UNIT TESTING

The code sets up a Flask application that allows users to submit a form with participant details and sends emails with attachments to the provided email addresses. The main components of the code are as follows:

1. Flask App Initialization: -

The Flask application is created using the `Flask` class from the `flask` module. - Environment variables are loaded from a `. env` file using `load_dotenv` from the `dotenv` package. The application configuration is set, including the password retrieved from the environment variable.

```
1  from flask import Flask, url_for, redirect, request, render_template, json
2  from dotenv import load_dotenv
3  import smtplib
4  from email.mime.text import MIMEText
5  from email.mime.multipart import MIMEMultipart
6  from email.mime.application import MIMEApplication
7
8  import os
9  # creates flask instance
10 app = Flask(__name__)
11 load_dotenv()
12
13 app.config['PASSWORD'] = os.environ.get('PASSWORD')
14 password = app.config['PASSWORD']
```

2. Routes and Request Handling: - The root route (`'/'`) is defined using the `@app.route()` decorator. - It handles both GET and POST requests. - For GET requests, the `''index.html''` template is rendered, and the `participants` list is passed to the template. For POST requests, form data is collected from the request, and a dictionary with the data is appended to the `participants` list. The user is then redirected back to the root URL.

```
19  @app.route('/', methods=['GET', 'POST'])
20  def index():
21      # collecting values from the form
22      if request.method == 'POST':
23          firstName = request.form['firstName']
24          lastName = request.form['lastName']
25          email = request.form['email']
26          ipaddress = request.form['ipaddress']
27          port = request.form['port']
28          name = firstName + " " + lastName
29          form_data = {
30              'name': name,
31              'email': email,
32              'ipaddress': ipaddress,
33              'port': port
34          }
35          participants.append(form_data)
36          return redirect(url_for('index'))
37      return render_template("index.html", participants=participants)
```

3. Sending Invites: - Another route (`'/sendInvites'`) is defined to handle sending email invites. - It also handles both GET and POST requests. - For POST requests, the SMTP server details are set, and for each participant in the `participants` list, an email is composed with an attached JSON file containing participant data. - The email is sent using the configured SMTP server and the `smtplib` module. - After sending the emails, the `participants` list is cleared, and the user is redirected back to the root URL.

```
41  def sendkInvites():
42      global participants
43      # establishing the smtp server
44      sender_email = 'vamsichowdary.dk@gmail.com'
45      sender_password = password
46      message = MIMEMultipart()
47      message['From'] = sender_email
48      smtp_server = 'smtp.gmail.com'
49      smtp_port = 587
50      smtp_username = sender_email
51      smtp_password = sender_password
52
53      # sending email to every paticipant
54      for participant in participants:
55          # formating the json object
56          individual = [{'main': participant}]
57          others = [d for d in participants if d != participant]
58          individual.append({'others': others})
59
60          # creating a json file
61          with open('form_data.json', 'w') as f:
62              json.dump(individual, f)
63          # creating email body
64          message['Subject'] = 'Form Data'
65          recipient_email = participant['email']
66          message['To'] = recipient_email
67          body = "Please see attached JSON file for form data."
68          message.attach(MIMEText(body, 'plain'))
```

4. Running the Application: - The `if __name__ == "__main__":` block ensures that the Flask app is only run if the script is executed directly (not imported as a module). - The development server is started using the `app.run()` method.

Overall, the code sets up a Flask app with routes for displaying a form, collecting participant data, and sending emails with attachments. The `participants` list serves as a temporary storage for the submitted participant data, and the SMTP server details are used to send the emails.

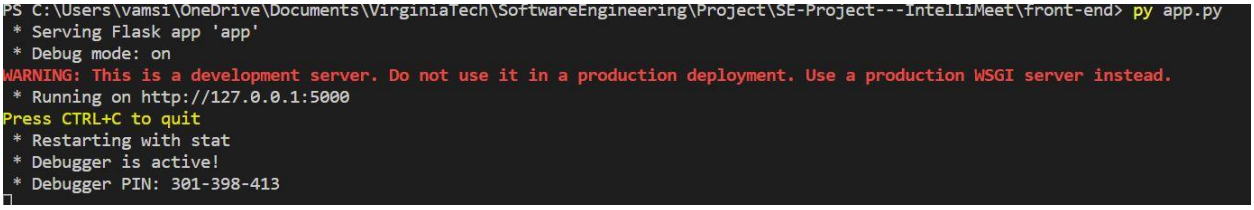
Code related to Testing:

```python
from unittest import TestCase
from flask import Flask
from flask_testing import TestCase as Ft
from app import app

class FlaskAppTestCase(TestCase):
    def create_app(self):
        app.config['TESTING'] = True
        return app

    def test_index_page(self):
        response = self.client.get('/')
        self.assert200(response)
        self.assert_template_used('index.html')

    def test_send_invites(self):
        # Add participant data for testing
        participant_data = {
            'firstName': 'John',
            'lastName': 'Doe',
            'email': 'johndoe@example.com',
            'ipaddress': '127.0.0.1',
            'port': '8080'
        }
        with self.client.session_transaction() as session:
            session['participants'] = [participant_data]

        response = self.client.post('/sendInvites')
        self.assert200(response)
        # Assert that the participants list is empty after sending invites
        with self.client.session_transaction() as session:
            self.assertEqual(session.get('participants'), [])

if __name__ == '__main__':
    unittest.main()
```

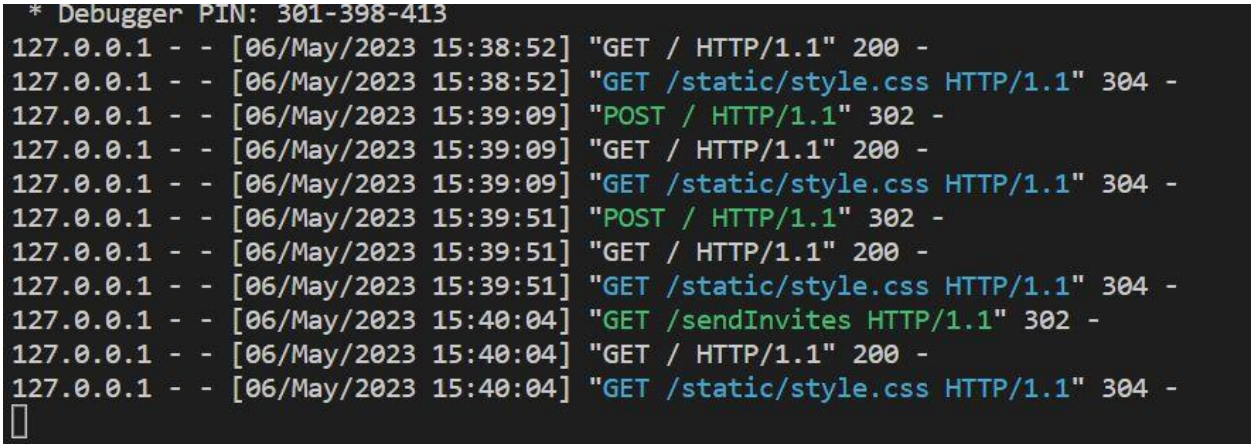
Unit Test:

- The assert200 assertion is a convenient method provided by the FlaskTestCase class from flask_testing.
- It verifies that the HTTP response status code is equal to 200, indicating a successful response.
- By using self.assert200(response) in test cases, they are verifying that the responses received from the Flask application for the respective routes are returning a status code of 200.
- Passing test cases with a status code of 200 indicate that the routes are functioning correctly and returning the expected responses.

Server Started:



Test Cases passed.



The 2 functions passed sucessfully.