AUTOMATIC BIRTHDAY&ANNIVERSARY WISHES SENDER USING FLASK

ABSTRACT:

This Flask project aims to automate the process of sending wishes via email on specific dates. The application is structured around Flask's routing system, allowing users to interact with various HTML templates through defined endpoints. Users can input necessary details such as sender's email, sender's password, receiver's email, message content, and the date for sending the wish. Upon submission of the form, the application saves this data to a log file for tracking purposes. To handle the email sending functionality, a separate function is defined within the application. This function encapsulates the logic required to send an email using the provided sender credentials, recipient email, and message content.

Overall, this project leverages Flask's capabilities for web development and integrates email sending functionality to automate the process of sending wishes, offering users a convenient and efficient means of scheduling and dispatching personalized messages.

TABLE OF CONTENT

LIST OF FIGURES

CHAPTER 1

INTRODUTION

Welcome to our innovative project designed to streamline the process of sending heartfelt wishes and greetings through automated email dispatches. In today's dynamic world, maintaining personal connections and fostering relationships is paramount, yet the busyness of daily life often makes it challenging to remember and execute these gestures in a timely manner. Our Flask-based solution addresses this by providing a user-friendly platform where users can effortlessly schedule and send wishes on specific dates without the need for manual intervention.

This project harnesses the versatility of Flask, a powerful web framework in Python, to create an intuitive interface where users can input essential details such as sender and recipient email addresses, personalized message content, and the date for dispatching the wish. With just a few clicks, users can schedule wishes for various occasions, ensuring that their sentiments are conveyed promptly and thoughtfully.

Behind the scenes, our application seamlessly integrates with email sending functionalities, enabling automated dispatches based on the scheduled dates provided by the user. Through effective file handling mechanisms, the system maintains comprehensive logs of user interactions and scheduled wishes, ensuring transparency and accountability

Our project caters to individuals seeking to stay connected with loved ones, colleagues, and acquaintances by offering a convenient and efficient solution for sending wishes on important dates. Whether it's birthdays, anniversaries, or special milestones, our platform empowers users to spread joy and positivity effortlessly, enhancing relationships and fostering meaningful connections in the digital age. Join us on this journey to simplify communication and make every occasion memorable with automated wishes.

1.1Overview:

Our Flask-based project aims to revolutionize the way individuals send wishes and greetings by automating the process through email dispatches. At its core, the application offers a user-friendly web interface where users can input necessary details such as sender and recipient email addresses, message content, and the desired date for sending the wish.

The project leverages Flask's robust routing system to provide seamless navigation between different HTML templates, ensuring an intuitive user experience. Through form submissions, users can schedule wishes for various occasions, from birthdays to anniversaries, with ease and efficiency.

Behind the scenes, the application integrates with email sending functionalities, enabling automated dispatches based on the scheduled dates specified by the user. Additionally, comprehensive file handling mechanisms are implemented to log user interactions and scheduled wishes, facilitating transparency and accountability.

In summary, our project offers a convenient solution for individuals seeking to maintain meaningful connections and spread joy through timely wishes. By combining Flask's web development capabilities with email automation, we aim to simplify communication and enhance relationships in the digital age. Join us on this journey to make every occasion memorable with automated wishes

1.2 MODULE DESCRIPTION

MODULES:

- Data Collection.
- Email Server Configuration and Dispatch.
- User Interaction Webpage.

Data Collection and Text File Storage

Objective:

This module is dedicated to capturing user information and organizing Systematically within a text file, creating a foundational dataset for the automated email system.

1.3Data Collection:

Design an intuitive interface or mechanism for users to input their details. This could be a form within a web application, an input terminal in a desktop application, or any user-friendly interface

Data Validation and Formatting:

Implement validation checks to ensure the correctness and completeness of the provided information. Validate email formats, ensure accurate date inputs, and handle potential errors gracefully to maintain data integrity.

File Handling for Data Storage:

Define a file path and filename for storing the user data. Utilize Python's file handling capabilities to open a text file in write mode ('w') for data storage.

Writing User Data to Text File:

Iterate through the collected user data and write it to the text file. Employ formatting techniques to organize user information systematically within the file, ensuring readability and structure.

Error Handling and File Closure:

Implement error handling mechanisms to address potential issues during file operations, ensuring robustness. Utilize Python's with statement to automatically close the file after data writing operations are completed, promoting efficient file handling.

1.4 Email Server Configuration and Dispatch:

Building upon the foundation of collected user data, Module 2 focuses on the establishment of a robust server-side mechanism dedicated to the dispatch of personalized celebratory emails. Central to this module is the configuration of the Simple Mail Transfer Protocol (SMTP) settings essential for email delivery. This includes specifying the SMTP server address, port, and authentication credentials, ensuring a secure and reliable communication channel for outbound emails.

A crucial aspect of this module is the generation of personalized email content using the acquired user data. Functions are developed to craft individualized messages, integrating user-specific details such as names, upcoming celebration

dates, and personalized greetings. These tailored emails serve as a testament to the system's ability to engage users in a meaningful and personalized manner.

Following the email content generation, the module proceeds to implement the email dispatch functionalities. These functions utilize the SMTP configuration to establish connections with the designated server and dispatch the personalized celebratory emails to the respective users. To ensure the reliability of the email delivery process, robust error handling mechanisms and logging

functionalities are integrated, providing insights into potential delivery issues and enhancing the system's resilience.

Optionally, security measures such as encryption for sensitive information like SMTP passwords are considered to fortify the system against potential vulnerabilities.

Furthermore, the exploration of secure email delivery methods might be warranted, especially if the system handles sensitive user information. The culmination of these efforts in Module 2 facilitates a seamless and efficient process for delivering personalized celebratory emails, further enriching the user experience and fostering stronger connections with the user base.

1.5 Flask Integration for User Interface:

Module 3 marks the integration of the automated email system into a userfriendly web interface, leveraging the Flask framework to facilitate interaction and data input. This module is characterized by its emphasis on user-centric design and seamless integration of functionalities within the Flask application.

The configuration of the Flask application serves as the initial step, setting up routes and defining the structure for user interaction. The application's routes are meticulously designed to accommodate various stages of the user journey, including data input, submission, and potential data review. The user interface is carefullycrafted using HTML templates or forms within Flask, providing an intuitive platform for users to input their information.

A paramount consideration within this module is user feedback and data validation. Robust feedback mechanisms are implemented to inform users about the status of their data submissions, ensuring transparency and user reassurance. Data validation routines are integrated to scrutinize and validate user inputs, safeguarding against erroneous or incomplete data submissions.

Integration with the email system constitutes a pivotal aspect of this module. Upon successful data submission, the Flask application triggers the email-sending functionalities, orchestrating the generation and dispatch of personalized celebratory emails. This seamless integration empowers users to engage with the system effortlessly, fostering a cohesive user experience.

Optionally, enhancements to the user interface can be explored to offer additional features such as data review, editing, or deletion of user information. This expands the usability of the application, providing users with control over their submitted data and reinforcing the system's transparency.

