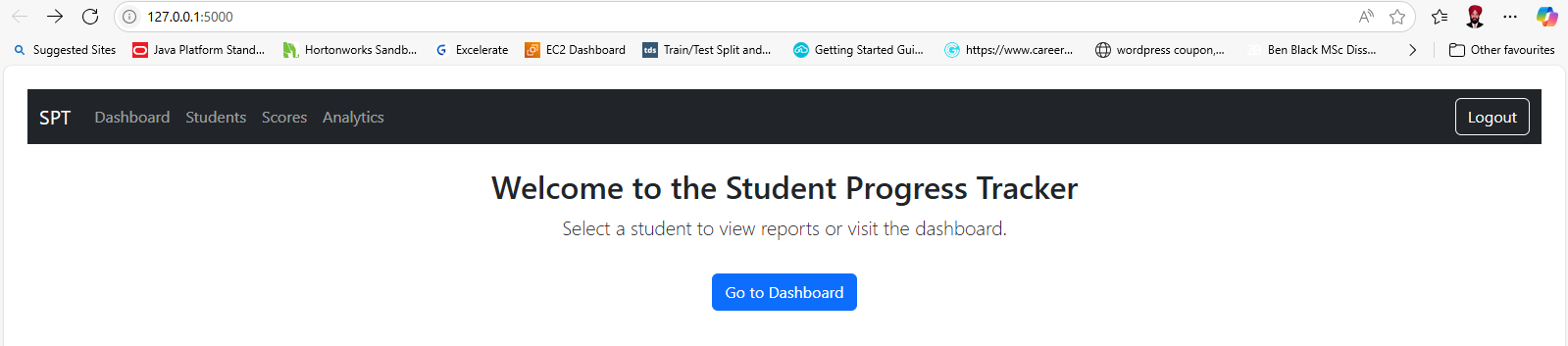
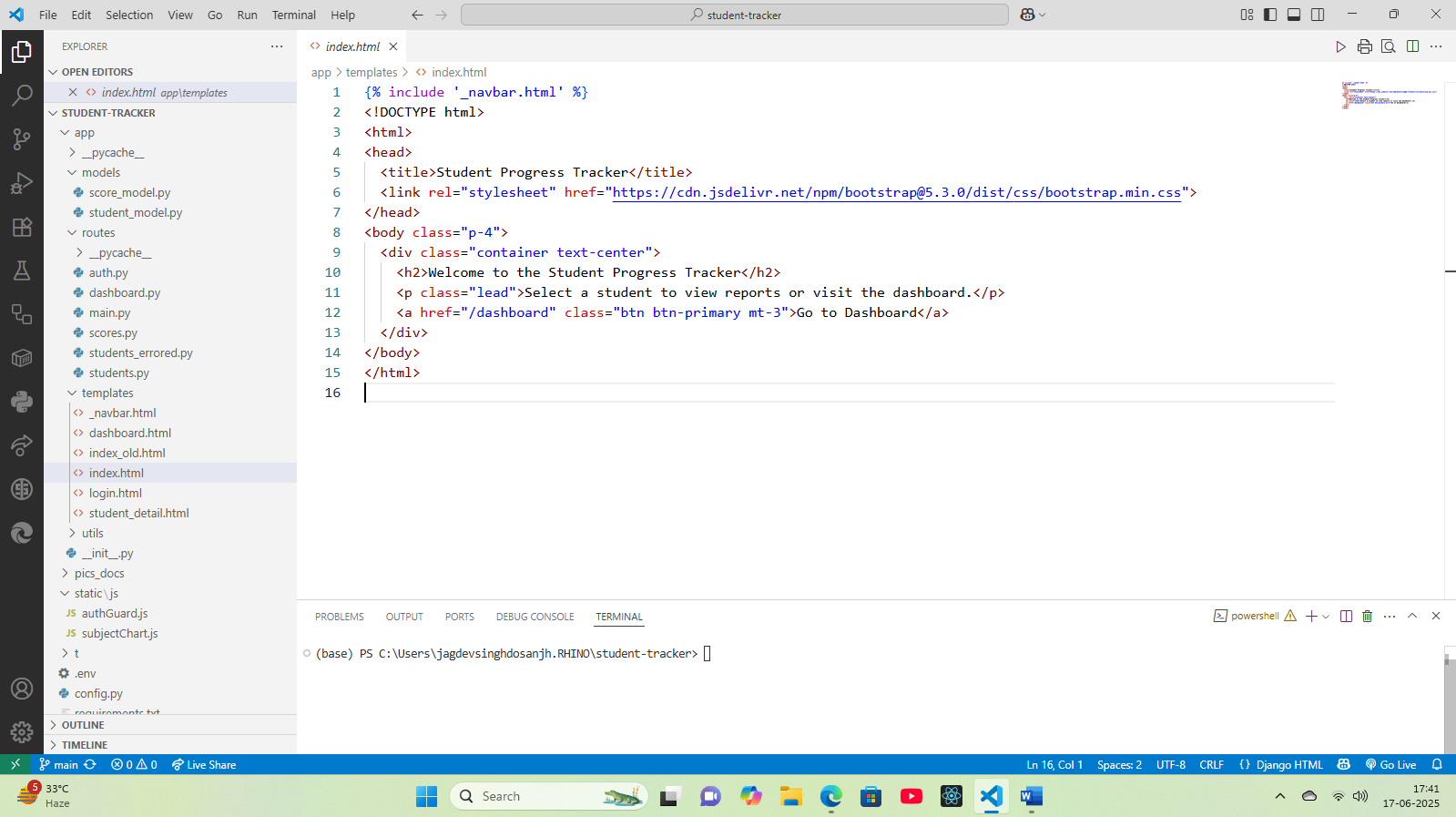
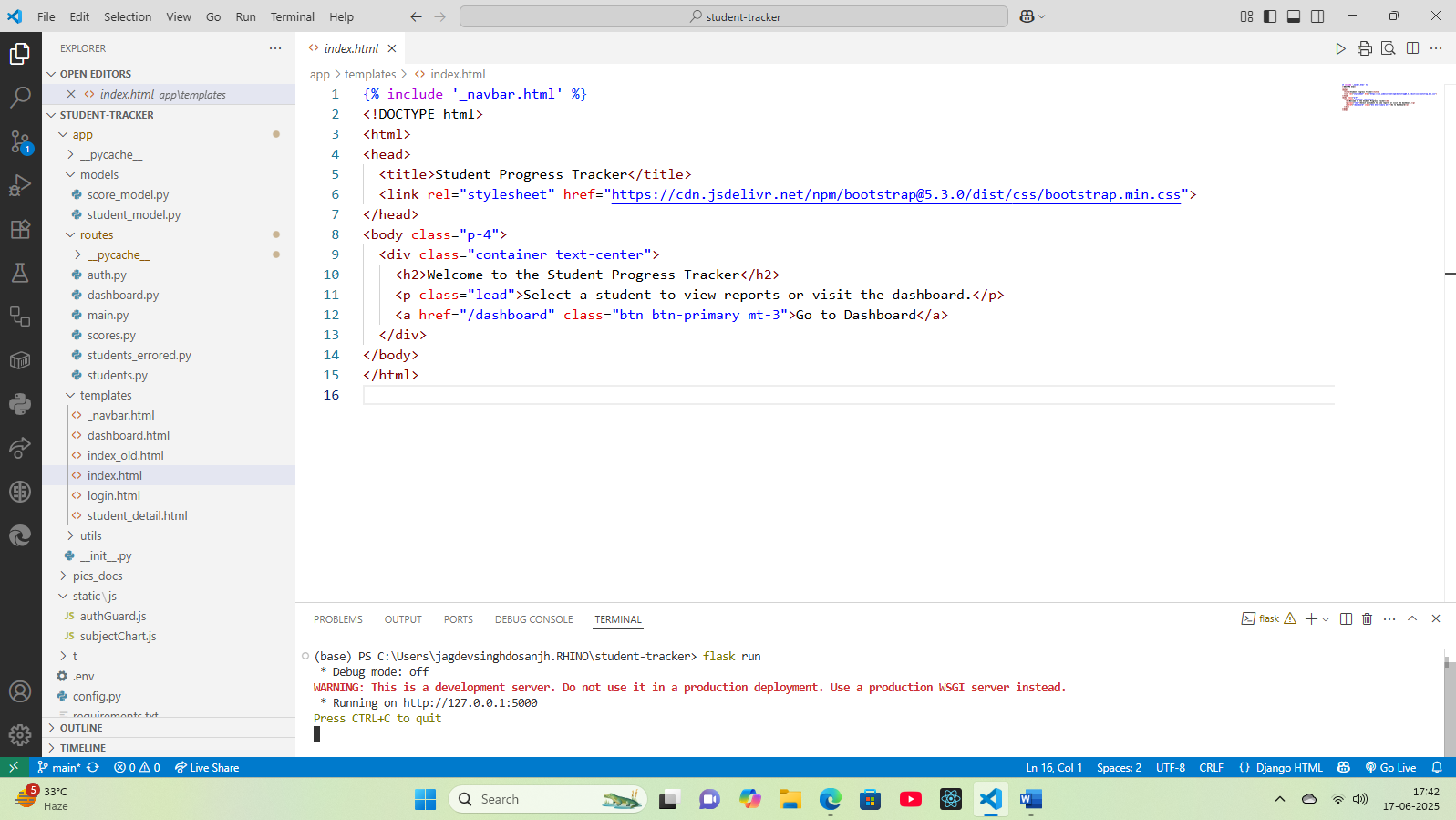
Student Progress Tracker



