CS 340 Project Two README

**README File – Grazioso Salvare Dashboard Project**

**1. Project Overview and Required Functionality**

Grazioso Salvare Dashboard was created to meet the demand for an intelligent, interactive system to view, filter, and decide on animal rescue candidates. The company Grazioso Salvare trains animals for emergency situations. The dashboard requires a user-friendly interface for data manipulation, visualization, and geospatial mapping to support these tasks.

Key functionality requirements were:

* **Database Integration**: A Python CRUD module (AnimalShelter class in crud.py) allows seamless read() operations on MongoDB.
* **Data Filtering Interface**: Create interactive controls (e.g., radio buttons or dropdowns) to filter animals by rescue type (Water Rescue, Mountain/Wilderness Rescue, Disaster Rescue).
* **Dynamic Table Rendering**: Dash DataTables render database contents, provide sorting, column selection, and real-time user input updates.
* **Graphical Visualization**: Plotly Express-integrated responsive pie graphic showing animal breed distribution.
* **Geolocation Mapping**: Dash\_leaflet map embedding to illustrate animals' latitude and longitude locations.
* **Branding and UI Enhancements**: Adding a corporate logo and project header for visual appeal and branding.

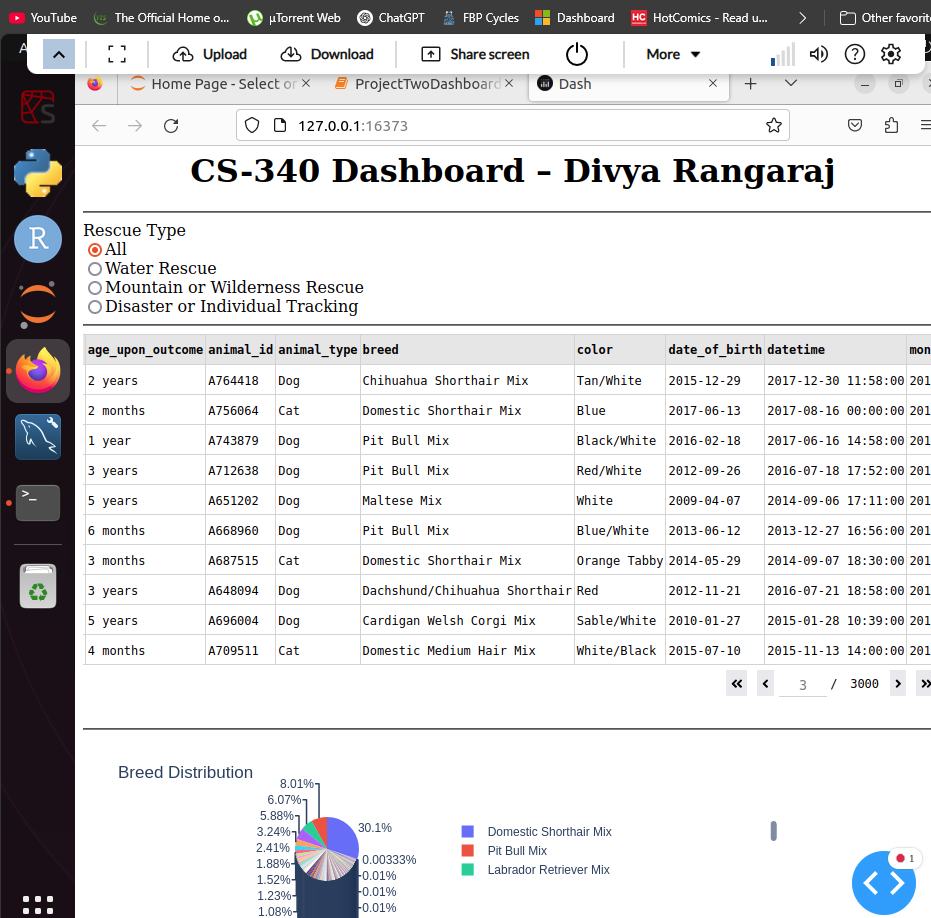
**Screenshots of Functionality**

Screenshots showed operational aspects during testing and validation (Step 6).

Screenshot 1: Full dashboard layout with table, chart, and map.

A screenshot of a computer

AI-generated content may be incorrect.



Screenshot 2: Filtering by Water Rescue with updated chart and records.

A screenshot of a computer

AI-generated content may be incorrect.

Screenshot 3: Terminal output confirming successful server start and callback functionality.

A screenshot of a computer

AI-generated content may be incorrect.

These screenshots show that the dashboard satisfied all customer functional requirements.

**2. Tools and Technologies Used**

Modern online and data tools were used for flexibility, integration, and community support for this project. The chosen tools and rationale are below:

**a. Python (Version 3.9+)**

Python was the main language because of its simplicity, large library ecosystem, and data science, web development, and database interaction capability. Full-stack programming was possible without JavaScript or CSS.

**b. Dash Framework (by Plotly)**

Python web application framework Dash is powerful and open-source. It builds interactive web apps with Python and little code. Python-savvy developers who prefer Python over JavaScript or HTML will love it.

* **Why Dash?**  
  Dash was selected because it provides:
  + UI component layout.
  + Callbacks for event-driven programming.
  + Simple graph, table, and HTML embedding.
  + A seamless Jupyter Notebook integration with JupyterDash.

