Software Requirements Specification

for

Community Learning Management System

Version 1.0 approved

Prepared by Muhammad Touseef,Ali Bilal, Huzaifa Ayyaz

FAST NUCES, Karachi

December 8, 2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

This report is generated as part of the "Software Design And Analysis - CS324" course at FAST NUCES, and is related to the SRS and SDS report as accompanied by the semester project.
The Software Requirement Specification (SRS) is meant to serve as a guide that details the exact nature, software, hardware, and the business requirements that lead to the development of a software project, in this case, this is for Terrabuzz. The guide provides an overview of the thought process, the nature of development, the workflow, the main conceiving ideas and driving force and list of business fulfilling points that the product tried to fill in the gaps for.
The Software Design Specification (SDS) entails the system architecture and the system design kept it mind while developing the product. How different components are interleaved with one another, and what dependencies exist, if any.

1.2 Intended Audience and Reading Suggestions

This document is for anyone who would like to know about the decisions the developers took from a design and analysis perspective, and for future engineers who wish to understand the nature of the project, and how it became what it did, with emphasis on what ideas and what features. This report will talk about those lower level details, as well as the higher business level use cases that are part of the product.

1.3 Product Scope

The **Community Learning Management System (CLMS)** is a web-based application designed to integrate course delivery with community engagement. It addresses the gap in traditional learning platforms by combining learning management features with collaborative tools like forums, event calendars, and real-time discussions. CLMS enables instructors to focus on content delivery while fostering peer interaction and community learning.

Purpose, Benefits, Objectives, and Goals:

- **Purpose:** To create a unified platform that enhances online learning through community collaboration.
- **Benefits:** Encourages active participation, seamless communication, and an interactive learning experience.
- **Objectives:** Deliver user-friendly tools for course management, community building, and event planning.
- **Goals:** Support corporate strategies by promoting a connected, engaging, and scalable learning ecosystem.

This aligns with modern education needs, empowering users with robust learning and networking opportunities within a single platform.

2. Overall Description

2.1 Product Perspective

The **Community Learning Management System (CLMS)** is a newly developed, standalone platform designed to address the limitations of existing Learning Management Systems (LMS). Unlike platforms such as Udemy or Coursera, which focus primarily on course delivery, CLMS integrates community-building features to provide a holistic learning environment. This unique combination eliminates the need for external tools like Discord or Reddit for community interaction, offering a unified space for learning and collaboration.

Context and Origin:
CLMS originates from the need to bridge the gap between isolated course learning and community engagement. It is not an extension of any existing system but a self-contained product specifically tailored to foster collaboration, interaction, and knowledge sharing among learners and instructors.

System Integration:
The system comprises three major components:

1. **Frontend (React.js):** User interface for interacting with courses, communities, and events.
2. **Backend (Node.js with Express.js):** Business logic and APIs for user authentication, course management, and event handling.
3. **Database (MySQL):** Relational data storage for users, courses, community posts, and events.

External Interfaces:

- **User Devices:** Access via web browsers on desktops, tablets, and smartphones.
- **Third-party APIs:** Integration for optional services like youtube videos integration and calendar component for user friendly navigation to community events.

The system's modular architecture ensures flexibility for future enhancements and easy scalability to meet growing user demands.

2.2 Product Functions

1. **User Authentication:** Secure account creation, login, and logout using JWT.
2. **Profile Management:** Users can update personal details, including names, passwords, and avatars.
3. **Public/Private Communities:** Creation of both public and private communities to foster group interactions.
4. **Role Management:** Admin and moderator roles for community management.
5. **Community Administration:** Approve/decline membership requests for private communities.
6. **Community Visibility:** View details of community members, including their roles.
7. **Post Management:** Create, view, like, comment, and delete posts within communities.
8. **Comment Replies:** Users can reply to comments for threaded discussions.
9. **Tagging System:** Community-specific tags for organizing and categorizing posts.

10. **Event Creation:** Admins and moderators can create events visible to community members.
11. **Community Calendar:** Events displayed in a calendar with date-based filtering.
12. **Course Hosting:** Instructors can create and manage courses within communities.
13. **Video-Based Learning:** Watch and navigate course videos with "Next" and "Previous" controls.
14. **Discussions and feeds:** Integrated forums to discuss course topics and community matters.
15. **Search Functionality:** Search for communities and filter posts based on tags.
16. **Responsive Design:** Optimized user experience across devices, including desktops, tablets, and smartphones.
17. **Access Control:** Role-based access to features like course creation and event management.

2.3 User Classes and Characteristics

The **Community Learning Management System (CLMS)** is designed for several distinct user classes, each with specific roles, privileges, and responsibilities. The following describes the primary user classes and their characteristics:

1. **Administrators**
    - **Frequency of Use:** High; frequently manage and monitor the platform's overall functionality.
    - **Functions Used:** Full access to all system features including community creation, role assignment, course management, event creation, and system configurations.
    - **Technical Expertise:** High; expected to have strong technical and administrative skills to handle user and system management.
    - **Privileges:** Can create and manage communities, approve/decline membership requests, assign roles, and manage user profiles. They have control over all aspects of the platform.
    - **Educational Level/Experience:** Typically experienced in system management or education administration.
2. **Moderators**
    - **Frequency of Use:** Medium to High; moderate users, primarily engaged with managing communities and content.
    - **Functions Used:** Community and post management, moderation of discussions, event creation, and member role adjustments.
    - **Technical Expertise:** Medium; capable of managing day-to-day operations and moderating user-generated content without needing deep technical knowledge.
    - **Privileges:** Can approve posts, remove inappropriate content, manage community members, and create events. They cannot create new communities.
    - **Educational Level/Experience:** Usually experienced in moderating online communities or forums.
3. **Instructors**

- **Frequency of Use:** Medium to High; actively involved in course creation, management, and student interaction.
- **Functions Used:** Course creation and management, video content delivery, and interaction with students via posts and discussions.
- **Technical Expertise:** Medium; familiarity with educational tools and course management systems.
- **Privileges:** Can create and manage courses, view student progress, and engage with students through posts and discussions.
- **Educational Level/Experience:** Experienced educators or industry professionals responsible for course creation and instruction.

4. **Students**

   - **Frequency of Use:** High; regularly use the platform to engage in learning activities.
   - **Functions Used:** Access and participate in courses, engage in discussions, attend events, interact with instructors and peers, and track their progress.
   - **Technical Expertise:** Low to Medium; basic understanding of web-based applications.
   - **Privileges:** Can access and interact with courses, participate in discussions, view posts, and RSVP for events. They have limited interaction with system administration.
   - **Educational Level/Experience:** Varies; includes learners of all levels who are engaging in the learning content offered by instructors.

5. **Guests**

   - **Frequency of Use:** Low; typically for browsing and initial exploration of the platform.
   - **Functions Used:** Limited to viewing available courses and community descriptions.
   - **Technical Expertise:** Low; no significant technical skills required.
   - **Privileges:** Can view community and course overviews, but cannot interact, post, or engage in events without registering as a full user.
   - **Educational Level/Experience:** No particular requirement.

Most Important User Classes:

- **Administrators, Moderators, and Instructors** are the key user classes whose needs directly impact the platform's functionality and success. These users control and manage the platform's content, security, and educational features, making them the primary focus of system features and access controls.

Less Important User Class:

- **Guests** have limited access and are considered a secondary class, with fewer specific needs beyond basic access to platform overviews.

2.4 Operating Environment

The **Community Learning Management System (CLMS)** will operate within a web-based environment, leveraging modern technologies to deliver a seamless user experience. The following details outline the hardware, software, and system environment where CLMS will function:

1. Hardware Platform

- **Client Side:**
  - **Devices:** Desktops, laptops, tablets, and smartphones (iOS, Android).
  - **Browser Support:** Chrome, Firefox, Safari, Microsoft Edge (latest stable versions).
  - **Screen Resolution:** Optimized for responsive design, supporting screens as small as 320px wide (mobile devices) to large desktop monitors.
- **Server Side:**
  - **Servers:** Cloud-based or dedicated servers (AWS, Azure, DigitalOcean, etc.) with sufficient computational power to handle concurrent users and large volumes of data.
  - **Database Server:** MySQL-compatible relational database management system (RDS for cloud hosting or dedicated MySQL server).

2. Operating System

- **Client Side:**
  - Web browsers are supported on all major operating systems, including:
    - **Windows** (Windows 10 and newer).
    - **Mac OS** (MacOS Mojave and newer).
    - **Linux** distributions (Ubuntu, Fedora, etc.).
    - **Mobile OS:** iOS (iOS 12 and newer) and Android (version 8 and newer).
- **Server Side:**
  - **Backend Server OS:** Linux (Ubuntu, CentOS, Debian) for running the Node.js and Express.js backend.
  - **Database OS:** Compatible with Linux-based distributions for MySQL deployment, though Windows and macOS can be used for local development environments.

3. Software Components

- **Frontend:**
  - **Framework:** React.js for building responsive, dynamic user interfaces.
  - **Styling:** CSS, Bootstrap, or Material-UI for consistent styling across devices.
  - **JavaScript:** ES6+ for modern JavaScript syntax.
- **Backend:**
  - **Runtime:** Node.js (LTS version) for executing JavaScript server-side.
  - **Framework:** Express.js for creating RESTful APIs and managing server routes.
  - **Authentication:** JWT (JSON Web Tokens) for secure user authentication.
- **Database:**
  - **Database Management System (DBMS):** MySQL (relational database) for storing and managing user data, courses, community posts, events, and other platform-related data.
  - **ORM:** MySQL2 npm package(for database interaction via JavaScript).

2.5 Design and Implementation Constraints

The development and implementation of the **Community Learning Management System (CLMS)** are subject to several constraints that will influence the choice of technologies, design decisions, and overall architecture. These constraints ensure that the software aligns with organizational goals, adheres to technical standards, and operates effectively in the target environment. Below are the key constraints:

1. Corporate and Regulatory Policies

- **Data Privacy and Security:**
  - CLMS must comply with global data privacy regulations, such as the **General Data Protection Regulation (GDPR)** (for users in the EU), and the **California Consumer Privacy Act (CCPA)** (for users in California).
  - The system must implement stringent data encryption and secure user authentication protocols, particularly for personal information and payment data.
- **Accessibility Standards:**
  - The platform must meet **WCAG 2.1** (Web Content Accessibility Guidelines) for web accessibility, ensuring usability for users with disabilities.

2. Hardware Limitations

- **Server and Client Hardware Requirements:**
  - The system must be able to scale efficiently, with considerations for both cloud-based servers (AWS, Azure) and on-premise deployments. The server hardware must meet performance requirements to handle an increasing number of concurrent users, with adequate memory, CPU resources, and network throughput.
- **Mobile Devices:**
  - The application must be optimized for a wide range of mobile devices, considering limitations in processing power, memory, and network speed on lower-end devices.

3. Technologies and Tools

- **Frontend Framework:**
  - **React.js** is mandated for the frontend development, which may limit the use of other frontend frameworks (e.g., Angular, Vue.js).
- **Backend Technology Stack:**
  - **Node.js (Express.js)** must be used for backend development, which may constrain the use of other backend technologies such as Django or Ruby on Rails.
- **Database:**

- **MySQL** must be used as the relational database, limiting the use of NoSQL databases like MongoDB or PostgreSQL unless specifically required by new features.

4. Parallel Operations

- **Concurrency Management:**
  - The system must be designed to handle multiple simultaneous users performing various actions (e.g., course enrollment, event registrations, video streaming). Proper concurrency handling will be required to ensure smooth operation, especially during peak traffic times.

5. Security Considerations

- **Data Encryption:**
  - The system must encrypt sensitive data both in transit (using **TLS/SSL**) and at rest (using **AES-256 encryption** or similar standards).
- **User Authentication:**
  - Use of **JWT (JSON Web Tokens)** for user authentication is mandatory, with secure token generation, storage, and expiration policies in place.
- **Authorization and Role Management:**
  - Strict role-based access control (RBAC) must be enforced, ensuring that only authorized users (e.g., admins, instructors) can perform high-privilege actions like community creation or course management.

6. Design Conventions and Programming Standards

- **Coding Standards:**
  - The development team must follow industry-standard coding practices (e.g., naming conventions, modular design, and code commenting) to ensure maintainability and readability.
- **Version Control:**
  - Use of **Git** for version control, following a defined branching strategy (e.g., Git Flow) for collaborative development.
- **Documentation:**
  - Detailed technical documentation must be provided, including API documentation and system architecture diagrams, for ease of maintenance and future development.

7. Interfaces with Other Applications

- **Third-Party Integrations:**
  - The system may need to interface with external systems such as payment gateways (e.g., **Stripe, PayPal**) and email services (e.g., **SendGrid**, **Mailgun**) for user notifications and payment processing.
- **Video Hosting Services:**

- If external video hosting platforms (e.g., **YouTube**, **Vimeo**) are integrated, the system will be constrained by the limitations and API requirements of these platforms.

8. Language Requirements

- **Internationalization and Localization:**
  - The system must be prepared to support multiple languages, starting with English and potentially expanding to others (e.g., Spanish, French, German). This will require the use of localization libraries or frameworks and structured support for dynamic content translation.

9. Communications Protocols

- **HTTP/HTTPS:**
  - The system will rely on standard HTTP/HTTPS protocols for communication between the frontend and backend.
- **WebSocket (Optional):**
  - If real-time communication (e.g., for live discussions or notifications) is needed, **WebSocket** protocol may be used to facilitate bidirectional communication between the client and server.

2.6 User Documentation

At the moment, there is no such thing, but we can plan on delivering a tutorial on how to use the platform for new users unfamiliar with the platform to get them started quickly.

2.7 Assumptions and Dependencies

The list of assumptions at the moment are,

- A module for verifying the email address exists, but we did not manage the time to integrate this into the actual flow, so we're assuming that the email entered and given to us actually belongs to a user
- If external video hosting services (e.g., YouTube, Vimeo) are integrated for hosting course videos, the platform depends on their API stability and performance. Any API changes or service disruptions could affect video delivery.

The list of dependencies is split into two aspects. We then have a client side of dependencies, and a server side of dependencies.

- **Server Level**
  - Bcrypt - Need to hash, decry-pt, and encrypt passwords into the database
  - Cookie Parser - For parsing cookies on the client level
  - CORS - To prevent cross origin JavaScript files running on the Terrabuzz domain
  - Dotenv - For incorporating the required security keys into the files
  - Express - JS framework for dealing with NodeJS on a higher level with more functionalities

- **Client Level**

- ReactJS - The frontend library we decided to go with for this project.

3. External Interface Requirements

3.1 User Interfaces

The **Community Learning Management System (CLMS)** user interface will be designed to be intuitive, accessible, and responsive, providing a seamless experience across all devices. Below are the key logical characteristics and components of the interface:

1. General User Interface (UI) Design Guidelines

- **Consistency:**
  - Consistent design elements (buttons, icons, typography) will be used throughout the platform to maintain a unified look and feel.
- **Responsiveness:**
  - The layout will adapt dynamically to different screen sizes, ensuring compatibility with desktops, tablets, and smartphones.
- **Minimalism:**
  - The interface will emphasize simplicity and clarity, avoiding cluttered layouts and providing a focused user experience.

2. Standard User Interface Components

- **Header and Navigation Bar:**
  - Each screen will feature a consistent header with links to the home page, user profile, notifications, and settings.
  - A horizontal or vertical navigation menu will be present to access key sections: Courses, Communities, Events, and Discussions.
- **Standard Buttons and Actions:**
  - Common buttons such as **Save**, **Cancel**, **Edit**, **Delete**, and **Back** will be displayed consistently across forms and pages.
  - Action buttons (e.g., **Join Community**, **Cancel join request**, **Post Comment**) will be prominently displayed with clear labels.
- **Error Message Display:**
  - Errors will be shown in a clear, non-intrusive manner using red-colored alert boxes, with concise descriptions of the problem (e.g., "Invalid login credentials").
  - Validation error messages will appear beside the respective form fields (e.g., "Username is already in use by another user").
- **Success Feedback:**
  - Upon completing actions such as saving changes or enrolling in a course, success messages will appear in green (e.g., "Account registered successfully").

3. Screen Layout and Constraints

- **Homepage Layout:**

    - The homepage will feature a hero section with quick access to the most popular communities.
    - A navigation bar for the platform and for each community will be provided at the top for easy access to different pages.
- **Course and Community Pages:**
    - Each course and community will have a header section with a title, description, and action buttons (e.g., **Join or Leave Community**).
    - A sidebar will display related content (e.g., course modules, community discussions, upcoming events).
    - A content area will show detailed course materials, posts, or event information.
- **Forms:**
    - Forms (e.g., login, registration, profile update) will include standard input fields, drop-downs, and checkboxes.
    - Forms will have **Submit** and **Reset** buttons. Inline validation messages will inform users of incorrect inputs.

4. Sample Screen Elements (Example)

- **Login Screen:**
    - Email and password fields.
    - **Login** and **Join Now** buttons.
- **Course Page:**
    - Course title, description, and module list.
    - **Watch Video**, **Next**, and **Previous** buttons for video-based learning.
- **Community Post Section:**
    - Post text area with options to **Like**, **Comment**, and **Share**.
    - Tagging options for categorizing posts.

5. User Interface Software Components

- **Login/Registration Forms:** User credentials input and validation.
- **Course Management Interface:** Allows instructors to create and manage courses (add videos, content, etc.).
- **Community Management Interface:** Used by admins/moderators to create, manage, and moderate communities.
- **Event Calendar Interface:** Displays events for each community, with the ability to filter by date.
- **Profile Management Interface:** Allows users to update personal information, including password and avatar.
- **Discussion Forum:** For students and instructors to post, comment, and engage with course material.

6. User Interface Standards

- **Typography:** Consistent use of fonts for readability (e.g.,Arial).
- **Color Scheme:** A balanced color palette with accessible contrast (primary colors for headings, secondary for buttons and links).
- **Icons:** Use of modern icons (e.g., SVG icons) for actions like adding posts, liking content, and navigating.

This user interface design ensures a user-friendly, responsive, and aesthetically pleasing experience across all devices, guiding users through the various functionalities of CLMS with ease.

3.2 Hardware Interfaces

1. Supported Device Types

- **Client Devices:**
  - Desktops, laptops, tablets, and smartphones (iOS, Android).
  - Browsers: Chrome, Firefox, Safari, and Microsoft Edge.
- **Server Devices:**
  - Cloud-based or on-premise servers (AWS, Azure, DigitalOcean).
  - Database servers running MySQL for data storage.

2. Data and Control Interactions

- **Client to Server:**
  - Client devices interact with the server using HTTP/HTTPS protocols to send requests and receive responses (e.g., course enrollment, post management).
  - Data transferred includes user inputs (e.g., login credentials, course registrations) and server responses (e.g., course details, posts).
- **Server to Database:**
  - Server communicates with the MySQL database to fetch, insert, and update data (e.g., course materials, user information, community posts).
  - Communication occurs via SQL queries over the server-side backend (Node.js with Express.js).

3. Communication Protocols

- **HTTP/HTTPS** for secure communication between the client and server.
- **WebSockets** (optional) for real-time updates (e.g., live chat or notifications).

3.3 Software Interfaces

Software Interfaces

1. Frontend and Backend (React.js and Node.js/Express.js)

- **Connection:**
  - The frontend, built with **React.js**, communicates with the backend, built using **Node.js** with the **Express.js** framework, through **RESTful APIs**.
- **Data Items:**
  - Incoming: User authentication data (e.g., username, password), course selection, community posts, event registrations.
  - Outgoing: User data (e.g., course lists, community details, posts), success/error responses.
- **Communication Protocols:**

    - **HTTP/HTTPS** is used for client-server communication, with **JSON** as the data format for requests and responses.
- **Services Needed:**
    - APIs for user authentication, course management, community management, event management, and post interactions.

2. Frontend and Video Hosting Service (YouTube, Vimeo, or Custom Hosting)

- **Connection:**
    - The frontend (React.js) embeds video content from a third-party video hosting platform (e.g., **YouTube API** or **Vimeo API**) or a custom media server.
- **Data Items:**
    - Incoming: Video URL or metadata (e.g., video ID, title, description).
    - Outgoing: Playback data (e.g., video progress, pause/play status).
- **Communication Protocols:**
    - **HTTP/HTTPS** for embedding and fetching video content using external video API calls.
- **Services Needed:**
    - Video streaming services or APIs for course video delivery.

3. Backend and Database (Node.js and MySQL)

- **Connection:**
    - The backend communicates with a **MySQL database** via an **Object-Relational Mapping (ORM) library** such as **Sequelize.js** or **TypeORM**.
- **Data Items:**
    - Incoming: User inputs (e.g., login credentials, course enrollments), community interactions (e.g., post data, comment data).
    - Outgoing: User profile information, community data, event schedules, and course materials.
- **Communication Protocols:**
    - SQL queries (via ORM) for data insertion, retrieval, and updates.
- **Services Needed:**
    - SQL queries for database interaction and transaction management.
- **Shared Data:**
    - User information, course data, community posts, event information, and student progress.

4. Data Sharing and Global Data Area

- **Data Sharing Mechanism:**
    - Data such as user profiles, courses, and community posts will be shared across the frontend and backend systems. The backend must ensure that shared data is correctly synchronized and updated in the database.
    - Use of ORM (MySQL2) ensures that data is consistently shared between the backend and the MySQL database.

3.4 Communications Interfaces

Communication Requirements

1. Web Browser Communication

- **Protocol: HTTP/HTTPS** will be used for communication between the client (web browser) and the server.
- **Message Formatting:** Requests and responses will use **JSON** for data transfer.
- **Encryption:** HTTPS with **SSL/TLS** for secure, encrypted communication between client and server.
- **Data Transfer Rates:** The system will optimize payload sizes to ensure fast load times, particularly for mobile users with lower bandwidth.

2. Network Server Communication

- **Protocol: HTTP/HTTPS** for client-server communication; **WebSocket** (optional) for real-time notifications or live events.
- **Synchronization:** The server will handle requests in real-time, ensuring data consistency using RESTful APIs for synchronous actions (e.g., course enrollments) and asynchronous methods (e.g., notifications).
- **Security: JWT (JSON Web Tokens)** for secure, token-based user authentication to ensure only authorized users can interact with sensitive endpoints.

3. Electronic Forms (e.g., Login, Registration)

- **Protocol:** Form submissions will use **HTTPS** to securely transmit data to the server.
- **Data Validation:** Input data will be validated both client-side (JavaScript) and server-side (Node.js/Express.js).
- **Message Formatting:** Form data will be submitted in **application/x-www-form-urlencoded** or **application/json** format.

4. General Communication Standards

- **HTTP/HTTPS** for web communication.
- **JSON** for structured data exchange.
- **JWT** for secure user authentication.
- **WebSocket** for real-time communication (optional).

4. System Features

4.1 Post creation and deletion

4.1.1 Description and Priority

The **Post Creation and Deletion** feature allows users (Admins, Moderators, and Students) to create and delete posts within communities. Creating posts enables members to share information, ask questions, or start discussions, while deleting posts ensures that inappropriate or outdated content is removed. This feature is of **High priority** as it directly impacts user interaction and community engagement.

- **Benefit:** Enhances user engagement and content management.
- **Penalty:** Inactive or outdated posts may negatively affect user experience.
- **Risk:** High if content moderation fails to prevent inappropriate posts.

4.1.2 Stimulus/Response Sequences

1. **Post Creation:**
    - **User Action:** The user clicks on the "Create Post" button in a community.
    - **System Response:** The system opens a form for post creation (title, content, tags).
    - **User Action:** The user enters the title, content, and selects tags (if any).
    - **System Response:** The system validates the input. If valid, the post is created and displayed in the community; if invalid (e.g., empty content or title), an error message is displayed.
    - **User Action:** The user submits the form.
    - **System Response:** The system creates the post and displays it in the community feed.
2. **Post Deletion:**
    - **User Action:** The user clicks on the "Delete" button associated with an existing post (available for Admins, Moderators, or the post author).
    - **System Response:** The system prompts for confirmation ("Are you sure you want to delete this post?").
    - **User Action:** The user confirms the deletion.
    - **System Response:** The post is permanently deleted from the community feed.

4.1.3 Functional Requirements

- **REQ-1:** The system must provide a "Create Post" button within each community.
- **REQ-2:** The post creation form must include fields for **Title**, **Content**, and **Tags** (optional).
- **REQ-3:** The system must validate that **Title** and **Content** are not empty when submitting a post. If either field is empty, an error message should be displayed.
- **REQ-4:** Only **Admins**, **Moderators**, and the post **author** can delete posts.

- **REQ-5:** Upon deletion, the system must display a confirmation dialog ("Are you sure you want to delete this post?"). If the user confirms, the post should be permanently removed.
- **REQ-6:** Deleted posts should no longer be visible in the community feed for any user.
- **REQ-7:** In case of an invalid input (empty title/content), the system should display an error message near the affected field, such as "Title cannot be empty" or "Content cannot be empty."
- **REQ-8:** The system should log post creation and deletion actions for auditing purposes, capturing user ID, timestamp, and action type.

4.2 Create Community

The **Create Community** feature enables **Admins** and **Moderators** to create new public or private communities within the platform. Communities allow users to collaborate, share content, and interact with others around common interests or educational topics. This feature is of **High priority** as it is central to the platform's social and collaborative functionality.

- **Benefit:** Encourages user engagement and interaction.
- **Penalty:** Lack of community creation may reduce user engagement and platform growth.
- **Risk:** Mismanagement of community creation could lead to unauthorized or inappropriate communities.

4.2.2 Stimulus/Response Sequences

1. **Community Creation:**
    - **User Action:** The user (Admin or Moderator) clicks on the "Create Community" button on the dashboard or community section.
    - **System Response:** The system displays a form for creating a new community with fields for **Community Name**, **Description**, **Community Type** (Public/Private), and **Community Tags** (optional).
    - **User Action:** The user enters the required details: community name, description, and selects community type (Public/Private).
    - **System Response:** The system validates the input. If the input is valid, the system creates the community; if invalid (e.g., duplicate name or missing description), an error message is displayed.
    - **User Action:** The user submits the form.

- **System Response:** The system creates the community and redirects the user to the newly created community page.

4.2.3 Functional Requirements

- **REQ-1**: The system must provide a "Create Community" button visible to **Admins** and **Moderators**.
- **REQ-2:** The community creation form must include the following fields:
  - **Community Name** (required, unique across the platform).
  - **Community Description** (required).
  - **Community Type** (dropdown with options: Public, Private).
  - **Community Tags** (optional, for categorizing communities).
- **REQ-3:** The system must validate that the **Community Name** is unique and not already in use. If the name is already taken, the system should display an error message: "Community Name already exists."
- **REQ-4:** The **Community Description** field must not be empty. If the description is missing, an error message should be displayed: "Community Description cannot be empty."
- **REQ-5:** If the user selects "Private" as the community type, the system must display an option to set **Membership Approval** (i.e., whether Admins/Moderators must approve join requests).
- **REQ-6:** Upon submission of valid input, the system must create the community and redirect the user to the new community's main page.

4.3 Retrieving community information

The **Retrieving Community Information** feature allows users to view detailed information about a community, including its name, description, membership, posts, and upcoming events. This feature enables users to explore available communities, decide whether to join, and engage with the content shared within each community. It is of **High priority** as it supports user discovery and engagement, which are core aspects of the platform.

- **Benefit:** Facilitates user participation and interaction by providing accessible community information.
- **Penalty:** Lack of community visibility may hinder user engagement and participation.

- **Risk:** Incorrect or outdated community data may lead to confusion and reduced user trust.

4.3.2 Stimulus/Response Sequences

1. **Retrieving Community Information:**
    - **User Action:** The user clicks on the name or icon of a community listed on the platform or from a search result.
    - **System Response:** The system retrieves and displays the community's detailed information, including the community name, description, membership list, recent posts, and any scheduled events.
    - **User Action:** The user views the community details and decides whether to join or interact with the community content.
    - **System Response:** If the user is not yet a member, the system may prompt them with an option to "Join Community" (for public communities or after admin/moderator approval for private communities).

4.3.3 Functional Requirements

- **REQ-1:** The system must display the following community details upon user request (clicking on a community):
    - **Community Name**
    - **Community Description**
    - **Membership List** (showing the number of members, clickable for viewing individual profiles)
    - **Recent Posts** (most recent posts with options to like, comment, or view the full post)
    - **Upcoming Events** (if any, with options to RSVP or learn more)
- **REQ-2:** The system must provide an option to view a list of **community members**, showing their names, profile pictures, and roles (e.g., Admin, Moderator, Member).
- **REQ-3:** The system should display **recent posts** within the community in a scrollable list, including the post's title, author, and date.
- **REQ-4:** If the community has **scheduled events**, the system must display them in chronological order, including the event title, description, date, and time.

- **REQ-5:** The system must allow users to easily navigate between sections (e.g., posts, events, members) within the community page using tabs or navigation links.
- **REQ-6:** For private communities, the system must indicate the **privacy status** (e.g., “Private Community – Membership approval required”) to inform users of access restrictions.
- **REQ-7:** The system must display a **Join Community** button for public communities, and an **Apply to Join** button for private communities that require admin/moderator approval.
- **REQ-8:** The system should ensure that **real-time updates** are reflected in the community page, including new posts or event changes. If using WebSockets, the system should update the content without requiring the user to refresh the page.
- **REQ-9:** The system must ensure that community information is always up-to-date, particularly membership lists and posts. Information should be fetched in real-time or with minimal delay.
- **REQ-10:** The system should provide an **error message** if the community information cannot be retrieved (e.g., due to network issues or server errors), stating, "Unable to load community information. Please try again later."

