**SE 4485: Software Engineering Projects**

Spring 2024

Detailed Design Document

| Group Number | **8** |
| --- | --- |
| Project Title | **Crop Yield Prediction Model** |
| Sponsoring Company |  |
| Sponsor(s) | **Marc J. Perna (*Main Sponsor)*** |
| Students | **1. Ryan Havens (*Team Leader*)**  **2. Cameron Sutton**  **3. Melvin Sajeev**  **4. Ibrahim Barney**  **5. Nisai Sun** |

# Abstract

This document will go into detail the design decisions, layout, processes, and procedures used to construct the systems atop the current software architecture. It will go over the reasons for making key decisions behind the GUI design, as well as the description for the main class and sequence diagrams. This document will also show and describe the deployment diagram, which will outline what happens when the software is made available for use. Finally, a traceability table will link the class diagrams with the requirements outlined around the first stages of development, as well as evidence of configuration management and standards plus additional references.

# 

# Table of Contents

[**Abstract 2**](#_u9n5317oy4uf)

[**Table of Contents 3**](#_hqg7w9c5bp5v)

[**List of Figures 4**](#_qzrld7alsg92)

[**List of Tables 5**](#_7uoo1tcd6opu)

[**GUI Design 6**](#_67ypzdfrmz62)

[**GUI Description 6**](#_s844cgl9w2bb)

[**Static Model 7**](#_9o3v7w2k2wi0)

[Class Diagram 7](#_xj4rsuyfwasu)

[Class Diagram Description 7](#_mjmu0qjw07fs)

[Deployment Diagram 8](#_wo15yd9mf2rm)

[Deployment Diagram Description 8](#_jd3yuv9wjdzu)

[**Dynamic Models 9**](#_s7yelosevh5y)

[Sequence Diagram 9](#_87dne5f6tseq)

[Sequence Diagram Description 9](#_dcd9crxo158s)

[Sequence Diagram predict() Diagram 10](#_ahbdzngdifwb)

[Sequence Diagram predict() Description 10](#_e57mlbk3a5tu)

[**Rationale for Detailed Design Model 11**](#_y850y8tp7vtt)

[**Traceability From Requirements To Design Model 12**](#_kovbj2m7avvc)

[**Evidence of Configuration Management 13**](#_6jza41efl2j2)

[**Configuration Management Table 14**](#_h4b435poyraf)

[**Engineering Standards And Constraints 15**](#_dmxitehervxj)

[**Additional References 16**](#_2n2sn7zi224)

# 

# List of Figures

* [GUI Design](#_67ypzdfrmz62)
* [Class Diagram](#_xj4rsuyfwasu)
* [Deployment Diagram](#_wo15yd9mf2rm)
* [Sequence Diagram](#_87dne5f6tseq)
  + [Sequence predict() Diagram](#_ahbdzngdifwb)

# 

# List of Tables

* [Traceability From Requirements To Design Model](#_kovbj2m7avvc)
* [Configuration Management Table](#_h4b435poyraf)

# GUI Design

# 

### GUI Description

* **State/County Code:** The code submissions are set up with a simple React form submission. The reasoning for this GUI design is to have an easy and simplistic approach to make for an easy user experience. (Subject to be modified to state name and county name)
* **Output: Numerical/Graphical Display:** Prediction results are displayed to the user in a simple statement of the numerical value of the yield. The user is also displayed all previous years crop yield data for that county.

