

# WeatherLy – Software Architecture Documentation

WeatherLy is a complex web application for displaying weather conditions, designed to offer current, past, and future weather information—and more—in an innovative form. It is developed in Python, using the Flask framework with the Jinja2 template engine, together with a front-end based on HTML, CSS, and JavaScript. The purpose of the application is to address people who want to quickly check the weather, travelers interested in conditions across several cities to properly prepare their trips, as well as authenticated users who want to personalize their experience by setting both their home city and favorite destinations or places for which they can receive tailored data.

The main features of WeatherLy differentiate our application from other competitors on the market, as it does not only provide simple and dry weather data—it is also intended for trip preparation and planning, acting as a sort of travel guide for users (during vacations), and as a helper in everyday “outdoor life” by offering clothing recommendations and precise, personalized predictions. Thus, the application creates a connection with the user, not being just a simple data source. Our features include displaying real-time weather for any city, retroactive access to data for up to one year (to observe patterns), long-term forecasts for the same time frame, intelligent pop-up messages depending on current conditions, and recommendations (from clothing to tourist suggestions). The application integrates a ChatBot with approximately 30 predefined questions about weather, tourism, precipitation, and other useful information (for example, you can select a type of destination, such as a sunny one, along with a potential budget, and the Bot will make recommendations based on the options chosen).

The app also allows viewing a dedicated page for the 100 most popular cities, as well as a "More details" page offering advanced weather metrics such as UV index, atmospheric pressure, precipitation, and visibility. The user system

supports authentication and registration, the creation of a personalized dashboard with saved cities, and even the implementation of certain usual routes regularly traveled by the user.

The technologies used are: Python and Flask for the backend, HTML, CSS, JavaScript, and Jinja2 (templates) for the frontend, and WeatherAPI.com for retrieving weather data, with a token valid until March 2026. Versioning is done through GitHub, with one main branch and individual feature branches for each of the two team members, and the application architecture is modular.

The architecture follows a client–server model, where the client side runs in the browser (locally: 127.0.0.1/5000) and displays the HTML pages generated by Flask, while the server side manages the application logic and communication with the external API.

The frontend is responsible for visual display (animations, interfaces, etc.), user interactions, and initiating requests to the backend, while the backend handles request processing, weather data management, ChatBot handling, user authentication, and storing personalized user information. The system interacts with the external API via HTTP requests to obtain weather data, which are processed and sent to the frontend through the Jinja2 template engine.

```
/**
```

For now, we have not included all potential front-/backend functions and files, because the application will go through several development stages, so their description is high-level and mainly orientative (in the final state we will most likely have more .py and .html files).

```
**/
```

The backend is organized modularly, with the main file app.py controlling the “behind-the-scenes” functionality of the application. The module weather\_data.py handles communication with the API, requests, and JSON responses. The chatbot.py module manages everything related to the ChatBot

(predefined questions, options, and various easter eggs ☺), while auth.py manages user authentication and registration, handling the mini “database.” We will also have common functions for these components placed in utils.py, and city\_manager.py is dedicated to managing the cities, Top 100, and all other destination-related options and recommendations. The frontend is structured in the “templates” and “static” folders, where we will have the HTML pages, CSS files, JavaScript logic, animations, and other dependencies, images, etc. (styling area) (it remains to be seen whether animations will be imported externally or created with Blender). Communication between frontend and backend is done through HTTP requests, the backend communicating with the external API through JSON-type requests as well, the processed data being inserted into Jinja2 templates for display to the final user.

The application runs locally via Flask at the address localhost:5000, and the environments required for building and running the application are: Python, Flask, a valid WeatherAPI token, and a browser (Chrome, Firefox, Edge, etc.). The application can also be containerized using Docker, with a single container requiring the application code and port 5000. Finally, testing and running the application for functionality verification will be performed manually. Python has simple libraries for handling JSON and HTTP requests, which allows efficient integration with WeatherAPI and easy manipulation of the data received. Python and Flask offer excellent support for using request libraries, facilitating stable and fast communication with APIs such as WeatherAPI. Docker ensures that the application runs identically on any system, eliminating issues caused by differences in environments and facilitating testing and distribution. The application can be easily containerized because Flask and Python require only a minimal environment to run. Flask manages the server logic and access to APIs, while HTML/CSS/JS handle the interface, allowing each component to be developed and modified independently without affecting the rest of the system. WeatherLy is a medium-difficulty project whose main advantages are its innovative and interactive graphical interface and the variety of useful options offered to the user, aiming to keep them engaged.

We estimate a development time of approximately 50–60 hours for the full implementation of all features. To accomplish all of the above, we believe an advanced level of Python knowledge is required, as the application includes fairly complex modules (such as user data management and ChatBot implementation), as well as medium-to-advanced frontend knowledge (HTML, CSS, and JavaScript), due to our desire to implement an intuitive and visually appealing interface.

Therefore, we can say that WeatherLy, at least at the project level, is a rather interesting attempt to reinterpret (and quite feature-rich compared to) traditional weather applications, making the user's life a bit easier—helping them at least walk out the door prepared ☺.

Github link: <https://github.com/SanduStefan/Project-IA4---WeatherLy>

Members:

Iosif Ianis-Cosmin 321CC

Sandu Bogdan-Ştefan 321CC