INDEX:

Chatbot using NLP

Concept:

Eatopia is a smart chatbot powered by Natural Language Processing (NLP), designed to understand and respond to user queries about food ordering in a natural, human-like way. With its advanced NLP capabilities, it simplifies the process of menu browsing, order placement, and order tracking.

What is NLP?

Natural Language Processing (NLP) is a branch of artificial intelligence (AI) focused on the interaction between computers and human (natural) languages. It involves the ability of machines to understand, interpret, and generate human language in a way that is both meaningful and useful.

Why we are using it?

Intent Matching: Understands what the user wants.
Entity Recognition: Grabs important details (like food, size, toppings).
Context Management: Keeps track of the conversation.
Synonym Handling: Recognizes different ways to say the same thing.
Handling Ambiguity: Asks for more details when needed.

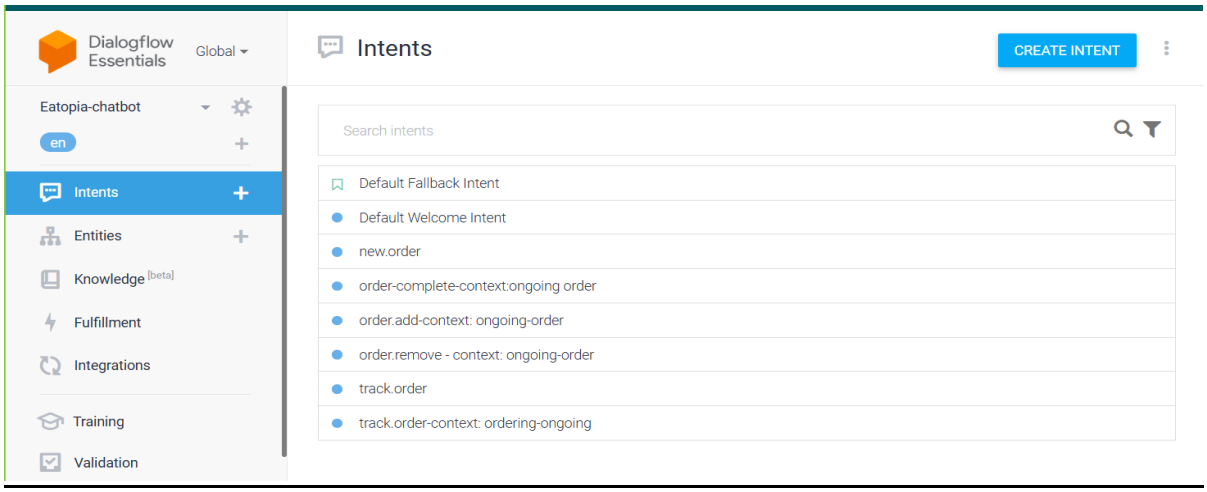
Key Features:

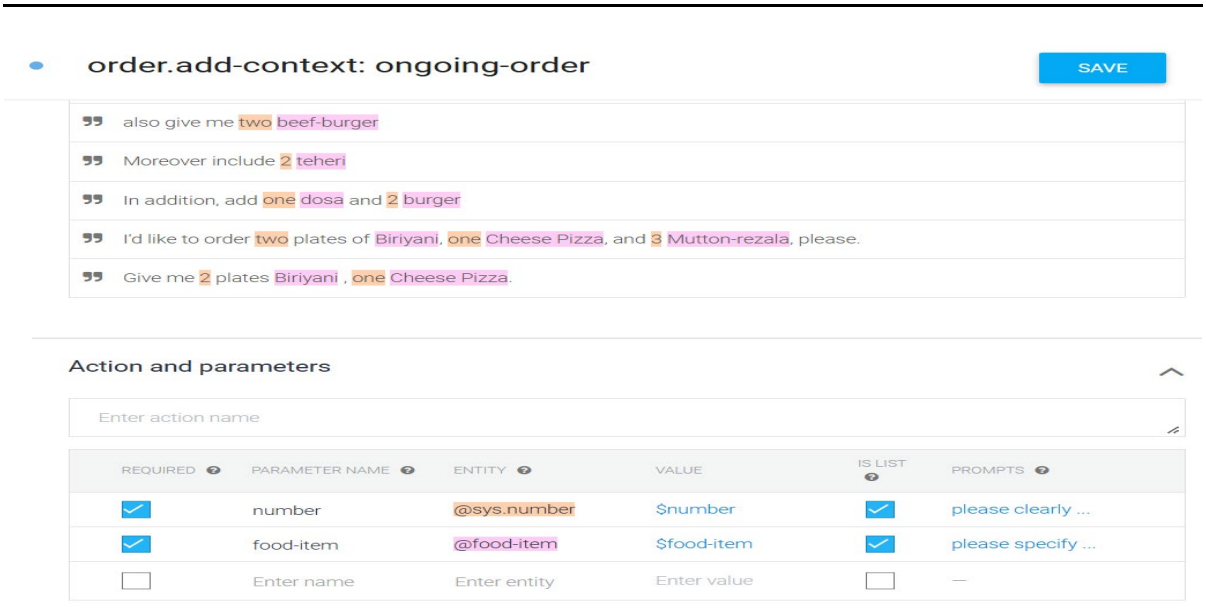
- Order placement
- Order Modification
- Payment Integration
- Order tracking

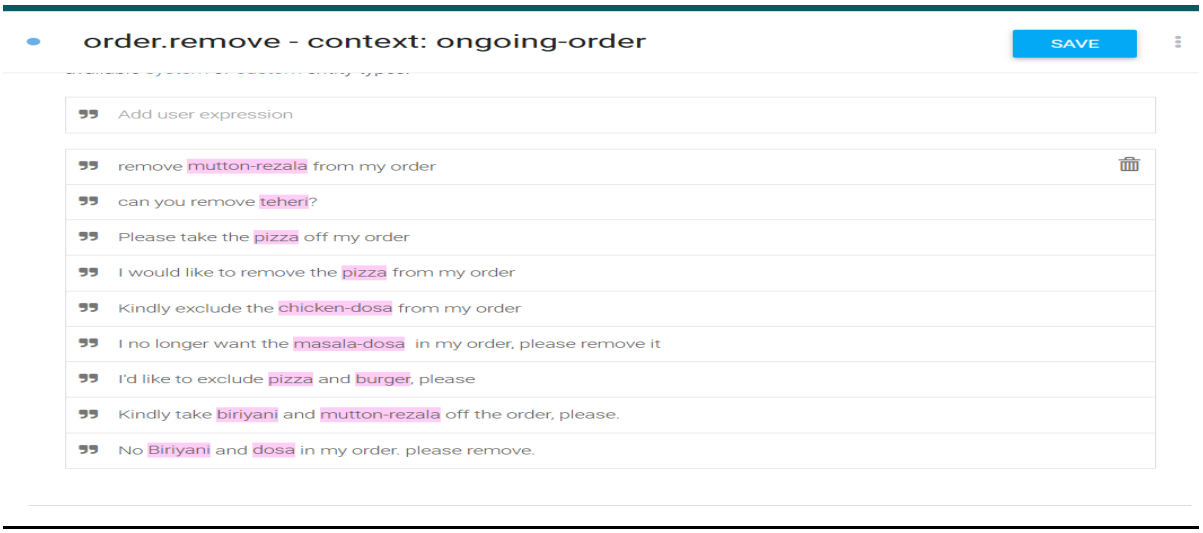
Tools and Technologies Used:

Platform: Dialogflow ES.
Frontend: HTML, CSS for the user interface.
Languages: Python (for webhook integration).
Backend: Mysql.
NLP Features: Intent matching, entity extraction, synonym handling etc.
Deployment: Google Assistant.
Integration and Testing: Ngrok for creating a secure tunnel to the local server during development.

