**COMSATS University Islamabad,   
Abbottabad Campus**

**SOFTWARE DESIGN DESCRIPTION**

**for**

**AI-Assisted Learning Management System**  
Version 0.1

***By***

**Hanzla Nouman CIIT/FA21-BSE-015/ATD**

**Laiba Binta Tahir CIIT/FA21-BSE-019/ATD**

**Arfah Ali CIIT/FA21-BSE-080/ATD**

***Supervisor*****Mr. Mukhtiar Zamin**

***Bachelor of Science in Computer Science (2021-2025)***

**Table of Contents**

**Revision History 5**

**1.** **Introduction 7**

**2.** **Design Methodology and software process model 7**

**3.** **System Overview 7**

3.1 Architectural Design 8

3.2 Process Flow/Representation 9

**4.** **Design Models 13**

**5.** **Data Design 14**

5.1 Data Dictionary 15

**6.** **Algorithm & Implementation 18**

**7.** **Software Requirements Traceability Matrix 23**

**8.** **Human Interface Design 24**

8.1 Screen Images 24

8.2 Screen Objects and Actions 29

**List of Figures**

Figure 1 Architecture diagram ………..8

Figure 2 Activity diagram: Access recommended resources ………..9

Figure 3 Activity diagram: Create Lecture ………..10

Figure 4 Activity diagram: Create Quiz ………..11

Figure 5 Class diagram ………..12

Figure 6 Sequence diagram ………..13

Figure 7 LMS Dashboard ………..23

Figure 8 Enrolled Courses ………..24

Figure 9 Specific course screen ………..24

Figure 10 Content Generation Screen ………..25

Figure 11 Assignment Screen ………..25

Figure 12 Create Assignment Screen ………..26

Figure 13 Evaluate Assignment Screen ………..26

Figure 14 Quizzes Screen ………..27

Figure 15 Create Quiz Screen ………..27

Figure 16 Quiz Submission Screen ........ ………..28

Figure 17 Evaluate Quiz Screen ........ ………..28

**List of Tables**

Table 1 Software requirements traceability matrix 22

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason for changes** | **Version** |
|  |  |  |  |
|  |  |  |  |

**Application Evaluation History**

|  |  |
| --- | --- |
| **Comments (by committee)**  **\*include the ones given at scope time both in doc and presentation** | **Action Taken** |
|  |  |
|  |  |

**Supervised by**

**Mr. Mukhtiar Zamin**

Signature\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Introduction**

The AI-Assisted Learning Management System (LMS) has made significant progress, emphasizing modularity, scalability, and efficiency. A key achievement is the Content Generation Module, powered by a Retrieval-Augmented Generation (RAG) system that produces high-quality educational materials, including lecture notes, quizzes, and supplementary resources. The Authentication System ensures role-based access for students, teachers, and administrators, using unique identifiers and secure token management for reliable session handling.

The LMS features a scalable Data Model and Backend Infrastructure, with NestJS as the framework and MongoDB as the database, enabling efficient storage and retrieval. AI integration, supported by a Llama-based server and vector embeddings, enhances personalized content recommendations and adaptive learning analytics. Operational features include tailored content generation, AI-assisted lecture and quiz creation, and a robust role-based authentication system, establishing a solid foundation for future enhancements.

**Design methodology and software process model**

The object-oriented programming (OOP) approach has been chosen for the design of the AI-Assisted LMS. This methodology ensures modularity, reusability, and scalability by representing the system as a collection of interacting objects, each encapsulating data and behavior. The system’s components, such as user management, content generation, and analytics, align well with OOP concepts like inheritance, polymorphism, and encapsulation.

OOP allows the team to implement features independently, ensuring better collaboration and debugging during development. The design aligns with industry-standard practices, promoting maintainability and extensibility for future updates.

The Agile Software Development Model is employed for this project. Agile emphasizes iterative development and continuous feedback, ensuring that the evolving requirements of users (teachers, students, and administrators) are addressed promptly. Agile’s adaptability is ideal for this project, as features like AI-based content generation and recommendations may require frequent adjustments based on usability testing and stakeholder feedback. The iterative approach also allows for modular delivery, where each sprint focuses on developing and validating specific modules of the system.

**System overview**

The AI-Assisted LMS is a comprehensive platform that integrates traditional learning management features with advanced AI functionalities:

* Personalized Learning: Provides tailored recommendations for learning materials based on user preferences, performance, and engagement patterns.
* Automated Evaluations: Offers instant feedback for quizzes and AI-driven assessment of written responses.
* Content Generation: Assists teachers in creating lecture content using AI tools.
* Performance Analytics: Generates actionable insights for teachers and administrators.
* Attendance Management: Automates attendance tracking.

Traditional LMS platforms often lack adaptability and fail to cater to diverse learning needs. This system bridges the gap by automating routine tasks, enhancing user engagement, and providing data-driven insights. It serves a wide range of educational institutions, from schools to universities, ensuring scalability and flexibility in operation.

**Architectural design**

The AI-Assisted LMS system is designed using a Modular Monolithic Architecture approach, ensuring that each module focuses on a specific functionality while remaining part of a unified system. This enables scalability, maintainability, and enhanced collaboration between components.

