# DataMaster:
# Software Architecture

DnD
*Computer Science Department*
*California Polytechnic State University*
*San Luis Obispo, CA USA*

December 6, 2018

# Contents

# Credits

| **Name** | **Date** | **Role** | **Version** |
|---|---|---|---|
| Griffin Aswegan | November 6, 2018 | Author | 1.0 |
| Steven Bradley | November 6, 2018 | Author | 2.0 |
| Christina Daley | November 6, 2018 | Author | 2.0 |
| Larry Hu | November 6, 2018 | Author | 1.0 |
| Shane Villalpando | November 6, 2018 | Author | 1.0 |
| Dustyn Zierman-Felix | November 6, 2018 | Author | 1.0 |

# Revision History

| **Name** | **Date** | **Reason for Changes** | **Version** |
|---|---|---|---|
| Steven Bradley | December 2, 2018 | Added more info in Test section | 2.0 |
| Christina Daley | December 2, 2018 | Fixed formatting | 2.0 |
| Team DnD | November 6, 2018 | Initial Revision | 1.0 |

# 1 Introduction

This document describes the various components and pieces that make up the DataMaster software. This document details the parts of the system, how the parts relate to each other, how each part is connected, and how each part relates or interacts with the user.

This document also contains images and visuals, such as Use Case Diagrams, Activity Diagrams, and Component and Deployment Diagrams, to help with visual clarity and understanding.

# 2 Problem Description

MarkLogic is looking for software that can help organize and collect common information from large data silos while leaving the data itself untouched.

This solution is designed to provide a flexible and intelligent system for finding commonalities between data sets. Specifically, this system is designed to look for and find connections between different collections of data and classify them so that groups of common data sets can be found.

For more details, the System Requirements Specification document (the SRS, found below under Glossary) provides more specifics on how the system is supposed to work, how it solves the solution from a higher-level perspective, and more details on how the system interracts with the user.

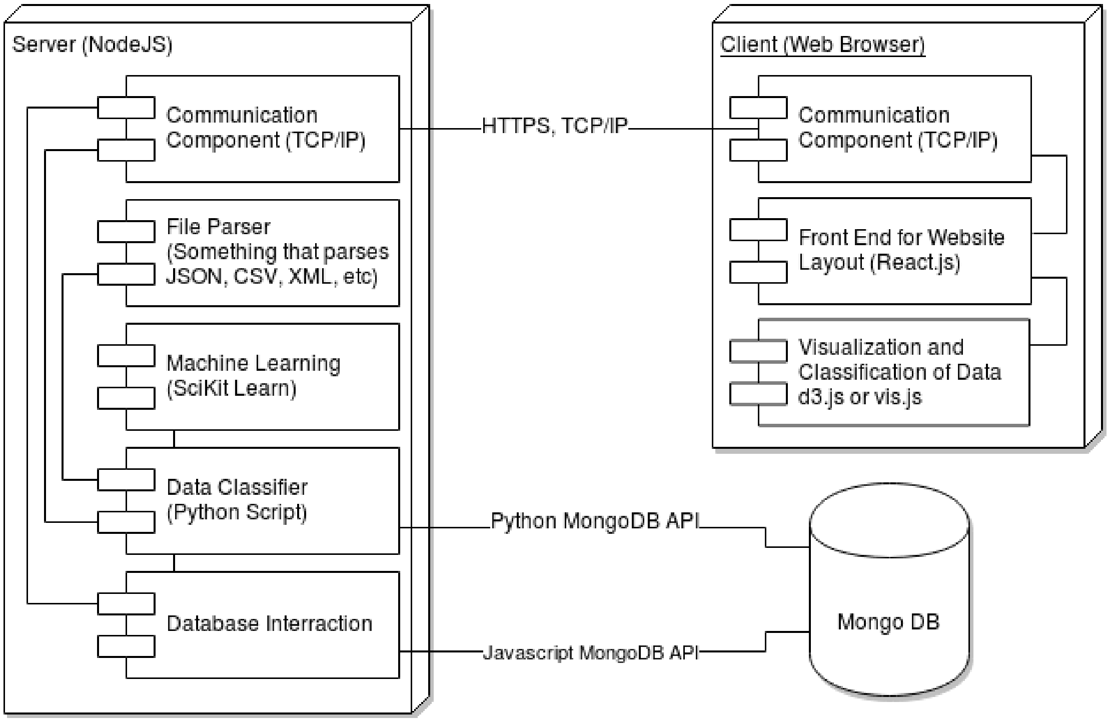
# 3 Solution

## 3.1 Overview

The main components of the system are as follows: the primary back end of the system will be written in Python, using SciKit Learn for Machine Learning of classifications, Pandas for data wrangling, and Django as a REST API to push data to the front end. The back end will also utilize Node.JS to provide a front-end webpage that will be written primarily in Javascript, utilizing React.JS and D3.JS for visualization. The system will be run on AWSs servers during development, but will be designed to be run on any server that has Node and Python installed.

