Ai Chatbots & Agents

A smarter way to learn Dialogflow

P2P Clouds

DialogFlow Agents

Chapter 1 : Getting Started

DialogFlow History: Dialogflow, originally known as API.AI, was founded in 2010 by Speaktoit. It was initially developed to provide conversational interfaces for virtual assistants and chatbots. Google acquired API.AI in 2016 and rebranded it as Dialogflow, integrating it with its cloud ecosystem to enhance its natural language processing (NLP) capabilities

Since then, Dialogflow has evolved into a powerful platform for building AI-powered chatbots and voice assistants across multiple channels. Over time, Google introduced new features, including the transition from Dialogflow ES (Essentials) to Dialogflow CX (Customer Experience), offering more advanced conversational flows and state management.

Reference: https://cloud.google.com/dialogflow/es/docs/history

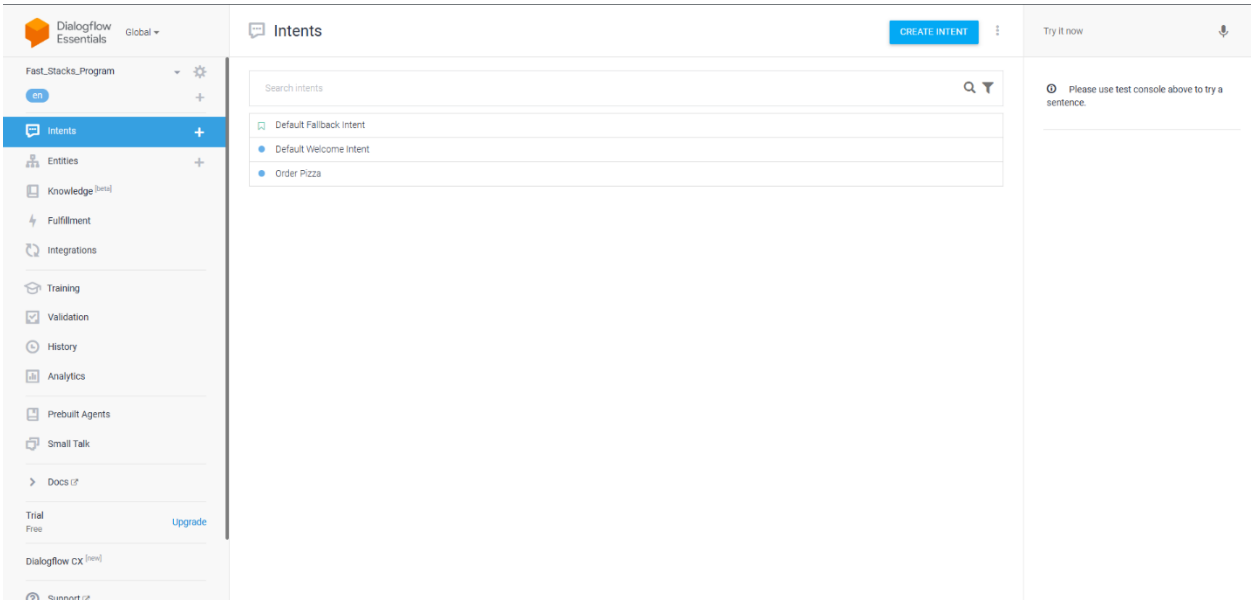
Dialogflow Introduction

Dialogflow is a cloud-based platform that enables developers to design and implement conversational interfaces using natural language understanding (NLU) and machine learning (ML). It helps businesses create AI-driven chatbots and virtual agents that can interact with users via various communication channels, such as websites, mobile apps, messaging apps, and voice-enabled devices. Dialogflow supports multiple languages, making it a global solution for conversational AI.

Capabilities of Dialogflow

- **Natural Language Understanding (NLU):** Detects user intent and extracts relevant entities from conversations.
- **Omnichannel Integration:** Deploy chatbots on platforms like Google Assistant, Facebook Messenger, Slack, WhatsApp, and more.
- **Context Management:** Maintains conversation context to provide personalized interactions.
- **Fulfillment:** Connects with backend systems via webhooks to fetch real-time data and execute business logic.
- **Multi-language Support:** Offers support for multiple languages to cater to global users.
- **Rich Responses:** Enables multimedia-rich interactions such as images, cards, and quick replies.
- **Analytics and Insights:** Provides metrics and logs to track chatbot performance and user engagement.

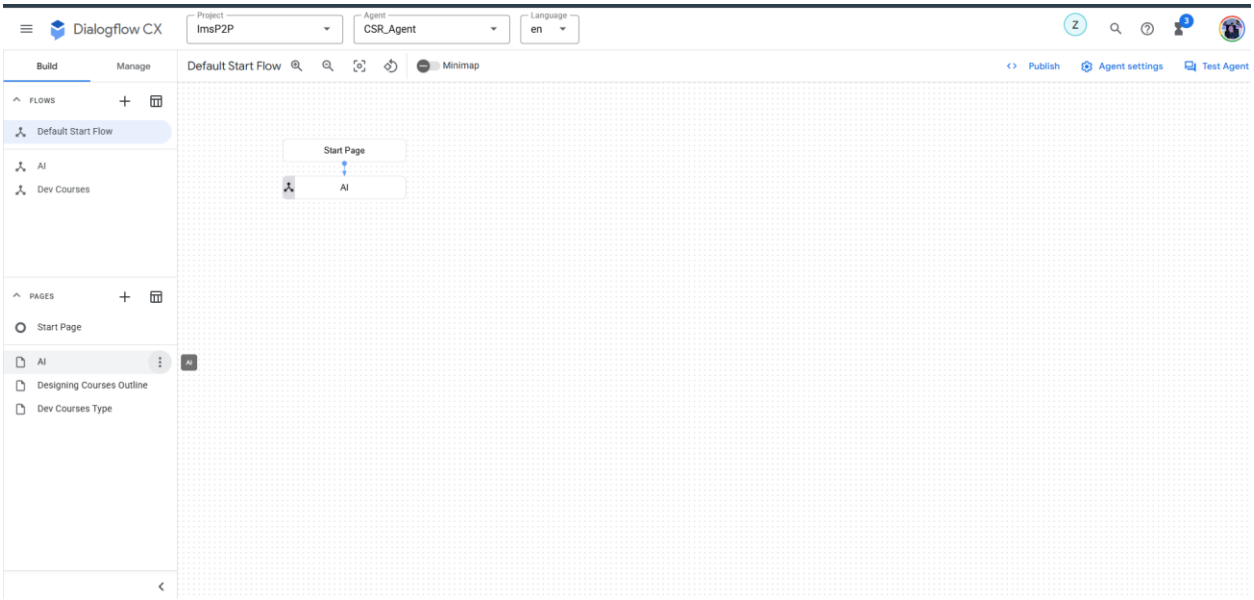
Difference between Dialogflow ES and CX



Dialogflow ES (Essentials):

1. Designed for simpler chatbot implementations.
2. Uses intents and contexts to manage conversations.
3. Best for small to medium-sized projects with linear workflows.
4. Limited state management capabilities.
5. Easier to set up and requires minimal training.

