

Development Paper: Weather Chatbot

Table of contents

1. Implementation summary	1
2. System Architecture	3
2.1 Google Dialogflow ES (GDF)	3
2.2 Flask-based Python Webhook	3
2.3 OWM (OpenWeatherMap API)	3
3. Development Process	4
3.1 Setting Up GDF	4
3.1.1 Intents	4
3.1.2 Entities	6
3.1.3 Rich Content	7
3.1.4 Context	8
3.1.5 Fulfillment	8
3.2 Implementing the Flask-based Python Webhook	9
3.2.1 Python File Descriptions	9
3.2.2 Deployment	9
4. Conclusion	10
5. Reference	10

1. Implementation summary

This paper presents the development phase of the weather chatbot, as outlined in the conception paper (see Figure 1 - Sequence diagram user intents and architecture). The chatbot is designed to provide real weather information through a modular architecture that integrates natural language processing (NLP), API-based data retrieval, and structured response generation.

The implementation follows a modular structure to ensure flexibility and scalability. GDF serves as the NLP engine, enabling user intent recognition and interaction management, as well as providing a graphical user interface (GUI). A Flask-based Python webhook functions as the backend, facilitating API requests and response formatting. Real weather data, including weather conditions (e.g., snow, rain, cloudy), temperature, and wind speed, is retrieved via OWM. The chatbot was developed with logging functionalities for debugging; every intent and output will be kept in a log file and will serve in the long run as new data input for chatbot improvement. Testing was conducted to ensure accuracy and reliability. A small test set was added on the [hosted website](#), as well as a bigger [test set](#) included within the GitHub repository, which can be adapted and enlarged for more testing purposes.

The machine learning training data (intents and entities), images, test questions, and source code are publicly available in my [GitHub repository](#).

For easy access, the chatbot has been publicly deployed on this [website](#), allowing users to interact with it without requiring manual setup, which is outlined in Chapter 3.

While implementing the chatbot, I faced a couple of challenges. First of all, I wanted to ensure seamless communication similar to ChatGPT because the majority of users are already accustomed to it. As a result, users take for granted that a chatbot should have seamless and context-keeping behavior. Unfortunately, this is not entirely possible, as even weather-related interactions can become complex with shifting contexts and intent levels. Although GDF performs well in NLP, even on a small training dataset of fewer than 200 intents, there is still something missing that prevents conversations from feeling like natural communication, as is possible with large language models. Another mind-boggling issue was handling different time formats. GDF's sys.date-time entity provides four different time format responses, which is why I created a [date handler](#) within the web application to extract the correct time frame and respond accordingly to the user's intent.

Weather Chatbot Testing Page

This chatbot provides weather data updates and answers waether related queries. Below are some example questions and expected responses.

Test Questions

Test Question	Expected Outcome
Hello how are you?	Example answer: I'm always feeling bright and ready to deliver forecasts! How about you?
I need help!	Example answer: Whether it's heavy rain, strong winds, or perfect beach weather, let me know what you're looking for!
What's the weather like in Bangkok?	Bangkok: {day name}, {month} {day}, {year} Weather: {weather condition} Temperature: {min temp} → {max temp} Wind Speed: {wind speed} in m/s
Will it rain tomorrow in Phuket?	Phuket: {day name}, {month} {day}, {year} {weahter icon - condition} Weather: If yes: {weather condition} Else: No {weather condition}
Will it be warm in Munich?	Munich: {day name}, {month} {day}, {year} Temperature: {min temp} Temperature: {max temp}
What's the UV index today?	Example answer: Sorry, I can only help with weather updates. Try asking something like: 'What's the temperature in Bangkok?'

