

An-Najah National University



Faculty of Engineering & Information Technology

Department of Computer Engineering

CircuitAI: an Interactive Hardware Learning Platform

Software Graduation Project 1

*A Project Presented to the Faculty of Engineering & Information Technology in Partial
Fulfillment of the Requirements for the Bachelor's degree in Computer Engineering*

By

Ahmed Taher Amin Gharib

Maen Haseeb Omar Thaher

Supervised by

Eng. Haya Samanneh

23/1/2026

Acknowledgment

We would like to dedicate this project to everyone who supported our endeavor; whether directly, such as our academic supervisor, Eng. Haya Samanea, whose invaluable contribution, feedback and assistance greatly shaped the direction of this project. Furthermore, we would also like to thank the students we surveyed before starting this project, whose honest experiences and struggles in studying at An-Najah University (whether in general or in hardware-related courses) gave us real insight and helped us understand real pain points and guided many of our design decisions.

We also would like to extend this gratitude to our family and friends for their unwavering support throughout this challenging academic journey or the development of this project.

As developers, we take immense pride in building projects that benefit the people around us and address real challenges. With this project, we aimed to make the learning experience more accessible, intuitive, and effective for instructors and students alike. We hope this work will be considered a contribution to a smoother, more engaging, and meaningful learning process at An-Najah University.

Abstract

Engineering students have been facing significant challenges bridging theoretical circuit knowledge with its practical implementation. While current general purpose AI systems exist, they have showcased inadequate accuracy in addressing hardware design questions, frequently providing incorrect instructions, flawed circuit analysis, and misleading debugging guidance, leaving students with no reliable assistance once outside of the classroom.

This is where our project comes in: an interactive cross-platform hardware learning application that integrates three foundational components into a single system. The platform includes a Moodle-like learning management layer responsible for course structuring, enrollment, assignments, quizzes, grading, and progress tracking. It also provides interactive circuit simulation, featuring a real-time, client-side digital logic simulator for immediate feedback and conceptual validation, alongside a SPICE-based electrical circuit simulator capable of DC operating point analysis, DC sweep, AC frequency analysis, and transient time-domain simulation. Last but not least, a domain-specific AI engine, enhanced through Retrieval-Augmented Generation (RAG) and grounded in An-Najah University course materials and circuit design principles, which ensures that it delivers accurate, context-specific tutoring, and then we applied the same advanced AI capabilities to quiz generation and AI-assisted grading designed to support instructors while preserving human oversight.

The resulting system enables students to learn theory, design circuits, perform simulations, and complete assignments within a unified workflow aligned with academic and laboratory practices. Integrating learning management, accurate circuit simulation, and curriculum-grounded AI ensures coherence between instruction, practical application, and evaluation.

The front-end development was done using NextJS and CSS modules, while the back-end development was done in python for the AI inference server and setup, while the RESTful API endpoints were tied closely to our front-end via NextJS API routes. As for database choice, we used a PostgreSQL database for user data and general storage, and ChromaDB vector database for AI embeddings.

While general circuit tools such as Multisim and Proteus support heavy simulation, and TinkerCAD and CircuitLab offer lighter educational interfaces, and while general-purpose AI assistants are becoming widely available, no project combines the best of all worlds into a single, easily accessible platform. This work demonstrates the effectiveness of such integration, with An-Najah University as its starting point.

Introduction

General Background

Electrical and digital circuit designs is a core pillar of many engineering majors (such as computer, electrical, mechatronics), yet it is one of the areas where students most frequently struggle to translate theory into reality. In the typical fashion, the student first encounters circuit laws, device models, and logical reasoning in lectures, and later attempts to apply them in laboratories, assignments and design exercises. This transition exposes a gap: solving a circuit on paper is fundamentally different from building, simulating, debugging, and validating it under real constraints such as incorrect wiring, missing ground references, non-ideal component behavior, misunderstanding of measurement points, and even faulty hardware components. And there exists learning workflows that could help mitigate that gap, but they are scattered across multiple systems; Course content and assessments are handled in a learning management system (LMS), simulation is usually performed in specialized heavy desktop tools. And students often rely on informal sources such as online forums for explanations and debugging help. This fragmentation introduces a real-world barrier: complex installations, licensing constraints, device and platform incompatibility, and a lack of continuity between learning materials, lab activities, and evaluation. As a result, students may spend significant time switching tools instead of focusing on conceptual understanding and hands-on skills. At the same time, general-purpose AI assistants have become widely available, and they are supposed to be a real help when it comes to bridging those gaps, and answering students hardware-related questions accurately, since they do use the entire internet as a source of info. However, traditional large language models [LLMs] often struggle with hallucinations, which causes it to produce confident but incorrect explanations, flawed circuit reasoning, or misleading debugging steps. And that is especially true when the question depends on a certain convention that is used in a specific course material but not necessarily all.

These challenges highlight the need for a more coherent learning environment that connects course content, hands-on practice, and reliable support in one place. By reducing tool-switching and confusion, students can focus on understanding circuits and developing practical skills rather than struggling with fragmented workflows.

Objectives and Purpose of the Work

This work was done to reduce the practical gap between theoretical hardware education and real-world applications by building a single software platform that supports the full learning cycle: instruction, practice, validation, and evaluation.

The main objectives of this project are:

1. Provide an integrated circuit-learning environment that combines course delivery (modules, lectures, slides), student management (enrollment, role-based access), and assessment tools (assignments, quizzes, grading, progress tracking) in one system rather than distributed tools.
2. Enable hands-on circuit practice with immediate feedback by delivering two complementary simulators
 - a) A client-side digital logic simulator that runs in real time in the browser for fast conceptual validation of combinational and sequential logic.
 - b) A SPICE-based electrical circuit simulator powered by a Python back-end capable of DC operating point analysis, DC sweep, AC frequency analysis, and transient simulation for realistic analog behavior.
3. Deliver reliable, curriculum-grounded AI tutoring using a RAG pipeline that retrieves relevant content from ingested course materials (Books and Slide) and generates answers with source awareness, reducing hallucination risk, grounding the answer, and showing proper citations directly from the material the student is studying.
4. Support instructors with AI-assisted automation while preserving oversight, whether through AI-driven quiz generation from lecture slides to reduce workload while keeping questions aligned with course content. Or AI-assisted grading that provides suggested grades and feedback as a decision-support tool, not as a replacement for the instructor.

Significance and Importance of the Work

This work is tied to the increasing demand for accessible, scalable, and interactive engineering education tools. Universities worldwide are steadily moving towards digital and hybrid learning models. Where students expect easy access to course materials, labs, and assessments with minimal friction. At the same time, engineering programs in many universities around the world face increasing pressure to deliver practical, industry-relevant skills while managing large class sizes and limited laboratory time and staffing. These trends create a strong need for platforms that can provide hands-on practice, consistent evaluation, and continuous support beyond the physical lab in a way that allows the instructor to scale their teaching effort without turning the grading, tracking and support process into a bottleneck.

As engineering education continues to shift toward blended delivery, tools that combine learning, practice, and evaluation in one workflow are becoming a baseline expectation rather than an extra feature.

In addition, there is a growing demand for AI-integrated software all around the globe, but we end up with many programs that are just a generic LLM wrapper or out-of-place AI usage, where the “AI feature” feels cosmetic rather than functional. In contrast, our software introduces a real, high-impact use case for AI within engineering education by embedding a domain-specific support layer directly into the learning workflow, so students receive help that stays aligned with their course context instead of generic, disconnected answers. At the same time, the AI is explicitly designed as a support tool, not a replacement for instructors or a shortcut around learning. Because human expertise is essential for engineering judgment, proper evaluation, and oversight; the system uses AI to assist students and reduce repetitive instructor workload while keeping teaching authority and final decisions in human hands

Organization of the Report

This report is organized as follows: Chapter 2 outlines the system requirements and design goals. Chapter 3 presents the methodology in detail, including the selected tools, programming languages, architecture decisions, and how course materials were processed for the AI pipeline. Chapter 4 carries the main weight of the work, presenting the implemented features and results. Chapter 5 discusses limitations, trade-offs, and future improvements.

System Requirements and Design Goals

Problem Scope and Boundaries

This project targets a well-defined problem within engineering education: the difficulty students face when translating theoretical circuit concepts into practical design, simulation, and validation workflows. The scope is focused on university-level hardware courses, with An-Najah University serving as the primary academic context.

Rather than separating learning management, circuit simulation, and student support across multiple tools, the system integrates these components into a single, coherent platform. Students can move smoothly between studying course material, designing circuits, simulating behavior, and completing assessments without leaving the environment.

The project emphasizes educational correctness, integration, and accessibility, prioritizing alignment with academic workflows over industrial-scale simulation depth or commercial deployment concerns.

The following aspects are considered out of scope for this software graduation project:

1. Support for a broad or extensible library of circuit components, whether digital or analog, beyond the predefined educational set needed for An-Najah students.
2. Highly specialized analysis domains, such as RF circuit design, power electronics, signal integrity analysis, or simulations requiring frequency ranges and numerical precision beyond standard undergraduate coursework.
3. Physical laboratory hardware integration or real-time embedded execution
4. Academic materials and curricula outside An-Najah University
5. Commercial deployment concerns such as large-scale scalability, licensing, or high-availability guarantees.
6. Any unique mobile application features (it represents a mirror of the web version)

Stakeholders and Usage Scenarios

1. Students: They use the platform as a unified learning environment that connects course materials, circuit design, simulation, and assessment. The system allows them to validate understanding through simulation and receive AI-assisted explanations grounded in their actual course content.

2. Instructors: They use the platform to organize courses, manage assignments and quizzes, and review student submissions. AI-assisted features reduce repetitive workload by generating quiz drafts and grading suggestions, while all academic judgment remains under instructor control.
3. Administrators: They should be able to manage user accounts, course structure, and platform configuration to ensure alignment with institutional requirements.

Functional Requirements

1. User authentication and role-based access control
2. Course and module management
3. Lecture material upload and organization
4. Student enrollment and progress tracking
5. Assignment creation, submission, and grading
6. Quiz creation, performing (taking) and evaluation
7. Browser-based digital logic circuit simulation
8. SPICE-based electrical circuit simulation (DC, AC, sweep, transient)
9. Visualization of simulation results
10. Circuit saving and reuse
11. AI-powered course-grounded tutoring
12. Retrieval-Augmented Generation for accurate, source-aware answers
13. AI-assisted quiz generation
14. AI-assisted grading with structured feedback
15. REST API-based front-end to back-end communication
16. Mobile application shall provide access to core platform features.
17. Mobile application shall synchronize user data and progress with the web platform through shared back-end services.

Non-Functional Requirements

1. Correct and reliable simulation behavior for educational circuits
2. Stable operation during normal academic use and demonstrations
3. Optimized response times for simulations and AI interactions
4. Clear and intuitive user interface
5. Maintainable and modular codebase
6. Secure handling of user data and credentials
7. Compatibility with modern web browsers

8. Mobile application shall provide an interface adapted to smaller screens while preserving usability and content clarity.

Methodology

Methodological Overview

This project follows a software engineering design and implementation methodology rather than a traditional physical experimental approach. The methodology focuses on system architecture design, technology selection, implementation procedures, and validation through functional behavior and reproducibility.

Since the work is software-based, the “materials” consist primarily of programming frameworks, libraries, databases, and APIs, while the “methods” consist of system design practices, back-end to front-end integration procedures, and algorithmic approaches for simulation and AI-assisted features.

All components were built using standard, well-documented technologies, so that the system can be reproduced or extended by other experienced developers.

System Architecture and Design Principles

The system follows a modular, layered architectural approach to ensure maintainability, scalability, and clear boundaries throughout the entire design. Each layer communicates through well-defined interfaces, which enables individual subsystems to evolve independently.

- **Presentation Layer (Web and Mobile Clients):** Implements user interfaces (UI) using modern web technologies, handling rendering, user interaction, and client-side validation while remaining independent of back-end computation logic.
- **Application Logic Layer (Next.js API Routes):** Acts as a lightweight orchestration layer responsible for request handling, authentication enforcement, and coordination between front-end clients and back-end services.
- **Computation Layer (Python Back-end Services):** Isolates compute-intensive tasks such as electrical circuit simulation and AI inference, enabling the use of scientific and machine learning libraries on a suitable python environment without impacting front-end performance.

- **Data Layer (Relational and Vector Databases):** Manages persistent storage for structured educational data and unstructured AI embeddings using databases optimized for each data type.

The following design principles guided the system architecture:

- **Separation of Concerns:** Each layer has a single, well-defined responsibility to reduce complexity and coupling.
- **Modular Design:** Simulation, AI assistance, learning management, and file handling are implemented as independent modules.
- **Reusability:** Shared components and services are reused across multiple features and user roles.
- **Clear API Boundaries:** Communication between layers occurs exclusively through documented REST interfaces.
- **Incremental Development and Testing:** Features were developed and validated independently to reduce integration risk.

Technology Stack and Tools

Front-end Technology Stack

1. **Next.js 15 (App Router):** Serves as the primary web framework, providing server-side rendering, optimized routing, and bullet-proof yet simple integration between front-end components and back-end API routes.
2. **React 19:** JavaScript/TypeScript library for building scalable UI interfaces, following a component-based approach and managing dynamic content and state effectively

TypeScript 5: Enforces static typing throughout the front-end codebase, improving code reliability.
3. **CSS Modules:** Enables locally scoped component styles, preventing global namespace collisions and ensuring modular, maintainable UI styling.
4. **Fabric.js:** Powers the interactive circuit design canvas, enabling drag-and-drop components, wire connections, and real-time visual feedback for digital and electrical circuit simulations.

5. Axios: Handles HTTP communication with back-end services, providing centralized request configuration, interceptors, and consistent error handling.
6. React Native: Enables cross-platform mobile application development using the same component-based architecture and business logic principles as the web application.
7. Expo: Simplifies mobile development and deployment by providing managed tooling, native API access, and streamlined build pipelines for the React Native application.

Back-end Technology Stack

1. Next.js 15 API Routes: Implements server-side API endpoints for handling authentication, authorization, request routing, and orchestration between front-end clients and back-end services, enabling secure server-side execution tightly integrated with the front-end.
2. Python back-end Server: Handles compute-intensive operations that are not suitable for execution in the JavaScript runtime, including circuit simulation and AI inference.
3. PySpice with NGSpice: Provides SPICE-based electrical circuit simulation by solving circuit equations derived from Kirchhoff's laws, supporting DC operating point analysis, DC sweep, AC frequency analysis, and transient time-domain simulation.
4. JOSE (JSON Web Token): Used for secure session handling through signed and verified JWTs stored in HTTP-only cookies, enabling stateless authentication.
5. bcryptjs: Implements secure password hashing with salting to protect user credentials following standard security practices.

AI Technology Stack

1. Groq API: Provides high-performance large language model inference for chatbot responses, quiz generation, and grading assistance.
2. ChromaDB: Vector database used to store and retrieve embeddings of ingested lecture slides and reference documents for semantic search.
3. Sentence Transformers: Generates dense vector embeddings for PDF documents, slide content, and user queries.
4. PyPDF / PyMuPDF (fitz) / python-pptx: Used to ingest both PDF documents and PowerPoint slide decks, enabling text extraction (including layout-aware extraction for slide-style files), access to speaker notes and slide metadata, and retrieval of embedded images for OCR when needed.

5. Tesseract OCR (via pytesseract): Used to extract text from image-based slides or diagrams when native text extraction is insufficient.

External Technologies:

1. Resend API: Handles transactional email delivery for account verification, password reset, and system notifications.
2. Google Drive API: Used for uploading and managing lecture videos, allowing instructors to host large media files without overloading the main database.

Database Design and Technology Stack

The database schema was designed using relational modeling principles with clear separation of concerns between academic entities, system entities, and AI-related data.

Key design decisions include:

1. Clear Entity Separation: Academic, system, and AI-related data are modeled independently to reduce coupling.
2. Relational Integrity & Access Control: Foreign keys, roles, and RLS enforce consistency and secure access at the database level.
3. Logic at the Database Layer: Views and RPC functions handle aggregations and complex queries for performance and consistency.
4. Vector / Transactional Separation: AI embeddings are isolated from relational data to avoid performance interference.

Key technologies used:

1. PostgreSQL (Supabase): Primary relational database for users, courses, assessments, submissions, and circuit data, with built-in auth and row-level security.
2. Supabase Storage: Object storage for PDFs, slide decks, assignments, and submission files linked via relational metadata.
3. ChromaDB: Dedicated vector database for storing and querying AI embeddings used in semantic search and RAG workflows.

Standards and Specifications (Codes)

Although the project does not rely on hardware communication standards, several engineering and software standards were applied:

- SPICE-based circuit analysis conventions for electrical simulation
- IEEE logic symbol conventions for digital circuit representation
- REST architectural style for API design
- SQL relational database standards for data storage
- Web standards (HTML, CSS, JavaScript) for browser compatibility
- Secure password handling standards (hashing, session control)

Project Constraints

1. Time constraint: Development was bounded by academic timelines, forcing parallel feature development and limiting deep iteration on individual subsystems.
2. Data gathering constraint: Access to high-quality, well-structured educational datasets beyond An-Najah University (particularly for practical circuit principles and real-world hardware implementations) was limited. As a result, the system relied heavily on instructor-provided materials, which varied significantly in depth, structure, formatting quality, and pedagogical consistency.
3. LLM capability constraint: AI features rely on cost-constrained, open-source / lower-tier models, which trade reasoning depth and mathematical rigor for affordability and deployability.
4. Breadth-over-depth constraint: The project intentionally prioritized horizontal system coverage (simulation, LMS, AI, assessment) over vertical specialization in any single domain.
5. Document and text extraction constraint: PDF and PPTX inputs vary widely in structure, cleanliness, and formatting, including noisy layouts, inconsistent headers and footers, visual-first slide designs, and embedded or scanned content. As a result, text extraction must operate under imperfect conditions, where fragmentation and OCR inaccuracies are unavoidable.

Results and Discussion

Feature 1: Course Management System

High-level summary

The Course Management System delivers core LMS functionality for course discovery, enrollment, and structured content delivery. It organizes content in a two-level hierarchy (Course → Ordered Modules) and exposes role-specific views for students, instructors, and administrators. The main outputs are course lists with progress/deadlines, course detail pages with an ordered module timeline, and instructor/admin management interfaces.

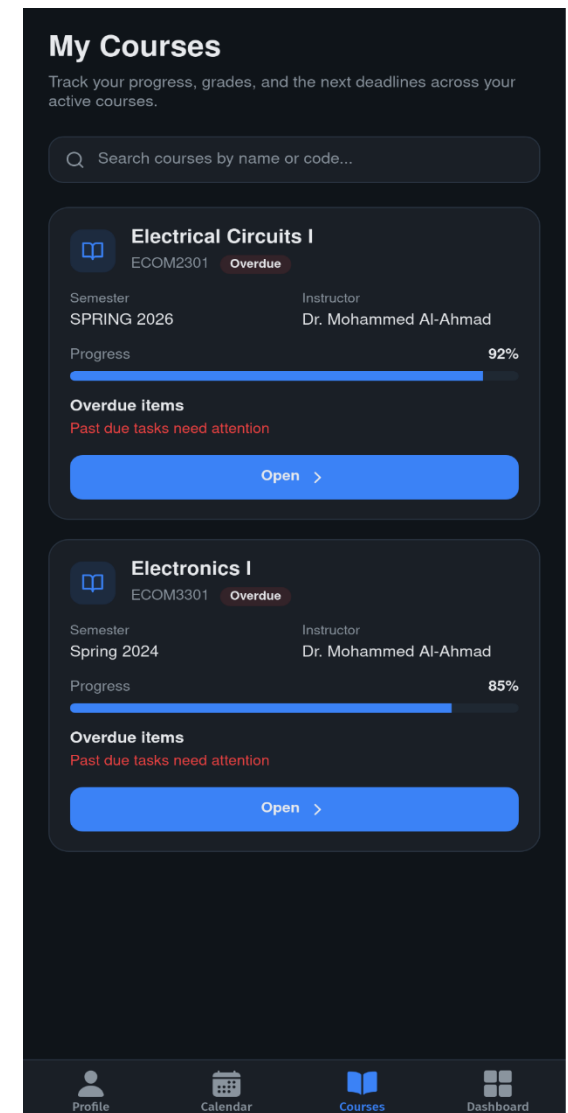
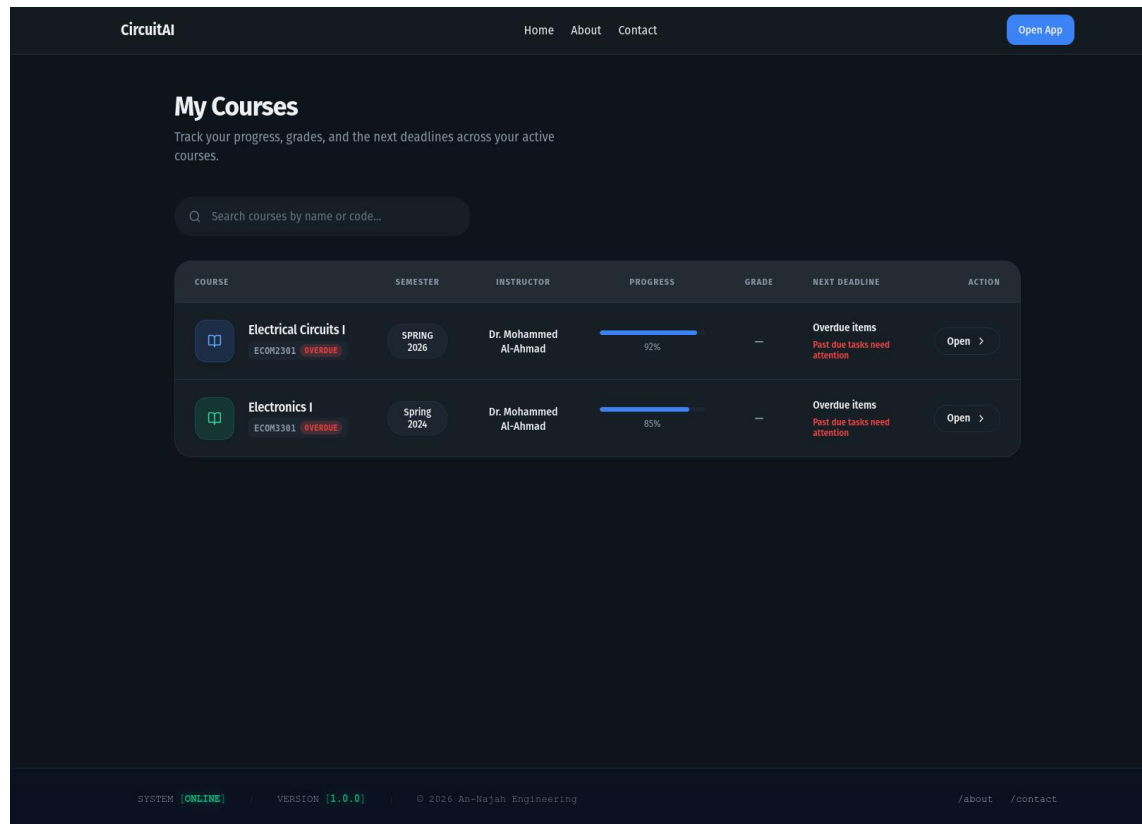
Feature list

- Browse enrolled courses with search, status, progress, and next-deadline indicators.
- Open a course detail page that shows published modules in sequence and tracks progress.
- Manage courses as an instructor: view owned courses, create/edit modules, reorder modules, and publish/unpublish content.
- Manage student enrollments (add/remove) with search and progress visibility.
- Provide admin oversight across all courses, including instructor reassignment.
- Support multiple module types (Lecture, Slides, Lab, Quiz, Assignment) with type-specific handling and content integration.

Sub-features

Student Course List & Discovery (Fig. 1)

- **What it does:** Displays the student's enrolled courses in a card-based layout with progress, status, and upcoming deadline cues. Supports search by course name or code.
- **Inputs:** Student navigates to /student/courses; optional search term (name/code).
- **Outputs:** Course cards showing course name/code/semester, enrollment date, status badge, progress indicator (0–100%), next deadline info, and grade summaries when available.
- **Typical workflow:**
 1. Student navigates to /student/courses.
 2. System fetches enrolled courses and progress data.
 3. Courses render in a grid/list of cards.
 4. Student searches by name or code (optional).
 5. Student clicks a course card to open the course detail page.



(Figure. 1) Student Course List & Discovery

Student Course Detail & Content Access (Fig. 2)

- **What it does:** Provides students with a read-only, comprehensive course overview that combines course metadata with an ordered module timeline restricted to published modules only. It tracks overall progress and completion, highlights upcoming due items that require urgent attention, and identifies the next item in the student's learning pathway within the course progression. The interface also offers quick access to course resources, including instructor-uploaded lecture slides and assignment files, alongside dedicated sections for quizzes and assignments that display their status, marks, grades, and feedback when available, thereby delivering a unified and intuitive view of course structure, progress, and assessment outcomes.
- **Inputs:** Student navigates to /student/courses/[courseId]; student actions include clicking modules and accessing different types content.
- **Outputs:** Dashboard header (personalized greeting and date), key metrics (enrolled courses, upcoming modules, average progress), prioritized upcoming tasks with status and deadlines, course cards showing name/code and progress, recent feedback summaries (grades and comments), resumable work items, and activity feed entries.
- **Typical workflow:**
 1. Student navigates to /student/courses/[courseId] via dashboard or direct URL.
 2. System validates enrollment and loads course metadata, published modules, assessments, resources, and student progress.
 3. Page renders course header, overall progress, and ordered module timeline with type and status indicators.
 4. System highlights next and overdue items within the course progression.
 5. Student selects a module to access lectures, assignments, or quizzes in read-only mode.
 6. System displays content, grades, feedback, and resource files where applicable.
 7. System tracks progression and resume state and updates overall completion metrics while enforcing access rules.

