

A PROJECT REPORT ON
GyanShristi: Content Management System (CMS)



BY

Manoj Shrestha (76288013)

Seezan Shrestha (76288024)

Usha Gurung (76288026)

An Educational Project Report Submitted in

Faculty of Education, Tribhuvan University

in partial fulfillment of the requirements for the degree of

Bachelor of Education in Information Communication Technology

at the

Aadikavi Bhanubhakta Campus

Tribhuvan University

(January, 2025)

CANDIDATE’S DECLARATION

This is to certify that we have completed the Educational Project entitled "**Content Management System**" under the guidance of Er. Ghan Bahadur Thapa, Supervisor of Aadikavi Bhanubhakta Campus, in the partial fulfillment of the requirements for the degree of **Bachelor of Education in Information Communication Technology** at Faculty of Education, Tribhuvan University. This is our original work, and we have not submitted it earlier elsewhere.

Date: January, 2025

Name: Manoj Shrestha

Name: Seezan Shrestha

Name: Usha Gurung

SUPERVISOR’S RECOMMENDATION

It is with great satisfaction that I recommend the project report titled “**Content Management System**” which has been meticulously prepared under my supervision by **Manoj Shrestha, Seezan Shrestha and Usha Gurung**. This report fulfills a portion of the requirements for the Bachelor in Education of Information Communication Technology (B.Ed. ICT) degree. The work presented in this report is both original and satisfactory, reflecting their diligent efforts and adherence to academic standards, and is ready for further evaluation.

.....

Er. Ghan Bahadur Thapa

Project Supervisor

Aadikavi Bhanubhakta Campus

CERTIFICATE OF APPROVAL

These undersigned certify that has read and recommended to the Department of Information and Communication Technology for acceptance, a project report entitled **“CONTENT MANAGEMENT SYSTEM”** submitted by **Manoj Shrestha, Seezan Shrestha** and **Usha Gurung** in partial fulfillment for the degree of Bachelor of Information Communication and Technology (B.Ed. ICT), Institute of Information and Communication Technology, Tribhuvan University.

.....

Er. Ghan Bahadur Thapa

Internal Examiner

.....

Mr. Parshuram Upadhyaya

External Examiner

ACKNOWLEDGMENT

I would like to express my sincere gratitude to the following individuals and organizations who have played a crucial role in the completion of this final project. We are also deeply thankful to everyone for the guidance and support throughout the development of this project, titled "**Content Management System.**"

We are highly indebted to **Mr. Mahaprashad Hadkhale**, Campus Chief of **Aadikavi Bhanubhakta Campus** for providing us with the opportunity to work on this project and for supporting a learning environment conducive to our academic growth. We would like to thank **Er. Ghan Bahadur Thapa**, ICT Coordinator, Aadikavi Bhanubhakta Campus, whose expertise and insights have been invaluable throughout the entire project and guide us throughout the project by providing all the necessary ideas, information and knowledge for the project. We are grateful for the time and effort you dedicated to helping us succeed. His constant support and feedback have been invaluable in overcoming the challenges we faced. Without his support, the project would not have been completed.

Last but not the least a special thanks to my teammates and classmates who contributed to meaningful discussion and provided constructive feedback. Your collaboration has enhanced the quality of this project, and I appreciate the teamwork that made this possible.

Manoj Shrestha (76288013)

Seezan Shrestha (76288024)

Usha Gurung (76288026)

EXECUTIVE SUMMARY

We propose a 'Content Management System'. Content Management System is a comprehensive digital platform designed to simplify the creation, management, and delivery of content while fostering collaboration among users. This system is tailored to meet the needs of educational institutions, organizations, and individuals by offering an efficient, scalable, and user-friendly solution. Gyanshristi integrates advanced features such as voice recognition or hands-free content creation and text-to-speech for inclusive content consumption, enhancing accessibility and inclusivity for users with diverse needs and preferences. This system provides users, including viewers, authors, editors, and administrators, with robust tools for content creation, category management, and structured interaction. Admin Manages all user roles, and has full control over all system functions; Editor Oversees the management of Viewers and Authors, as well as content, categories, and Editors also approve requests from Viewers to become Author; Author create content, manage categories; Viewer can search, view, download content, and provide comments and can also apply to become Authors and contribute content upon approval. This project focuses on the development and implementation of an advanced Content Management System aimed at enhancing the efficiency, accessibility, and user experience of content management processes. With its focus on modern web technologies and AI-powered features, Gyanshristi addresses the evolving needs of content creators and managers, delivering a streamlined and intuitive platform that transforms how content is created, managed, and consumed.

This system is implemented using; MongoDB for database and HTML, Tailwind CSS and Nextjs framework and TypeScript for frontend offering a responsive and dynamic user interface. Node JS, JavaScript for backend and Express for backend framework.

Gyanshristi is designed to be a versatile solution capable of handling various types of content, including textual articles, PDF documents, multimedia files, and videos, ensuring a rich and diverse repository for its users. With a focus on user experience, Gyanshristi emphasizes an intuitive interface that minimizes the learning curve for new users while providing advanced tools and functionalities for power users. With its futuristic design and powerful capabilities, Gyanshristi not only meets current industry standards but also sets a benchmark for the future of content management system.

TABLE OF CONTENT

<i>SUPERVISOR'S RECOMMENDATION</i>	<i>iii</i>
<i>CERTIFICATE OF APPROVAL</i>	<i>iv</i>
<i>ACKNOWLEDGMENT</i>	<i>v</i>
<i>EXECUTIVE SUMMARY</i>	<i>vi</i>
LIST OF FIGURES	1
LIST OF ABBREVIATIONS	4
CHAPTER I: INTRODUCTION.....	5
1.1 Introduction	5
1.2 Background of the study	5
1.3 Problem Statement and Motivation.....	6
1.4 Objectives.....	6
1.4.1 Primary Objectives	6
1.4.2 Secondary Objectives	6
1.5 Project Scope and Direction.....	7
1.6 Limitations	7
1.7 Report Organization	7
CHAPTER II: REQUIREMENT ANALYSIS	8
2.1 Literature Review	8
2.2 Problem Definition.....	9
2.3 Proposed Solution	10
2.4 Requirement Analysis	10
2.5 Feasibility Study.....	14
2.5.1 Economic Feasibility Study.....	14
2.5.2 Technical Feasibility Study	15
2.5.3 Operational Feasibility Study	17
2.5.4 Schedule Feasibility Study	19
2.6 Structuring System Requirements.....	20
2.6.1 System Flowchart	20
2.6.2 Use Case Diagram	22

2.6.2 ER Diagram	48
2.6.3 Data Flow Diagram	49
2.6.6 Activity Diagram	51
CHAPTER III: SYSTEM DESIGN.....	55
3.1. System Architecture and Overview.....	55
3.2. System Design.....	56
3.3. Interface Design	56
3.4. Database Schema.....	59
CHAPTER IV: IMPLEMENTATION AND TESTING	64
4.1 Implementation and Overview	64
4.2 Tools Used.....	65
4.2.1 System Design Tools	65
4.2.1 Collaboration Tools	65
4.2.1 Front End Tools	66
4.2.2 Back End Tools	67
4.2.2 Testing Tools	68
4.3 Testing.....	69
4.3.1 Component Testing.....	69
4.3.2 API Testing.....	78
4.3.3 System Testing	86
CHAPTER V: CONCLUSION AND RECOMMENDATION	100
5.1. Conclusion.....	100
5.2. Recommendation.....	100
5.3. References	101

LIST OF FIGURES

Figure 2.1: Gantt Chart (Project Timeline).....	19
Figure 2.2: System Flowchart 1	21
Figure 2.3: System Flowchart 2	22
Figure 2.4: Use Case Diagram of Admin.....	23
Figure 2.5: Use Case Diagram of Editor.....	24
Figure 2.6: Use Case Diagram of Author	25
Figure 2.7: Use Case Diagram of Viewer	26
Figure 2.8: Entity Relationship Diagram	49
Figure 2.9: DFD Level 0	50
Figure 2.10: DFD Level 1	50
Figure 2.11: Activity Diagram of Viewer.....	51
Figure 2.12: Activity Diagram of Author	52
Figure 2.13: Activity Diagram of Editor.....	53
Figure 2.14: Activity Diagram of Admin	54
Figure 3.1: System Architecture	55
Figure 3.2: Implementation architecture of Content Management System	56
Figure 3.3: Homepage of Viewer.....	57
Figure 3.4: Category Wise Content of Viewer	57
Figure 3.5: Sign up Page of Non-registered User	58
Figure 3.6: Login Page.....	58
Figure 3.7: Dashboard Page of Admin	59
Figure 4.1: Waterfall Model	64

LIST OF TABLES

Table 2.1: Cost estimation table.....	14
Table 2.2: SDLC Phases Duration	20
Table 2.3: Use Case Template of Login	27
Table 2.4: Use Case Template of Add User	28
Table 2.5: Use Case Template of Update User.....	29
Table 2.6: Use Case Template of Delete User	30
Table 2.7: Use Case Template of View User.....	31
Table 2.8: Use Case Template of Approve Viewer	32
Table 2.9: Use Case Template of Add Content	33
Table 2.10: Use Case Template of Update Content.....	34
Table 2.11: Use Case Template of Delete Content.....	35
Table 2.12: Use Case Template of Approve Content	36
Table 2.13: Use Case Template of View Content.....	37
Table 2.14: Use Case Template of Add Comment	38
Table 2.15: Use Case Template of View Comment	39
Table 2.16: Use Case Template of Delete Comment.....	40
Table 2.17: Use Case Template of Logout	41
Table 2.18: Use Case Template of Register.....	42
Table 2.19: Use Case Template of Apply as Author	43
Table 2.20: Use Case Template of Add Category	44
Table 2.21: Use Case Template of Update Category.....	45
Table 2.22: Use Case Template of Delete Category	46
Table 2.23: Use Case Template of View Category.....	47
Table 4.1: Component testing of Login.test.tsx	69
Table 4.2: Component testing of Register.test.tsx	70
Table 4.3: Component testing of ContentTable.test.tsx.....	71
Table 4.4: Component testing of CategoryTable.test.tsx.....	72
Table 4.5: Component testing of ShowPdf.test.tsx.....	73
Table 4.6: Component testing of ShowPost.test.tsx	74
Table 4.7: Component testing of ShowVideo.test.tsx.....	75
Table 4.8: Component testing of Logout.test.tsx	77

Table 4.9: API Testing	78
Table 4.10: System Testing of Admin	86
Table 4.11: System Testing of Editor	91
Table 4.12: System Testing of Author	94
Table 4.13: System Testing of Registered Viewer	97
Table 4.14: System Testing of Non-registered Viewer	98

LIST OF ABBREVIATIONS

AI: Artificial Intelligence
API: Application Programming Interface
CI/CD: Continuous Integration/Continuous Delivery
CLI: Command Line Interface
CMS: Content Management System
CSS: Cascading Style Sheet
DB: Database
DFD: Data Flow Diagram
ERD: Entity Relationship Diagram
GUI: Graphical User Interface
HTTP: Hypertext Transfer Protocol
JS: JavaScript
JWT: JSON Web Token
OTP: One Time Password
PDF: Portable Document Format
ROI: Return on Investment
SDLC: Software Development Life Cycle
SEO: Search Engine Optimization
SQL: Structured Query Language
UML: Unified Modeling Language

CHAPTER I: INTRODUCTION

1.1 Introduction

Gyanshristi is a robust and innovative Content Management System (CMS) designed to simplify content creation, management, and delivery while fostering collaboration among users. This system plays a crucial role in enabling organizations, institutions, and individuals to efficiently manage digital information without requiring advanced technical skills.

This project aims to provide content to every user whoever is involved to get knowledge by using technology. It is devoted to providing the content creation and content delivery features in a more managed and entertaining way. This platform is tailored to provide an engaging and inclusive experience for diverse users by integrating advanced features such as voice recognition and text-to-speech, making content creation and interaction seamless and accessible. The voice recognition feature allows users to interact with the system through voice commands, while the text-to-speech functionality provides an auditory way to consume content, making the system inclusive for users with different preferences and needs. Now, the school and any other institutions can use AI-based CMS with minimal cost. This project is focused on enhancing the user experience by streamlining the process of content management. By incorporating AI technologies, it is easy to simplify interactions and ensure that content is easily accessible to all users.

Our system has four types of accessing modes each with specific roles and permissions to ensure an organized and efficient content management process. They are: -

- Admin
- Author
- Editor
- Viewer

1.2 Background of the study

A Content Management System is a digital platform designed to empower users to create, manage, and share content seamlessly in a collaborative digital environment. As digital technologies continue to transform how information is shared and consumed, Content Management Systems (CMS) have become essential tools for efficiently organizing, managing, and delivering content. Traditionally, content creation and management required extensive manual effort, technical expertise, and significant resources for structuring and

maintaining information. However, the emergence of CMS platforms has revolutionized these processes by providing flexible, user-friendly, and scalable solutions for individuals and organizations alike. Content Management Systems (CMS) have revolutionized the way digital content is created, managed, and delivered. From the early days of static platforms like Jekyll to the modern, dynamic systems such as WordPress, CMS technology has evolved to offer responsive designs, multilingual support, and enhanced interactivity. However, existing systems often fail to address critical user needs, such as accessibility for visually impaired users, seamless voice-based interactions, and offline content access.

1.3 Problem Statement and Motivation

In this time, students are often bored to read from the same textbooks and it is very difficult to find the right content from the Google like search engine. Finding the appropriate and right content according to their syllabus is a huge task. Similarly, teachers are not able to provide note to every student and writing on the board could be really painful. While students cannot just sit and read the notes given by the teachers. The collaboration and interaction about the educators and learners are a huge problem during content creation and content delivery. Other external persons are not able to provide some help to the students by providing note to each and every student.

1.4 Objectives

1.4.1 Primary Objectives

- To provide content in a managed way to different entities according to user roles.
- To create content through voice recognition and read content through text-to-speech technology.

1.4.2 Secondary Objectives

- To manage users with different accountable features.
- To provide feedback or clear doubts to the contents through comments.
- To design a responsive and intuitive user interface that works well on various devices.
- To implement search functionalities.
- To implement search functionalities.

1.5 Project Scope and Direction

The main intention of this project is to solve the issues encountered during the traditional content authoring and delivery system where it was completely based on the paper works. Some institutions are using the web applications for this but there occurs issue of ease in content creation and content delivery. In this project, a web application can be used to create and provide contents using voice recognition for content creation and text-to-speech for content delivery. Not only that, students can also ask their doubts through the comments.

1.6 Limitations

- The targeted groups for GyanShristi mainly includes colleges, schools and similar educational institutions.
- It cannot provide accurate voice recognition at the beginning.
- The admin, editor, and author side web pages are not responsive. So, they need desktop or laptop for a better experience.
- Teachers must have good English-speaking skills.

1.7 Report Organization

Chapter I: This chapter explains the overview, introduction, problem statement, objectives, scope and limitation of the system.

Chapter II: This chapter covers all the history, methods, requirement specification and feasibility analysis and structured system requirements.

Chapter III: Design of Content Management System project is explained in detail with all the necessary diagrams and brief functionality.

Chapter IV: Process of implementation and testing is described along with all the tools used for development.

Chapter V: Conclusion and future scope of the application are explained.

CHAPTER II: REQUIREMENT ANALYSIS

2.1 Literature Review

Content Management system is a software for the creation, organization, and distribution of digital content in a digital aspect. This system provides authors with an opportunity to publish content and helps users reach out to that content in a single platform. Content Management System has become an important platform for managing content in an organized way and delivering users with specific content. With the growth of tools and technologies, there is a global change in how users are receiving content. From books to pdf files, and pdf files to videos, users expect the content to be delivered in a more appealing form. As a result, there is a need for an efficient and effective content management system that addresses the needs of the users. This literature review aims to provide a thorough overview of key concepts, evolutions, challenges, and trends of the content management system.

There is a drastic change in the evolution of the Content Management System. The Content Management System evolved with the development of the first static CMS called Jekyll. Jekyll is a simple CMS that helps in generating static content that does not require any database and server-side scripts. As the pages are static in Jekyll, it loads fast. Due to the blooming of the large content and widening of the business, dynamic content becomes obvious. The development of the Video Text Management System provided an opportunity for the user to experience the dynamic content. Many open-source dynamic Content Management Systems like Wix, Drupal, Joomla, and WordPress [1] came to the hype because they provided a platform to create customizable content with rich user interactivity. WordPress is now very popular for its flexibility and user-friendly interface. It stores the content in a database and retrieves the content easily and dynamically. WordPress provides a theme that is responsive to many devices. It provides regular updates to avoid any kind of susceptibility. It also provides multilingual support for supporting users in different languages. [2] Similarly, the platform of the content management system is evolving continuously with underway enhancement of content management, user interactivity, security, and responsiveness.

Although these Content Management Systems provide user-friendly content with responsive design, Content Management Systems are still not able to address the problem of blind users. Up to date, no CMS is friendly to blind users. [3] In addition, there is a trend

in people that they like to search through voice. They don't like to type. Excessive typing may result in strained muscles, eye strain, and poor posture. That's why users try to avoid typing. The website called listening.io accepts the input texts from the users and converts them into the audio file that reads out the text. [4] The development of this website sets the proof that users are now in need of a Content Management System that drifts the CMS towards a text-to-speech converter. Though this is the era of the internet, users are not always accessible on the internet. Users want to download the content and access the content even if it does not connect them to the internet. Likewise, users want to interact with the author or editor easily whenever they are using a CMS. A website like GeeksforGeek [5] does not give the facilities for downloading the content and interacting with the author or editor. Although Wikipedia allows users to interact with the author or editor, the process of interacting is long and difficult.

