Software Requirements Specification

for

<Learning Managment System>

Version 1.0

Prepared by <Noorain Eqbal, Dyaneshwar Hawal, Ratul Jha>

<Manipal Institute of Technology>

<23-03-2025>

Table of Contents

Team Members

Noorain Eqbal – 230953454
Dyaneshwar Hawal - 230953376
Ratul Jha - 230953338

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) document outlines the software requirements for the Learning Management System (LMS) application. This version of the SRS corresponds to release 1.0 and covers the core functionalities of the LMS, including user management, course administration, module and quiz creation, enrollments, and submissions. It provides detailed specifications for the backend schema and interactions that support both instructor and student functionalities. The document is intended to serve as a guide for developers, project managers, testers, and technical documentation writers throughout the development lifecycle.

1.2 Document Conventions

This document follows these conventions:

- **Bold text** is used for section headings.
- Italicized text is used for emphasis.
- **Code snippets** (if any) will be formatted in monospace.
- Each requirement is uniquely numbered for easy reference.
- Priorities:
  - **Must-have** (critical for system functionality).
  - **Should-have** (important but not essential).
  - **Nice-to-have** (enhancements or future scope).

1.3 Intended Audience and Reading Suggestions

This document is intended for:

- **Developers:** To understand the detailed data structures and relationships for backend implementation.
- **Project Managers:** To gauge scope and scheduling based on clearly defined functionalities.
- **Testers:** To design test cases that validate each aspect of the system, from user authentication to course and module management.
- **Documentation Writers:** To reference for creating user manuals and technical guides.

Reading Sequence:

1. Begin with the Overview and Purpose sections to understand the high-level goals of the LMS.
2. Proceed to the detailed schema descriptions and model relationships in the subsequent sections.
3. Refer to the Document Conventions to understand the formatting and priority of requirements.
4. Finally, review the References section for additional context and supporting documents.

1.4 Product Scope

The LMS application is designed to streamline and enhance the learning process for both instructors and students. Key functionalities include:

- **User Management:** Secure authentication and role-based access control for students and instructors.
- **Course Administration:** Creation and management of courses, including start and end date/time configurations.
- **Module and Quiz Management:** Instructors can upload modules and create quizzes, complete with metadata such as upload dates and maximum marks.
- **Enrollment Handling:** Students can enroll in courses, with additional features for tracking course status and completion percentages.
- **Submission Processing:** Students can submit quiz responses, with the system recording marks and managing server-side file paths.

This product supports academic institutions and training organizations by providing a robust, scalable platform that aligns with modern educational practices and business strategies. It integrates with existing corporate IT infrastructure to ensure a seamless user experience and operational efficiency.

1.5 References

- **Django Documentation:** Details on models, views, and the REST framework for API development. URL: https://docs.djangoproject.com
- **React Documentation:** Guidelines and best practices for building dynamic user interfaces. URL: https://reactjs.org/docs/getting-started.html
- IEEE 830-1998: Software Requirements Specification Standard.

2. Overall Description

2.1 Product Perspective

The LMS application is a new, self-contained product designed to streamline educational processes for academic institutions and training organizations. It is not a mere extension of an existing system; rather, it replaces legacy manual or semi-automated methods of course management and learning delivery. This product is part of a broader digital transformation initiative aimed at integrating educational processes into a centralized platform.

System Context:

- **External Systems:** The LMS may interface with university registration systems, external content repositories, and authentication services (such as Single Sign-On).
- **Internal Components:** It comprises user management, course and module management, quiz administration,

enrollment processing, and submission tracking.

A high-level system diagram might include:

- **User Interface Layer (ReactJS):** Provides the dynamic front-end experience.
- **API Layer (Django REST Framework):** Handles business logic and database interactions.
- **Database Layer:** Stores data according to the provided schema.
- **External Interfaces:** Connects to third-party services for authentication, notifications, and content storage.