- **Validation & constraints:** Student view filters modules to status = 'Published' (draft modules are hidden).

CircuitAI

HomeAboutContact

Open App

← Courses

ECOM2301

ECOM2301

SPRING 2026

Electrical Circuits I

Introduction to circuit analysis, Ohm's law, Kirchhoff's laws, and basic circuit theorems. This course provides the foundation for all future electronics engineering studies.

INSTRUCTOR
Dr. Mohammed Al-Ahmad
m.alahmad@najah.edu

92%

Completion

📅 NEXT DUE

Quiz: XXXX

Closed 4 days ago - Sun, Jan 18, 23:54

Module timeline

10

📄 Ohm's Law and Basic Circuits

Lecture

Self-paced

AVAILABLE

Open →

📄 Kirchhoff's Laws

Lecture

Self-paced

AVAILABLE

Open →

📄 Series and Parallel Circuits Lab

AVAILABLE

Start next item

Start next item →

Next up: Quiz: XXXX

Resources

View all →

Slide decks

Ch06-circuit.pdf

PDF - 21

Unavailable

Ch04-circuit.pdf

PDF - 24

Unavailable

Ch08-circuit.pdf

PDF - 10

Unavailable

SYSTEM ONLINE

VERSION 1.0.0

© 2026 An-Najah Engineering

/about

/contact

1

CircuitAI

[Home](#)[About](#)[Contact](#)

Open App

Kirchhoff's Laws Application

Overdue by 81 days Sun, Nov 2, 21:28

Status: Draft saved

DRAFT SAVED

Continue →

Series and Parallel Circuits Lab

Overdue by 74 days Sun, Nov 9, 21:28

Status: Graded

GRADED

View Feedback →

Voltage Divider Design

Overdue by 67 days Sun, Nov 16, 21:28

Status: Graded

GRADED

View Feedback →

RC Circuit Analysis 123

Overdue by 60 days Sun, Nov 23, 21:28

Status: Graded

GRADED

View Feedback →

Quizzes 2

View all →

COMPLETED

0/2

IN PROGRESS

0

XXXX

CLOSED Closed 4 days ago Sun, Jan 18, 23:54 0/6 answered

View Details →

Slide decks

Ch06-circuit.pdf

PDF - 21

Unavailable

Ch04-circuit.pdf

PDF - 24

Unavailable

Ch08-circuit.pdf

PDF - 18

Unavailable

Ch03-circuit.pdf

PDF - 15

Unavailable

Ch02-circuit.pdf

PDF - 10

Unavailable

Attachments

10_18 Context Pack Strategy .pdf

Assignment: RC Circuit Analysis 123

Download

hw1-unrolling.pdf

Assignment: unrolling test

Download

attachment

Quiz: Test

Download

Ahmed_CV.pdf

Slides: Test

Download

attachment

Lecture: xxx

Download

attachment

Quiz: XXXX

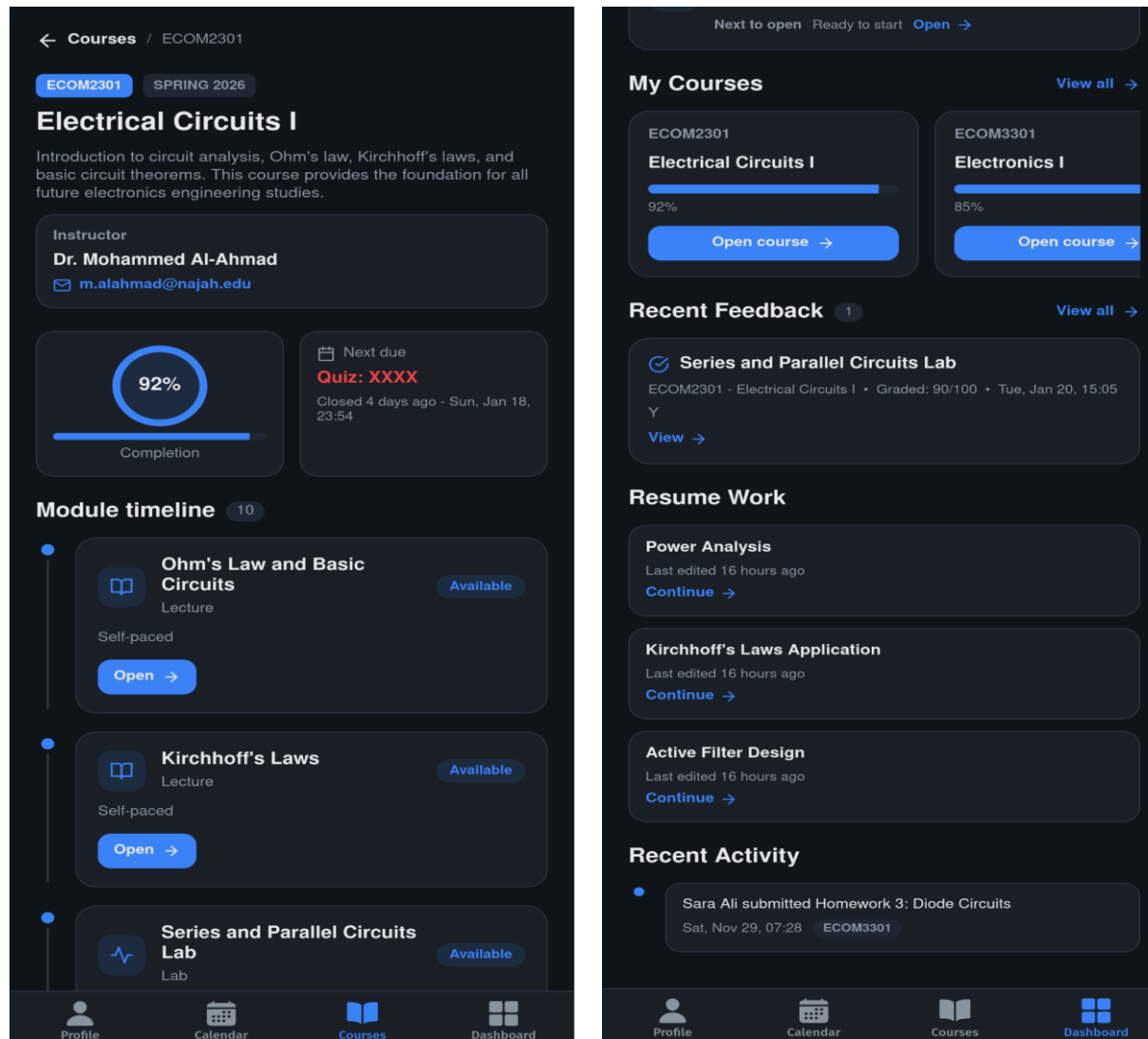
Download

SYSTEM ONLINE

VERSION 1.0.0

© 2026 Ah-Najah Engineering

[/about](#)[/contact](#)



(Figure. 2) Student Course Detail & Content Access

Instructor Course List (Fig. 3)

- **What it does:** Lists courses owned by the current instructor in a table with key metadata and summary stats. Supports search and provides quick actions (edit/manage students/view details).
- **Inputs:** Instructor navigates to /instructor/courses; optional search/filter actions; “Create new course” action.
- **Outputs:** Table rows showing course name/code/semester, student count stats, status badges (active/archived/draft), and per-course action menus.
- **Typical workflow:**
 1. Instructor navigates to /instructor/courses.
 2. System fetches courses where instructor_id matches the current user.
 3. Courses render in a table with statistics.
 4. Instructor searches/filters (optional) or creates a new course.
 5. Instructor selects a course to open the course management page.
- **Validation & constraints:** Ownership filtering based on instructor_id.

CircuitAI

[Home](#)[About](#)[Contact](#)

Open App

Your Courses

Manage your curriculum and student cohorts.

Q Search courses by name or code...

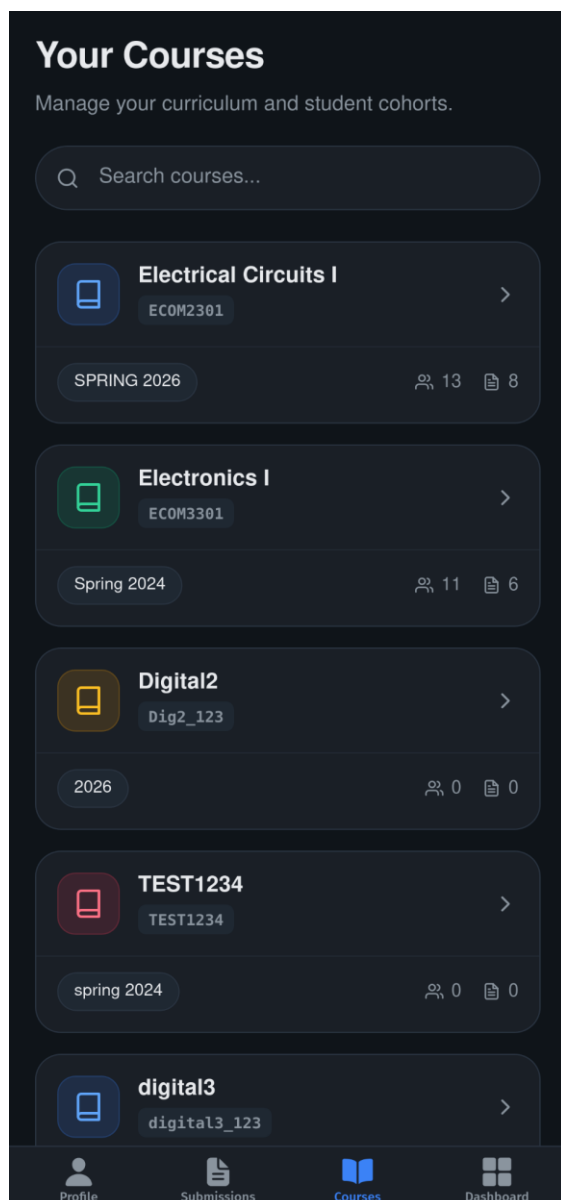
COURSE	SEMESTER	STATS	
 Electrical Circuits I ECON2301	SPRING 2026	 12  8	Open >
 Electronics I ECON3301	Spring 2026	 10  6	Open >
 Digital2 Dig2_123	2026	 0  0	Open >
 DIG1 Digital Circuit Design I	spring 2026	 0  0	Open >
 digital3 digital3_123	2026	 0  0	Open >
 electronic2 electronic2_123	2026	 0  0	Open >

SYSTEM **ONLINE**

VERSION **1.0.0**

© 2026 An-Najah Engineering

[/about](#)[/contact](#)



(Figure. 3) Instructor Course List

Instructor Course Detail & Management (Fig. 4)

- **What it does:** Provides instructors with centralized control over a single course, including metadata management, module lifecycle control (draft → published), assessment configuration, order content sequencing managing resources, and general course-level analytics.
- **Inputs:** Instructor navigates to `/instructor/courses/[courseId]`; edits course metadata; creates, configures, reorders, publishes, and deletes modules; manages assignments and quizzes; uploads resources; selects circuit templates for labs; monitors enrollments and submissions; exports course and performance data.
- **Outputs:** Editable course header with metadata and enrollment metrics, ordered module timeline with type and status indicators, module creation/edit modals, assessment management views (submissions, grading, feedback), student progress overview, resource management section, and course analytics summaries.
- **Typical workflow:**
 1. Instructor opens `/instructor/courses/[courseId]`; system validates ownership and loads course structure, assessments, enrollments, and analytics.
 2. Interface renders course overview and ordered module timeline with draft/published states
 3. Instructor creates or edits modules and configures type-specific content (lecture, lab, quiz, assignment).
 4. Instructor reorders modules and publishes content to make it visible to students.
 5. Instructor manages assignments and quizzes, reviews submissions, and performs grading.
 6. Instructor monitors student progress and course analytics.
 7. System validates changes, updates course state, and logs actions.
- **Validation & constraints:** The system applies a unified validation schema for course metadata, module configuration, and assessment rules, with strict separation between draft and published content. Core constraints include role-based access control, ordered module integrity, and validated assessment parameters (deadlines, attempt limits, content types).

Referential integrity and audit logging are enforced across modules, submissions, and grading workflows.

CircuitAI

HomeAboutContact

Open App

← Courses / ECOM2301

ECOM2301SPRING 2026

StudentsEdit Course

Electrical Circuits I

Introduction to circuit analysis, Ohm's law, Kirchhoff's laws, and basic circuit theorems. This course provides the foundation for all future electronics engineering studies.

12Enrolled Students

8Total Assignments

11Course Modules

Course Modules11

+ Add Module

Ohm's Law and Basic Circuits

Lecture

PUBLISHED

Kirchhoff's Laws

Lecture

PUBLISHED

Test

quiz

PUBLISHED

Series and Parallel Circuits Lab

Lab

PUBLISHED

RC and RL Circuits

Lecture

PUBLISHED

AC Circuit Fundamentals

Lecture

PUBLISHED

DRAFT

PUBLISHED

PUBLISHED

Circuit Templates

+ Add Template

Voltage Divider Circuit

Standard voltage divider configuration for reference voltage generation.

RC circuit template

RC circuit for time constant analysis and filter design.

Series resistor network

Basic series circuit template for learning voltage division and current flow.

Browse Template Library

SYSTEM [ONLINE]

VERSION [1.0.0]

© 2026 An-Najah Engineering

/about /contact

Assignments8

View All →

Assignment Title	Due Date	Status	Submissions
unrolling test	Sat, Dec 20, 15:00	Published	1 / 12 Review
TXAS	Sun, Dec 7, 11:45	Draft	0 / 12 Review
Power Analysis	Sat, Dec 20, 21:28	Published	0 / 12 Review
Voltage Divider Design	Sun, Nov 16, 21:28	Published	10 / 12 Review
AC Circuit Analysis	Tue, Jan 13, 21:28	Published	0 / 12 Review
Kirchhoff's Laws Application	Sun, Nov 2, 21:28	Published	0 / 12 Review
RC Circuit Analysis 123	Sun, Nov 23, 21:28	Published	13 / 12 Review
Series and Parallel Circuits Lab	Sun, Nov 9, 21:28	Published	1 / 12 Review

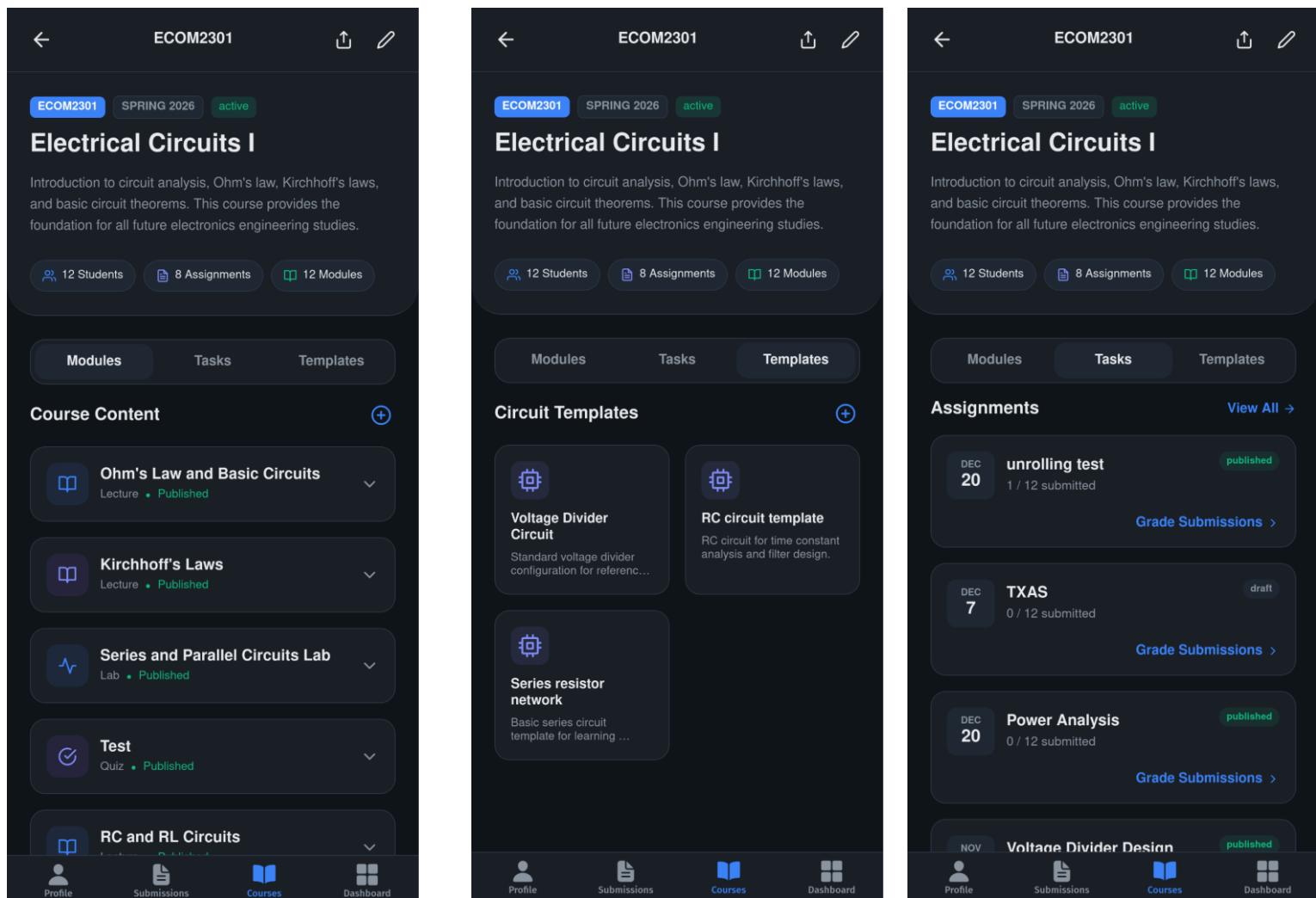
SYSTEM [ONLINE]

VERSION [1.0.0]

© 2026 An-Najah Engineering

/about /contact

2



(Figure. 4) Instructor Course Detail & Management

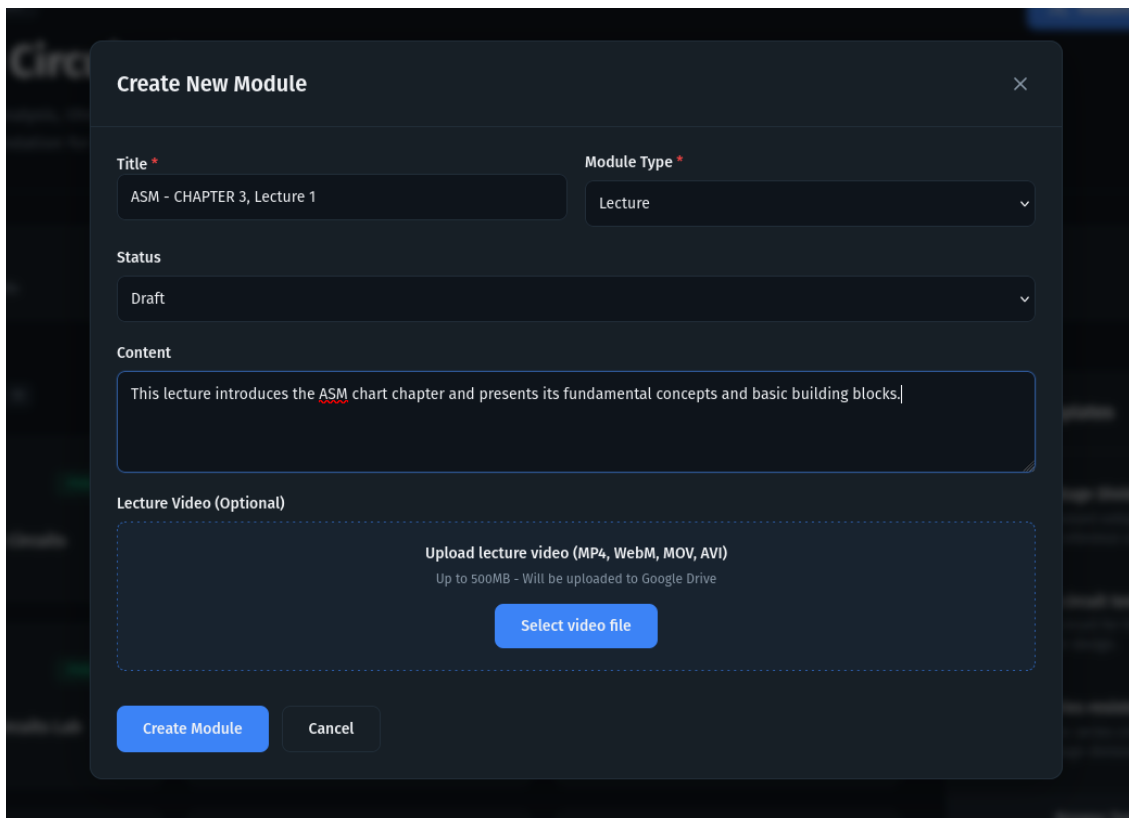
Module System (Fig. 5)

- **What it does:** Defines module types, visibility states, and ordering rules used across student and instructor experiences. This is the shared layer that drives the module timeline and content delivery behavior.
- **Inputs:** Instructor sets `module_type`, `module_status`, and ordering via `order_index`; may attach content via uploads or template association.
- **Outputs:** Consistent module timeline behavior and routing across module types; type-specific content rendering based on stored fields (e.g., `attachment_url`, `template_id`, `starts_at/ends_at`).
- **Typical workflow:**

1. Instructor creates a module with a `module_type`.
2. System stores the module and assigns/updates `order_index`.
3. Instructor sets `module_status` to Draft or Published.
4. Student timeline queries include only Published modules.
5. Front-end routes module interactions based on type.

- **Validation & constraints:**

1. `module_type` ENUM values: 'Lecture', 'Lab', 'Quiz', 'Assignment', 'Slides'.
2. `module_status` ENUM values: 'Published', 'Draft'.
3. Draft modules are hidden from students; Published modules appear in the student timeline.



The image shows a 'Create New Module' form with the following fields and options:

- Title ***: Text input containing 'ASM - CHAPTER 3, Lecture 1'.
- Module Type ***: Dropdown menu with 'Lecture' selected.
- Status**: Dropdown menu with 'Draft' selected.
- Content**: Text area containing 'This lecture introduces the ASM chart chapter and presents its fundamental concepts and basic building blocks.'
- Lecture Video (Optional)**: Section with instructions 'Upload lecture video (MP4, WebM, MOV, AVI)' and 'Up to 500MB - Will be uploaded to Google Drive'. It includes a 'Select video file' button.
- Buttons**: 'Create Module' and 'Cancel' buttons at the bottom.

Edit Module

Title *

ASM - CHAPTER 3, Lecture 1

Module Type *

Lecture

Status

Draft

Content

This lecture introduces the ASM chart chapter and presents its fundamental concepts and basic building blocks.

Lecture Video (Optional)

Upload lecture video (MP4, WebM, MOV, AVI)

Up to 500MB - Will be uploaded to Google Drive

Current file: Attachment

Replace video

simplescreenrecorder-2025-10-26_12.57.25.mp4

Remove

Update Module

Cancel

Delete Module

Are you sure you want to delete "ASM - CHAPTER 3, Lecture 1"? This action cannot be undone and will permanently remove the module and all associated content.

Cancel

DELETE MODULE

(Figure. 5) Module System

Module Creation Modal (Fig. 6)

- **What it does:** Provides a type-driven interface for creating course modules, dynamically adapting fields and validation based on module type, handling file uploads and special workflows (quizzes and assignments), and automatically integrating new modules into the course structure.
- **Inputs:**
 - Instructor selects module type: Lecture, Slides, Lab, Quiz, Assignment.
 - Common fields: title, description, status (Draft/Published).
 - Type-specific inputs:
 - Lecture: video file upload (auto-upload to Google Drive).
 - Slides: PDF/PPTX upload (auto-upload to Supabase Storage, triggers RAG ingestion).
 - Lab: template selection or creation.
 - Quiz: time windows, question counts.
 - Assignment: due date, max points, instructions.
- **Outputs:** Creation of a new module with automatic `order_index` assignment, populated type-specific fields, triggered workflows for quizzes and assignments, real-time validation feedback, file upload handling, and updated course module list.
- **Typical workflow:**
 1. Instructor clicks “Add Module”; system opens modal with type selection.
 2. instructor selects module type; system activates type-specific fields and validation rules.
 3. Instructor provides required metadata and content (files, templates, or parameters).
 4. System processes uploads and validates inputs.
 5. On submission, system creates the module, assigns the next `order_index`, and updates the course timeline.

6. For quizzes or assignments, system redirects to specialized creation workflows.
 7. System displays success or error feedback and preserves form state when needed.
- **Validation & constraints:** The modal applies a unified validation schema with type-specific rules and strict separation between draft and published modules. Critical constraints include valid module typing, chronological quiz timing, file format and size limits, and consistent module ordering. Instructor ownership and referential integrity are enforced across module creation and associated resources. Supported formats for lecture slides include PDF, PPTX, and legacy PPT files, with a maximum size of 40 MB, while supported formats for lecture media uploads include MP4, WebM, MOV, and AVI, with a maximum size of 500 MB.