Floating throughout the evolution of the Content Management System, we are planning to make an SEO-friendly Content Management System that will fulfill the shortcomings of the previous CMS. Our Content Management System will allow users to create, post, and organize content that will be in the form of text, images, and video by strictly addressing the needs of blind users. Our CMS will use a speech-to-text converter that allows the users to write content through the voice command. In the similar way, this system will include a text-to-speech converter that will be helpful for blind users to read the pdf files. This system will be friendly to blind people, which will help in taking education one step closer towards inclusive education. Likewise, users will download and access the PDF files even in the absence of the internet. This will help users to access the content from anywhere at any time. Our system will provide rich interactivity by providing users with a facility to add a comment and interact with the author and editor. This system provides an opportunity for the users to apply for the author and can create and post content like an author after the editor or admin has accepted the request. This system will store the content in the database and retrieve it dynamically. Overall, our Content Management System will address diverse needs, prioritize accessibility, and encourage user engagement through innovative features.

2.2 Problem Definition

In this time, students are often bored to read from the same textbooks and it is very difficult to find the right content from the Google like search engine. Finding the appropriate and right content according to their syllabus is a huge task. Similarly, teachers are not able to provide note to every student and writing on the board could be really painful. While

students cannot just sit and read the notes given by the teachers. The collaboration and interaction about the educators and learners are a huge problem during content creation and content delivery. Other external persons are not able to provide some help to the students by providing note to each and every student.

2.3 Proposed Solution

Our proposed solution for a Content Management System is to develop a web-based platform that addresses key challenges in content creation, delivery, and collaboration for educators, students, and contributors. To enhance accessibility, the system incorporates voice-based input, streamlining content creation and allowing educators to focus on delivering high-quality educational materials. This transition reduces time spent on administrative tasks while supporting students with diverse learning needs.

To promote self-paced learning, the platform allows students to listen to course content during their free time, promoting self-paced learning, reinforcing concepts, and improving retention. Teachers can efficiently upload and share notes, videos, and other learning resources, ensuring that students can easily access syllabus-specific content without the distraction of irrelevant materials from generic search engines. The system also fosters collaboration through interactive features, enabling users to comment, interact with authors, and apply to contribute content after approval from editors or admins.

Additionally, it supports offline learning by allowing users to download notes and PDFs, ensuring continuity even without internet access. With tools for category and user management, as well as a focus on voice-based innovation and interactivity, this Content Management System transforms the educational experience into a dynamic, engaging, and accessible journey for all users.

2.4 Requirement Analysis

In our project, we have collected a list of documents with sufficient and necessary requirements for the project development. To derive the requirements, we have done better understanding of the products under the development which we achieved through detailed and continuous communications with the project team throughout the software development process.

2.4.1 Requirement Identification

2.4.1.1 Study of Existing System

GyanShristi: Content Management System is a web application to automate the content creation and delivery process easier, faster, and interactive. The purpose of this system is to provide a better environment for students to learn.

2.4.1.2 Requirement collection

There are two types of requirement collection i.e. functional and non-functional requirement.

A. Functional Requirement

1. Viewer

1.1 Search for the Content

- 1.1.1 Search based on categories. Each content has a category which might have sub categories.
- 1.1.2 Search for the result based on specific keyword of the heading.
- 1.1.3 Search the content based on voice recognition.
- 1.1.4 View the contents even without logging in.
- 1.1.5 Listen to the text content through text to speech.

1.2 Provide Comment

- 1.2.1 Create an account and comment on the content or post to provide feedback.
- 1.2.2 Reply to the comments on the posts/contents.

1.3 Manage the content

- 1.3.1 Apply as an author and wait for approval from author or editor.
- 1.3.2 Upload contents which need approval from admin or editor after approval.

1.4 Apply as Author

- 1.4.1 Apply as author to create posts/contents

1.5 Login

- 1.5.1 Login into the system.

1.6 Register

- 1.6.1 Register to use different functionalities.

1.7 Logout

- 1.7.1 Viewers can logout from the system.

2. Author

2.1 Manage the contents

- 2.1.1 Add categories to manage the content properly according to category and sub category
- 2.1.2 Upload the contents which needs approval from admin or editor.
- 2.1.3 Update the contents and needs approval from admin or editor.
- 2.1.4 Create content based on voice recognition.

2.2 Provide Comment

- 2.2.1 Create an account and comment on the content or post to provide feedback.
- 2.2.2 Reply to the comments on the post/comments.

2.3 Login

- 2.3.1 Login into the system.

2.4 Register

- 2.4.1 Register to use different functionalities.

2.5 Logout

- 2.5.1 Viewers can logout from the system.

3. Editor

3.1 Manage the contents

- 3.1.1 Add categories to manage the content properly according to category and sub category
- 3.1.2 Approve or unapproved content written by the author.
- 3.1.3 Create content based on voice recognition.
- 3.1.4 Upload contents without approval.
- 3.1.5 Update the contents without approval.

3.2 Create and manage author

- 3.2.1 Create authors.
- 3.2.2 Update authors.
- 3.2.3 Delete authors.
- 3.2.4 Approve viewer to be author.

3.3 Provide Comment

- 3.3.1 Create an account and comment on the content or post to provide feedback.
- 3.3.2 Reply to the comments on the post/comments.

3.4 Login

3.4.1 Login into the system.

3.5 Register

3.5.1 Register to use different functionalities.

3.6 Logout

3.6.1 Viewers can logout from the system.

4. Admin

4.1 Manage users

4.1.1 Create every user in the system.

4.1.2 View users list.

4.1.3 Update every user in the system.

4.1.4 Delete every user in the system.

4.1.5 Approve the viewer to be the author.

4.2 Manage the content

4.2.1 Create content.

4.2.2 Update content.

4.2.3 Delete content.

4.2.4 View content.

4.2.5 Approve contents written by the authors.

4.2.6 Add categories to manage the content properly according to category
and sub category

4.2.7 Create content based on voice recognition.

4.3 Provide Comment

4.3.1 Comment on the content or post to provide feedback.

4.3.2 Reply to the comment on the posts/contents.

4.4 Login

4.4.1 Login into the system

4.5 Register

4.5.1 Register to use different functionalities

4.6 Logout

4.6.1 Viewers can logout from the system.

B. Non-Functional Requirement

- **Performance:** The performance of the website depends upon the system specification and network strength.
- **Security:** The details of the users are well managed and accountable to every user.
- **Scalability:** The system can be extended later due to well managed codes and documents.
- **Maintainability:** The system is easy to maintain.
- **Accountability:** Every user has their own responsibilities which are tracked.
- **Supportability:** The system is supported by every device.

2.5 Feasibility Study

2.5.1 Economic Feasibility Study

Economic feasibility evaluates the financial aspects of the “GyanShristi” CMS project to determine whether the benefits cross the costs or not.

2.5.1.1 Cost Analysis

Table 2.1: Cost estimation table

SN	Category	Estimated Cost (in Rs)	Frequency
1	Paper print cost	5,000	One time
2	Development charge per person	3*5000=15,000	One time
3	Hosting and Domain	500*12=6,000	Annual
4	User support and Maintenance	5,000	Annual
5	Miscellaneous Expense (Training, Minor Fixing, Travel)	3,100	Annual
	Total Cost	33,500	

The total estimated cost for implementing and maintaining the project is Rs. 33,500, covering one-time expenses such as paper printing and development charges, alongside annual costs for hosting, user support, and miscellaneous expenses. This budget ensures efficient project execution and ongoing support.

2.5.1.2 Payback Period

Payback period is a financial metric used to assess the time it takes for an investment to recoup its initial cost through the cash flows it generates.

$$\begin{aligned}\text{Net annual cash inflow} &= \text{Total Annual Cash Inflows} - \text{Total Annual Cash Outflow} \\ &= 60,000 - 10,000 \\ &= 50,000\end{aligned}$$

$$\begin{aligned}\text{Payback period} &= \text{initial investment} / \text{net annual cash inflow} \\ &= 33,500 / 50,000 \\ &= 8.1 \text{ months}\end{aligned}$$

The payback period for the project is calculated to be 8.1 months, indicating that the initial investment of Rs. 33,500 will be recovered through net annual cash inflows of Rs. 50,000 within this time frame. This short payback period demonstrates the project's financial viability and quick return on investment.

2.5.1.3 Return on Investment (ROI)

The return on investment (ROI) is a financial metric used to evaluate the profitability of an investment relative to its cost. It measures the percentage return or profit generated from an investment over a specific period of time.

$$\text{ROI} = (\text{Net profit} / \text{initial investment}) * 100$$

Here,

$$\begin{aligned}\text{Net profit} &= (\text{Net annual cash inflow} - \text{Initial Investment}) \\ &= 50,000 - 33,500 \\ &= 16,500\end{aligned}$$

$$\begin{aligned}\text{So, ROI} &= \text{Net profit} / \text{initial investment} * 100\% \\ &= 16,500 / 33,500 * 100\% \\ &= 49.25\%\end{aligned}$$

The return on investment (ROI) for the project is calculated to be 49.25%, indicating that the project generates a profit of nearly half its initial investment. This high ROI demonstrates the project's strong profitability and its potential to deliver significant financial benefits over time.

2.5.2 Technical Feasibility Study

The technical feasibility of “GyanShristi” give us the information where the system is technically feasible. It lets us know whether we have enough resources or not to complete the project.

1. Technology Stack

Front-end Technologies

- **Languages:** HTML5, CSS3, Tailwind CSS, JavaScript
- **TypeScript:** For starting static typing to JavaScript, improving code quality and maintainability.
- **Frameworks:** NextJS to simplify the server-side rendering and static site generation, enhancing performance and SEO.

Back-end Technologies

- **Language:** JavaScript
- **Framework:** Express.js for server-side development
- **API architecture:** Restful APIs to ensure modularity, scalability and ease of integration.

Database

- **Type:** NoSQL
- **Database System:** MongoDB for flexible and schema less data management
- **Database Management Tool:** MongoDB GUI Compass for a user-friendly interface to explore and manipulate data.

Hosting and Development:

- **Cloud Provider:** Vercel for hosting the website that offers automatic deployment.

2. Team Expertise and Resources

Team Expertise

- **Front-end Development:** Proficiency in NextJS and TypeScript
- **Back-end Development:** Experience with Express.js
- **Database Administration:** Familiarity with MongoDB and management tools

Development Tools

- **IDE:** Visual Studio Code
- **Version Control:** Git and GitHub for code management and collaboration

3. Technical Demonstrations

- User authentication with JWT Tokens
- User validation with ZOD validation
- Encrypting and Decrypting passwords with bcrypt
- Implementing voice recognition with react-speech-recognition

- Implement text-to-speech with Web speech
- Basic Froala Text editor integrated into the content creation page.

4. System Architecture

- Use of API gateway for routing and managing API requests.
- Separate controllers for user management, content management, and media handling

5. Integrated Tools

- **Firestore:** Firestore for handling the email verification.

6. Nonfunctional Tools

- Uses Jest framework for testing the code for improved security.
- Uses bcrypt encryption algorithm for hashing the password for enhanced security.
- Uses token authentication mechanism. (JWT)

7. Technical Risks and Mitigation

Technical Risks

- Performance issues with more user load.

Mitigation Measures

- Frequent performance testing and optimizations

The technical feasibility study for “GyanShristi” CMS confirms that the project is technically feasible. The chosen technology stack aligns with the project requirements, and the team has necessary skills. Key technical challenges have been identified, ensuring the project can proceed successfully.

2.5.3 Operational Feasibility Study

1. Introduction

Operational feasibility evaluates whether the “GyanShristi” CMS can be effectively integrated into the daily operations of schools and colleges, ensuring it meets the needs of teachers, students, administrations or any other users related to this CMS.

2. User Needs and Acceptance

Stakeholder Analysis:

- **Teachers:** Needs an easy-to-use platform for creating and sharing content, including voice recognition for creating content.

- **Students:** Require engaging content, the ability to listen to content, and features to interact with content, like commenting,
- **Administrators:** Needs content approval system for managing the content in a better way.
- **Other Users:** Need to participate in the content creation program of CMS as an author.

User Involvement:

- Conduct surveys and focus groups with teachers and students to gather feedback on the propose features.
- Gather feedback from every category of users to enhance requirements.

Training and Support:

- Provide comprehensive training for the teachers, students, and administrators to ensure they are comfortable with the system.
- Create tutorial videos for other users who are users of the system.
- Provide technical support and user assistance through the help desk.

3. System Integration:

- Analyze the existing content creation and delivery process in different institutions.
- Ensure CMS integrates smoothly with the current system of different institutions.
- Evaluate any necessary data migration from current systems to the new CMS.

4. Operational Impact:

- Assess how the new system will change daily routines for users.
- Implement a change management plan to help users transition to the new system. This plan includes communications, training sessions, and support.

5. Resource Requirement:

- Assess the infrastructure required for CMS such as servers, storage, and network bandwidth.
- Ensure there are backups in one place.

6. Operational Costs:

- Estimated cost provides benefits as per the calculation of ROI.

The operational feasibility indicates that the “GyanShristi” can be effectively integrated into the existing educational environments. With appropriate training, support and change

management strategies, the system can meet the needs of teachers, students, administrators, and other users.

2.5.4 Schedule Feasibility Study

Time planning deals with the measures to plan the available time within the time frame. If a project takes more than the time frame, it is likely to get rejected. Utilizing tools like Gantt charts facilitates effective time management by visually organizing tasks, dependencies, and deadlines. This ensures timely completion of project milestones and enhances the project's chances of meeting client expectations within the designated time frame.

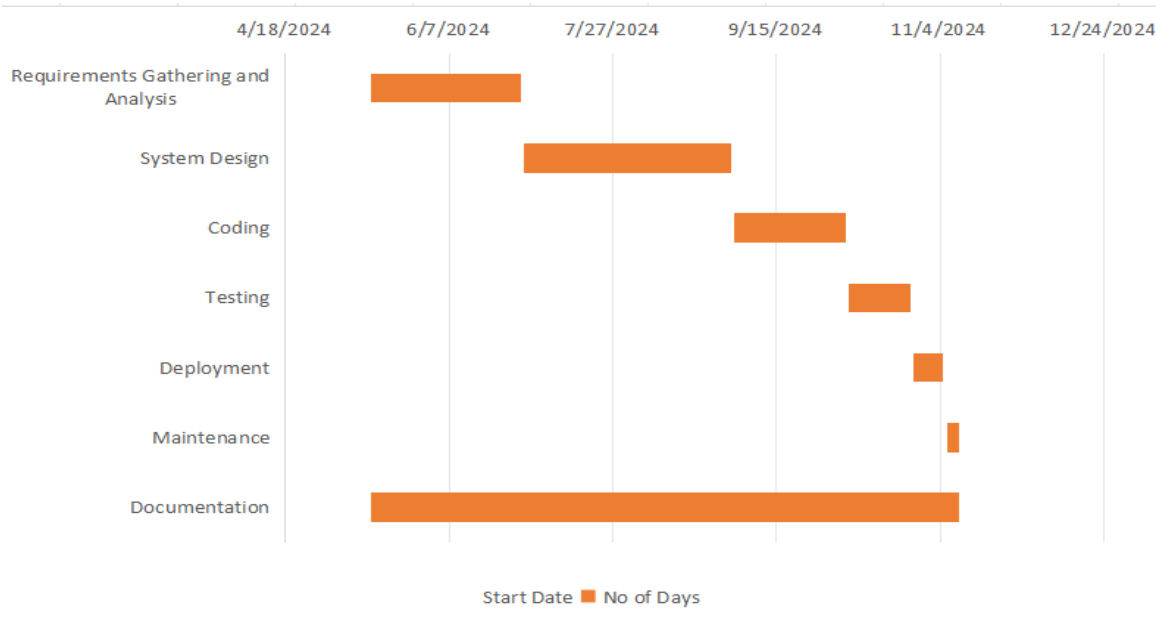


Figure 2.1: Gantt Chart (Project Timeline)

Presenting project phases and their durations in a table clarifies the sequential timeline, aiding in planning, resource allocation, and tracking progress efficiently within a structured format.

Table 2.2: SDLC Phases Duration

Description	Number of Days
Requirements Gathering and Analysis	46
System Design	63
Coding	34
Testing	19
Deployment	9
Maintenance	4
Documentation	180

2.6 Structuring System Requirements

Structuring system requirements is crucial in software development and systems engineering, as it ensures clarity and mutual understanding among all stakeholders. This practice reduces ambiguity and miscommunication, streamlines task tracking, and facilitates prioritization.

Moreover, structured requirements aid in early risk identification and management, simplify change processes, and promote system consistency. They also serve as a basis for legal agreements, ensuring all parties understand the deliverables.

In essence, well-structured requirements are fundamental to delivering high-quality systems on time and within scope.

2.6.1 System Flowchart

A system flowchart is a graphical representation of the processes and workflows within a system, illustrating how data flows and is processed. It uses standardized symbols to depict inputs, outputs, processes, decisions, and storage, providing a clear and concise overview of the system's functionality. System flowcharts are used to analyze, design, and document systems, making them easier to understand and troubleshoot.