In summary, Module 3 encapsulates the user-facing aspect of the automated email system, employing Flask's versatility to create an engaging and interactive platform for users to input their information and experience the seamless process of receiving personalized celebratory emails

1.File Handling:

- Responsible for managing logging operations, ensuring user interactions and scheduled wishes are recorded.

- Facilitates transparency and accountability by maintaining comprehensive logs of application activities.
- Enables error tracking and debugging, crucial for diagnosing issues and improving system reliability. - Provides a structured approach to storing and retrieving data, enhancing the organization of the project.

- Supports scalability, allowing the application to handle a growing volume of user interactions and data.

2. Sending Email:

- Essential for automating the process of sending wishes and greetings via email on scheduled dates.

- Enables seamless communication between users by delivering personalized messages promptly.

- Facilitates engagement and relationship-building, fostering meaningful connections between individuals.

- Enhances user experience by providing a convenient and efficient means of conveying sentiments.

- Empowers users to maintain connections and celebrate special occasions regardless of geographical distance.

3. Validating Date

- Crucial for determining when scheduled wishes should be dispatched, ensuring timely delivery. - Facilitates automation by comparing the current date with scheduled dates specified by users.

- Enables the application to trigger email sending processes only when the scheduled date matches the current date.

- Enhances reliability by preventing premature or delayed dispatch of wishes based oninaccurate date comparisons.

- Promotes user satisfaction by ensuring that wishes are sent precisely when intended, enhancing the effectiveness of the application.

4. Interface:

- Provides a user-friendly platform for interacting with the application, enhancing accessibility and usability.
- Enables users to input essential details such as sender and recipient email addresses, message content, and scheduled dates.

- Facilitates seamless navigation between different

- Facilitates seamless navigation between different pages and functionalities, optimizing the user experience.

- Enhances engagement by presenting information in a visually appealing and intuitive manner.

- Promotes adoption and usage of the application by offering an intuitive and responsive interface.

CHAPTER 2

SYSTEM ANALYSIS

2.1 TECHNICAL FESIBILITY

Analysis of Required Technology:

Flask, Python's built-in modules (datetime), Jinja2, Flask-Mail, SMTP servers, HTML/CSS/JavaScript for frontend.

Availability of Resources:

All required technologies are readily available and widely used, ensuring ease of access.

Scalability:

The project can be scaled up to accommodate increased user demand by optimizing code and infrastructure.

Integration Potential:

Integration with external services like SMTP servers for email sending is straightforward, enhancing feasibility.

2.2 OPERATIONAL FESIBILITY

User Acceptance:

The project aims to simplify the process of sending wishes, garnering positive reception from users

Training Requirements:

Minimal training required for users due to the intuitive interface and straightforward functionalities.

Resource Utilization:

Efficient utilization of system resources ensures smooth operation even under peak loads.

Impact on Existing Operations:

Integration of the system won't disrupt existing operations and can enhance productivity.

2.3 ECONOMICAL FESIBILITY:

Cost-Benefit Analysis:

The benefits of automated wish sending, such as time savings and enhanced communication, outweigh the development and maintenance costs.

Resource Allocation:

Resources allocated for development, hosting, and maintenance are reasonable and justifiable.

Return on Investment:

The project is expected to yield positive returns by improving user satisfaction and fostering relationships.

Long-Term Viability:

The system's sustainability is ensured by its cost-effectiveness and potential for future enhancements.

2.4 EXISITING SYSTEM:

Manual Process:

Sending wishes manually via email involves significant time and effort, leading to delays and oversight.

Limited Automation:

Existing systems may lack automation features, making it challenging to schedule and send wishes efficiently.

Risk of Human Error:

Reliance on manual processes increases the risk of errors, such as forgetting to send wishes on time.

2.5 PROPOSED SYSTEM:

Automated Process:

The proposed system automates the process of sending wishes, eliminating manual intervention and reducing the risk of errors.

Enhanced Efficiency:

Users can schedule wishes in advance, ensuring timely delivery and improving overall efficiency.

Improved User Experience:

The intuitive interface and seamless functionalities enhance user experience, leading to higher satisfaction levels.

Cost Savings:

Automation reduces the time and effort required to send wishes, resulting in cost savings over time.

HARDWARE REQUIREMENTS:

System : AMD PRO A4-430B R4, 5 computer cores 2C+3G 2.50 GHZ

Hard Disk: 256GB

Ram: 4GB

Internet Connection: Yes

Display: LCD 16:9 (1366X768 Pixels HD)

SOFTWARE REQUIREMENTS

Operating system: windows

Front End: HTML, CSS

Back End: Python , Flask Schedule, VS Code.

SOFTWARE DESCRIPTION:

Python:

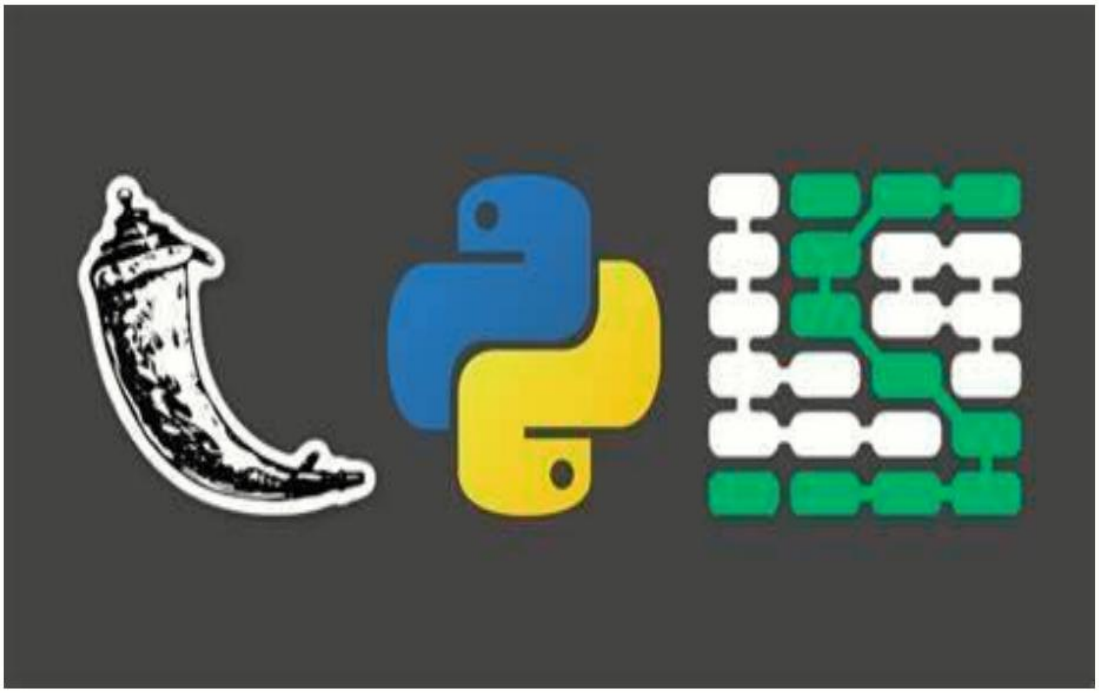
Python stands as a cornerstone in the realm of programming languages, revered for its versatility and robustness across a multitude of applications. With its expansive library ecosystem and dynamic typing, Python caters to a wide spectrum of needs, from web development to scientific computing and artificial intelligence. Its hallmark readability and conciseness foster a collaborative coding environment, where developers can easily comprehend and build upon each other's work.