Hello! 🌤️ Wondering about the weather?
Just tap below to find out! 🌤️

Figure 1: Hosted Weather Chatbot - Website

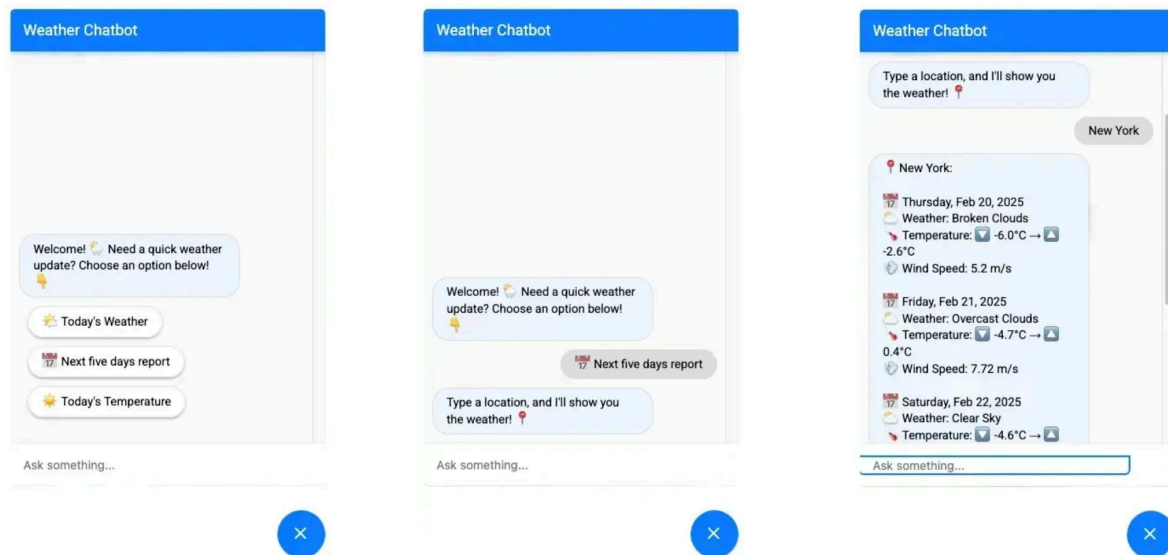


Figure 2- Weather Chatbot - Dialog example

2. System Architecture

The chatbot architecture (see Figure 1 - conception paper) has three main components, ensuring maintainability, scalability, and efficiency through separation of concerns. GDF handles NLP and ensures understanding English intents. The webhook connects to OWM for structured, weather forecasts, providing users with accurate and well-presented weather updates.

2.1 Google Dialogflow ES (GDF)

GDF handles NLP, enabling conversational interactions and providing a GUI. The chatbot can be integrated into various platforms (Google 2025b); for testing, I deployed it on a website. GDF Messenger integration can also be activated via the GDF console (Google 2025a). However, the core function of GDF is training the NLP model to improve intent and entity recognition, enhancing the user experience (Google 2025c).

2.2 Flask-based Python Webhook

The webhook bridges GDF and OWM, processing user queries, retrieving weather data, formatting responses, and ensuring seamless communication between the chatbot and external services.

2.3 OWM (OpenWeatherMap API)

OWM is the primary weather data source. This project uses its free forecast API, which provides a 5-day forecast with 3-hour intervals for any global location (OWM 2025).

3. Development Process

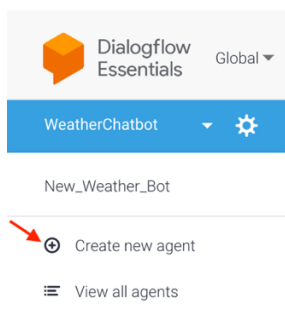
This chapter provides a step-by-step guide of the development process. I will explain how to set up the weather chatbot with a locally running webhook and an activated DGF Messenger inside the GDF console.

3.1 Setting Up GDF

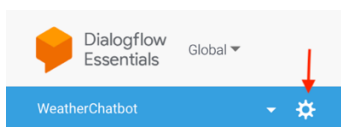
GDF is responsible for providing an interface, handling user intents, identifying entities, calling the webhook, and generating responses.

Here are the steps to set up GDF:

1. Open [GDF](#)
2. Sign in with your Google account. If no Google account exist, [here](#) you can create one
3. Click on “Create new Agent”



4. Enter the Agent Name (e.g., “WeahterChatbot”)
5. Set Default Language to “English – en” to meet the requirements
6. Select Google Project and “create a new one” to continue
7. Click “Create” to initialize the proect
8. Optional: Click on “Settings”



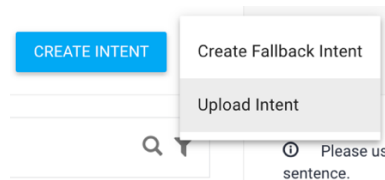
9. Optional: Adjust “Project Avatar” and the “Description”, click on “Save”

After a successfully initializing the GDF Weather Chatbot, the next step is training the model with intents and entities.

3.1.1 Intents

Intents are training data for the machine learning model. All intents are available [here](#).