This feature enhances the user experience by providing rich, accessible information about each community, helping users make informed decisions about their engagement.

4.4 Join/Leave Community

4.4.1 Description and Priority

The **Joining and Leaving Community** feature allows users to join or leave communities based on their interests or needs. This feature is crucial for enabling user participation and engagement within the platform. Joining a community lets users interact with its content and members, while leaving a community removes them from the group, ensuring users maintain control over their participation. This feature is of **High priority** as it directly influences user retention and community dynamics.

- **Benefit:** Enhances user engagement by allowing users to easily participate or disengage from communities.

- **Penalty:** Lack of flexibility in joining or leaving communities could reduce user participation and satisfaction.
- **Risk:** Inadequate access control for joining or leaving private communities may lead to security issues or unauthorized access.

4.4.2 Stimulus/Response Sequences

1. **Joining a Community:**
    - **User Action:** The user clicks the "Join Community" button on the community page (for public communities or after approval for private communities).
    - **System Response:** If the community is public, the system adds the user to the community and updates the community membership list. For private communities, the system prompts the user to request membership approval.
    - **User Action:** The user submits a membership request for private communities.
    - **System Response:** The system notifies the community Admin or Moderator to review the request and approves or declines it.
    - **User Action:** If approved, the user receives a notification confirming their membership in the community.
    - **System Response:** The system updates the community's membership list and grants the user access to community posts, discussions, and events.
2. **Leaving a Community:**
    - **User Action:** The user clicks the "Leave Community" button on the community page.
    - **System Response:** The system prompts the user with a confirmation dialog ("Are you sure you want to leave this community?").
    - **User Action:** The user confirms the decision to leave.
    - **System Response:** The system removes the user from the community, updates the membership list, and prevents access to community posts and events.

4.4.3 Functional Requirements

- **REQ-1**: The system must provide a **"Join Community"** button for public communities, visible to users who are not currently members.
- **REQ-2:** The system must provide an **"Apply to Join"** button for private communities, visible to users who are not yet members, with an option to send a membership request for approval.
- **REQ-3:** For private communities, the system must notify **Admins** or **Moderators** when a user applies to join, and allow them to approve or reject the request.
- **REQ-4:** If a user's request to join a private community is approved, the system must send a confirmation notification to the user.
- **REQ-5:** The system must provide a **"Leave Community"** button for users who are members of the community, allowing them to leave at any time.
- **REQ-6:** Upon clicking "Leave Community," the system must prompt the user with a confirmation message: "Are you sure you want to leave this community?"
- **REQ-7:** After confirming to leave, the system must remove the user from the community, update the membership list, and restrict access to community content (posts, events, etc.).
- **REQ-8:** The system must ensure that users can rejoin public communities at any time by clicking the "Join Community" button again.
- **REQ-9:** For private communities, users must be able to reapply for membership if they previously left, but rejoining is subject to **Admin/Moderator approval**.

This feature ensures that users have full control over their community participation, promoting flexibility and personalization within the platform.

4.5 Update Privilege

4.5.1 Description and Priority

The **Update Privileges** feature allows **Admins** and **Moderators** to modify the roles and permissions of community members. This includes upgrading or downgrading roles such as Admin, Moderator, or Member based on user behavior or need. It ensures that proper access control is maintained within communities. This feature is of **High priority** as it impacts the security and management of the community, ensuring only authorized users have control over sensitive actions.

- **Benefit:** Ensures proper access control and management within communities.
- **Penalty:** Lack of privilege management could lead to unauthorized actions, security risks, and mismanagement of community resources.
- **Risk:** Incorrect privilege assignments could disrupt community operations or lead to unauthorized content management.

4.5.2 Stimulus/Response Sequences

1. **Updating Privileges:**
    - **User Action:** The Admin or Moderator clicks the "Manage Members" button in the community settings to view the list of members.
    - **System Response:** The system displays the list of members along with their current roles (e.g., Admin, Moderator, Member).
    - **User Action:** The Admin or Moderator selects a member and clicks "Update Privileges" next to their name.
    - **System Response:** The system presents options to modify the user's role (e.g., "Promote to Moderator," "Demote to Member").
    - **User Action:** The Admin or Moderator selects the new role and confirms the change.
    - **System Response:** The system updates the user's role and sends a notification to the user about the privilege update (if required).
    - **User Action:** The system updates the member’s role in the community, reflecting the new permissions.

4.5.3 Functional Requirements

- **REQ-1**: The system must allow **Admins** and **Moderators** to view a list of community members and their current roles (Admin, Moderator, Member).
- **REQ-2:** The system must allow **Admins** and **Moderators** to update the roles of members by selecting an option to promote or demote a user (e.g., **Promote to Moderator**, **Demote to Member**).
- **REQ-3:** The system must ensure that only **Admins** and **Moderators** can update the privileges of community members.
- **REQ-4:** The system must send a **notification** to the affected user when their privileges are updated, informing them of the new role.
- **REQ-5:** The system must update the user's role immediately after confirmation, reflecting the new permissions for community management.

- **REQ-6:** The system must prevent **Members** from updating their own privileges or those of other users.
- **REQ-7:** When an Admin or Moderator updates a user's privilege, the system must log the change, including the **user ID**, **updated role**, **timestamp**, and **admin/moderator ID** making the change.
- **REQ-8:** The system must ensure that updated privileges take effect immediately, allowing users to access or restrict access to certain community features based on their new roles.
- **REQ-9:** The system must enforce role-based access control (RBAC) after updating privileges, ensuring users with elevated roles (e.g., Admin, Moderator) can perform higher-level actions, while Members have limited access.
- **REQ-10:** In case of invalid input (e.g., attempting to promote a user without sufficient permissions), the system must display an error message like, "You do not have permission to update this user's privileges."

4.6 Update Community information

4.6.1 Description and Priority

The **Update Community Information** feature allows **Admins** and **Moderators** to modify the details of an existing community, such as the community name, description, and banner. This ensures that the community's identity and content stay relevant and up to date. This feature is of **High priority** as it directly impacts the management and branding of the community, ensuring that it aligns with its purpose and audience.

- **Benefit:** Ensures community information is accurate, relevant, and reflective of its purpose.
- **Penalty:** Outdated or inaccurate community details can lead to confusion and reduce user engagement.
- **Risk:** Unauthorized or incorrect updates may misrepresent the community or disrupt its functioning.

4.6.2 Stimulus/Response Sequences

1. **Updating Community Information:**

- **User Action:** The **Admin** or **Moderator** clicks on the "Edit Community" button in the community settings.
- **System Response:** The system opens the community information edit form, displaying the current community name, description, banner image, and other editable fields.
- **User Action:** The Admin or Moderator updates the community name, description, and optionally uploads a new banner image.
- **System Response:** The system validates the input to ensure the community name and description are not empty. If validation fails, an error message is displayed.
- **User Action:** The Admin or Moderator submits the changes.
- **System Response:** The system updates the community's information and displays a confirmation message. The updated details are immediately reflected on the community page.

