CS Capstone Final Document

June 12, 2018

Linking Seasonal Weather Data to AgBizClimate^{TM}

Prepared for

Oregon State University

Clark Seavert _____ _____
Signature *Date*

Prepared by

Group26

Thomas Noelcke _____ _____
Signature *Date*

Shane Barrantes _____ _____
Signature *Date*

Shengpei Yuan _____ _____
Signature *Date*

Abstract

The purpose of this document is to provide documentation regarding the *AgBizClimate* project. We will start off the document by giving a general overview of the *AgBizClimate Project*. This will inclue information about the goals of the project, and information about the project stakeholders. Next we have our requirements document. In this section we will also discuss how our requirements have changed over the last several terms. Next we will be discussing the design document. In this section we will first display our original design document. We will then discuss how our design has changed over the term. After the design document we will display the tech review. Next, we will also display the project poster. Finally we will provide some project documentation regarding how project setup, running the project, how the project works and guids for any API's we are using. After this we will discuss any technical resources for learning more about the technologies that our project uses. Finally, We will end this document with our conclusions and reflections section. This section will involve reflecting on this project and discussing what went well and what didn't go well.

Contents

1 Introduction

1.1 The Team

Thomas Noelcke Through out this project Thomas played the role of team lead as well as front end developer and atmospheric science technical expert.

Shane Barantes This project depended on Shanes skills in devops and environment setup as well as back end web programming.

Shengpei Yuan Shempei was our python expert as well as our QA testing expert.

1.2 Project Sponsors

This project was sponsored by Clark Seavert from OSU the applied economics department at Oregon State University. Professor Seavert specializes in Enterprise budgets, capital investments, and crop and livestock leasing.

1.3 Project Importance

The importance and goal of the AgBizClimate project was to provide accurate short term climate data (up to seven months out) to farmers, ranchers, and climate researchers in order for them to more accurately make decisions for their respective crafts.

1.4 Client Involvement

Clark Seavert was largely hands off during this project. However, that's not to say he didn't play a role. Clark was very helpful we reviewing documents and presentations. Clark also provided timely feed back on any change requests to the project.

It is also important to note that though our project sponsor was not directly involved in the development of the short term climate tool his project manage Sean Hammond was a great resource through out the project. Through out the project we worked with Sean to work through technical problems and road blocks. Through out the term we met with Sean on a weekly basis to discuss progress on the project along with any major blockers. During these meetings Sean provided guidance on any problems we may have run into.

1.5 Definitions, Acronyms and Abbreviations

REST - Representation State Transfer, This is a type of architecture that manages preforms operations on the state of the program. This is especially popular in web development.
API- Application Programming Interface. This is a piece of software that allows a connection to another piece of software providing some sort of service.
Thredds Data Server - This is a web server that provides meta-data and data access for scientific data sets using OPeNDAP along with some other remote data access protocols.
OPeNDAP - Open-source Project for a Network Data Access Protocol. This is the protocol we will be using to retrieve the data sets from the Thredds data server.
NMME - North American Multi-Model Ensemble. This is a data set that brings together a variety of different

weather models into one data set.
Climate Scenario - This is a theoretical calculation of yields, inputs and of the overall budget for one situation based on the climate data.
NETCDF - This is a file storage format for large scientific data sets especially good for any data that is referenced on a grid and related to is geo-location.
SQL Database - This is a relational database that makes storing and accessing data easy.
Container - A virtualized operating system that is used to host and deploy web development projects. This allows projects to be easily portable between different operating systems and platforms.
Mount Bind - This is the practice of mounting a directory from the native os to the container such that if either the container or the native operating system. This allows changes in the files to be reflected in both the container and the native operating system.

1.6 References

[1] C. F. Seavert, "Negotiating new lease arrangements with the transition to direct seed intensive cropping systems," 2017.

[2] S. Y. Thomas Noelcke, Shane Barrantes, "Problem statement," 2017.

[3] S. Y. Shane Barrantes, Thomas Noelcke, "Requirements document," 2017.

[4] S. Y. Thomas Noelcke, Shane Barrantes, "Tech review," 2017.

[5] S. Y. Shane Barrantes, Thomas Noelcke, "Design document," 2017.

1.7 Overview

Seasonal climate is one of the essential factors that affects harvest, returns of crops and farming investment programs of enterprises and organizations. As a sub-project of AgBizLogic, AgBizClimate is dedicated to deliver essential information about climate change to farmers, and help professionals to develop management pathways that best fit their operations under a changing climate. This project aims to link the crucial seasonal climate data from the Northwest Climate Toolbox database to AgBizLogic so that it can provide specific analysis and illustrations through powerful graphics. AgBizClimate is designed to enable farmers and agriculture enterprises to decide appropriate farming investment projects for their crops and products.

Currently AgBizClimate has a long-term climate tool but no such tool exists for short term climate data. We will implement a tool to extract short-term climate data from the Northwest Climate Toolbox database, display it to the user and allow the user to adjust crop yields, inputs and costs. Moreover, a landing tool will be developed to allow users to switch between short term seasonal tool and long-term climate data tool.

Seasonal weather data is important for farmers and land managers as climate has great impact on crops. For instance, the precipitation on different days across the life cycles of crops may have different impacts on the harvest. Land managers and farmers used to rely on past experiences of climate to make decisions for specific farming operations to reduce the negative impacts and make use of favorable climate conditions. However, these individual experiences of weather data are often limited and inaccurate. Professional tools like software systems could be adopted to build models for decision making based on available seasonal climate data.

1.8 Project Scope

This project is a part of a much larger AgBiz Logic^{TM}program. However, the purpose of this project is to add a short term climate tool to the *AgBizClimate* module. This limits the scope of the project to the *AgBizClimate* Module. Additionally, we will only be adding the short term climate data tool as the long term climate data tool already exists.

1.9 Product Functions

AgBizClimate is a web based decision tool that will allow users to gain specific insight on how localized climate data for the next seven months will affect their crop and livestock yields or quality of products sold and production inputs. The *AgBizClimate* tool will allow users to input their location (state, county) and a budget for the specific crop or livestock enterprise. *AgBizClimate* will select climate data for the next seven months for that location and provide graphical data showing temperature and precipitation. Users will then be able to change yields or quality of product sold by a percentage they think these factors will affect and modify production inputs. Finally the tool will calculate the net returns.

1.10 User Characteristics

AgBizClimate users can be split into two subgroups, agricultural producers and climate researchers. The first subgroup, agricultural users who use this product tend to be between fifty and sixty years old of mixed gender. Their educational background ranges from high school to the completion of college. The primary language this group uses is English, but there are some Spanish users as well. Most of the users in this group tend to have novice computational skills. The primary domain for these users is agricultural and business management. Most agricultural producers who use this product are motivated by the potential profit that the decision tool *AgBizClimate* could potentially offer. The second subgroup, climate researchers range from ages twenty to forty and are of mixed gender. The educational background for most climate researchers exceed the postgraduate level with their primary language being English. These users generally have advanced computational skills and are motivated by the easily accessible climate and weather data.

1.11 Constraints

There are several key constraints that this product has to work within. Firstly, We are limited by the availability and completeness of the data from the NMME data set. The NMME Data set does not have data for every point on in the country. Secondly, we must use the Threadds server hosted by University of Idaho to get the data in the NetCDF format. The tooling provides a variety of access methods however, the only method that currently works is downloading the whole file. Fourthly, we dont have access to any of the hardware that *AgBizClimate* is exists on as it is being managed by a third party. This will prevent us from improving the hardware or cause roadblocks if their servers are having issues. Lastly, we are limited to using the languages Python and JavaScript since we are integrating our product into an already existing project.

1.12 Assumptions and Dependencies

We are assuming that the NMME Thredds data base will allow us to pull location based temperature and precipitation data. This data will come in the form of a NetCDF files which we will then read and format a JSON response. Due to the fact that we are writing an addition to an existing project we do not need to interact with the user budgets as these have already been defined. This fact extends to the calculations portion of the *AgBizClimate* product. Our team will simply be accessing data via the NMME threadds database, will then format the data, store the data, and hand the data over to the front end or some other sort of client.

2 Requirements Document

2.1 Original Document

CS Capstone Software Requirements Document

November 3, 2017

Linking Seasonal Weather Data to AgBizClimate^{TM}

Prepared for

Oregon State University

Clark Seavert _____ _____
Signature *Date*

Prepared by

Group26

Thomas Noelcke _____ _____
Signature *Date*

Shane Barrantes _____ _____
Signature *Date*

Shengpei Yuan _____ _____
Signature *Date*

Abstract

This Software requirements document will cover the requirements for the AgBizClimate project. We will first give a general introduction of the project. This section will provide some context for why we are doing this project and what this project hopes to accomplish. Next will give an overall description of what our application does. This section will what the application does in broad no technical terms. Then we will discuss specific Requirements. This section will provide the technical details of the application including the functional requirements for the application. In the next section there is an approximate project schedule displayed in a Gantt chart. Finally, The last section provides the Tables and Figures referenced throughout the document.

Contents

List of Figures

1 Introduction

1.1 Purpose

This SRS describes the requirements and specifications of the AgBizClimate^{TM}web application. This document will explain the functional features of this web application. This includes the interface details, design constraints and considerations such as performance characteristics. This SRS is intended out outline how we will proceed with the development of the *AgBizClimate* system.

1.2 Scope

This project is a part of a much larger AgBiz Logic^{TM}program. However, the purpose of this project is to add a short term climate tool to the *AgBizClimate* module. This limits the scope of the project to the *AgBizClimate* Module. Additionally, we will only be adding the short term climate data tool as the long term climate data tool already exists.

1.3 Definitions, Acronyms and Abbreviations

REST - Representaional State Transfer, This is a type of architecture that manages the state of the program. This is esspecially populare in web development.
API- Application Programming Interface. This is a peice of softwared that allows a connection to another peice of software providing some sort of service.
NWCTB - Northwest Climate Toolbox. This is the database we will be connecting to that will provide the short term climate data we plan to use.
Climate Scenario - This is a theoretical calculation of yields, inputs and of the overall budget for one situation based on the climate data.
SQL Database - This is a relational database that makes storing and accessing data easy.

1.4 References

[1] C. F. Seavert, "Negotiating new lease arrangements with the transition to direct seed intensive cropping systems," 2017.

[2] S. Y. Thomas Noelcke, Shane Barrantes, "Problem statement," 2017.

1.5 Overview

Seasonal climate is one of the essential factors that affects agricultural production. As a module of *AgBiz Logic*, *AgBizClimate* delivers essential information about climate change to farmers, and help professionals to develop management pathways that best fit their operations under a changing climate. This project aims to link the crucial seasonal climate data from the Northwest Climate Toolbox database to *AgBiz Logic* so that it can provide changes in net returns of crop and livestock enterprises through powerful graphics and tables.
Currently *AgBizClimate* has a long-term climate tool but no such tool exists for short term climate data. We will implement a tool to extract short-term climate data from the Northwest Climate Toolbox database, display it to the user and allow the user to adjust crop and livestock yields or quality of products sold and, production inputs. Moreover, a landing tool will be developed to allow users to switch between short-term seasonal tool and long-term climate data tool.

2 Overall Description

2.1 Product Functions

AgBizClimate is a web based decision tool that will allow users to gain specific insight on how localized climate data for the next seven months will affect their crop and livestock yields or quality of products sold and production inputs. The *AgBizClimate* tool will allow users to input their location (state, county) and a budget for the specific crop or livestock enterprise. *AgBizClimate* will select climate data for the next seven months for that location and provide graphical data showing temperature and precipitation. Users will then be able to change yields or quality of product sold by a percentage they think these factors will affect and modify production inputs. Finally the tool will calculate the net returns.

2.2 User Characteristics

AgBizClimate users can be split into two subgroups, agricultural producers and climate researchers. The first subgroup, agricultural users who use this product tend to be between fifty and sixty years old of mixed gender. Their educational background ranges from high school to the completion of college. The primary language this group uses is English, but there are some Spanish users as well. Most of the users in this group tend to have novice computational skills. The primary domain for these users is agricultural and business management. Most agricultural producers who use this product are motivated by the potential profit that the decision tool *AgBizClimate* could potentially offer. The second subgroup, climate researchers range from ages twenty to forty and are of mixed gender. The educational background for most climate researchers exceed the postgraduate level with their primary language being English. These users generally have advanced computational skills and are motivated by the easily accessible climate and weather data.

2.3 Constraints

There are several key constraints that this product has to work within. The first constraint is that we only have access to two data parameters from the North West Climate Tool box, precipitation and temperature. Secondly, we only have access to their data via the NWCTB API which could have additional restrictions such as limited usage per day, mislabeled data, or poor documentation. Thirdly, we dont have access to any of the hardware that *AgBizClimate* is exists on as it is being managed by a third party. This will prevent us from improving the hardware or cause roadblocks if their servers are having issues. Lastly, we are limited to using the languages Python and JavaScript since we are integrating our product into an already existing project.

2.4 Assumptions and Dependencies

We are assuming that the Northwest Climate Toolbox is a functional API that will allow us to pull location based temperature and precipitation data. This data will most likely come in the form of a text body of which we will then format into a JSON object and store in a Mongo database for future use. Due to the fact that we are writing an addition to an existing project we do not need to interact with the user budgets as these have already been defined. This fact extends to the calculations portion of the *AgBizClimate* product. Our team will simply be accessing data via the NWCT API, then format the data, store the data, and hand the data over to the tool while will provide some additional front end support.

3 Specific Requirements

This section contains all of the functional and quality requirements of the *AgBizClimate* System. We will give detailed description of what's being added to the *AgBiz Logic* system along with the features we will implement.

3.1 External Interface Requirements

This section provides a detailed description of all inputs into and outputs from the short term climate tool for the *AgBizClimate* system. This section will also provide descriptions for the hardware, software and communication interfaces. Additionally we will provide detailed prototypes for the user interface for the short term climate data tool.

3.1.1 User Interface

When the user first navigates to *AgBizClimate* from the *AgBiz Logic* main page the user will be directed to a landing page. This landing page will allow the user to either select the existing long term climate tool or the short term climate tool that we will be developing. On this page there will be a brief description of the tool and what does. This description will also describe the difference between the long term climate data tool and the short term data climate tool. Below this description will be two buttons one to run the long term climate data tool and one to run the short term climate data tool. Clicking the long term climate data tool button will take you to the long term climate data tool page. Clicking the short term climate data tool button will take you to the short term climate data tool page. For a prototype of this page see section 5 Figures and Tables figure 2.

After clicking on the short term climate data tool page you will be directed to a page that will allow you to create a new climate scenario. This page will allow the user to chose which budgets they would like to adjust and make notes about this scenario. This page also allows them to cancel or delete the scenario they are currently working on. For a prototype of this page see section 5 Figures and Tables figure 3.

Once the user Selects their budgets, makes notes on the scenario and clicks continue the will be redirected to a page where they will be prompted to enter their location. This page will take a county and a sate as input. This page will have a drop down menu for the state and the county. Before the user can enter a County they will be forced to enter a state. Once a state is entered the county drop down menu will auto populate and the user will be allowed to enter a county. For a prototype of this page see section 5 Figures and Tables figure 4.

After clicking continue the user will be taken to the plots for the location they selected. This page will initially have the 7 month average precipitation plot selected as the default. However, the user can select from four options via a drop down menu. Changing what is selected in the drop down menu will change which plot is displayed. Once the user has reviewed the plots they can then enter in how much they think their yield will be effected based on the climate data. For a prototype of this page see section 5 Figures and Tables figure 5.

Next the user will be directed to the budget review page. This page will display a summary of the budget. The summary of the budget will include Income, General Cash Costs and A total summary of the budget. This page will allow the user to adjust the cost per unit, input costs, and quantity sold and will adjust their budget in near real time. This page will also allow the user to remove or add inputs to their general costs. This page will also allow the user to save their budget and output it as a PDF. For a prototype of this page see 5 Figures and Tables figure 6.

3.1.2 Hardware Interfaces

This project requires no designed hardware and the hardware this system is running on is managed by the Operating system as a result no hardware interfaces are necessary. Additionally the connection the the NWCTB is managed

over the network and requires no hardware interface.

3.1.3 Software Interface

There are several software interfaces in this application, one between the back end and the front end, one between the back end and the Database and finally a software interface between our RESTFUl API and the NWCTB. The majority of the connections between the front end and the back end are managed for us the the current *AgBiz Logic* system. However, we will still need to connect our API to the back end to provide the data from the NWCTB. To make this connection we will use a RESTFULL API. This will allow for the back end code to make a direct call to our API to extract the data.
We will also need a connection between the NWCTB and our API. At the time of writing this document we have not received any information from the NWCTB regarding the API they have promised access too. However, We do know that we will need to authenticate the connection with the data base, send a request and then receive the result. Authenticating the connection will involve some sort of hand shake where we pass the NWCTB a key. Once we authenticate the connection we can then send a request. A request will contain a variety of information including a location for which we want the data. Once we have sent the request we will then wait for the response and accept the data. We may also need to store this data is a data base. If this is necessary we will need an additional software interface between the database and the API.

3.1.4 Communications Interfaces

This project will require communication between its various parts. One key lane of communication is between the front end and back end of the application. Another key lane of communication will be between the NWCTB and our API. Most of the communication between components will be carried out through HTTP requests. These are managed by the operating system and will not effect the way our application works.

3.2 Functional Requirements

In this Section we will list and discuss the functional requirements for this project. Each functional requirement shall have an ID, Title, Description and Dependencies. The ID shall be unique for each requirement. The description will give a detailed expiation of the requirement. The dependencies section will list the requirements that will need to be complete before starting work on that requirement.

3.2.1 Functional Requirements For API

Functional Requirement 1.1 ID: FR1.1
Title: Request for Users Location Data
Description: The API shall take an HTTP Post request with the users State and County as parameters. Upon receiving the HTTP Post request the API will strip the parameters off the Request and store the values in variables for later use.
Dependencies: FR2.1 and FR2.2

Functional Requirement 1.2 ID: FR1.2
Title: Transforming Users Location data.
Description: The API shall take the users Sate and county information and transform it into latitude and longitude. It shall then store the results in a variable for later use.
Dependencies: FR1.1 and FR2.1

Functional Requirement 1.3 ID: FR1.3
Title: NWCTB Authentication
Description: The API shall authenticate the connection with the NWCTB by sending a request to connect to the NWCTB. This request will include a validation key to ensure that the connection is valid.
Dependencies: None.

Functional Requirement 1.4 ID: FR1.4
Title: Requesting data
Description: After the correct user information has been gathered and the connection with the NWCTB has been authenticated the API shall send a request to the NWCTB API for the information the user requested. This request shall be made via an HTTP Get request.
Dependencies: FR1.1, FR1.2, FR1.3 and FR2.1

Functional Requirement 1.5 ID: FR1.5
Title: Receive Response from NWCTB
Description: After sending the request for the data to the NWCTB API the API shall wait for a response to the request. Once a response has been send the API will receive this response through an HTTP Response. If the request for the user data results in an error the API shall send the appropriate response code along with a useful error message. This shall be done through an HTTP Post response.
Dependencies: FR1.1, FR1.2, FR1.3, FR1.4 and FR1.5.

Functional Requirement 1.6 ID: FR1.6
Title: Processing the Data to JSON
Description: Once the request has been successfully received the API shall process the raw data in the response body of the HTTP response sent by the NWCTB API. The raw data shall be placed into a JSON object and useful labels shall be applied to the JSON object so it can be easily displayed later.
Dependencies: FR1.1, FR1.2, FR1.3, FR1.4, FR1.5 and FR2.1

Functional Requirement 1.7 ID: FR1.7
Title: Send the Data to the Front End
Description: Once the data has been place into a JSON object and formatted the API shall send the resulting data to the front end application so it can be displayed. This shall be done via an HTTP Post response. The JSON object that contains the formatted data shall be placed in the response body. The appropriate headers for the Response shall be set to indicate that the response contains a JSON object.
Dependencies: FR1.1, FR1.2, FR1.3, FR1.4, FR1.5, FR1.6 and FR2.1

3.2.2 Functional Requirements For Front End

Functional Requirement 2.1 ID: FR2.1
Title: Landing Page.
Description: Upon loading the *AgBizClimate* application the user will be directed to a loading page this page will allow the user to select between the short term climate tool and the long term climate tool.
Dependencies: None.

Functional Requirement 2.2 ID: FR2.2
Title: Request data from API
Description: After selecting the short term data tool the user will be directed to a page where they will be required

to enter the location they would like to analyze. This page will require the user to first enter a state via a drop down menu and then a county via a drop down menu. Clicking the continue button will send an HTTP request to the API for the climate data for the location the user requested. The Front end will then wait for a response to the request sent to the API. This request shall be made through an HTTP Post request.
Dependencies: FR2.1

Functional Requirement 2.3 ID: FR2.3
Title: Plot the Data
Description: After the API processes the request for the data for the location the the user specified the front end will receive the response from the API. The front end shall then take the data out of the response body and use a plotting library to plot the data and display it to the user.
Dependencies: FR1.1, FR1.2, FR1.3, FR1.4, FR1.5, FR, 1.6, FR1.7, FR2.1 and FR2.2

Functional Requirement 2.4 ID: FR2.4
Title: Entering Adjustments to yield
Description: Once the data has been plotted and displayed the the user front end shall allow the user to make adjustments to the expected yield of the crop they are analyzing. This shall be done via a text box. The user input shall be limited to a decimal number with no more than one decimal place.
Dependencies: FR1.1, FR1.2, FR1.3, FR1.4, FR1.5, FR, 1.6, FR1.7, FR2.1, FR2.2 and FR2.3

Functional Requirement 2.5 ID: FR2.5
Title: Redirect To Budget Tool.
Description: Once the user has entered the adjustments to the yield for the crop they are analyzing they shall then be redirected to the existing *AgBizClimate* budgeting tool. This tool will be sent the data they entered in the previous step and shall reflect the adjustments to yield that they made.
Dependencies: FR1.1, FR1.2, FR1.3, FR1.4, FR1.5, FR, 1.6, FR1.7, FR2.1, FR2.2, FR2.3 and FR2.4

3.2.3 Functional Requirements for Testing

Functional Requirement 3.1 ID: FR3.1
Title: Unit Tests For API Routes.
Description: We shall write unit tests to cover 100 percent of our API routes. Each route shall have a test case that is non-trivial and seeks to emulate how the route will be used by the application. These test cases shall test that the correct response is returned from each API route.
Dependencies: FR1.1, FR1.2, FR1.3, FR1.4, FR1.5, FR1.6 and FR2.1

Functional Requirement 3.2 ID: FR3.2
Title: Unit Tests for User Actions on Front End
Description: We shall provide unit tests for the various user actions on the front end of the application. The front end tests shall ensure that clickable objects on the page perform the correct action. We shall provide tests for all clickable objects that redirect the application to another page. Our tests shall ensure that if a clickable action is clicked that the correct action is taken for each clickable object.
Dependencies: FR1.1, FR1.2, FR1.3, FR1.4, FR1.5, FR, 1.6, FR1.7, FR2.1, FR2.2, FR2.3, FR2.4 and FR2.5

Functional Requirement 3.3 ID: FR3.3
Title: Unit Tests for Budget Save
Description: We shall provide unit tests that ensure that the budget is being correctly saved once the user is all

done making adjustments to their budget. This test case shall ensure that the values sent to the back-end of the application are posted to the database. The test will also ensure that the values posted in the database are correct. Dependencies: FR1.1, FR1.2, FR1.3, FR1.4, FR1.5, FR, 1.6, FR1.7, FR2.1, FR2.2, FR2.3, FR2.4 and FR2.5

3.3 Performance Requirements

In this section we will list the performance metrics for our application. This performance metrics will describe acceptable performance for our application. Each requirement will have a title and a description that provides details for how that metric will be measured.

3.3.1 Performance Metric 1

Title: Run Time Performance
Our application shall provide the plotted data in a reasonable amount of time 100 percent of the time. Reasonable being defined as less than 5 seconds. This time shall be measured from the time the user presses the next button after selecting their state and county to the time the web page is loaded with the plot being displayed.

3.3.2 Performance Metric 2

Title: Reliability
Our application shall provide the correct result to the user 100 percent of the time. This result can include an error if the user enters a location that doesn't exist or if the data the user request is unavailable.

3.4 Design Constraints

The constraints of this project can be divided into two parts. Software constraints and hardware constraints. In this section we will discuss these constraints.

3.4.1 Software Constraints

This project shall be implemented using Django, Angular JS and SQL databases. What we add must also be done using Django, Angular JS and SQL.

Additionally, what we create must work with the the existing *AgBiz Logic* system. This means we will be forced to use software interfaces and databases that have already been implemented that we have no control over.

3.4.2 Hardware Constraints

Another factor that will effect how our software works is the hardware that this system is running on. The hardware this system runs on has already been determined and is managed by the university. We don't expect that the hardware will affect our project. However, we have no control over the hardware the *AgBiz Logic* system is running on.

3.5 Other Requirements

This section will cover any additional information pertinent to this project but not covered in other sections.

3.5.1 Stretch Requirements

In this section we will discuss requirements not apart of our contract with our client. These goals will not be required as apart of this project but are objectives that we would like to complete.

Stretch Requirement 1.1 ID: SR1.1
Title: Location Via a Map
Description: In requirement 2.2 instead of using two drop down menus to select the location the user would be able to use a pin on a map of the US to select their location. This would be done through an existing library that would allow us to take a pin on a map and produce a latitude and longitude.
Dependencies: FR2.1 and FR2.2

4 Gantt Chart

An approximate schedule for the AgBizClimate project is shown below. Note that this schedule is subject to change.

