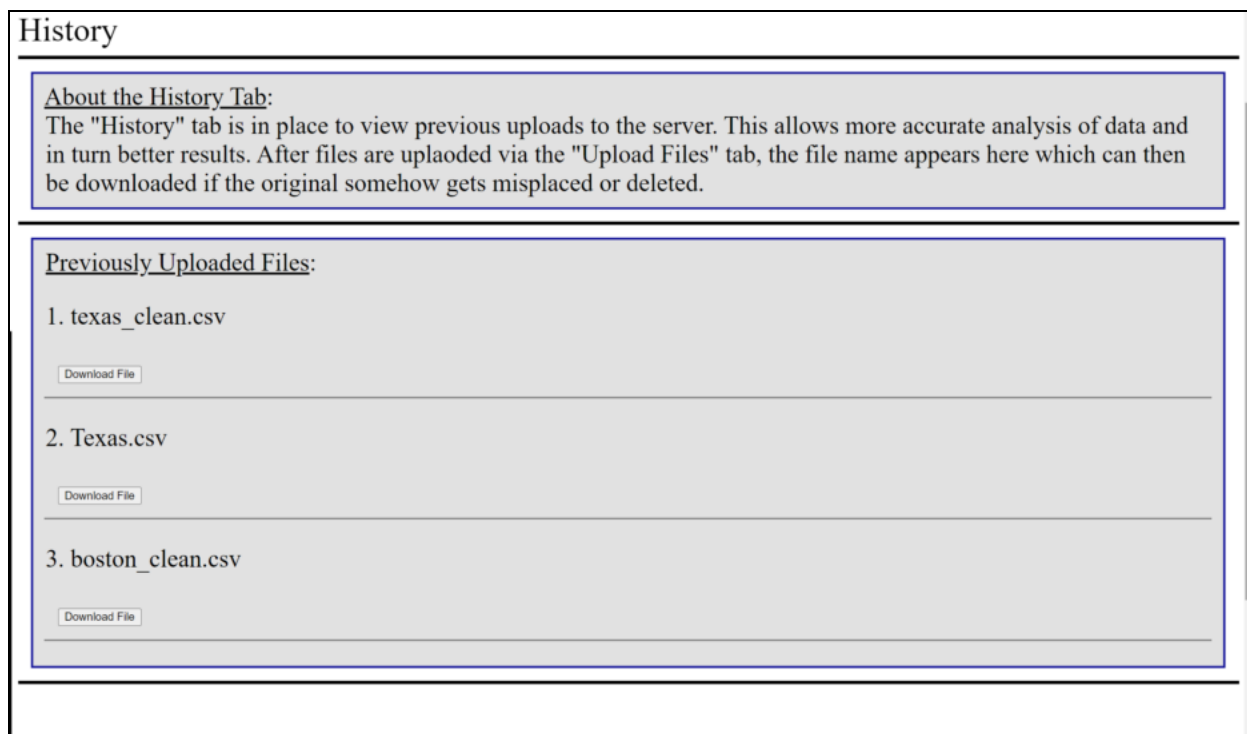


Figure 28: History Page of Web Application

5.2.2.5. GitHub

The Github tab directs the user to the team's Github page. This allows the user to view the code, reports and data associated with the web application. The tab button can be seen through Figure 23 above.

5.2.2.6. TAMU

The TAMU tab directs the user to the Texas A&M web page. This allows the user to get more insight into the University that the project is being completed in. The tab button can be seen through Figure 23 above.

5.2.3. File Upload Button

The File Upload Button of the web application is what links the user to the AWS server. This allows the user to click the 'Choose File' button, choose a file from the file explorer application and finally click the 'Upload' button which sends said file to the AWS server. From there, the file's data can be used to create a plot and prediction which are then both sent to the Graphs & Data tab. The File Upload button can be seen on the File Upload page though Figure 25 above.

5.2.4. Graph

The Graph of the web application is the outcome of the user uploading a file to the web application. This allows the user to view the raw data (directly from the uploaded file) and also view the predicted trend for 1 week in advance (see Integration subsystem in section 4.1. for details). The graph can be seen on the Graphs & Data page though Figure 27 above.

5.2.5. History Downloads

The History Downloads of the web application is where the user can view and download previously uploaded files. This works by having the file name and a download button appear in the History page that the user can view after a file is uploaded onto the AWS server. The History Downloads can be seen on the History page though Figure 28 above.