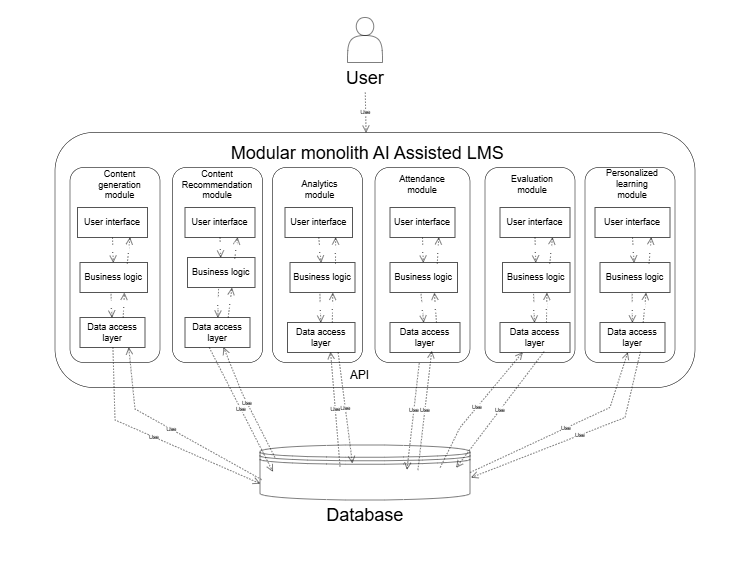


Figure 1 Architecture diagram

**Process flow/Representation**

We have created activity diagrams for major processes of the system to represent the flow of actions effectively. Below are the descriptions of each activity:

**Activity Diagram: Access recommended resources**

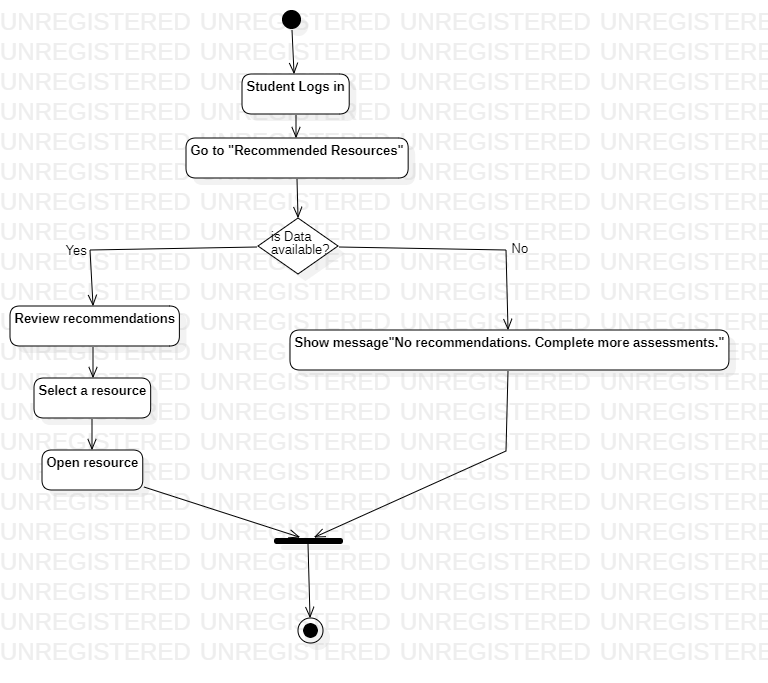


Figure 2 Activity diagram: Access recommended resources

**Activity Diagram: Create lecture**

The teacher creates lecture content using the AI-assisted content generation feature. Optionally, the teacher can manually edit AI-generated content.

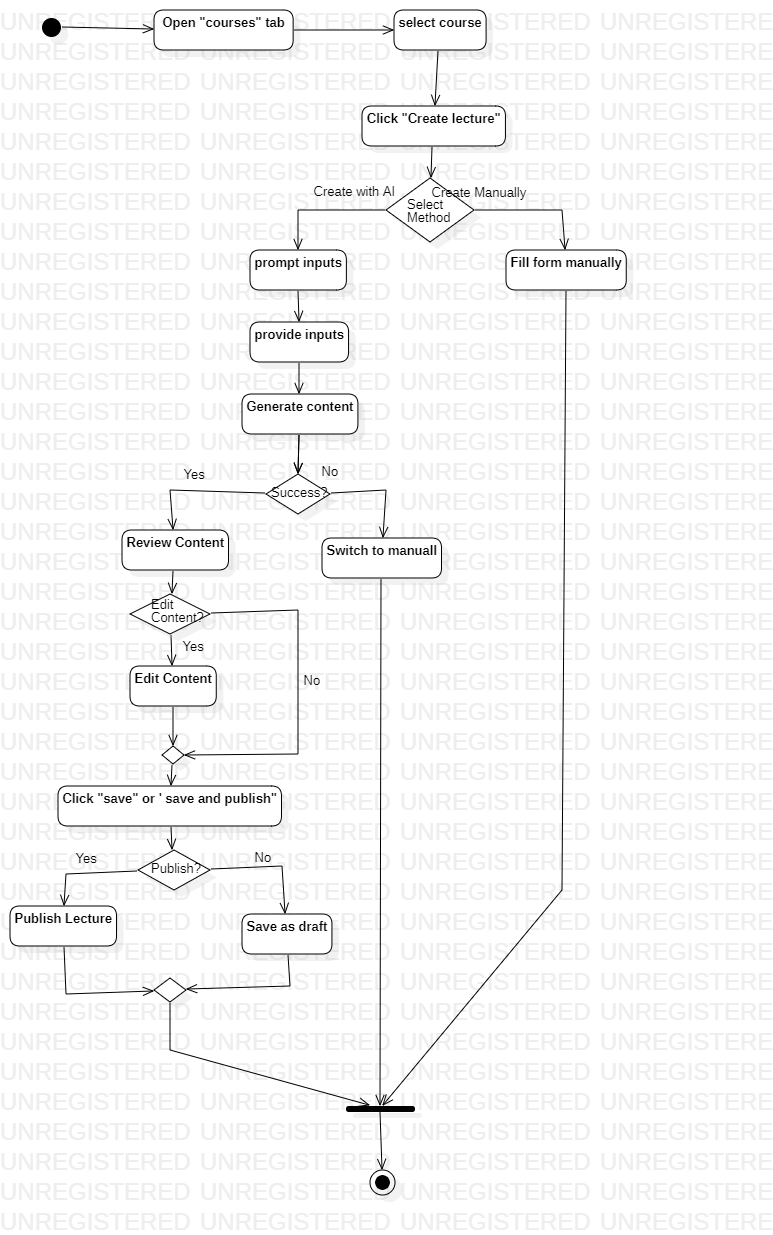


Figure 3 Activity Diagram: Create lecture

**Activity Diagram: Create Quiz**

The teacher creates and schedules a quiz by entering questions, defining correct answers, and assigning marks.