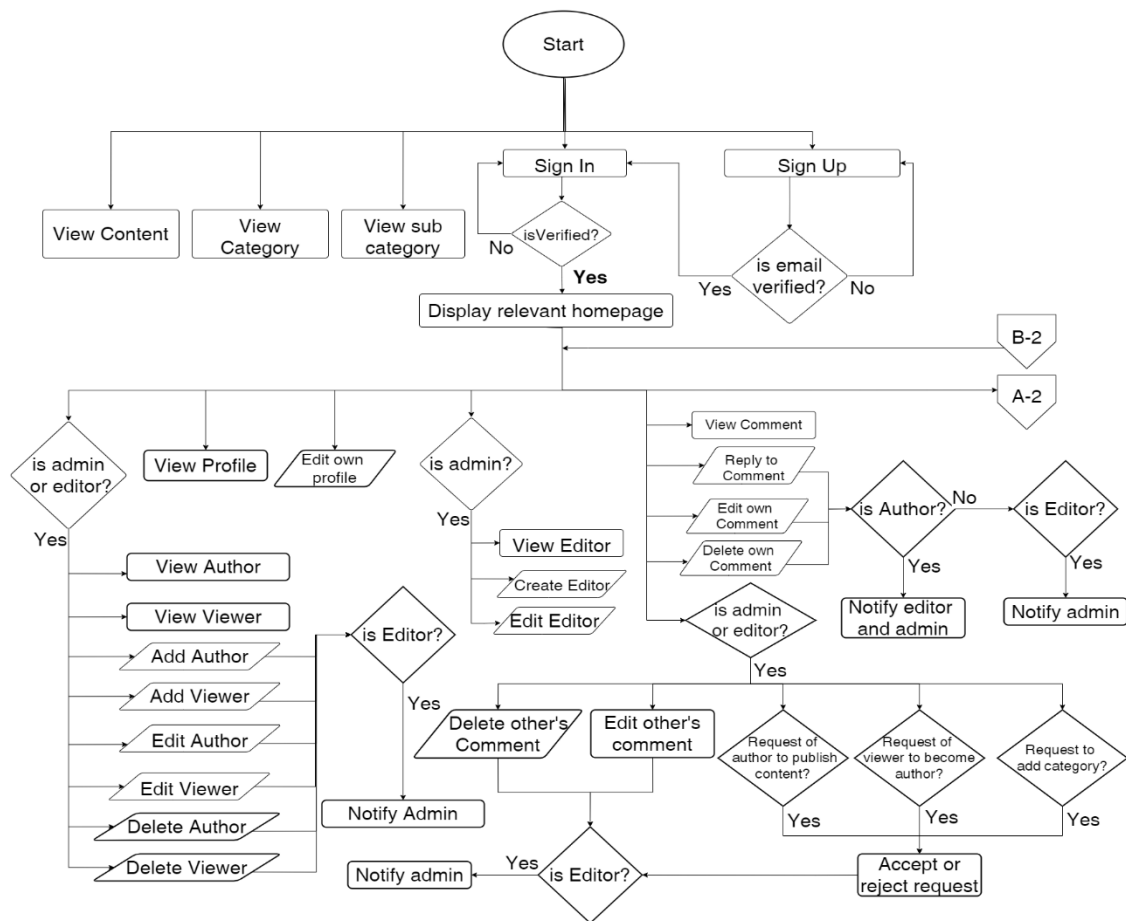


Figure 2.2: System Flowchart 1

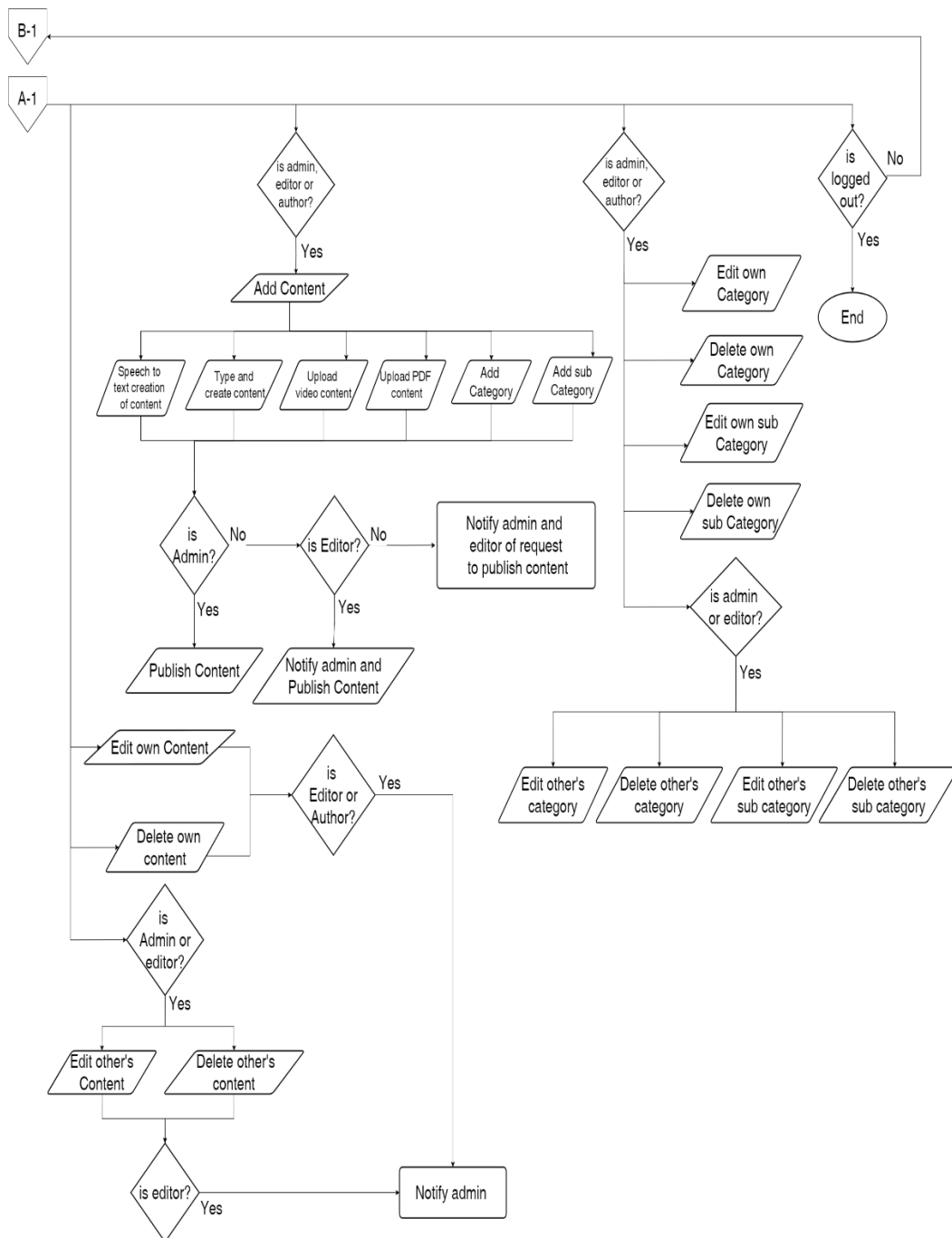


Figure 2.3: System Flowchart 2

2.6.2 Use Case Diagram

A use case diagram is a UML tool that shows the interactions between users (actors) and a system, highlighting key functions (use cases) and their relationships to define system requirements.



Figure 2.4: Use Case Diagram of Admin



Figure 2.5: Use Case Diagram of Editor



Figure 2.6: Use Case Diagram of Author



Figure 2.7: Use Case Diagram of Viewer

Table 2.3: Use Case Template of Login

Use Case Id:	1.0
Use Case Name:	Login
Created By:	Seezan Shrestha
Date Created:	January 3,2024
Actor(s):	Admin, Author, Editor, Viewer
Description:	This use case allows users to login into the system and perform different functions according to the user roles. All the users have to enter their unique username and password.
Precondition:	Users have to validate their account.
Post-condition:	System displays the relevant homepage.
Normal Courses:	<ul style="list-style-type: none"> • Users enter the username, password and choose the user role. • System validates the username, password and user role. • System verifies the username, password and user role. • System displays the relevant homepage. • The use case ends.
Alternative Courses:	<ol style="list-style-type: none"> 1. Upon missing username, password or user role: <ul style="list-style-type: none"> • System prompts for empty username, password, or user role. • The use case resumes from the previous step. 2. Upon entering less than 6-character password: <ul style="list-style-type: none"> • System prompts a message to enter a password having more than 6 characters. 3. Upon entering more than 32-character password: <ul style="list-style-type: none"> • System prompts a message to enter a password having less than 32 characters.
Exceptions:	Users may not login.
Includes:	None
Priority:	High
Frequency of use:	24 hours a day
Business Rule:	Users must login through a registered and verified account.

Special Requirement:	None
Assumptions:	Nil

Table 2.4: Use Case Template of Add User

Use Case Id:	2.0
Use Case Name:	Add User
Created By:	Seezan Shrestha
Date Created:	January 3,2024
Actor(s):	Admin, Editor
Description:	This use case allows the admin to add editors into the system by filling up their details. Whereas both editor and admin can add the author into the system.
Precondition:	Admin or Editor are authenticated and given permission to add users according to their role.
Post-condition:	<ul style="list-style-type: none"> • The added user is registered and accessible to the system. • User roles are assigned correctly.
Normal Courses:	<ul style="list-style-type: none"> • Admin or editor log into the system. • They click on the user on the navigation bar and click on the add user. • They enter the required details and click on the add button. • System validates the data. • The user is added to the system according to their role.
Alternative Courses:	Upon missing any detail, the system prompts for necessary information.
Exceptions:	System failure prevents the add user process.
Includes:	None
Priority:	High
Frequency of use:	24 hours a day
Business Rule:	Newly added users must have a unique username and password.

Special Requirement:	None
Assumptions:	The system is operational and accessible during add user processes.

Table 2.5: Use Case Template of Update User

Use Case Id:	3.0	
Use Case Name:	Update User	
Created By:	Seezan Shrestha	Last updated by:
Date Created:	January 3,2024	Date last updated:
Actor(s):	Admin, Editor	
Description:	This use case allows the admin to update any information of the editor. Whereas both admin and editor can update the information of the author and viewer.	
Precondition:	Admin or editor are authenticated and given permission to update users according to their role.	
Post-condition:	The user’s information is successfully updated.	
Normal Courses:	<ul style="list-style-type: none">● Admin or editor log into the system.● They click on the user on the navigation bar and click the update button of the required user.● Update the required information.● System validates and saves the updated information.	
Alternative Courses:	Upon missing any detail, the system prompts for necessary information.	
Exceptions:	Nil	
Includes:	None	
Priority:	Medium	
Frequency of use:	24 hours a day	
Business Rule:	Updated information must adhere to specified data format or validation rules.	

Special Requirement:	None
Assumptions:	The system is operational and accessible during user information update processes.

Table 2.6: Use Case Template of Delete User

Use Case Id:	4.0		
Use Case Name:	Delete User		
Created By:	Seezan Shrestha	Last updated by:	
Date Created:	January 3,2024	Date last updated:	
Actor(s):	Admin, Editor		
Description:	This use case allows the admin to delete editor, viewer or author. Whereas the editor can only delete the author.		
Precondition:	Admin or editor are authenticated and given permission to delete users according to their role.		
Post-condition:	The deleted user’s information is removed from the system.		
Normal Courses:	<ul style="list-style-type: none">● Admin or editor logs into the system.● They click on the user on the navigation bar and click the delete button of the required user.● System Prompts a confirmation box.● They confirm the action from the confirmation box.● The user is successfully deleted from the system.		
Alternative Courses:	None		
Exceptions:	System failure prevents user deletion.		
Includes:	None		
Priority:	High		
Frequency of use:	24 hours a day		
Business Rule:	Deletion should be confirmed to avoid accidental removal of users.		

Special Requirement:	None
Assumptions:	The system is operational and accessible during user deletion processes.

Table 2.7: Use Case Template of View User

Use Case Id:	5.0		
Use Case Name:	View User		
Created By:	Seezan Shrestha	Last updated by:	
Date Created:	January 3,2024	Date last updated:	
Actor(s):	Admin, Editor		
Description:	This use case allows the admin and editor to view all the information about the editor, viewer and author.		
Precondition:	Admin or Editor have to gain access to their account.		
Post-condition:	System displays the information about users according to the user’s role and access level.		
Normal Courses:	<ul style="list-style-type: none">● Admin, editor or author logs into the system.● They click on the user from the navigation bar.● Admins view all the information about both editors and authors by clicking on the category editor and author.● Editors view all the information about the authors.● Authors view basic details of other authors.		
Alternative Courses:	Upon not getting required information, the system displays an error.		
Exceptions:	System failure prevents displaying user information.		
Includes:	None		
Priority:	Medium		
Frequency of use:	24 hours a day		
Business Rule:	Access to user information is restricted based on user roles and permissions.		

Special Requirement:	None
Assumptions:	The system is operational and accessible during the user viewing processes.

Table 2.8: Use Case Template of Approve Viewer

Use Case Id:	6.0		
Use Case Name:	Approve Viewer		
Created By:	Seezan Shrestha	Last updated by:	
Date Created:	January 3,2024	Date last updated:	
Actor(s):	Admin, Editor		
Description:	This use case allows the admin and editor to approve the request sent by the viewer to be an author.		
Precondition:	Admin and editor get notification of pending requests to become an author.		
Post-condition:	<ul style="list-style-type: none">● The viewer’s status is updated to an author upon approval.● System records the approval action and updated user roles accordingly.		
Normal Courses:	<ul style="list-style-type: none">● Admin or Editor logs into the system.● They click on the notification from the navigation bar and access the pending requests.● Accept or reject the request.● Viewer’s status is updated to an author if the request is accepted.		
Alternative Courses:	If the request has already been approved, display a relevant message.		
Exceptions:	System failure prevents the approval process.		
Includes:	None		
Priority:	Medium		
Frequency of use:	24 hours a day		
Business Rule:	Viewer requests need approval to become authors.		
Special Requirement:	Only admins and editors have the authority to approve viewer requests to be an author.		

Assumptions:	The system is operational and accessible during the approval processes.
---------------------	-------------------------------------------------------------------------

Table 2.9: Use Case Template of Add Content

Use Case Id:	7.0		
Use Case Name:	Add Content		
Created By:	Seezan Shrestha	Last updated by:	
Date Created:	January 3,2024	Date last updated:	
Actor(s):	Admin, Editor, Author		
Description:	This use case allows admin, editor and author to add content like posts, videos and PDFs to the system.		
Precondition:	<ul style="list-style-type: none">● The user initiating the action is authenticated and has necessary permissions to add content.● Suitable storage or space is available to accommodate the new content.		
Post-condition:	The new content is successfully added to the system and accessible to users based on access rights.		
Normal Courses:	<ul style="list-style-type: none">● Admin, Editor or Author logs into the system.● They click on the content from the navigation bar.● They click on the type of content (post, video, PDF) and click on the add content button inside the relevant content.● Creates and submits the content for the approval.● Content is uploaded after approval.		
Alternative Courses:	<ul style="list-style-type: none">● If the content upload fails, provide appropriate error messages and retry options.● If there's insufficient storage space, prompt for additional space or notify administrators.		
Exceptions:	System failure prevents adding content to the system.		
Includes:	None		
Priority:	High		
Frequency of use:	24 hours a day		
Business Rule:	Uploaded content must adhere to specified formats and guidelines.		

Special Requirement:	None
Assumptions:	The system is operational and accessible during the content addition process.

Table 2.10: Use Case Template of Update Content

Use Case Id:	8.0		
Use Case Name:	Update Content		
Created By:	Seezan Shrestha	Last updated by:	
Date Created:	January 3,2024	Date last updated:	
Actor(s):	Admin, Author, Editor		
Description:	This use case allows admin, editor and author to update their own content. Admin has the authority to update other’s content too. Whereas the editor has the authority to update the author's content.		
Precondition:	<ul style="list-style-type: none">● The user initiating the action is authenticated and has necessary permissions to update content.● The content being updated exists in the system and is accessible to the user.		
Post-condition:	The updated content is successfully modified in the system.		
Normal Courses:	<ul style="list-style-type: none">● Admin, Editor or Author logs into the system.● They click on the content from the navigation bar.● They click on the type of content (post, video, PDF) from the category.● Click on the update button on the required content.● Submits the content for approval.● Content is updated after approval.		
Alternative Courses:	If the content update fails, provide appropriate error messages and retry options.		
Exceptions:	System failure prevents updating content.		
Includes:	None		
Priority:	High		

Frequency of use:	24 hours a day
Business Rule:	Updated content must adhere to specified formats and guidelines.
Special Requirement:	Users must have to login.
Assumptions:	The system is operational and accessible during the content update process.

Table 2.11: Use Case Template of Delete Content

Use Case Id:	9.0		
Use Case Name:	Delete Content		
Created By:	Seezan Shrestha	Last updated by:	
Date Created:	January 3,2024	Date last updated:	
Actor(s):	Admin, Editor, Author		
Description:	This use case allows the admin and editor to delete content. Whereas authors can only delete their own content.		
Precondition:	<ul style="list-style-type: none">● The user initiating the action is authenticated and has the necessary permissions to delete content.● The content being deleted exists in the system and is accessible to the user.		
Post-condition:	The deleted content is removed from the system and no longer accessible to the users.		
Normal Courses:	<ul style="list-style-type: none">● Admin, Editor or Author logs into the system.● They click on the content from the navigation bar.● They click on the type of content (post, video, PDF) from the category.● Click the delete button of the required content.● System Prompts a confirmation box.● They confirm the action from the confirmation box.● System removes the content from the system.		
Alternative Courses:	If the content deletion fails, provide appropriate error messages and retry options.		
Exceptions:	<ul style="list-style-type: none">● System failure prevents deleting content.		

