Project Report

on

“Task Scheduling Web App”

Submitted By

Ayush Jain - A41

Sanchi Hisariya - A37

Aditya Kulat - A53

Pratik Lad - A57

Under the guidance of

Mrs.Sphurti Deshmukh

Department of Information Technology
Pimpri Chinchwad Education Trust's
Pimpri Chinchwad College of Engineering

Academic Year 2023-24

Abstract

This project aims to create a Todo application using Flask, a micro web framework in Python, and MongoDB, a NoSQL database. The application provides users with the ability to manage their tasks efficiently in a user-friendly interface.

Introduction

Introducing our cutting-edge Todo List App, where simplicity meets power, driven by MongoDB as its backend engine. This app redefines task management, providing a seamless and scalable solution for users seeking efficiency and flexibility. With MongoDB's agile data storage, our app adapts to your needs, offering a dynamic and personalized task management experience. Say goodbye to rigid systems and embrace a new era of productivity. Welcome to a world where your tasks, backed by MongoDB, become effortlessly manageable, allowing you to focus on what matters most—getting things done.

Scope

The scope of this project is to develop a simple and intuitive Todo application that allows users to add, edit, and delete tasks. The application will be built using Flask for the backend and MongoDB to store task-related data.

Purpose

The purpose of the project is to demonstrate the integration of Flask and MongoDB in developing a web application. The Todo application serves as a practical example of building a functional and dynamic web application using these technologies.

Functional Requirements

User Registration and Authentication: Users should be able to register and log in securely. Authentication mechanisms to protect user data.

Task Management:

Users can add new tasks with details such as title, description, and due date. Ability to mark tasks as complete or incomplete.

Users can edit or delete existing tasks. Task Filtering: Filter tasks based on completion status or due date.

User Profile: Users can view and update their profiles.

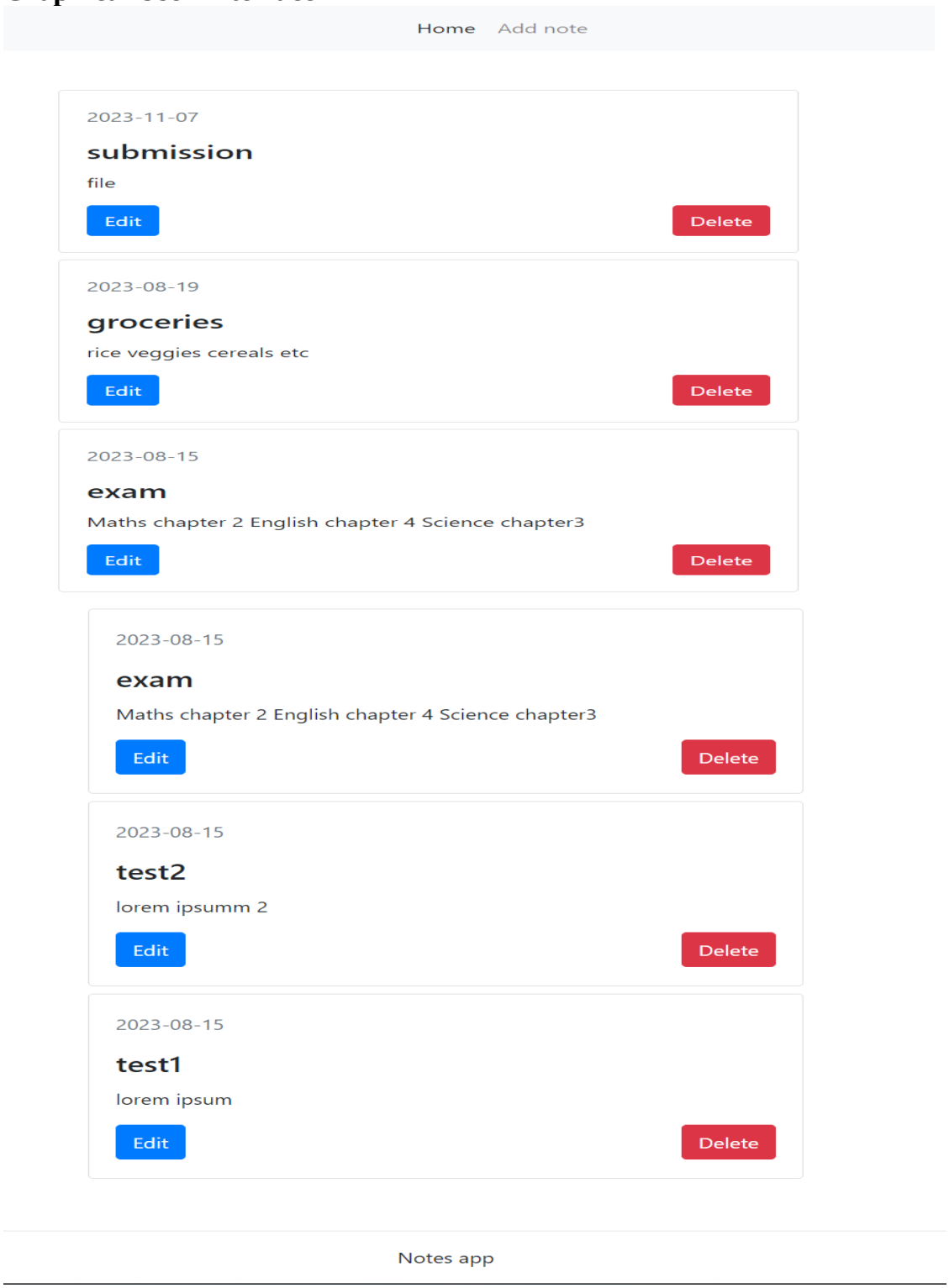
Data Modeling Features User Model:

