Automated email to customer message

Using Generative AI concept, openAI and chatGPT

Prepared by:Kruti Dhyani

Table of content

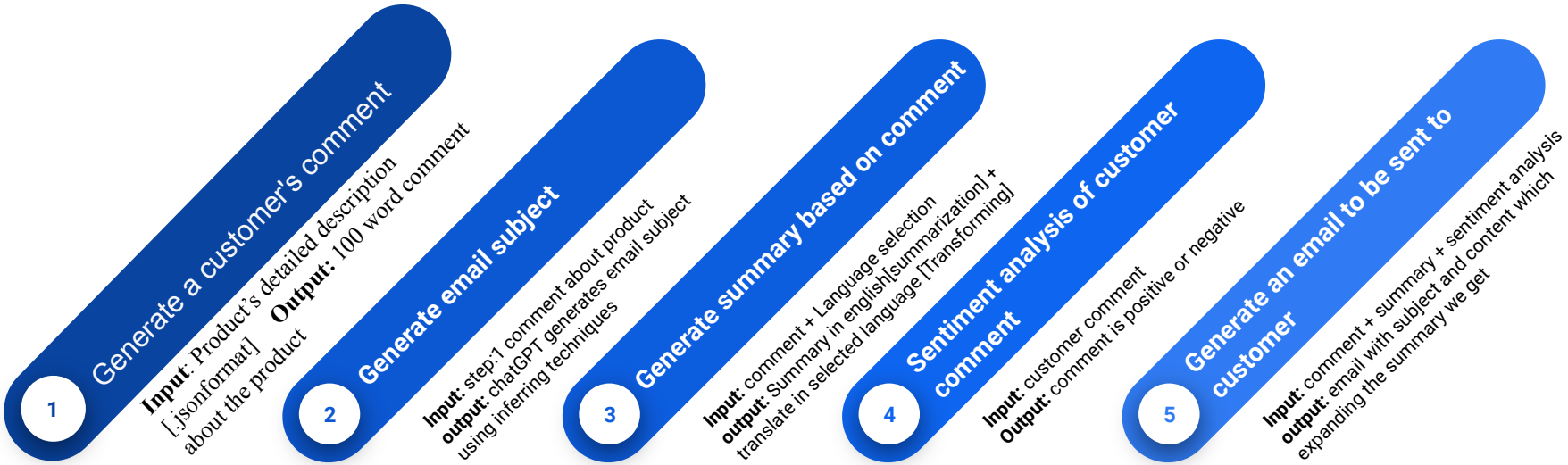
1. Introduction
2. Design
3. Implementation
4. Test
5. Enhancement
6. Conclusion
7. Github repository Link:
8. Bibliography / References

Introduction

This project aims to streamline customer communication by automatically generating informative product-related emails, ensuring excellent customer support across various languages.

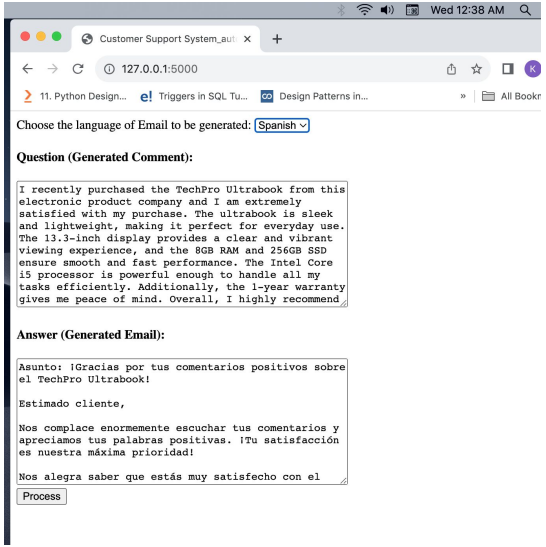
- **Automated Email Customer Support System:** This project centers around the automation of sending personalized emails to customers with product-related messages, leveraging ChatGPT's capabilities to create a web-based system.
- **Prerequisite:** It is recommended to first complete the "Customer Support System" project, where ChatGPT is used to develop a system for answering website-related questions, as this project builds upon that foundation.
- **Overview:** Imagine you are a customer service assistant at a major electronics store. The store's website offers various features, including language selection, an extensive product range categorized with detailed descriptions, and a web interface designed for this project.
- **Language Selection:** Customers can select their preferred language on the website, enhancing their experience.
- **Product Messaging:** The store's products are categorized into different segments, each with comprehensive descriptions. The objective of this project is to automate the process of sending customized product messages to customers via email.

