Grade Management System

Planning document

Course / Module:
BTSCC – Project Management
School:
Lycée Guillaume Kroll
Class:
PROMA1
Team Members:
Moshe Sarkis Marios
Mustafa Murtadha
Yona Yarvis
Teacher
Sam Hoffman

Table of Contents

1. Introduction

This project is about building a grade management web application that supports both students and professors. The system will allow students to log in and check their grades, while professors will be able to log in and create or update grades. To ensure reliability and portability, the application will be developed in Python, connected to a relational database, and deployed using Docker.

Planning Approach

Before starting development, the project requires a structured planning phase. The planning work is organized into the following activities:

- Personas: Defining typical users such as students, professors, and administrators to understand their goals and frustrations.
- User Stories: Writing at least twelve short, user-focused descriptions of system features, each with acceptance criteria.
- MoSCoW Prioritization: Ranking features into Must-have, should-have, could-have, and won't-have categories to manage scope.
- MS Planner Board: Creating a task board with columns (Backlog, Ready, In Progress, Review, Done) to track progress.
- Research: Investigating technologies and workflows that are new to the team, including Python frameworks, database integration, password hashing, and Docker.

Non-Functional Requirements

The system must also meet several quality standards:

- Passwords stored securely using hashing.
- Code written in a clean and well-structured way.
- The entire system launched with a single command (docker composed up).
- A README file included to explain setup and usage.

Deliverables

The project will produce the following outputs:

- Planning Document (PDF): Personas, user stories with MoSCoW prioritization, system architecture sketch, ER diagram.
- Research Document (PDF): Frameworks and tools investigated, justification of architectural decisions, sources used.
- MS Planner Board: Export or screenshots showing task tracking.
- Source Code Repository (ZIP): Python application, docker-compose.yml, and README.
- Demonstration: A 3–5-minute live demo of the system.
- Reflection Document (1–2 pages): Summary of team collaboration, successes, and challenges.

Personas

Persona 1 – Student (Primary User)

Name: Diddy Johnsen

Role: 1st year student 2nd semester, BTSCC

Goal: See all his grades in one place and know if he passed a module.

Motivation: Does not want to email teachers for marks.

Frustrations/Pain Points:

Different teachers use different tools.

Does not know his overall situation (passed/failed).

Needs from the system:

Secure login

List of courses + grades

See grading date and professor.

Persona 2 – Professor (Data Input User)

Name: Prof. Marc Schmit

Role: Teaches multiple modules

Goal: Quickly enter/update grades for many students.

Motivation: Wants grading to be transparent and fast.

Frustrations/Pain Points:

Manual Excel files

Students asking, "did you grade me?"

Needs from the system:

Login

See list of students in a course

Add/update grades.

Persona 3 – Admin / Program Coordinator (Support User)

Name: Marc Schmit

Role: School admin

Goal: Keep users and courses organized.

Motivation: System should be ready before semester.

Frustrations/Pain Points:

Having to call IT for simple user changes

Needs from the system:

Manage users (create student/professor)

Manage courses.

User stories

Authentication & Access

User 1 – Student Login

As a student, *I want to* log in to the system *so that* I can view my grades.

Acceptance Criteria:

1. Given valid username/password user is logged in and redirected to dashboard.
2. Given invalid credentials error message “Invalid credentials.”
3. Logout button is available after login.
4. Passwords are stored hashed (non-functional link).

User 2 – Professor Login

As a professor, *I want to* log in *so that* I can manage grades for my courses.

Acceptance Criteria:

1. Valid credentials → access to professor view.
2. Professors cannot see other professors’ private courses (only assigned ones).
3. Unauthorized users are blocked.

User 3 – Role-Based Access

As an admin/professor/student, *I want* the system to show only the functions for my role *so that* I don’t see options I can’t use.

Acceptance Criteria:

1. Students don’t see “Edit Grade”.
2. Professors don’t see “Create Admin”.
3. Admins see user/course management.

User 4 – View my grades

As a student I'd like to see my grades if available

Acceptance Criteria:

1. Table show course name is a grade date professor.
2. If you have no grades, it should show a message "No grades available yet."
3. Only the logged in student grades are shown.

User 5 API Access & Integration

As an admin I'd like to have a good API to integrate grades, the teachers, and the students to easily integrate into other platforms.

Acceptance Criteria:

1. To be able to use the grading system via the API.
2. To be able to use it efficiently it should be well documented.
3. It should be able to call by a Bash script if we need to automate it.

User 6- Head Teacher authority

As a head teacher I should be able to see the grades from all the courses.

Acceptance Criteria:

1. I should be able to see the grades and the teacher on that course and the names of the students.
2. Should not be able to edit the grades.
3. See the grades if updated.

User 8 -Teacher and student (registration date).

