PROJECT DEVELOPMENT AND TESTING

VERSION 001 – BEP09 Bootcamp Management System

CODING STANDARD - PEP 8, Style Guide for Python Code (https://peps.python.org/pep-0008/)

1. **Plan for frontend and backend integration**

Step 1: Define the Architecture of the Integration

We'll create APIs that handle requests from the frontend (web or mobile) to the backend. The frontend will have a user-friendly interface that matches the backend's features, with form validations (like email, date of birth, and roles) to reduce errors before data reaches the backend.

Step 2: Create an API Layer

We'll build an API layer with endpoints for each function (e.g., student registration, fee validation, scoring). Security is crucial because the API exposes our system to external requests. We will use a reliable authentication method like JWT (JSON Web Tokens) to ensure only authorized users can access certain features.

Step 3: Setup Database

All data (students, courses, results) will be stored in a secure, scalable database like PostgreSQL or MySQL. We'll map our data structures to database tables for efficient data management. Scalability for us means that the database grows as more students, courses, and results are added without slowing down.

Step 4: Security Measures and Error handling

All data exchanged between the frontend and backend will be encrypted (using HTTPS). The backend's role-based access control (RBAC) will be mirrored in the frontend. This consistency ensures that unauthorized actions are blocked at every step, making the system more secure. We'll also ensure that the frontend shows clear, user-friendly error messages based on backend feedback.

Step 5: Testing and Deployment

We will create unit tests for each backend function and test the integration with the frontend by simulating real-world use cases (like registering a student through the frontend and checking if their details are correctly stored in the backend). User Acceptance Testing (UAT) will confirm that the system meets user needs. Finally, we'll deploy both the backend and frontend on a server or as a mobile app.

1. **Unit Testing: Write and perform unit tests to validate the functionality of individual components.**

A comprehensive unit test script has been written which validates the efficacy of the various components of the system. This document will be uploaded to the BEP09 repository on GitHub.

1. **System Testing: Ensure the system works as a whole, testing for bugs, performance issues, and edge cases.**

Manual Test Cases testing the overall BEP09 Management system for bugs and performance deficiencies. Our system was able to pass all test cases with flying colours.

1. Student Registration

Test Case 1.1: Successful Registration

Objective: Verify that a student can be registered successfully.

Preconditions: The system is up and running, and the registration form is accessible.

Steps:

1. Navigate to the registration page.
2. Enter valid student details.
3. Submit the form.

Expected Result: Registration is successful, and a confirmation message is displayed. The student's data is saved in the database.

Test Case 1.2: Invalid Email Format

Objective: Ensure the system handles invalid email formats correctly.

Preconditions: The system is up and running, and the registration form is accessible.

Steps:

1. Navigate to the registration page.
2. Enter an invalid email format
3. Submit the form.

Expected Result: An error message is displayed indicating that the email format is invalid.

Test Case 1.3: Future Date of Birth

Objective: Verify that the system prevents registration with a future date of birth.

Preconditions: The system is up and running, and the registration form is accessible.

Steps:

1. Navigate to the registration page.
2. Enter a future date for the date of birth (e.g., 2025-01-01).
3. Submit the form.

Expected Result: An error message is displayed indicating that the date of birth cannot be in the future.

1. Student Enrollment

Test Case 2.1: Successful Enrollment

Objective: Verify that a student can be enrolled in a course successfully.

Preconditions: The student is registered, and the admin is logged in.

Steps:

1. Log in as an admin.
2. Navigate to the enrollment section.
3. Select a student and a course (e.g., Data Science).
4. Enroll the student in the selected course.

Expected Result: A success message is displayed, and the student is enrolled in the course.

Test Case 2.2: Enrollment in Invalid Course

Objective: Ensure the system handles enrollment in an invalid course properly.

Preconditions: The student is registered, and the admin is logged in.

Steps:

1. Log in as an admin.
2. Navigate to the enrollment section.
3. Attempt to enroll a student in a non-existent course.

Expected Result: An error message is displayed indicating that the course is invalid.

Test Case 2.3: Unauthorized Enrollment Attempt

Objective: Verify that only admins can enroll students in courses.

Preconditions: A student is registered, and a non-admin user is logged in.

Steps:

1. Log in as a non-admin user.
2. Attempt to enroll a student in a course.

Expected Result: An access denied message is displayed, indicating that only admins can perform this action.

1. Student Scoring

Test Case 3.1: Successful Scoring

Objective: Verify that a student can be assigned a score successfully.

Preconditions: The student is enrolled in a course, and the admin is logged in.

Steps:

1. Log in as an admin.
2. Navigate to the scoring section.
3. Select a student and enter a score (e.g., 95).

Save the score.

Expected Result: A success message is displayed, and the score is updated for the student.

Test Case 3.2: Invalid Score Input

Objective: Ensure the system handles invalid score inputs correctly.

Preconditions: The student is enrolled in a course, and the admin is logged in.

Steps:

1. Log in as an admin.
2. Navigate to the scoring section.
3. Enter an invalid score (e.g., a string or negative number).
4. Attempt to save the score.

Expected Result: An error message is displayed indicating that the score is invalid.

1. Fee Payment Validation

Test Case 4.1: Successful Fee Payment

Objective: Verify that fee payments can be validated successfully.

Preconditions: The student is registered, and the admin is logged in.

Steps:

1. Log in as an admin.
2. Navigate to the fee payment validation section.
3. Mark the student's fees as paid.
4. Save the payment status.

Expected Result: A success message is displayed, and the fee payment status is updated.

Test Case 4.2: Unauthorized Fee Payment Validation

Objective: Ensure that only admins can validate fee payments.

Preconditions: The student is registered, and a non-admin user is logged in.

Steps:

1. Log in as a non-admin user.
2. Attempt to validate fee payments for a student.

Expected Result: An access denied message is displayed, indicating that only admins can perform this action.

5. Report Generation

Test Case 5.1: Generate Progress Report

Objective: Verify that a progress report can be generated for a student successfully.

Preconditions: The student is registered, scores are assigned, and the admin is logged in.

Steps:

1. Log in as an admin.
2. Navigate to the report generation section.
3. Generate a progress report for a student (e.g., John Doe).

Expected Result: The progress report is displayed, showing the student's performance.

Test Case 5.2: View Student Results

Objective: Ensure that student results can be viewed correctly.

Preconditions: The student is registered, and scores are assigned.

Steps:

1. Log in as an admin or authorized user.
2. Navigate to the student results section.
3. View the results for a specific student.

Expected Result: The results for the student are displayed correctly.