FEDERAL UNIVERSITY OF TECHNOLOGY, MINNA

SCHOOL OF ELECTRICAL ENGINEERING AND TECHNOLOGY

DEPARTMENT OF MECHATRONICS ENGINEERING

MCE 324 – SOFTWARE ENGINEERING

SmartLearn E-Learning Platform - Complete System Documentation

SUPERVISED BY

ENGR DR. JIBRIL ABDULLAHI BALA

SEPTEMBER 2025

Table of Contents

List of Figures

List of Tables

List of Abbreviations

- **LMS**: Learning Management System
- **API**: Application Programming Interface
- **ERD**: Entity Relationship Diagram

- **UI/UX**: User Interface / User Experience
- **JWT**: JSON Web Token
- **CSV**: Comma-Separated Values

Executive Summary

1.1 Project Overview

SmartLearn is a university e-learning and academic management platform that supports student enrollment, course registration, content delivery, assessments, results computation/approvals, feedback, analytics, and support. It provides role-based dashboards for students, lecturers, and administrators.

1.2 System Features Highlights

- Course registration and availability management
- Learning content upload/download with tracking
- Continuous assessment, grading, GPA/CGPA, transcripts
- Result approvals across department, school, and senate
- Live chat rooms and basic Q&A
- Admin dashboards and analytics
- Feedback, evaluations, FAQs, knowledge base
- Support tickets and internal notes

Target users: Students, Lecturers, Department/School/Senate Administrators.

1.3 Technology Stack Summary

- Frontend: Next.js (pages router), React, TypeScript, Tailwind CSS, shadcn/ui, SWR
- Backend: Next.js API routes, NextAuth.js
- Database: Prisma ORM with PostgreSQL
- Integrations: Cloudinary (documents), xlsx (exports)

Exact versions (from `package.json`):

- next: 15.4.5 · react: 19.1.0 · typescript: ^5
- prisma: ^6.16.2 · @prisma/client: ^6.16.2 · pg: ^8.16.3
- next-auth: ^4.24.11 · @auth/prisma-adapter: ^2.10.0

- tailwindcss: ^3.4.11 · lucide-react: ^0.462.0
- swr: ^2.3.6 · xlsx: ^0.18.5 · cloudinary: ^2.7.0

1.4 Project Team

10 groups (1–10), each owning a module such as authentication, course registration, LMS content, assessments, chat, design system, admin analytics, approvals, feedback/notifications, and support.

Introduction

2.1 Background

2.1.1 Problem Statement

Universities need a unified system for course management, learning content, assessments, approvals, and communication.

2.1.2 Motivation

Provide a robust, modular LMS aligned with academic workflows and Nigerian university contexts.

2.1.3 Significance

Streamlines academic processes, improves transparency, and enhances learning outcomes.

2.2 Project Objectives

2.2.1 Primary Objectives

- Deliver end-to-end course, content, assessment, and result workflows.

2.2.2 Secondary Objectives

- Provide analytics, feedback, and support tooling.

2.2.3 Success Criteria

- All key flows are functional, role-gated access, exports are available, and build passes without errors.

2.3 Project Scope

2.3.1 In Scope

Course registration, content management, grading, approvals, chat, support, analytics.

2.3.2 Out of Scope

Real-time video conferencing, payment gateway, production monitoring.

2.3.3 Assumptions and Constraints

Assumes PostgreSQL and environment configuration are available; constraints include academic calendar and team size.

2.4 Target Audience

Students, Lecturers, Administrators.

System Requirements

3.1 Functional Requirements

- User management and authentication
- Course listing, registration, approvals
- Content upload/download with access control
- Assessment entry, GPA/CGPA calculation, exports
- Communication (live chat, Q&A)
- Administrative dashboards and analytics

3.2 Non-Functional Requirements

3.2.1 Performance

Responsive UI, paginated lists, efficient queries.

3.2.2 Security

NextAuth sessions/JWT, role-based access, validated inputs.

3.2.3 Usability

Consistent design system, accessible components.

