# CS 340 Project Two Grazioso Salvare Dashboard README

*Joe Clancy*

*6/18/2024*

*CS340*

*Project Two*

## About the Project/Project Title

The purpose of this project was to build an interactive, web-based interface for Grazioso Salvare to use for reviewing candidate animals available from the Austin Animal Shelter.

The project operates on three separate layers:

1. A MongoDB datastore to store and serve Austin Animal Shelter data.
2. A Python, Juptyer Notebook-based controller for the front end.
3. A Dash-based dashboard frontend for visualization and user interaction.

Additionally, the Python layer relies on a custom Create/Read/Update/Delete (CRUD) module to handle interactions with MongoDB through the Pymongo distribution. Access credentials for the database are integrated into this module.

## Project Components

MongoDB was selected to handle the backend for this project due to two valuable features:

* Document model translates easily to Python data structures
* Pymongo distribution allows simple, effective CRUD operations

The Dash framework, which handles the frontend for the project, is based on the Model-View-Controller (MVC) architecture. In this architecture, the Model represents data sources – a MongoDB docstore with a data set from the Austin Animal Shelter, in our case. View defines user interface elements, like the data table, map, and pie chart in this project. The controller manages user interactions and processes inputs through callback functions that attach logic to view elements.

This project uses a function to manage the width of column headers based on the following source:

<https://github.com/plotly/dash-table/issues/432>

## MongoDB Configuration

The project relies on an active instance of MongoDB with a database named ‘AAC’ containing a collection named ‘animals. Using [this](https://learn.snhu.edu/content/enforced/1609887-CS-340-11376.202451-1/course_documents/aac_shelter_outcomes.csv?_&d2lSessionVal=ScQoRYLN9OTGv4T9RbCyYwie6&ou=1332057&ou=1609887) dataset, such a collection can be imported to MongoDB with a command like:

A screenshot of a computer program

Description automatically generated

## Once the collection has been imported, user credentials will need to be created for the database for the Python layer to perform CRUD operations. Example:

A screenshot of a computer screen

Description automatically generated

## CRUD Module

CRUD is a Python module which provides the AnimalControl() class to handle CRUD operations with MongoDB.

CRUD relies on the pymongo library, which can be installed with pip using the command:

pip install pymongo

CRUD uses the following connection configuration for accessing MongoDB, initialized when the AnimalControl() object is instantiated:  
 user = 'aacuser'

pwd = 'barkwoof'

host = 'nv-desktop-services.apporto.com'

port = 31368

database = 'AAC'

collection = 'animals'

CRUD could be reconfigured to function on a different MongoDB collection through modification of this configuration.

**AnimalControl() Class Functionality**

**create\_document(***document: dict***)**

Insert *document* into the ‘animals’ collection. Returns *true* if the document was successfully inserted. Returns *false* if an error was thrown.

**read\_documents**(*query: dict*)

Returns a *list* of documents as *dicts* matching the key:value pair *query*. Returns an empty *list* if an error is thrown.

**update\_documents**(*query: dict, update: dict, update\_many: bool*)

Modifies every key:value pair matching *update* in document(s) matching *query*. If update\_many is set true, uses update\_many() API call and updates all documents matching *query*, otherwise uses update\_one() API call and updates only the first document matching *query*. update\_many is false by default.

**delete\_documents**(*query: dict, delete\_many: bool*)

Deletes document(s) matching *query* from the ‘animals’ collection. If delete\_many is set true, uses delete\_many() API call and deletes all documents matching *query*, otherwise uses delete\_one() API call and deletes only the first document matching *query*. delete\_many is false by default.

**Dashboard Branding**

Grazioso Salvare has requested the inclusion of the following components somewhere on the project dashboard:

* The Grazioso Salvare logo. The company has requested that this logo include a link to the client’s home page: [www.snhu.edu](http://www.snhu.edu).



* A unique identifier (text or image) containing identifying the project creator.

**Dashboard Functional Requirements**

Gravioso Salvare requires the following interface elements be present for the dashboard:

* Interactive filter options to filter the Austin Animal Center Outcomes dataset by:
  + Water Rescue
  + Mountain or Wilderness Rescue
  + Disaster Rescue or Individual Tracking
  + Reset (returns all widgets to their original, unfiltered state)
* A data table which dynamically responds to filtering options.
* A geolocation chart and a second chart (such as a pie chart) that dynamically responds to filtering options.