2.2 Product Functions

The LMS application will provide the following major functions:

- **User Management:**
    - Secure login and authentication.
    - Role-based access control for instructors and students.
- **Course Administration:**
    - Creation and management of courses, including scheduling.
    - Course description and metadata handling.
- **Module Management:**
    - Upload and organization of course modules.
    - Tracking module creation details and instructor attribution.
- **Quiz and Assessment Management:**
    - Creation and scheduling of quizzes.
    - Handling quiz submissions, grading, and feedback.
- **Enrollment Processing:**
    - Student enrollment in courses.
    - Tracking course progress and completion percentages.
- **Reporting and Analytics:**
    - Generation of reports for instructors and administrators.

- Dashboard for course and quiz performance metrics.

A top-level data flow diagram would show user interactions with the front-end, data processing through the API, and persistent storage in the database.

2.3 User Classes and Characteristics

The primary user classes anticipated for this LMS include:

- **Students:**
  - **Characteristics:** Varying levels of technical expertise; primarily interact with the course content, modules, quizzes, and submissions.
  - **Usage:** Frequent use to access learning materials, enroll in courses, and complete assessments.
- **Instructors:**
  - **Characteristics:** Higher technical proficiency with an emphasis on content creation and management.
  - **Usage:** Use the system to create courses, modules, quizzes, and manage student enrollments and assessments.
- **Administrators:**
  - **Characteristics:** Oversee system configuration, user management, and overall data integrity.
  - **Usage:** Infrequent, yet critical interactions to ensure smooth operation and compliance with policies.
- **Support Staff:**
  - **Characteristics:** Provide technical and academic support to users.
  - **Usage:** Utilize system reporting and user data to assist with troubleshooting and guide users.

2.4 Operating Environment

The LMS application is designed to operate in a web-based environment with the following conditions:

- **Client-side:**
  - Modern web browsers (Chrome, Firefox, Edge, Safari) supporting HTML5, CSS3, and JavaScript ES6+.
- **Server-side:**
  - A web server hosting a Django-based backend on Linux or Windows environments.

    - Database support (e.g., PostgreSQL, MySQL) configured according to performance and scalability needs.

- **Development Environment:**

    - Django (Python) for server-side logic.

    - ReactJS for front-end development.

- **Third-party Services:**

    - External authentication and file storage services as needed.

2.5 Design and Implementation Constraints

Several constraints will guide the design and implementation:

- **Corporate and Regulatory Policies:**

    - Adherence to data protection regulations (e.g., GDPR) for user information.

    - Compliance with institutional IT policies regarding data security and privacy.

- **Technology Stack:**

    - The backend must be implemented using Django and Django REST Framework.

    - The front-end must be built using ReactJS.

- **Hardware Limitations:**

    - The system should be optimized for performance on standard hardware configurations common in educational institutions.

- **Interfacing Requirements:**

    - Must interface seamlessly with existing academic systems (e.g., registration systems, email servers).

- **Security Considerations:**

    - Secure file handling for uploads (modules and quiz submissions).

    - Role-based access control and proper authentication mechanisms.

- **Development Standards:**

    - Follow best practices for coding, testing, and documentation in both Django and ReactJS communities.

2.6 User Documentation

The following documentation will be delivered along with the LMS application:

➓ **User Manual:** Comprehensive guide for students, instructors, and administrators on using the system.

➓ **On-line Help:** Contextual help and FAQs embedded within the application.

➓ **Tutorials:** Step-by-step tutorials and video guides demonstrating common tasks (e.g., course creation, quiz setup).

➓ **API Documentation:** Detailed reference for developers interfacing with the LMS back-end APIs.

2.7 Assumptions and Dependencies

The following assumptions and dependencies are inherent to the project:

➓ **Third-Party Components:**

  ➓ The project assumes the availability of reliable third-party services for authentication and file storage.

➓ **Existing Infrastructure:**

  ➓ The institution will provide necessary hardware and network infrastructure to support the deployment.

➓ **Development Environment:**

  ➓ The development team is assumed to have expertise in Django and ReactJS.

➓ **External Dependencies:**

  ➓ Dependencies on external systems (e.g., registration systems, payment gateways) are assumed to be stable and accessible.

