TODO-LIST

PROJECT REPORT

SUBMITTED BY
SHESHU AKULA(MSS2018061)

GRADUATING IN
UNIVERSITY COLLEGE OF ENGINEERING, OSMANIA UNIVERSITY

SUBMITTED TO
MEDHA CHARITABLE TRUST

ACKNOWLEDGEMENT

I want to express my sincere gratitude to everyone who played a role in the development of this project. This endeavor wouldn't have been possible without the support and assistance from various individuals and resources.

First and foremost, I extend my heartfelt thanks to Praneeth Sir for providing invaluable guidance and encouragement throughout the project. My gratitude also goes to my friends for their unwavering support, which kept me motivated and helped me overcome obstacles.

I would like to acknowledge the open-source community and online resources that offered valuable tutorials and documentation. The wealth of knowledge available online significantly contributed to the success of the project.

In conclusion, I appreciate everyone who contributed, whether in a big or small way, to the development of the project. Your efforts have been invaluable, and I am thankful for the support and motivation you provided throughout this journey.

ABSTRACT

The project aims to develop a robust ToDo list application using Python, Flask, and SMTP for sending email notifications. This web-based application provides users with a seamless experience for managing their tasks efficiently. With an intuitive user interface, users can easily create, update, and delete tasks, organizing them based on priority and category.

The integration of SMTP ensures that users receive timely email notifications for upcoming tasks, enabling them to stay organized and on top of their commitments while also offering customizable settings for personalization.

Leveraging Flask's flexibility and scalability, along with Python's rich ecosystem of libraries, the project delivers a reliable and user-friendly solution for effective task management.

INDEX

INTRODUCTION

In today's hectic lifestyle, managing tasks effectively is essential for productivity and success. To address this need, we present a ToDo list application designed to simplify task management.

Developed using Python, Flask, HTML, CSS, and MySQL, this application offers a user-friendly interface for creating and organizing tasks. With features such as deadline tracking, users can stay organized and focused on their goals. The integration of MySQL ensures data integrity and reliability, while Flask facilitates seamless web development.

This project report provides insights into the implementation process, technologies employed, and the significance of the ToDo list application in enhancing productivity.

TECHNOLOGIES USED

Python: Primary programming language for backend development.

Flask: Web framework for building the application.

HTML: Markup language for structuring web pages.

CSS: Styling language for customizing the appearance of web pages.

MySQL: Relational database management system for storing task data.

MySQL Connector: Library for connecting Flask application with MySQL database.

SMTP: Protocol for sending email notifications.

Git: Version control system for managing project code.

IMPLEMENTATION

Backend Development: Python and Flask were used to develop the backend logic of the application. Flask provided a lightweight and flexible framework for building web applications, while Python served as the primary programming language for implementing the business logic.

Frontend Design: HTML and CSS were utilized for designing the frontend interface of the application. HTML was used to structure the web pages and define the content, while CSS was used for styling and layout customization.

Database Management: MySQL was chosen as the database management system for storing task data. The MySQL connector library facilitated seamless communication between the Flask application and the MySQL database, enabling efficient data storage and retrieval.

Task Management Features: The application offered a range of task management features, including task creation, deletion and deadline

tracking. Users could easily add new tasks and mark tasks as completed.

Email Notifications: Integration with SMTP allowed the application to send email notifications for upcoming tasks and reminders. Users could configure their notification preferences and receive timely reminders via email.

Background Scheduler: Integration of background scheduler allows the application to send mails and run program in background so that it can send mails even in background.