## 3.2 Components

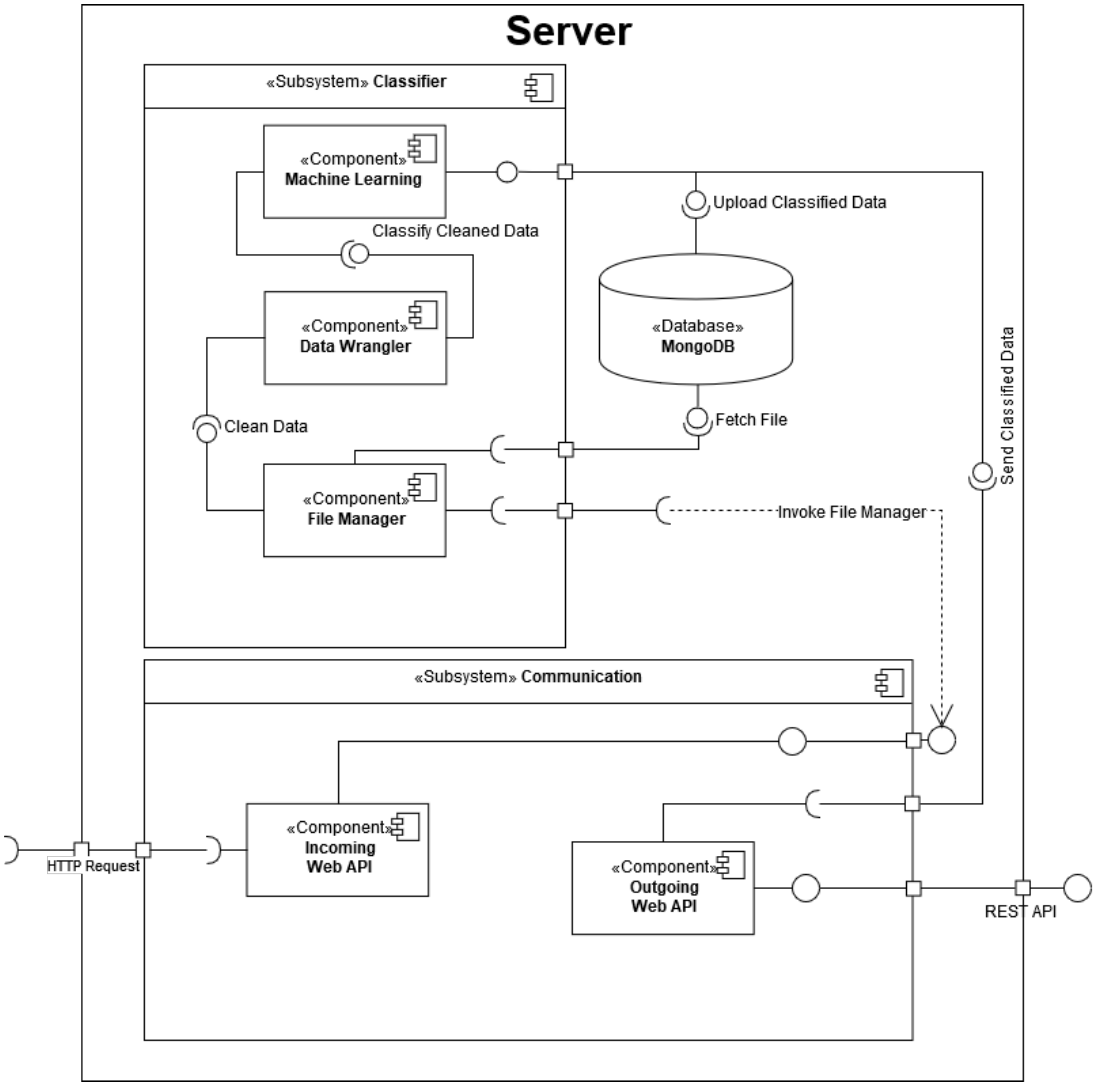
### 3.2.1 Deployment Diagram (Griffin Aswegan)


Server (NodeJS)
Communication Component (TCP/IP)
File Parser (Something that parses JSON, CSV, XML, etc)
Machine Learning (SciKit Learn)
Data Classifier (Python Script)
Database Interraction
HTTPS, TCP/IP
Client (Web Browser)
Communication Component (TCP/IP)
Front End for Website Layout (React.js)
Visualization and Classification of Data d3.js or vis.js
Python MongoDB API
Javascript MongoDB API
Mongo DB

DataMaster is broken up into two major parts - a back-end server, and a front-end web page. The back-end server has a couple of major components - the main classification component, which consists of a machine learning component, a file parser, and the actual classifier, and a system running as a host for the front-end web page.

The client-side front-end is going to be composed of a web page (running on the aforementioned host system) that will display information to the user. Most of the information displayed will be from the classifier in the server. The front-end is planned to perform no computation involved in classifying, but some light computation in drawing some visualizations.

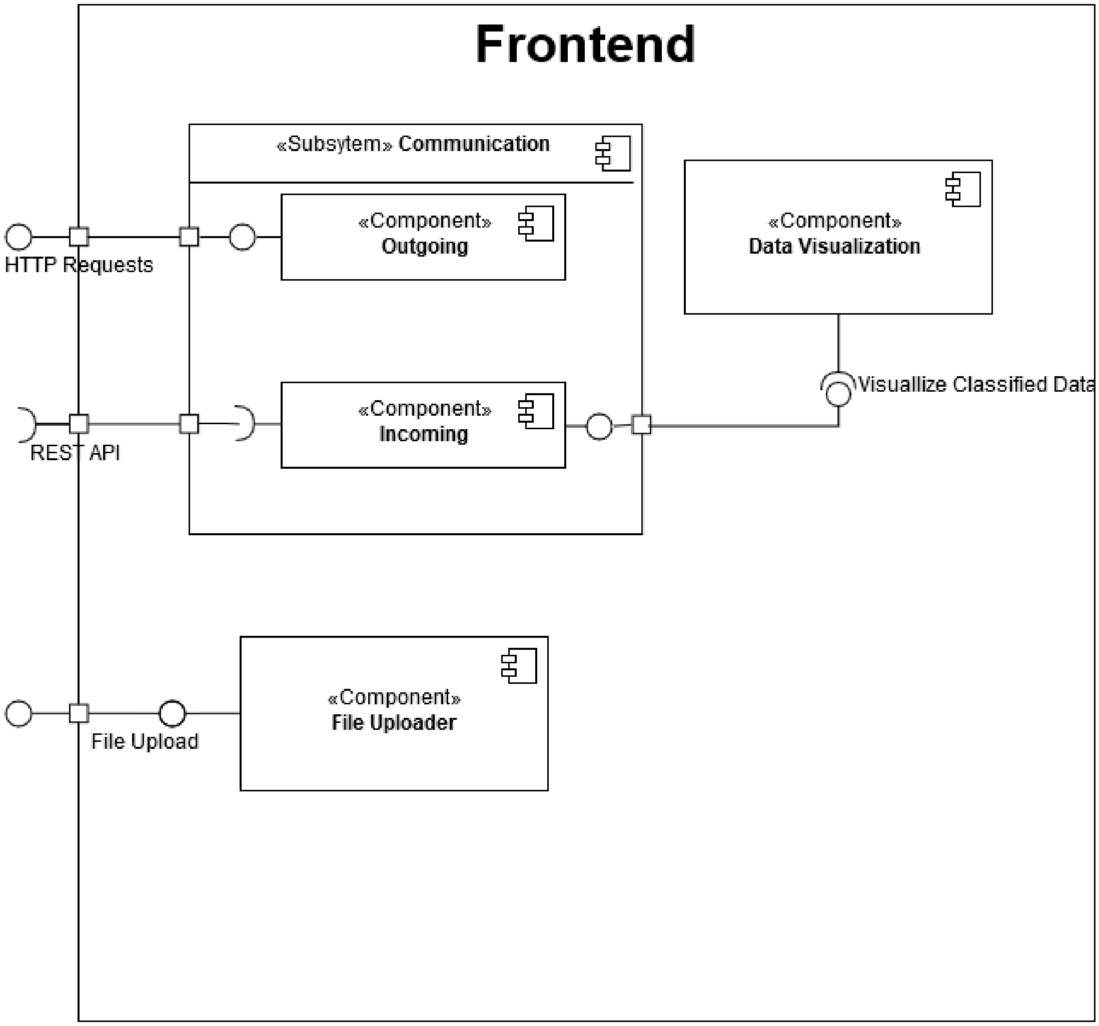
### 3.2.2 Backend Component Diagram (Steven Bradley)



