Project 3: Simple Web-Based To-Do List and Calendar View Application Using Python

EE129 - Computer Communication Networks

Aliyah Weiss

Mon & Wed 4:30-5:45 pm

December 18th 2024

Tufts University, ECE School of Engineering

Introduction

The goal of this project is to use Python to create a project of our choosing. I decided to build a web-based to-do list application that integrated user accounts, task management, and a dynamic calendar view, similar to the known Google Calendar. This project requires the use of a broad range of topics from computer communication networks, web development, and software design. The project utilizes Python and the Flask framework, which allows the use of a language and ecosystem known to be reliable, readable, and clear. Users can create an account, manage personalized tasks, and visualize their schedules on a dynamic calendar using HTTP requests that showcase the server-client model and the flow of data on the web. The project also utilizes SQLAlchemy and SQLite for data persistence, Bootstrap for a polished interface, and FullCalendar for the dynamic client-side interactivity. The project combines theories with practice, allowing a functional, user-friendly application to use HTTP and database queries. Sessions secure user data and the logical routes serve dynamic content, while providing a simple UI to allow users to navigate easily. Understanding how to build, secure, and present these web-based tools is a great foundation for more advanced networking applications.

Experimental Setup

To begin, I launched Visual Studio Code to create a project folder on my local computer. In my case, I have already downloaded Python3. A student replicating this project should ensure this version is downloaded. On a Mac with Homebrew, use "`brew install python`". Then run "`python3 –version`" or "`python –version`" in the terminal to confirm a Python 3.X installation is being used. Once this has been confirmed and VSCode is open, I opened my new folder named `todo_app` to hold all of the project files. Inside this folder, I used a virtual environment by running "`python3 -m venv venv`" and activated it with "`source venv/bin/activate`" on Mac. This ensures that all dependencies used in the project such as Flask and Flask-SQLAlchemy are installed in the correct location in this isolated environment. Then I used "`pip install flask flask_sqlalchemy werkzeug`" to get the necessary packages. Since the project uses user authentication, a few additional libraries were also installed. These are noted in the "`requirements.txt`" file and can be downloaded as such.

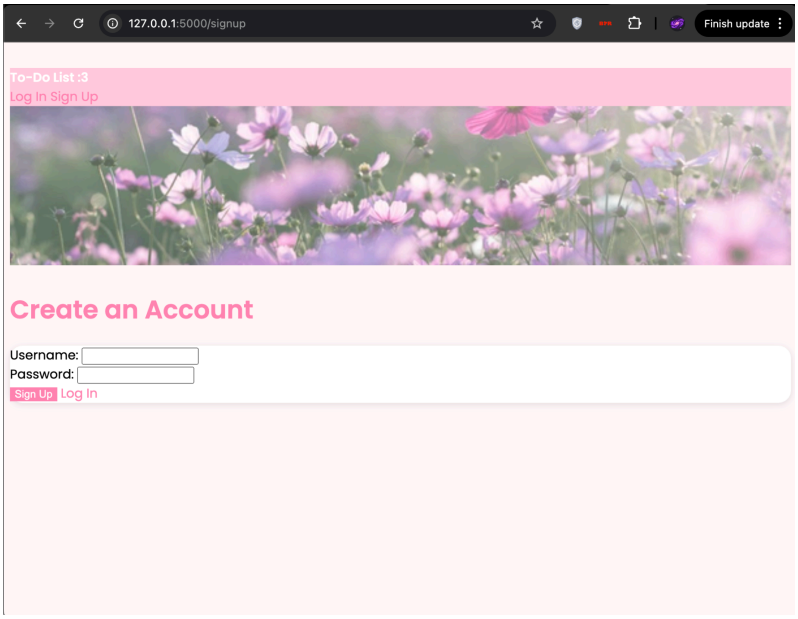
After preparing the environment, I created the essential files: "`app.py`" for the Flask application logic, "`models.py`" for database models (User and Task), "`templates/`" for HTML templates, and "`static/`" for images. I added a "`base.html`" file to templates as a template layout, plus "`index.html, add_task.html, view_task.html, edit_task.html,` and `calendar.html`" for functionality. I also added an image of a cat and an image of flowers to the "`static/images`" directory to use later on for decoration.

