VIETNAM INTERNATIONAL UNIVERSITY – HO CHI MINH CITY
INTERNATIONAL UNIVERSITY

WEB APPLICATION DEVELOPMENT PROJECT
DISTANCE LEARNING ACTIVITY MANAGER

By
Nguyễn Đức Thành Công - Student ID: ITCSIU22023
Phạm Chấn Quân - Student ID: ITCSIU22313
Nguyễn Hữu Thụy - Student ID: ITCSIU22141

Advisor: Prof. Nguyen Trung Nghia
A report submitted to the School of Computer Science and Engineering as part of the requirements for the Final Project in Web Application Development course, Semester 1, Academic Year 2025-2026

Ho Chi Minh city, Vietnam, 2025

TABLE OF CONTENTS

I. INTRODUCTION

1. ABOUT US

The Distance Learning Activity Manager (DLAM) is a web-based educational platform designed to bridge the gap between instructors and students in a remote learning environment. The logic within the codebase focuses on streamlining the management of courses, activities, resources, and submissions. It facilitates a structured learning path where instructors can organize content and assess student performance, while students can track their progress and submit assignments efficiently.

2. PRODUCT INFORMATION

With the rapid growth of online and blended learning, educational institutions require a centralized system to manage teaching activities effectively. Traditional learning management processes such as tracking assignments, monitoring participation, and evaluating student performance are often fragmented and inefficient.

The Distance Learning Activity Manager (DLAM) is a web-based application designed to support distance education by managing:

- Courses and learning activities
- Assignment distribution and submission
- Student participation tracking
- Grading and feedback

The system enables students and instructors to interact seamlessly through an online platform, improving accessibility, transparency, and learning outcomes.

3. WORK BREAKDOWN STRUCTURE

This project was developed by a team of three members following an Agile methodology. To ensure efficiency and parallel development, the workload was divided based on functional modules. Each member was responsible for the full-stack implementation (Database, Backend Logic, and Frontend View) of their assigned features.

Nguyễn Đức Thành Công: Team Leader & System Core

Role: Project Setup, Database Architect, Authentication Module.

Responsibilities:

1. **Project Initialization:**
- Set up the Spring Boot project structure and dependencies (Maven, pom.xml).
- Configured the database connection (application.properties) and static resource folders.
2. **Database Design:**
- Designed the initial Entity-Relationship Diagram (ERD).
- Created the base SQL scripts and User entity relationships.
3. **Authentication & User Management:**
- Implemented AuthController for Login, Registration, and Logout logic.
- Developed the "Remember Me" cookie persistence functionality.

- Built the "Profile Management" feature, including the file handling logic for Profile Picture uploads.

4. **Security:**

- Implemented basic session management and role-based access checks (Student vs. Instructor).

Phạm Chấn Quân: Course Management & UI Integration

Role: Backend Developer, Frontend Lead.

Responsibilities:

1. **Course Module:**

- Implemented the Course entity and CourseRepository.
- Built the CourseController to handle CRUD operations (Create, Edit, Delete Course).
- Designed the specific logic for the "Delete Course" cascade, ensuring activities and enrollments are removed safely.

2. **Enrollment System:**

- Implemented the specific logic for Students to Enroll and Unenroll in courses.

3. **Dashboard & UI Engine:**

- Developed the complex logic for the Main Dashboard (/courses endpoint).
- Implemented the filtering logic to separate "My Courses" from "Available Courses" for students.
- Designed the base CSS styles and Thymeleaf layouts to ensure a consistent look and feel across the application.

Nguyễn Hữu Thuỵ: Assessment System & Activities

Role: Backend Developer, Testing.

Responsibilities:

1. **Activity Management:**

- Implemented the Activity entity and controller logic.
- Built the "Create/Edit Activity" forms, including the "Single Submission" constraint logic.

2. **Submission System:**

- Developed the SubmissionController and file upload handling for assignments.
- Implemented the validation logic to check for due dates and enforce deadlines.
- Created the student view for "My Submissions" to track their own progress.

3. **Grading & Feedback:**

- Built the Instructor's "Gradebook" view (/view-submissions).
- Implemented the logic for Instructors to save numeric grades and text feedback for each student.

4. **Resources & Announcements:**

- Implemented the smaller helper modules for Course Resources and Announcements (ResourceController, AnnouncementController).

4. DEVELOPMENT PROCESS

The Agile Scrum methodology is adopted for this project.

Development is carried out in multiple sprints, with each sprint delivering a functional subset of the system.

Key advantages of this approach include:

- Flexibility in handling changing requirements
- Continuous feedback and improvement
- Early delivery of usable features

5. DEVELOPMENT ENVIRONMENT

The Distance Learning Activity Manager is developed as a web application following the MVC architecture.

Core Tech Stack

- **Backend Language:** Java 17 (LTS), Selected for its strong typing, object-oriented features, and long-term support. The project uses modern Java 17 features.
- **Framework:** Spring Boot 3.1.5, Provides a production-ready environment with embedded Tomcat server, dependency injection, and automatic configuration, significantly reducing boilerplate code.
- **Build Tool:** Maven 3.8+, Manages project dependencies (as defined in pom.xml) and handles the build lifecycle (compile, test, package).
- **Database:** MySQL 8.0.33.
- **Version Control:** Git is used for source code management (indicated by the .git directory).
- **API Testing:** Postman or browser-based developer tools were used to verify Controller endpoints.
- **Database Client:** Tools like MySQL Workbench or VS Code extensions were used to verify schema changes and data entry.

II. REQUIREMENT ANALYSIS AND DESIGN

---

1. REQUIREMENT ANALYSIS

Actors: Student, Instructor

Major Use Cases:

- **User authentication:**
  - Users must be able to register with a role (Student or Instructor).
  - Users must be able to login and maintain a session (using HTTP Session and persistent Cookies).
  - Users must be able to logout and clear data.
- **Course enrollment:**
  - Instructors can create, update, and delete their own courses.
  - Students can browse available courses and enroll/unenroll.
  - Users can search for courses by title.
- **Activity creation:**
  - Instructors can add activities with titles, descriptions, due dates, and attachment files to a course.
  - Instructors can set "Single Submission" constraints.
- **Assignment submission:**
  - Students can upload files or write text content to submit an assignment.
  - Students can view their submission history.
  - Instructors can view all submissions for an activity.
- **Grading and feedback:**
  - Instructors can assign grades and write feedback for student submissions.
  - Report generation

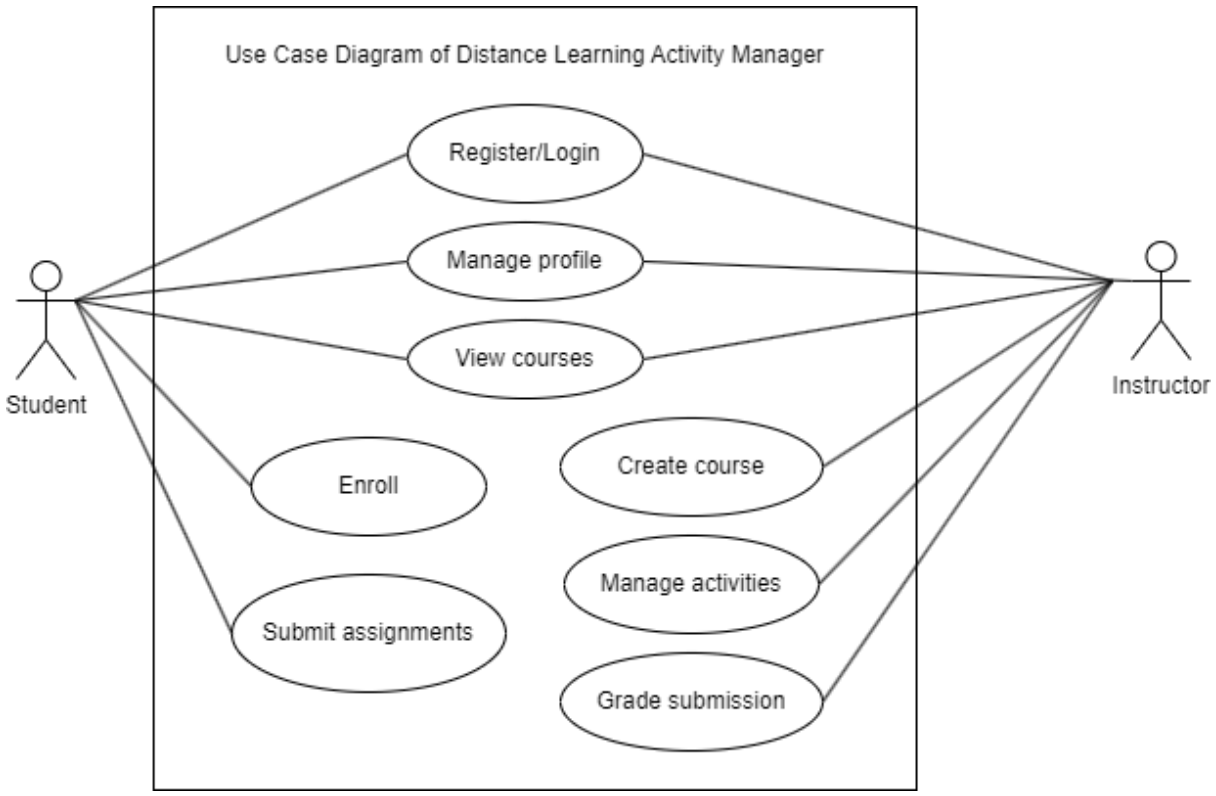


Figure 1

a. Use Case 1: User Login

Actor: Student / Instructor / Administrator

Input: Username, Password

Output:

- Dashboard page (success)
- Error message (failure)

User Story:

As a system user, I want to log in securely so that I can access features according to my role.

b. Use Case 2: Manage Learning Activities

Actor: Instructor

Description:

Create, update, and publish learning activities such as lectures, discussions, and quizzes.

User Story:

As an instructor, I want to manage learning activities so that students can follow the course schedule effectively.

c. Use Case 3: Submit Assignment

Actor: Student

Description:

Upload assignment files before the deadline and view submission status.

User Story:

As a student, I want to submit assignments online so that my work is recorded and graded properly.

d. Use Case 4: Grade and Feedback

Actor: Instructor
Description:
Review submissions, assign grades, and provide feedback.
User Story:
As an instructor, I want to grade assignments and give feedback to support student learning.

e. Use Case 5: Generate Academic Reports

Actor: Administrator / Instructor
Description:
Generate reports on student participation, performance, and course progress.
User Story:
As an administrator, I want reports to evaluate learning effectiveness and academic performance.

2. DESIGN

a. Entity-Relationship Diagram

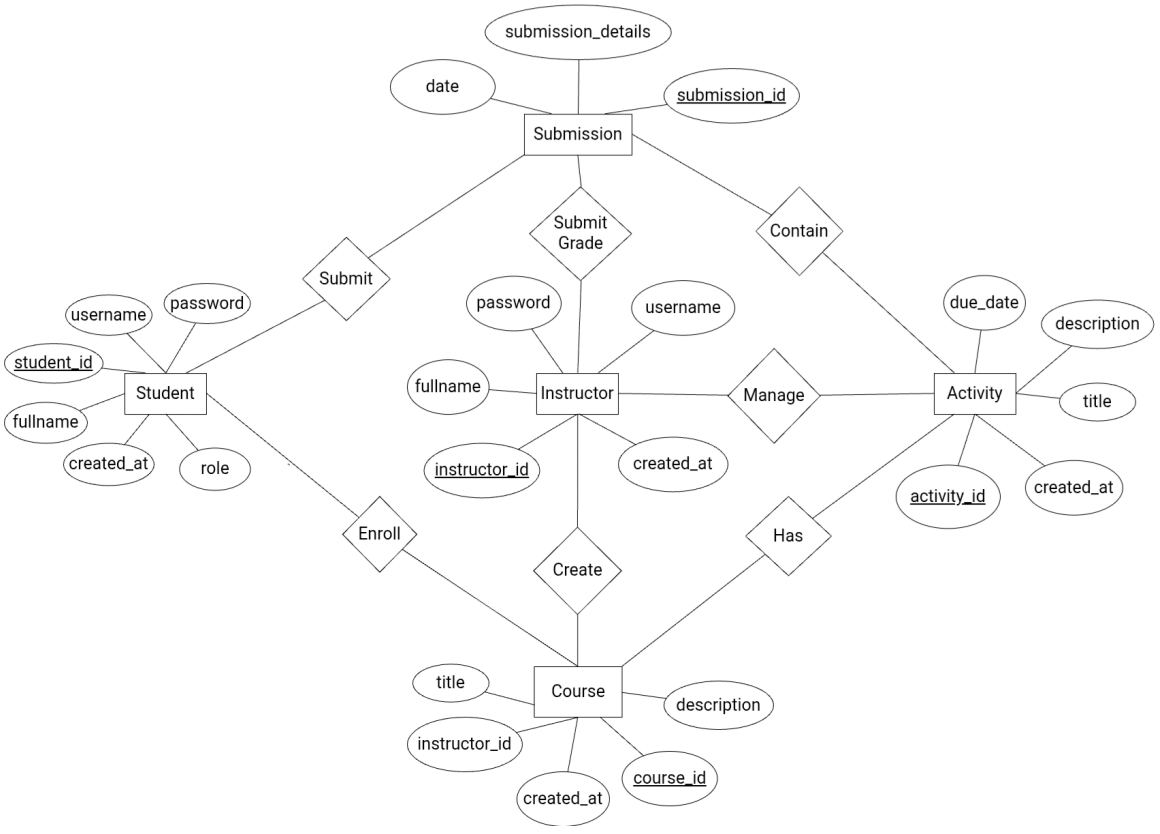


Figure 2

user-schema(id, username, password, full_name, role, created_at)

course-schema(id, title, description, instructor_id, created_at)
from course-schema.instructor_id to user-schema.id

enrollment-schema(course_id, student_id, enrolled_at)
from enrollment-schema.student_id to user-schema.id
from enrollment-schema.course_id to course-schema.id

activity-schema(id, title, description, due_date, created_at, course_id)
from activity-schema.course_id to course-schema.id

submission-schema(id, content, grade, feedback, submitted_at, activity_id, user_id)
from submission-schema.activity_id to activity-schema.id
from submission-schema.user_id to user-schema.id

b. FUNCTIONAL REQUIREMENTS

- The system shall allow users to register and log in securely.
- The system shall support role-based access control.
- The system shall allow instructors to create and manage courses.
- The system shall allow students to enroll and participate in courses.
- The system shall allow assignment upload and deadline enforcement.
- The system shall support grading and feedback.
- The system shall generate academic performance reports.

c. NON-FUNCTIONAL REQUIREMENTS

- Usability
- Easy-to-use interface
- Clear navigation
- Responsive design for multiple devices
- Performance
- Support concurrent users
- Fast response time for submissions and grading
- Security
- Encrypted passwords
- Session management
- Role-based access control
- Maintainability
- Modular code structure
- Well-documented source code
- Scalable database design

d. SYSTEM DESIGN

The system follows a three-tier architecture:

The application follows the Model-View-Controller (MVC) architectural pattern:

- **Model:** POJO classes annotated with @Entity (e.g., User, Course) representing the data structure.
- **View:** Thymeleaf templates (HTML files) responsible for rendering the user interface.
- **Controller:** Spring @Controller classes that handle HTTP requests, process business logic, and determine which View to render.
- **Repository Integration:** Spring Data JPA repositories interface between the Models and the Database.

III. IMPLEMENTATION

---

1. User Account Functions

The user account module serves as the gateway to the application, ensuring that only authorized individuals can access specific resources based on their designated roles (Student or Instructor).

1.1 User Registration

User Perspective (UI Flow): The journey begins at the registration page. A new user is presented with a form requiring personal details such as their full name, a desired username, a secure password, and their role selection (Student or Instructor).

- The user navigates to the "Sign Up" link from the landing page.
- They fill in the required fields. The system provides immediate feedback if a field is left empty.
- Upon clicking the "Register" button, the user waits briefly for processing.
- If successful, the user is redirected to the login page with a success message indicating their account has been created.
- If the username is already taken, the page reloads with an error message prompting the user to choose a different identifier.

Register

Full Name

Nguyen Van A

Username

vana

Password

••••

Role

Student

Register

Already have an account? Login here

Register

Full Name

Tran Van B

Username

vanb

Password

••••

Role

Instructor

Register

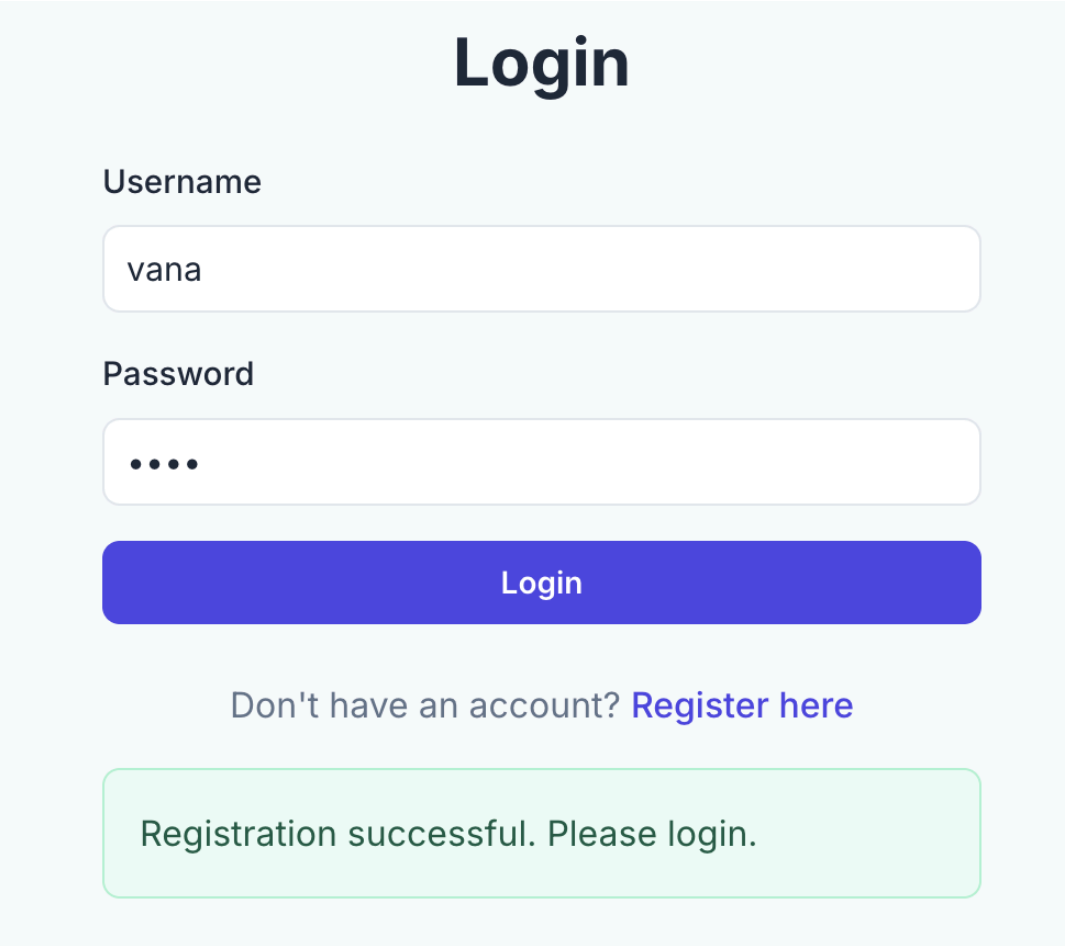
Already have an account? Login here

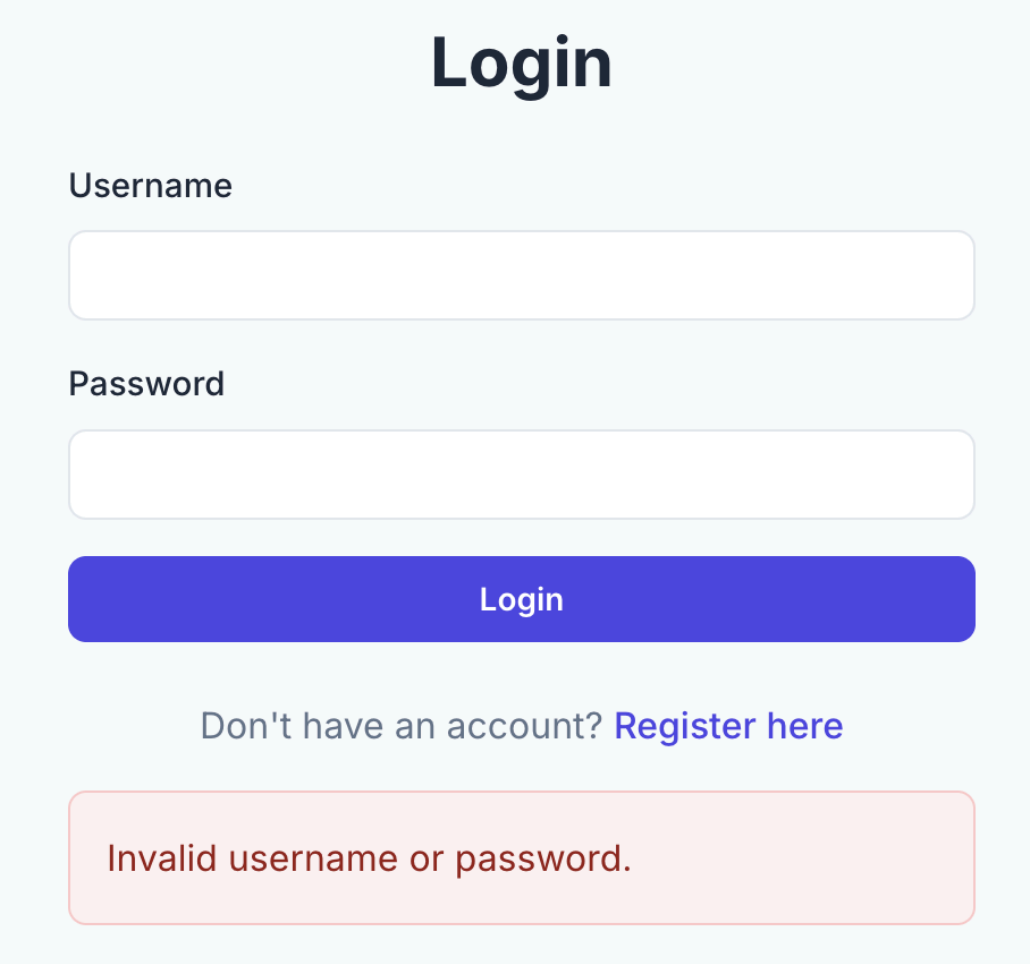
Technical Implementation: When the registration form is submitted, the data is transmitted to the backend server via a secure POST request. The system first sanitizes the input and performs a database query to check for the existence of the provided username. This ensures data integrity and prevents duplicate accounts. If the username is unique, the system creates a new user entity. The password is processed and stored securely. The new user record, including the selected role (which determines future access permissions), is committed to the database. Finally, the controller logic triggers a redirect response to the login route, passing a success flag to trigger the UI notification.

1.2 User Login & Session Management

User Perspective (UI Flow): Registered users access the system via the Login page.

- The user enters their credentials.
- They have the option to check a "Remember Me" box to stay logged in across browser sessions.
- Upon clicking "Login", the system validates their identity.
- Successful authentication redirects the user immediately to their personalized dashboard.
- Invalid credentials result in a reloading of the login page with a visual error indicator.



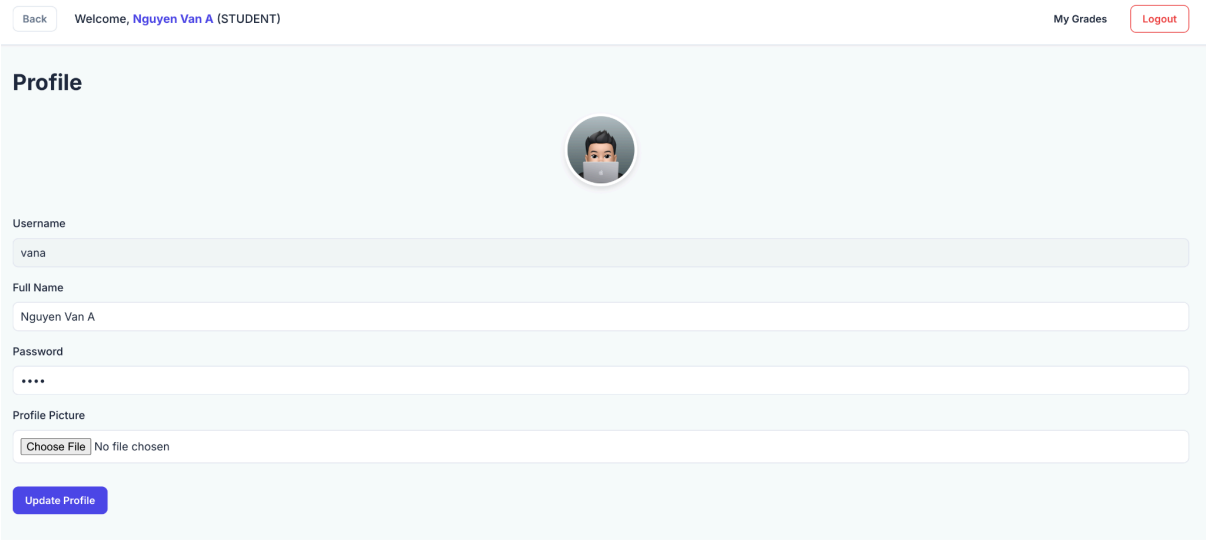


Technical Implementation: The login mechanism relies on verifying the provided credentials against the stored user records. When the login request is received, the backend queries the database for a user matching the input username and password. If a match is found, the system initializes a server-side session. This session object serves as a temporary storage container for the user's profile and role information, following the user as they navigate different pages. This eliminates the need to re-authenticate for every request. If the "Remember Me" option was selected, the system generates a persistent HTTP cookie containing a unique identifier. This cookie is sent to the user's browser, allowing the system to automatically recognize and re-authenticate the user on subsequent visits even if the browser has been closed.

1.3 Profile Management

User Perspective (UI Flow): From the dashboard, users can access their profile settings.

- The user clicks on their avatar or name in the navigation bar.
- The "My Profile" page displays their current information (Name, Role).
- Users can update their display name or change their password.
- Crucially, users can upload a profile picture. They select an image file from their local device and click "Update".
- The page refreshes to confirm the changes, showing the new profile image immediately.



Technical Implementation: Profile updates involve handling multipart form data, specifically for image uploads. When a user uploads a picture, the backend receives the binary file data. To prevent file name conflicts (e.g., multiple users uploading "profile.jpg"), the system generates a universally unique identifier (UUID) and appends it to the original filename. The file is then saved to a dedicated directory on the server's file system. The user's database record is updated to point to this new file path. When the profile page is rendered, the system effectively links the user entity to the static file resource, displaying the image.

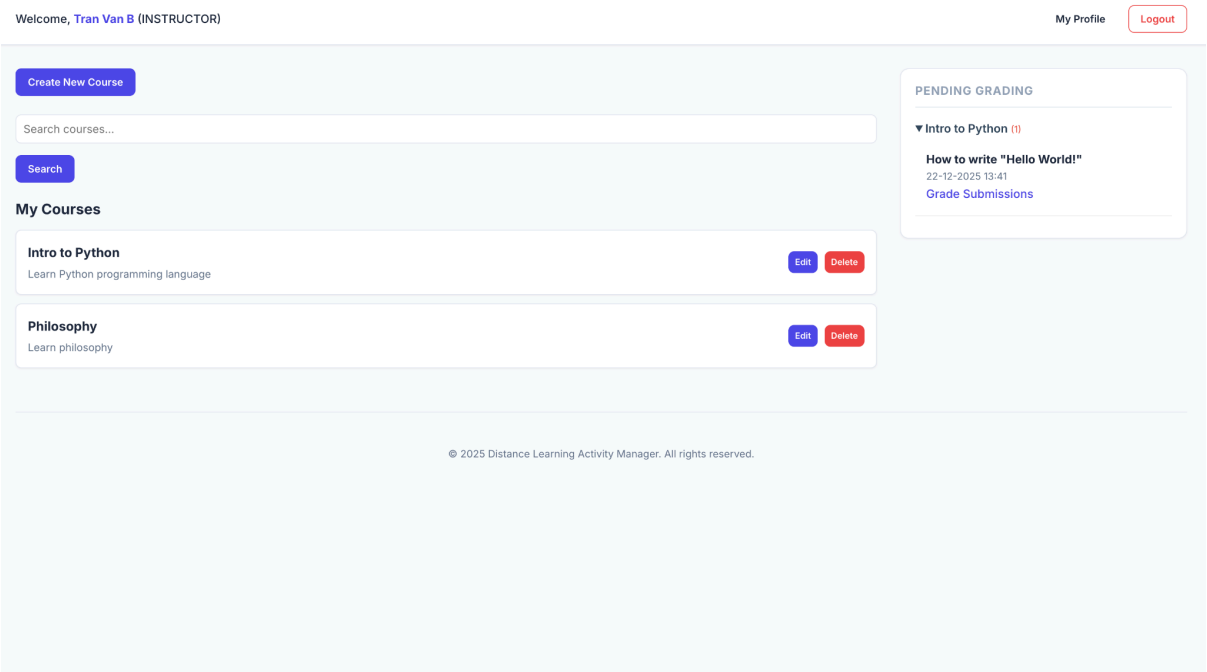
2. Course Viewing and Management Functions

The core educational content is organized into "Courses". The experience differs significantly depending on whether the user is an Instructor (Creator) or a Student (Consumer).

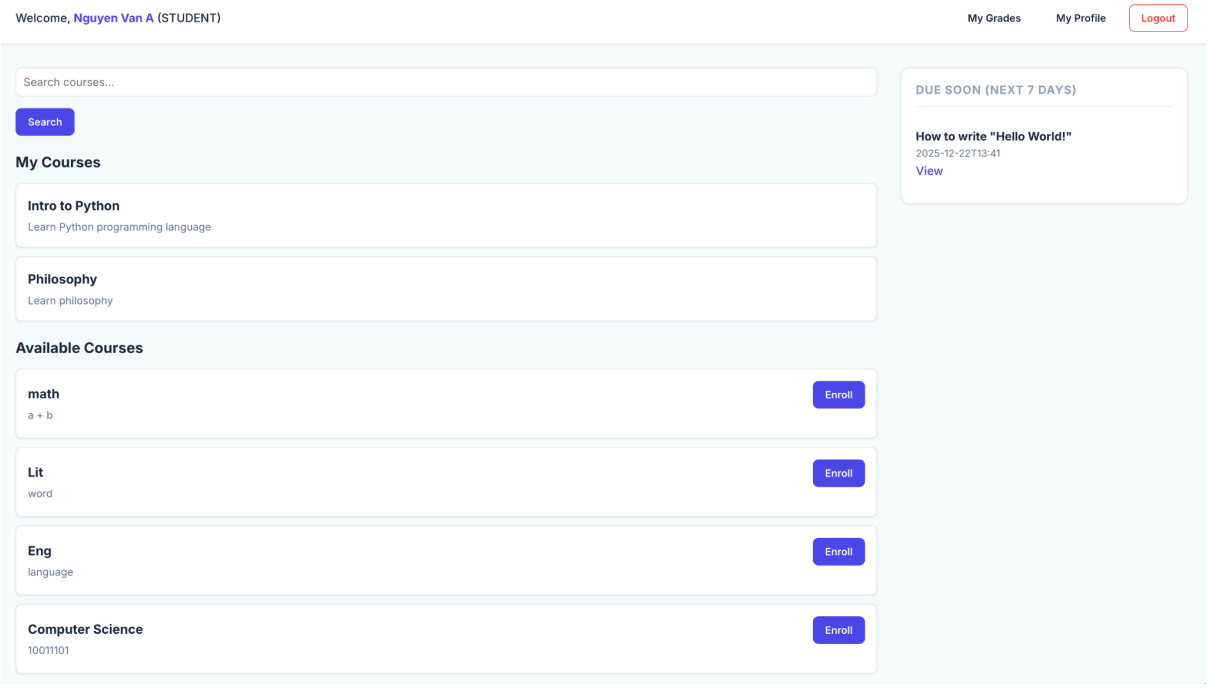
2.1 Dashboard & Course Listing

User Perspective (UI Flow): After logging in, the user lands on the Dashboard.

- **For Instructors:** The view is an administrative command center. They see a grid of courses they have created. A generic search bar allows them to filter their courses by title.



- **For Students:** The view is split into two sections: "My Enrolled Courses" and "Available Courses". This clear separation allows students to distinguish between their active classes and potential new electives.



- Widgets on the side display "Upcoming Deadlines" (for students) or "Pending Grading" (for instructors), providing immediate context on urgent tasks.

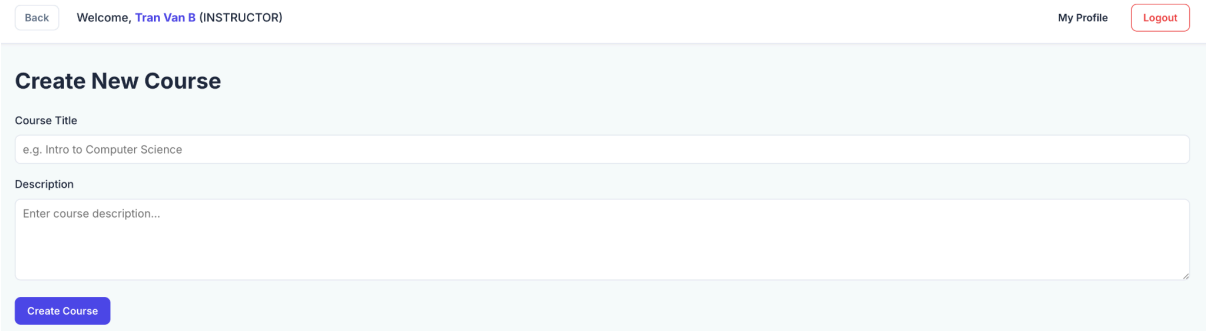
Technical Implementation: This dynamic dashboard is built by querying multiple data repositories based on the user's ID and role stored in the session.

- **Student Logic:** The system performs a complex join operation. It first retrieves the list of courses the student is enrolled in from the enrollment table. It then fetches all available courses and filters them into two lists: one containing the IDs found in the enrollment list, and another containing the rest.
- **Instructor Logic:** The system executes a simpler query to fetch only the courses where the "Creator ID" matches the current logged-in user. The resulting data lists are injected into the HTML template, which allows the frontend to loop through and render the course cards dynamically.

2.2 Course Creation (Instructor Only)

User Perspective (UI Flow): Instructors have a persistent "Create Course" button.

- Clicking it opens a dedicated form.
- The instructor enters a Course Title and a detailed Description.
- Upon submission, the new course appears immediately on their dashboard.

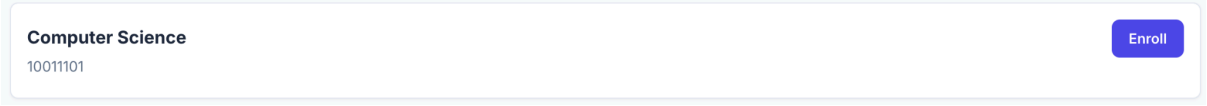


Technical Implementation: This function is strictly role-protected. The backend strictly checks the session ensuring the user has the "Instructor" role before processing the request. The submitted data is instantiated into a new Course object, associated with the instructor's ID, and persisted to the database.

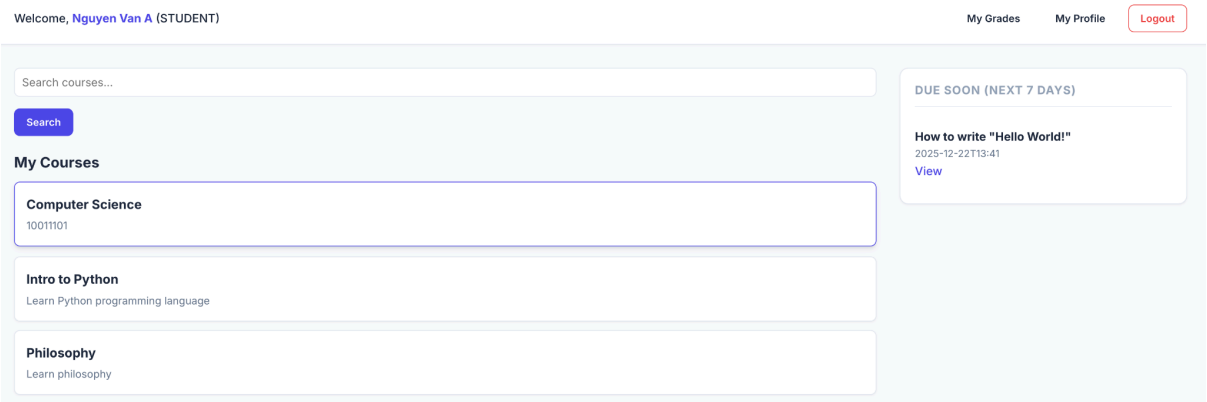
2.3 Enrollment System (Student Only)

User Perspective (UI Flow): In the "Available Courses" section, students see an "Enroll" button on each course card.

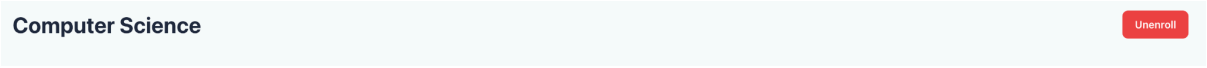
- Clicking "Enroll" triggers a request.



- The page refreshes, and the course moves from the "Available" list to the "My Courses" list.



- Conversely, enrolled courses have an "Unenroll" button, allowing students to drop a class if needed.



Technical Implementation: The enrollment logic manages a many-to-many relationship between Users and Courses. When a student enrolls, the system creates a new record in the junction table (Enrollments) linking the specific Student ID and Course ID. The backend includes validation to prevent double-enrollment (e.g., checking if the record already exists before saving). Unenrolling simply deletes this specific relationship record.

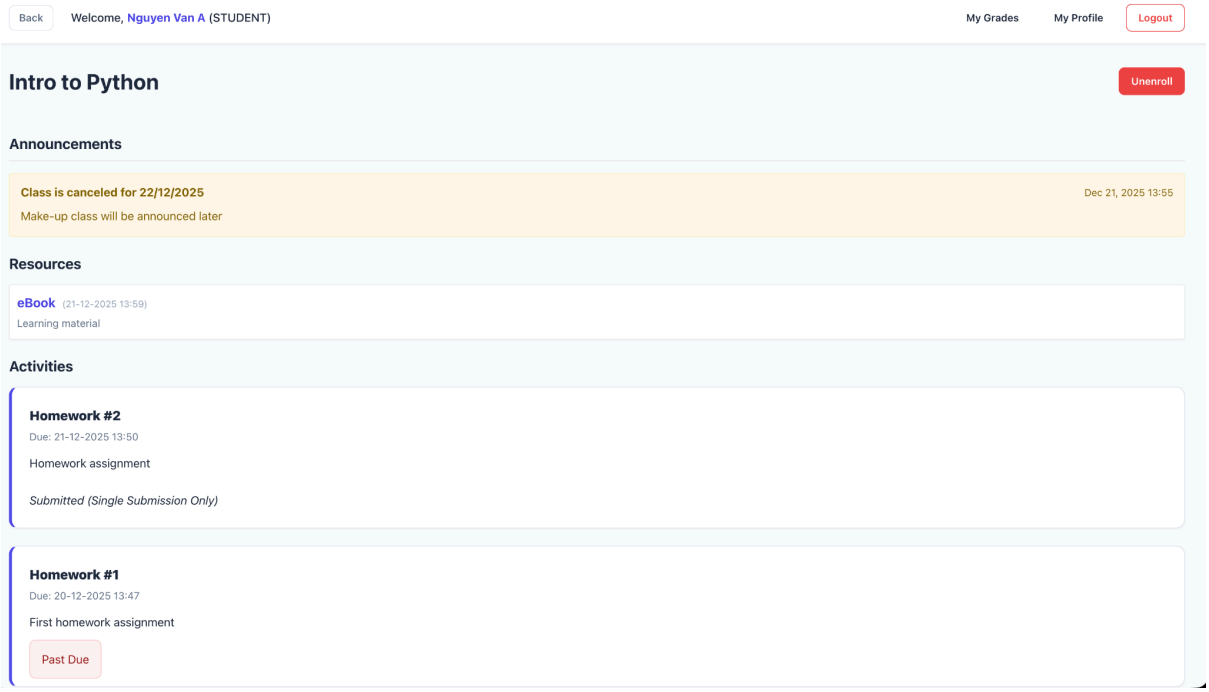
3. Activity Submission

The heart of the learning process involves students interacting with Activities (assignments) created by instructors.

3.1 Viewing Activities and Resources

User Perspective (UI Flow): Clicking on a course card takes the user to the Course Details page.

- The page is organized into sections: Announcements, Learning Resources, and Activities.
- Activities are listed in chronological order. Each activity card shows the title, description, and status (e.g., "Submitted").
- Different colors or icons indicate whether an activity is overdue or completed.



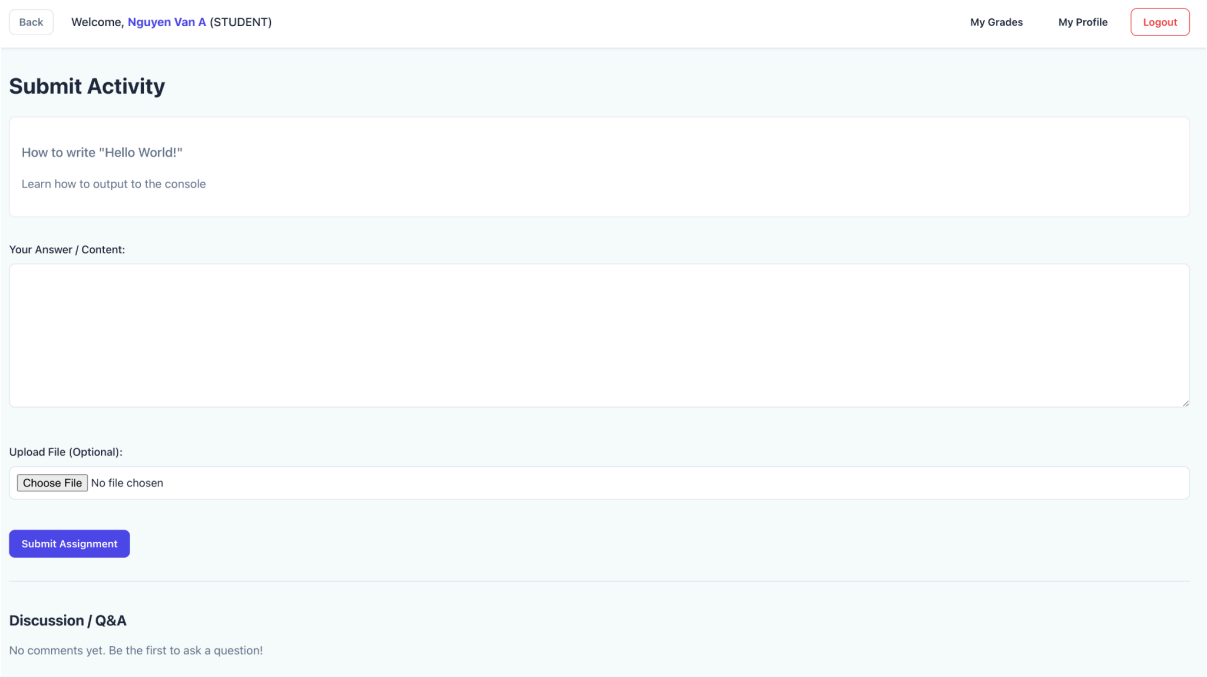
Technical Implementation: This page acts as an aggregation point. The backend performs multiple parallel queries:

- Fetching the Course details.
- Fetching all Activities linked to this course.
- Fetching Resources and Announcements.
- **For Students:** It also queries the Submission repository to determine which activities have already been completed. This list of "submitted activity IDs" is passed to the view to conditionally render "Completed" badges or "Submit" buttons.

3.2 Assignment Submission

User Perspective (UI Flow): When a student selects an activity to complete:

- They are taken to a Submission Interface.
- They see the activity instructions and any attached files from the instructor.
- The submission form allows two modes of input: a text box for written answers and a file picker for uploading documents (PDFs, request forms, zip files).
- After clicking "Submit Assignment", the system uploads the work.
- The user is redirected back to the course page, where the activity now shows a "Submitted" status.



Technical Implementation: This requires robust file handling. When the form is submitted:

- **Validation:** The system checks if the activity is past its due date. If so, it rejects the submission logic to enforce deadlines. It also checks if the activity allows single or multiple submissions.
- **File Processing:** Similar to profile pictures, uploaded assignment files are renamed with a unique ID to prevent overwrites and stored securely.
- **Data Persistence:** A new Submission record is created, capturing the timestamp, the text content, the file path (if applicable), the Student ID, and the Activity ID. This record is the permanent proof of the student's work.

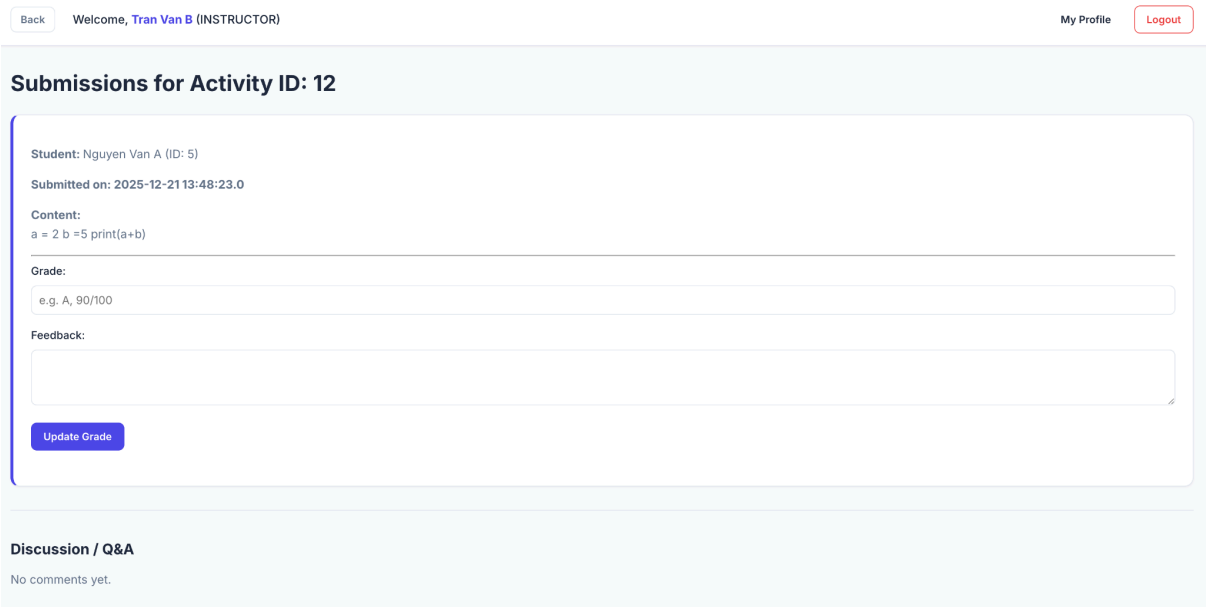
4. Grading

The final phase of the workflow is the assessment of student work by the instructor.

4.1 Viewing Submissions

User Perspective (UI Flow): Instructors can click on each activity card in their course view and see the submissions.

- Clicking this opens a table listing all students who have submitted work for that specific activity.
- The table displays the student's name, submission time, and links to download their attached files.
- There is a status column showing whether the work has been graded yet.



Technical Implementation: The backend retrieves all submission records linked to the specific Activity ID. It often performs a "fetch join" or a secondary query to resolve the User ID in the submission to a human-readable "Student Name" to display in the table. The list is sorted, typically by submission date, to help instructors organize their grading workflow.

4.2 Grading and Feedback

User Perspective (UI Flow): Under each submission is a "Update Grade" action.

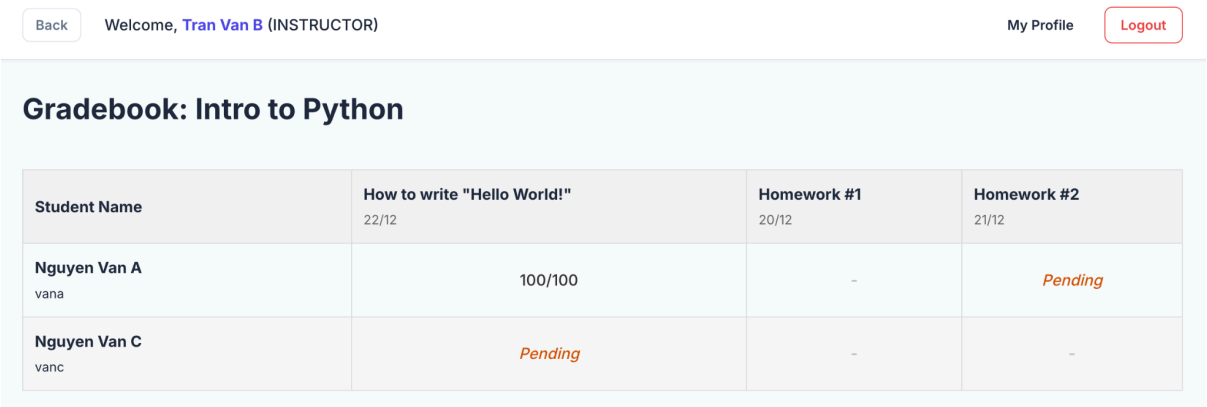
- The instructor enters a score (e.g., "A", "85/100") and qualitative feedback in a text area.
- Upon saving, the page updates.
- The student, upon logging in next, can see this grade and feedback on their submission history page.

