CROSS-PLATFORM MOBILE APPLICATION DEVELOPMENT - 502071

SEMESTER 1 – ACADEMIC YEAR 2025 – 2026

Lecturer: Mai Van Manh

FINAL PROJECT

E-learning Management App

For content highlighted in **purple**, students must adhere strictly to the descriptions provided. Any deviation, modification, or addition of features will result in the submission being disqualified from evaluation.

I. OVERVIEW

The final project requires each team to develop a cross-platform E-learning application using Flutter, inspired by the design and interface organization of **Google Classroom** but tailored to the course scope. The application operates in a single context: the system supports only two roles—Instructor (a single user, also acting as the content administrator, with the mandatory and fixed account credentials admin/admin) and Student (multiple users). No additional roles are permitted. All academic content, courses, and data must pertain exclusively to the Faculty of Information Technology (e.g., programming, databases, AI). Any irrelevant content (e.g., cooking, sports, arts) will result in a score of 0 and the submission will not be graded.

The application must fully reflect the operational lifecycle of a digital classroom: the instructor defines semesters, creates courses, assigns groups, and manages students; the instructor publishes and distributes learning resources (announcements, assignments, quizzes, materials); the system tracks learning activities to generate statistics, reports, and CSV exports for end-of-semester evaluations; students receive notifications, track progress, participate in discussions, and submit work by deadlines. Beyond functional correctness, the submission must demonstrate practical deployability: it must include an Android APK (arm64), Windows executable (64-bit) or macOS application, and a publicly accessible web version.

During grading, the web application must operate smoothly, with teams responsible for resolving any cold start issues with free backend services—proactively prepare wake-up scripts for the grading session.

II. PROJECT REQUIREMENTS

1. Interface and User Experience Design

The application should organize its interface into three distinct layers:

- Homepage with Role-Based Context:
  - For students, the homepage displays enrolled courses as cards, each featuring a cover image, course name, instructor name, and other relevant details.
  - For the instructor, the homepage transforms into a dashboard summarizing key metrics for the current semester: number of courses, groups, and students managed; number of assignments and quizzes; and progress charts for quick insights.
  - Both roles have a semester switcher in a convenient interface location (student-defined). By default, the system loads the current (latest) semester; users can switch to past semesters for reference. For students, past semesters are read-only, prohibiting assignment submissions or quiz attempts.
- Course Space with Three Tabs:
  Each course is organized into three tabs:
  - Stream: Displays recent announcements with short comment threads for quick interaction.
  - Classwork: Centralizes assignments, quizzes, and materials, organized systematically with search and sorting capabilities for large datasets.
  - People: Lists groups and students enrolled in the course.
  - Students can view all three tabs but cannot message other students directly; valid interaction channels include course/group forums and private messaging with the instructor.
- User Profile with Avatar: Both instructor and students have a Profile page to view and edit basic information (e.g., avatar, supplementary fields), but display names cannot be modified. Usernames must be real names (e.g., no “user1”, “user2”).

2. Semester–Course–Group–Student Model

The instructor is responsible for CRUD operations on core entities:

- Semester: Requires only a code and name, encompassing multiple courses.
- Course: Includes a course code, name, number of sessions (10 or 15), and belongs to a single semester.
- Group: Belongs to a single course in a specific semester; within a course, each student can belong to only one group (e.g., Web Programming & Applications course has Group 1, Group 2, Group 3).

- Student: Managed independently; the instructor creates student accounts first, then assigns them to appropriate groups. For example, in a new semester, a list of 50 students may include 40 existing accounts and 10 new ones; the system must support this process seamlessly and efficiently.

To enhance operational efficiency, all bulk import features must support CSV uploads with validation and preview. For instance, when uploading a 50-student CSV, the system must not reject the entire batch due to duplicates; instead, it should display a preview listing all 50 entries with their status ("already exists," "will be added"), allowing the user to proceed with importing only the 20 new students while skipping the 30 existing ones. A post-import results screen must show the status of each item. This preview rule applies similarly to semesters, courses, groups, and student-group assignments.