Implementation screenshot:







new.order

SAVE

Responses

DEFAULT +

Text Response

1. Ok, starting a new order. You can say things like "I want two pizzas and one biriyani". Make sure to specify a quantity for every food item! Also, we have only the following items on our menu: Pizza, Burger, Mutton-rezala, Teheri, Biriyani, Dosa.
2. Starting new order. Specify food items and quantities. For example, you can say, "I would like to order two pizzas and one biriyani. Also, we have only the following items on our menu: Pizza, Burger, Mutton-rezala, Teheri, Biriyani, Dosa.
3. Enter a text response variant

ADD RESPONSES

Set this intent as end of conversation

Dialogflow Essentials — Global

Eatopia-chatbot

en

- Intents
- Entities
- Knowledge [beta]
- Fulfillment
- Integrations
- Training
- Validation
- History

food-item

SAVE

Define synonyms | Regexp entity | Allow automated expansion | Fuzzy matching

Pizza	Pizza, Cheese pizza, pizzas
Bueger	Bueger, beef-burger, cheese burger, burgers
Mutton-rezala	Mutton-rezala, mutton, mutton-curry
Teheri	Teheri, beef-teheri, mutton-teheri
Biriyani	Biriyani, chicken-biriyani, beef-biriyani
Dosa	Dosa, masala-dosa, chicken-dosa, vegetable-dosa
	Click here to edit entry

+ Add a row

Fulfillment

Webhook

ENABLED

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.

URL* https://0054-103-145-135-18.ngrok-free.app

BASIC AUTH Enter username Enter password

HEADERS Enter key Enter value

Add header

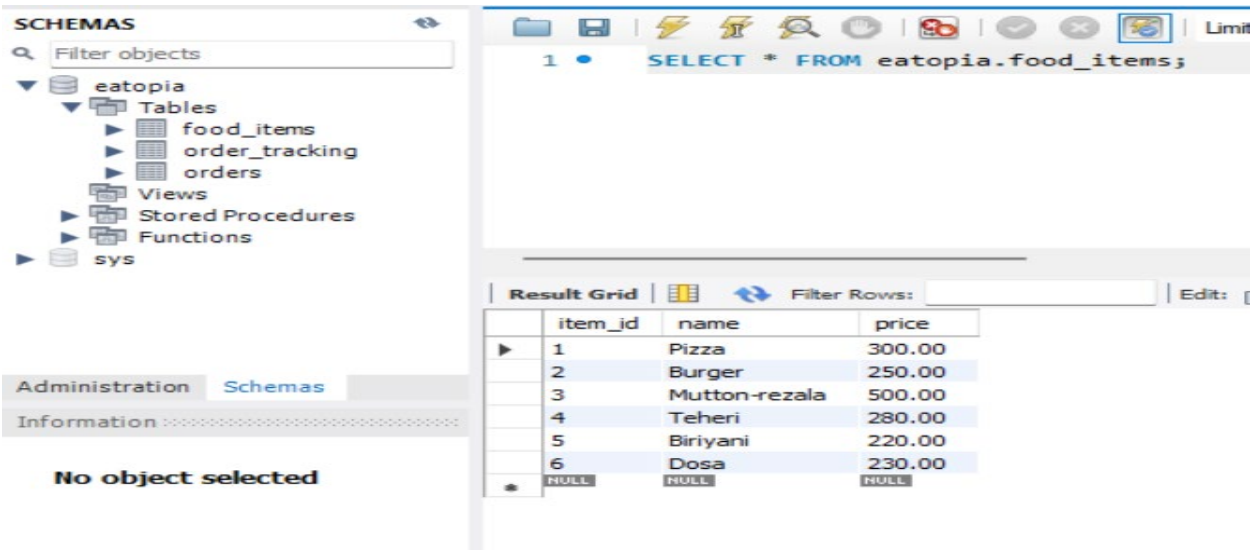
SMALL TALK Disable webhook for Smalltalk

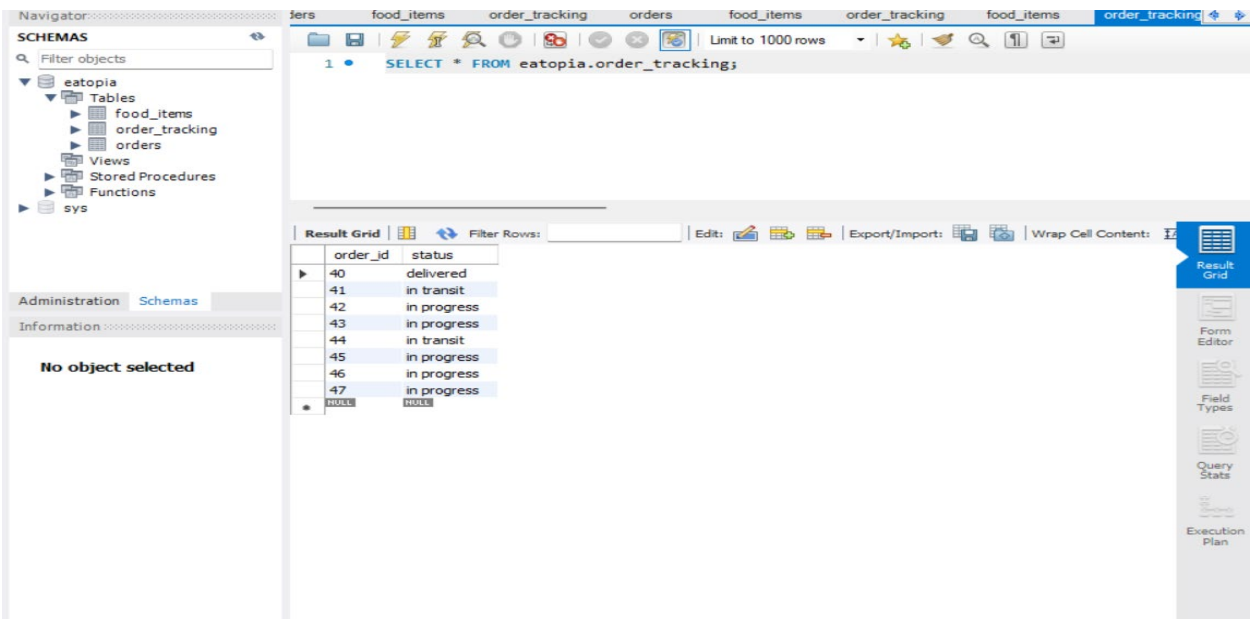
Inline Editor (Powered by Google Cloud Functions)

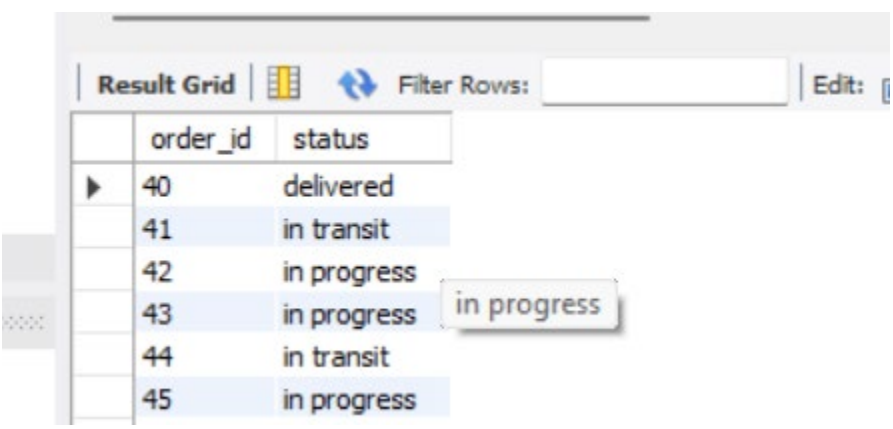
DISABLED

Build and manage fulfillment directly in Dialogflow via Cloud Functions. Docs

Newly created cloud functions now use Node.js 10 as runtime engine. Check migration guide for more details.