# 

# Static Model

## *Class Diagram*

### Class Diagram Description

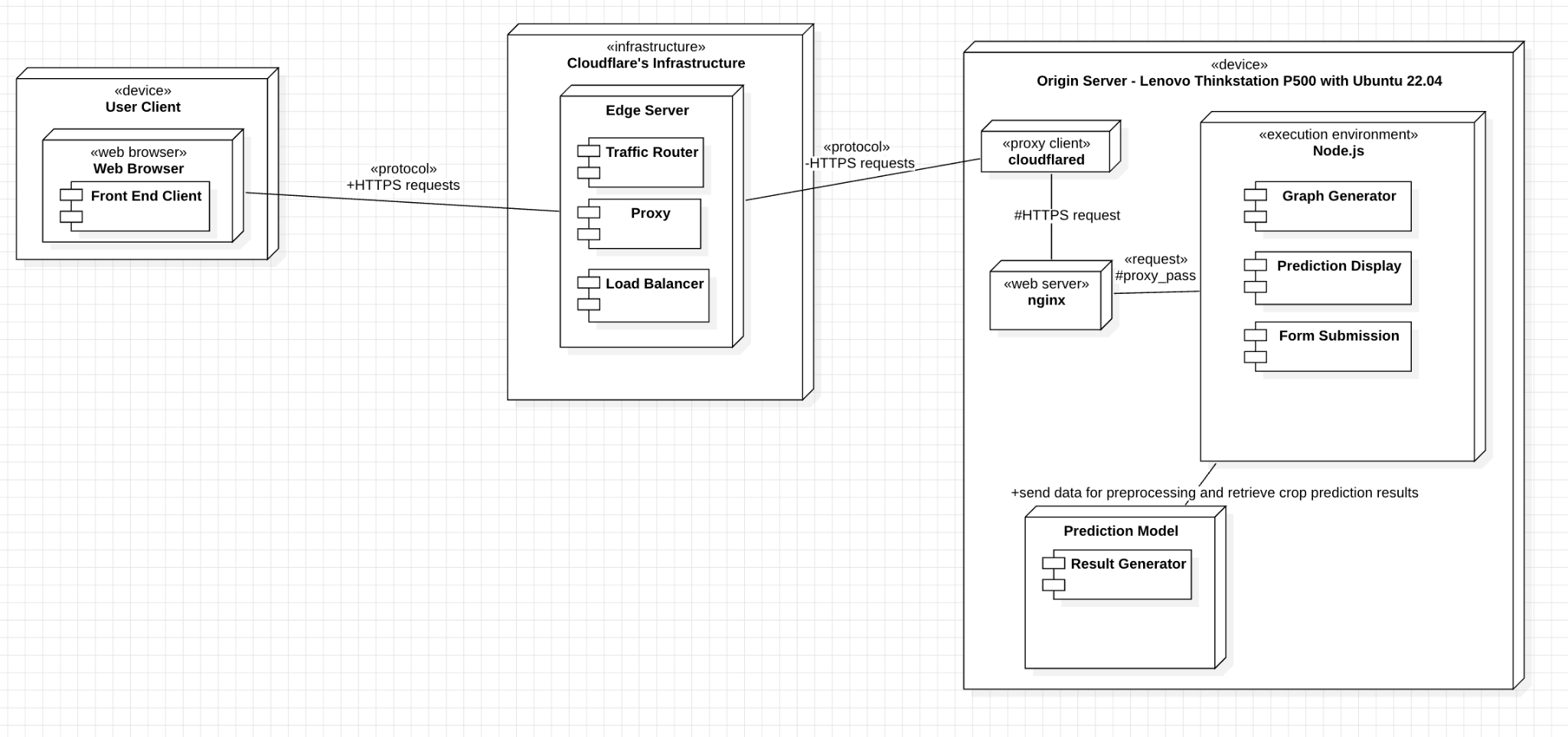
**Client**: Client side contains a form submission. The submitted data consists of two values: a county code and state county, which are passed back as integer values. It also displays the prediction for the data.

**Server**: consists of handling incoming state and county data. Once data is received it holds the predict function that runs the main task of slimming the data, running ML algorithm, and passing back data

**DataFilter**: Class used on the backend that takes in the state and county code from the front end and slims the main dataset down to just that state and county. Once slimmed it passes the cleaned dataset back to the Server class to run the prediction.

## 

## *Deployment Diagram*



### Deployment Diagram Description

**User Client:** For the deployment diagram, the user client(s) will display and access the web application through their ***web browser***. From there, the user client using a web browser will communicate to Cloudflare’s web infrastructure. It will communicate on both ends with HTTPS requests protocol; since many modern internet browsers default to HTTPS traffic as a security precaution, by default.

**Cloudflare’s Infrastructure:** Then, the browser sends out HTTPS requests in order to connect to their edge server in an attempt to communicate with the origin server through a proxy service, which helps to improve security and obfuscate the origin server’s IP address in the event of attempting to find where the origin server is located and mitigate DDoS (*distributed* denial of service) attempts. It will communicate between the***edge server*** and the origin server’s ***cloudflared*** proxy client, by the HTTPS requests protocol.

**Origin Server:** After connecting to the origin server and communicating via the default HTTP/HTTPS ports, the proxy client ***cloudflared*** will communicate with ***nginx*** to proxy traffic into the execution environment using ***Node.js***. It will listen for the customer to generate crop yield predictions based on information entered by the user (if incorrect data is inputted, it will take care of that), and then communicate with the prediction model stored on the origin server. Many of the tasks will be processed in and out of the server to be piped through Cloudflare’s infrastructure, back to the user’s internet browser, using the HTTPS protocol, primarily.