The image shows a 'Create New Module' modal form with a dark theme. The form contains the following fields and sections:

- Title ***: Text input with 'First Quiz' entered.
- Module Type ***: Dropdown menu with 'Quiz' selected.
- Status**: Dropdown menu with 'Draft' selected.
- Description**: Text area with 'First Quiz for Digital circuit design I' entered.
- Start Date & Time**: Date and time picker showing '2026-01-23 21:30'.
- End Date & Time**: Date and time picker showing '2026-01-23 23:00'.
- Number of Questions**: Section with the instruction 'Specify how many questions of each type you want to create'. It contains three sub-sections:
 - Multiple Choice**: Input field with '2' and a plus/minus icon.
 - True/False**: Input field with '2' and a plus/minus icon.
 - Short Answer**: Input field with '2' and a plus/minus icon.

At the bottom of the modal are two buttons: 'Create Module' (highlighted in blue) and 'Cancel'.

Create New Module

Module Type *

Assignment

Click "Create Module" to open the assignment creation modal.

Create Module

Cancel

Create New Module

Title *

RC Circuit Analysis Educational Lab

Module Type *

Lab

Status

Draft

Description

Add lab description...

Circuit Template

RC circuit template

Create Module

Cancel

Create New Module

Title *

Unit1_Device_Oveview_PIC18

Module Type *

Slides

Status

Published

Slides File (Optional)

Upload slides file (PDF, PPTX)

Up to 40MB

Select slides file

Create Module

Cancel

Create New Module

Title *

ASM - CHAPTER 3, Lecture 1

Module Type *

Lecture

Status

Draft

Content

This lecture introduces the ASM chart chapter and presents its fundamental concepts and basic building blocks.

Lecture Video (Optional)

Upload lecture video (MP4, WebM, MOV, AVI)

Up to 500MB - Will be uploaded to Google Drive

Select video file

Create Module

Cancel

← ECOM2301 ↑ ↗

Edit Course ×

Course Code *

ECOM2301

Unique identifier for the course (e.g., CS101)

Course Name *

Electrical Circuits I

Status

Active ▾

Semester

SPRING 2026

Description

Introduction to circuit analysis, Ohm's law, Kirchhoff's laws, and basic circuit theorems. This course provides the foundation for all future electronics engineering studies.

Cancel Update Course

(Figure. 6) Module Creation Modal

Student Enrollment Management (Fig. 7)

- **What it does:** Lets instructors enroll/remove students for a course, with search-based selection and visibility into enrollment/progress metrics.
- **Inputs:** Instructor navigates to `/instructor/courses/[courseId]/students`; searches by name/email; selects students to add; confirms removals.
- **Outputs:** Enrolled student list with name/email, enrollment date, progress percentage, assignment completion stats; add-student modal; confirmation dialog for removals.
- **Typical workflow (enroll):**
 1. Instructor opens `/instructor/courses/[courseId]/students`.
 2. Instructor clicks “Add Student.”
 3. Search modal opens; instructor searches by name or email.
 4. System returns only students not already enrolled.

5. Instructor selects a student and confirms.

6. RPC `enroll_student_in_course` creates the enrollment; student appears immediately in the list.

- **Typical workflow (remove):**

1. Instructor clicks remove for a student.

2. Confirmation dialog warns about cascade deletion (submissions will be deleted).

3. Instructor confirms removal.

4. RPC `remove_student_from_course` executes; enrollment and associated submissions are deleted.

- **Validation & constraints:**

1. Enrollment add/remove uses RPC functions (`enroll_student_in_course`, `remove_student_from_course`).

2. Removal triggers cascade deletion for associated submissions (and related attempt/answer data as described later in the input).

Students

Track progress and performance.

Search students...

Add Student

NAME	PROGRESS	ASSIGNMENTS	LAST ACTIVE	
 Layla Ibrahim s1226774@najah.edu	88%	3/7	54 days ago	
 Fatima Ali s1231903@najah.edu	82%	2/7	55 days ago	
 Khalid Mahmoud s1203686@najah.edu	65%	2/7	55 days ago	
 Nour Salman s1224323@najah.edu	70%	2/7	55 days ago	
 Yusuf Ahmad s1211160@najah.edu	68%	2/7	56 days ago	
 Mariam Khaled s1237263@najah.edu	90%	2/7	54 days ago	
 Hassan Nasser s1246757@najah.edu	72%	2/7	55 days ago	
 Rania Fadi s1218918@najah.edu	80%	2/7	54 days ago	
 Tariq Sami s1245152@najah.edu	87%	2/7	54 days ago	
 Lina Ziad s1214151@najah.edu	75%	1/7	57 days ago	
 Bilal Rami s1236428@najah.edu	83%	1/7	54 days ago	

F

Fatima Ali

s1231903@najah.edu

OVERVIEW

82%

Progress

2/7

Assignments

RECENT ACTIVITY

Submitted Lab 3

2h ago

Logged in

5h ago

Add Student to Course

×

Search by name, email, or ID

Q a

D

Dina Waleed

s1232171@najah.edu

+

Enroll

S

Sara Ali mohammad

s12219691@stu.najah.edu

92%

3

F

Fatima Ali

s1231903@najah.edu

OVERVIEW

82%

Progress

2/7

Assignments

RECENT ACTIVITY

Submitted Lab 3

2h ago

Logged in

Add Student to Course

×

Search by name, email, or ID

Q Type to search...

Enter a student's name, email, or ID to search.

The search will return up to 3 matching results.

(Figure. 7) Student Enrollment Management

Admin Course Oversight (Fig. 8)

- **What it does:** Provides administrators with platform-wide visibility and control over all courses, including creation, modification, instructor reassignment, and lifecycle management.
- **Inputs:** Admin navigates to /admin/courses; creates a new course; edits course metadata; reassigns instructors; changes course status; deletes courses.
- **Outputs:** Tabular view of all courses showing core metadata (code, name, semester), assigned instructor (name/email), status indicators, and per-course action controls (edit, reassign, delete), along with modal interfaces for creating, updating, and confirming course operations.
- **Typical workflow:**
 1. Admin opens /admin/courses.
 2. System Admin navigates to /admin/courses; system fetches the full course catalog with metadata, instructors, and status.
 3. Interface renders a sortable/filterable courses table with loading and error states.
 4. Admin creates a course via modal form, entering metadata and assigning an instructor with validation.
 5. Admin edits existing courses through pre-filled forms, updating metadata and status.
 6. Admin reassigns instructors with permission and conflict checks.
 7. Admin changes course lifecycle state (draft/active/archived) according to business rules.
 8. Admin deletes courses through confirmation flow with impact validation
 9. System validates inputs, enforces constraints, persists changes, and refreshes the table with success/error feedback.
- **Validation & constraints:** Admin view is unfiltered across instructors (platform-wide access). The feature applies a centralized validation schema to enforce consistent course metadata, instructor assignment rules, and lifecycle state transitions across the platform. Critical constraints include globally unique course codes, permission-based instructor assignment, and controlled status changes to prevent invalid course states. Referential

integrity is preserved through dependency checks on enrollments and assessments, with all administrative actions validated and logged.

Course Catalog

+ Create course

Course Name	Code	Semester	Instructor	Status	Actions
electric_circuit	electrical_Circuit_123	2026	Dr. Mohammed Al-Ahmad	Active	EditDelete
electronic2	electronic2_123	2026	Dr. Mohammed Al-Ahmad	Active	EditDelete
digital3	digital3_123	2026	Dr. Mohammed Al-Ahmad	Active	EditDelete
Digital2	Dig2_123	2026	Dr. Mohammed Al-Ahmad	Active	EditDelete
Digital1	dig1_123	Spring 2026	Dr. Mohammed Al-Ahmad	Active	EditDelete
Shsh	Zsn	2030	Dr. Mohammed Al-Ahmad	Active	EditDelete
Math	Math123	2026	Dr. Mohammed Al-Ahmad	Active	EditDelete
DIG1	Digital Circuit Design I	spring 2024	Dr. Mohammed Al-Ahmad	Active	EditDelete
Electronics I	ECOM3301	Spring 2024	Dr. Mohammed Al-Ahmad	Active	EditDelete
Electrical Circuits I	ECOM2301	SPRING 2026	Dr. Mohammed Al-Ahmad	Active	EditDelete

(Figure. 8) Admin Course Oversight

Content Integration

What it does:

Implements a unified integration layer that connects course modules with external storage systems, AI processing pipelines, and simulation environments, enabling rich media delivery and interactive lab experiences within the learning workflow.

Lecture Video Integration (Google Drive)

Inputs:

- Instructor uploads a video file during lecture module creation, which is routed to Google Drive through the storage API.
- Module metadata determines upload handling and content binding.

Outputs:

- Google Drive file identifiers and embeddable URLs stored in the module record.
- Lecture modules rendered with embedded video playback in the student interface.

Typical workflow:

1. Instructor selects a video file in the lecture module form.
2. System uploads the file to Google Drive via OAuth2-authenticated API.
3. System generates public and embed URLs and stores them in the module.
4. Student interface embeds the video using the stored URL.

Slide Deck Integration + RAG Processing (Supabase + AI Pipeline)

Inputs:

- Instructor uploads PDF/PPTX slide decks during slides module creation, routed to Supabase Storage.
- Files are forwarded to the RAG back-end for content extraction and indexing.

Outputs:

- Public slide deck URLs stored in the module and metadata stored in the slide_decks table.
- Vectorized slide content available for AI-powered querying and contextual responses.

Typical workflow:

1. Instructor uploads slide deck in PDF/PPTX format.
2. System stores the file in Supabase Storage and generates a public URL.
3. System triggers RAG ingestion for text extraction and chunking.
4. Student interface displays slides, while the AI system enables content-grounded Q&A.

Lab Module Integration (Circuit Templates + Simulator)

Inputs:

- Instructor selects a circuit template when creating a lab module.
- Template identifiers are stored as references in the module record.

Outputs:

- Lab modules linked to simulator-ready circuit templates.
- Student simulator sessions initialized with forked circuit instances.

Typical workflow:

1. Instructor selects a circuit template during lab module creation.
2. System associates the module with the selected template via `template_id`.
3. Student opens the lab module and launches the simulator.
4. Simulator loads the template and forks it into the student's workspace.

Design constraints:

The integration layer follows a type-driven routing model with unified validation schemas and strict separation between storage, processing, and presentation responsibilities. Slide modules uniquely trigger automated ingestion through the Python back-end when `moduleType == "Slides"`, while other module types follow independent storage or simulation pipelines. Core constraints include file type and size limits, referential integrity for circuit template associations, consistent module–content binding, and abstraction of external services (Google Drive, Supabase, RAG back-end, simulator) behind stable service interfaces.

Discussion

Design Goals

The Course Management System was designed to support flexible content delivery without enforcing a single pedagogical format, while maintaining strict role boundaries and predictable system behavior.

Primary goals:

1. Structured but flexible course modeling

A simple two-level hierarchy (Course → Module) that reflects real academic structures while supporting heterogeneous content types.

2. Role-aligned capabilities

- Students: consume published content and track progress.
- Instructors: manage course structure, content lifecycle, and assessments.
- Admins: oversee platform-wide operations without scope restrictions.

3. Unified multi-content abstraction

Videos, slides, labs, quizzes, and assignments are treated as polymorphic “modules,” with type-driven upload, processing, and rendering pipelines.

Architectural Overview

Core Data Model

The system is built around a hierarchical and polymorphic schema, with key entities being:

- **courses:** owned by a single instructor (`instructor_id`) with lifecycle states (active, archived, draft).
- **course_modules:** ordered via `order_index`, visibility controlled by `module_status`, behavior determined by `module_type`.
- **course_enrollments:** join table enabling progress aggregation and dashboard analytics.

Polymorphic Module Behavior

Module behavior is driven by enums and type-specific fields:

- `module_type`: Lecture | Slides | Lab | Quiz | Assignment
- `module_status`: Draft | Published

Type-specific bindings

- **Lecture:** attachment_url → Google Drive embed link
- **Slides:** attachment_url → Supabase public URL; deck_id → RAG grounding
- **Lab:** template_id → circuit_templates
- **Quiz:** starts_at, ends_at, question relations
- **Assignment:** submission and grading relations

Content Integration Architecture

All uploads pass through a unified routing endpoint:

/api/storage/modules

The system dispatches content based on module type:

- **Lecture videos → Google Drive pipeline**
- **Slides → Supabase Storage + Python RAG ingestion**
- **Labs → Circuit template association**

This design decouples storage, processing, and presentation layers while preserving a consistent module abstraction.

Key System Mechanisms

Role-Based Views

Visible Modules	Scope	Role
Published only	Enrolled courses	Student
Draft + Published	Owned courses	Instructor
All	Platform-wide	Admin

Draft → Published Workflow

- Draft modules are instructor-only.

- Published modules become instantly visible to students.
- Student queries must filter by status, increasing query and UI complexity but enabling safe content staging.

Module Timeline Logic

- Modules ordered by `order_index`.
- Student state overlays: completed, in-progress, upcoming, overdue.
- Progress computed via aggregated completion signals (view, submit, pass).

Type-Driven Routing

front-end and back-end behavior is determined by `module_type`:

- Lecture → video viewer
- Slides → slide viewer + AI features
- Lab → simulator
- Quiz → runner
- Assignment → submission workflow

Media Pipelines

- **Google Drive (videos):** upload → permissioning → embed.
- **Supabase + RAG (slides):** upload → async ingestion → vectorization.
- Slide viewing remains functional even if ingestion fails.

Enrollment Lifecycle

- Managed via PostgreSQL RPC functions.
- CASCADE DELETE ensures referential integrity but makes removals irreversible.

4.7 Module Ordering

- Controlled via `order_index`.
- Reordering handled via RPC to avoid collisions and gaps.

Implementation Architecture

Database Layer

- Foreign keys + CASCADE DELETE enforce integrity.
- Enums enforce correctness.
- Polymorphic interpretation of shared fields (attachment_url, template_id).

Authentication & Authorization

- NextAuth + Google OAuth for sessions.
- Role checks in middleware and API endpoints.
- Instructor ownership enforced at API level.

Front-end State Management

- React hooks orchestrate modules, uploads, and modals.
- Optimistic updates with rollback.
- Shared modal state for CRUD flows.
- React Hook Form for validation and uploads.
- Yup for schema creation and validation

Back-end Service Layer

Core services:

- createInstructorModule
- uploadLectureVideoToDrive
- ingestSlidesModule

Storage and AI integrations abstracted behind service interfaces.

RPC & Aggregation

Key functions:

- get_student_dashboard_data
- get_instructor_course_detail
- enroll_student / remove_student
- update_module_order

RPC reduces N+1 queries and centralizes business logic.

Resilience & Performance

- Multi-layer validation (client → API → DB).
- Graceful degradation for external services.
- Indexing on course_id, status, order_index.
- Pre-aggregation via RPC.
- Public media buckets to reduce auth overhead.

Design Trade-offs

Costs	Benefits	Decision
Added filtering complexity	Safe content staging	Draft/Published model
Reorder maintenance	Instructor control	Manual ordering
External dependency	Better streaming	Google Drive external dependency
Async complexity	AI-powered slides	Dual storage in Supabase and ChromaDB adds complexity and latency.

Costs	Benefits	Decision
Irreversible data loss	Clean DB	Cascade deletion
No co-teaching	Simple authorization	Single instructor

Constraints & Limitations

- Fixed module types (ENUM-based extensibility cost).
- Binary visibility model (no scheduled publishing).
- Async RAG ingestion without progress tracking.
- Linear progress weighting.
- Upload size and pipeline limits.
- Python back-end dependency for ingestion.
- External service single points of failure.

Feature 2: Dashboard & Progress Tracking

High-level summary

The Dashboard & Progress Tracking system provides role-specific, aggregated views that help users understand status, prioritize work, and navigate with minimal cognitive overhead. It consolidates cross-course progress, deadlines, and feedback into unified dashboards, a calendar-based deadline view, and platform-wide admin analytics. Core outputs are dashboard interfaces (student/instructor/admin), a student calendar (month/week views), and an admin analytics surface powered by pre-aggregated RPC responses.

Feature list

- View a unified student dashboard with enrolled courses, prioritized upcoming tasks, progress indicators, resume items, recent feedback, and activity.

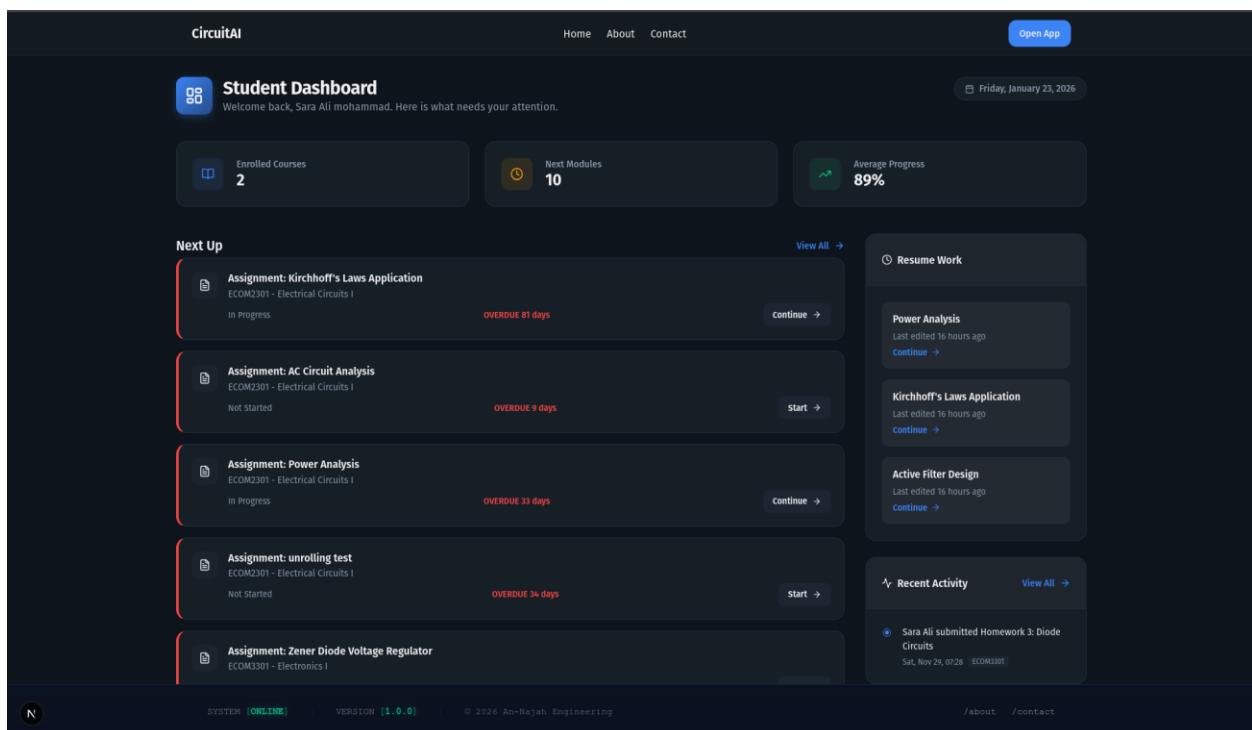
- Track assignment deadlines and quiz windows in a calendar (month/week views) with status cues and quick navigation.
- Monitor an instructor dashboard showing active courses, aggregate student counts, and pending grading workload.
- Access an admin home dashboard as a navigation hub to management areas.
- Review platform-wide analytics: engagement, course popularity, submission trends/status, and chat/AI usage metrics.

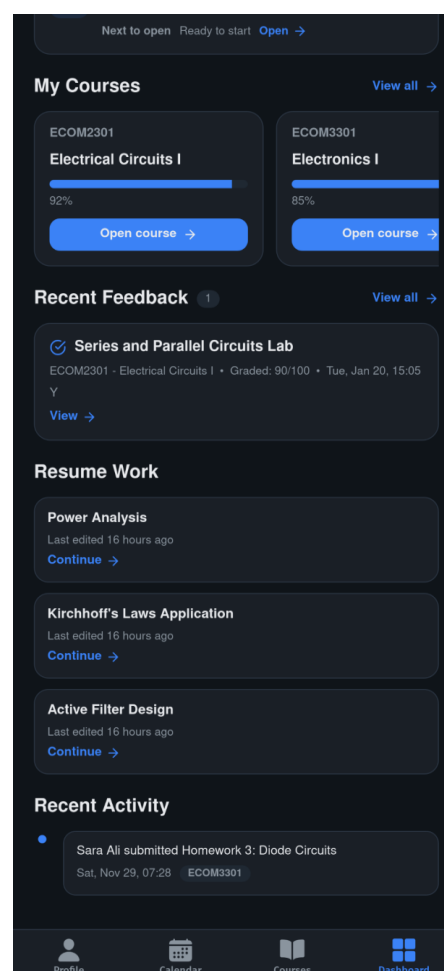
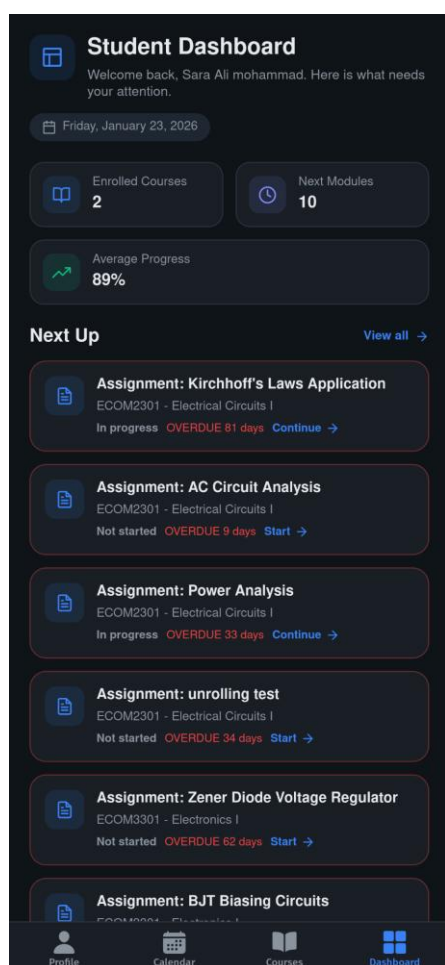
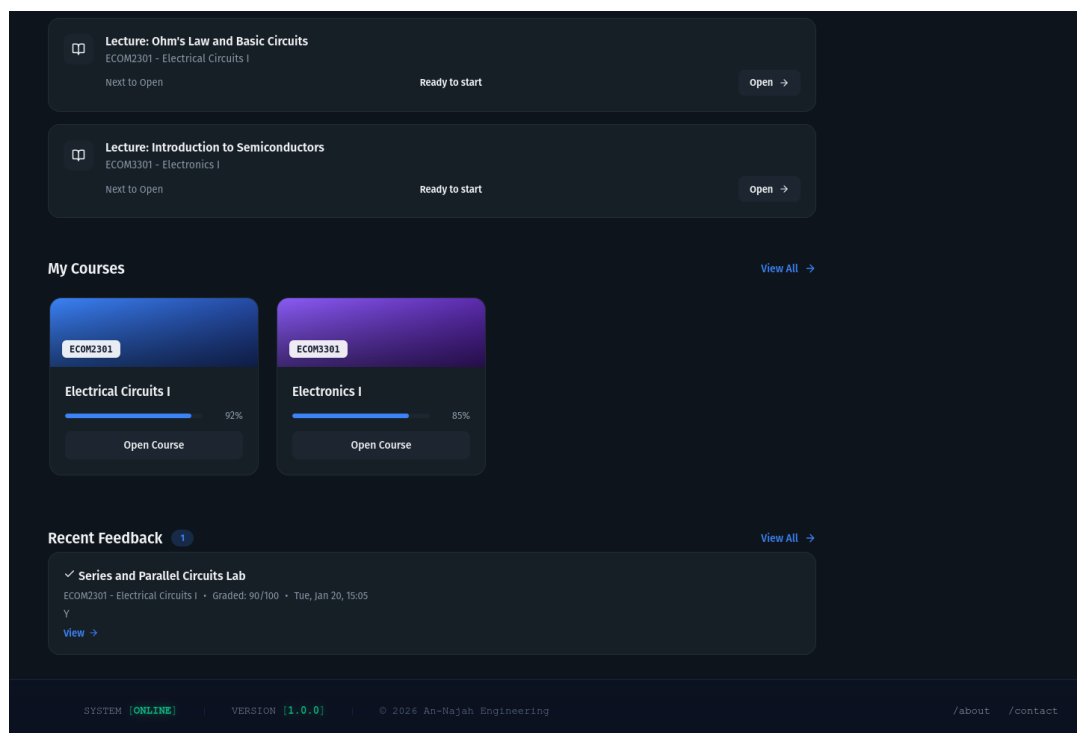
Sub-features

Student Dashboard (Fig. 9)

- **What it does:** Presents a single consolidated view across all enrolled courses: prioritized tasks (assignments/quizzes), progress, resumable work, recent feedback, and a recent activity timeline. Implements fixed urgency tiers (overdue / due-soon / normal) for clear prioritization. Uses a single optimized RPC call to minimize latency and DB round-trips.
- **Inputs:** Student navigates to /student/dashboard; system fetches dashboard payload via get_student_dashboard_overview(studentId) RPC.
- **Outputs:**
 - Header with greeting + current date
 - Stats grid: enrolled course count, next modules/tasks count, average progress
 - **Next Up:** prioritized tasks sorted by urgency then order index
 - **My Courses:** carousel of enrolled courses with progress indicators
 - **Recent Feedback:** graded work from last 7 days (score + instructor comments)
 - **Resume Work:** in-progress items (quiz/assignment/module) with unified prioritization
 - **Recent Activity:** chronological activity log
- **Typical workflow:**
 - Student opens /student/dashboard.
 - System calls get_student_dashboard_overview with the student ID.

