

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.

A screenshot of a computer

AI-generated content may be incorrect.

* **Describe the required functionality** of the project. Include the screenshots or screencast taken while testing and deploying your dashboard (Step 6) as proof that you have achieved the required functionality.
  + I have Included all the screen shot above and the required functionality that I have used in project two is MongoDB Integration which help to connects to the animal shelter database using a custom CRUD class. I have also use the filter, Interactive Data Table, Pie Chart, Map, and Logo Display functionality which I have added the screen shot above.
* **Describe the tools used to achieve this functionality and a rationale for why these tools were used.**
  + I have used the tools MongoDB served as the database, JSON-like document structure and compatibility with Python through the PyMongo library. The Dash framework was also used for building the web application. And JupyterDash was used to run the dashboards within a Jupyter Notebook which help to provide a convenient development environment.

Be sure to explain why MongoDB was used as the model component of the development, including what specific qualities or capabilities it provides for interfacing with Python.

* + MondoDB was used because it is a NoSQL database, perfect for handling large volumes of semi-structure data like animal shelter records. It is also compatible with PyMongo, allowing for smooth integration into Python application. It supports flexible scheme, ideal for a project where records may vary slightly.

Be sure to explain the Dash framework that provides the view and controller structure for the web application.

* + Dash helps us to build on top flask, Plotly, and react, allowing for rapid creation of interactive dashboards. Callbacks allow for linking data updates, UI events, and visualizations dynamically and controller Dash @callback functions manage interaction logic between components. And View handles layout and rendering with HTML- like Python code.
* **Explain the steps that were taken to complete the project.**
  + The steps that were taken to complete the projects are:

Step 1: (Database Setup) A connection was established to a remote MongoDB instance using login credentials. A custom AnimalShelter class (CRUD module) was used to interact with the database using PyMongo.

Step 2: (Data Retrieval and Cleaning) The read() method was called to fetch all animal records. The data was converted into a Pandas DataFrame, and the \_id field was removed to avoid display issued in the dashboard.

Step 3: (Dashboard Design) The layout of the dashboard was created using the Dash framework within JupyterDash. This included a Title , logo, breed filter, interactive data table, pie chart, and map component.

Step 4: (Interactivity) The Callback functionality were implemented to allow filtering of animals by breed, highlight selected data row, update the pie chart dynamically, and show geolocation on a map based on the selected row.

Step 5: (Logo Integration) The Graziozo Salvare logo was embedded using base64 encoding to ensure proper display inside the JupyterDash app.

Step 6: (Testing and Debugging) The dashboard was tested in the Jupyter environment to ensure that all components worked as intended, including breed filtering, map rendering and chart updates.

Last but not the least, Step 7: (Deployment Proof) I took Screenshot to demonstrate successful implementation and functionality of the dashboard components.

* **Identify any challenges that were encountered and explain how those challenges were overcome.**
  + One of the main challenges encountered during the project was integrating the Grazioso Salvare logo into the dashboard, as using a direct file path caused issues with image rendering in the Jupyter environment. This was resolved by encoding the image using Python’s base64 module and embedding it directly in the HTML layout using a data:image/png;base64 source string. Another challenge was handling the \_id field from MongoDB, which caused rendering errors in the data table due to its ObjectID type. This was addressed by dropping the column before passing the data to the table. In Addition, ensuring interactivity between components like filtering, chart updates, and map rendering required careful debugging of callback function and data handling to avoid NoneType or index errors.