Before coding the main functionality, I tested that the Flask server could run. By typing "`python app.py`" in the terminal, I confirmed that the program ran on "`http://127.0.0.1:5000`". I opened this URL in Chrome and it allowed me to see the initial setup and confirm the environment and file structure worked. I then began implementing the required features. First, I added a user signup and login functionality, which required setting up the database with "`db.create_all()`" in the Flask app context and ensuring the User and Task models were all properly defined in "`models.py`". To test this, I registered my own user "`leebee`" with the password "`meow`", and verified that I could login and add unique tasks.

Next, I integrated the dynamic calendar by adding FullCalendar through a CDN link in "`calendar.html`". I tested this by creating tasks with future due dates and then accessing the calendar page to confirm the tasks appeared on the correct calendar dates. This same process applied to verifying styling enhancements when I added pastel colors, Bootstrap components, and a flower banner image. I would also refresh the page to ensure the design changes stayed on a refreshed version.

Results

After implementing the functionality discussed above. The URL performed the desired tasks as follows:



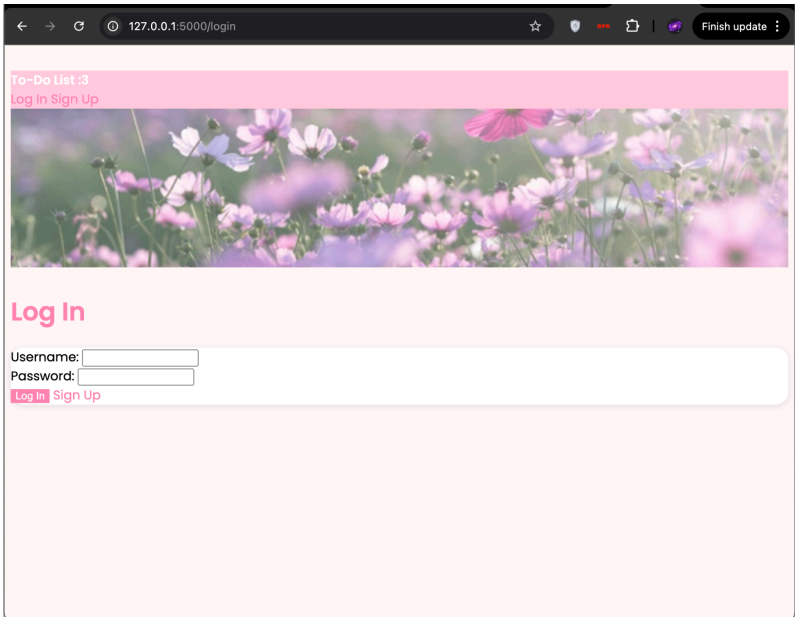


Figure 1&2: User is greeted with a Login page where they can enter pre-existing credentials or create a new account

