**1. Project Overview & Required Functionality**

The Grazioso Salvare Animal Shelter Dashboard is a **web-based data visualization tool** designed to connect directly to the Austin Animal Center (AAC) MongoDB database.  
Its core functions include:

* **Secure Authentication** – Prompts the user for a valid MongoDB username and password, preventing unauthorized database access.
* **Real-Time Data Queries** – Pulls live data from MongoDB using the CRUD Python module.
* **Geolocation Map** – Plots intake locations on a Leaflet-based map, updating dynamically based on user-selected filters.
* **Responsive Dashboard Layout** – Built with Dash for an intuitive, interactive experience.

**Proof of Functionality**:  
  
A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.A screenshot of a computer

AI-generated content may be incorrect.

**2. Tools & Rationale**

**MongoDB (Model Layer)**

* **Why Chosen**: Flexible schema to store variable animal attributes (breed, intake type, location, etc.).
* **Capabilities**: Supports **fast aggregation queries** for generating breed distributions, integrates seamlessly with Python via PyMongo, and handles large datasets efficiently.

**PyMongo**

* Python driver for MongoDB enabling **Create, Read, Update, Delete** operations.
* Chosen for direct compatibility with the Dash environment.

**Dash by Plotly (View + Controller Layers)**

* Provides a **component-based UI framework** for building interactive web apps in Python.
* Allows integration of Plotly Express charts and Leaflet maps.
* Supports dynamic callbacks, enabling real-time updates without refreshing the page.

**Plotly Express**

* Used to create interactive charts with hover labels, legend filtering, and color customization.
* Enabled grouping of small percentage breeds into an “Other” category.

**Dash Leaflet**

* Used for geolocation visualization.
* Ideal for mapping animal intake points using latitude/longitude from the database.

**pandas**

* Handles data manipulation, filtering, and aggregation before rendering charts and tables.

**3. Steps to Reproduce**

**Prerequisites**

* Python 3.x installed
* MongoDB instance containing the aac\_shelter\_outcomes.csv dataset
* The crud.py module implemented with connection/authentication logic

**Install Required Packages**

pip install dash jupyter-dash pandas plotly dash-leaflet pymongo

**Import AAC Dataset into MongoDB**

mongoimport --username <MONGO\_USER> \

--password <MONGO\_PASS> \

--authenticationDatabase admin \

--db AAC --collection animals \

--type csv --headerline \

--file aac\_shelter\_outcomes.csv

**Run the Dashboard**

1. Open the provided Jupyter Notebook (animal\_shelter\_dashboard.ipynb).
2. Execute all cells in sequence.
3. Enter valid MongoDB credentials and click **Submit**.
4. Interact with the chart, map, and table.

**4. Challenges & Solutions**

**Challenge 1 – Missing geolocation data**

* **Problem**: The map failed to load when some records had no coordinates.
* **Solution**: Filtered out any entries without valid location\_lat and location\_long.

**Challenge 2 – Repeated DB connection attempts**

* **Problem**: Callback triggered on every keystroke in username/password fields.
* **Solution**: Added a n\_clicks trigger on the submit button to control when the DB connection is attempted.

**5. Resources**

 [MongoDB Documentation](https://www.mongodb.com/docs/)

 <https://pymongo.readthedocs.io/en/stable/>

 <https://dash.plotly.com/>

 <https://plotly.com/python/plotly-express/>

 <https://dash-leaflet.herokuapp.com/>

**How do you write programs that are maintainable, readable, and adaptable?**

In this course, I focused on building a CRUD Python module in Project One and then connecting it to the dashboard in Project Two. By designing the module with clear function names, consistent formatting, and comments, the code became more readable and easier to debug. Breaking down the CRUD operations into separate functions also made it adaptable—if the database structure changes or a new collection is added, I can reuse the same framework with only minor adjustments. This modular approach saves time and effort in the long term. Beyond this project, I could apply the same CRUD module to other dashboards, APIs, or even mobile applications that need database access.

**How do you approach a problem as a computer scientist?**

I start by analyzing the requirements of the problem—in this case, Grazioso Salvare’s request for a dashboard that filters rescue animals by training status and breed. I then break the larger problem into smaller, solvable tasks, such as connecting to the database, building queries, and rendering results in the dashboard. Compared to earlier courses, this project required a deeper focus on both backend (MongoDB) and frontend (Dash) integration, which helped me see how the pieces connect. In the future, I would continue to use this strategy of requirements analysis, modular design, and incremental testing when creating databases and dashboards for other client requests.

**What do computer scientists do, and why does it matter?**

Computer scientists solve real-world problems by creating tools and systems that make organizations more efficient. In this project, I created a dashboard that allows Grazioso Salvare to quickly identify and filter animals that meet search and rescue criteria. Without such tools, they would spend much more time manually searching through large datasets. By automating this process, my work directly helps them make faster and better-informed decisions. More broadly, computer scientists add value by combining problem-solving skills with technical expertise to design solutions that impact businesses, communities, and even society at large.