➓ **User Adoption:**

  ➓ It is assumed that users will have basic proficiency with web-based applications.

➓ **Regulatory Changes:**

  ➓ Any future changes in regulatory requirements might necessitate adjustments in data handling or user privacy features.

3. External Interface Requirements

3.1 User Interfaces

The LMS application will feature a responsive, web-based graphical user interface (GUI) designed for both desktop and mobile browsers. Key characteristics include:

- **Consistent Look and Feel:**
    - Follows a unified style guide that incorporates the institution’s branding.
    - Standardized headers, footers, navigation menus, and action buttons (e.g., “Submit”, “Cancel”, “Help”) across all pages.
- **Screen Layouts:**
    - **Dashboard:** Displays quick links to courses, upcoming assignments, and notifications.
    - **Course Pages:** Show course details, modules, and quizzes in a clearly structured format.
    - **User Profiles:** Present personal information, enrollment history, and progress metrics.
- **GUI Standards and Controls:**
    - Use of modern UI frameworks (e.g., Material-UI or Bootstrap) to ensure consistency.
    - Standard keyboard shortcuts and accessible design for improved usability.
    - Error messages and form validations follow consistent formatting (e.g., red highlights for errors, contextual help links).
- **User Interface Components:**
    - Login/registration forms.
    - Navigation menus.
    - Interactive forms for course creation, module uploads, and quiz submissions.
    - Data visualization components (charts, progress bars) for performance tracking.

Detailed designs and screen mockups will be documented in a separate User Interface Specification.

3.2 Hardware Interfaces

*T*he LMS is designed to be hardware agnostic, operating efficiently on commonly available computing devices. Key hardware interface considerations include:

- **Supported Device Types:**

    - Desktop and laptop computers (Windows, macOS, Linux).
    - Tablets and smartphones (iOS and Android).
- **Peripheral Integration:**
    - Standard keyboard and mouse input.
    - Touchscreen capabilities for mobile devices.
    - Printers for printing course materials and reports.
- **Data and Control Interactions:**
    - No direct hardware control; interactions are managed through the operating system's standard input/output channels.
- **Communication Protocols:**
    - Reliance on standard internet protocols (e.g., HTTP/HTTPS) for data exchange between devices and the server.

3.3 Software Interfaces

The LMS must seamlessly integrate with a variety of software components to ensure robust functionality. Interfaces include:

- **Backend and Frontend Integration:**
    - **Django REST Framework (Backend):**
        - Manages API endpoints for user authentication, course management, module and quiz administration, and enrollment processing.
        - Data exchange in JSON format.
    - **ReactJS (Frontend):**
        - Consumes APIs to dynamically render content.
        - Utilizes state management libraries (e.g., Redux) for managing user interactions and data flow.
- **Database Systems:**
    - **Relational Database:**
        - MySQL is used to store and manage data as per the defined schema (USER, STUDENT, INSTRUCTOR, ENROLLMENT, COURSE, MODULE, QUIZ, SUBMISSION, etc.).
        - Interactions are conducted via Django's ORM.

- **External Software Components:**
    - **Authentication Services:**
        - Uses JWT system.
    - **File Storage Services:**
        - Use of cloud storage (e.g., AWS S3) or local file servers for managing module and quiz submission files.
- **APIs and Data Sharing:**
    - Detailed API documentation will outline the expected request and response structures, authentication methods, and error handling.
    - Shared data items include user credentials, course identifiers, and enrollment statuses that are exchanged between different components.

3.4 Communications Interfaces

The LMS application includes several communications interfaces to support both internal and external messaging:

- **Web Communications:**
    - All client-server interactions use the HTTP/HTTPS protocols to ensure secure data transmission.
    - RESTful API endpoints are exposed for CRUD operations on system entities.
- **Email Communications:**
    - Automated emails for notifications such as enrollment confirmations, assignment deadlines, and password resets.
    - Supports integration with SMTP servers (e.g., Gmail SMTP, enterprise email systems).
- **Data Synchronization:**
    - Real-time data updates via WebSocket or polling mechanisms to ensure that users have the latest course information and notifications.
- **Security and Encryption:**
    - All communications use SSL/TLS encryption to protect sensitive user data.
    - Use of secure token-based authentication (e.g., JWT) to maintain session integrity.
- **Protocol Standards:**
    - Adherence to standard communication protocols, including REST for API communication and SMTP for email services.