Python's scalability further bolsters its appeal, accommodating projects ranging from small scripts to enterprise-grade software solutions. Embraced by a thriving community of developers and enthusiasts, Python continues to evolve as a go-to language for innovation and problem-solving across industries worldwide.

Flask:

Flask emerges as a beacon of simplicity and flexibility in the realm of web development, offering developers a lightweight yet powerful framework for crafting dynamic and scalable web applications. Its minimalist design and unobtrusive architecture provide a refreshing departure from the complexities of traditional frameworks, enabling rapid prototyping and iterative development. Flask's modular structure empowers developers to cherry-pick the components and extensions that align with their project requirements, facilitating a bespoke approach to web

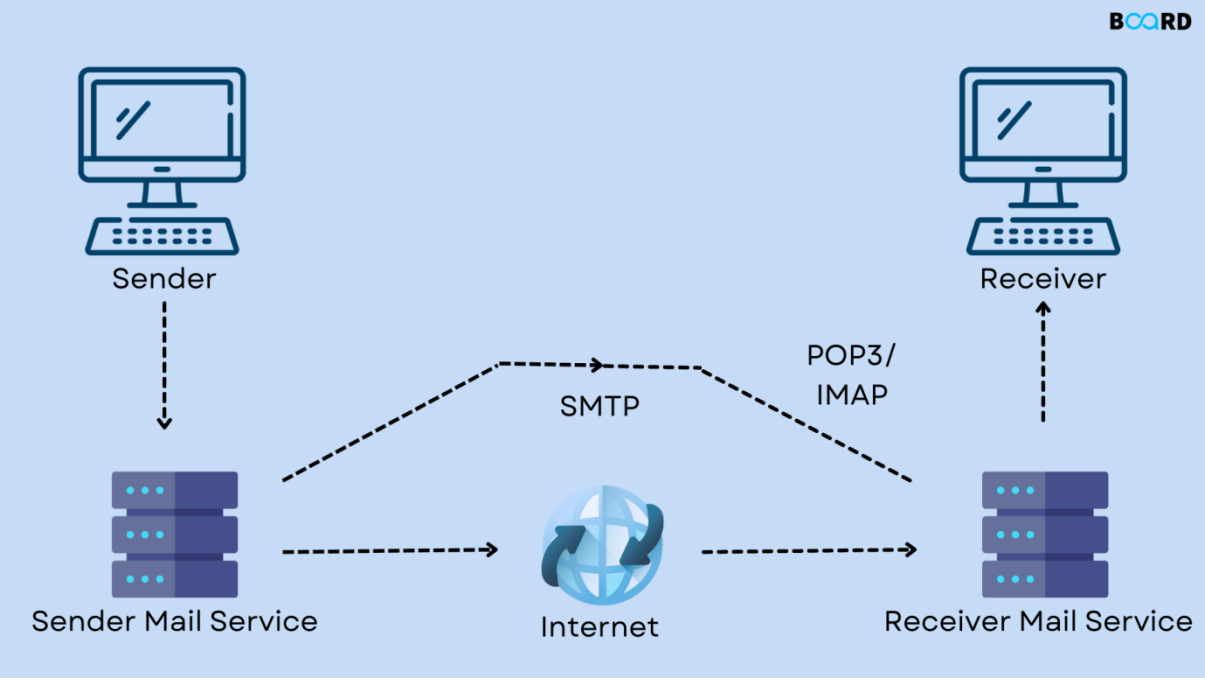
application development. Leveraging a vibrant ecosystem of extensions and plugins, Flask enables seamless integration of additional features and functionalities, from authentication and authorization to database management and beyond. Whether embarking on a solo project or collaborating within a team, Flask empowers developers to bring their web development visions to life with elegance and efficiency.



2.5.1 Flask Diagram

SMTP:

SMTP, or Simple Mail Transfer Protocol, serves as the backbone of electronic communication, facilitating the seamless exchange of email messages across the vast expanse of the internet. Its robust architecture and standardized protocols ensure the reliable transmission of emails between mail servers and clients, underpinning the global ecosystem of electronic correspondence. With mechanisms for error detection and recovery, SMTP minimizes the risk of message loss or delivery delays, maintaining the integrity and efficiency of email communications. Moreover, SMTP's support for authentication and encryption mechanisms, such as SSL/TLS, bolsters the security of sensitive information transmitted via email, safeguarding against unauthorized access and interception. Embraced by email service providers and organizations worldwide, SMTP continues to play an indispensable role in fostering efficient and secure electronic messaging for individuals and businesses alike.

