SE 4485: Software Engineering Projects

Spring 2024

Architecture Document

| Group Number | **8** |
| --- | --- |
| Project Title | **Crop Yield Prediction Model** |
| Sponsoring Company |  |
| Sponsor(s) | **Marc J. Perna (*Main Sponsor*)**  **Daryl P. Nelson (*Former, as of 03/14/24*)** |
| Students | **1. Ryan Havens (*Team Leader*)**  **2. Cameron Sutton**  **3. Melvin Sajeev**  **4. Ibrahim Barney**  **5. Nisai Sun** |

# 

# Abstract

This document presented will outline the software infrastructure style, behavior, and relationships for our crop yield data prediction model. It will go over the reason for choosing this architecture, the project interfaces within it, interactions between the components of the architecture, the technology used, and how it fits with the project's requirements. The goal for this architecture is to clearly outline levels of abstraction and interfaces in preparation for the design and implementation phase of the software, as well as addressing various functional and non-functional requirements.

# Table Of Contents

[Abstract 1](#_heading=h.tchc4poy90ud)

[Table Of Contents 2](#_heading=h.30j0zll)

[List Of Figures 3](#_heading=h.6q1p6d4q02dn)

[List Of Tables 4](#_heading=h.3znysh7)

[Introduction 5](#_heading=h.slgr0uvy31tv)

[Rationale For Using Client-Server Model 6](#_heading=h.brwyaij4idwl)

[Architectural Style Used 7](#_heading=h.qqu484gy3u8p)

[Architectural Model 8](#_heading=h.4f03i457y281)

[Full Application Architecture 8](#_heading=h.oya6m0wf3zps)

[Full Application Components Table 9](#_heading=h.nnma0zhqfdw9)

[Frontend Internal Architecture 10](#_heading=h.kfrhvbqnod5c)

[Detailed Interface Description 10](#_heading=h.7c5ocrv2d7il)

[Backend Internal Architecture 11](#_heading=h.pgbmym5bmny3)

[Detailed Interface Description 12](#_heading=h.gcl7yj5ruus3)

[Technology, Software, and Hardware Used 13](#_heading=h.2s8eyo1)

[Traceability From Requirements To Architecture 15](#_heading=h.vpuqjfci5e92)

[Configuration Management Table 17](#_heading=h.kakoxgw30dg8)

[Engineering Standards and Multiple Constraints 18](#_heading=h.4cf7pj2m96v3)

[Additional References 19](#_heading=h.puetd9fv5rwk)

# List Of Figures

* [Full Application Architecture](#_heading=h.4f03i457y281)
* [Frontend Internal Architecture](#_heading=h.kfrhvbqnod5c)
* [Backend Internal Architecture](#_heading=h.pgbmym5bmny3)

# List Of Tables

* [Full Application Components Table](#_heading=h.nnma0zhqfdw9)
* [Traceability from Requirements to Architecture](#_heading=h.vpuqjfci5e92)
* [Configuration Management Table](#_heading=h.kakoxgw30dg8)

# Introduction

This document goes over our client-server architecture style and why this style works best with our model. Architecture for the crop yield prediction model will be constrained to a client connecting to a server containing crucial data and information used to display, predict, and calculate results based on historical data of crop yields. This document explains the architecture style itself, the reason we chose it and how it applies to our project. It then goes into the structure of the architecture and the technology used in the project.

# Rationale For Using Client-Server Model

* The client-server architecture style allows us to compartmentalize the model, since each part needs to work well for the entire project to work, this style works well with our project, as it allows us to give each component the attention it needs.
* This style allows the project to be divided into easy-to-work-with parts, allowing our team members to divide the work and manage the whole project easily.
* Distinguishing between the back end and the front end will allow us to focus on a user-friendly interface.
* This style works well with the plan we had for our project.

# 

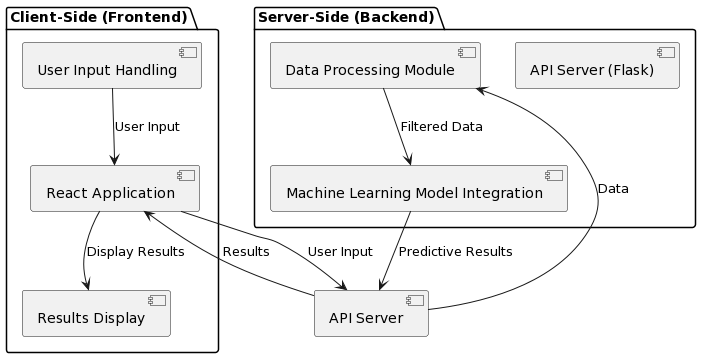
# Architectural Style Used

**The Client-Server Architecture Style**

* The client-server architecture style works with our model since this will allow us to divide the crop yield data model into various parts and we can manage the different parts with different members of the team.
* The predictive model requires us to have different components, each one is as important as the next. The style allows us to separate each one and give them the proper attention needed.
  + On the *server* side we have: the data preprocessing, the model training, etc.
  + On the *client* side we have: the user interface and data being submitted.

# Architectural Model

## Full Application Architecture

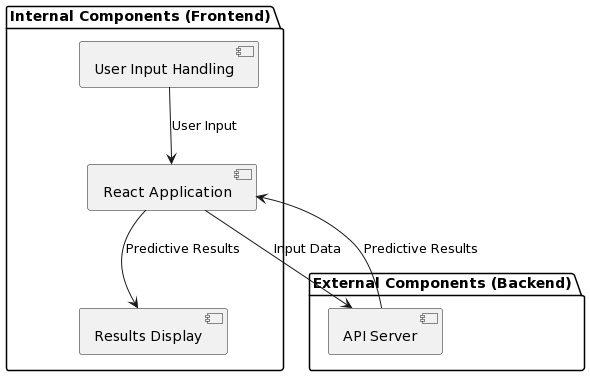


## Full Application Components Table

| **Interface** | **Description** | **Sender** | **Receiver** | **Data Type** | **Internal / External** | **Protocols** |
| --- | --- | --- | --- | --- | --- | --- |
| User Input Handling | UI components capturing user input events and sending data to the backend | User | React Application  (Frontend) | User input data | Internal | HTTPS |
| React Application | Handles communication with external components | User Input Handling  (Frontend)  Api Server  (External) | Results Display  (Frontend)  Api Server  (External) | User input data and the  Predictive Results | Internal and External | HTTPS |
| Results Display | UI components displaying predictive results to the user | React Application (Frontend) | Results Display (Frontend) | Predictive results | Internal | HTTPS |
| API Server | HTTP endpoints handling requests from the frontend and responding with results | React Application(Frontend), Machine Learning Model Integration (Backend) | Data Processing Module, (Backend)  React Application  (Frontend) | User input data, Predictive results | Internal & External | HTTPS |
| Data Processing Module | Module processing incoming data, filtering it, and passing it to the ML model | API Server (Backend) | Machine Learning Model Integration (Backend) | Filtered data | Internal | - |
| Machine Learning Integration | Module integrating with the ML model to execute predictions on filtered data | Data Processing Module (Backend) | API Server (Backend) | Predictive results | Internal | - |

*Note: The* ***user*** *is outside the system boundary, used as a stimulus documented in the table above.*