5.3. Subsystem Validation

5.3.1. Validation for Title Block

The validation for the title block is done by clicking onto each of the tabs and ensuring that the title block remains consistent in each of the tabs. Another important validation for the title block is to ensure that the color of the tabs changes when hovering over a tab (lightest blue), when in a specific tab (medium blue) and when not interacting with a tab (darkest blue). Each of these validation points can be seen though Figure 29 below.

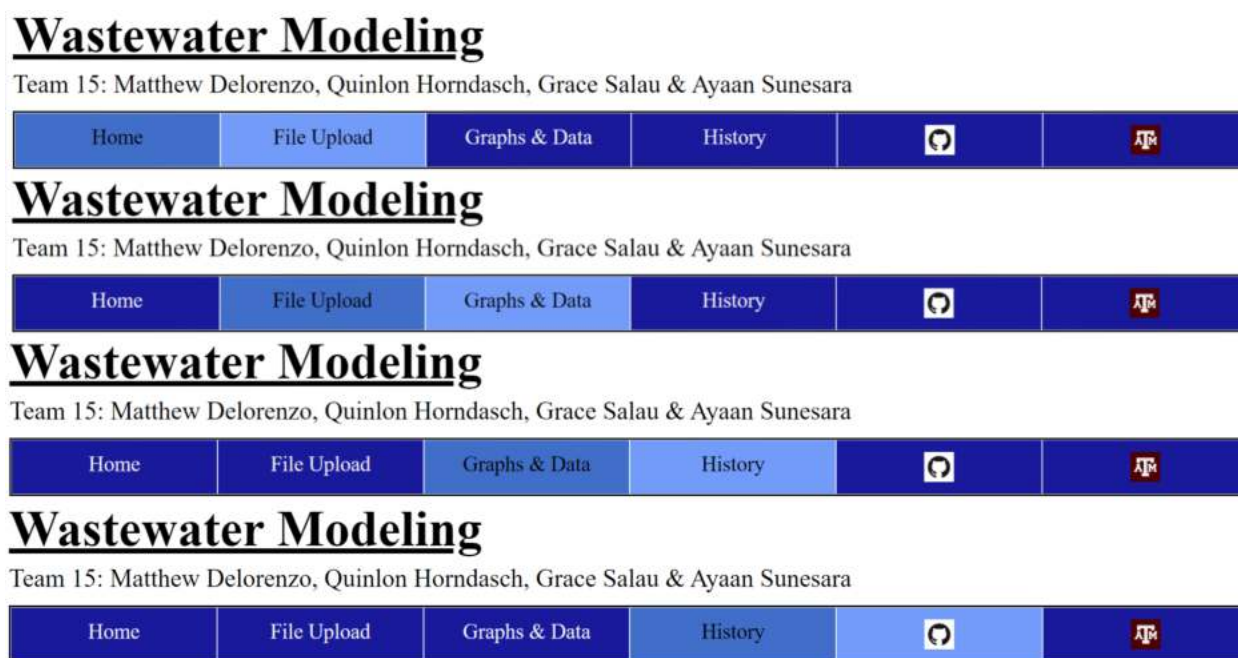


Figure 29: Validation of Title Block of Web Application

5.3.2. Validation for Functional Tabs

The validation for the tabs is very similar in execution as the validation for the title block (section 5.3.1.). This is done by individually going from tab to tab to ensure that the proper page is pulled up each time. Here the primary goal is making sure that the tabs are functioning and the structure of each page is correct whereas specific features are validated below (sections 5.3.3. through 5.3.5.).

The Home page can be viewed in Figure 24 above. The File Upload page can be seen in Figure 25 above. The Graphs & Data page can be viewed in Figures 26 and 27 above. Finally the History page can be seen in Figure 28 above.

5.3.3. Validation for File Upload Button

The primary validations for the upload button relevant to the project are that files can be uploaded from the user's PC, the files are uploaded correctly, an error message is displayed for incorrect file types (not .csv), and that an error message is displayed for incorrect file formatting (anything not specified in the 'General Guidelines' description).

Being able to upload files from the user's PC can be validated by clicking on the 'Choose File' button in the File Upload page then seeing if it shows the user's file explorer application (user's files). The next step is to see if once the file is selected, it is viewable before the 'Upload' button is pressed. This can be viewed in Figure 30 below.

Ensuring the Files are uploaded correctly can be seen in section 3.3.1., however, having a successful upload must be prompted to the user. This ensures that the user knows the file was uploaded correctly and does not need to worry about reuploading or backtracking to check. This success message can be seen in Figure 31 below.