The back end of DataMaster will consist of two main subsystems, the data classifier and a communication subsystem. The communication subsystem will be responsible for all communication between the server and the client. Clients will send HTTP requests to the server. These HTTP requests will be handled by the communication subsystem, which will invoke the data classifier.

The data classifier will have 3 main components: a file manager, a data wrangler, and a machine learning component. The file manager will be able to fetch newly uploaded files from the database. It will also be able to fetch previously used data from the database. After the file manager gets data from the database, it will pass the data to the data wrangler which is responsible for cleaning the dataset and transforming it into tables usable by the machine learning component. Once these tables have been formed, the machine learning component will use the tables to classify the dataset. The classification data will then be uploaded to the database and then passed back to the communication subsystem. In the communication subsystem, the data will be jsonified and then sent back to the client via a REST API.

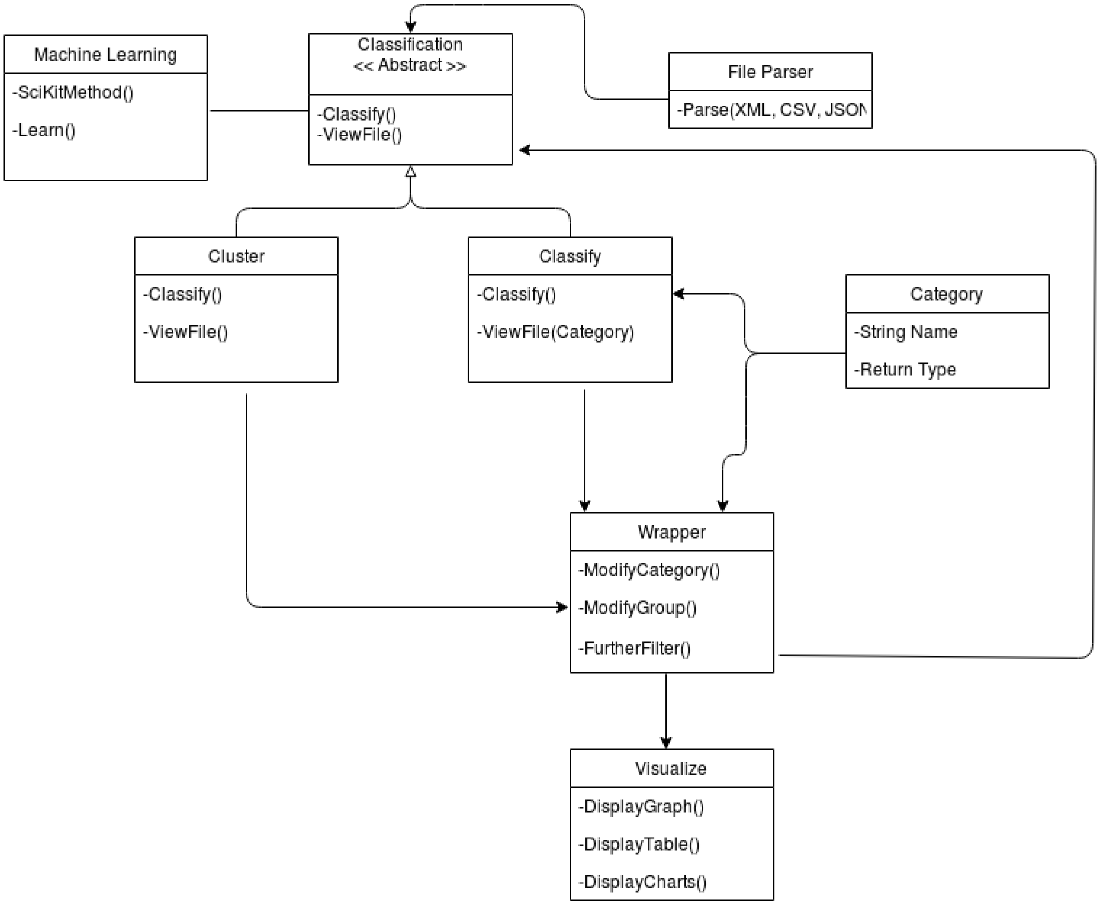
### 3.2.3 Frontend Component Diagram (Steven Bradley)



The frontend of DataMaster is relatively simple. There is a communication subsystem which is responsible for handling all communication between the frontend and the backend, a data visualization component, and a file upload component.