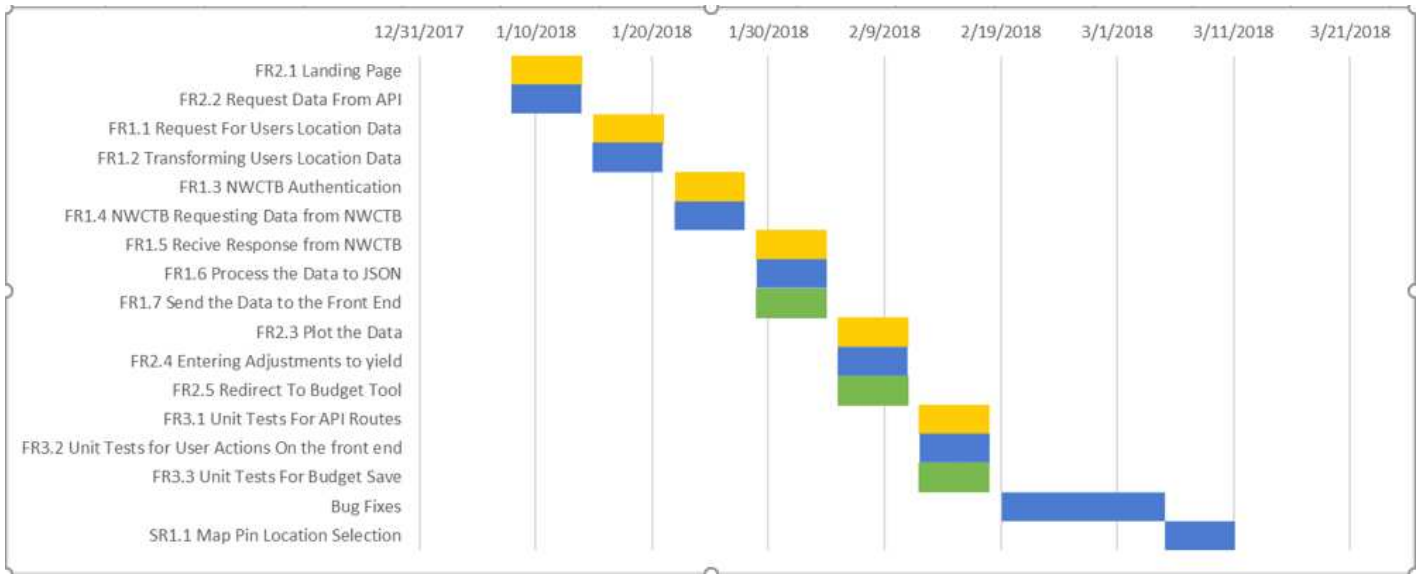


Figure 1: Project Schedual

5 Figures and Tables

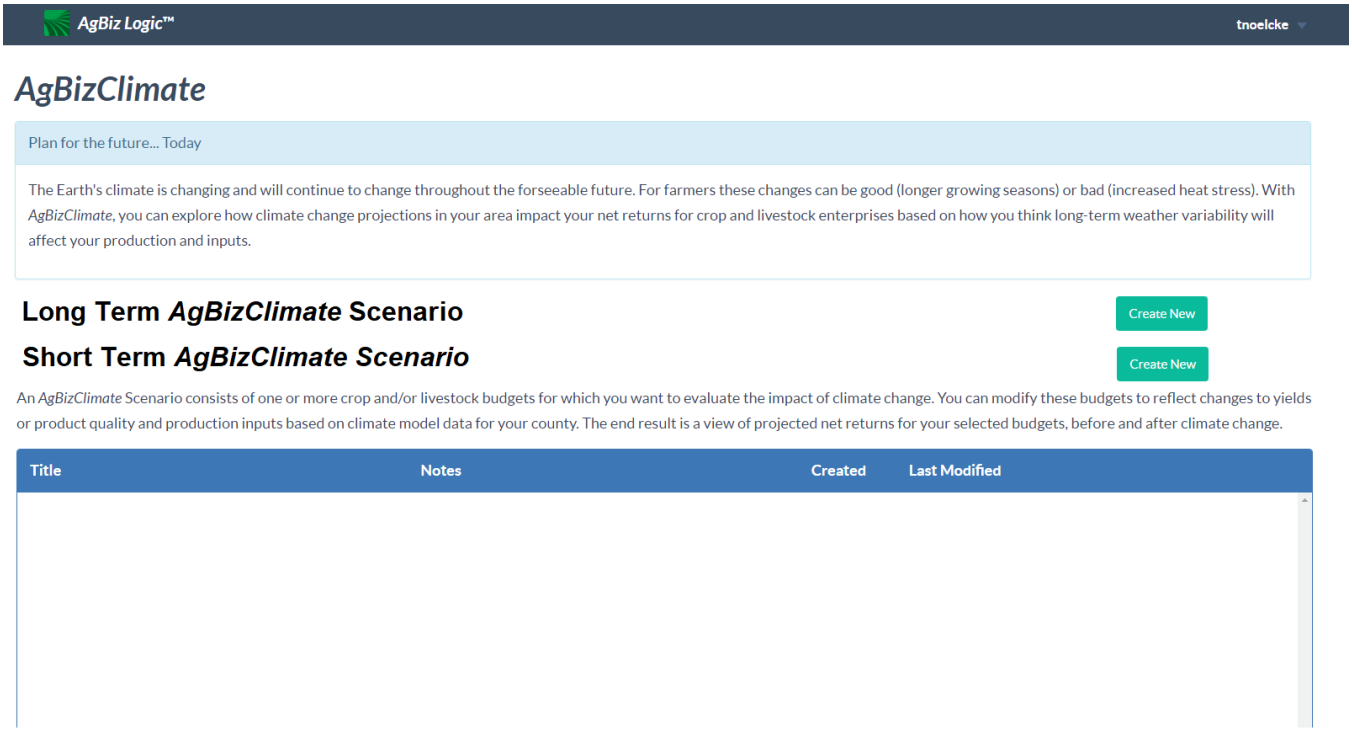


Figure 2: Landing Page

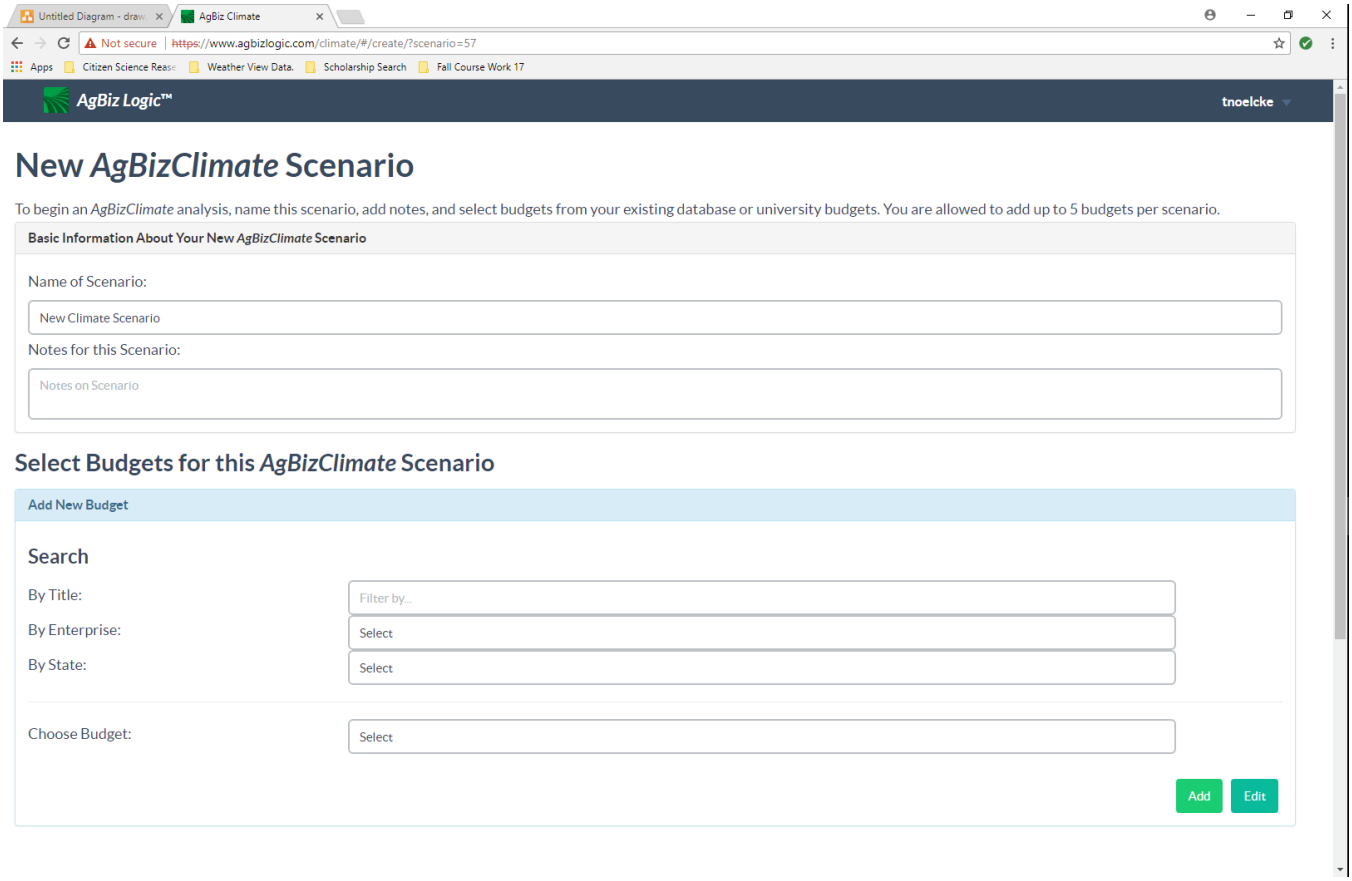


Figure 3: Allows user to select budgets and make notes

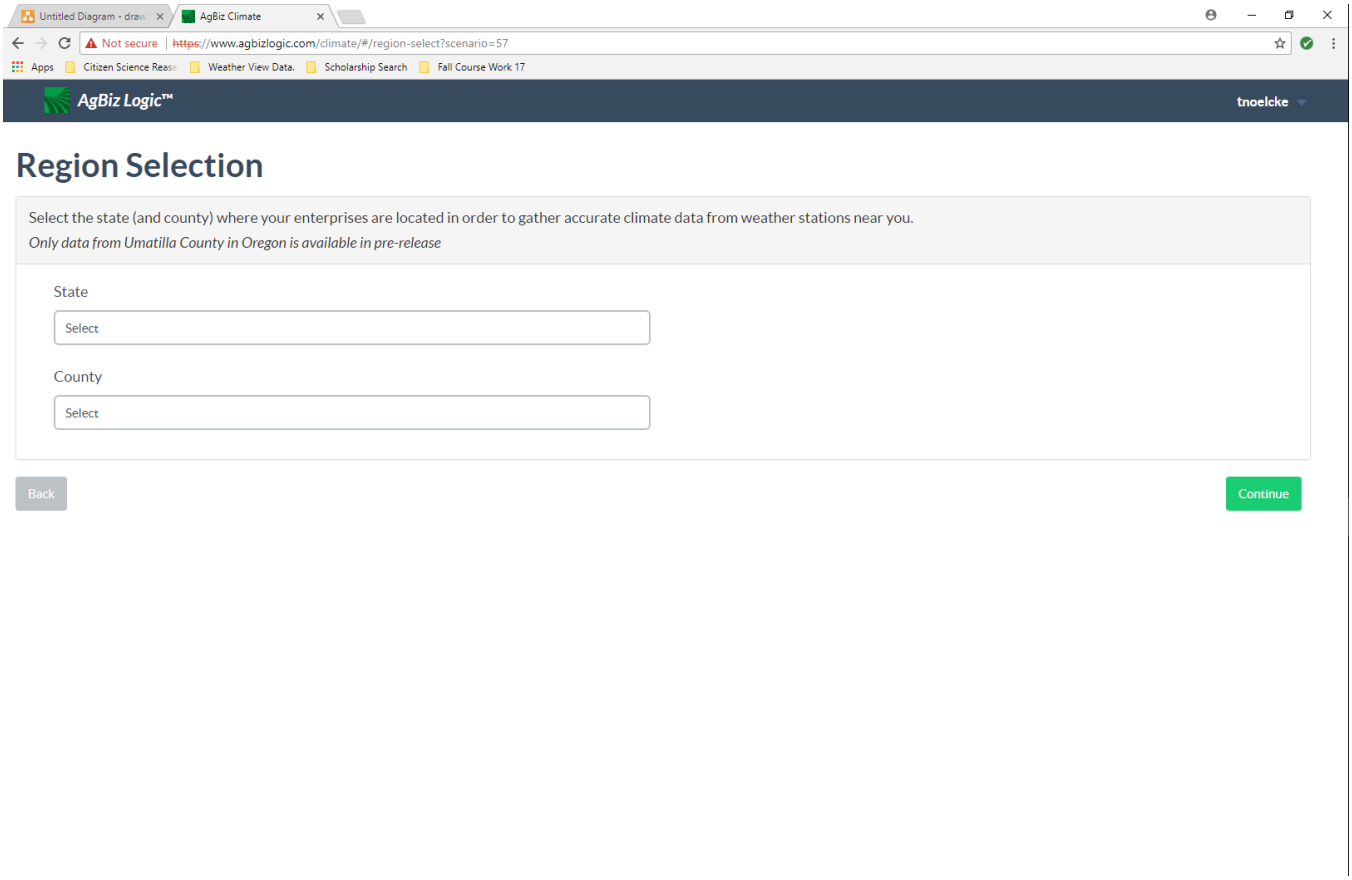


Figure 4: Allows User to select location

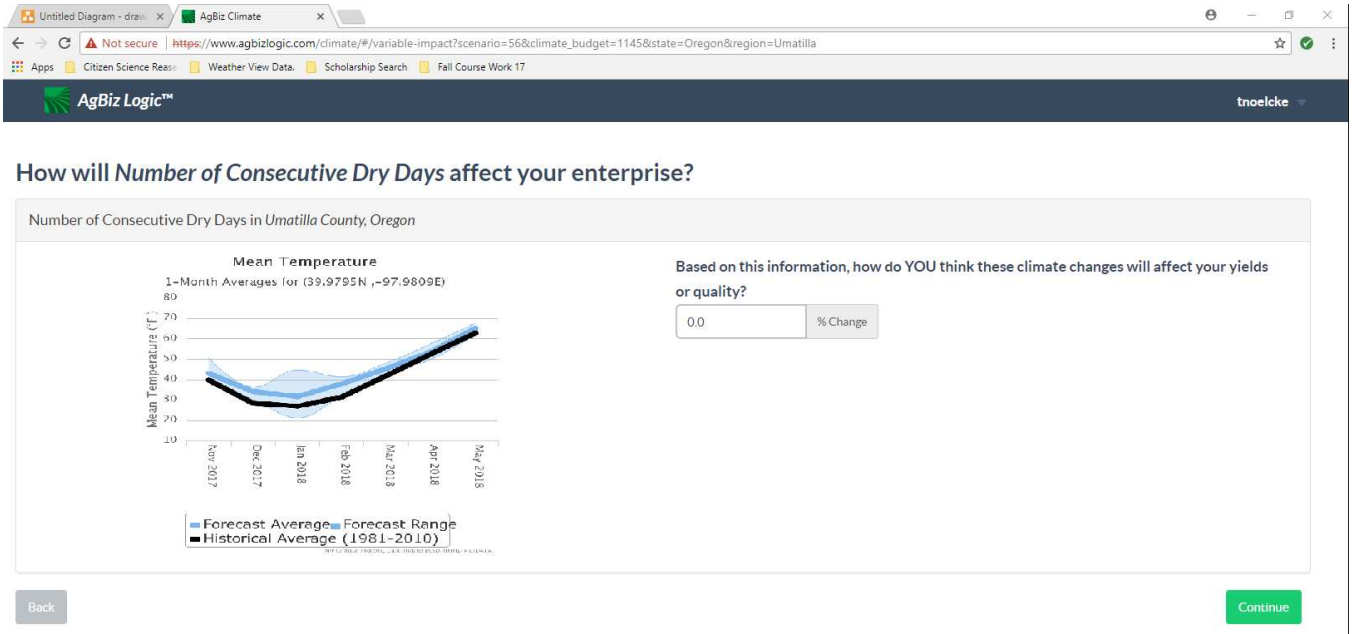


Figure 5: Displays the Data in a plot

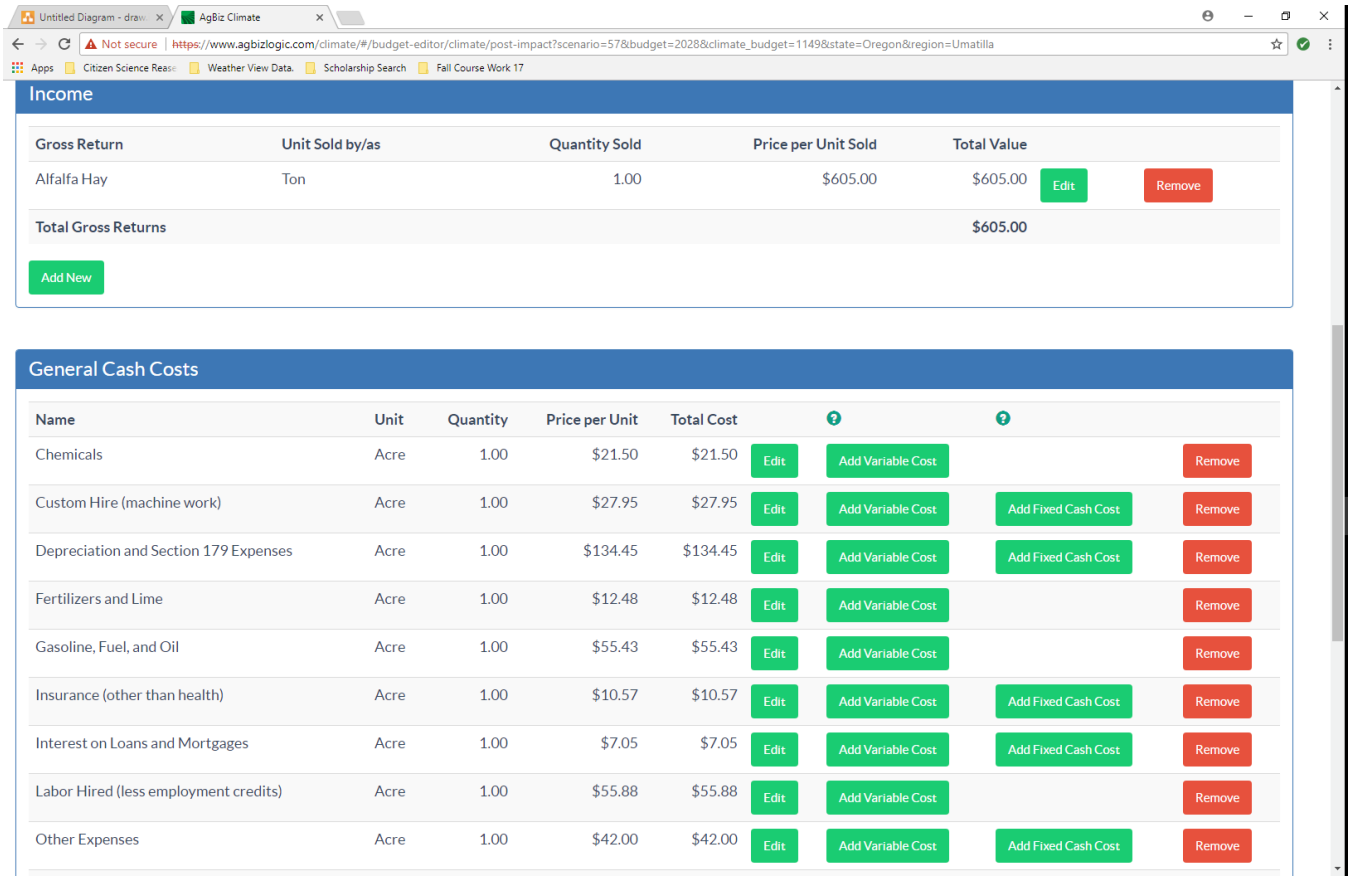


Figure 6: Allows user to view their budget

2.2 Changes

2.2.1 Introduction

Through out this process some things mentioned above the introduction has changed to reflect the changes to the application. Specifically, We are no longer using NWCTB as we had originally assumed. Instead we are using a Threadds server hosted by the university of Idaho. As such we would change the references to the NWCTB to NMME Threadds database.

2.2.2 Overall description

Several things changed over the term in the overall descriptions section, constraints, and Assumptions and Dependencies. The constraints changed because we were originally thinking we were going to be required to use the NWCTB API to get the data. Instead of the NWCTB were actually constrained by the downscale NMME Threadds database hosted at University of Idaho. The assumptions and dependencies for similar reasons. We were originally assuming that we were going to need to use the NWCTB to get the data. However, because we ended using the Threadds database this introduced a slew of new dependencies including, netcdf4, the OPeNDAP protocol and HDF5.

2.2.3 User Interface Changes

Through out the process of building this application we found that there were a variety of changes were made to the UI design we originally proposed. Below we will discuss the changes we made to the UI. We will break down these changes by section as that's how it was done in the design document. Each section represents a page.

Landing page Originally we though we were going to use the climate manager page to branch between the different climate scenario types. We were going to do this using a button that would create a new short term climate scenario. We ended up choosing to branch a bit later in the process as it made our application simpler. As such the climate manager now only contains one button to create a new climate scenario. The climate manager also allows you to review old scenarios and to delete old scenarios. This was not functionality included in the original document.

New Climate Scenario Page The New Climate Scenario page design was more or less unchanged however we did not adequately describe the page in our initial requirements document. We never specified that the user would be able to add up to five budgets using this page.

Region Select Page Originally, the region select page was just going to be to select the region. However, early on in development we realized that if we had our user select the region and forecast scope on this page that it would simplify our work flow. So instead of just selecting state and county the user now selects climate forecast type, state and county. We also did not mention that the user will not be allow to continue with out entering all three of these items.

Charts Page In our initial draft of our requirements document we had planed to use a drop down menu to allow the user to select different plots that they might want to look at. We decided that it would be better to use tabs for the same functionality. So now the user may switch between different charts using tabs rather than using a drop down menu.

Climate Summary Page We added a new page to the User Interface description. In our original requirements document we did not know we would have a summary page, however the client already had a summary page implemented that we were able to plug the data we created in the short term tool. This page allows the user to see a summary and visual representation of what they did during their short term scenario.

2.2.4 Interfaces

Given that the source of our data changed we also made some changes to the hardware and software interfaces. The big change to the hardware interface is that we are now using a Threadds server located at the university of Idaho. Previously we were assuming we would be able to use the NWCTB to get the data.

We also made some changes the software interface. Given that the source of our data changed several times through out the term we are no longer using the NWCTB to get our data. In fact we are now downloading the data monthly and placing in on the dev server. We are then passing the file to the container our application is run in via a mount bind. As such the software interfaces for our application changed drastically over the term. We now have to include the dev server as part of our software interface. It is also important to note that we are interfacing with the Threadds server at University of Idaho.

2.2.5 Functional Requirements changes

In this section we will discuss how the functional requirements have changed through out the course of the project. We will only discuss the requirements that have changed and will not discuss requirements that have not changed.

Functional Requirement 1.3: The data source As mentioned previously the source of the data changed from the NWCTB to the threads server hosted at University of Idaho. Additionally, the way our application uses this data changed through out the project. We were initially planning getting each data point as needed. However, we ended up storing the whole data file on the dev server.

Functional Requirement 1.4: Getting the Data In our original draft of the document we hadn't had a chance to look at the data yet and did not know that the data would be somewhat sparsely populated. So as the project moved along we realized that it would be possible to request a point and get NAN as a result even though it was in the middle of the data set. As such we were required to do some searching if the initial result was NAN. We eventually settled on searching with in a 20 mile radius as it was unlikely that there was not data with a 20 mile radius. We also realized that 20 miles was an acceptable margin for error.

Functional Requirement 1.5: Calculating Historical Data In our original document this requirement was actually sending a request for the data from the server. We swapped this requirement for calculating historic data because our client wanted to be able to see the projection vs the historical data and we are no longer making requests for the data as it is available locally.

Functional Requirement 1.6: Formatting the Data In our original document we were under the assumption that we would be sending a request for the data and processing the result. So this requirement changed from processing the data returned by our call to an API to formatting the data into a usable json object.

3 Design Document

3.1 Original Document

CS Capstone Software Design Document

June 12, 2018

Linking Seasonal Weather Data to AgBizClimate™

Prepared for

Oregon State University

Clark Seavert

Sean Hammond

Prepared by

Group26

Thomas Noelcke

Shane Barrantes

Shengpei Yuan

Revision History

Revision	Date	Author(s)	Description
1.0	1.12.17	Thomas Noelcke, Shane Barrantes, Shengpei Yuan	Preliminary Draft

Abstract

This design document will cover the proposed design of the AgBizClimate™project. We will first give a general introduction to the project. This section will provide some context for why we are doing this project and what this project hopes to accomplish. Next we will talk about Architecture design. This section will describe a high level structure for the project. After that we will discuss the data for the project and how it will be structured. Then we will discuss in detail the design of each component in the *AgBizClimate* system. We will then discus the

design of each view in the user interface. Finally, we will provide a requirements matrix which will show how each component fulfills the functional requirements outlined in the requirements document.

Contents

List of Figures

1 Introduction

1.1 Purpose

The purpose of this Software Design Description (SDD) is to describe the architecture and system design of the *AgBizClimate* project. This document will provide a high level design for the *AgBizClimate* short term climate tool. This document will also provide a detailed description of the design of the data for this project. We will also break down each component and discuss the design of each component in detail. After that we will discuss the design of each view in the user interface. Finally, we will have a requirements matrix. The requirements matrix will show how each component fulfills the functional requirements for this project. This document is intended for the project owners and software developers of the *AgBizClimate* system. This document is intended to be a guide for the implementation of the *AgBizClimate* short term climate tool.

1.2 Overview

Seasonal climate is one of the essential factors that affects agricultural production. As a module of *AgBiz Logic*, *AgBizClimate* delivers essential information about climate change to farmers, and help professionals to develop management pathways that best fit their operations under a changing climate. This project aims to link the crucial seasonal climate data from the Northwest Climate Toolbox database to *AgBiz Logic* so that it can provide changes in net returns of crop and livestock enterprises through powerful graphics and tables.

1.3 Scope

This project is a part of a much larger AgBiz Logic^{TM}program. However, the purpose of this project is to add a short term climate tool to the *AgBizClimate* module. This limits the scope of the project to the *AgBizClimate* Module. Additionally, we will only be adding the short term climate data tool as the long term climate data tool already exists.

Currently *AgBizClimate* has a long-term climate tool but no such tool exists for short term climate data. We will implement a tool to extract short-term climate data from the Northwest Climate Toolbox database, display it to the user and allow the user to adjust crop and livestock yields or quality of products sold and, production inputs. Moreover, a landing tool will be developed to allow users to switch between short-term seasonal tool and long-term climate data tool.

1.4 Definitions, Acronyms and Abbreviations

REST - Representational State Transfer, This is a type of architecture that manages the state of the program. This is especially popular in web development.
API- Application Programming Interface. This is a piece of software that allows a connection to another piece of software providing some sort of service.
NWCTB - Northwest Climate Toolbox. This is the database we will be connecting to that will provide the short term climate data we plan to use.
Climate Scenario - This is a theoretical calculation of yields, inputs and of the overall budget for one situation based on the climate data.
SQL Database - This is a relational database that allows for storing and accessing data.
NOSQL Database - This is a non-relational database that allows for data storage and data access.
UI - User Interface, This is a piece of software that allows a human to interact with the software. Often this is what the user sees while using software.

1.5 References

[1] C. F. Seavert, "Negotiating new lease arrangements with the transition to direct seed intensive cropping systems," 2017.

[2] S. Y. Thomas Noelcke, Shane Barrantes, "Problem statement," 2017.

2 System Overview

2.1 Product Functions

AgBizClimate is a web based decision tool that will allow users to gain specific insight on how localized climate data for the next seven months will affect their crop and livestock yields or quality of products sold and production inputs. The *AgBizClimate* tool will allow users to input their location (state, county) and a budget for the specific crop or livestock enterprise. *AgBizClimate* will select climate data for the next seven months for that location and provide graphical data showing temperature and precipitation. Users will then be able to change yields or quality of product sold by a percentage they think these factors will affect and modify production inputs. Finally the tool will calculate the net returns.

2.2 User Characteristics

AgBizClimate users can be split into two subgroups, agricultural producers and climate researchers. The first subgroup, agricultural users who use this product tend to be between fifty and sixty years old of mixed gender. Their educational background ranges from high school to the completion of college. The primary language this group uses is English, but there are some Spanish users as well. Most of the users in this group tend to have novice computational skills. The primary domain for these users is agricultural and business management. Most agricultural producers who use this product are motivated by the potential profit that the decision tool *AgBizClimate* could potentially offer. The second subgroup, climate researchers range from ages twenty to forty and are of mixed gender. The educational background for most climate researchers exceed the postgraduate level with their primary language being English. These users generally have advanced computational skills and are motivated by the easily accessible climate and weather data.

2.3 Constraints

There are several key constraints that this product has to work within. The first constraint is that we only have access to two data parameters from the North West Climate Tool box, precipitation and temperature. Secondly, we only have access to their data via the NWCTB API which could have additional restrictions such as limited usage per day, mislabeled data, or poor documentation. Thirdly, we dont have access to any of the hardware that *AgBizClimate* is exists on as it is being managed by a third party. This will prevent us from improving the hardware or cause roadblocks if their servers are having issues. Lastly, we are limited to using the languages Python and JavaScript since we are integrating our product into an already existing project.

2.4 Assumptions and Dependencies

We are assuming that the Northwest Climate Toolbox is a functional API that will allow us to pull location based temperature and precipitation data. This data will most likely come in the form of a text body of which we will then format into a JSON object and store in a MongoDB database for future use. Due to the fact that we are writing an addition to an existing project we do not need to interact with the user budgets as these have already been defined. This fact extends to the calculations portion of the *AgBizClimate* product. Our team will simply be accessing data via the NWCTB API, then format the data, store the data, and hand the data over to the tool while will provide

some additional front end support.

3 System Architecture

3.1 Architectural Design

Shown Below is the architectural design for the *AgBizClimate* project. This UML diagram shows the high level components of this application. This Diagram also shows how these components will interact.

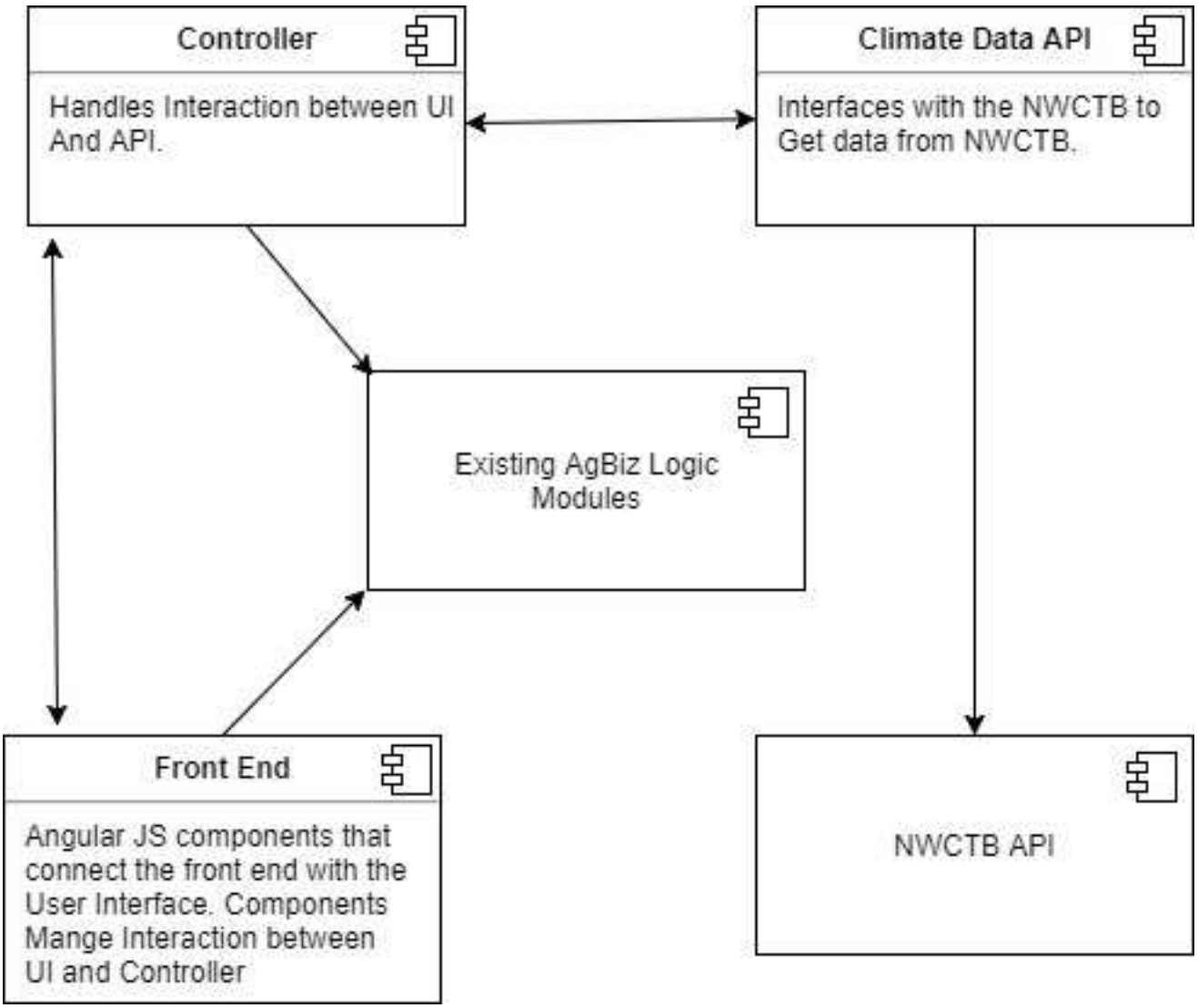


Figure 1: System Architecture Design for *AgBizClimate* project

3.2 Decomposition Description

The *AgBizClimate* project can broken down into six components. The six components are the Backend Controller, The Front End Controller, UI, Existing *AgBiz Logic* Modules, Climate Data API, and the NWCTB API. In this section we will describe each models function and how it interacts with the other modules.