Displaying an error message when the incorrect file type is uploaded ensures that the web application can continue operation for the user. For example, uploading a .txt can cause the entire web application to shutdown due to the incorrect file type. To counteract this, displaying an error message ensures that the user can upload the correct type of file while also interrupting the calculation process from continuing. This error message can be seen in Figure 32 below.

Displaying an error message when the incorrect file format is uploaded also ensures that the web application can correctly work for the user. Displaying an error message ensures that the user can upload the correct type of file while also interrupting the calculation process much like in above (error for an upload of an incorrect file type). This error message can also be seen in Figure 32 below.

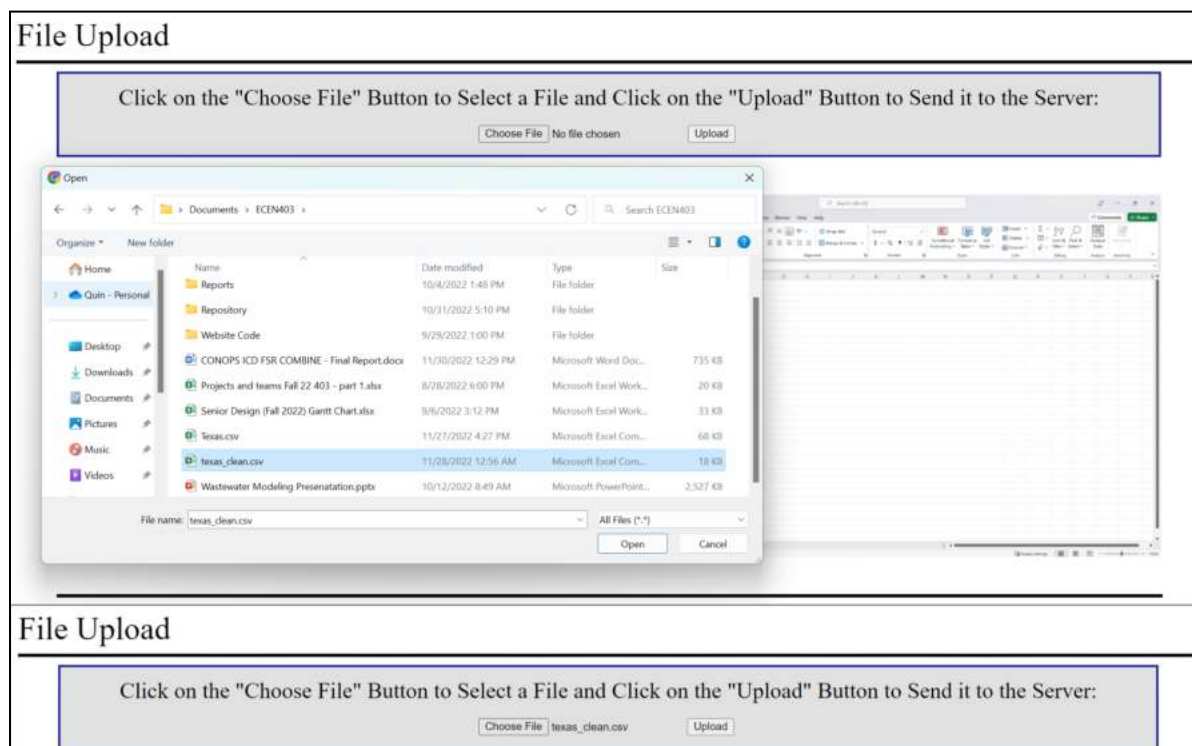


Figure 30: Validation of 'Choose File' Button of Web Application (top is selection of file from user's file explorer and bottom is file successfully selected in application)