SCHEMAS
Filter objects
eatopia
Tables
food_items
order_tracking
orders
Views
Stored Procedures
Functions
sys
Administration
Schemas
Information
No object selected
SELECT * FROM eatopia.food_items;
Result Grid
Filter Rows:
Edit:
item_id name price
1 Pizza 300.00
2 Burger 250.00
3 Mutton-rezala 500.00
4 Teheri 280.00
5 Biriyani 220.00
6 Dosa 230.00


Navigator
SCHEMAS
Filter objects
eatopia
Tables
food_items
order_tracking
orders
Views
Stored Procedures
Functions
sys
Administration
Schemas
Information
No object selected
Limit to 1000 rows
SELECT * FROM eatopia.order_tracking;
Result Grid
Filter Rows:
Edit:
Export/Import:
Wrap Cell Content:
order_id status
40 delivered
41 in transit
42 in progress
43 in progress
44 in transit
45 in progress
46 in progress
47 in progress
Result Grid
Form Editor
Field Types
Query Stats
Execution Plan


Result Grid
Filter Rows:
Edit:
order_id status
40 delivered
41 in transit
42 in progress
43 in progress
44 in transit
45 in progress
in progress

Result Grid | Filter Rows:

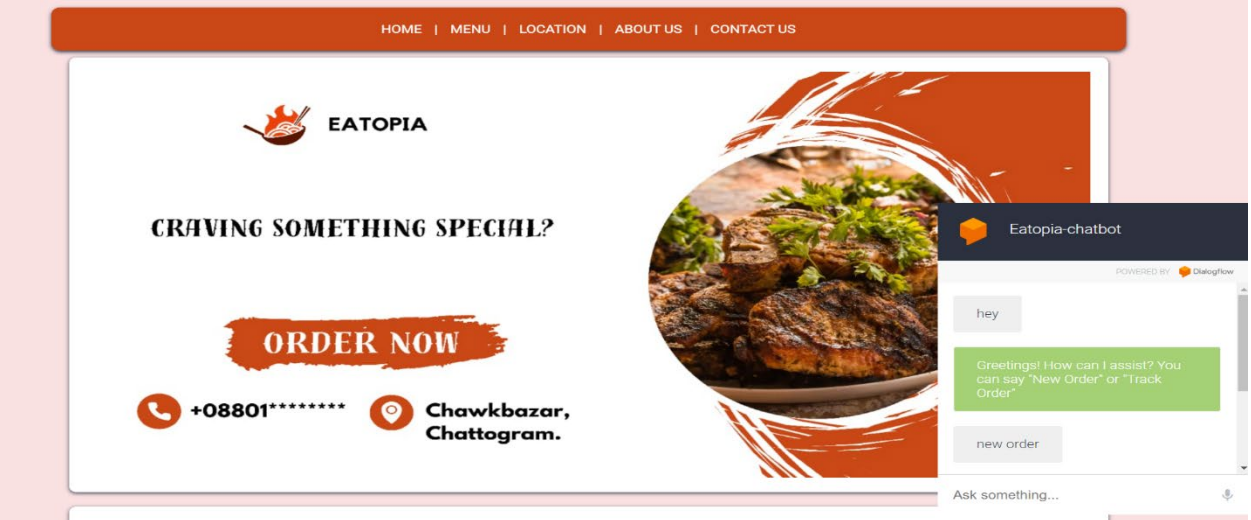
item_id	name	price
1	Pizza	300.00
2	Burger	250.00
3	Mutton-rezala	500.00
4	Teheri	280.00
5	Biriyani	220.00
6	Dosa	230.00
NULL	NULL	NULL

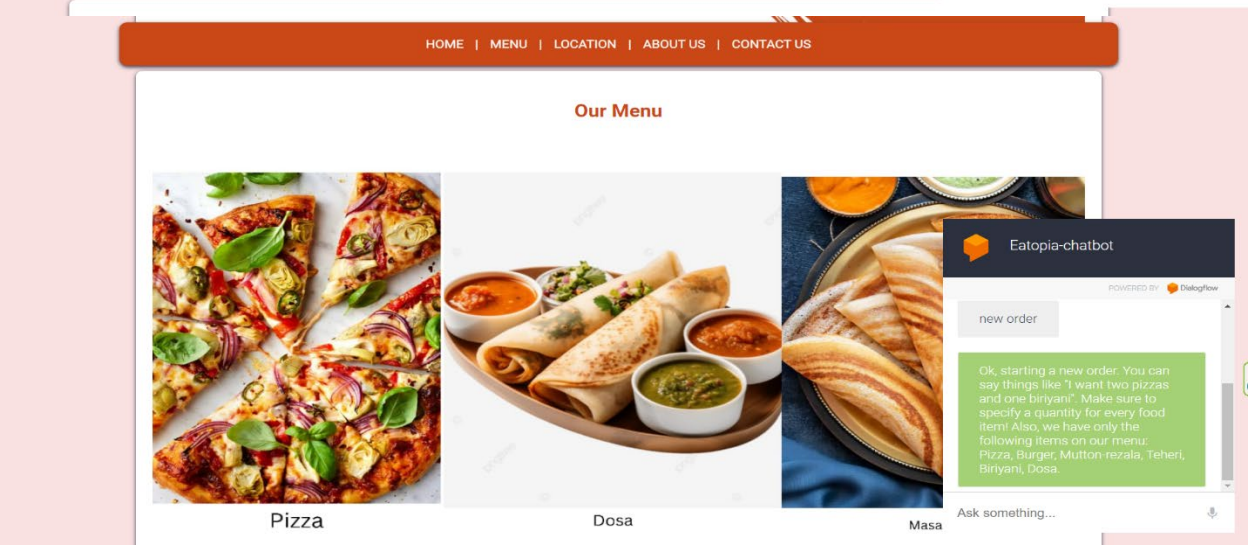
Result Grid | Filter Rows:

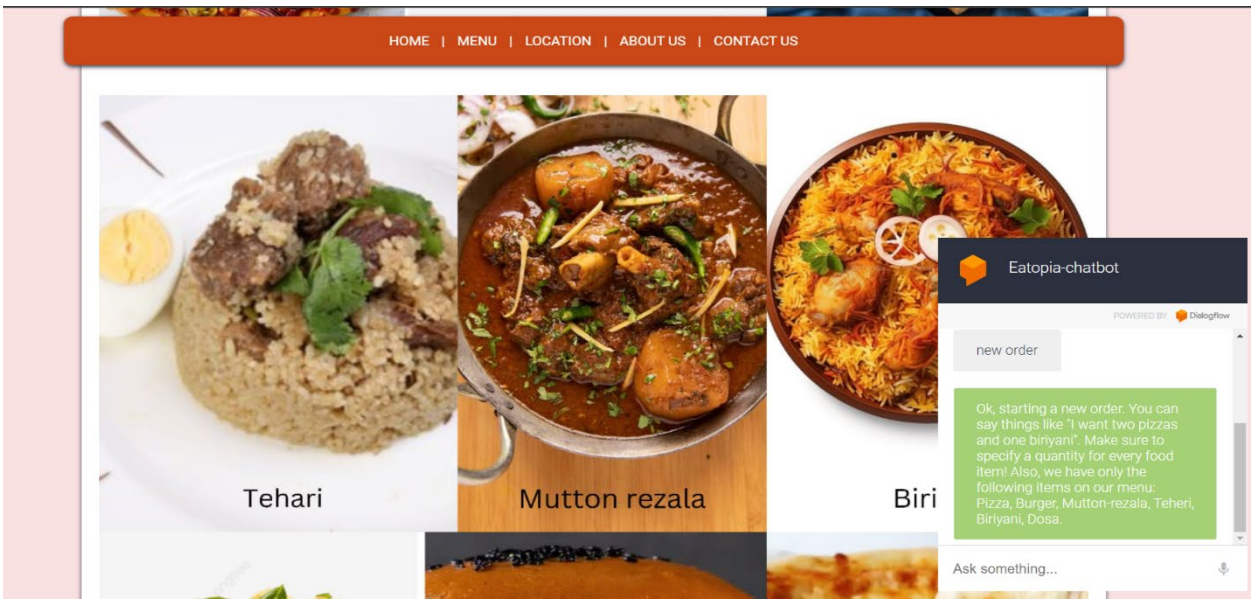
order_id	status
40	delivered
41	in transit
42	in progress
43	in progress
44	in transit
45	in progress
46	in progress
47	in progress
NULL	NULL