3.2.1 Backend Controller

This component is responsible for connecting the dots between the rest of the components. Generally, the backend controller will handle incoming requests from the front end controller and return the requested content. This component will also interface with the existing *AgBiz Logic* modules and Climate Data API so it can provide all the requested information.

3.2.2 Front End Controller

The Front End Controller will act as an interface between the UI, the existing *AgBiz Locic* modules and the Backend Controller. This component will handle UI action made by the user and will use those to make requests to the Backend and existing *AgBiz Logic* Modules. This component will then take the result of these requests and display the relevant information to the user. This component will also handle user inputs such as a button push or clicking on a drop down menu. This component will take these actions and modify the UI to reflect the actions the user preformed.

3.2.3 UI

This is the portion of the application that the user will see and interact with. The primary responsibility of this component is to display information to the user. This component will also be responsible for interacting with the front end Controller to ensure that user actions so the program responds correctly to user action.

3.2.4 Existing *AgBiz Logic* modules

This is not a component but rather a collection of components that already exists as part of the *AgBiz logic* system. We will use these components to preform a variate of actions including, retrieving budget data, managing user information, making modifications to budget data and saving budget data back to the database. We will interface with these component from the Front End Controller to handle budget data. We will also interface with these components from the back end to handle user data.

3.2.5 Climate Data API

The Climate Data API component will interface withe the NWCTB to provide long term forecast data. This component will take requests, with location data, from the Backend controller and will respond with the formatted data from the NWCTB. To do we will interface with the NWCTB API to retrieve the data. Then we will take the data from the NWCTB, parse it into JSON, apply some formatting and pass it back to the Backend. For the purposes of this project this component is only going to interface with the Backend Controller. However in the future this API maybe used by other sections of the application as well.

3.2.6 NWCTB API

The NWCTB API will be our data source for this project. This component will provide the climate data by interfacing with the back end controller. Currently we are not sure how we will interface with the NWCTB as the NWCTB has not responded to our requests for API access.

3.3 Design Rationale

We've chosen to design this system this way in part because of the nature of our system. We need a front end controller to handle the clients interaction with the server because this is a web development project and the front end will be separate from the server. The front end controller will facilitate the communication between the client and the server.
We also chose to use a Backend Controller so we can facilitate the communication between the front end controller and the various components on the backend. This makes the application easier to build, test and maintain. This also allows for one line of communication between the backend and the front end. This is necessary to keep the interactions between the backend code and front end code simple. This allows for large changes to be made to both the front end and the backend with out causing them to impact each other.
We also chose to create the Climate Data API as its own service. We chose to do this because it's easier to test and then the Climate Data API can be reused in future projects and with the other components. If we had built the Climate Data API into the backend controller this would not have been possible.
The NWCTB and Existing *AgBiz Logic* components have already been implemented out side the scope of our project. However, we are still planning to use them in our project. This is why we have created these modules in our design because we will be using them as part of the design of our system but do not want to tightly couple our project with these existing components.
More generally the components in our system use the REST API architecture type. We chose to do this because it allows for flexible reusable modules. The REST API Architecture also allows us to break our application into independent modules that are easier to develop, test and maintain. This division of our application also makes it more scalable allowing our system to keep up with future demand.

4 Data Design

4.1 Data Description

In this section we will discuss the design of the data required for this system. The data needed to implement this system includes the user data, climate scenarios, Budget Data and the climate data. It should be noted that this project adds the climate data to the system. User data, climate scenarios and budget data have already been implemented as part of the existing *AgBiz Logic* system. However, since we will be using this data as part of our project I've included their design in this section.
Shown below is the design for the data we will use in this program. In this UML diagram are all the various entities required by this system. Additionally we also show the relationship between different entities. Shown below are the UML diagrams for User Data, Climate Scenarios, Budget Data, Climate Data and Related Entities.

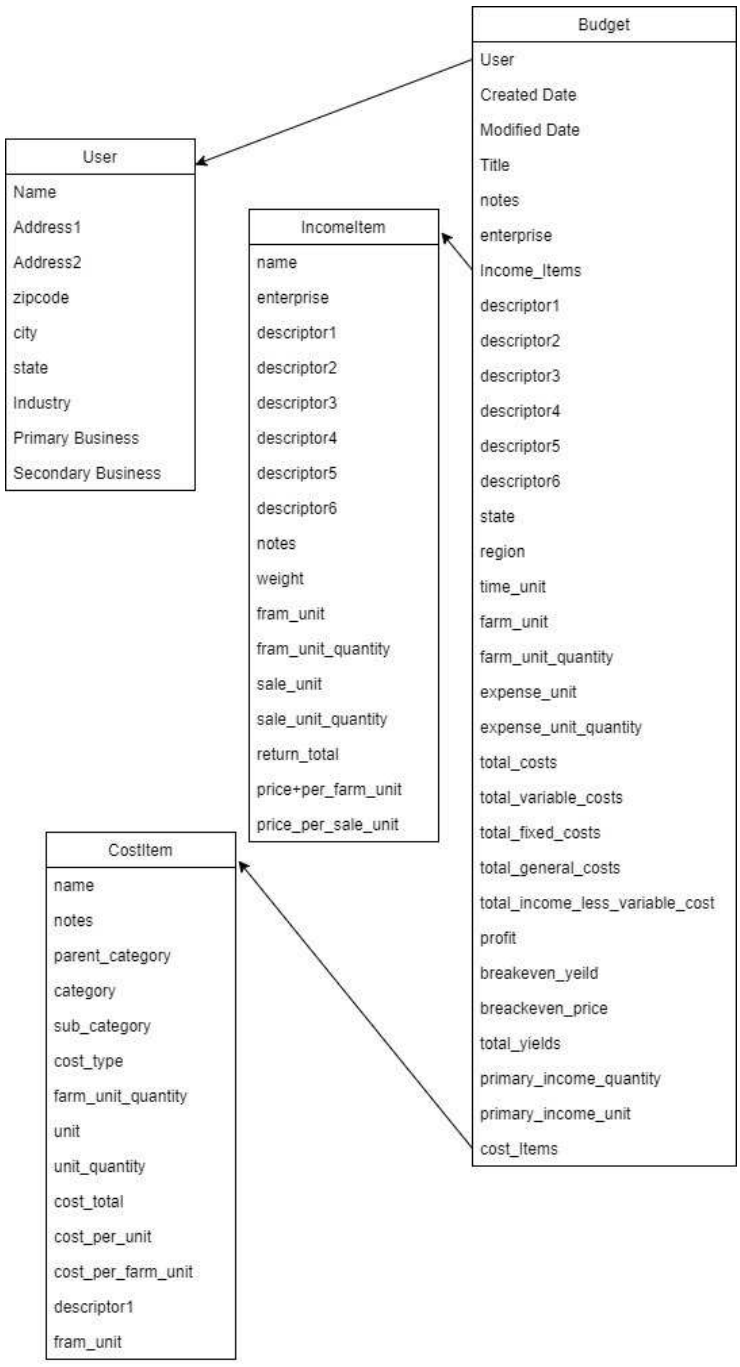


Figure 2: User Data Currently Implemented in the *AgBiz Logic* project

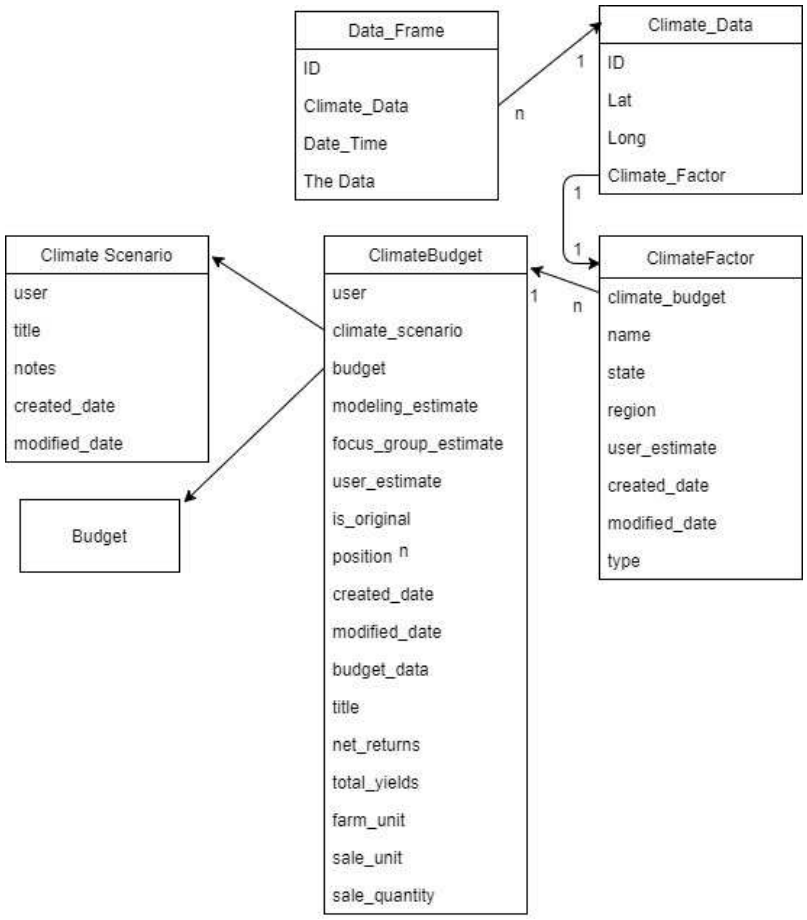


Figure 3: Design For the Climate Data for the *AgBizClimate* project

4.2 Data Dictionary

In this section we will describe each piece of data and what it represents. We will also discuss possible values for each piece of data.

4.2.1 User Data

In this section we will define the entities and fields that relate to user data and user budgets.

4.2.2 User

This entity represents a user along with the data we will need for each user.
Name - This is a string that stores the user name of a user.

Address 1 - This stores the first line of a users address.

Address 2 - This stores the second line of a users address.

zip code - This stores the zip code for the address the user entered.

city - This stores the name of the city for the address the user entered.

state - This stores the name of the state that the user entered. The state must be a valid US state as climate data is not available for out side the US.

Industry - This stores what industry the user is involved in. Currently the user may choose from two options Agriculture or Non-Agriculture.

Primary Business - This stores what sort of business a user is employed in. For instance if a user were a farmer they would select Producer.

Secondary Business - This is an optional parameter that stores what if any other business that the user may engage in. For instance if a user is a producer but also packages their product them selves they would select packager for this option.

4.2.3 Budget

This entity represents a budget and associated data. It is important to note that Budgets are related to users by a many to many relationship. It is also important to note that this entity is related to the costItem entity and the IncomeItem entity. This relation ship is a many one to many relationship.
User - This is a reference to the User entity because a budget should be associated with a user.

Created Date - This stores the date that this budget was created.

Modified Date - Keeps track of the last time the budget was modified.

Title - The user entered title for the budget. This helps users keep their budgets organized.

notes - Users entered notes about a budget. This helps a user keep track of their budgets.

enterprise - The enterprise type that this budget represents. If it is a conventional crop we would say conventional. If it were an Organic type crop this would be Organic.

descriptor1 through descriptor6 - Generic descriptor for the budget. May be used might not be used. This item will hold descriptions relevant to the budget we are storing.

state - Stores the state that this budget is intended for.

region - Stores what region in the state that the budget is intended for.

time_unit - Stores how the user would like to measure time for that budget. Some budgets may be measured in years but some may be measured in months or weeks.

farm_unit - This item stores how we plan to measure how much land we have to work with. Most farms will choose to use acres. However for some crops its more useful to use a different type of measurement.

farm_unit_quantity - This describes how large our farm is based on the farm_unit.

expense_unit - This item stores the unit of measurement for an expense. This is similar to a farm_unit.

expense_unit_quantity This item stores how many of the expense_units we have in our budget.

total_cost Total cost represents the total cost of the crop we are using this budget for.

total_variable_costs This represents the total variable cost for this budget. Variable costs are cost that can change and depend on other variables such as fertilizer use.

total_fixed_costs - Represents the the total fixed costs for this budget. Fixed costs are costs that will be the same regardless of other variables such as land lease.

total_general_costs - This is the total general cost associated with this crop. This represents how much it will cost to produce this crop.

total_income_less_variable_cost - This is total income with out subtracting the total costs.

profit - This is how much money we will make after we adjust for the total cost of producing the crop.

breakeven_yield - This represents the quantity of a crop needed to off set the total cost.

breakeven_price - This represents the price we will have to sell our crop at given that the quantity is the breakeven_yield to off set the total cost.

total_yields This represents how many units of a crop we produced.

primary_income_quantity - This represents the quantity of the most profitable crop in my budget.

primary_income_unit - This represents the unit of measure for the primary_income_quantity.

4.2.4 Cost Item

This entity represents an item that is a cost in regards to ranching or farming. It should be noted that this entity is related to a budget via a many to one relationship.

name - This represents the name of the cost. For instance if the cost is fertilizer the name of this cost item would be fertilizer.

notes - This is user input data that allows the user to make notes about a cost item.

parent_budget - This is a reference to the budget that this cost item is associated with.

parent_category - This a broad category that this item belongs to. For instance if this item were round up it would belong to the broader category of pesticides.

category - This is the category the item falls into. For instance if the cost is paying one of my farm workers this would be labor.

sub_category - This allows for even farther categorization of cost items.

cost_type - This represents what kind of cost this item is. Mainly weather its a fixed or variable type cost.

farm_unit_quantity - This stores how many farm units we will need to apply this cost too when making calculations.

unit - This represents the unit that this cost item we be measured in.

unit_quantity - This is the number of units of this cost item that we will need.

cost_total - This is how much the this cost item will cost based on the number units needed and the cost per unit.

cost_per_unit - This represents how much each unit of this cost item will cost.

cost_per_farm_unit this represents how much it will cost to per farm_unit for this cost item.

4.2.5 Income Item

This entity represents an item that provides income in to the budget such as a crop or livestock. This item is related to the budget entity by a many to one relationship.

Name - This represents the name of the Income item. For instance if the income item is corn it would be name corn.

enterprise - he enterprise type that this budget represents. If it is a conventional crop we would say conventional. If it were an Organic type crop this would be Organic.

descriptor1 descriptor6 - Generic descriptor for the Income Item. May be used might not be used. This item will hold descriptions relevant to the Income Item we are storing.

notes - This will store notes about the Income Item to help the user keep income items organized.

weight -

farm_unit - This stores the unit that we will measure how much space we have to grow this Income item.

farm_unit_quantity This represents the number of farm_units we have to produce this income item.

sale_unit This specifies how we plan to measure the quantity of this income item.

sale_unit_quantity - This specifies how many sale units of this cost item we have to sell.

return_total - This is the total amount we would get from selling this income item.

price_per_farm_unit - This is how much money we sell our income item per farm unit we have produced.

price_per_sale_unit - This is the price we can sell this income item for per sale unit.

4.3 Climate Data

In this section we will discuss and define the entities that are required to represent climate data.

4.3.1 ClimateScenario

This entity represents a Climate Scenario. This entity keeps track of important user information in regards to Climate Simulations. It is important to note that this entity is related to the Climate Budget entity via a one to many relationship. This Entity is also related to the user entity via a many to one relationship.

user - This is a reference to the user who created the climate scenario.

Title - This a user entered title that represents the name for the scenario. This allows the user to keep their scenarios organized.

notes - This is user entered data about this climate scenario. This allows user to better track what each scenario is for.

created_date - This keeps track of the date that the climate scenario was created.

modified_date - This keeps track of the date that the climate scenario was lasted modified.

4.3.2 ClimateBudget

This entity represents one Budget simulation in a climate scenario. This entity is related to the ClimateScenario entity by a many to one relationship. This entity is also related to the user entity via a many to one relationship. This entity is also related to the budget entity via a many to one relationship.

user - This is a reference to the user entity. This represents the user who created this ClimateBudget.

climate_scenario - This is reference to the ClimateScenario entity this represents the climate scenario that this climate budget belongs with.

budget - This is a reference to the budget entity. This budget the budget we are considering for this climate budget.

modeling_estimate - This represents the estimation that the model produces for how much the climate prediction will effect the climate.

focus_group_estimate - This is an estimation of how much a model will effect a budget based on a focus group.

user_estimate - This is the estimation that the user provides for how much the climate factors will effect the budget being considered in this climate budget.

is_original - This is a Boolean flag that indicates weather or not this is an original climate budget or a university budget.

position - This indicates the geographical location that this climate budget is trying to display.

created_date - This tracks the date that the user created this climate budget.

4.3.3 ClimateFactor

This entity represents one climate factor that may effect the climate budget. For example we may consider the number of freezing nights in one year. This item is related to the climate budget entity via a many to one relationship.

climate_budget - This is reference to the ClimateBudget entity. This represents the budget we are considering for this climate factor.

name - This is the name of the climate factor. For instance average temperature.

state - This is the state for which this climate factor is being considered.

region - This is the region for which this climate factor is being considered.

user_estimate - This represents the amount the user estimates that this climate factor will impact total crop yield.

created_date - This keeps track of when this climate factor was created by the user.

modified_date This keeps track of when this climate factor was last modified by the user.

type this keeps track of what type of climate factor this climate factor represents. Currently we have two types long term and short term.

4.3.4 ClimateData

This entity represents Climate data that will be used to determine how climate change may effect a budget. This entity is related to the ClimateFactor entity in a one to one relationship.

ID - this is a unique ID by which we can identify this set of climate data.

Lat - This stores the Latitude of the climate data represented by this climate data entity.

Long - This stores the Longitude of the climate data represented by this climate data entity.

Climate_Factor - This is a reference to the ClimateFactor entity.

4.3.5 DataFrame

This entity represents one single point in time of a climate data model run. A collection of these DataFrames can represent the data for the whole model run. This entity is related to the ClimateData entity in a many to one relationship.

ID - This is a unique identifier we can use to identify this data frame.

Climate_data - This is a reference to the ClimateData entity.

DateTime - This keeps track of the date and time that this data point occurs at in time.

The Data - This is simply a place holder for the data we will need to store for each data frame. We can't currently know how this section of this entity will need to be formatted as we don't know what the data we will get from the NWCTB will look like.

5 Component Design

5.1 Front End Controller

The front end controller is essential as it is what passes the user input to the climate data API and receives valuable data from the backend to display too the user. This section will discuss individual angular components and their functions. On every page there will a toolbar that provides a link to the AgBizClimate landing page via clicking on the *AgBiz Logic* and a drop down box that provides user actions on user clicks.

5.1.1 Angular Components Design

In this section we will look at each page of AgBizClimate and view the angular components and their function. Shown below in figure 4 is the overall design for the angular components for each page. It should be noted that each box represents one page in the UI. The first section above the second bar represents the various components in the UI. The second section represents the various function that will be needed to respond to user input on the UI. For each each page we will discuss and describe the function of each UI component. We will also discuss each function in each component along with its intended purpose.

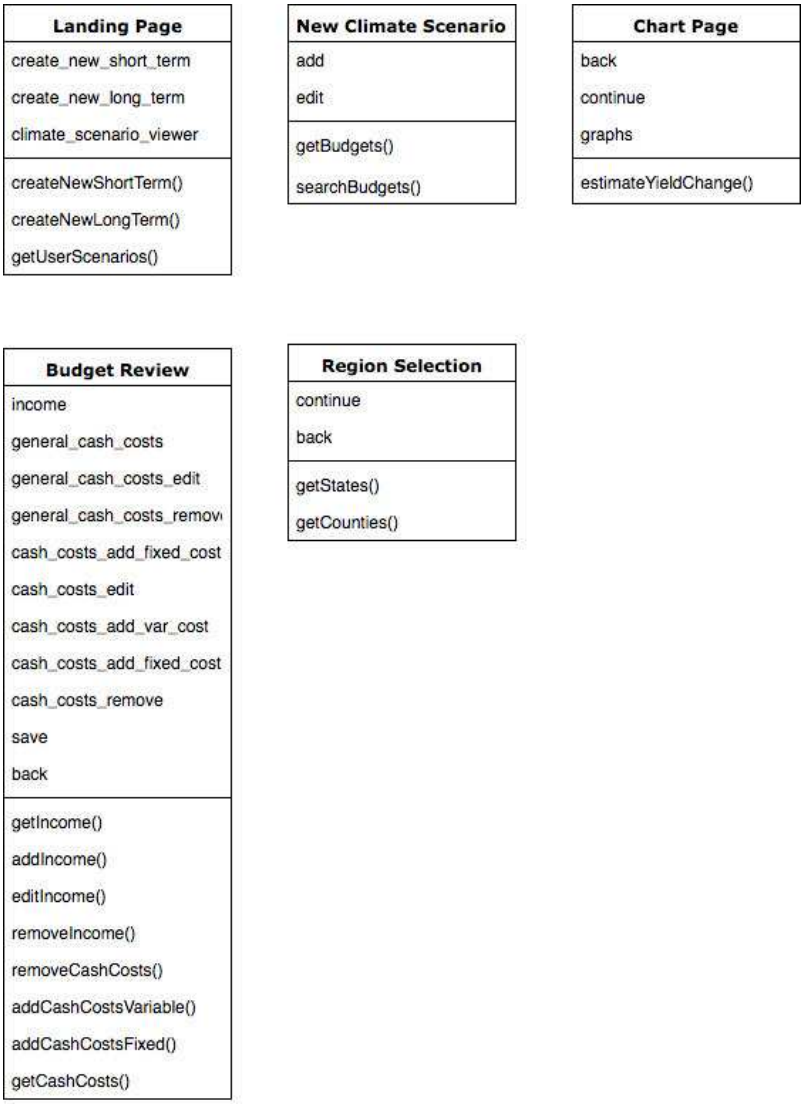


Figure 4: Overall structure for Controller and View

5.1.2 Landing Page

The landing page has three angular components: create_new_short_term, create_new_long_term, and climate_scenario_viewer. The create_new_short_term and create_new_long_term are clickable action components that utuilize the createNewShortTerm() and createNewLongTerm() functions to make calls to the back-end and begin the climate creation scenario processes. The climate_scenario_viewer component is used to make a call to the back-end using getUserScenarios() which grabs all scenarios tied to that user and presents them in a table.

5.1.3 New Climate Scenario

The new climate scenario page has two angular components: add, and edit. The add component allows users to add a new budget to their scenario from a list of provided budgets and the edit component allows users to alter budgets they already have attached to their accounts. There are two primary function calls associated to this page. The first function is getBudgets() which makes a call to the back-end to grab all available budgets for the user to select from. The second function is searchBudgets() which allows users to search for a specific budget from the list of all budgets.

5.1.4 Region Selection

The region selection page has two angular components: back, and continue. These components are clickable and cause redirects to the previous new climate scenario page or the chart page. The two functions included in this page are getStates(), and getCounties() which make calls to the back-end to grab a list of all available states and counties that AgBizClimate has seasonal weather data for.

5.1.5 Chart Page

The chart page has three angular components: back, continue, and grab. Back, and continue clickable event buttons that are used to return to the previous page or move on to the next page. The graph component is a static component that displays the seasonal precipitation and temperature for the location they selected. This page also makes use of the function estimateYieldChange() which takes user input and makes a call to the back-end to use an AgBizLogic formula to make a more accurate budget review for the user.

5.1.6 Budget Review Page

The budget review page has the most angular components for viewing and manipulating data. There are also clickable navigation components back, and continue which allow users to go back to the previous chart page or continue on and save their current climate scenario. Next, there are a series of components that allows actions to be taken on the income and cash_cost data objects. New income objects can be added via the income_add component, edited via the income_edit component, or removed from the list via the income_remove component. cash_costs can be edited via the cash_costs_edit component, assigned additional variable costs via the cash_costs_add_var_cost, assigned additional fixed costs via the cash_costs_add_fixed_cost component, or removed via the cash_costs_remove component. Finally, There are a series of functions that these components use to make calls to the back-end to retrieve and manipulate data. The getIncome() and getCashCosts() functions retrieve the income and cash_costs associated with the budget to be displayed in the view. The addIncome(), addCashCostsFixed(), and addCashCostsVariable() functions are all used to add an aditional data object to the data objects or data object lists. The removeIncome() and removeCashCosts() functions are used to remove specific items from the list. Lastly, the editIncome(), and editCashCosts() functions are used to manipulate selected income and cast_costs data objects.

5.2 Controller Design

5.2.1 Overview

As an interface between Model and View, the Controller plays a very important role in the entire application architecture. First, it controls the way the application responds to the requests that result from the end-users' operations on UI. Secondly, it maintains the underlying relations between user interface (View) and data (Model). Thirdly, it accepts input from end-users, makes some necessary transformations, and then applies the final results on the data (Model). In this way, it controls the computation of the data (Models). At the same time, it could also reflect the variations on data (Model) back into UI (Views).

5.2.2 Overall Design

Overall, the controller tries to decompose the coupling between Model and View. However, as we would mainly employ the python Django framework for the whole AgBizClimate project. The Controller component acts as intermediary agent between front-end UI and back-end Climate APIs. However, it should be noted that the Django framework does not do well at separating the View and Controller clearly. As a result the view and controller are tightly interwoven. Therefore, it is better to deliberately define a middle layer between view and controllers. In this project, we could adapt some design pattern techniques called Mediator and Event dispatch mechanism. The figure

5 depicts the overall structure for Controller and View. We can see that the user action is sent to Controller from View through a Mediator to Controller. And the final effects of Controller on View are sent back to View through Mediator. And the user actions are transferred by event dispatching mechanism. Therefore, the complex dependency relations between Controller and View are largely broken.

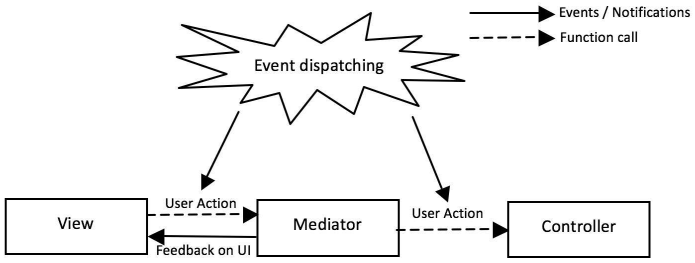


Figure 5: Overall structure for Controller and View

According to the data design, there are four kinds of general data types: user data, climate scenarios, budget data and the climate data. Consequently, we will need four general controllers to handle each data item. However, it is not possible or reasonable to deal with all of the business logic within the four general controllers. We plan to define some specific controllers to deal with more concrete tasks and organize them hierarchically. That is, underneath the four high level general controllers there are more low level small detailed controllers for various businesses / events. And the events are transferred gradually from lower levels to higher levels. In figure 6 presents the top Controller design of the AgBizClimate project. Basically there are two general controller types: Frontend Controller and Backend Controller, which mainly handles the operations/events from UI and data models, respectively. Additionally, much of the work are actually finished by the four specific controller under Backend Controller: Climate Scenario Controller, Climate Data Controller, Climate Factor Controller and Climate Budget Controller. Also, it should be noted that the Backend Controller will call the existing AgBiz Logic Model for certain tasks.

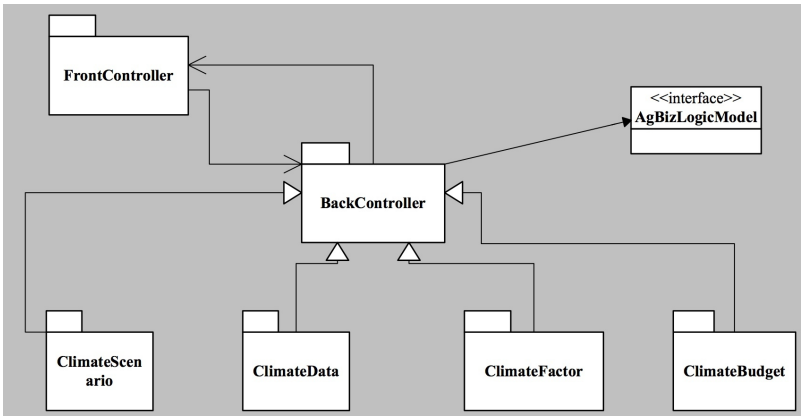


Figure 6: Top controller design diagram

5.3 API Design

5.3.1 Overview

In this section we will discuss the overall design of the API that will interface with the NWCTB. This API needs to get the data from the NWCTB, format the data, and send it to the client. Currently, there is a lot of uncertainty around the design of this API because we do not know what sort of API access that we will be given from the NWCTB. We are trying to contact the NWCTB development team regarding our API access but the NWCTB hasn't been very responsive. Because we still don't have NWCTB API access yet and have no date when this might

be accomplished, we will discuss several possible options that do not require NWCTB API access along with one design option that includes NWCTB API access.