1. Go to Intents in the left menu
2. Click on the three dots next to the “Create Intent” button and then click on “Upload Intent”



3. The intents in the specified intent folder must be uploaded individually
4. After successfully uploading all intents, the list will contain 13 intents, which will be briefly described

Intent	Description
welcome	Training set for Greeting intents like Hello and Hi. The response of the chatbot is a welcome greeting plus three pills, that suggest actions (see 3.1.3)
weather	It is the main intent that primarily handles all weather-related queries. My goal was to develop a seamless and intuitive chatbot where the only mandatory input required to request weather data is the city name.
city.followup	If the user has already requested: <i>“forecast for the next three days in Moscow”</i> and now wants to know the forecast for Cologne, users can simply type <i>“and in Cologne”</i> The chatbot will understand the context and provide the requested weather data accordingly.
condition.followup	If the user has already requested the weather <i>“forecast for today in Dubai”</i> and now wants to know if it will rain? Users can simply type <i>“will it rain?”</i> . The chatbot will understand the context and provide the answer.
date.followup	If the user has already requested the weather <i>“forecast for today in Munich”</i> and now wants to know how the weather will

	be on the next day, the user can simply ask: <i>“and for the next day?”</i>
past.forecast	If the user requests for a date in the past. The system message will inform the user that only present and future dates are possible to request.
temperature.followup	If the user has already requested the weather <i>“forecast for the next 2 days in Maastricht”</i> and now wants to know if it is hot? Users can simply type <i>“is it hot?”</i> . The chatbot will understand the context and provide the answer.
windspeed.followup	If the user already requested <i>“weather data for Berlin”</i> but only wants to see the wind speed, the customer can type: <i>“And the windspeed”</i>
help	Offers the user help, by showing three pills (see 3.1.3) and highlighting, that it is a weather chatbot
fallback	When the user types something in, that the bot does not understand, then the bot replies friendly and offer three pills (see 3.1.3).
feedback.negative	If the customer is not happy with the bot, the bot replies friendly.
feedback.positive	If the customer is happy with the bot, the bot replies in a very positive way
goodbye	The bot replies in a friendly way to say goodbye

Table 1: Intent Description

3.1.2 Entities

Entities are training data for the machine learning model. All entities are available [here](#).

1. Go to Entities in the left menu
2. Click on the three dots next to the “Create Entities” button and click on “Upload Entity”



3. After successfully uploading all entities, the list will contain four custom entities

Entity	Description
past	This entity contains words that refer to the past. I created this time-tense entity because the sys.date-time entity in GDF also recognizes past dates, whereas I only need present and future dates. By using this dedicated entity, I can respond to user intents such as: <i>“What was the weather in Berlin last week?”</i> on GDF level
temperature	To request the temperature of a city, I created a dedicated entity called temperature. It includes various terms related to temperature, allowing users to ask questions like <i>“Is it warm today in London?”</i> or <i>“Will it be super-hot in Ouagadougou tonight?”</i>
weather-condition	This entity is used to handle queries related to weather conditions, enabling the chatbot to respond to questions such as: <i>“Will it rain later on?”</i> or <i>“Will it be snowy in London within the next five days?”</i>
wind-speed	This entity is used to handle queries related to wind conditions, enabling the chatbot to respond to questions such as: <i>“Will it be windy?”</i> or <i>“What’s the air movement speed in Tokyo right now?”</i>

Table 2: Entity Description

In addition to custom entities, I use GDF’s built-in entities, specifically sys.geo-city and sys.date-time. The sys.geo-city entity recognizes city names, essential for requesting weather data and is the only mandatory entity in the chatbot. All other entities are optional, allowing for a more natural query style and reducing unnecessary loops before users receive a response. The sys.date-time entity specifies the requested time frame for weather data; if not provided, the webhook defaults it to today.

3.1.3 Rich Content

To display pill-style options within the GDF messenger, I had to add a custom payload to each intent where I found it beneficial to offer these options to the user.

```
{
  "richContent": [
    [
      {
        "type": "chips",
        "options": [
          {
            "text": "☀️ Today's Weather"
          },
          {
            "text": "📅 Next five days report"
          },
          {
            "text": "☀️ Today's Temperature"
          }
        ]
      }
    ]
  ]
}
```

The following intents contain the custom payload: welcome, help, and fallback.

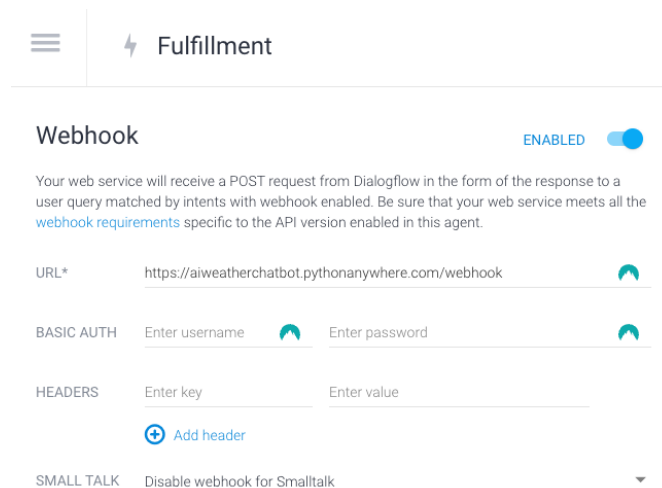
3.1.4 Context

To maintain context, I use only the weather-check context. This context retains the user's query along with previously identified entities and passes them to the next intent. As a result, follow-up questions such as *"Will it rain?"* can be understood in relation to a prior query like *"Tell me if it is going to be hot from today until the next three days in Hamburg."*

The weather-check context is carried over to all follow-up intents, ensuring a seamless and intuitive conversation flow.