Design


1
Generate a customer's comment
Input: Product's detailed description [.jsonformat] about the product
Output: 100 word comment
2
Generate email subject
Input: step:1 comment about product
output: chatGPT generates email subject using inferring techniques
3
Generate summary based on comment
Input: comment + Language selection
output: Summary in english[summarization] + translate in selected language [Transforming]
4
Sentiment analysis of customer comment
Input: customer comment
Output: comment is positive or negative
5
Generate an email to be sent to customer
Input: comment + summary + sentiment analysis
output: email with subject and content which expanding the summary we get

Design view

Web interface which supports all the steps of the designing process. Like the below screenshot, which has language selection dropdown menu, automated message section, answer section and button which initiate the process.



Implementation

- ➔ **Tools and TEchnology:** python Flask to run web applications, OpenAI secret key
- ➔ **Prerequisites:**
  - ◆ Create virtual environment or go to already created virtual environment using command: **workon <virtual environment name>**
  - ◆ Download Flask and translate packages in your virtual environment. Using command:

```
bash: no job control in this shell
macs-MacBook-Pro:CS589 mac$ workon chatGPT
(chatgpt) macs-MacBook-Pro:CS589 mac$ pip install translate
Collecting translate
```

- ➔ Flask command: pip install flask
- ➔ Or you can combine install:

```
(chatgpt) macs-MacBook-Pro:CS589 mac$ pip install flask
Requirement already satisfied: flask in /Users/mac/.virtualenvs/chatgpt/lib/python3.10/site-packages (2.3.3)
Requirement already satisfied: Werkzeug>=2.3.7 in /Users/mac/.virtualenvs/chatgpt/lib/python3.10/site-packages (from flask) (2.3.7)
Requirement already satisfied: Jinja2>=3.1.2 in /Users/mac/.virtualenvs/chatgpt/lib/python3.10/site-packages (from flask) (3.1.2)
Requirement already satisfied: itsdangerous>=2.1.2 in /Users/mac/.virtualenvs/chatgpt/lib/python3.10/site-packages (from flask) (2.1.2)
Requirement already satisfied: click>=8.1.3 in /Users/mac/.virtualenvs/chatgpt/lib/python3.10/site-packages (from flask) (8.1.7)
Requirement already satisfied: blinker>=1.6.2 in /Users/mac/.virtualenvs/chatgpt/lib/python3.10/site-packages (from flask) (1.6.2)
Requirement already satisfied: MarkupSafe>=2.0 in /Users/mac/.virtualenvs/chatgpt/lib/python3.10/site-packages (from Jinja2>=3.1.2->flask) (2.1.3)
(chatgpt) macs-MacBook-Pro:CS589 mac$ pip install Flask openai translate
Requirement already satisfied: Flask in /Users/mac/.virtualenvs/chatgpt/lib/python3.10/site-packages (2.3.3)
Requirement already satisfied: openai in /Users/mac/.virtualenvs/chatgpt/lib/python3.10/site-packages (0.28.0)
Requirement already satisfied: translate in /Users/mac/.virtualenvs/chatgpt/lib/python3.10/site-packages (3.6.1)
Requirement already satisfied: Werkzeug>=2.3.7 in /Users/mac/.virtualenvs/chatgpt/lib/python3.10/site-packages (from Flask) (2.3.7)
```

- Create .env file in your project folder and set the variable OPENAI_API_KEY= “your key” and ORG_ID = “your ID”