As a teacher and a student, I should be able to see the registration date and the date of ending the account.

Acceptance Criteria:

1. I should be able to see the registration date and the account end date.
2. The dates must be displayed in a consistent format such as YYYY-MM-DD.
3. The dates should accurately reflect when the account was created and when it will end.
4. I should not be able to edit these dates.

User story 9- Teacher update Existing grade.

As a head teacher and an admin, I want to update my grade so that I can correct mistakes.

Acceptance Criteria:

1. Head teacher and admin can open existing grade history.
2. Changes are saved and overwrite the previous value.
3. System saves the old grades in a text file.

User Story 10 – Password Recovery

As a student, I want to reset my password so that I can regain access if I forget it.
Acceptance Criteria:

1. Users can request a password reset via email.
2. System sends a secure reset link that expires after a set time.
3. New password must meet security requirements (length, complexity).
4. Passwords are stored hashed after reset.

User Story 11 – Admin Course Management (Admin)

As an admin, I want to create, update, and delete courses so that professors and students are linked to the correct modules.
Acceptance Criteria:

1. Admin can add new courses with name, code, and assigned professor.
2. Admin can update course details (e.g., professor assignment).
3. Admin can delete courses only if no grades are linked.
4. Changes are reflected immediately in the system.

User Story 12 – Admin Audit Log

As an admin, I want to see an audit log of grade changes so that I can track who updated grades and when.
Acceptance Criteria:

1. Every grade update is logged with timestamp, user role, and course.
2. Logs are read-only and cannot be altered.
3. Admin can filter logs by course, professor, or student.
4. Logs are exportable for compliance reporting.

Program Overview

Templates

- ### GraPe System HTML Template (Base.html)

  This HTML template displays a **table of student grades** using **Tailwind CSS** for styling. The table has a **header row** (Student, Subject, Grade) and a **body** that dynamically lists grades from the grade's variable using a template loop ({% for g in grades %}). Each row shows the **student's name, subject, and grade**, with hover effects for better readability. If there are no grades, a **placeholder row** appears stating "No grades yet." The table is fully responsive and styled for clarity and usability in a web application.

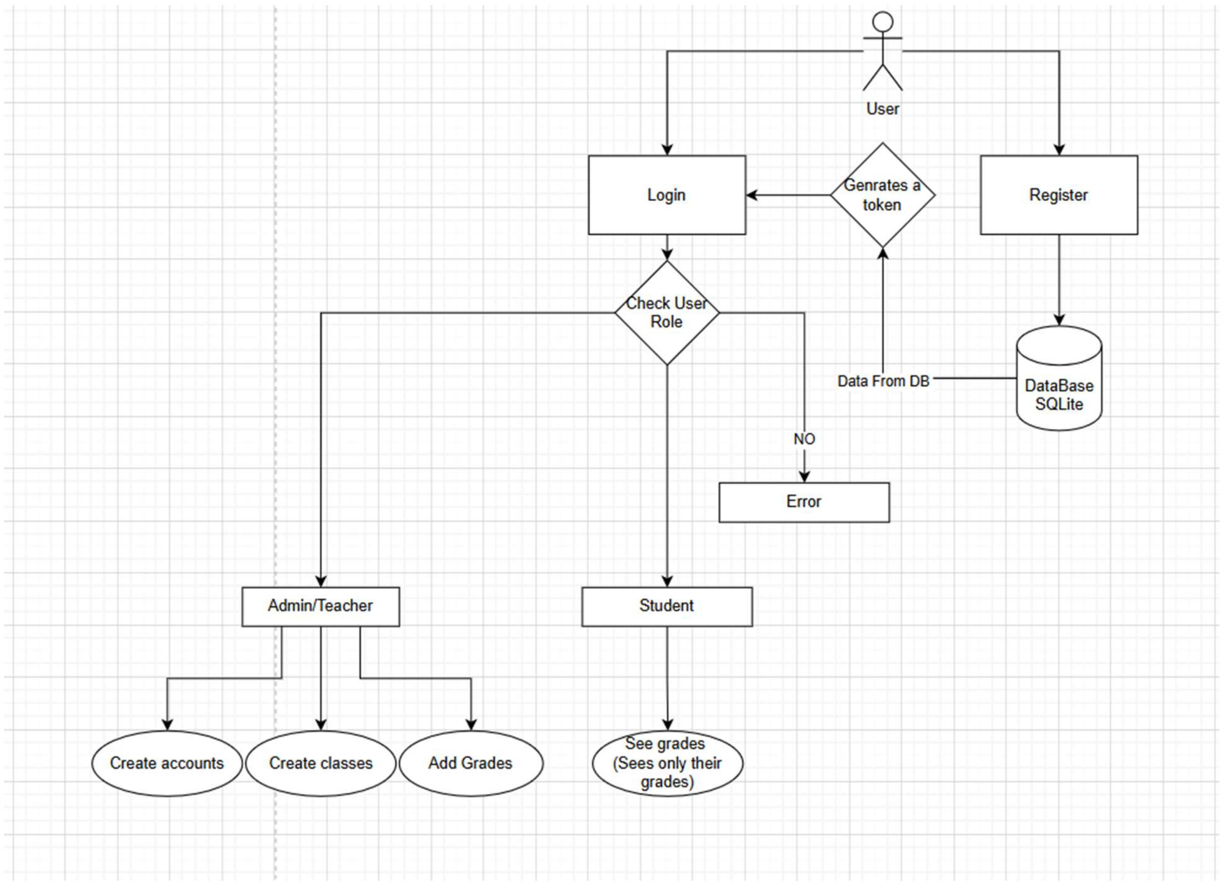
- ### Grades Table Template (grades_table.html)