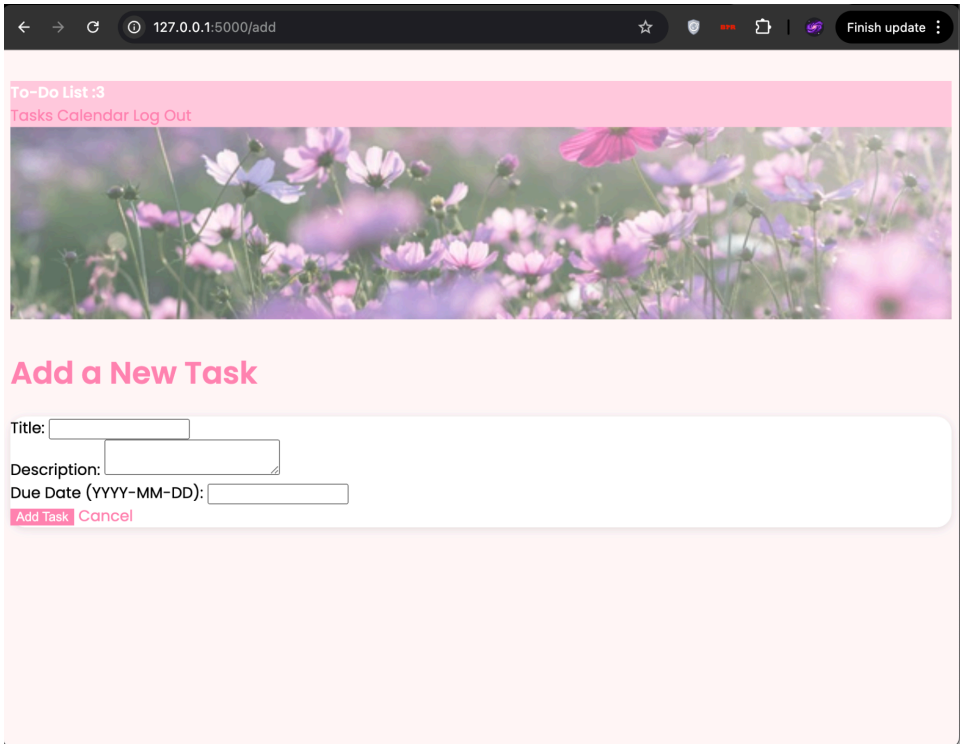
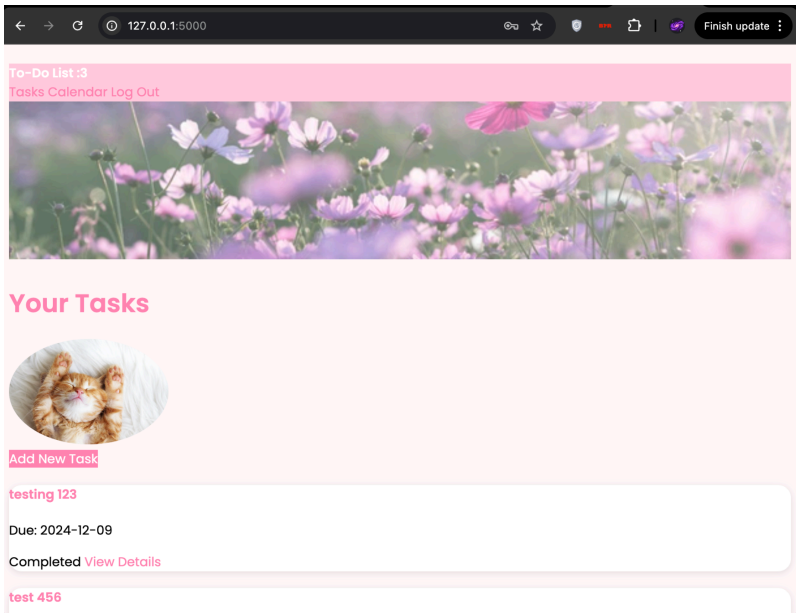


Figure 3: The user can then add a task, allowing for a title, description, and due date to be personalized for each task added



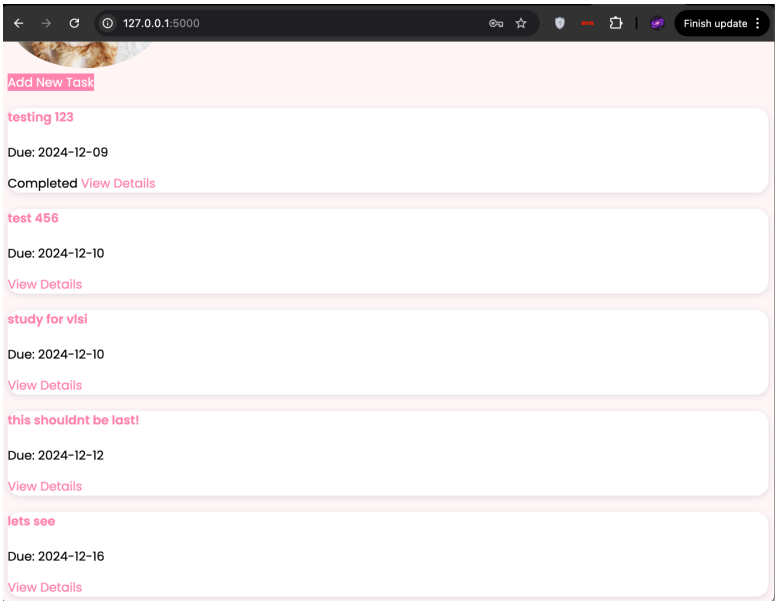


Figure 4&5: Once logged in, the homepage of the website appears as follows, showcasing tasks that have been added in due-date order, as well as their status if completed