Figure 31: Validation for Successful File Upload

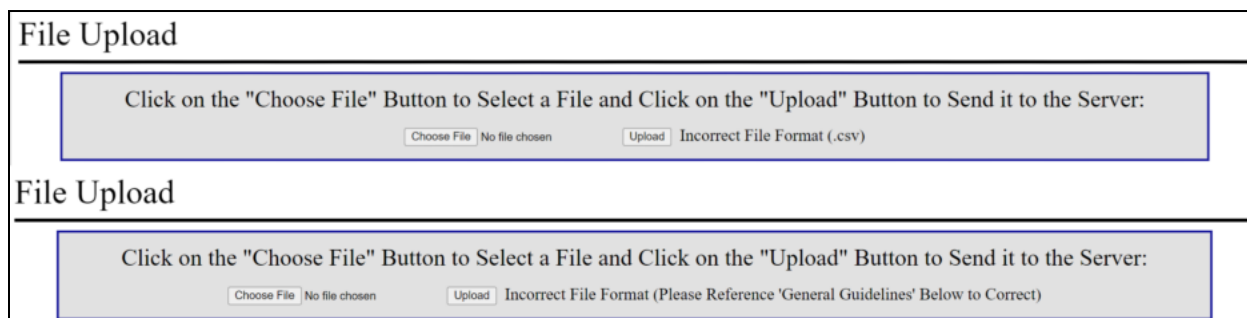


Figure 32. Validation of Unsuccessful File Upload (top is for incorrect file type [not .csv] and bottom is incorrect file formatting [does not follow 'General Guidelines'])

5.3.4. Validation for Graph

The primary validations of the graph relevant to the project are that the 'Generate Prediction' button works correctly, and that a warning message is displayed when needed, which prompts the user to upload a file to view a graph.

To validate that the 'Generate Prediction' button works, the graph must be viewed in 3 phases. These phases include: before the button is clicked (raw data only), after pressing the button but before prediction is finished (in the middle of loading the prediction) and after the prediction is finished running (both raw data and predicted trends). These 3 phases can be seen in Figures 33, 34 and 35 below.

To ensure that a blank graph or that an error (not planned) is not shown to the user, a message telling the user to upload a file before viewing anything must be put in place. This makes sure that the user only sees results when the proper steps are followed. This message can be seen in Figure 36 below.

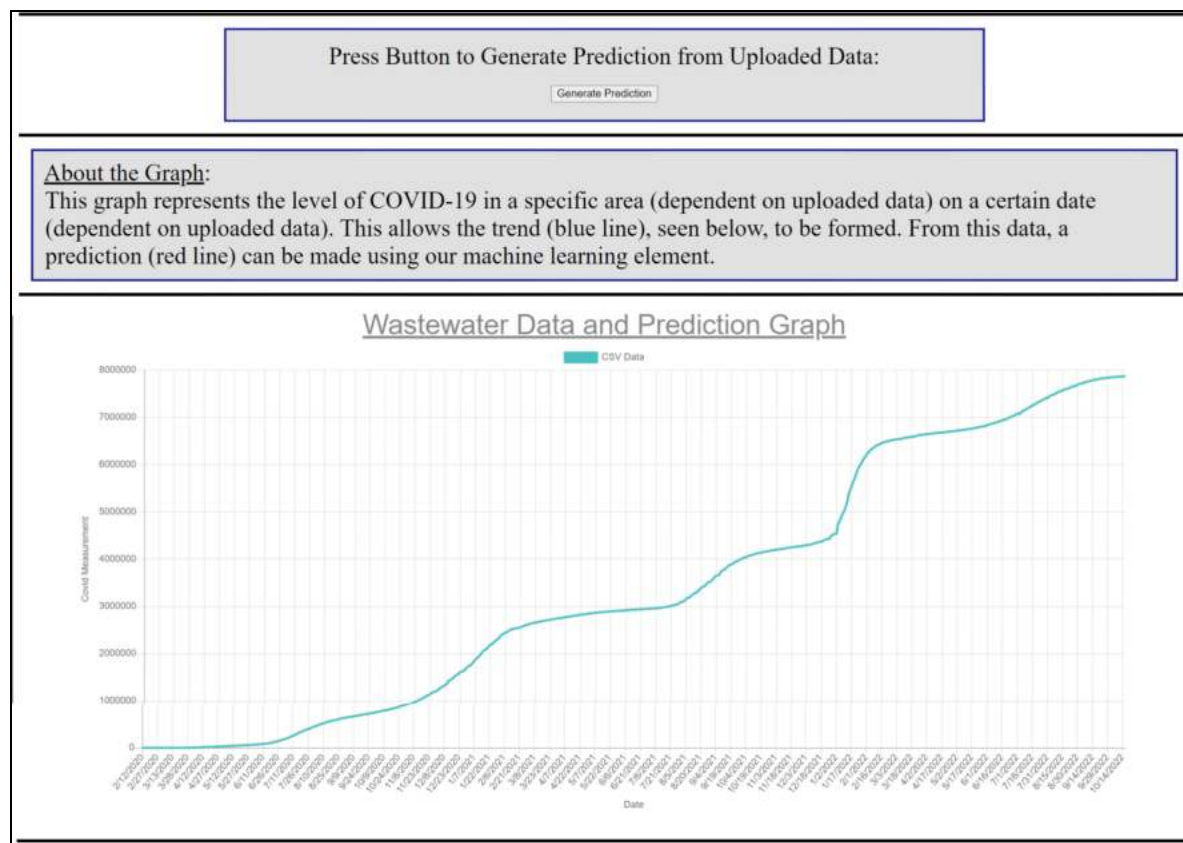


Figure 33: Validation of 'Generate Prediction' Button (Before Prediction)

