CS Capstone Software Design Document

December 1, 2017

Linking Seasonal Weather Data to AgBizClimate^{TM}

Prepared for

Oregon State University

Clark Seavert

Sean Hammond

Prepared by

Group26

Thomas Noelcke

Shane Barrantes

Shengpei Yuan

Abstract

This design document will cover the proposed design of the AgBizClimate^{TM}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. Then we will discus the design of the user interface. Finally, we will provide a requirements matrix which will show how each component fulfills the functional requirements.

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 this project will need to use. We will also break down each component and discuss the design of each component in detail. After that we will discuss the design of 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 implimentatino 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.

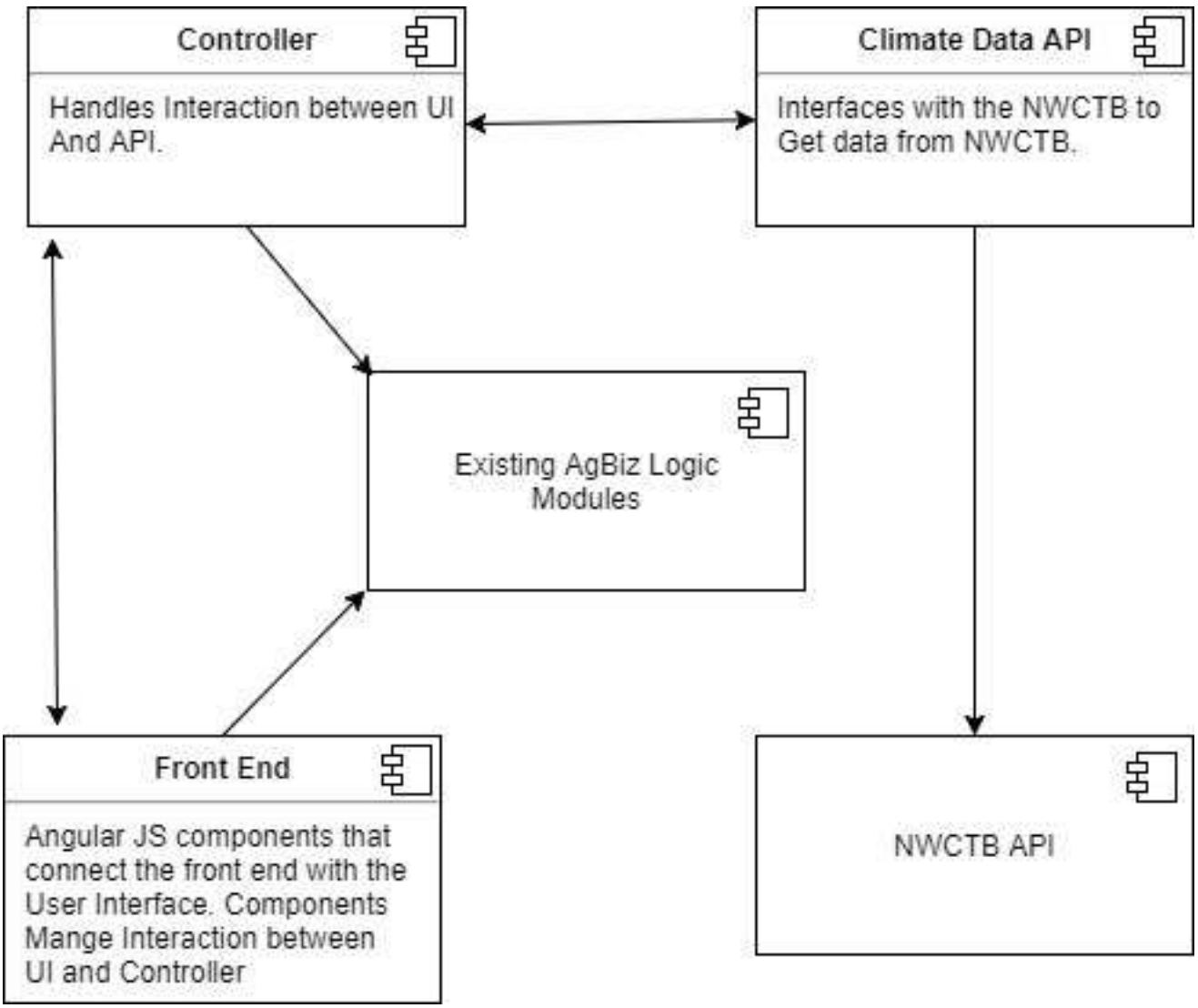

Figure 1: System Architecture Design for *AgBizClimate* project

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 this will happen 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 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 create these modules 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 it 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.