3.2.4 Reliability

Builds pass, error handling with toasts and safe defaults.

3.2.5 Scalability

Modular APIs, Prisma schema designed for growth.

3.2.6 Maintainability

TypeScript, ESLint rules, clear directory structure.

3.3 System Requirements

3.3.1 Hardware

Server: 2 vCPU, 4GB RAM+ for development/test; Client: modern browser on laptop/desktop.

3.3.2 Software

Node.js 18+, PostgreSQL, modern browsers, development via npm.

3.4 User Roles and Permissions

- Administrator (Department/School/Senate): manage users, courses, approvals, exports.
- Lecturer: manage course content, enter results, and view students.
- Student: register courses, access materials, view grades, chat.

System Architecture

4.1 Architecture Overview

- Next.js monorepo using pages router for UI and API routes.
- Three-tier: UI (React) → API (Next.js routes) → Data (Prisma/PostgreSQL).

- Modules interact via REST endpoints and shared Prisma models.

4.2 Technology Stack

4.2.1 Frontend

Next.js 15, React 19, TypeScript, Tailwind CSS, shadcn/ui, SWR.

4.2.2 Backend

Next.js API routes, NextAuth for auth, and Zod-for manual validation.

4.2.3 Database

PostgreSQL via Prisma; see prisma/schema.prisma.

4.2.4 Third-Party Services

Cloudinary (documents), Email notifications stubs, XLSX exports.

4.3 System Integration

Module interconnections through API calls; data flows from UI pages under pages/dashboard/* to pages/api/*, persisted via Prisma.
Key directories and files:

- UI Pages: pages/dashboard/*, pages/index.tsx, pages/_app.tsx, pages/_document.tsx
- API Routes: pages/api/* (see per-module tables below)
- Auth: pages/api/auth/[...nextauth].ts, middleware.ts, contexts/AuthContext.tsx, components/auth/LoginForm.tsx
- Database: prisma/schema.prisma
- Exports: pages/api/admin/export-students.ts, pages/api/admin/export-transcripts.ts, pages/api/lecturer/export-students.ts, pages/api/lecturer/export-grades.ts

4.4 Security Architecture

NextAuth credentials provider with JWT/session callbacks, middleware route protection, role-based checks in APIs, and environment secrets.

System Modules

This section summarizes modules (per Groups 1–10). For detailed developer-sized contributions, see Group contributions.md.
For each module, API endpoints are representative and derived from pages/api/*.

5.1 Student Information Management (Group 1)

- Overview: Authentication, profiles, role-based access.
- API Endpoints (examples):
  - POST /api/auth/[...nextauth] – NextAuth.js
  - GET/PUT /api/user/profile – profile
- UI: Login/Register forms, profile page.

Key Files:

- pages/api/auth/[...nextauth].ts
- pages/api/auth/password-reset.ts
- pages/api/user/profile.ts
- middleware.ts
- components/auth/LoginForm.tsx
- contexts/AuthContext.tsx

API Table:

Method	Endpoint	Description
POST	/api/auth/[...nextauth]	Authentication (NextAuth)
POST	/api/auth/password-reset	Request/reset password
GET	/api/user/profile	Get current user profile
PUT	/api/user/profile	Update current user profile

5.2 Course Registration System (Group 2)

- Overview: Course listing, selection, approvals.

- API Endpoints:
  - GET /api/course/available
  - POST /api/student/course-selection
  - POST /api/student/course-registration
  - POST /api/admin/course-registration-approval
- UI: pages/dashboard/courses.tsx (selection and progress).

Key Files:

- pages/api/course/available.ts
- pages/api/student/course-selection.ts
- pages/api/student/course-registration.ts
- pages/api/admin/course-registration-approval.ts
- pages/api/admin/department-course-selection.ts
- pages/api/student/enrolled-courses.ts

API Table:

Method	**Endpoint**	**Description**
GET	/api/course/available	List available courses
POST	/api/student/course-selection	Save student-selected courses
POST	/api/student/course-registration	Submit registration for approval
POST	/api/admin/course-registration-approval	Approve/decline registrations
GET	/api/student/enrolled-courses	Fetch enrolled courses
POST	/api/admin/department-course-selection	Admin sets department course availability

5.3 LMS Content Management (Group 3)

- Overview: Upload/download lecturer documents.
- API Endpoints:
  - POST/GET /api/lecturer/documents
  - GET /api/student/documents
- Integration: Cloudinary for storage.

