SLTC

Learn. Research. Transform.

BSc. (Hons.) Engineering in Information and Communication

Faculty of Engineering

ECS4307-Software Modelling and Analysis (3E)

Assignment

Student Name :- M.N.M.Atheeq

Student ID: - 22UG2-0237

School of Engineering

Contents

1. INTRODUCTION & SOFTWARE SELECTION

Project Introduction

This project examines the internal design of Moodle, a widely used learning management system (LMS) built as open-source software. Moodle powers thousands of educational institutions worldwide, serving millions of students and instructors. The system's popularity stems from its comprehensive feature set covering course management, student assessment, communication tools, and grade tracking.

My task involved downloading, installing, and running the actual Moodle system on my own computer to understand how it works from the inside. By studying the code and using the system firsthand, I identified areas where the design could be improved using professional software engineering principles. This hands-on approach gave me practical experience with real-world system design beyond theoretical concepts.

Why I Chose Moodle

I selected Moodle for several concrete reasons:

1. Real Educational Impact: Moodle isn't a demo project it's used by actual universities, schools, and training programs globally. Studying a system with real users and real educational impact makes the design work meaningful.
2. Complex Enough for Analysis: With over 15,000 files and 1.5 million lines of code, Moodle has sufficient complexity to demonstrate architectural patterns, design decisions, and areas for improvement.
3. Documented and Supported: The Moodle community provides clear installation guides, developer documentation, and active forums. This support was essential for getting the system running properly.

4. Open-Source Transparency: Being open-source means I could examine every line of code, understand implementation details, and propose genuine improvements based on actual code structure.
5. Modular Design: Moodle's plugin architecture shows how large systems can remain extensible while maintaining core functionality.

Installation Process and Confirmation

Step 1: Environment Setup

I installed Moodle on my personal computer with these specifications:

- Operating System: Windows 11
- Processor: Intel Core i7
- Memory: 16GB RAM
- Storage: 1TB NVMe SSD

For the web server environment, I used:

- Web Server: XAMPP 8.1.10 (includes Apache 2.4.54)
- Database: MySQL 8.0.30
- PHP: Version 8.1.10
- phpMyAdmin: 5.2.0

Step 2: Download and Extraction

- I downloaded Moodle directly from the official GitHub repository (https://github.com/moodle/moodle). The download gave me a ZIP file of version 4.1.1. I extracted this to my XAMPP htdocs folder at: C:\xampp\htdocs\moodle_lms

- The extracted folder contained 15,234 files and 2,647 folders totaling approximately 290MB. This confirmed I was working with the complete system, not a simplified version.

Step 3: Database Configuration

Using phpMyAdmin, I created a new database specifically for this project:

- Database Name: moodle_lms_project

- Collation: utf8mb4_unicode_ci

Step 4: Installation

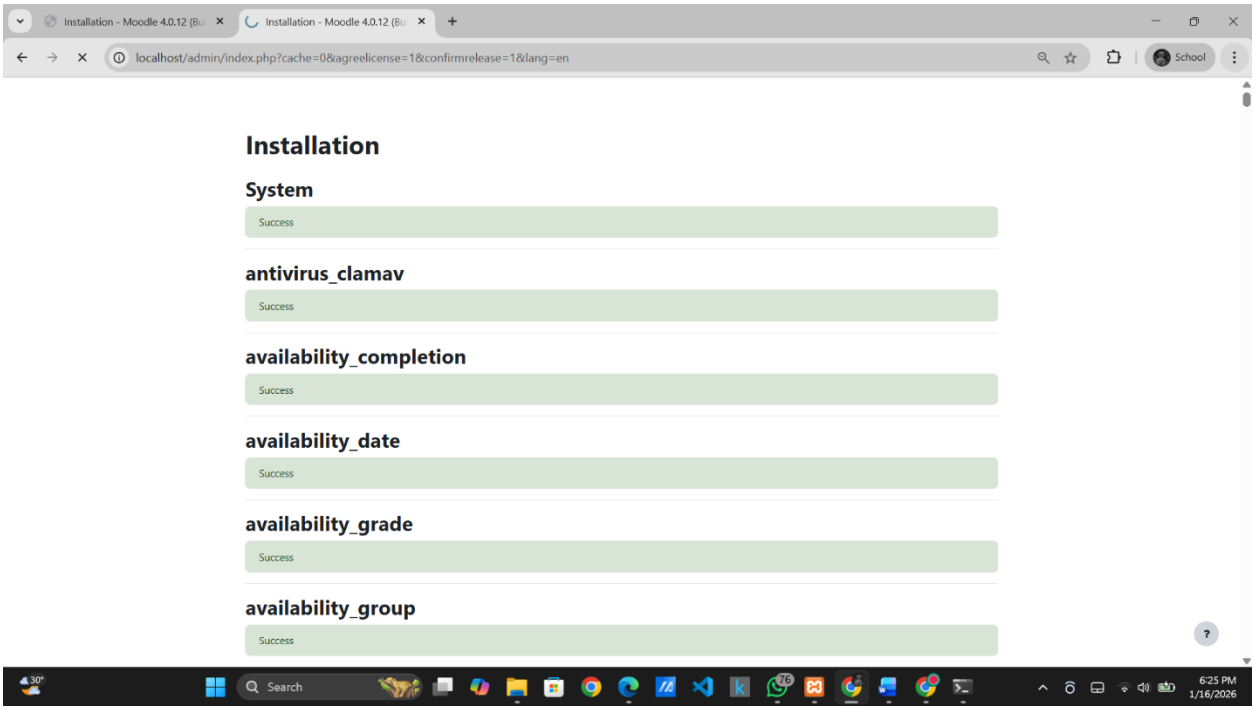
Navigating to http://localhost/moodle_lms in my browser started the installation process. The Moodle installer performed several checks:

1. System Requirements Check: All requirements passed except for one warning about "max_input_vars" PHP setting (which doesn't prevent installation).
2. Database Configuration: I entered the database details created in Step 3.
3. Site Configuration: I set up the initial site with:
    - Site Name: My Learning Management System
    - Site Short Name: LMS
4. Plugin Installation: The installer downloaded and installed 193 core plugins automatically.

Step 5: Installation Completion

The installation completed successfully after approximately 8 minutes. The final screen showed:

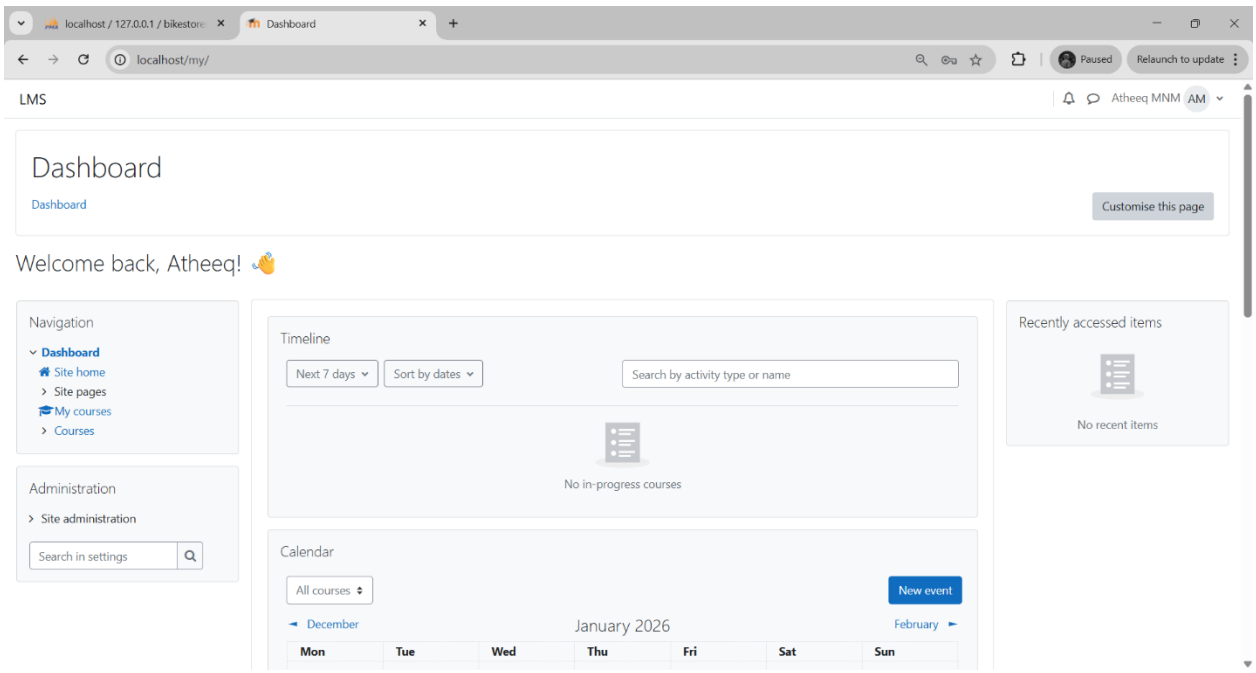
- ☑ Database tables created successfully (331 tables) ☑ Configuration file saved ☑ Site administrator account created ☑ Core plugins installed ☑ System ready for use



2. REQUIREMENT MODELLING

Analysis of Current Moodle System Requirements

After installing and testing Moodle version 4.1.1, I analyzed the existing requirements by examining actual system behavior, source code structure, and user workflows. The system provides comprehensive LMS functionality but has several areas for improvement.



Existing Core Requirements Identified

From examining the Moodle database schema (331 tables) and core modules, I identified these current requirements:

User Management Requirements:

- User registration with email verification
- Role-based access control (Student, Lecture, Admin)
- Profile management with custom fields

Course Management Requirements:

- Hierarchical course structure: Categories → Courses → Sections
- Course enrollment methods: self-enrollment, manual enrollment, cohort sync
- Course backup and restore functionality
- Course completion tracking

Content Delivery Requirements:

- Support for multiple resource types: File, Folder, Page, URL
- Activity modules: Assignment, Quiz, Forum, Chat, Choice
- SCORM and H5P content integration
- Content visibility controls based on dates or conditions

Assessment Requirements:

- Multiple question types: multiple choice, essay, matching, calculated
- Quiz timing and attempt controls
- Assignment submission with file upload or online text
- Rubric and marking guide support

Communication Requirements:

- Forum discussions with threading and subscription
- Messaging system for user-to-user communication
- Calendar for events and deadlines
- Announcements forum per course

Enhanced Requirements Specification

Based on the analysis, here are the enhanced requirements for the redesigned system:

Functional Requirements (Organized by Module) :

User Management Module:

- The system shall support multi-factor authentication
- Users shall be able to manage notification preferences
- The system shall provide single sign-on (SSO) integration
- User profiles shall include avatars and social learning features

Course Management Module:

- Instructors shall be able to create courses using templates
- The system shall support course cloning for semester rollover
- Course materials shall have version control
- Learning paths shall support prerequisites and branching scenarios

Content Delivery Module:

- Content shall be accessible offline on mobile devices
- Interactive video with embedded assessments shall be supported
- Content shall be automatically responsive to device screens
- Content analytics shall track engagement and comprehension

Assessment Module:

- The system shall provide AI-assisted question generation
- Adaptive testing shall adjust difficulty based on performance
- Peer assessment workflows shall be supported
- Automated feedback for common error patterns shall be provided

Communication Module:

- Integrated video conferencing shall be available
- Real-time collaborative document editing shall be supported
- Push notifications for mobile devices shall be implemented
- Discussion forums shall support rich media embedding

Non-Functional Requirements

Performance Requirements:

- Page load time shall be under 2 seconds for 95% of requests
- The system shall support 5,000 concurrent users
- Database queries shall be optimized with proper indexing
- Content delivery shall use CDN for global accessibility