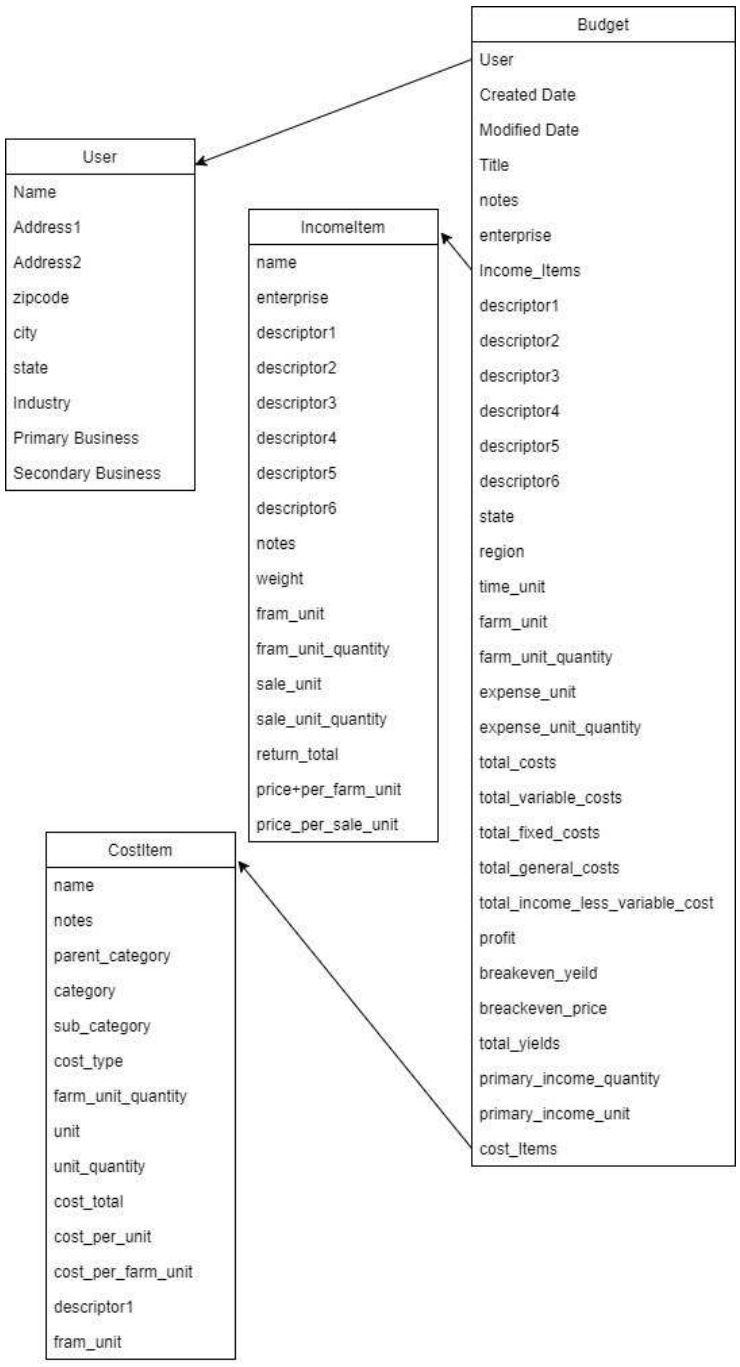

Figure 2: User Data Currently Implemented in the *AgBiz Logic* project

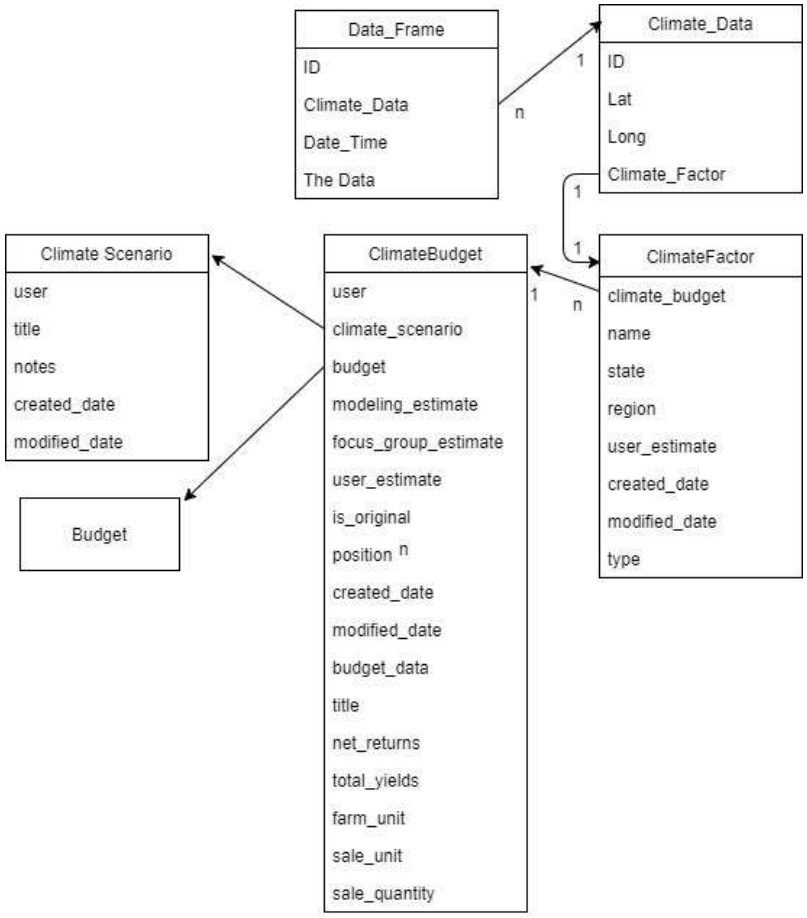

Figure 3: Design For the Climate Data for the *AgBizClimate* project

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 retrieves valuable data back to 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 trademark 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.

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). Third, 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 is not doing well at separating the View and Controller clearly, and most times they 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 4 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.