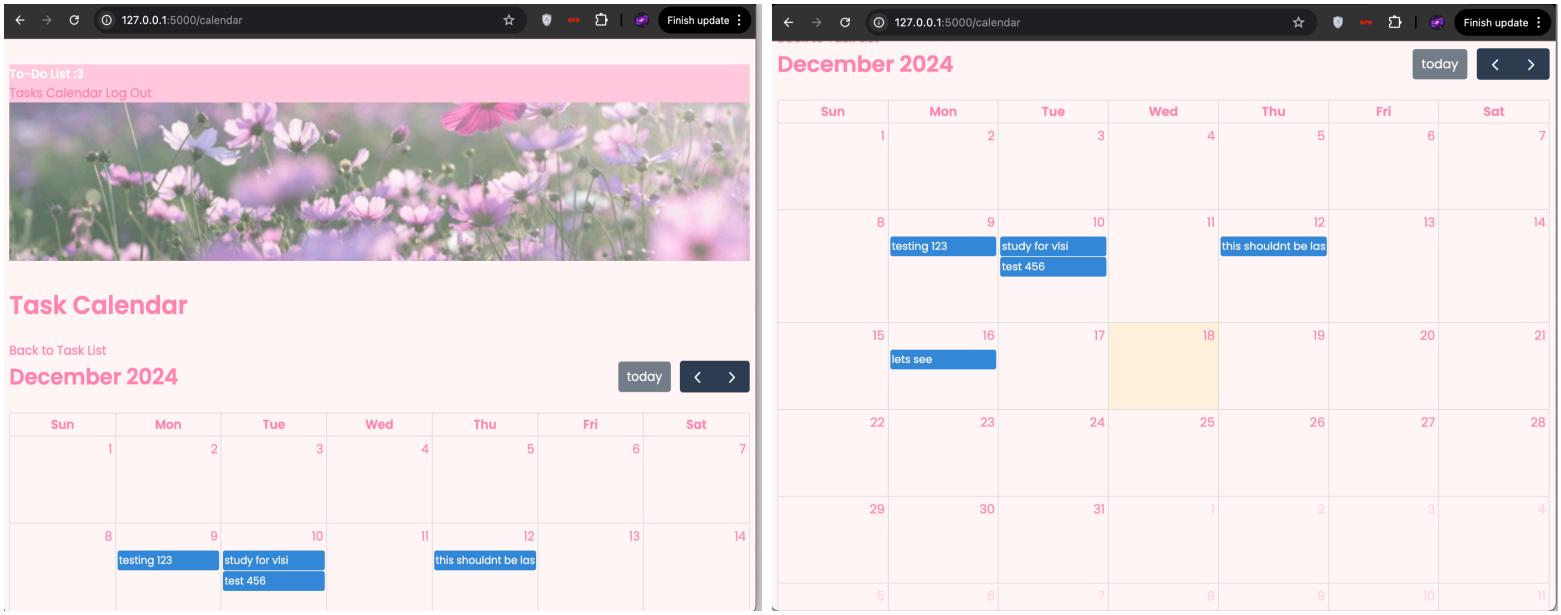


Figure 6&7: Using the Navigation bar, a user can navigate to the Calendar to view their tasks on a monthly calendar

The above images display the functionality of the website, recognizing the user input and adjusting the output behavior on the application based on this input. HTTP is able to GET, POST, PUT, and DELETE successfully. As well as utilize the database and included dependencies to run a calendar, and utilize user accounts to remember each user's tasks.

Discussion

During this project, the final application was able to successfully demonstrate the networking principles and design considerations that were outlined above. I integrated user accounts, personal task management, and a dynamic calendar to show how HTTP and data persistence work together to create a responsive, user-focused program. However, during this process I faced a few issues. One issue was the serialization of Task objects to JSON. I had problems with the syntax when passing these objects, and received error messages saying "Object of type Task is not JSON serializable". I decided to convert them to dictionaries before rendering to solve this issue. This emphasized the importance of understanding data formats and serialization. Similarly, I had issues with the calendar appearing on the page using FullCalendar. This gave me an error that said "FullCalendar is not defined". I realized I was using an outdated URL link for FullCalendar, and I ended up finding the correct one on their website giving instructions on how to utilize the script.

Lastly, I had challenges with my environment setup. I had originally attempted to create the program without a virtual environment, however I had issues getting the correct libraries in the correct directories. I had faced errors trying to install Flask-SWLAlchemy when attempting to run the program this way. The fastest solution was to switch to the use of the virtual environment to install everything in a more organized and isolated manner. After resolving these issues, I understood a bit more about the foundations of networking principles and the importance of attention to detail when building these real-world applications.