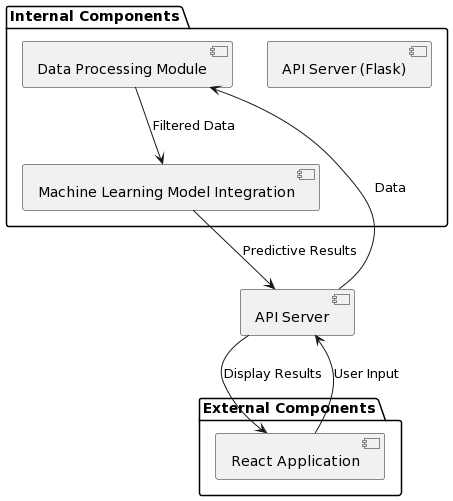
# Frontend Internal Architecture



## Detailed Interface Description

* **React Application:** This component handles the primary form submission.
* **User Input Handling:** This component exists on the React/JS frontend of the application. The user input consists of a form that takes in integer values for a state and county code. It also takes in a String value of yes or no for pesticide use. This data is then sent to the backend of the application via a POST request.
* **Results Display:** The results component receives data from the back end of the application with predictive results based upon the user input provided from the User Input Handling component. The data is then displayed on the web page screen for the user to view the results. Results come in the format of numbers/images.
* **API Server (Flask):** The API server is a Python Flask server that abides by CORS security measures for the web browser. The server is equipped to handle OPTION and POST requests from the front end of the application. It then returns results from the machine learning model to display back to the front end of the application. It is used for test purposes during development.
* **API Server:** Production server that will be utilized during production. Utilizes Nginx.

# Backend Internal Architecture



# 

## Detailed Interface Description

* **API Server (Flask):** The API server is a Python Flask server that abides by CORS security measures for the web browser. The server is equipped to handle OPTION and POST requests from the front end of the application. It then returns results from the machine learning model to display back to the front end of the application. It is used for test purposes during development.
* **API Server:** Production server that will be utilized during production. Utilizes Nginx.
* **Data Processing Module:** The processing module is a Python file that receives input from the user on the frontend of the application. It then accesses the data and reduces the data to only that state and county code. It then sends the filtered data to the machine learning model to be processed. This data is exported as a csv to be used by the model.
* **Machine Learning Model Integration:** The machine learning model is where the predictive analysis takes place. The interface takes in the filtered data that is sent from the data processing module, and uses that data to create predictive results. Once these results are created, the model sends out the data to be sent by the Flask server to send back to the frontend of the application for displaying information.

# 

# Technology, Software, and Hardware Used

The software used for this project includes use of Python and Javascript, in conjunction with various frameworks and libraries. Flask and React.js are chosen for the front-end GUI portion of the software, in order to make it easy to use for users. Python was used to also help pre-process the data used for the crop yields, and to keep the project focused and less bloated. Database management system of choice is PostGreSQL, due to its community support, high performance under even stressful and high loads, and wide industry adoption; however, of note, the database management system (DBMS) for this project, PostgreSQL, will be used as a repository for data records to train the model that will interface with the client.

For the hardware, using a decommissioned workstation as a compute server, this is used to help train the neural network models. The compute GPU, an NVIDIA Tesla P40, is for training the model that we have setup, to predict the future crop yield based on a wide variety of factors and previous datasets that the model is fed and trained under. ECC RAM is used to make sure that any data that goes through the RAM will not be susceptible to data corruption or errors, in critical situations. As a risk mitigation, a CyberPower 1500VA/900W UPS battery backup will be used in the event of a brownout, or worst, a blackout to make sure critical information going in-and-out of the server will not be corrupted, so the system will be able to safely shut down in a critical situation, mentioned previously.

**Hardware**

* + Lenovo ThinkStation P500 Workstation
    - GPUs:
      * NVIDIA Tesla P40 (24GB GDDR5 VRAM, *Compute*)
      * AMD Radeon GPU R9 255 OEM (*Display/Primary*)
    - RAM: 64GB (RDIMM, ECC, DDR4)
    - CPU: Intel Xeon E5-2699 v3 (18-core, 36-threads, 2.3 GHz)
  + CyberPower UPS 1500VA/900W Battery Backup

**Software**

* + Operating System
    - Ubuntu Linux 64-bit, 22.04
  + Programming Languages
    - Python 3
    - JavaScript
  + Programming Libraries / Frameworks
    - React.js (JavaScript library)
    - Flask (Python framework)
  + Database Management System
    - PostgresSQL 16.x
  + Programs
    - JetBrains’ Data Grip
    - Git
    - VSCode

**Communication between the Application Server and the Database Server**

In this current server infrastructure, the application server communicates with the dataserver by communicating to the local IP address at a different port (ex. 127.0.0.1:8006, localhost IP at a different listening port) to retrieve specific data found within the database, matching certain conditions of the user’s input. For purposes of recordkeeping many datasets, in order for the model training to be set, the PostgreSQL database will only for data record keeping to feed, store, and train the crop yield prediction model. Python libraries and programs will also be utilizing the *virtual environment* and *psycop2* library (C language-based PostgreSQL database adapter for the Python programming language) to communicate with the DBMS (PostgreSQL), sending various PostgreSQL commands and interfacing with the database(s), when necessary.