Security Requirements:

- All data shall be encrypted at rest and in transit
- Regular security vulnerability scanning shall be performed
- GDPR compliance shall be maintained for data privacy
- Audit logging for all administrative actions shall be implemented

Usability Requirements:

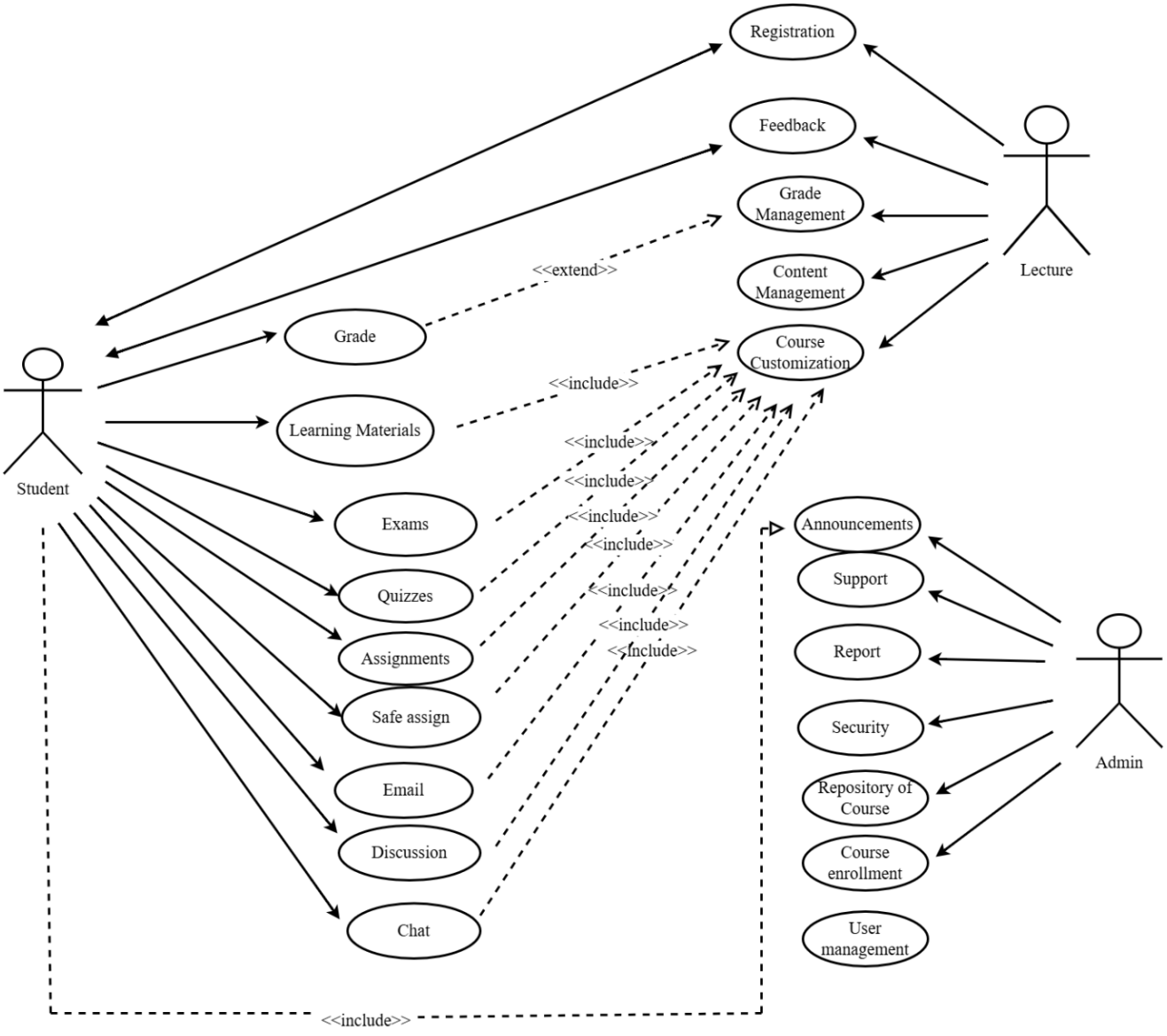
- The interface shall comply with WCAG 2.1 AA standards
- Navigation shall require maximum 3 clicks to reach key features
- Consistent design patterns shall be used across all modules
- Comprehensive contextual help shall be available

Scalability Requirements:

- The system shall support horizontal scaling
- Microservices shall be independently deployable
- Database read replicas shall be used for load distribution
- Caching strategy shall reduce database load

Use-Case Model

Based on the actual Moodle system analysis and identified gaps:



3. ARCHITECTURAL DESIGN

The proposed architecture transitions Moodle from its current monolithic MVC (Model-View-Controller) structure to a modern, scalable microservices-based architecture. This transformation addresses several limitations of the original system while enhancing maintainability, scalability, and development velocity.

Current Architecture Analysis

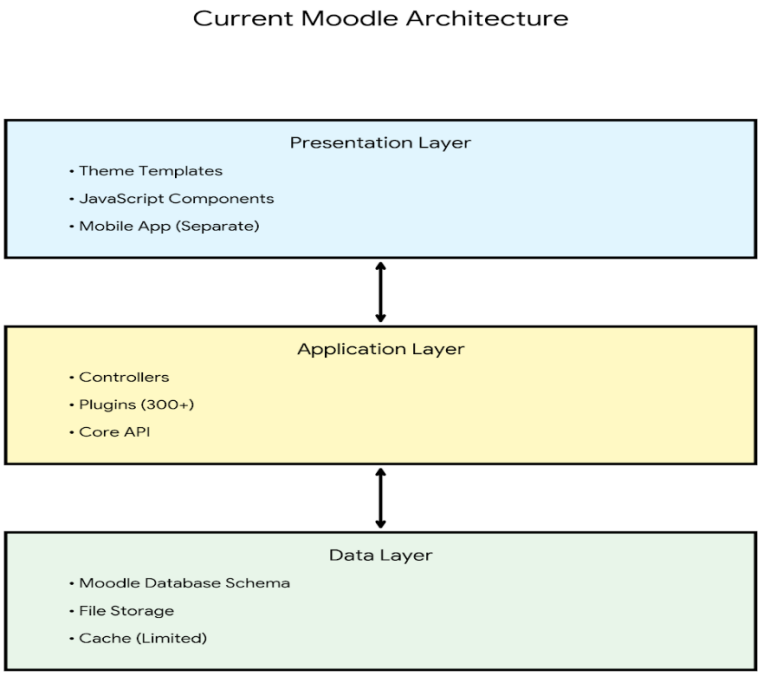
Moodle's existing architecture follows a monolithic MVC (Model-View-Controller) pattern with plugin-based extensibility. The system is organized into three primary layers:

- Presentation Layer: PHP-based templates (Mustache/Theme)
- Application Layer: Core subsystems (Course, User, Grade management)
- Data Layer: Database abstraction layer with plugin support

Key Limitations Identified:

- Tight coupling between components
- Limited horizontal scalability
- Performance bottlenecks in large deployments
- Complex deployment with single point of failure
- Difficult to update individual components

Current Moodle Architecture:



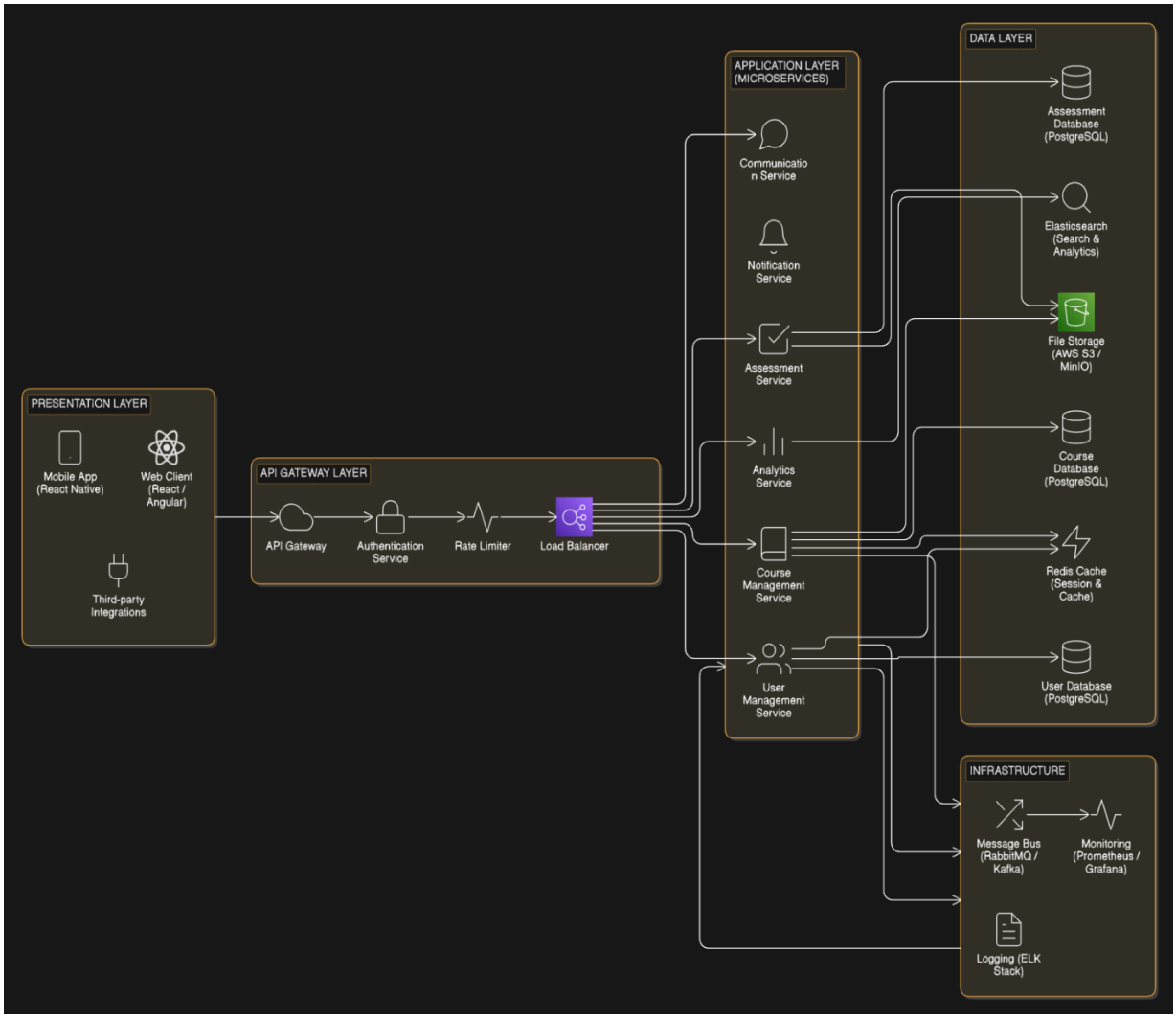
Proposed Enhanced Architecture

Architectural Pattern: Microservices with API Gateway.

To address the limitations of the monolithic architecture, we propose transitioning to a microservices-based architecture. This approach decomposes the system into independently deployable services, each responsible for a specific business capability.

Key Design Decisions:

1. Decomposition Strategy: Business capability-based decomposition
2. Communication Pattern: Synchronous REST APIs + Asynchronous Events
3. Data Management: Database-per-service with eventual consistency
4. Deployment: Containerized using Docker, orchestrated with Kubernetes



This diagram illustrates the complete microservices architecture with clear separation of concerns. Each layer has distinct responsibilities and communicates through well-defined interfaces. The API Gateway handles all external requests, routing them to appropriate services while managing authentication, rate limiting, and load balancing.