- RPC returns a JSONB payload: enrollments, assignments, quizzes, feedback, resume items, activities, next modules.
- Front-end classifies urgency (overdue / due-soon / normal) for “Next Up”.
- Front-end merges resume sources into a unified list (Quiz > Assignment > Module).
- UI renders all sections with loading + empty states.
- Student clicks a task/resume item to navigate to the underlying assignment/quiz/module.
- **Validation & constraints:**
 - Student must be valid; data filtered to enrolled courses only.
 - Progress calculation is linear (all modules weighted equally).
 - Urgency thresholds are fixed: overdue if past due; due-soon if within 24 hours.
 - Resume work capped (4 most recent); recent feedback limited to last 7 days; activity list limited to most recent entries.



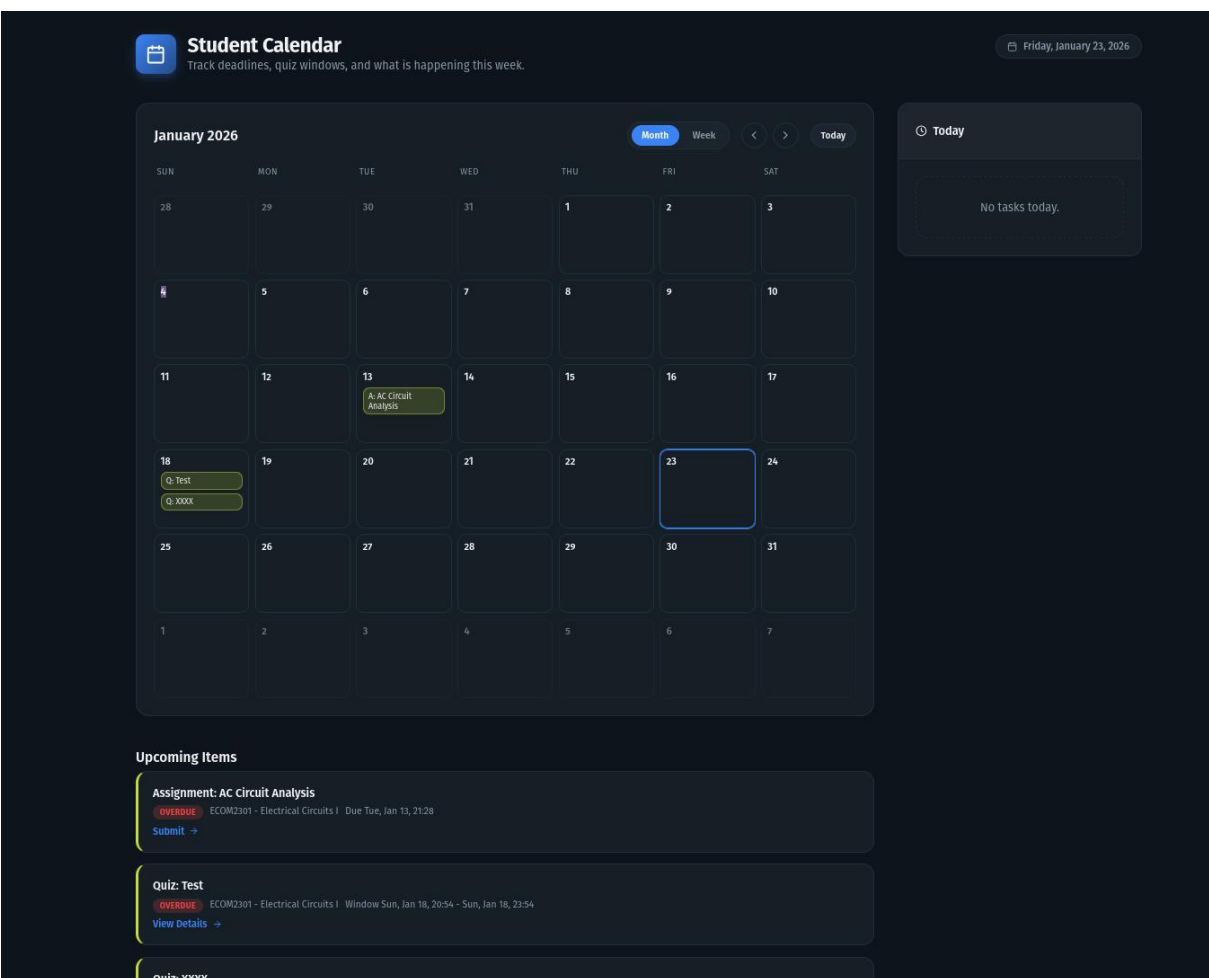


(Figure. 9) Student Dashboard

Student Calendar (Fig. 10)

- **What it does:** Aggregates assignment deadlines and quiz time windows across enrolled courses into a visual calendar with month/week modes. Uses deterministic per-course color assignment and event status (upcoming/overdue/completed). Displays clickable event chips with overflow handling on dense days.
- **Inputs:** Student navigates to /student/calendar; selects view mode (month/week); navigates time periods via prev/next or “Today”.
- **Outputs:**
 - Calendar header: current period label + view toggle
 - Calendar grid (month = 42-day grid, week = 7-day grid)
 - Event chips per day with course-coded styling + overflow (“+N more”)
 - **Upcoming Items:** list for visible range with status badges
 - **Today sidebar:** events occurring today + currently active quiz windows
 - Navigation controls: prev/next/today
- **Typical workflow:**
 - Student opens /student/calendar.
 - System fetches calendar payload via /api/student/calendar.
 - Front-end normalizes assignments + quizzes into unified CalendarEvent objects.
 - Front-end generates calendar grid dates based on focused date + view mode.
 - Events are grouped by date and rendered as chips in day cells.
 - Student switches month/week, navigates periods, or jumps to today.
 - Student clicks an event chip to open the assignment/quiz detail.
- **Validation & constraints:**
 - Calendar includes **assignments + quizzes only** (excludes lectures/labs/other modules).
 - Status requires valid due/windows + submission/attempt state.

- Color assignment is deterministic (hash-based) and may collide visually in edge cases.
- Overflow capped to 3 events per day with “+N more”.
- “Today” includes quizzes where now is within [starts_at, ends_at].



(Figure. 10) Student Calendar

Instructor Dashboard (Fig. 11)

- **What it does:** Provides an instructor overview across owned courses: active course list, aggregate student count, and assignments requiring grading (ungraded submission counts).

Prioritizes quick navigation into grading and course management. Includes a recent activity timeline.

- **Inputs:** Instructor navigates to /instructor/dashboard; system fetches dashboard payload via `get_instructor_dashboard_overview(instructorId)` RPC.
- **Outputs:**
 - Header with greeting + current date
 - Stats grid: total students, active courses, pending reviews
 - **Active Courses:** carousel with student counts and assignment counts
 - **Needs Review:** assignments with ungraded counts + due dates + “Grade Now”
 - **Recent Activity:** timeline of course/submission/grading events
- **Typical workflow:**
 - Instructor opens /instructor/dashboard.
 - System calls `get_instructor_dashboard_overview` with instructor ID.
 - RPC returns JSONB: active courses, assignments with ungraded counts, activity log.
 - Front-end computes aggregate stats (students, pending reviews).
 - UI renders carousels and activity with loading/empty states.
 - Instructor clicks “Grade Now” or a course card to navigate.
- **Validation & constraints:**
 - Instructor must be valid; data filtered to `courses.instructor_id = current_user`.
 - “Needs Review” limited to assignments with `ungraded_count > 0`.
 - Activity timeline shows most recent entries only.

Instructor Dashboard

Overview of your courses, students, and pending tasks.

Total Students

22

Active Courses

10

Pending Reviews

9

Active Courses

electrical_circuit_123

electric_circuit

0 Students

0 Assignments

Manage Course

electronic2_123

electronic2

0 Students

0 Assignments

Manage Course

digital3_123

digital3

0 Students

0 Assignments

Manage Course

Needs Review 12

Due Sun, Nov 2, 21:28

Kirchhoff's Laws Application

ECOM2301

1 Ungraded

Due Sun, Nov 9, 21:28

Series and Parallel Circuits Lab

ECOM2301

0 Ungraded

Due Fri, Nov 14, 21:28

BJT Biasing Circuits

ECOM3301

0 Ungraded

Recent Activity

Fatima Ali used your 'RC Circuit' template

Sat, Nov 29, 17:28 • ECOM2301

15 students completed Module 4

Sat, Nov 29, 16:28 • ECOM2301

Layla Ibrahim submitted Voltage Divider Design

Sat, Nov 29, 15:28 • ECOM2301

Mariam Khaled used your 'Series resistor network' template

Sat, Nov 29, 13:28 • ECOM2301

Tariq Sami submitted RC Circuit Analysis

Sat, Nov 29, 11:28 • ECOM2301

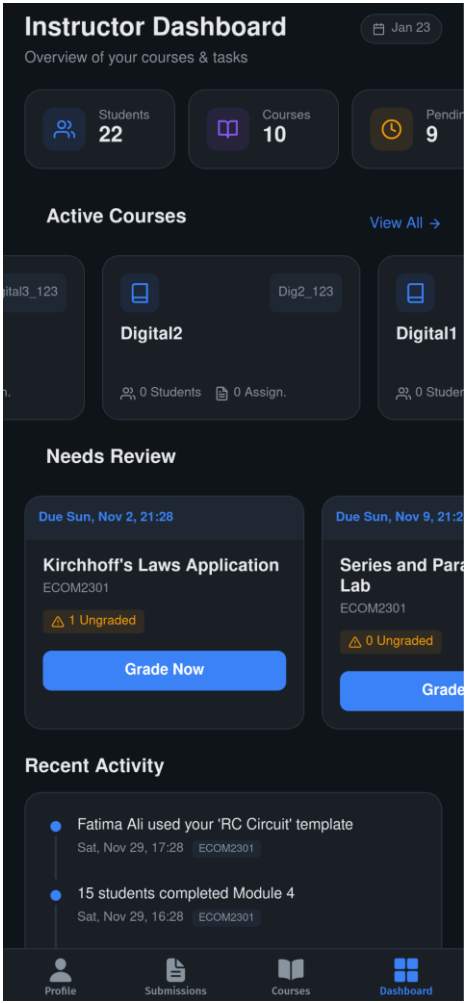
12 students completed Module 2

Sat, Nov 29, 09:28 • ECOM3301

Sara Ali submitted Homework 3: Diode Circuits

Sat, Nov 29, 07:28 • ECOM3301

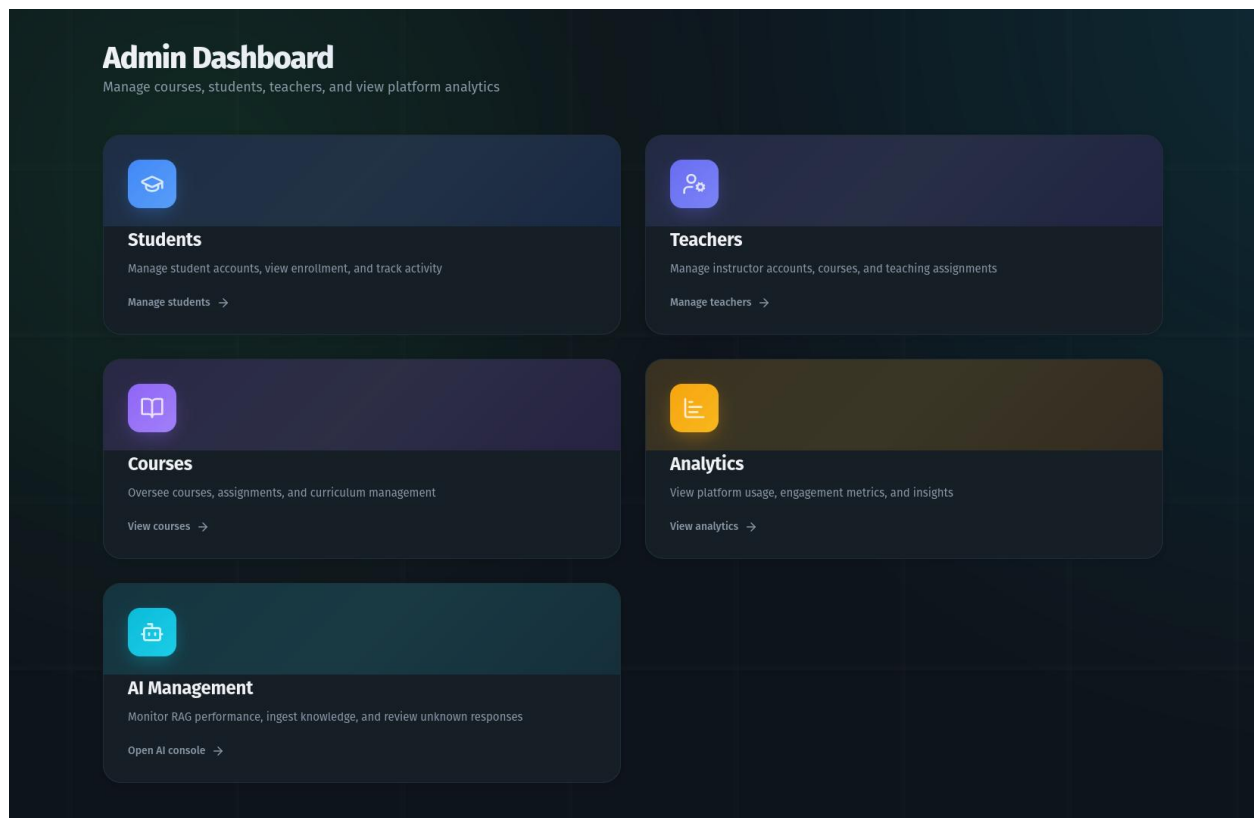
20 students completed Module 3



(Figure. 11) Instructor Dashboard

Admin Dashboard (Fig. 12)

- **What it does:** Acts as an admin control center and navigation hub to management areas (students, instructors, courses, analytics, AI management). This page is primarily a routing surface, not a data-heavy dashboard.
- **Inputs:** Admin navigates to /admin; clicks management cards.
- **Outputs:**
 - Header with title/description
 - Management cards: Students, Teachers, Courses, Analytics, AI Management
- **Typical workflow:**
 - Admin opens /admin.
 - UI renders static grid of navigation cards.
 - Admin clicks a card; system routes to the target page (e.g., /admin/students).
- **Validation & constraints:**
 - Admin role required.
 - No major data fetch required on this page (static navigation surface).

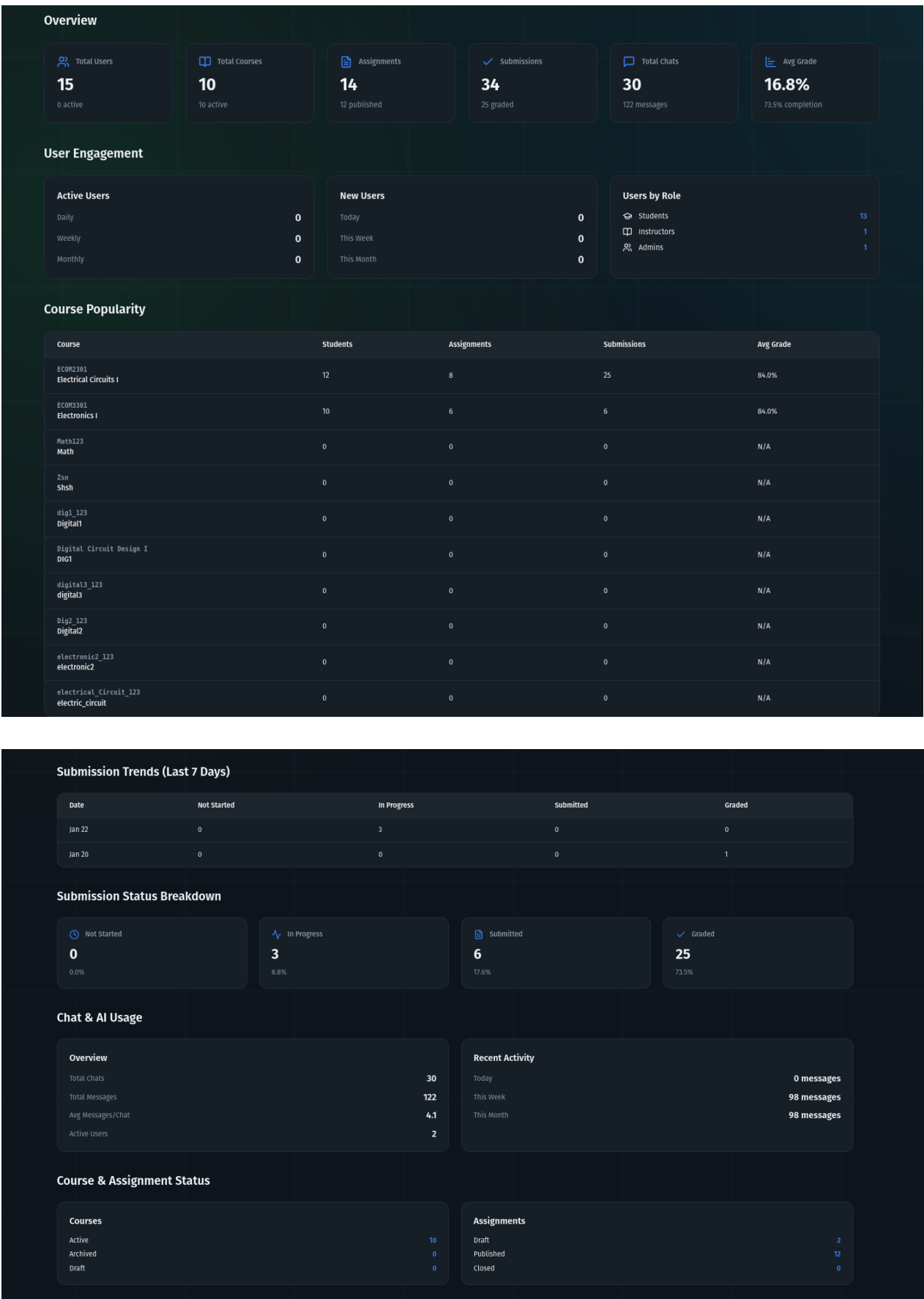


(Figure. 12) Admin Dashboard

Admin Analytics (Fig. 13)

- **What it does:** Provides platform-wide metrics and insights: engagement (DAU/WAU/MAU), course popularity, submission trends/status, chat + AI usage, and course/assignment status distributions. Powered by a single RPC aggregation call to reduce query overhead and centralize business logic.
- **Inputs:** Admin navigates to /admin/analytics; system fetches analytics payload via get_admin_analytics() RPC.
- **Outputs:**
 - Overview metrics: totals + active counts (users, courses, assignments, submissions, chats/messages), average grade, submission completion rate
 - User engagement: daily/weekly/monthly active, new users (time buckets), users by role
 - Course popularity table: course code/name, student count, assignment/submission counts, average grade

- Submission trends (last 7 days): date + counts by submission status
- Submission status breakdown: counts + percentages
- Chat & AI usage: totals, unique users, message volume (today/week/month)
- Course & assignment status distributions (active/archived/draft; published/draft/closed, etc.)
- **Typical workflow:**
 - Admin opens /admin/analytics.
 - System calls get_admin_analytics() (platform-wide, no params).
 - RPC returns JSONB covering all metric sections.
 - Front-end parses payload, computes percentages, formats values.
 - UI renders dashboards/tables with loading/empty handling.
 - Admin uses findings to navigate into management areas for action.
- **Validation & constraints:**
 - Admin role required; no filtering (platform-wide).
 - Submission trends limited to last 7 days.
 - Calculations must handle zero denominators safely.
 - Analytics are platform-wide only (no per-course drill-down without navigation).



Course Popularity

Submission Trends (Last 7 Days)

Date	Not Started	In Progress	Submitted	Graded
Jan 22	0	3	0	0
Jan 20	0	0	0	1

Submission Status Breakdown

Not Started

0

0.0%

In Progress

3

8.8%

Submitted

6

17.6%

Graded

25

73.5%

Chat & AI Usage

Overview

Total Chats

30

Total Messages

122

Avg Messages/Chat

4.1

Active Users

2

Recent Activity

Today

0 messages

This Week

98 messages

This Month

98 messages

Course & Assignment Status

Courses

Active

10

Archived

0

Draft

0

Assignments

Draft

2

Published

12

Closed

0

(Figure. 13) Admin Analytics

Discussion

Design goals

1. Single-glance status + priorities

Consolidate cross-course state into one place so users don't need to navigate multiple pages to understand what matters.

2. Reduce cognitive load via DB-side aggregation

Use optimized RPC functions for joins/aggregations and return precomputed payloads to reduce client complexity and DB round-trips.

3. Fast navigation to action

Task cards, event chips, and action buttons route directly to the relevant assignment/quiz/module/grading surfaces.

Architectural overview

Core aggregation model (RPC-first)

- **Student dashboard:** `get_student_dashboard_overview(p_student_id)` → JSONB with enrollments, tasks, feedback, resume items, activities, and “next modules”.
- **Instructor dashboard:** `get_instructor_dashboard_overview(p_instructor_id)` → JSONB with active courses, ungraded assignment counts, activity log.
- **Admin analytics:** `get_admin_analytics()` → JSONB with all platform metrics.

Role-specific data filtering

Core Filtering Behavior	Scope	Role
Access to published learning content and own submissions/attempts only	Enrolled courses	Student
Access limited to owned courses; grading workload derived from owned assignments	Owned courses	Instructor
No filtering (full access)	Platform-wide	Admin

Key system mechanisms

Student task prioritization

- **Urgency tiers (fixed):**
 - Overdue: due/end date in the past
 - Due soon: within 24 hours
 - Normal: everything else
- **Sorting:** overdue first → then by order_index (ascending).

Progress percentage

- **Formula:** $(\text{completed_modules} / \text{total_modules}) * 100$
- **Model:** linear weighting; all modules treated equally.

Resume work unification

- Sources merged into one list:
 - in-progress assignments (submission status = in_progress)
 - active quiz attempts (completed_at IS NULL)
 - recently opened modules (from activity log)
- Priority ordering: **Quiz > Assignment > Module**
- Cap: **4 items**

Student calendar event model

- Assignment event date: due_date
- Quiz event date: ends_at (or starts_at if not started yet)
- “Today” includes active quiz windows: $\text{now} \in [\text{starts_at}, \text{ends_at}]$
- Overflow: show up to 3 chips/day, then +N more

Instructor pending review logic

- Ungraded count per assignment:

- submissions where status in (not_started, in_progress, submitted) and grade IS NULL
- “Needs Review” includes only assignments where ungraded_count > 0.

Admin analytics aggregation (platform-wide)

- Metrics computed via aggregated counts/averages across core tables (profiles/courses/assignments/submissions/chats/messages).
- Submission trends grouped by date and status for the last 7 days.
- Popularity sorted by student_count DESC.

Implementation architecture

Database layer

- RPC functions return JSONB payloads to minimize API chattiness.
- Query optimization patterns:
 - CTEs for complex joins/aggregations
 - Index usage on foreign keys and status fields
 - STABLE volatility to support plan caching where applicable

API layer

- Endpoints as thin wrappers around service calls, examples:
 - GET /api/student/dashboard?studentId=...
 - GET /api/student/calendar?studentId=...
 - GET /api/instructor/dashboard?instructorId=...
 - GET /api/admin/analytics

Front-end state management

- Hooks for fetch lifecycle: loading/error/data
- Derived computation via memoization (urgency classification, event grouping, grid generation)
- UI patterns: skeleton loaders, empty states, navigation-first actions

Resilience & performance

- **Single RPC call per dashboard** reduces DB round-trips and centralizes aggregation logic.
- **Graceful degradation:** empty states when no data; dashboards remain usable without real-time streaming.
- **Freshness:** data fetched on page load; requires refresh for real-time changes (no WebSocket/SSE).

Design trade-offs

Costs	Benefits	Decision
Larger JSONB payload; less flexible; harder debugging	Reduced latency; consistent aggregation; fewer round-trips	Single RPC payload per dashboard
Fixed thresholds (24h “due soon”) may be arbitrary	Clear prioritization; easy scanning	Three-level urgency system
No day view for dense schedules	Covers common cases; simpler UI	Month/week calendar only
Doesn’t reflect real effort differences by module type	Predictable and simple	Linear progress weighting
Adds client-side compute (mitigated with memoization)	Flexible presentation	Client-side event normalization
Higher server load; slower repeat visits	Always fresh on load	No client-side caching

Constraints & limitations

- **No real-time updates:** dashboards can become stale if left open; refresh required.
- **Large JSONB payloads:** may impact performance for heavy users (many enrollments/courses).
- **Progress model limitations:** equal weighting; no partial-credit awareness; updates may lag depending on trigger/RPC timing.

- **Calendar scope:** assignments + quizzes only; overflow may hide same-day deadlines; deterministic colors can collide.
- **Analytics scope:** platform-wide only; trends limited to last 7 days; calculated on-demand (may be slow at scale); limited historical comparisons.

