Infinite Loopers

Project Description

HurriScan is an advanced hurricane tracking and scanning system designed to offer unparalleled insights into hurricane activities in the Pacific Ocean. The platform utilizes wind and temperature data from a network of strategically positioned buoys throughout the equatorial Pacific, providing hurricane tracking, detailed forecasts, and potential impact assessments.

HurriScan aims to cater to a diverse range of users, including meteorologists, climate researchers, companies engaged in shipping goods across the Pacific, and media outlets alerting coastal cities. The platform accomplishes this by delivering an interactive oceanographic data map, machine-learning-powered predictions, helpful data visualizations, and timely alerts. These features empower users to stay safe and well-informed about natural events and changes occurring throughout the Pacific.

User Groups

Group 1: The Meteorologist

- Goals: To provide accurate and timely weather updates; to understand weather patterns, especially hurricanes.
- Needs: Real-time data on hurricane locations, paths, and intensities; historical hurricane data for comparative analysis.
- Behaviors: Checks hurricane data multiple times a day, especially during hurricane season; uses data to make forecasts and weather reports.

Group 2: The Shipping Coordinator

- Goals: To ensure safe and timely delivery of goods across the Pacific; to reroute ships away from dangerous weather.
- Needs: Up-to-date information on predicted hurricanes; alerts on weather changes.
- Behaviors: Monitors weather conditions daily; adjusts shipping routes based on weather forecasts.

Group 3: The System Administrator

- Goals: To maintain and update the HurriScan platform; ensure data accuracy and system reliability.
- Needs: Efficient tools for updating and managing the database; clear feedback channels from users.
- Behaviors: Regularly reviews system performance and updates datasets

Infinite Loopers

User Requirements

Users will create an account to view the application and login with the same information in all future logins. After logging in, users may choose to log out once they are finished viewing the site.

Users will view geographical data on the user dashboard and may filter the data based on month/year and a temperature range. Users will view charts displaying humidity, air, temperature, and longitude and latitude wind speeds. Additionally, a summary chart regarding the average temperature per month can be viewed as well.

Users may navigate to the alerts page in which they can specify their region, phone number, and email address for various alerts from the application.

Users may navigate to the predictions page and select a point on the map to make a hurricane prediction based on the longitude and latitude. Users can also view the temperature, humidity, and air average for each given month in a graph on the predictions page. Finally, users can specify their location for receiving alerts for predictions.

Admins may create alerts and delete users on an admin page that is accessible only to admins.

Non-Functional Requirements

- Login and register page ensure valid credentials
- User dashboard displays geographical and chart information
- Alert preferences allow personalized notification configuration
- Predictions page has a responsive layout with clear indications of what the user can interact with.
- Admin dashboard displays only the user information to admins

Functional Requirements

- User credentials verified via the SQLite database for the login and register pages
- Hurricane data from SQLite database loaded onto maps and graphs/
- User information loaded onto admin dashboard and handling of alert creation

Infinite Loopers

Status of the software implementation:

Overall, we as a group are incredibly proud of the current status of our project, as we have successfully completed all the features we set out to finish over the project's span. After never partaking in a project of this scale, it was amazing to see the growth and sense of accomplishment upon completing these weekly tasks as a team. The minimum requirements that we were given are as follows:

- Our Alerting system, a key feature of our software, was implemented using Twilio. It utilizes our predictions model and SQLite database to send email and text message alerts to users, notifying them of hurricane activity based on their preferences.
- Visualizations (at least three different types): We implemented an interactive map, heatmap as well and chart visualizations for users and admins to interact with
- We have successfully created two distinct roles, user and Admin. Each role comes with a unique set of features and permissions, ensuring a tailored user experience for each user type.
- Filtering - Users can filter hurricane data by date and location.
- Predictions - Created a hurricane risk prediction model that relies on the previous data to predict upcoming hurricane data for a user-specified location.
- Dashboards - We implemented a user and admin dashboard, giving different permissions and features to the roles based on how we wanted these users to interact with the system.
- Login/Registration System - We implemented a login and registration system that utilizes basic validation.
- Database: Implemented an SQLite database that stores user and hurricane data, making our system usable and scalable.

Infinite Loopers

After completing our requirements, we have 0 tests left in the backlog since we have accomplished everything we set out to do!

Reflections:

Once the requirements and overall project started getting into motion, our overall project management worked very well. The hardest part was the lack of direction offered at the beginning of the course surrounding the project, which set us back at the start. The only thing we would change about our process next time would be a better understanding of our tech stack before stepping in. Better file structure and testing would have helped us save time later on, which was an exciting learning experience through this project. Initially, the initial requirements needed to be better detailed, but the project felt much more apparent once we received clarification from our TA in early February. We did not miss or overlook any of the requirements. The only thing that needed to be added in our initial planning was direction, and it would have been good if we looked more into Flask before the coding process. As a team, we will deal with testing differently in the future. Though most of our struggles here were because we lacked understanding of Pytest and Selenium, it would have been better to fix our testing process earlier (i.e. ensuring they all passed and writing more beforehand) to make the project closing a more straightforward process. Overall, the effort to complete the project was about what was expected; we did feel that being expected to put in 8 hours a week consistently was unfair since everyone has other things going on in their lives, but overall with finishing on time, I feel that we correctly understood the amount of work that needed to be put into the project. Our team is proud of the many features that we implemented, especially getting everything connected into one web app. We are most proud of our prediction model/alerting system. This is because these are both things

Infinite Loopers

that we knew very little about before starting the project, so it was interesting to learn how to do this and then see it all in action.