Key Files:

- pages/api/lecturer/documents.ts
- pages/api/student/documents.ts

API Table:

Method	Endpoint	Description
POST	/api/lecturer/documents	Upload lecturer documents
GET	/api/lecturer/documents	List/delete lecturer documents
GET	/api/student/documents	Access documents for a student

5.4 Assessment and Result Computation (Group 4)

- Overview: Grades, GPA/CGPA, approvals.
- API Endpoints:
    - POST /api/lecturer/results
    - GET /api/student/grades
    - POST /api/admin/result-approval
    - GET /api/admin/export-students, /api/admin/export-transcripts
- Lib: lib/gpa-calculator.ts.

Key Files:

- pages/api/lecturer/results.ts
- pages/api/student/grades.ts
- pages/api/admin/result-approval.ts
- pages/api/admin/export-students.ts
- pages/api/admin/export-transcripts.ts
- lib/gpa-calculator.ts

API Table:

Method	Endpoint	Description
POST	/api/lecturer/results	Enter/submit student results

Method	Endpoint	Description
GET	/api/student/grades	Fetch student grades
POST	/api/admin/result-approval	Multi-stage approval actions
GET	/api/admin/export-students	Export student lists
GET	/api/admin/export-transcripts	Export transcripts

5.5 Virtual Meetings and Chat Rooms (Group 5)

- Overview: Live chat sessions, messages, Q&A stubs.
- API Endpoints:
  - POST/GET /api/live-chat/sessions
  - POST/GET /api/live-chat/messages
  - Q&A: /api/qa/questions, /api/qa/answers
- UI: pages/dashboard/chatrooms.tsx, student chat pages.

Key Files:

- pages/api/live-chat/sessions.ts
- pages/api/live-chat/messages.ts
- pages/api/qa/questions.ts, pages/api/qa/answers.ts

API Table:

Method	Endpoint	Description
GET	/api/live-chat/sessions	List/join chat sessions
POST	/api/live-chat/sessions	Create/close chat session
GET	/api/live-chat/messages	List messages in a session
POST	/api/live-chat/messages	Send a message

Method	Endpoint	Description
GET	/api/qa/questions	List questions
POST	/api/qa/questions	Ask question
POST	/api/qa/answers	Post answer

5.6 System Integration and UI/UX (Group 6)

- Overview: Design system, layouts, accessibility.
- Components: Tailwind, shadcn/ui, lucide icons, toasts.

Key Files:

- styles/globals.css, tailwind.config.ts
- components/layout/DashboardLayout.tsx

5.7 Admin Dashboard and Analytics (Group 7)

- Overview: Admin management and analytics.
- API Endpoints:
  - /api/dashboard/admin
  - /api/admin/users, /api/admin/courses, /api/admin/departments, /api/admin/schools
  - /api/analytics/overview, /api/analytics/performance, /api/analytics/logs

Key Files:

- pages/api/dashboard/admin.ts
- pages/api/admin/users.ts, pages/api/admin/courses.ts, pages/api/admin/departments.ts, pages/api/admin/schools.ts
- pages/api/analytics/overview.ts, pages/api/analytics/performance.ts, pages/api/analytics/logs.ts

API Table:

Method	Endpoint	Description
GET	/api/dashboard/admin	Admin overview metrics

Method	Endpoint	Description
GET	/api/admin/users	Manage users
POST	/api/admin/users	Create/update users
GET	/api/admin/courses	Manage courses
GET	/api/admin/departments	Manage departments
GET	/api/admin/schools	Manage schools
GET	/api/analytics/overview	Analytics overview
GET	/api/analytics/performance	Performance metrics
GET	/api/analytics/logs	Activity logs

5.8 Results Consideration and Approval (Group 8)

- Overview: Multi-stage approvals and visibility gating.
- API Endpoints: /api/admin/result-approval, exports, student visibility /api/student/grades.