4.6.3 Functional Requirements

- **REQ-1:** The system must provide an **"Edit Community"** button visible to **Admins** and **Moderators** in the community settings.
- **REQ-2:** The community information form must allow **Admins** and **Moderators** to edit the following fields:
  - **Community Name** (required, must be unique).
  - **Community Description** (required).
  - **Community Banner Image** (optional, must support standard image formats like JPG, PNG).
  - **Community Tags** (optional).
- **REQ-3:** The system must validate that the **Community Name** and **Community Description** fields are not empty before submission. If either field is empty, an error message should be displayed: "Community Name cannot be empty" or "Community Description cannot be empty."
- **REQ-4:** The system must ensure that **Community Name** is unique across the platform, and display an error message if the name already exists: "Community Name already taken."
- **REQ-5:** The system must allow **Admins** and **Moderators** to upload a new banner image. The banner image must be displayed at the top of the community page.
- **REQ-6:** Once the user submits the updated information, the system must immediately reflect the changes on the community page.

- **REQ-7:** The system must send a **confirmation notification** to the **Admin** or **Moderator** after successfully updating the community information.
- **REQ-8:** The system must log the update action, including the **user ID** of the Admin/Moderator, **updated fields**, **timestamp**, and **community ID**.
- **REQ-9:** The system should prevent **Members** from updating community information, allowing only **Admins** and **Moderators** to make changes.
- **REQ-10:** In case of failure (e.g., invalid banner image format, duplicate community name), the system should display an appropriate error message such as "Invalid image format" or "Community name already exists."

4.7 Accept/Reject community join request

4.7.1 Description and Priority

The **Accept/Reject Community Join Request** feature enables **Admins** and **Moderators** to review and manage requests from users wishing to join a **private community**. This ensures that membership is controlled and that only authorized users gain access to the community. This feature is of **High priority** as it is essential for maintaining community integrity and security.

- **Benefit:** Ensures that only authorized users join private communities.
- **Penalty:** Failure to manage requests may lead to unauthorized access.
- **Risk:** Mismanagement of requests could lead to the inclusion of inappropriate or non-relevant members.

4.7.2 Stimulus/Response Sequences

1. **Accepting a Join Request:**
    - **User Action:** The Admin or Moderator views pending join requests in the community management dashboard.
    - **System Response:** The system displays a list of pending requests with user details.
    - **User Action:** The Admin or Moderator clicks "Accept" next to a user's request.
    - **System Response:** The system adds the user to the community, updates the membership list, and sends a notification to the user confirming their membership.

2. **Rejecting a Join Request:**

    - **User Action:** The Admin or Moderator clicks "Reject" next to a pending join request.
    - **System Response:** The system removes the request from the list and notifies the user that their request has been denied.

4.7.3 Functional Requirements

- **REQ-1**: The system must display a list of **pending join requests** for private communities, including user names and details (e.g., profile picture, reason for joining).
- **REQ-2:** The system must allow **Admins** and **Moderators** to **accept** or **reject** join requests for private communities.
- **REQ-3:** Upon accepting a request, the system must add the user to the community, update the membership list, and send a **confirmation notification** to the user.
- **REQ-4:** Upon rejecting a request, the system must remove the request from the pending list and send a **rejection notification** to the user, explaining the reason (if applicable).
- **REQ-5:** The system must log each action (accept or reject) with the **Admin/Moderator's user ID**, **timestamp**, and **community ID** for auditing purposes.

4.8 Like on a Post

4.8.1 Description and Priority

The **Like on a Post** feature allows users to express their approval or interest in a community post by liking it. This feature is fundamental for user interaction, encouraging engagement and feedback within community discussions. It is of **Medium priority** as it enhances social engagement but is not critical to the functionality of the platform.

- **Benefit:** Increases user interaction and engagement with posts.
- **Penalty:** Lack of interaction features may reduce user participation and content visibility.

- **Risk:** Users may misuse the like feature if not properly managed (e.g., excessive liking or spamming).

4.8.2 Stimulus/Response Sequences

1. **Liking a Post:**
    - **User Action:** The user clicks the "Like" button under a post.
    - **System Response:** The system increases the like count and changes the "Like" button to a "Liked" state (e.g., showing a filled heart icon).
    - **User Action:** The user can click the "Like" button again to undo the like.
    - **System Response:** The system decreases the like count and reverts the "Liked" state to "Like."

4.8.3 Functional Requirements

- **REQ-1**: The system must provide a **"Like" button** below each post for users to express approval.
- **REQ-2:** The system must increase the post's **like count** by 1 each time a user clicks the "Like" button.
- **REQ-3:** The system must allow users to **undo a like**, decreasing the like count by 1 if they click the "Liked" button again.
- **REQ-4:** The system must visually indicate the like status (e.g., filled heart icon for "Liked," empty heart for "Like").
- **REQ-5:** The system must ensure that each user can like a post only once, and prevent multiple likes from the same user.

4.9 Replying on a comment

4.9.1 Description and Priority

The **Replying on a Comment** feature allows users to respond to comments on community posts. This feature promotes interactive discussions and allows users to engage in threaded conversations. It is of **High priority** as it encourages deeper interaction and fosters a sense of community within the platform.

- **Benefit:** Enhances discussion and engagement through threaded conversations.

- **Penalty:** Without replies, discussions may become fragmented and lack context.
- **Risk:** Mismanagement of comment threads (e.g., excessive replies) could lead to disorganized conversations.

4.9.2 Stimulus/Response Sequences

1. **Replying to a Comment:**
    - **User Action:** The user clicks the "Reply" button beneath a comment.
    - **System Response:** The system opens a text input field for the user to type their reply.
    - **User Action:** The user enters their reply and submits it.
    - **System Response:** The system adds the reply to the thread under the original comment and updates the conversation view with the new reply.

4.9.3 Functional Requirements

- **REQ-1**: The system must provide a **"Reply" button** beneath each comment for users to reply to it.
- **REQ-2:** The system must open a **text input field** upon clicking the "Reply" button, allowing users to type and submit their reply.
- **REQ-3:** The system must display the reply in a **threaded** format, directly beneath the comment it is replying to.
- **REQ-4:** The system must update the comment section in real-time to reflect the new reply without needing a page refresh.
- **REQ-5:** The system must ensure that replies are associated with the correct comment, maintaining the proper thread structure for clarity and context.

4.10 Attach tag on a Post

4.10.1 Description and Priority

The **Attach Tag on a Post** feature allows users to categorize and label posts by attaching relevant tags. This feature enhances post discoverability and improves content organization within the community. It is of **Medium priority** as it aids in content management and searchability but is not critical to basic functionality.

- **Benefit:** Improves content discovery and helps users find relevant posts easily.
- **Penalty:** Lack of tagging could lead to disorganized content and difficulty in finding related posts.
- **Risk:** Improper tagging or misuse of tags could lead to confusion or clutter.

4.10.2 Stimulus/Response Sequences

1. **Attaching a Tag to a Post:**
    - **User Action:** The user clicks the "Add Tag" button on a post.
    - **System Response:** The system opens a text field or dropdown where the user can type or select tags.
    - **User Action:** The user types a tag or selects from predefined options.
    - **System Response:** The system attaches the tag to the post and displays it alongside the post content.
    - **User Action:** The user submits the post or the tag.
    - **System Response:** The system saves the tag and associates it with the post for search and filtering purposes.

4.10.3 Functional Requirements

- **REQ-1**: The system must provide an **"Add Tag"** button for users to add tags to a post.
- **REQ-2:** The system must allow users to either **type** new tags or **select** from a list of predefined tags (if available).
- **REQ-3:** The system must display the tags clearly alongside the post, with the option to click on tags for related content.
- **REQ-4:** The system must allow **multiple tags** to be attached to a single post, separated by commas or spaces.
- **REQ-5:** The system must validate tag input to ensure that only valid characters (e.g., alphanumeric) are allowed and prevent submission of empty or duplicate tags.