**Filter Criteria and Methodology**

Gravioso Salvare has provided specific criteria for each filter option regarding the breed, sex, and age of candidate animals, articulated below:

A table with different breeds

Description automatically generated

However, due to the variety of breeds and mixes across the animal population, there is still some ambiguity in these criteria. Does an Alaskan Malamute/Siberian Husky mix meet the criteria for a Mountain or Wilderness rescue dog, for example. This project is configured to use the most inclusive definition of these criteria to provide the client with the largest possible pool of candidate animals, however it could be reconfigured for stricter breed requirements or the option for stricter breed requirements.

The following regular expression (REGEX) MongoDB query for each set of criteria accomplishes this result:

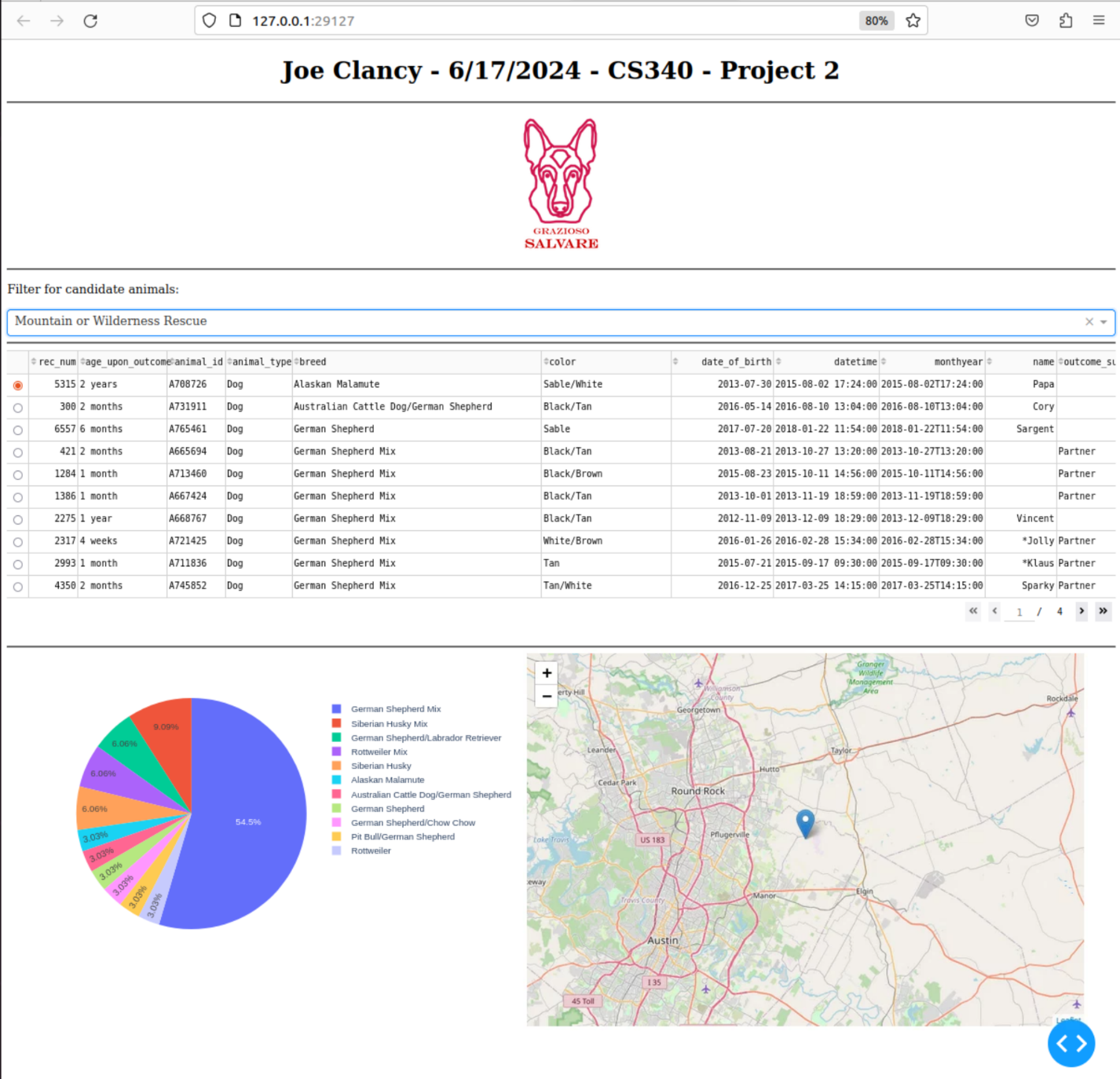


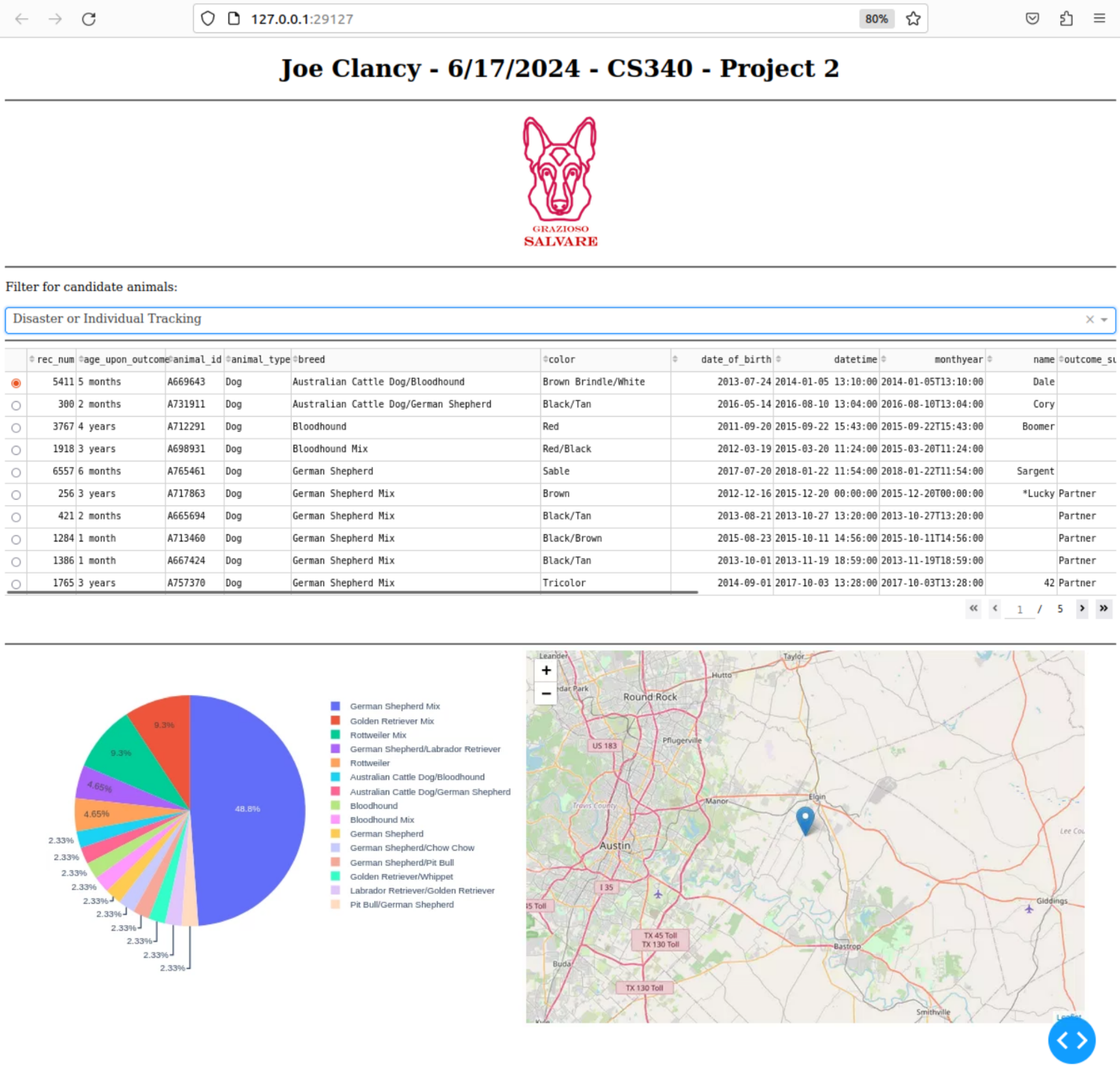
### Dashboard Functionality Demonstration