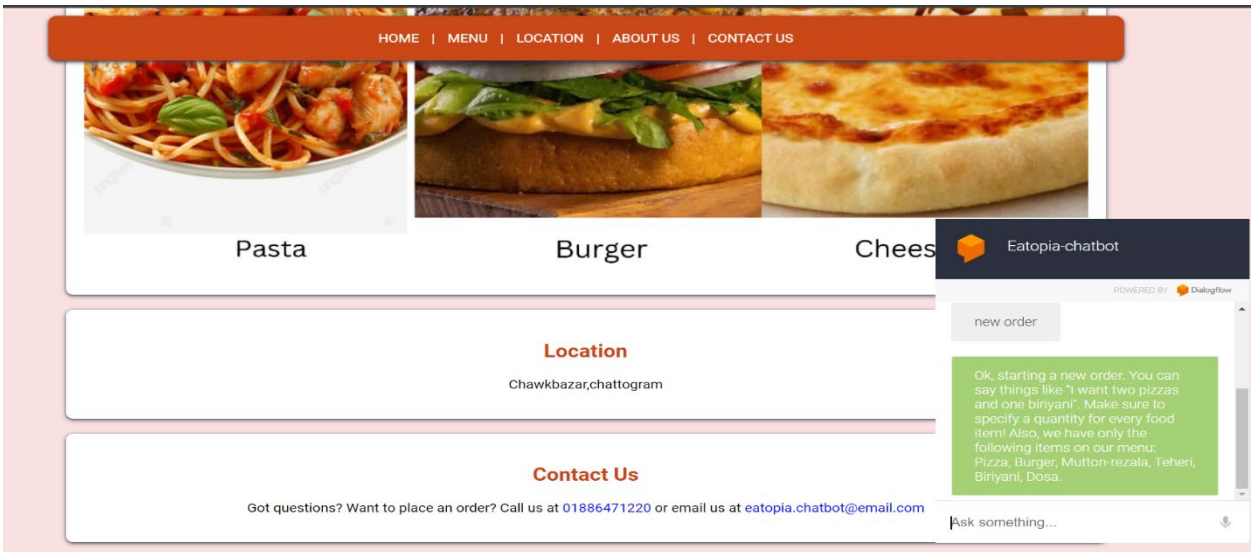
Result Grid | Filter Rows: | Edit:

order_id	item_id	quantity	total_price
40	1	2	600.00
40	3	1	500.00
41	4	3	750.00
41	6	2	840.00
41	9	4	460.00
42	1	1	300.00
42	4	1	280.00
43	1	2	600.00
44	3	1	500.00
44	6	1	230.00
45	1	2	600.00
45	4	1	280.00
46	4	2	560.00
46	5	3	660.00
47	3	1	500.00
47	6	2	460.00
NULL	NULL	NULL	NULL


HOME | MENU | LOCATION | ABOUT US | CONTACT US
EATOPIA
CRAVING SOMETHING SPECIAL?
ORDER NOW
+08801********
Chawkbazar, Chattogram.
Eatopia-chatbot
POWERED BY Dialogflow
hey
Greetings! How can I assist? You can say "New Order" or "Track Order"
new order
Ask something...


HOME | MENU | LOCATION | ABOUT US | CONTACT US
Our Menu
Pizza
Dosa
Masa
Eatopia-chatbot
POWERED BY Dialogflow
new order
Ok, starting a new order. You can say things like "I want two pizzas and one biriyani". Make sure to specify a quantity for every food item! Also, we have only the following items on our menu: Pizza, Burger, Mutton-rezala, Teheri, Biriyani, Dosa.
Ask something...


HOME | MENU | LOCATION | ABOUT US | CONTACT US
Tehari
Mutton rezala
Biri
Eatopia-chatbot
POWERED BY Dialogflow
new order
Ok, starting a new order. You can say things like "I want two pizzas and one biriyani". Make sure to specify a quantity for every food item! Also, we have only the following items on our menu: Pizza, Burger, Mutton-rezala, Teheri, Biriyani, Dosa.
Ask something...




```mysql
DELIMITER ;;
CREATE DEFINER=`root`@`localhost` FUNCTION `get_total_order_price`(p_order_id INT) RETURNS decimal(10,2)
    DETERMINISTIC
BEGIN
    DECLARE v_total_price DECIMAL(10, 2);

    -- Check if the order_id exists in the orders table
    IF (SELECT COUNT(*) FROM orders WHERE order_id = p_order_id) > 0 THEN
        -- Calculate the total price
        SELECT SUM(total_price) INTO v_total_price
        FROM orders
        WHERE order_id = p_order_id;

        RETURN v_total_price;
    ELSE
        -- Invalid order_id, return -1
        RETURN -1;
    END IF;
END ;;
```

```mysql
CREATE DEFINER=`root`@`localhost` PROCEDURE `insert_order_item`(
  IN p_food_item VARCHAR(255),
  IN p_quantity INT,
  IN p_order_id INT
)
BEGIN
    DECLARE v_item_id INT;
    DECLARE v_price DECIMAL(10, 2);
    DECLARE v_total_price DECIMAL(10, 2);

    -- Get the item_id and price for the food item
    SET v_item_id = (SELECT item_id FROM food_items WHERE name = p_food_item);
    SET v_price = (SELECT get_price_for_item(p_food_item));

    -- Calculate the total price for the order item
    SET v_total_price = v_price * p_quantity;

    -- Insert the order item into the orders table
    INSERT INTO orders (order_id, item_id, quantity, total_price)
    VALUES (p_order_id, v_item_id, p_quantity, v_total_price);
END ;;
DELIMITER ;
```