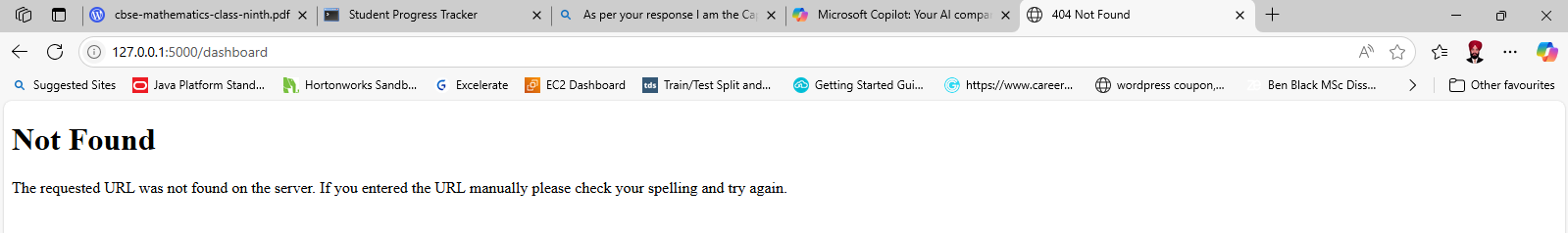
Outcome in MS Edge Browser



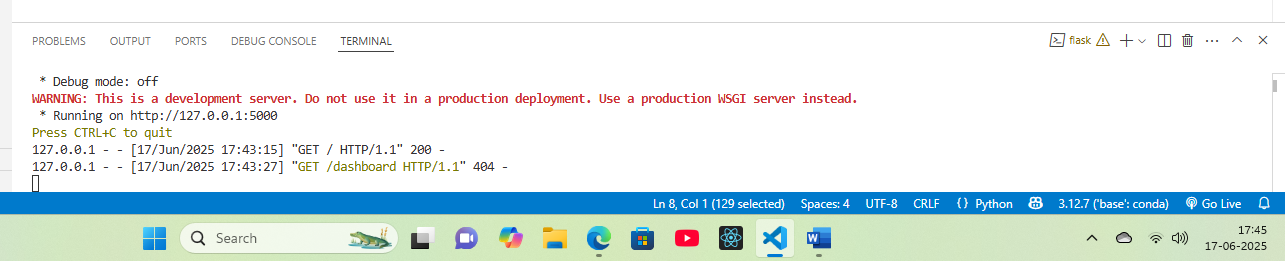
VS Code view 1



VS Code View 2 (after flask run command execution)