Conclusion

This project illustrates how a Python-based web application can integrate user authentication, data persistence, and a dynamic interface to enhance user experience. By building a to-do list system with personalized tasks, a calendar view, and secure user logins, I was able to demonstrate the interplay of network protocols, database management, and front end design. The outcome was a fully functioning application that not only could handle HTTP communication and data storage, but also had a visually appealing and user friendly interface. The key takeaway is a deeper understanding of how all of these components work together to build a stable, secure, and intuitive web application that utilizes concepts that drive our modern Internet services today.

Appendix

app.py

```
import os
from flask import Flask, render_template, request, redirect, url_for, session, g
from models import db, User, Task
from werkzeug.security import generate_password_hash, check_password_hash
from flask import abort

app = Flask(__name__)
# Configure the database URI and the secret key for session management.
app.config['SQLALCHEMY_DATABASE_URI'] = 'sqlite:///app.db'
app.config['SECRET_KEY'] = 'some-secure-secret-key'  # change this in production
db.init_app(app)

# Create database tables if they do not already exist.
with app.app_context():
    db.create_all()

tasks = [
```

```
    # Example:
    # {"id": 1, "title": "Example Task", "description": "A sample task", "due_date":
"2024-12-10", "completed": False}
]


#
#   A decorator to ensure that the user is logged in before accessing certain routes.
#   If the user is not logged in, they are redirected to the login page.
def login_required(view):
    def wrapped_view(**kwargs):
        if g.user is None:
            return redirect(url_for('login'))
        return view(**kwargs)
    wrapped_view.__name__ = view.__name__
    return wrapped_view

# @app.route('/')
# @login_required
# def index():
#     tasks = Task.query.filter_by(user_id=g.user.id).all()
#     return render_template("index.html", tasks=tasks)
#   The home page displaying a list of tasks for the currently logged-in user.
#   Tasks are ordered by their due date for better organization.
@app.route('/')
@login_required
def index():
    tasks = Task.query.filter_by(user_id=g.user.id).order_by(Task.due_date).all()
    return render_template("index.html", tasks=tasks)



#   Allows the user to add a new task.
#   On GET: Display the form to create a new task.
#   On POST: Create the new task and save it to the database.
@app.route('/add', methods=['GET', 'POST'])
@login_required
def add_task():
    if request.method == 'POST':
        title = request.form.get("title")
        description = request.form.get("description")
        due_date = request.form.get("due_date")
        new_task = Task(
```

```
            title=title,
            description=description,
            due_date=due_date,
            user_id=g.user.id
        )
        db.session.add(new_task)
        db.session.commit()
        return redirect(url_for("index"))

    return render_template("add_task.html")

# Update other task routes similarly, ensuring user_id matches g.user.id
#    Displays detailed information about a single task identified by task_id.
#    If the task does not belong to the current user or doesn't exist, return 404.
@app.route("/task/<int:task_id>")
@login_required
def view_task(task_id):
    task = Task.query.filter_by(id=task_id, user_id=g.user.id).first()
    if task is None:
        abort(404)
    return render_template("view_task.html", task=task)

#    Marks a specified task as completed.
#    After updating the task, redirect back to the task's detail view.
@app.route("/complete/<int:task_id>")
@login_required
def mark_complete(task_id):
    task = Task.query.filter_by(id=task_id, user_id=g.user.id).first()
    if task:
        task.completed = True
        db.session.commit()
    return redirect(url_for("view_task", task_id=task_id))

#    Deletes a specified task from the database.
#    If successful, redirect back to the main task list.
@app.route("/delete/<int:task_id>")
@login_required
def delete_task(task_id):
    task = Task.query.filter_by(id=task_id, user_id=g.user.id).first()
    if task:
        db.session.delete(task)
        db.session.commit()
```

```
    return redirect(url_for("index"))

#    Allows the user to edit an existing task.
#    On GET: Display the current task details in a form.
#    On POST: Update the task in the database.
@app.route("/edit/<int:task_id>", methods=["GET", "POST"])
@login_required
def edit_task(task_id):
    task = Task.query.filter_by(id=task_id, user_id=g.user.id).first()
    if not task:
        abort(404)

    if request.method == "POST":
        title = request.form.get("title")
        description = request.form.get("description")
        due_date = request.form.get("due_date")

        if title and due_date:
            task.title = title
            task.description = description
            task.due_date = due_date
            db.session.commit()

        return redirect(url_for("view_task", task_id=task_id))

    return render_template("edit_task.html", task=task)

# @app.route("/calendar")
# @login_required
# def calendar_view():
#     tasks = Task.query.filter_by(user_id=g.user.id).all()
#     return render_template("calendar.html", tasks=tasks)
#    Displays a calendar view of all tasks. The tasks are converted into a list of
dictionaries
#    to ensure they can be serialized into JSON and displayed on the calendar.
@app.route("/calendar")
@login_required
def calendar_view():
    user_tasks = Task.query.filter_by(user_id=g.user.id).all()
    # Convert each Task object into a dict
    tasks_list = []
    for t in user_tasks:
```

```
        tasks_list.append({
            "id": t.id,
            "title": t.title,
            "description": t.description,
            "due_date": t.due_date,
            "completed": t.completed
        })

    return render_template("calendar.html", tasks=tasks_list)


#    Allows a new user to create an account.
#    On POST: Creates the user with a hashed password.
#    If the username already exists, return an error.
@app.route('/signup', methods=['GET', 'POST'])
def signup():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')

        if User.query.filter_by(username=username).first():
            return "Username already exists!", 400

        hashed_pw = generate_password_hash(password)
        new_user = User(username=username, password_hash=hashed_pw)
        db.session.add(new_user)
        db.session.commit()
        return redirect(url_for('login'))

    return render_template('signup.html')

#    Allows an existing user to log in.
#    On POST: Checks credentials and, if valid, stores user_id in session.
#    If invalid, returns an error.
@app.route('/login', methods=['GET', 'POST'])
def login():
    if request.method == 'POST':
        username = request.form.get('username')
        password = request.form.get('password')

        user = User.query.filter_by(username=username).first()
        if user and check_password_hash(user.password_hash, password):
```

```
            session['user_id'] = user.id
            return redirect(url_for('index'))
        else:
            return "Invalid credentials!", 401
    return render_template('login.html')


#    Logs the user out by clearing the session.
#    Redirects to the login page.
@app.route('/logout')
def logout():
    session.clear()
    return redirect(url_for('login'))


#    Runs before each request. Loads the currently logged-in user from the database
#    using the user_id stored in the session, if any. This makes g.user available
#    globally in templates and routes.
@app.before_request
def load_logged_in_user():
    user_id = session.get('user_id')
    if user_id is None:
        g.user = None
    else:
        g.user = User.query.get(user_id)


# Run the Flask development server with debugging enabled.
if __name__ == "__main__":
    app.run(debug=True)
```