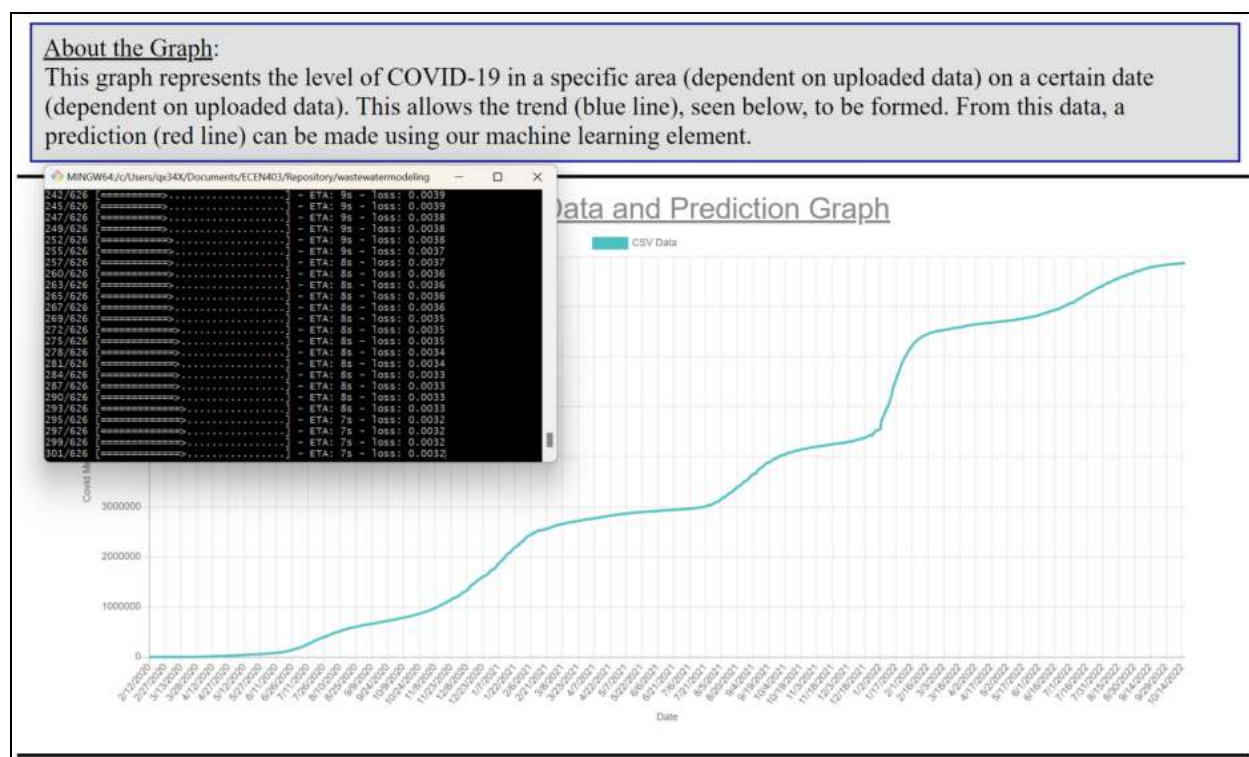


Figure 34: Validation of 'Generate Prediction' Button (During Prediction)