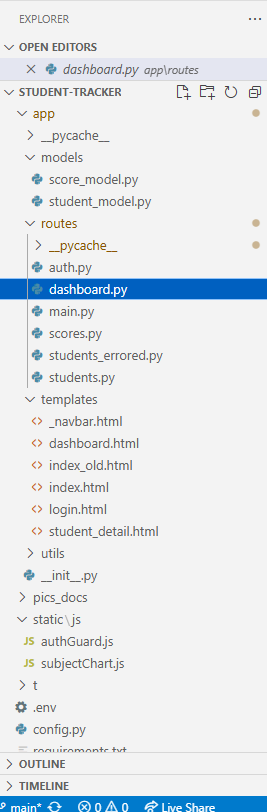
Dashboard Not found when navbar link Dashboard (as per blueprint) clicked to open it.



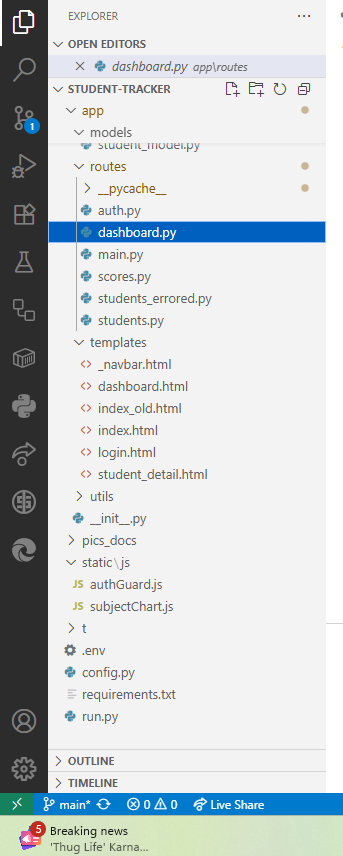
Warning 404 at VS Code Terminal for Dashboard resource

Other resources like Students Scores Analytics are also not working. Just 404 Messages for these resources.

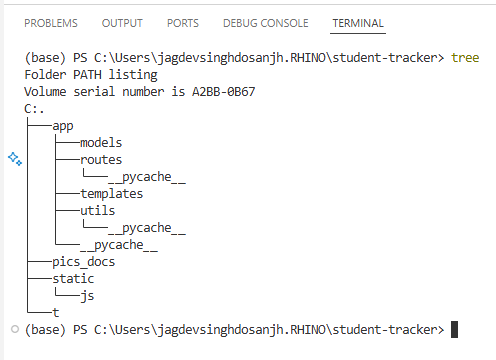
Directory Structure Snapshot 1 Top View



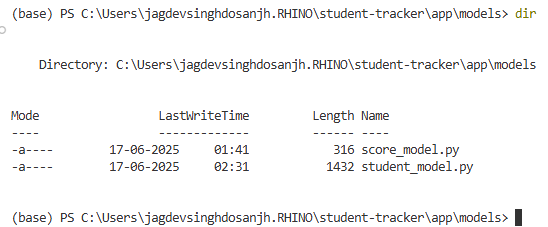
Directory Structure Snapshot 1 Bottom View

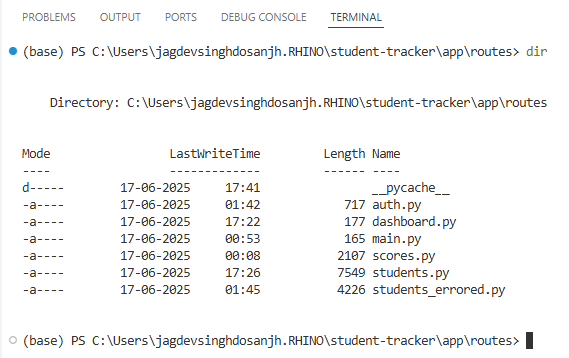


Directory Tree (SPT)