models.py

```
# models.py
from flask_sqlalchemy import SQLAlchemy
from datetime import datetime

db = SQLAlchemy()


#   The User model represents an individual user of the application.
#   Each user can have multiple associated tasks.
```

```python
class User(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    username = db.Column(db.String(80), unique=True, nullable=False)
    password_hash = db.Column(db.String(200), nullable=False)

    # Establish a one-to-many relationship: one user can have many tasks.
    # 'tasks' is accessible as a list of Task objects related to this user.
    tasks = db.relationship('Task', backref='user', lazy=True)


#    The Task model represents an individual to-do item created by a user.
#    Each task is linked to a specific user and contains details like title,
#    description, and due date.
class Task(db.Model):
    id = db.Column(db.Integer, primary_key=True)
    title = db.Column(db.String(120), nullable=False)
    description = db.Column(db.Text, nullable=True)
    due_date = db.Column(db.String(10), nullable=False)  # YYYY-MM-DD format
    completed = db.Column(db.Boolean, default=False)

    # Foreign key linking this task to a specific user.
    # 'user_id' references the 'id' field in the User model.
    user_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)
```

Add_task.html

```
{% extends "base.html" %}
{% block title %}Add Task{% endblock %}
{% block content %}
<h1>Add a New Task</h1>
<div class="card p-4">
    <form action="{{ url_for('add_task') }}" method="post">
        <div class="mb-3">
            <label for="title" class="form-label">Title:</label>
            <input type="text" id="title" name="title" required class="form-control">
        </div>
        <div class="mb-3">
            <label for="description" class="form-label">Description:</label>
            <textarea id="description" name="description"
class="form-control"></textarea>
        </div>
```

```
        <div class="mb-3">
            <label for="due_date" class="form-label">Due Date (YYYY-MM-DD):</label>
            <input type="text" id="due_date" name="due_date" required
class="form-control">
        </div>
        <button type="submit" class="btn btn-pink">Add Task</button>
        <a href="{{ url_for('index') }}" class="btn btn-light ms-2">Cancel</a>
    </form>
</div>
{% endblock %}
```