Implementation: import libraries, data & set key

CS589

app_json.py 2

emailToCustomer_automated > app_json.py > ...

```python
# Works perfectly but slow without usinf Json file. Data is stored in .py file
import openai
import os
import time
from flask import Flask, render_template, request
from dotenv import load_dotenv
#from products_data import products_data
import json
import random

app = Flask(__name__)

# Load product data from product.json
with open('products_data.json', 'r') as json_file:
    products_data = json.load(json_file)

load_dotenv()
openai.api_key = os.getenv("OPENAI_API_KEY")
if openai.api_key is None:
    raise ValueError("API key not found in environment variables")
```

ailToCustomer_automated > {} products_data.json > ...


```json
{
    "TechPro Ultrabook": {
        "name": "TechPro Ultrabook",
        "category": "Computers and Laptops",
        "brand": "TechPro",
        "model_number": "TP-UB100",
        "warranty": "1 year",
        "rating": 4.5,
        "features": ["13.3-inch display", "8GB RAM",
                "256GB SSD", "Intel Core i5 processor"],
        "description": "A sleek and lightweight ultrabook for everyday use.",
        "price": 799.99
    },
    "BlueWave Gaming Laptop": {
        "name": "BlueWave Gaming Laptop",
        "category": "Computers and Laptops",
        "brand": "BlueWave",
        "model_number": "BW-GL200",
        "warranty": "2 years",
        "rating": 4.7,
        "features": ["15.6-inch display", "16GB RAM",
                "512GB SSD", "NVIDIA GeForce RTX 3060"],
        "description": "A high-performance gaming laptop for an immersive experience.",
        "price": 1199.99
    },
    "PowerLite Convertible": {
        "name": "PowerLite Convertible",
        "category": "Computers and Laptops",
        "brand": "PowerLite",
        "model_number": "PL-CV300",
        "warranty": "1 year",
        "rating": 4.3,
        "features": ["14-inch touchscreen", "8GB RAM",
                "256GB SSD", "360-degree hinge"],
        "description": "A versatile convertible laptop with a responsive touchscreen.",
        "price": 699.99
    },
```

Implementation: Step wise

Step:1=>

```python
# Step 1: Generate a Customer's Comment
def generate_customer_comment(products_data):
    prompt=f"""
    Assume that you are a customer to an electronic product company.
    Write a 100-word only comment about the products delimited by triple backticks in its own language.
    Products: ```{products_data}```
    """
    response = get_completion_with_rate_limit(prompt)
    return response
```

Step:2 =>

```python
# Step 2: Generate Email Su
def generate_email_subject(comment):
    prompt=f"""
    Assuming that you provide customer support for an electronic product company.
    Based on the customer comment delimited in triple backticks, suggest a short email subject to respond to the customer.
    Comment= ```{comment}```
    """
    response = get_completion_with_rate_limit(prompt)
    return response
```

Step:3=>

```python
# Step 3: Generate Customer Comment Summary
def summarize_comment(comment):
    prompt=f"""
    Assuming that you provide customer support for an electronic product company.
    Provide a concise summary in 50 words of the following customer comment delimited in triple backticks. Comment: ```{comment}```
    """
    response = get_completion_with_rate_limit(prompt)
    return response

def translate_summary_with_chatgpt(language, summary):
    prompt= f"""
    Translate the following summary delimited by triple backticks to the language delimited by <>.
    Language:```{language}```
    Summary:<{summary}>
    """
    response = get_completion_with_rate_limit(prompt)
    return response
```

Implementation: Step wise

Step:4=>

```
# Step 4: Analyze Customer Comment Sentiment
def analyze_sentiment(comment):
    prompt=f"""
    Assuming that you provide customer support for an electronic product company.
    What is the sentiment of the comment delimited in triple backticks? Is it positive or negative?
    Comment: ```{comment}```
    """
    max_tokens=10
    response = get_completion_with_rate_limit(prompt)
    sentiment = response.lower()
    if "positive" in sentiment:
        return "positive"
    elif "negative" in sentiment:
        return "negative"
    else:
        return "neutral"
```