	<ul style="list-style-type: none"> ● Lack of necessary permissions prevents deleting content.
Includes:	None
Priority:	High
Frequency of use:	24 hours a day
Business Rule:	Deletion should be confirmed to avoid accidental removal of content.
Special Requirement:	User must have to login
Assumptions:	The system is operational and accessible during content deletion processes.

Table 2.12: Use Case Template of Approve Content

Use Case Id:	10.0	
Use Case Name:	Approve Content	
Created By:	Seezan Shrestha	Last updated by:
Date Created:	January 3,2024	Date last updated:
Actor(s):	Admin, Editor	
Description:	This use case allows the admin, and editor to approve the content uploaded by the authors. Whereas content uploaded by admin and editors does not need any approval.	
Precondition:	<ul style="list-style-type: none"> ● The user initiating the action is authenticated and has the necessary permissions. ● There exist notifications about pending content uploaded by authors requiring approval. 	
Post-condition:	The approved content is marked as verified and becomes accessible to users.	
Normal Courses:	<ul style="list-style-type: none"> ● Admin or Editor logs into the system. ● They click on the notification and access the pending content uploads. ● Selects and approves the content, marking it as verified. 	
Alternative Courses:	<ul style="list-style-type: none"> ● If the content approval fails or encounters issues, provide appropriate error messages and retry options. 	

	<ul style="list-style-type: none"> ● If there's no pending content for approval, display a relevant message.
Exceptions:	System failure prevents the approval of content.
Includes:	None
Priority:	High
Frequency of use:	24 hours a day
Business Rule:	Content uploaded by authors requires approval before being accessible to users.
Special Requirement:	Content uploaded by Admins and Editors does not need approval.
Assumptions:	The system is operational and accessible during the content approval process.

Table 2.13: Use Case Template of View Content

Use Case Id:	11.0		
Use Case Name:	View Content		
Created By:	Seezan Shrestha	Last updated by:	
Date Created:	January 3,2024	Date last updated:	
Actor(s):	Admin, Editor, Author, Viewer		
Description:	This use case allows all the users to view the content according to its types, such as videos, posts and PDFs.		
Precondition:	The requested content exists in the system and is accessible based on user roles and permissions.		
Post-condition:	The requested content is displayed to the user based on their access level.		
Normal Courses:	<ul style="list-style-type: none">● User logs into the system.● Choose the content according to the category.● The system presents the available content of the chosen type of the user.		
Alternative Courses:	If the required content is not found, display a relevant message or suggest similar content.		
Exceptions:	System failure prevents displaying content.		

Includes:	None
Priority:	High
Frequency of use:	24 hours a day
Business Rule:	None
Special Requirement:	None
Assumptions:	The system is operational and accessible during the content viewing process.

Table 2.14: Use Case Template of Add Comment

Use Case Id:	12.0
Use Case Name:	Add Comment
Created By:	Usha Gurung
Date Created:	January 4,2024
Actor(s):	Admin, Editor, Author, viewer
Description:	This Use Case allows admin, editor, registered viewer and author to add comments to the contents.
Precondition:	The user must be logged into the system.
Post-condition:	The comment is successfully added to the selected content.
Normal Courses:	<ul style="list-style-type: none"> • User logs into the system. • User navigates to the content they want to comment on. • User clicks on the “Add Comment” button. • User enters the comment and submits. • System associates the comment with the selected content.
Alternative Courses:	If the user cancels the comment, the system will remain the same .
Exceptions:	If the user’s session expires, they are prompted to log in again.
Includes:	None
Priority:	Medium

Frequency of use:	Daily
Business Rule:	Comments cannot exceed 500 characters.
Special Requirement:	The system should support plain text for comments.
Assumptions:	Users have the necessary permissions to add comments to the content.

Table 2.15: Use Case Template of View Comment

Use Case Id:	13.0
Use Case Name:	View Comment
Created By:	Usha Gurung
Date Created:	January 4,2024
Actor(s):	Admin, Editor, Author, viewer
Description:	This use case allows the user to view comments to the contents.
Precondition:	None
Post-condition:	The user successfully views comments on the selected content.
Normal Courses:	<ul style="list-style-type: none"> • User logs into the system. • User navigates to the content they want to view comments on. • System retrieves and displays existing comments for the selected content.
Alternative Courses:	If there are no comments for the selected content, the system displays a message indicating “no comments are available”.
Exceptions:	If there is an issue retrieving comments, the system displays an error message.
Includes:	None
Priority:	Medium
Frequency of use:	Daily
Business Rule:	None

Special Requirement:	The system should support scroll down if there are a large number of comments.
Assumptions:	None

Table 2.16: Use Case Template of Delete Comment

Use Case Id:	14.0
Use Case Name:	Delete Comment
Created By:	Usha Gurung
Date Created:	January 4,2024
Actor(s):	Admin, Editor, Author, viewer
Description:	This use case allows the user to delete their comments to the contents.
Precondition:	<ul style="list-style-type: none"> • The user must be logged into the system. • The user must have previously added the comment.
Post-condition:	The user successfully deletes their comment on their selected content.
Normal Courses:	<ul style="list-style-type: none"> • User logs into the system. • Users navigate to the content with their existing comment. • User clicks on the “Delete” button next to their comment. • System prompts the user for confirmation. • User confirms the deletion. • System removes the comment from the content.
Alternative Courses:	If the user cancels the delete action , the comments remain unchanged.
Exceptions:	If there is an issue saving the deleting comment, the system displays an error message.
Includes:	None
Priority:	High
Frequency of use:	24 hours a day
Business Rule:	<ul style="list-style-type: none"> • Users can only delete their own comments.

	<ul style="list-style-type: none"> Admin can delete other users' comments.
Special Requirement:	The system should maintain a record of deleted comments for audit purposes.
Assumptions:	Users have the necessary permissions to delete their comments on the content.

Table 2.17: Use Case Template of Logout

Use Case Id:	15.0
Use Case Name:	Logout
Created By:	Usha Gurung
	Last updated by:
Date Created:	January 4,2024
	Date last updated:
Actor(s):	Admin, Editor, Author, viewer
Description:	This use case allows the user to logout of the system.
Precondition:	The user must be currently logged into the system.
Post-condition:	The user is successfully logged out of the system.
Normal Courses:	<ul style="list-style-type: none"> User clicks on the “Logout” button or navigates to the logout option. System logs the user out and redirects to the login page. Session variables and user data are cleared.
Alternative Courses:	If the user cancels the logout action, they remain logged in.
Exceptions:	None
Includes:	None
Priority:	Medium
Frequency of use:	Daily
Business Rule:	None
Special Requirement:	The system should display a confirmation message before logging the user out.

Assumptions:	Users have the necessary permissions to log out, and their session data is successfully cleared upon logout.
---------------------	--------------------------------------------------------------------------------------------------------------

Table 2.18: Use Case Template of Register

Use Case Id:	16.0
Use Case Name:	Register
Created By:	Usha Gurung
Date Created:	January 4,2024
Actor(s):	Viewer
Description:	This use case allows viewer to register
Precondition:	The viewer must not be already registered in the system.
Post-condition:	The viewer is successfully registered in the system.
Normal Courses:	<ul style="list-style-type: none"> • Viewer accesses the registration page. • Viewer clicks on the “Register” button. • System presents the registration form to the viewer. • Viewer fills in the required information (e.g., username, email,password). • Viewer submits the registration form. • System validates the information and creates a new viewer account. • System logs the viewer into the system.
Alternative Courses:	If the viewer cancels the registration, they are returned to the homepage.
Exceptions:	If there is an issue during registration, the system displays an error message.
Includes:	None
Priority:	High
Frequency of use:	One-time (per viewer)
Business Rule:	Username and email address must be unique within the system.

Special Requirement:	The system should send a confirmation email to the viewer upon successful registration.
Assumptions:	<ul style="list-style-type: none"> • The viewer provides valid and unique information during registration. • The system admin is responsible for handling account verification and activation.

Table 2.19: Use Case Template of Apply as Author

Use Case Id:	17.0
Use Case Name:	Apply as author
Created By:	Usha Gurung
Date Created:	January 4,2024
Actor(s):	Viewer
Description:	This use case allows the viewer to apply as author.
Precondition:	<ul style="list-style-type: none"> • The user must be a registered viewer in the system. • The viewer is logged into the system.
Post-condition:	The system receives the application for authorship from the viewer.
Normal Courses:	<ul style="list-style-type: none"> • Viewer logs into the system. • Viewer navigates to the “Apply as Author” section. • Viewer clicks on the “Apply as Author” button. • System prompts the viewer to provide necessary details for the author application. • Viewer fills in the required information (e.g., bio, writing sample) and submits the application. • System records the author's application for review.
Alternative Courses:	If the viewer cancels the application, they are returned to the homepage.
Exceptions:	If there is an issue submitting the application, the system displays an error.
Includes:	None
Priority:	High

Frequency of use:	Infrequent
Business Rule:	The system admin will review and approve author applications.
Special Requirement:	The system should send a confirmation email to the viewer upon successful submission of the author application.
Assumptions:	<ul style="list-style-type: none"> • The viewer has the intention and necessary qualifications to become an author. • The system admin or designated authority is responsible for reviewing and approving author applications.

Table 2.20: Use Case Template of Add Category

Use Case Id:	18.0
Use Case Name:	Add Category
Created By:	Usha Gurung
Date Created:	January 4,2024
Actor(s):	Admin, Editor, Author
Description:	This use case allows the admin, editor and author to add categories for content.
Precondition:	<ul style="list-style-type: none"> • The user must be logged into the system. • The user must have the necessary permissions to add a category.
Post-condition:	The new category is successfully added to the system.
Normal Courses:	<ul style="list-style-type: none"> • User logs into the system. • User navigates to the “Add Category” section. • User clicks on the “Add Category” button. • System requests the user to provide details for the new category (e.g., name,description). • User fills in the required information and submits the form. • System validates and adds the new category to the system.
Alternative Courses:	If the user cancels adding a new category, there will be no changes.
Exceptions:	If there is an issue during category addition (e.g.,duplicate category name), the system displays an error message.

Includes:	None
Priority:	Medium
Frequency of use:	Occasional
Business Rule:	Category names must be unique within the system.
Special Requirement:	The system should support the hierarchical structure of categories (parents-child relationships).
Assumptions:	<ul style="list-style-type: none"> • Users have the necessary permissions to add categories. • The system supports the concept of categories for organizing content.

Table 2.21: Use Case Template of Update Category

Use Case Id:	19.0
Use Case Name:	Update Category
Created By:	Usha Gurung
Date Created:	January 4,2024
Actor(s):	Admin, Editor, Author
Description:	This use case allows the admin, editor and author to update categories.
Precondition:	<ul style="list-style-type: none"> • The user must be logged into the system. • The user must have the necessary permissions to update a category. • There must be existing categories in the system.
Post-condition:	The selected category is successfully updated in the system.
Normal Courses:	<ul style="list-style-type: none"> • User logs into the system. • User navigates to the “Update Category” section. • User selects the category they want to update • User clicks on the “Update Category” button for the selected category. • System loads the existing information for the category into an editable form. • User modifies the category details (e.g., name, description) and submit the form. • System validates and updates the category information.

Alternative Courses:	If the user cancels the update action, the system retains the original category details.
Exceptions:	If there is an issue during category update (e.g., invalid data), the system displays an error message.
Includes:	None
Priority:	Medium
Frequency of use:	Occasional
Business Rule:	Category names must be unique within the system.
Special Requirement:	The system should maintain consistency in the hierarchical structure of categories.
Assumptions:	<ul style="list-style-type: none"> • Users have the necessary permissions to update categories. • There are existing categories in the system to be updated.

Table 2.22: Use Case Template of Delete Category

Use Case Id:	20.0		
Use Case Name:	Delete Category		
Created By:	Usha Gurung	Last updated by:	
Date Created:	January 4,2024	Date last updated:	
Actor(s):	Admin, Editor, Author		
Description:	This use case allows the admin, editor and author to delete categories.		
Precondition:	<ul style="list-style-type: none">● The user must be logged into the system.● The user must have the necessary permissions to delete a category.● There must be existing categories in the system.		
Post-condition:	The selected category is successfully deleted in the system.		
Normal Courses:	<ul style="list-style-type: none">● User logs into the system.● User navigates to the “Delete Category” section.● User selects the category they want to delete.● User clicks on the “Delete Category” button for the selected category.● System prompts the user for confirmation to delete the category.● User confirms the deletion.		

	<ul style="list-style-type: none"> System removes the selected category and associated content.
Alternative Courses:	If the user cancels the delete action, the category remains unchanged.
Exceptions:	If there is an issue during category deletion (e.g., category has associated content), the system displays an error message.
Includes:	None
Priority:	High
Frequency of use:	Occasional
Business Rule:	Categories with associated content cannot be deleted directly; users must first move or delete the associated content.
Special Requirement:	The system should provide an option to reassign or move content before deleting a category.
Assumptions:	<ul style="list-style-type: none"> Users have the necessary permissions to delete categories. There are existing categories in the system to be deleted.

Table 2.23: Use Case Template of View Category

Use Case Id:	21.0
Use Case Name:	View Category
Created By:	Usha Gurung
Date Created:	January 4, 2024
Actor(s):	Admin, Editor, Author, Viewer
Description:	This use case allows users to view categories.
Precondition:	The user must be logged into the system.
Post-condition:	The user successfully views the list of categories.
Normal Courses:	<ul style="list-style-type: none"> User logs into the system. User navigates to the “View Categories” section. System retrieves and displays a list of existing categories.
Alternative Courses:	None

Exceptions:	If there is an issue retrieving categories, the system displays an error message.
Includes:	None
Priority:	Medium
Frequency of use:	Daily
Business Rule:	Categories are organized hierarchically, displaying parent and child relationships.
Special Requirement:	The system should support filtering or sorting options for an enhancing viewing experience.
Assumptions:	<ul style="list-style-type: none"> • Users have the necessary permissions to view categories. • Categories are organized in a hierarchical structure.

2.6.2 ER Diagram

An ER (Entity-Relationship) Diagram is a graphical representation of entities, their attributes, and the relationships between them, used in database design. It helps visualize the data structure, making it easier to understand how data is organized and connected. Entities are represented as rectangles, attributes as ovals, and relationships as diamonds, with lines connecting them to indicate associations. ER diagrams are essential for creating efficient, logical, and well-structured databases by identifying primary keys, foreign keys, and cardinality constraints, ensuring a clear blueprint for implementation.

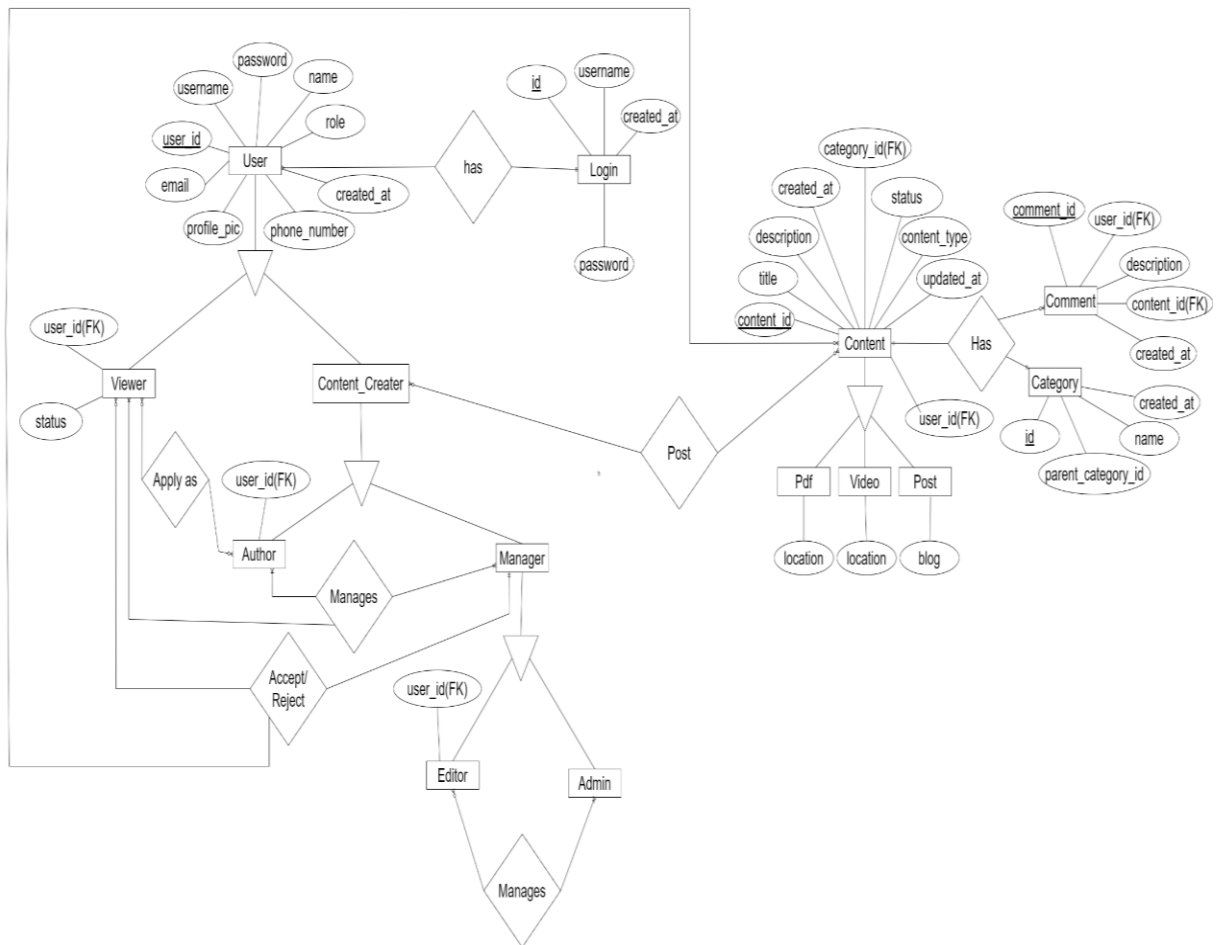


Figure 2.8: Entity Relationship Diagram

2.6.3 Data Flow Diagram

A Data Flow Diagram (DFD) is a visual representation of how data moves through a system, showing processes, data stores, and external entities to illustrate the system's functional flow.