5.3.2 Diagrams

Shown below are two diagrams. The first diagram shows a UML design of this module shown in Figure 7. Including the route for the API, the functions public and private and the data models used as part of this module. Also shown below in Figure 8 is how a typical transaction will work with the Climate Data API.

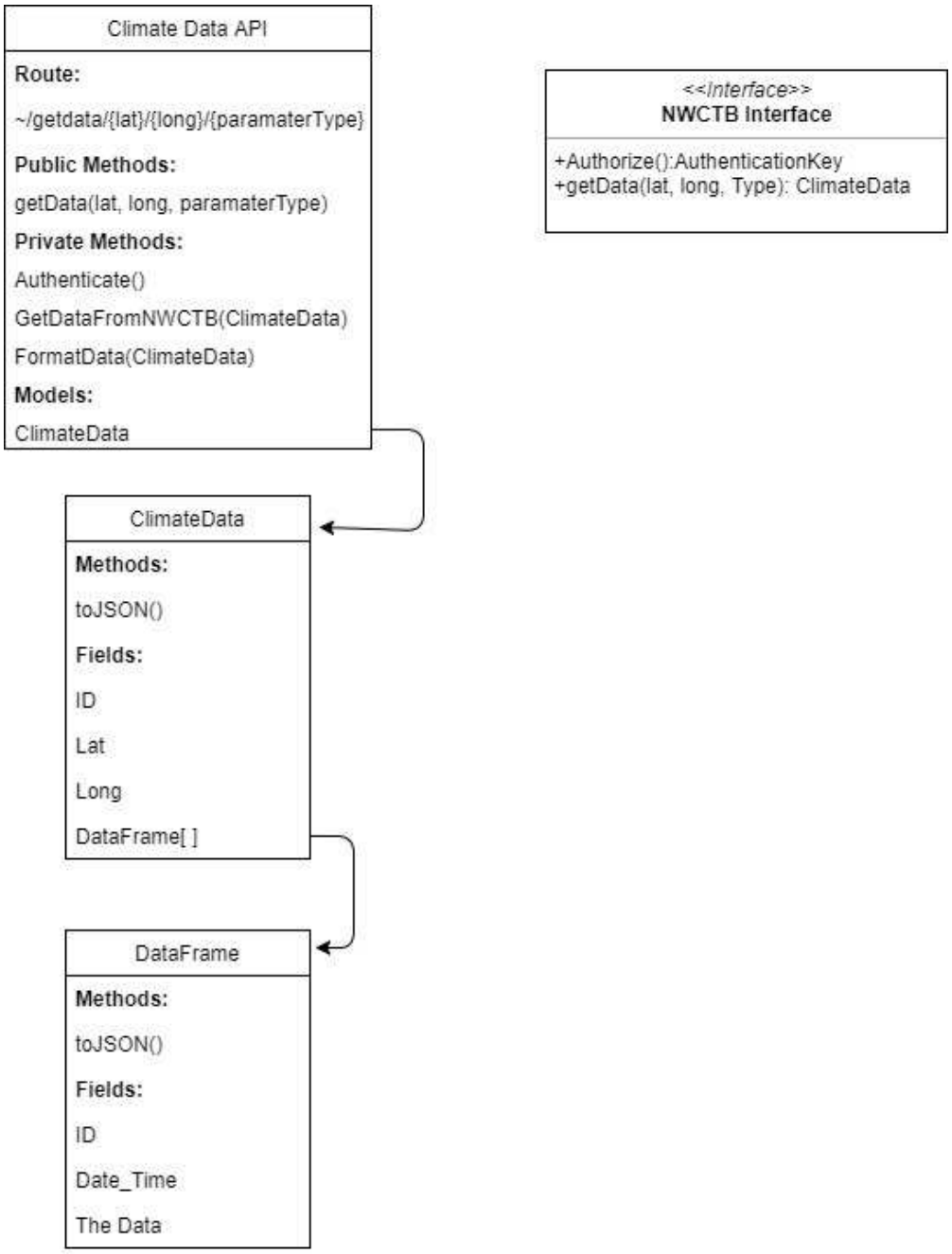


Figure 7: Design of Climate Data API and associated models

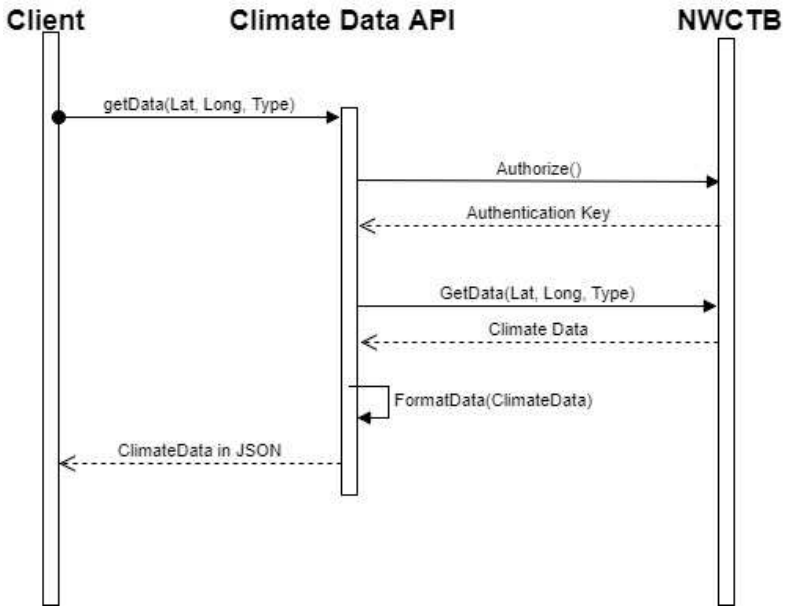


Figure 8: Typical Climate Data API Transaction

5.3.3 Overall Design

The climate data API can be divided into three different parts. The API Controller it self, the Data that the controller will be using to represent the climate data and the NWCTB interface. The following paragraphs will briefly describe each of these components.

5.3.4 Function Design

In this section we will define and describe each function in the Climate API design. We will also discuss what each function will take as input and what each function will return.

Public Methods

GetData(lat, long, parameterType) - This function call takes three paramaters as input. latitude, longitude and parameter type. The parameter type is the type of data that the client is requesting either precipitation or temperature. This call will then return the requested climate data as a JSON object to the client.

RouteCall: " /getdata/{lat}/{long}/{paramaterType}" - This route takes three parameters via the URL as input. These parameters are the latitude, longitude and the ParameterType. The parameter type is the type of data the user requested. In this case it's either precipitation or temperature. The data will then be formatted as a JSON object and will be returned to the calling client.

Private Methods

Authenticate() - This function will preform a hand shake with the NWCTB API to let the NWCTB API know who we are. This function will make a call to the Authorize function in the interface of the NWCTB API. This function will then get the authentication key for later use when we go to make requests to the NWCTB API.

GetDataFromNWCTB(ClimateData) - This function will set up and make the request to the NWCTB API. This function will take a ClimateData obejct as a parameter preloaded with the ID, Lat, and Long fields completed. This function will return the climateData object after initializing the DataFrame collection with the data received from the NWCTB.

FormatData(ClimateData) - This function will take a climate data object and will transform the object into a JSON object and return it.

5.3.5 Model Design

In this section we will define and discuss the models we plan to use in the Climate Data API. For each model we will define and describe each data member and function.

5.3.6 NWCTB Interface

We will not be talking about how the NWCTB Interface is designed as for the purposes of this project this is a black box that returns the climate data we requested. However we will discuss the method calls we plan to use for the NWCTB API.

It should also be noted that this sections is what we are guessing the API Access should look like. Currently we are unsure how we will be accessing the API and what the calls will look like. But for the purposes of the design of this system we have taken our best guess and what it might look like.

Methods

Authroize() - This function takes no parameters and preforms a handshake between our application and the NWCTB application. This function call will return an authentication key that we can use later to make function calls to the NWCTB API.

getData(lat, long, Type) - This call takes latitude, longitude and the data type we are requesting. The data type in this case will either be precipitation or temperature. This function will then return the data we requested or an error if the request we made was unauthorized or was a bad request.

5.3.7 Data Design

In this section we will discuss the design of the models that we will use for the Climate Data API. For each model we will define and describe the fields and methods.

ClimateData This model represents the climate data for one whole climate simulation.

Fields ID - This is an ID that uniquely identifies this ClimateData object. This will be used to find specific model runs. Lat - This represents that latitude at which this climate situation is located.
Long - This represents the longitude at which this climate situation is located.
DataFrame[] - This is a collection of data frames that make up the climate data for this climate simulation.

Methods toJSON() - This method takes the ClimateData object and turns it into the equivalent JSON representation.

DataFrame This model represents one frame of data for a climate simulation. A data frame represents one moment in time in a climate simulation. A collection of these data frames make up a climate simulation.

Fields ID - This is a Unique identifier that will allow us to identify the order that the data frames need to go in.
Date_time - This is the date time that this data frame occurs on in time.
The Data - This is a place holder for the data. We currently do not have API access to the NWCTB. As a result we are not sure how the data will be formatted. Once we have the data from the northwest climate tool box we will able replace this place holder with the actual data for each frame.

Methods ToJSON() - This produces the equivalent JSON representation of the data frame.

5.3.8 Possible Alternative to the NWCTB

In this section we will discuss other possible solutions for getting our climate data other than using the NWCTB. We are discussing this because it is possible that we may not get API access to the NWCTB or that we will get access after this project is completed. Because there is some uncertainty regarding the API access we have come up with a few different options if we are unable to get the necessary API Access.

Automate Downloading the Data
One possible solution to getting the data without downloading the data is to write a script to download the data from the website. The data is available for download on their website and It wouldn't be too difficult to create a scrip that goes to their website and download the data. There are some potential issues we may run it using this approach. Firstly, they may throttle our speed if we try to download to many data entries at once or to often. This may happen many times as a person goes through climate scenarios for different crops. One way to over come this problem would be to cache the data in a database locally. This would allow us to only ever grab a data set for a location once. However, with this approach we would need to find a way to ensure that this data is the most updated data. This is the preferred alternative if we do not end up getting the NWCTB API access.

Build a New Service From the Ground Up
Another possible solution would be to build a new service that went out to the NOAA got the different climate predictions and averaged together the result. The model data used by the NWCTB is public record available to the public for download. It would be possible to create a new service that did essentially the same thing as the NWCTB but without the user interface. There are a few nice things about this solution in that it would give us control over the data from start to finish and would allow us to make calls to an internal service rather than an external one. However, this would have serious impacts on our development time. This would also be a rather complex problem that would require a lot of research. Preferably we will not use this option.

Find A New Climate Data API
Finally, Another solution would be to find a different API for the climate data we need. Currently we are unaware if another Climate Data API exists that would suite the needs of our application. However, if we were to find one that provided the data we need changing the design of our application to accommodate the new API probably wouldn't cause big issues. Assuming we can find another data source this approach would be ideal.

5.4 Testing Design

5.4.1 Front End Testing

We plan to test the front end using a Angular testing framework. Using this testing frame work we will test that every possible user interaction produces some sort of UI action. Essentially the idea is that we ant to test that our UI is responsive for every possible UI action.

5.4.2 Controller testing

Controllers are the workhorse of MVC. And the tests of them would to some extent guarantee the correctness of the business logic of the application. Basically, most controllers either render a view or handle form submissions. Thus the test of controller should mainly focus on the concrete user operations or response but not the entire framework or functionalists of the application. And these tests would be done in the phase of unit test. Generally, each test for controller may consist of two steps.
Firstly, one launches a request to the controller method to be tested. This is mainly done by user inputs from the view. In practice, the mocking actions from testing frameworks would replace human operations to do this.
Secondly, one verifies that expected response is received or certain effect has taken place. This step could also be done easily by auto testing frameworks after one has define appropriate test cases.

5.4.3 API Testing

For the API we will provide testing for every possible API route. We will also test that erogenous resumes result in correct response. Beyond that we will also test each function in the API. For each function we will test a variety of different inputs for each function including valid and invalid inputs. We will then ensure that the correct result is returned for valid requests. For invalid request we will ensure that the correct error message is returned.

6 User Interface Design

6.1 Overview of User Interface

The user interface is an essential part of any web application as it is the layer between the user and what they want and need out of an application. In this section we will be discussing the User Interface. Specifically, we will be detailing the specific UI elements that make up each page.

6.2 Screen Images

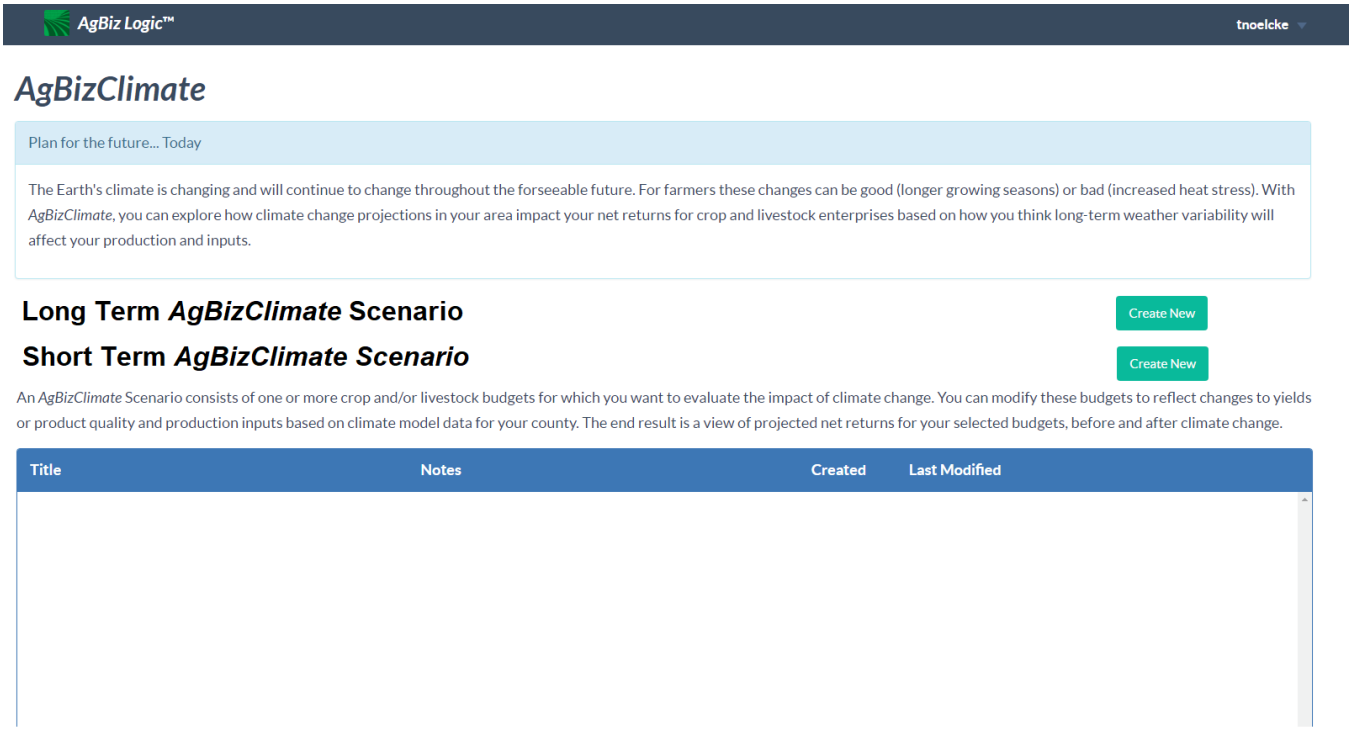


Figure 9: Landing Page

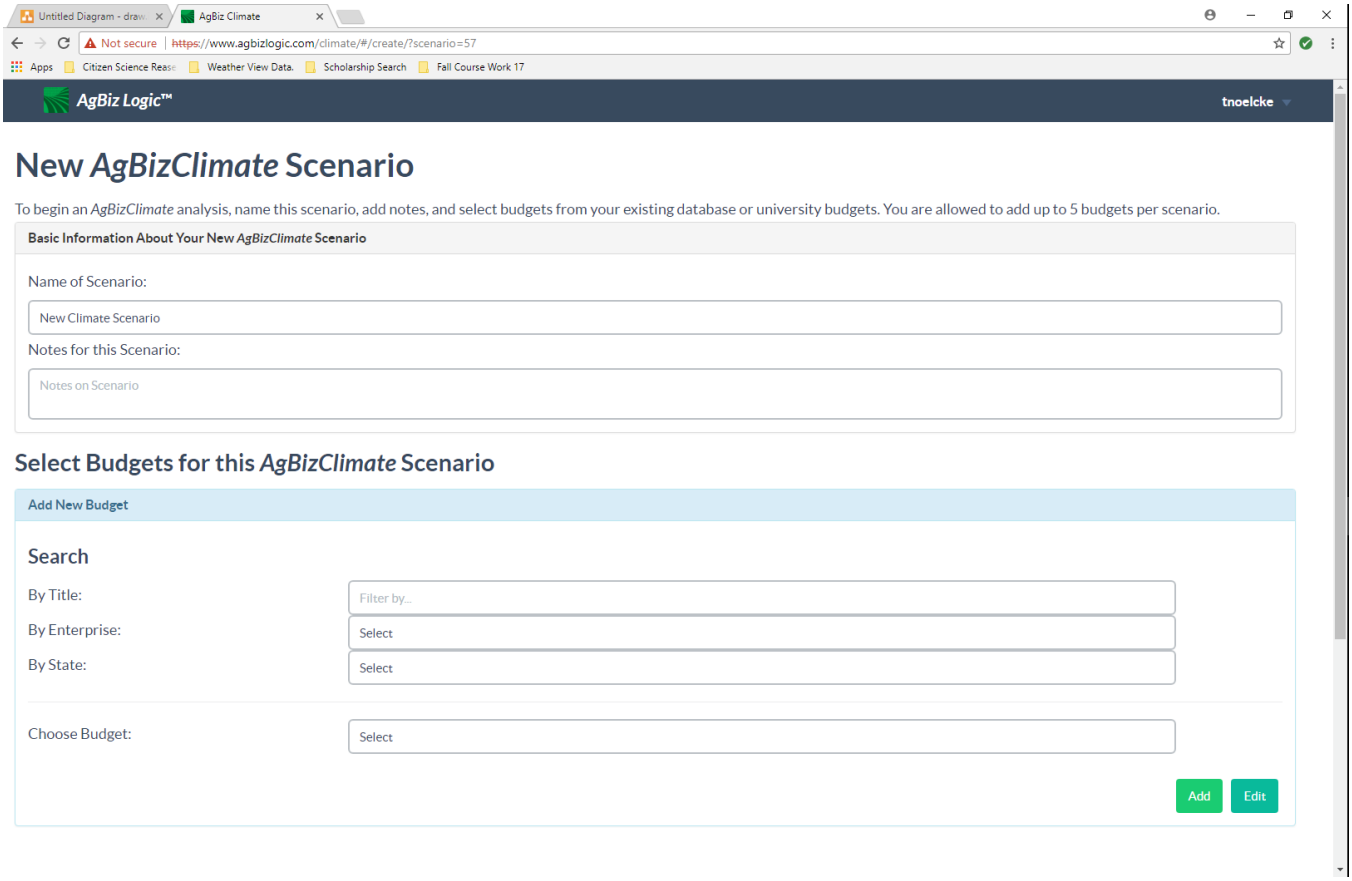


Figure 10: Allows user to select budgets and make notes

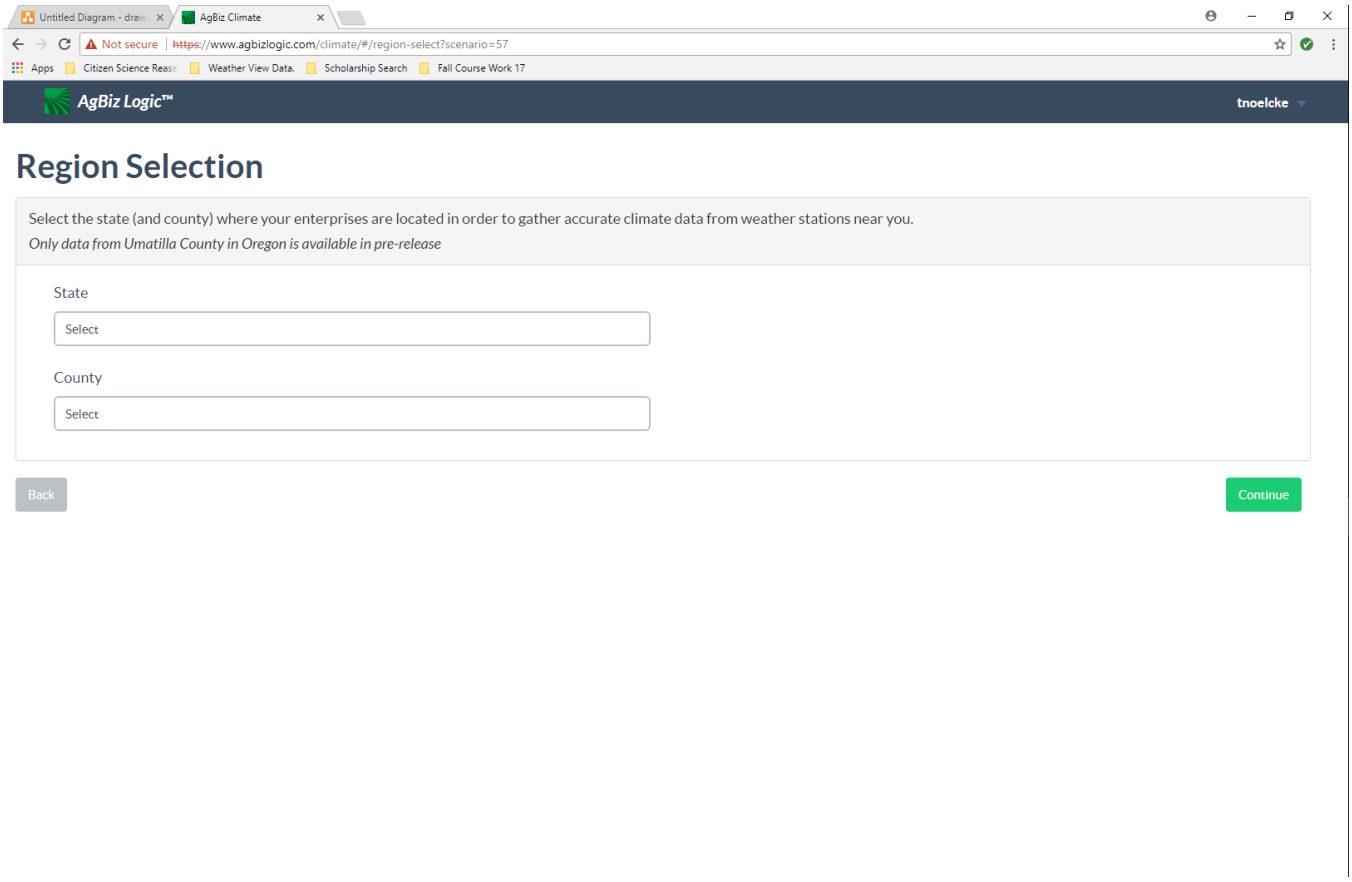


Figure 11: Allows User to select location

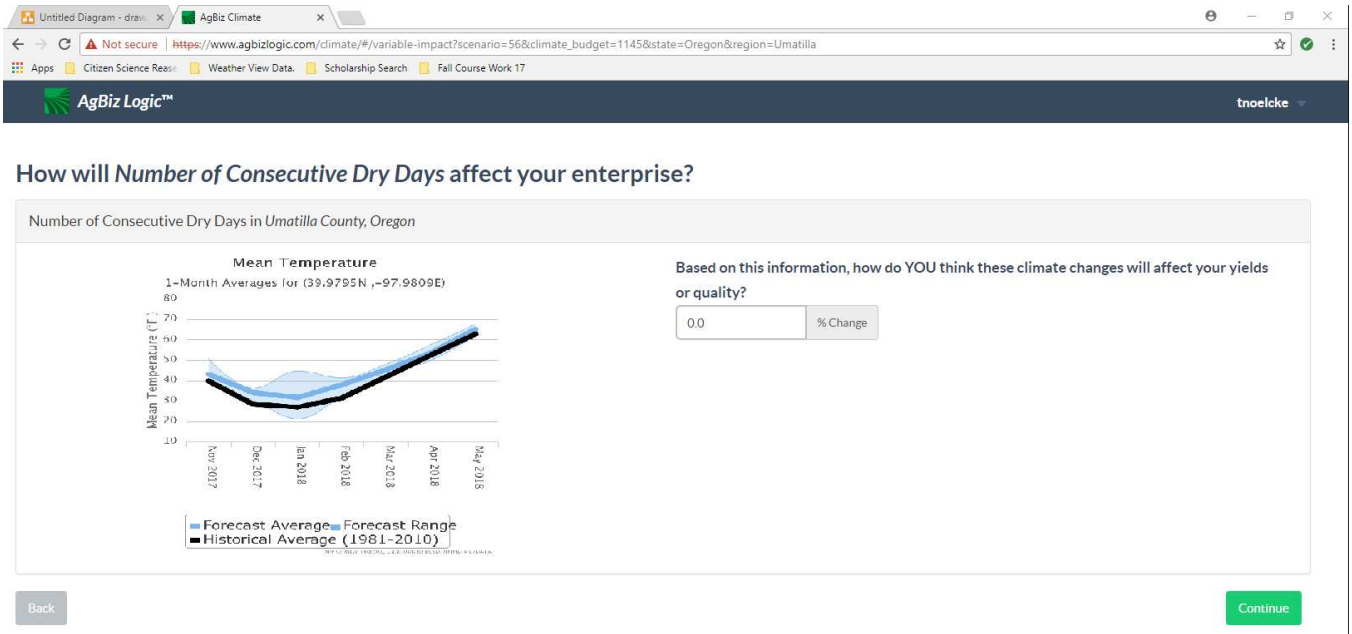


Figure 12: Displays the Data in a plot

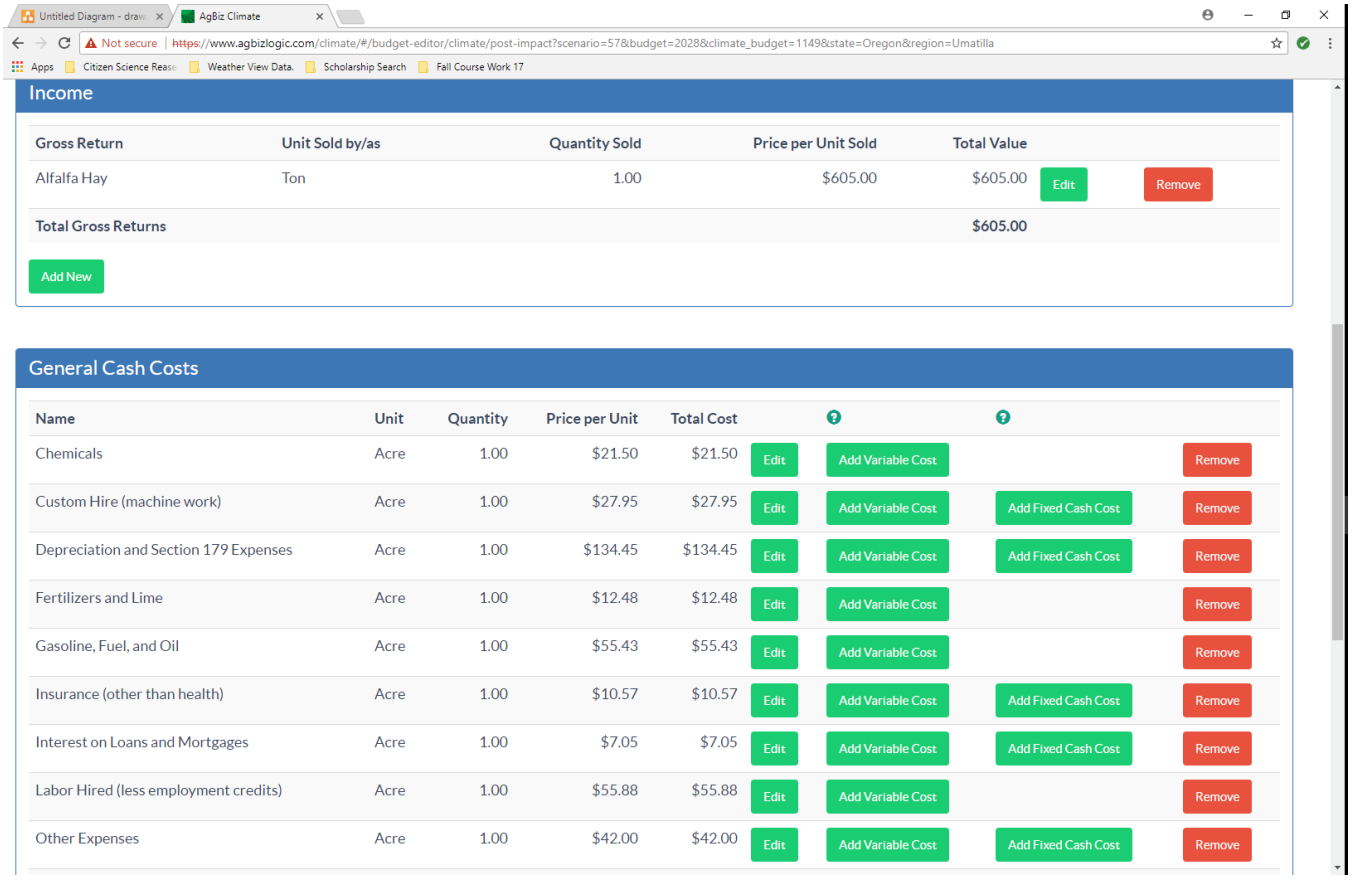


Figure 13: Allows user to view their budget

6.3 Screen Objects and Actions

6.3.1 Landing Page

The Landing page consists of two primary parts: the information jumbotron, new scenario creation, and a scenario listings. The information jumbotron is a static and gives the mission statement of the AgBizClimate product. The new scenario creation selection allows the user to click on one of two boxes specifying if they would like to create a new short or long term climate scenario. Finally, the scenario listings section is a dynamic table that provides information on the title, notes, date created, and date last modified for all entered climate scenarios.