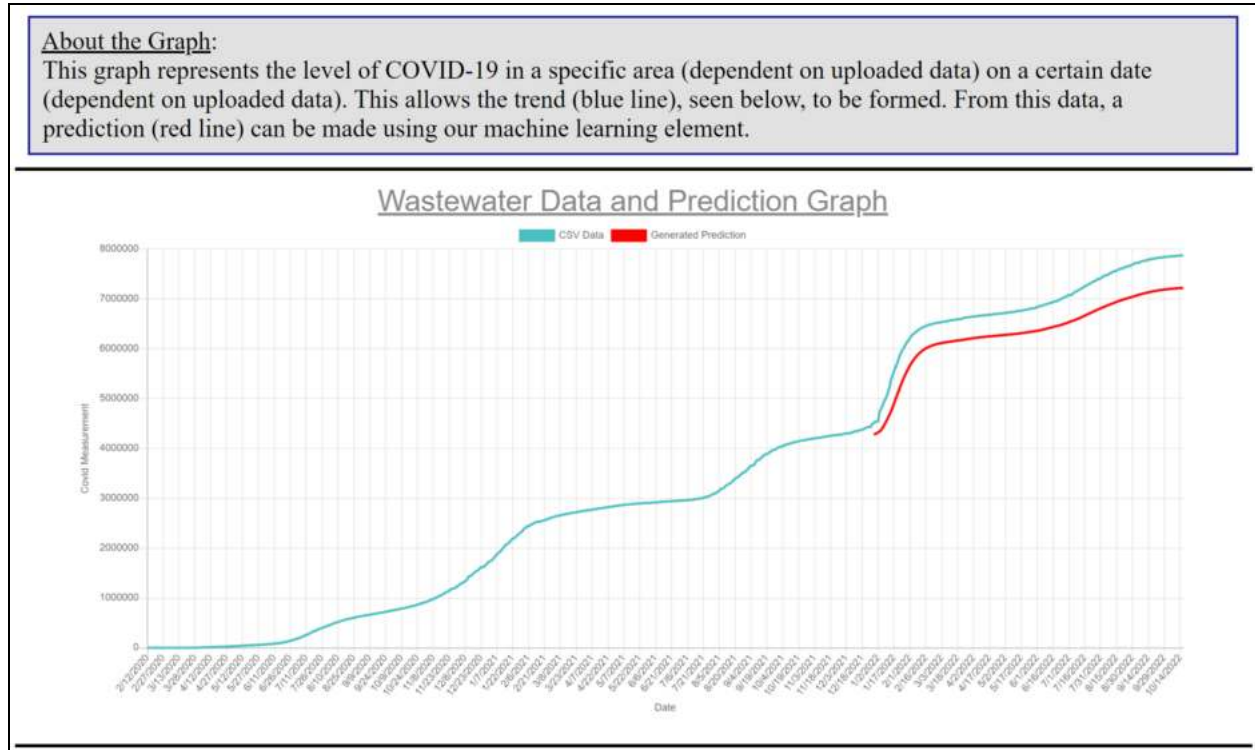


Figure 35: Validation of 'Generate Prediction' Button (After Prediction)

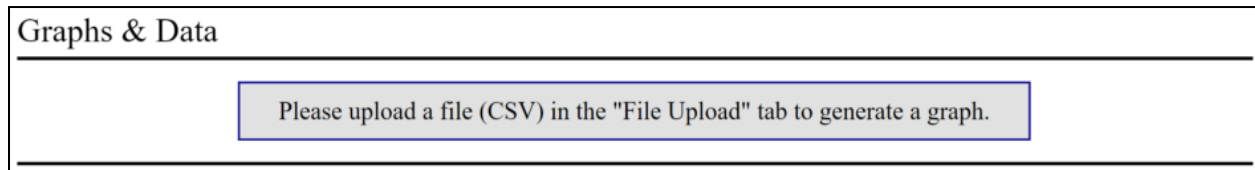


Figure 36: Validation Message for Graph (Before File is Uploaded)

5.3.5. Validation for History Download

The primary validations for the history page relevant to the project are: displaying a message if a file has not been uploaded, ensuring that each file is shown (one by one) when multiple files are uploaded, and confirming that the 'Download File' button works.

Like in section 5.3.4., a message must appear in order to ensure that a blank 'Previously Uploaded Files' section is not displayed to the user. This makes sure that the user only sees results when the proper steps are followed. This message can be seen in Figure 37 below. When uploading multiple files (one by one), the History page should show each file as it is uploaded. In order to validate this, the tab must be viewed after each upload to ensure that the proper output is displayed. This can be seen in Figure 38 below.

The final part to validate the History tab is to ensure the 'Download File' button is working properly. The way this is validated is by clicking on the 'Download File' button and checking the file explorer application for the specified file. This can be seen in Figure 39 below.