- The system will also support JSON for data interchange, ensuring interoperability between different software components.

4. System Features

4.1 User Authentication and Management

4.1.1 Description and Priority

This feature provides secure user authentication, role-based access control, and profile management. It is **High priority** because it forms the foundation of system security and personalized user experiences.

4.1.2 Stimulus/Response Sequences

- **User Action:** A user enters login credentials on the login page.
  System Response: The system validates the credentials against the database and grants access to the appropriate user dashboard.

- ***User Action:*** A user requests a password reset.
  System Response: The system sends an email with a secure reset link after verifying the user's identity.

4.1.3 Functional Requirements

- **REQ-4.1.3.1:** The system shall validate user credentials against stored data in the USER table.

- **REQ-4.1.3.2:** The system shall support role-based access for students, instructors, and administrators.

- **REQ-4.1.3.3:** The system shall provide secure password reset functionality via email verification.

- **REQ-4.1.3.4:** The system shall log all authentication attempts and display appropriate error messages for invalid inputs.

4.2 Course Management

4.2.1 Description and Priority

This feature enables instructors and administrators to create, update, and delete courses, along with managing course details such as scheduling and description. It is **High priority** due to its central role in content delivery.

4.2.2 Stimulus/Response Sequences

- **User Action:** An instructor fills out a course creation form and submits it.
  System Response: The system validates the input, creates a new course record, and confirms course creation.

- **User Action:** An administrator modifies course details.
  System Response: The system updates the course record and reflects the changes immediately.

4.2.3 Functional Requirements

- **REQ-4.2.3.1:** The system shall allow instructors to create new courses by specifying title, description, start/end dates, and department.
- **REQ-4.2.3.2:** The system shall allow administrators to update or delete existing courses.
- **REQ-4.2.3.3:** The system shall validate that the course start date precedes the end date.
- **REQ-4.2.3.4:** The system shall display error messages when invalid course data is entered.

4.3 Module and Quiz Management

4.3.1 Description and Priority

This feature allows instructors to upload and manage course modules and quizzes. It includes handling file uploads, associating content with courses, and scheduling quizzes. It is **Medium priority** because it directly impacts content delivery and assessments.

4.3.2 Stimulus/Response Sequences

- **User Action:** An instructor uploads a module file and selects the corresponding course.
  System Response: The system stores the module file, records the metadata (including upload date and instructor ID), and associates it with the course.
- **User Action:** An instructor schedules a quiz by entering details such as timing and maximum marks.
  System Response: The system validates the quiz details and creates a new quiz record, linking it to the course.

4.3.3 Functional Requirements

- **REQ-4.3.3.1:** The system shall allow instructors to upload module files and link them to a specific course.
- **REQ-4.3.3.2:** The system shall record module metadata, including upload date and the instructor who created the module.
- **REQ-4.3.3.3:** The system shall allow instructors to create quizzes by providing quiz details, such as start/end dates and maximum marks.
- **REQ-4.3.3.4:** The system shall validate quiz scheduling to prevent conflicts with course timelines.

4.4 Enrollment and Progress Tracking

4.4.1 Description and Priority

This feature allows students to enroll in courses and provides mechanisms for tracking their progress and completion percentages. It is **High priority** as it directly impacts the learning experience and monitoring.

4.4.2 Stimulus/Response Sequences

- **User Action:** A student selects a course and requests enrollment.
  System Response: The system registers the enrollment in the ENROLLMENT table and updates the student's dashboard.

- **User Action:** A student submits a quiz or module assessment.
  System Response: The system records the submission, calculates marks, and updates the course progress status.