**c. Dash Leaflet**

Dash Leaflet uses Leaflet.js to display interactive maps. Allows dashboard to:

* Show map tiles.
* Plot markers with tooltips and popups.
* Center the map on Austin, TX (the operational hub).

**d. MongoDB**

Back-end database was MongoDB. It is a document-oriented NoSQL database for flexible, schema-less animal data with variable properties.

* **Why MongoDB?**
  + **Dynamic schema**: Can handle inconsistently structured data.
  + **JSON/BSON structure**: Easily translates to Python dictionaries.
  + **Scalable and fast**: Supports real-time queries for filtering data.
  + **PyMongo integration**: Enables Python-based data access with minimal setup.

**e. Pandas and NumPy**

Pandas converted MongoDB documents into a structured DataFrame for tabular presentation and analysis. For calculations and visualizations, NumPy was used.

**f. Plotly Express**

Plotly Express created dynamic, interactive breed distribution pie charts. Its Dash integration and low code needs make it the best pick.

**g. JupyterDash**

JupyterDash ran Dash programs in Jupyter Notebooks for academic-friendly testing and development.

**3. MongoDB as the Model Layer**

MongoDB was used as the MVC model. Unstructured and semi-structured data, typical in animal shelter records, is its specialty.

**Key Benefits for This Project**

* MongoDB's natural compatibility with the pymongo module lets Python programs connect, read, write, and manipulate data using Python dictionaries.
* Diverse animal intake reports can have varying field counts.
* Flexible query operations: This dashboard uses MongoDB's filtering, sorting, projections, and conditional retrieval.
* Local, Lightweight Hosting: Very easy to deploy locally or in a container, making it excellent for academics.

**4. Dash as the View and Controller**

The Dash framework is an MVC View and Controller.

* **View**: Dash's layout system (html.Div, dcc.Graph, dash\_table.DataTable, etc.) lets developers design the dashboard in Python.
* **Controller**: Decorator-defined dash callbacks update charts, tables, and maps in real time based on user interaction.

**Example:**

@app.callback(

Output('graph-id', 'children'),

[Input('datatable-id', 'derived\_virtual\_data')]

)

def update\_graphs(viewData):

dff = pd.DataFrame.from\_dict(viewData)

return [dcc.Graph(figure=px.pie(dff, names='breed', title='Preferred Breeds'))]

This function dynamically changes the breed pie chart with table data loading or filtering.

**5. Steps Taken to Complete the Project**

**Step 1: Database Configuration**

* Created MongoDB and added animal data.
* Created a crud.py module using AnimalShelter for database authentication and reading.

**Step 2: JupyterDash Setup**

* Jupyter\_dash, dash, dash\_leaflet, and associated packages were imported.
* Validated Jupyter Notebook Dash application display.

**Step 3: Data Retrieval**

* Read documents from MongoDB with read().
* Created a Pandas DataFrame from the returned dictionaries.
* Dropped the Dash DataTable \_id field to prevent data type errors.

**Step 4: UI Layout and Branding**

* Designing the HTML structure with a company logo, header, horizontal dividers, and container rows.
* The base64-encoded logo was rendered using an HTML image tag.

**Step 5: Interactive Features**

* Created a selectable, sortable DataTable.
* Added a callback to filter animals by rescue kind.
* Added a dynamic pie chart for animal breed distribution.
* Used Dash Leaflet to plot animal map markers.

**Step 6: Testing and Screenshot Capture**

* Callbacks and data loading were verified.
* Screenshots for verification and submission.

**6. Challenges Encountered and Solutions**

**a. MongoDB ObjectId Compatibility**

Due to the ObjectId format, MongoDB's \_id field created Dash's DataTable display difficulties. Dropping the field soon after DataFrame creation fixed this.

df.drop(columns=['\_id'], inplace=True)

**b. Missing Coordinates**

Latitude and longitude fields in certain records were invalid, preventing map visualization. Data validation was applied to map only records with valid coordinates.

**c. Callback Synchronization**

Mismatched input/output IDs prevented initial callback routines from returning expected outcomes. Systematic testing and callback dependency re-alignment fixed these concerns.

**d. DataTable Interactivity**

To update the DataTable based on filters and allow single-row selection, derived\_virtual\_data and derived\_virtual\_selected\_rows needed careful configuration.

**7. Conclusion**

A Python-based full-stack online application including database management, data analytics, and dynamic user interfaces is shown here. The Grazioso Salvare dashboard has all necessary features and provides a solid platform for future upgrades like full CRUD operations or cloud deployment. The client gets a scalable, interactive, and maintainable system tailored to their operational needs with MongoDB and Dash. Clear documentation, solid functionality, and modular design prepare the project for handover, maintenance, and development.

**8. Resources and Links**

**Dash Documentation**: <https://dash.plotly.com/>

**Dash Leaflet**: https://dash-leaflet.herokuapp.com/

**MongoDB Docs**: <https://www.mongodb.com/docs/>

**PyMongo**: <https://pymongo.readthedocs.io/>

**Pandas**: https://pandas.pydata.org/

**Plotly Express**: https://plotly.com/python/plotly-express/

**JupyterDash**: <https://github.com/plotly/jupyter-dash>