Main.py

```
from fastapi import FastAPI, Request
from fastapi.responses import JSONResponse
import logging
import db_helper
from db_helper import get_connection, get_order_status
import generic_helper
# Initialize FastAPI app
app = FastAPI()
# Configure logging
logging.basicConfig(level=logging.INFO)
# In-progress orders dictionary
inprogress_orders = {}
def add_to_order(parameters: dict, session_id: str) -> str:
    """
    Handles adding items to the order based on parameters.
    """
    food_items = parameters.get("food-item", [])
    quantities = parameters.get("number", [])
    if len(food_items) != len(quantities):
        fulfillment_text = "Sorry, I didn't understand. Can you please specify food items and
quantities?"
    else:
        new_food_dict = dict(zip(food_items, quantities))
        if session_id in inprogress_orders:
            current_food_dict = inprogress_orders[session_id]
            current_food_dict.update(new_food_dict)
        else:
            current_food_dict = new_food_dict
        inprogress_orders[session_id] = current_food_dict
        logging.info(f"Updated in-progress orders: {inprogress_orders}")
        order_str = generic_helper.get_str_from_food_dict(inprogress_orders[session_id])
        fulfillment_text = f"So far you have: {order_str}. Do you need anything else?"
    return fulfillment_text
def track_order(cnx, parameters: dict) -> str:
    """
    Handles the logic for tracking an order based on parameters.
    """
    order_id = parameters.get('order_id')
    logging.info(f"Tracking order with parameters: {parameters}")
    if not order_id:
        return "Order ID is missing. Please provide a valid order ID."
    # Convert to integer, with error handling
```

```
    try:
      order_id = int(order_id)
    except ValueError:
        return "Invalid Order ID. Please provide a numeric order ID."
    # Fetch order status from the database
    order_status = get_order_status(cnx, order_id)
    if order_status and order_status.lower() != "no order found for the provided order id.":
        return f"The order status for order ID {order_id} is: {order_status}."
    else:
        return f"No order found with order ID: {order_id}."
def save_to_db(cnx, order: dict):
    """
    Saves an order to the database.
    """
    next_order_id = db_helper.get_next_order_id(cnx)
    # Insert individual items along with quantity in orders table
    for food_item, quantity in order.items():
        rcode = db_helper.insert_order_item(cnx, food_item, quantity, next_order_id)
        if rcode == -1:
            return -1
    # Now insert order tracking status
    db_helper.insert_order_tracking(cnx, next_order_id, "in progress")
    return next_order_id
def complete_order(cnx, parameters: dict, session_id: str):
    """
    Completes the user's order by saving it to the database.
    """
    if session_id not in inprogress_orders:
        return "I'm having trouble finding your order. Sorry! Can you place a new order, please?"
    order = inprogress_orders[session_id]
    order_id = save_to_db(cnx, order)
    if order_id == -1:
        fulfillment_text = "Sorry, I couldn't process your order due to a backend error. " \
                           "Please place a new order again."
    else:
        order_total = db_helper.get_total_order_price(cnx, order_id)
        fulfillment_text = f"Awesome. We have placed your order. " \
                           f"Here is your order ID # {order_id}. " \
                           f"Your order total is {order_total}, which you can pay at the time of delivery!"
    del inprogress_orders[session_id]
    return fulfillment_text
def remove_from_order(parameters: dict, session_id: str) -> str:
    """
    Handles removing items from the order based on parameters.
```

```
    """
    if session_id not in inprogress_orders:
        return "I'm having trouble finding your order. Sorry! Can you place a new order, please?"
    food_items_to_remove = parameters.get("food-item", [])
    if not food_items_to_remove:
        return "I didn't understand what you want to remove. Please specify the food items."
    current_food_dict = inprogress_orders[session_id]
    # Remove specified items from the current order
    for item in food_items_to_remove:
        if item in current_food_dict:
            del current_food_dict[item]
        else:
            return f"I couldn't find {item} in your order. Please check again."
    # Update the session's order
    inprogress_orders[session_id] = current_food_dict
    # If the order is empty after removal, clear the session
    if not current_food_dict:
        del inprogress_orders[session_id]
        return "Your order is now empty. You can start a new order if you'd like."
    order_str = generic_helper.get_str_from_food_dict(current_food_dict)
    return f"Okay, I've removed the specified items. Your updated order is: {order_str}. Do you
need anything else?"
@app.post("/")
async def handle_request(request: Request):
    """
    Main handler for processing requests from Dialogflow.
    """
    try:
        payload = await request.json()
        logging.info(f"Payload received: {payload}")
        # Extract intent, parameters, and session ID
        intent = payload['queryResult']['intent']['displayName']
        parameters = payload['queryResult']['parameters']
        session_id = payload['session'].split('/')[-1]  # Extract session ID
        cnx = get_connection()  # Get database connection
        # Intent handler dictionary
        intent_handler_dict = {
            'order.add-context: ongoing-order': lambda params: add_to_order(params, session_id),
            'order.remove - context: ongoing-order': lambda params: remove_from_order(params,
session_id),
            'order-complete-context:ongoing order': lambda params: complete_order(cnx, params,
session_id),
            'track.order-context: ordering-ongoing': lambda params: track_order(cnx, params),
        }
```

```
        # Match the intent and call the corresponding handler function
        if intent in intent_handler_dict:
            fulfillment_text = intent_handler_dict[intent](parameters)
            return JSONResponse(content={
                "fulfillmentText": fulfillment_text
            })
        # Default response for unrecognized intents
        return JSONResponse(content={
            "fulfillmentText": "Intent not recognized"
        })
    except Exception as e:
        logging.error(f"Error processing request: {e}")
        return JSONResponse(content={
            "fulfillmentText": "An error occurred while processing your request."
        })
    finally:
        # Close the database connection
        if 'cnx' in locals() and cnx.is_connected():
            cnx.close()
# Entry point for the application
if _name_ == "_main_":
    import uvicorn
    uvicorn.run(app, host="0.0.0.0", port=8000)
```

db_helper.py:

```
import mysql.connector
from mysql.connector import Error
# Helper function to establish a database connection
def get_connection():
    """
    Establishes and returns a new database connection.
    """
    try:
        return mysql.connector.connect(
            host="localhost",       # Replace with your database host
            user="root",            # Replace with your MySQL username
            password="221251",      # Replace with your MySQL password
            database="eatopia"      # Replace with your database name
        )
    except Error as e:
        print(f"Error connecting to the database: {e}")
        raise e
def insert_order_tracking(cnx, order_id, status):
```

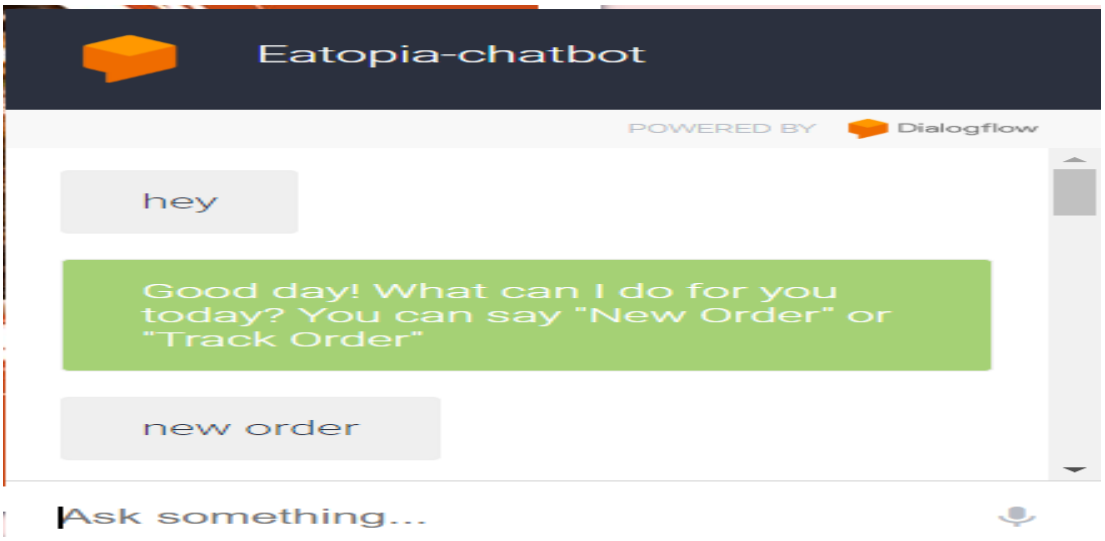
```
    """
    Inserts an order tracking record into the order_tracking table.
    """
    cursor = cnx.cursor()
    insert_query = "INSERT INTO order_tracking (order_id, status) VALUES (%s, %s)"
    cursor.execute(insert_query, (order_id, status))
    cnx.commit()
    cursor.close()
def insert_order_item(cnx, food_item, quantity, order_id):
    """
    Inserts an item into the orders table using a stored procedure.
    """
    try:
        cursor = cnx.cursor()
        cursor.callproc('insert_order_item', (food_item, quantity, order_id))
        cnx.commit()
        cursor.close()
        print("Order item inserted successfully!")
        return 1
    except mysql.connector.Error as err:
        print(f"Error inserting order item: {err}")
        cnx.rollback()
        return -1
    except Exception as e:
        print(f"An error occurred: {e}")
        cnx.rollback()
        return -1
def get_next_order_id(cnx):
    """
    Fetches the next available order ID by finding the maximum order_id in the orders table.
    """
    cursor = cnx.cursor()
    query = "SELECT MAX(order_id) FROM orders"
    cursor.execute(query)
    result = cursor.fetchone()[0]
    cursor.close()
    return 1 if result is None else result + 1
def get_total_order_price(cnx, order_id):
    """
    Fetches the total price of an order using a stored procedure or query.
    """
    cursor = cnx.cursor()
    query = f"SELECT get_total_order_price({order_id})"
    cursor.execute(query)
```