4.4.3 Functional Requirements

- **REQ-4.4.3.1:** The system shall allow students to enroll in courses through an intuitive enrollment interface.

- **REQ-4.4.3.2:** The system shall record enrollment data, including student ID, course ID, and initial status.

- **REQ-4.4.3.3:** The system shall track and display the progress and completion percentage for each enrolled course.

- **REQ-4.4.3.4:** The system shall provide error handling for duplicate enrollments by notifying the user if they are already enrolled in a course

5. Other Nonfunctional Requirements

5.1 Performance Requirements

Response Time:

- The system shall respond to any user action (e.g., login, course enrollment, module upload) within 2 seconds under normal operating conditions.

- The system shall be capable of handling concurrent access by up to 500 users without noticeable performance degradation.

Scalability:

- The backend architecture must support horizontal scaling to accommodate increasing user loads, ensuring that performance remains consistent during peak usage times.

Data Processing:

- API endpoints shall process and return data (such as course details and enrollment information) within 3 seconds, even when handling complex queries or large datasets.

These performance metrics are specified to ensure a responsive and efficient user experience, particularly during critical operations such as enrollment and quiz submissions.

5.2 Safety Requirements

Data Integrity and Recovery:

- ❿ The system must implement automated backups of all critical data (user profiles, course data, enrollment records) at regular intervals.
- ❿ In the event of a system failure, a recovery procedure must restore data to the most recent backup with a maximum data loss window of 5 minutes.
- ❿ **Error Handling:**
- ❿ The system shall include safeguards to prevent data corruption during file uploads or transactional operations.
- ❿ Clear, non-technical error messages must be displayed to users to prevent misuse and reduce operational errors.
- ❿ **Compliance:**
- ❿ The system must comply with all relevant safety and regulatory standards, including institutional guidelines for digital learning environments.

5.3 Security Requirements

User Authentication and Authorization:

- ❿ *The system shall enforce secure login procedures using encrypted passwords (e.g., bcrypt or similar).*
- ❿ *Role-based access control (RBAC) must be implemented to restrict access to sensitive functions (e.g., course**Data Protection:***
- ❿ *All sensitive data transmitted between the client and server shall be secured using SSL/TLS encryption.*
- ❿ *Stored sensitive information (e.g., passwords, personal details) shall be encrypted in the database.*
- ❿ ***Vulnerability Management:***
- ❿ *The system shall be regularly scanned for vulnerabilities, and security patches must be applied promptly.*
- ❿ *Any third-party services (e.g., external file storage) must meet defined security standards and be reviewed periodically.*

5.4 Software Quality Attributes

Adaptability:

- ❿ *The system should be designed using modular components that allow for easy addition of new features or integration with other systems.*

Availability:

- ❿ *The LMS shall guarantee an uptime of 99.5% during operational hours through redundancy and fault-tolerant designs.*

Maintainability:

- ❿ *Code and documentation must adhere to industry standards to facilitate ease of maintenance and future enhancements.*

Usability:

- ❿ *The user interface must be intuitive, ensuring that users with minimal technical expertise can navigate the management* **system effective***ly.*
- ❿ *User experience (UX) testing will be performed to verify that the system meets accessibility and usability standards.*

Interoperability:

- ❿ *The system must seamlessly integrate with external systems (e.g., authentication services, content repositories) through well-documented APIs and standard protocols.*

5.5 Business Rules

Role-Specific Access:

- ❿ *Only users with instructor or administrator roles shall have the authority to create, update, or delete course and module content.*
- ❿ *Students are allowed to view and enroll in courses but cannot alter course content.*

Enrollment Management:

- ❿ *A student may enroll in a course only once; duplicate enrollment attempts must trigger an appropriate error notification.*

Assessment Submission:

- ❿ *Quiz and module submissions must be timestamped; late submissions are subject to predefined rules (e.g., grace periods, penalties) as defined by the academic policy.*

Data Retention:

- ❿ *Records such as course enrollments and submission histories shall be retained in accordance with institutional data retention policies.*

6. Other Requirements

Database Requirements:

- The system shall use a relational database (e.g., PostgreSQL or MySQL) to store all user, course, module, quiz, and enrollment data.
- The database must support ACID properties and allow for efficient indexing to ensure rapid retrieval of large datasets.
- Data integrity constraints (such as foreign key relationships) must be strictly enforced as defined in the schema.