3.1.5 Fulfillment

In the left menu there is the Fulfillment section. This section is important for linking the webhook URL.



The screenshot shows the 'Fulfillment' section in the Dialogflow console. The 'Webhook' toggle is turned on (ENABLED). Below the toggle, there is a description: 'Your web service will receive a POST request from Dialogflow in the form of the response to a user query matched by intents with webhook enabled. Be sure that your web service meets all the [webhook requirements](#) specific to the API version enabled in this agent.' The 'URL*' field is set to 'https://aiweatherchatbot.pythonanywhere.com/webhook'. The 'BASIC AUTH' section has fields for 'Enter username' and 'Enter password'. The 'HEADERS' section has fields for 'Enter key' and 'Enter value', with an 'Add header' button below. At the bottom, there is a 'SMALL TALK' dropdown menu with the option 'Disable webhook for Smalltalk'.

After linking, enabling, and saving the webhook, each intent that should respond with weather data must be explicitly enabled for fulfillment. In particular, this applies to all follow-up intents and the weather intent. However, only the weather intent also requires the "Enable webhook call for slot filling" option to be activated. This ensures that the webhook is triggered when the city is not specified or when the timeframe exceeds five days from today.

3.2 Implementing the Flask-based Python Webhook

The Webhook is a Flask-based web service designed to handle weather-related queries. It integrates with OWM to fetch weather data and responds to user queries via a webhook endpoint. The application also supports dialog management and date handling for user requests.

3.2.1 Python File Descriptions

app.py - This is the main entry point for the Flask application. It includes:

- A root endpoint (/) to verify the server status and hosting the chatbot for test purposes
- A /webhook endpoint to process user queries from GDF.
- Integration with Weather and DialogHandler classes.

web_application.py - Contains static HTML content for a test page and includes a script to start the GDF Messenger

date_handler.py - This module is responsible for:

- Parsing and validating date ranges in user queries.
- Ensuring requested dates fall within an allowable range (e.g., up to 5 days ahead).
- Utilizing DialogHandler for response generation to handle errors.

dialog.py - Handles user dialog management:

- Generates responses based on user intent, city, weather conditions, temperature, and wind speed.
- Uses a predefined set of response templates.
- Integrates with precipitation.py for condition-specific emoji representation.

precipitation.py - Defines a mapping of weather conditions to relevant emojis for user-friendly responses.

weather.py - Manages interaction with the OWM:

- Retrieves weather data using an API key stored in an .env file.
- Parses and formats the response before passing it to the chatbot.
- Integrates with DialogHandler and DateHandler.

init .py – The init file inside the weather_condition module will activate the logging function. Every user intent and system output will be stored in a log file (weather_query.log) Based on this information, the intents and the model can be updated over time.

3.2.2 Deployment

To deploy the webhook and run it locally with [Ngrok](#)

1. Install Ngrok
2. git clone https://github.com/MKalder/ai_weather_chatbot.git

3. `cd ai_weather_bot`
4. Optional, create a virtual python environment
5. Install dependencies using `pip install -r requirements.txt`
6. Set up an `.env` file with the OWM API key, which can be created [here](#)
7. Run the Flask application using `python app.py`
8. Run `ngrok --ngrok http 5050`
9. Link the ngrok URL in GDF Fulfillment section
10. Activate all intents for Fulfillment in GDF

4. Conclusion

The development phase successfully implemented a functional and scalable weather chatbot. By utilizing GDF for NLP and a GUI, a Flask-based Python webhook for data processing, and OWM for weather information, the chatbot provides accurate and user-friendly weather updates. Despite challenges such as handling multiple time formats and maintaining context across interactions, the modular architecture ensures flexibility and scalability. Future improvements could focus on enhancing contextual awareness and integrating more advanced natural language models to improve conversational fluidity.

5. Reference

- Google. (2025a, February 18). Dialogflow Console overview. <https://cloud.google.com/dialogflow/es/docs/console>
- Google. (2025b, February 18). Integration. <https://cloud.google.com/dialogflow/es/docs/integrations>
- Google. (2025c, February 18). Training . <https://cloud.google.com/dialogflow/es/docs/training>
- OWM. (2025, February 18). 5 day weather forecast. <https://openweathermap.org/forecast5>