Feature 3: Assignment System

High-level summary

The Assignment System manages the full assignment lifecycle—**creation** → **draft work** → **submission** → **grading**—with a **multi-modal submission model** (text, files, circuit links) and an explicit **submission state machine**. Students can save drafts repeatedly to prevent data loss, submit when ready, and then view grades/feedback after evaluation. Instructors manage assignments, review submissions (per-assignment and cross-course), and grade using manual inputs or **AI auto-grading suggestions** (assistive layer only; instructors retain final authority). The system integrates with the circuit simulator via **template associations** so lab-style assignments can start from a predefined circuit and end with a linked student circuit artifact.

Feature list

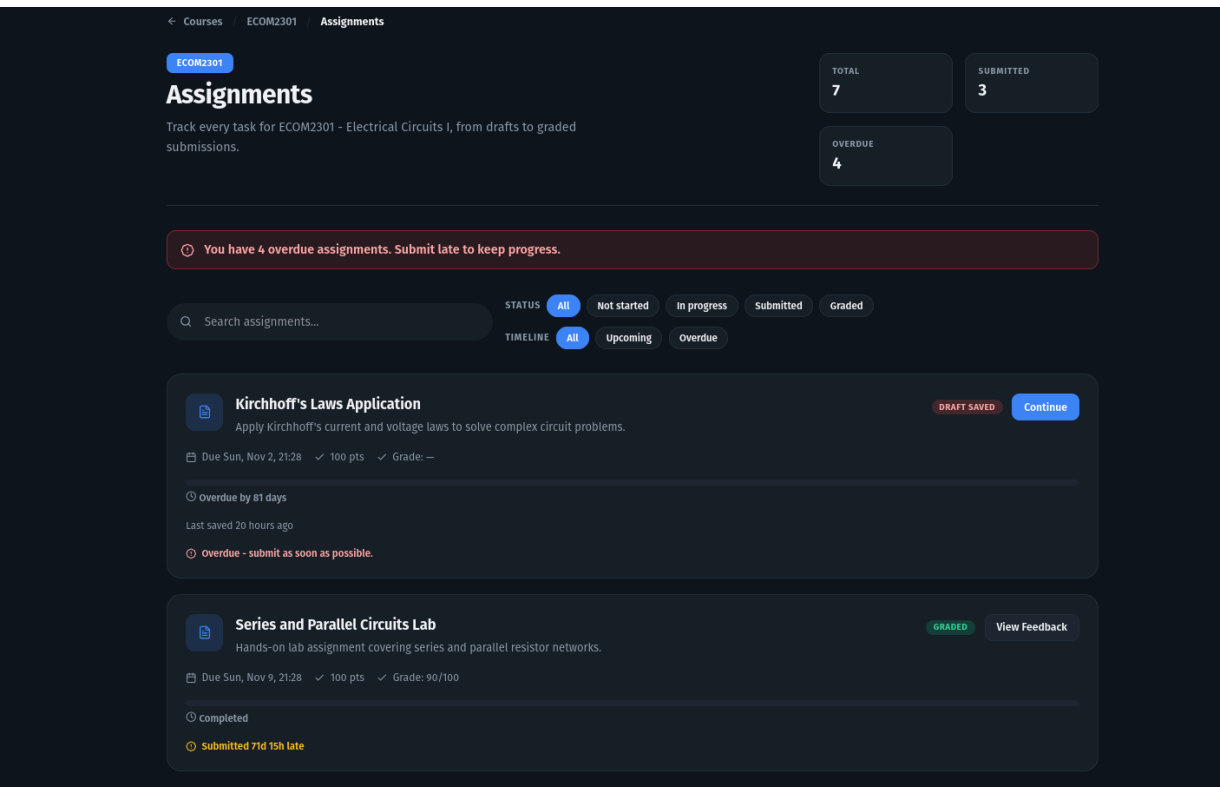
- List course assignments with status tracking, search, and filters.
- Support multi-modal submissions:
 - **Text content**
 - **File uploads** (PDF, DOCX, PPTX, images, ZIP/RAR; up to **50MB**)
 - **Circuit links** (manual circuit ID entry)
- Save drafts (unlimited) with persistence and safe recovery.
- Track submission history/milestones (draft saves, submission, grading).
- Instructor management: create/edit assignments, configure due dates/max points, attach templates, set visibility (draft/published/closed).
- Instructor review: per-assignment submissions table with filtering and Excel export.
- Cross-course pending submissions view for efficient grading workflow.
- Grading UI with manual grading + AI auto-grade suggestions (review/edit before finalization).

- Circuit template integration: fork template → build → link circuit to submission → instructor preview.

Sub-features

Student Assignment List (Fig. 14)

- **What it does:** Student navigates to /student/courses/[courseId]/assignments. system shows all assignments for a course in a card layout with status, progress cues, overdue highlighting, and filters (status + timeline) plus search.
- **Inputs:** enrollment + courseId; optional search; status filter; timeline filter.
- **Outputs:** assignment cards (title, due date, points, status badge, progress, time remaining/overdue, grade when available), summary stats, and actions (Start / Continue / View Grade).
- **Typical workflow:**
 1. Student opens the assignments page for a course.
 2. System fetches assignments + student submission status per assignment.
 3. UI derives overdue/due-soon metadata and renders cards.
 4. Student filters/searches.
 5. Student opens an assignment detail page via action button.
- **Validation & constraints:**
 1. Requires valid course enrollment.
 2. Status derived from submission record existence + submission status field.
 3. Overdue determined by due_date vs current time.
 4. Progress is mapped from submission status (not_started→0%, in_progress→50%, submitted/graded→100%).
 5. Filtering/search performed client-side after fetch.



(Figure. 14) Student Assignment List

Student Assignment Detail & Submission (Fig. 15)

- **What it does:** Students visit `/student/courses/[courseId]/assignments/[assignmentId]` , then the system provides the submission workspace (text + file upload + circuit ID) with draft saving, submission validation, milestone timeline, template launch (when applicable), and post-grading feedback display.
- **Inputs:**
 - Student enters text content
 - Uploads a file via drag/drop or picker
 - Pastes a circuit ID from the simulator
 - Clicks **Save Draft** or **Submit**
- **Outputs:**

- Assignment header + due/overdue indicators + points + status (and grade if graded)
- Description/instructions + instructor attachments
- Template card + “Open in Simulator” link (if template associated)
- Submission controls (text area, file upload, circuit ID input)
- Draft/save/submit actions
- Timeline view of key milestones (started, last saved, submitted, graded)
- Feedback section (grade + instructor comments) after grading
- Lock banners when editing is disallowed
- **Typical workflow:**
 - Student opens assignment detail; system loads assignment + submission (if exists).
 - Form fields auto-populate from saved submission content.
 - Student reviews instructions and any attached files.
 - If a template exists, student opens simulator and forks a starting circuit.
 - Student works via any combination of text/file/circuit link and saves drafts repeatedly.
 - Student submits; system validates at least one mode contains content.
 - Submission becomes reviewable for grading; editing rules apply based on status/late policy.
 - After grading, student sees grade + feedback.
- **Validation & constraints:**
 - Submission requires at least one of: text content, file attachment, or circuit ID.
 - File size limit: **50MB**. Types: PDF, DOCX, PPTX, images, ZIP, RAR.
 - Circuit ID validated on submission (must reference an existing circuit in DB).
 - Draft saves allowed unlimited times; overwrites prior draft (no draft version history specified).
 - Editing locks after grading; late submission locking rules apply as described (late detection via submitted_at vs due_date).

←

Courses

/

ECOM2301

/

Assignments

/

Series and Parallel Circuits Lab

ECOM2301

GRADED

Series and Parallel Circuits Lab

Work on your submission for ECOM2301 - Electrical Circuits I and keep track of every milestone.

DUE

Sun, Nov 9, 21:28

Completed

POINTS

100

Max score

GRADE

90/100

Latest status

Submitted 71d 15h late

Description

Hands-on lab assignment covering series and parallel resistor networks.

Instructions

Build and measure three different resistor networks: one series, one parallel, and one combination. Measure total resistance, current, and voltage drops. Compare measured values with calculated values and explain any discrepancies.

Attached Files

No supporting files provided.

Submission Workspace

SUBMISSION NOTES

Write your solution, approach, or notes...

Upload submission file

PDF, DOCX, PPTX, Images, ZIP/RAR up to 50MB

Current submission: HW2.pdf

Select file

HW2.pdfRemove

HW2.pdfRemove

CIRCUIT TEMPLATE

Template ID: 0f111111-1111-1111-1111-111111111111

View template

CIRCUIT ID

Paste your circuit ID once saved

This assignment has been graded and cannot be edited.

Save DraftSubmit

Submission History

STARTED

Tue, Jan 20, 12:55

LAST SAVED

3 days ago

SUBMITTED

Tue, Jan 20, 12:55

GRADED

Tue, Jan 20, 15:05

GRADE

90/100

Feedback

GRADE

90/100

STATUS

Graded

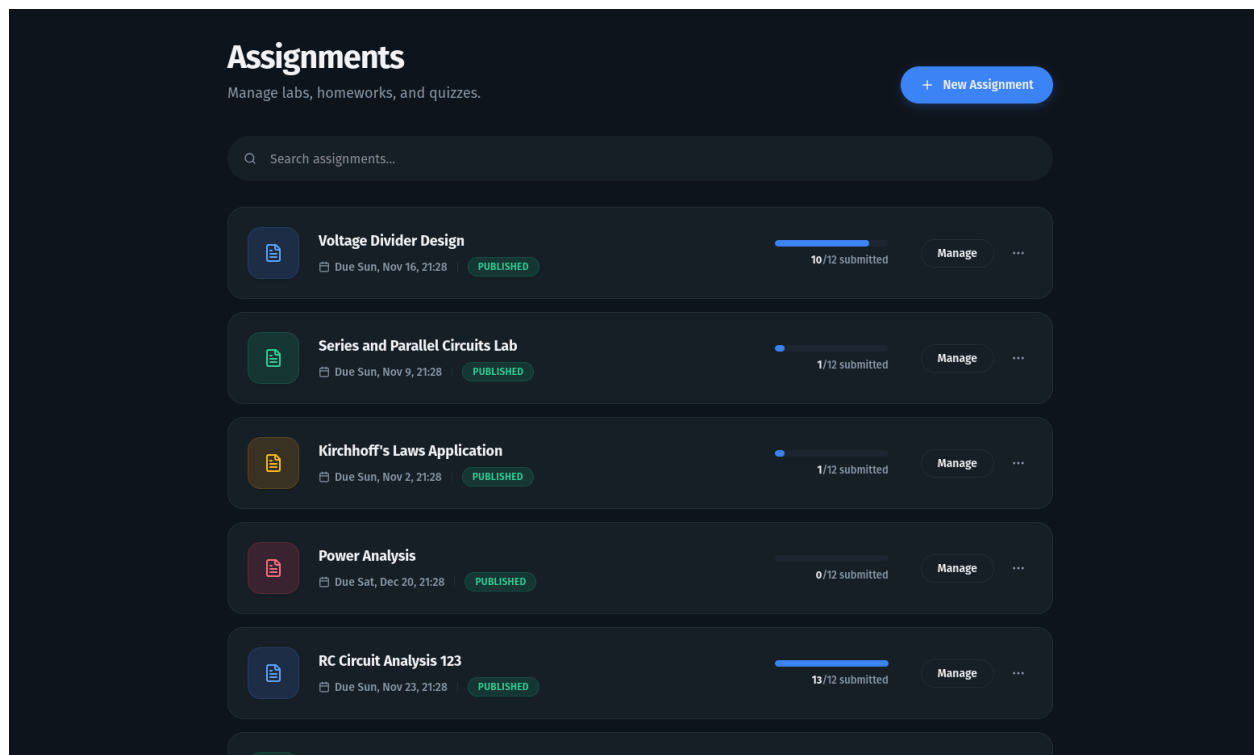
Good work on the series and parallel circuits lab overall. The resistor networks were correctly constructed, and the measured values for resistance, current, and voltage were largely consistent with calculated results, demonstrating a solid understanding of basic circuit principles. The discussion of discrepancies was adequate but could have been more detailed, particularly in addressing experimental error and component tolerances. Despite the late submission, the work shows strong effort and competence in both measurement and analysis

43

(Figure. 15) Student Assignment Detail & Submission

Instructor Assignment List (Fig. 16)

- **What it does:** User routes to `/instructor/courses/[courseId]/assignments`, then the system displays all assignments in a course for instructors with status badges, submission stats (submitted vs enrolled), quick actions, and a modal workflow for creation/editing.
- **Inputs:** instructor navigates; optional search; clicks New Assignment; manage/edit/delete actions.
- **Outputs:** assignment cards (title, due date, status: draft/published/closed, submission progress, actions), creation/edit/delete modals with validation.
- **Typical workflow:**
 1. Instructor opens the course assignments page.
 2. System fetches assignments + aggregated submission counts.
 3. Instructor creates an assignment (title, instructions, due date, max points, optional template, status).
 4. Instructor opens “Manage” to view submissions, or edits/deletes via card actions.
- **Validation & constraints:**
 1. Requires instructor ownership of the course.
 2. Due date must be valid; max points must be positive.
 3. Status rules:
 - **draft:** hidden from students
 - **published:** visible to students
 - **closed:** no new submissions accepted
 4. Submission stats computed via aggregation across enrollments/submissions.



(Figure. 16) Instructor Assignment List

Instructor Assignment Submissions View (Fig. 17)

- **What it does:** Instructor navigates to `/instructor/courses/[courseId]/assignments/[assignmentId]`, system shows all submissions for a single assignment in a table with grading-status filters and Excel export.
- **Inputs:** filter selection (all/to grade/graded); row click; export click.
- **Outputs:** submissions table (student, status, submitted date, grade), filter controls, export file, empty state.
- **Typical workflow:**
 1. Instructor opens submissions from assignment list.
 2. System fetches submissions + student info.
 3. Instructor filters to “to grade” and opens a submission for detailed review.
 4. Instructor exports visible results to Excel if needed.
- **Validation & constraints:**

1. Requires instructor ownership of the course/assignment.
2. Filtering performed client-side after fetch.
3. Table includes enrolled students including not_started (as specified).

← Back to Assignments

Published

Edit Settings Grade All

Voltage Divider Design

ECOM2301 - Electrical Circuits I Due Nov 16, 2025, 09:28 PM

Student Submissions				All (10)	To Grade (1)	Graded (9)
STUDENT	SUBMITTED	STATUS	GRADE			
R Rania Fadi	Nov 17, 01:28 PM	Graded	92/100			Review
H Hassan Nasser	Nov 17, 11:28 AM	Graded	78/100			Review
M Mariam Khaled	Nov 17, 09:28 AM	Graded	91/100			Review
Y Yusuf Ahmad	Nov 17, 07:28 AM	Graded	74/100			Review
N Nour Salman	Nov 17, 05:28 AM	Graded	76/100			Review
K Khalid Mahmoud	Nov 17, 03:28 AM	Graded	71/100			Review
F Fatima Ali	Nov 17, 01:28 AM	Graded	87/100			Review
L Layla Ibrahim	Nov 16, 11:28 PM	Graded	94/100			Review
S Sara Ali mohammad	Nov 16, 07:28 PM	Graded	92/100			Review

ASSIGNMENT DETAILS

Description
Students will design and implement voltage divider circuits for specific output voltage requirements.

Instructions
Design a voltage divider circuit that produces 3.3V from a 5V input. Choose appropriate resistor values and verify your design using simulation. Submit the circuit schematic, calculations, and simulation results showing the output voltage.

Linked Template
Template ID: 0f111111-1111-1111-1111-11111113

PERFORMANCE

(Figure. 17) Instructor Assignment Submissions View

Instructor Cross-Course Submissions (Fig. 18)

- **What it does:** Instructor goes to /instructor/submissions, instructor aggregates pending submissions across all courses owned by the instructor (assignments + quizzes), enabling a unified grading queue with filtering and search.
- **Inputs:** course filter; type filter (assignments/quizzes); search (student/course/assignment); open review action.
- **Outputs:** consolidated submissions table with navigation to the correct grading view per submission type.
- **Typical workflow:**
 1. Instructor opens cross-course submissions.
 2. System fetches data via RPC aggregating instructor-owned course submissions.
 3. Instructor filters by course/type, searches, and opens specific items to grade.

- **Validation & constraints:**

1. Requires instructor role; aggregation limited to courses where instructor_id matches.
2. Filtering/search performed client-side after fetch.

TYPE	TITLE	STUDENT	COURSE	SUBMITTED
Assignment	RC Circuit Analysis 123 Due Sun, Nov 23, 21:28	Dina Waleed	ECOM2301 Electrical Circuits I	Sat, Nov 29, 17:28 55 days ago
Assignment	RC Circuit Analysis 123 Due Sun, Nov 23, 21:28	Bilal Rami	ECOM2301 Electrical Circuits I	Sat, Nov 29, 15:28 55 days ago
Assignment	RC Circuit Analysis 123 Due Sun, Nov 23, 21:28	Lina Ziad	ECOM2301 Electrical Circuits I	Sat, Nov 29, 13:28 55 days ago
Assignment	Homework 3: Diode Circuits Due Fri, Nov 28, 21:28	Nour Salman	ECOM3301 Electronics I	Sat, Nov 29, 00:28 55 days ago
Assignment	Homework 3: Diode Circuits Due Fri, Nov 28, 21:28	Fatima Ali	ECOM3301 Electronics I	Fri, Nov 28, 22:28 55 days ago
Assignment	Voltage Divider Design	Taria Sami	ECOM2301	

(Figure. 18) Instructor Cross-Course Submissions

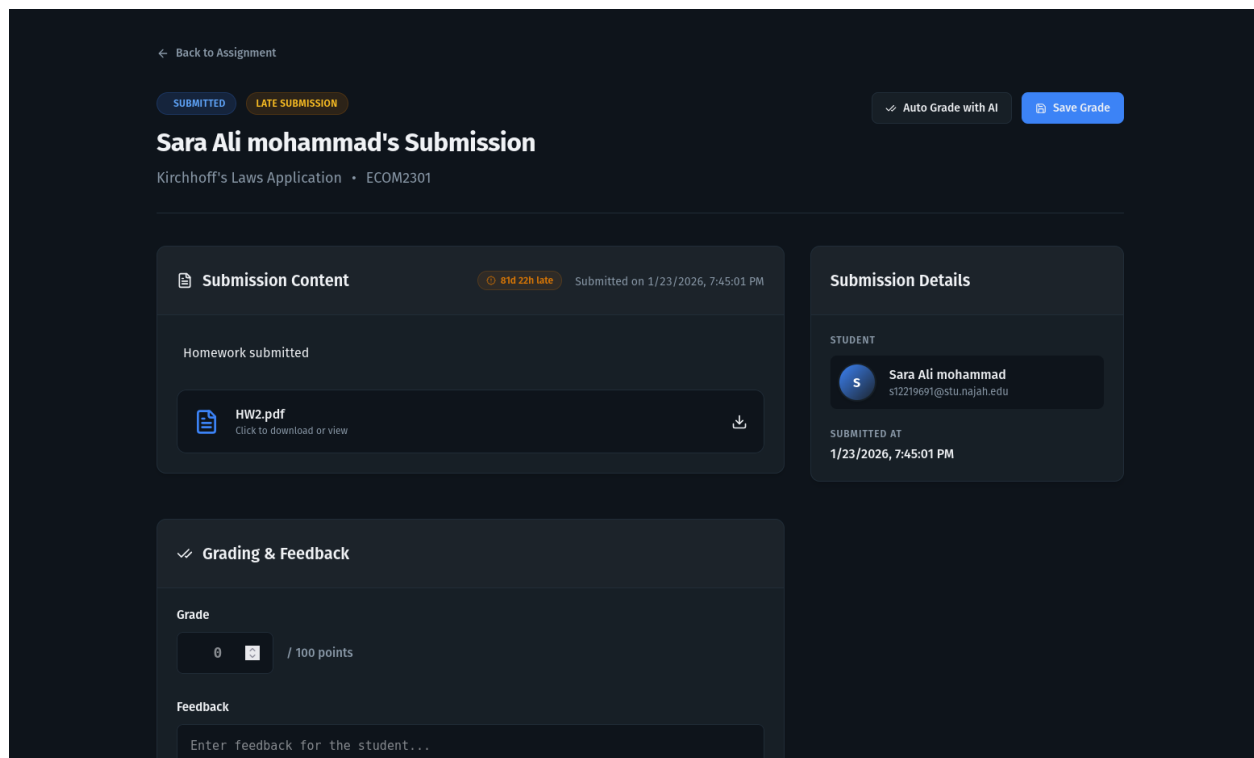
Instructor Submission Grading (Fig. 19)

- **What it does:** Instructor navigates to /instructor/submissions/[submissionId], system displays all submission artifacts (text, file, circuit link) and provides grading controls. Supports manual grading and AI auto-grading suggestions that prefill grade/feedback fields for instructor review before final submission.
- **Inputs:** instructor reviews; clicks Auto-Grade; edits suggested grade/feedback; submits grade.
- **Outputs:** submission content viewer(s), grading form (grade 0..max_points + feedback), AI suggestion results, finalization action + success/error states, grade history section (if present).
- **Typical workflow:**

1. Instructor opens a submission from a submissions list.
2. System loads submission + assignment benchmark + student metadata.
3. Instructor reviews text/file/circuit artifacts.
4. Instructor optionally clicks **Auto-Grade** to generate suggestions.
5. Suggestions populate grade/feedback fields; instructor edits as needed.
6. Instructor submits grade; system marks submission graded and locks student editing.

- **Validation & constraints:**

1. Requires instructor ownership of the course/assignment.
2. Grade must be numeric within **[0, max_points]**.
3. Auto-grading requires:
 - Benchmark context: at least one of description/instructions/assignment attachment URL
 - Submission content: at least one of submission text or submission attachment URL
4. AI suggestions are editable and do not finalize grading by themselves.
5. Late detection: submitted_at vs due_date.



(Figure. 19) Instructor Submission Grading

Discussion

Design goals

1. Flexible submission formats

Support varied coursework (written reasoning, uploaded reports, interactive circuit artifacts) without forcing a single submission modality.

2. Reduce grading burden without removing instructor control

AI generates suggested grades/feedback, but instructors review and finalize.

3. Clear, enforceable submission state

Explicit state machine removes ambiguity about what's draft vs submitted vs graded.

4. Seamless simulator integration

Templates act as assignment starting points; final work can be verified via linked student circuits. Architectural overview

Core data model

- assignments: course-owned, includes lifecycle state (draft/published/closed), due_date/max_points, description/instructions, optional template_id, optional attachment_url.
- submissions: one per (assignment, student) with status (not_started/in_progress/submitted/graded), content fields (text + attachment URL + circuit_id), and grading fields (grade/feedback/graded_at/graded_by).
- circuits + circuit_templates: support template association and circuit linking.

Submission state machine

not_started → in_progress → submitted → graded

- not_started: no submission record exists.
- in_progress: record exists (draft saved).
- submitted: final submission with submitted_at.
- graded: grade/feedback applied with graded_at.

Content storage

- Text: stored in DB (submissions.content).
- Files: stored in Supabase Storage (submissions bucket), URL stored in submissions.attachment_url.
- Circuits: stored by ID reference (submissions.circuit_id → circuits.id).

Key system mechanisms

Draft persistence

- Draft saves upsert a submission with status = in_progress.
- Saves include all active modalities (text/file URL/circuit ID).
- Draft saving is repeatable and designed to prevent data loss.

Submission validation + locking

- Submit requires at least one modality populated.
- Lock rules prevent edits after grading (and late-graded submissions lock as specified).
- UI explains lock reasons to reduce confusion.

AI auto-grading (suggestion layer)

- Triggered explicitly by instructor.
- Next.js API route forwards to Python backend.
- Backend downloads/extracts text from attachments (when present), constructs:
 - Benchmark: description + instructions + assignment attachment text (if available)
 - Submission: student text + submission attachment text (if available)
- Backend calls Groq LLM and returns {grade, feedback}; grade is validated/clamped to range.
- Frontend pre-fills fields; instructor edits before final submission.

Circuit integration workflow

- Instructor selects template on assignment creation (template_id stored).
- Student opens template via “Open in Simulator”, forks to a personal circuit, completes work.
- Student links circuit via circuit ID input; validated on submission.
- Instructor opens linked circuit directly from grading UI.

Design trade-offs

- **Multi-modal submissions:** flexibility vs more validation and UI complexity.
- **Draft system:** prevents data loss vs potential student confusion (draft $\neq$ submitted).
- **AI grading as suggestion:** reduces workload while preserving authority vs requires review step.
- **Circuit linking by manual ID:** simplest integration vs copy/paste friction (no direct handoff).
- **Lock-after-grading:** integrity and reduced disputes vs limits regrade/iteration workflows.

Constraints & limitations

- **Files:** 50MB hard limit; fixed allowed types; no attachment version history.
- **Drafts:** no draft versioning/history; overwrites prior state.
- **Circuit integration:** manual ID entry; student ownership enforcement is backend-side; no embedded preview in submission UI.

- **Grading:** single numeric grade + freeform feedback; no rubric breakdown; no batch grading; audit trail for grade changes not specified.
- **Operational:** AI auto-grading depends on Python backend availability; large uploads can be slow; large course submission lists may render slowly (no pagination specified).
- **Late policy:** timestamp-based; no explicit grace period/extension workflow specified.

Feature 4: Quiz System

High-level summary

The Quiz System delivers end-to-end assessments as a first-class module type. It supports three question types (multiple choice, true/false, short answer), enforces time-based availability windows, and maintains attempt state across sessions through immediate answer persistence. Objective questions are auto-graded on submission, while short answers flow into instructor grading. Students get a low-friction quiz-taking experience (free navigation + recovery), and instructors get full authoring tools—including AI-powered quiz generation grounded in slide decks—with edit-lock constraints that preserve fairness. Primary outputs are instructor quiz creation/editing surfaces, student quiz-taking flows, auto/manual grading, and results review controlled via per-attempt release.

