Abstract

**Task manager using Python**

This mini project explores task updates, this task manager is used to add task

Retrieve the task data from the database. we run this application through the web

Potential benefits:

Decentralization: Eliminates reliance on centralized servers for file sharing, enhancing privacy and security.

Offline accessibility: Enables file transfer without internet connectivity, overcoming accessibility limitations.

Open-source development: The project will be open-source, allowing for community collaboration and future enhancements.

**Features:**

First, we create the MongoDB instance, we have to create a database named “Task Manager”

In the backend python starts the flask server.

After inserting the task we secure the task data.

When the task is being started for every interval of time the task will be displayed in the command prompt.

**Requirements:**

* Python3.x
* Flask
* MongoDB

**Flask**

1. Creating the Web Application:

app = Flask(\_\_name\_\_): This line creates an instance of the Flask application, which forms the core of your web app. It handles essential tasks like routing, request handling, and response generation.

2. Defining Routes:

@app.route('/'): This decorator associates the following function with the root URL (/). When a user accesses your app's homepage, this function will be responsible for generating the response.

@app.route('/tasks', methods=['POST', 'GET']): This decorator defines two routes for the /tasks URL, one for handling POST requests (adding tasks) and another for GET requests (retrieving tasks).

3. Handling Requests and Responses:

The functions associated with each route (like add\_task and get\_tasks) handle the specific logic for processing requests and generating responses. They interact with your MongoDB database through PyMongo to add, retrieve, or manage tasks.

Flask helps you easily handle different HTTP methods (POST, GET), access request data (like JSON payloads), and format responses (like JSON for APIs).

4. Rendering Templates:

render\_template('index.html'): This line tells Flask to render the index.html template when a user requests the root URL. This template likely contains the HTML structure and presentation logic for your task manager's user interface.

**Mongo-DB**

MongoDB is a NoSQL database that is document-oriented, meaning it stores data in flexible, JSON-like documents instead of rigid rows and columns like traditional relational databases. This makes it well-suited for applications where data schema may evolve or where data has complex relationships.

Here are some key aspects of MongoDB used in the project:

1. Connection and Configuration:

You've defined the mongodb\_uri variable to connect to your MongoDB Atlas database.

You've created a MongoClient instance and accessed the specific database and collection containing your tasks.

2. Data Storage and Retrieval:

You use tasks\_collection.insert\_one to add new tasks as documents to the collection.

You use tasks\_collection.find to retrieve all tasks from the collection, excluding the \_id field.

3. Document Structure:

Your tasks are stored as documents with a single field named task containing the task description.

4. Advantages in your project:

MongoDB's flexible schema allows you to easily add new fields or modify existing tasks without changing the database structure.

The document-oriented approach aligns well with how you represent tasks as self-contained units with their descriptions.

Additional Notes:

Consider adding comments to your code explaining how you interact with MongoDB, including specific operations and their purposes.

You might explore advanced features like indexing or querying based on specific task attributes for more complex task management scenarios.

**Installation:**

Install the required libraries using pip:

Pip install Flask

Save the provided code as a Python file

Code for HTML:

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Task Manager</title>

<link rel="stylesheet" href="{{ url\_for('static', filename='styles.css') }}">

</head>

<body>

<h1>Task Manager</h1>

<div id="tasks-container">

<!-- Task list will be displayed here -->

</div>

<form id="add-task-form">

<label for="task">New Task:</label>

<input type="text" id="task" name="task" required>

<button type="submit">Add Task</button>

</form>

<script src="{{ url\_for('static', filename='scripts.js') }}"></script>

</body>

</html>

Code for CSS:

body {

font-family: 'Arial', sans-serif;

margin: 20px;

}

h1 {

color: #333;

}

#tasks-container {

margin-top: 20px;

}

form {

margin-top: 10px;

}

**Code for JavaScript:**

document.addEventListener('DOMContentLoaded', function () {

// Fetch and display tasks on page load

fetchTasks();

// Add event listener for form submission

document.getElementById('add-task-form').addEventListener('submit', function (event) {

event.preventDefault();

addTask();

});

});

function fetchTasks() {

// Fetch tasks from the backend

fetch('/tasks')

.then(response => response.json())

.then(data => {

const tasksContainer = document.getElementById('tasks-container');

tasksContainer.innerHTML = ''; // Clear existing tasks

// Display tasks

data.tasks.forEach(task => {

const taskElement = document.createElement('div');

taskElement.textContent = task.task;

tasksContainer.appendChild(taskElement);