4.11 Access courses of a community

4.11.1 Description and Priority

The **Access Courses of a Community** feature allows users to view and participate in courses associated with a community. This feature provides an easy way for users to engage in learning materials shared within specific communities. It is of **High priority** as it directly supports the platform's educational functionality, enabling content delivery and knowledge sharing.

- **Benefit:** Facilitates learning and course participation within communities.
- **Penalty:** Without easy access to courses, user engagement and learning experiences may decrease.
- **Risk:** Lack of access control or mismanagement of course availability could result in unauthorized users viewing or participating in courses.

4.11.2 Stimulus/Response Sequences

1. **Accessing a Course:**
    - **User Action:** The user navigates to a community page and clicks on the "Courses" tab.
    - **System Response:** The system displays a list of available courses within the community.
    - **User Action:** The user clicks on a course title to view course details.
    - **System Response:** The system opens the course page, displaying course materials, video content, and navigation options.
    - **User Action:** The user clicks "Start Course" or "Continue" to begin or resume learning.
    - **System Response:** The system allows the user to access and interact with the course materials.

4.11.3 Functional Requirements

- **REQ-1:** The system must display a **"Courses"** tab or section within the community page to allow users to access available courses.
- **REQ-2:** The system must display a list of available courses, including course titles, descriptions, and enrollment status (e.g., enrolled, not enrolled).
- **REQ-3:** The system must allow users to **view course details** by clicking on a course title, which includes information such as course description, syllabus, and instructor.

- **REQ-4:** The system must provide a **"Start Course"** or **"Continue"** button for users to begin or resume a course they are enrolled in.
- **REQ-5:** The system must ensure that only **authorized users** (those who are enrolled or have access permissions) can access the course materials, preventing unauthorized access to private or restricted courses.

4.12 Add/Remove/Access Events of a community

4.12.1 Description and Priority

The **Add/Remove/Access Events of a Community** feature enables **Admins** and **Moderators** to manage events within a community, allowing users to view, add, or remove events such as webinars, meetups, or discussions. This feature enhances community engagement by organizing events that encourage participation. It is of **High priority** as it is crucial for managing community activities and fostering engagement.

- **Benefit:** Organizes and promotes participation in community events.
- **Penalty:** Lack of event management could reduce user interaction and community activity.
- **Risk:** Mismanagement of event visibility or access could cause confusion and reduced event participation.

4.12.2 Stimulus/Response Sequences

1. **Adding an Event:**
    - **User Action:** The Admin or Moderator clicks on the "Add Event" button in the community's event section.
    - **System Response:** The system displays a form with fields for event details such as title, description, date, and time.
    - **User Action:** The Admin or Moderator enters the event details and clicks "Submit."
    - **System Response:** The system validates the event details (e.g., ensuring the date is not in the past) and adds the event to the community calendar.
    - **User Action:** The system confirms the event addition and displays the new event on the community page.

2. **Removing an Event:**
    - **User Action:** The Admin or Moderator clicks the "Remove" button next to an event.
    - **System Response:** The system asks for confirmation to remove the event.
    - **User Action:** The Admin or Moderator confirms the removal.
    - **System Response:** The system deletes the event from the community calendar and notifies relevant users of the cancellation.
3. **Accessing an Event:**
    - **User Action:** The user clicks on the event title in the community's event section.
    - **System Response:** The system opens the event details page, showing the event description, date, time, and RSVP options.
    - **User Action:** The user clicks "RSVP" to register for the event.
    - **System Response:** The system records the RSVP and confirms the registration.

4.12.3 Functional Requirements

- **REQ-1:** The system must provide an **"Add Event"** button for **Admins** and **Moderators** to create new events.
- **REQ-2:** The event creation form must include the following fields: **Event Title**, **Description**, **Date and Time**, and **Location** (optional).
- **REQ-3:** The system must validate that the event **date** is not in the past and that all required fields are filled out before submission.
- **REQ-4:** The system must allow **Admins** and **Moderators** to **remove events**, with a confirmation prompt before deletion.
- **REQ-5:** The system must allow **users** to **RSVP** for events, with a clear confirmation message upon successful registration and an option to cancel the RSVP.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

For our application, the main criteria is the smoothness and delivery of content at real

time and maintaining a consistent view against many users about some information. This can include seeing the same number of comments, the same number of likes. The high performance criteria here in this case are,

- Very smooth UX experience and good delivery of content with speed without too much lag
- Reliability in terms of the security measure taken to make sure no privacy credentials are released to the malicious hackers with miss intent.
- High availability of the application is a huge driving force. Being down for a few hours has a huge impact on the consumer base.

These are very important factors to consider, only after we put our project into deployment and production. Unfortunately, we have not gotten the time to do so

5.2 Safety Requirements

There is room for a group of people to misuse the system to encourage inappropriate action towards communities by igniting discussions that spiral into hate and disgust. These have real impacts on the social lives of people, and so far at this scope, we have not been able to address this issue.

5.3 Security Requirements

Some criteria can be,

- Not storing passwords as simple plain text
- Keeping secrets out of the main repository
- Queries made to the database MUST be sanitized
- No internal logging of the systems should be view-able by a user.
- Root access to the databases should be protected by a a hashed password
- A rate limiter to make sure no user makes too many requests

5.4 Software Quality Attributes

1. **Usability**
    - The platform should be designed with **ease of use** as a top priority, ensuring that users can easily navigate the system, complete tasks, and find information with minimal effort.
    - **Task completion time** should not exceed **3 minutes** for common tasks such as enrolling in a course, posting in a community, or registering for an event.
    - **User satisfaction** should be measured through surveys, aiming for a **minimum 85% user satisfaction rate** regarding the ease of use.
2. **Availability**
    - The system must ensure **99.9% uptime** to ensure reliable access for users, including during peak hours.
    - Scheduled maintenance should be planned outside of peak usage times to minimize disruptions.

3. **Scalability**
    - The platform must be designed to **scale horizontally** to handle growth in user base, courses, and content. It should accommodate a **minimum of 10,000 users** without performance degradation.
    - System performance should maintain response times (page load, data retrieval) under **2 seconds** even with a substantial increase in user load.
4. **Maintainability**
    - The system must include **unit tests** that cover at least **80% of the code** to detect and fix issues early.
    - Documentation (code, APIs, and system architecture) should be up-to-date and comprehensive, ensuring that developers can maintain and extend the system efficiently.
5. **Interoperability**
    - The system must be able to integrate with external services, such as **email providers (SendGrid, Mailgun)**, **payment processors (Stripe, PayPal)**, and **video hosting platforms (YouTube, Vimeo)** using their standard APIs.
    - The platform should also support **cross-platform compatibility**, working seamlessly across modern web browsers and mobile devices.
6. **Reliability**
    - The system should handle **fault tolerance** with minimal impact on user experience, especially for critical functions like user login and course access.
    - In case of failure, the system should provide **graceful error messages** and allow users to retry actions, ensuring no loss of data during network or server interruptions.
7. **Flexibility**
    - The system should allow **customization** of course layouts, community settings, and notification preferences, offering users the ability to personalize their experience.
    - New features should be easy to integrate into the system without significant rework of existing functionality.
8. **Portability**
    - The platform should be operable across a variety of environments, including cloud-based services (e.g., AWS, Azure) and on-premise servers, with the ability to migrate between these environments with minimal configuration changes.
9. **Reusability**
    - Components and services (e.g., user authentication, course creation, event scheduling) should be designed for **maximum reusability**, allowing future expansions or features to leverage existing code with minimal changes.
10. **Testability**
    - The system must include automated **unit, integration, and end-to-end tests** to ensure functional correctness and prevent regressions.