Internationalization Requirements:

- The application shall support multiple languages for the user interface, enabling dynamic language selection based on user preferences.
- Date and time formats, numerical displays, and currency (if applicable) shall be adaptable to the locale settings of the user.

Legal Requirements:

- The LMS must comply with all applicable educational data privacy regulations (e.g., FERPA, GDPR) to protect user information.
- The system must include mechanisms to handle user consent and data deletion requests in accordance with privacy policies.

Reuse Objectives:

- The project aims to develop reusable components, particularly in user authentication, file handling, and API services, to be leveraged in future educational applications.
- Documentation and code modules shall follow industry standards to promote reuse and facilitate maintenance.

Additional Considerations:

- The system shall provide logging and audit trails for critical operations (e.g., user authentication, course modifications) to support troubleshooting and compliance audits.
- The application must be designed with scalability in mind, ensuring that components can be updated or replaced with minimal impact on the overall system.

Appendix A: Glossary

- **LMS:** Learning Management System – a platform for delivering educational courses and training programs.
- **SRS:** Software Requirements Specification – a document that describes the software system's functions, features, and constraints.

- **API:** Application Programming Interface – a set of protocols for building and interacting with software applications.
- **ACID:** Atomicity, Consistency, Isolation, Durability – principles that ensure reliable processing of database transactions.
- **SSO:** Single Sign-On – an authentication process that allows a user to access multiple applications with one set of credentials.
- **RBAC:** Role-Based Access Control – an approach to restricting system access to authorized users.
- **JSON:** JavaScript Object Notation – a lightweight data-interchange format used in API communication.
- **TBD:** To Be Determined – placeholder used for requirements that are not fully defined at the time of writing.
- **ORM:** Object-Relational Mapping – a programming technique for converting data between incompatible type systems in object-oriented programming languages.

Appendix B: Analysis Models

High-Level Architecture Diagram:

- ***User Interface Layer (ReactJS):*** *Manages presentation and client-side interactions.*
- ***API Layer (Django REST Framework):*** *Processes business logic and communicates with the database.*
- ***Database Layer (PostgreSQL/MySQL):*** *Persists application data according to the defined schema.*
- ***External Interfaces:*** *Integrations for authentication (SSO, LDAP), file storage (cloud services), and email services (SMTP).*

