# CS 340 README Template

**Project Functionality Description**

The project's goal was to build an interactive data dashboard for an animal shelter using the Dash framework. The dashboard provides filtering options for rescue-type suitability (Water, Mountain, Disaster), displays animal data in a sortable and filterable table, shows breed distributions in a histogram, and maps animal locations interactively. This was achieved through MongoDB as the data backend and Dash for the web-based front end.

Key Functionalities Implemented:

* Radio button filters for different animal rescue roles
* Interactive dash\_table.DataTable supporting sorting, filtering, selection, and pagination
* Dynamic histogram showing the breed distribution of filtered results
* A geolocation map with breed and color tooltips, updating on table selection
* Dashboard layout includes a Grazioso Salvare logo and a structured page layout using Dash components

**Screenshots**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a map

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a map

AI-generated content may be incorrect.

A screenshot of a map

AI-generated content may be incorrect.

| **Tool** | **Purpose** | **Rationale** |
| --- | --- | --- |
| **Dash (by Plotly)** | Web app framework | Allows reactive UIs in Python using familiar components (HTML, DCC, Plotly, Leaflet) |
| **Dash Leaflet** | Mapping | Lightweight map rendering tool that integrates with Dash |
| **Plotly Express** | Charting (Histogram) | Quick generation of interactive visualizations |
| **Dash Table** | Interactive tabular data | Provides sorting, filtering, selection, and pagination capabilities |
| **MongoDB** | Backend storage (Model) | Stores unstructured animal data, queried via a custom AnimalShelter module |
| **Pandas** | Data handling | DataFrame abstraction allows easy tabular manipulation and conversion to/from MongoDB |
| **Base64** | Image encoding | Used to embed a PNG logo into the dashboard |
| **JupyterDash** | Runs Dash inside Jupyter | Convenient testing environment for development and prototyping |

**Why MongoDB?**

MongoDB was chosen for this project as the *Model* in the MVC structure. Its flexible, schema-less document format (BSON/JSON) suits animal records, which may vary in attributes. Specific benefits include:

* **Python Compatibility**: The pymongo library allows seamless integration between Python and MongoDB.
* **Powerful Queries**: Supports advanced filtering with Mongo-style selectors (e.g., $in, $gte, $lte).
* **Scalability**: Easily handles large amounts of data with fast indexing and read operations.
* **Ease of Use with Pandas**: Documents retrieved from Mongo can be converted directly into Pandas DataFrames.

**Dash Framework Explanation (MVC)**

Dash by Plotly is a Python framework that enables the creation of web-based data apps. It follows an MVC-style architecture:

* **Model**: MongoDB via the AnimalShelter class provides data to the view.
* **View**: Dash components like html.Div, dash\_table.DataTable, dcc.Graph, and dash\_leaflet.Map renders the interface.
* **Controller**: Dash’s callback decorators (@app.callback) handle user interactions, input changes, and update the UI dynamically.

Dash’s advantage is that it keeps the entire development in Python, enabling a single-language full stack approach.

**Resources and Software**

* [Dash Documentation](https://dash.plotly.com/)
* [Dash Leaflet](https://dash-leaflet.herokuapp.com/)
* [[MongoDB](https://www.mongodb.com/)](https://www.mongodb.com/)
* [Pandas](https://pandas.pydata.org/)
* [Plotly Express](https://plotly.com/python/plotly-express/)
* [[Base64 Encoding](https://docs.python.org/3/library/base64.html)](https://docs.python.org/3/library/base64.html)
* [Jupyter Notebook](https://jupyter.org/)
* [Jupyter Dash](https://github.com/plotly/jupyter-dash)

**Steps Taken**

1. **Setup & Imports**: Imported necessary libraries for Dash, MongoDB, Pandas, and visual components.
2. **CRUD Module Integration**: Connected to MongoDB using credentials and a custom AnimalShelter class.
3. **DataFrame Conversion**: Retrieved animal documents and preprocessed them into a DataFrame.
4. **Dashboard Layout**: Using Dash HTML and core components, I created a layout with an image, filters, table, graph, and map.
5. **Callbacks**: Defined interactivity using Dash callback functions:
   * Filter logic using Mongo queries
   * Updating the data table
   * Generating histograms by breed
   * Updating map markers dynamically
6. **Testing**: Run and debug the app within Jupyter using app.run\_server(debug=True).

**Challenges**

| **Challenge** | **Resolution** |
| --- | --- |
| **MongoDB \_id field incompatibility** | Dropped \_id column from the DataFrame to prevent table rendering issues |
| **Dynamic map integration** | Used Dash Leaflet and handled potential nulls with conditional checks before plotting |
| **Filtering by breed/age/sex** | Encoded Mongo queries in callbacks to filter data appropriately based on rescue type |
| **UI Responsiveness** | Implemented native sorting/filtering/pagination to handle larger data sets |
| **JupyterDash not rendering** | Ensured correct image encoding and server startup within notebook constraints |

## Contact

Your name: Daniel Vidmar