Base.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8" />
    <title>{% block title %}My To-Do App{% endblock %}</title>
    <!-- Bootstrap CSS -->
    <link
        rel="stylesheet"
        href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css"

integrity="sha384-ENjdO4Dr2bkBIFxQ2l6tW6M5WruT8vc8Jp0UIjRgkZwqpp1YnjcFNOZcdwFRRZ9"
        crossorigin="anonymous"
    >
    <!-- Google Fonts -->
    <link rel="stylesheet"
href="https://fonts.googleapis.com/css2?family=Poppins:wght@300;400;600&display=swap">

    <style>
        body {
            font-family: 'Poppins', sans-serif;
            background: #fff7f9;
        }
        .navbar {
            background: #ffcce0; /* Light pastel pink */
        }
        .navbar-brand {
            font-weight: 600;
            color: #fff;
        }
        .card {
```

```
            border-radius: 1rem;
            background: #fff;
            box-shadow: 0 2px 8px rgba(0,0,0,0.1);
        }
        h1, h2, h3, h4 {
            color: #ff85b3;
        }
        a, a:hover, a:focus {
            text-decoration: none;
            color: #ff85b3;
        }
        .btn-pink {
            background: #ff85b3;
            color: #fff;
            border: none;
        }
        .btn-pink:hover {
            background: #ff9cc1;
        }

        /* Banner Styles */
        .banner {
            position: relative;
            height: 200px; /* adjust as desired */
            background: url("{{ url_for('static', filename='images/flowers.jpg') }}")
no-repeat center center;
            background-size: cover;
        }
        /* Semi-transparent overlay for softness */
        .banner::after {
            content: "";
            position: absolute;
            top: 0;
            left: 0;
            height: 100%;
            width: 100%;
            background: rgba(255,255,255,0.3); /* White overlay with 30% opacity */
        }

        .container {
            margin-top: 2rem;
        }
```

```
    </style>
    {% block head %}{% endblock %}
</head>
<body>
    <nav class="navbar navbar-light">
      <div class="container">
        <a class="navbar-brand" href="{{ url_for('index') }}">To-Do List :3</a>
        <div>
            {% if g.user %}
                <a href="{{ url_for('index') }}" class="me-3 text-white">Tasks</a>
                <a href="{{ url_for('calendar_view') }}" class="me-3
text-white">Calendar</a>
                <a href="{{ url_for('logout') }}" class="text-white">Log Out</a>
            {% else %}
                <a href="{{ url_for('login') }}" class="me-3 text-white">Log In</a>
                <a href="{{ url_for('signup') }}" class="text-white">Sign Up</a>
            {% endif %}
        </div>
      </div>
    </nav>

    <!-- Add the banner here -->
    <div class="banner"></div>

    <div class="container">
        {% block content %}{% endblock %}
    </div>

    <script

src="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/js/bootstrap.bundle.min.js"

integrity="sha384-kenU1KFdBIe4zVF0s0G1M5b4hcpxyD9F7jL+2IrQAKSz3al8NLI8q8Zvs30yw9jx"
        crossorigin="anonymous">
    </script>
</body>
</html>
```

Calendar.html

```
```

```
{% extends "base.html" %}
{% block title %}Calendar{% endblock %}
{% block content %}
<h1>Task Calendar</h1>
<a href="{{ url_for('index') }}" class="text-muted mb-3 d-inline-block">Back to Task
List</a>

<div id="calendar" class="mt-3"></div>

<!-- FullCalendar JS -->
<script
src="https://cdn.jsdelivr.net/npm/fullcalendar@6.1.15/index.global.min.js"></script>
<script>
   const myTasks = JSON.parse('{{ tasks|tojson|safe }}');
   const events = myTasks.map(task => ({
       title: task.title,
       start: task.due_date,
       url: "/task/" + task.id
   }));

   document.addEventListener('DOMContentLoaded', function() {
       var calendarEl = document.getElementById('calendar');
       var calendar = new FullCalendar.Calendar(calendarEl, {
           initialView: 'dayGridMonth',
           events: events,
           eventClick: function(info) {
               info.jsEvent.preventDefault();
               window.location.href = info.event.url;
           }
       });
       calendar.render();
   });
</script>
{% endblock %}
```

edit_task.html

```
{% extends "base.html" %}
{% block title %}Edit Task{% endblock %}
{% block content %}
```