Fields: username, email, password hash.

Task Model: Fields: title, description, due date, completion status, user reference.
Data Dictionary
User Model: username (string): User's unique username.
email (string): User's email address.
password_hash (string): Hashed password for security.
Task Model: title (string): Task title.
description (string): Task description.
due_date (date): Due date of the task.
completed (boolean): Completion status of the task.
user (reference): Reference to the user who created the task.
Relational Database Design The data is structured in a non-relational manner using MongoDB. The user and task models are connected through references.
Graphical User Interface The application's user interface is designed to be simple and user-friendly. It includes pages for task management, user profile, and authentication features.
Source Code (Important Modules) app.py: Main Flask application file. Contains routes for task management, user authentication, and profile. models.py: Defines the data models for users and tasks. forms.py: Includes forms for user registration, login, and task creation. templates/: Contains HTML templates for rendering pages. static/: Static files such as stylesheets and client-side scripts. Testing Document Unit Testing Unit tests have been implemented for critical functions, ensuring the reliability of the application. Tests cover user authentication, task creation, and filtering functionalities. Integration Testing Integration tests verify the interaction between different components, including the Flask application and MongoDB.

Data Dictionary Features Data Model:
The unique username chosen by the user during registration.The email address associated with the user's account.The hashed representation of the user's password for security.A detailed description of the task to provide additional context.The date by which the task should be completed.A flag indicating whether the task is completed or not. A reference to the user who created the task.

Graphical User Interface



Source Code(Important Modules):

HTML CODE

```
from flask import Flask, request,redirect,render_template,get_flashed_messages
from flask_pymongo import PyMongo, ObjectId
from datetime import datetime

app=Flask(_name_)
app.config['MONGO_URI']='mongodb://localhost:27017/todo1'

mongo=PyMongo(app)
@app.route('/')
def home():
  # get the notes from the database
   notes = list(mongo.db.notes.find({}).sort("createdAt",-1))

   # render a view
   return
render_template("/home.html",homeIsActive=True,addNoteIsActive=False,notes=notes)
   @app.route('/add-note', methods=['GET','POST'])
def addNote():
   if(request.method == "GET"):
      return render_template("/add-note.html",homeIsActive=False,addNoteIsActive=True)
   elif (request.method == "POST"):
      # get the fields data
      title = request.form['title']
      description = request.form['description']
      createdAt = datetime.now()

      # save the record to the database

mongo.db.notes.insert_one({"title":title,"description":description,"createdAt":createdAt})

      # redirect to home page
      return redirect("/")

@app.route('/edit-note',methods=['GET','POST'])
def editNote():
   if request.method == "GET":

      # get the id of the note to edit
      noteId = request.args.get('form')


      # get the note details from the db
      note = dict(mongo.db.notes.find_one({"_id":ObjectId(noteId)}))

      # direct to edit note page
      return render_template('/edit-note.html',note=note)

   elif request.method == "POST":

      #get the data of the note
```

```
        noteId = request.form['_id']
        title = request.form['title']
        description = request.form['description']

        # update the data in the db

mongo.db.notes.update_one({"_id":ObjectId(noteId)},{"$set":{"title":title,"description":desc
ription}})

     # redirect to home page
        return redirect("/")
    @app.route('/delete-note',methods=['POST'])
def deleteNote():
    # get the id of the note to delete
    noteId = request.form['_id']

    # delete from the database
    mongo.db.notes.delete_one({ "_id": ObjectId(noteId)})

    # redirect to home page
    return redirect("/")


if(_name)=='__main_' :
    app.run(debug=True)
```

Conclusion:

The Flask-based Todo application successfully demonstrates the integration of Flask and MongoDB for building a dynamic web application. The project highlights the importance of user authentication, efficient task management, and a well-structured data model. Future improvements may include additional features, such as task prioritization and collaboration functionalities.