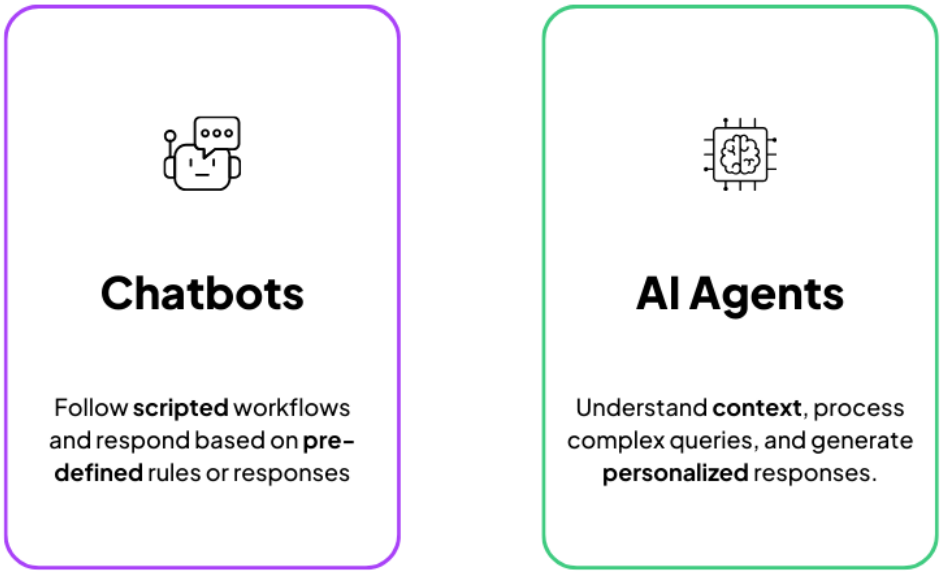
Dialogflow CX (Customer Experience):



https://dialogflow.cloud.google.com/cx/

- Provides advanced features with state machine-based flows.
- Designed for complex, large-scale conversational applications.
- Supports a visual flow builder for intuitive design.
- Allows for better scalability and more granular control over conversation paths.
- Ideal for enterprises needing sophisticated conversational flows.

Chatbots and Ai Agents



A **chatbot** is a software application that simulates human conversation through text or voice interactions. It typically follows predefined scripts or simple decision trees to respond to user inputs.

An **AI agent**, on the other hand, is a more advanced version of a chatbot that leverages machine learning, natural language processing, and artificial intelligence to understand user intent, maintain context, and provide more intelligent, context-aware responses.

How to Start with Dialogflow Tool

- **Sign up for Google Cloud Platform (GCP):**
    - Visit the Google Cloud Console and create an account.
- **Enable Dialogflow API:**
    - Navigate to the API & Services section and enable the Dialogflow API.

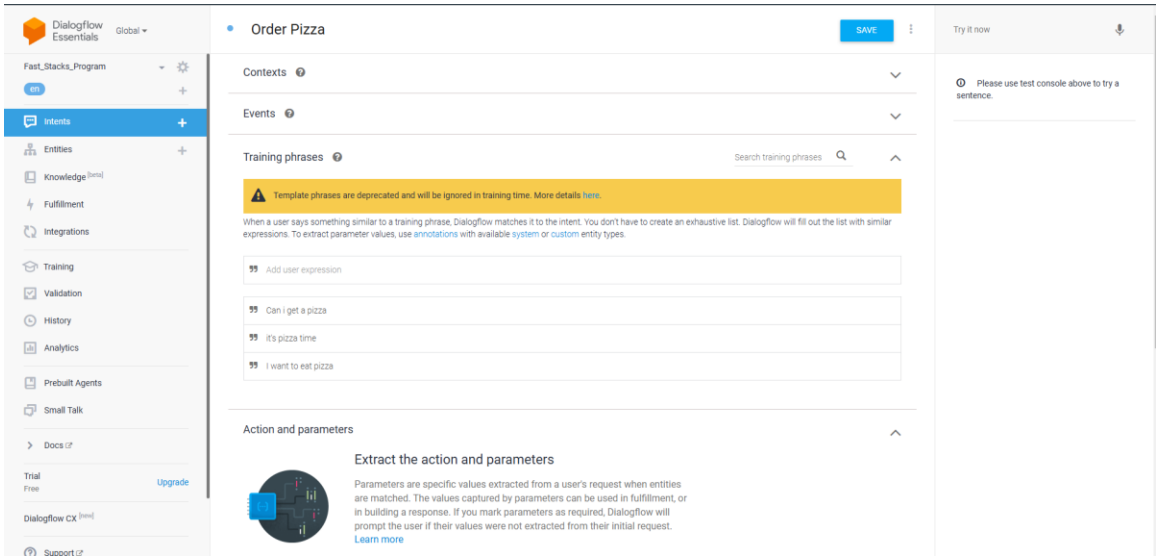
- **Access Dialogflow Console:**
    - Go to Dialogflow Console and create a new agent.
- **Create an Agent:**
    - Define a project, set the default language, and choose a time zone.
- **Define Intents:**
    - Create intents to handle different user queries.
- **Train the Agent:**
    - Add training phrases and responses to fine-tune the chatbot's understanding.
- **Integrate with Platforms:**
    - Use built-in integrations to deploy the agent on platforms like Google Assistant, Facebook Messenger, and more.
- **Testing and Optimization:**
    - Use the Dialogflow simulator to test responses and refine the agent based on user interactions.

What are Intents with Example

Intents in Dialogflow represent the mapping between what a user says and how the system should respond. They are used to categorize user requests and provide appropriate responses. Each intent contains training phrases, responses, and optional actions.

Example: Pizza Order Intent

Intent Name: OrderPizza



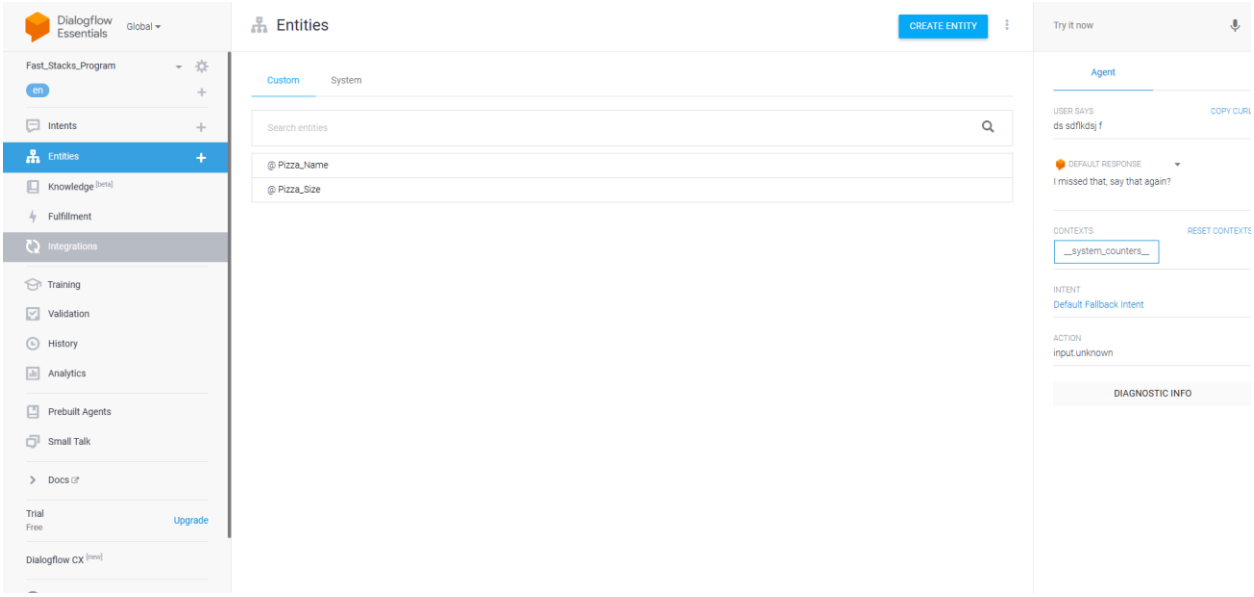
Training Phrases:

