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.

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 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

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`.

- 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):

- 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.