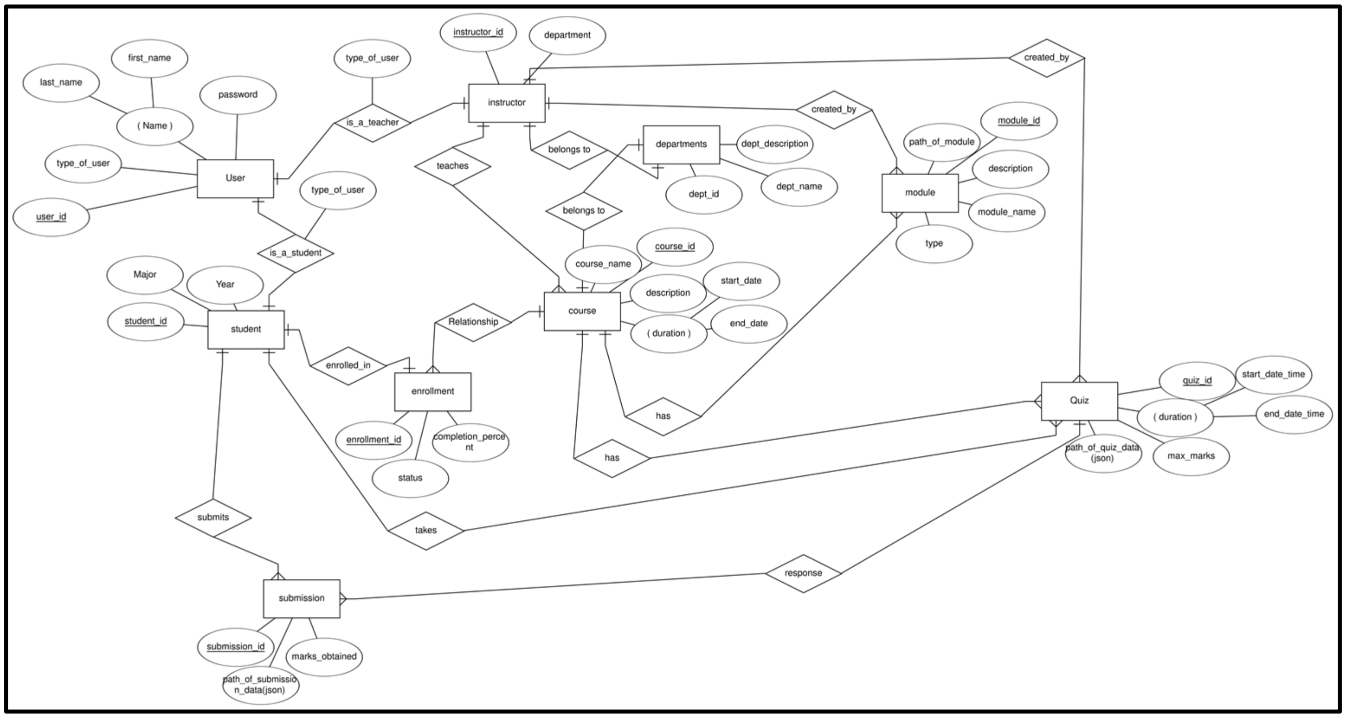
Entity-Relationship Diagram (ERD):

Appendix C: To Be Determined List

TBD-1: Specific third-party authentication and file storage service details.

- **TBD-2:** Detailed user interface mockups and interaction diagrams.
- **TBD-3:** Performance benchmarks and load testing results.
- **TBD-4:** Finalized API documentation including endpoint parameters and response formats.
- **TBD-5:** Regulatory compliance details and da*ta retention policies specific to target jurisdictions.*
- ***TBD-6:*** *Localization and internationalization support specifics (languages, formats).*

ER DIAGRAM



first_name
last_name
password
(Name)
type_of_user
User
user_id
type_of_user
is_a_teacher
instructor_id
department
instructor
created_by
created_by
departments
dept_description
dept_name
dept_id
belongs to
belongs to
teaches
type_of_user
is_a_student
module
path_of_module
module_id
description
module_name
type
Major
Year
student_id
student
course_name
course_id
course
description
start_date
(duration)
end_date
Relationship
enrolled_in
enrollment
enrollment_id
completion_percent
status
has
has
Quiz
quiz_id
start_date_time
(duration)
end_date_time
path_of_quiz_data (json)
max_marks
submits
takes
response
submission
submission_id
marks_obtained
path_of_submission_data(json)

SCHEMA

1. USER(user_id, password, first_name, last_name, type_of_user)

2. STUDENT(student_id, user_id ,major, year)

3. INSTRUCTOR(instructor_id, user_id, department_id)

4. ENROLLMENT(student_id, course_id, status, completion_percentage)

5. COURSE(course_id, dept_id, desc, start_date_time, end_date_time)

6. MODULE(module_id, course_id, name, type, path_of_module)

7. MODULE_CREATED_BY(instructor_id, module_id, upload_date)

8. QUIZ(quiz_id, course_id, start_date_time, end_date_time, max_marks, path_on_server)

9. QUIZ_CREATED_BY(instructor_id, quiz_id)

10. SUBMISSION(submission_id, student_id, quiz_id, marks_obtained, path_on_server)

11. TEACHES(instructor_id, course_id)

Normalization

USER(user_id, password, first_name, last_name, type_of_user)

- **Primary Key:** user_id
- **Analysis:** Every attribute is atomic, and all non-key attributes (password, first_name, last_name, type_of_user) depend solely on user_id. There are no partial or transitive dependencies.
- **Conclusion:** In 1NF, 2NF, 3NF, and BCNF.

