**Animal Shelter Database and Dashboard**

**Project Overview**

This project implements a full-stack application for Grazioso Salvare, an organization that identifies and trains dogs for search-and-rescue operations. The application consists of:

1. A Python CRUD module that provides middleware functionality between MongoDB and client applications
2. An interactive dashboard that allows Grazioso Salvare staff to filter, visualize, and analyze dog data from the Austin Animal Center

The system enables quick identification of dogs suitable for specific types of rescue training (water rescue, mountain/wilderness rescue, and disaster/individual tracking) based on breed, sex, and age criteria.

**Required Functionality**

**CRUD Module Functionality**

The AnimalShelter class provides four primary methods for interacting with the MongoDB database:

* **Create**: Adds new animal records to the database
* **Read**: Retrieves animal records based on query criteria
* **Update**: Modifies existing animal records
* **Delete**: Removes animal records from the database

All operations include proper error handling, data validation, and logging.

**Dashboard Functionality**

The dashboard implements all required features specified by Grazioso Salvare:

1. **Interactive Filter Options:** 
   * Water Rescue button - filters for Labrador Retriever Mix, Chesapeake Bay Retriever, or Newfoundland breeds that are intact females between 26 and 156 weeks old
   * Mountain/Wilderness Rescue button - filters for German Shepherd, Alaskan Malamute, Old English Sheepdog, Siberian Husky, or Rottweiler breeds that are intact males between 26 and 156 weeks old
   * Disaster/Individual Tracking button - filters for Doberman Pinscher, German Shepherd, Golden Retriever, Bloodhound, or Rottweiler breeds that are intact males between 20 and 300 weeks old
   * Reset button - returns all widgets to their original, unfiltered state
2. **Dynamic Data Table:** 
   * Displays animal records that match the selected filter criteria
   * Updates automatically when filter options are selected
   * Shows key animal information including ID, breed, sex, age, and location
3. **Geolocation Chart:** 
   * Displays the locations of filtered animals on an interactive map
   * Allows users to see the geographic distribution of rescue-eligible dogs
4. **Breed Analysis Chart:** 
   * Pie chart showing the breed distribution of filtered animals
   * Provides visual analysis of which breeds are most prevalent in each rescue category
5. **Branding Elements:** 
   * Includes the Grazioso Salvare logo with a link to the client's homepage
   * Contains a unique identifier crediting the creator (Brett Plemons)

**Dashboard Demonstration**

The following screenshots demonstrate the successful implementation of the dashboard:

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Tools and Technologies**

**Database: MongoDB**

MongoDB was selected as the database component for several reasons:

1. **Document-Oriented Storage**: MongoDB's flexible JSON-like document structure is ideal for animal records that contain varying attributes and information. Unlike traditional relational databases that require a predefined schema, MongoDB allows each record to have its own structure, which is perfect for the diverse animal data from multiple shelters.
2. **Schema Flexibility**: The database can easily accommodate new fields or changes to existing data structures without requiring migrations or restructuring. This is particularly useful for animal shelter data that might evolve over time with new attributes or categories.
3. **Geospatial Capabilities**: MongoDB offers built-in geospatial indexing and query operators that made implementing the map visualization straightforward. This was crucial for showing the locations of potential rescue dogs.
4. **Query Performance**: MongoDB's indexing capabilities allow for fast querying across large datasets, which is essential when filtering through thousands of animal records based on multiple criteria (breed, age, sex).
5. **Python Integration**: The PyMongo driver provides a natural interface between Python and MongoDB, allowing data to be treated as native Python dictionaries and lists. This integration simplifies the code and improves developer productivity.

**Resources:**

* [MongoDB Documentation](https://www.mongodb.com/docs/)
* [PyMongo Documentation](https://pymongo.readthedocs.io/)

**Frontend Framework: Dash by Plotly**

Dash was chosen as the web application framework for the following reasons:

1. **MVC Architecture:** Dash implements the Model-View-Controller pattern perfectly for this application. The model (data) comes from MongoDB, the views are the Dash components (table, charts, filters), and the controller is implemented through Dash callbacks that respond to user interactions.
2. **Python-Based:** Dash allows for building interactive web applications entirely in Python, eliminating the need for separate frontend technologies. This creates a seamless development experience as both the database interface and UI are implemented in the same language.
3. **Interactive Components:** Dash provides ready-to-use components for tables, dropdowns, buttons, and charts that match Grazioso Salvare's requirements exactly. These components update reactively in response to user input.
4. **Visualization Capabilities:** The integration with Plotly graphics allows for creating the required geolocation map and pie chart visualizations with minimal code.
5. **Callback System:** Dash's callback decorator system makes it easy to connect user interface elements to data processing functions, creating a responsive application that updates all components when filters are applied.