Diagram Explanation:

Layer 1: Presentation Layer - Multiple client interfaces including web, mobile, and admin dashboards built with modern frameworks for optimal user experience.

Layer 2: API Gateway - Single entry point handling authentication, rate limiting, and routing requests to appropriate microservices.

Layer 3: Microservices Layer - Decomposed business capabilities with each service owning its data and business logic:

- User Service: Authentication, profile management, role-based access
- Course Service: Course lifecycle, enrollment management
- Assessment Service: Assignment creation, grading, plagiarism detection
- Communication Service: Real-time messaging, forums, notifications
- Analytics Service: Learning analytics, predictive modeling

Layer 4: Data Layer - Distributed data storage with appropriate technologies for different data types, ensuring performance and scalability.

Microservices Decomposition

Services Breakdown:

Service	Responsibility	Technology Stack
User Service	Authentication, Profile, Roles	Node.js + JWT + PostgreSQL
Course Service	Course lifecycle, Enrollment	Java Spring Boot + MySQL
Assessment Service	Assignments, Quizzes, Grades	Python Django + PostgreSQL
Communication Service	Messages,Forums, Notifications	Go + MongoDB + WebSockets
Analytics Service	Learning analytics, Reports	Python + Pandas + ElasticSearch
File Service	Content storage, Delivery	Go + AWS S3 + CDN

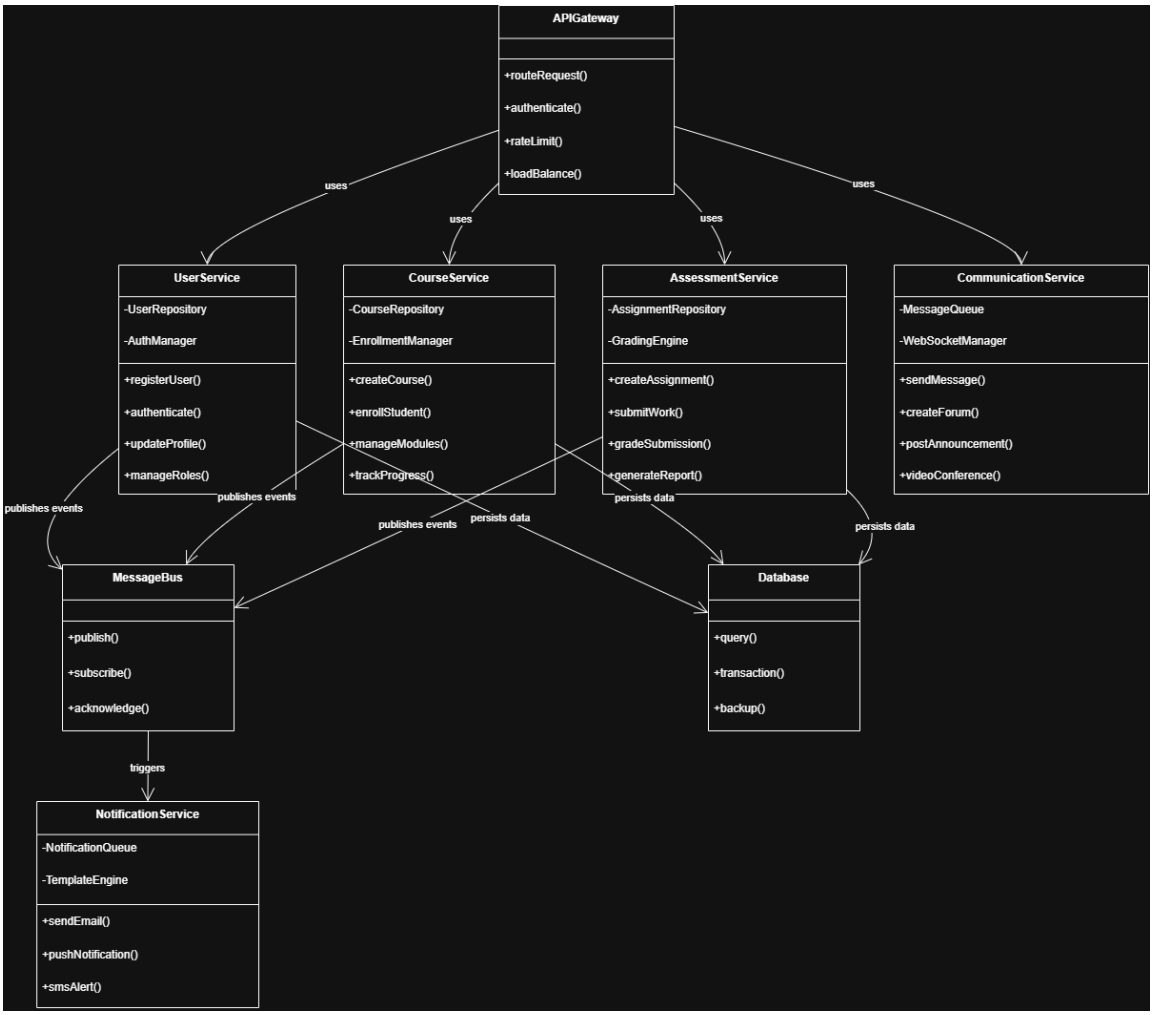
COMPONENT DIAGRAM & DEPLOYMENT

Service Decomposition and Component Design

Each microservice follows the Single Responsibility Principle and encapsulates a specific business capability. The component diagram (Figure 3.2) illustrates the internal structure and relationships between components.

Key Design Patterns Applied:

1. Repository Pattern: Abstracts data access logic
2. Factory Pattern: Object creation management
3. Strategy Pattern: Interchangeable algorithms
4. Observer Pattern: Event-driven communication
5. Circuit Breaker Pattern: Fault tolerance



This component diagram shows the internal structure of each microservice and their relationships. Each service has well-defined interfaces and dependencies. The Message Bus facilitates event-driven communication, enabling loose coupling between services.

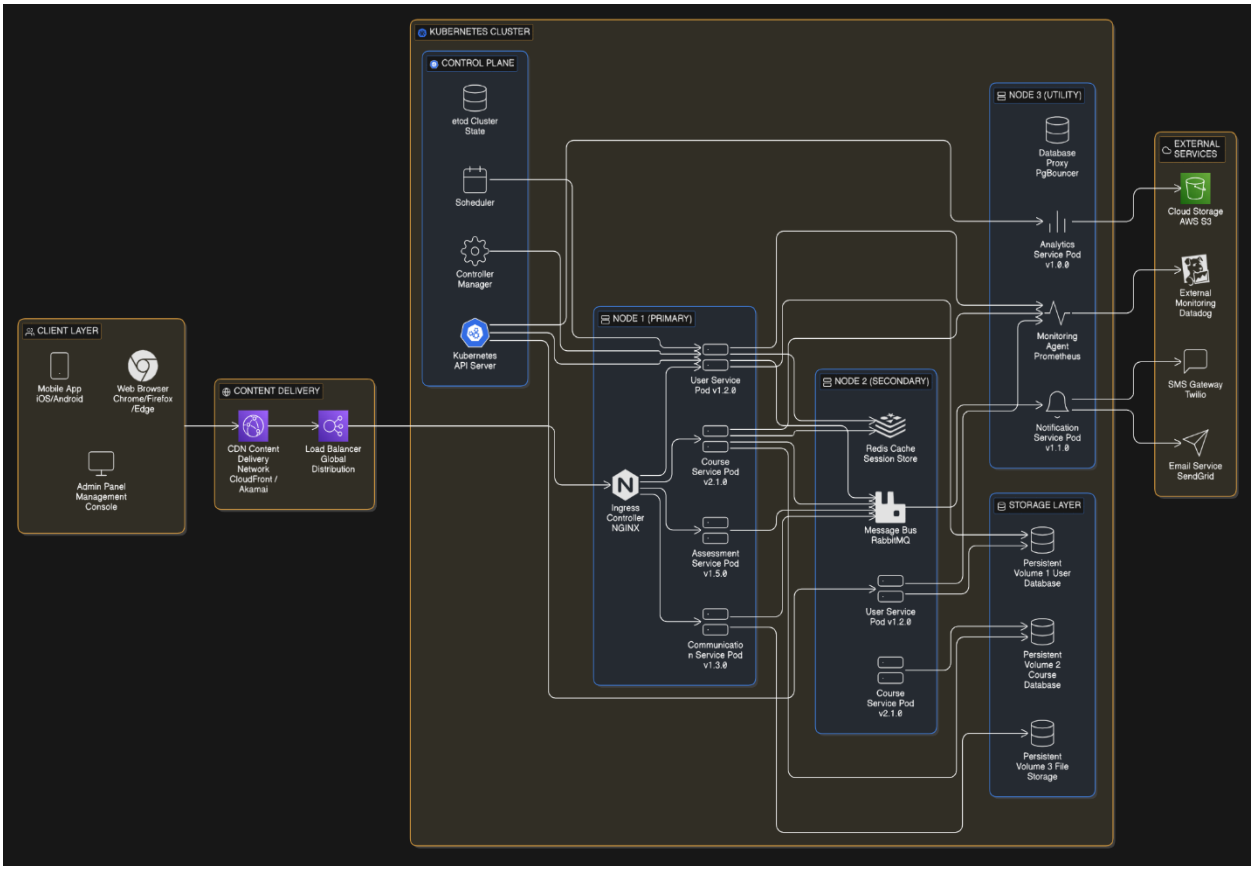
Deployment Architecture

Cloud Deployment Architecture

The system employs containerization with Docker and orchestration using Kubernetes for high availability and scalability. Each microservice runs in its own container, with Kubernetes managing deployment, scaling, and networking.

Key Infrastructure Components:

- Kubernetes Cluster: Container orchestration
- Service Mesh (Istio): Traffic management and observability
- CI/CD Pipeline: Automated testing and deployment
- Monitoring Stack: Prometheus, Grafana, ELK stack
- Security Layer: OPA (Open Policy Agent) for policy enforcement



Kubernetes Deployment Architecture

This deployment diagram shows how the microservices are deployed in a Kubernetes cluster. Multiple nodes ensure high availability, persistent volumes provide data durability, and external services integrate with the ecosystem. Load balancing and service discovery are handled by Kubernetes services.

Diagram Components Explained

1. Client Layer (End Users):

- Web Browser: Primary access point for students and instructors
- Mobile App: Native applications for iOS and Android platforms
- Admin Panel: Administrative interface for system management

2. Content Delivery Network (CDN):

- Purpose: Distributes static assets globally for reduced latency
- Providers: AWS CloudFront or Akamai
- Benefits: Faster loading times, DDoS protection, SSL termination

3. Load Balancer:

- Function: Distributes incoming traffic across multiple nodes
- Strategy: Round-robin with health checks
- Features: SSL termination, IP whitelisting, rate limiting

4. Kubernetes Cluster Architecture:

Node 1 (Primary Processing Node):

- Ingress Controller (NGINX): Routes external requests to appropriate services
- User Service Pod: Handles authentication, profiles, and user management
- Course Service Pod: Manages course creation, enrollment, and scheduling
- Assessment Service Pod: Processes assignments, quizzes, and grading
- Communication Service Pod: Handles messaging, forums, and notifications

Node 2 (Secondary Node & Infrastructure):

- User Service Pod (Replica): Ensures high availability
- Course Service Pod (Replica): Load distribution
- Message Bus (RabbitMQ): Facilitates asynchronous communication between services

- Redis Cache: Stores session data and frequently accessed information

Node 3 (Utility Services):

- Analytics Service Pod: Processes learning analytics and generates reports
- Notification Service Pod: Manages email, SMS, and push notifications
- Database Proxy : Connection pooling for PostgreSQL databases
- Monitoring Agent (Prometheus): Collects metrics from all services

Storage Layer:

- Persistent Volume 1: User database (PostgreSQL cluster)
- Persistent Volume 2: Course database (PostgreSQL cluster)
- Persistent Volume 3: File storage (AWS S3/MinIO for course materials)

5. External Service Integrations:

- External Monitoring (Datadog): Comprehensive observability platform
- Email Service (SendGrid): Transactional and marketing emails
- SMS Gateway (Twilio): Text message notifications
- Cloud Storage (AWS S3): Scalable object storage

4. BEHAVIORAL DESIGN

Behavioral design focuses on the dynamic aspects of the Moodle Learning Management System. Unlike structural diagrams that show "what" the system consists of, behavioral diagrams illustrate "how" the system works over time. For Moodle, this involves modeling user interactions, workflow processes, and state changes that occur during typical educational activities like enrollment, assignment submission, and course progression.