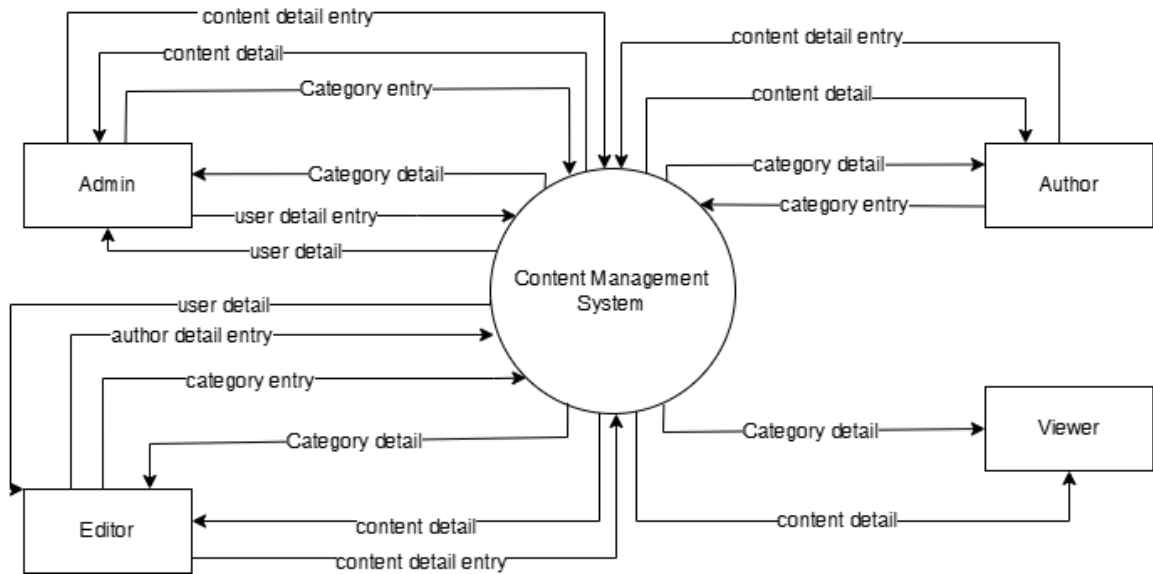


Figure 2.9: DFD Level 0

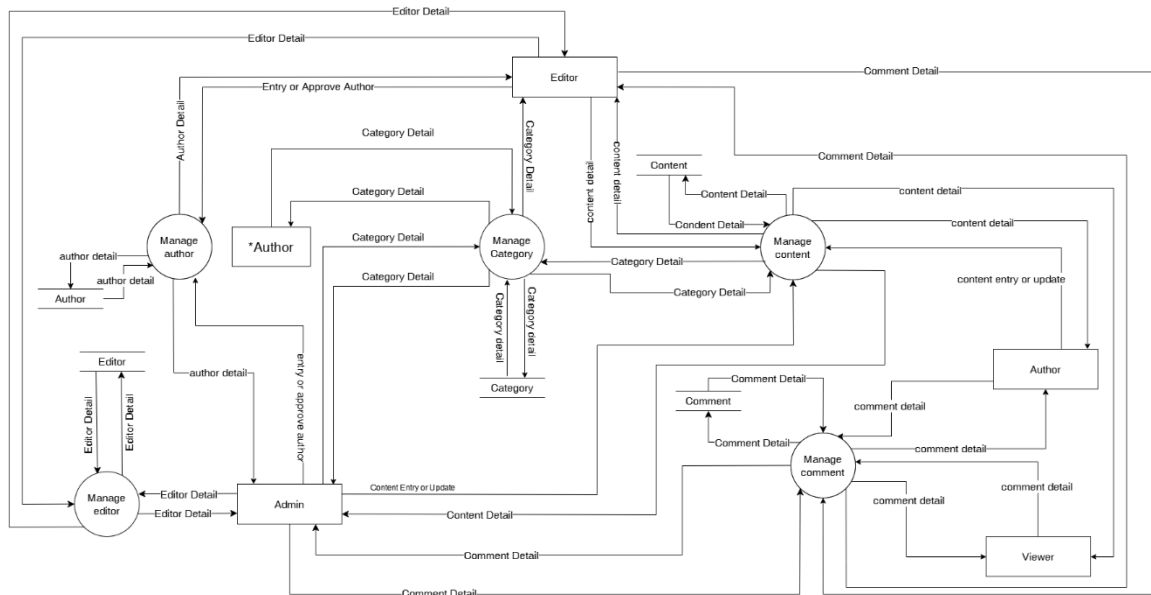


Figure 2.10: DFD Level 1

2.6.6 Activity Diagram

An activity diagram is a UML tool that visually represents the workflow or processes in a system, showing the sequence of activities, decision points, and parallel actions.