a

# Dynamic Models

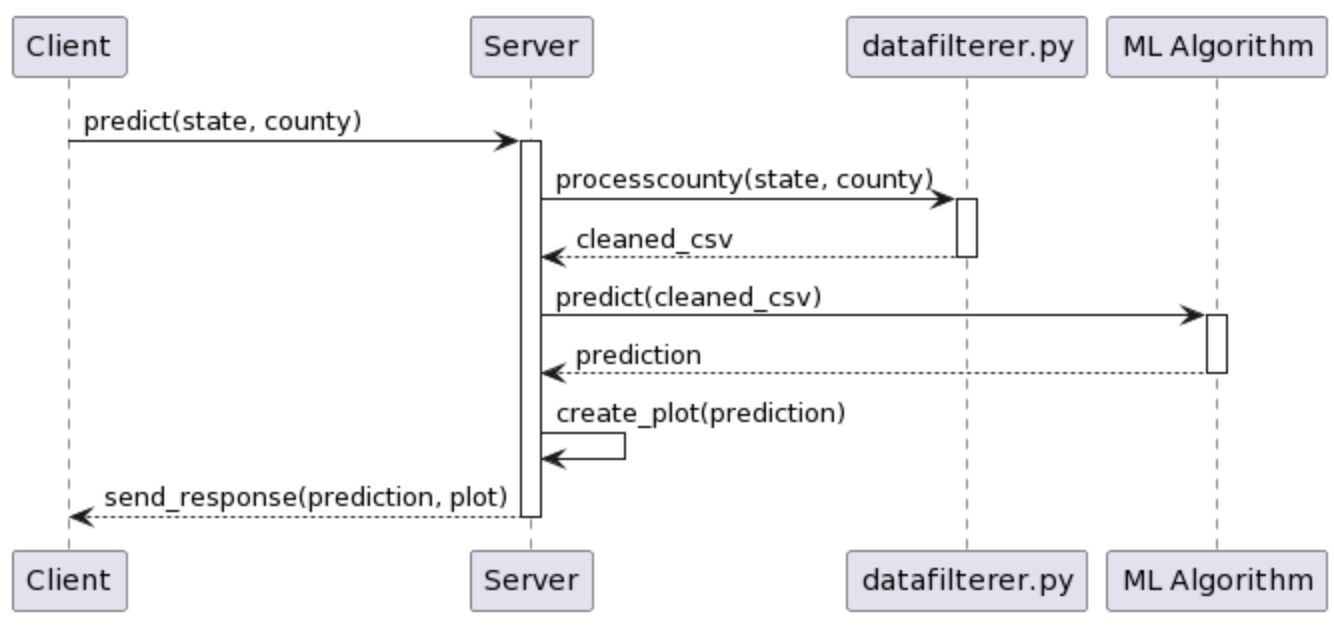
## *Sequence Diagram*

### Sequence Diagram Description

**User State/County Entry Sequence**: User submits the state and county information into the form on the front end react component. The data is then sent to the backend to be predicted within the predict function. Once the prediction results are created the results are then returned and received by the front end. The front end then displays the results to the user on the front web page.

## 

## *Sequence Diagram predict() Diagram*



### Sequence Diagram predict() Description

**predict() Sequence**: When the data enters the predict function it is first run through the data filterer file to be processed down into a cleaned csv file with only that state and county code. The cleaned data is then fed into the linear regression model to create a prediction for the future year’s yield. It finally also creates a graph for the prediction to also send back to the front end.

## 

# Rationale for Detailed Design Model

* *Server-side Design*
  + On the ***server*** side, this design model prioritizes operational securities, uptime, and mitigating potential cyberattacks from bad actors.
    - Much of the online space is filled with DDoS attacks that have a wide variety of different motives, not exclusive to a few.
    - Bad actors can show up at a very inconvenient time and place, for both users and developers; it is best to have failsafe plans in case, in order to mitigate potential cyberattacks.
    - Hide the origin server’s IP address from being publicly exposed and potentially putting the host of the server in danger for bad actors, in regards to *operational security*.
    - Keep dependencies to a minimum, especially with the choice to use a pre-trained model to process and output data.
      * Thanks to this approach, there is no need to use or host a database management system (DBMS) on the server.
        + Thus, mitigating a major point of vulnerability for bad actors, given how precious and valuable data records are, as a commodity.
* *Client-side Design*
  + On the ***client*** side, make it very easy and simple for a wide variety of users.
    - The target demographic of users potentially use devices that lack computational resources, so it is very important to offload the burden to the server, rather than to the user’s local device.
    - The GUI, on the client side, is designed the way it is in order to provide a straightforward and seamless user experience that prioritizes performance and simplicity.