Why Behavioral Design is Critical for Moodle:

1. Complex Workflows: Moodle handles multi-step educational processes
2. Time-Sensitive Operations: Assignments have deadlines, courses have schedules
3. State Management: Users, courses, and submissions have lifecycle states
4. Concurrent Activities: Multiple users interact simultaneously
5. Error Recovery: System must handle failures gracefully in educational contexts

Moodle-Specific Behavioral Patterns Identified:

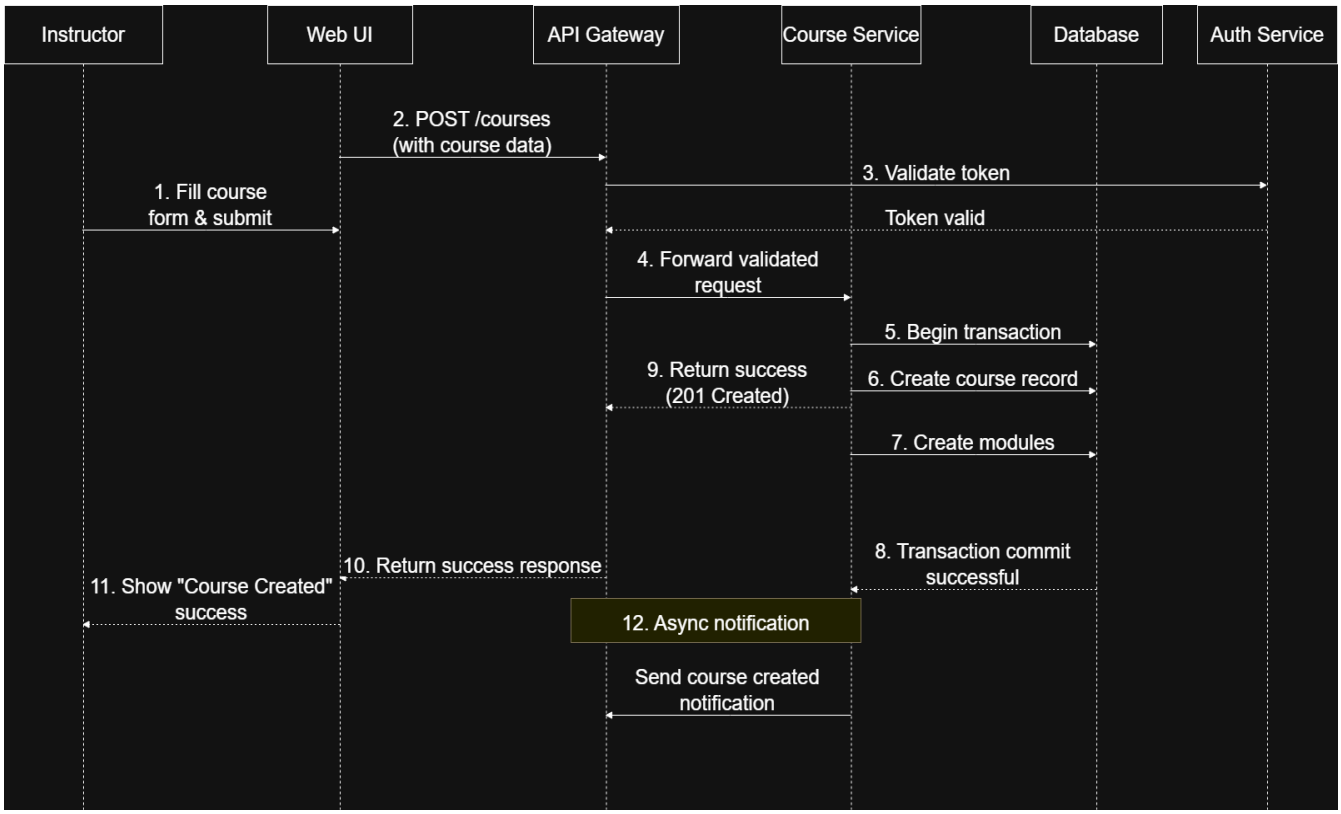
- Asynchronous Operations: Email notifications, file processing
- Synchronous Validation: Real-time form validation, plagiarism checking
- State Transitions: Course publishing, enrollment approval, grade release
- Event-Driven Architecture: Notifications triggered by system events

DIAGRAM PLACEMENT & EXPLANATIONS

Sequence Diagrams Section

Course Creation Sequence Diagram

Shows step-by-step interaction between components during course creation.



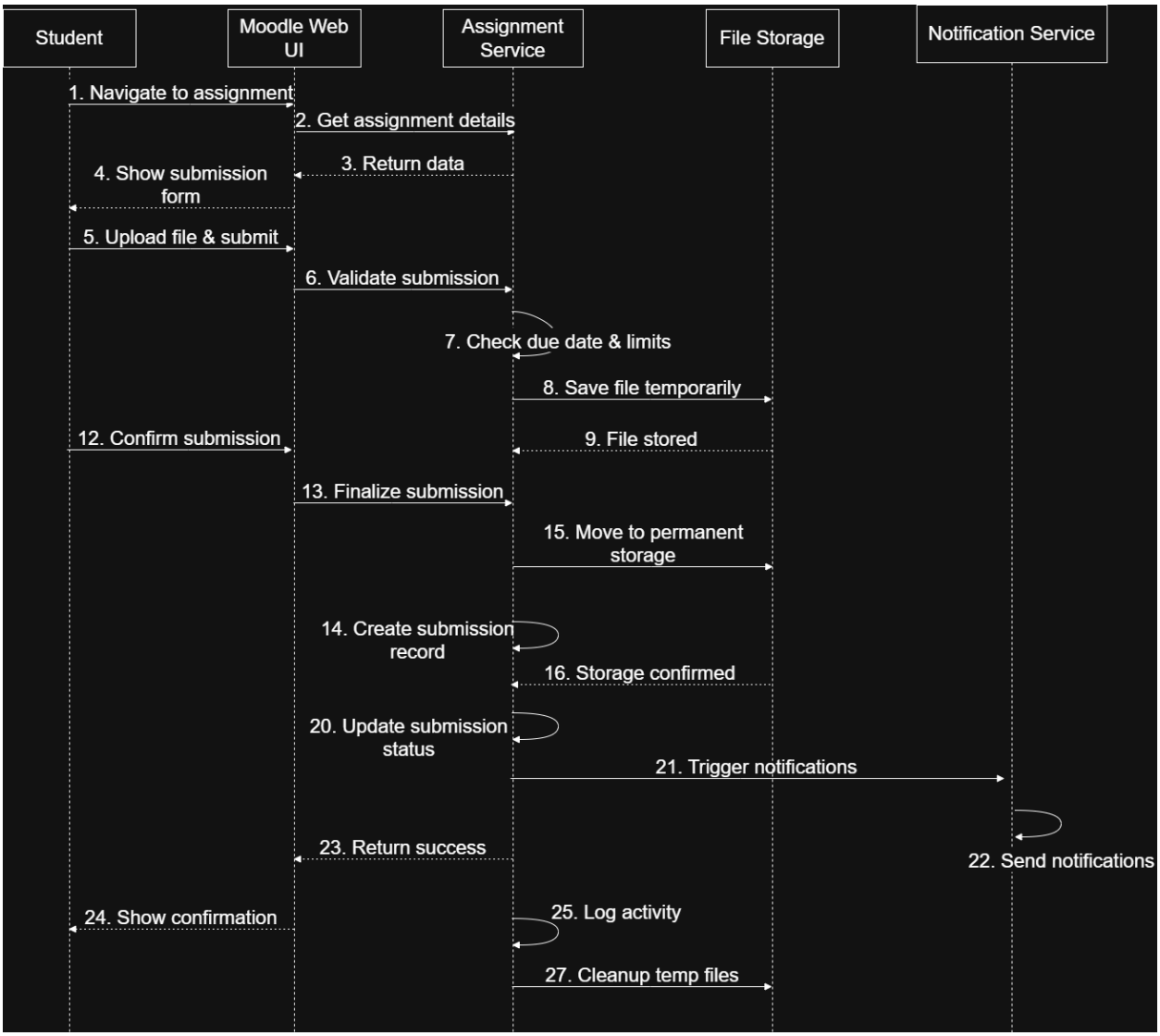
Moodle-Specific Features Illustrated:

- Request/Response Pattern - Synchronous communication
- Transaction Management - Database integrity
- Layered Architecture - Clear separation
- Error Handling - Graceful failure management

Why Important: Understanding real-time system behavior and component interactions

Assignment Submission Sequence

Moodle's assignment module handles file uploads, online text, and various submission types with pluggable feedback systems.



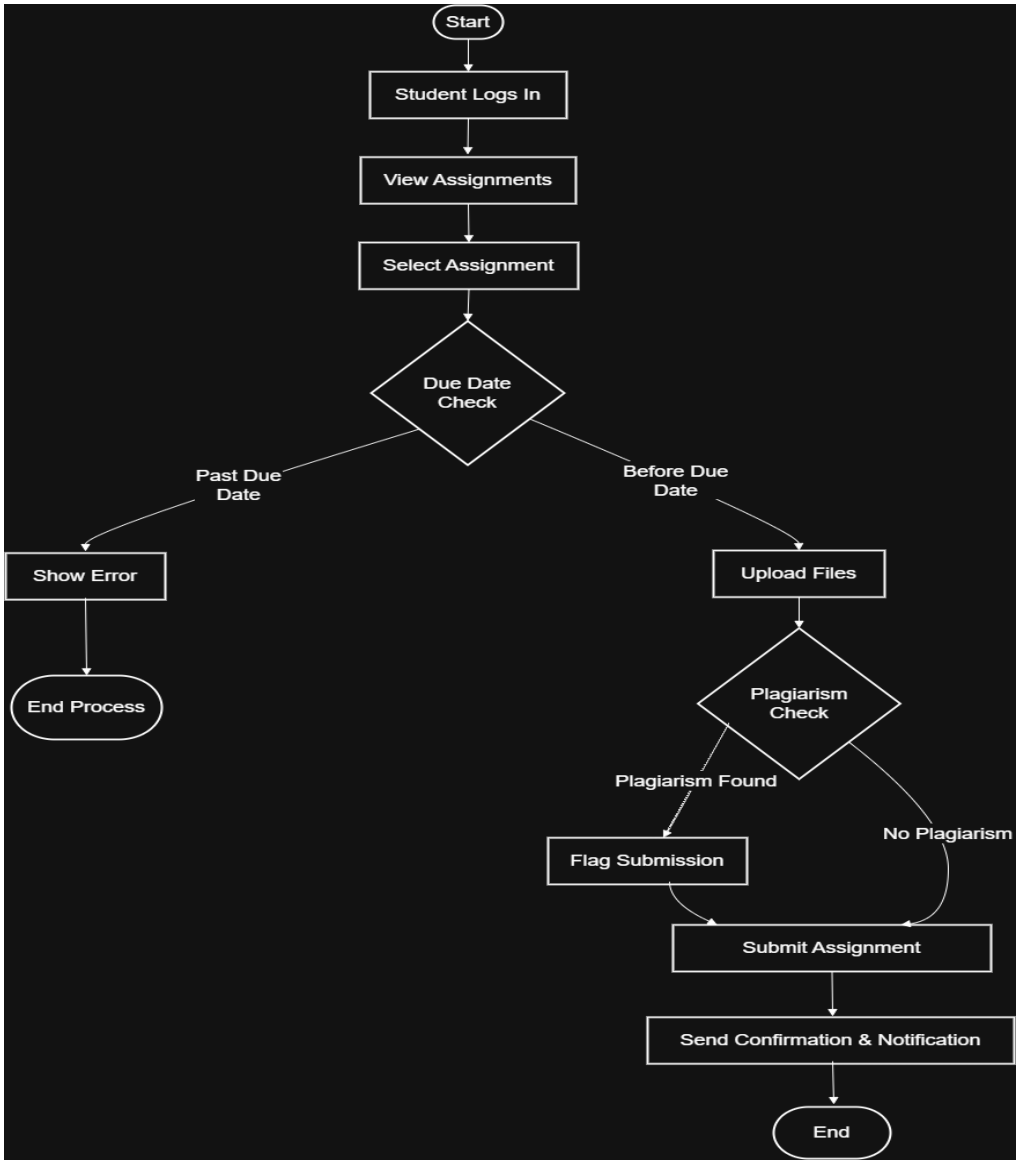
Moodle Components Shown:

1. Student accesses assignment and submits work
2. System validates submission (due date, file type, limits)
3. File gets stored temporarily, then permanently
4. Optional plagiarism check if enabled
5. Notifications sent to relevant parties
6. Confirmation shown to student

Activity Diagrams Section

Course Enrollment Workflow

Moodle supports multiple enrollment methods: manual, self, guest, cohort, and plugin-based enrollments.



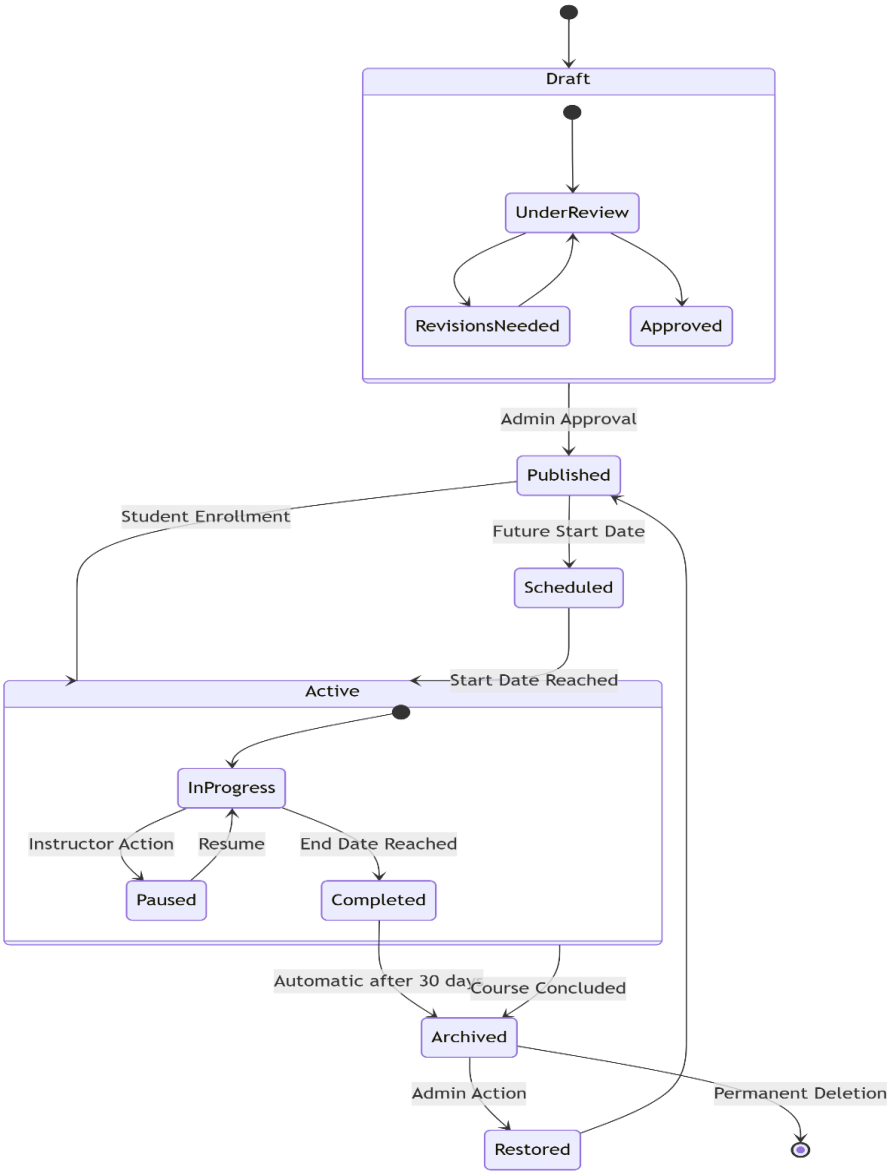
Moodle Enrollment Types:

- **Student Logs In** → Authentication to access the learning platform
- **Navigate to Course** → Access the specific course containing assignments
- **View Assignments** → List all available assignments
- **Select Assignment** → Choose the specific assignment to submit

State Machine Diagrams Section

Course Lifecycle States

Courses in Moodle have visibility settings and availability dates that control student access.

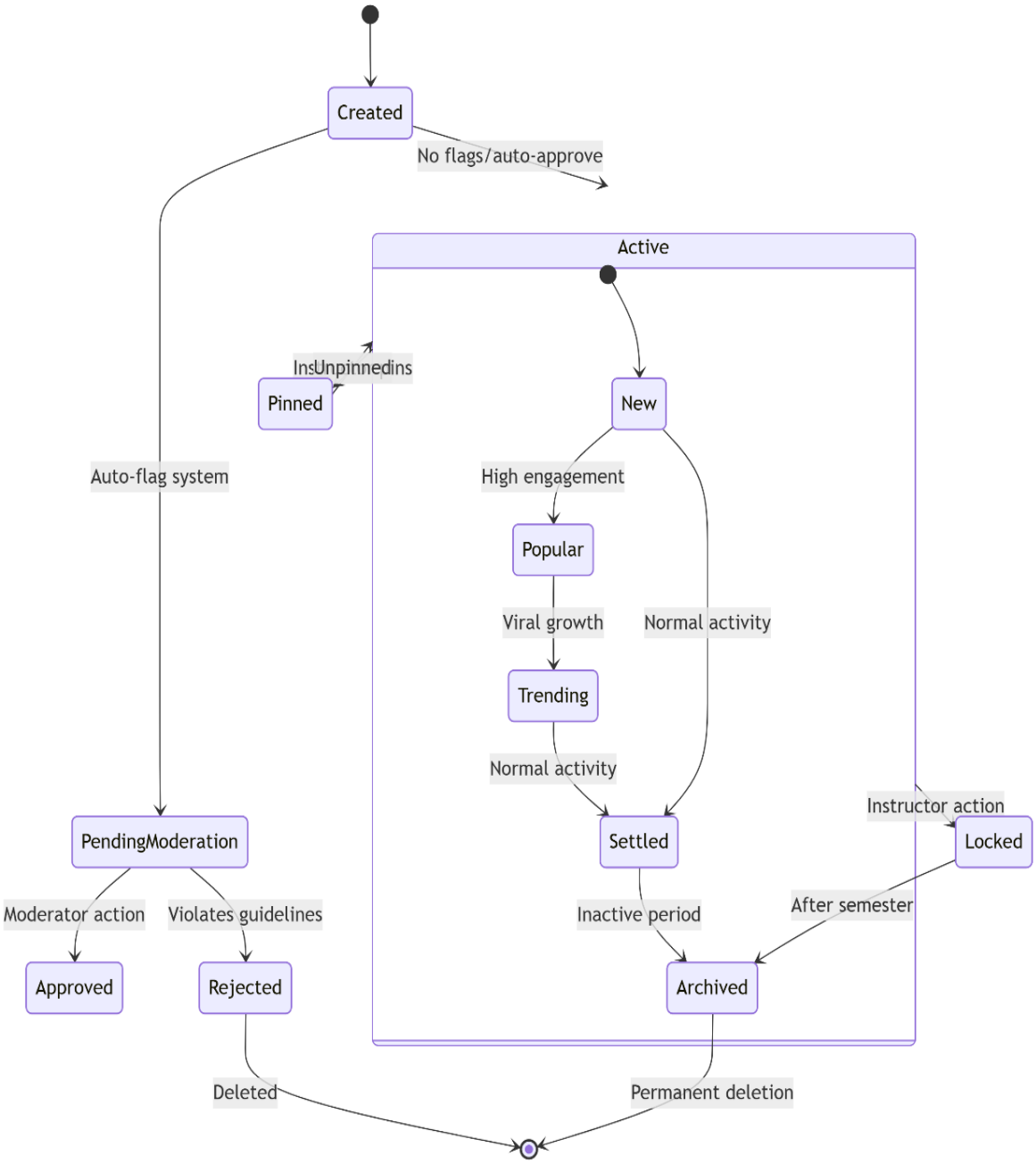


Moodle Course States Explained:

- Draft: Course being created (visible=0)
- Published: Course available (visible=1)
- Restricted: Published but before start date
- Active: Students actively participating
- Completed: Past end date but still accessible

Forum Discussion States

Moodle forums support moderated discussions, graded discussions, and multiple forum types.



Moodle Forum Features:

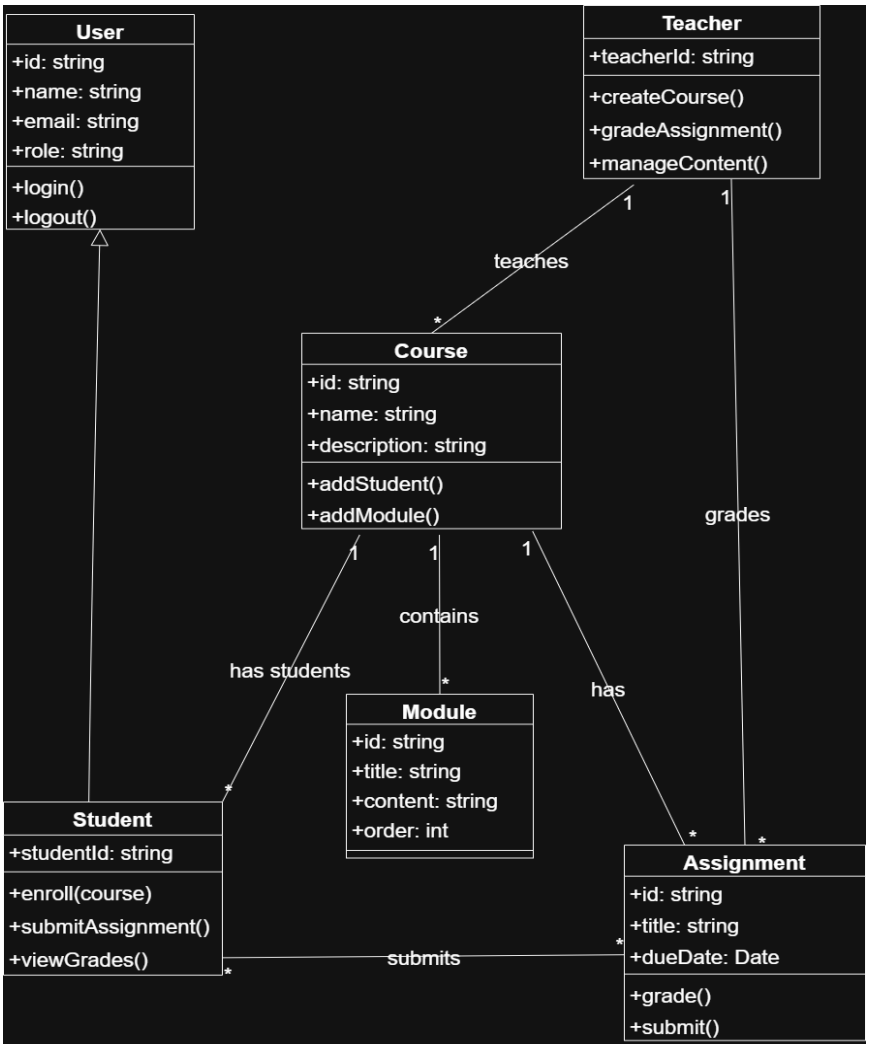
1. Post timing control: Cut-off dates for posting
2. Subscription options: Email digests of new posts
3. Attachment support: Files can be attached to posts
4. Rating system: Posts can be rated by users.