Feature list

- Show quiz modules in the course module timeline with status + time window indicators.
- Take quizzes with free navigation, immediate answer persistence, and session recovery.
- Support question types: multiple choice, true/false, short answer.
- Visual progress tracking via a question navigation sidebar (answered/unanswered).
- Submit attempts with confirmation + auto-grading for objective questions.
- Review results with per-question breakdown, explanations, and release gating.
- Instructor manual quiz authoring with question editor, points, and explanations.
- AI-generated quizzes from selected slide decks with configurable counts per type.
- Edit quizzes pre-start: reorder questions, adjust points, add explanations.
- View attempt lists with statuses (in progress / needs grading / graded) and scoring.

- Manual grading workflow for short answers with per-attempt results release.
- Enforce quiz availability via starts_at / ends_at windows.
- One attempt per student per quiz via DB uniqueness.

Sub-features

Student Quiz List (Integrated in Course Detail)

- **What it does:** Renders quizzes as modules inside the ordered course timeline (alongside lectures/labs/assignments). Communicates availability and attempt state using status badges and time window messaging.
- **Inputs:** Student navigates to /student/courses/[courseId]; quiz modules appear in timeline; student clicks a quiz module.
- **Outputs:** Timeline card showing title/description, status (not started / in progress / completed), time window (opens/closes), availability message (pre-start / open / closed), attempt indicator, and quick action (Start / Continue / View Results).
- **Typical workflow:**
 1. Student opens course detail.
 2. System fetches published modules (including quiz modules with starts_at, ends_at).
 3. UI computes availability from now vs. starts_at/ends_at.
 4. UI computes attempt status from an existing quiz_attempt and completed_at.
 5. Student clicks the quiz module to enter the quiz flow.
- **Validation & constraints:**
 1. Requires valid course enrollment.
 2. Students only see **published** quiz modules.
 3. Availability derives from timestamp comparisons; attempt derives from existence of quiz_attempt + completed_at.

Student Quiz Taking (Entry Page) (Fig. 20)

- **What it does:** Serves as the quiz entry/landing state machine: not started (countdown), available (start/continue), finished (results flow), locked (window ended without attempt). Enforces time windows while preserving resume for in-progress attempts.
- **Inputs:** Student navigates to `/student/courses/[courseId]/modules/[quizId]/quiz`; student clicks Start/Continue; system checks window + attempt.
- **Outputs:** Quiz info (title/description), window display, countdown (pre-start), Start button (during window), Continue button (if in progress), finished messaging/results route, locked message (after end without attempt).
- **Typical workflow:**
 1. Student opens quiz entry page.
 2. System loads `starts_at/ends_at` and existing attempt (if any).
 3. System classifies state: not started / available / finished / locked.
 4. Student starts or continues.
 5. System creates or retrieves attempt and routes to question 1.
- **Validation & constraints:**
 1. Requires enrollment.
 2. Access rules: quiz usable when $\text{starts_at} \leq \text{now} \leq \text{ends_at}$ (**except continuing an in-progress attempt**).
 3. Attempt uniqueness: `(module_id, student_id)` prevents multiple attempts.
 4. Resume only if `completed_at` is null.
 5. Locked if time ended and no attempt exists.



Quiz Unavailable

This quiz is currently locked. Please check back at the scheduled start time.



OPENS

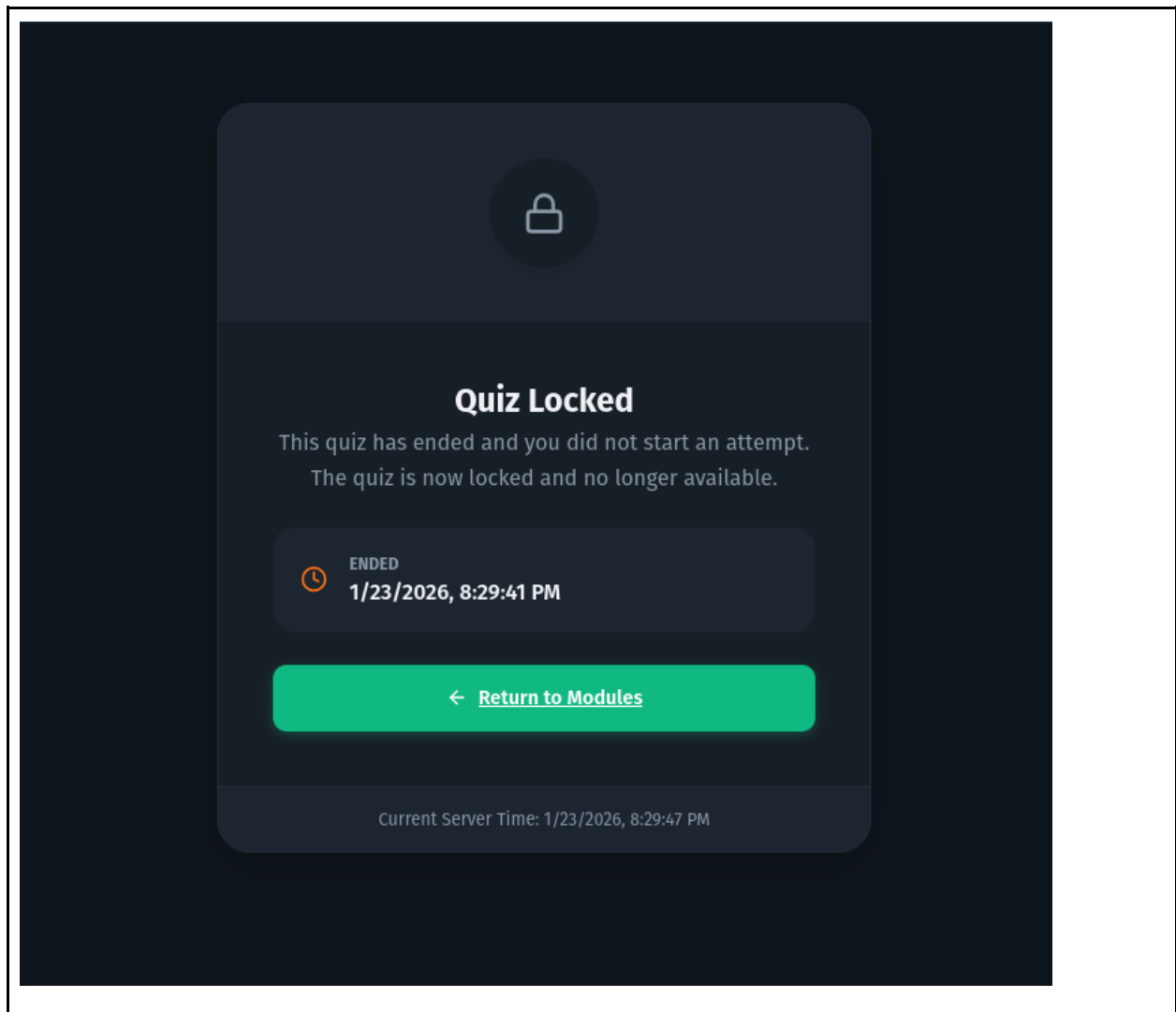
1/23/2026, 8:27:29 PM



CLOSES

1/23/2026, 9:26:07 PM

Current Server Time: 1/23/2026, 8:27:11 PM

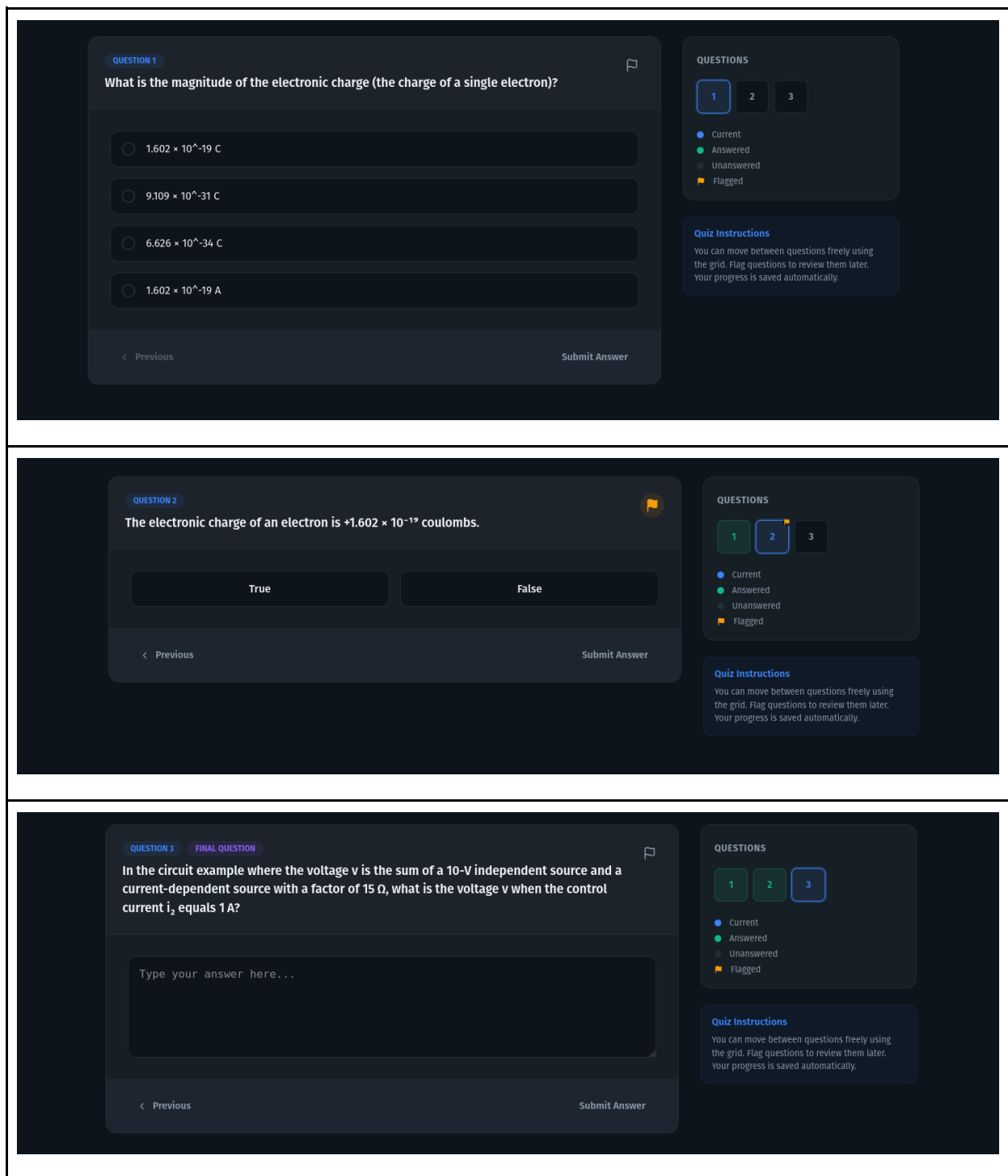


(Figure. 20) Student Quiz Taking (Entry Page)

Student Quiz Question (Per Question) (Fig. 21)

- **What it does:** The core quiz-taking interface: renders question content with type-specific inputs, allows free navigation, saves answers immediately via upsert, and keeps a sidebar showing answered/unanswered state across all questions.
- **Inputs:** Student navigates to `/student/courses/[courseId]/modules/[quizId]/quiz/question/[index]`; selects/enters answer; clicks Submit Answer; navigates via sidebar or next/prev; clicks Finish Quiz.
- **Outputs:**
 - Question card: text + type label
 - Inputs:

- **MC:** radio options (A–D)
- **T/F:** two-option toggle
- **Short answer:** text input
- Sidebar: numbered question buttons with state (unanswered/answered/current)
- Controls: next/prev, submit answer, finish quiz (+ confirmation)
- Progress: answered count vs total
- Optional question flagging stored locally (localStorage)
- **Typical workflow:**
 - Student opens a question page (or lands there after Start/Continue).
 - System ensures attempt exists (created on first question access if needed).
 - System fetches current question + existing answer + sidebar statuses.
 - Student selects/types an answer.
 - Student submits; system upserts quiz_answers (unique on attempt_id, question_id).
 - Sidebar updates to answered state.
 - Student navigates freely until done.
 - Student clicks Finish; sees confirmation with answered count.
 - On confirm, system sets completed_at, runs auto-grading for objective questions, and returns to finished state.
- **Validation & constraints:**
 - Question access requires a valid attempt (created on first access).
 - Answer submission requires non-empty answer.
 - Upsert overwrites previous answers (no version history).
 - Finish requires confirmation.
 - Submission blocked after ends_at (finish action cannot complete after window).



(Figure. 21) Student Quiz Question (Per Question)

Student Quiz Results Review (Fig. 22)

- **What it does:** Shows the completed attempt breakdown with score, correctness indicators, earned points, and explanations—**only if** results are released via `show_results`. Otherwise displays a release-pending message.

- **Inputs:** Student navigates to `/student/courses/[courseId]/modules/[quizId]/quiz/review`; system loads completed attempt + answers + grading fields.
- **Outputs:**
 - Header: total score (X/Y), percentage, completion timestamp
 - Per-question cards: question text/type, student answer, correctness indicator, points earned/possible
 - Correct answer shown for objective questions (MC/T-F)
 - Explanations shown when authored
 - If `show_results = false`: “Results not yet released” message
- **Typical workflow:**
 - Student submits quiz.
 - System auto-grades objective questions.
 - Instructor grades short answers (if present).
 - Instructor toggles `show_results` per attempt.
 - Student opens review page.
 - System gates output by `show_results`.
- **Validation & constraints:**
 - Requires `completed_at` attempt.
 - Visibility controlled by `show_results` per attempt.
 - Short answers show earned points without a correct-answer reveal.



Quiz Completed

Your attempt has been successfully submitted and recorded.

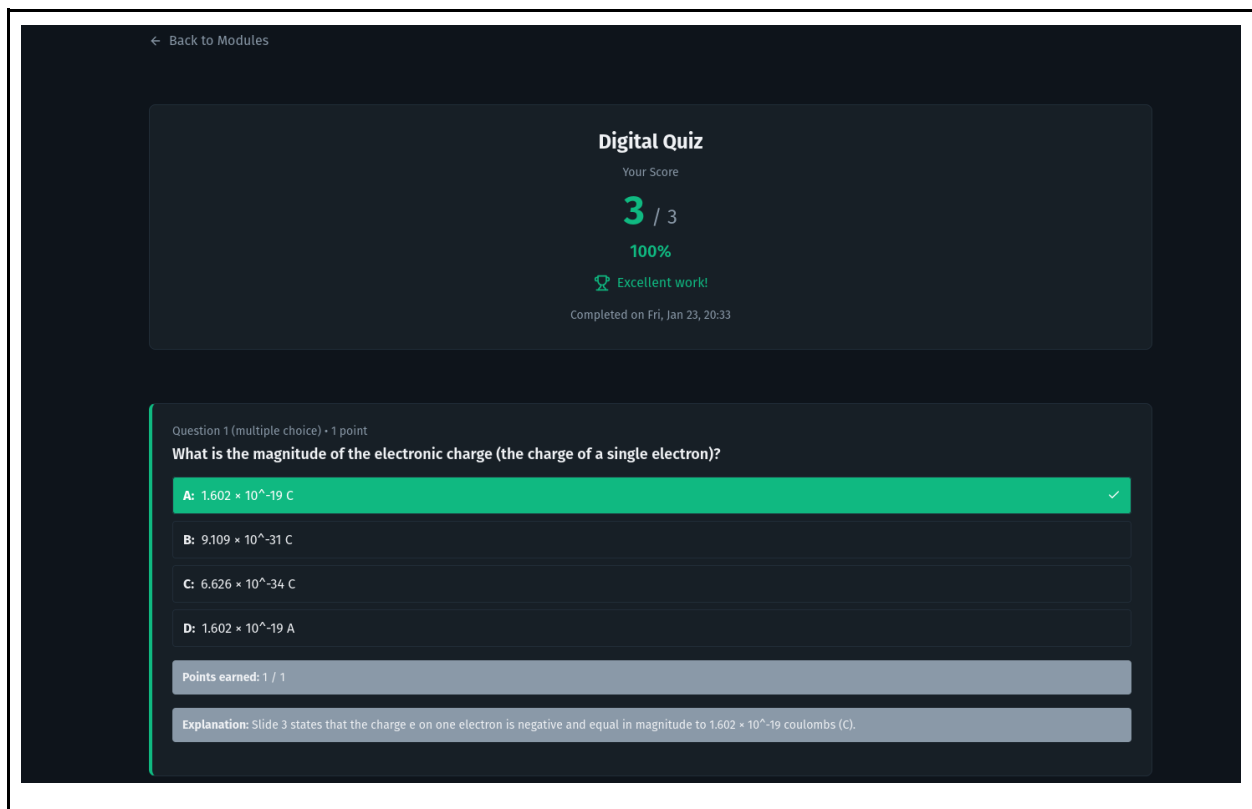
YOUR SCORE

2 / 3

 Submitted on 1/23/2026, 8:33:32 PM

[View Review](#) →

[Return to Modules](#) →



(Figure. 22) Student Quiz Results Review

Instructor Quiz Creation (Fig. 23)

- **What it does:** Full quiz authoring: metadata (title/description/windows/status), manual question creation (all types), AI generation from slide decks, review/edit of generated content, reorder, points/explanations, and save to DB as a quiz module + questions.
- **Inputs:** Instructor navigates to /instructor/courses/[courseId]/quiz/create; enters metadata; creates questions manually or via AI; selects decks + counts per type; saves.
- **Outputs:** Metadata form, question editor, AI generation modal (deck select + counts + optional focus), editable generated question list, reorder controls, Create Quiz action that persists module + questions.
- **Typical workflow (Manual):**
 1. Instructor enters metadata.
 2. Adds questions by type with required fields.
 3. Reorders as needed.
 4. Creates quiz (module + questions persisted).

- **Typical workflow (AI):**

1. Instructor enters metadata and opens AI modal.
2. Selects one/more slide decks.
3. Sets question counts per type; optional quiz focus text.
4. System calls Python backend /rag/generate-quiz via an API wrapper.
5. Backend retrieves slide chunks (top_k=30, slide chunks only), generates structured JSON, returns questions.
6. UI populates questions; instructor edits/reorders; saves quiz.

- **Validation & constraints:**

1. Requires instructor ownership of course.
2. starts_at < ends_at.
3. At least one question required.
4. Per-type requirements:
 - MC: ≥ 2 options (typically 4); correct answer in A–D format
 - T/F: correct answer “true/false”
 - SA: text answer field (used as reference)
5. Points must be positive integer.
6. AI generation requires ≥ 1 deck and ≥ 1 requested question count > 0.
7. Status: Draft (hidden) / Published (visible).

Create Quiz

Build questions manually, or auto-fill empty slots from your course slide decks.

Back

Create Quiz

Quiz Details

Edit

These values will be used when saving the quiz module.

TITLE

Digital Quiz

STATUS

Published

START DATE

2026-01-23T22:00

END DATE

2026-01-23T23:45

Question Counts

Edit

Counts are read from the URL and control how many question forms render.

MULTIPLE CHOICE

2

TRUE/FALSE

2

SHORT ANSWER

2

Auto-Fill With AI

Auto-fill All Empty

Select deck IDs tied to this course, then auto-fill any empty question slots.

Add deck_id manually (e.g., pptx_deck_abc123)

+ Add

DECK ID

FILE

TYPE

SLIDES

Auto-Fill With AI

 Auto-fill All Empty

Select deck IDs tied to this course, then auto-fill any empty question slots.

Add deck_id manually (e.g., pptx_deck_abc123)

+ Add

	DECK ID	FILE	TYPE	SLIDES
☐	pdf_deck_4cb6b4c9	Ch02-circuit.pdf	pdf	10
☐	pdf_deck_890f077e	Ch03-circuit.pdf	pdf	15
☐	pdf_deck_9abff69a	Ch08-circuit.pdf	pdf	18
☐	pdf_deck_ab55a907	Ch04-circuit.pdf	pdf	24
☐	pdf_deck_857804b3	Ch06-circuit.pdf	pdf	21

AI INSTRUCTIONS (OPTIONAL)

e.g. Focus on slides 1-10, emphasize practical applications...

Multiple Choice 1

 Auto-fill Empty

All questions filled

Multiple Choice #1 ✓ FILLED

QUESTION

What is the magnitude of the electronic charge (the charge of a single electron)?

OPTIONS

A

$1.602 \times 10^{-19} \text{ C}$

B

$9.109 \times 10^{-31} \text{ C}$

C

$6.626 \times 10^{-34} \text{ C}$

D

$1.602 \times 10^{-19} \text{ A}$

CORRECT ANSWER

A

POINTS

1

EXPLANATION (OPTIONAL)

Slide 3 states that the charge e on one electron is negative and equal in magnitude to 1.602×10^{-19} coulombs (C).

The image shows a dark-themed user interface for creating quiz questions. It features two main sections, each with a title, a count of questions, and an 'Auto-fill Empty' button.

True / False 1 Auto-fill Empty
 1 empty question

True / False #1 EMPTY

QUESTION

Write the question text...

CORRECT ANSWER POINTS

True 1

EXPLANATION (OPTIONAL)

Shown to students after answering...

Short Answer 1 Auto-fill Empty
 1 empty question

Short Answer #1 EMPTY

QUESTION

Write the question text...

CORRECT ANSWER

Write the expected answer...

POINTS

1

EXPLANATION (OPTIONAL)

(Figure. 23) Instructor Quiz Creation

Instructor Quiz Editing (Fig. 24)

- **What it does:** Pre-start edit surface for quiz metadata and questions (CRUD, reorder, points/explanations). Enforces an edit lock after starts_at to maintain fairness.
- **Inputs:** Instructor navigates to /instructor/courses/[courseId]/quiz/[quizId]/edit; edits metadata/questions; saves.
- **Outputs:** Pre-populated edit form, question list with edit/delete, reorder controls, lock warning when started, save action.

- **Typical workflow:**

1. Instructor opens edit page.
2. System loads quiz + questions.
3. System checks whether starts_at has passed.
4. If not started: full editing allowed; save persists updates.
5. If started: show lock warning; disable question modifications/reorder/points/explanations.

- **Validation & constraints:**

1. Requires instructor ownership.
2. Edit lock: no question modifications after starts_at.
3. Quiz deletion only if no attempts exist.

Instructor Quiz Submissions List

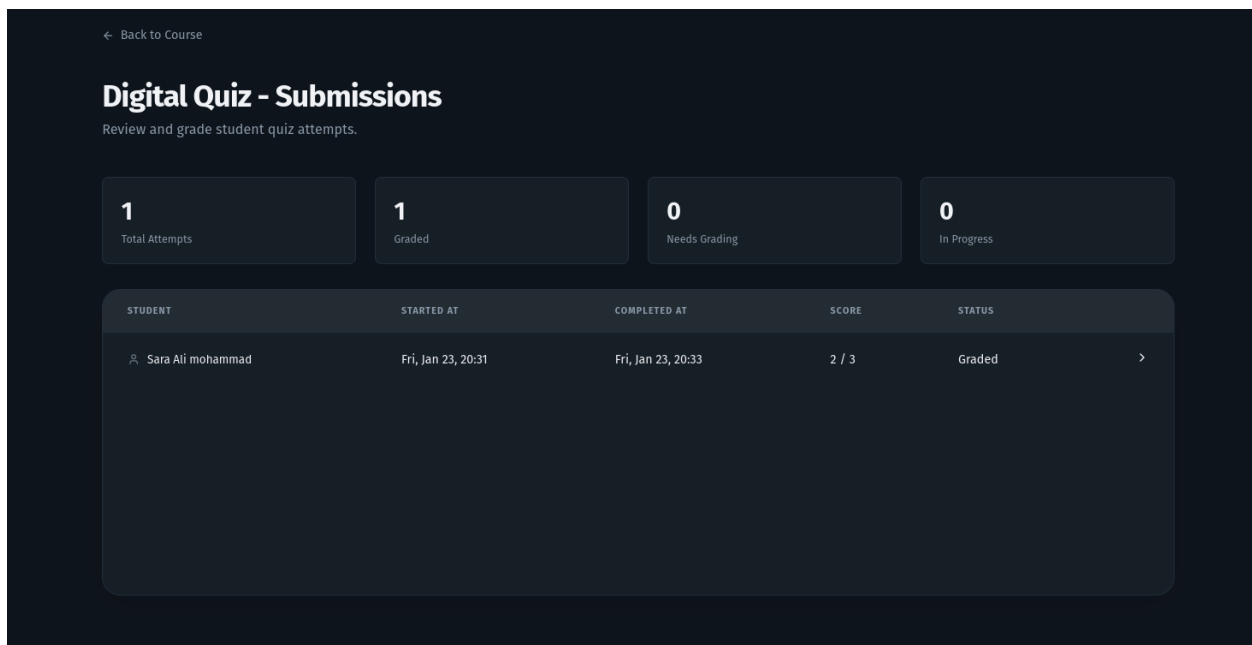
- **What it does:** Centralized attempt tracking: a table of attempts with student info, timestamps, score, status, filters, and per-attempt results release toggles. Designed to surface “needs grading” attempts quickly.
- **Inputs:** Instructor navigates to /instructor/courses/[courseId]/quiz/[quizId]/submissions; filters; opens attempt detail; toggles results release.
- **Outputs:** Header with quiz stats, status filters, attempt table (student, started/completed, score/%, status badge), View Details action, per-row show_results toggle, empty state.