Key Files:

- pages/api/admin/result-approval.ts
- pages/api/student/grades.ts

5.9 Feedback, Evaluation & Notification System (Group 9)

- Overview: Evaluations, knowledge base, FAQs, notifications.
- API Endpoints: /api/evaluations/*, /api/knowledge/*, /api/faqs, /api/notifications/*.

Key Files:

- pages/api/evaluations/course-evaluations.ts, pages/api/evaluations/feedback-forms.ts, pages/api/evaluations/feedback-responses.ts
- pages/api/knowledge/articles.ts, pages/api/knowledge/feedback.ts

- pages/api/faqs.ts and pages/api/admin/faqs.ts
- pages/api/notifications.ts, pages/api/notifications/email.ts

API Table:

Method	Endpoint	Description
GET	/api/evaluations/course-evaluations	List course evaluations
POST	/api/evaluations/feedback-forms	Create evaluation forms
POST	/api/evaluations/feedback-responses	Submit evaluation responses
GET	/api/knowledge/articles	Knowledge base articles
POST	/api/knowledge/feedback	KB feedback
GET	/api/faqs	Public FAQs
GET	/api/notifications	Notification fetch (stub)
POST	/api/notifications/email	Send email notifications (stub)

5.10 User Support and Help Center (Group 10)

- Overview: Support tickets, responses, and attachments placeholder.
- API Endpoints: /api/support/tickets, /api/support/responses.

Key Files:

- pages/api/support/tickets.ts, pages/api/support/responses.ts

API Table:

Method	Endpoint	Description
POST	/api/support/tickets	Create ticket

Method	Endpoint	Description
GET	/api/support/tickets	List tickets
POST	/api/support/responses	Add ticket response
GET	/api/support/responses	List responses (by ticket)

Database Design

6.1 Overview

Prisma ORM with PostgreSQL. Models cover users, roles, departments, courses, registrations, results, evaluations, content, tickets, and logs.

6.2 ERD

See prisma/schema.prisma. Generate Prisma ERD using community tools (placeholder).

6.3 Data Dictionary (format)

Table	Column	Type	Constraints	Description
users	id	UUID	PK	User identifier
users	email	TEXT	UNIQUE, NOT NULL	Login email
roles	name	ENUM	NOT NULL	User role

6.4 Relationships & Constraints

Foreign keys between users, courses, registrations, and results; cascade on delete where safe; indexes on user email and course code.

6.5 Normalization

3NF, where applicable, to reduce redundancy while supporting exports and reporting.

System Implementation

7.1 Development Methodology

Lightweight iterative sprints with module ownership by groups; feature branches merged via code review.

7.2 Development Environment

- Version Control: Git
- Repo Structure: Next.js pages, API routes under pages/api/*, Prisma under prisma/
- Tools: ESLint, TypeScript, Tailwind, Prisma

Scripts (from package.json):

- npm run dev → Next.js dev server
- npm run build → Production build
- npm start → Start production server
- npm run lint → Lint codebase

7.3 Coding Standards

Type-safe API handlers, consistent naming, error handling via toasts, and minimal coupling.

7.4 Testing Strategy

Manual flows per module; build checks (npm run build); export verification; endpoint smoke tests.

7.5 Quality Assurance

Linting (npm run lint), peer review, seed data for predictable testing.