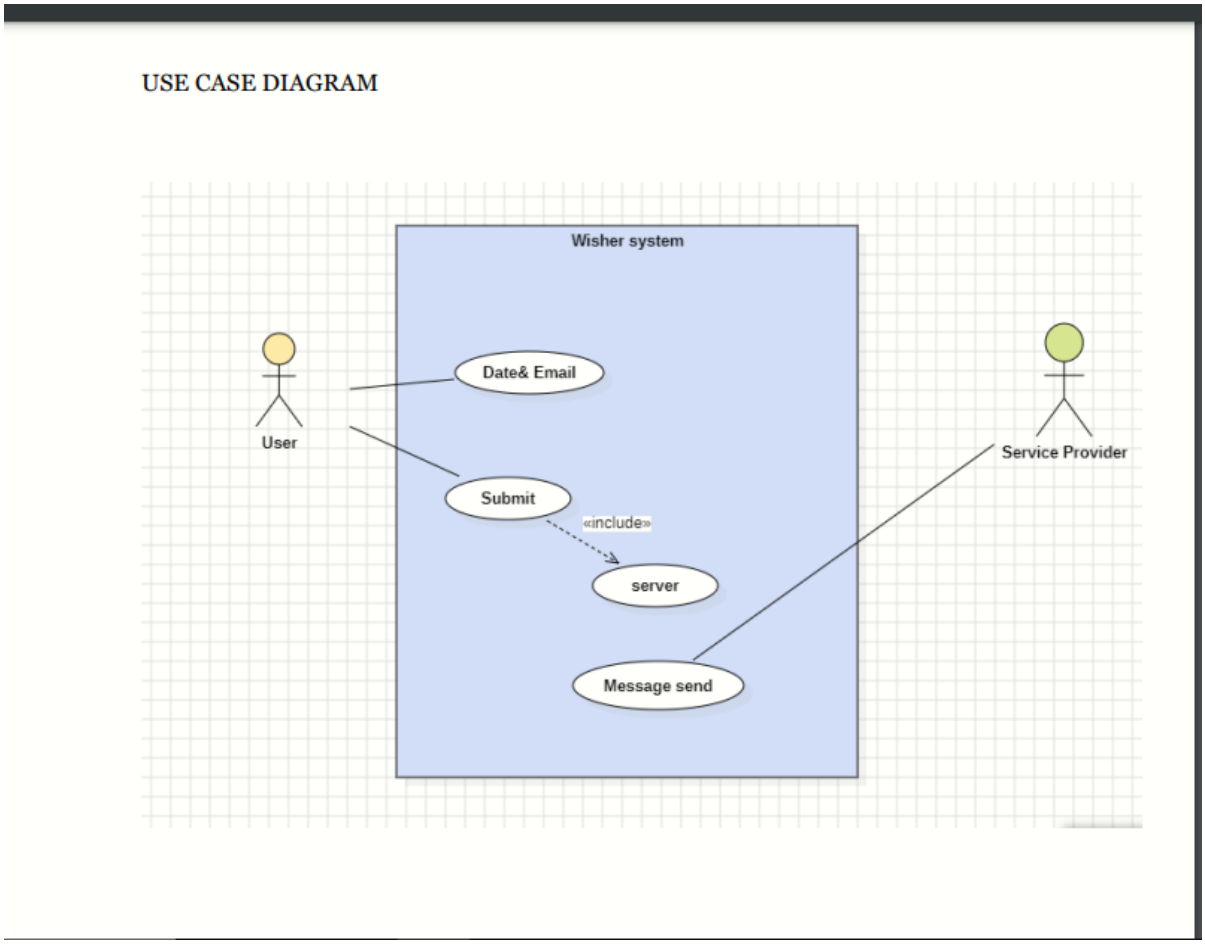


2.5.2 SMTP Diagram

VS Code:

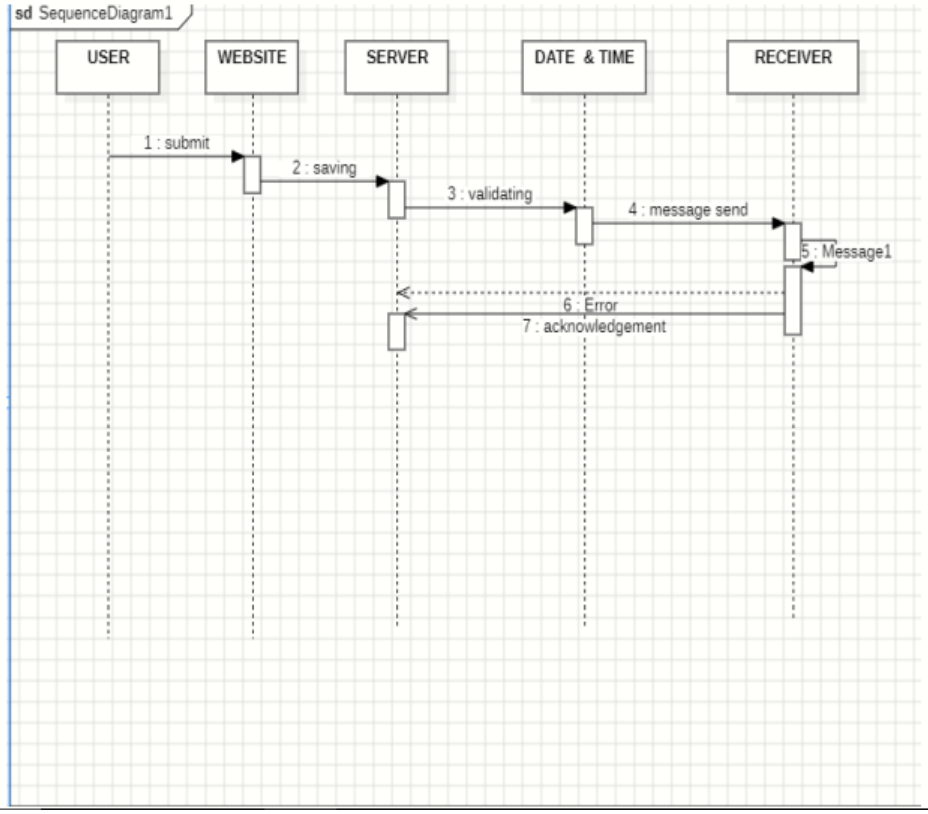
Visual Studio Code (VS Code), a popular code editor developed by Microsoft, offers developers a versatile and feature-rich environment for writing, debugging, and managing Python scripts. With its intuitive user interface and extensive plugin ecosystem, VS Code provides developers with powerful tools for enhancing their Python development experience. Features such as IntelliSense, which offers intelligent code completion and syntax highlighting, streamline the coding process and improve productivity. Additionally, VS Code's built-in Git integration simplifies version control, allowing developers to collaborate seamlessly on Python projects. With support for debugging, unit testing, and task running, VS Code empowers developers to build and deploy Python scheduling applications with confidence and efficiency

SYSTEM OVERVIEW:



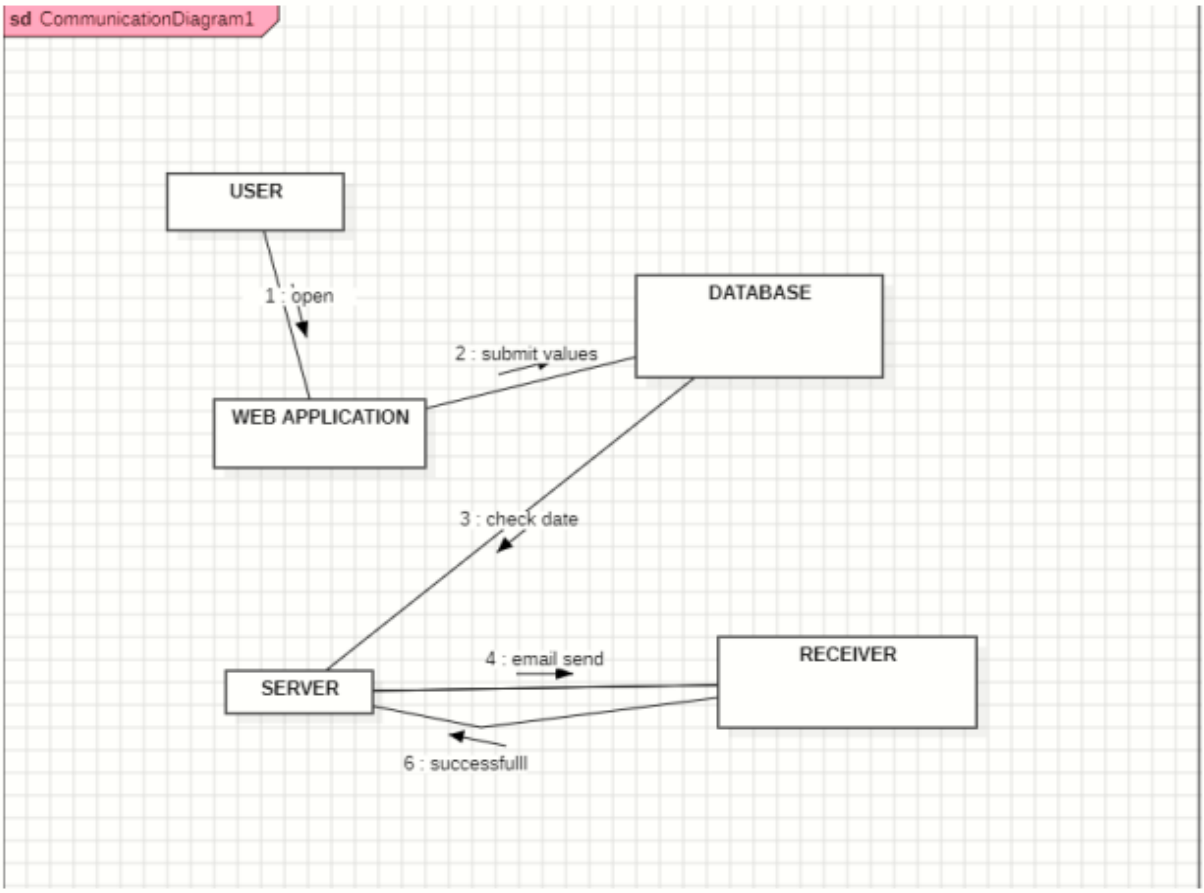
2.5.3 Use Case Diagram

SEQUENCE DIAGRAM



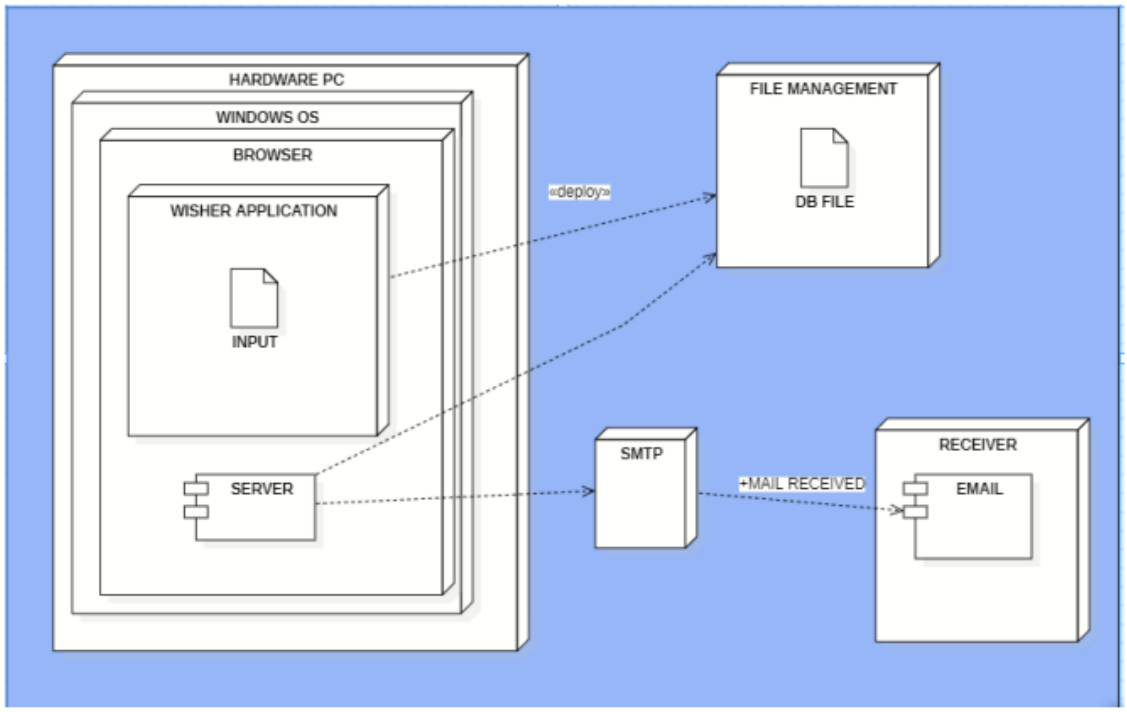
2.5.4 Sequence Diagram

COMMUNICATION DIAGRAM:



2.5.5 Communication Diagram

DEPLOYMENT DIAGRAM:



2.5.6 Deployment Diagram

CHAPTER 3

SYSTEM TESTING

System testing is an integral part of ensuring the quality and reliability of the Automated Wish Sender project. This phase of testing encompasses several key components aimed at validating the functionality, security, and performance of the entire software system. Here's how each component applies to the project:

3.1 Unit Testing:

Unit testing focuses on verifying individual components of the Automated Wish Sender, such as the email sending functionality, input validation, and scheduling mechanism. Developers write test cases to assess each unit's functionality in isolation, ensuring that

they perform as expected and meet the specified requirements.

3.2 Integration Testing:

Integration testing evaluates the interactions and interfaces between different modules of the Automated Wish Sender. Test cases are designed to verify the integration points between components, such as the interface with Flask for web rendering and Flask-Mail for email sending. This testing ensures that all components work together seamlessly and communicate effectively to achieve the desired functionality.

3.3 System Testing:

22 System testing involves testing the entire Automated Wish Sender system as a cohesive unit. Test cases cover various scenarios, including user interactions, scheduling wishes, and email dispatch, to validate the end-to-end functionality of the system. System testing ensures

that the Automated Wish Sender meets user requirements and performs reliably under different conditions.

3.4 Security Testing:

Security testing assesses the Automated Wish Sender's resilience against potential security threats and vulnerabilities. Test cases simulate attacks, such as SQL injection or cross-site scripting, to identify and mitigate security risks. Security testing ensures that user data is protected, and the system maintains confidentiality, integrity, and availability.

3.5Performance Testing:

Performance testing evaluates the Automated Wish Sender's responsiveness and scalability under various load conditions. Test scenarios include stress testing, load testing, and endurance testing to assess the system's performance under peak usage.Performance testing ensures that the system can handle multiple concurrent users and scheduled wishes without degradation in performance.

By conducting thorough system testing across these key components, the Automated

Wish Sender project aims to deliver a high-quality, reliable, and secure solution for .
automating wish sending. This testing phase ensures that the system meets user .

expectations, performs as intended, and provides a positive user experience.

CHAPTER 4

SOURCE CODE

4.1.1 HOME.HTML

```
<!DOCTYPE html>
<html>
<head>
<style type="text/css">
body{
background-size: cover;
background-attachment: fixed;
background-repeat: repeat;
background: rgb(56,222,232);
background: linear-gradient(90deg, rgba(56,222,232,1) 22%,
rgba(128,30,143,0.8575805322128851) 63%, rgba(48,48,215,1) 74%);
}
.details{
color: white;
margin: 1rem;

display: flex;
```

flex

-direction: column;

align

-items: center;

justify

-content: center;

max

-width: 80rem;

font

-size: 30px;