# 

# Traceability From Requirements To Design Model

| **Client/Server** | **FR** | **Mapping** | **Description** |
| --- | --- | --- | --- |
| Server-side | FR1 - FR7 | Data Filter file | Architecture must include components for importing, validating, cleansing, and standardizing data, as well as exporting the cleaned data for further use. |
| Server-side | FR14, FR16 | Server Class, predict() function | Components for training machine learning models on integrated data and evaluating their performance are required. |
| Client-side | FR9 - FR13 | Client class, React component | The architecture needs to include a user interface that allows input of parameters and displays predictions in an understandable format. |
| Client-side | FR12, FR13, FR16 | Client class, React component | Ensures the system can format the output correctly and measures performance accuracy and precision |

# Evidence of Configuration Management

1. **Name of the CM tool used by your team**: Google Docs

1. **Version number of each document after it is checked in**: Google Docs does not have version numbers, but has versions based on timestamps for each session. We can view these in the version history in *File > Version History*.

1. **Version number of each document before it is checked out**: Google Docs does not have check out, or check in, the team works on the latest time stamped version of the document.

1. **Difference between two consecutive versions**: Users can view the revision history through Google docs to see the different versions of the document.

1. **Review of each change**:
   1. After the document is finalized, we have the team double check it to make sure all the input is correct.

1. **Other information that helps the understanding of each change**:
   1. Different team members are assigned different tasks and put this information in the table below.
   2. Every team member is added with their gmail accounts so we know who made each change.

# Configuration Management Table

| **Section Updated** | **Update Description** | **Submitted By** | **Date Updated** |
| --- | --- | --- | --- |
| Formatting, Initial set up | Completed | Ryan | 04-02-2024 |
| GUI Design | Completed | Ryan | 04-02-2024 |
| Evidence of Configuration Management | Completed | Ryan | 04-02-2024 |
| Title Page | Completed | Ryan | 04-02-2024 |
| Sequence Diagram | Completed | Ryan | 04-02-2024 |
| Engineering Standards and Constraints | Completed | Nisai | 04-03-2024 |
| Additional References | Completed | Nisai | 04-03-2024 |
| Table of Contents | Completed | Nisai | 04-03-2024 |
| Abstract | Completed | Nisai | 04-03-2024 |
| Class Diagram | Completed | Ibrahim and Cameron | 04-03-2024 |
| Deployment Diagram | Completed | Nisai | 04-05-2024 |
| Engineering Standards and Constraints | Completed | Ibrahim | 04-05-2024 |
| Traceability from Requirements to Design Model | Completed | Ibrahim, Melvin,  and Ryan | 04-05-2024 |
| Class Diagram description | Completed | Ryan | 04-05-2024 |
| Sequence Diagram Descriptions | Completed | Ryan | 04-05-2024 |
| Rationale for Detailed Design Model | Completed | Ibrahim and Nisai | 04-05-2024 |

# 

# Engineering Standards And Constraints

* IEEE Std 1016-1998-(Revision-2009): Software Design [[pdf](https://cengproject.cankaya.edu.tr/wp-content/uploads/sites/10/2017/12/SDD-ieee-1016-2009.pdf)]
* ISO/IEC/IEEE Std 29148-2018: Systems and software engineering
  + Life Cycle Processes
  + Requirements Engineering [[PDF]](https://paris.utdallas.edu/ewong/se4485/IEEE/ISO-IEC-IEEE-29148-2018.pdf)
* ISO/IEC/IEEE Std 42030:2019: Software, systems and enterprise
  + Architecture evaluation framework [[PDF]](https://paris.utdallas.edu/ewong/se4485/IEEE/ISO-IEC-IEEE-42030-2019.pdf)
* IEEE Std 830-1998: Software Requirements [[PDF]](https://paris.utdallas.edu/ewong/se4485/IEEE/IEEE%20Std%20830-1998-Software-Requirements.pdf)
* IEEE Std 1471-2000: Software Architecture [[PDF]](https://paris.utdallas.edu/ewong/se4485/IEEE/IEEE-Std-1471-2000-Software-Architecture.pdf)
* IEEE Std 12207: Software Life Cycle Processes [[PDF]](https://paris.utdallas.edu/ewong/se4485/IEEE/IEEE%2012207%20(2017)%20-%20Software%20Life%20Cycle%20Processes.pdf)

# Additional References

* Cloudflare Inc., 2024. Cloudflare Docs: Cloudflare Tunnel
* Larman, C., 2012. Applying UML and Patterns: An Introduction to Object Oriented Analysis and Design and Iterative Development. Pearson Education
* Hyman, B., 1998. Fundamentals of Engineering Design. New Jersey: Prentice Hall
* “Reference architecture: Evolving to a SASE architecture with Cloudflare · Cloudflare Reference Architecture Docs,” Cloudflare Docs, https://developers.cloudflare.com/reference-architecture/architectures/sase/ (accessed Apr. 5, 2024).
* Simon, H.A., 2014. A Student's Introduction to Engineering Design: Pergamon Unified Engineering Series (Vol. 21). Elsevier