A screenshot of a phone

Description automatically generated

Figure 4 Activity Diagram: Create quiz

**Design models**

For our project, we are utilizing an object-oriented approach, and therefore, the following design models are included: The applicable models may include:

**Class diagram**

This diagram captures the relationships between classes using associations (e.g., courses enrolling users), aggregations (e.g., quizzes containing questions), and inheritance (e.g., lectures and quizzes extending content). The design reflects a modular, scalable structure capable of supporting advanced LMS functionalities.

A diagram of a computer flowchart

Description automatically generated

Figure 5 Class diagram

**Sequence diagram**

This sequence diagram illustrates the process of lecture creation, review, publishing, and access in a Learning Management System (LMS), showing interactions between Teachers, Students, the LMS Frontend, Backend, Database, and Content Generation module (LLM).

A black and white text on a black background

Description automatically generated

Figure 6 Sequence diagram

**Data design**

The system's data design transforms its information domain into structured data models. The LMS relies on non-relational databases and vector databases

**Data Organization**

* User Data: Includes profiles, roles, and login details, stored securely with encryption.
* Resource Data: Metadata for educational materials, including topics, difficulty levels, and engagement statistics.
* Engagement Metrics: Tracks time spent, completion rates, and feedback for resources.
* Assessment Data: Stores quiz scores, assignment grades, and flagged items for review.
* Analytics Data: Aggregates performance trends and predictive insights for reporting

**Data dictionary**

Below is an alphabetically organized data dictionary that lists the objects in the system, along with their attributes, methods, and method parameters, based on an Object-Oriented (OO) description. This dictionary serves as a reference for understanding the data structure and behavior of the system.

**1. Analytics**

Attributes:

id: Unique identifier.

user: Reference to the user (student or teacher).

course: Reference to the associated course.

scores: Map of assessment IDs to scores.

engagementRate: Percentage of course content accessed.

feedback: Map of assessment IDs to feedback comments.

Methods:

addScore(assessmentId: string, score: number): Adds a score for a specific assessment.

updateEngagementRate(rate: number): Updates the engagement percentage.

addFeedback(assessmentId: string, feedback: string): Adds feedback for a specific assessment.

**2. Assignment**

Attributes:

id: Unique identifier.

title: Title of the assignment.

course: Reference to the associated course.

dueDate: Deadline for the assignment.

createdBy: Reference to the teacher who created the assignment.

questions: List of question IDs.

Methods:

addQuestion(questionId: string): Adds a question to the assignment.

removeQuestion(questionId: string): Removes a question from the assignment.

updateDueDate(newDate: Date): Updates the due date for the assignment.

**3. Attendance**

Attributes:

id: Unique identifier.

course: Reference to the course.

date: Date of the attendance session.

presentStudents: List of student IDs marked present.

absentStudents: List of student IDs marked absent.

Methods:

markPresent(studentId: string): Marks a student as present.

markAbsent(studentId: string): Marks a student as absent.

generateReport(): Generates a summary of attendance for the session.

**4. Content**

Attributes:

id: Unique identifier.

title: Title of the content.

course: Reference to the associated course.

type: Type of content (e.g., lecture, quiz, note).

description: Brief summary.

createdBy: Reference to the teacher who created it.

attachments: List of file or media links.

Methods:

addAttachment(fileLink: string): Adds an attachment to the content.

updateDescription(newDescription: string): Updates the content description.

**5. Course**

Attributes:

id: Unique identifier.

name: Name of the course.

code: Unique course code.

description: Description of the course.

students: List of student IDs.

teachers: List of teacher IDs.

content: List of content IDs.

isActive: Boolean indicating if the course is active.

Methods:

enrollStudent(studentId: string): Adds a student to the course.

assignTeacher(teacherId: string): Assigns a teacher to the course.

addContent(contentId: string): Adds content to the course.

deactivateCourse(): Marks the course as inactive.

**6. Event**

Attributes:

id: Unique identifier.

type: Type of event (e.g., login, content\_view).

user: Reference to the user who triggered the event.

content: Reference to the related content (optional).

metadata: Additional information about the event.

Methods:

logEvent(): Saves the event to the system log.

addMetadata(key: string, value: any): Adds extra information to the event metadata.

**7. Lecture**

Attributes:

id: Unique identifier.

title: Title of the lecture.

course: Reference to the associated course.

description: Brief overview.

attachments: List of multimedia or supporting files.

createdBy: Reference to the teacher who created the lecture.

Methods:

addAttachment(fileLink: string): Adds an attachment to the lecture.

updateDescription(newDescription: string): Updates the lecture description.

**8. Quiz**

Attributes:

id: Unique identifier.

title: Title of the quiz.

course: Reference to the associated course.

questions: List of question IDs.

dueDate: Deadline for quiz submission.

createdBy: Reference to the teacher who created the quiz.

Methods:

addQuestion(questionId: string): Adds a question to the quiz.

removeQuestion(questionId: string): Removes a question from the quiz.

updateDueDate(newDate: Date): Updates the quiz deadline.

**9. Question**

Attributes:

id: Unique identifier.

text: Text of the question.

type: Type of question (e.g., multiple-choice, short answer).

options: List of answer options (for multiple-choice).

correctAnswer: The correct answer.

Methods:

addOption(option: string): Adds an answer option.

updateAnswer(correctAnswer: string): Updates the correct answer.

**10. Token**

Attributes:

id: Unique identifier.

user: Reference to the associated user.

token: JWT or session token.

expiresAt: Expiration date and time.

Methods:

refreshToken(): Refreshes the session token.

isValid(): Checks if the token is still valid.

**11. User**

Attributes:

id: Unique identifier.

name: Full name of the user.

identifier: Unique login identifier (email or registration number).

password: Encrypted password.

role: User role (student, teacher, admin).

email: Email address (optional for students).

registrationNumber: Registration number (only for students).

coursesEnrolled: List of course IDs the user is enrolled in.

coursesAssigned: List of course IDs the user is assigned to teach.

Methods:

enrollCourse(courseId: string): Enrolls the user in a course.

assignCourse(courseId: string): Assigns the user to teach a course.

updatePassword(newPassword: string): Updates the user's password.