5. STRUCTURAL DESIGN

Structural design focuses on the static architecture of the system - the "bones" that don't change during runtime. We'll create three key diagrams that show different perspectives of the system structure:

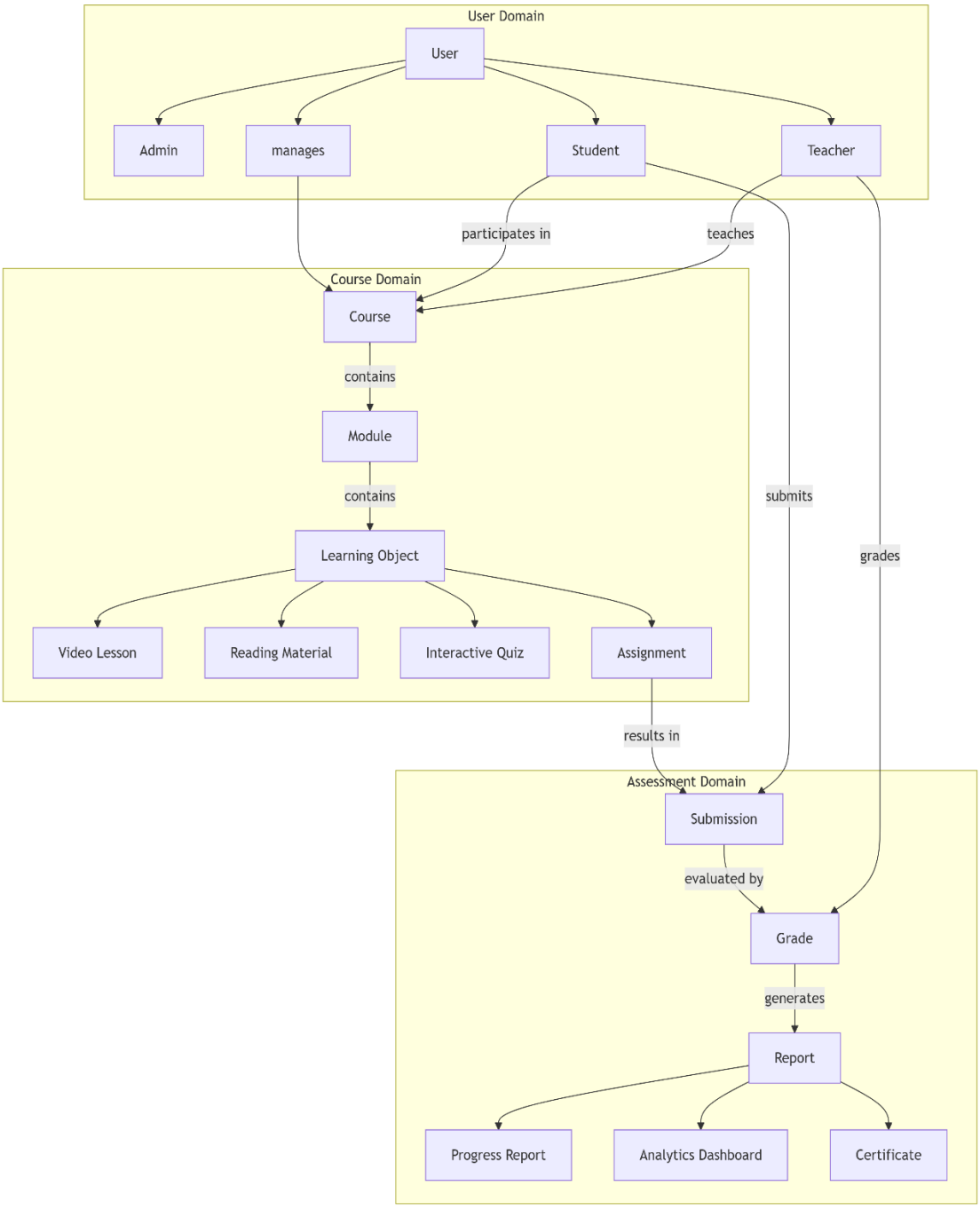
- Class Diagram: Shows object-oriented relationships
- Domain Model: Shows business concepts and relationships
- ERD (Entity Relationship Diagram): Shows database tables and relationships

Class Diagram



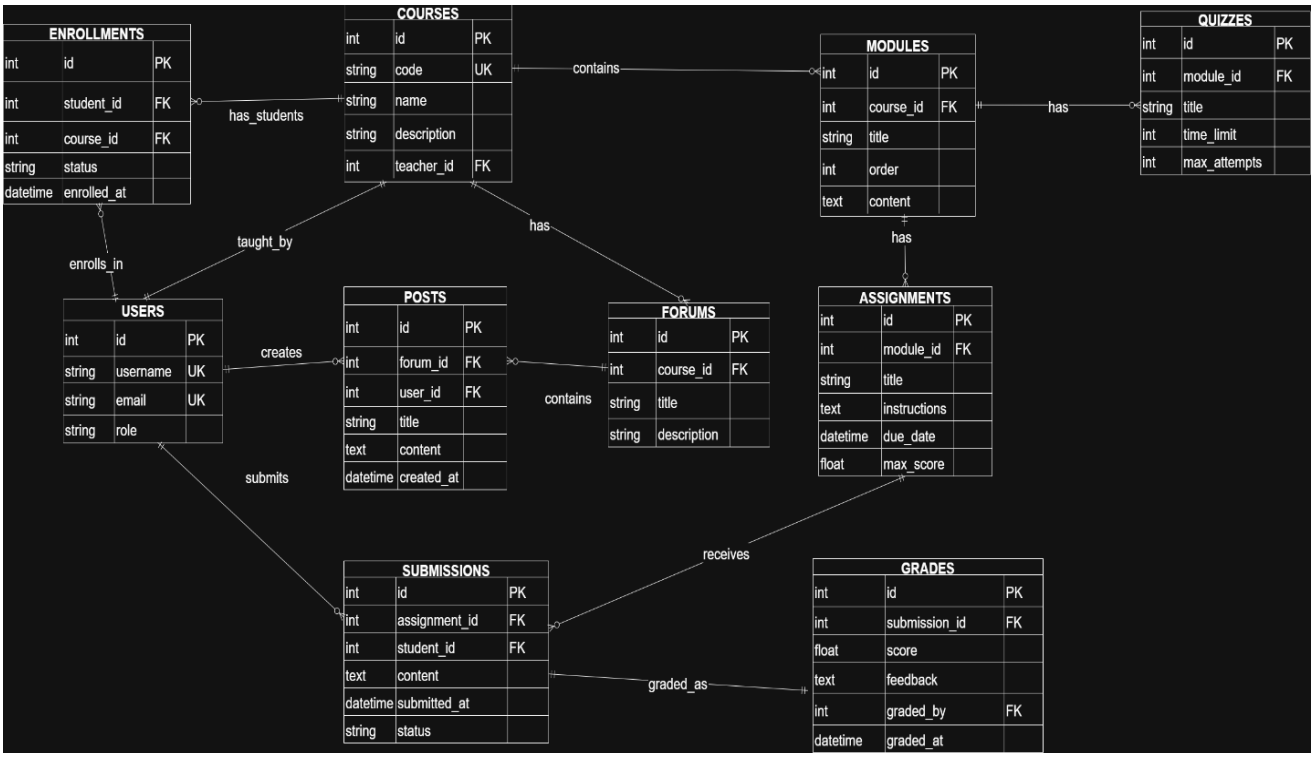
- User is base class inherited by Student and Teacher
- Course contains multiple Modules
- Modules can have multiple Assignments
- Students enroll in Courses and submit Assignments

Domain Model



- **Four main domains**: User, Course, Assessment
- **User Domain**: Different user roles with specific capabilities
- **Course Domain**: Hierarchical structure from Course to learning resources
- **Assessment Domain**: Submission → Grade → Feedback flow

ERD (Entity Relationship Diagram)



Explanation:

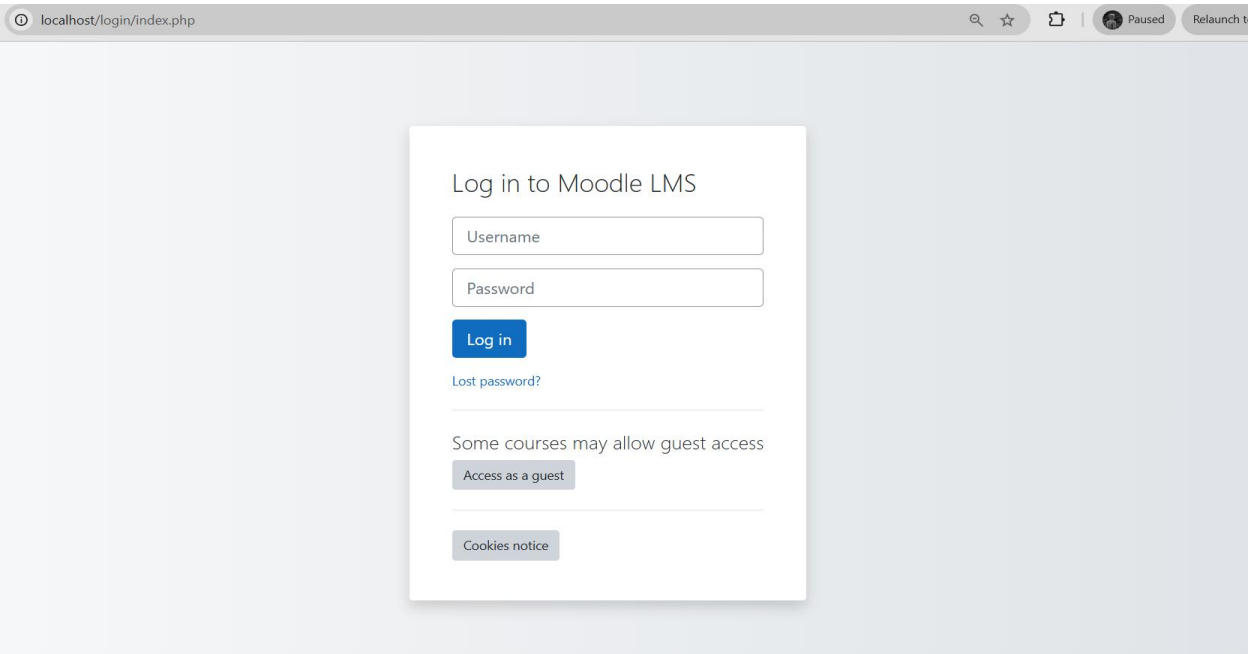
- **Core tables**: USERS, COURSES, MODULES, ASSIGNMENTS
- **Relationship tables**: ENROLLMENTS (M:N between USERS and COURSES)
- **Submission flow**: ASSIGNMENTS → SUBMISSIONS → GRADES
- **Forum system**: USERS create FORUM_POSTS within COURSES
- **Foreign keys**: All relationships properly defined with FK constraints

6. UI/UX Design Analysis

Moodle, as the world's most popular open-source Learning Management System, has undergone significant UI/UX transformations since its inception. This analysis explores the journey from Moodle's traditional interface to modern design approaches, examining how evolving educational needs and technological advancements have shaped its user experience.