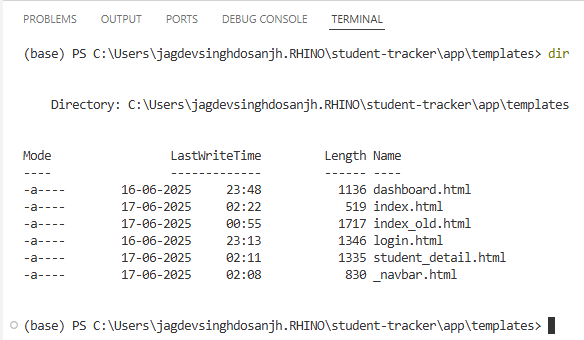


**Models directory Contents**

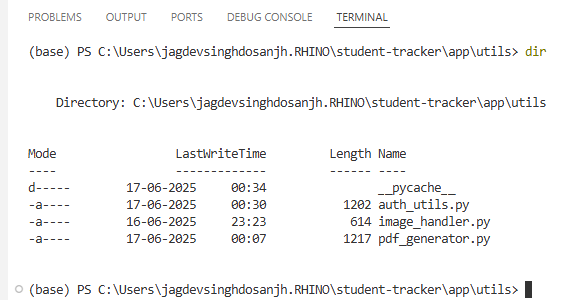


**Routes Directory Contents**

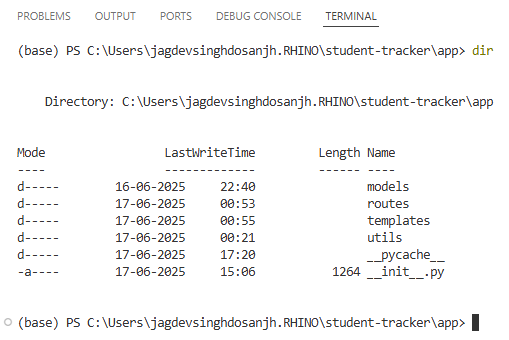
Templates directory contents



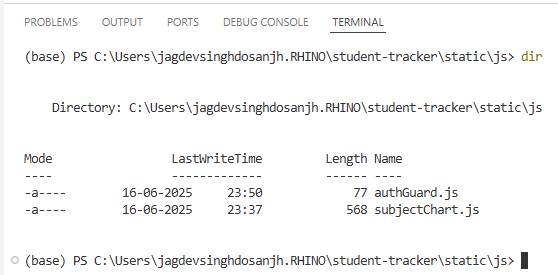
**Utils Directory Contents**



Complete **app** folder in **student-tracker** project



static/js contents



**Files and codes**

**Files**

1. score\_model.py : (inside models of app directory)
2. student\_model.py : (inside models)
3. auth.py: (inside routes of app directory)
4. dashboard.py: (inside routes)
5. main.py: (inside routes)
6. scores.py: (inside routes)
7. students.py: (inside routes)
8. \_navbar.html: (inside templates of app directory)
9. dashboard.html: (inside templates)
10. index.html: (inside templates)
11. login.html: (inside templates)
12. student\_detail.html: (inside templates)
13. auth\_utils.py: (inside utils of app directory)
14. image\_handler.py: (inside utils)
15. pdf\_generator.py(inside utils)
16. \_\_init\_\_.py (inside app directory)
17. authGuard.js: (inside static\js)
18. subjectChart.js: (inside static\js)
19. .env : (inside student-tracker project directly)
20. config.py: (inside student-tracker project directly)
21. requirements.txt: (inside student-tracker project directly)
22. run.py : (inside student-tracker project directly)

**Codes**

1. **socre\_model.py**

# score\_model.py

from datetime import datetime

from bson import ObjectId

class Score:

    def \_\_init\_\_(self, student\_id, subject, marks\_obtained, total\_marks, exam\_date, created\_at=None, updated\_at=None, \_id=None):

        self.id = str(\_id) if \_id else None

        self.student\_id = student\_id  # Should already be ObjectId

        self.subject = subject

        self.marks\_obtained = marks\_obtained

        self.total\_marks = total\_marks

        self.exam\_date = exam\_date

        self.created\_at = created\_at or datetime.utcnow()

        self.updated\_at = updated\_at or datetime.utcnow()

    def to\_dict(self):

        return {

            "\_id": ObjectId(self.id) if self.id else None,

            "student\_id": ObjectId(self.student\_id),

            "subject": self.subject,

            "marks\_obtained": self.marks\_obtained,

            "total\_marks": self.total\_marks,

            "exam\_date": self.exam\_date,

            "created\_at": self.created\_at,

            "updated\_at": self.updated\_at

        }

    @staticmethod

    def from\_dict(data):

        return Score(

            student\_id=data["student\_id"],

            subject=data["subject"],

            marks\_obtained=data["marks\_obtained"],

            total\_marks=data["total\_marks"],

            exam\_date=data["exam\_date"],

            created\_at=data.get("created\_at"),

            updated\_at=data.get("updated\_at"),

            \_id=data.get("\_id")

        )

1. **student\_model.py**

# student\_model.py

from bson import ObjectId

class Student:

    def \_\_init\_\_(self, name, roll\_no, email, class\_name=None, date\_of\_birth=None, guardian\_contact=None, \_id=None):

        self.id = str(\_id) if \_id else None

        self.name = name

        self.roll\_no = roll\_no

        self.email = email

        self.class\_name = class\_name

        self.date\_of\_birth = date\_of\_birth

        self.guardian\_contact = guardian\_contact

    def to\_dict(self):

        return {

            "full\_name": self.name,

            "roll\_number": self.roll\_no,

            "class\_name": self.class\_name,

            "date\_of\_birth": self.date\_of\_birth,

            "guardian\_contact": self.guardian\_contact,

            "\_id": ObjectId(self.id) if self.id else None,

            "email": self.email

        }

    @staticmethod

    def from\_dict(data):

        return Student(

            name=data.get("full\_name"),

            roll\_no=data.get("roll\_number"),

            email=data.get("email"),

            class\_name=data.get("class\_name"),

            date\_of\_birth=data.get("date\_of\_birth"),

            guardian\_contact=data.get("guardian\_contact"),

            \_id=data.get("\_id")

        )

1. **auth.py**

# app/routes/auth.py

import jwt

import datetime

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

from app import mongo

import os

from dotenv import load\_dotenv

load\_dotenv()

auth\_bp = Blueprint('auth', \_\_name\_\_)

SECRET\_KEY = os.getenv("SECRET\_KEY", "default\_secret\_key")

# 🔐 Login API (POST)

@auth\_bp.route('/login', methods=['POST'])

def login\_api():

    data = request.get\_json()

    if not data or "username" not in data or "password" not in data:

        return jsonify({"error": "Username and password required"}), 400

    user = mongo.db.users.find\_one({"username": data["username"]})

    if not user or user["password"] != data["password"]:

        return jsonify({"error": "Invalid credentials"}), 401

    token = jwt.encode({

        "user\_id": str(user["\_id"]),

        "exp": datetime.datetime.utcnow() + datetime.timedelta(hours=2)

    }, SECRET\_KEY, algorithm="HS256")

    return jsonify({"token": token})

# 🧭 Login Page (GET)

@auth\_bp.route('/login', methods=['GET'])

def login\_page():

    return render\_template('login.html')

1. **dashboard.py**

# In dashboard.py

from flask import Blueprint, render\_template

dashboard\_bp = Blueprint('dashboard', \_\_name\_\_)

@dashboard\_bp.route('/dashboard')

def dashboard():

    return render\_template('dashboard.html')

1. **main.py**

from flask import Blueprint, render\_template

main\_bp = Blueprint('main', \_\_name\_\_)

@main\_bp.route('/')

def home():

    return render\_template('index.html')

1. **scores.py**

from flask import Blueprint, request, jsonify

from app import mongo

from datetime import datetime

from bson.objectid import ObjectId

from app.utils.auth\_utils import token\_required

from flask import send\_file

from app.utils.pdf\_generator import generate\_student\_report

score\_bp = Blueprint('scores', \_\_name\_\_)

@score\_bp.route('/', methods=['POST'])

@token\_required

def add\_score():

    data = request.get\_json()

    required = ["student\_id", "subject", "marks\_obtained", "total\_marks", "exam\_date"]

    if not all(k in data for k in required):

        return jsonify({"error": "Missing fields"}), 400

    score = {

        "student\_id": ObjectId(data["student\_id"]),

        "subject": data["subject"],

        "marks\_obtained": int(data["marks\_obtained"]),

        "total\_marks": int(data["total\_marks"]),

        "exam\_date": data["exam\_date"],

        "created\_at": datetime.utcnow(),

        "updated\_at": datetime.utcnow()

    }

    mongo.db.scores.insert\_one(score)

    return jsonify({"message": "Score added"}), 201

@score\_bp.route('/student/<student\_id>', methods=['GET'])

def get\_scores\_by\_student(student\_id):

    scores = mongo.db.scores.find({"student\_id": ObjectId(student\_id)})

    result = []

    for s in scores:

        result.append({

            "subject": s["subject"],

            "marks\_obtained": s["marks\_obtained"],

            "total\_marks": s["total\_marks"],

            "exam\_date": s["exam\_date"]

        })

    return jsonify(result), 200

@score\_bp.route('/report/<student\_id>', methods=['GET'])

@token\_required

def download\_report(student\_id):

    student = mongo.db.students.find\_one({"\_id": ObjectId(student\_id)})

    if not student:

        return jsonify({"error": "Student not found"}), 404

    scores = list(mongo.db.scores.find({"student\_id": ObjectId(student\_id)}))

    pdf\_buffer = generate\_student\_report(student, scores)

    filename = f"{student['full\_name'].replace(' ', '\_')}\_Report.pdf"

    return send\_file(pdf\_buffer, as\_attachment=True, download\_name=filename, mimetype='application/pdf')

1. **students.py**

# app/routes/students.py

from flask import Blueprint, request, jsonify, render\_template, send\_file

from bson.objectid import ObjectId

from datetime import datetime

from app import mongo

from app.utils.auth\_utils import token\_required

from app.utils.image\_handler import save\_image, get\_image, delete\_image

import io

student\_bp = Blueprint('students', \_\_name\_\_)

# 🆕 Add a new student

@student\_bp.route('/', methods=['POST'])

@token\_required

def add\_student():

    data = request.form

    photo = request.files.get('photo')

    required\_fields = ("full\_name", "roll\_number", "class\_name", "date\_of\_birth", "guardian\_contact")

    if not all(k in data for k in required\_fields):

        return jsonify({"error": "Missing required fields"}), 400

    photo\_id = save\_image(photo) if photo else None

    student\_doc = {

        "full\_name": data['full\_name'],

        "roll\_number": data['roll\_number'],

        "class\_name": data['class\_name'],

        "date\_of\_birth": data['date\_of\_birth'],

        "guardian\_contact": data['guardian\_contact'],

        "photo\_id": photo\_id,

        "created\_at": datetime.utcnow(),

        "updated\_at": datetime.utcnow()

    }

    result = mongo.db.students.insert\_one(student\_doc)

    return jsonify({"message": "Student added successfully", "id": str(result.inserted\_id)}), 201

# 📥 Fetch all students

@student\_bp.route('/', methods=['GET'])

def get\_students():

    students = mongo.db.students.find()

    result = []

    for student in students:

        result.append({

            "id": str(student["\_id"]),

            "full\_name": student["full\_name"],

            "roll\_number": student["roll\_number"],

            "class\_name": student["class\_name"],

            "date\_of\_birth": student["date\_of\_birth"],

            "guardian\_contact": student["guardian\_contact"],

            "photo\_id": str(student["photo\_id"]) if student.get("photo\_id") else None,

            "created\_at": student["created\_at"],

            "updated\_at": student["updated\_at"]

        })

    return jsonify(result), 200

# 🖼️ View student photo

@student\_bp.route('/photo/<photo\_id>', methods=['GET'])

@token\_required

def get\_photo(photo\_id):

    binary\_data, mime, filename = get\_image(photo\_id)

    if binary\_data:

        return send\_file(io.BytesIO(binary\_data), mimetype=mime or 'image/jpeg', as\_attachment=False, download\_name=filename or 'photo.jpg')

    else:

        return jsonify({"error": "Photo not found"}), 404

# ❌ Delete student photo

@student\_bp.route('/photo/<photo\_id>', methods=['DELETE'])

@token\_required

def delete\_photo(photo\_id):

    if delete\_image(photo\_id):

        return jsonify({"message": "Photo deleted successfully"}), 200

    else:

        return jsonify({"error": "Photo not found"}), 404

# ✏️ Update student

@student\_bp.route('/<student\_id>', methods=['PUT'])

@token\_required

def update\_student(student\_id):

    data = request.form

    photo = request.files.get('photo')

    required\_fields = ("full\_name", "roll\_number", "class\_name", "date\_of\_birth", "guardian\_contact")

    if not all(k in data for k in required\_fields):

        return jsonify({"error": "Missing required fields"}), 400

    student\_doc = {

        "full\_name": data['full\_name'],

        "roll\_number": data['roll\_number'],

        "class\_name": data['class\_name'],

        "date\_of\_birth": data['date\_of\_birth'],

        "guardian\_contact": data['guardian\_contact'],

        "updated\_at": datetime.utcnow()

    }

    if photo:

        photo\_id = save\_image(photo)

        student\_doc["photo\_id"] = photo\_id

    result = mongo.db.students.update\_one({"\_id": ObjectId(student\_id)}, {"$set": student\_doc})

    if result.matched\_count == 0:

        return jsonify({"error": "Student not found"}), 404

    return jsonify({"message": "Student updated successfully"}), 200

# ❌ Delete student

@student\_bp.route('/<student\_id>', methods=['DELETE'])

@token\_required

def delete\_student(student\_id):

    result = mongo.db.students.delete\_one({"\_id": ObjectId(student\_id)})

    if result.deleted\_count == 0:

        return jsonify({"error": "Student not found"}), 404

    return jsonify({"message": "Student deleted successfully"}), 200

# 📄 View student profile with scores

@student\_bp.route('/<student\_id>', methods=['GET'])

@token\_required

def view\_student(student\_id):

    student = mongo.db.students.find\_one({"\_id": ObjectId(student\_id)})

    if not student:

        return jsonify({"error": "Student not found"}), 404

    scores = list(mongo.db.scores.find({"student\_id": ObjectId(student\_id)}))

    student["\_id"] = str(student["\_id"])  # Needed for safe Jinja2 rendering

    return render\_template('student\_detail.html', student=student, scores=scores)

# 📊 View student scores separately

@student\_bp.route('/<student\_id>/scores', methods=['GET'])

@token\_required

def view\_student\_scores(student\_id):

    scores = list(mongo.db.scores.find({"student\_id": ObjectId(student\_id)}))

    if not scores:

        return jsonify({"message": "No scores found for this student"}), 404

    return render\_template('student\_scores.html', scores=scores)

# ➕ Add a score for student

@student\_bp.route('/<student\_id>/scores', methods=['POST'])

@token\_required

def add\_student\_score(student\_id):

    data = request.json

    if not all(k in data for k in ("subject", "score", "date")):

        return jsonify({"error": "Missing required fields"}), 400

    score\_doc = {

        "student\_id": ObjectId(student\_id),

        "subject": data['subject'],

        "marks\_obtained": data['score'],

        "total\_marks": 100,  # default or adjust via UI later

        "exam\_date": data['date'],

        "created\_at": datetime.utcnow(),

        "updated\_at": datetime.utcnow()

    }

    result = mongo.db.scores.insert\_one(score\_doc)

    return jsonify({"message": "Score added successfully", "id": str(result.inserted\_id)}), 201

# ✏️ Update a student's score

@student\_bp.route('/<student\_id>/scores/<score\_id>', methods=['PUT'])

@token\_required

def update\_student\_score(student\_id, score\_id):

    data = request.json

    if not all(k in data for k in ("subject", "score", "date")):

        return jsonify({"error": "Missing required fields"}), 400

    score\_doc = {

        "subject": data['subject'],

        "marks\_obtained": data['score'],

        "exam\_date": data['date'],

        "updated\_at": datetime.utcnow()

    }

    result = mongo.db.scores.update\_one(

        {"\_id": ObjectId(score\_id), "student\_id": ObjectId(student\_id)},

        {"$set": score\_doc}

    )

    if result.matched\_count == 0:

        return jsonify({"error": "Score not found"}), 404

    return jsonify({"message": "Score updated successfully"}), 200

# ❌ Delete a student's score

@student\_bp.route('/<student\_id>/scores/<score\_id>', methods=['DELETE'])

@token\_required

def delete\_student\_score(student\_id, score\_id):

    result = mongo.db.scores.delete\_one({"\_id": ObjectId(score\_id), "student\_id": ObjectId(student\_id)})

    if result.deleted\_count == 0:

        return jsonify({"error": "Score not found"}), 404

    return jsonify({"message": "Score deleted successfully"}), 200

1. **\_navbar.html**

<nav class="navbar navbar-expand-lg navbar-dark bg-dark mb-4">

  <div class="container-fluid">

    <a class="navbar-brand" href="/">SPT</a>

    <div class="collapse navbar-collapse">

      <ul class="navbar-nav me-auto">

        <li class="nav-item"><a class="nav-link" href="/dashboard">Dashboard</a></li>

        <li class="nav-item"><a class="nav-link" href="/students">Students</a></li>

        <li class="nav-item"><a class="nav-link" href="/scores/dashboard">Scores</a></li>

        <li class="nav-item"><a class="nav-link" href="/scores/analytics">Analytics</a></li>

      </ul>

      <button class="btn btn-outline-light" onclick="logout()">Logout</button>

    </div>

  </div>

</nav>

<script>

  function logout() {

    localStorage.removeItem('token');

    window.location.href = '/login';

  }

</script>

1. **dashboard.html**

{% include '\_navbar.html' %}

<html lang="en">

<head>

  <meta charset="UTF-8">

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

  <title>Dashboard - Student Progress Tracker</title>

  <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">

<style>

    body {

      padding-top: 20px;

    }

    .container {

      max-width: 800px;

    }

  </style>

<script>

  if (!localStorage.getItem('token')) {

    window.location.href = '/login';

  }

</script>

<title>Welcome to Dashboard</title>

</head>

<body>

<div class="container">

  <h1 class="mt-4">Dashboard</h1>

  <p>Welcome to the Student Progress Tracker Dashboard!</p>

  {% comment %} <p>Use the navigation bar to access different sections.</p> {% endcomment %}

</div>

<div class="container">

  <h4 class="mb-3">Recent Student Scores</h4>

  <table class="table table-striped">

    <thead>

      <tr>

        <th>Student Name</th>

        <th>Subject</th>

        <th>Score (%)</th>

        <th>Date</th>

      </tr>

    </thead>

    <tbody id="recentScores">

      <!-- Recent scores will be populated here -->

    </tbody>

  </table>

<div class="container">

  <h4 class="mb-3">Subject-wise Performance Overview</h4>

  <canvas id="subjectChart" width="600" height="300"></canvas>

</div>

<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>

<script>

  async function loadChart() {

    const token = localStorage.getItem('token');

    const res = await fetch('/scores/analytics/subject-averages', {

      headers: { 'Authorization': 'Bearer ' + token }

    });

    const data = await res.json();

    const subjects = data.map(x => x.subject);

    const averages = data.map(x => x.average);

    new Chart(document.getElementById('subjectChart'), {

      type: 'bar',

      data: {

        labels: subjects,

        datasets: [{ label: 'Average %', data: averages, backgroundColor: 'steelblue' }]

      },

      options: { scales: { y: { beginAtZero: true, max: 100 } } }

    });

  }

  loadChart();

</script>

</body>

</html>

1. **index.html**

{% include '\_navbar.html' %}

<!DOCTYPE html>

<html>

<head>

  <title>Student Progress Tracker</title>

  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

</head>

<body class="p-4">

  <div class="container text-center">

    <h2>Welcome to the Student Progress Tracker</h2>

    <p class="lead">Select a student to view reports or visit the dashboard.</p>

    <a href="/dashboard" class="btn btn-primary mt-3">Go to Dashboard</a>

  </div>

</body>

</html>

1. **login.html**

{% include '\_navbar.html' %}

<!DOCTYPE html>

<html>

<head>

  <title>Login | SPT</title>

  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">

</head>

<body class="p-5">

  <div class="container col-md-4">

    <h3 class="mb-4">Teacher Login</h3>

    <form id="loginForm">

      <input type="text" class="form-control mb-2" name="username" placeholder="Username" required />

      <input type="password" class="form-control mb-2" name="password" placeholder="Password" required />

      <button class="btn btn-primary w-100">Log In</button>

    </form>

    <div id="msg" class="mt-3 text-danger"></div>

  </div>

  <script>

    document.getElementById('loginForm').onsubmit = async (e) => {

      e.preventDefault();

      const data = Object.fromEntries(new FormData(e.target).entries());

      const res = await fetch('/login', {

        method: 'POST',

        headers: {'Content-Type': 'application/json'},

        body: JSON.stringify(data)

      });

      const result = await res.json();

      if (res.ok) {

        localStorage.setItem('token', result.token);

        window.location.href = '/dashboard';  // redirect to dashboard page

      } else {

        document.getElementById('msg').innerText = result.error || 'Login failed';

      }

    };

  </script>

</body>

</html>

1. **student\_detail.html**

{% include '\_navbar.html' %}

<!DOCTYPE html>

<html>

  <head>

    <title>{{ student.full\_name }} - Progress Report</title>

    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css" />

  </head>

  <body class="p-4">

    <div class="container">

      <h3>{{ student.full\_name }}'s Progress Report</h3>

      <p>

        <strong>Class:</strong> {{ student.class\_name }} &nbsp;&nbsp; <strong>Roll No:</strong> {{ student.roll\_number }}

      </p>

      <p>

        <strong>Date of Birth:</strong> {{ student.date\_of\_birth }}

      </p>

      <hr />

      <h5>Subject Scores</h5>

      <table class="table table-bordered table-striped">

        <thead>

          <tr>

            <th>Subject</th>

            <th>Marks Obtained</th>

            <th>Total Marks</th>

            <th>Exam Date</th>

          </tr>

        </thead>

        <tbody>

          {% for score in scores %}

            <tr>

              <td>{{ score.subject }}</td>

              <td>{{ score.marks\_obtained }}</td>

              <td>{{ score.total\_marks }}</td>

              <td>{{ score.exam\_date }}</td>

            </tr>

          {% endfor %}

        </tbody>

      </table>

      <div class="mt-4">

        <a href="/scores/report/{{ student.\_id }}" class="btn btn-outline-primary" target="\_blank">Download PDF Report</a>

      </div>

    </div>

  </body>

</html>

1. **auth\_utils.py**

# app/utils/auth\_utils.py

from functools import wraps

from flask import request, jsonify

import jwt

import os

from dotenv import load\_dotenv

load\_dotenv()

SECRET\_KEY = os.getenv("SECRET\_KEY")

def token\_required(f):