The communication subsystem will send HTTP requests to the backend and will consume the REST API responses from the backend. When the backend sends JSONified objects(classified data) to the frontend, the communication subsytem will hand this data off to the data visualizer. The data visualizer has a single responsibility, visualizing data. The file upload component will be responsible for uploading files that are on the user's local computer to our database.

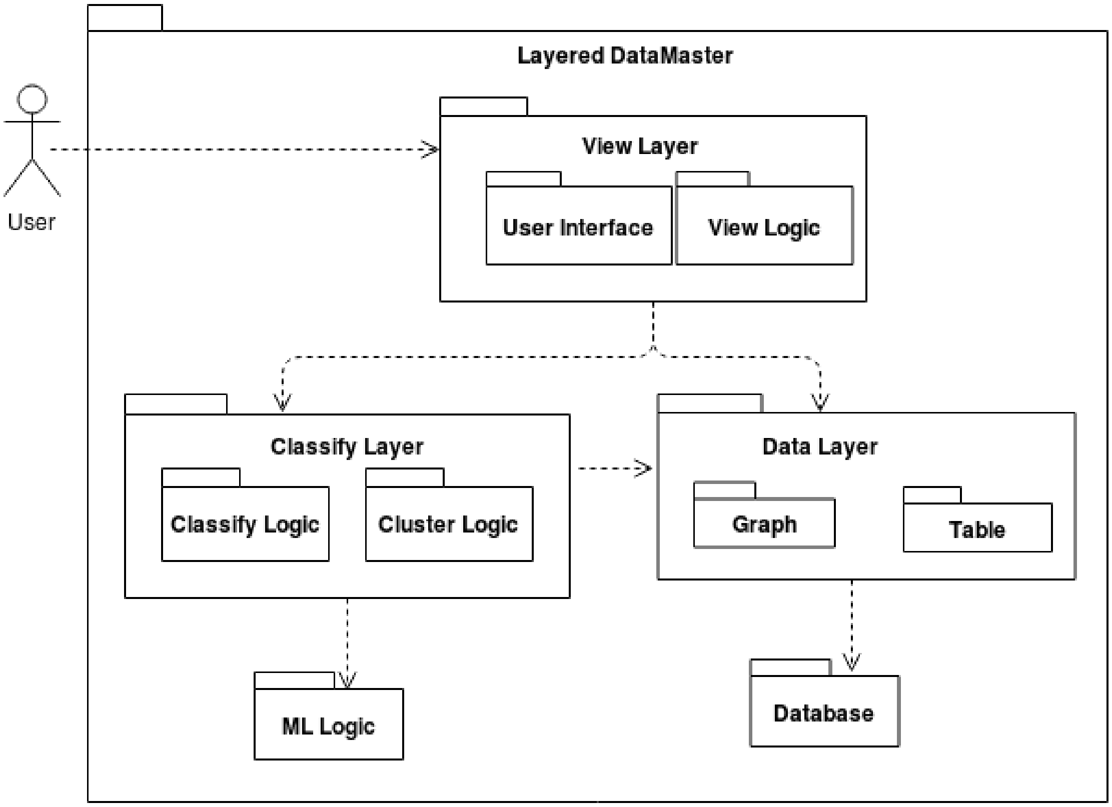
### 3.2.4 Class Diagram (Larry Hu)



The above diagram is the class diagram of the system. This is a static diagram that shows the relationships of all the individual pieces of the classification portion of DataMaster. The purpose of this system is the Classify Data and eventually display the data to the user.

We start with the File Parser that takes in standard file types such as XML, CSV, and JSON and sends it to the Classification Abstract class. Here the data is classified by the Machine Learning class and there is also an method to ViewFile. Cluster and Classify classes extend Classification and are subclasses. They have the option to Classify the data with or without given categories. Category is an attribute to of classify and Wrapper. Once the data is classified it is entered into a wrapper class and can be sent to the Visual Class to be ready to be displayed to the user.

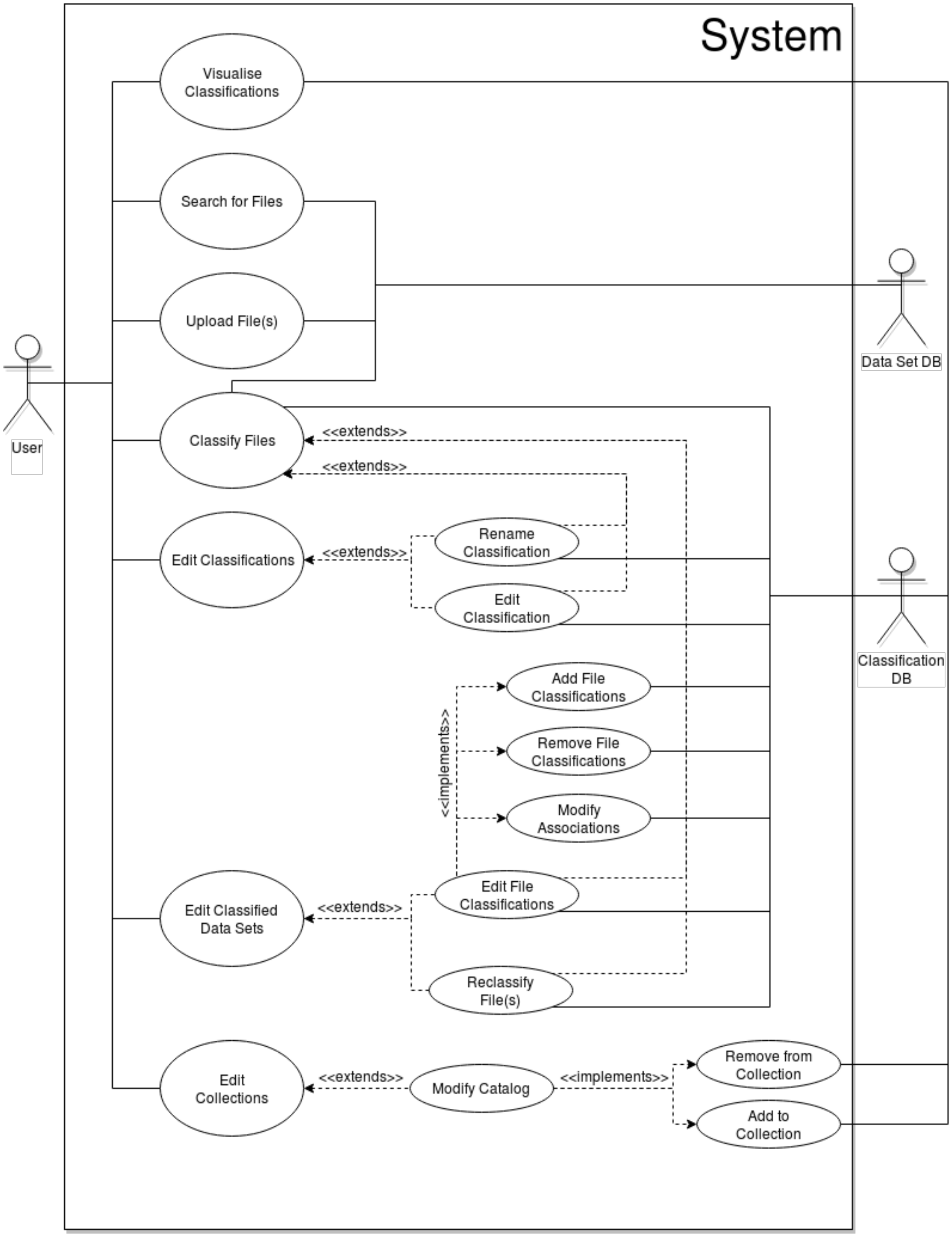
### 3.2.5 Package Diagram (Christina Daley)