Technical Implementation: This is an update operation on the existing Submission entity. The system takes the Grade string and Feedback string provided by the instructor and updates the corresponding fields in the database record. This closes the feedback loop, making the data immediately available to the student view via the previously described submission retrieval logic.

4.3 Instructor Gradebook

User Perspective (UI Flow): For a comprehensive view of student performance, instructors can access the "Gradebook" tab within a course.

- The view presents a matrix (grid) layout. Rows represent enrolled students, and columns represent the course activities.
- Each cell displays the grade for that specific student and assignment. Empty cells indicate missing work.
- This high-level overview allows instructors to quickly identify at-risk students who are falling behind across multiple assignments.



Technical Implementation: Generating this grid requires a computationally intensive process handled by the GradebookController.

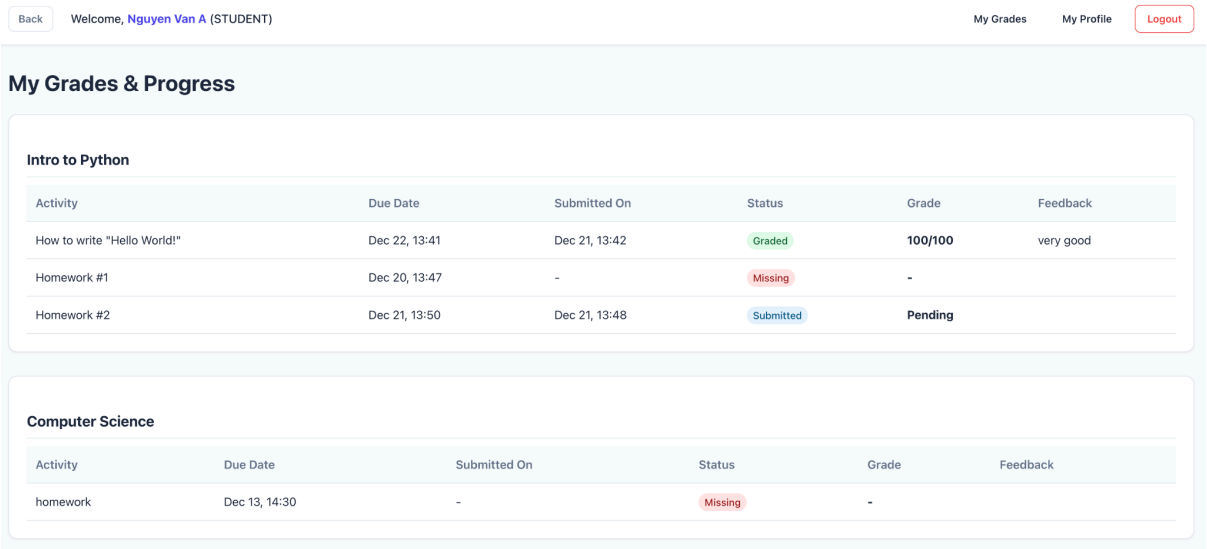
- The backend fetches three distinct lists: all Activities for the course, all Enrolled Students, and all Submissions.
- It then constructs a nested Map structure (Map<Student.Id, Map<ActivityId, Submission>>).
- The controller iterates through every student and maps their submissions to the corresponding activity IDs. This data structure allows the frontend template to render a dynamic table, iterating rows for students and columns for activities, simply by looking up the value in the map for each cell coordinate.

5. Student Progress Tracking Functions

My Grades Page

User Perspective (UI Flow): Students have a dedicated "My Grades" page accessible from the main navigation.

- This page aggregates performance data across all enrolled courses, not just one.
- The view is grouped by Course Name. Under each course header, a list of assignments is shown.
- Each item displays the Activity Title, Due Date, Submission Status (e.g., "Submitted", "Missing", "Pending"), and the Grade/Feedback if available.
- "Missing" items (past due date with no submission) are effectively highlighted to prompt immediate action.



Technical Implementation: The ProgressController orchestrates this logic.

- It first retrieves the list of courses the student is enrolled in.
- It then fetches all activities associated with those course IDs.
- Simultaneously, it fetches the student's entire submission history.
- The core logic involves a correlation step: It iterates through every assigned activity and checks against the submission history.
    - If a submission exists and has a grade, status is "Graded".
    - If a submission exists but no grade, status is "Submitted".
    - If no submission exists and the current time is past the due date, status is computed as "Missing".
- This processed list of "StudentGradeDTO" objects is passed to the view, providing a computed, real-time snapshot of academic standing.

IV. DISCUSSION AND CONCLUSION

---

1. Conclusion

The Distance Learning Activity Manager (DLAM) project has successfully delivered a functional web-based platform that addresses the core needs of remote education: centralized course management, streamlined assignment submission, and effective communication between instructors and students.

By leveraging the Spring Boot framework and MVC architecture, the team created a modular and maintainable system.

- **For Instructors**, the application reduces the administrative burden of tracking student progress via email or disjointed tools. The "Create Course" and "Gradebook" features provide a clear workflow for content delivery and assessment.
- **For Students**, the dashboard offers an organized view of their academic responsibilities, removing ambiguity about deadlines and submission status.

The project not only met the initial functional requirements—Authentication, Course CRUD, and File Submissions—but also implemented a responsive UI using Thymeleaf and Bootstrap, ensuring accessibility across different devices.

2. Limitations

Although the project is functional, several limitations were identified during the development and testing phases, which should be addressed in a production environment:

- **Security Implementation:** Passwords are currently stored, but the implementation details of hashing encryption were simplified for this prototype level.
- **Scalability of File Storage:** The system stores uploaded files (profile pictures, assignment submissions) directly on the local server's file system (uploads/ directory). This approach is not scalable for a cloud deployment (e.g., Heroku, AWS), as these files would be lost during server restarts or deployments.
- Search and Discovery:
- **Real-time Interaction:** The application relies on page refreshes to update state. For example, if an instructor posts an announcement, a student currently viewing the course page will not see it until they refresh the browser.

3. Future Work

To evolve DLAM from a prototype into a production-ready system, the following enhancements are proposed:

- **Cloud Storage Integration:** Replace the local file I/O logic with an object storage service like AWS S3 or Google Cloud Storage. This will ensure data persistence and better performance for large file uploads.
- **Advanced Analytics Dashboard:** Develop an "Admin" view that aggregates system-wide data, such as "Total Active Users," "Submission Rates per Course," and "Average Grades." This would provide valuable insights for educational administrators.
- **Notification System:** Implement an email or in-app notification system. Students should receive alerts when a grade is posted or a deadline is approaching (e.g., "Assignment due in 24 hours"). This could be achieved using JavaMailSender and scheduled tasks (@Scheduled).

V. REFERENCES

Agile Methodology – https://agilemethodology.org
Scrum Framework – https://www.scrum.org
Spring Boot Documentation: https://spring.io/projects/spring-boot
Thymeleaf Documentation: https://www.thymeleaf.org/documentation.html
MySQL Connector/J Developer Guide: https://dev.mysql.com/doc/connector-j/8.0/en/
Bootstrap Documentation: https://getbootstrap.com/docs/5.3/getting-started/introduction/