Step:5 =>

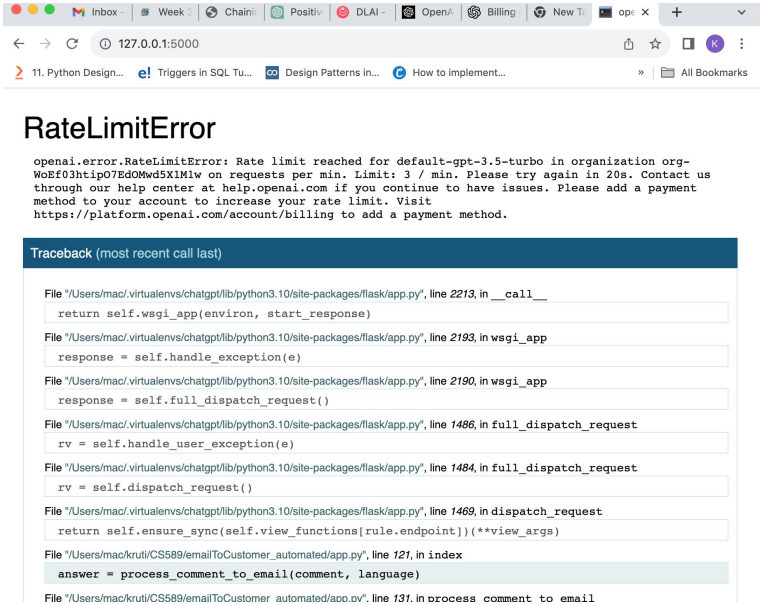
```
# Step 5: Generate Customer Email
def generate_customer_email(summary, sentiment, email_subject, language):
    if sentiment == "positive":
        response_text = "We're thrilled to hear your feedback and appreciate your positive words. Your satisfaction is our top priority!"
    elif sentiment == "negative":
        response_text = "We're truly sorry to hear about your experience. Your feedback is crucial, and we'll strive to address your concerns."
    else:
        response_text = "Thank you for your feedback! We're always looking to improve, and your insights are valuable."
    prompt = f"""
    Assuming that you provide customer support for an electronic product company.
    Given the specified parameters below:
    - Comment summary enclosed in backticks (`{summary}`)
    - Our response text enclosed in triple quotes (\"\"\"{response_text}\"\"\")
    - Translate the Email subject enclosed in angle brackets ({email_subject}) to language \"{language}\"
    Write a complete email responding to the customer's comment using the language \"{language}\".
    """
    response = get_completion_with_rate_limit(prompt)
    return response
```

Implementation: main

```python
@app.route('/', methods=['GET', 'POST'])
def index():
    answer = ""
    comment = generate_customer_comment(products_data)
    print("A customer comment has been generated.")
    if request.method == 'POST':
        language = request.form.get('language')  # Fetch language input from the webpage
        print(f"Selected language: {language}")
        answer = process_comment_to_email(comment, language)
    return render_template('index.html', question=comment, answer=answer)

def process_comment_to_email(comment, language):
    # Step 2: Generate Email Subject
    email_subject = generate_email_subject(comment)
    print(f"An email subject is generated from the customer's comment.")
    # Step 3: Generate Customer Comment Summary
    summary = summarize_comment(comment)
    print("A Summary is generated from the customer comment.")
    translated_summary = translate_summary_with_chatgpt(language, summary)
    print("The summary has been translated to the requested language.")
    # Step 4: Analyze Customer Comment Sentiment
    comment_sentiment = analyze_sentiment(comment)
    print(f"Sentiment of the comment is detected as: {comment_sentiment}")
    # Step 5: Generate Customer Email
    email_content = generate_customer_email(translated_summary, comment_sentiment, email_subject, language)
    print("A customer email has been generated.")
    return email_content

if __name__ == '__main__':
    app.run(debug=True)
```

Error and Error Solution

While running the app it Throws ‘RateLimitError’ which is not the coding error but openAI do not allow you to run continuously so following is error screenshot and I added line of code to my file to solve the error.