**12. Feedback**

Attributes:

id: Unique identifier.

user: Reference to the user receiving feedback.

course: Reference to the related course.

content: Reference to the related content (optional).

message: Feedback message.

Methods:

createFeedback(message: string): Creates feedback for a user.

updateFeedback(newMessage: string): Updates the feedback message.

**13. Submission**

Attributes:

id: Unique identifier.

assignment: Reference to the associated assignment or quiz.

student: Reference to the student making the submission.

submittedAt: Timestamp of the submission.

score: Grade or score for the submission.

feedback: Reference to feedback provided for the submission.

Methods:

updateScore(newScore: number): Updates the score for the submission.

addFeedback(feedbackId: string): Links feedback to the submission.

**Algorithm & Implementation**

Below is a high-level overview and pseudo-code (PDL) for each model discussed, organized by its function within the LMS. This provides systematic descriptions of algorithms, local data requirements, and their integration into the overall system.

**1. Knowledge Gap Analysis**

Identify weak, average, or strong topic areas for each student based on assessments.

**Algorithm:**

Input:

Assessment data: quiz\_scores, question\_accuracy.

Engagement metrics: time\_spent, completion\_rates.

Process:

Calculate average performance per topic.

Classify topics based on thresholds:

Weak: <50%

Average: 50%-80%

Strong: >80%.

Output:

Topic classifications for each student.

**Pseudo-Code:**

function knowledge\_gap\_analysis(quiz\_scores, question\_accuracy, time\_spent):

topic\_scores = {}

for question in quiz\_scores:

topic = question.topic

if topic not in topic\_scores:

topic\_scores[topic] = []

topic\_scores[topic].append(question.score)

classifications = {}

for topic, scores in topic\_scores.items():

avg\_score = average(scores)

if avg\_score < 50:

classifications[topic] = "Weak"

elif avg\_score < 80:

classifications[topic] = "Average"

else:

classifications[topic] = "Strong"

return classifications

**2. Personalized Recommendations**

Provide tailored resources to students based on their performance and preferences.

**Algorithm:**

Input:

Weak topics from Knowledge Gap Analysis.

Student interaction data: time\_spent, resource\_completions.

Resource metadata: topic, difficulty.

Process:

Rank resources using a collaborative filtering algorithm.

Include semantic similarity using vector embeddings.

Output:

Ranked list of resources for each student.

**Pseudo-Code:**

function personalized\_recommendations(student\_id, weak\_topics, interaction\_data, resources):

relevant\_resources = filter\_by\_topics(resources, weak\_topics)

scores = {}

for resource in relevant\_resources:

scores[resource.id] = calculate\_similarity(student\_id, resource) + engagement\_score(resource, interaction\_data)

ranked\_resources = sort\_by\_score(scores)

return ranked\_resources

**3. Content Clustering**

Organize resources into meaningful groups by topic and difficulty.

**Algorithm:**

Input:

Resource metadata: topics, difficulty, engagement\_stats.

Process:

Convert metadata into feature vectors.

Apply k-Means clustering to group resources.

Output:

Clustered resources.

**Pseudo-Code:**

function content\_clustering(resources):

vectors = encode\_resources(resources)

clusters = k\_means(vectors, k=5) # Predefined k value

return clusters

**4. Performance Forecasting**

Predict future performance trends and identify at-risk students.

**Algorithm:**

Input:

Time-series data: quiz\_scores, attendance, engagement\_metrics.

Process:

Train an LSTM model on historical data.

Predict future scores and engagement levels.

Output:

Predicted performance trends and risk scores.

**Pseudo-Code:**

function performance\_forecasting(student\_data):

lstm\_model = train\_lstm(student\_data)

predictions = lstm\_model.predict(next\_semester\_data)

return predictions

**5. Behavioral Analytics**

Classify students into engagement levels based on activity data.

**Algorithm:**

Input:

Attendance records, login frequency, session durations.

Process:

Train a logistic regression model to classify engagement levels.

Use engagement thresholds to define categories.

Output:

Engagement classifications (e.g., Engaged, Passive, Disengaged).

**Pseudo-Code:**

function behavioral\_analytics(attendance, login\_frequency, session\_durations):

features = preprocess([attendance, login\_frequency, session\_durations])

model = train\_logistic\_regression(features, labels)

engagement\_levels = model.predict(features)

return engagement\_levels

**6. Content Effectiveness**

Evaluate the impact of resources on student performance.

**Algorithm:**

Input:

Pre- and post-assessment scores.

Engagement metrics.

Process:

Calculate resource effectiveness as the delta between pre- and post-assessment scores.

Correlate engagement metrics with score improvement.

Output:

Effectiveness scores for each resource.

**Pseudo-Code:**

function content\_effectiveness(pre\_scores, post\_scores, engagement\_metrics):

deltas = calculate\_differences(pre\_scores, post\_scores)

effectiveness = correlate(deltas, engagement\_metrics)

return effectiveness

**7. Lecture Creation Process**

To create a lecture using preprocessed insights and AI-generated content, followed by teacher review and publishing.

**Algorithm**

Input Collection:

Collect inputs from the teacher, including:

Topic.

Objectives.

Duration.

Preferred formats (notes, quizzes, multimedia).

Any references or attachments.

Fetch Preprocessed Insights:

Retrieve insights from the database, including:

Weak topics from Knowledge Gap Analysis.

Clustered and tagged resources from prior preprocessing.

Content Generation:

Pass collected inputs and preprocessed insights to the AI content generation engine (e.g., LLM).

Generate the following:

Structured lecture notes.

Relevant quizzes.

Multimedia placeholders or suggestions.

Teacher Review:

Deliver the generated content to the teacher via the frontend editor.

Allow the teacher to:

Edit notes and quizzes.

Replace or add multimedia resources.

Save as draft or finalize the content.

Save and Publish:

If saved as a draft:

Store metadata and content in the database (marked as "Draft").

If published:

Update the lecture status in the database (marked as "Published").

Make the content available for students.

Content Delivery to Students:

Students can access the published lecture via their dashboards.