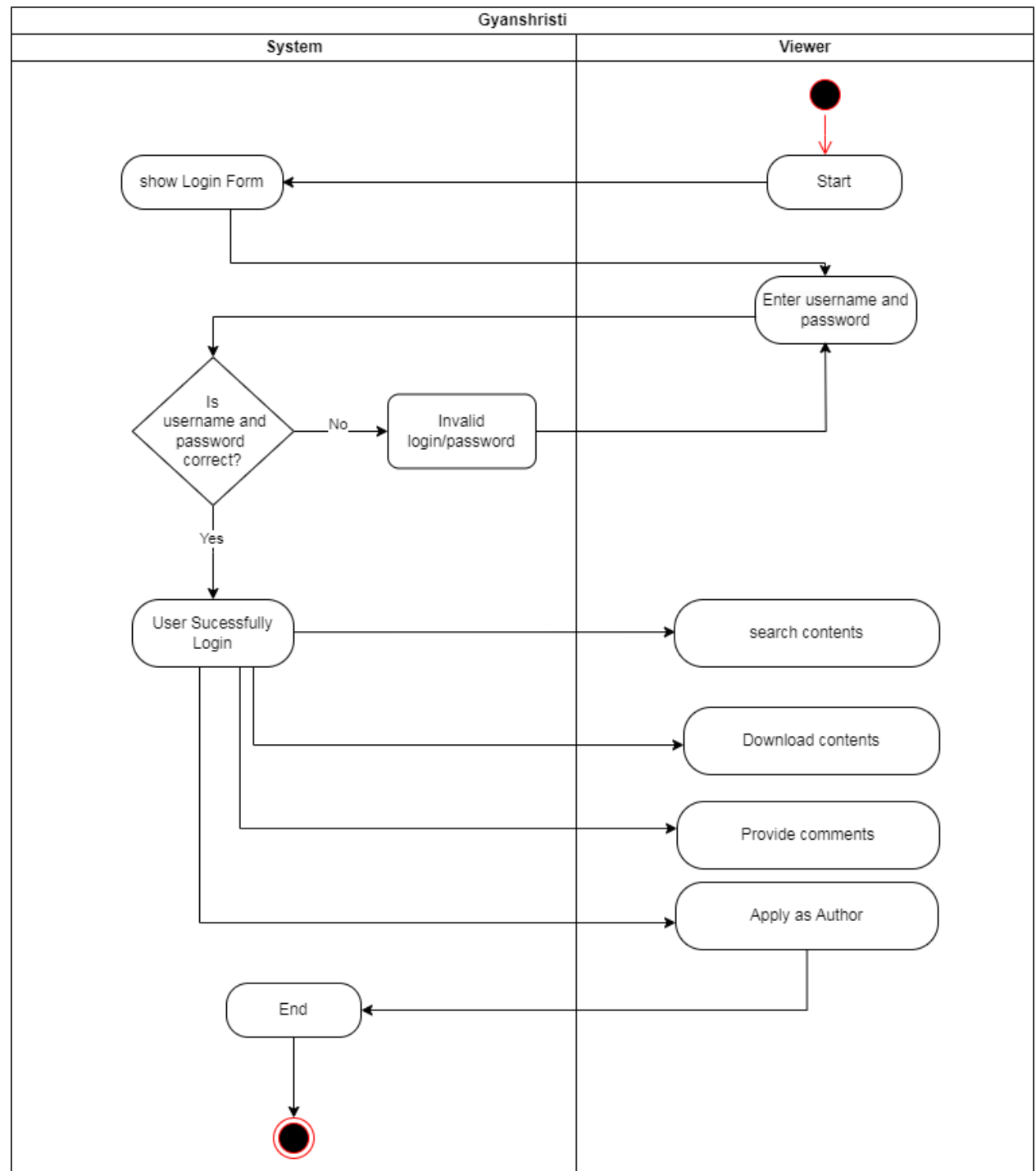


Figure 2.11: Activity Diagram of Viewer

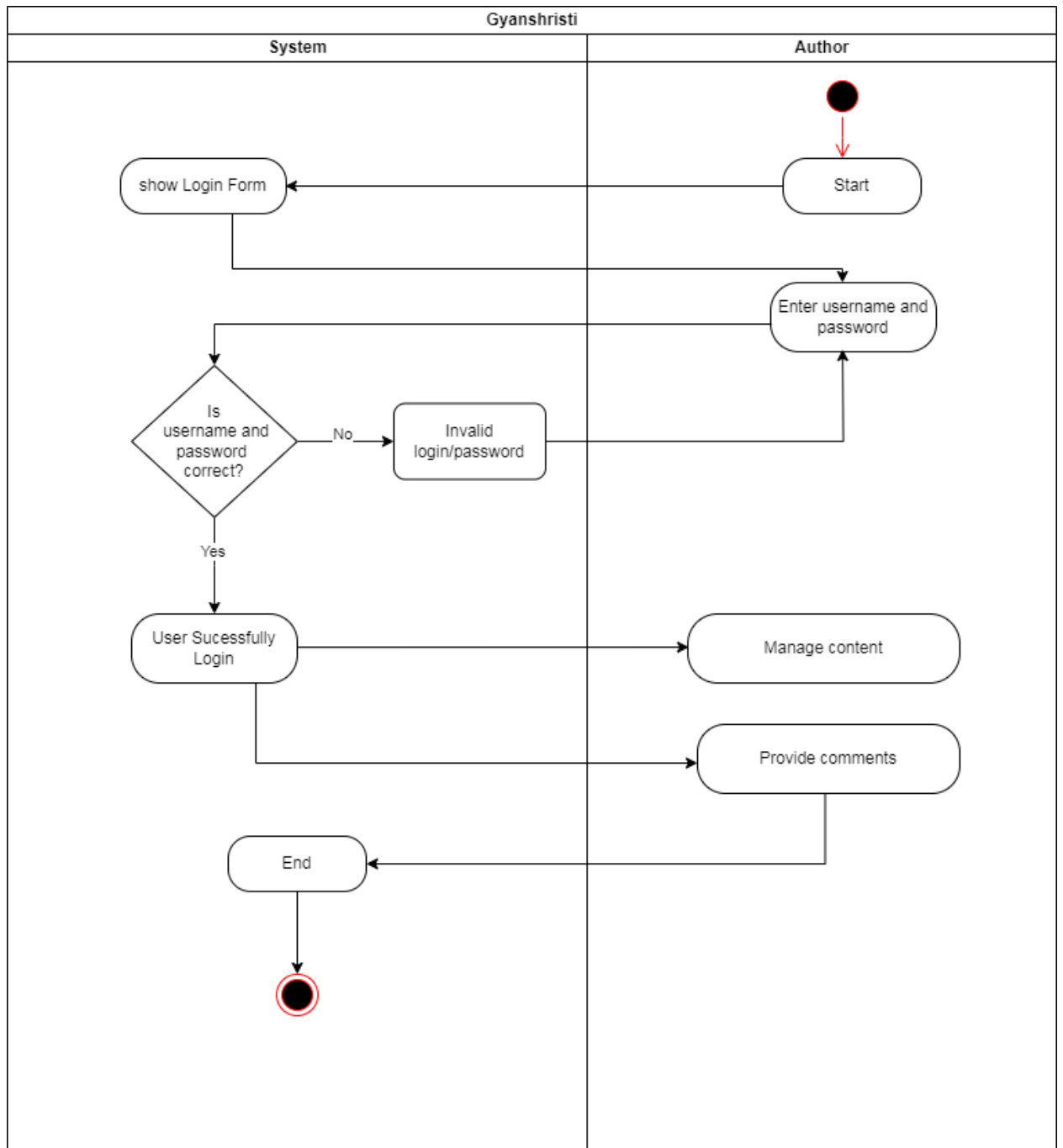


Figure 2.12: Activity Diagram of Author

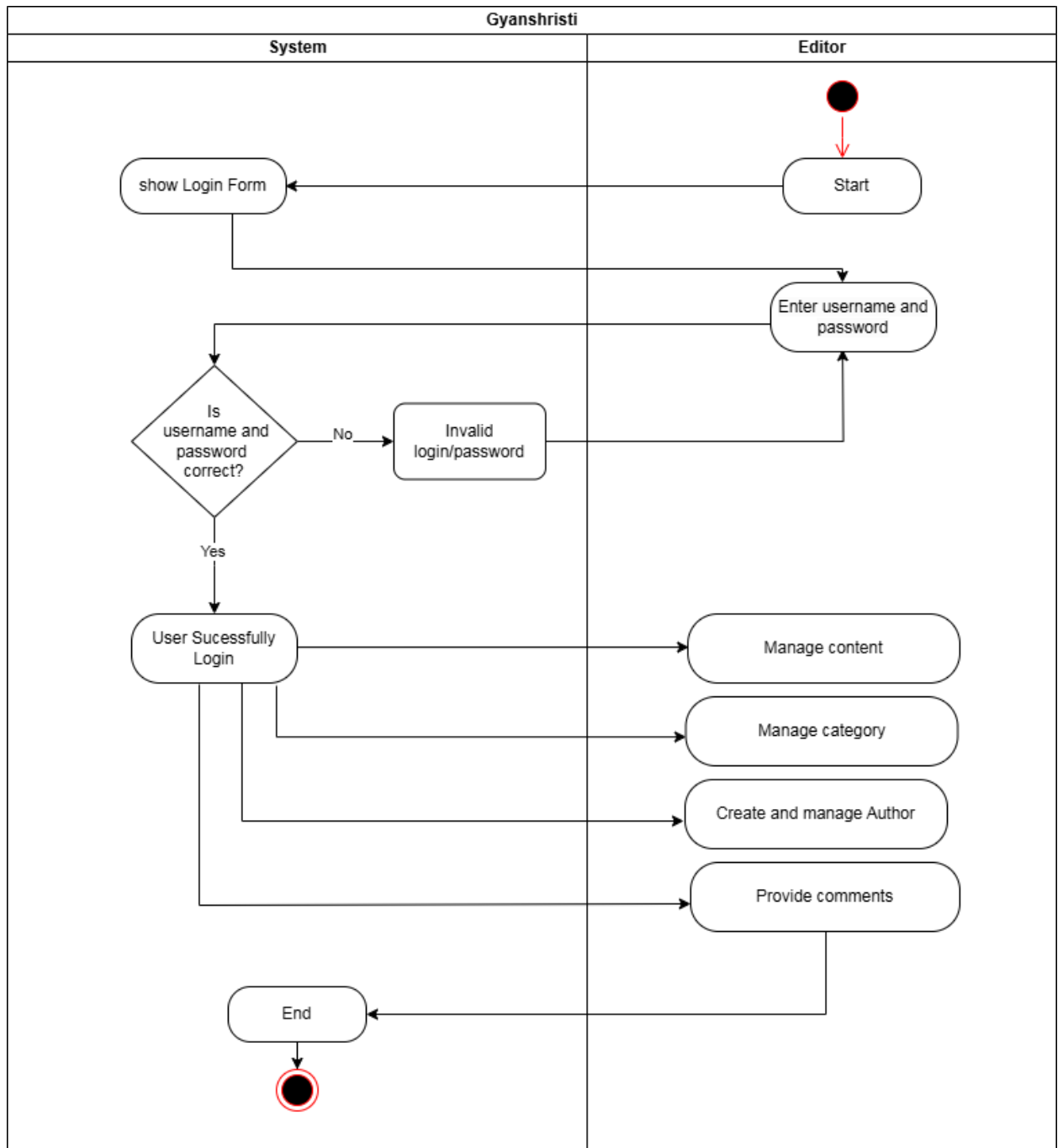


Figure 2.13: Activity Diagram of Editor

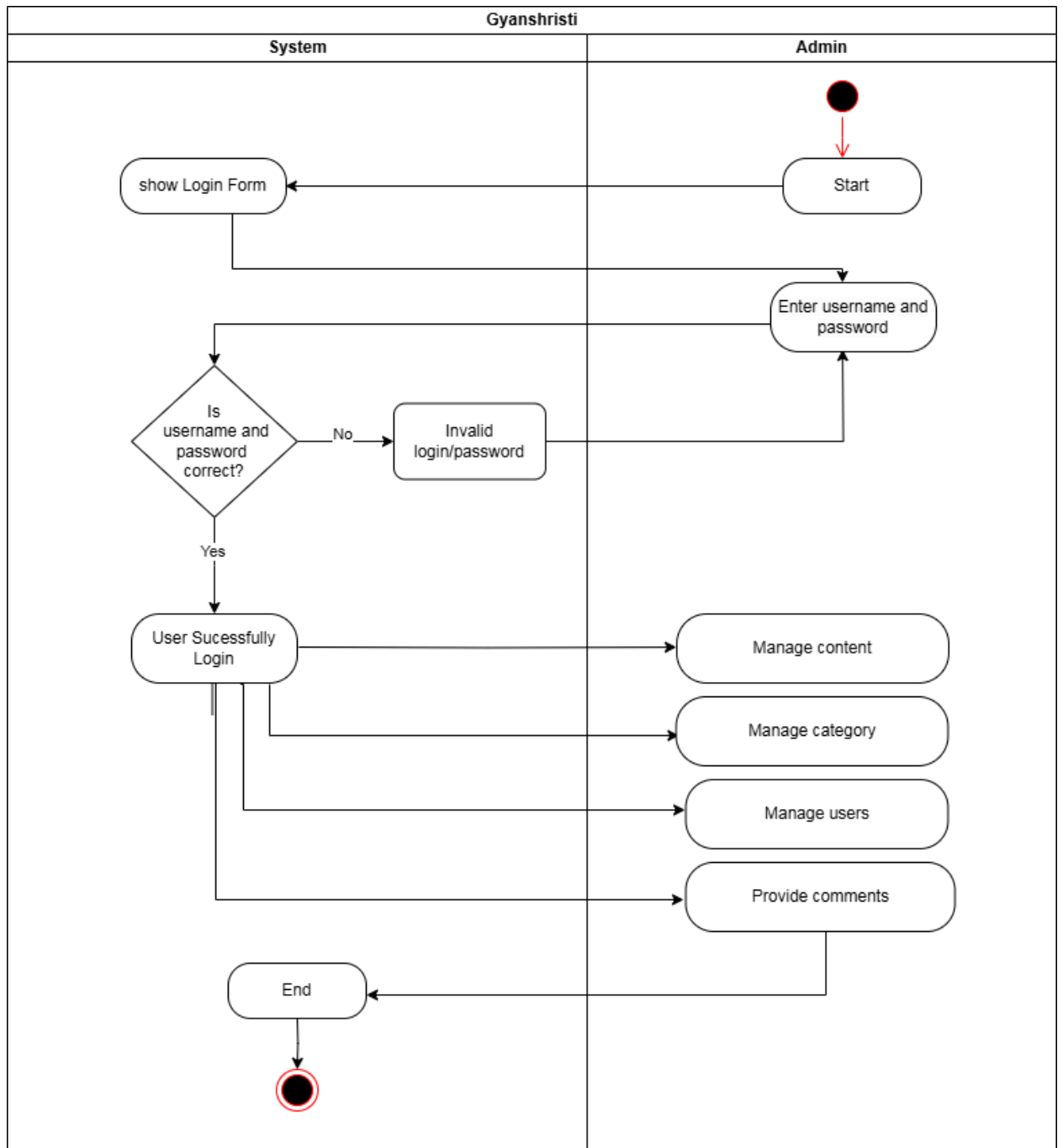


Figure 2.14: Activity Diagram of Admin

CHAPTER III: SYSTEM DESIGN

3.1. System Architecture and Overview

System design is the process of defining the architecture components, modules, interface and data for a system to satisfy specified requirements. It involves specifying how the system will accomplish its objectives and meet the needs of users, considering factors such as efficiency, scalability and maintainability. It is a crucial phase in the software development life cycle (SDLC) and involves transforming the requirements gathered during the analysis phase into a blueprint for the actual system. We have developed "CONTENT MANAGEMENT SYSTEM (GYANSHRISTI)" which required MongoDB for its functioning. To use this system, we need a web browser, internet, laptop/ desktop and the users.

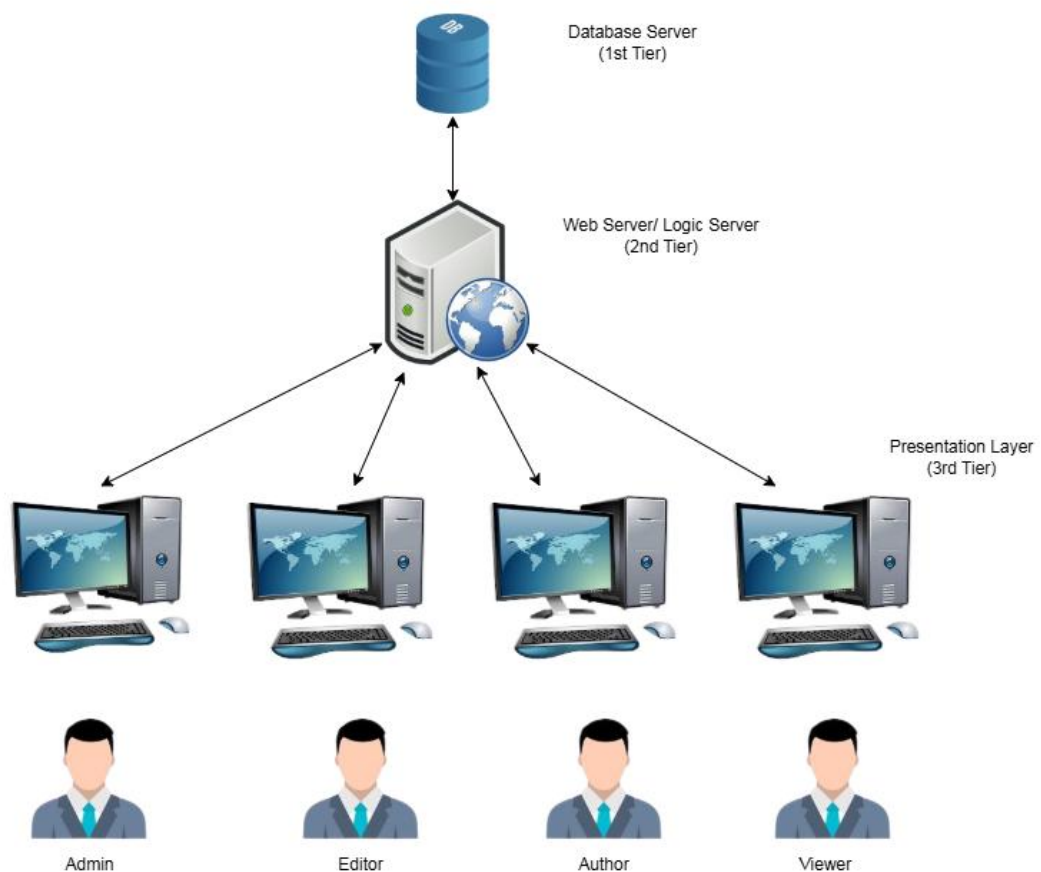


Figure 3.1: System Architecture

3.2. System Design

System design is the process of planning and defining the structure, components, interfaces, and data of a system to fulfill specified requirements. It involves creating a detailed blueprint that guides the implementation of the system, covering aspects such as architecture, components, data organization, interfaces, procedures, and security.

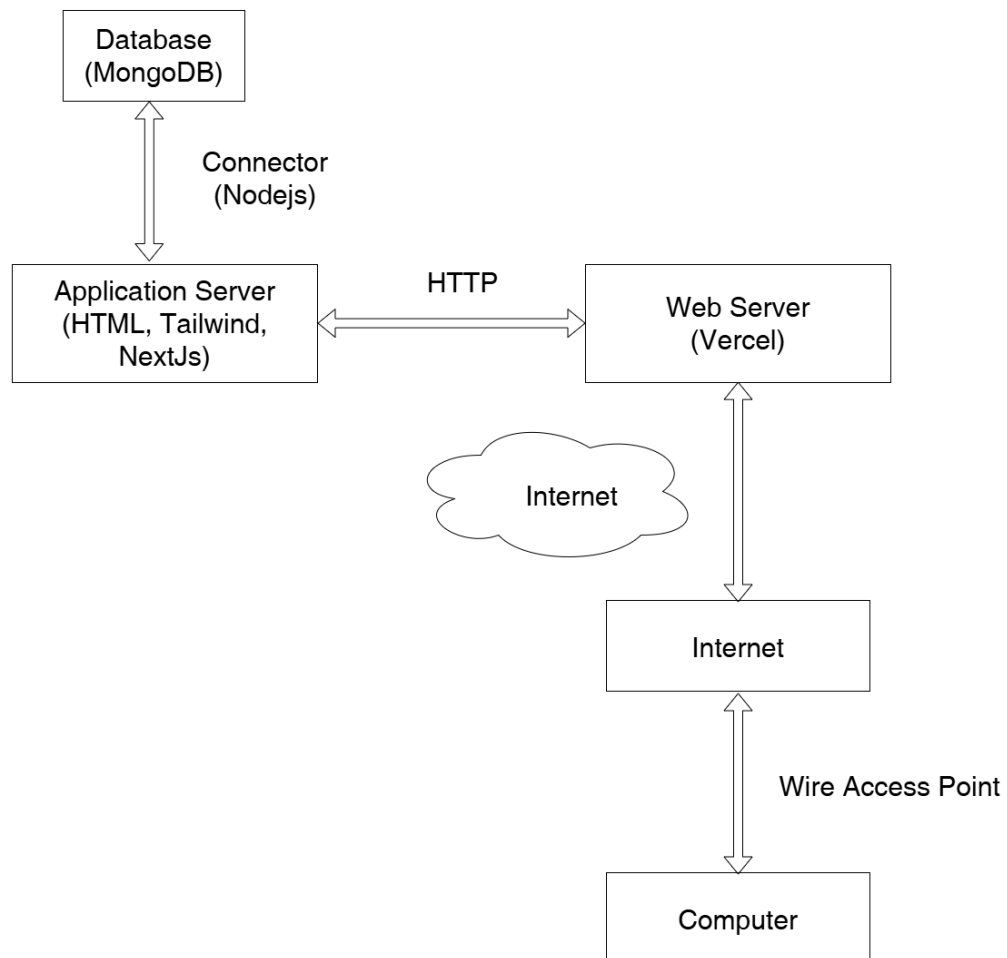


Figure 3.2: Implementation architecture of Content Management System

3.3. Interface Design

Interface design is the process of creating visual and interactive elements for computer systems, software, websites, or applications to make them user-friendly and aesthetically pleasing. It involves designing the layout, appearance, and functionality of on-screen elements such as buttons, menus, and navigation, with the aim of providing a positive and efficient user experience.

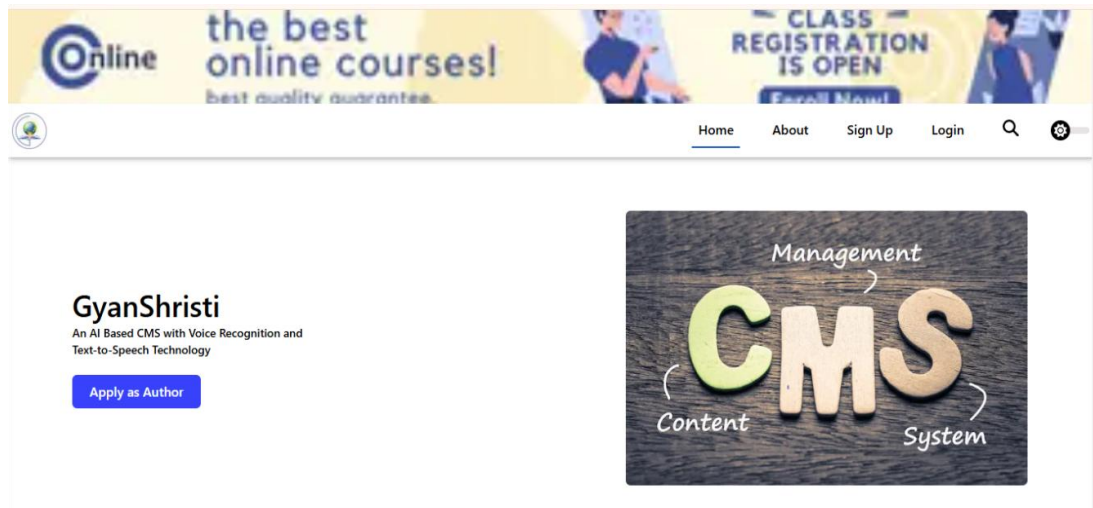


Figure 3.3: Homepage of Viewer

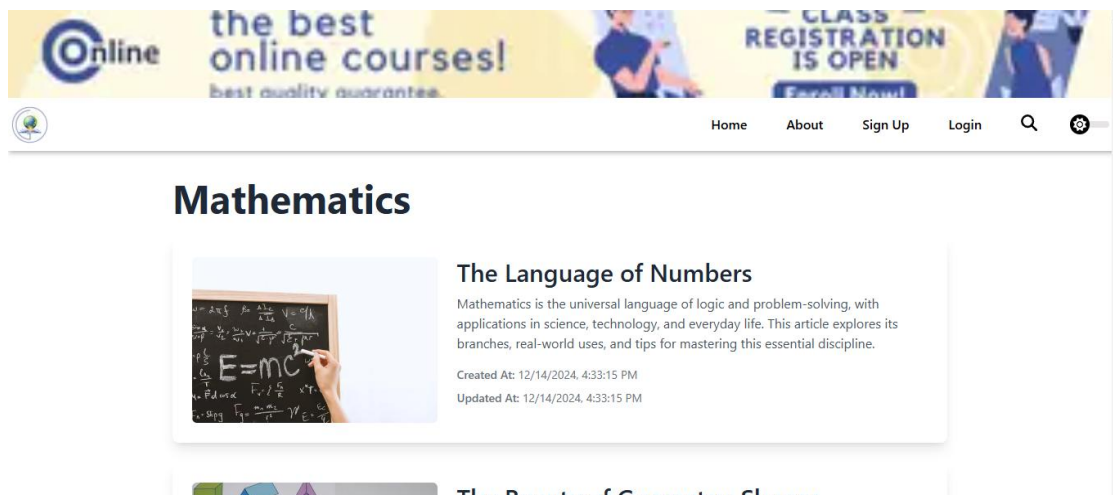



Figure 3.4: Category Wise Content of Viewer




Home About Sign Up Login 🔍 ⚙️

GyanShristi CMS

A CMS based on Voice Recognition and Text to Speech Technology for the better management of the contents.

[Know About Us](#)



Full Name


Email ID

Contact Number


Address

[Cancel](#) [Next](#)


Figure 3.5: Sign up Page of Non-registered User



Home About Sign Up Login 🔍 ⚙️



Username

Password 

[Login](#)

Figure 3.6: Login Page



Figure 3.7: Dashboard Page of Admin

3.4. Database Schema

A database schema in MongoDB defines the structure of data in collections and documents, allowing flexibility with varying fields, while tools like Mongoose can enforce consistent patterns and validation.

3.4.1 UserSchema

```
const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
  },
  username: {
    type: String,
    required: true,
  },
  password: {
    type: String,
    required: true,
  },
  email: {
    type: String,
    required: true,
  },
})
```

```

address: {
  type : String,
  required : true,
},
profile_pic: {
  type: String,
},
phone_number: {
  type: Number,
  required: true,
},
role: {
  type: String,
  enum: ["admin", "editor", "author", "viewer"],
  default: "viewer",
},
status: {
  type: String,
  enum: ["unrequested", "pending", "approved", "rejected"],
},
otp: {
  type: String
},
created_at: {
  type: Date,
  default: Date.now,
},
});

```

3.4.2 ContentSchema

```

const ContentSchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },

```

```
description: {
  type: String,
  required: true,
},
content_type: {
  type: String,
  enum: ["pdf", "video", "post"],
  required: true,
},
location: {
  type: String,
},
blog: {
  type: String,
},
thumbnail: {
  type: String,
  default:
    "https://images.pexels.com/photos/1323550/pexels-photo-1323550.jpeg?auto=compress&cs=tinysrgb&w=1260&h=750&dpr=1",
},
status: {
  type: String,
  enum: ["Pending", "Uploaded", "Rejected"],
  default: "Pending",
},
user_id: {
  type: mongoose.Schema.Types.ObjectId,
  ref: "User",
},
category_id: {
  type: mongoose.Schema.Types.ObjectId,
  ref: "Category",
  default: null,
}
```

```

    },
    created_at: {
      type: Date,
      default: Date.now,
    },
    updated_at: {
      type: Date,
      default: Date.now,
    },
  });

```

3.4.3 CommentSchema

```

const CommentSchema = new mongoose.Schema({
  description : {
    type : String,
    required : true
  },
  user_id : {
    type : mongoose.Schema.Types.ObjectId,
    ref : 'User'
  },
  content_id : {
    type : mongoose.Schema.Types.ObjectId,
    ref : 'Content'
  },
  parent_comment_id : [{
    type : mongoose.Schema.Types.ObjectId,
    ref : 'Comment'
  }],
  created_at : {
    type : Date,
    default : Date.now
  }
});

```

3.4.4 CategorySchema

```
const CategorySchema = new mongoose.Schema({
  title: {
    type: String,
    required: true,
  },
  user_id: {
    type: mongoose.Schema.Types.ObjectId,
    ref : "User"
  }
});
```

3.4.5 LoginSchema

```
const LoginSchema = new mongoose.Schema({
  user_id: {
    type: mongoose.Schema.Types.ObjectId,
    required: true,
  },
  created_at: {
    type: Date,
    default: Date.now,
  },
});
```

CHAPTER IV: IMPLEMENTATION AND TESTING

4.1 Implementation and Overview

The Waterfall model is applied to our project as it follows a structured approach where development progresses through distinct and sequential phases. These phases include requirements analysis, system design, implementation, testing, deployment, and maintenance. Each phase must be fully completed before moving on to the next, ensuring a systematic and organized development process. This model emphasizes detailed documentation and a clear progression, making it easier to manage, track progress, and ensure deliverables are well-defined. It is particularly suitable for projects with fixed requirements, schedules, and budgets, where a thorough plan and predictable outcomes are essential. Although the Waterfall model is well-suited for smaller or straightforward projects due to its structured approach and emphasis on thorough documentation, it is less adaptable to changes and iterative feedback, which can be a limitation for more complex or dynamic project environments.