  This HTML template displays a **table of student grades** using **Tailwind CSS** for styling. The table has a **header row** (Student, Subject, Grade) and a **body** that dynamically lists grades from the grade's variable using a template loop ({% for g in grades %}). Each row shows the **student's name, subject, and grade**, with hover effects for better readability. If there are no grades, a **placeholder row** appears stating "No grades yet." The table is fully responsive and styled for clarity and usability in a web application.

- ### Index Page Template

  This HTML template extends base.html and defines the **main content** for the GraPe System. It displays a **page title** (School Grades) and a **form** to add new grades, with fields for **student name, subject, and grade**. The form uses **HTMX** (hx-post, hx-target, hx-swap) to dynamically update the **grades table** (grades_table.html) without reloading the page. The grades table is included below the form and updates automatically when a new grade is submitted. Tailwind CSS classes provide a **responsive, styled layout** with grid alignment and hover effects.

System Architecture Sketch



After the user logs in, the system retrieves their account information from the database and **checks the user's role**. This decision step determines whether the user is an **Admin/Teacher** or a **Student**. If the role is Admin/Teacher, the system directs them to the admin panel where they can create accounts, create classes, and add grades. If the role is Student, the system takes them to their student dashboard to view only their own grades. If the system cannot match a valid role, it returns an error

Work Timesheet

Week 1 – Planning & Research

- **Day 1 (School):**
    - Communicated individual strengths and weaknesses.
    - Read and analysed the project document together.
    - Documented key requirements and deliverables.
    - Created a task list and decided on task distribution based on skills.
    - Agreed to keep each other updated on progress.
- **Rest of Week 1:**
    - Continued planning and refining user stories.
    - Performed MoSCoW prioritization.
    - Researched Python frameworks, database integration, secure password hashing, and Docker.
    - Documented research findings and architectural decisions.

Week 2 – Initial Development

- Set up Docker environment and created docker-compose.yml.
- Implemented basic project structure in Python (Flask/Django).
- Developed authentication module (login/logout for students and professors).
- Connected Python app to relational database.
- Updated MS Planner board with tasks and progress.

Week 3 – Full Development

- Completed grade management features (create, update, view grades).
- Added security measures (password hashing, input validation).
- Conducted internal testing and fixed bugs.
- Continued updating Planner board and documentation.

Week 4 – Finalization

- Finalized System Architecture Sketch.
- Completed Planning Document and Research Document.
- Prepared live demo and tested Docker Compose setup.
- Wrote Reflection document (team collaboration, challenges, lessons learned).
- Packaged source code and documentation for submission.

Reflection

Team Collaboration

Our collaboration improved significantly over time. Initially, we had difficulties working together because of technical issues we couldn't work on the code simultaneously at home due to environment setup problems. Additionally, we had a slow start because of different interpretations of the project requirements and conflicting ideas, which made decision-making hard.
To overcome this, we introduced regular communication, shared research, and used MS Planner to organize tasks. We also agreed on a clear architecture and workflow, which helped us move forward efficiently.

What Worked Well

- **Improved Communication:** Regular meetings and updates aligned the team.
- **Task Distribution by Strengths:** Assigning tasks based on skills increased productivity.
- **Use of MS Planner:** Visual task tracking kept everyone organized.
- **Knowledge Sharing:** Team members helped each other with Docker and database issues.
- **Early Planning:** Creating user stories and MoSCoW prioritization gave a clear roadmap.

Challenges

- **Technical Issues:** We couldn't work on the code together at home due to environment setup problems.
- **Slow Start:** Different interpretations of the project and conflicting ideas delayed progress.
- **Decision-Making:** It was hard to agree on frameworks and architecture at first.
- **Integration Problems:** Connecting Python app with the database inside Docker required troubleshooting.
- **Time Management:** Balancing school schedules with project deadlines was challenging.