TECHNICAL RESEARCH & DEVELOPMENT

WEATHER FORECAST SOCIAL MEDIA AUTOMATION APP

Felipe De Souza

Full Sail University - WCO | 3300 University Blvd, Winter Park FL 32792

Table of Contents

REPOSITORY

https://github.com/fdesouza1992/Weather_Automate_Forecast.git

FEATURES LIST

Current Features:

- **Real-Time Weather Data Integration:** Seamlessly fetches live weather data using the OpenWeatherMap API, ensuring up-to-date information for users.
- **User-Friendly Interface:** Provides an intuitive GUI developed with Tkinter, allowing users to input city names and generate weather updates effortlessly.
- **Secure API Key Management:** Implements best practices by storing API keys securely using the python-dotenv library, preventing exposure in the codebase.
- **Version Control with GitHub:** Utilizes Git for version control, maintaining a clear history of changes and facilitating collaborative development.Hop+2Hamed Alemohammad+2UC San Diego Library Guides+2
- **Error Handling and Notifications:** Incorporates robust error handling to alert users of invalid inputs or issues during data retrieval, enhancing user experience.

Features in Progress:

- **Automated Image Generation:** Developing functionality to overlay weather data onto pre-designed templates using the Pillow library, automating the creation of shareable weather graphics.
- **Support for Multiple Cities:** Expanding the application to handle weather data for multiple cities simultaneously, enabling batch processing and broader usability.

Planned Features for Future Iterations:

- **Cloud-Based Image Storage:** Planning to integrate cloud storage solutions, such as AWS S3 or Firebase Storage, to store generated images, facilitating easy access and sharing.
- **Automated Social Media Posting:** Aiming to implement features that allow automatic posting of weather updates to various social media platforms, streamlining content distribution.

OVERVIEW

In an increasingly digital world, social media plays a vital role in communication, marketing, and information dissemination. One niche yet consistent form of content is the daily weather update—used by small businesses, influencers, and local pages to engage their audience. However, manually designing and posting weather graphics can be time-consuming and error prone. The Weather Forecast Social Media Automation App aims to solve this problem by automating the process of generating social media-ready weather update graphics using real-time data.

This project will use Python as the primary programming language and integrate the OpenWeatherMap API to fetch live weather data. The app will also utilize the Pillow library to dynamically overlay weather information onto a professionally designed image template. For a basic user interface, the Tkinter library will be used to allow users to input a city name and trigger the generation process via a simple GUI. Development will take place in Visual Studio Code (VSCode), and all project versions and updates will be tracked using GitHub to maintain version control and support collaborative documentation. The goal is to provide a one-click solution to generate and save weather updates, creating a seamless workflow for social media managers. This document outlines the technical research and development process behind the application.

The core data source for this app is the OpenWeatherMap API, a widely used weather data service. This API offers access to current weather conditions, forecasts, and additional environmental information based on geographic location. To retrieve weather data, developers send an HTTP GET request to the API's endpoint, along with an API key and query parameters such as the city name or geographic coordinates.

Upon signing up for an OpenWeatherMap account, it can take up to three hours for the API key to be emailed to the user. Interestingly, the email includes a link to the One Call API 3.0 documentation, which provides access to historical weather data through a subscription service. However, for the purposes of this project, it was necessary to navigate back manually to the "Current Weather Data" API documentation to find the appropriate syntax and examples needed to make simple weather queries for real-time data. This caused a bit of initial confusion and added extra time to the setup process.

Once the API key was activated and the correct documentation reviewed, API requests could be properly constructed using Python’s requests library. The response from the API is returned in JSON format and includes essential data fields like temperature, humidity, weather condition descriptions, and wind speed. To better visualize this information during testing, I used jsonbeautifier.org to reformat the response, making the

structure more readable and easier to understand. This was helpful in identifying which keys and nested values to extract when building the automation logic.

In addition to navigating the documentation, there was a learning curve with Python's library management system. Before being able to use external libraries like requests, I had to understand how to physically install them via pip (Python's package installer). Without doing this first, the import statement would result in an error, which was a new concept for me coming from other environments where some libraries are preloaded.

By default, OpenWeatherMap's free tier allows for 60 API calls per minute and up to 1,000,000 calls per month. To avoid any limitations during development or testing, I applied for an educational license, which is also free for students and educators. This automatically upgraded my access to 3,000 API calls per minute and 100,000,000 monthly calls, providing much more flexibility to explore, test, and refine the app without worrying about hitting the limit.