- Test coverage should be **80% or higher** for key functionality, ensuring that any changes are adequately tested before deployment.

These quality characteristics define the expectations for the platform's performance, user experience, and maintainability, ensuring that the system meets both customer and developer needs effectively.

5.5 Business Rules

1. **User Roles and Permissions**
    - **Administrators**:
        - Have full access to all system functionalities, including creating and managing communities, courses, events, and user accounts.
        - Can assign or modify roles (Admin, Moderator, Instructor, Student) within communities.
        - Responsible for system settings and enforcing privacy policies.
    - **Moderators**:
        - Can manage content within communities (e.g., approve or delete posts, moderate discussions).
        - Can manage community member roles but cannot create new communities or modify system settings.
    - **Instructors**:
        - Can create and manage courses, including uploading course materials (e.g., videos, documents).
        - Have limited control over community management within their course's community but cannot manage system settings or roles outside their scope.
    - **Students**:
        - Can enroll in courses, participate in community discussions, and access learning materials.
        - Cannot modify courses, communities, or roles; only able to interact with content.
2. **Community Management**
    - **Private Communities**:
        - Only **Administrators** and **Moderators** can approve or decline membership requests for private communities.
        - **Members** of private communities can only invite others with permission from Admins or Moderators.
    - **Public Communities**:
        - Anyone can join without approval, but once joined, **Members** can only participate in discussions, not manage the community or moderate posts.
3. **Course Enrollment**
    - **Instructors**:
        - Can create courses but cannot enroll students manually.
        - Courses can be free or paid, with **Admins** responsible for integrating payment gateways if needed.

- **Students**:
  - Can only enroll in courses that are open or free. Paid courses require a valid payment method through a **secure payment gateway** (e.g., Stripe, PayPal).
  - Students cannot delete or modify course content; they only have access to learning materials.

4. **Event Management**

   - **Admins and Moderators**:
     - Can create, edit, and delete community-specific events (e.g., webinars, group study sessions).
     - **Students** can RSVP for events but cannot modify event details.
     - **Admins** will have the ability to delete events if needed, while **Moderators** can manage attendance.

5. **Data Privacy**

   - **Admin Access to User Data**:
     - **Administrators** can view all user data but must ensure that access to sensitive information (e.g., passwords, personal details) complies with data privacy regulations (e.g., GDPR, CCPA).
     - **Moderators** and **Instructors** have limited access to user data, primarily to interact with students in the context of their courses or communities, without seeing sensitive personal data.

6. **Content Moderation**

   - **Post Deletion**:
     - Only **Moderators** and **Admins** can delete posts or comments in communities.
     - **Students** can report inappropriate content, but the action will require approval from a **Moderator** or **Admin**.

6. Other Requirements

1. Database Requirements

- **Database Management System (DBMS):**
  - The system will use **MySQL** as the relational database for storing user data, course materials, community posts, and event information.
  - The database must support **ACID** (Atomicity, Consistency, Isolation, Durability) properties to ensure data integrity and reliability during transactions.
- **Data Backup:**
  - The platform must have **daily backups** of all critical data to prevent data loss. Backups should be stored securely, with redundancy across different physical locations for disaster recovery.

2. Internationalization and Localization Requirements

- **Language Support:**

    - The platform must support **multi-language with** the capability to switch between languages, starting with English, Spanish, French, and German.
    - All static text (e.g., labels, messages) should be stored in **external language files** to allow easy localization.
- **Time Zone Handling:**
    - The system should handle **time zones** for events and course schedules, ensuring users in different regions see event times in their local time zone.
    - Date and time formats should be adaptable to different regional conventions (e.g., MM/DD/YYYY for the US, DD/MM/YYYY for the UK).

3. Legal and Compliance Requirements

- **Data Privacy Regulations:**
    - The system must comply with **General Data Protection Regulation (GDPR)** for EU users and **California Consumer Privacy Act (CCPA)** for California-based users.
    - Users must be informed of the system's data collection and privacy policies, and consent must be obtained for the collection of personal data during account creation.
- **Intellectual Property (IP):**
    - Any content created by users (e.g., course materials, posts, images) must remain the intellectual property of the content creator unless otherwise agreed.
    - **Terms of Service** and **Privacy Policy** documents must clearly state the ownership and licensing terms for content and user data.

4. Reuse Objectives

- **Code Reusability:**
    - The system should be modular, allowing for reuse of common components across the platform, such as authentication, user profile management, and course enrollment.
    - **APIs** should be designed for potential future reuse, enabling integration with third-party services (e.g., payment gateways, video platforms) without major changes to the core system.
- **Componentization:**
    - The system should use a **microservices architecture** or **modular design**, where distinct components (e.g., user management, course management, event scheduling) can be independently updated or replaced with minimal impact on the rest of the system.

5. Scalability and Performance Requirements

- The platform must be able to scale horizontally to accommodate up to **10,000 users** and **1,000 concurrent users** without significant performance degradation.
- **Load Balancing** and **Auto-scaling** mechanisms should be implemented to distribute traffic and optimize performance during peak usage periods.
- Database queries should be optimized to handle high-volume transactions efficiently, with indexing on frequently queried fields (e.g., user IDs, course IDs, community tags).

6. System Monitoring and Logging

- The system should include **real-time monitoring** to track server performance, user activity, and potential errors.
- **Logging** should capture relevant events (e.g., login attempts, error messages, system events) to ensure proper debugging and auditing. Logs should be stored securely and retain data for at least **30 days** for troubleshooting purposes.

7. User Feedback and Reporting

- The system should include a **feedback mechanism** that allows users to submit feedback on courses, communities, and system functionality.
- **Admin Analytics** should be provided to track user engagement, course popularity, and community activity, enabling continuous improvement of the platform.

Appendix A: Glossary

ACID (Atomicity, Consistency, Isolation, Durability)

- A set of properties that guarantee database transactions are processed reliably and ensure data integrity.

API (Application Programming Interface)

- A set of protocols and tools that allow different software applications to communicate with each other.

AJAX (Asynchronous JavaScript and XML)

- A technique used in web development to update parts of a web page asynchronously without reloading the entire page.

JWT (JSON Web Token)

- A compact, URL-safe token used to represent claims securely between two parties, typically for user authentication.

MFA (Multi-Factor Authentication)

- A security mechanism that requires two or more forms of identification to access a system, enhancing security.

MySQL

- An open-source relational database management system (RDBMS) that uses SQL (Structured Query Language) for managing and organizing data.

ORM (Object-Relational Mapping)

- A technique that allows developers to interact with a relational database using object-oriented programming languages.

REST (Representational State Transfer)

- A software architectural style that defines a set of constraints for creating scalable web services, often using HTTP and JSON for communication.

SSL (Secure Sockets Layer)

- A standard security protocol for establishing encrypted links between a server and a client, ensuring secure data transmission over the internet.

TLS (Transport Layer Security)

- A cryptographic protocol designed to provide secure communication over a computer network, the successor to SSL.

UI (User Interface)

- The space where interactions between humans and machines occur, focusing on the design of screens, pages, and other visual elements.

URL (Uniform Resource Locator)

- The address used to access resources on the internet, such as a webpage or API endpoint.

UX (User Experience)

- The overall experience a user has when interacting with a product, focusing on the ease of use and satisfaction.

WebSocket

- A communication protocol that enables full-duplex, real-time communication between client and server over a single, long-lived connection.