});

});

}

function addTask() {

const taskInput = document.getElementById('task');

const task = taskInput.value;

// Add new task to the backend

fetch('/tasks', {

method: 'POST',

headers: {

'Content-Type': 'application/json',

},

body: JSON.stringify({ task: task }),

})

.then(response => response.json())

.then(data => {

// Fetch and display updated tasks

fetchTasks();

// Clear input field

taskInput.value = '';

});

}

**Code for Python file:**

from flask import Flask, render\_template, request, jsonify

from pymongo import MongoClient

app = Flask(\_\_name\_\_)

# MongoDB Atlas configuration

mongodb\_uri = "mongodb+srv://jeevansrinivas77:N%40dh2306@cluster0.yode8tw.mongodb.net/TaskManager"

database\_name = "TaskManager"

collection\_name = "tasksData"

client = MongoClient(mongodb\_uri)

db = client[database\_name]

tasks\_collection = db[collection\_name]

@app.route('/')

def index():

return render\_template('index.html')

@app.route('/tasks', methods=['POST'])

def add\_task():

data = request.json

new\_task = {'task': data['task']}

result = tasks\_collection.insert\_one(new\_task)

return jsonify({'message': 'Task added successfully', 'task\_id': str(result.inserted\_id)})

@app.route('/tasks', methods=['GET'])

def get\_tasks():

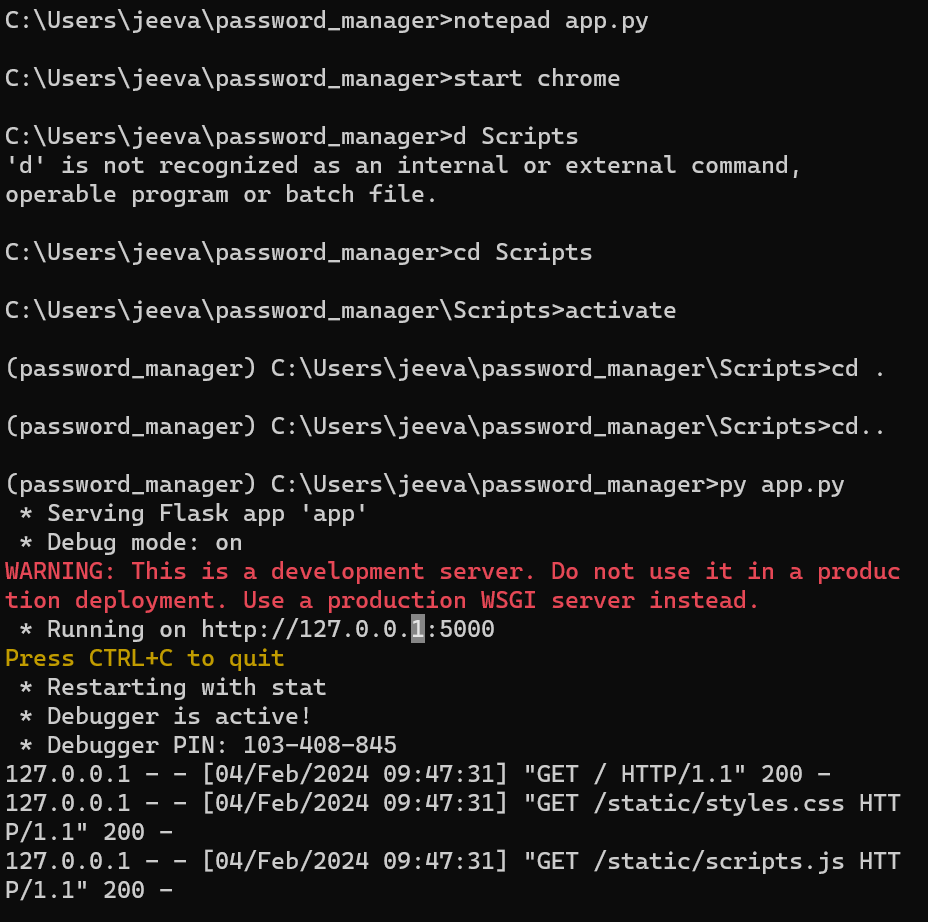
tasks = list(tasks\_collection.find({}, {'\_id': 0}))