**Resources:**

* [Dash Documentation](https://dash.plotly.com/)
* [Plotly Python Documentation](https://plotly.com/python/)

**Additional Libraries**

* **Pandas:** Used for data manipulation and preparation before visualization
* **Pydantic:** Implemented for data validation and settings management
* **Jupyter Notebook:** Used for development, testing, and demonstration
* **Poetry:** Used for project management and script utilization
* **Ruff:** Python linting and formatting to enforce alignment with PEP8 standards
* **Pydantic:** Data validation for runtime type checking
* **PyRight:** Static type checking

**Implementation Steps**

The project was completed following these steps:

1. **Requirements Analysis** 
   * Reviewed the specifications document provided by Grazioso Salvare
   * Identified key requirements for the CRUD module and dashboard
   * Analyzed the rescue type specifications and preferred dog breeds
2. **Database Setup and Configuration** 
   * Installed MongoDB and imported the Austin Animal Center dataset
   * Created appropriate user credentials for database access
   * Set up indexes to optimize query performance for filtering operations
3. **CRUD Module Development** 
   * Implemented the AnimalShelter class with comprehensive error handling
   * Created methods for creating, reading, updating, and deleting animal records
   * Added data validation for input parameters
   * Developed extensive testing to verify all operations worked correctly
4. **Query Development** 
   * Created specialized MongoDB queries for each rescue type based on the client's specifications
   * Implemented filtering for breed, sex, and age according to the requirements table
   * Optimized queries to ensure efficient data retrieval
5. **Dashboard Design and Implementation** 
   * Designed the layout with Dash HTML and Core components
   * Created interactive filter buttons for each rescue type
   * Implemented the data table with dynamic updating
   * Added geolocation mapping using Dash's Plotly integration
   * Created the breed distribution pie chart
   * Implemented callback functions to connect all components
6. **Testing and Verification** 
   * Tested all dashboard components individually and as an integrated system
   * Verified that filter buttons correctly applied the specified criteria
   * Confirmed that all visualizations updated properly in response to filter changes
   * Tested edge cases and error handling
7. **Documentation** 
   * Created comprehensive documentation including this README
   * Added screenshots demonstrating dashboard functionality
   * Documented code with appropriate comments and docstrings

**Challenges and Solutions**

Several challenges were encountered during development:

1. **Challenge:** Displaying geolocation data correctly on the map component Solution: Created a data transformation function that converted the latitude and longitude values from the database into the format required by Plotly's map component. Added fallback handling for missing location data.
2. **Challenge:** Handling the dynamic updating of all dashboard components when filter buttons were clicked Solution: Implemented a callback structure that efficiently filtered data between components. Used Dash's PreventUpdate feature to ensure components only updated when necessary, improving performance.
3. **Challenge:** Ensuring the dashboard was intuitive and user-friendly Solution: Followed the layout mockups provided by Grazioso Salvare while making small usability improvements based on testing. Added clear labels and tooltips to enhance user understanding.
4. **Challenge:** Optimizing performance for large datasets Solution: Implemented database-level filtering rather than client-side filtering, reducing the amount of data transferred. Used MongoDB's indexing capabilities to speed up queries on frequently filtered fields.

**How to Use the Application**

**Installation Requirements**

Ensure you have MongoDB installed and running on your system

Import the Austin Animal Center Outcomes dataset into MongoDB

Create a user account with appropriate permissions for database access

Install the required Python libraries:

poetry install

To run the application:

poetry run dashboard

Access the dashboard by navigating to the URL displayed in the console (typically <http://127.0.0.1:8050/>)

**Using the Dashboard**

1. Click one of the filter buttons to view dogs suitable for:
   * Water Rescue
   * Mountain or Wilderness Rescue
   * Disaster Rescue or Individual Tracking
2. Use the Reset button to clear all filters and view the complete dataset
3. Click on any row in the data table to see the corresponding location highlighted on the map
4. View the breed distribution in the pie chart, which updates automatically based on the selected filter

**Future Enhancements**

Potential future improvements include:

1. User authentication system to control access to the dashboard
2. Email notification feature for when new rescue-eligible dogs enter the system
3. Enhanced filtering options for more specific criteria
4. Mobile-responsive design for field use
5. Export functionality to download filtered data as CSV or PDF reports

**Contact**

**Brett Plemons**