Implementation of python code with flask:

```python
def execute_query(query, fetch=False, fetchall=False):
    connection = connect_to_mysql()
    cursor = connection.cursor()
    try:
        cursor.execute(query)
        if query.strip().lower().startswith("select"):
            result = None
            if fe (parameter) fetch: bool
                r
            elif fetch:
                result = cursor.fetchone()
            else:
                result = cursor.fetchall()  # Fetch all by default for SELECT queries
        else:  # For INSERT, UPDATE, DELETE
            connection.commit()  # Commit changes for non-SELECT queries
            result = None
    except Exception as e:
        connection.rollback()  # Rollback changes if an exception occurs
        print(f"Error executing query: {str(e)}")
        result = None
    finally:
        cursor.close()
        connection.close()
    return result
```

```python
@app.route('/', methods=["GET", "POST"])
def home():
    query = "SELECT DATE_FORMAT(CREATED, '%d %b %Y'),TASK,DATE_FORMAT(DEADLINE,'%Y-%m-%d'),COMPLETED,EMAIL,EMAIL_SENT FROM TODO ORDER BY DEADLINE"
    data = execute_query(query, fetchall=True)
    d=defaultdict(list)
    for i in data:
        if not is_valid_date(i[2]):
            query = f"DELETE FROM TODO WHERE DEADLINE = '{i[2]}'"
            execute_query(query)
    data=execute_query(query,fetchall=True)
    time=datetime.now().strftime("%H:%M")
    for i in data:
        l=""
        if (datetime.strptime(i[2], '%Y-%m-%d')-datetime.today()).days==4 and i[3]==0 and i[5]==0:
            query=f"UPDATE TODO SET EMAIL_SENT=1 where (TASK='{i[1]}' and ((DATEDIFF('{i[2]}',SYSDATE())-1)=4 or (DATEDIFF('{i[2]}',SYSDATE())-2)=4))";
            execute_query(query,fetchall=True)
            l+=i[1]+"->"+i[2]+"\n"
            d[i[4]].append(l)
            print(l)
    print(d)
    c=0
    try:
        for i,j in d.items():
            j=",".join(j)
            if j!='':
                send_email("Tasks to be completed in 4 days",i,j)
                c+=1
    except:
        print("Error in sending mail")
    print(f"Emails sent : {c}")

    query = "SELECT DATE_FORMAT(CREATED, '%d %b %Y'),TASK,DATE_FORMAT(DEADLINE,'%d %b %Y'),COMPLETED FROM TODO ORDER BY DEADLINE"
    data = execute_query(query, fetchall=True)
    return render_template("index.html" ,data=data)
```

```python
@app.route('/add', methods=["POST"])
def add():
    task = request.form.get('task')
    deadline = request.form.get('deadline')
    mail=request.form.get('mail')
    current_date_time = datetime.now()
    current_date = current_date_time.date()
    if is_valid_date(deadline):
        query = f"INSERT INTO TODO (CREATED, TASK, DEADLINE,EMAIL) VALUES ('{current_date}', '{task}', '{deadline}','{mail}')"
        execute_query(query)
    return redirect(url_for('home'))

@app.route('/update/<task>', methods=["POST"])
def update(task):
    # Use the received value in your SQL query to update the database
    query = f"UPDATE TODO SET COMPLETED = CASE WHEN COMPLETED = 0 THEN 1 ELSE 0 END WHERE TASK = '{task}'"
    execute_query(query)
    return redirect(url_for('home'))

@app.route('/delete/<task>', methods=["GET","POST"])
def delete(task):
    query=f"DELETE FROM TODO WHERE TASK='{task}'"
    execute_query(query)
    return redirect(url_for('home'))


def is_valid_date(date_str, format='%Y-%m-%d'):
    try:
        current_date = datetime.now().date()
        if date_str >= str(current_date):
            datetime.strptime(date_str, format)
            return True
    except ValueError:
        return False
```

Implementation of send_email.py code:

```python
import os

from dotenv import load_dotenv  # pip install python-dotenv
load_dotenv()


import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

# SMTP2GO server settings
smtp_server = 'mail.smtp2go.com'
smtp_port = 2525 # SMTP2GO port number
smtp_username = os.getenv("USERNAME_SMTP")
smtp_password = os.getenv("PASSWORD_SMTP")

# Sender and recipient email addresses
sender_email = os.getenv("EMAIL_NEW")
```

```
# Email content
def send_email(subject, receiver_email, l):
    body = f'''\
        <html>
        <body>
            <p>Hello there,</p>
            <p>I hope you are well.</p>
            <p>I just wanted to drop you a quick note to remind you that the following tasks are pending which are to be completed by <strong>next 4 days</strong> .
            <br>
            <p><strong>{l}</strong></p>
            <br>
            <p>Best regards</p>
        </body>
        </html>
        '''

    # Create MIMEText object
    msg = MIMEMultipart()
    msg['From'] = sender_email
    msg['To'] = receiver_email
    msg['Subject'] = subject
    msg.attach(MIMEText(body, 'html'))

    # Connect to SMTP server and send email
    try:
        server = smtplib.SMTP(smtp_server, smtp_port)
        server.starttls()
        server.login(smtp_username, smtp_password)
        text = msg.as_string()
        server.sendmail(sender_email, receiver_email, text)
        print('Email sent successfully!')
    except Exception as e:
        print('Error sending email:', str(e))
    finally:
        server.quit()
```

```
if __name__ == "__main__":
    print(sender_email,smtp_username,smtp_password)
    send_email(
        subject="Tasks to be done in 4 days",
        # name="John Doe",
        receiver_email="akulasheshu4@gmail.com",
        # due_date="11, Aug 2022",
        # invoice_no="INV-21-12-009",
        # amount="5",
        l='HIII sfuhsdufbieuhfu'
    )
```