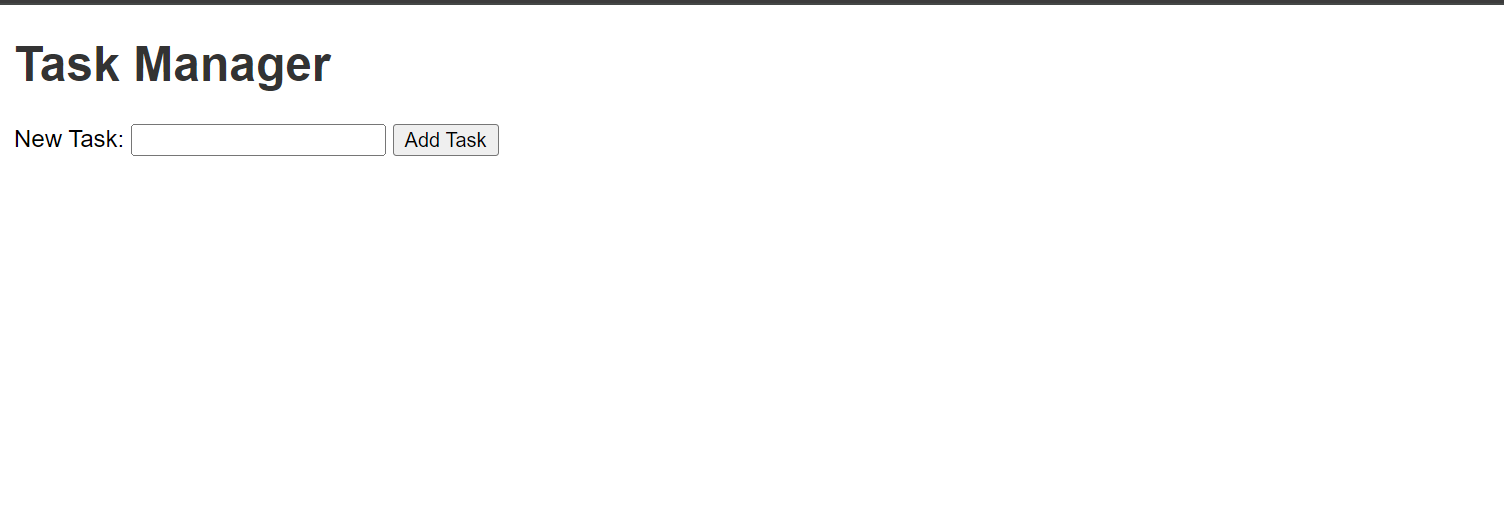
return jsonify({'tasks': tasks})

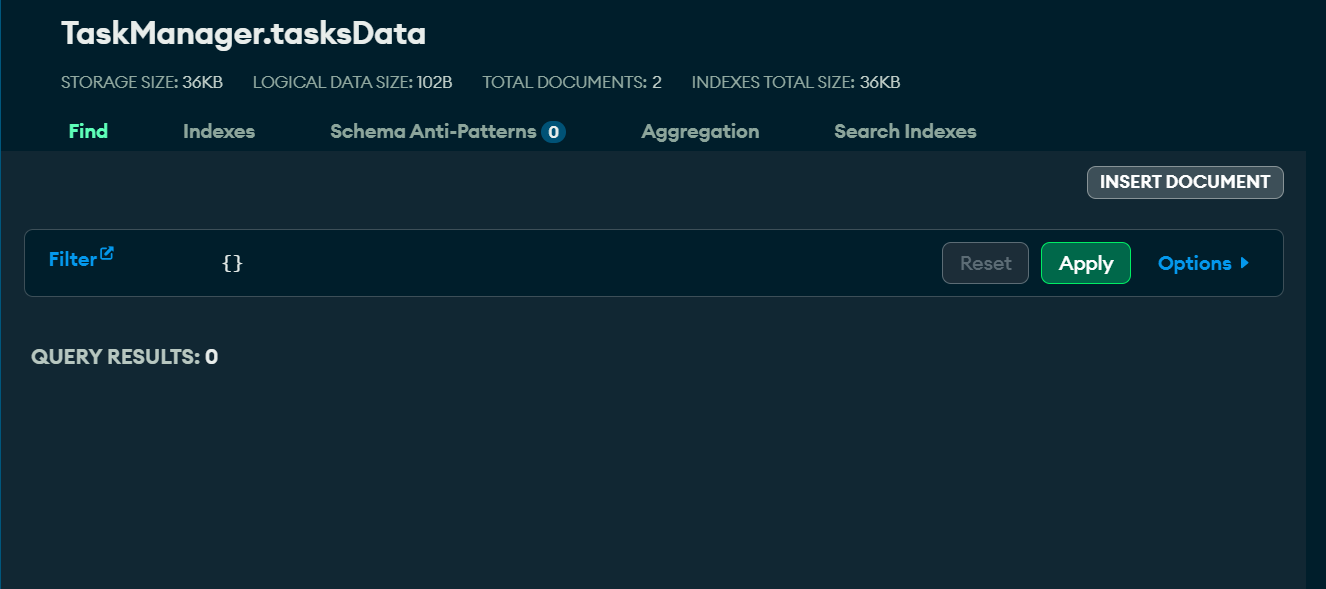
if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True)