DataMaster layers several different packages to successfully create a working classifier. The DataMaster takes user input through the view layer. Within this layer the user interface and view logic is contained. This layer also accesses the data layer and the classification layer. The data layer holds the graph and table organization of the data and accesses the database. The classification layer holds the classify and cluster logic. The classification layer accesses the machine learning logic.

## 3.3 Design

### 3.3.1 Use Case (Griffin Aswegan)

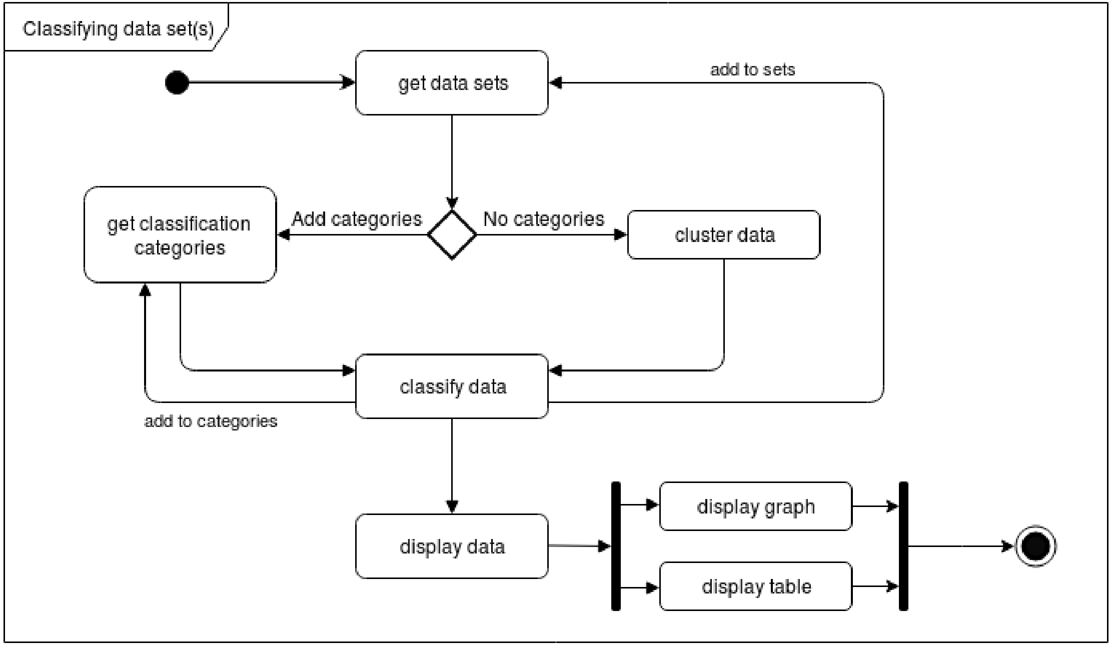


The above diagram is a use-case example for the system. There are three primary actors - active users of the systems, a data base to store the data sets themselves, and a data base to store information about classification (note that these do not need to be separate tables - they are marked as separate to illustrate that they have no interaction with each other except through the system, and that classification should be connected, but independent, of the data itself).

Users are able to perform a multitude of tasks, such as uploading files for future classifiction, classify files based on information within though files, search for specific files or data sets, visualize connections and similarities between classifications and data sets, edit data sets that have been classified, and edit collections of similar classifications. The Classification Database stores all of the information involving the classifications, including which files have what classifications, what classifications are parts for collections, and more. The Data Set Database stores all of the information related to the physical files, such as name, column information, metadata, and more.