STUDENT(student_id, user_id, major, year)

- ***Primary Key:*** *student_id*
- ***Analysis:*** *All attributes (user_id, major, year) are fully functionally dependent on student_id. The user_id is a foreign key linking to USER, and its attributes are stored in that table.*
- **Conclusion:** In 1NF, 2NF, 3NF, and BCNF.

INSTRUCTOR(instructor_id, user_id, department_id)

- ***Primary Key:*** *instructor_id*
- ***Analysis:*** *The non-key attributes (user_id, department_id) depend entirely on instructor_id.*
- **Conclusion:** In 1NF, 2NF, 3NF, and BCNF.

ENROLLMENT(student_id, course_id, status, completion_percentage)

- ***Primary Key:*** *A composite key (student_id, course_id)*
- ***Analysis:*** *The attributes status and completion_percentage depend on the full combination of (student_id, course_id). No attribute depends on only one part of the composite key.*
- **Conclusion:** In 1NF, 2NF, 3NF, and BCNF.

COURSE(course_id, dept_id, desc, start_date_time, end_date_time)

- ***Primary Key:*** *course_id*
- ***Analysis:*** *All other attributes depend solely on course_id.*
- **Conclusion:** In 1NF, 2NF, 3NF, and BCNF.

MODULE(module_id, course_id, name, type, path_of_module)

- ***Primary Key:*** *module_id*
- ***Analysis:*** *The attributes (course_id, name, type, path_of_module) depend only on module_id.*
- **Conclusion:** In 1NF, 2NF, 3NF, and BCNF.

MODULE_CREATED_BY(instructor_id, module_id, upload_date)

- ***Primary Key:*** *Likely a composite key (instructor_id, module_id)*
- ***Analysis:*** *Assuming that a module can be created by one or more instructors, the combination of instructor_id and module_id is the key and upload_date depends on that combination. If, however, business rules state that a module is created by only one instructor, then module_id could determine instructor_id and upload_date—but the given design supports the composite key scenario.*
- **Conclusion:** In 1NF, 2NF, 3NF, and BCNF.

QUIZ(quiz_id, course_id, start_date_time, end_date_time, max_marks, path_on_server)

- **Primary Key:** quiz_id
- **Analysis:** All other attributes are fully functionally dependent on quiz_id.
- **Conclusion:** In 1NF, 2NF, 3NF, and BCNF.

QUIZ_CREATED_BY(instructor_id, quiz_id)

- **Primary Key:** Likely a composite key (instructor_id, quiz_id)

- **Analysis:** This table maps which instructor created which quiz. If every quiz is created by exactly one instructor, a functional dependency may exist (quiz_id → instructor_id); however, the composite design covers both one-to-many and many-to-many scenarios.
- **Conclusion:** In 1NF, 2NF, 3NF, and BCNF.

SUBMISSION(submission_id, student_id, quiz_id, marks_obtained, path_on_server)

- ***Primary Key:*** submission_id
- ***Analysis:*** All non-key attributes depend entirely on submission_id.
- **Conclusion:** In 1NF, 2NF, 3NF, and BCNF.

TEACHES(instructor_id, course_id)

• **Primary Key**: Composite key (instructor_id, course_id)
• **Analysis**: This relation serves as a many-to-many mapping between instructors and courses with no additional non-key attributes.
• **Conclusion**: In 1NF, 2NF, 3NF, and BCNF.

Each relation in the schema:
• Uses atomic values (satisfies 1NF).
• Has non-key attributes fully functionally dependent on the entire primary key (satisfies 2NF).
• Has no transitive dependencies among non-key attributes (satisfies 3NF).
• Contains only candidate keys as determinants (satisfies BCNF).
Thus, the schema follows all the normal forms up to BCNF.

Therefore the final schema will be the same as derived above schema