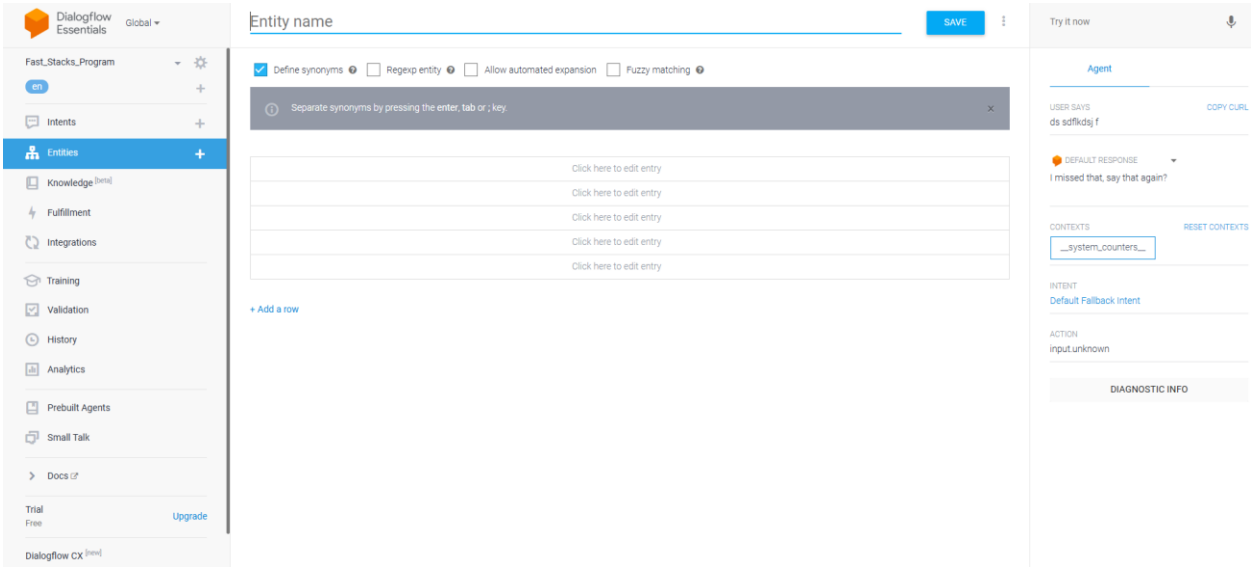
- I want to order a pizza
- Can I get a large pepperoni pizza?
- I'd like to order two medium-sized pizzas

Response:

- Sure! You choosen large one pepporoni pizza?
- Great choice! Would you like to add any drinks?

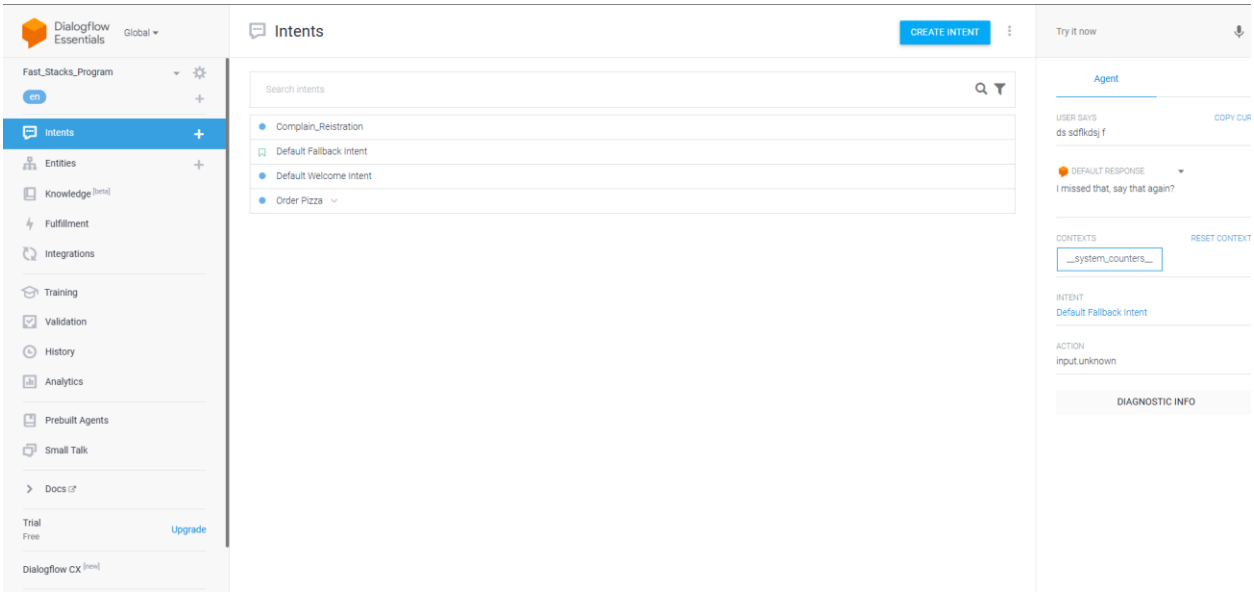
By setting up an intent like this, Dialogflow can handle different ways users request a pizza order and respond accordingly.