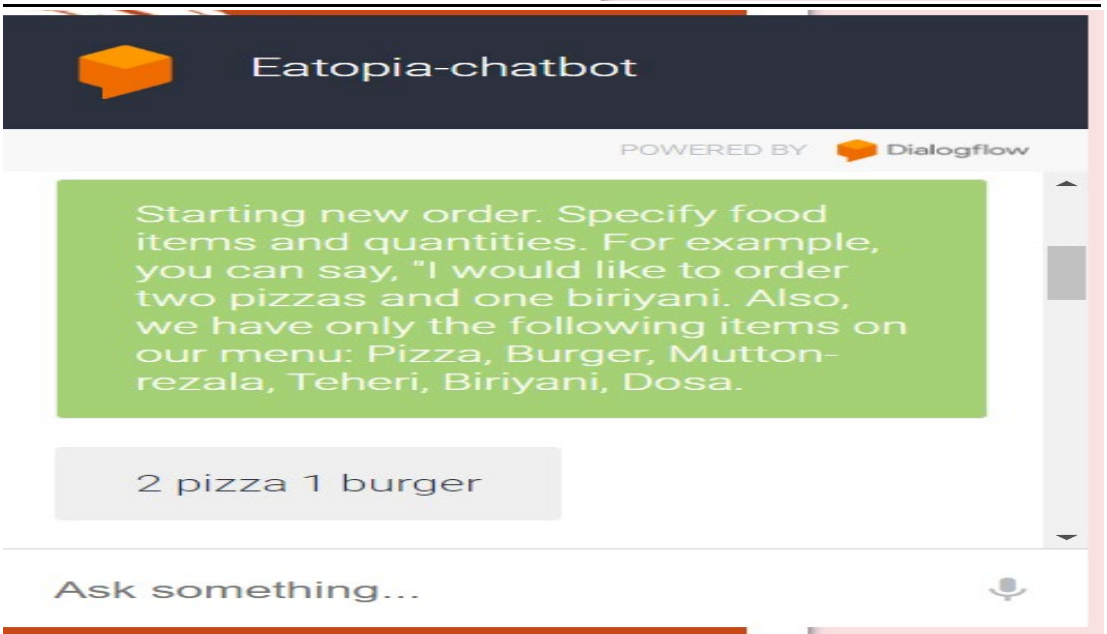
```
    result = cursor.fetchone()[0]
    cursor.close()
    return result
def get_order_status(cnx, order_id):
    """
    Fetches the status of an order from the order_tracking table using order_id.
    """
    cursor = cnx.cursor()
    query = "SELECT status FROM order_tracking WHERE order_id = %s"
    cursor.execute(query, (order_id,))
    result = cursor.fetchone()
    cursor.close()
    return result[0] if result else "No order found for the provided order ID."
```

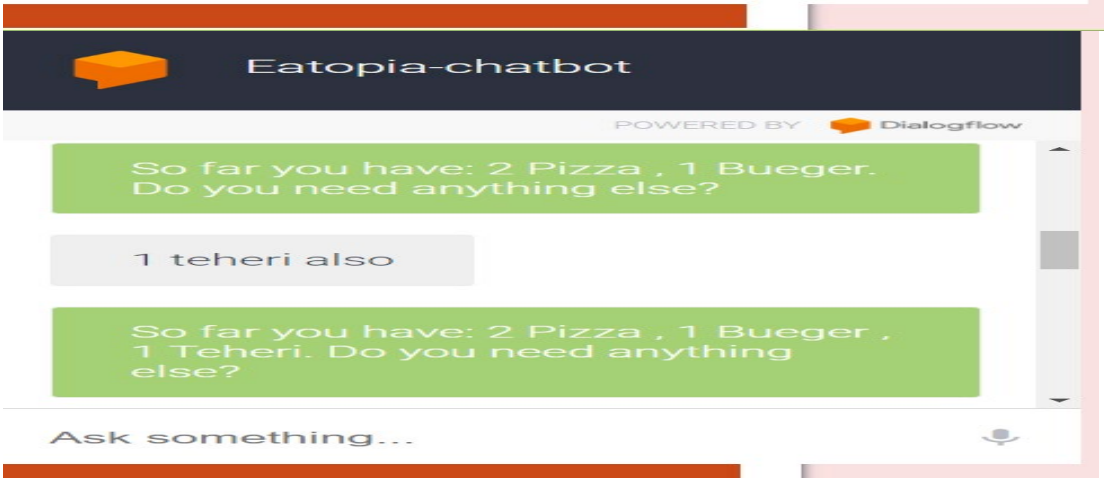
generic_helper.py:

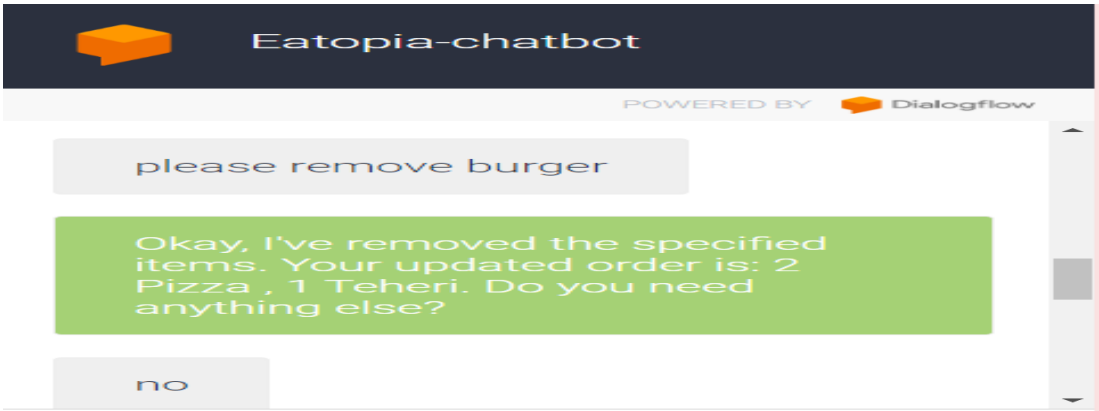
```
import re
def extract_session_id(session_str: str):
    match = re.search(r"/sessions/(.*:?)/contexts/", session_str)
    if match:
        extracted_string = match.group(1)
        return extracted_string
    return ""
def get_str_from_food_dict(food_dict: dict):
     return " , ".join([f"{int(value)} {key}" for key, value in food_dict.items()])
if _name=="main_":
    print(get_str_from_food_dict({"samosa": 2, "pizza": 1}))
    #print(extract_session_id("projects/eatopia-rode/agent/sessions/123/contexts/ongoing-
order"))
```

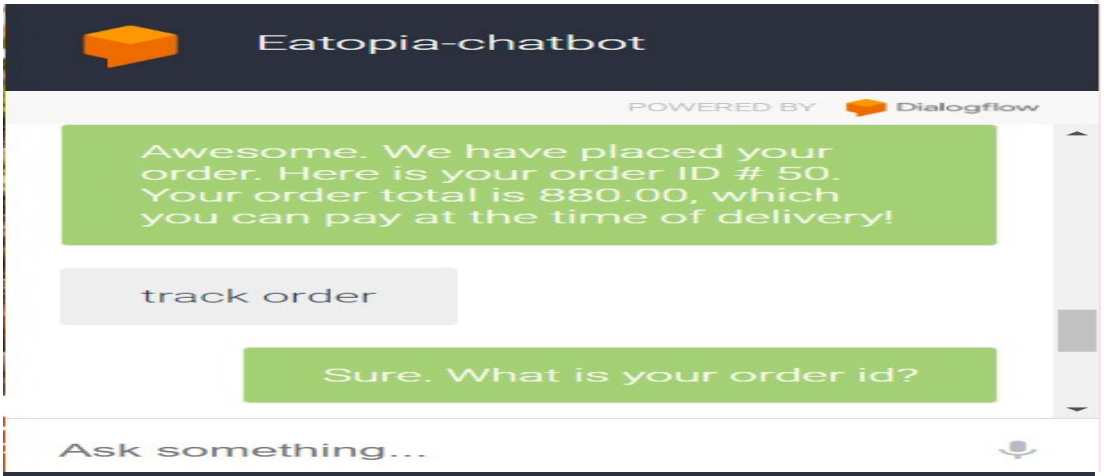
Input and Output screenshot:

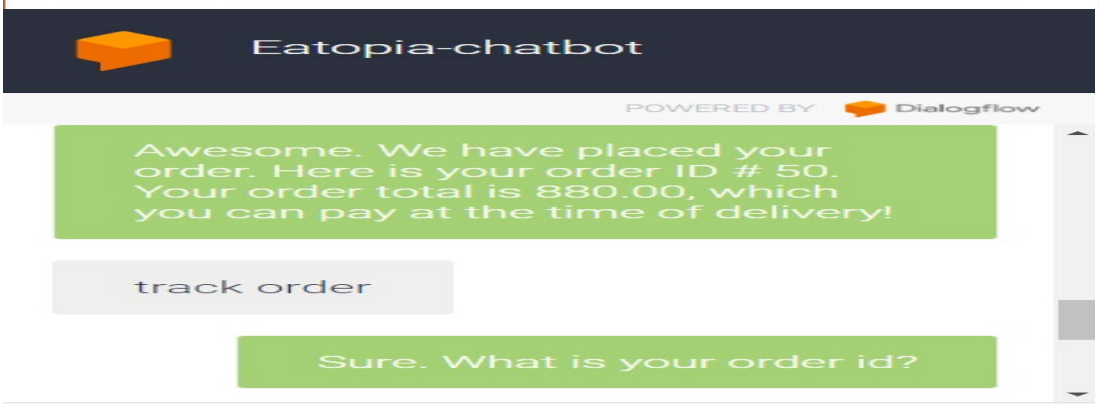


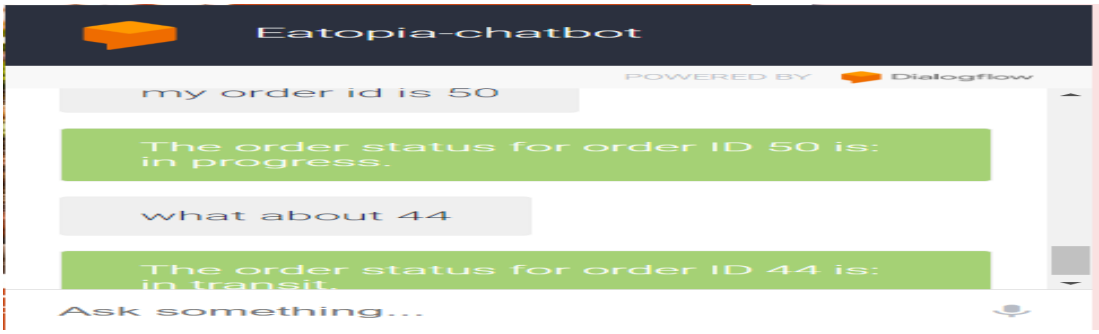





Eatopia-chatbot
POWERED BY Dialogflow
please remove burger
Okay, I've removed the specified items. Your updated order is: 2 Pizza , 1 Teheri. Do you need anything else?
no
Ask something...

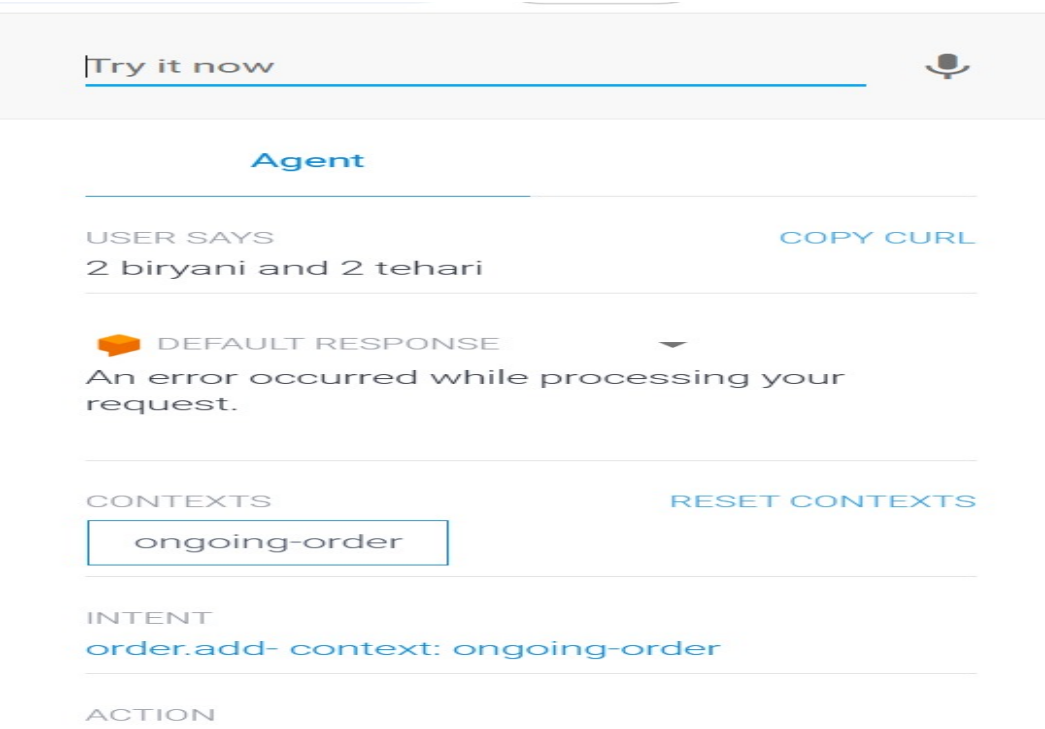

Eatopia-chatbot
POWERED BY Dialogflow
Awesome. We have placed your order. Here is your order ID # 50. Your order total is 880.00, which you can pay at the time of delivery!
track order
Sure. What is your order id?
Ask something...


Eatopia-chatbot
POWERED BY Dialogflow
Awesome. We have placed your order. Here is your order ID # 50. Your order total is 880.00, which you can pay at the time of delivery!
track order
Sure. What is your order id?
Ask something...


Eatopia-chatbot
POWERED BY Dialogflow
my order id is 50
The order status for order ID 50 is: in progress.
what about 44
The order status for order ID 44 is:
Ask something...

The problems we face:

Problem 1:



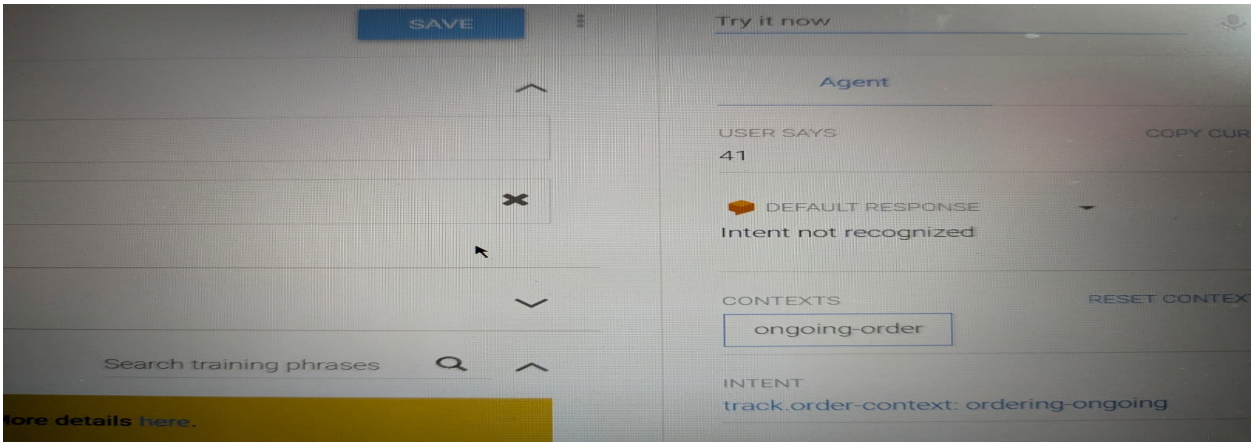
Why it Happens:

- Due to server errors or database query failures.