LOGIN PAGE

OLD MOODLE LOGIN:



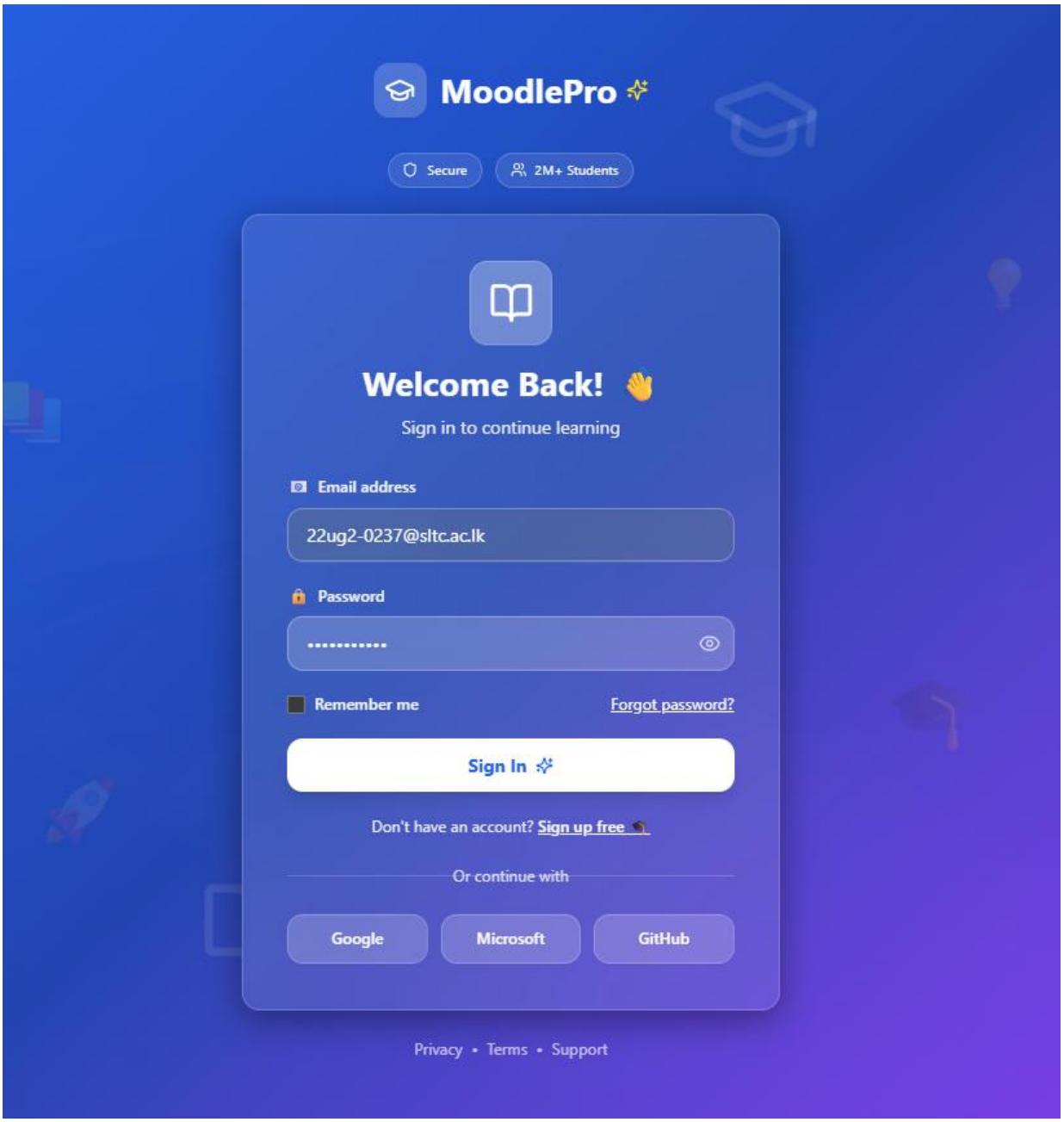
Problems Identified:

- Outdated Visual Design: Plain, minimal styling (circa 2000s)
- Poor Accessibility: Low contrast ratios
- No Modern Auth Options: Missing social login, SSO
- Limited Error Feedback: Basic error messages
- No Visual Hierarchy: All elements have equal weight
- Not Mobile-Friendly: Fixed width, poor responsive design

Usability Issues:

- Small click targets
- No password visibility toggle
- No "Remember me" functionality
- Missing form validation in real-time
- Poor error recovery

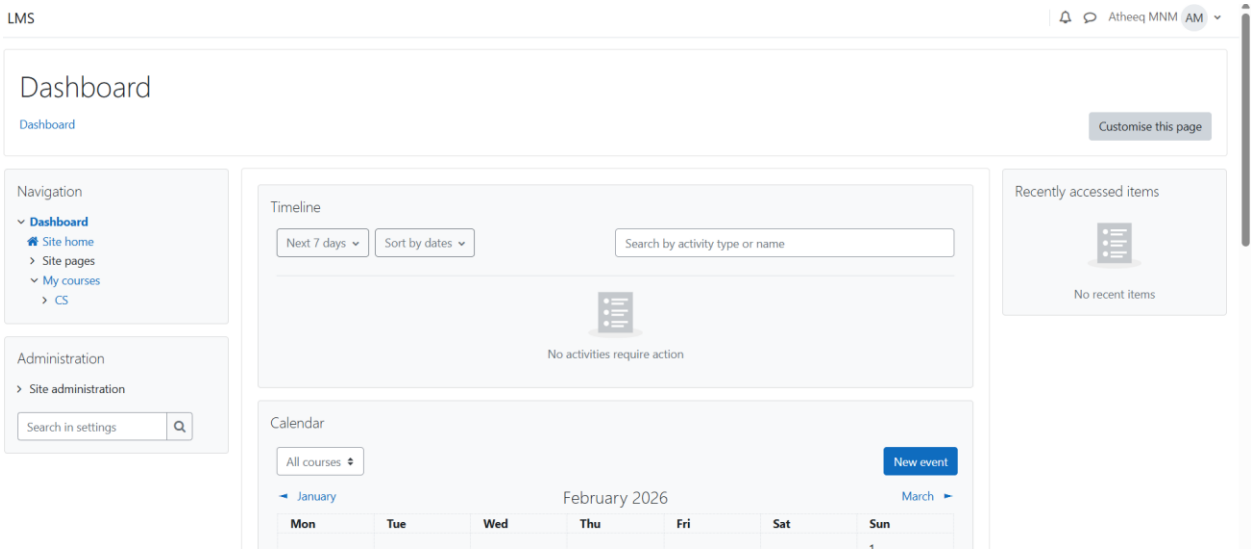
NEW MOODLE REDESIGN:



Moodle's redesign elevates user experience through modern aesthetics, enhanced usability, and inclusive accessibility. Clean gradients, card-based layouts, and consistent typography create visual appeal. Practical features like password visibility toggle and real-time validation improve usability.

DASHBOARD

OLD MOODLE DASHBOARD:



Problems Identified:

1. Information Overload:
   - Dense text without visual hierarchy
   - No prioritization of important information
   - Limited personalization

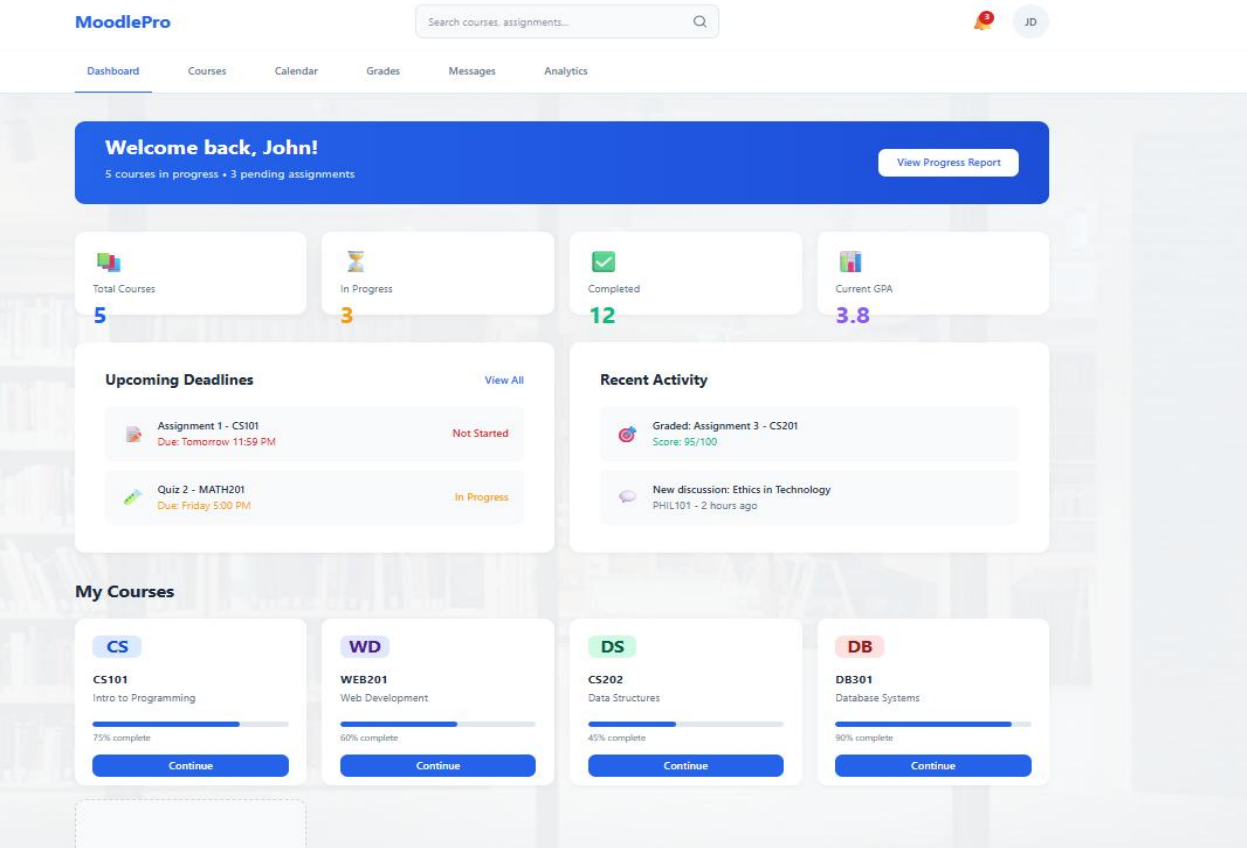
2. Poor Visual Design:
   - Table-based layouts with borders
   - Basic white color scheme
   - No visual indicators for status
   - Minimal use of icons
   - Fixed-width design