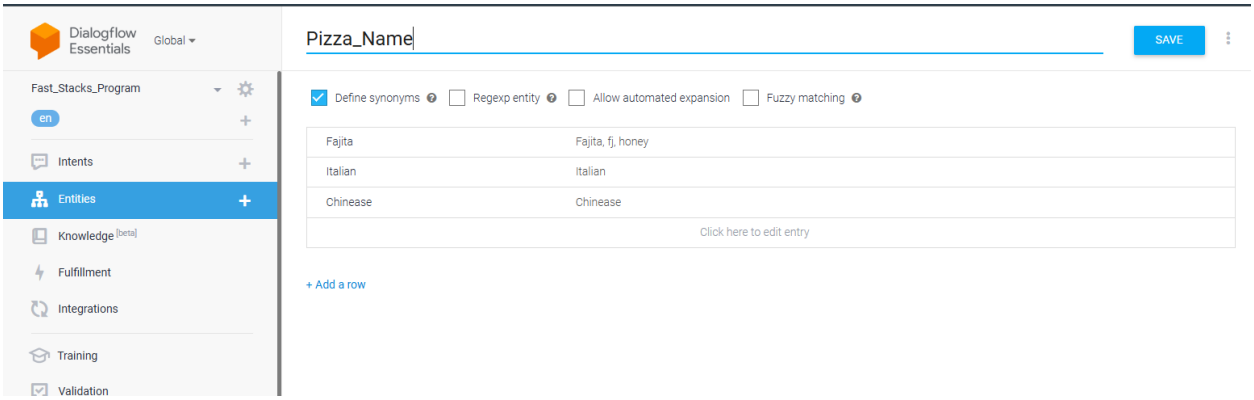


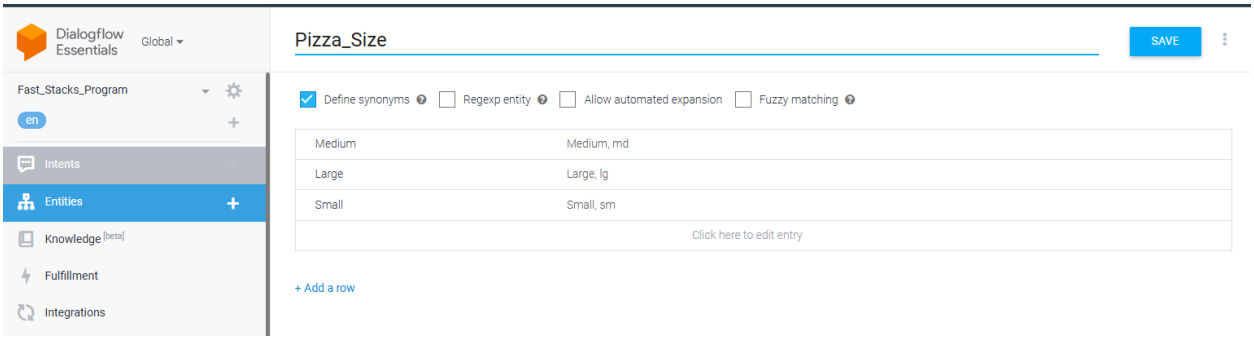


★ **Create Entities:**

- Go to the Dialogflow console and navigate to the "Entities" section.
- Create an entity named pizza_name with values like "pepperoni," "margherita," "veggie," etc.
- Create another entity named pizza_size with values like "small," "medium," "large."





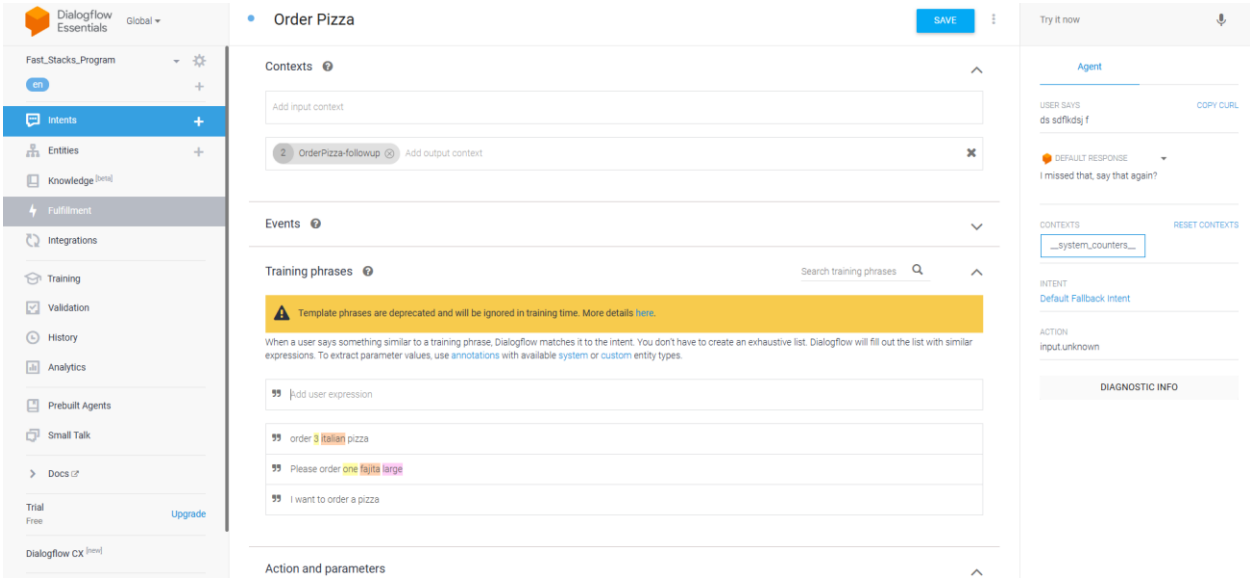


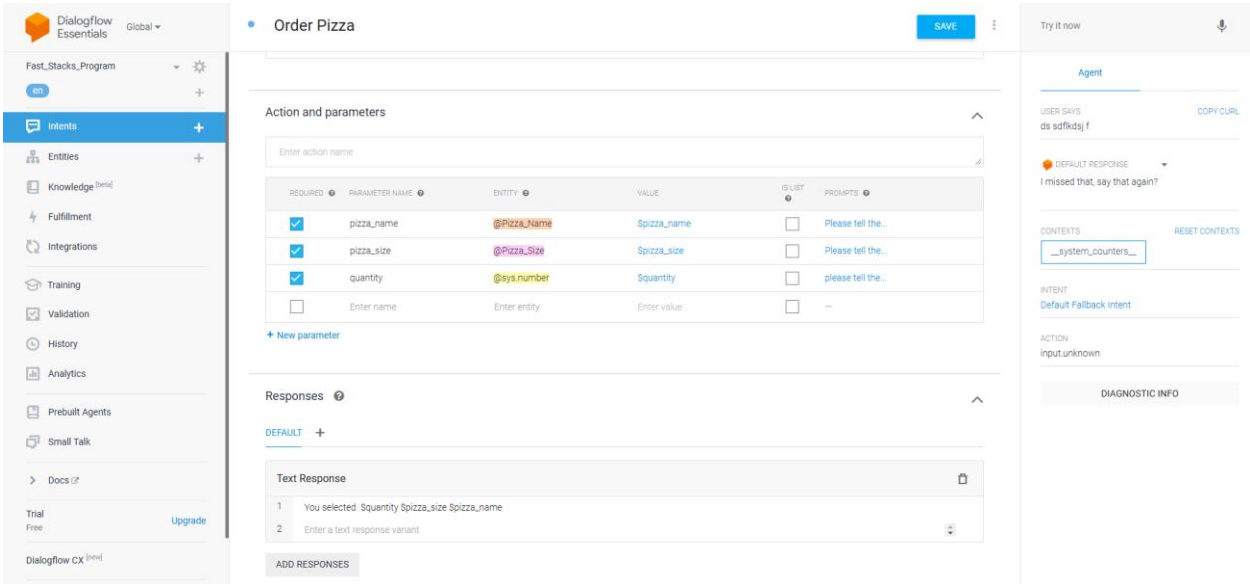
★ **Define an Order Pizza Intent:**

- Create an intent named OrderPizza.
- Add training phrases such as:

- "I want a medium pepperoni pizza."
- "Can I get a large margherita?"

- Use the pizza_name and pizza_size entities in the training phrases.





★ **Create the order_pizza Entity:**

- Define a composite entity called order_pizza.
- Include pizza_name and pizza_size as attributes.

★ **Configure Fulfillment:**

- Enable webhook fulfillment to connect the chatbot with a backend system for processing orders.

★ **Test the Agent:**

- Use the simulator in Dialogflow to test user inputs and verify correct responses.

Contexts

Contexts are an essential feature in Dialogflow that help manage the flow of conversation by maintaining relevant information across multiple interactions. They allow the chatbot to understand and remember what the user has said earlier and use it to provide meaningful responses.

There are two types of contexts in Dialogflow:

Input Contexts

Input contexts are used to control which intents should be triggered based on the current conversation context. They help the agent identify relevant intents when multiple intents could match a user's query.

Example: If a user says "I want to order a pizza," an input context named order-pizza can be set. Subsequent intents, such as choosing a pizza type or size, will only be triggered if the input context is active.

Output Contexts

Output contexts are used to maintain state and pass information from one intent to another. They help in carrying forward user-provided data through the conversation.

Example: After the user specifies the pizza type, an output context can be set to retain that information for the next intent, such as selecting the pizza size.

Adding Input and Output Contexts in Dialogflow

To add contexts in Dialogflow:

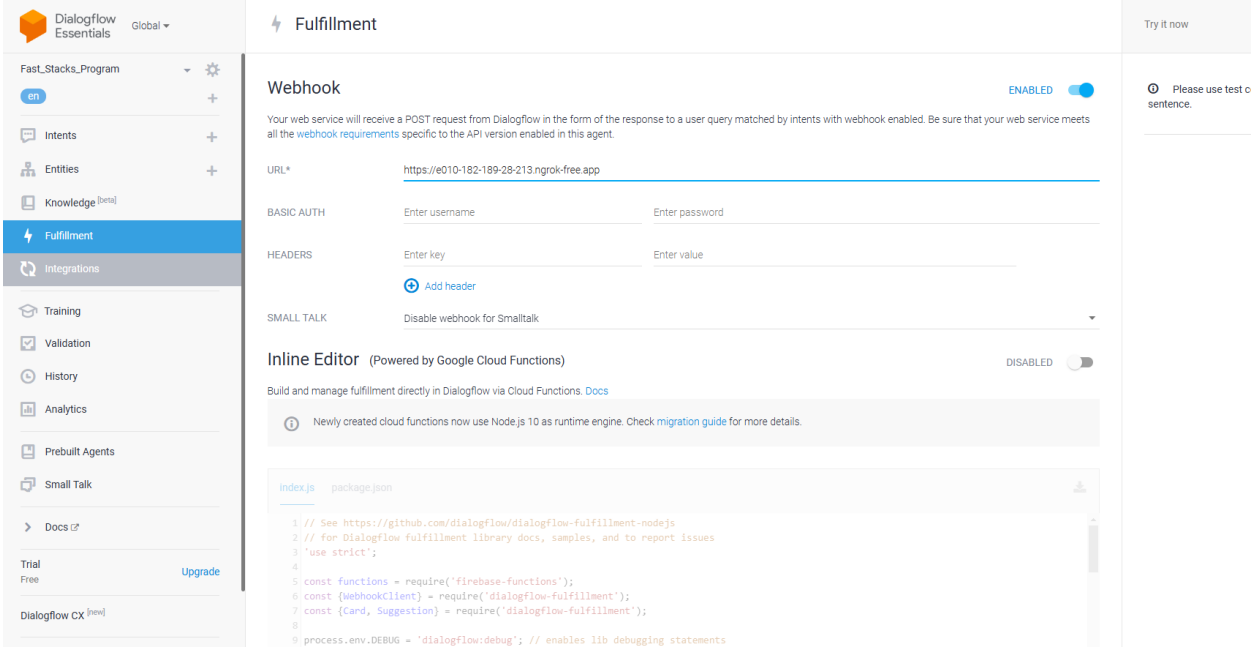
1. Go to the Dialogflow console and open the desired intent.

2. In the "Contexts" section, add an **input context** to restrict the intent to a specific conversational state.
3. Add an **output context** to maintain state across interactions.
4. Test the agent to ensure the context is being set and carried correctly.

Using contexts effectively helps create smooth and personalized conversations that feel natural and efficient to the user.

Chapter 2: Fulfillment and Webhooks with FastAPI and ngrok

What is Fulfillment in Dialogflow ES?



Fulfillment is the mechanism that allows your Dialogflow agent to interact with external services. It lets you perform dynamic actions such as querying a database, calling APIs, or responding with personalized content.

Key Components of Fulfillment:

- **Webhook**: An HTTP endpoint that processes requests sent by Dialogflow.
- **Request and Response Structure**: JSON format for communication between Dialogflow and your webhook.
- **Enablement**: Fulfillment must be enabled on the Dialogflow Console at the intent level.

Building a Webhook with FastAPI

FastAPI is a high-performance, easy-to-use Python framework ideal for building webhooks for Dialogflow.

Fast API Doc: https://fastapi.tiangolo.com/

Uvicorn: https://www.uvicorn.org/

Step 1: Setting Up FastAPI

Install FastAPI and Uvicorn:

```
pip install fastapi uvicorn
```

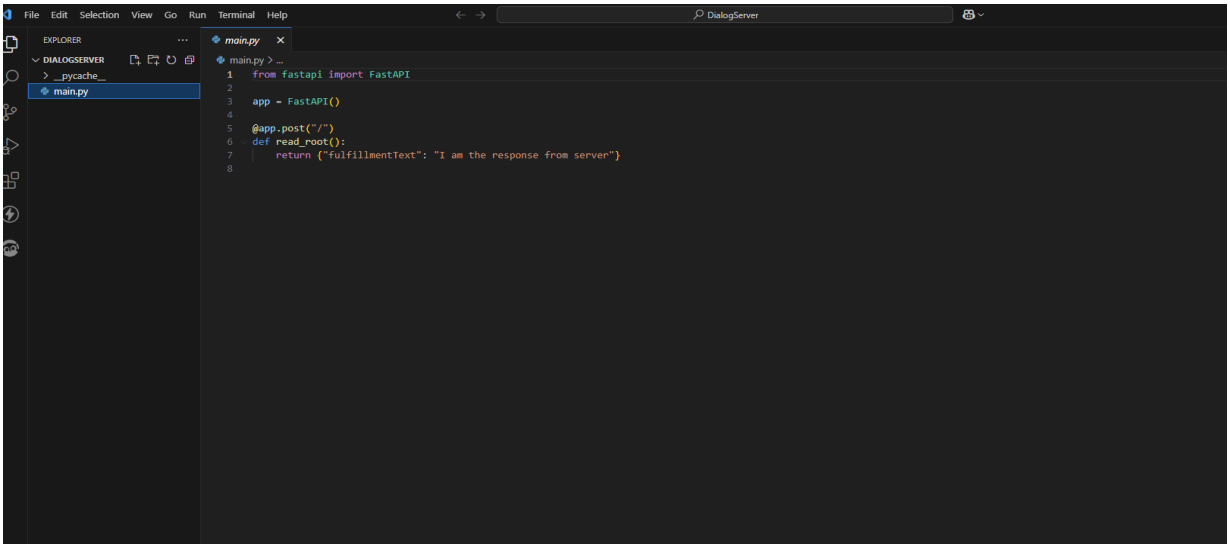
Create a Basic FastAPI Application:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")
def read_root():
    return {"message": "Webhook is running!"}
```

Run the Application:

```
uvicorn main:app --reload
```

Step 2: Implementing Dialogflow Webhook Logic



1. **Define the Webhook Endpoint**:

Easy Starter Way

```
from fastapi import FastAPI, Request

app = FastAPI()

@app.post("/webhook")
async def webhook(request: Request):
    req_data = await request.json()
```

```
    print("Request received:", req_data)

    # Construct a response
    fulfillment_response = {
        "fulfillmentText": "This is a response from FastAPI"
    }
    return fulfillment_response
```

Normal Way

```
from fastapi import FastAPI, Request

app = FastAPI()

@app.post("/webhook")
async def webhook(request: Request):
    req_data = await request.json()
    print("Request received:", req_data)

