Air Quality Final Project

Project Goal:

Create a responsive web application to monitor and display real-time air quality data, including temperature, humidity, PM2.5, PM10, NOx, NH3, CO2, SO2, and VOC levels. The application will utilize HTML, CSS, and JavaScript.

First Steps:

1. Creating a Figma project
2. Searching for free APIs
3. Getting to know Chart.js

First Implementation:

- Used WeatherAPI, Chart.js, and no enhanced backend.
- Displayed temperature, humidity, and air quality values.
- Graphs only for temperature and humidity.
- Sidebar opens a new webpage.

Problems Faced in First Implementation:

1. WeatherAPI had no historical values for air quality in the free tier.
2. Opening a new page with the sidebar resulted in repeated API calls.

Solution:

- Introduced a second API: OpenWeather API.
- Replaced new-page navigation with a scrollable main page to reduce API calls.

Second Implementation:

- Used two APIs to display historical air quality values.
- Displayed temperature, humidity, and air quality values.
- Added graphs for all air quality values, temperature, and humidity.
- Dashboard was made scrollable with the sidebar scrolling to corresponding sections.

Problems Faced in Second Implementation:

- CSS styling for responsiveness was challenging.
- Decided against using a style template to learn CSS through direct implementation.

New Ideas:

- Adding a search bar for cities, allowing users to search and update data and graphs accordingly.

Third Implementation:

- Found a city data API at OpenDataSoft
- Attempted to implement the API but failed.
- Shifted to downloading the CSV file and converting it into a SQLite database.
- Encountered issues with database size (~1GB), requiring GitHub Large File Storage (LFS).
- Hardcoded a number of cities instead.

Problems Faced After Third Implementation:

- City updates required changes in multiple functions.
- Data needed to refresh at set intervals.

New Idea:

- Include AI forecasting for CO2 values in the website.

Solution:

- Decided to use Flask for the backend to enhance functionality and learn something new.

Fourth Implementation:

- Converted existing code to Flask:
    1. API call functions moved to Python.
    2. Routes configured in `routes.py`.
    3. JavaScript modified to call the Flask backend instead of APIs directly.
    4. Updated JS and CSS script inclusion in HTML.
- Added a background scheduler to call APIs at set intervals.
- Introduced a global `cities` variable for dynamic city updates in API calls.

- Added informative videos for each pollution component using YouTube iframes.
- Included a footer.
- Added Privacy and Terms & Conditions pages using tools like Termly and TermsFeed.
- Created a Contacts page with an embedded Google Maps location (PNU).

Problems After Fourth Implementation:

- Flask backend meant hosting on GitHub Pages was no longer possible.
- AI forecasting required further testing before integration.

New Idea:

- Display additional weather data (e.g., wind speed, snow, feels-like temperature).
- Change background based on the weather in the selected city.

Solution:

- Created a new weather-specific website linked in the sidebar.

Fifth Implementation:

- Updated the sidebar with links to Weather and AI Websites.
- Updated the dashboard value boxes with icons and dynamic background that changes color depending on the value of the air pollution component
- Developed the Weather Website:
    1. Used Weatherbit API to test multiple APIs.
    2. Added functions to collect data from the API.
    3. Designed a new stylesheet.
    4. Added images for different weather situations.
    5. Implemented dynamic background changes based on API-provided weather codes.

Sixth Implementation:

- Integrated AI forecasting into the website:
    1. Added an AI Website linked via the sidebar.
    2. Created a function to fetch and append new data from the Recoglass API every 10 minutes.
    3. Added functions to display loss and prediction graphs.
    4. Implemented CO2 prediction using self-pretrained models:
        a. GRU
        b. LSTM
        c. Bidirectional LSTM

    d. CNN with LSTM
    e. Deep LSTM
5. Organized models in a `models` folder.
6. Stored `loss.json` and `prediction.csv` in `static/data` for each model.
7. Added functionality to export original Recoglass API data as a CSV file.
8. Used plotly for the graphs and not chart.js for better graphs and to try something new

Problems in Sixth Implementation:

- Displaying prediction values for different models was challenging.
- Preprocessing and postprocessing issues caused errors but were eventually debugged.
- Predictions now display correctly.

Hosting Challenges:

1. **Heroku:**
    - Added `Procfile` and `requirements.txt`, linked GitHub with Heroku.
    - Failed due to GitHub LFS support.
    - Removed LFS support using `git-filter-repo`, but Heroku didn't support some pip requirements.

2. **Vercel:**
    - Added `vercel.json` but failed to detect Flask backend.

3. **Railway:**
    - Connected to GitHub; build failed due to pip requirements.

4. **PythonAnywhere:**
    - Recreated file structure and updated file paths manually.
    - Successfully hosted, but TensorFlow version mismatch broke CO2 predictions.

Security Updates:

- Addressed API key handling issues using a `.env` file and adding it to `.gitignore`.

Final Steps:

- Added a README.md.
- Cleaned up the project by moving unused code to a `legacy_files` folder.

- Included a screenshot of the AI website view from localhost:5000 to demonstrate functionality on my machine.

Key Learnings and Takeaways:

This project provided me with valuable insights into API integration, responsive design, Flask development, AI model deployment, and hosting challenges. While encountering numerous issues, each step contributed to a deeper understanding of web development and problem-solving.

Screenshot from AI Forecasting Website: