AIAC Lab Test

Batch – AI 04

2403A52092

Q1.a:

Task: Connect to an Email API (SendGrid/Mailgun).

Prompt:

I need to integrate an email sending func Θon into a Python applica Θon using the SendGrid API . I've already signed up for a SendGrid account. Provide a step-by-step guide on how to set up the environment and include a complete, runnable Python code snippet that sends a simple text email to a single recipient. The code should clearly show where to insert the API key and the recipient/sender details.

Code:

```
"""
SendGrid Email Integration - Complete Working Example
This script demonstrates how to send a simple text email using the
SendGrid API.
"""

import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
from dotenv import load_dotenv


# Load environment variables from .env file
load_dotenv()


def send_email(sender_email, recipient_email, subject,
plain_text_content):
    """
    Send a simple text email using SendGrid.


    Args:
        sender_email (str): Email address of the sender
        recipient_email (str): Email address of the recipient


        subject (str): Email subject line
        plain_text_content (str): Plain text content of the email


    Returns:
        bool: True if email sent successfully, False otherwise
    """
    try:
        # Get API key from environment variable
        api_key = os.getenv( 'SENDGRID_API_KEY' )


        if not api_key:
            print( "Error: SENDGRID_API_KEY environment variable not
found." )
            print( "Please set your API key in the .env file or as an
environment variable." )
            return False

        # Create SendGrid client
        sg = SendGridAPIClient(api_key)

        # Create email message
        message = Mail(
            from_email=sender_email,
            to_emails=recipient_email,
            subject=subject,
            plain_text_content=plain_text_content
        )

        # Send email
        response = sg.send(message)

        # Check if email was sent successfully (status code 202)
        if response.status_code == 202:
            print( f"✓ Email sent successfully to {recipient_email} ")
            print( f"  Status Code: {response.status_code} ")
            return True
        else:
            print( f"✗ Failed to send email. Status Code:
```

```
{response.status_code} ")
          return False

   except Exception as e:
      print( f"✗ Error sending email: {str(e)} ")
      return False



def send_email_with_html(sender_email, recipient_email, subject,
                    plain_text_content, html_content):
   """
   Send an email with both plain text and HTML content using
SendGrid.

   Args:
      sender_email (str): Email address of the sender
      recipient_email (str): Email address of the recipient
      subject (str): Email subject line
      plain_text_content (str): Plain text version of the email
      html_content (str): HTML version of the email

   Returns:
      bool: True if email sent successfully, False otherwise
   """
   try:
      api_key = os.getenv( 'SENDGRID_API_KEY' )
```

```
    if not api_key:
        print( "Error: SENDGRID_API_KEY environment variable not
found." )
        return False

    sg = SendGridAPIClient(api_key)

    # Create email message with both plain text and HTML
    message = Mail(
        from_email=sender_email,
        to_emails=recipient_email,
        subject=subject,
        plain_text_content=plain_text_content,
        html_content=html_content
    )

    response = sg.send(message)

    if response.status_code == 202:
        print( f"✓ HTML Email sent successfully to
{recipient_email} ")
        print( f"  Status Code: {response.status_code} ")
        return True
    else:
        print( f"✗ Failed to send HTML email. Status Code:
```

```
{response.status_code} ")
      return False


   except Exception as e:
     print( f"✗ Error sending HTML email: {str(e)} ")
     return False

#
=============================================================================
========
# EXAMPLE USAGE - Modify these values with your actual email
addresses
#
=============================================================================
========

if __name__ == "__main__" :
   # ========== CONFIGURE THESE VALUES ==========
   SENDER_EMAIL = "2403a52089@sru.edu.in"   # Use a verified sender
email in SendGrid
   RECIPIENT_EMAIL = "nishantbommathiofficial@gmail.com"   # Email
address to send to
   EMAIL_SUBJECT = "Hello from SendGrid!"
   EMAIL_BODY = "This is a test email sent using the SendGrid API
with Python."
```

```
# ========== END CONFIGURATION ==========

print( "=" * 60)
print( "SendGrid Email Integration Example" )
print( "=" * 60)

# Send simple text email
print( "\n[1] Sending simple text email..." )
send_email(
  sender_email=SENDER_EMAIL,
  recipient_email=RECIPIENT_EMAIL,
  subject=EMAIL_SUBJECT,
  plain_text_content=EMAIL_BODY
)

# Send HTML email
print( "\n[2] Sending HTML email..." )
html_body = """
<html>
  <body>
    <h1>Hello from SendGrid!</h1>

    <p>This is an <strong>HTML email</strong> sent using the
SendGrid API.</p>
    <p><em>This format supports rich text
```

```
formatting.</em></p>
        </body>
    </html>
    """

    send_email_with_html(
        sender_email=SENDER_EMAIL,
        recipient_email=RECIPIENT_EMAIL,
        subject= "HTML Email Test" ,
        plain_text_content= "This is the plain text version for email
clients that don't support HTML." ,
        html_content=html_body
    )

    print( "\n" + "=" * 60)
    print( "Email sending demonstration complete!" )
    print( "=" * 60)
```

OUTPUT:

OBSERVATION:

Security: The API key must be managed via an environment variable (SENDGRID_API_KEY), not hardcoded

Q1.b:

Task: Connect to an Email API (SendGrid/Mailgun).

Prompt:

I need to implement SendGrid failure handling in Python.

1. Provide the Python try...except structure to catch immediate API delivery excep Θons (SendGridAPIClientError).

2. Explain the setup for handling asynchronous bounce events (delayed failures). Specifically, detail the necessary SendGrid Webhook configura Θon and the basic Flask endpoint logic required to securely receive and process the incoming bounce data.

.

CODE:

```
import os
from sendgrid import SendGridAPIClient
from sendgrid.helpers.mail import Mail
from python_http_client.exceptions import SendGridAPIClientError

# --- Configuration (Assuming API Key is set as an Environment
Variable) ---
API_KEY = os.environ.get( 'SENDGRID_API_KEY' )
SENDER_EMAIL = 'verified_sender@example.com'
RECIPIENT_EMAIL = 'recipient@example.com'
```

```
def send_email_with_error_handling():
    if not API_KEY:
        print( "ERROR: SENDGRID_API_KEY is not set." )
        return

    message = Mail(
        from_email=SENDER_EMAIL,
        to_emails=RECIPIENT_EMAIL,
        subject= 'Test Email' ,
        html_content= 'Sending test.'
    )

    try:
        # Initialize the SendGrid client
        sg = SendGridAPIClient(API_KEY)

        # Attempt to send the email

        response = sg.send(message)

        # Successful delivery request (Status 202: Accepted)
        print( f"✅ Email request accepted. Status Code:
{response.status_code} ")

    except SendGridAPIClientError as e:
        # Catches failures like invalid API key, formatting issues,
etc.
        print( "❌ Immediate API Delivery Failure Caught!" )
        print( f"Status Code: {e.status_code} ")
        print( f"Error Details (Response Body): {e.body.decode( 'utf-
8')}")

        # --- Custom Failure Logic ---
        # Log the error, notify an admin, or queue for manual
review.
        # ---

    except Exception as e:
        # Catches any other unexpected issues (e.g., network
timeout)
        print( f"⚠ An unexpected error occurred: {e}")

if __name__ == "__main__" :
    send_email_with_error_handling()
```

Output:

ObservaƟon:

Dual Handling:  Need to handle synchronous errors  (via try...except

SendGridAPIClientError) and asynchronous bounces  (via SendGrid Webhooks ).

Q2.a:

Task: Convert a Python dic Θonary-based script into Go. .

Prompt:  Translate the following Python script, which defines a dic Θonary (user_data) and iterates through its items to print key-value pairs, into an equivalent, runnable Go program. The Go code should use a map data structure and clearly demonstrate the ini ΘalizaΘon and itera Θon paΣern

Python Script:

```
user_data = {
   "username": "coder123",
   "id": 4091,
   "acΘve": True
}
for key, value in user_data.items():
print(f'{key}: {value}')
```

CODE:

PYTHON:

```
# Python equivalent of 2.go
```

```
def main():
    # Equivalent user data map from Go example
    user_data = {
        "username" : "coder123" ,
        "id": 4091,
        "active" : True,
    }

    print( "--- User Data (Python dict) ---" )
    for key, value in user_data.items():
        print( f"{key}: {value} ")

if __name__ == "__main__" :
    main()
```

GO:

```
package  main

import (
    "fmt"
)
```

```
func main() {
   // In Go, a map key must be consistent (here, string), and
   // the value type must also be consistent. We use interface{}
   // (Go's equivalent of Python's dynamic typing) to hold mixed
types.
   userData := map[string]interface {}{
      "username" : "coder123" ,
      "id":      4091,
      "active" :   true,
   }

   // Iteration in Go's map: use a 'for range' loop.
   // It automatically returns the key and value for each element.
   fmt.Println( "--- User Data (Go Map) ---" )
   for key, value := range userData {
      // The value is printed, and Go's fmt package handles the
formatting
      // of the underlying data type held by the interface{}.
      fmt.Printf( "%s: %v\n" , key, value)
   }
}
```

Ouput:

ObservaΘon:

Type Bridge: To translate a heterogeneous Python dict, the Go map must use interface{} as the value type

Q2.b:

Task: Explain how data structures change.

Prompt:

Explain the differences in the primary key- value data structure when transla Θng from Python's dict to Go's map. Specifically, address:

Go's requirement for type homogeneity and how the heterogeneous Python dic Θonary must be handled.

The difference between Python's direct key access and Go's 'comma ok' idiom for checking key existence.

OUTPUT:

Data Structure Changes (Short Explana Θons)

 Type Homogeneity: Python dict is heterogeneous (mixed types allowed).

Go map is homogeneous (all values must be the same type). To handle mixed data, Go requires se ţng the value type as interface{} (the empty interface). This allows it to hold any type, but requires type asser Θon upon retrieval.

 Key Existence Check:  Python raises a KeyError if a key is missing. Go map returns the zero value  (e.g., 0, "") and a boolean flag. You must use the 'comma ok' idiom  (value, ok := myMap["key"]) to check if the key was actually present (ok is true) or if you just got the zero value (ok is false).

OBSERVATION:

Existence Check: Go lacks Python's KeyError and requires the 'comma ok' idiom (value, ok := ...) for safe key existence checks.