Provide dynamic sections for notes, quizzes, and multimedia.

**Pseudo-Code:**

function create\_lecture(teacher\_inputs):

Step 1: Input Collection

topic = teacher\_inputs["topic"]

objectives = teacher\_inputs["objectives"]

duration = teacher\_inputs["duration"]

formats = teacher\_inputs["formats"] # e.g., notes, quizzes, multimedia

references = teacher\_inputs.get("references", [])

Step 2: Fetch Preprocessed Insights

insights = fetch\_insights\_from\_database(topic)

weak\_topics = insights["weak\_topics"]

clustered\_resources = insights["clustered\_resources"]

Step 3: Content Generation

content = generate\_content\_with\_ai(topic, objectives, weak\_topics, clustered\_resources)

AI Output

lecture\_notes = content["notes"]

quizzes = content["quizzes"]

multimedia\_suggestions = content["multimedia"]

Step 4: Teacher Review

teacher\_reviewed\_content = get\_teacher\_review(

notes=lecture\_notes,

quizzes=quizzes,

multimedia=multimedia\_suggestions

)

if teacher\_reviewed\_content["status"] == "Draft":

# Step 5a: Save as Draft

save\_to\_database(

status="Draft",

metadata={

"topic": topic,

"objectives": objectives,

"teacher\_id": teacher\_inputs["teacher\_id"]

},

content=teacher\_reviewed\_content

)

elif teacher\_reviewed\_content["status"] == "Published":

# Step 5b: Save and Publish

save\_to\_database(

status="Published",

metadata={

"topic": topic,

"objectives": objectives,

"teacher\_id": teacher\_inputs["teacher\_id"]

},

content=teacher\_reviewed\_content

)

Step 6: Notify Success

return "Lecture created successfully!"

Key Functions in the Process

fetch\_insights\_from\_database(topic):

Retrieves weak topics and clustered resources for the lecture topic from the database.

generate\_content\_with\_ai(topic, objectives, weak\_topics, clustered\_resources):

Passes insights and teacher inputs to the AI content generation engine.

Returns structured lecture content (notes, quizzes, multimedia).

get\_teacher\_review(notes, quizzes, multimedia):

Displays generated content to the teacher for editing and approval.

Returns the finalized content and status (Draft or Published).

save\_to\_database(status, metadata, content):

Saves lecture metadata and content into the database (SQL for metadata, NoSQL for content).

Expected Outputs

For Drafts:

Lecture is stored in the database with status "Draft."

Teacher can revisit and finalize later.

For Published Lectures:

Lecture content is made available to students.

Includes organized sections for notes, quizzes, and multimedia

**Software requirements traceability matrix**

This section describes how each software requirement is fulfilled through the design components and their corresponding elements. Here's a suggested template and an example of what to include:

Table 1 Software requirements traceability matrix

|  |  |  |  |
| --- | --- | --- | --- |
| **Req. Number** | **Ref. Item** | **Design Component** | **Component Items** |
| FR-PL-001 | Table 9 | Class Diagram | Personalized Content Recommendations |
| FR-PL-002 | Table 10 | Class Diagram | Interactive Material Generation |
| FR-AE-001 | Table 11 | Class Diagram | Instant Feedback Mechanism |
| FR-AE-002 | Table 12 | Class Diagram | Self-Evaluation Tools |
| FR-PA-001 | Table 13 | Class Diagram | Analytics Dashboard for Teachers |
| FR-PA-002 | Table 14 | Class Diagram | Predictive Performance Insights |
| FR-CG-01 | Table 15 | Class Diagram | AI-Assisted Lecture Creation |
| FR-CG-02 | Table 16 | Class Diagram | Manual Editing of AI-Generated Content |
| FR-CG-03 | Table 17 | Class Diagram | Save and Publish Options |

**Human interface design**

Describe the functionality of the system from the user’s perspective. Explain how the user will be able to use  your system to complete  all the  expected  features and  the  feedback  information that will be displayed for the user.