How to Solve it:

- By checking for 500 internal server errors and review backend logs.
- Ensure database connection is active and queries are correct.
- Add error handling in the code to return meaningful responses.

Problem 2:



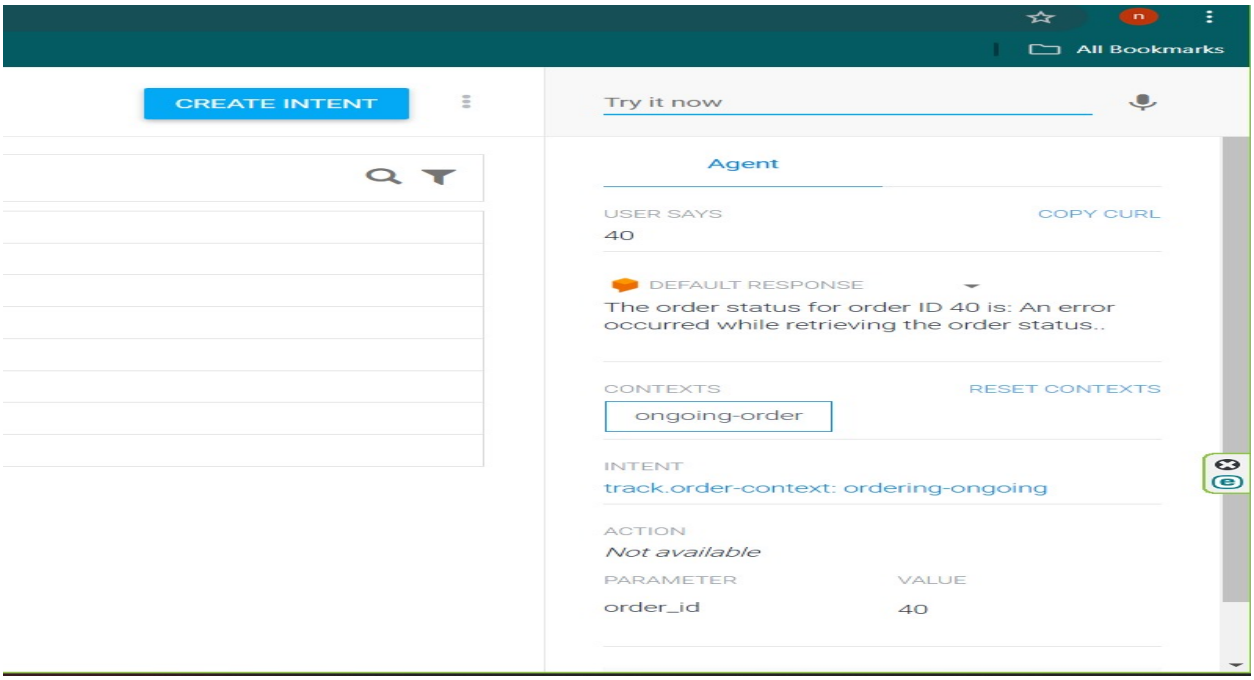
Why it Happens:

- Dialogflow fails to match user input to the right intent.
- This could be due to:Misconfigured or missing intent.
- Insufficient or ambiguous training phrases.
- Incorrectly defined or unrecognized entities.

How to Solve it:

- We can add more training Phrases: Ensure a variety of phrases to cover different user expressions.
- Review Entities: we have to make sure entities are defined properly and match user input.
- Test with Dialogflow Console:We used the testing tool to verify if the correct intent is triggered and entities are extracted.

Problem 3:



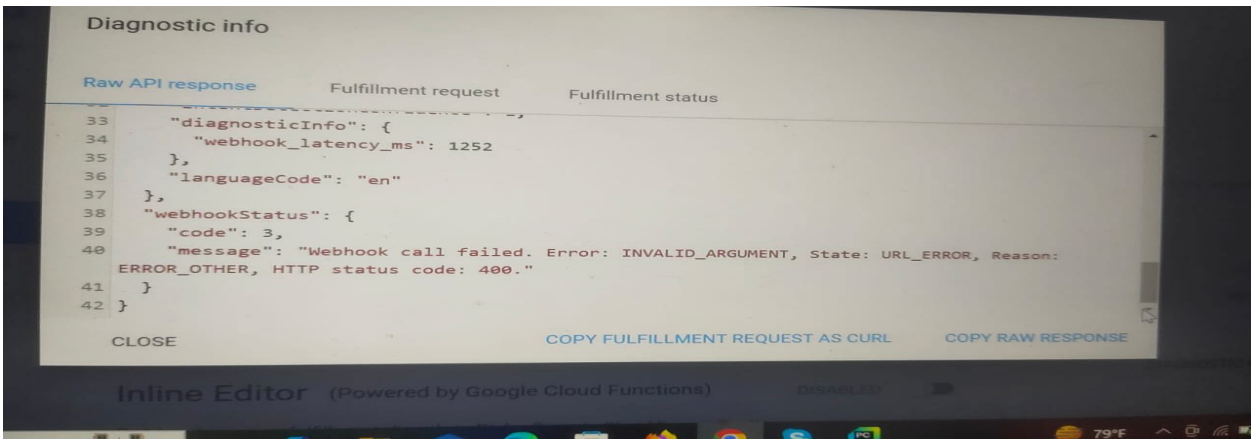
Why it Happens:

- Incorrect URL: Webhook URL may be wrong or unreachable.
- Server Errors: Backend server issues.
- Timeouts: Slow responses from the server.

How to Solve it:

- Verify Webhook URL: Ensure the URL is correct and accessible.
- Check Server Logs: Look for errors in the server logs.
- Test API Endpoints: Use Postman to test API responses.

Problem 4:



Why it Happens:

- Network issues: Connection problems between Dialogflow and the backend.
- API key errors: Incorrect or missing authentication keys.
- Invalid response format: The response from the backend is not in the correct format.

How to Solve it:

- Check API Keys: Ensure correct API keys or tokens are set up.
- Validate Response Format: Confirm that the backend returns a valid JSON response.
- Review Backend Logs: Check server logs for any error details to identify the issue.

Problem 5:

Why it Happens:

- Network issues or slow server response.
- Incorrect URL.

How to Solve it:

- Verify URL: Ensure the webhook URL is correct and accessible.
- Optimize Server: Improve backend performance.
- Increase Timeout: Adjust timeout settings in Dialogflow and the backend.

Besides these problems we also face ngrok authentication problem and pycham establishment problem.

Conclusion:

The Food Chatbot project successfully integrates Dialogflow for NLP, Python for backend processing, and ngrok for tunneling, allowing users to place orders, track statuses, and provide feedback. Challenges like intent recognition, webhook failures, and setup issues with ngrok and PyCharm were resolved through configuration adjustments and backend improvements. The project enhances user experience by offering quick, efficient responses and sets a foundation for future enhancements, such as personalized recommendations and advanced AI features.

Reference:

https://youtu.be/aFwrNSfthxU?si=PpJUj32cuWH148q-
https://youtu.be/TfZzPjMJDeY?si=RWn41kCbAJ0K0p0f
https://youtu.be/CMrHM8a3hqw?si=VrJMy9--KbGkZTRn