### 3.3.2 Activity Diagram (Christina Daley)

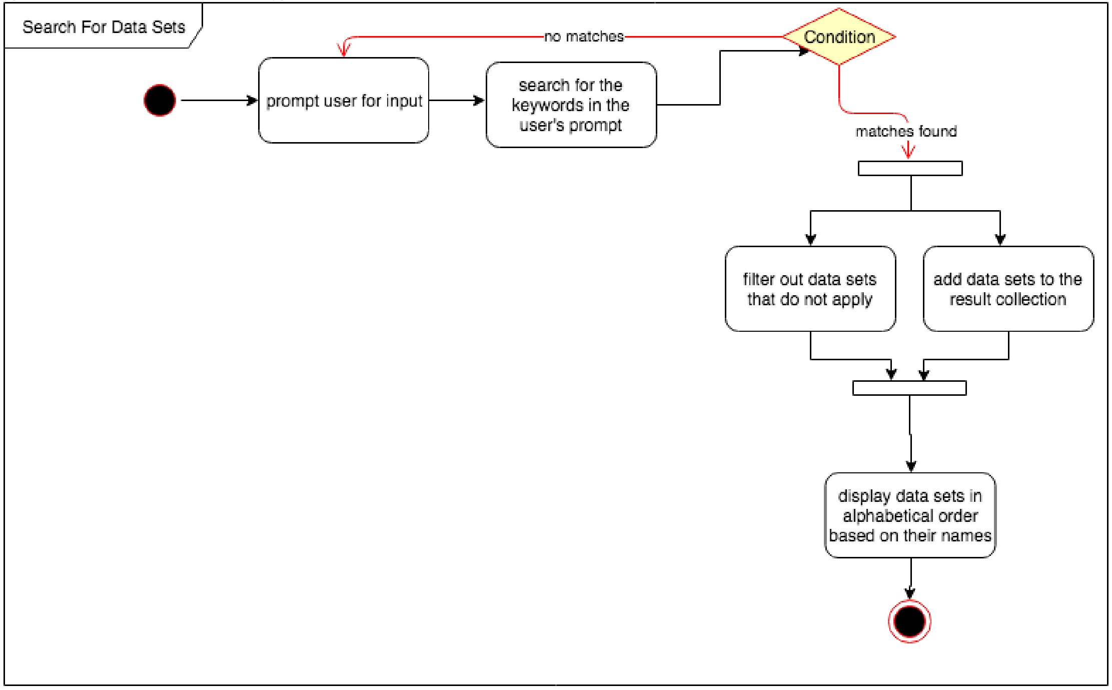
**Classifying Data Sets:**



DataMaster has a specific flow of actions it will take to classify data sets. To start, DataMaster will get the data set or sets it will process. It then will either classify or cluster the data set depending on if it received classification categories. Then it will classify the data and has the ability to display the data in graph and/or table view.

### 3.3.3 Activity Diagram (Dustyn Zierman-Felix)

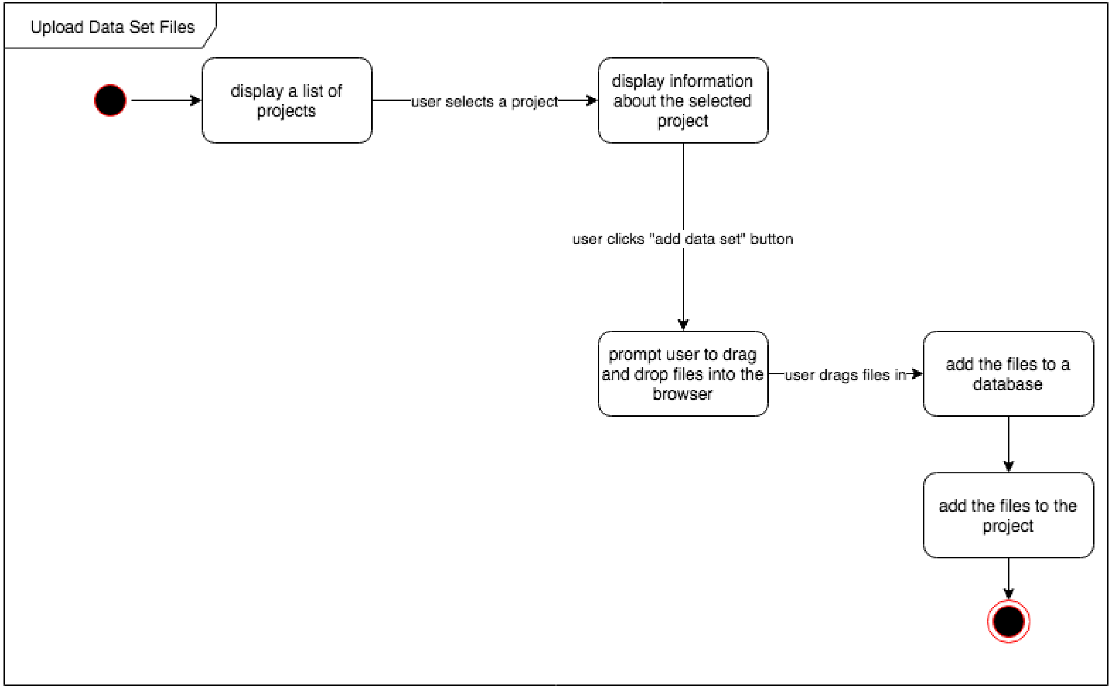
**Search For Data Sets:**


Search For Data Sets
no matches
Condition
prompt user for input
search for the keywords in the user's prompt
matches found
filter out data sets that do not apply
add data sets to the result collection
display data sets in alphabetical order based on their names

DataMaster gives the user the option of searching for data sets, which will help the user build projects and make new classifications on different data. This activity diagram displays the flow that the user must go through in order to find new data sets. The system will display the results of the user's search in alphabetical order, which will be helpful to the user.

### 3.3.4 Activity Diagram (Shane Villalpando)

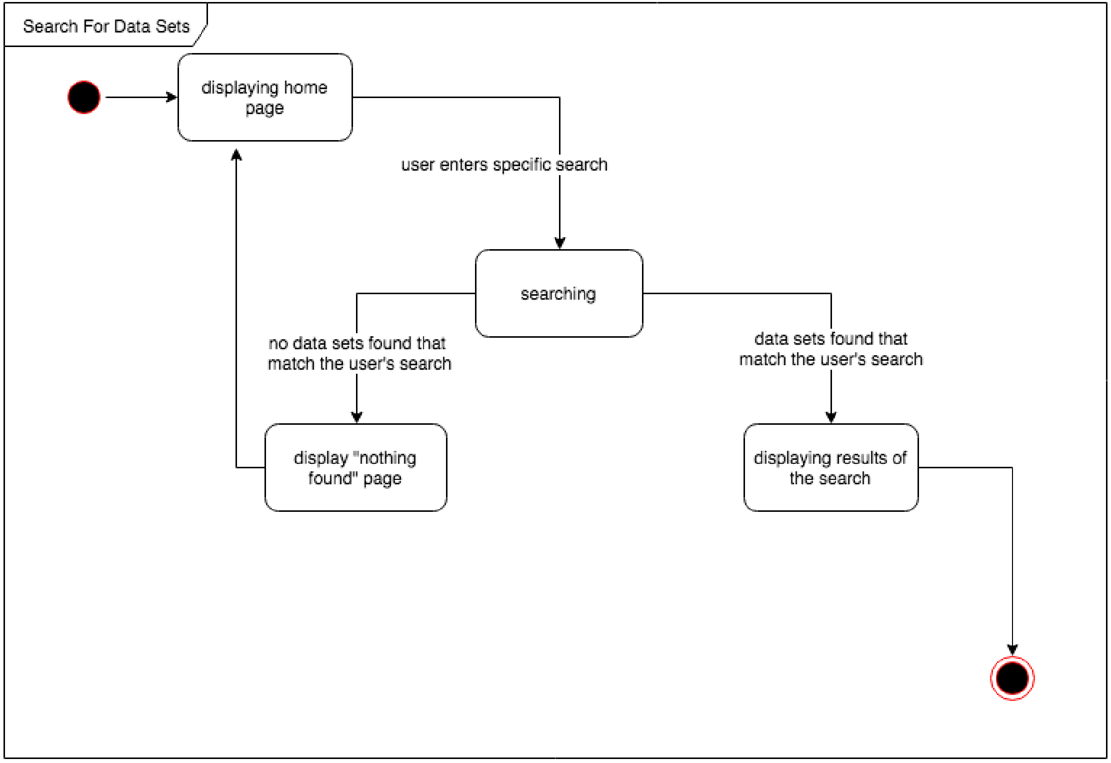
**Upload Data Set Files:**


Upload Data Set Files
display a list of projects
user selects a project
display information about the selected project
user clicks "add data set" button
prompt user to drag and drop files into the browser
user drags files in
add the files to a database
add the files to the project

DataMaster conveniently allows users to add files to projects. DataMaster utilizes a convenient drag-and-drop method of doing this. Files that the user adds to projects will be stored in a database and inside the user's project.

### 3.3.5 State Diagram (Dustyn Zierman-Felix)

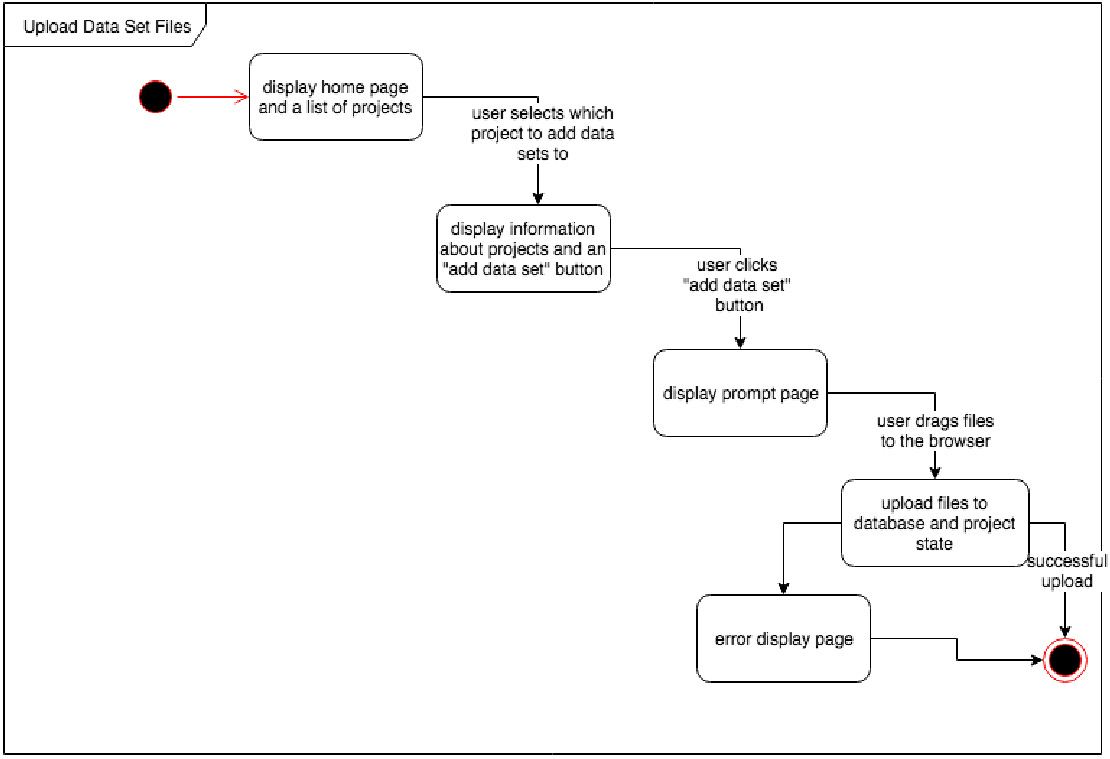
**Search For Data Sets:**


Search For Data Sets
displaying home page
user enters specific search
searching
no data sets found that match the user's search
data sets found that match the user's search
display "nothing found" page
displaying results of the search

The system of DataMaster will be in the state of displaying the home page at the beginning of this state diagram. Once the user enters a search, the system will be in the state of searching for results. If the system does not find a single match, the system will display a page that shows the user that nothing matched the search. Then the state will return to displaying the home page. If there were results found, the system will display the results on a new page.