```
<h1>Edit Task</h1>
<div class="card p-4">
   <form action="{{ url_for('edit_task', task_id=task.id) }}" method="post">
       <div class="mb-3">
           <label for="title" class="form-label">Title:</label>
           <input type="text" id="title" name="title" value="{{ task.title }}"
required class="form-control">
       </div>
       <div class="mb-3">
           <label for="description" class="form-label">Description:</label>
           <textarea id="description" name="description" class="form-control">{{
task.description }}</textarea>
       </div>
       <div class="mb-3">
           <label for="due_date" class="form-label">Due Date (YYYY-MM-DD):</label>
           <input type="text" id="due_date" name="due_date" value="{{ task.due_date
}}" required class="form-control">
       </div>
       <button type="submit" class="btn btn-pink">Save Changes</button>
       <a href="{{ url_for('view_task', task_id=task.id) }}" class="btn btn-light
ms-2">Cancel</a>
   </form>
</div>
{% endblock %}
```

Index.html

```
{% extends "base.html" %}
{% block title %}Task List{% endblock %}
{% block content %}
<h1>Your Tasks</h1>
<img src="{{ url_for('static', filename='images/cute_cat.jpg') }}"
    alt="A cute cat" class="img-fluid mb-4"
    style="max-width:200px; border-radius:50%;">
<div class="d-flex justify-content-end mb-3">
   <a href="{{ url_for('add_task') }}" class="btn btn-pink">Add New Task</a>
</div>
<div class="row">
 {% for task in tasks %}
 <div class="col-12 col-md-6 col-lg-4 mb-3">
   <div class="card p-3">
```

```
    <h4>{{ task.title }}</h4>
    <p class="text-muted">Due: {{ task.due_date }}</p>
    {% if task.completed %}
      <span class="badge bg-success mb-2">Completed</span>
    {% endif %}
    <a href="{{ url_for('view_task', task_id=task.id) }}" class="stretched-link
text-muted">View Details</a>
  </div>
 </div>
 {% endfor %}
</div>
{% endblock %}
```

login.html

```
{% extends "base.html" %}
{% block title %}Log In{% endblock %}
{% block content %}
<h1>Log In</h1>
<div class="card p-4">
  <form action="{{ url_for('login') }}" method="post">
      <div class="mb-3">
          <label for="username" class="form-label">Username:</label>
          <input type="text" id="username" name="username" required
class="form-control">
      </div>
      <div class="mb-3">
          <label for="password" class="form-label">Password:</label>
          <input type="password" id="password" name="password" required
class="form-control">
      </div>
      <button type="submit" class="btn btn-pink">Log In</button>
      <a href="{{ url_for('signup') }}" class="btn btn-light ms-2">Sign Up</a>
  </form>
</div>
{% endblock %}
```

Signup.html

```
{% extends "base.html" %}
{% block title %}Sign Up{% endblock %}
{% block content %}
<h1>Create an Account</h1>
```

```
<div class="card p-4">
   <form action="{{ url_for('signup') }}" method="post">
       <div class="mb-3">
           <label for="username" class="form-label">Username:</label>
           <input type="text" id="username" name="username" required
class="form-control">
       </div>
       <div class="mb-3">
           <label for="password" class="form-label">Password:</label>
           <input type="password" id="password" name="password" required
class="form-control">
       </div>
       <button type="submit" class="btn btn-pink">Sign Up</button>
       <a href="{{ url_for('login') }}" class="btn btn-light ms-2">Log In</a>
   </form>
</div>
{% endblock %}
```

View_task.html

```
{% extends "base.html" %}
{% block title %}View Task{% endblock %}
{% block content %}
<div class="card p-4">
   <h1>{{ task.title }}</h1>
   <p><strong>Description:</strong> {{ task.description }}</p>
   <p><strong>Due Date:</strong> {{ task.due_date }}</p>
   <p><strong>Status:</strong> {{ 'Completed' if task.completed else 'Pending' }}</p>
   <div class="mt-3">
       <a href="{{ url_for('mark_complete', task_id=task.id) }}" class="btn btn-pink
me-2">Mark as Complete</a>
       <a href="{{ url_for('edit_task', task_id=task.id) }}" class="btn btn-light
me-2">Edit</a>
       <a href="{{ url_for('delete_task', task_id=task.id) }}" class="btn
btn-danger">Delete</a>
   </div>
   <a href="{{ url_for('index') }}" class="d-block mt-3 text-muted">Back to Task
List</a>
</div>
{% endblock %}
```