- **Typical workflow:**

1. Instructor opens submissions page.
2. System fetches attempts with student names + computed statuses.
3. Instructor filters to “needs grading”.
4. Instructor opens an attempt for grading.
5. Instructor toggles results release as needed.

- **Validation & constraints:**

1. Requires instructor ownership.
2. Status definitions:
 - in progress: no completed_at
 - needs grading: completed with ungraded short answers
 - graded: all questions graded
3. Filtering performed client-side after fetch.
4. Toggle updates show_results on the attempt.



(Figure. 24) Instructor Quiz Editing

Instructor Attempt Detail & Manual / Automatic Grading

- **What it does:** Full attempt review with auto-graded objective results and manual grading inputs for short answers. Supports score recalculation and results release control.
- **Inputs:** Instructor navigates to `/instructor/courses/[courseId]/quiz/[quizId]/attempt/[attemptId]`; enters short answer points; saves; toggles show_results.

- **Outputs:** Attempt header (student + timestamps + total score), per-question breakdown, auto-graded correctness for objective questions, point inputs for short answers (0..max), save grades action, show_results checkbox.
- **Typical workflow:**
 1. Instructor opens attempt detail from submissions list.
 2. System loads attempt + questions + answers.
 3. Objective questions display locked auto-grade results.
 4. Instructor assigns points for short answers.
 5. Save updates points_earned for short answers and recalculates totals.
 6. Instructor toggles results visibility; updates show_results.
- **Validation & constraints:**
 1. Requires instructor ownership.
 2. Short answer points must be numeric and within [0, max_points].
 3. Auto-graded objective questions cannot be overridden.
 4. Grade updates overwrite previous values (no audit trail stated).

Discussion

Design goals

1. **Flexible assessment formats** via MC, T/F, and short answer.
2. **Low-anxiety student experience** through free navigation, immediate persistence, and recovery.
3. **Instructor efficiency** through AI generation + auto-grading for objective questions.
4. **Academic integrity** through time windows, one-attempt policy, and edit locks.

Architectural overview

Core data model

- course_modules: quiz module (module_type = 'Quiz'), window fields (starts_at, ends_at), and status (Draft/Published).

- **quiz_questions:** typed questions with options (JSONB for MC), correct answer, points, order index, explanation.
- **quiz_attempts:** unique (module_id, student_id), scoring fields (score, max_score), timestamps (started_at, completed_at), results release (show_results).
- **quiz_answers:** per-question answer text plus grading fields (is_correct, points_earned) and timestamps.

Attempt state model

- **not started:** no attempt record
- **in progress:** attempt exists, completed_at is null
- **completed:** completed_at set, grading applied (auto + manual)

Key system mechanisms

Time window enforcement

- Pre-start: visible but not startable; countdown messaging.
- Within window: start/continue enabled.
- Post-end: no new attempts; finish/submission blocked after ends_at; completed attempts can be reviewed only if released.

Answer persistence + recovery

- Each answer save performs an **upsert** into quiz_answers (unique on attempt_id, question_id).
- Saves happen immediately (not batched at final submit), enabling session recovery after refresh/interruption.
- Updates overwrite earlier responses (no answer history).

Auto-grading (objective only)

- Runs on quiz completion.
- MC + T/F: sets is_correct and points_earned (points or zero).
- Short answer: left ungraded until instructor assigns points.
- Attempt totals: sum of points_earned; max_score sum of question points.

Results release control

- show_results is per attempt.
- Students see either a full breakdown or a “not released” state.

AI quiz generation (slide-grounded)

- Instructor selects decks + counts → request forwarded to Python backend /rag/generate-quiz.
- Backend retrieves slide chunks (top_k=30, slide chunks only), generates structured questions JSON.
- Instructor reviews/edits/reorders before saving.

Implementation architecture

Database layer

- Foreign keys with CASCADE DELETE across quiz modules → questions → attempts → answers.
- Uniqueness:
 - quiz_attempts(module_id, student_id) one attempt per student per quiz
 - quiz_answers(attempt_id, question_id) one answer per question per attempt
- Indexing described for module/attempt/answer lookup.

API layer

- Student endpoints for window fetch, attempt create/get, combined question/status fetch, answer upsert, finish + auto-grade.
- Instructor endpoints for quiz CRUD, submissions list, attempt detail, grading, results toggle.
- AI endpoint wrapper that forwards generation requests to Python service.

Front-end behavior

- Hook-driven data fetch + derived UI state (sidebar statuses, navigation).
- Prepopulation of saved answers on load.
- Immediate save (MC/T-F on selection; SA on submit per description).

- Local-only question flags via localStorage (not critical state).

Design trade-offs

Costs	Benefits	Decision
Higher pressure; no practice/retake path	Fairness + prevents retake abuse	One attempt policy
Can encourage end-loaded answering	Reduces anxiety; flexible pacing	Free navigation
Higher DB write volume (upsert per answer)	Recovery + no data loss	Immediate answer persistence
No built-in accommodation/extension mechanism described	Controlled availability	Time windows
Requires review; quality depends on slide content + ingestion readiness	Faster authoring	AI quiz generation
No partial credit; limited matching sophistication	Reduces instructor workload	Auto-grading objective questions
Errors can't be corrected mid-window	Fairness	Edit lock after start

Constraints & limitations

- **Attempts:** one attempt per student; no retake mechanism described.
- **Windows:** fixed; no grace period; submission blocked at ends_at; no extension flow described; server-time reliance.
- **Editing:** questions/ordering/points/explanations locked after starts_at; quiz deletion only if no attempts exist.
- **AI generation:** limited to slide deck content; depends on ingestion availability; may produce weak questions requiring edits; counts are targets (may generate fewer if content insufficient).
- **Auto-grading:** MC + T/F only; no partial credit; exact/basic normalization (T/F trim/lowercase); no fuzzy matching for SA.

- **Results:** binary release per attempt (show_results); no partial reveal controls described; objective correct answers shown when results are released.
- **Question types:** fixed three types; MC limited to A–D; no described character limits/validation for short answers.
- **UI/navigation:** no randomization/shuffling; flagging stored only in localStorage; sidebar always shows all questions (large quizzes may impact UI performance).
- **Data integrity:** answer updates overwrite history; no grade audit trail/versions described; no soft-delete path described.
- **Performance:** synchronous auto-grading on finish may delay completion flow for large quizzes; AI backend is a dependency.

Feature 5: Circuit Simulation System

High-level summary

The Circuit Simulation System provides two simulators—a Digital Logic editor with client-side, always-on evaluation, and an Electrical/Analog editor backed by a Python SPICE pipeline for analysis-grade simulation. Both environments share a consistent canvas interaction model (drag/drop placement, wiring, editing, save/load), while diverging at the simulation layer and result depth (real-time propagation vs. analysis runs + plots). A shared Circuit Templates Library enables discovery, preview, and fork-to-editor flows that integrate directly with course labs/assignments. Primary outputs include the simulator landing page, digital/electrical editors (new + saved), electrical analysis tables/plots, and template browsing + fork.

Feature list

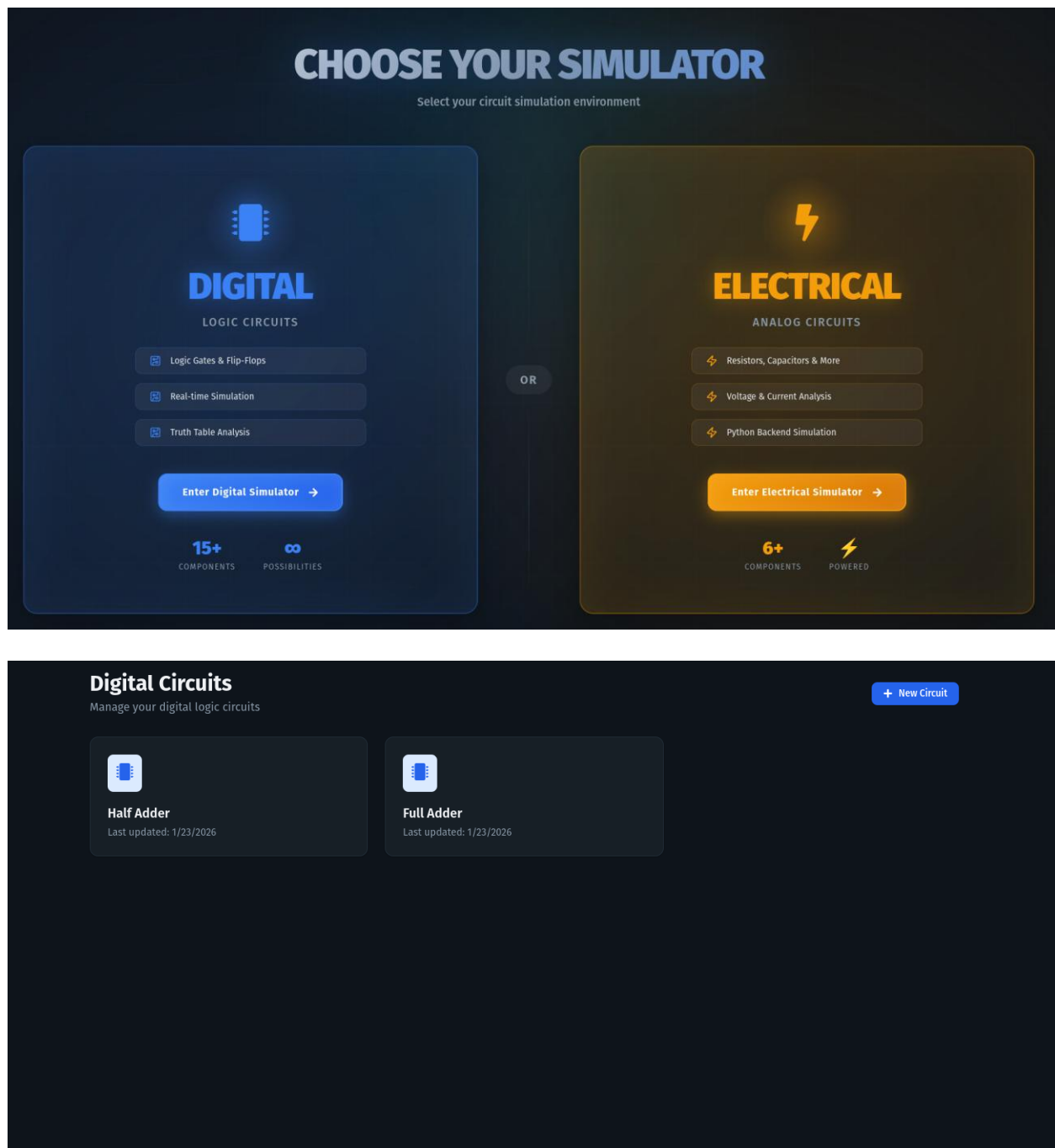
- Simulator landing page that routes users to **Digital** or **Electrical** environments.
- Digital logic circuit editor with always-on real-time evaluation and visual signal propagation.
- Electrical circuit editor with SPICE-grade analyses (DC op point, DC sweep, transient, AC sweep/Bode).
- Persist circuits (digital/electrical) and support saved-circuit editing routes.
- Templates library with search + filters (type, difficulty) and template detail pages.

- Fork templates into personal editable circuits while preserving template provenance (template_id link).
- Integrate templates into course workflows (templates can be associated with modules; students fork for lab work).

Sub-features

Simulator Landing Page (Fig. 25)

- **Route:** /simulator
- **What it does:** Provides a two-option entry point (Digital vs Electrical) and routes the user into the selected simulator.
- **Inputs:** User navigates to /simulator; selects a simulator type.
- **Outputs:** Split-screen selection UI, capability summaries, navigation to /simulator/digital or /simulator/electrical.
- **Typical workflow:**
 1. User opens /simulator.
 2. System renders Digital + Electrical options with summaries.
 3. User selects one.
 4. System routes to the chosen simulator.



(Figure. 25) Simulator Landing Page

Digital Circuit Simulator (Create & Load)

- **What it does:** Provides an interactive digital logic circuit building environment that combines a palette-based component editor with a canvas-driven wiring system and an always-on simulation engine. Users can create new circuits or load existing ones, with the system reconstructing component layout, wiring topology, properties, and internal state

from persisted circuit data. The simulation engine continuously re-evaluates circuit behavior on topology or input changes, propagates signal states across wires and components, visualizes logic levels in real time, supports sequential logic with persistent flip-flop states, and enables circuit persistence with save and save-as functionality.

- **Inputs:** User navigates to `/simulator/digital` or `/simulator/digital/[id]`; user drags components from the palette to the canvas; user connects components using wires; user toggles input switches or clock signals; user edits component properties; user deletes components or wires; user clicks Save or Save As actions.
- **Outputs:** Canvas-based digital circuit editor with grid-aligned layout, visual wire connections, and component symbols; real-time signal visualization on wires and output indicators (logic HIGH/LOW); reconstructed circuit state when loading existing circuits; persisted circuit records containing layout, topology, properties, and simulation state.
- **Key elements:**
 - **Component palette**
 1. Logic gates: AND, OR, NOT, NAND, NOR, XOR, XNOR.
 2. Sequential components: D, JK, SR, T flip-flops.
 3. Inputs: switches, push buttons, clock generators.
 4. Outputs: LEDs, seven-segment displays, probes.
 - **Editor features**
 1. Grid-aligned placement with snap-to-grid behavior.
 2. Click-to-connect wire routing with visual feedback.
 3. Component property editing and configuration dialogs.
 4. Circuit persistence with save, load, and save-as operations.
 - **Simulation engine**
 1. Always-on simulation without an explicit Run action.
 2. Signal visualization with logic HIGH rendered in red and logic LOW rendered in black.
 3. Sequential state persistence across simulation cycles.
 4. Clock-driven periodic re-evaluation of dependent components.
 5. Iterative signal propagation with convergence control.
- **Typical workflow:**
 - User opens the digital simulator via `/simulator/digital` or `/simulator/digital/[id]`.
 - System initializes a new canvas or fetches and deserializes existing circuit data.

- If loading an existing circuit, the system reconstructs components, wiring, properties, and internal states.
- User places logic gates, sequential components, inputs, and outputs from the palette.
- User connects components using wires to form logic paths.
- User toggles inputs or clocks and observes real-time signal propagation.
- Simulation engine re-evaluates circuit behavior on every topology or input change.
- User edits components, wiring, or properties as needed.
- User saves the circuit or creates a save-as copy.
- **Validation & constraints:**
 - Circuit re-evaluation is triggered on topology edits and input state changes.
 - Signal propagation uses an iterative convergence loop capped at a maximum of 8 iterations to prevent infinite feedback loops.
 - Flip-flop states persist across simulation cycles and are not reset on each evaluation.
 - Clock generators trigger periodic re-evaluation of dependent components.
 - Loaded circuits must reference valid component and wire definitions in persisted circuit data.
 - Save operations serialize layout, topology, properties, and simulation state into a unified circuit representation.

Electrical Circuit Simulator (Create & Load)

What it does: Provides an analog/electrical circuit building environment with backend-driven SPICE simulation. Users construct circuits on a canvas, configure component values, define node connections, select analysis types and parameters, and execute simulations that return numerical tables and plots. The simulator supports both creating new circuits and loading existing ones by reconstructing component layout, wiring topology, and parameter values from persisted circuit data. A backend pipeline converts JSON circuit topology into SPICE netlists, executes simulations, and returns structured JSON results and server-rendered plots, enabling iterative simulation, visualization, and persistence of electrical circuits.

- **Inputs:** User navigates to `/simulator/electrical` or `/simulator/electrical/[id]`; user places components on the canvas; user configures component values; user wires nodes and ensures reference ground; user selects analysis type and parameters; user runs simulations; user edits topology or values; user saves or clones circuits.
- **Outputs:** Electrical circuit editor with palette, grid-based canvas, and wiring; analysis result tables including node voltages and branch currents; simulation plots including DC

sweep curves, transient waveforms, and Bode magnitude/phase responses; reconstructed circuit state when loading saved circuits; persisted circuit records containing topology, values, and layout.

- **Key elements:**

- a. **Component palette**

- 1. Passive components: resistors (R), capacitors (C), inductors (L).
 - 2. Sources: DC voltage, AC voltage (sinusoidal), DC current.
 - 3. Reference component: ground (mandatory).
 - 4. Devices: diode (1N4148), BJT (2N2222 NPN).

- b. **Value configuration**

- 1. Parameter dialogs for R, C, L, voltage, current, frequency, and device parameters.

- c. **Analysis controls**

- 1. DC operating point.
 - 2. DC sweep.
 - 3. Transient analysis.
 - 4. AC sweep (Bode magnitude and phase).

- d. **Backend simulation pipeline**

- 1. JSON topology generation from canvas state.
 - 2. SPICE netlist construction and simulation execution.
 - 3. JSON-based result extraction and normalization.
 - 4. Server-side plot rendering with Base64-encoded PNG output.

- **Typical workflow:**

- a. User opens the electrical simulator via `/simulator/electrical` or `/simulator/electrical/[id]`.
 - b. System initializes a new canvas or fetches and deserializes existing circuit data.
 - c. If loading an existing circuit, the system reconstructs components, values, wiring, and layout.
 - d. User places components and configures their values.
 - e. User wires the circuit and ensures a valid ground reference.
 - f. User selects an analysis type and configures its parameters.
 - g. User runs the simulation and receives tables and plots in the UI.
 - h. User iterates by modifying topology or parameters and re-running analyses.
 - i. User saves the circuit or creates a save-as copy.

- **Validation & constraints:**

- a. Simulation requires backend availability configured as localhost:5000.
- b. Each analysis validates required parameters including ranges, step sizes, time windows, and frequency points.
- c. Circuits must include a ground reference node.
- d. Branch current extraction follows the PySpice naming convention `vr{name}_plus`.
- e. AC sweep node tracking is limited to a single node per request as defined by the analysis specification.
- f. Save operations serialize topology, values, layout, and metadata into a unified circuit representation.

Circuit Templates Library

- **Route:** /circuit-templates
- **What it does:** Searchable/filterable catalog of templates spanning both simulators, enabling discovery and navigation to template detail.
- **Inputs:** Navigate to library; optional filters (type, difficulty); optional search term.
- **Outputs:** Template card grid, filtered results, empty state, authentication redirect when required.
- **Key elements:**
 - Template cards: name, description preview, type badge (Digital/Electrical), difficulty badge, creator, date, optional thumbnail
 - Filters:
 - Type: All / Digital / Electrical
 - Difficulty: All / Beginner / Intermediate / Advanced
 - Search across name/description/creator

- AND-combined filtering (type + difficulty + search)
- **Typical workflow:**
 - Open /circuit-templates.
 - System fetches templates and renders cards.
 - User filters/searches.
 - User opens a template detail page.

Template Detail & Fork

- **Route:** /circuit-templates/[templateId]
- **What it does:** Displays template metadata + a read-only preview canvas, and supports “Fork to Editor” to create a personal editable copy in the correct simulator.
- **Inputs:** Template ID; fork action; access rules (public vs private).
- **Outputs:** Preview + metadata; new circuit record created on fork; redirect to /simulator/{type}/{circuitId}.
- **Typical workflow:**
 1. User opens /circuit-templates/[templateId].
 2. System renders read-only preview + metadata.
 3. User clicks “Fork to Editor”.
 4. System creates a new circuit record:
 - copies circuit_data
 - sets template_id reference
 - sets name to {template_name} (Copy)
 - sets created_by to current user
 - preserves type (digital/electrical)

5. System redirects to the saved-circuit editor route.

- **Validation & constraints:**

1. Public templates: visible/forkable by authenticated users.
2. Private templates: visible/forkable only by creator.
3. Templates may be associated with course modules via `module_id`.

Discussion

Design goals

1. **Hands-on learning through interactive simulation**

Immediate feedback loops (real-time logic; fast iteration on analysis runs) to support experimentation.

2. **Professional-grade analysis without local SPICE installation**

Backend SPICE execution provides analysis depth while keeping the client lightweight.

3. **Share and reuse circuit designs**

Templates enable instructor-provided starting points and reusable patterns via forking.

4. **Seamless integration with assignments/labs**

Templates can anchor lab modules; students fork and iterate as part of coursework.

Architectural overview

Dual simulation architecture

- **Digital (client-side)**

- Browser-based boolean evaluation
- Always-on propagation/visualization (no network latency)
- Iterative convergence for loops/sequential logic
- Flip-flop state persistence; clocks trigger periodic updates

- **Electrical (server-side)**

- Canvas topology serialized to JSON
- Python backend runs PySpice/ngspice analyses
- Returns structured result JSON + plot images (Base64 PNG)

- Supports: DC op point, DC sweep, transient, AC sweep/Bode

Shared infrastructure

- **Canvas interaction model:** Fabric.js-based placement/wiring/editing shared across both simulators.
- **Persistence:** Circuits stored as serialized JSON (circuit_data JSONB) with type differentiation (digital vs electrical).
- **Templates:** Template records store circuit JSON in a format aligned with saved circuits, enabling faithful fork behavior.

Key system mechanisms

Digital: always-on evaluation loop

- Evaluates components + propagates signals automatically on edits and input changes.
- Convergence is bounded (max iterations) to prevent infinite evaluation in feedback loops.
- Visual state is first-class (wire colors, LEDs, display outputs).

Electrical: SPICE analysis pipeline

- Frontend sends JSON topology; backend generates netlist and executes analysis.
- Results returned as tables + server-rendered plots.
- Resistor branch current mapping depends on the discovered naming convention (vr{name}_plus).
- AC sweep tracking limited to one node per request (as specified).

Templates: fork mechanism

- Fork produces a fully independent circuit copy linked back by template_id.
- Fork redirects directly into the correct simulator editor for immediate iteration.

Design trade-offs

- **Client-side digital vs server-side digital:** chosen to preserve immediacy for boolean logic and avoid latency.
- **Always-on simulation vs manual run:** chosen to reduce friction and emphasize causality; costs continuous (lightweight) computation.

- **Server-rendered plots vs client-rendered:** chosen for plot quality + Python tooling; costs larger payloads and static images.
- **JSON blob storage vs normalized schema:** chosen for flexibility and simple serialization; limits query-ability and DB-level validation.

Constraints & limitations

Digital simulator

- Fixed component library; no custom components without code changes.
- No timing-level modeling (prop delays, setup/hold), no analog behavior, no buses/tri-state.
- Grid-aligned layout; limited routing control; no hierarchical subcircuits.
- Convergence cap may fail for complex feedback loops.

Electrical simulator

- Fixed device set (notably 1N4148 diode, 2N2222 BJT as specified).
- Analysis constraints: current probing emphasized for resistors; AC tracking limited to one node per request.
- Backend dependency and latency; no offline electrical simulation.

Templates

- No version history; updates overwrite.
- Forks do not inherit future template updates.
- No collaboration primitives (comments, reviews, co-authoring).

General

- Limited import/export to standard EDA formats.
- Performance risks for large circuits.
- Assignment/lab linking is template-driven and requires the described manual fork flow.

Feature 6: AI-Powered Educational Assistant

High-level summary

The AI-Powered Educational Assistant delivers a course-aware chat experience built on **Retrieval-Augmented Generation (RAG)** so users can ask questions and receive **document-grounded answers with verifiable citations**. The system supports three modes: (1) **general RAG chat** across all ingested content, (2) **deck-grounded chat** scoped to selected slide decks, and (3) a **fallback general LLM** mode when grounding is unavailable or fails. Responses stream in real time via **SSE**, chats persist with multi-turn continuity, and admins manage ingestion + monitoring through a dedicated AI dashboard. Primary outputs are streaming answers with numbered citations, persistent chat history + sidebar navigation, deck selection/attachment flows, and admin ingestion + metrics surfaces.

Feature list

- Ask questions over ingested course materials using RAG-powered semantic retrieval.
- Return answers grounded in sources with **numbered citations** (pages/slides).
- Support **deck-scoped** Q&A by selecting slide decks from enrolled courses.
- Maintain multi-turn conversations with context continuity (history + rollup memory).
- Stream responses via **Server-Sent Events (SSE)**.
- Fall back to general LLM chat when grounding fails/unavailable (no citations).
- Admin ingestion for PDFs and slide decks (PDF/PPTX), with OCR support when required.
- Admin monitoring for usage + knowledge base health (docs/chunks/decks/slides).

Sub-features

Chat Interface (Student/Instructor)

- **Route:** /chatbot
- **What it does:** Provides a conversational UI for querying course materials. Supports three modes (RAG, deck-grounded, fallback LLM), streams responses live, persists multi-turn

history, and displays citations with document/deck name + page/slide numbers. Includes an attachment flow to select decks for focused sessions.

- **Inputs:**

- User navigates to /chatbot
- User enters a question
- Optional: user selects one or more decks via attachment modal
- Optional: user resumes an existing chat from sidebar

- **Outputs:**

- Streaming assistant response (token-by-token)
- Citations block with numbered references + source metadata
- Persistent chat + messages saved to DB (when authenticated)
- Sidebar chat list (all conversations)
- Attachment badge showing selected deck count
- Metadata events (sources + used queries) and error states

- **Typical workflow:**

- User opens /chatbot; system loads chat list (if authenticated).
- User types a question; optionally attaches decks.
- System selects mode:
 - Decks attached → deck-grounded mode
 - No decks → general RAG mode
- User sends; UI inserts the message optimistically.
- UI opens an SSE stream to backend; shows retrieval/thinking indicator.
- Backend emits metadata (sources + used queries).
- Backend streams token data events until completion.
- UI renders final answer with citations (collapsible section).
- Messages persist; chat title auto-generated from first user message.