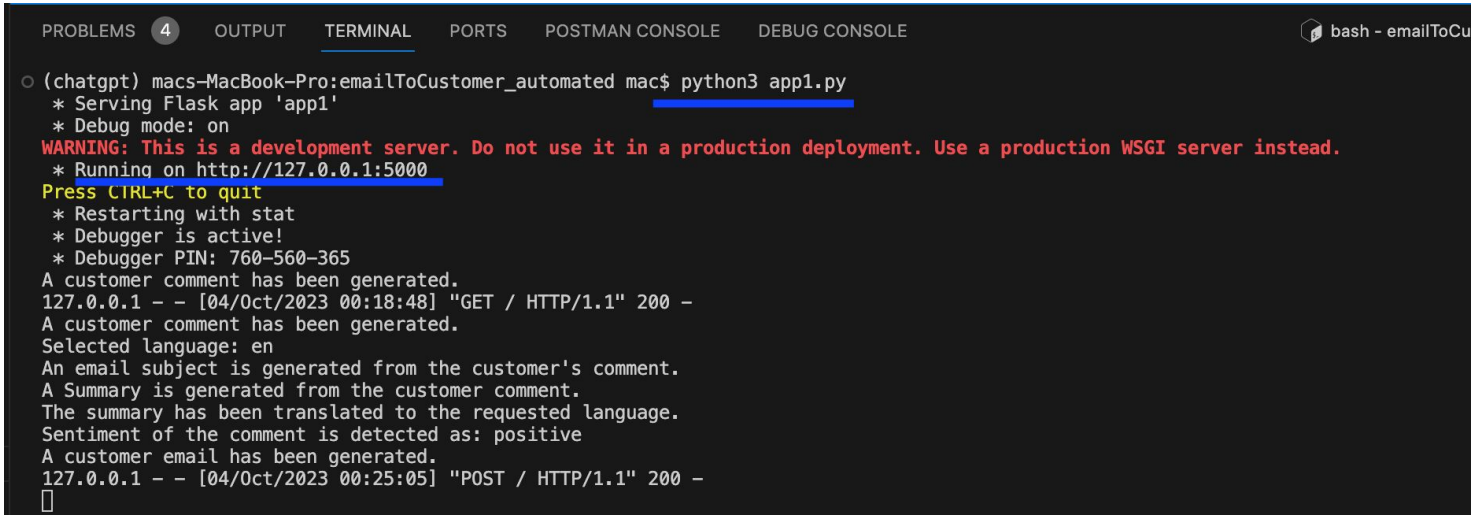



```python
# Define a rate limiting mechanism
RATE_LIMIT = 3  # Requests per minute
RATE_LIMIT_PERIOD = 60  # 60 seconds in a minute

# Define a timestamp variable to track the last request time
last_request_time = 0

def get_completion_with_rate_limit(prompt):
    global last_request_time

    # Calculate the time since the last request
    current_time = time.time()
    time_since_last_request = current_time - last_request_time

    # Check if the rate limit has been reached, and if so, wait
    if time_since_last_request < RATE_LIMIT_PERIOD:
        time_to_wait = RATE_LIMIT_PERIOD - time_since_last_request
        time.sleep(time_to_wait)

    # Make the API request
    messages = [{"role": "user", "content": prompt}]
    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=messages,
        temperature=0,
    )

    last_request_time = time.time()

    return response.choices[0].message["content"]
```

Executing project

To run the code write python3 <filename>.py and server is running on port 5000.

Open browser and type http://127.0.0.1:5000 to use the project.