3. Distribution of Learning Content

The instructor creates and publishes four types of content, each with distinct behaviors, scopes, and tracking:

- **Announcement**: Includes a title, rich-text content, and optional file attachments. When published, the instructor selects the scope (one, multiple, or all groups in a course). Both instructor and students can comment briefly under announcements in a simplified "social media" style. The instructor can track who has viewed the announcement and, if applicable, who has downloaded attached files.
- **Assignment**: Includes a title, description, and multiple file/image attachments. The instructor sets the start date, deadline, whether late submissions are allowed (with a specified late deadline), maximum submission attempts, file format, and size limits. Assignments are scoped to groups, similar to announcements. The instructor's interface must provide real-time tracking: who has submitted, who hasn't, late submissions, multiple attempts (e.g., 2nd or 3rd submission), and current grades. All tracking tables must support filtering, searching, and sorting by name, group, time, or status, with CSV export for individual assignments or all assignments in a course/semester for final evaluations.
- **Quiz**: Built on a course-specific question bank reusable across semesters. Each question has multiple choices, one correct answer, and a difficulty label (easy, medium, hard). When creating a quiz, the instructor configures the time window (open/close), number of attempts, duration, and can randomly select a structure (e.g., x easy, y medium, z hard questions). Post-release, the system must show who has completed the quiz, who hasn't, scores, submission times, and support CSV export for individual quizzes or all quizzes in a course/semester.
- **Material**: Includes a title, description, and one or more files/links. Unlike announcements/assignments, materials are automatically visible to all students in a course, with no group scoping. The instructor can track who has viewed or downloaded materials.

4. Interaction Rules, Forums, Messaging, and Notifications

- **Forum**: Allows topic creation per course, with all enrolled students having equal discussion rights, threaded replies, file attachments, and search functionality.

- **Private Messaging**: Exists only between students and the instructor; direct student-to-student messaging is prohibited—students must use course/group forums for peer interaction.
- **Notifications**: The system implements in-app notifications for students with clear read/unread status; instructors do not require in-app notifications. Email notifications (for students only) must be sent for key events, such as new announcements, approaching assignment/quiz deadlines, important feedback, or confirmation of assignment/quiz submissions.

5. Search, Sorting, and Performance Optimization

Due to potentially large datasets, list screens (courses, groups, students, assignments, quizzes, materials, submissions) must support keyword search, filtering (e.g., by group, status, or time), and sorting (e.g., by name, deadline, score, or update date). The application should implement appropriate caching (e.g., caching "category" data and recent query results) to reduce API calls, minimize response times, and maintain a smooth experience under unstable network conditions. When switching semesters or refreshing data, cache synchronization must ensure no data inconsistencies.

6. Student Privileges, Personal Dashboard, and Action Restrictions

Beyond accessing resources, students must have a personal dashboard displaying learning progress: submitted, pending, or late assignments; completed quizzes with scores (if available); and a chart/timeline of upcoming deadlines. For past semesters, all actions (submitting assignments, taking quizzes, editing) are disabled, allowing only read-only access for reference.

7. Deployment

For the backend of the application, you have the freedom to decide how to implement it. However, for the front end (Flutter), you need to provide the following: an APK version for Android (arm64), a 64-bit version for Windows/macOS, and deploy the web version to an online hosting service (e.g., Firebase Hosting or GitHub Pages). Afterward, you should provide the URL of the website in your submission. The web deployment section, if successful, will earn you 0.5 points. Although the Windows/macOS and APK versions do not contribute to the score, they are still mandatory. If you only submit the source code without providing any working deployment versions, or if they are faulty and cannot run, your submission will not be graded.

8. Other requirements

- UI/UX: The web app must have a clear, user-friendly design with intuitive navigation. Focus on user experience, quick load times, and easy interaction with elements.
- Responsive Design: The web app should be responsive, adapting seamlessly to different devices and screen sizes. Use frameworks like Bootstrap or CSS Grid to ensure compatibility.

- Team Collaboration: Team members must work together using version control (e.g., Git), dividing tasks and ensuring smooth integration of contributions. Regular communication is essential. You are required to demonstrate the group work process by capturing screenshots of team members' contributions using the **GitHub Insights** feature. The evidence must clearly show that the project has been ongoing for at least one month from its start date, with each member making a minimum of two commits per week. Teamwork is **mandatory**, so if the team has only one member or lacks evidence of collaboration, **0.5** points will be deducted from the total score.
- Offline Capability: The application must support an offline mode to ensure functionality when the user has no internet connection. At a minimum, students should be able to view previously accessed course materials, announcements, and their personal dashboard (including submitted assignments, completed quizzes with scores, and upcoming deadlines). The instructor should be able to view previously accessed course data, student lists, and tracking metrics (e.g., who has viewed materials or submitted assignments). To achieve this, the application must integrate both an online database and an offline database (e.g., SQLite, Hive). The offline database should maintain a synchronized copy of critical data from the online database, enabling faster access or serving as a fallback when the internet is unavailable.

9. Bonus features

To encourage technical depth, teams may implement the following extensions, with descriptions and evidence placed in a bonus/ directory. Each well-executed feature earns 0.25–0.5 points, with a maximum bonus of 4 features:

- Self-built backend (without using Firebase or similar services).
- Microservices architecture with message queues (e.g., RabbitMQ/Redis) and/or Kubernetes for deployment.
- AI chatbot for learning support.
- AI-driven question and answer generation for quizzes from instructor materials, with parameterized difficulty and minimal validity checks.
- Other AI features that enhance learning or teaching (with quantitative/qualitative evidence and clear descriptions, prominently showcased in the final demo video; final bonus points are at the instructor's discretion).
- Publishing the application on Google Play, Apple App Store, or Microsoft Store.

How to determine bonus points (0.25 vs. 0.5): For example, for the AI chatbot, simply calling OpenAI/Gemini APIs without customization, contextualization, or specialized testing for learning support earns only 0.25 points. To achieve 0.5 points, the chatbot must be designed and validated for the specific context (e.g., using RAG or fine-tuning the model for learning support, with evidence via specific examples or benchmarks).

If your team successfully implemented any of these bonus features, you should clearly state it in the self-evaluation form, the README file, the product introduction video along with specific, convincing evidence. This ensures that your contributions are recognized and properly communicated to reviewers.

10. Notes

The description provided above is intended as a general guideline and cannot be considered as a detailed step-by-step implementation for each feature, specifying what is right or wrong. However, during the grading process, the features must be developed based on these descriptions, particularly those highlighted in a distinct color. These features must be implemented to a relatively high standard in order to qualify for maximum points. Teams should actively refer to related applications and apply their daily user experience with such applications to the task. For example:

- **When displaying a list of courses (or other resources)**:
  - **Poor approach**: Only display the information available in the Course table of the database, even showing the createdAt field in a raw machine-like format (e.g., 2025-08-16T13:45:30.000Z).
  - **Better approach**: Display not only the information from the Course table with a user-friendly format (e.g., 15/08/2025), but also include related information that enhances usability (e.g., the number of groups belonging to the course, the total number of students enrolled in the course).
- **When displaying a list of courses (or other resources)**:
  - **Poor approach**: Always show a loading screen while fetching data from the API, even when the user revisits the same screen. This forces repeated waits and unnecessary API calls.
  - **Better approach**: Use a user-friendly loading indicator (e.g., skeleton screen) combined with local caching, so data only needs to load once and is refreshed only when updated. This reduces waiting time and API usage.
- **When adding a new student**:
  - **Poor approach**: Only allow adding through a form or uploading a CSV in a simplistic way. If the CSV happens to be in the correct format, it uploads successfully; if not, the outcome is uncertain. After a successful upload, the system merely displays a generic message such as “Upload completed.”
  - **Better approach**: Support adding students via both the form and CSV upload. On the CSV upload interface, provide clear instructions so users know the correct column order and whether headers are required. Once a file is selected, show a preview of the parsed results: highlight missing or incorrectly formatted fields, and list which students already exist and which do not. Allow the user to proceed only if the file is valid. Upon confirming the upload, create accounts only for students who do not already exist, skip duplicates, and display a summary screen so the user clearly understands the import results.
- **When attaching image files**:
  - **Poor approach**: After selecting, the system only indicates that one image has been chosen, without letting the user verify whether the correct file was selected.

- **Better approach**: For formats that can be previewed (e.g., images, text files, etc.), allow the user to view them directly in the app by clicking on the file, instead of forcing them to download it first. Furthermore, if multiple images are included, enable swipe navigation to quickly move between files, rather than requiring the user to close one image and click another to view it.

- **Other examples**: students may look up and study further by themselves.

II. RUBRIK

Category	Points	25% (Low Achievement)	25%–75% (Moderate Achievement)	75%–100% (High Achievement)
Semester, Course, Group, and Student Management	**2.0**	Basic CRUD operations are implemented for only 1–2 entities (e.g., semester or course), with significant errors (e.g., unable to add students or groups). CSV upload is either missing or non-functional, failing to import data correctly. Student management lacks independent account creation, and group assignment is broken or causes crashes. User interface for these operations is confusing or incomplete, making it unusable for practical purposes.	CRUD operations are implemented for most entities (semester, course, group, student), but with noticeable limitations or errors (e.g., CSV import processes data incorrectly or lacks preview functionality). Student management supports independent account creation, but group assignment may have minor issues (e.g., duplicate entries or slow processing). CSV upload works partially, with preview or import functionality incomplete, causing occasional errors. Interface is functional but may have minor usability issues, such as unclear error messages or slow response times.	Full CRUD operations are seamlessly implemented for all entities (semester, course, group, student) with no errors. CSV upload includes robust preview and smart import functionality, correctly handling duplicates and providing clear feedback on import status. Student management allows independent account creation and efficient group assignment, with intuitive workflows. Interface is user-friendly, with clear navigation, error handling, and fast response times, ensuring practical usability for both instructors and students.
Content Delivery	**2.0**	Only 1–2 content types (e.g., announcement or assignment) are partially functional, with critical errors (e.g., unable to attach files or scope content to groups). Question bank for quizzes is either missing or unusable, with no search or difficulty tagging. View/download tracking is non-existent, and content delivery fails to meet basic requirements (e.g., materials not accessible to students). Interface for content management is broken or severely limited, preventing practical use.	3–4 content types (announcement, assignment, quiz, material) are implemented, but with missing secondary features (e.g., no view/download tracking, limited question bank search, or partial group scoping). File attachments work but may have errors (e.g., size limits not enforced). Question bank supports basic tagging but lacks advanced features like cross-semester reuse. Interface is functional but has minor issues, such as slow loading of content or inconsistent display of attachments.	All 4 content types (announcement, assignment, quiz, material) are fully functional, with seamless group scoping, file attachments, and view/download tracking. Question bank is robust, supporting search, difficulty tagging, and cross-semester reuse. Content delivery is reliable, with clear options for instructors to set deadlines, file formats, and submission limits. Interface is intuitive, with fast loading, clear organization, and no errors, ensuring a smooth experience for all users.

Interaction and Notifications	**2.0**	Interaction features (forum, private chat) or notifications (in-app/email) are largely non-functional or have critical errors (e.g., students can chat with each other, violating rules, or email notifications fail to send). Forum lacks threading or file attachment capabilities, and in-app notifications are either missing or do not display read/unread status. System is unusable for communication, with frequent crashes or incorrect behavior.	Forum and private chat (student-instructor only) are implemented but lack key features (e.g., forum search is missing, or email notifications are inconsistent for events like deadlines). In-app notifications work but may have issues, such as missing read/unread status or delayed updates. Interaction features are functional but have minor errors, such as slow loading of forum threads or occasional notification failures. Interface is usable but may cause confusion due to inconsistent behavior.	Forum and private chat (student-instructor only) are fully functional, with threaded discussions, file attachments, and search capabilities in the forum. In-app notifications are reliable, with clear read/unread status and real-time updates. Email notifications are sent consistently for key events (e.g., new announcements, deadlines, submission confirmations) with no failures. Interface is intuitive, with fast and error-free interaction, ensuring effective communication for all users.
Reports and Analytics	**2.0**	Dashboard is either missing or severely limited, showing no meaningful metrics for instructors or students (e.g., no progress tracking or assignment status). CSV export is non-functional or produces unusable files (e.g., incorrect data or formatting). Search/filter/sort features are absent, and progress charts are either missing or broken, making analytics unusable for practical purposes.	Dashboard is implemented but lacks some metrics (e.g., incomplete progress tracking or missing charts for instructors). CSV export works but has minor errors (e.g., missing columns or incorrect data for some entries). Search/filter/sort features are present but slow or incomplete (e.g., limited filtering options). Progress charts are basic and may have display issues, but the system is partially usable for analytics.	Comprehensive dashboard for both instructors and students, displaying all required metrics (e.g., assignment/quiz status, progress tracking) with intuitive progress charts. CSV export is accurate, supporting individual assignments/quizzes and course/semester summaries. Search/filter/sort functions are fast and comprehensive, with no errors. Analytics interface is user-friendly, visually clear, and fully supports end-of-semester reporting.
Teamwork (GitHub Insights)	**0.5**	No GitHub Insights provided, or commits are limited to 1–2 members with minimal activity (e.g., fewer than 2 commits total). Commits are trivial (e.g., formatting changes) or lack descriptions, showing no evidence of task distribution or collaborative effort. Alternative formats (e.g., text files, screenshots) are used instead of GitHub Insights.	GitHub Insights is provided, but commits are uneven across members (e.g., some members have <2 commits/week) or lack clear task distribution. Commit messages are vague or inconsistent, making it difficult to assess individual contributions. Team collaboration is evident but lacks consistency or organization over the required one-month period.	GitHub Insights is fully provided, showing consistent activity (≥2 commits/week/member) over at least one month. Commits have clear, descriptive messages reflecting specific tasks (e.g., "implemented CSV import," "fixed UI for course page"). Task distribution is well-organized, with balanced contributions from all members, demonstrating strong teamwork.
Web Deployment	**0.5**	Web version is inaccessible, crashes frequently, or fails to support core functionalities (e.g., cannot view courses, submit assignments, or access	Web version is accessible but has minor issues, such as slow loading, UI glitches, or occasional errors in core functionalities (e.g., some features work inconsistently). Cold start issues	Web version is publicly accessible, runs smoothly without cold start issues, and fully supports all core functionalities (e.g., course management, content delivery, notifications). A script or clear

		dashboards). No script or instructions provided for cold start mitigation, making the system unusable during grading. Deployment is effectively non-existent or broken.	require manual intervention, and the deployment may not fully support all required features. Documentation for deployment is incomplete or unclear.	instructions for service wake-up is provided, ensuring immediate access during grading. Deployment is reliable and user-friendly.
User Interface (UI) Scored based on the teacher's perception; only above-average work earns points (a basic interface gets no points)	**0.5**	UI lacks critical components (e.g., no homepage, missing course tabs, or no profile page) or is not responsive (e.g., unusable on mobile or desktop). Design is inconsistent, with broken layouts, missing images, or unreadable text, making the application visually unusable.	UI includes most required components (homepage with role-based views, 3-tab course structure, profile page) but has minor display issues (e.g., misaligned elements, poor scaling on tablets). Responsive design is partially implemented, with some devices showing suboptimal visuals. Design is functional but lacks polish or consistency.	UI is visually appealing, with all required components (homepage with role-based views, 3-tab course structure, profile page) implemented correctly. Fully responsive across mobile, tablet, and desktop, with consistent design, clear layouts, and no display errors. Visuals enhance usability and align with project requirements.
User Experience (UX) Scored based on the teacher's perception; only above-average work earns points (a basic interface gets no points)	**0.5**	UX is poor, with confusing navigation, frequent crashes, or critical errors (e.g., incorrect data display, broken semester switching). Search/filter/sort features are non-functional or cause errors. Users cannot complete basic tasks without significant frustration, rendering the system impractical.	UX is functional but has noticeable issues, such as slow search/filter/sort, clunky semester switching, or minor cache synchronization errors. Users can complete tasks, but the experience is inconsistent or inconvenient (e.g., delayed responses or unclear error messages). Basic usability is achieved but not optimized.	UX is smooth and intuitive, with fast, accurate search/filter/sort and seamless semester switching. Cache synchronization is reliable, preventing data mismatches. All interactions are user-friendly, with clear feedback and no errors, ensuring an efficient and pleasant experience for both instructors and students.

III. OUTPUT REQUIREMENTS

- Required submission components include:
  - The "**source**" folder: Contains the entire source code of the Flutter application, web server, and API if applicable, along with relevant database files. Ensure that this source code can be compiled and run on the teacher's computer. The project needs to be "cleaned" to remove unnecessary content before submission and to reduce the size of the compressed file.
  - The "**bin**" folder: This folder should contain executable files for at least two platforms, including Android (APK) and Windows (EXE). Teachers will use these files for installation on their real/virtual machines for assessment. If necessary, teachers can choose to build the source code from scratch.
  - The demo video, titled "**demo.mp4**" must include the participation of all team members to introduce the group's product. The video should provide a brief overview of the technologies and architecture of the web application. Following this, the team must sequentially demonstrate each feature developed. Any criteria not demonstrated in the video will be considered as not implemented, even if the

group claims otherwise in their self-assessment form. The video must have a minimum resolution of 1080p, clear audio, and be easy to understand. If the video file is too large, the team should upload it to **YouTube** and include the link in the submission.

- The "**git**" folder must contain screenshots taken only from **GitHub Insights** that demonstrate collaboration between team members on the GitHub or GitLab repository, while any other forms of evidence (such as .txt files, terminal git log captures, or photos of command-line history) are strictly **not** acceptable. The submitted screenshots should provide clear proof of effective teamwork with balanced workload distribution, early and regular commits, and proactive collaboration among all members. In addition, the evidence must confirm that the project has lasted for at least one month, during which each member has consistently maintained a minimum of two commits per week.
- **Readme.txt** file: Provide all necessary information for the evaluation process, such as project building and running instructions, URL + server login information (if applicable), and usernames/passwords for accounts with pre-loaded data for assessment. Include any relevant notes on building, running, and using the application for teachers to reproduce the project. If your team implements some optional features (which get extra points), it should be clearly stated in the readme file.
- **The "Bonus"** folder**:** The bonus folder should include a description of the extra features your team implemented for additional points, along with evidence. Organize the information clearly, concisely, and convincingly.
- **Rubrik.docx:** This file lists the 36 required features for the project. Teams should self-assess their completion level in this file. The instructor will provide this file at the submission time. The file will also include the public URL to the web application and any required username/password for login.

- Organize all the above contents into a folder named **id1_fullname1_id2_fullname2**, then compress this folder in ZIP format with the same name, e.g., id1_fullname1_id2_fullname2.zip. A team representative should submit this file on the online learning system as instructed by the course instructor.
- Teachers do not accept sumissions via email, only elearning is accepted.

IV. IMPORTANT NOTES

- The Final Project must be implemented using a **Flutter** project using the **Dart** programming language. Groups are allowed to use any libraries within the framework of a Flutter project.
- Teams may use online storage services such as Firebase or equivalent services, or set up their own web server.
- If your group submits a project that is not relevant, it will not be graded, and the entire group will receive a score of 0. For example, if your group obtains the source code of another email client application for some reason, and only a few features are related to the description in this requirement, while the majority are completely unrelated, the project will be given 0 point.
- The Essay is entirely independent of the Final Project. Therefore, all team members must participate in both the Essay and the Final Project. The Essay will be assessed by the lab instructor, while the Final Project will be assessed by the theoretical instructor.

- Groups are prohibited from sharing code with each other, obtaining source code from the internet, and must take responsibility for protecting their group's source code. Groups with similar source code (verified by specialized software) or code found online, even if only in part, will receive a score of 0 for all members, regardless of which group shared or received the code.
- Failure to submit source code will result in the entire team receiving a score of 0.
- If the team does not fill out and submit the **rubric.docx** file, the submission will not be graded.
- Failure to deploy the web version to public hosting or submit the apk/exe file will result in the entire team receiving a score of 0.
- In case the submission is missing or there are no specific instructions on how to run the project, and the lecturer has tried his best but still cannot run the project on his computer, leading to no way to check the results, the whole group will also receive 0 points.
- Deductions will also apply in the following situations:
    - Late submission: 1-day late deducts 1 point. Submissions late by 1 second to less than 1 day are considered 1 day late.
    - Complex project configuration without specific instructions for instructors to compile and run the program: Deduction of 2 points.
    - Submit the entire project without performing a clean to remove unnecessary files: 0.5 point.
    - Failure to provide necessary grading information, such as missing usernames/passwords, incorrect file naming, or not submitting required content: 1.0 point.