6.3.2 Climate Scenarios

The new climate scenario page consists of three primary parts: the information section, and budget selection. The information section will state that the user is beginning a new AgBizClimate scenario and provide rules and data entry components for the scenario and notes for the new scenario. Specifically it will allow users to enter text specifying the scenario name and notes for the new scenario. Finally budget selection portion will allow users to add a budget or select a preexisting budget from the existing database.

6.3.3 Region Selection

The region selection page will consist of a jumbotron stating that the user is on the region selection page, a brief message explaining what the page is for, and two drop down boxes that the user can use to select the desired state

and county for their new climate scenario. This page will also have a back, and continue button that allow users to either backtrack through the new climate creation process or move on to the new step.

6.3.4 Chart Page

The chart page will consist of a title prompting the user to consider how the following graph will affect their enterprise, a graph containing localized temperature, and precipitation data based on previous user input in the region selection page and a text box where the user can enter the percent they think the forecast will affect their yields or quality. This page will also contain the back, and continue buttons the other pages have which allow the user to more easily move around the tool.

6.3.5 Budget Review

The budget review page will have two major section. The first section is the income section which is a table containing the following information: income gross returns, unit sold as, quantity sold, price per unit, and total value. This table will also sum the total value from this section for the user. The second section is a general cash costs section and contains the the following information: name, unit, quantity, price per unit, and total cost. Finally for each table users will have access to buttons that will allow them to add new entries, remove entries, and edit entries.

7 Requirements Matrix

Shown below in figure 14 is our requirements matrix. This matrix shows how the different components will fufill the project requirements. For a full list of requirements please see the AgBizClimate requirements document.

Organization	OSU				
Project Manager Name:	Clark Seavert				
Project Description:	Adding A short term Climate Module to the AgBiz Logic Project				
ID	**Assoc ID**	**Functional Requirement**	**Status**	**System Component(s)**	**Software Module(s)**
001	FR1.1	Accepting location data in a request to Climate Data API	In Progress	Climate API	AgBizClimate
002	FR1.2	Transform users Location data from State and County to Lat Lon.	In Progress	Climate API	AgBizClimate
003	FR1.3	Authenticate with NWCTB	In Progress	Climate API	AgBizClimate
004	FR1.4	Request Data from NWCTB	In Progress	Climate API	AgBizClimate
005	FR1.5	Receive Response from NWCTB	In Progress	Climate API	AgBizClimate
006	FR1.6	Process the Data to JSON	In Progress	Climate API	AgBizClimate
007	FR1.7	Send the Data to the front end controller to be displayed	In Progress	Climate API	AgBizClimate
008	FR2.1	Give the user a landing page so they can choose between a long term climate scenario or a short term climate scenario	In Progress	U.I.	AgBizClimate
009	FR2.2	Request data from the Climate Data API after user has entered location for the Data	In Progress	Front End Controller	AgBizClimate
010	FR2.3	Plot the Resulting Data from the Climate Data API	In Progress	Front End Controller	AgBizClimate
011	FR2.4	Allow user to make adjustments to yeild and budgets after displaying data to user.	In Progress	Existing AgBiz Logic Modules	AgBiz Logic
012	FR2.5	Redirect uesr to Existing Budget Tool	In Progress	Front End Controller	AgBizClimate
013	FR3.1	Provide testing for Climate API	In Progress	Cliamte Data API	AgBizClimate
014	FR3.2	Provide Testing for UI Responsiveness	In Progress	Front End Controller	AgBizClimate
015	FR3.3	Provide Unit test for budget save	In Progress	Back End Controller	AgBizClimate

Figure 14: Requirements Matrix

3.2 Changes

3.2.1 Front End

In this section we will discuss the changes to the design of the front end of this application. We will discuss how our design changed over the course of the term.

The design of the front end of the application is largely unchanged from how we envisioned it at the beginning of the term. We were pretty much right about how the interaction between the front end and the back end would work in that we just made rest calls from the front end to the back end. However, It is fair to say that some things did change mostly the order of our pages.

The main change in our design over the term relates to the fact that we chose where the branch between short term and long term scenarios occurs later in our work flow. Originally, this was going to take place much earlier at the climate manager. After doing some thinking we realized that it made much more sense for this branch to happen at the region select page as the two scenario types share identical steps until after this point. Given that some of the routes in this design depend on that that fact the design has changed to include only one additional page and route that did not exist previously. This is the charts page for the short term scenario. This is the main page that we added. it is fair to say though, that many of the other exist pages did require minor adjustments in order for this to work.

3.2.2 Backend

Our original design for the back end of this document was in adequate as we really only had a surface level understanding for how the back end of this project worked. This section should have still been in the original document but would have been a small section where we described how we interacted with the existing back end code. Below I will provide a brief description of how we interact with the back end code.

It should be noted that there are other parts of the back end that I'm not going to discuss here that our application does rely on but that we are not making any changes too. As such we will not discuss the parts of the back end that we did not have to make changes too. The only major part of the back end that we changed was the model and database entries for climate scenarios. We added one field so we could denote a discernible difference between a short term scenario and a long term scenario. All of the other changes to existing code occurred on the front end.

3.2.3 API

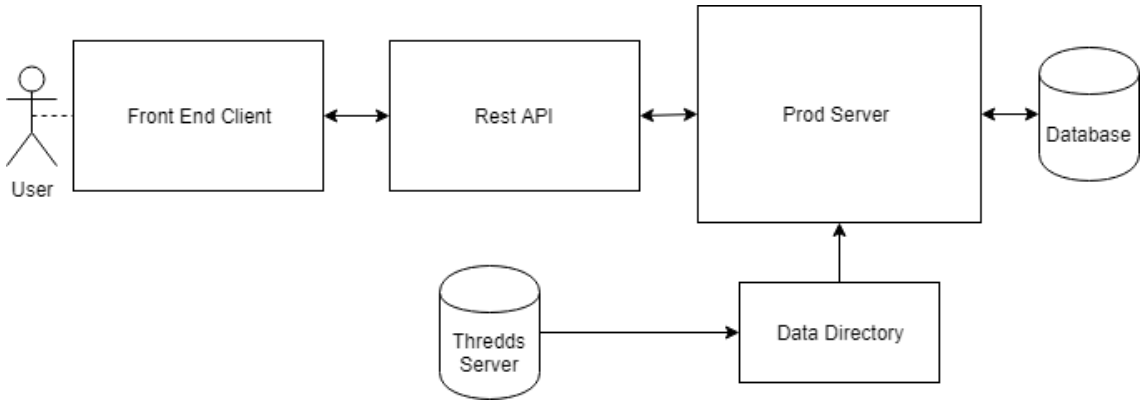
In this section we will discuss the changes to the design of the back end and API of this application. We will discuss how our design changed over the course of the term.

At the beginning of the term the data API was meant to be an endpoint that sent a URL request to the Northwest Climate Toolbox to grab climate data for a specific state and county in the United States. However, due to an assortment of issues with their server and components involved this was no longer a viable solution. Instead we downloaded the entire data file and data mounted it. The final API simply takes a state and county, then does a look up in the local data file to return the data to the request sent by the user.

3.2.4 Architecture

The last section of the design that changed over the course of the term was the Architecture of the application. Originally, we were planning on being able to make calls to a rest API that contained the climate data. We were planning on doing this as part of the back-end or if it was really simple as part of the front end. However, as the project progressed we realized that this was not going to work as the implementation of the API we were trying to use wasn't stable. As such we had to make some changes to the Architecture of the application so we could still

serve the data. What we ended up doing was mounting the vitalized container that our project is running in to file on the prod server. This now makes the Prod server part of the architecture design in a way that it wasn't before. Shown below is a diagram reflecting this change.



4 Tech Review

Oregon State University

CS 461

Fall 2017

Tech Review AgBizClimate©

Author:
Thomas Noelcke

Instructor:
D. Kevin McGrath
Kirsten Winters

Abstract

The purpose of this document is to research and consider different technical options for our application. In this document we research different options for data storage, HTTP request frame works, and testing frameworks. We will consider three possible choices for each section of the application. For each of these options we will weight the pros and cons of each. After comparing the different options we will select the option we would like to use for the *AgBizClimate* application. We have divided our application into 9 different section and each of us has preformed this analysis for each of the nine sections.

Contents

Oregon State University

CS 461

Fall 2017

Tech Review: Linking Seasonal Weather Data to AgBizClimate

Author:
Shane Barrantes, 29

Instructor:
D. Kevin McGrath
Kirsten Winters

Abstract

This document will provide an overarching analysis on front-end frameworks, graphing frameworks, and back-end design that we considered and selected for the AgBizClimate project. The topics included in this document will mirror my primary responsibilities for this project.

1 Front-End Frameworks

1.1 Overview

Front-end technologies are extremely important to web applications because they supply the foundation and interface for user interaction. This interaction is the first glimpse that users get into the application and will use it to judge the appearance, feel, and usability of the product. We want to use a front-end framework to give structure to our front-end interface, creating a groundwork for our application and making sure we can add any additional features with minimal time expenditure. We will be using the framework to build a user interface that is easily navigable and will provide dynamic graphing output based on user input.

1.2 Criteria

The front-end framework we pick needs to accomplish several things. First, it must enable our project to be highly customizable. Secondly, it needs to have a logical structure that allows us to create new features easily and produce a functional interface. Lastly, it needs to play nicely with graphing libraries so that we can cleanly display the graphing output we generate from the seasonal weather data to the user.

1.3 Potential Choices

1.3.1 Angular JavaScript

Angular JavaScript is a relatively new web framework built around HTML5, CSS3, and JavaScript that is developed and maintained by Google[1]. Angular JavaScript is not Googles first framework, but it is the most modern so it is highly dependable. Since the debut of this framework it has standardized the way web applications are structured and is used as a model for front-end design[1]. Angular supports easy REST actions, Model View Controller, and Model View View-Model.

1.3.2 React JavaScript

React is the latest and greatest front-end JavaScript library. It is only four years old and quickly becoming the most used front-end technology due to its simplicity and strength in building user interfaces[2]. This strength comes from the fact that you dont have to write separate HTML with react, but instead you describe what you want and React builds the HTML for the designer. Perhaps Reacts biggest strength is its reactive updating. When an input is changed React instantly updates the component without refreshing the page.

1.3.3 Raw JavaScript

JavaScript without additional frameworks is still fully functional for designing a front-end interface and structure and brings several strengths. The first strength is that raw JavaScript designers dont have to spend additional time learning and choosing which framework they want to use. Secondly, getting off the ground and creating the application can be easier due to not having to spend time setting up the application framework which can sometimes be overkill in comparison with the project goals. Lastly expanding the code base once its created will be easier since you wont have to pull in and learn additional frameworks, but this strength only exists if creator is also the maintainer of the wrote the code.

1.4 Discussion

Raw JavaScript's benefit is that we wont have to learn additional frameworks and can immediately write our own code base. However, for this project I believe it is a weakness since the rest of the AgBizTeam will have to maintain our code after we complete the module; so existing within a standardized framework is a good idea. The existing model for AgBizClimate is model view controller which Angular JavaScript was built to support therefore making the product is consistent and reliable. React is a phenomenal front-end technology with no real weaknesses except for not being a self contained framework.

1.5 Conclusion

For the purposes of this project raw JavaScript is not a viable solution for the previously mentioned reasons. React JavaScript and Angular JavaScript dont play well together so its important that we only use one of them for the front-end technology. Since our team is producing a submodule of an existing product we are required to use the technologies the development chose at the start of development; so we will be using Angular JavaScript for our front-end framework. As a side note we will also be using the front-end HTML framework Bootstrap with React.

2 Graphing Frameworks

2.1 Overview

One of the essential and most valuable components of the AgBizClimate submodule is its ability to generate useful and explicit graphs based on user input. These graphs need to be visually pleasing and easy to interpret so that users can quickly ascertain the relevant information they can use to assist their normal decision making process.

2.2 Criteria

The graphing framework that we choose needs to work closely with JavaScript and HTML5 since that is the primary base for our dynamic web application. It also needs to work quickly since we are trying to provide user with rapid feedback based on input and have high responsiveness and feedback to increase overall user satisfaction. Ideally the graph selection we chose will have minimal overhead so integration is quick and easy.

2.3 Potential Choices

2.3.1 Angular Chart.js

Angular chart.js is an open source graphing library for JavaScript. It is responsive and works well with JavaScript and HTML5. It can provide eight different types of charts and can interleave different chart types together [4]. This interleaving process creates singular graphs that can illustrate data differences more clearly. Chart.js is script-able and also supports animations which could assist users in understanding the data[4].

2.3.2 Plotly.js

Plotly.js is a high level JavaScript graphing library that is built on top of d3.js and stack.gl[5]. It provides the ability to chart data with 20 different chart types including 3D graphing, animations, sub-plotting, and mixed plotting[5]. One of the stand out aspects of Plotly.js is its ability to stream data in and dynamically produce graphs.

2.3.3 D3.js

D3.js for data driven documents is a JavaScript library that was created solely to manipulate documents based on data. It is one of the most widely used and popular JavaScript graphing libraries in existence. It works closely with front-end frameworks to produce high quality HTML5 tables and visualizations. D3 is sleek, fast, and effective; allowing high quality graphs to be generated with almost no overhead and assisting in high responsiveness and usability[6].

2.4 Discussion

Chart.JS is an effective open source JavaScript library that would work well for our project. Its primary drawback is the limited number of chart types. With the current AgBizClimate setup this is not an issue, but when the submodule expands it could create problems down the road due to limitations on visualizations. Plotly.js is an extremely powerful JavaScript tool built on top of the other option, D3.js and provides a wide variety of graphing options. However, due to the size and high level of abstraction Plotly.js is slower than the other two graphing libraries with more required overhead. D3.js is the nice middle ground between these three libraries. It's faster with more responsiveness than plotly.js and it provides a wider range of chart types than Charts.js.

2.5 Conclusion

All of the graphing frameworks this document has discussed have their varying strengths and weaknesses, however I believe D3.js is best suited for the job. Since our team is producing a submodule of an existing product we are required to use the technologies the development chose at the start of development; so we will be using Angular Chart.js for our graphing framework.

3 Back-end Designs

3.1 Overview

The Back-end design or software architectural pattern is an essential part of building a functioning web application. Deciding on which model to use will influence design decisions for all parts of the web application including the front-end, and the back-end The following choices for back-end design are the most popular and modern options that are being used today.

3.2 Criteria

The back-end technology has two key requirements in order for it to be successful with the AgBizClimate project. The first requirement is that there needs to be a distinct front-end and back-end. Secondly, it is essential that the front-end is stated and sessioned, so that the application remembers the user inputted steps to reach the decision assistance stage.

3.3 Potential Choices

3.3.1 REST API

REST APIs also known as a representational state transfer application passing interfaces have a clear separation between the client and server which is highly desirable for creating seamless user interfaces[7]. This separation makes products using REST extremely scalable with very little effort. Rest APIs are also useful due to the API itself being separate from the code-base. This means that you can have servers running different languages, but as long as the API is the same then the application will still function.

3.3.2 Model View Controller

The Model View Controller is one of the most basic and widely used back-end architectural patterns. The Model consists of the logic and collection of classes necessary for the web application. The View is what the user sees and where the user interface resides and the controller is what handles requests and passes information between the model and the view[9]. The Model View Controller Paradigms main strength is the separation between the visual components of a web application and the functional back-end. This allows the front-end interface to be altered with no real effect on back-end functionality.

3.3.3 Model view ViewModel

The Model View View-model design pattern is utilized to separate the front-end from the back-end with an integrated ViewModel component. The Model consists of the logic and collection of classes necessary for the web application. The View is what the user sees and where the user interface resides. The ViewModel is responsible for altering the state of the view and manipulating the model with the information that was gained from the altered view[8]. This paradigm allows events to trigger in the view itself.

3.4 Discussion

Each of the previous paradigms have their strengths and weaknesses. The REST API is fantastic for simple queries to the back-end, but it has a harder time tracking progression through multiple steps which is what we need for AgBizClimate. The Model View ViewModel method is great for projects that have long forums and require more dynamic views, but its a bit overboard for our needs on this project. The Model View Controller method allows the designer to have a distinct front-end and back-end while processing data between the components. Maintaining this structures allows the development team to spend very little time working on the front end after it's initial design and completion.

3.5 Conclusion

Since our team is producing a submodule of an existing product we are required to use the technologies the development chose at the start of development; so we will be using the Model View Controller paradigm for our back-end design.

References

[1] Angular JavaScript
https://www.linkedin.com/pulse/20140613173601-45832080-why-to-choose-angularjs-javascript-framewo
Pankaj Kumar Jha, June 13th, 2014.

[2] React JavaScript,
https://medium.freecodecamp.org/yes-react-is-taking-over-front-end-development-the-question-is-wh
Samer Buna March 30th, 2017.

[3] Raw JavaScript,
https://www.sitepoint.com/frameworkless-javascript/. Pawel Zagrobelny.

[4] Chart.js
http://www.chartjs.org/.

[5] Plotly.js,
https://plot.ly/javascript/.

[6] d3js,
https://d3js.org/.

[7] The Representational State Transfer,
https://www.service-architecture.com/articles Douglas K Barry.

[8] The MVVM Pattern,
https://msdn.microsoft.com/en-us/library/hh848246.aspx

[9] The MVC Pattern,
https://msdn.microsoft.com/en-us/library/windows/hardware/ff550694(v=vs.85).aspx

Oregon State University

CS 461

Fall 2017

Tech Review AgBizClimate©

Author:
Thomas Noelcke

Instructor:
D. Kevin McGrath
Kirsten Winters

Abstract

The purpose of this document is to research and consider different technical options for our application. In this document we research different options for data storage, HTTP request frame works, and testing frameworks. I will consider three possible choices for each section of the application. For each of these options I will weight the pros and cons of each. After comparing the different options I will select the option I would like to use for the *AgBizClimate* application.

Contents

1 Data Storage

1.1 Overview

For the *AgBizClimate* application we will need a way to store data so it can be easily retrieved later. For this application we will store a variety of data including budget data, weather data, and user information. This information will need to be quickly recalled so it can be used in our application. Generally, we will want to select a data storage option that will be easy to set up and allow us a lot of flexibility with what kind of data that can be stored. We will also want a data storage option that will quickly recall stored data so it can be used by the application.

1.2 Criteria

To determine the best choice for our application i will analyze the performance of the data storage based on, Ease of Development, and Ease of Set Up and Flexibility. I will analyze run time speed however, this will not be one of the criteria as it is not critical that our data is retrieved as quickly as possible. Generally, its much more important to consider Ease of development, Set up and flexibility because the primary concern is being able to get the application up and running quickly. Ease of development and ease of set up will be a subjective measures for each option. In those sections I will use the opinion of other software developers along with my own experience to compare each option.For these criteria I will rate each option on a scale from very easy to very hard. Flexibility will be a measure of how easy it is to store different kinds of data using each option. For Flexibility I will measure each option from flexible to rigid.

1.3 Potential Choices

1.3.1 PostGreSQL

PostGreSQL is an open-source relational database management system. PostGreSQL uses the Sever Querying Language (SQL). PostGreSQL uses the relational database model. This model sets up tables that represent a certain type of data. We can then run SQL quires on this data base to get the data we need for our application [1].
Though SQL is quick, In my experience it is not as developer friendly as the other options in this analysis. Generally, writing raw SQL queries is difficult and time consuming. This is especially true when your data models are very complicated. Using Raw SQL also requires the developer to manually figure out how to make the mapping between the database and the models used at the application level. With raw SQL it is also more difficult to change the structure of the database once you have created the database. These changes will require potentially complicated scripts along with scripts that convert the data to fit into the new structure [2].
Another problem with SQL in my experience is that it is more difficult to set up. The configuration process can be complicated. Additionally, if you choose to use raw SQL you must also set up your data base as third party application separately from your actual application.
Another problem with SQL is that it is rather rigid in the way that you must store data. For example if you want to store a list object in an SQL data base you must create a new table where one row represents one item in the list. This item must have its own unique id. Additionally, if you want to have a list of lists it gets even more complicated. Now you need to create another table to represent a name and ID for each list you want to store and you must relate every item you want to put in that list back to the parent item in another table. This can get very complicated in a hurry. Another problem is if you don't know what the structure of the data is going to look like before run time it is impossible to store this data in an SQL database. This makes PostGreSQL rather rigid in

terms of flexibility.

1.3.2 Python ORM

Python ORM or Object Relational Mapper does not substitute for an SQL database. There will still need to be an SQL database running on the back end. However, the ORM framework allows developers to create objects in python that then map to the data base. Often times this type of frame work allows the developer to create the objects first and let the framework deal with creating the SQL database on the backend. Given that this is not a replacement for an SQL it does make working with an SQL database much easier.
This type of frame work has several advantages, it allows for easy development and set up. The ORM frame work allows the developer to be completely insulated from the SQL data base on the back end. This means that instead of writing SQL queries to get data from the database the developer is able to use objects in python to access data that is stored in a database. This makes development and set up Easier on the developer. This is because the developer doesn't have to worry about writing complicated SQL statements or setting up complicated relationships between tables. This frame work also allows the developer to develop the code first and let the framework worry about creating the database the data will ultimately be stored in [3]
The ORM framework also allows for great flexibility in what you can store in a database. This type of frame work allows for mappings between python objects and the data base. So nearly anything you can store in an object in python, you can also store in a database. However, it should be noted that you must know the structure the object you are trying to store before run time.
Though this approach is very easy for the developer it isn't with out cost. The ORM approach does take a hit in terms of run time performance. The ORM framework will be slower than raw SQL. Another problem with this type of framework is that it doesnt leave the developer very much control over the database. This means that you are stuck with what you get. If the frame work structures something in the database in a way you don't like you don't have a ton of choice about that. Additionally, making database chances can cause you to loose data in the database if you are not careful about how you handle the migrations [3].

1.3.3 MongoDB

MongoDB is an open source database NoSQL database. This means that instead of using tables and rows like an SQL relational database, MongoDB uses collections and documents. Documents are individual data members where there is a key value associated with each document. Collections contain multiple documents. Collections allow the developer to store many items in a database. This structure allows for the structure of the data being stored to be determined after run time. This allows for greater flexibility because you can store dynamically generated objects[4].
MongoDB is also fairly simple to set up. This is because of the way that objects are stored we don't need to set up and complicated tables. We can simply set up our data base and insert the data. This also means that if we want to change the structure of the data down the road we don't need to make massive changes to the data base. This makes MongoDB much more developer friendly and also very flexible[2].
However, It should be noted that the usability and flexibility of MongoDB is not with out cost. The biggest cost of the flexibility of MongoDB is that it is harder to relate two items in a database. For instance if want to have one entity that represents a user and another that represents a user role, this becomes a challenge in MongoDB. There are work around and ways to solve this problem however it should be mentioned that this is a problem[5]. Another trade off in using MongoDB read and write operations are generally asynchronous. This means that you can tell the database to do something and then move on other tasks and the data base will return the result of your query later.

On the surface this sounds nice but if you are doing a lot of reading and writing operations in the same block of code this can cause problems. For instance if you write several items to the data base and then need to read items out that depend on those items you just added you may get errors because the first write hasn't finished yet[6]. Finally, MongoDB may also be much slower for some operations than traditional SQL. One example where this becomes apparent are aggregate functions where you want to preform some sort of manipulation of the order of the data[7].

1.4 Discussion

As mentioned earlier we can divide the types of data we need to store into two distinct groups, weather data and user data. The user data will generally have relationships between different parts of the data. For instance a user will have an address and a phone number. The user will also have various different budgets. These relationships make the data easier to store in an SQL database such as PostGreSQL. To farther simplify storing the data we could also use Python ORM. This would allow us to create relationships in the data, easily develop the database and store the data. The weather data on the other hand is much more like a large list. This sort of data can be sotred in an SQL data base such as PostGreSQL however, this would make the development of the project more difficult. A more suitable choice for storing the weather data would be MongoDB. This is because MongoDB allows for easy storage of lists with little complexity than PostGreSQL. MongoDB provides the easiest and most flexible way to store the weather data.

1.5 Conclusion

For this project we will use PostGreSQL and MongoDB. We decided to use these frame works because our client requested it. This is because we are adding to our clients system that has already been created. However, the client did put in some good though into these choices. They chose PostGreSQL because its a powerful relation database management system that will work well for the user data. They also chose MongoDB for the weather data because it provides a simple to set up and easy to use storage tool for storing large lists of data.

2 HTTP Request FrameWork

2.1 Overview

The *AgBizClimate* project will need a way for the client on the front end of the application to communicate with the API at the back end of the application. We have already determined that we will use HTTP request to facilitate this communication. However, we have not decided which framework we should use to make the HTTP requests with. Generally, the frame work we use to make HTTP requests should be easy to use, easy to read and allow us to quickly right requests.

2.2 Criteria

For the purposes of this analysis we will consider ease of development, readability and compatibility with AngularJS. We are considering compatibility with AngularJS because our client has required us to use angular. This will be measured on a scale from compatible to uncompilable. Ease of development will be a subjective measure of how hard it will be to develop software with a framework. This will be measured on a scale from easy to very hard. Readability will be a subjective measure

of how easy it is to decipher the meaning of the request from the code. This shall be measured on a scale from readable to incomprehensible.

2.3 Potential Choices

2.3.1 jQuery Ajax

When talking about HTTP requests in java script Ajax is probably what immediately comes to most software developers minds. Ajax is one of the original frameworks for making HTTP requests with out updating the page. Ajax allows for asynchronous requests to the backend. Generally, Ajax is not very human readable. Ajax can be read by humans but those humans are going to need to have had experience with Ajax before. It is important to note that Ajax requests will require more work that most of the other HTTP request frameworks we are discussing. Additionally, Ajax can be use with Angular however we will be required to take extra steps to ensure that the requests are handled correctly. If you choose to use this framework with angular you will also need to do more work to be able to test your code.

2.3.2 axios

Axios is a promise based HTTP client for web browsers. Axios will also allow us to make asynchronous calls to the backend of our application. It is also important to note that axios HTTP requests are much more concise making them human readable[8]. Axios can also be integrated into Angular however, this will require some set up. If you choose to use axios with angular you will need to set up your own custom tests because this frame work will not integrate into angular testing framework. Additionally, you may run into headaches down the road when angular updates are made because axios is not maintained by angular.

2.3.3 AngularJS $http

$http is a module in AngularJS that enables communication with remote HTTP servers by using the browsers built in tools. This frame work provides a layer over top of an ajax request that makes the request easier to read and write. These requests are much easier to read and writing the request is much easier that raw Ajax. This HTTP frame work also integrates seamlessly with AngularJS because its a core service. This means that testing these requests with the Angular testing frame work is fairly easy and updates to Angular wont break your HTTP requests [8][9].

2.4 Discussion

In comparing these three frame works I noticed that $http really stood out from the other two frame works. This is because $http is a core service in the AngularJS framework. This makes it much more compatible with Angular than Ajax or axios. Comparing $http and axios out side of the context of Angular they are pretty similar. Both produce much more readable http requests than Ajax and both are are fairly easy to use.

2.5 Conclusion

For this project we will use $http to make http requests between the front end of our application and the back end of our application. We chose to use this framework because it integrates seamlessly with Angular while also making HTTP request readable and easy to develop.

3 Testing Frame Work

3.1 Overview

For the AgBizCliate system we will be required to write unit tests for our code. Unit tests will help us ensure that our application meats the functional requirements. Unit tests will also be helpful for feature projects to ensure that parts of the application have not been broken by a change. In general good unit tests will improve the reliability and extend the life of a project. For this project we want a testing frame work that will allow us to quickly create unit tests in Python for Django Applications. We want to be able to do this because our client has already specified that we will be using Python and Django.

3.2 Criteria

For the purposes of this analysis we will consider ease of development, compatibility with Django, and simplicity of setup. Ease of development is a subjective measure of how easily tests can be developed with each framework. Ease of development will be rated on a scale of easy to hard. This frame work will also need to be compatible with Django as this is the frame work our application will be written in. Finally, we will also consider ease of set up. This is a measure of how easy a framework is to set up so we can begin writing unit tests.

3.3 Potential Choices

3.3.1 Python unittest

Python unittest is a testing frame work built into Pythons standard library. The unittest framework provides the necessary scaffolding to setup, shutdown and run unit tests. This testing framework will feel familiar is it is part of the greater Junit project. This makes writing unit tests feel familiar and fairly easy. However, for this application using this testing frame work would require a lot of set up. This is because this is a web application and unittest does not include tools that make it easier to set up and run tests for web applications. More generally, unittest wills take longer to develop because it provides less scaffolding[10]. This makes unittest less developer friendly. Unit test can easily be used with Django as it is part of the python library. For this reason unittest is also very simple to set up.

3.3.2 py.test

py.test is a popular python testing framework that helps reduce the repletion required in unittest. py.test also contains powerful utilities that make testing many kinds of applications easier. For our project py.test would make testing our web app much simpler than unittest. This would make test development quick and easy. Though this framework makes tests easier to develop that does come at a cost. This framework requires extra setup to integrate with Django. This will make setting up this framework more difficult that either Django.test or unittest. Though the set up isn't complicated it would make testing our application more difficult and more complicated [11].