Github Link:
https://github.com/Shesh009/TODO-Flask

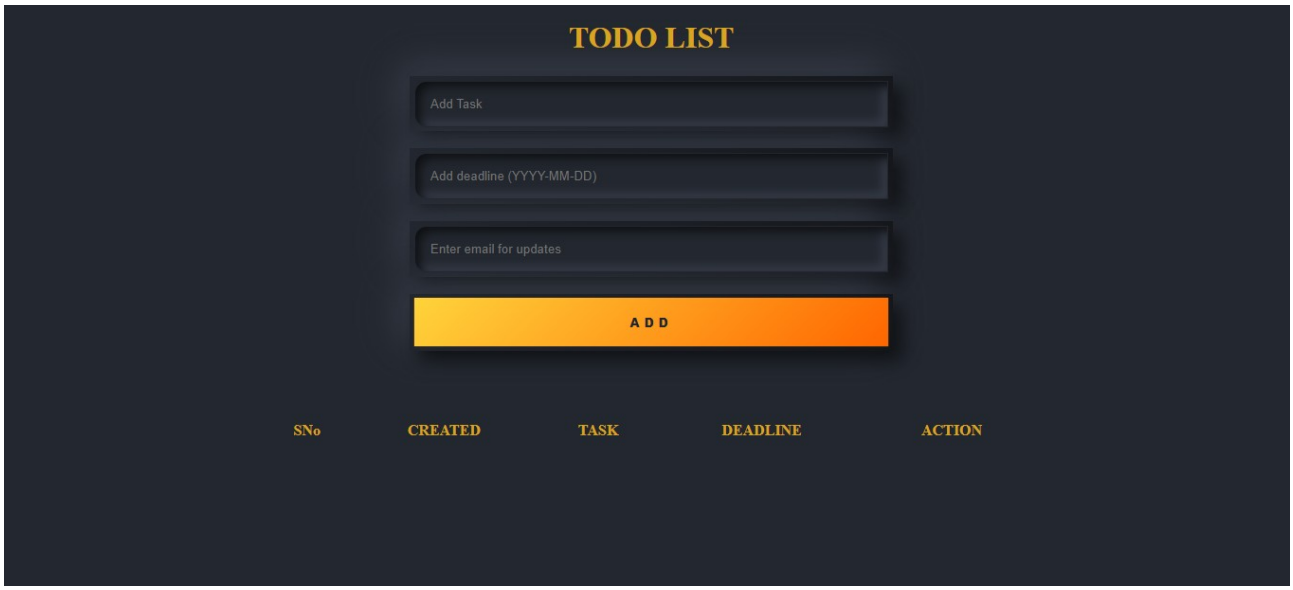
Implementation of HTML code along with inbuilt CSS:

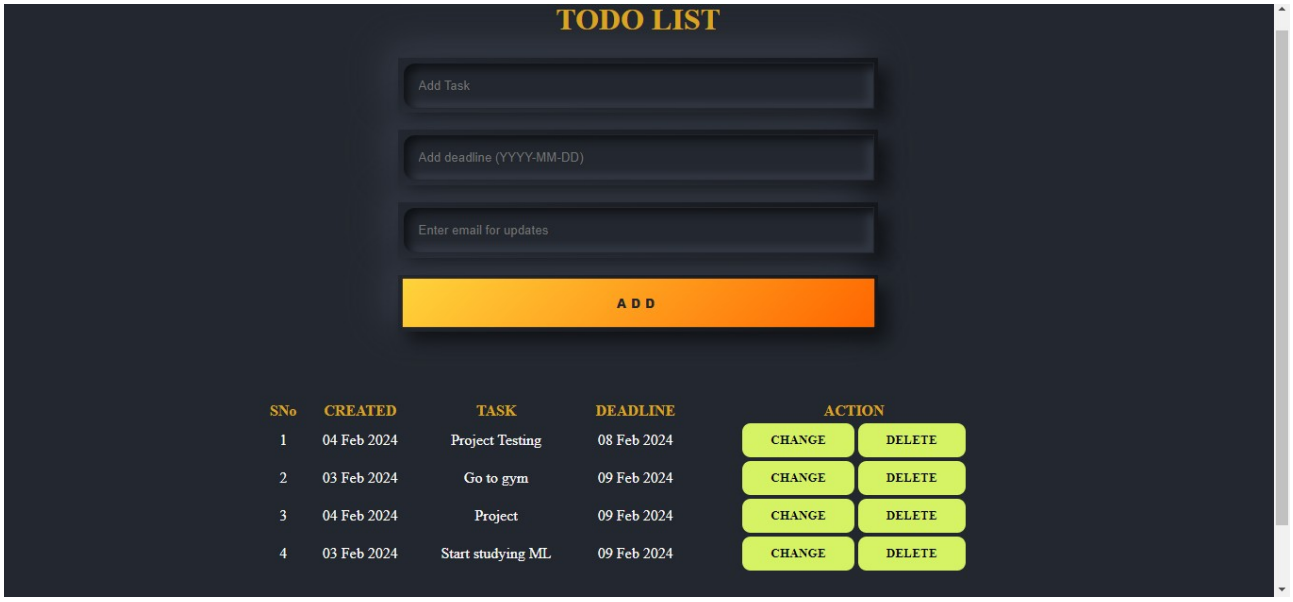
```
<body>
    <form action="/add" method="post" name="add_form">
        <div class="container">
            <h1 style="text-align: center; color: goldenrod;">TODO LIST</h1>
        </div>
        <div class="input-container">
            <input type="text" name="task" id="task" placeholder="Add Task" required>
        </div>
        <br>
        <div class="input-container">
            <input type="text" name="deadline" id="deadline" placeholder="Add deadline (YYYY-MM-DD)" required>
        </div>
        <br>
        <div class="input-container">
          <input type="email" name="mail" id="mail" placeholder="Enter email for updates">
      </div>
        </div>
        <br>
        <div class="input-container">
            <button type="submit">ADD</button>

        </div>
    </form>
    {% if error %}
    <p style="color: red;">{{ error }}</p>
    {% endif %}
    <br><br><br>
```

```
                    <td>
                        <div align="center">
                            <form action="/update/{{ row[1] }}" method="POST" style="display: inline;">
                                <button id="bottone1"><strong>CHANGE</strong></button>
                            </form>
                            <form action="/delete/{{ row[1] }}" method="POST" style="display: inline;">
                                <button id="bottone1"><strong>DELETE</strong></button>
                            </form>
                        </div>
                    </td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
        <br><br><br><br><br>
    </div>
</body>

</html>
```

```
                    <td>
                        <div align="center">
                            <form action="/update/{{ row[1] }}" method="POST" style="display: inline;">
                                <button id="bottone1"><strong>CHANGE</strong></button>
                            </form>
                            <form action="/delete/{{ row[1] }}" method="POST" style="display: inline;">
                                <button id="bottone1"><strong>DELETE</strong></button>
                            </form>
                        </div>
                    </td>
                </tr>
                {% endfor %}
            </tbody>
        </table>
        <br><br><br><br><br>
    </div>
</body>

</html>
```

OUTPUT


TODO LIST
Add Task
Add deadline (YYYY-MM-DD)
Enter email for updates
ADD
SNo
CREATED
TASK
DEADLINE
ACTION


TODO LIST
Add Task
Add deadline (YYYY-MM-DD)
Enter email for updates
ADD
SNo
CREATED
TASK
DEADLINE
ACTION
1
04 Feb 2024
Project Testing
08 Feb 2024
CHANGE
DELETE
2
03 Feb 2024
Go to gym
09 Feb 2024
CHANGE
DELETE
3
04 Feb 2024
Project
09 Feb 2024
CHANGE
DELETE
4
03 Feb 2024
Start studying ML
09 Feb 2024
CHANGE
DELETE

CONCLUSION

In conclusion, the ToDo list application represents a comprehensive solution for managing tasks efficiently, leveraging a stack of robust technologies including Python, Flask, HTML, CSS, and MySQL. By providing users with a user-friendly interface and essential features such as task creation, editing, categorization, and deadline tracking, the application empowers individuals to stay organized and focused on their objectives.

The integration of MySQL ensures seamless data management, while SMTP enhances user engagement through timely email notifications. As we navigate the demands of modern life, tools like this ToDo list application serve as invaluable aids in maximizing productivity and achieving personal and professional goals. Moving forward, continued refinement and expansion of features could further enhance the application's utility and impact on users' lives.