**Screen images**

A screenshot of a computer

Description automatically generated

Figure 7 LMS Dashboard

A screenshot of a computer

Description automatically generated

Figure 8 Enrolled Courses

**A screenshot of a computer

Description automatically generated**

Figure 9 Specific course screen

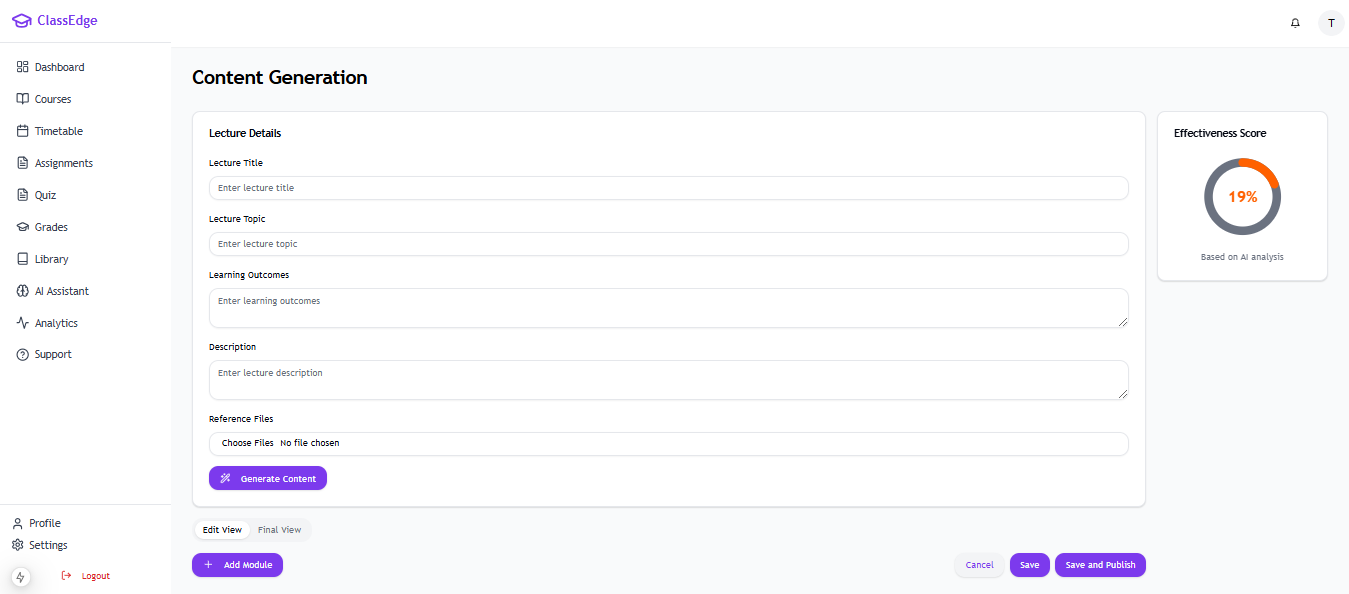
****

Figure 10 Content generation screen

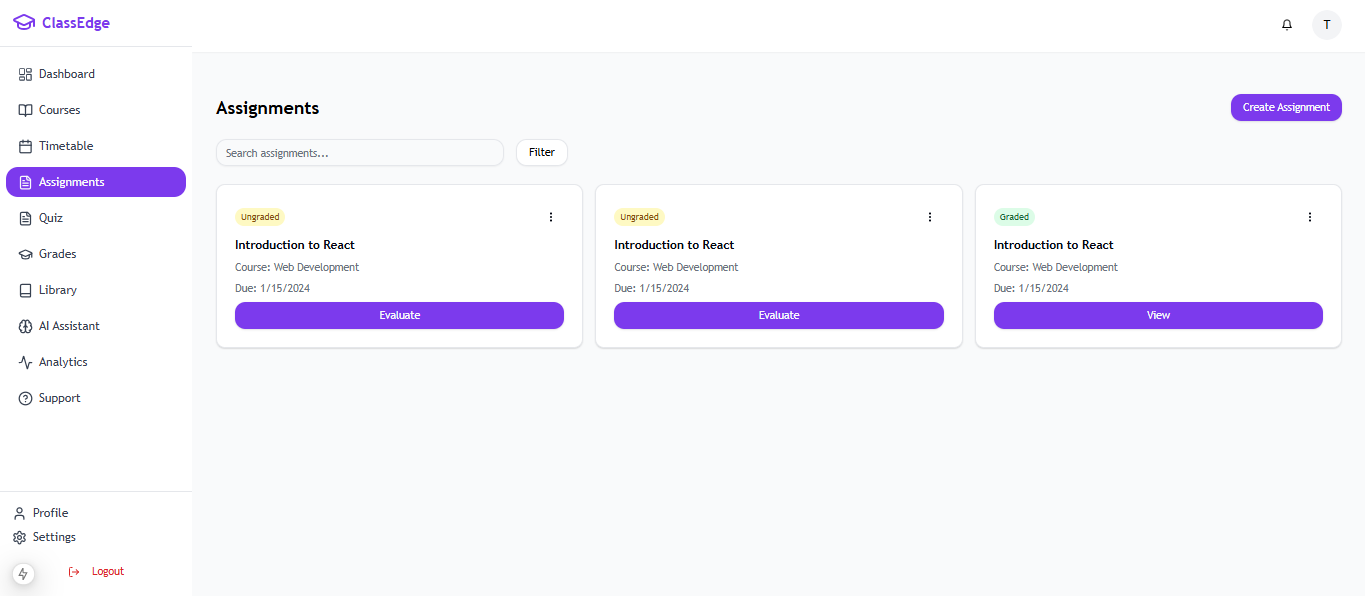
****

Figure 11 Assignment screen

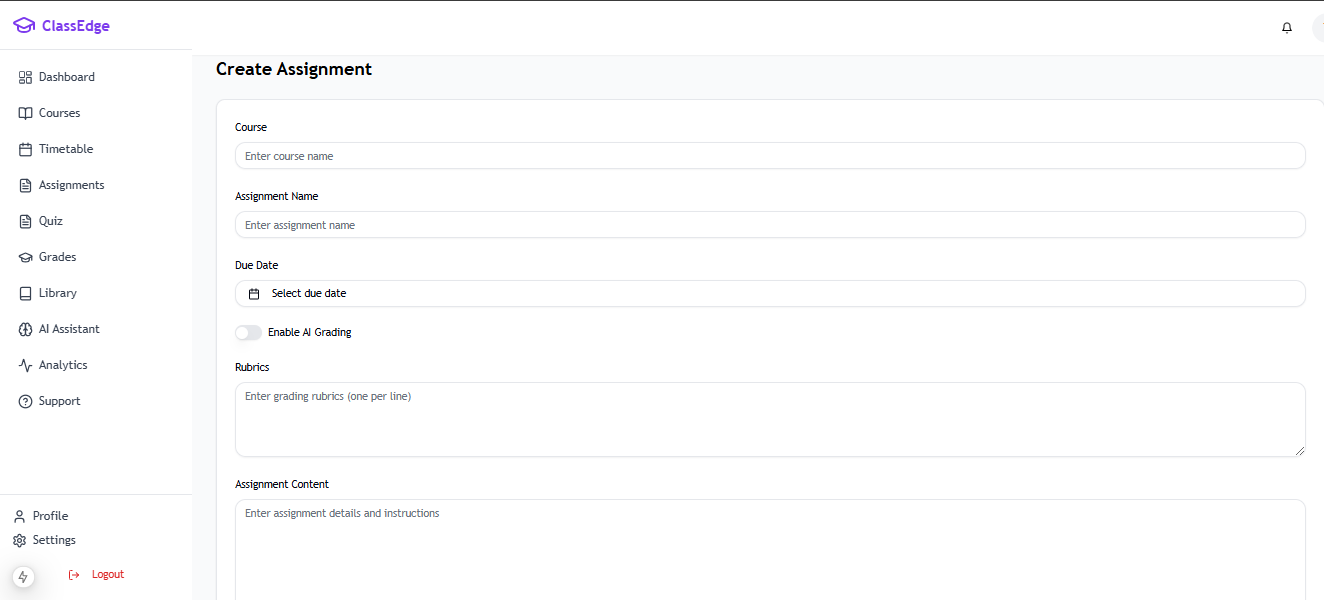
****

Figure 12 create Assignment screen

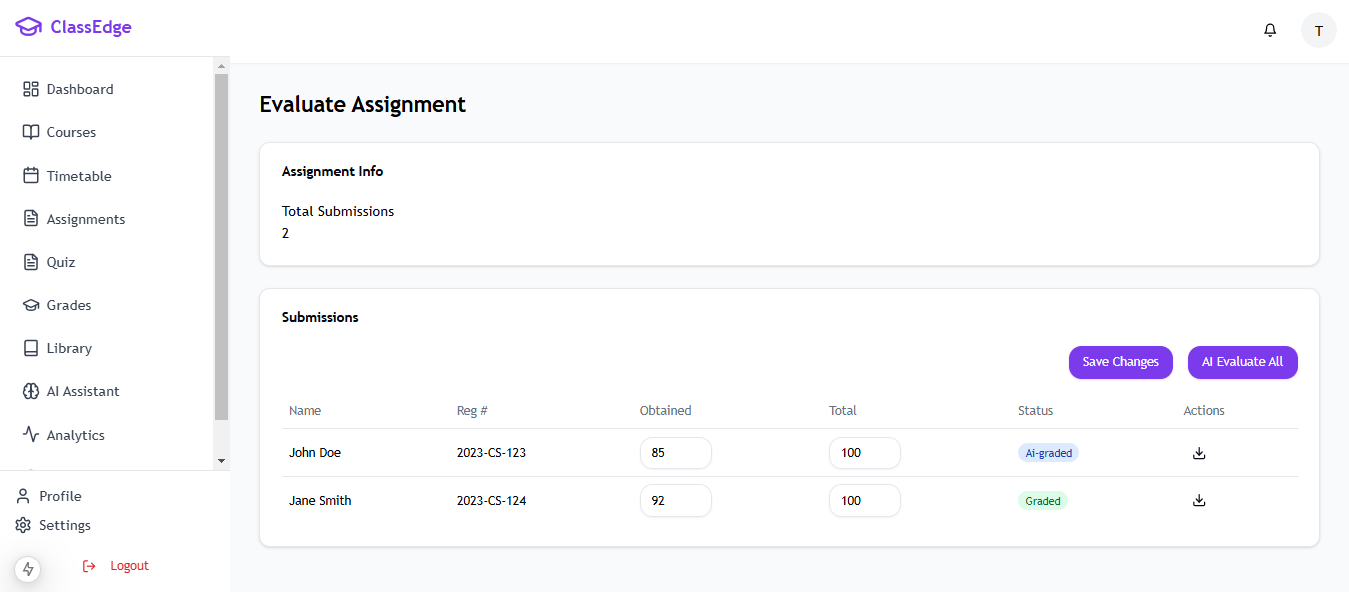
****

Figure 13 Evaluate assignment screen

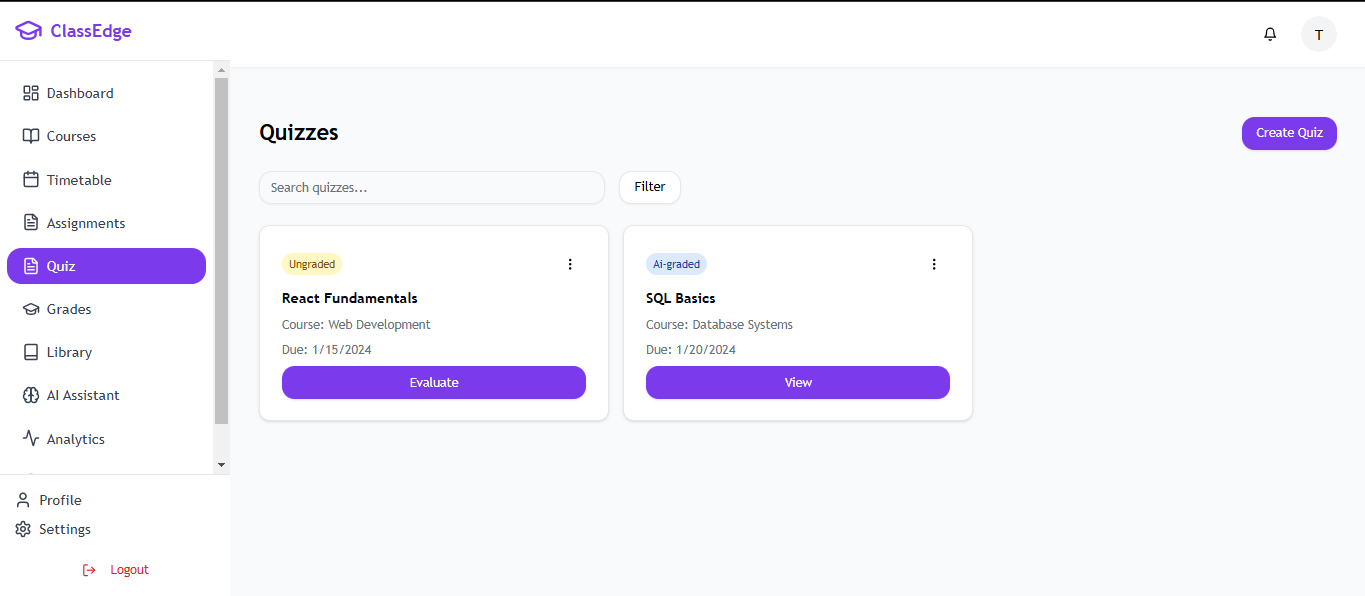
****

Figure 14 Quizzes screen

**A close-up of a notebook

Description automatically generated**

Figure 15 Create quiz screen

**A screenshot of a computer

Description automatically generated**

Figure 16 Quiz submission screen

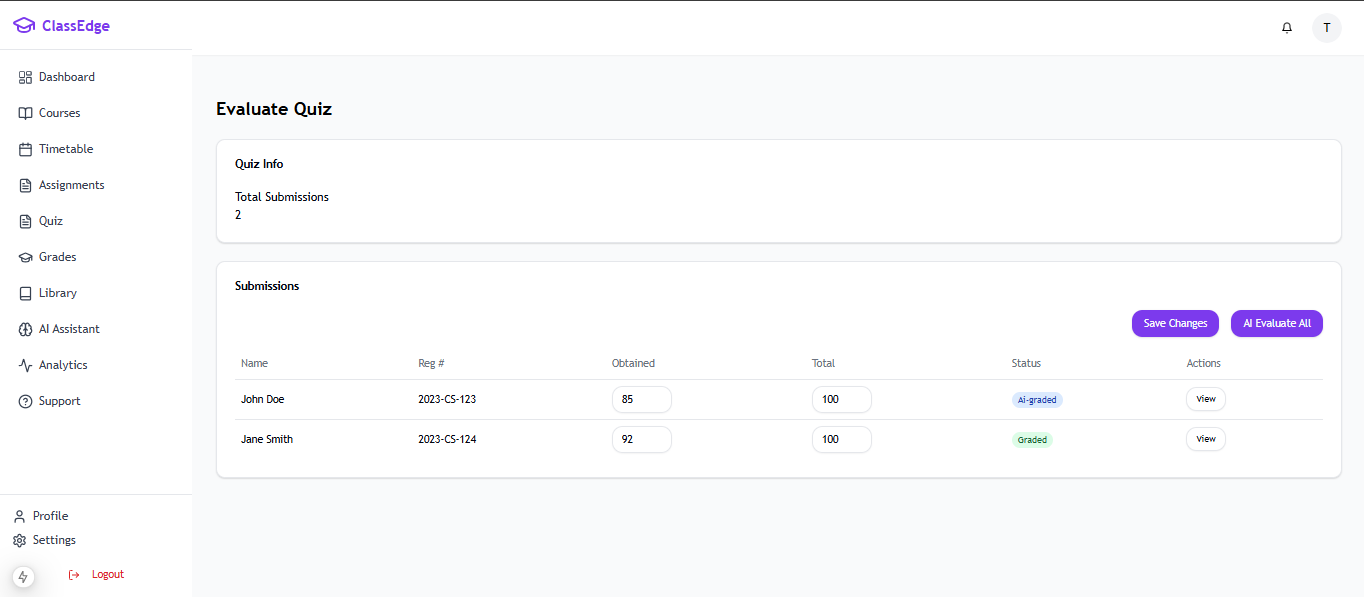
****

Figure 17 Evaluate quiz screen

**8.2 Screen objects and actions**

Here is a breakdown of the screen objects and their associated actions based on the images provided:

**1. Evaluate Assignment Screen**

**Screen Objects:**

* Assignment Info: Displays total submissions with static information.
* Submissions Table
* Columns: Name, Registration Number, Obtained Marks, Total Marks, Status

**Actions.**

* Manually input grades for assignments.
* Save manually graded changes.
* Trigger AI-based evaluation for all submissions.
* Download individual student submissions.