}

.explain{

margin: 1rem;

display: flex;

flex

-direction: column;

align

-items: center;

justify

-content: center;

max

-width: 60rem;

font

```
-size: large;
color: white;
}
h1{
color: blue;
position: static;
background
-color: transparent;
}
.name,.email,.db,.op{
font
-size: medium;
width: 75%;
max
-width: 20rem;
padding: 0.5rem;
margin: 0.5rem;
color: white;
border: none;
border-radius: 0.75rem;
padding: 0.75rem 3rem;
background-color: rgb(50, 50, 50);
}
```

```
button{
font-size: 20px;
background-color:rgb(80, 218, 223);
border-radius: 5px;
width: 40%;
padding:3px;
}
.heading{
width: 100%;
height: 10vh;
background-color: aqua;
}
</style>
<title>WISHER</title>
</head>
<body >
<div class="heading">
<h1 id="hea" style=" font-style: bold; font-size: 60px"
align="center">Automatic- Wisher</h1>
</div>
<marquee style="font-size: 18px; color:white; background-color:
red;">Our
project caters to individuals seeking to stay connected with loved ones,
```

colleagues, and acquaintances by offering a convenient and efficient solution for

sending wishes on important dates. Whether it's birthdays, anniversaries, or

special

milestones, our platform empowers users to spread joy and positivity

effortlessly,

enhancing relationships and fostering meaningful connections in the digital

age.</marquee>

** **

<form class="details" method="post" action="/getdata">

Enter the Name:<input type="text" placeholder="Name" name="name"

class="name">

Enter the DOB(MMDD):<input type="text" name="DOB" placeholder="Date of

birth" class="db">

Enter the Email:<input type="Email" name="email" placeholder="Email"

class="email">

Enter Your Name:<input type="text" name="send" placeholder="Your name"

class="email">

Enter the Event:<select class="op" name="event"><option

value="Birthday">Birthday</option>

<option value="Anniversary">Anniversay</option></select>

<button value="Submit" type="Submit" >Submit</button>

</form>

** **

</body>

</html>

4.1.2 INDEX.HTML

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8">
<meta name="viewport" content="width=device-width, initial-scale=1.0">
<title>Successfull!</title>
<style>
body{
background-color: darkslateblue;
background-repeat: no-repeat;
}
h1{
font-size: 50px;
font-style: bold;
color: aqua;
}
a{
color: red;
font-size: 27px;
margin: top 4px;
```

```
justify-content: center;

align-content: center;

flex-direction: column;

flex-wrap: wrap;

}

p{

text-align: center;
color: black;

ont-size: 30px;

}

</style>

</head>

<body>

<div class="ex" align="center">

<h1>Successfully added!</h1>

<p>The Date has been added to the database..!</p>

<a href="/home">Added Another!</a>

</div>

</body>

</html>
```

4.2 PYTHON CODE

APP.PY

```
#Project: Automatic Anniversary and Birthday wishing using Email.
#Team Members: k.sathish, Ashok kumar, Krishna.M
import time
from flask import Flask, render_template, request #flask is used for integrate a
html and python code
app = Flask(__name__) #Initalize the flask.
@app.route('/')
def entry():
return render_template('home.html')
@app.route('/home')
def home():
return render_template('home.html')
@app.route('/getdata', methods=['POST','GET'])
def submit():
name= request.form['name']
db = request.form['DOB']
event = request.form['event']
email = request.form['email']
```

```
sender = request.form['send']

x = name # This is used for store the data in the file

y = db

z = event

v=email

s =sender

f = open("member.txt","a+")

f.write(y+' ')

f.write(z+' ')

f.write(x+' ')

f.write(v+' ')

f.write(s+' ')

f.write('\n')

f.close()

print("value writted")

return render_template('index.html',)

if __name__=="__main__":

app.run(debug=True)

#The server can be run 24/7 to check the birthday and anniversary date for

wishing.

#server.py is used for that work.
```

4.2.2 SERVER.PY

```
import time
import schedule
import smtplib
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
from email.mime.base import MIMEBase
from email import encoders
def picture(subject):
if subject=="Birthday":
attachment_path = "C:\\Users\\SATHISH
K\\Documents\\sem2\\samplepython\\templates\\birthday.png"
else:
attachment_path = "C:\\Users\\SATHISH
K\\Documents\\sem2\\Pytproject\\templates\\Designer.png"
return attachment_path
def send_email(message, recipient_email, subject, body, attachment_path):
sender_email = "sathishksv2003@gmail.com"
recipient_email=str(recipient_email)
subject =str(subject)
message =str(message)
try:
```

```
msg = MIMEMultipart()
msg['From'] = sender_email
msg['To'] = recipient_email
msg['Subject'] = subject
msg.attach(MIMEText(body, 'plain'))
# Attach the file
with open(attachment_path, 'rb') as attachment:
part = MIMEBase('application', 'octet-stream')
part.set_payload(attachment.read())
encoders.encode_base64(part)

part.add_header('Content-Disposition', f'attachment;
filename="{attachment_path}"')
msg.attach(part)
# Set up the SMTP server
smtp_username = "sathishksv2003@gmail.com"
# Connect to the server and send the email
with smtplib.SMTP("smtp.gmail.com",587) as server:
server.starttls()
server.login(smtp_username,"zczdmmddvfslrihe")
server.sendmail(sender_email, recipient_email, msg.as_string())
print("email send succesfully")
except:
print("Unable to send!")
```

```
def ctb():

f=open("member.txt",'r')

today = time.strftime('%m%d')

flag =0

for line in f:

if today in line:

line = line.split(' ')

flag=1

message="hello"

subject=line[1]

recipient_email =line[3]

attachment_path=picture(subject)

body = "Happy "+line[1].strip()+" my dear "+line[2].strip()+" I wish you
all

the very best on this special day. May you be blessed today, tomorrow, and
in the

upcoming days to come. May you have a wonderful birthday and many
more to

come. " "\n Greated By \n\n\n "+line[4].strip()
send_email(message, recipient_email, subject, body, attachment_path)

if flag ==0:

print("no_events")