- User continues with follow-ups using conversation context.
- **Chat modes:**
 - **RAG chat:** /rag/chat over full knowledge base; citations reference document + page numbers.
 - **Deck-grounded chat:** /rag/deck-chat filtered by deck_ids; citations reference deck name + slide numbers.
 - **General LLM fallback:** /groq/general-llm; ungrounded responses with no citations.
- **Deck selection workflow:**
 - User clicks attachment icon.
 - Modal lists slide decks grouped by enrolled courses.
 - User filters by course and/or searches by deck name.
 - User selects one or more decks; attachment count updates.
 - Closing modal keeps session in deck-grounded mode until deselected.
- **Streaming (SSE) mechanics:**
 - Event ordering: metadata → token data → done (or error)
 - Unicode handling: \u2003 converted to newlines to preserve markdown formatting.
- **Validation & constraints:**
 - Message must be non-empty.
 - Deck IDs must be valid and visible within the user's enrollment scope.
 - Auth required for persistence; anonymous chat allowed but not persisted.
 - SSE timeout: 60 seconds.
 - Backend enforces max message length (noted: ~2000 chars as specified).

AI System Management (Admin)

- **Route:** /admin/ai

- **What it does:** Admin control center for ingestion and monitoring. Supports PDF ingestion into global KB and slide deck ingestion (PDF/PPTX) for course-scoped retrieval, with OCR stats + chunking details. Exposes knowledge base health and usage metrics.
- **Inputs:**
 - Admin navigates to /admin/ai
 - Upload document (PDF) via drag/drop or file select
 - Upload slide deck (PDF/PPTX) via drag/drop or file select
 - Refresh stats; browse paginated document list
- **Outputs:**
 - Upload status indicators + ingestion success/error messages (with per-file stats)
 - Knowledge base metrics (docs/pages/chunks; decks/slides; latest timestamps; storage size)
 - Usage metrics (chats/messages/users with time period breakdowns)
 - Paginated document list + metadata
- **Typical workflow (PDF ingestion):**
 - Admin opens /admin/ai; dashboard loads stats + document list.
 - Admin uploads a PDF.
 - System validates file type and forwards to /api/admin/ai/ingest/documents.
 - API forwards to backend /rag/ingest.
 - Backend extracts text, chunks, embeds, and stores in ChromaDB.
 - Backend returns page/chunk summary.
 - API stores metadata in rag_documents.
 - UI shows success and refreshes totals.
- **Typical workflow (slide deck ingestion):**
 - Admin uploads PDF/PPTX via slides ingestion section.
 - Upload forwarded to /api/admin/ai/ingest/slides then backend /rag/ingest-slides.

- Backend extracts slide structure and triggers OCR under defined conditions (low native text / image-heavy slides).
- Backend creates slide chunks + window chunks and returns deck stats (deck_id, slide count, OCR stats).
- API stores metadata in slide_decks.
- UI shows success and refreshes deck totals.
- **Validation & constraints:**
 - Slides: PDF/PPTX only; documents: PDF only.
 - Size limits enforced (noted: ~40MB for slides; documents vary).
 - OCR requires Tesseract (optional; graceful degradation).
 - Ingestion runs synchronously until completion (large files may take minutes).
 - Duplicate filenames allowed (separate records).

Discussion

Design goals

1. Document-grounded accuracy

Answers are driven by retrieved context from ingested materials (not generic recall).

2. Verifiable attribution

Numbered citations point to specific pages/slides for auditability.

3. Flexible interaction modes

General RAG for broad coverage, deck-grounded for focused study, fallback LLM for reliability.

4. Context-aware conversations

Multi-turn continuity via recent-history window + rollup memory for long threads.

5. Administrative control

Central ingestion and monitoring to maintain coverage, quality, and operational visibility.

Architectural overview

Core data model

- chats: conversation sessions (ownership, name/title, timestamps)
- chat_messages: message storage (role, content, timestamp, optional metadata)
- rag_documents: ingested PDFs (page/chunk counts + metadata)
- slide_decks: ingested decks (type, slide/OCR stats, course association)
- **ChromaDB collection:** embeddings + chunk metadata
(source/page/deck_id/slide_number/etc.)

RAG pipeline

- **Ingestion**
 - PDFs: extract → two-stage chunking → embeddings → ChromaDB
 - Decks: structure extraction → conditional OCR → slide + window chunking → embeddings → ChromaDB
- **Query processing**
 - Query enhancement (expansion/reformulation/decomposition)
 - Semantic retrieval with optional deck metadata filters
 - Neighbor expansion for slide continuity
 - Deduplication
- **Answer generation**
 - Context formatting with numbered citations
 - Prompt construction (system + retrieved context + conversation history + rollup)
 - Streaming generation via Groq with SSE
 - Persist response + sources metadata

Conversation memory model

- Fixed window of recent messages (last 30)
- Older messages compressed into rollup memory
- Rollup injected back into context to preserve constraints/open loops

Key system mechanisms

Retrieval & grounding

- Embeddings: sentence-transformers/all-MiniLM-L6-v2 (384-dim)
- Multi-query retrieval: original + 4 expanded queries → merge + dedupe
- Routing by question type (direct, slide-specific, coding, comparative, follow-up)
- Deck chat uses metadata filtering (deck_ids) to constrain retrieval surface area

Slide ingestion strategy

- Dual chunking:
 - Per-slide chunks for precision
 - Window chunks (10 slides, stride 7) for continuity
- OCR triggered when native text is sparse / slides are image-heavy
- Normalization steps (bullets, header/footer stripping, OCR cleanup)

Streaming via SSE

- Events: metadata first (sources + used queries) → token data → done/error
- Frontend accumulates tokens and updates rendering progressively

Design trade-offs

- **RAG vs pure LLM:** grounded citations vs dependency on ingestion coverage + retrieval quality.
- **Dual chunking:** precision + continuity vs more storage + more dedupe complexity.
- **Query expansion:** higher recall vs increased ChromaDB load and latency.
- **Rollup memory:** long-thread continuity vs compression risk (nuance loss).
- **Server-side OCR:** better coverage vs longer ingestion time + optional dependency.
- **Single LLM provider (Groq):** simpler ops vs outage/rate-limit risk.

Constraints & limitations

RAG quality & coverage

- Answer quality is bounded by ingested content + retrieval relevance.

- Ambiguous questions can retrieve irrelevant chunks; exact-text dedupe may miss near-duplicates.
- Embedding model is fixed; changing it requires re-ingestion.

Persistence vs backend state

- Backend session/rollup cache is in-memory (lost on restart); DB persistence exists, but backend cache is not durable.
- Rollup threshold is fixed (30 messages).

Ingestion & operations

- Scanned PDFs may require OCR pre-processing (documents pipeline assumes extractable text).
- Ingestion is synchronous and blocks until completion (no progress UI).
- No deletion flow noted; duplicates can accumulate.

LLM failure modes

- Fallback LLM is ungrounded and has no citations.
- Prompting reduces hallucination risk but cannot guarantee strict compliance.
- Streaming complicates retries; client must handle partial responses.

Deck scoping limitations

- Students only see decks tied to enrolled courses.
- Deck selection is session-scoped (not persisted across new chats).
- Multi-deck selection can introduce conflicting context without explicit conflict resolution.

Feature 7: Authentication & User Management

High-level summary

Authentication & User Management provides secure platform access and centralized user administration. It supports two sign-in methods—email/password and Google OAuth 2.0—while enforcing **institutional access** via domain restriction (@najah.edu, @najah.edu.jo). Access is gated through role-based middleware that protects route groups (/student/*, /instructor/*, /admin/*). Admins can manage user accounts end-to-end (CRUD) with search/filter tooling and integrity-preserving cascade deletion.

Feature list

- Email/password authentication with bcrypt password hashing.
- Google OAuth 2.0 sign-in restricted to institutional domains.
- Automatic Google account linking to existing email accounts.
- Password reset via email verification + expiring reset token links.
- Role-based route protection via Next.js middleware.
- Admin user management (create/read/update/delete) across roles.
- Search/filter users by name/email for admin workflows.

Sub-features

Authentication Page (Fig. 26)

- **What it does:** A unified entry point for authentication (Login / Sign Up) supporting both email/password and Google OAuth flows. Includes validation, password strength enforcement, and institutional account linking for Google users.
- **Inputs:** User navigates to /authentication; submits email/password credentials or initiates Google OAuth; sign-up includes full name.
- **Outputs:** Tabbed Login/Sign Up UI, form validation feedback, Google sign-in button, and a created session on success.
- **Typical workflow (Email sign-up):**
 1. User opens /authentication → selects **Sign Up**.
 2. User enters email, password, full name.
 3. System validates email format + password strength + required fields.
 4. System hashes password with bcrypt (10 salt rounds).
 5. System creates profile with default role student.
 6. System creates encrypted session cookie.
 7. User is redirected to the student dashboard.
- **Typical workflow (Email sign-in):**
 1. User opens /authentication → selects **Login**.

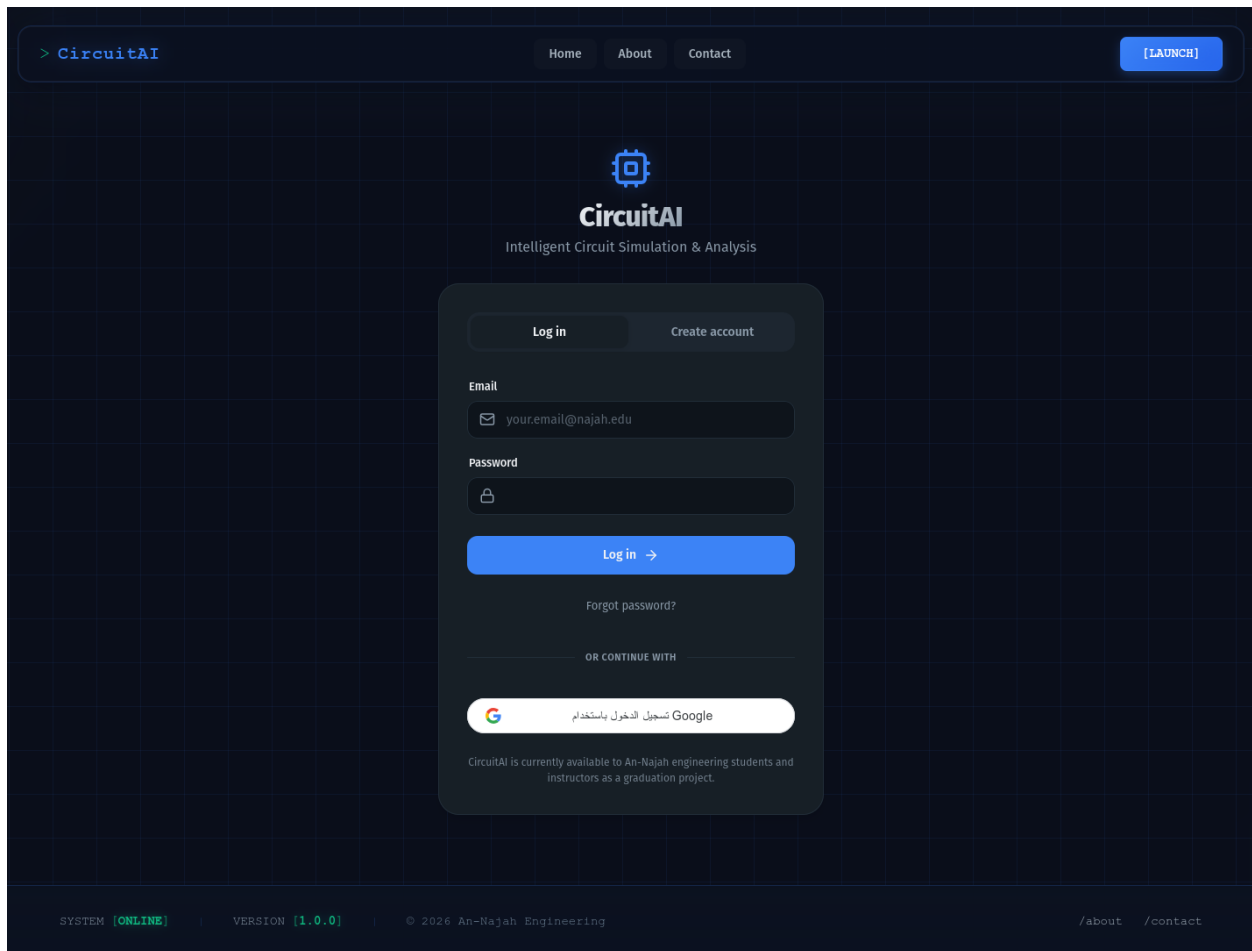
2. User enters email + password.
3. System fetches profile by email.
4. System verifies password via bcrypt.compare.
5. System creates encrypted session cookie.
6. User is redirected to role-appropriate dashboard.

- **Typical workflow (Google OAuth):**

1. User clicks **Sign in with Google**.
2. Google Identity Services prompts for authentication.
3. User authenticates with institutional Google account.
4. Google returns an ID token to the front-end.
5. Front-end posts token to /api/authentication/google.
6. Backend performs token verification + domain validation + lookup/link/create.
7. System creates session and redirects to role-appropriate dashboard.

- **Validation & constraints:**

1. Email must be valid format and globally unique.
2. Password must meet strength requirements.
3. **Both email sign-up and Google OAuth restricted to @najah.edu and @najah.edu.jo.**
4. Default role on sign-up is student.



(Figure. 26) Authentication Page

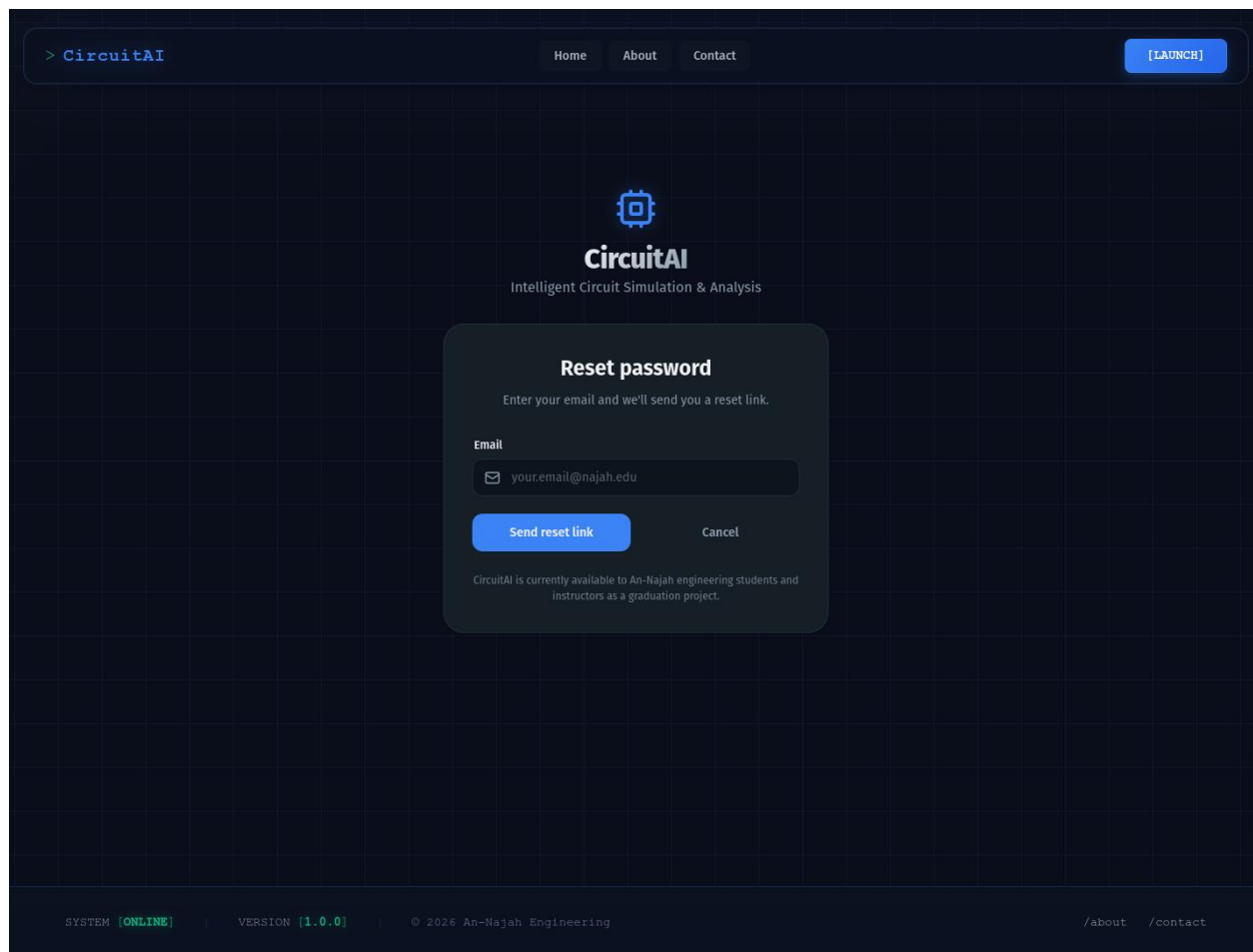
Password Reset (Fig. 27)

- **What it does:** Account recovery via email-driven reset flow. Generates an encrypted reset token with expiration, emails a reset link, validates the token, then updates the password using bcrypt hashing.
- **Inputs:** User navigates to /authentication/reset-password; submits email; uses emailed link (?token=...); submits new password + confirmation.
- **Outputs:** Email verification step, token validation, reset form (with prefilled email), success/failure messaging, redirect back to authentication on completion.
- **Typical workflow:**
 1. User clicks **Forgot Password**.
 2. User enters email.
 3. System validates email exists in profiles.

4. System generates encrypted reset token (expires in 1 hour).
5. System emails reset link with token.
6. User opens /authentication/reset-password?token=....
7. System decrypts token + checks expiration.
8. If valid, shows reset form with email pre-filled.
9. User enters new password + confirmation.
10. System validates strength + match.
11. System hashes new password with bcrypt.
12. System updates stored password hash.
13. Shows success state and redirects to authentication.

- **Validation & constraints:**

1. Reset token expiry: **1 hour**.
2. Token must be valid encrypted JWT.
3. Email must exist in profiles.
4. Password strength enforced; confirmation must match.



(Figure. 27) Password Reset

Admin Student Management

- **What it does:** Admin CRUD for student accounts, with search/filter and visibility into enrollment and activity signals. Deletion triggers cascade removal of related student-owned records.
- **Inputs:** Admin navigates to /admin/students; optional search; create/edit/delete actions via modals/confirmations.
- **Outputs:** Student table (name, email, enrolled course count, created date, last active status), create/edit modals, delete confirmation, filtered results.
- **Typical workflow (Create):**
 1. Admin opens /admin/students → clicks **Create Student**.
 2. Modal opens (name, email, password).

3. System validates + checks email uniqueness.
4. System hashes password with bcrypt.
5. System creates profile role = student.
6. List refreshes with new record.

- **Typical workflow (Edit):**

1. Admin clicks edit.
2. Modal opens prefilled.
3. Admin edits name/email and optionally password.
4. System validates; hashes password if provided.
5. List reflects updates.

- **Typical workflow (Delete):**

1. Admin clicks delete.
2. Confirmation warns about cascade deletion.
3. Admin confirms.
4. Profile deletion triggers cascade deletion (enrollments, submissions, quiz attempts, answers).
5. Student removed from list.

- **Validation & constraints:**

1. Email must be unique; password strength enforced if provided.
2. Admin role required.
3. Deletion is destructive (CASCADE DELETE).

Admin Instructor Management (Fig. 35)

- **What it does:** Admin CRUD for instructor accounts, with search/filter and statistics surfaces (courses taught, students taught). Instructor deletion cascades across course-owned resources.
- **Inputs:** Admin navigates to /admin/teachers; optional search; create/edit/delete flows.

- **Outputs:** Instructor table (name, email, courses taught, students taught, created date), create/edit modals, delete confirmation, filtered results.
 - **Typical workflow (Create/Edit/Delete):** Mirrors student management, with role instructor and instructor-specific cascade impacts.
 - **Validation & constraints:**
 - Email must be unique; password strength enforced if provided.
 - Admin role required.
 - Deletion triggers CASCADE DELETE for courses and dependent records.
-

Middleware & Role-Based Access Control (Fig. 36)

- **What it does:** Enforces role boundaries before rendering protected routes. Middleware extracts and validates the session (cookie or Bearer token), checks route prefix requirements, and redirects unauthorized requests. Also adds CORS headers for API routes (with endpoint-level auth still required).
- **Inputs:** Any protected navigation to /student/*, /instructor/*, /admin/*; middleware reads session cookie or Authorization: Bearer
- **Outputs:** Request proceeds (authorized) or redirects to / (unauthorized). API routes receive CORS headers.
- **Typical workflow:**
 1. User requests /student/courses.
 2. Middleware intercepts request.
 3. Middleware extracts session from cookie or bearer token.
 4. Middleware decrypts session to obtain {id, role, email, ...}.
 5. Middleware checks route prefix and required role.
 6. If role matches → allow; else → redirect to /.
- **Route protection rules:**

1. /student/* → role === student
2. /instructor/* → role === instructor
3. /admin/* → role === admin
4. Public routes: /, /about, /contact, /authentication → no middleware gating
5. API routes: CORS headers added; authentication enforced at endpoint level

- **Validation & constraints:**

1. Middleware runs pre-render.
2. Session must be valid encrypted JWT.
3. Prefix-based RBAC only; role mismatch redirects to /.

Discussion

Design goals

1. **Secure authentication with multiple options**

Email/password and Google OAuth support different user preferences without relaxing security requirements.

2. **Institutional alignment via domain restriction**

Domain gating ensures only Najah institutional users can access the platform.

3. **Seamless identity linking**

Google users map cleanly onto existing email identities without manual account merges.

4. **Hard role separation**

Middleware-level protection enforces clear student/instructor/admin boundaries.

5. **Centralized administration**

Admin tooling supports full lifecycle management with integrity enforced via validation + cascade deletion.

Architectural overview

Core session model

- Stateless, cookie-based session tokens stored in **HTTP-only cookies**.
- Token format: encrypted JWT-like payload (jose SignJWT/JWTVerify, HS256).
- Session expiry: **7 days**.
- Session payload includes core profile fields (id, role, email, name, etc.) + timestamps.

OAuth integration (verification + linking)

Google sign-in follows a staged flow:

- Verify Google ID token authenticity (JWKS-based verification).
- Validate allowed email domain (@najah.edu, @najah.edu.jo).
- Lookup by google_sub for returning OAuth users.
- Lookup by email for account linking if no google_sub match.
- Create new profile if no match exists.

User management model

- Admin CRUD operates against the profiles table.
- Email uniqueness enforced.
- CASCADE DELETE preserves referential integrity for dependent rows.
- Search/filter implemented client-side (loads then filters).

Key system mechanisms

Email/password authentication

- Sign-up: validate → bcrypt hash → create profile (role=student) → create session → redirect.
- Sign-in: lookup → bcrypt compare → create session → redirect.

Password reset

- Token: encrypted JWT carrying email + expiration (1 hour).

- Validate token on reset page load, then allow password update via bcrypt hashing.

Middleware route protection

- Prefix-based RBAC: /student, /instructor, /admin.
 - Session extraction supports cookie (web) and bearer (mobile).
 - Unauthorized requests redirect to /.
-

Implementation architecture

Database layer

- profiles table with role enum (student, instructor, admin).
- Email uniqueness constraint + index for lookup.
- Foreign keys with CASCADE DELETE for dependent records.

Service layer

- emailSignUp, emailSignIn
- googleSignIn (token verify + domain gate + link/create)
- resetPassword (token generate/verify + update)
- Admin services: fetch lists + statistics, create/update/delete users

Session utilities

- createSession, getCurrentUser, deleteSession
- generateResetPasswordToken, verifyResetPasswordToken
- Uses SESSION_SECRET env var

Middleware

- Matcher-based interception for protected routes.
- Cookie/Bearer extraction, decrypt/verify, role compare.
- Adds CORS headers for API routes and handles preflight.

Design trade-offs

Costs	Benefits	Decision
No server-side revocation; relies on cookie lifecycle	Simple, scalable (stateless), HTTP-only protection	Cookie-based encrypted sessions
Excludes external users; not configurable per deployment	Strong access boundary for university members	Institutional domain restriction
Assumes email ownership; linking risks if email compromised	Unified identity, preserves user data	Automatic account linking
No multi-role users (e.g., student + instructor)	Clear boundaries, simpler authz	Fixed single role per user
Prefix-only granularity; runs on every request	Strong default protection	Middleware RBAC enforcement
Not scalable for large userbases; requires loading all users	Fast UI, low server complexity	Client-side admin search
Permanent loss; no soft delete / archival path	Clean integrity, prevents orphaned records	Cascade deletion

Constraints & limitations

- **Auth scope:** Google only (no other OAuth providers); domain allowlist hardcoded; password reset depends on email service.
- **Sessions:** cookie-only, fixed 7-day expiration, no refresh tokens, no session limit across devices, no suspicious activity detection.
- **RBAC:** roles are mutually exclusive; no fine-grained permissions; prefix-based route protection only; unauthorized redirects to / (not /authentication).
- **Admin UX:** no bulk import/export; no deactivation (delete only); client-side search doesn't scale.

- **Password reset security:** no rate limiting (email enumeration risk); token reuse prevention is tied to successful update (no explicit invalidation store)