3. Navigation Issues:
   - Deep nested menus
   - Inconsistent navigation patterns
   - Poor mobile navigation
   - Hidden important features

4. Performance Problems:
   - Slow loading with many courses

- No lazy loading of content
- No offline capabilities

NEW MOODLE DASHBOARD REDESIGN:



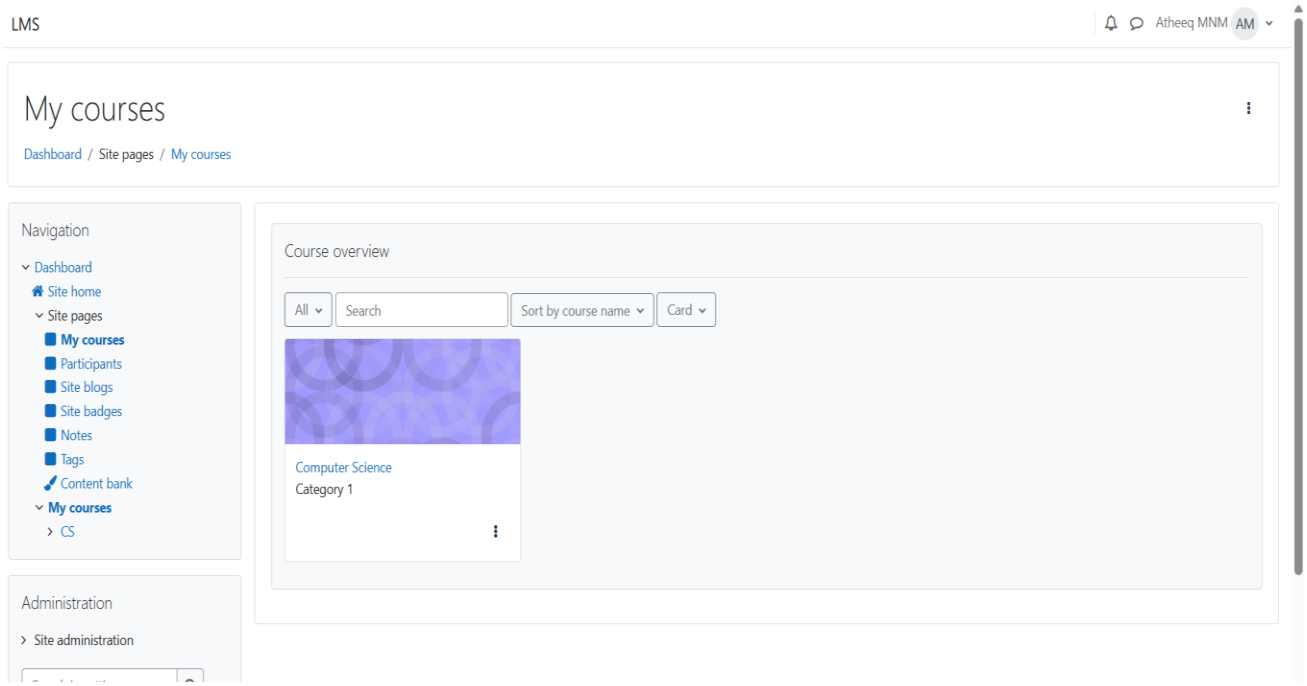
Improvements Implemented:

1. Personalized Dashboard:
   - Dynamic content: Based on user role and preferences
   - Smart prioritization: Important deadlines highlighted
   - Customizable widgets: Users can rearrange dashboard components
2. Visual Hierarchy & Design:
   - Card-based layout: Clear separation of content areas
   - Modern typography: Clear hierarchy with proper font sizing
3. Enhanced Navigation:
   - Persistent top navigation: Always accessible core features
   - Breadcrumb trails: Clear location awareness
   - Quick access menu: Frequently used actions
   - Smart search: Predictive search with filters
4. Performance Optimizations:
   - Lazy loading: Content loads as needed

- Progressive enhancement: Core functionality first
- Caching strategy: Smart caching of frequent data

COURSE MANAGEMENT

OLD MOODLE COURSE INTERFACE:

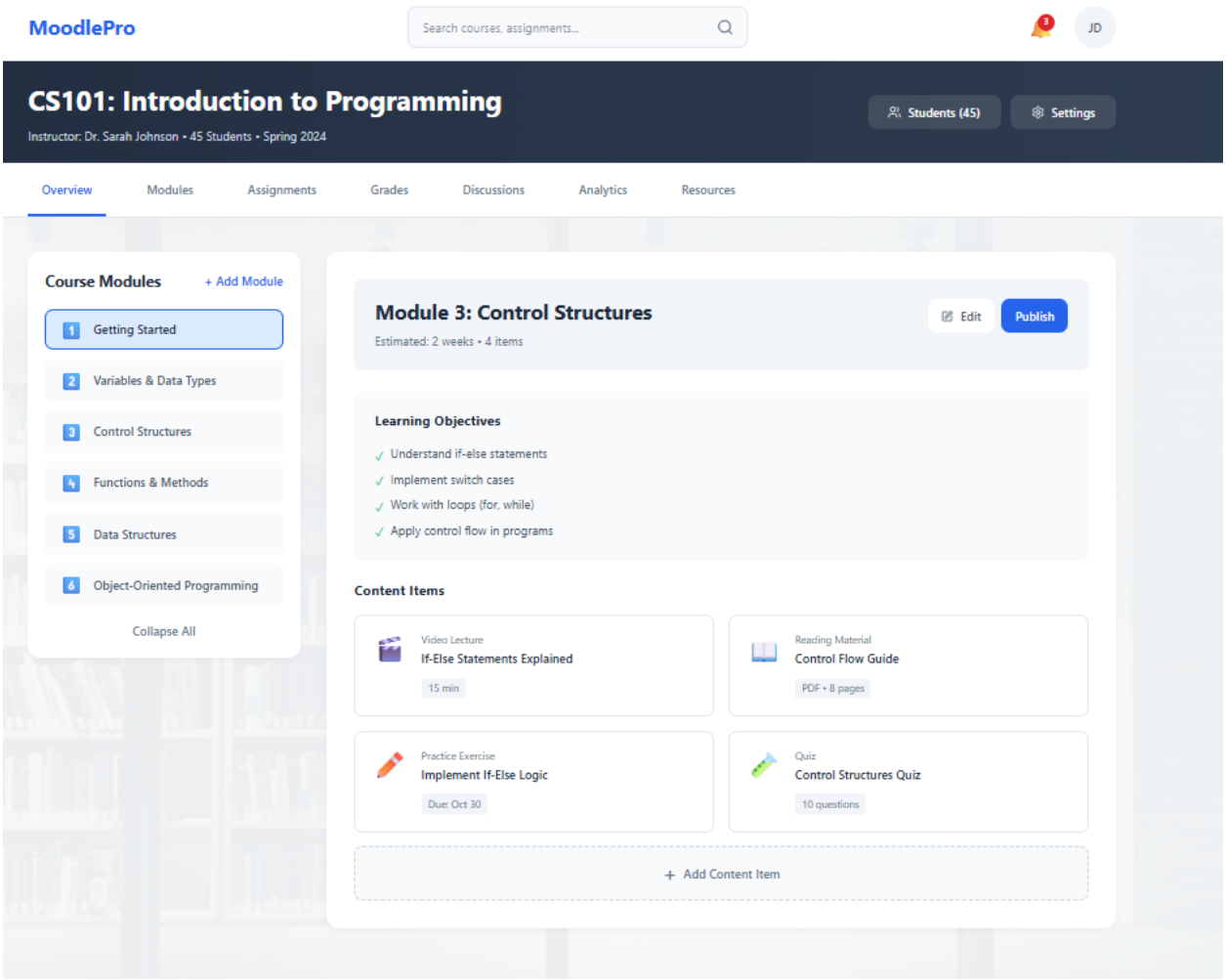


Problems Identified:

1. Poor Content Organization:
   - Linear list structure
   - No visual hierarchy
   - Limited content types
   - Hidden important features
2. Limited Interaction Design:
   - Click-heavy navigation
   - No real-time updates
   - Poor content discovery
   - Limited customization
3. Instructor Experience Issues:
   - No templates
   - Limited analytics
   - Poor student management tools
4. Student Experience Problems:
   - Overwhelming content lists

- Poor mobile experience
- Limited engagement tools

NEW MOODLE COURSE MANAGEMENT REDESIGN:



Improvements Implemented:

1. Modern Content Management:
   - Visual module builder: Drag-and-drop interface
   - Rich content types: Video, interactive, documents, quizzes
   - Learning path creation: Sequence content logically
   - Template library: Pre-built course structures
2. Enhanced Teaching Tools:
   - Real-time analytics: Student progress dashboards
   - Automated grading: Rubric-based assessment
   - Plagiarism detection: Integrated tools
   - Peer review: Built-in peer assessment workflows

- Communication tools: Announcements, discussions, messaging

3. Student-Centric Design:
   - Progress tracking: Visual completion indicators
   - Personalized learning: Adaptive content recommendations
   - Engagement features: Gamification elements
   - Mobile optimization: Full course access on mobile
   - Offline mode: Download content for offline access

4. Accessibility & Inclusivity:
   - Universal design: Accessible to all learners
   - Multiple content formats: Text, audio, video, interactive
   - Language support: Multi-language course materials
   - Accommodation tools: Extended time, alternative formats
   - Screen reader optimization: Course content accessible

7. CONCLUSION

This project has been a comprehensive exploration of software design principles applied to a real-world system. By analysing Moodle an established, complex open-source LMS I've had the opportunity to bridge academic theory with professional practice in a meaningful way.

Starting with the software selection, Moodle proved to be an excellent candidate with its modular architecture and clear workflows. Installing and navigating the system gave me firsthand insight into how large educational platforms function in practice. The requirement modelling phase revealed both the system's strengths and areas where modern enhancements could significantly improve user experience.

The architectural redesign from a monolithic to microservices-based approach represents the project's most substantial contribution. This transition directly addresses scalability and maintainability concerns that educational institutions face as they grow. The proposed architecture doesn't just copy modern trends but specifically targets Moodle's real limitations while respecting its educational mission.

Through the behavioral and structural design activities, I translated high-level architecture into concrete system behaviors and static structures. Creating sequence diagrams for workflows like course creation and assignment submission forced me to think through every interaction detail—exactly what professional software engineers do daily. The enhanced class structure and database design provide a blueprint that could actually be implemented.

The UI/UX redesign focused on making the system more intuitive and accessible. Educational technology should reduce friction, not create it. By applying established usability principles and designing with actual users in mind, the proposed interfaces aim to make learning management more efficient for everyone students, instructors, and administrators alike.

Throughout this project, I've learned that good software design isn't about choosing the trendiest patterns or technologies. It's about making thoughtful, justified decisions that balance competing priorities performance versus complexity, features versus usability, innovation versus stability. The diagrams and explanations in this document represent my best attempt at that balancing act.

What began as an assignment has become a case study in how educational technology can evolve to better serve its purpose. The proposed enhancements real-time collaboration, predictive analytics, modern interfaces aren't just technical improvements. They're ways to make learning more engaging, teaching more effective, and administration more efficient.

This work demonstrates that applying professional software engineering principles to existing systems isn't just theoretical it's how we make technology better serve human needs. The skills practiced here from requirement analysis to architectural design to evaluation—are exactly what the industry needs and what will make me a more effective software engineer.

The journey from existing system analysis to comprehensive redesign has been challenging but rewarding. It has transformed my understanding of what it means to design software that doesn't just work, but works well for real people in real educational contexts. That's ultimately what this project and good software design is all about.

GitHub Link –

https://github.com/atheeqnasrullah/Software-Modelling-and-Analysis-.git