message=None

return message

schedule_time="11:57"
```

schedule.every().day.at(schedule_time).do(ctb)

if __name__ == "__main__":

while True:

schedule.run_pending()

time.sleep(2)

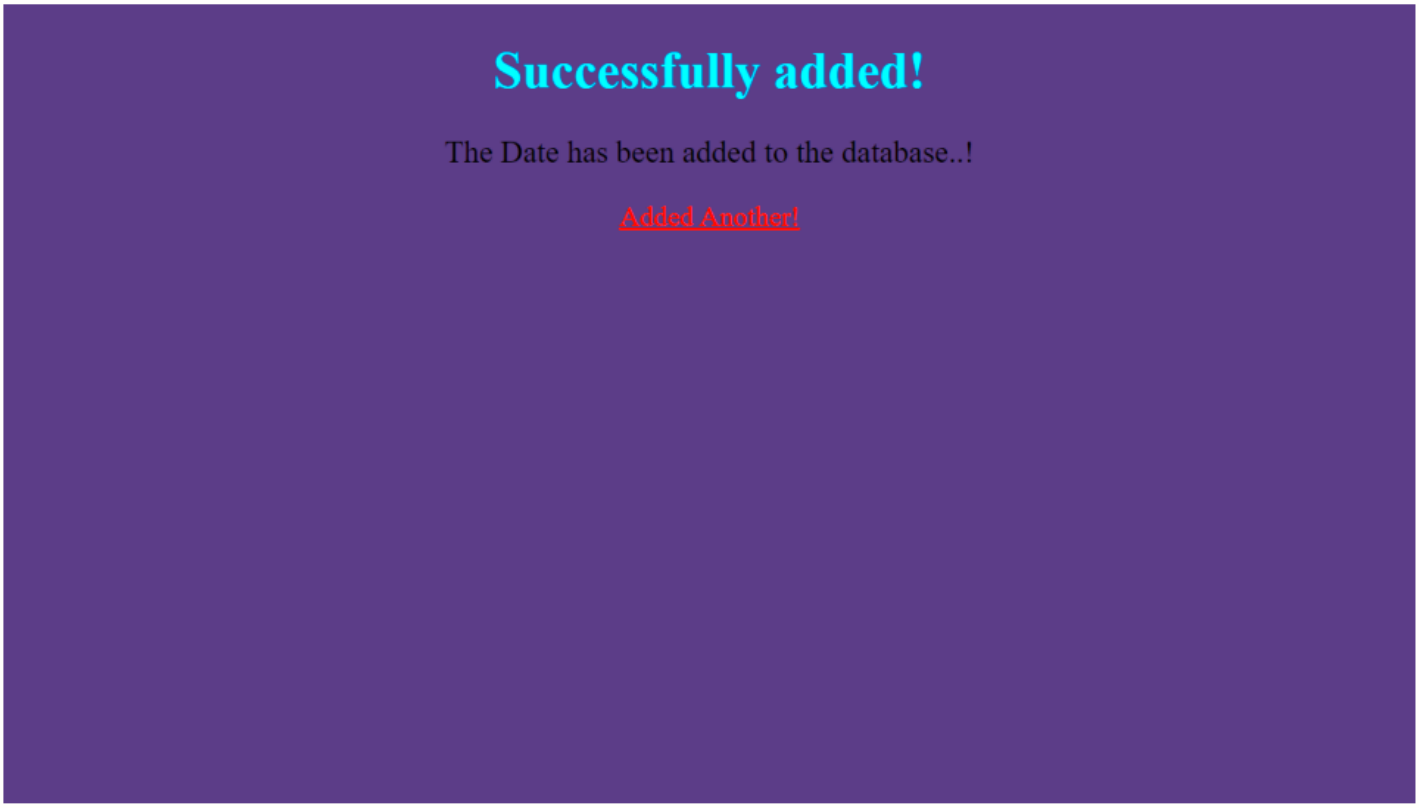
OUTPUT:

SCREENSHOTS:

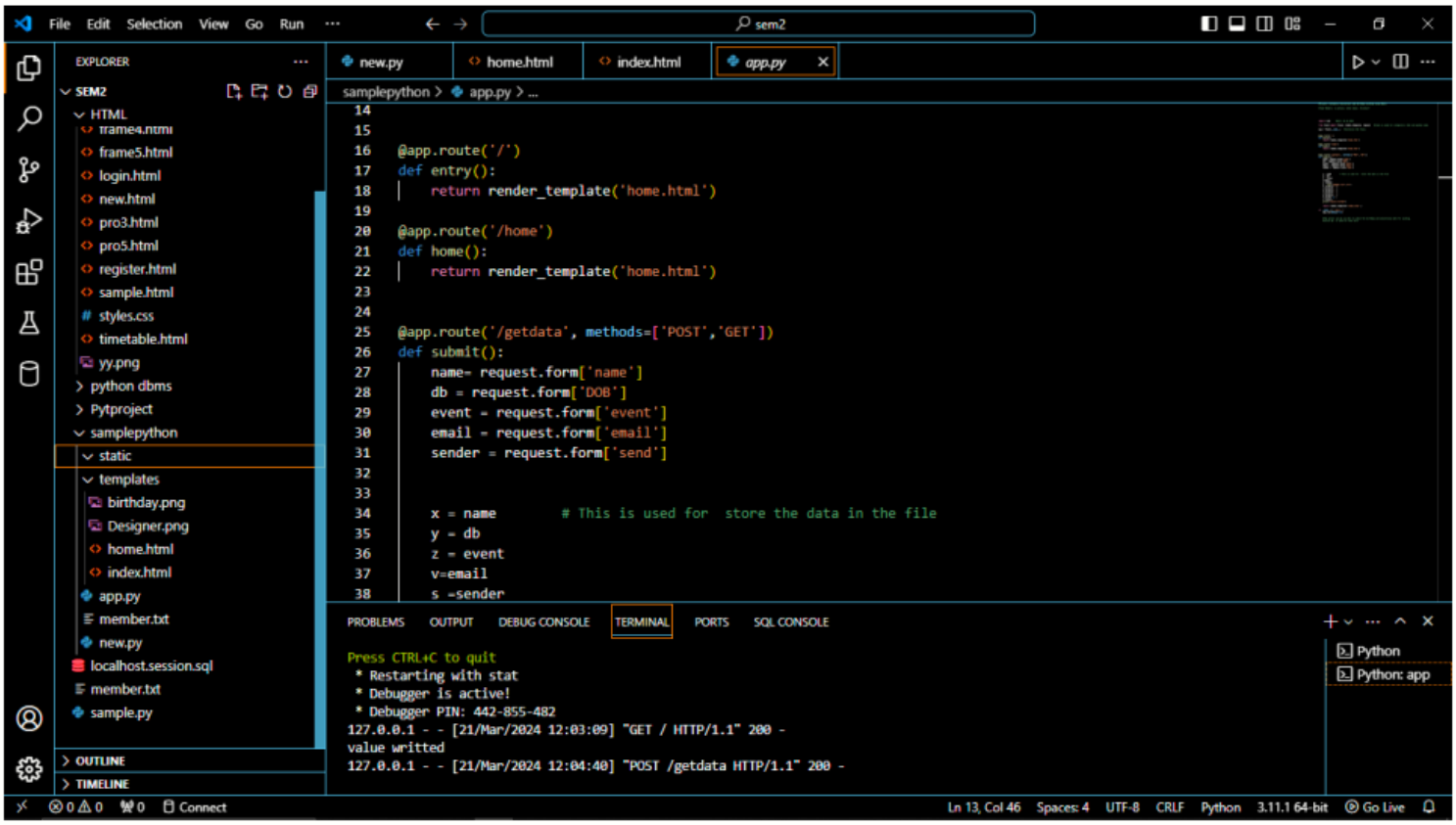
Interface the website:

Entering values for submission:

The value is submitted successfully.

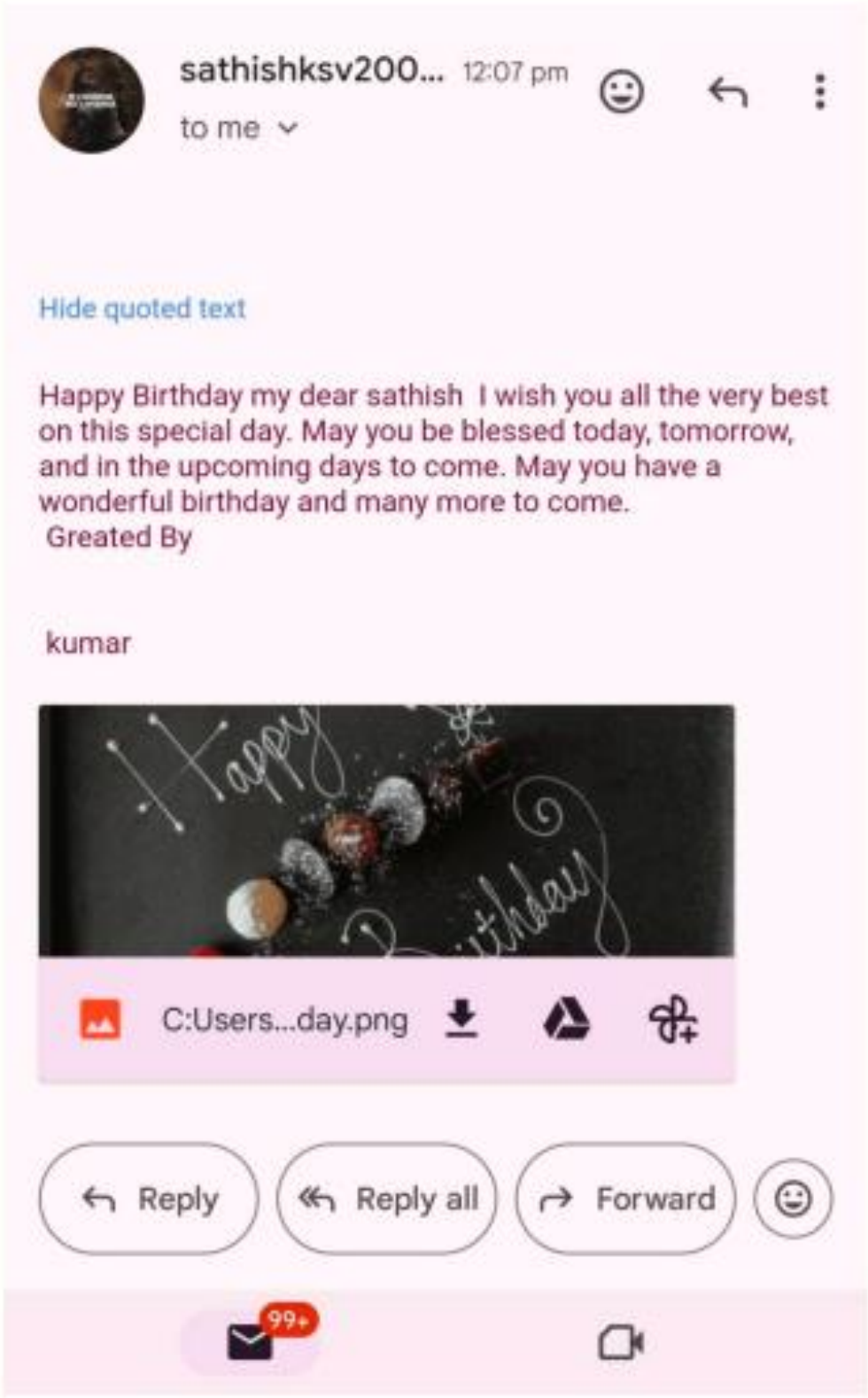


The value is been successfully stored in the file.



The email has been send to corresponding date.

The message received from the receiver side:


sathishksv200... 12:07 pm
to me
Hide quoted text
Happy Birthday my dear sathish I wish you all the very best on this special day. May you be blessed today, tomorrow, and in the upcoming days to come. May you have a wonderful birthday and many more to come.
Greated By
kumar
C:Users...day.png
Reply
Reply all
Forward
99+

FUTURE SCOPE AND ENHANCEMENT:

To further enhance the functionality and versatility of the Automated Wish Sender project, we propose the implementation of database connectivity and the addition of support for other events beyond birthdays and anniversaries. This enhancement will enable users to manage a wider range of occasions and personalize their wishes more effectively

Database Connectivity:

Integrate a relational database management system (e.g., SQLite, MySQL, PostgreSQL) into the Automated Wish Sender.

Establish a database schema to store user information, scheduled wishes, event details, and any other relevant data.

Implement database connectivity using an ORM (Object-Relational Mapping) framework like SQLAlchemy to interact with the database seamlessly.

Enable features such as user authentication, user profiles, and persistent storage of scheduled wishes for each user.

Support for Other Events:

Expand the list of supported events beyond birthdays and anniversaries to include holidays, festivals, milestones, and personal achievements.

Allow users to create custom events and define their own categories, event names, and associated dates.

Provide a user-friendly interface for managing events, including adding, editing, and deleting events as needed.

Implement advanced scheduling options, such as recurring events (e.g., weekly, monthly) and reminders for upcoming events.

Enhance the message customization feature to include event-specific templates and personalized content tailored to each occasion.

Benefits of the Enhancement:

Improved User Experience:

Database connectivity enables users to manage their wishes and events more efficiently, with features like user authentication and persistent storage enhancing usability. Enhanced Flexibility: Supporting a wider range of events allows users to celebrate diverse occasions and milestones, making the Automated Wish Sender more versatile and appealing to a broader audience. Increased Personalization: Customizable event management and message templates empower users to craft personalized wishes tailored to each recipient and occasion, fostering deeper connections and meaningful interactions.