**2. Assignments Overview Screen**

**Screen Objects:**

* Search Bar and Filter Options: Allows filtering assignments.
* Assignment Cards: Displays the assignment title, course, due date, and grading status (Ungraded/Graded).
* Evaluate Button: To access the evaluation page for ungraded assignments.
* View Button: For graded assignments.

**Actions:**

* Filter assignments based on criteria.
* Evaluate or view individual assignments by clicking the respective button.

**3. Evaluate Quiz Screen**

**Screen Objects:**

* Quiz Info: Displays total submissions.
* Submissions Table:
* Columns: Name, Registration Number, Obtained Marks, Total Marks, Status

**Actions.**

* Fields: Editable fields for “Obtained Marks.”
* Status Tags: “AI-Graded” (blue) or “Graded” (green).
* View Icon: To view individual submissions.

**4. Quiz Submission Detail View**

**Screen Objects:**

* Question Details: Displays the question, student’s answer, correct answer, and marks.
* Marked Correct Checkbox: Checkbox to mark the student’s response as correct.
* Marks Field: Editable field for assigning marks.

**Actions:**

* Review and update individual question scores.
* Mark answers as correct or incorrect.
* Save changes to the marks.

**5. Content Generation Screen**

**Screen Objects:**

* Lecture Details Form: Fields for title, topic, learning outcomes, description, and reference files.
* Generate Content Button: To use AI for generating lecture content.
* Effectiveness Score: Displays a percentage based on AI analysis.

**Actions:**

* Input details for a lecture.
* Generate lecture content using AI.
* View effectiveness score based on AI evaluation.