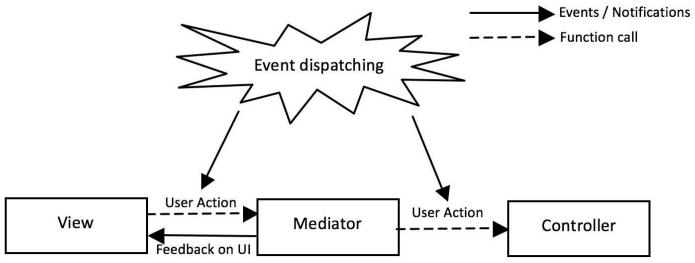

Figure 4: Overall structure for Controller and View

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 may need four general controllers to handle each of them. However, it is not possible or reasonable to deal all 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 5 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. And 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 could call the existing AgBiz Logic Model for certain tasks.

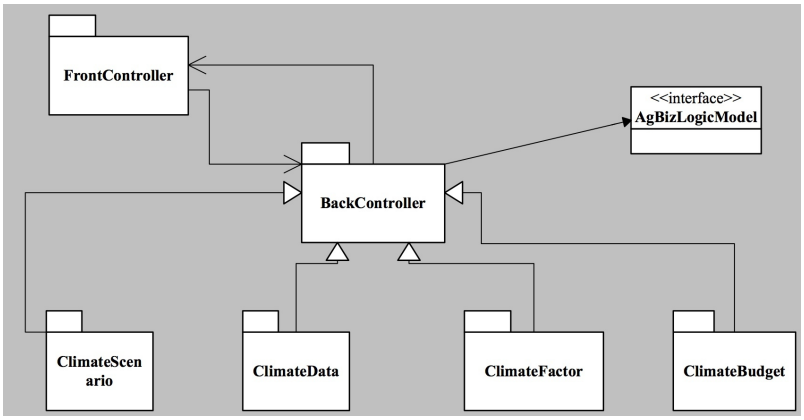

Figure 5: Top controller design diagram

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 6. 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 7 is how a typical transaction will work with the Climate Data API.