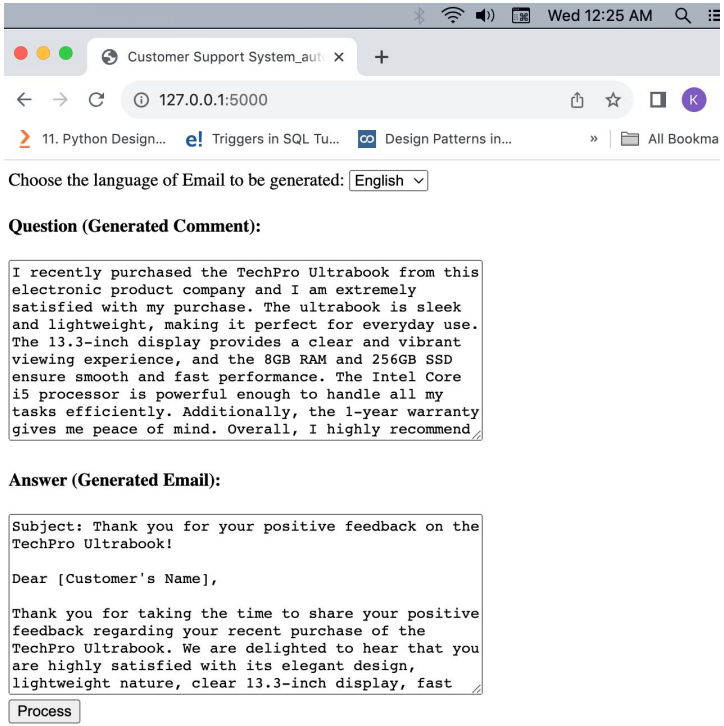
Test Cases

Expected output:

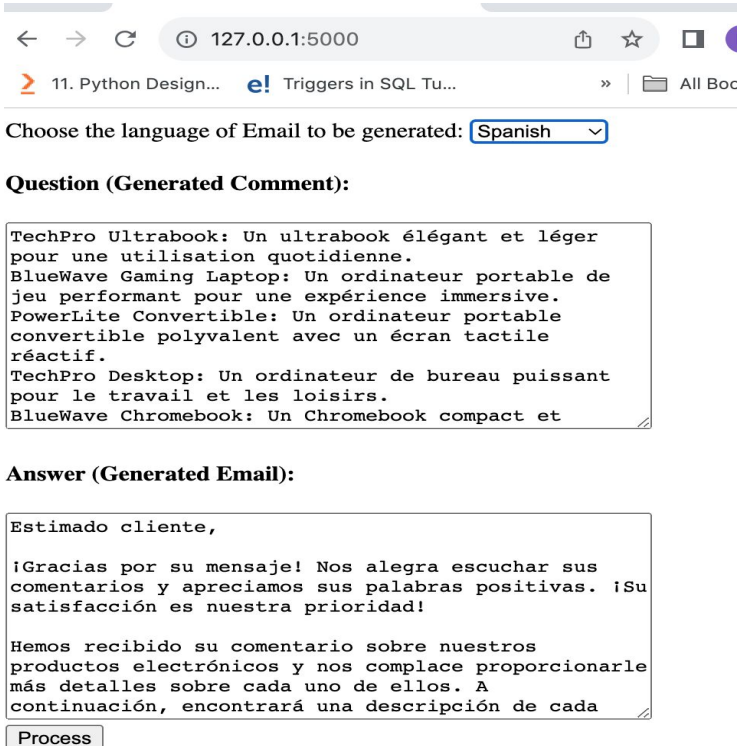
ID	Question	Answer
1	English	English
2	English	Non-English
3	Non-English	English
4	Non-English	Non-English Note: ▪ ChatGPT can infer the language used in Question and then generate the Answer with the same language

Test cases output for all combinations:

1) English to English

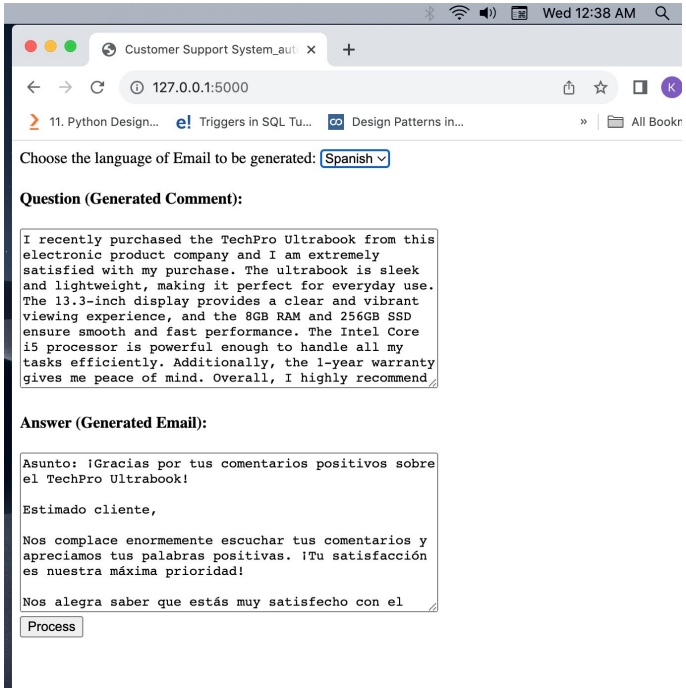


2)spanish to spanish

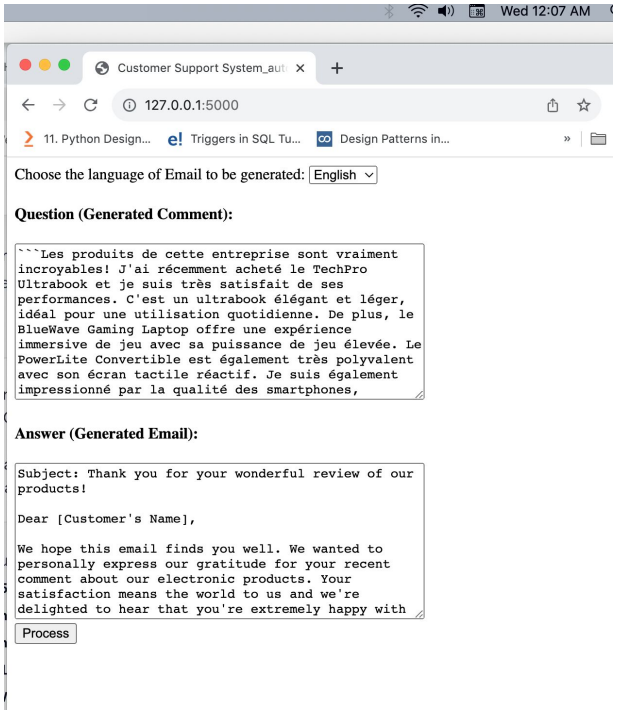


Test cases output for all combinations:

3) English to Spanish



4) spanish to english



Enhancements

1. Works as a translator app when person do not know the word or sentence meaning.
2. Integrate with customer Relationship management to maintain a comprehensive record of customer interactions, enabling better tracking and follow-up.
3. Integrate a chatbot alongside ChatGPT to provide immediate responses to common customer queries.
4. Implement real-time analytics to monitor customer interactions and sentiments. This data can be used to further improve the support system by identifying trends and areas where customers need assistance the most.

Conclusion

The Automated Email Customer Support System utilizes ChatGPT to automate personalized product email messages for enhanced customer support. With multi-language support, sentiment analysis, and future enhancements, it promises to elevate customer interactions and satisfaction. This project marks a significant step towards efficient, data-driven, and customer-centric support within a large electronics store.

Github Links:

https://github.com/DKruti/Machine-Learning/tree/master/Generative%20AI/Automated%20Email%20to%20customer

References

- https://hc.labnet.sfbu.edu/~henry/sfbu/course/deeplearning_ai/chatgpt_prompt_eng_for_developer/slide/exercise_chatgpt_prompt_eng_for_developer.html
- https://learn.deeplearning.ai/chatgpt-prompt-eng/lesson/2/guidelines