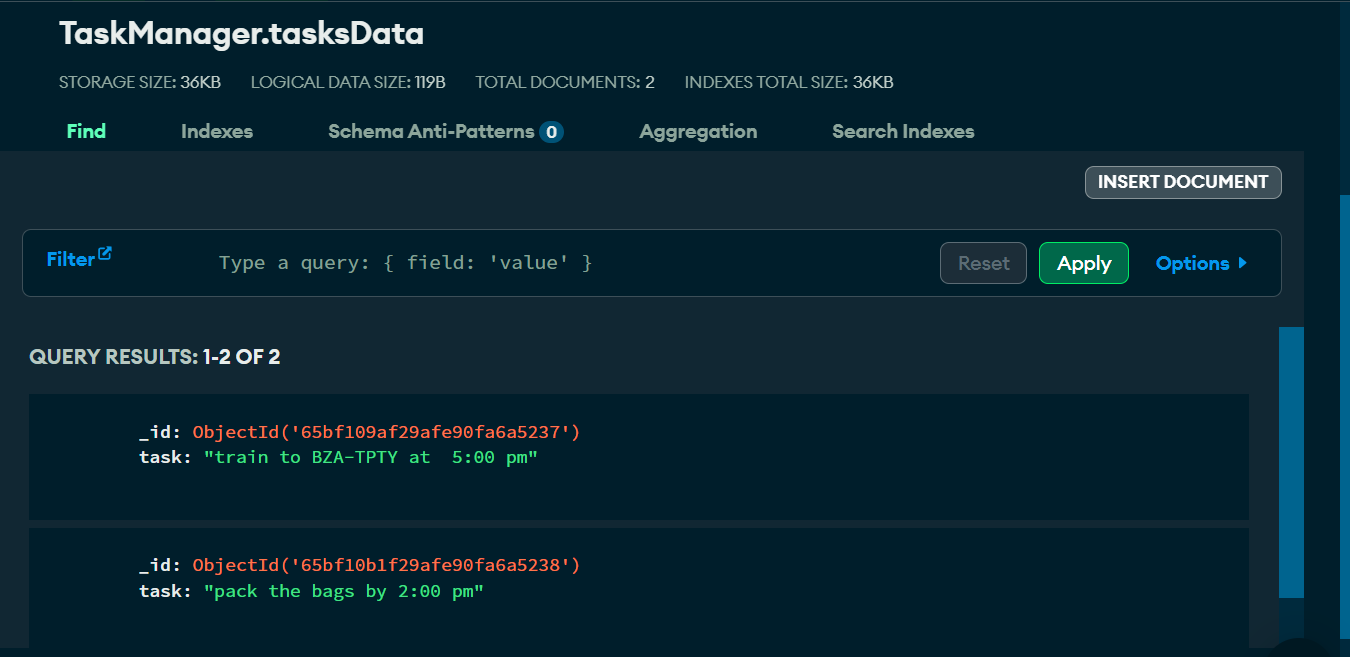
**output:**

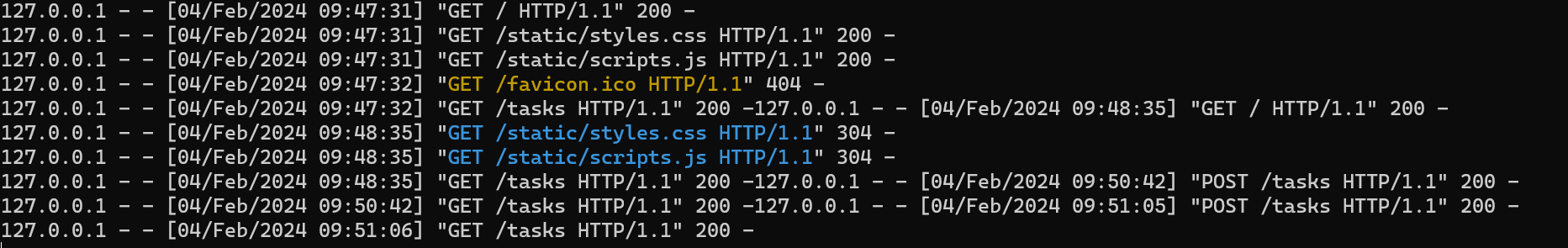






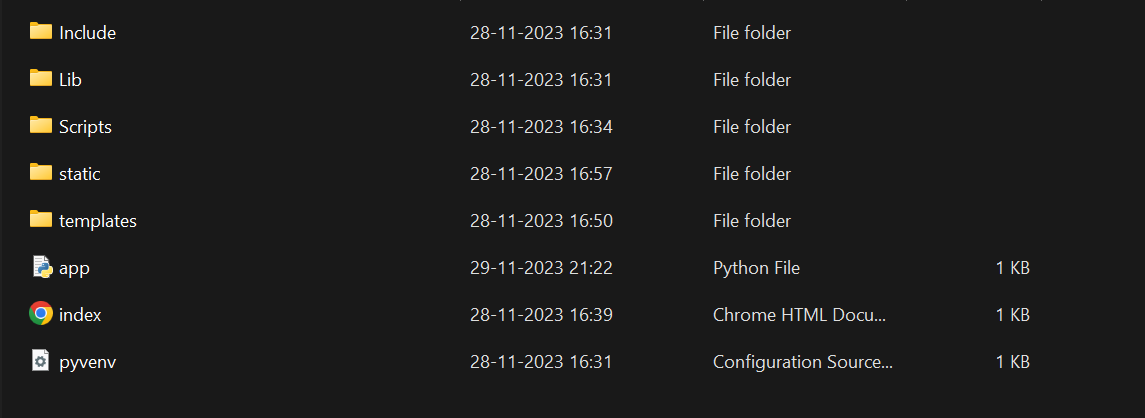






This is the structure of the Application Program

* We need to create a root folder named after your project
* Create a file within the root folder named templates
* Create a file within the root folder named static
* In The templates folder save your HTML,CSS Code
* In the static folder save your CSS,JAVASCRIPT Code
* In the main root folder save your python code



Testing:

Unit Testing:

• Test server startup:

1. Verified that the server binds to the correct port.

2. Checked for exceptions and error handling during initialization.

• Test file serving:

1. Mock file access and verify correct data retrieval.

2. Tested different file types and sizes.

3. Checked for error handling for missing or inaccessible files.

• Test QR code generation:

1. Verified that the generated QR code content matches the server URL.

2. Tested different QR code sizes and formats.

• Test user interface components:

1. Verified the button click events and corresponding actions.

2. Tested UI element visibility and functionality.

3. Check for proper display of server information and error messages.

Integration Testing:

• Test server response to HTTP requests:

1. Sent different request methods (GET, POST, etc.) and verified responses.

2. Tested file download and upload functionality.

• Test QR code scanning and server access:

1. Use a QR code reader app to scan the generated code and verify it opens the server URL.

2. Check for successful file access through the QR code scan.

• Test interaction between the user interface and server:

1. Verify clicking the "Start Server" button starts the server successfully.

2. Check if clicking the "Generate QR Code" button creates a valid QR code.

Functional Testing:

• Test file sharing from a user's perspective:

1. Share different types of files and verify successful access.

2. Test file sharing from different devices on the same network.

3. Check for the performance and responsiveness of the server.

• Test user interface experience:

1. Verify the buttons are easily accessible and intuitive.

2. Check for clear error messages and informative notifications.

3. Ensure the overall user experience is smooth and user-friendly.

Security Testing:

• Test for directory traversal vulnerabilities:

1. Verify that users cannot access unauthorized files and folders.

2. Check for potential exploits in file paths and access control mechanisms.

• Test for unauthorized file access and modification:

1. Ensure only authorized users can upload, download, and modify files.

2. Implement authentication and authorization mechanisms to restrict access.

• Test for network security vulnerabilities:

1. Check for potential network attacks like denial-of-service attacks.

2. Implement encryption and secure communication protocols.

References :

W3 Schools

Source for Project

Conclusion:

The Process of This project is keen on leaping the task manager in a way such that the task would store the task data in the database and retrieve the data in the command prompt. We used The Flask Web-Server to retain this project