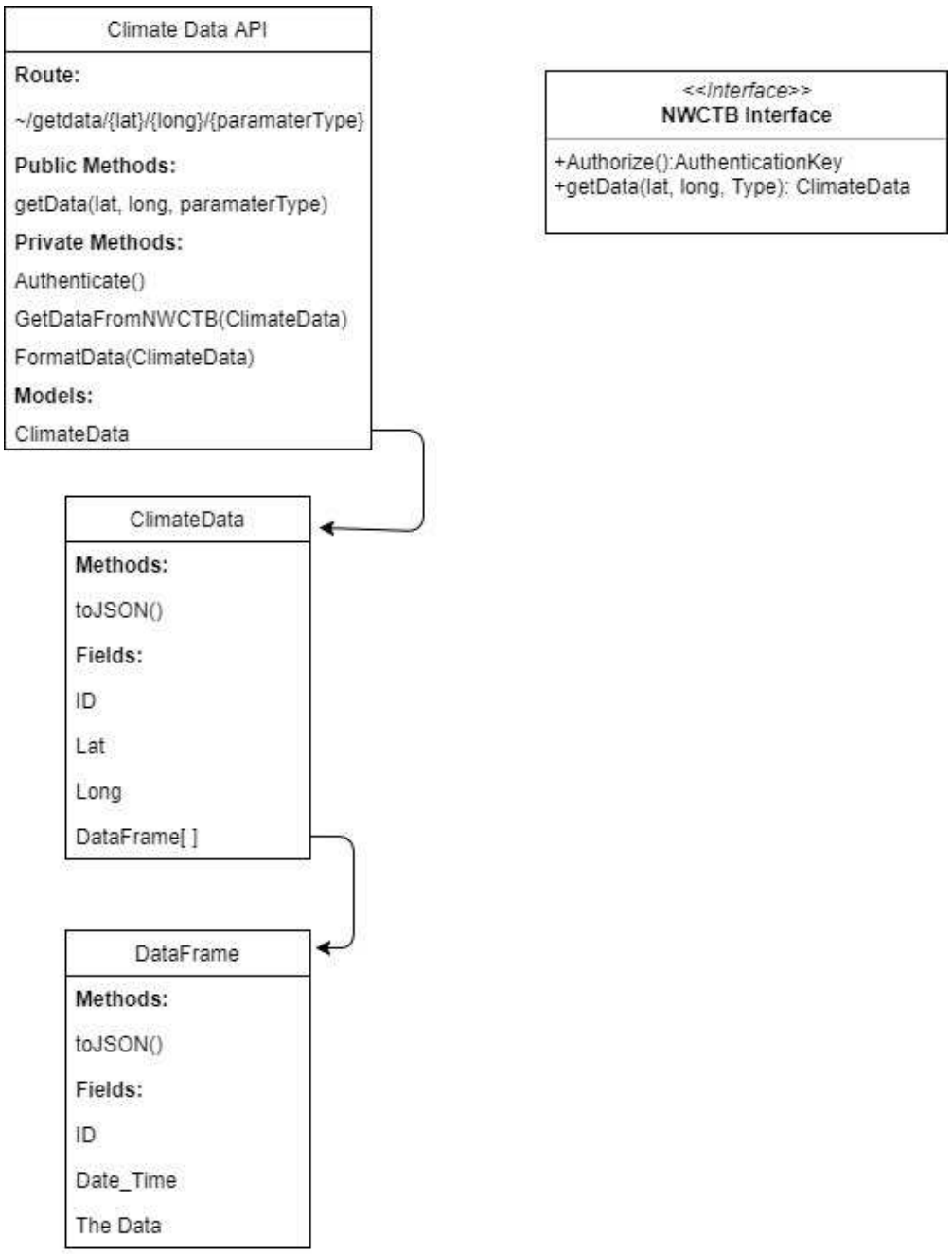

Figure 6: Design of Climate Data API and associated models

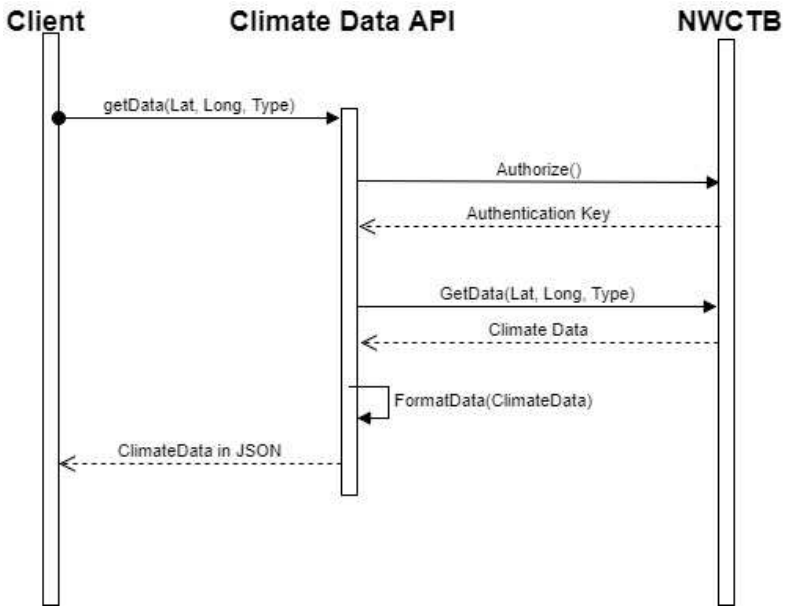

Figure 7: Typical Climate Data API Transaction

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

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

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

These figures are currently in the figured folder...

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 8 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 8: Requirements Matrix