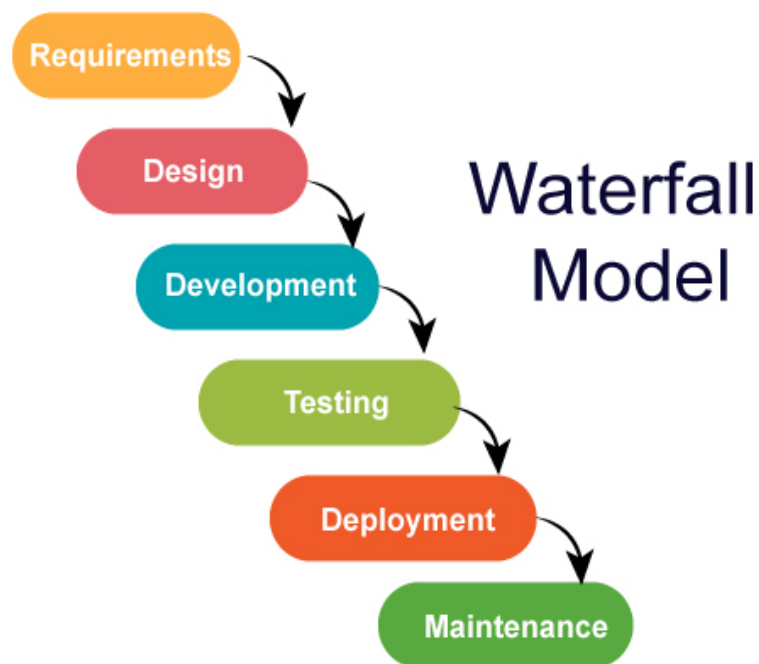


Figure 4.1: Waterfall Model

4.2 Tools Used

4.2.1 System Design Tools

- **Draw.io**

Draw.io is a free, web-based diagramming tool that allows users to create flowcharts, UML diagrams, network diagrams, and other types of visual representations for planning and documenting systems. It offers a wide range of shapes, templates, and collaboration features to create professional diagrams.

In our CMS, draw.io helps to visualize the architecture, data flow, and overall system design. It is used to map out the user roles, permissions, and the interaction between different entities.

- **Figma**

Draw.io is a cloud-based design and prototyping that allows teams to collaborate in real-time on UI/UX design projects. It enables designers to create interactive prototypes, wireframes, and visual designs that can be shared and edited by team members simultaneously.

In our CMS, Figma is used to design the user interface, create wireframes, and prototype user flows. It is used in our project to have a clear visual about the UI of our system and converting the design into the final product.

- **Canva**

Canva is an online graphic design tool that allows users to create professional-quality designs easily with its drag-and-drop interface. It offers for a wide range of projects, including social media posts, presentations, posters, and more, making it accessible for both designers and non-designers.

In our CMS, Canva is used to create logo and advertisement banner. Canva has helped to make visually appealing design banners.

4.2.1 Collaboration Tools

- **Git**

Git is a distributed version control system that allows developers to track changes in code, collaborate with team members, and manage different versions of a project. It enables efficient branching, merging and rollback of changes.

In our CMS, it is used to manage and version control the codebase. It helped us developers to work on different features or bug fixes in isolated branches, and later merge them into main branch without disrupting the main code.

- **GitHub**

GitHub is a web-based platform built on Git that provides hosting for version-controlled repositories, along with tools for collaboration, code review, and project management.

In our CMS, GitHub serves as the central hub for hosting the codebase and managing collaborative among team members. It has helped us to track the bugs and feature requests through issues, and document the project using README files or wikis.

4.2.1 Front End Tools

- **HTML 5**

HTML5 is the latest version of the language used to create web pages. It improves web development with better structure, multimedia support, interactivity, and offline features, making content management more efficient and accessible.

In our content management system, HTML5 ensures the proper structure of the webpage, content organization, and seamless multimedia integration while improving the structure and scalability.

- **Tailwind CSS**

Tailwind CSS is a CSS framework that makes building modern, responsive websites faster and easier. Instead of writing custom CSS, you use it pre-made utility classes directly in your HTML. This speed up development and makes customization simple, which is why many developers like using it.

In our content management system, Tailwind CSS enables efficient and responsive design by providing pre-built utilities classes for layout, spacing, typography, and colors.

- **Framework: NextJs**

NextJS is a React framework that simplifies the development of server-side rendered and static websites, providing features like routing, server-side rendering, and automatic code splitting for optimized performance.

In our content management system, Next.js provides fast, scalable, and SEO friendly pages by doing server-side rendering and static generation. It helps in easy routing, dynamic content handling, and easy API integration.

- **Language: TypeScript**

TypeScript is a programming language that adds static typing to JavaScript, helping developers catch errors and write more maintainable code.

In our CMS, TypeScript enhances the development process by offering type safety, which reduces runtime errors and improves code clarity. It helps in managing complex data structures and ensures the data are handled correctly while passing from one component to another.

4.2.2 Back End Tools

- **Framework: ExpressJS**

ExpressJS is a minimalist application framework for NodeJS, providing a robust set of features for building web servers and APIs quickly and efficiently. It simplifies the development process by offering powerful tools for handling HTTP requests, middleware integration, routing, and session management.

In our CMS, due to its middleware system, Express allows us to add custom functionality like logging, error handling, and validation for our application.

- **Language: JavaScript**

JavaScript is a versatile programming language used primarily for building interactive websites and web applications. It enables developers to create dynamic and interactive user experiences on websites and applications. Originally designed to run in web browsers, JavaScript is now widely used both on the client side (front-end) and server side (back-end, with platforms like Node.js).

In our CMS, it helps in implementing dynamic features like form validation, and interactive UI elements.

- **Database: MongoDB**

MongoDB is a popular NoSQL database that uses a document-oriented data model. It is designed for flexibility, scalability, and performance, making it suitable for a wide range of applications.

In our CMS, it helps in the storage and management of various types of data like users, contents, categories, comments, and logins. Also, it helps in creating flexible schema throughout the project.

- **Database Management Tools: MongoDB Compass**

MongoDB Compass is a GUI-based platform for MongoDB to explore and manipulate data. Here, we can visualize the data stored in the database in a user-friendly manner.

In our CMS, MongoDB Compass helps by providing a user-friendly interface to visualize data, optimize queries, and manage collections and documents.

4.2.2 Testing Tools

- **Postman**

Postman is a popular API testing tool ideal for validating Express.js projects. It allows developers to simulate HTTP requests (GET, POST, PUT, DELETE, etc.) to test routes, handle dynamic inputs like query parameters or headers, and inspect server responses. With features like scripting, environment management, and collections, Postman simplifies manual and automated testing. It ensures robust error handling and seamless integration with CI/CD workflows through its CLI tool, Newman.

In our CMS, it is used to test and validate the RESTful APIs built with Express.js. It helps in ensuring that the API responses are accurate and efficient.

- **Jest**

Jest is a powerful JavaScript testing framework widely used for testing Express.js projects. It provides a simple setup for writing unit and integration tests to validate API routes, middleware, and server logic. With features like test mocking, snapshot testing, and built-in assertions, Jest ensures reliable and maintainable tests. Its fast execution, detailed reports, and coverage tracking make it ideal for ensuring code quality and robustness in Express.js applications.

In our CMS, it is used to write unit and integration tests for both frontend and backend components. It helps to maintain high code quality for improving the overall stability and reliability of the system.

4.3 Testing

Testing is the systematic process of evaluating a software application or system to identify and resolve defects, ensuring that it operates as intended. The primary objective of testing is to verify that the software complies with specified requirements, functions correctly, and provides a reliable and satisfactory user experience. This process involves executing the software or its components, using various test cases and scenarios, to detect errors or bugs.

4.3.1 Component Testing

Component testing is a type of software testing that focuses on verifying the functionality and behavior of individual components or modules in isolation to ensure they perform as expected.

Table 4.1: Component testing of Login.test.tsx

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-01	Render Form Elements	Verify that the login form elements and links are rendered properly	N/A	All elements are present: username, password, login button, links.	All elements are present as expected.	Pass
T-02	Update Form State	Verify that the form state updates correctly on input changes.	Username: "testuser", Password: "password123"	Form state updates with entered values.	Form state updated with values: "testuser" and "password123".	Pass
T-03	Handle Empty Submission	Verify that an empty form submission triggers an error notification.	Click the login button without entering any input.	Error notification displayed: "Enter valid credentials."	Error notification displayed: "Enter valid credentials."	Pass
T-04	Handle Successful Login	Verify successful login flow with correct	Username: "testuser", Password: "password"	Notification: "Login successful", calls login	Notification: "Login successful", called login	Pass

		credentials.	123". Mock API returns valid token.	with token and user ID.	with token and user ID.	
T-05	Handle Validatio n Error	Verify that incorrect login credentials result in an error notification.	Username: "wronguser ", Password: "wrongpas s". Mock API response: { ok: false, status: 400, msg: "Invalid credentials " }.	Notification: "Invalid credentials".	Notification : "Invalid credentials".	Pass

Table 4.2: Component testing of Register.test.tsx

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-06	Render First Form	Verify that the first form of the registration component renders correctly.	N/A	Fields: Full Name, Email ID, Contact Number, Address are displayed.	Fields rendered as expected.	Pass
T-07	Switch to Second Form	Verify that clicking "Next" switches to the second form.	Click "Next" without filling the first form.	Fields: Username, Password, Confirm Password are displayed.	Fields rendered as expected.	Pass
T-08	Password Mismatch	Verify that mismatched passwords trigger an error notification.	Username: "testuser", Password: "password12 3", Confirm Password: "password45 6".	Notification: "Please match the password."	Notification displayed as expected.	Pass

T-09	Handle OTP Sending	Verify successful OTP sending notification after filling the forms correctly.	Full Name: "Test User", Email: "test@example.com", Contact: "1234567890", Address: "Test Address", Password: "123".	Notification: "OTP sent successfully."	Notification displayed as expected.	Pass
T-10	Successful Registration	Verify that successful registration navigates to the login page.	Same as T-04, with OTP: "123456".	Navigates to "/login".	Navigated as expected.	Pass
T-11	Handle API Errors	Verify that API errors are handled gracefully, and an error notification is shown.	Same as T-04, but with mocked API response: { ok: false, status: 400, msg: "API Error" }.	Notification: "API Error."	Notification displayed as expected.	Pass

Table 4.3: Component testing of ContentTable.test.tsx

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-12	Handle API Error Gracefully	Verify that the component gracefully handles API errors when fetching content.	Simulate an API rejection (mockRejectedValue).	Content is not displayed, and no errors are thrown in the UI.	As expected.	Pass
T-13	Search Filtering	Verify that the content table filters results based on search input.	Input: "Test" and "NonExistent" in the search field.	Content "Test Content" is displayed for "Test" but not displayed for "NonExistent"	As expected.	Pass

				t".		
T-14	Handle Delete Failure	Verify that the component handles failures when attempting to delete a content item.	Simulate delete API response with { ok: false } and msg: "Delete failed".	"Delete failed" notification is displayed, and the content remains in the list.	Delete failure handled correctly.	Pass
T-15	Pagination Update	Verify that pagination updates correctly when new data is fetched and pages are switched.	Use 15 content items and switch to page 2 using the pagination button.	Items on page 1 are not displayed after switching to page 2, and new items appear correctly.	Pagination works as expected.	Pass
T-16	Sorting by Creation Date	Verify that the content is sorted by creation date when fetched.	Provide items with creation dates 2024-01-02 and 2024-01-01.	The content with the newest date appears first in the list.	Content is sorted correctly.	Pass

Table 4.4: Component testing of CategoryTable.test.tsx

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-17	Render Table Headers	Verify that the table headers are rendered correctly.	None	Headers "SN," "Title," "Created By," and "Action" are displayed.	Headers are displayed as expected.	Pass
T-18	Fetch and Display Categories	Verify that category data is fetched from the API and displayed correctly in the table.	Mock category data: [{ id: 1, title: "Test Category", user: { name: "Test User" } }]	The category title "Test Category" and creator name "Test User" are displayed in the table.	Category data is displayed correctly.	Pass

Table 4.5: Component testing of ShowPdf.test.tsx

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-19	Render Required Elements	Verify that all required elements (title, description, user, etc.) are rendered correctly.	Valid mockProps object	Title, description, user name, thumbnail, profile picture, PDF viewer, and download link are displayed correctly.	All elements are displayed as expected.	Pass
T-20	Fetch Comments on Mount	Verify that the component fetches comments when mounted.	Valid mockProps.id	API call is made to fetch comments using contentId.	API call to fetch comments is verified.	Pass
T-21	Handle Fetch Error	Verify error handling when the fetch for comments fails.	Invalid API response (e.g., ok: false)	Error handling occurs without breaking the UI.	Fetch error is handled gracefully.	Pass
T-22	Default Profile Picture	Verify that the default profile picture is displayed when profilePic is null.	mockProps with profilePic: null	Default profile picture (/default.jpg) is displayed.	Default profile picture is displayed as expected.	Pass
T-23	Format Dates Correctly	Verify that the created and updated dates are formatted correctly.	Valid mockProps.createdAt and mockProps.updatedAt	Dates are displayed in MM/DD/YYYY format.	Dates are formatted and displayed correctly.	Pass

T-24	Render Advertisement Component	Verify that the ShowAdvertisement child component is rendered.	Valid mockProps	Advertisement component is rendered.	Advertisement component is displayed as expected.	Pass
T-25	Render Comment Component	Verify that the ShowComment child component is rendered.	Valid mockProps	Comment component is rendered.	Comment component is displayed as expected.	Pass

Table 4.6: Component testing of ShowPost.test.tsx

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-26	Render Component with Data	Verify that the component renders all content data correctly.	Valid mockContentData	Component displays the title, description, user info, and other details from mockContentData.	All content is rendered correctly.	Pass
T-27	Fetch Content Data on Mount	Verify that the API call is made to fetch content data on component mount.	Valid API endpoint and id in URL	API call to api/content/post/123 is made successfully.	API call verified.	Pass
T-28	Speech Synthesis Controls	Verify that the Speak button initializes the speech synthesis functionality.	Valid SpeechSynthesis mock and button click simulation	SpeechSynthesisUtterance is called and initializes the speech functionality.	Speech synthesis functionality works as expected.	Pass
T-29	Render Advertisement Component	Verify that the ShowAdvertisement	Valid mockContentData	Advertisement component is displayed	Advertisement component is displayed	Pass

		child component is rendered.		correctly.	as expected.	
T-30	Render Comment Component	Verify that the ShowComment child component is rendered.	Valid mockContentData	Comment component is displayed correctly.	Comment component is displayed as expected.	Pass
T-31	Format Dates Correctly	Verify that created and updated dates are formatted and displayed correctly.	Valid mockContentData.created_at	Dates are displayed in the correct format (e.g., MM/DD/YYYY).	Dates are formatted and displayed correctly.	Pass
T-32	Handle Download Functionality	Verify that the download button is rendered and functional.	Valid mockContentData.location	Download button is displayed with a title attribute and works as expected when clicked.	Download button is displayed as expected.	Pass

Table 4.7: Component testing of ShowVideo.test.tsx

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-33	Render Component Elements	Verify that all required elements (title, description, user info, etc.) are rendered correctly.	Valid mockProps	Component displays all required elements, including user info, title, description, and media-related elements.	Pass	N/A
T-34	Fetch Comments on Mount	Verify that comments are fetched when the component	Valid mockProps.id	API call to api/comments?contentId=123 is made	Pass	N/A

		mounts.		successfully.		
T-35	Handle Fetch Errors	Verify that the component handles comment-fetching errors gracefully.	Simulate API failure (ok: false)	Component gracefully handles error (e.g., shows no comments or an error message).	Pass	N/A
T-36	Default Profile Picture	Verify that the default profile picture is displayed when profilePic is null.	mockProps.profilePic: null	Component renders the default profile picture (/default.jpg).	Pass	N/A
T-37	Format Dates Correctly	Verify that created and updated dates are formatted and displayed correctly.	Valid mockProps.createdAt and mockProps.updatedAt	Dates are displayed in the correct format (e.g., MM/DD/YYYY).	Pass	N/A
T-38	Render Advertisement Component	Verify that the ShowAdvertisement child component is rendered.	Valid mockProps	Advertisement component is displayed correctly.	Pass	N/A
T-39	Render Comment Component	Verify that the ShowComment child component is rendered.	Valid mockProps	Comment component is displayed correctly.	Pass	N/A
T-40	Render Video Iframe Source	Verify that the iframe for video playback is rendered with the correct source URL.	Valid mockProps.location	Iframe source URL matches \${process.env.NEXT_PUBLIC_BACKEND_API}\${mockProps.location}.	Pass	N/A

T-41	Render Download Link	Verify that the download link for the video is rendered with the correct href.	Valid mockProps.location	Download link href matches <code>\${process.env.NEXT_PUBLIC_BACKEND_API}\${mockProps.location}</code> .	Pass	N/A
------	----------------------	--------------------------------------------------------------------------------	--------------------------	---------------------------------------------------------------------------------------------------------	------	-----

Table 4.8: Component testing of Logout.test.tsx

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-42	Render Nothing When Closed	Verify that the component does not render anything when isOpen is false.	isOpen: false	Modal content is not present in the DOM.	Pass	N/A
T-43	Render Modal When Open	Verify that the modal content is rendered when isOpen is true.	isOpen: true	Modal displays confirmation message and Yes and Cancel buttons.	Pass	N/A
T-44	Call onClose on Cancel	Verify that onClose is called when the Cancel button is clicked.	isOpen: true + Click Cancel button	onClose callback function is invoked.	Pass	N/A
T-45	Call onClose on X Button	Verify that onClose is called when the close (X) button is clicked.	isOpen: true + Click close button	(Currently commented out) Expect onClose callback to be invoked.	N/A	Test commented.
T-46	Call Logout on Yes	Verify that the logout function is called when	isOpen: true + Click Yes button	Logout function is invoked successfully.	Pass	N/A

		the Yes button is clicked.				
T-47	Handle Logout Errors Gracefully	Verify that errors during the logout process are logged without breaking the application.	Simulated logout rejection (mock error)	Console logs error message (e.g., "Logout failed").	N/A	Test commented.