    # Construct a response
    fulfillment_response = {
        "fulfillmentMessages": [
            {"text": {"text": ["This is a response from FastAPI!"]}}
        ]
    }
    return fulfillment_response
```

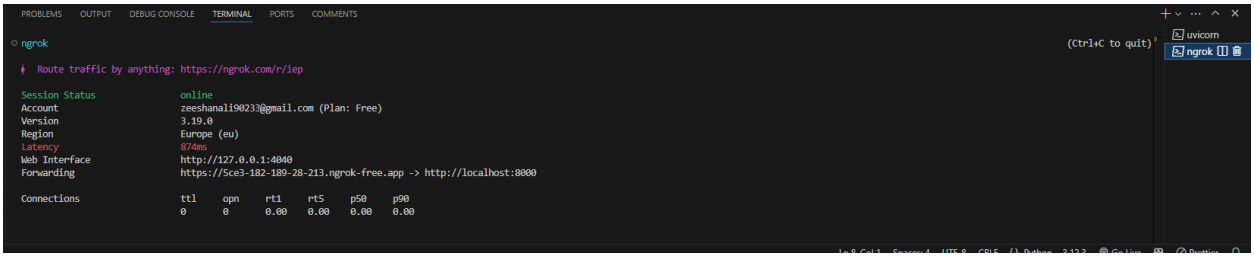
2. **Test the Endpoint Locally**: Use tools like Postman or curl to send a test request and ensure the endpoint behaves as expected.

Exposing Localhost with ngrok

For Dialogflow to communicate with your webhook, it needs a publicly accessible URL. Ngrok creates secure tunnels to your localhost, making it accessible over the internet.

Step 1: Install ngrok

1. **Download and Install**: Visit ngrok's website to download and install the tool for your OS.
2. **Authenticate**:
3. ngrok authtoken YOUR_AUTH_TOKEN



Step 2: Run ngrok

1. **Expose Localhost**:
2. ngrok http 8000
3. **Copy the Public URL**: Ngrok will display a URL like https://<random-string>.ngrok.io. Use this in the Dialogflow Console.

Connecting the Webhook to Dialogflow

1. **Enable Fulfillment**:
    - Go to the intent where you want to use fulfillment.
    - Enable the "Use webhook" checkbox.
2. **Set the Webhook URL**:
    - Navigate to the Dialogflow Console -> Fulfillment section.
    - Add the ngrok public URL (https://<random-string>.ngrok.io/webhook).
3. **Test Your Integration**:
    - Use the Dialogflow Simulator to trigger the intent and verify that your webhook is responding correctly.

Advanced Use Cases with FastAPI Webhooks

1. **Custom Payloads for Rich Responses:**

```
rich_response = {
  "fulfillmentMessages": [
    {
      "payload": {
        "richContent": [
          [{
            "type": "info",
            "title": "Custom Card",
            "subtitle": "This is a subtitle",
            "image": {
              "src": {
                "rawUrl": "https://example.com/image.png"
              }
            }
          }]
        ]
      }
    }
  ]
}
```

2. Accessing User Parameters:

Extract and use parameters provided by the user:

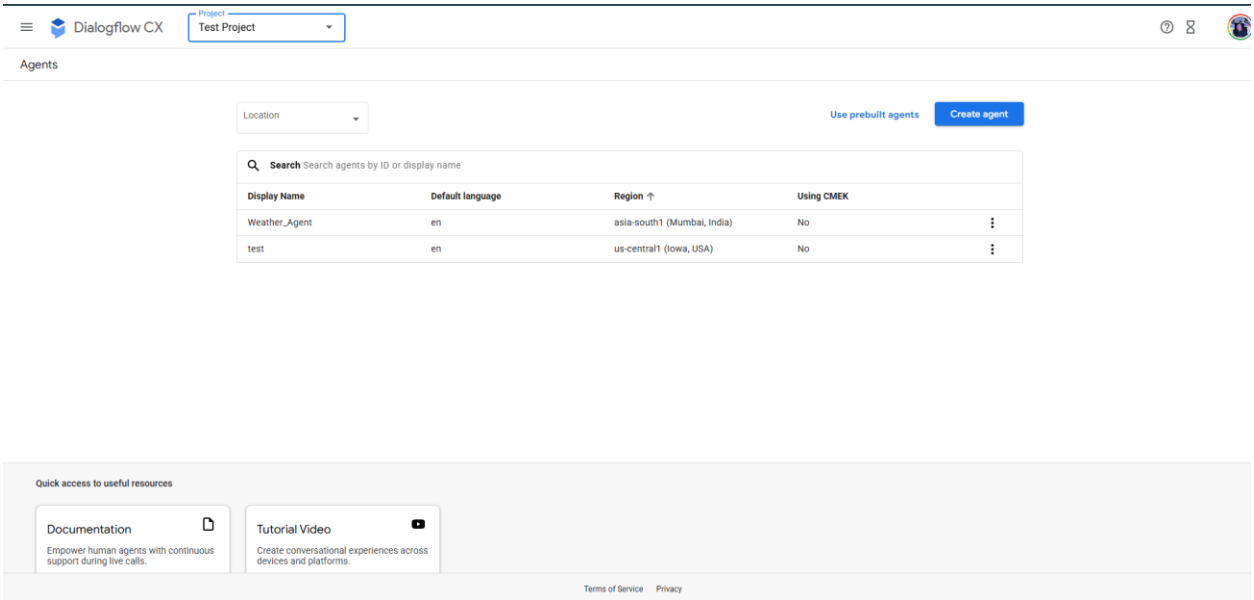
```
user_params = req_data["queryResult"]["parameters"]
print("User Parameters:", user_params)
```

3. Error Handling:

Add robust error handling to ensure reliable responses:

```
try:
  # Processing logic here
  return fulfillment_response
except Exception as e:
  return {"fulfillmentText": "Something went wrong!"}
```

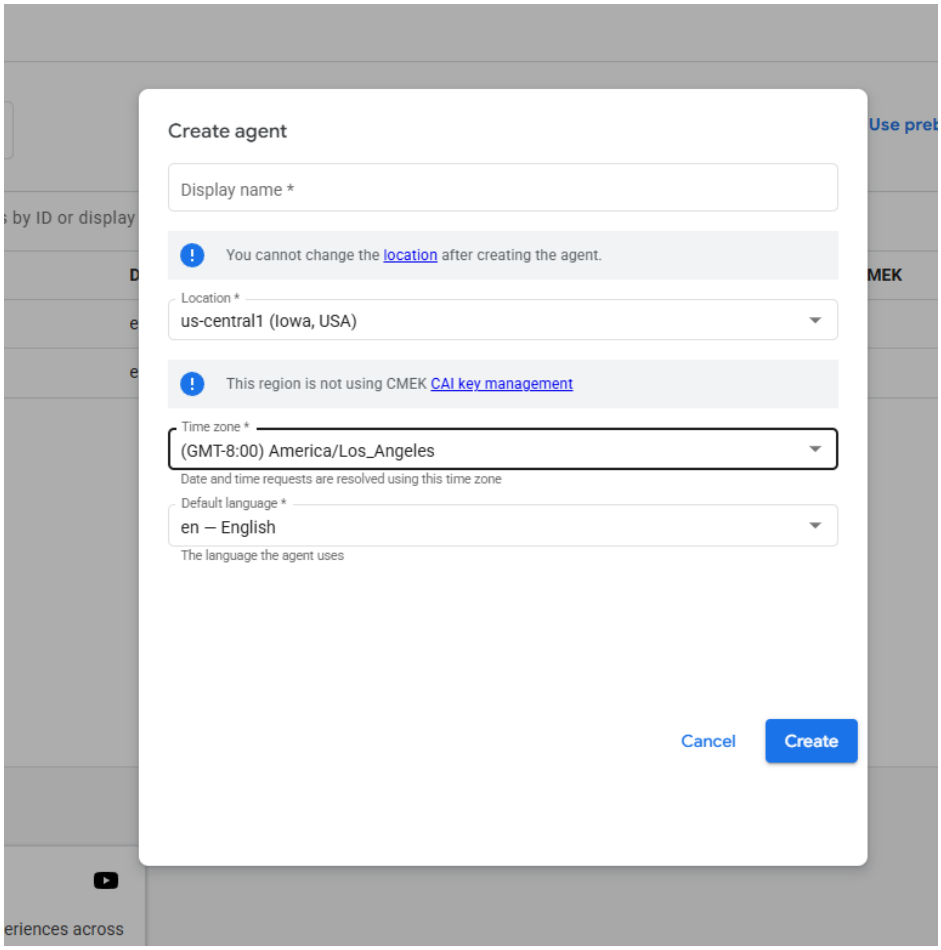
Chapter 3: Dialogflow CX Setup and Pizza Chatbot



Introduction to Dialogflow CX

Dialogflow CX (Customer Experience) is designed for **complex, multi-turn conversations** with advanced state management. Unlike ES, CX uses a **visual flow builder** to map conversational journeys, making it ideal for enterprise-grade chatbots. Key components include:

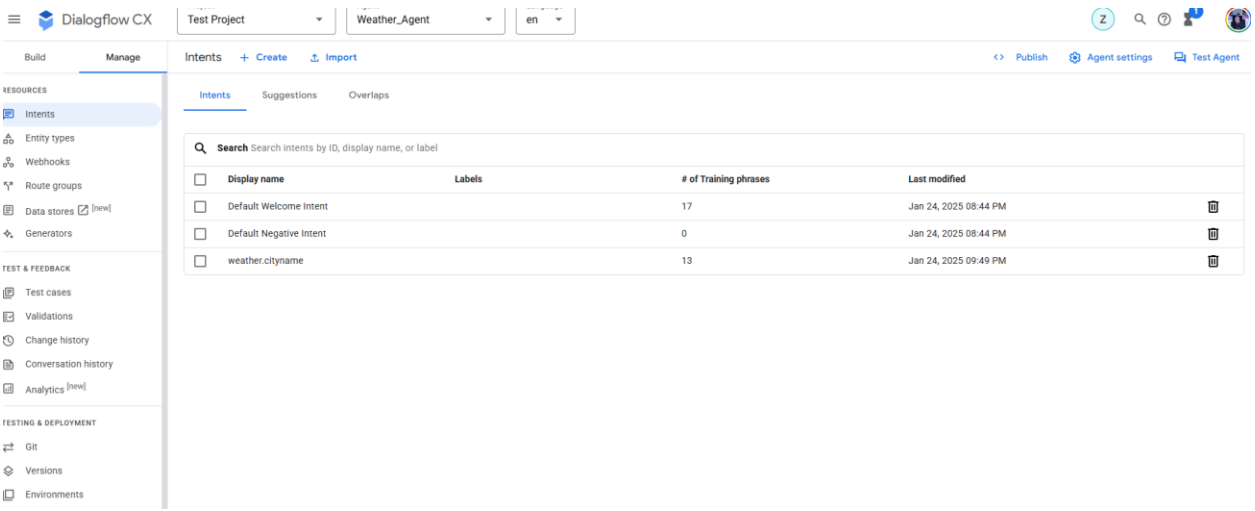
- **Agents**: The core chatbot instance.
- **Flows**: Modular sections of a conversation (e.g., "Order Pizza" flow).
- **Pages**: Steps within a flow (e.g., "Choose Size," "Select Toppings").
- **Transitions**: Rules to navigate between pages.
- **Parameters**: Variables to store user inputs (e.g., pizza_size).



Setting Up a Dialogflow CX Agent

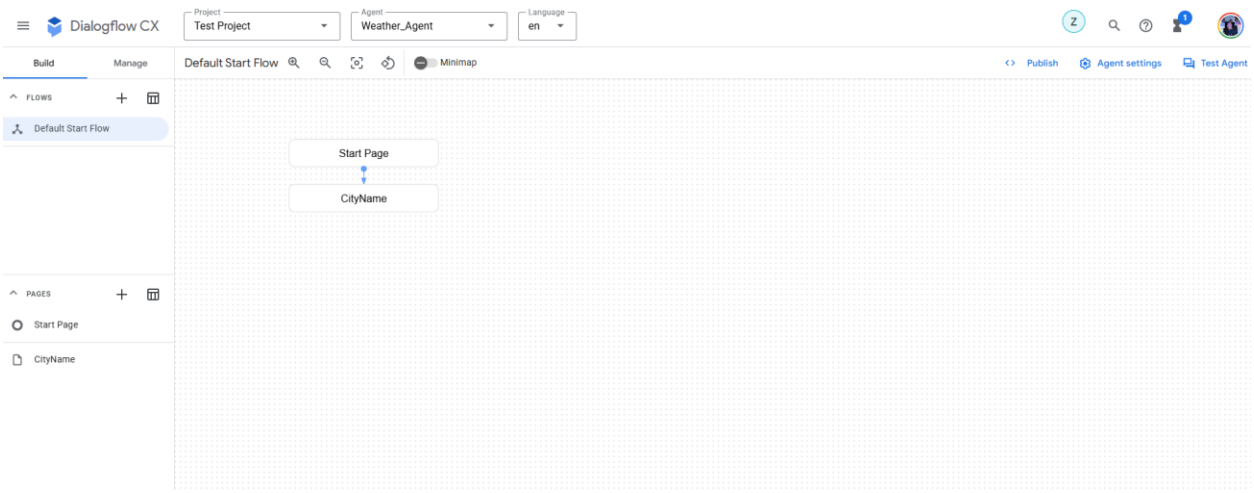
1. **Create a CX Agent**:
    - Go to Dialogflow CX Console.
    - Click **Create Agent** → Name it "PizzaBotCX" → Set language/time zone.
    - Enable **Advanced Settings** for Google Cloud logging if needed.
2. **Design the Flow**:
    - Click **Build a Flow** → Name it "OrderPizzaFlow".
    - Start with the **Default Start Page** to handle initial user requests.

Building the Pizza Order Flow



Step 1: Define Intents and Entities

1. **Create Intents**:
    - **StartOrderIntent**:
        - Training Phrases: *"I want to order a pizza," "Start pizza order."*
    - **SizeSelectionIntent**:
        - Training Phrases: *"Large," "Medium size," "I'll take a small."*
    - **ToppingsIntent**:
        - Training Phrases: *"Add pepperoni," "Mushrooms and olives."*
2. **Create Entities**:
    - **pizza_size**: Values: small, medium, large.
    - **toppings**: Values: pepperoni, mushrooms, olives.



Step 2: Configure Pages and Transitions

1. **Start Page**:
    - **Entry Route**: Triggered by **StartOrderIntent**.
        - Set Parameter: pizza_type (if needed).
    - **Transition**: Route to **SizePage**.
2. **SizePage**:
    - **Parameter**: pizza_size (required).
    - **Entry Route**: Use **SizeSelectionIntent** to capture size.
    - **Condition**: $session.params.pizza_size != null.
    - **Transition**: Route to **ToppingsPage**.
3. **ToppingsPage**:
    - **Parameter**: toppings (list).
    - **Entry Route**: Use **ToppingsIntent**.
    - **Transition**: Route to **ConfirmationPage**.

Step 3: Add Conditional Responses
In **ConfirmationPage**, use a **conditional response**:

Fulfillment in CX with FastAPI
Dialogflow CX uses a **different JSON structure** for webhook requests compared to ES. Adapt the FastAPI webhook from Chapter 2:

Sample CX Webhook Request:

```
{
  "session":
"projects/PROJECT_ID/locations/LOCATION/agents/AGENT_ID/sessions/SES
SION_ID",
  "pageInfo": {
    "currentPage":
"projects/PROJECT_ID/locations/LOCATION/agents/AGENT_ID/flows/FLOW_
ID/pages/PAGE_ID"
  },
  "sessionInfo": {
    "parameters": {
      "pizza_size": "large",
      "toppings": ["pepperoni"]
    }
  }
}
```

FastAPI Endpoint:

```
from fastapi import FastAPI, Request

app = FastAPI()

@app.post("/cx-webhook")
async def cx_webhook(request: Request):
    data = await request.json()
    params = data.get("sessionInfo", {}).get("parameters", {})

    size = params.get("pizza_size", "medium")
    toppings = params.get("toppings", [])

    response_text = f"Order confirmed: {size} pizza with {toppings if toppings
else 'no toppings'}."

    return {
        "sessionInfo": {
            "parameters": params  # Preserve parameters for future steps
            "fulfillmentResponse": {
```

```
            "messages": [{"text": {"text": [response_text]}}]
        }
    }
```

Configure Fulfillment in CX:

1. In the **ConfirmationPage**, enable **Webhook** under *Entry Fulfillment*.
2. Set the endpoint to your ngrok URL (e.g., https://abcd1234.ngrok.io/cx-webhook).

Testing the Flow

1. Use the **CX Simulator** to test:
    - User: *"I want to order a pizza."*
    - Bot: *"What size would you like?"*
    - User: *"Large with pepperoni."*
    - Bot: *"Order confirmed: large pizza with pepperoni."*

Advanced CX Features

1. **Slot Filling**:
    - Mark parameters as **required** in a page. The bot will auto-prompt users for missing data.
2. **Event Handlers**:
    - Trigger intents on specific events (e.g., timeout, retry).
3. **Parameter Persistence**:
    - Use $session.params.param_name to retain data across flows.

Migrating from ES to CX

- **Flows ≈ Contexts**: CX flows replace ES's input/output contexts with visual workflows.
- **Parameters ≈ Entities**: Store user inputs in parameters instead of composite entities.

- **State Management**: CX automatically preserves session parameters, reducing manual context handling.

Conclusion
Dialogflow CX empowers developers to build **non-linear, scalable chatbots** with intuitive visual tools. By structuring conversations into flows and pages, you can handle complex scenarios like pizza orders effortlessly.

Chapter 4: Building a Weather Chatbot and FastAPI

Introduction to the Weather Chatbot

In this chapter, you'll build a **weather chatbot** using **Dialogflow CX** and **FastAPI**. The chatbot will:

1. Capture a **city name** and **country name** from the user.
2. Call an external weather API (e.g., WeatherAPI) to fetch real-time weather data.
3. Return a formatted response with temperature, humidity, wind speed, and more.

Step 1: Set Up the Dialogflow CX Agent

1. **Create a New Agent**:
    - Go to Dialogflow CX Console.
    - Name the agent **WeatherBotCX** and set the default language.
2. **Define Entities**:
    - **city**: System entity type @sys.geo-city (prebuilt).
    - **country**: System entity type @sys.geo-country (prebuilt).

Step 2: Create the Weather Intent

1. **Intent Name**: GetWeather
2. **Training Phrases**:
    - *"What's the weather in Lahore, Pakistan?"*
    - *"Tell me the weather forecast for Tokyo, Japan."*
    - *"How's the weather in Paris?"*

3. **Parameters**:
    - **cityname**: Map to @sys.geo-city.
    - **countryname**: Map to @sys.geo-country (optional but recommended for accuracy).

Step 3: Design the Flow

1. **Start Page**:
    - Add an **entry route** triggered by GetWeather intent.

```
Set parameters:

"parameters": {
  "cityname": "$session.params.cityname",
  "countryname": "$session.params.countryname"
}
```

Transition to **WeatherPage**.

WeatherPage:

- Enable **webhook fulfillment** to call your FastAPI endpoint.

Step 4: Implement the FastAPI Webhook

Use the provided code to create a webhook that processes city/country inputs and fetches weather data.

Code Explanation

```
from fastapi import FastAPI, Request
import requests

app = FastAPI()

@app.post("/webhook/weather")
async def webhook(request: Request):
    try:
        body = await request.json()
```

```
        # Extract city and country from Dialogflow parameters
        city_name = body.get("intentInfo", {}).get("parameters",
{}).get("cityname", {}).get("resolvedValue")
        country_name = body.get("intentInfo", {}).get("parameters",
{}).get("countryname", {}).get("resolvedValue")

        response_text = ""
        try:
            # Call the weather API (replace with your API endpoint)
            url =
f"https://p2pclouds.up.railway.app/v1/learn/weather?city={city_name}"
            response = requests.get(url)
            data = response.json()

            # Extract weather details
            temp_c = data.get("current", {}).get("temp_c")
            feelslike_c = data.get("current", {}).get("feelslike_c")
            wind_kph = data.get("current", {}).get("wind_kph")
            humidity = data.get("current", {}).get("humidity")
            api_city = data.get("location", {}).get("name")
            api_country = data.get("location", {}).get("country")

            # Format the response
            response_text = f"""
            🌤 **Weather in {api_city}, {api_country}**:
            - Temperature: {temp_c}°C (Feels like {feelslike_c}°C)
            - Wind Speed: {wind_kph} km/h
            - Humidity: {humidity}%
            """
        except Exception as e:
            response_text = f"⚠ Failed to fetch weather for {city_name}. Please
try again later."

        # Return the response to Dialogflow
        return {
            "fulfillmentResponse": {
                "messages": [{"text": {"text": [response_text]}}]
            }
        }
    except Exception as e:
        return {
            "fulfillmentResponse": {
                "messages": [{"text": {"text": ["❌ An error occurred. Please try
again!"]}}]
            }
        }
```

Step 5: Configure Fulfillment in Dialogflow CX

1. **Enable Webhook**:
    - In the **WeatherPage**, go to *Entry Fulfillment* → Enable **Webhook**.
    - Set the URL to your FastAPI endpoint (e.g., https://your-ngrok-url.ngrok.io/webhook/weather).
2. **Deploy with ngrok**:
    - Run ngrok http 8000 to expose your local FastAPI server.
    - Update the webhook URL in Dialogflow CX with the ngrok HTTPS URL.

Step 6: Test the Chatbot

1. Use the **Dialogflow CX Simulator**:
    - **User**: *"What's the weather in London, UK?"*
    - **Bot**: "🌤️ *Weather in London, UK: Temperature: 18°C (Feels like 17°C), Wind Speed: 15 km/h, Humidity: 65%"*
2. **Edge Cases**:
    - Test invalid cities: *"What's the weather in XYZ?"* → Bot returns an error message.

Common Issues and Fixes

1. **API Rate Limits**:
    - Use a paid API key (e.g., WeatherAPI Premium) for higher request limits.
2. **Parameter Extraction Failures**:
    - Add validation logic to handle missing city/country names:

```
if not city_name:
    response_text = "🔍 Please specify a city name!"
```

Conclusion

You've now built a fully functional weather chatbot using **Dialogflow CX** and **FastAPI**. This chatbot can be integrated with platforms like WhatsApp, Google Assistant, or your website to provide real-time weather updates. In the next chapter, you'll learn how to **deploy the chatbot at scale** and add analytics for performance monitoring.