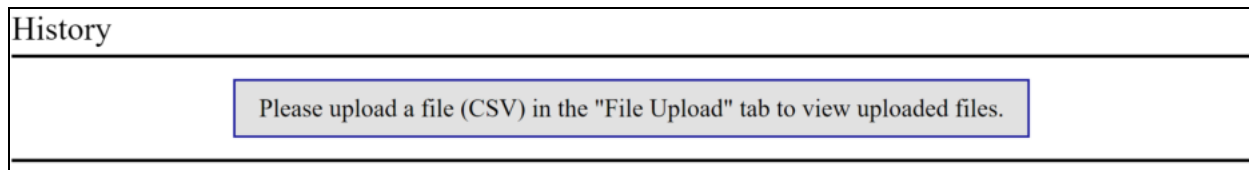


Figure 37: Validation Message for History Page (Before File is Uploaded)

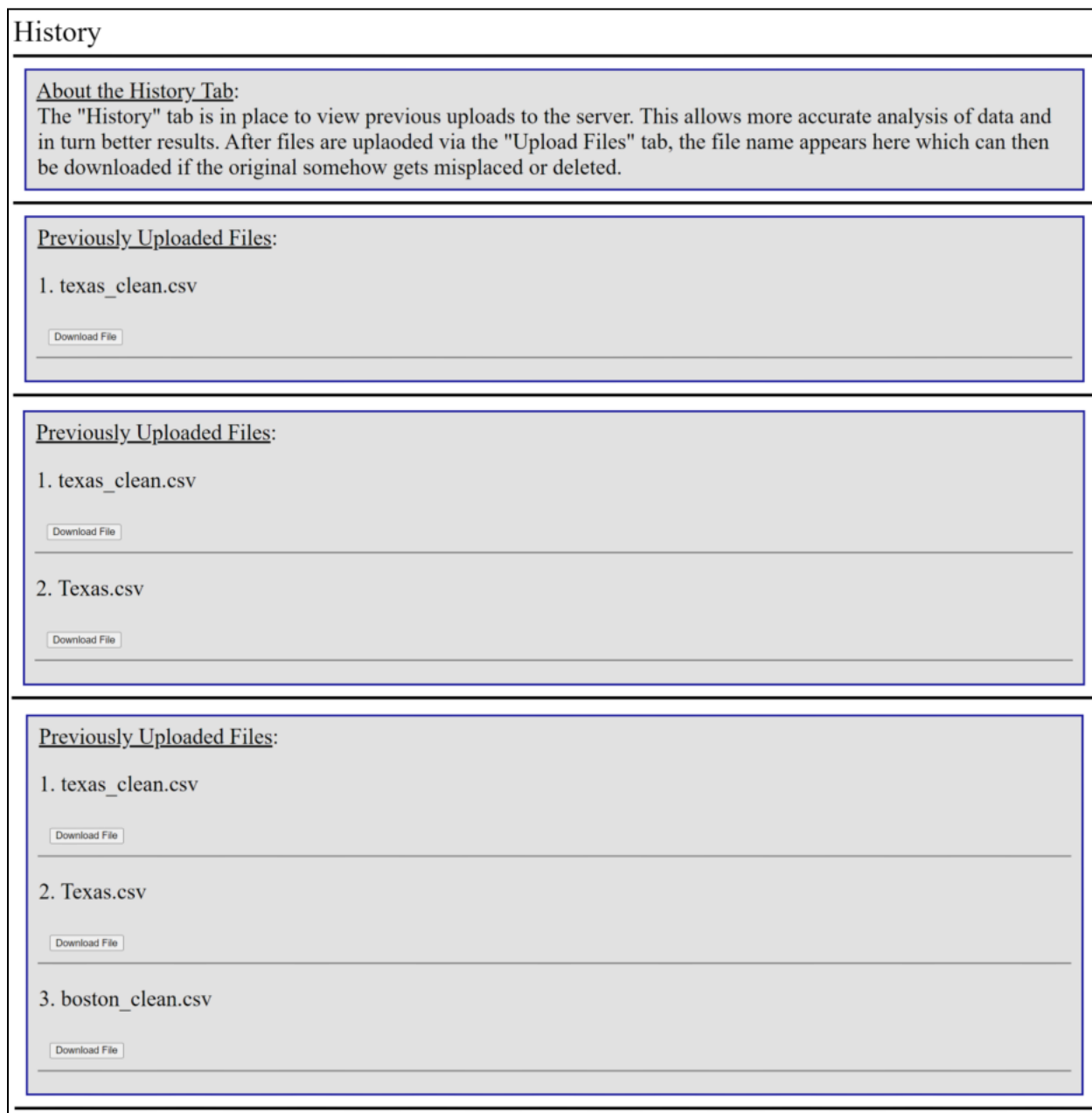


Figure 38. Validation of Multiple Uploaded Files in History Page (*Clarification*: This is a combined photo of 3 separate screenshots: after each file upload, the file is appended to the displayed list with a download button).