Overall, the integration of the OpenWeatherMap API was successful, but not without initial hurdles. These challenges provided valuable insight into working with live APIs, handling JSON data, and navigating Python's ecosystem of tools and libraries.

The second major technical component of the application involves automating image generation using Python's Pillow library. Pillow not only supports the programmatic creation and manipulation of image files, but it also allows for working with pre-designed base images that can serve as templates. This feature is especially helpful for ensuring a consistent and branded appearance in the social media graphics generated by the app.

For this project, the base template images will be designed using tools like Canva or Figma to ensure a clean, professional visual presentation. These tools allow for thoughtful design choices, such as layout, spacing, and color combinations, that can then be imported into Pillow for automated text overlays and content updates.

The automation process begins with fetching the latest weather data via the API, formatting it for visual display, and rendering it on the base image template. This includes placing key information such as city name, current temperature, and weather conditions on designated parts of the image. A notable challenge during development was dealing with unit conversions, as the OpenWeatherMap API returns temperature values in Kelvin by default. To enhance user experience, I implemented a conversion formula to Fahrenheit, with support for Celsius planned in future iterations.

Design and user interaction considerations also played a key role in this phase. Beyond selecting appropriate font sizes, it became clear that font color and overall color

combinations significantly affect how users perceive and interact with the app. Poor contrast or conflicting color schemes can lead to readability issues or an unappealing design. As such, efforts are being made to select fonts and color palettes that are visually appealing and accessible.

One of the current challenges lies in saving the generated images. While the image is successfully created and previewed using Pillow, I've encountered issues ensuring that it saves properly to the local file system. This has made the idea of moving to cloud-based image storage even more appealing. Although it's not yet implemented, once the local saving process is stable, I plan to incorporate cloud storage (such as Firebase Storage or AWS S3) to facilitate the automation process further—allowing for real-time accessibility and potentially enabling direct sharing or posting from the cloud.

This phase of the project has provided great insight into the visual and technical aspects of content automation and highlighted the importance of both backend logic and front-end presentation working in harmony.

When dealing with GitHub for version control, I went ahead and created a local dev branch for development and plan to merge into the main branch as the project progresses. During this process, I discovered that hardcoding my API key directly into the .py file was creating a potential security vulnerability, especially once the code was pushed to GitHub, making the key publicly visible.

To address this issue, I found a helpful YouTube video by John Watson Rooney, which outlined two secure methods of managing sensitive keys. I decided to follow his recommendation of using the dotenv approach, which is a preferred and scalable way to store environment variables—not just for API keys but also for future credentials or secrets.

This led me to research how to properly install dotenv into my Python environment, as well as how to configure a .gitignore file to prevent sensitive files from being committed. After some trial and error, I successfully integrated this setup into my project and confirmed that the API key is now safely stored in an external .env file and not exposed in the public repository.

This experience not only strengthened the security of the research app but also gave me firsthand experience in best practices for professional version control workflows.

While this research and early development phase has been informative and productive, it has also presented several technical challenges that are still being addressed. One of the current issues involves getting the text overlay to save correctly onto the templated image using Pillow. Although the overlay displays correctly during runtime, saving the final image with the weather data fully rendered on the template has proven

more difficult than anticipated. Additionally, the automation portion of the app—which includes scheduling and batch processing—has yet to be fully explored due to time constraints and the technical learning curve.

Despite these hurdles, there have been significant triumphs that demonstrate the project's strong foundation. One of the most rewarding successes so far has been the implementation of a Graphical User Interface (GUI) using Tkinter, which allows users to input the city name dynamically. This interface not only makes the application more user-friendly but also introduces new possibilities for expanding its features. Another major win has been the integration of error handling within the Tkinter interface, which gracefully alerts users if their input is invalid or if something goes wrong during the API request.

Looking ahead, I plan to continue improving the application's usability and robustness. One future goal is to support up to five city entries, allowing users to generate multiple weather update images at once. Once the overlay saving issue is resolved, implementing cloud-based image storage will be a logical next step, further streamlining the automation process and enabling real-time sharing or scheduling for social media posting.

This phase of development has provided valuable insights into backend integration, image processing, GUI design, and error handling. While there is still work to be done, the foundation is solid, and I am excited to continue building upon it.