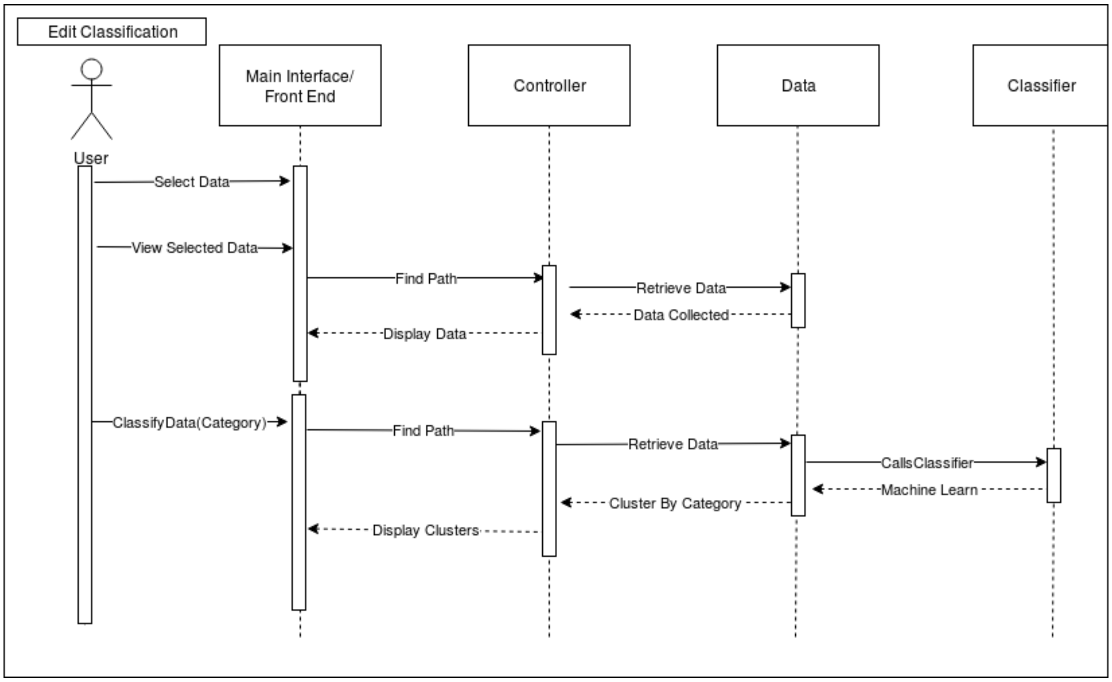
### 3.3.6 State Diagram (Shane Villalpando)

**Upload Data Set Files:**


Upload Data Set Files
display home page and a list of projects
user selects which project to add data sets to
display information about projects and an "add data set" button
user clicks "add data set" button
display prompt page
user drags files to the browser
upload files to database and project state
successful upload
error display page

DataMaster will be in the state of displaying the home page with a list of current projects that the user is working on. When the user selects a project to add data sets to, the state will change to display information about that project and an "add data set" button. When the user clicks on that button, the state will change to display a prompt page. When the user drags files into the browser, the state will change to uploading those files to the database and project. If the upload is successful, the task is done. Otherwise, the state will change to an error display screen.

### 3.3.7 Sequence Diagram (Larry Hu)



This is the Sequence Diagram for covering the use case to ”Edit Classification”. The user starts by selecting data and viewing selected data from the main interface. The main interface finds the related path and sends it to the controller to retrieve the data from the database which is passed back to the controller. Then the controller sends the received data to the main interface to be displayed.
Next the user selects the Classify Data button and inputs his desired category. The front end sends the category from the controller to the Database. The database then calls classify with Classifier and uses machine learning to change the current Database. Database is now clustered by the specific given category, and the clusters are displayed in the main interface.

# 4 Test

In order to ensure that, after each sprint, we have a working, potentially shippable increment, we will follow an industry standard testing and integration process. We will be using GitHub to manage our code repository, with a ’master’ branch that will act as the primary release branch. Each developer will work on their own separate branch, to ensure modularity and a lack of interdependence. Each developer must submit a pull request that must be reviewed by at least one other developer before it is merged. Each pull must have at least one or two meaningful automated tests (unit, UI, acceptance, or other). Developers will be using a TravisCI integration server that will be run after each pull request, to ensure code that is quality and consistent, and if the code does not pass the TravisCI integration test, then the code will not be merged until it is fixed. The developer that submitted the pull request will be responsible for fixing the failing code, and no further pull requests will be made until the pull request in question is repaired.

## 4.1 Testing Tools

This section includes information about the various testing frameworks that we use to test all parts of the system.

### 4.1.1 Unit Testing

For unit testing of our python modules we plan on using the PyTest framework. The developers who write the python modules will be responsible for all unit testing of that module.

### 4.1.2 UI Testing

UI testing frameworks will be determined during the first sprint.

### 4.1.3 Acceptance Testing

Acceptance testing of Python modules will be done using the Robot framework. The responsibility of writing acceptance tests will be mostly on the developers of the Python modules with some assistance from the UI developers.

### 4.1.4 Continuous Integration

A TravisCI server will be used for continuous integration. The TravisCI server is connected to our GitHub repository. Whenever a developer makes a pull request or a commit is made to the repository, the TravisCI server shall run a subset of our test suite. This subset shall consist of integration tests that test whether the various components of our system work together.

## 4.2 Test Example

| | |
|---|---|
| Test: | Drag and Drop File Uploads |
| Use Case: | As a data analyst, I want to be able to drag-and-drop files to upload |

| | |
|---|---|
| Requirements | 1. The system shall allow users to drag and drop file upload<br>2. The system shall store uploaded files received from the front-end on the server<br>3. The system shall upload those files to an AWS S3 bucket for later use<br>4. The system shall delete the file from the server |
| Steps | 1. The user drags a file to upload.<br>2. The front end will process the file and push the file to the back end.<br>3. The back end receives and save the file for later use.<br>4. The back end pushes the files to an Amazon Web Services S3 bucket.<br>5. The back end removes the file itself from the server. |

# 5 Issues

No issues to be aware of.

# A Glossary

*Dataset*: A file, or collection of files, that represents a single table within a data base.

*Classification*: A label that identifies information about the data inside of a data set under a common identifier.

*Category/Collection*: A collection of classifications that provide information about a single topic (for example, "Phone Number", "Address", and "Email" could all be categorized/collected as "Contact Information".