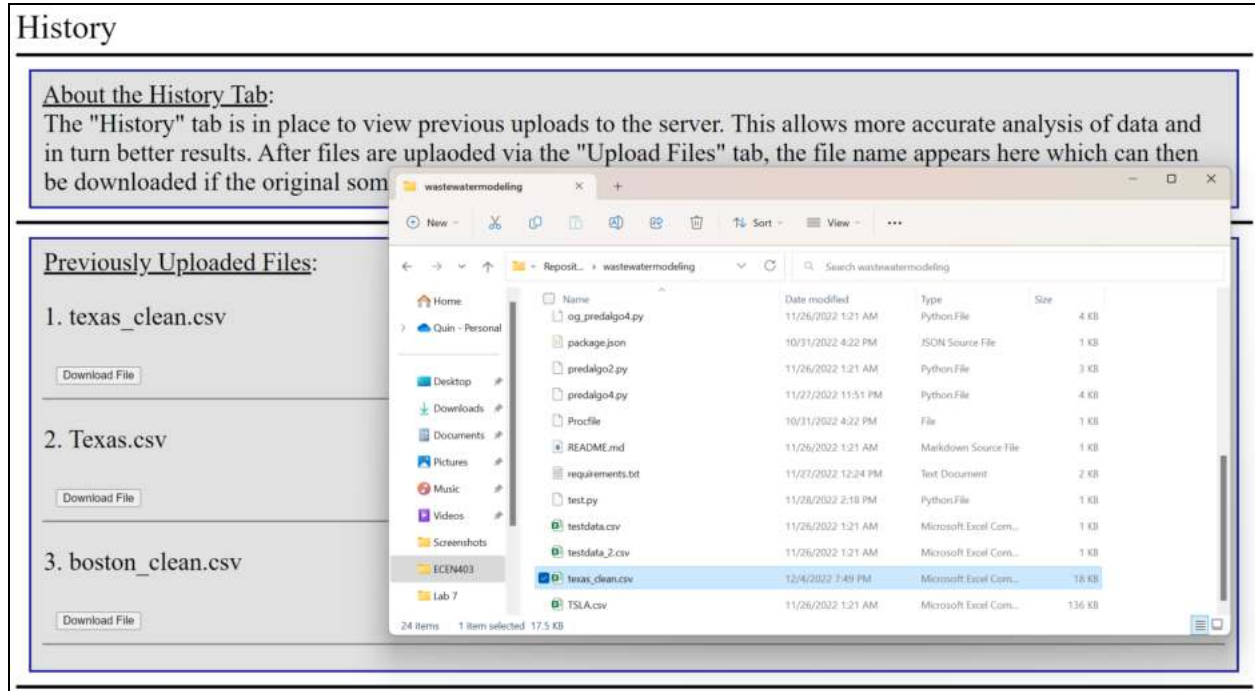


Figure 39. Validation of History's Download (Downloaded file is 'texas_clean.csv which is 1 on the web application and highlighted in file explorer)

5.4. Subsystem Conclusion

5.4.1. Progress

The subsystem contains all intended functionality to support the current web application. This includes all tasks being validated. Such tasks include error detection and proper functionality for the title block, tabs, file upload button, graph and history download features. This has enabled the user to properly use and navigate the web application.

5.4.2. Difficulties and Solutions

The primary difficulties encountered include combining all subsystems into one cohesive web application and ensuring any possible errors are taken care of. The primary reason that combining the subsystems was difficult was because each subsystem uses a different programming language. The solution to this was to use a program acting as a middle man between frontend and backend. This allowed the frontend (HTML and CSS) to be able to connect to the backend (AWS server) through Flask (Python) creating a bridge between the 2 ends of the application (this is accomplished through the Integration Subsystem). The other issue was taking any errors into account. This was difficult because there are so many possibilities of things going wrong that it is hard to think of how to fix it. The solution to this was to spend considerable time "breaking" the web application to find the errors and then fixing them one by one. This allowed the error to be taken care of individually rather than multiple at the same time.

5.4.3. Final Remarks

This subsystem is designed to ensure that the user can navigate the web application with little to no difficulty. This was done through the use of functional tabs to go from one feature to the next in an organized manner. Each of these UI features allows the user to view the work of the other subsystems so ensuring that the UI works is of utmost importance. Not only does the UI need to work, it also needs to be designed in such a way that the user does not have to search for things so ensuring that everything was placed in a location such that it flows was also important