Step by Step Guide

Before you begin, ensure you have the following installed on your local machine:

- Python 3.x

- pip (Python package installer)

- Git Repo cloned onto your machine trevorwinser/Infinite-Loopers (github.com)

Setting Up the Virtual Environment

To set up a virtual environment enter the following in the Vscode CLI:

1. Navigate to the project directory: cd server

2. Create a virtual environment named `venv` python -m venv venv

3. Activate the virtual environment:

On Windows:

.\venv\Scripts\activate

On macOS and Linux:

source venv/bin/activate

Installing Dependencies

After activating the virtual environment, install the project dependencies by entering the following command in the CLI:

pip install -r ../requirements.txt

Infinite Loopers

This command reads the `requirements.txt` file from the project directory and installs all the necessary packages.

Setting Up the Database

1. Ensure SQLite is installed on your machine (comes pre-installed with Python).

2. Navigate to the project root directory and run the following command:

python sqlite_setup.py

Running the Flask Application

1. Ensure your virtual environment is activated.

2. Set the environment variable for the Flask application:

- On Windows:

set FLASK_APP=app.py

- On macOS and Linux:

export FLASK_APP=app.py

3. Run the Flask development server:

flask run

4. Open a web browser and navigate to http://127.0.0.1:5000/ to view the application.

Troubleshooting Issues

Include common issues and their solutions here. For example:

- If you encounter errors while installing dependencies, try updating pip: `pip install --upgrade pip` and then rerun `pip install -r requirements.txt`.

Infinite Loopers

- If the Flask app doesn't run, ensure the `FLASK_APP` environment variable is set correctly and points to `app.py`

Software Implementation

The architecture of the system centers around a web application designed to monitor and provide alerts based on data collected from IoT sensors, particularly focusing on historical hurricane data in the Pacific Ocean. The system architecture is modular, with each key component responsible for distinct functionalities, facilitating easier maintenance, updates, and potential scaling.

System Architecture and Key Components

1. Database (Singleton Design Pattern):

- The system uses a relational database to store sensor data, user information, and alert logs. The Singleton design pattern ensures that only one instance of the database connection is active at any given time, preventing unnecessary resource utilization and potential data inconsistencies.

2. Web Server:

- The backend logic is handled by a web server, built using a Flask framework for Python. This server processes requests from the front end, interacts with the database, and handles business logic such as data analysis and alert generation.

3. Frontend:

- The user interface is web-based, allowing users to interact with the system through forms, dashboards, and notifications. It communicates with the backend server via API calls.

4. Alert System (Observer Design Pattern):

Infinite Loopers

- The alert system uses the Observer pattern, where the system monitors data and, upon detecting certain conditions, notifies subscribed users through various channels (e.g., email and text messaging). This pattern allows for decoupling the alert conditions from the notification mechanism.

5. Analysis and Prediction Module:

- Implements algorithms to analyze collected data, predict future conditions, and trigger alerts. This module uses machine learning libraries for prediction and charts.js for the graph visualizations as well as Leaflet for the map.

Degree and Level of Re-use Achieved

The team was able to achieve a high degree of code reusability by modularizing the system into distinct components, utilizing libraries for common tasks (e.g., data analysis, web server setup, frontend components), and adhering to design patterns that facilitate code reuse and extension.

Reflection on Planned vs. Actual Build

The initial plan outlined a straightforward system for collecting, analyzing, and alerting on sensor data. Throughout the development, the team introduced modular components for better separation of concerns, utilized design patterns like Singleton and Observer for efficiency and maintainability, and developed a web-based UI for user interaction. While the core objectives were maintained, the actual implementation evolved to address practical concerns such as database management, scalability of the alert system, and user interface usability.

Twilio Email Testing

1. Download any new requirements in the requirements.txt file

Infinite Loopers

2. echo "export SENDGRID_API_KEY='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'" > sendgrid.env
3. echo "sendgrid.env" >> .gitignore
4. source ./sendgrid.env
5. Run ‘python -m test.test_sendgrid’
6. Run ‘python -m test.test_twilio’

Notes

- Always activate the virtual environment (`venv`) before working on the project locally.
- After pulling new changes from the repository, run `pip install -r requirements.txt` to ensure your local environment matches the project's dependencies.
- Do not push changes to the `venv` directory, `hurriscan.db`, or any other files that contain sensitive information or personal configurations.
- Before committing changes to Git, ensure no sensitive information is included, and only necessary files are added.

Troubleshooting

Include common issues and their solutions here. For example:
- If you encounter errors while installing dependencies, try updating pip: `pip install --upgrade pip` and then rerun `pip install -r requirements.txt`.
- If the Flask app doesn't run, ensure the `FLASK_APP` environment variable is set correctly and points to `app.py`.

.env folder

```
PYTHONPATH=.:${PYTHONPATH}:./server
TWILIO_ACCOUNT_SID=AC44993aed976dd5210997b2519df5a254
TWILIO_AUTH_TOKEN=dadc7ffcb6e3d6867cd51f23885375c6
SENDGRID_API_KEY=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
```