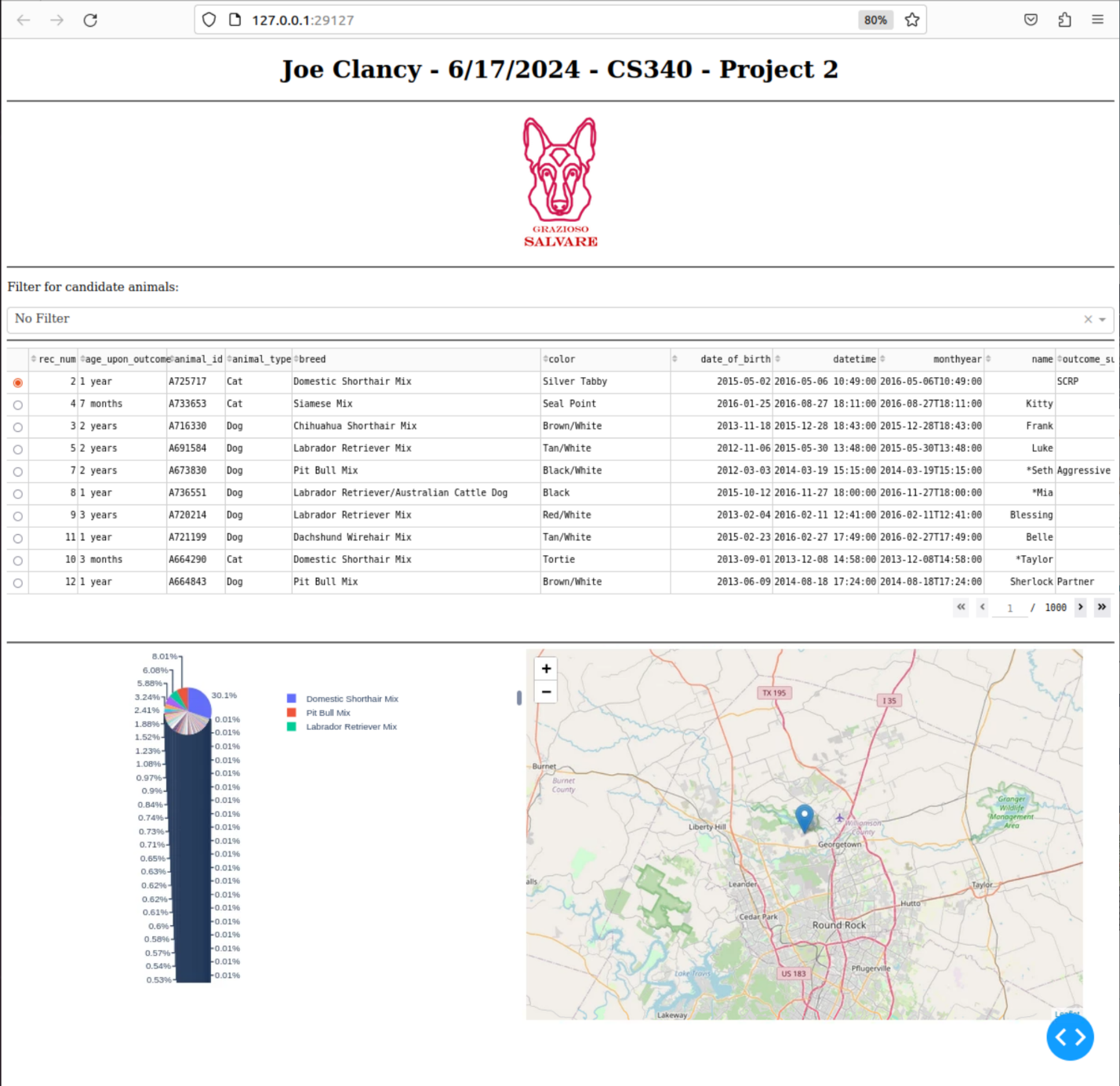
Selecting the appropriate filter from the drop-down menu will yield the following dashboard views:

**Water Rescue**A screenshot of a computer

Description automatically generated

**Mountain or Wilderness Rescue**

**Disaster or Individual Tracking**

**No Filter/Reset**

## Project Recreation Instructions

If the project needed to be recreated, it could be accomplished with the following steps:

1. Install MongoDB, pyMongo, and required Dash components.
2. Use an admin account to load Austin Animal Control dataset into MongoDB. (see: *MongoDB Configuration*)
3. Generate a MongoDB user account with at least read/write permissions. (see: *MongoDB configuration*)
4. Add CRUD module and Jupyter Notebook controller to the host server.
5. Update CRUD module to include new MongoDB account credentials and server URI.
6. Start MongoDB.
7. Start Jupyter Notebook controller.

**Challenges**

The Austin Animal Control dataset has many inconsistencies in the way it reports animal breeds, and in some cases uses abbreviations that cannot be intuitively anticipated. The breed Chesapeake Bay Retriever, for example, is abbreviated as Chesa Bay Retr, which is not a abbreviation used for any other breed of retriever. Additionally, variations in how animals with hybrid or mix breeds are documented leads to ambiguity in identifying the client’s preferred breeds. These challenges were overcome with regular expressions, however, pyMongo’s native REGEX functionality is very limited, so refactoring may be necessary if the dataset becomes less reliable or the client’s requirements become more complicated in the future.

## Contact

Your name: Joe Clancy, joseph.clancy@snhu.edu