    @wraps(f)

    def decorated(\*args, \*\*kwargs):

        token = None

        if 'Authorization' in request.headers:

            bearer = request.headers['Authorization']

            token = bearer.split(" ")[1] if " " in bearer else bearer

        if not token:

            return jsonify({'error': 'Token is missing!'}), 401

        try:

            data = jwt.decode(token, SECRET\_KEY, algorithms=["HS256"])

            request.user\_id = data['user\_id']

        except jwt.ExpiredSignatureError:

            return jsonify({'error': 'Token expired!'}), 401

        except jwt.InvalidTokenError:

            return jsonify({'error': 'Invalid token!'}), 401

        return f(\*args, \*\*kwargs)

    return decorated

1. **imge\_handler.py**

# app/utils/image\_handler.py

from gridfs import GridFS

from bson.objectid import ObjectId

from flask import current\_app

def save\_image(file):

    fs = GridFS(current\_app.mongo.db)

    return fs.put(file, filename=file.filename)

def get\_image(photo\_id):

    fs = GridFS(current\_app.mongo.db)

    try:

        image = fs.get(ObjectId(photo\_id))

        return image.read(), image.content\_type, image.filename

    except:

        return None, None, None

def delete\_image(photo\_id):

    fs = GridFS(current\_app.mongo.db)

    try:

        fs.delete(ObjectId(photo\_id))

        return True

    except:

        return False

1. **pdf\_generator.py**

# app/utils/pdf\_generator.py

from reportlab.lib.pagesizes import A4

from reportlab.pdfgen import canvas

from io import BytesIO

def generate\_student\_report(student, scores):

    buffer = BytesIO()

    c = canvas.Canvas(buffer, pagesize=A4)

    width, height = A4

    # Header

    c.setFont("Helvetica-Bold", 16)

    c.drawString(50, height - 50, f"Progress Report: {student['full\_name']}")

    c.setFont("Helvetica", 12)

    c.drawString(50, height - 80, f"Class: {student['class\_name']} | Roll No: {student['roll\_number']}")

    c.drawString(50, height - 100, f"Date of Birth: {student['date\_of\_birth']}")

    # Table Header

    y = height - 140

    c.setFont("Helvetica-Bold", 12)

    c.drawString(50, y, "Subject")

    c.drawString(200, y, "Marks Obtained")

    c.drawString(350, y, "Total Marks")

    c.drawString(470, y, "Exam Date")

    # Table Rows

    c.setFont("Helvetica", 12)

    for score in scores:

        y -= 20

        c.drawString(50, y, score["subject"])

        c.drawString(200, y, str(score["marks\_obtained"]))

        c.drawString(350, y, str(score["total\_marks"]))

        c.drawString(470, y, score["exam\_date"])

    c.showPage()

    c.save()

    buffer.seek(0)

    return buffer

1. **\_\_init\_\_.py**

# app/\_\_init\_\_.py

from flask import Flask

from flask\_pymongo import PyMongo

from dotenv import load\_dotenv

import os

load\_dotenv()

SECRET\_KEY = os.getenv("SECRET\_KEY")

mongo = PyMongo()

def create\_app():

    load\_dotenv()

    app = Flask(\_\_name\_\_)

    app.config["MONGO\_URI"] = os.getenv("MONGO\_URI")

    mongo.init\_app(app)

    # Blueprint registration (move these inside the function)

    from app.routes.students import student\_bp

    from app.routes.scores import score\_bp

    from app.routes.main import main\_bp

    from app.routes.dashboard import dashboard\_bp  # Add this

    from app.routes.auth import auth\_bp

    # Then register it:

    #app.register\_blueprint(auth\_bp, url\_prefix='/auth')

    app.register\_blueprint(auth\_bp)  # NO prefix

    app.register\_blueprint(dashboard\_bp)  # This line tells Flask about your dashboard routes

    app.register\_blueprint(main\_bp)

    app.register\_blueprint(score\_bp, url\_prefix='/scores')

    app.register\_blueprint(student\_bp, url\_prefix='/students')

    return app

1. **authGuard.js**

// inside static/js/authGuard.js

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

    const authToken = localStorage.getItem('authToken');

    const loginButton = document.getElementById('loginButton');

    const logoutButton = document.getElementById('logoutButton');

    if (authToken) {

        // User is authenticated

        loginButton.style.display = 'none';

        logoutButton.style.display = 'block';

    } else {

        // User is not authenticated

        loginButton.style.display = 'block';

        logoutButton.style.display = 'none';

    }

    logoutButton.addEventListener('click', function() {

        localStorage.removeItem('authToken');

        window.location.href = '/login'; // Redirect to login page

    });

});

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

1. **subjectChart.js**

//inside static/js/subjectChart.js

const token = localStorage.getItem('token');

fetch('/scores/analytics/subject-averages', {

  headers: { 'Authorization': 'Bearer ' + token }

})

.then(res => res.json())

.then(data => {

  const subjects = data.map(x => x.subject);

  const averages = data.map(x => x.average);

  new Chart(document.getElementById('subjectChart'), {

    type: 'bar',

    data: {

      labels: subjects,

      datasets: [{ label: 'Average %', data: averages, backgroundColor: 'steelblue' }]

    },

    options: { scales: { y: { beginAtZero: true, max: 100 } } }

  });

});

1. **.env**

MONGO\_URI=mongodb://localhost:27017/spt\_db

SECRET\_KEY=d10952582a68ffc37c05a283d1715aedd5d1890256f73475d8b5972bd6d10608

1. **config.py**

SECRET\_KEY="d10952582a68ffc37c05a283d1715aedd5d1890256f73475d8b5972bd6d10608"

1. **requirements.txt**

Flask

Flask-PyMongo

pymongo

python-dotenv

gunicorn

reportlab

1. **run.py**

from app import create\_app

app = create\_app()

if \_\_name\_\_ == "\_\_main\_\_":

    app.run(debug=True)

What I got today