**6. Create Assignment Screen**

**Screen Objects:**

* Assignment Form
* Fields for course name, assignment name, due date, rubrics, and assignment content.
* Enable AI Grading Toggle: To enable AI grading functionality.

**Actions:**

* Input assignment details.
* Toggle AI grading functionality.

**7. Course Overview Screen**

**Screen Objects:**

* Course Cards:
* Displays course title, instructor, completion percentage, and recent materials.
* Go to Course Button: To access course-specific details.

**Actions:**

* View course progress and materials.
* Access course-specific details.

**8. Create Quiz Screen**

**Screen Objects:**

* Quiz Form:
* Fields for course, quiz title, due date, and question types.
* Question builder for multiple-choice, fill-in-the-blanks, or short questions.
* Enable AI Grading Toggle: To allow AI to evaluate quizzes.

**Actions:**

* Input quiz details and questions.
* Enable AI grading for the quiz.

**9. Quiz Overview Screen**

**Screen Objects:**

* Quiz Cards:
* Displays quiz title, course, due date, and grading status (Ungraded/AI-Graded).
* Evaluate Button: To access the evaluation page for ungraded quizzes.
* View Button: For graded quizzes.

**Actions:**

* Evaluate or view quizzes.
* Filter quizzes using search options.

**10. My Courses Screen**

**Screen Objects:**

* Course Cards:
* Displays course title, instructor, progress percentage, and recent materials.
* Go to Course Button: To access course content.

**Actions:**

* View course progress and materials.
* Access specific course content.