Installation & Deployment

8.1 Prerequisites

- Node.js 18+
- PostgreSQL

8.2 Installation Guide

Step 1: Clone Repository

```
git clone <repository-url>
cd mce-324-smartlearn-project
```

Step 2: Install Dependencies

```
npm install
```

Step 3: Configure Environment Create .env.local with at least:

```
DATABASE_URL=postgres://user:pass@localhost:5432/smartlearn
NEXTAUTH_SECRET=your-secret
NEXTAUTH_URL=http://localhost:3000
CLOUDINARY_CLOUD_NAME=...
CLOUDINARY_API_KEY=...
CLOUDINARY_API_SECRET=...
```

Step 4: Database Setup

```
npx prisma generate
npx prisma db push
```

Optional: Seed Sample Data

```
# Ensure dev server is running (npm run dev)
curl -X POST http://localhost:3000/api/seed-organized
curl -X POST http://localhost:3000/api/seed-course
```

Step 5: Start Application

```
npm run dev
```

8.3 Configuration

- Database connection via DATABASE_URL
- NextAuth secrets and URL
- Cloudinary credentials for document uploads

8.4 Deployment

- Build: npm run build
- Start: npm start
- Ensure environment variables and database are configured in production; set up reverse proxy and SSL.

User Manual

9.1 Getting Started

- Access via browser at application URL.
- Login with role-specific credentials (after seeding or admin creation).

9.2 Student Guide

- Profile: Update via dashboard profile page.
- Course Registration: Select courses from available list and submit.
- Materials: Access course materials under each course.
- Assessments: View grades when released; transcripts via exports.
- Chat: Join course chat rooms and send messages.
- Feedback: Submit course evaluations and ratings.

9.3 Lecturer Guide

- Courses: Manage assigned courses and upload materials.
- Assessments: Enter results and submit for approvals.
- Students: View enrolled students and analytics.

9.4 Administrator Guide

- Users: Manage users and roles.
- Courses: Configure departments, schools, course availability.

- Approvals: Process result approvals; export students and transcripts.
- Analytics: View dashboards and activity logs.

Testing & Results

10.1 Methodology

Manual end-to-end checks per module; API smoke tests; export file validation.

10.2 Test Cases

Test ID	Module	Test Case	Expected	Actual	Status
TC001	Login	Valid credentials	Success	Success	Pass

10.3 Results Summary

All critical paths verified; issues tracked and addressed during sprints.

10.4 Performance Testing

Basic load checks on exports and list views.

10.5 User Acceptance Testing

Feedback collected from team; UI/UX refinements applied.

Challenges & Solutions

11.1 Technical Challenges

- Export file size handling → Streamed XLSX and CSV generation.
- Role-based access consistency → Centralized middleware and helpers.

11.2 Integration Challenges

- Module API alignment → Shared route contracts and SWR key conventions.

11.3 Resource Constraints

- Timeboxed sprints and prioritized feature set.

11.4 Lessons Learned

Modular ownership accelerates delivery; early schema alignment reduces rework.

Conclusion & Recommendations

12.1 Project Summary

Delivered core academic and LMS workflows with role-gated access and exports.

12.2 System Benefits

Improved efficiency, transparency, and user experience for all roles.

12.3 Future Enhancements

Mobile app, real-time video, AI-driven analytics, richer notifications.

12.4 Recommendations

Regular maintenance, upgrade path for dependencies, user training sessions.

12.5 Conclusion

SmartLearn meets MCE 324 objectives and provides a foundation for future growth.

Appendices

Appendix A: API Reference (by module)

- Auth: /api/auth/[...nextauth], /api/auth/password-reset
- Student: /api/student/*
- Lecturer: /api/lecturer/*
- Admin: /api/admin/*
- Evaluations: /api/evaluations/*
- Knowledge: /api/knowledge/*
- Live Chat: /api/live-chat/*
- Support: /api/support/*
- Analytics: /api/analytics/*

Appendix B: Database Schema Scripts

Manage schema with Prisma: see prisma/schema.prisma, use npx prisma db push for migrations in development.

Appendix C: Configuration Files

- .env.local example values documented in Section 8.3.

Appendix D: Error Codes Reference

Return JSON errors with message and code fields (module-specific; see API handlers).

Appendix E: Glossary

See List of Abbreviations.

Appendix F: Libraries and Tools

- Next.js
- React
- TypeScript
- Prisma
- PostgreSQL
- NextAuth.js
- Tailwind CSS
- shadcn/ui (Radix UI components)

- lucide-react (icons)
- SWR
- xlsx
- Cloudinary SDK