# Traceability From Requirements To Architecture

| **Layer** | **FR** | **Use Cases** | **Description** |
| --- | --- | --- | --- |
| Data Processing | FR1 - FR7 | Data Pre-Processing | Architecture must include components for importing, validating, cleansing, and standardizing data, as well as exporting the cleaned data for further use. |
| Data Integration | FR15 | Data Integration | The architecture should facilitate the merging of different data sources (crop yield and pesticide usage) into a unified dataset for analysis. |
| Model Training | FR14, FR16 | Machine Learning Model | Components for training machine learning models on integrated data and evaluating their performance are required. |
| Presentation/User Interface | FR9 - FR13 | User Interface | The architecture needs to include a user interface that allows input of parameters and displays predictions in an understandable format. |
| Data Output and Analysis | FR12, FR13, FR16 | Output Formatting, Accuracy and Performance | 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
2. **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*.
3. **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.
4. **Difference between two consecutive versions**: Users can view the revision history through Google docs to see the different versions of the document.
5. **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.
6. **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** |
| --- | --- | --- | --- |
| Technology, Software, and Hardware Used | Completed | Nisai | 02-27-2024 |
| Abstract | Completed | Nisai and Ibrahim | 03-01-2024 |
| Introduction | Completed | Nisai and Ibrahim | 03-01-2024 |
| Architectural Style Used | Completed | Ibrahim | 03-01-2024 |
| Architectural Model | Completed | Ibrahim | 03-01-2024 |
| Rationale | Completed | Ibrahim | 03-01-2024 |
| Traceability from Requirements to Architecture | Completed | Melvin | 03-02-2024 |
| Configuration Management Table | Completed | Ibrahim | 03-09-2024 |
| Evidence of Configuration Management | Completed | Ibrahim | 03-09-2024 |
| Engineering Standards and Multiple Constraints + Additional References | Completed | Nisai | 03-15-2024 |
| Architecture Model | Completed | Ryan | 03-19-2024 |
| Components Table | Completed | Ryan | 03-19-2024 |
| Frontend Model / Descriptions | Completed | Ryan | 03-19-2024 |
| Backend Model / Descriptions | Completed | Ryan | 03-19-2024 |
| Formatting / Links Update | Completed | Ryan | 03-19-2024 |
| Revisions and Fixes (Based on Sponsors’ feedback and a second overall review by the group) | Completed | Ryan and Nisai | 03-22-2024 |

# Engineering Standards and Multiple Constraints

* IEEE Std 1471-2000: Software Architecture
* ISO/IEC/IEEE Std 42030:2019: Software, Systems and Enterprise — Architecture Evaluation Framework
* ISO/IEC/IEEE Std 29148-2018: Systems and software engineering — Life cycle processes — Requirements engineering
* IEEE Std 830-1998: Software Requirements
* IEEE Std 1016-1998-(Revision-2009): Software Design

# Additional References

* Lattanze, A.J., 2008. *Architecting Software Intensive Systems: A Practitioner’s Guide.* CRC Press
* Bass, L., Clements, P. and Kazman, R., 2003. *Software Architecture in Practice.* Addison-Wesley
* Vasav (2023) ‘Client Server Architecture: Types, Examples, & Benefits’, *Red Switches*, 20 October. Available at: https://www.redswitches.com/blog/client-server-architecture/ (Accessed: 18 March 2024).
* Ashanin, N. (2019, June 4). *Documentation in software architecture*. Medium. https://medium.com/@nvashanin/documentation-in-software-architecture-4f2e4159c4fc