3.3.3 Django.test

Django.test is a testing frame work that is baked into Django. This frame work provides powerful tools for testing Django applications. This frame work is part of the greater Django framework. Because Django.test is built into the Django framework, there is very little set up. This also means that there will be no compatibility issues with Django either now or in the future. This testing suite

also provides powerful utilities that make it easier to develop tests. This frame work also gives us specific tools designed to test Django applications[13].

3.4 Discussion

While considering the options there is one option that fits our application the best which is Django.test. Django.test will allow us to test our Django application with minimal effort as it requires no special set up. Additionally, Django.test also contains tools like py.test that will enable quick and easy development of tests. Though py.test may also be a good option, it doesn't integrate into Django as well as Django.test. Unittest is also another good testing option as it is apart of the python language. However for the purposes of this project unittest would make the development of tests more difficult.

3.5 Conclusion

For this project we will be using Django.test to write unit tests for our application. We chose to do this in part because our client asked us to use this frame work. However, our client asked us to use this frame work with the goals for our application in mind. For this application we want to be able to quickly and easily generate unit tests and Django.test is one of the best tools available for testing Django applications.

4 References

[1] Home, PostgreSQL Tutorial. [Online]. Available: http://www.postgresqltutorial.com/what-is-postgresql/. [Accessed: 22-Nov-2017].
[2] MongoDB vs SQL: Day 1-2, MongoDB. [Online]. Available: https://www.mongodb.com/blog/post/mongodb-vs-sql-day-1-2. [Accessed: 22-Nov-2017].
[3] M. Makai, Object-relational mappers (ORMs), Object-relational Mappers (ORMs) - Full Stack Python. [Online]. Available: https://www.fullstackpython.com/object-relational-mappers-orms.html. [Accessed: 22-Nov-2017].
[4] What is MongoDB? - Definition from WhatIs.com, SearchDataManagement. [Online]. Available: http://searchdatamanagement.techtarget.com/definition/MongoDB. [Accessed: 22-Nov-2017].
[5] J. Headley, The Problem with MongoDB Hacker Noon, Hacker Noon, 12-Feb-2017. [Online]. Available: https://hackernoon.com/the-problem-with-mongodb-d255e897b4b. [Accessed: 22-Nov-2017].
[6] T. S. Chief, Potential problems and issues with using MongoDB, Stackchief. [Online]. Available: https://www.stackchief.com/blog/Problems%20with%20MongoDB. [Accessed: 22-Nov-2017].
[7] A. C. Weinberger, Benchmark: PostgreSQL, MongoDB, Neo4j, OrientDB and ArangoDB, ArangoDB, 13-Oct-2017. [Online]. Available: https://www.arangodb.com/2015/10/benchmark-postgresql-mongodb-arangodb/. [Accessed: 22-Nov-2017].
[8] axios, npm. [Online]. Available: https://www.npmjs.com/package/axios. [Accessed: 22-Nov-2017].
[9] $http, AngularJS. [Online]. Available: https://docs.angularjs.org/api/ng/service/$http. [Accessed: 22-Nov-2017].
[10] unittest vs py.test, Bytes IT Community. [Online]. Available: https://bytes.com/topic/python/answers/43330-unittest-vs-py-test. [Accessed: 22-Nov-2017].
[11] 26.4. unittest - Unit testing framework 26.4. unittest - Unit testing framework Python 3.6.3 documentation. [Online]. Available: https://docs.python.org/3/library/unittest.html. [Accessed: 22-Nov-2017].
[12] C. Maske, The Engine Room, Using pytest with Django - The Engine Room - TrackMaven. [Online]. Available: http://engineroom.trackmaven.com/blog/using-pytest-with-django/. [Accessed:

22-Nov-2017].
[13] Django Tutorial Part 10: Testing a Django web application, Mozilla Developer Network. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Testing. [Accessed: 22-Nov-2017].

Oregon State University

CS 461

Fall 2017

Tech Review

Author:
Shengpei Yuan

Instructor:
D. Kevin McGrath
Kirsten Winters

Abstract

The purpose of this document is to research different technical options and consider possible choices for our application. In this document, I have researched different options for containers, back end design, and styling framework, and I will analyze three choices for each section of the application. For each option, I will introduce the principle, advantages and disadvantages as it relates to this project. After comparing all three choices, I will choose the appropriate option for the AgBizClimate application.

1 Container

1.1 Overview

Containers are a powerful virtualization technology that is helpful for software development with the cloud environment. Generally speaking, they greatly help environment construction and deployment work for developers and maintainers. Containers also improve efficiency and reduce cost of developing complicated software systems.

1.2 Criteria

Containers are one important technology in cloud computing. It provides independent running environment for various software applications. It is believed that containers may replace virtual machines in the near future as the dominant technology for constructing cloud-computing environments. Basically, containers work on the OS layer and provide runtime environments for programs. The OS distributes computing, memory and peripheral resources for containers, and it owns isolated namespaces for programs.
The performance of containers should be considered carefully as there are substantial differences between computing architectures and their native hosts. Ideally, we would expect little or no reduction on computing or memory access efficiency when using containers relative to the native host. To test whether containers meets this requirement, one feasible method is to examine and compare the specifics of the same application deploying both on containers and native host.

1.3 Potential Choices

1.3.1 LXC

LXC is one of the earliest containers to provide virtual Linux runtime environment for software applications. The core characteristics of LXC are built on the basis of resource management and isolations techniques of the Linux kernel. For instance, the cgroups feature of Linux kernel that divides processes into groups for management is the essential framework for managing resources for processes in LXC.

1.3.2 Docker

Docker is a mature container tool has almost replaced the conventional Linux Containers (LXC) and helped solve the availability problems of many software systems. It is claimed that Docker is the prevailing virtualization technology in todays software building environment. However, with the rapid development of the Docker ecosystem as a tool for cloud-computing it has made constructing other businesses based on it harder. Docker is one of the best components for constructing a software system in virtual computing environments. One perspective is that Docker is or will become an ecosystem for software construction. This will make it vulnerable in many ways, whereas Rocket makes a fresh start and only focus on working as a component to help construct complex software systems. Another perspective is that the core value of Rocket is exactly the start point of Docker, integrating complicated software systems into one independent platform. The founder of Docker, Hykes agrees that Rocket could be one important tool for customizing configurations of containers, and will define the future of containers technology.

1.3.3 Rocket

Rocket is a quite new open-source container technology for software developing created by CoreOS in 2014. It is implemented in Go. As a command-line tool, Rocket works quite like Docker to package software applications and their dependency libraries or systems to a plantable container.

It is known that Rocket would have no compatibility issues with Docker. Also, rocket will bring new ideas for software container technology and relevant markets. It never tries to provide broad friendly functionalities like cloud acceleration tools, and integrated systems. Rocket will become more purified as a standard container tool for the software industry. It has a start point that multiple heterogeneous container platforms could be integrated into. This will help solve the availability problem. Moreover, it would try to alleviate the security issues of Docker. It should be noted that Docker and Rocket both have their own strength and weakness. Generally speaking, Docker would be more competent at complicated and huge systems, while Rocket is a better fit for lightweight and small software applications.

1.4 Conclusion

The vast adoption of Rocket among great software systems proves that it satisfies the requirements of the software world. Rocket is designed to boost the success of software systems, including specific measures like guaranteed security, flexible components and open standards. Different companies may choose different container tools between Docker and Rocket according to the specific contexts of their systems. As a result, for this project, we will adopt Rocket as an independent container library, so that it could be easily integrated into the existing project AgBizClimate and the corresponding environment. In other words, we need a container tool to construct components for our system rather than to create a new system from scratch.

2 Back end Design

2.1 Overview

There are several powerful and popular models for building the back end of a web applications. RESTAPI and MVC are two important technologies among them. Besides, we would also describe a little about MMVC, an extended version of MVC. We will introduce them in the order of when they were proposed in order to help readers to understand them more easily.

2.2 Criteria

The backend architecture technologies mainly solve the problem of decomposing the functionality of the application reasonably and efficiently for software developers. Ideally, all modules have clear bounds between each other and tight connections within themselves. Besides this, it is also important to present the structure and functionality of the application clearly and easily to both developers and end-users.
The performance of the back end architecture is important as once determined they dominate the entire development process of the application. The back end architecture has great impact on the extendability and robustness of the program. It can be hard to test whether a back-end architecture can fully meet our design requirements as it is very abstract high layer design. However, it is commonly believed that a good back-end architecture must make improvements on the efficiency of entire process of system development, function extensibility, code readability, and maintainability.

2.3 Potential Choices

2.3.1 MVC

The MVC (Model View Controller) model was proposed as early as 1970s and used by Smalltalk, and it is still popular and used to develop large software applications in various fields today. Many software languages and frameworks like the prevailing Java Struts and Spring MVC use the MVC architecture. The core concepts behind the MVC model is that no matter how simple or complex a

software system is, it could always be stratified into three layers from the perspective of structure. First, the View layer, which lies on the top, is the one directly seen by the end user. It usually works as an interface between users and the program, and is also called the shell of the program. Secondly, the Model layer, which lies in the bottom, usually represents and stores the data or information of the program. For many programs, this layer is the core layer that largely dominates the structure of the other two layers. Thirdly, the Controller layer, which lies in the middle, is mainly responsible for the interactions between the Model and View layers. More specifically, it accepts user instructions from View layer, retrieves data, manipulates the model, and sends back the resulting data to view for to be displayed. The above three layers are tightly connected together as well as being independent. The internal operations within each layer never affect other two layers, and each layer provides appropriate interfaces for the other layers. The essential point of this design is that the entire system could be divided into independent modules so that neither changing appearances nor changing data would alter other two layers. Hence, it greatly reduces the cost of maintaining, upgrading and testing the system.

2.3.2 RESTAPI

RESTAPI (Representational State Transfer) was proposed by Roy T. Fileding in 2000 as a simple and extensive standard for developing web applications. In fact the HTTP protocol is one typical application of the RESTAPI architecture. With the vast applications of cloud computing technology, RESTAPI is very popular used by many software architectures and developers to build large web applications. The most essential features of RESTAPI are resources, unified interface, URI and statelessness. The core concept of RESTAPI is state transfer. More specifically, the network resources and actions are clearly isolated from the state of the application. The transferring of states is described by the URI so that it is very clear for both developers and end-users. By Passing around states from back end to front end, a uniform interface is created by mapping HTTP methods to CRUD operations. The RESTAPIs, front end and back end, are stateless, so that the APIs are very efficient. Consequently, all information, including modifications, are saved and represented by the requesters, and servers contain no state information.

2.3.3 MMVC

The MMVC is an extension of MVC that optimized the Model or data layer. It is essentially the same as MVC except that it adds a view Model between Model layer and View layer. The new layer acts as an interface between Model and View layer since the data models in applications are becoming more and more complicated. We need to know three essential points about MMVC. Firstly, it is compatible with the existing MVC architecture. Secondly, it makes the software applications more testable. Thirdly, it works best with a binding mechanism. It should be mentioned that MMVC is a relatively new back-end architecture technology and is currently not used broadly in the software industry.

2.4 Conclusion

MVC and RESTAPI try to build a powerful general architecture for building various software systems. MVC tries to model the system based on actions, whereas RESTAPI mainly focuses on the data, the sate and transition to different states. MVC emphasizes dividing the internal implementations into three layers Model, View and Controller, and RESAPI decompose the web applications in terms of outside appearance, transitions of states of system. Basically, this project would try to follow RESTAPI standards for back-end design so that the programs are well structured and flexible for extension.

3 Styling framework

3.1 Overview

The styling frameworks are higher level programming languages based on CSS (Cascading Style Sheets) to make CSS (Cascading Style Sheets) development flexible and efficient. There are many popular styling frameworks for building beautiful and consistent looking web applications. We would talk about three popular ones LESS, SASS and Bootstrap.

3.2 Criteria

The core function of styling framework is to allow easier and efficient web UI design using CSS. A good styling framework will have following three features. First, it provides programming ability like logic examination, loops and functions for basic CSS code. Second, it provides rich and practical predefined templates, components, and a suite of themes for quick design of web UIs. Thirdly, it integrates other front-end languages like HTML and Javascript for flexible and easy developing.
The performance of front end styling framework is also important as it largely influences the UI design and system interaction. For instance, a good styling framework would provide rich dynamic UI components with well defined interfaces. To test whether the framework fully meets the requirements, one could compare the specs for the styling of HTML components before and after using relative frameworks. In fact, it is quite easy for testers or end-users to measure the differences between different UIs after using them.

3.3 Potential Choices

3.3.1 LESS

LESS extends CSS and introduces module conceptions into it. It provides much functionality for front-end developers beyond the basic CSS. It has similar grammar rules as basic CSS. The LESS grammar rules are similar to SASS grammar rules. Like SASS, LESS is also an extension of CSS3 that introduces rules, variables, selector and inheritance. It tries to generate well-formatted CSS codes that are easy to organize and maintain.

3.3.2 SASS

For front-end programmers of web applications, SASS provides more powerful functionality than LESS, which works more like a real programming language than LESS. However, for UI designers, LESS seems to be clearer.

3.3.3 Bootstrap

Unlike LESS and SASS, Bootstrap is one comprehensive and powerful front-end framework based on HTML, CSS and JavaScript. Generally speaking, Bootstrap tries to integrate tools like Compass, Blueprint and h5bp. Bootstrap is a comprehensive framework for web front-end development. It should be mentioned that Bootstrap use Normalize.css to reset CSS, which has becoming the practical standard (used more broadly than Eric meyer 2.0 implementation of Compass). It is also compatible with h5bp, meaning you can use h5bp and Bootstrap concurrently in one project. Generally, Bootstrap has three key features. Firstly, it includes the complete basic modules of CSS, although not as powerful as Compass. This make it possible for programmers to use basic CSS attributes for simple customization of HTML elements. Secondly, it provides some suites of predefined CSS, including one grid layout system like blueprint but with a different style. This helps programmers to quickly define the overall look of their web applications. Thirdly, it provides a group

of UI components based on jquery like dialogs, navigation menus, and edit boxes. They are all both powerful and ascetically pleasing. This may be the most powerful strength of Bootstrap. In fact, it is becoming a practical standard of most jquery-based web projects.

3.4 Conclusion

Since we are not professional UI designers and work as programmers for the AgbizClimate application We plan to use Bootstrap. Bootstrap will allow us to employ both the powerful programming ability and the rich UI components of Bootstrap without needing much graphic design.

5 Weekly Blog Posts

5.1 Shane Barrantes

In This section I've imported the notes from my weekly OneNote updates. I will be denoting the term and week with the following format [Term Number].[Week].

5.1.1 Week 1.1

Plans

Problems

Progress

Summary
I signed up for project preferences and reached out to multiple project managers, none responded back.

5.1.2 Week 1.2

Plans

Problems

Progress

Summary
I was assigned to the agbiz project. I met with my group and we emailed the client to setup a meeting for Tuesday of next week.

5.1.3 Week 1.3

Plans

Problems

Progress

Summary

- Met with client to get overview of project.
- Met with TA to better understand his role in project.
- Met with Senior tech of project to discuss the
- specifications of the project.

5.1.4 Week 1.4

Plans

Problems

Progress

Summary

- Met with group to make the final draft of the problem statement.
- Signed a newly sprung NDA, will turn in early next week...
- Met with the TA.

5.1.5 Week 1.5

Plans

Problems

Progress

Summary

- Turned in Problem statement.
- Met with Client to discuss requirements document.
- Met with the TA.

5.1.6 Week 1.6

Plans

Problems

Progress

Summary

- Met with client to discuss the requirements
- document again.
- Turned in rough draft of the requirements document.
- Met with the TA.

5.1.7 Week 1.8

Plans

Problems

Progress

Summary

- Turned in Rough draft of the technical review.
- Met with TA.

5.1.8 Week 1.9

Plans

Problems

Progress

Summary

- Meeting with TA cancelled.
- Worked on design document.

5.1.9 Week 1.10

Plans

Problems

Progress

Summary

- Didn't meet with TA, got cancelled.
- Finished/turned in design document.
- Made progress report

5.1.10 Week 2.1

Plans

Progress

Problems

Summary

- Met with the client
- Got repo permissions
- Set tasks

5.1.11 Week 2.2

Plans

Progress

Problems

Summary

- Met with the client
- Defined API access

5.1.12 Week 2.3

Plans

Progress

Problems

Summary

- Worked on concept script
- Installed NetCDF and associated packages.

5.1.13 week 2.4

Plans

Progress

Problems

Summary

- Met with the client
- Defined API access

5.1.14 week 2.5

Plans

Progress

Problems

Summary

- Met with the client
- Worked on building NetCDF pipeline from source and all associated packages

5.1.15 week 2.6

Plans

Progress

Problems

Summary

- Finished development of the primary charts page
- Completed the midterm progress report

5.1.16 week 2.7

Plans

Progress

Problems

Summary

- Showed client the alpha in meeting
- Found a solution to bypass NetCDF to get climate data via URL

5.1.17 week 2.8

Plans

Progress

Problems

Summary
Completed the concept script which allowed us to get climate data while bypassing NetCDF.

5.1.18 week 2.9

Plans

Progress

Problems

Summary
Created API endpoint for the climate data and charts to pull from.

5.1.19 week 2.10

Plans

Progress

Problems

Summary
Integrated the climate data API into the beta and showed it to the client.

5.1.20 Week 3.1

Plans

Progress

Problems

Summary

- Weren't able to get in contact with TA
- Talked with team about plans for the term

5.1.21 Week 3.2

Plans

Progress

Problems

Summary
Found out that the API solution we spent months on doesn't actually work.

5.1.22 Week 3.3

Plans

Progress

Problems

Summary
Tried R and MatLab NetCDF solutions for our API, however neither worked for this project.

5.1.23 Week 3.4

Plans

Progress

Problems

Summary
Came up with a data mount solution for the climate API.

5.1.24 Week 3.5

Plans

Progress

Problems

Summary
Created our midterm progress report.

5.1.25 Week 3.6

Plans

Progress

Problems

Summary
Started creating back-end unit tests.

5.1.26 Week 3.7

Plans

Progress

Problems

Summary

- Finished creating the back-end unit tests
- Attended Expo

5.1.27 Week 3.8

Plans

Progress

Problems

Summary
Finished final client recommended changes and re-based to master.

5.1.28 Week 3.9

Plans

Progress

Problems

Summary
Recorded the Final Presentation.

5.1.29 Week 3.10

Plans

Progress

Problems

Summary

- Finished Final Document
- Finished Final Report

5.2 Thomas Noelcke

5.2.1 Week 1.1

In this week our group had not yet been formed. As such there is no report for this week.

5.2.2 Week 1.1

Plans

- Set Up First Group Meeting
- Meet With Client
- Work On Problem Statement

Problems

- Group Communication

Progress

- Started Slack Channel with group 10/4
- Contacted Client 10/4
- met with group 10/5
- setup meeting with client for 10/10 on 10/5
- set up meeting with lead developer for 10/12 on 10/6

Summary
This week our group started to get organized and reached out to our client. We also had a group meeting where we briefly discussed project details along with what skill sets we had. We also had a discussion about group communication.

5.2.3 Week 1.3

Plans

- Meet with Client
- Create Problem Statement Rough draft
- start working on requirements document
- Technical meeting with Sean Hammond

Problems

- We don't know how the NWCTB interface will work as we don't have API Access yet.

Progress

- Finished Rough Draft of Problem Statement 10/9
- Met with Clark on 10/10
- Met with Seen Hammond on 10/12

Summary
This week we met with Clark and Sean to start the conversation about our project. The first meeting with Clark was a higher level overview of our project where the second meeting with Sean was a technical meeting where we discussed the technical details of the project. We found one major blocker moving forward which is we don't have NWCTB API Access. To design our project we will need to figure this out.

5.2.4 Week 1.4

Plans

- Turn in final drafts for individual problem statements
- Start working on Requirements Document
- Work with group to start compiling group problem statement
- Try to get NWCTB API Access

Problems

- We Still don't have NWCTB API Access.

Progress

- Met as a group to work on the group problem statement 10/18
- Finalized and turned in the group problem statement 10/19
- Followed up with Clark regarding setting up a meeting with the Northwest Climate Toolbox to gain API Access

Summary This week we worked as a group to get our problem statement finalized. We also followed up with Clark regarding setting up a meeting with the Northwest Climate toolbox as this is the major blocker for our project. Finally, we sent our final group problem statement off to Clark for final approval. Clark approved our problem statement and we turned it in. This week we also started to think about our requirements list and what we will need to do next week to get started on our requirements document.

5.2.5 Week 1.5

Plans

- Set up meeting with NWCTB
- Start requirements document
- line requirements document
- Turn in rough draft of requirements document

Problems

- We still don't have NWCTB API Access

Progress

- Set up meeting to discuss requirements document for 10/24 on 10/23
- Started requirements document outline 10/23
- Finished First rough draft of requirements document 10/27
- Turned In rough draft of requirements document 10/27
- Meet as group with TA to review this weeks progress 10/27
- Meet as a group to discuss progress on requirements document 10/27

Summary This week we started off the week meeting with our Clients lead developer to start drafting our requirements document. The purpose of this meeting was to direct our efforts on writing the requirements document. This meeting was a great help and helped us to get a good start on the document. We also met with the TA on Friday and discussed our progress this week. After our meeting with the TA we meet as a group to go over our progress on the requirements document and to plan for next week.

5.2.6 Week 1.6

Plans

- Meet with Client to review draft of requirements document
- work with group to finish SRS final draft.
- follow up with Clark about setting up meeting for NWCTB API Access.

Problems

- Still don't have API Access for NWCTB

Progress

- Worked on SRS 10/30
- Met with client to review SRS progress 10/31
- worked on SRS 10/31
- Worked on SRS 11/1
- Worked on SRS 11/2
- Worked on SRS 11/3
- Turned in SRS 11/3

Summary
This week we met with our client to go over our draft of our SRS. During this meeting he suggested some edits that we made the following day. Additionally, we also worked as a group on completing our SRS. We currently have the final draft done but are doing a final proof read before we turn it in this evening.

5.2.7 Week 1.7

Plans

- Divide Project up into 9 distinct parts for tech review.
- Start working on tech review
- Follow up with Clark regarding NWCTB API Access

Problems

- Still don't have NWCTB API Access
- Beginning to notice we are not making progress on API Issue.

Progress

- Divided project into 9 parts 11/8
- worked on rough draft of tech review 11/12

Summary
This week we divided up the project into 9 different parts so each group members could each have three parts. We then picked the parts of application that we wanted to review. After that we then got working on the tech review rough draft.

5.2.8 Week 1.8

Plans

- finish tech review
- set up meeting with Clark and Sean for design document.
- Meet up and discuss tech review and peer review each others tech review

Problems

- NWCTB Still hasn't responded to our request for API access

Progress

- worked on tech review 11/13
- set up meeting with Clark and Sean on 11/13 for 11/21 for design document
- Preformed peer review in class on tech review 11/14
- Shengpei and I met up to review each others tech review 11/16

Summary This week we continued to work on our tech reviews. These documents are nearly complete. Some group members met to discuss the structure and content of the tech review. We will be ready to turn in our tech review next Tuesday. Additionally we set up a meeting with Clark and Sean to review the tech review.

5.2.9 Week 1.9

Plans

- Meet with Clark and Sean to go over the design document
- finalize drafts of tech review
- start design document

Problems

- Still haven't heard back from NWCTB regarding API access.

Progress

- Worked on tech review 11/20
- Did a final review with Shengpei 11/20
- We meet with Sean on 11/21 at 1:30 PM
- discussed API access with Sean at our meeting 11/21.
- finished Tech review 11/21
- Started work on design document 11/24

Summary
This week we finished up our tech reviews. I helped Shengpei finish up his tech review on Monday. I finished up my tech review Tuesday evening. Additionally, We also met with Seen Hammond to discuss our design document. We also started our design document on 11/24/2017.

5.2.10 Week 1.10

Plans

- Finish Design Document
- Start Progress Report
- Research alternatives to NWCTB

Problems

- Still don't have NWCTB API Access.

Progress

- Worked on Design Document 11/26 - 12/1
- Submitted rough draft of design document to Sean Hammond and Clark Seavert 11/30.
- Started working on progress report 12/1.
- Researched alternatives to NWCTB 12/1.

Summary
This week was busy week for our project. This week we worked on the design document. Currently we are nearly completed with the design document and will be able to finish up the document before EOD today. This week we also started working on the progress report. We outlined the report and will start working on the doc as soon as we are done with the design doc. We will also need to produce content for the presentation we are going to get together and give on Sunday evening.

5.2.11 Week 2.1

Plans

- Meet with group to set up iteration one of project development.
- Meet with Sean to set up git branch and discuss git workflow.
- set tasks for iteration 1.

Progress

- Forked github repo from AgBiz-Logic
- Set up a meeting with Sean to discuss project development.
- Started setting up tasks for iteration one on the git repo.
- Started working on the Wiki page with common help items for the project.

Problems

- Still haven't heard any thing from the NWCTB team regarding API access for the climate data.

Summary
This week we tried to set up a meeting with Sean to do some project planning and set up for iteration one of our project. However, Sean was unavailable this week so we set up a meeting for next week. We also got our git repo, forked from *AgBiz-Logic* set up. We started planning the first iteration of development on the project by adding issues to the github repository. We also started compiling some help pages on the Wiki of our repo.

5.2.12 Week 2.2

Plans

- Meet with Sean.
- Start Iteration One.

- Get UI elements implemented along with most of the front end functionality.
- Plan iterations 2 and 3.

Progress

- Set up meeting with Sean Hammond for Friday at 1 pm.
- Finished setting up iteration one tasks.
- Finished adding content to the help wiki on the github repository.
- Finally defined Climate Data API Access.
- Set a Weekly status meeting time to meet with the group. We plan to meet every week at one pm.

Problems

- API Access is less than ideal and will require more work than we were planning on but is still better than having to write our own service from scratch.
- Finding time to meet up as a group has been more challenging that I had anticipated.

Summary
This week we didn't get much development work done on our project like we had planed on. However, we did do some set up work. we finished setting up the github repository and finished laying out tasks on our story board. We also started defining what tasks we'd like to have in future iterations of our project. Additionally, we finally know what our API access to the climate data looks like. this will allow us to get the data we will need to plot. However, this will also require much more work that we had planned on and may set us back a bit in terms of our project schedule. That being said we worked in some flex time in to our schedule so we should be able to make it work.

5.2.13 Week 2.3

Plans

- Create proof of concept script for connecting to the database and getting data.
- start working on front end changes.
- Update design document and requirement document.
- Meet with Sean for status update at 1pm on Friday.

Progress

- Started working on concept script.
- Managed to gt dev environment set up instructions completed.
- Installed netcdf.
- Created example script for getting climate data from the thredds database. However we get some errors on certain reads.
- Updated requirements document.

Problems

- We had a hard time getting NETCDF4 to install. We ended up using anaconda however we are guessing Sean doesn't want to use Anaconda and will want us to produce an install script.

Summary
This week was a primarily a week of setup. we spent most of our time trying to get the dev environment set up along with installing NETCDF4 and its dependencies. This week we did find a way to install NETCDF4 using anaconda. However, we anticipate that we will be required to find a better way to install it. In the mean time this will allow us to develop a concept script. We also managed to the development environment for AgBiz-Logic set up. This took us more time that we had anticipated but wasn't as difficult as we thought it might be. This week we also made some updates to the requirements document to reflect the changes to the climate data API.

5.2.14 week 2.4

Plans

- Start working on the front end of the application.
- Refine the proof of concept to be more dynamic.
- Write script to install netcdf and dependencies.
- Start working on backend changes.
- Update documents.

Progress

- Started working on refining proof of concept script to search for points if the point we asked for doesn't have data also added more advanced bounds checking.
- Determined that NETCDF4 is having issues reading in blocks for chunk three.

Problems

- requests past index 435 on latitude cause a runtime error.

Summary
This week was mostly focused on working on the proof of concept scrip that will be used later to access the data from the thredds server. This week we discovered that the NETCDF4 library throws errors on any lat index greater than 435. We also discovered through the database administrator that this is the boundary between chunk two and chunk three of the file we are trying to read. We think that the NETCDF4 library may have a bug in it. Regardless we are going to need to find a work around moving forward. Shane and Thomas also got together on Saturday and started working on front end changes.

5.2.15 week 2.5

Plans

- Work on front end changes.
- Follow up with NETCDF4 developers about potential bug.
- Continue to work on concept script to see if we can tease out the runtime error.
- Finish NETCDF5 install script.
- Research other potential options other than python or netcdf4 for reading in data from the serer.

Progress

- Followed up with netcdf4 people.
- Shane finished the netcdf4 install script.
- Researched alternatives to netcdf4 We can write a c program that will to the same thing. There are a few other libraries for reading data via opendap.
- made progress on frontend changes.

Problems

- Issues with netcdf4 library.
- netcdf4 developers will not fix unless I can produce a self contained example of the read failing.
- We think that netcdf4 dependencies may not be installed correctly.

Summary
This week we made progress on the front end development and installing the dependencies for netcdf. However, we've run into some issues with netcdf. We think we maybe able to fix it by installing the dependencies for netcdf from source with certain flags enabled but we aren't totally sure on that. We also started work on setting up the end point where the API will live. The plan for now is to have it serve mock data as to enable us to continue with the rest of the development work without getting behind.

5.2.16 week 2.6

Plans

- Finish development of the charts page.
- Set up API to mock data.
- Figure out work around for netCDF problems.
- write the midterm progress report.
- make the midterm progress presentation.
- finish the poster rough draft.