Scalability:

By leveraging a database-driven architecture, the project becomes more scalable and adaptable to future growth and feature expansions, accommodating evolving user needs and requirements.

Overall, implementing database connectivity and adding support for other events represents a significant enhancement to the Automated Wish Sender project, enriching its functionality, usability, and appeal to users. This enhancement aligns with the project's goal of simplifying wish sending and fostering meaningful connections, while also laying the foundation for future growth and innovation

CONCLUSION:

In conclusion, the Automated Wish Sender project represents a significant step forward in simplifying and enhancing the process of sending heartfelt wishes and greetings. Through the seamless integration of Flask web framework, Python scripting, and SMTP email functionality, the project offers users a convenient and efficient platform for scheduling and automating wish dispatches on special occasions. Throughout the development process, we have focused on delivering a solution that prioritizes user experience, reliability, and scalability. The implementation of system testing, including unit testing, integration testing, and performance testing, has ensured the project's robustness and adherence to quality standards. Additionally, the incorporation of security measures safeguards user data and communications, fostering trust and confidence among users. Looking ahead, the project has immense potential for further enhancements and expansions. The proposed future enhancement of implementing database connectivity and adding support for additional events will elevate the project's functionality, flexibility, and personalization options, catering to a broader range of user needs and preferences. Ultimately, the Automated Wish Sender project aims to bring joy, warmth, and connectivity to users' lives by facilitating meaningful interactions and celebrations. As we continue to refine and evolve the project, we remain committed to delivering a high-quality, user-centric solution that enriches the lives of our users and strengthens relationships across distances. With gratitude for the journey thus far, we look forward to the continued success and impact of the Automated Wish Sender in the lives of our users.

BIBLIOGRAPHY

SMTP :SMTP response code list and commands (mailslurp.com)

Simple Mail Transfer Protocol (SMTP) - GeeksforGeeks

FLASK: 7 Python Flask Project Ideas for Beginners – Pythonista Plane