REFERENCES

OpenWeatherMap. (n.d.). *Current weather data*. Retrieved from https://openweathermap.org/current

GeeksforGeeks. (2021). *Python - Find current weather of any city using OpenWeatherMap API*. Retrieved from https://www.geeksforgeeks.org/python-find-current-weather-of-any-city-using-openweathermap-api/

Programiz. (n.d.). *Python Programming: JSON*. Retrieved from https://www.programiz.com/python-programming/json

GeeksforGeeks. (n.d.). *Convert JSON to Dictionary in Python*. Retrieved from https://www.geeksforgeeks.org/python-convert-json-to-dictionary/

GeeksforGeeks. (n.d.). *Convert Dictionary to JSON in Python*. Retrieved from https://www.geeksforgeeks.org/python-convert-dictionary-to-json/

JsonBeautifier. (n.d.). *Online JSON Beautifier Tool*. Retrieved from https://jsonbeautifier.org

Python.org. (n.d.). *Beginner's Guide for Programmers*. Retrieved from https://wiki.python.org/moin/BeginnersGuide/Programmers

Unwired Learning. (2023). *Python Tips and Tutorials*. Retrieved from https://blog.unwiredlearning.com/python

Visual Studio Code Marketplace. (n.d.). *GitHub RemoteHub Extension*. Retrieved from https://marketplace.visualstudio.com/items?itemName=github.remotehub

Visual Studio Code. (n.d.). *Official Website*. Retrieved from https://code.visualstudio.com

Microsoft Learn. (2023). *Clone GitHub Repository Using Visual Studio Code*. Retrieved from https://learn.microsoft.com/en-us/azure/developer/javascript/how-to/with-visual-studio-code/clone-github-repository?tabs=activity-bar

TutorialsPoint. (n.d.). *How to install Tkinter in Python*. Retrieved from https://www.tutorialspoint.com/how-to-install-tkinter-in-python#:~:text=To%20install%20Tkinter%2C%20we%20need,need%20not%20install%20it%20separately.

GeeksforGeeks. (n.d.). *Adding text on image using Python PIL*. Retrieved from https://www.geeksforgeeks.org/adding-text-on-image-using-python-pil/

Stack Overflow. (2013). *Add text on image using PIL*. Retrieved from https://stackoverflow.com/questions/16373425/add-text-on-image-using-pil

Code Maven. (n.d.). *Create images with Python PIL (Pillow)*. Retrieved from https://python.code-maven.com/create-images-with-python-pil-pillow

PyPI. (n.d.). *Pillow*. Retrieved from https://pypi.org/project/pillow/

Pillow Documentation. (n.d.). *Pillow Docs - Stable Release*. Retrieved from https://pillow.readthedocs.io/en/stable/

Pillow Documentation. (n.d.). *ImageDraw Reference*. Retrieved from https://pillow.readthedocs.io/en/stable/reference/ImageDraw.html

FontSpace. (n.d.). *Free TTF Fonts*. Retrieved from https://www.fontspace.com/category/ttf

Programiz. (n.d.). *Python datetime strftime()*. Retrieved from https://www.programiz.com/python-programming/datetime/strftime

W3Schools. (n.d.). *HTML Color Picker*. Retrieved from https://www.w3schools.com/colors/colors_picker.asp

Google. (n.d.). *How to add a degree/Fahrenheit symbol in Python*. Retrieved from https://www.google.com/search?client=safari&rls=en&q=how+to+add+a+degree%2Ffahrenheit+symbol+in+python&ie=UTF-8&oe=UTF-8

Stack Overflow. (2010). *How to get character in a string in Python*. Retrieved from https://stackoverflow.com/questions/3215168/how-to-get-character-in-a-string-in-python

Rooney, J. W. (2023). *Hide API Keys Using Dotenv in Python*. YouTube Video

GeeksforGeeks. (n.d.). *Using Python Environment Variables with python-dotenv*. Retrieved from https://www.geeksforgeeks.org/using-python-environment-variables-with-python-dotenv/

PyPI. (n.d.). *python-dotenv*. Retrieved from https://pypi.org/project/python-dotenv/

GitHub Gist. (n.d.). *.gitignore Example for Python Projects*. Retrieved from https://gist.github.com/cmarteepants/8ee1a075169dd8e45b9fc77729731840

Microsoft. (n.d.). *VSCode Python GitHub Repo: .gitignore*. Retrieved from https://github.com/microsoft/vscode-python/blob/main/.gitignore

GeeksforGeeks. (n.d.). *Python Tkinter Label Widget*. Retrieved from https://www.geeksforgeeks.org/python-tkinter-label/

TutorialsPoint. (n.d.). *Python Tkinter Label*. Retrieved from https://www.tutorialspoint.com/python/tk_label.htm