Progress

- Finished Development on the charts page.
- Finished the midterm progress report.
- Made the midterm progress report presentation.
- Finished the Poster rough draft.
- Found a work around for NETCDF4 issues.

Problems

- Created some bugs by introducing short term climate scenarios.

  There are many ways we can fix this problem we will need to discuss with Sean how he wants this solved.

Summary
This week was a busy week. This week we accomplished most of the front end development required by this project in the maps page. However, we also introduced some bugs. Mostly we created and issue where if you have multiple budgets its not possible to tell what page you need to redirect to once you save your budget. There are many ways we can solve this so I want to ask Sean how he thinks the best way to go about this is. This week we also made our expo poster along with creating the midterm progress report document and presentation. Additionally, Shengpei found a work around to the NETCDF problems we've been having.

5.2.17 week 2.7

Plans

- Fix bug with short term climate scenario where rap around for multiple budgets doesn't work.
- Translate Shengpei's concept script from python 3.6 to 2.7.
- Create end point to host climate API.
- Serve static data from the end point to the front end until API is set up to get dynamic data.
- Integrate Shengpei's API into the end piont.

Progress

- Fixed Bug with rapping around for multiple budgets.
- Shengpei translated script into python 2.7.
- Shane worked on getting end point setup.

Problems

- Shengpei's script needs to refined.
- Inexperience with Django.

Summary
This week we continued to work on the frontend of the application. We made progress on getting some of the bugs fixed on the front end. Mainly, Now when you use multiple budgets the application correctly redirects to the correct page once reaching the end of the work flow. Shengpei also managed to get his script ported over from python 3.6 to python 2.7. Shane worked on setting up the endpoint that will evenutally host the climate data API. However, we ran into a few issues with this due to inexperience with Django.

5.2.18 week 2.8

Plans

- Create Unit tests for the Data-Impact page.
- update existing unit test to reflect changes in the application.
- Get end point up and running.
- Refine Shengpei's script so it returns data we can consume.

Progress

- Updated existing front end tests.
- Started working on Unit tests for the front end.
- started working on changes to API to make the data consumable.

Problems

- Having issues with mocking the services on the front end correctly.

Summary
This week we worked on setting up the back end to serve the data to the front end. We also started working on the front end unit tests. Additionally Shengpei started working on updating the API. It's looking increasingly more likely that we can have a beta next week if not early in week 10.

5.2.19 week 2.9

Plans

- Finish Front end testing.
- Get climate data endpoint up and running.
- Finish updates to the climate API.
- Set up front end to use endpoint to dynamically get data.

Progress

- Finished front end testing.
- Finished setting up end point.
- Finished refining the API.

Problems
We had no major problems during this week of development.

Summary
This week we continued to make progress on the development of our application. I finished up the front end unit tests and they now all pass. Shane set up the end point and has it ready to consume the API. Shengpei finished making the updates to the API so that the front end can easily consume the data. All that is left to do is to put the pieces together and fix any bugs we've created in the process. Next week we should be able to put the pieces together and produce a fully functioning beta release.

5.2.20 week 2.10

Plans

- Create Beta Release.
- Start Testing and bug fix phase of the project.
- Write and present final progress report.
- Integrate API into end point to serve data.
- set up front end to call the end point to get the dynamically generated data.

Progress

- Shane integrated the API in to the end point so our API is now serving dynamically generated data.
- Set up front end to call the end point and get the dynamically generated data. With the completion of this item we now have a fully functional beta release.
- Created progress report template.
- Meet with Sean to demo beta release.

Problems

- Need to write some code to deal with when the Climate data server is under maintenance.

Summary
This week we took all the pieces that we already had done and put them together. We found a few issues with the climate API and fixed them. Shane integrated the climate data end point with the climate data API. I then used that end point to get dynamically generated data on the front end. Thus we now have a fully functional beta.

5.2.21 Week 3.1

Plans

- Plan out term so we have a road map for the term.
- Set up some issues in github for our document projects.
- Set meeting with TA.
- Set meeting with Sean.

Progress

- Set issues in github for document projects.
- Tried to set meeting time with TA but TA never reasoned.
- Set up meeting time with Sean Hammond for Friday.

Problems

- Every one was sick this week so we weren't able to meet up and work on the project this week.
- TA did not respond to emails regarding setting up a meeting time.

Summary
This week our team decided to take it a bit easier as we don't have tons of work left to do and every one in our group is sick. So we didn't meet up and do a ton work this week but we did do some planning for our document projects along with creating some issues in github. Next week we will meet up and come up with a plan for finishing this project by the middle of the term.

5.2.22 Week 3.2

Plans

- Start working on QA.
- Finish up last development issues.
- Start adhoc testing.

Progress

- Stared working on Front end tests.
- Worked on diagnosis for issues with script to get data.
- Started researching alternative solutions for getting data.

Problems

- Currently our API isn't returning any data and is hanging indefinitely. I'm not sure if the URL changed or if something changed about the was data access works but currently our method of accessing data does not work.
- Found that our current solution isn't stable and isn't going to work out. So we need to find a new solution for getting the data from the Threadds server.

Summary
This week I went to do some testing and discovered that the climate API wasn't working. I started looking into why it wasn't working correctly and we also met as a group to discuss why the API is hanging. As a group we determined that this solution though clever is not stable. We will need to create a new solution from scratch probably from matlab as this is what the server admin recommended. The idea is that we will use the matlab scrip to get the data. We will call this script from a python script. This is possible because matlab provides extensive support for python.

5.2.23 Week 3.3

Plans

- Write Matlab API or figure out alternative wat to get data from database.
- If we solve data issues start writing tests for backend code.
- Add hoc testing.

Progress

- Started working on alternate data API.
- Found that MatLab was not going to work out as it was going to be hard to virtualize along with the fact that we can't export the interpreter.
- Tried R implimentation and discovered that R has the same problem as the python version.
- Meet with Sean to come up with a better solution for Accessing the data. We decied we will have to produce a chron job that gets the data every month and we will then spin that data into a mongodb database.

Problems

- Currently our API doesn't return and hangs indefinitely. We were previously accessing the data using a URL method. We've tested this method and determined its the database we are connecting to that is hanging.
- The underlying C library for the NETCDF library we are using has a bug in it or the server we are connecting to is set up incorrectly. Either way we are not going to be able to depend on this library to read files over the network.

Summary
This week we started looking for a solution to our data API problems. We discovered that matlab was not going to be a viable solution as it is impossible to virtualize along with the fact that exporting the interpreter is impossible. We also tried an R version of the same thing and were able to get the up and running but we get the same errors as the python library. In the end we are stuck downloading the whole file and storing it on the local server every month.

5.2.24 Week 3.4

Plans

- Meet to work on and discuss solution to problems getting data.
- create script to download data and place data in a database.
- Set up Django to use named database for short term data.
- Test this solution using a container.

Progress

- Started working on creating database script.
- Decided to abandon the idea of using DB all together and decided to use mount bind to share directory between the os and the container.
- Started working on proof of concept for mount bind.

Problems

- Having issues getting docker image to run using rocket.

Summary
This week we entertained the solution of using mongodb to store our climate database. However, after some research and thinking about the problem we decided that the best solution was really just to use a mount bind to share a directory between the os and container. We also started working on getting a proof of concept up for this solution.

We have run into some issues getting rkt to run a docker image.

5.2.25 Week 3.5

Plans

- Create Concept solution showing that we can bind mount on a native OS file in the rocket container.
- Produce API that uses local version of the netcdf file.
- start working on testing if we can get the API up and running.
- Update Poster and Submit.
- Complete midterm progress report.

Progress

- On sunday i produced a proof of concept showing that a mount bind works the way I expected it too.
- produced API that uses local version of the netcdf file.
- Created template for midterm progress report doc.
- Worked on midterm progress report and finished it.
- Worked with group to record progress report.
- Reimplimented API.
- Set up api to fail gracefully.

Problems

- To much work and not enough time.

Summary
This week we made alot of progress on the API. Most of this week development efforts were directed at the API. We managed to rewrite the API so that it uses a local netcdf file containing the climate data. Additionally the location of this file is provided through a configuration file so that it can be set differently for prod vs dev. The API has also been set to handle if the file doesn't exist with out throwing exception's. Finally, we will also set up the API to deal with searching the data when it finds a point that does not have a value. If no data can be found with in the nearest 99 locations then an error will be returned.

5.2.26 Week 3.6

Plans

- Start working on QA and testing.
- Bug fixes.
- Layout final schedule.
- Start Document Updates.

Progress

- Came up with final time line and schedule for project.
- Met with Sean to discuss final hand off date for the project.
- Started working on backend unit test along with updating front end unit tests.

Problems
No problems this week.

Summary
This week was a busy week for all of us as we all had midterms and other projects due. As such we did not spend very much time working on the project this week. Most of the work we got done this week was planning work. However, we did start working on unit tests for the API. I anticipate we will be able to finish up development and testing next week and will be able to hand off the project next week.

5.2.27 Week 3.7

Plans

- Expo.
- Finish testing.

Progress

- Finished updating front end unit tests.
- Attended Expo.
- Changed minor wording changes on chart page.

Problems

- Backend testing not working how we had expected.

Summary
This week the primary focus was on expo. The goal was to show up tot expo and inform people about the work we have been doing for the last several terms. We also spend some time working on some unit tests for the front end and the backend. However, we have been struggling to get the backend unit tests setup and working. It should also be noted that we spend some time updating wordage on the charts page in preparation for expo.

5.2.28 Week 3.8

Plans

- Fix workding and UI layout issues provided by Clark.
- Bug Fixes.
- Unit testing.
- Ad Hoc testing.
- Update requirements Document.
- Update Design Document.

Progress

- Updated Requirements Document.
- Finished Bug fixes.
- Made updates requested by Sean and Clarck.
- Preformed Ad Hoc testing.
- Rebased code repository to master.
- Handed code off to our client.

Problems
No problems this week.

Summary
This week we finished up our last cycle of development. This was a busy week of finishing up last minute changes and bug fixes. We started off the week by making some changes requested by Sean and Clark. We also made sure to update our tests and write some more unit tests for the API. We also fixed some bugs found through Ad Hoc testing. Finally, we made some updates to the wording and order of UI elements requested by Clark and Sean. We also rebased the code to master and passed if off to Sean. Currently all that is left to finish this project are the final document and final progress report.

5.2.29 Week 3.9

Plans

- Create and finish final document.
- Provide support for any bugs found in our project.
- Do the final presentation.

Progress

- Started final Document.
- scheduled final progress report presentation.

Problems
no problems this week.

Summary
The focus of this week was to finish up as much documentation as possible. As such we have worked on the final document and also produced the final progress report presentation. We also provided support for the product we've handed off to our client. Though we have handed our code off to our client we've agreed to provide bug support if they find any bugs.

5.2.30 Week 3.10

Plans

- Finish Final Presentation.
- Finish Final Report.
- Fill out peer evaluation.

Progress

- Filmed Final progress report.
- Started the final document.

Problems
No problems this week.

Summary
The primary focus of this week was to work on the final progress presentation and the final progress report document. Given that this document is rather large this took a fair bit of our time this week. We also spent some time working on and recording the final progress report.

5.3 Shengpei Yuan

5.3.1 Week 1.1

- I have thought through the project proposal and chosen top 5 projects I like.

5.3.2 Week 1.2

- On Tuesday, I have reviewed my project and thought about it.
- On Wednesday, I have contacted my group mate, and we have sent the email to the client.
- On Thursday, I have meet with my group mate, and we have talked the project and set up the time when we will meet the client.
- On Friday, I began to start write my problem statement.

5.3.3 Week 1.3

- On Monday, I have finished the problem statement for project.
- On Tuesday, our team have met our client.
- On Wednesday, I have modified my draft of problem statement.
- On Thursday, our team have met our technical guidance.
- On Friday, I modified some detail of my draft of problem statement.

5.3.4 Week 1.4

- On Wednesday, I have add some detail on my problem statement.
- On Thursday, my group have met together to modify the group problem statement, and I have signed NDA.
- On Friday, I have met TA and we begin to work on requirement draft.

5.3.5 Week 1.5

- On Tuesday, our group have met with Sean to specify the requirement.
- On Thursday, I have begin to draft requirement.

5.3.6 Week 1.6

- On Tuesday, our group have met our client to discuss what we have written for the requirement doc, and revised the requirement doc.
- On Wednesday, I have finished the references on requirement doc.
- On Thursday, I have changed some flaws on requirement doc.

5.3.7 Week 1.7

- On Tuesday, I begun to work on technology review.
- On Wednesday, I have researched options and added more detail on technology review.
- On Thursday, I revised some detail on technology review.
- On Sunday, I have finished my technology review.

5.3.8 Week 1.8

- On Tuesday, I have begun work on final draft Tech Review.
- On Thursday, my group mate have help me to correct some grammar issues in my Tech Review, and I have revised it.
- On Saturday, I have done more research for my Tech Review, and added more Infor.
- On Sunday, I have finished my Tech Review.

5.3.9 Week 1.9

- On Tuesday, I have begun work on design document and our group have met with our client to get more information for design document.
- On Saturday, I have searched more information for the design document.

5.3.10 Week 1.10

- On Tuesday, I have written part of design document.
- On Wednesday, I have added more information on design document.

5.3.11 Week 2.1

Plans

- Meet with group to set up iteration one of project development.
- Meet with Sean to set up git branch and discuss git workflow.

Progress

- Forked github repo from AgBiz-Logic
- Set up a meeting with Sean to discuss project development.
- Started setting up tasks for iteration one on the git repo.
- Started working on the Wiki page with common help items for the project.

Problems

- Still haven't heard any thing from the NWCTB team regarding API access for the climate data.

5.3.12 Week 2.2

Plans

- Meet with Sean.

Progress

- Set up software problem.
- Set a Weekly status meeting time to meet with the group. We plan to meet every week at one pm.

Problems

- API Access is a big problem we got.

5.3.13 Week 2.3

Plans

- Update design document and requirement document.
- Meet with Sean for status update at 1pm on Friday.

Progress

- Managed to gt dev environment set up instructions completed.
- Installed NETCDF4

Problems

- We had a hard time getting NETCDF4 to install.

5.3.14 week 2.4

Plans

- Start working on the front end of the application.
- Refine the proof of concept to be more dynamic.
- Write script to install netcdf and dependencies.
- Start working on backend changes.
- Update documents.

Progress

- Started working on refining proof of concept script to search for points if the point we asked for doesn't have data also added more advanced bounds checking.
- Determined that NETCDF4 is having issues reading in blocks for chunk three.

Problems

- requests past index 435 on latitude cause a runtime error.

5.3.15 week 2.5

Plans

- Work on front end changes.
- Continue to work on concept script to see if we can tease out the runtime error.
- Finish NETCDF5 install script.
- Research other potential options other than python or netcdf4 for reading in data from the serer.

Progress

- made progress on frontend changes.

Problems

- Issues with netcdf4 library.
- We think that netcdf4 dependencies may not be installed correctly.

5.3.16 week 2.6

Plans

- Finish development of the charts page.
- Set up API to mock data.
- Figure out work around for netCDF problems.
- write the midterm progress report.
- make the midterm progress presentation.
- finish the poster rough draft.

Progress

- Finished Development on the charts page.
- Finished the midterm progress report.
- Made the midterm progress report presentation.
- Finished the Poster rough draft.
- Found a work around for NETCDF4 issues.

Problems

- Created some bugs by introducing short term climate scenarios.

5.3.17 week 2.7

Plans

- Debug API
- Working on frontend

Progress

- Make the API can mock data correctly

Problems

- Try to figure out the parameters of the website link that is helpful to get the data correctly

5.3.18 week 2.8

Plans

- Make API output all the individual results
- Parse results into their own arrays
- Create another array contains labels for the data.

Progress

- The API can output individual results in each date set

Problems

- Try to find a way which can split results in different array

5.3.19 week 2.9

Plans

- Split results in individual array
- Write some debug test for frontend

Progress

- Got four individual array for four different type data

Problems

- Make the month label for four type data in different array is complicated

5.3.20 week 2.10

Plans

- Write another array to label each month data
- Write the final progress report
- Work on final progress presentation

Progress

- Finish month label for four type data in different array

5.3.21 Week 3.1

- Plan for the testing.

5.3.22 Week 3.2

- Found bug on API.

5.3.23 Week 3.3

- Try to find a solution for API.

5.3.24 Week 3.4

- Found data mount solution for the API.

5.3.25 Week 3.5

- Finished Midterm presentation and report.

5.3.26 Week 3.6

- Begin back-end unit test.

5.3.27 Week 3.7

- Attended Expo.

5.3.28 Week 3.8

- Debug some package issues.

5.3.29 Week 3.9

- Finished Final presentation.

5.3.30 Week 3.10

- Finished Final Document and report.

6 Final Poster

Shown below is a smaller version of the poster we presented at Expo.

AgBiz-Logic: AgBizClimate

AgBizClimate is a submodule of the *AgBiz-Logic* suite of decision tools. This submodule specializes in assisting farmers and ranchers by supplying them with short and long term weather data which they can use to make financial decisions. More specifically, *AgBizClimate* provides precipitation and temperature information in a series of graphs associated to the location the users selected via latitude and longitude. Users are then able to input the percent they believe these factors will influence their yields , *AgBizClimate* will then return the results of the calculations based on the users to assist their decisions.

AgBiz-Logic

- **What is AgBiz-Logic?**
  - *AgBiz-Logic is a suite* of **economic, financial, and environmental decision tools** for businesses that grow, harvest, package, add value, and sell agricultural products.
- **Where does the Financial Data come from?**
  - *AgBiz-Logic* allows users to enter their own budgets or selects University budgets for varying livestock and crops to plan for the future with with *AgBiz-Logic* tools.
- **What are the actual components that make up *AgBiz-Logic*?**
  - *AgBiz-Logic* consists of five distinct submodules including *AgBizProfit*, *AgBizLease*, *AgBizFinance*, *AgBizEnvironment*, and *AgBizClimate*.
- **What are the goals of *AgBiz-Logic*?**
  - *AgBiz-Logic* seeks to assist farmers and ranchers by supplying them with management tools. Additionally, the data produced by these modules will be used by researches to study the effects of climate change on farming and ranching.

Oregon State University

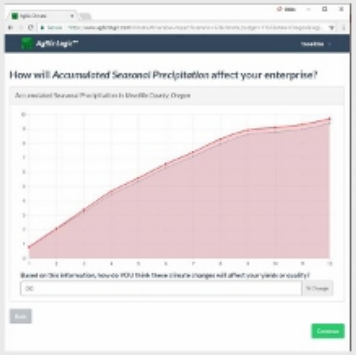


Figure 1: A screenshot of the *AgBiz-Logic* Short term Climate tool. This Screen Shot shows the plotting tool where the user can make choices about their crop based on the plot.

Figure 2: Shown Above is a use case diagram for the *AgBizClimate* tool.

DESCRIPTION OF THE PROBLEM

Climate change is a huge problem facing many farmers and ranchers. As the climate changes farmers will need to adapt and modify what they grow and how they grow it. To solve this problem will require intelligent data driven solutions.

This project seeks to arm farmers and rancher with a suit of climate tools to help mitigate the effects of climate change. Currently there is already a long term climate tool that helps make longer term predictions. The focus of this project is providing seasonal guidance to farmers and ranches, ultimately helping farmers to be more efficient and more profitable.

API DESIGN

- REST API that provides the climate data for the short term projections.
- Data: Downscaled version of NOAA's NMME dataset hosted by University Of Idaho.
- Data files are stored on the production server. The production sever updates the data at the same date every month.
- API Connects Reads Data From the Production server via a Bind Mount to a directory on the production server.
- API Uses latitude and longitude to determine the users location.
- Gets the data for the nearest location within a radius of 20 miles if no data is available for a certain location.

AgBizClimate Workflow

This workflow assumes that the users is already logged on and authenticated via the *AgBizLogic* Module and has navigated to the Climate Manager.

1. First, the user select the budgets that they want to analyze over the next 7 months.
2. Then they will then be redirected to the region selection page. At this page they will select the location by county state.
3. Next, The user selects the short term scenario type. Selecting short term climate Scenario will redirect the user to the charts page where the climate data for their location will be shown on a graph. If the user selects long term climate scenario the user will be redirected to a page where they will select which variables they would like to analyze.
4. Next, the user will be directed chart page where they may enter adjustments to the yield for each budget that they selected based on the climate variables shown.
5. The user will then be directed to the budget page where adjustments to prices, inputs, and other budget parameters may be made for selected budgets.
6. Finally the user will be directed to a climate summary page summarizing how the climate projections will affect their budget.

The Team

Thomas Noelcke

Shane Barrantes

Shengpie Yuan

7 Project Documentation

7.1 Theory of Operation

The AgBizClimate work flow begins when the use creates a new account. Once the user's account is created they are sent to a landing page where they can navigate to the budget selection page and either upload their own budgets or select pre-loaded university budgets. Once users have selected the budgets they would like to use they navigate to the region selection page where they can select state, county, and climate data type. We worked primarily on the short term climate data type so we will assume the user selected this option. The user is then brought to a charts page where they can view graphical data corresponding to the state and county they selected in the previous page. The variables these charts graph are precipitation, precipitation, temperature, and temperature anomaly. Users can then input the percent change they believe the graphs will affect their budget yields. They are then brought to a final budgets page where they can alter their budget. Finally users are brought to a summary pgae where they can export their new budget. This work flow process is iterative so users can go back and alters the percent change, or add additional budgets.

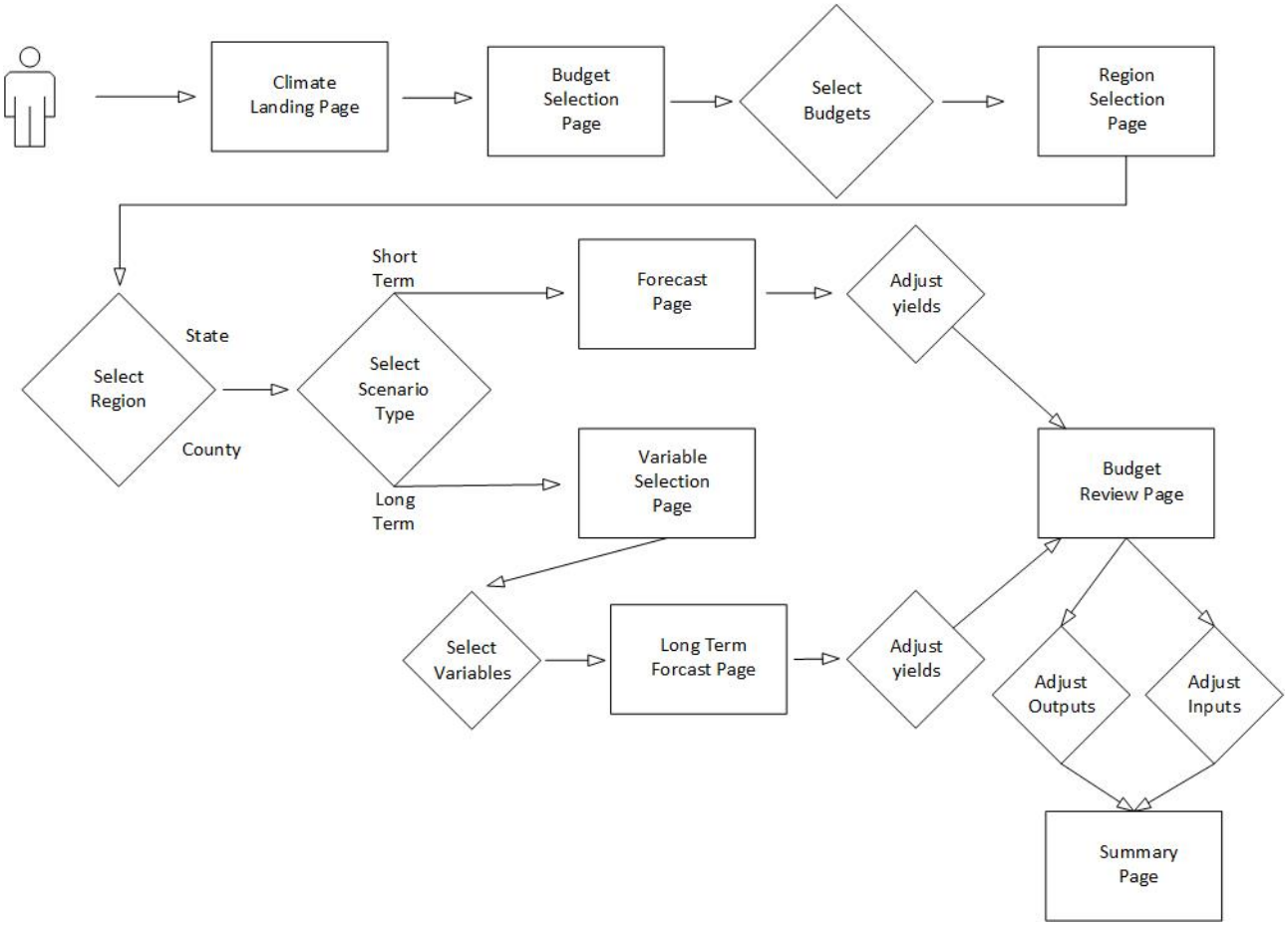


Figure 1: A sample use case diagram for the AgBizClimate project.

7.2 Configuration Steps

In this section we will include information on how to setup and configure our project for Mac OS X and Ubuntu.

7.2.1 Mac

- Install git: ruby -e "$(curl -fsSL https://raw.githubusercontent.com/Homebrew/install/master/install)"
  brew doctor
  brew install git

- Clone our project repo: git clone https://github.com/tnoelcke/AgBiz-Logic.git
- Install Python: brew install python
- Install Python-pip: sudo easy_install pip
- Install Nodejs: brew install node
- Move to the directory /AgBiz-Logic/
- Install Django dependencies: pip install -r requirements.txt
- Install node modules: npm install
- Run entrypoint script: ./entrypoint.sh

7.2.2 Ubuntu

- Install git: sudo apt-get install -y git-core
- Clone our project repo: git clone https://github.com/tnoelcke/AgBiz-Logic.git
- Install Python: sudo apt-get install -y python
- Install Python-pip: sudo apt-get install -y python-pip
- Install Nodejs: curl -sL https://deb.nodesource.com/setup_8.x — sudo -E bash -
  sudo apt-get install -y nodejs
- Move to the directory /AgBiz-Logic/
- Install Django dependencies: pip install -r requirements.txt
- Install node modules: npm install
- Run entrypoint script: ./entrypoint.sh

7.3 Running the project

Once the project and its dependencies have been successfully installed using the steps in the "Configuration Steps" section, the server can be restarted at any time by running the entrypoint.sh script or using the following command "python app/manage.py run server". We have tested our application on Mac OS X and Ubuntu.

7.4 Testing the project

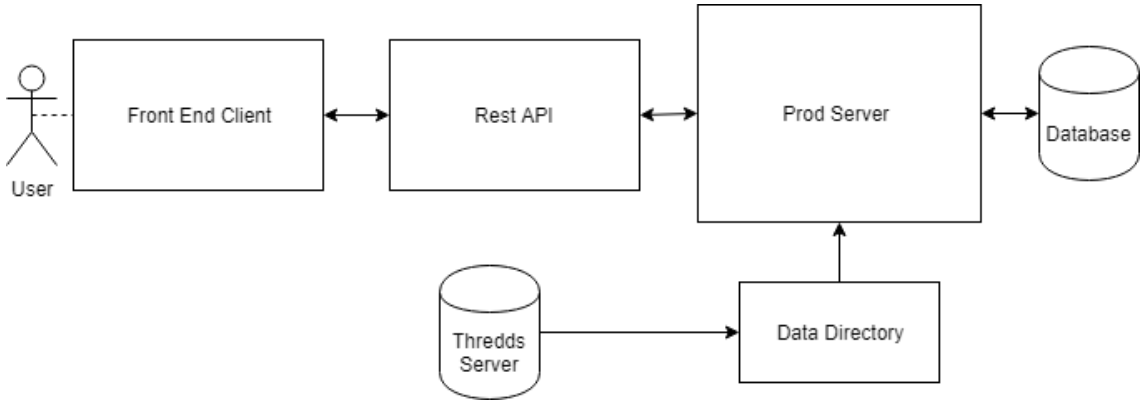
Once the project and its dependencies have been successfully installed using the steps in the "Configuration Steps" section the project can be easily tested by using the command "npm test".

7.5 Project Design

7.5.1 Architecture

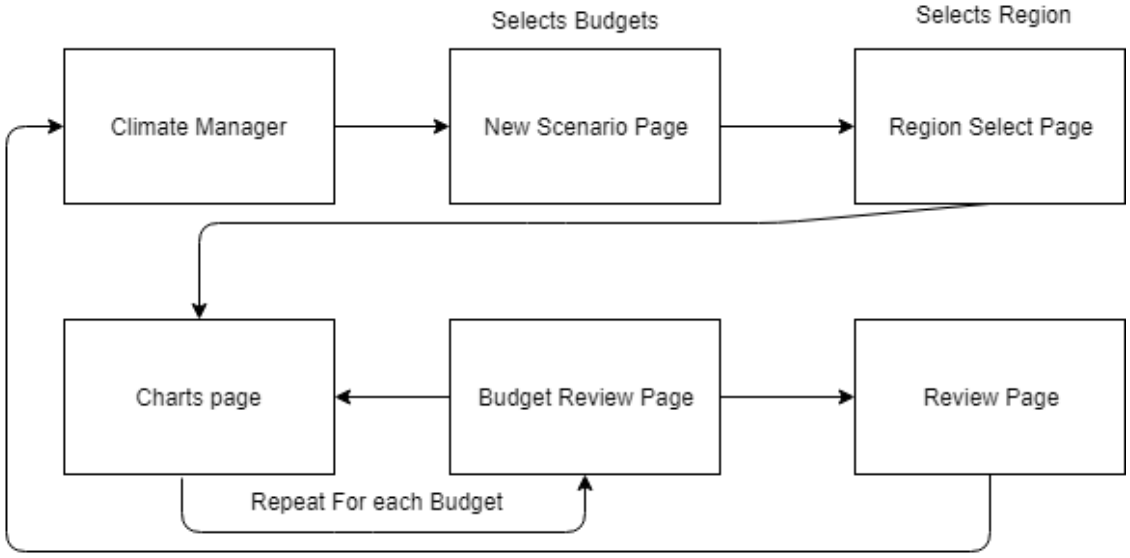
The AgBizCliamte application architecture is actually fairly simple. There are essentially six parts. The first of which is the Front end client. This client is a thick client that handles rendering the UI, interacting with the user and interacting with the back end API. The next part is the back end API. The back end api handles, processing

data, serving data and interfacing with the database. The rest API also interfaces with the production server to get data from the data directory. Next there is the production server. The production server is the hosting environment for the Rest API as well as for the data base and the the data directory. The server is also responsible for running a chron job that updates the data directory once a month. The thredds server is a third party server that we connect with to get the short term climate data that we serve to our users. The final part of the application is the database. The database handles storing and serving user data that we want to persist. This includes budgets and climate projections.



7.5.2 Front End

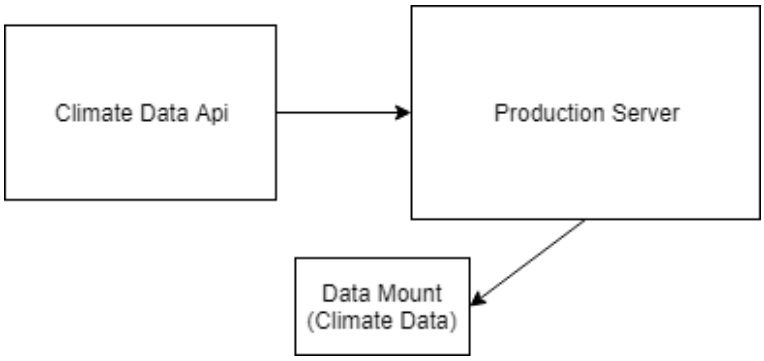
The front end of this application is a thick client. This thick client is responsible for rendering the user interface, making requests to the api and routing the front end of the application. In our application the front end is actually quite large as there are other tools other than the short term climate tool. However, we are only going to talk about the short term climate tool as this is the part of the project that we worked on. Shown below is a diagram of how the different pages relate to each other. I have omitted the long term climate pages for simplicity.



At the climate manager the user can review and delete previous scenarios. Form the climate manger they can also create new climate scenarios. This will direct them to the new scenario page where the user will enter notes about the scenario. Additionally, The user will add budgets up to five budgets. Next they will be directed to the region select page where the user will select their location. Next they will be directed to the charts page where they will enter a percentage yield based on the climate data. Next they will be directed to the budget review page where they can make any adjustments to their budget including yields and inputs. They will then repeat the cycle between the charts page and budget review page for each budget. Finally they will be directed the review page which provides a summery of what was done in the simulation along with a visual representation.

7.5.3 Back End

The AgBizClimate back end consists of a Django APIView that's used to route API requests. This APIView acts as an API endpoint and takes the state and country as arguments in the following format climate/climate_data/state/county. The state and county information is then used to look up a corresponding latitude and longitude using the Python package geopy. Once the latitude and longitude are found found The data API then does a look up in the climate_data.nc file in the data mount on the production server to find requested climate data. If the data is not found at that exact latitude and longitude the API then does a search in the surrounding area until valid data is found. Of course, there is a limited distance from the initial request in which the API will actually return data, otherwise an Error is thrown.



7.6 Essential Documents

7.7 Recommended Technical Resources

7.7.1 Helpful web pages

Django-rest-framework.org The django-rest-framework.org site was an essential resources which covered in-depth information on how to setup a django API view for our climate data API.

Angular JS: https://angularjs.org - We used this page as a reference for AngularJS syntax and design patterns.

Jasmine: https://jasmine.github.io/ - This page is a comprehensive reference on the Jasmine the front end JavaScript testing frame work. This page has documentation to do any thing you might want to do with jasmine.

netcdf4: http://unidata.github.io/netcdf4-python/ - This page contains a comprehensive guide to all things netcdf4. This reference contains everything including reading and writing netcdf4 files.

Rocket: https://coreos.com/rkt/docs/latest/ - This page contains the documentation for the visualization frame work rocket.

Stack Overflow It's also fair to mention that we used Stack Overflow for very specific problems that we couldn't find in documentation. This a good reference when we are facing a specific problem that was hard to find in the documentation.

7.7.2 Helpful people on campus

Seam Hammond Sean Hammond is our Clients technical adviser. When ever we ran into problems we weren't sure how to solve Sean was always our go to as he is very familiar with the project and the frame works used on the project.

Rob Hess Rob Hess is an instructor here on campus. He teaches web development, mobile development, cloud development and many other courses. During spring term Thomas and Shane were taking Rob's could development class where we were learning about vitalization. One of the problems we faced in this project was related to the best way to handle a vitalization problem at a design level. After coming up with a few ideas were able to run these by Rob and confirm that we were using best practice for our problem.

Graham Barber Graham is a coworker of Thomas's at CASS here on campus. One day we were stuck on a docker compatibility issue with rocket and Graham was very helpful in helping us work through this issue. Graham has had some experience with rocket and was able to point us to some documentation that solved our problem.

Zach Lerew Zach Lerew is another one of Thomas's coworkers at CASS. Zach Lerew has had some experience with Django through work done at CASS. As a result when faced with some Django problems we were able to ask Zach for some help when facing issues that the documentation couldn't answer.

8 Conclusions and Reflections

8.1 Shane Barrantes

8.1.1 Technical Information

Over the course of this project I learned a lot about the following technologies.

- **Django:** I learned how the front-end, back-end, and unit tests work as well as how to add Routes and API Routes.
- **AngularJS:** I learned basic front-end information about angularJS and got a little bit of insight to how it's often tested.
- **Rkt:** I learned the limitations of pulling down containers and how to implement bind mounts.
- **NetCDF:** I learned what NetCDF, it's installation process, how to interact with files, and fairly deep understanding of the functionality.
- **Github:** I learned how to work in a team environment, how to squash merge conflicts, and how to re-base.
- **DevOps:** I learned about the deployment process for our project and gained some software building insights.

8.1.2 Non-Technical Information

As far as non-technical information I learned how to frequently communicate with clients and how to work together in a team environment. I also learned how to work independently on a team project.

I don't think I learned a whole lot about project work that I hadn't already experienced in a work or internship environment. One thing that stands out is time management while working with a team that has very different active hours than you do. Another thing I learned was how to regularly communicate the work progress directly to the client since.

The project management skills I learned over the duration of this project mostly reside in inter-team communication and team scheduling. We frequently setup goals and deadlines that were in flux based on our term schedules, project state, and deadlines.

The most important things I learned about working in teams are communication and of how to divide the work based on skills, availability, and fairness.

If I could do this all over again I would look for an alternative solution for our client data API immediately. We took the client specification and tried our best to make it work, however this added months of work on tools and software that in the end were not used in the final product. If we were aware of our final solution sooner, we would have easily been able to do more testing and implement stretch goals.

8.2 Thomas Noelcke

8.2.1 Technical Information

In this section I will discuss the technical information that I learned through out working on this project. Below is a list technologies that I learned while working on this project.

- **Django:** I learned about how to set up and configure a django application. Additionally, I also learned how views and API views work in Django. I also learned about unit testing in Django.

- **AngularJS:** I learned how to implement a thick client in Angular js. This includes building the dom, routing, sending requests to the API and passing data from one view to another.

- **postgres SQL:** I learned a few of the differences between postgres and mysql.

- **NETCDF:** I learned what NETCDF files are and how to read in and interact with netcdf files. I also learned how to install and setup the netcdf package in python.

- **github:** I learned about managing projects via github using issues along with a project board in github. I also learned about the feature branch work flow process.

- **git:** I learned how to use git to rebase branches. I also learned how to use the command line client to view history and move the project thorugh different points in the history of the project.

- **Ubuntu:** I learned how to install and setup Ubuntu. I also learned how to set up and install projects on Ubuntu. I learned about the apt-get package manager and how to use to manage various different software packages.

- **Rocket:** I also learned about Rocket as a vitalization framework. Though this framework is much like Docker, I learned how to host docker images use rocket. I also learned how to mount a directory from to a docker image using rocket.

- **Devops:** In general I learned about devops quite a bit through this project. I learned about how applications are deployed and about the environment in which they are deployed.

8.2.2 Non-Technical Information

In this section I will discuss some of the things I learned from this project while working on this project that are not technical items. I will cover what I learned about, working on a team, working with a client, project management and what I would do differently if I were to it all again.

Working On a Team While working on this project I learned that when working on a team communication is key. I learned that communicating frequently and often is important. I also learned that it is especially import to communicate with your team about what your working on. This is important so your team can know what work is getting done so they are sure every thing is getting completed. This is also important because it also ensures that your team doesn't take a task that you are already working thus duplicating work.

I also learned about the value of pair programming over the course of this project. There were a few times when one of use would get stuck on something for long periods of time. When we eventually asked for help and hand some one look over our shoulder or take a look at the code, we often found the problem. In essence the lesson I learned is that four eyes are better than two and that two heads are better than one.

Project Management Through out this project I also learned a lot about project management. I learned how to set up a project board in git hub. I also learned about writing user stories and developing requirements for an application. Through out the course of this project I also learned how to lead a team on software development project.

The other thing I learned about project management through this project is working with a client. The big lesson I learned is again that communication is key. Communicating with your client is everything. I learned this lesson through positive experiences with our client due to our communication with our client. The main lesson being communicate early and offten.

What I would Do Differently If I could do this project over again there would be several things i would change. The first of which would be that I would have abandon trying to read netcdf files over the network. When I started to have stability problems I should have realized that netcdf was not going to work over the network for my application. I didn't see the warning signs early enough and wasted a bunch of time trying to figure out how to make a framework that was unstable work.

If I did this project again I would also be more effective at delegating work to my group mates. It is fair to say that I did more work on this project partially because I have more experience that my team mates but also in part because i was not effective in delegating tasks to my team mates. If I could do this project again I would have delegated tasks more effectively.

8.3 Shengpei Yuan

8.3.1 Technical Information

After finished project, I have learned several new skills.

- **Python:** I learned the difference between python 2.7 and 3, and used python to get data from climate website by modify parameter URL.
- **Django:** I learned unit tests, front-end and back-end.
- **NetCDF:** I learned how to install NetCDF and how it works with other files.

8.3.2 Non-Technical Information

For the non-technical information, I have learned how to work and cooperate with teammate, and how to communicate with client when meet problem. After overviewed the project, I have learned the way using past climate data to predict the future yield of corps and understood the value of past climate data. For the project management, I have understood that plan ahead and finish task on schedule are very important because a complete project is the primary goal for the project management. For the team work, I have learned that communication is very important for the team and how to keep communication with group member. If I could do the project again, I would like to find more climate data sources and build a better data API to ensure project can get data anytime.

9 Appendix

9.1 Appendix 1: Essential Code Listings

```
class ClimateDataView(APIView):
    """ API endpoint for ClimateData model.
    """
    authentication_classes = (SessionAuthentication, BasicAuthentication)
    permission_classes = (IsAuthenticated,)

    def get(self, request, format=None, **kwargs):
        #check that our local file for the climate data exits.
        if not os.path.isfile(settings.DATA_PATH):
            data = {
                "error": "Short term Climate Data Unavailable. No data plots will be displayed. If this continues to happen
            }
            return JsonResponse(data, status=500)



        #get the data
        data = []
        county = self.kwargs['county']
        state = self.kwargs['state']
        if not county or not state:
            data = {
                "error": "Please select a valid state and county."
                }
        geolocator = Nominatim()
        location = geolocator.geocode( county + " " + state)
        if(location == None):
            data = {
                "error": "Please select a valid state and county."
            }
        else:
            latitude = location[1][0]
            longitude = location[1][1]
            data = getData(latitude, longitude, settings.DATA_PATH)
        if data.has_key('error'):
            return JsonResponse(data, status=404)

        return JsonResponse(data)
```

Figure 2: This is the data APIView class which acts as an API endpoint to get climate data for the user.

```
def findData(latIndex, lonIndex, datahandle):
    interations = 5
    num = 0
    #if we are to close to the edge move the origin so that we don't over step our bounds.
    if latIndex + 5 > MAX_LAT:
        latIndex = latIndex - (MAX_LAT - latIndex)

    if lonIndex + 5 > MAX_LON:
        lonIndex = lonIndex - (MAX_LON - lonIndex)

    if latIndex - 5 <  0:
        latIndex = latIndex + (5 - latIndex)

    if lonIndex - 5 < 0:
        lonIndex = lonIndex + (5 - lonIndex)

    # preform grid search moving out from the origin.
    for i in range(num):
        for j in range(-i, i):
            if not math.isnan(datahandle[latIndex + i, lonIndex + j, 0]):
                return (latIndex + i,lonIndex + j)

            if not math.isnan(datahandle[latIndex - i,lonIndex + j, 0]):
                return(latIndex + i, lonIndex + j)

            if not math.isnan(datahandle[latIndex + j, lonIndex + i, 0]):
                return(latIndex + j, lonIndex + i)

            if not math.isnan(datahandle[latIndex + j, lonIndex - i, 0]):
                return(latIndex + j, lonIndex - i)
    return(-1, -1)
```

Figure 3: This is the data findData function in our ClimateDataView which utilizes a grid search to optimally find the latitude and longitude closest to the user entered state and county.

9.2 Appendix 2: Catch all

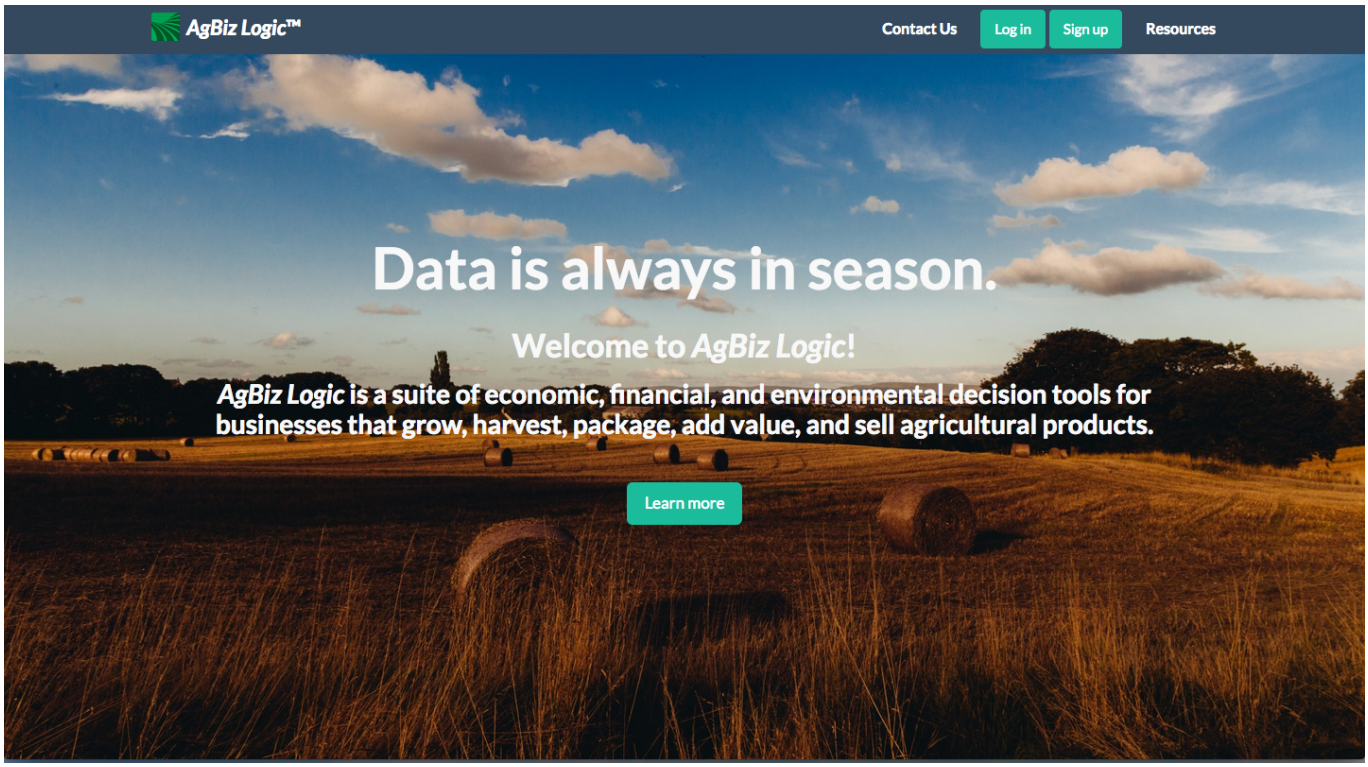


Figure 4: The landing page for AgBizLogic

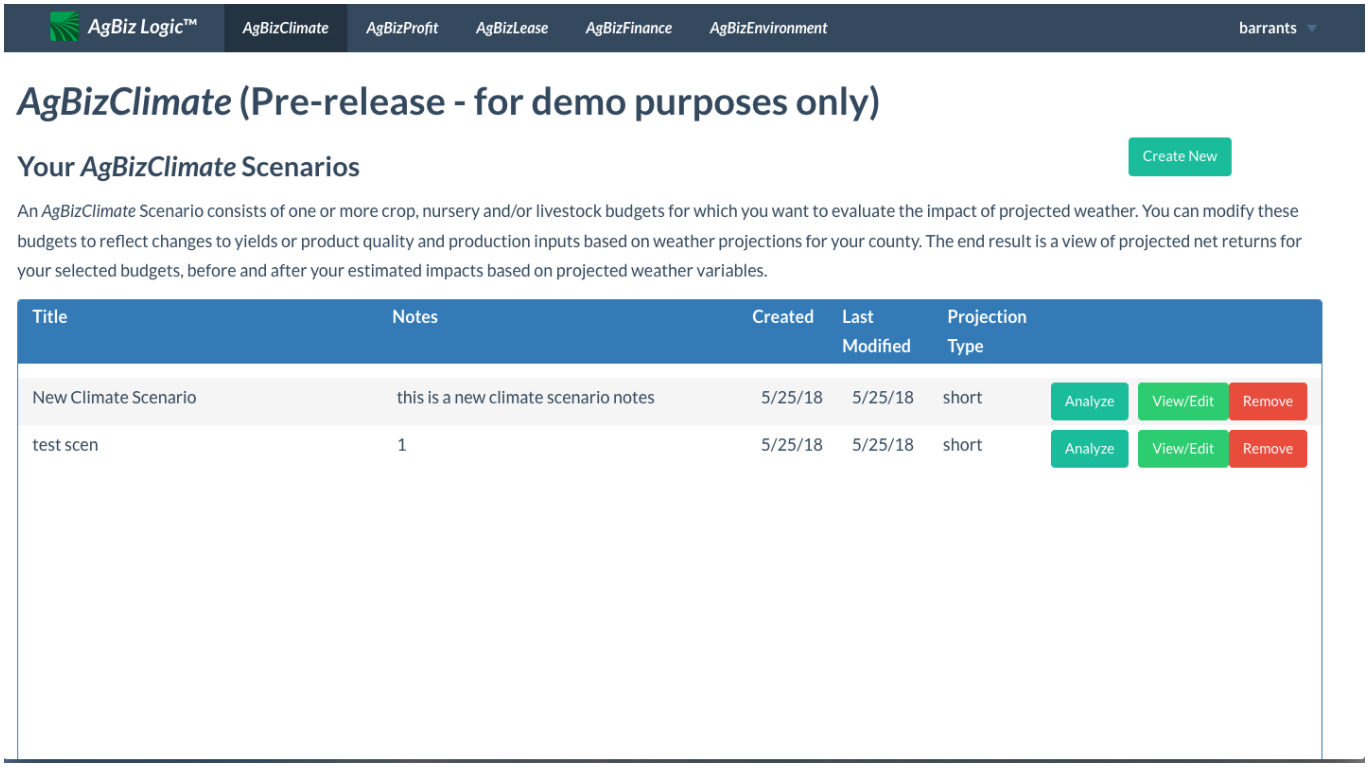


Figure 5: The scenarios page of AgBizClimate.

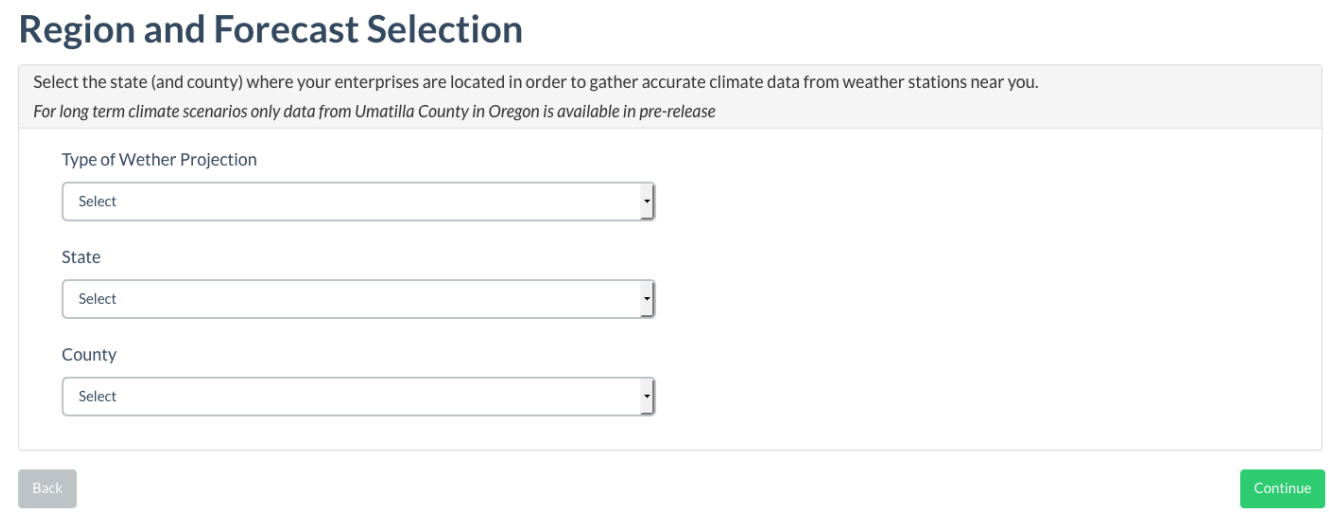


Figure 6: The region and climate selection page of AgBizClimate.

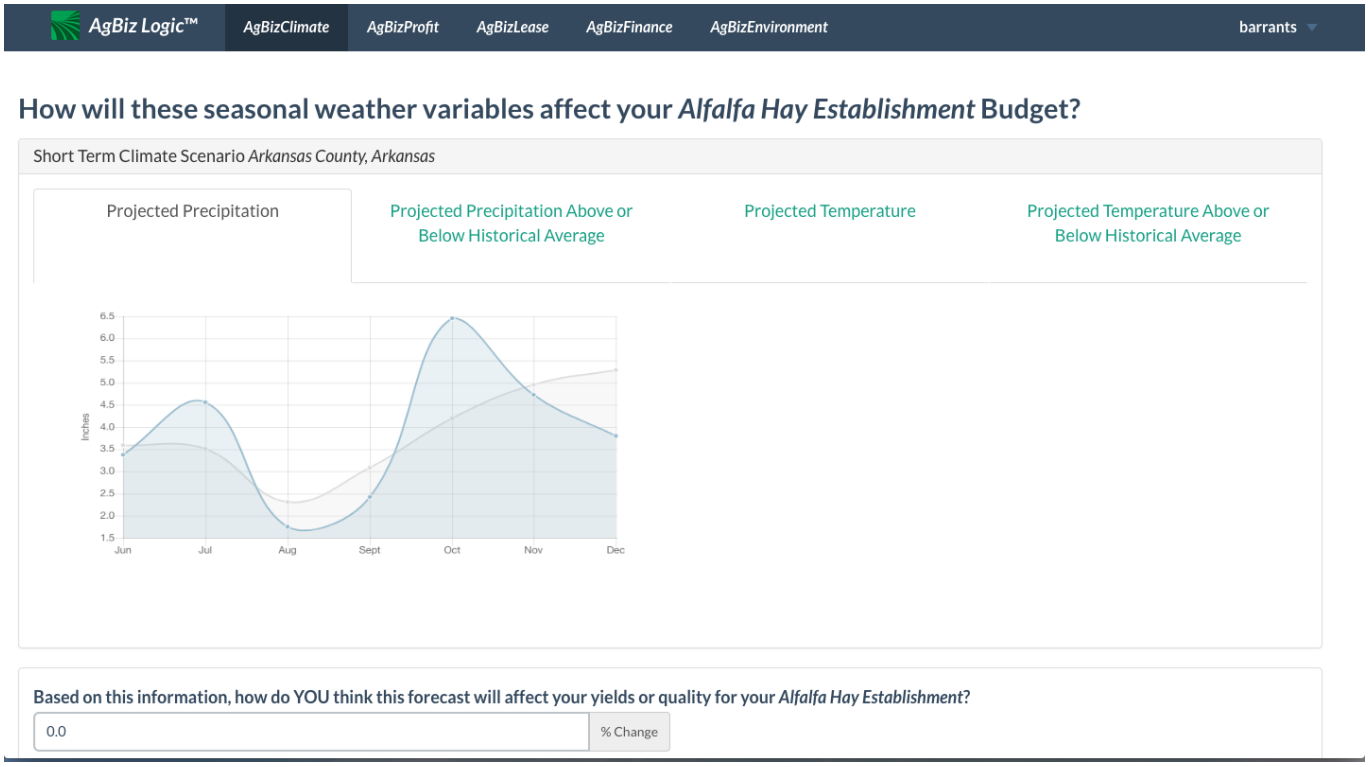


Figure 7: The charts page of AgBizClimate.

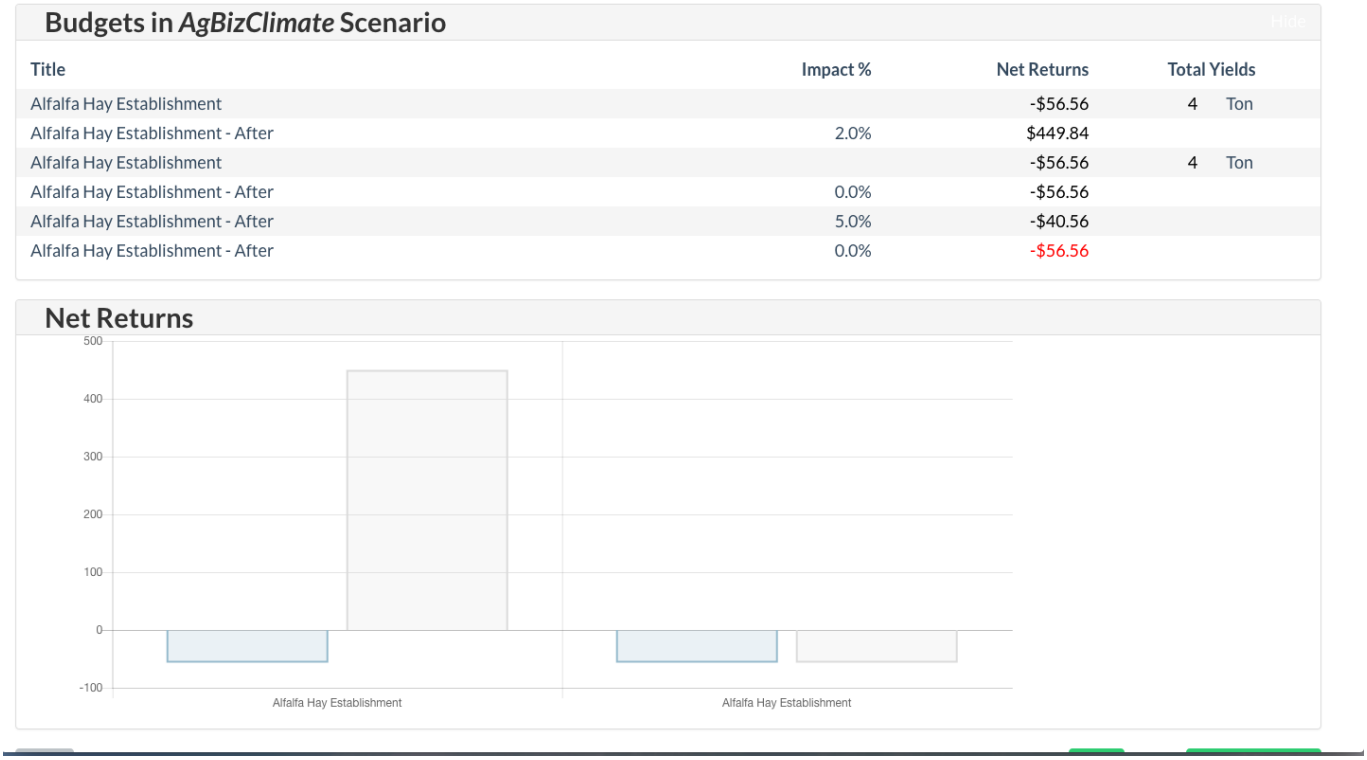


Figure 8: The long term climate scenario budget review page of AgBizClimate.