4.3.2 API Testing

API testing is the process of validating Application Programming Interfaces (APIs) to ensure they function correctly, reliably, and securely by checking responses, performance, and error handling for various requests.

Table 4.9: API Testing

Endpoint	Method and isToken Needed	Body	Response & Status
/api/auth/sendOTP	POST, no	{ "email" : "shresthanewar678@gmail.com" }	Shows message "Name is required" (500)
/api/auth/register	POST, no	{ "name": "Jghghhggh", "username": "johndoe1", "password": "SecureP@ssw0rd!", "email": "shresthanewar67@gmail.com",	Upon valid OTP and unique username and email shows "Registration successful" (200) or else show "Invalid OTP" (400)

		<pre>"address": "123 Main Street, Apt 4B, Springfield, USA", "phone_number": "1231234", "otp": "227973" }</pre>	
/api/auth/login	POST, no	<pre>{ "username": "admin", "password": "admin" }</pre>	Shows message “Credentials error” (400) on incorrect username and password
/api/auth/login	POST, no	<pre>{ "username": "test", "password": "test123" }</pre>	Shows message “Login successful” (200) on correct username and password
/api/user/	GET, yes	nil	Shows the list of users in the system with their details except password
/api/user/role?role=viewer	GET, yes	nil	Shows the list of viewers if user is admin or editor
/api/user/role?role=author	GET, yes	nil	Shows the list of authors if user is admin or editor
/api/user/role?role=editor	GET, yes	nil	Shows the list of editors if user is admin or editor

/api/user/role?role=admin	GET, yes	nil	Shows the list of admin if user is admin or editor
/api/user/66b610084996992b3d5db30c	GET, yes	nil	Shows the details of the user with id “66b610084996992b3d5db30c” (200)
/api/user/66b610084996992b3d5db30d	GET, yes	nil	Shows “No users found” if ID doesn’t match (404)
/api/user/66b610084996992b3d5db30c	PUT, yes	{ "name" : "Sophia Shrestha", "username" : "johndoe131", "phone_number" : "24234234234" }	Edit the detail of user with username “johndoe131” (200) if username is found.
/api/user/66b610944996992b3d5db318	DELETE, yes	nil	Delete the user with id “66b610944996992b3d5db318” from the database (200)
/api/user/change-password/66b6101c4996992b3d5db310	PUT, yes	{ "old_password": "password123", "new_password": "password12" }	Edit the password of the user with id “66b6101c4996992b3d5db310”

		}	to the new password (200)
/api/user/change-email/66b6101c4996992b3d5db310	PUT, yes	{ "new_email" : "shresthanewar678@gmail.com" }	Edit the email of the user with id "66b6101c4996992b3d5db310" to the new email (200)
/api/user/approve-author/66b6101c4996992b3d5db310?approve=true	PUT, yes	nil	Approve the user as author if user with id "66b6101c4996992b3d5db310" is a viewer (200) or else show "User is not a viewer"
/api/user/change-to-editor/66b6101c4996992b3d5db310	PUT, yes	nil	Change the user role of user with id "66b6101c4996992b3d5db310" to editor.
/api/user/promote-admin/668ba90852a9bfcfcf20b7f	PUT, yes	nil	Change the user role of user with id "668ba90852a9bfcfcf20b7f" to admin (200)
/api/user/upload-profile-picture/66b610084996992b3d5db30c	PUT, yes	form-data profile-pic : /C:/Users/MANOJ/Pictur	Uploads profile picture for the user with id "66b610084996

		es/Screenshots/Experience at Freelancer.com.png	992b3d5db30c” (200)
/api/content/add/post	POST, yes		
/api/content/add/pdf	POST, yes	form-data title: PDF sample description: This is a pdf pdf: /C:/Users/MANOJ/Downloads/SmartDustbin.pdf thumbnail: /C:/Users/MANOJ/Downloads/people-woman-yoga-meditation.jpg category_id: 66cdd525db61a9ffa4f48401 user_id: 66b610084996992b3d5db30c content_type: pdf	Adds pdf detail to the database and stores pdf in the server (200) with a message “PDF added successfully”
/api/content/add/video	POST, yes	form-data title: Video sample description: This is a video pdf: /C:/Users/MANOJ/Downloads/SmartDustbin.mp4	Adds video detail to the database and stores video in the server (200) with a message “Video added successfully”

		thumbnail: /C:/Users/MANOJ/Downloads/people-woman-yoga-meditation.jpg category_id: 66cdd525db61a9ffa4f48401 user_id: 66b610084996992b3d5db30c content_type: pdf	
/api/content/edit/post/66963e7d41fd5196a57e5a14	PUT, yes	<pre>{ "title": "Sample PDF Title", "description": "This is a sample description for the PDF content.", "blog" : "<h1>THis is Hellosadsfadsfdf</h1>" }</pre>	Edit the detail for the post with id “66963e7d41fd5196a57e5a14” (200)
/api/content/edit/pdf/669640684c469b87f9c9540b	PUT, yes	form-data	Edit the detail for the pdf with id “669640684c469b87f9c9540b” (200)
/api/content/edit/video/669641fe4c469b87f9c9540e	PUT, yes	form-data	Edit the detail for the video with id “669641fe4c469

			b87f9c9540e” (200)
/api/content/delete/6697137d9b4aa6c594cbc530	DELETE, yes	nil	Delete the content(post, pdf, video) with id “6697137d9b4aa6c594cbc530” (200) and remove content from server
/api/content/approve/6698e1e6d3046b6250a317c7	PUT, yes	nil	Approves content with id “6698e1e6d3046b6250a317c7” (200)
/api/content/post/66963f0b41fd5196a57e5a1a	GET, yes	nil	Get the detail of the post with id “66963f0b41fd5196a57e5a1a” (200)
/api/content/pdf/6697132b9b4aa6c594cbc52d	GET, yes	nil	Get the detail of the pdf with id “6697132b9b4aa6c594cbc52d” (200)
/api/content/video/669641fe4c469b87f9c9540e	GET, yes	nil	Get the detail of video with id “669641fe4c469b87f9c9540e” (200)

/api/content/	GET, yes	nil	Get the list of contents with details (200)
/api/category/add	POST, yes	{ "title" : "Green", "user_id": "66b610084996992b3d5db30c" }	Adds the title called green to the database (200)
/api/category/edit/66b6162193b17c060c4d4f8a	PUT, yes	{ "title": "Communication System", "user_id": "66b610084996992b3d5db30c" }	Edits the category detail with id "66b6162193b17c060c4d4f8a" (200)
/api/category/delete/66b6162193b17c060c4d4f8a	DELETE, yes	nil	Delete the category with id "66b6162193b17c060c4d4f8a" (200)
/api/category/66b6160993b17c060c4d4f85	GET, yes	nil	Get detail of category with id "66b6160993b17c060c4d4f85"
/api/category	GET, no	nil	Get the list of category detail (200)
/api/user/countuser	GET, yes	nil	Get the number of individual users (admin, editor, author)

			and viewer in the database) (200
/api/user/add	POST, yes	{ "name": "Jane Smith", "username": "editor", "address": "456 Elm St, Metropolis", "password": "editor123", "email": "editor@example.com", "role": "editor", "phone_number": "9845671234" }	Adds user to the database (200)

4.3.3 System Testing

System testing is a type of software testing that evaluates the complete and integrated system to ensure it meets specified requirements, functioning as intended in real-world scenarios.

Table 4.10: System Testing of Admin

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-01	Login to System	Enter username and password	admin, admin123	Log into admin dashboard	Logged into admin dashboard	Pass
T-02	Add Editor	Enter name, address, contact number, role as editor, email, username password,	Editor, damauli, 9800765443, editor, editor25@gmail.com, editor25, editor123,	Add new editor to the system	Added new editor to the system	Pass

		new password	editor123			
T-03	Add Author	Enter name, address, contact number, role as author, email, username password, new password	Author, damauli, 9800765443, author, author25@gmail.com, author25, author123, author123	Add new author to the system	Added new author to the system	Pass
T-04	Edit Editor	Enter name, username, address and contact	Editor, editor25, damauli, 9810234555	Update editor	Updated editor successfully	Pass
T-05	Edit Author	Enter name, username, address and contact	Author, author25, damauli, 9810234555	Update author	Updated author successfully	Pass
T-06	Edit Viewer	Enter name, username, address and contact	Viewer, viewer25, damauli, 9807657743	Update viewer	Updated viewer successfully	Pass
T-07	Delete Editor	Click to the delete icon in the editor		Delete the editor	Editor deleted	Pass
T-08	Delete Author	Click to the delete icon in the author		Delete the author	Author deleted	Pass
T-09	Delete Viewer	Click to the delete icon in the viewer		Delete the viewer	Viewer deleted	Pass
T-10	Add PDF	Enter title, description, author, PDF, thumbnail, category		PDF added successfully	PDF added successfully	Pass

T-11	Add Video	Enter title, description, author, video, thumbnail, category		Video added successfully	Video added successfully	Pass
T-12	Add Post using Speech to Text	Enter title, description, author, thumbnail, category, speech to text post		Post added successfully	Post added successfully	pass
T-13	Add Post using text editor	Enter title, description, thumbnail, category, write post		Post added successfully	Post added successfully	pass
T-14	Edit PDF	Edit title, description, author, PDF, thumbnail, category		PDF edited successfully	PDF edited successfully	pass
T-15	Edit Video	Edit title, description, author, thumbnail, category		Video edited successfully	Video edited successfully	pass
T-16	Edit Post	Edit title, description, author, thumbnail, category, post		Post edited successfully	Post is edited while typing but not edited while providing voice command	Fail
T-17	Delete PDF	Click to the delete icon in the PDF content		PDF deleted successfully	PDF deleted successfully	pass

T-18	Delete Video	Click to the delete icon in the video content		Video deleted successfully	Video deleted successfully	pass
T-19	Delete Post	Click to the delete icon in the post content		Post deleted successfully	Post deleted successfully	pass
T-20	Add Category	Enter category name, choose author		Category added successfully	Category added successfully	Pass
T-21	Edit Category	Enter category name, choose author		Category edited successfully	Category edited successfully	Pass
T-22	Delete Category	Click to the delete icon in the category		Category deleted successfully	Category deleted successfully	Pass
T-23	Accept Viewer as Author					
T-24	Reject Viewer as Author	Click to the reject button in the request		Reject viewer as author	Rejected viewer as author	Pass
T-25	Accept author's content					
T-26	Reject author's content	Click to the reject button in the request		Reject content request	Rejected content request	pass
T-27	Notify admin of pending requests	Click to the notification in the sidebar		Display pending requests	Display pending requests	Pass

T-28	Add comment on content	Enter comment and hit post button	Very helpful Content	Display the comment on the comment	Display the comment on the comment	Pass
T-29	Delete comment on content	Click on comment in the side bar and click delete icon on comment		Delete the comment	Comment deleted	Pass
T-30	Reply to comment	Click on reply text on the comment of the content	Thank You	Reply to the comment successfully	Replied to the comment successfully	Pass
T-31	View Content	Click on the content in the sidebar		Display content table	Displayed content table	Pass
T-32	View Users	Click on the user in the sidebar		Display user table successfully	Displayed user table successfully	Pass
T-33	View Category	Click on the Category in the sidebar		Display category table successfully	Displayed category table successfully	Pass
T-34	View Comments	Click on the comment in the sidebar		Displayed comment table successfully	Displayed comment table successfully	Pass
T-35	Logout	Click on logout in the sidebar and click yes		Logout of the admin page successfully	Logged out of the admin page successfully	Pass

Table 4.11: System Testing of Editor

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-01	Login to System	Enter username and password	editor, editor123	Log into editor dashboard	Logged into editor dashboard	Pass
T-02	Add Author	Enter name, address, contact number, role as author, email, username password, new password	Author, damauli, 9800765443, author, author25@gmail.com, author25, author123, author123	Add new author to the system	Added new author to the system	Pass
T-03	Edit Author	Enter name, username, address and contact	Editor, author25, damauli, 9810234555	Update author	Updated author successfully	Pass
T-04	Edit Viewer	Enter name, username, address and contact	viewer, viewer25, damauli, 9810234555	Update viewer	Updated viewer successfully	Pass
T-05	Delete Author	Click to the delete icon in the author		Delete the author	Author deleted	Pass
T-06	Delete Viewer	Click to the delete icon in the viewer		Delete the viewer	viewer deleted	Pass
T-07	Add PDF	Enter title, description, PDF, thumbnail, category		PDF added successfully	PDF added successfully	Pass
T-08	Add Video	Enter title, description,		Video added successfully	Video added	Pass

		video, thumbnail, category			successfully	
T-09	Add Post using Speech to Text	Enter title, description, thumbnail, category, speech to text post		Post added successfully	Post added successfully	pass
T-10	Add Post using text editor	Enter title, description, thumbnail, category, write post		Post added successfully	Post added successfully	pass
T-11	Edit PDF	Edit title, description, PDF, thumbnail, category		PDF edited successfully	PDF edited successfully	pass
T-12	Edit Video	Edit title, description, thumbnail, category		Video edited successfully	Video edited successfully	pass
T-13	Edit Post	Edit title, description, thumbnail, category, post		Post edited successfully	Post edited successfully	pass
T-14	Delete PDF	Click to the delete icon in the PDF content		PDF deleted successfully	PDF deleted successfully	pass
T-15	Delete Video	Click to the delete icon in the video content		Video deleted successfully	Video deleted successfully	pass
T-16	Delete Post	Click to the delete icon in the post content		Post deleted successfully	Post deleted successfully	pass

T-17	Add Category	Enter category name, choose author		Category added successfully	Category added successfully	Pass
T-18	Edit Category	Enter category name, choose author		Category edited successfully	Category edited successfully	Pass
T-19	Delete Category	Click to the delete icon in the category		Category deleted successfully	Category deleted successfully	Pass
T-20	Accept Viewer as Author	Click to the accept button in the request		Accept viewer as author	Accepted viewer as author	Pass
T-21	Reject Viewer as Author	Click to the reject button in the request		Reject viewer as author	Rejected viewer as author	Pass
T-22	Accept author's content	Click to the accept button in the request		Accept author's content	Accepted author's content	Pass
T-23	Reject author's content	Click to the reject button in the request		Reject content request	Rejected content request	pass
T-24	Notify editor of pending requests	Click to the notification in the sidebar		Display pending requests	Display pending requests	Pass
T-25	Add comment on content	Enter comment and hit post button	Very helpful Content	Display the comment on the comment	Display the comment on the comment	Pass
T-26	Delete comment on content	Click on comment in the side bar and click		Delete the comment	Comment is deleted	Pass

		delete icon on comment				
T-27	Reply to comment	Click on reply text on the comment of the content	Thank You	Reply to the comment successfully	Replied to the comment successfully	Pass
T-28	View Content	Click on the content in the sidebar		Display content table	Displayed content table	Pass
T-29	View Users	Click on the user in the sidebar		Display user table	Displayed user table	Pass
T-30	View Category	Click on the Category in the sidebar		Display category table	Displayed category table	Pass
T-31	View Comments	Click on the comment in the sidebar		Displayed comment table	Displayed comment table	Pass
T-32	Logout	Click on logout in the sidebar and click yes		Logout of the editor page	Logged out of the editor page	Pass

Table 4.12: System Testing of Author

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-01	Login to System	Enter username and password	author, author123	Log into author's dashboard	Logged into author's dashboard	Pass
T-02	Add PDF	Enter title, description, PDF, thumbnail, category		Send request to admin and editor to add PDF	Sent request to admin and editor to add PDF	Pass
T-03	Add Video	Enter title, description, video,		Send request to admin and editor to add	Sent request to admin and	Pass

		thumbnail, category		video	editor to add video	
T-04	Add Post using Speech to Text	Enter title, description, thumbnail, category, speech to text post		Send request to admin and editor to add post	Sent request to admin and editor to add post	pass
T-05	Add Post using text editor	Enter title, description, thumbnail, category, write post		Send request to admin and editor to add post	Sent request to admin and editor to add post	pass
T-06	Edit Own PDF	Edit title, description, PDF, thumbnail, category		PDF edited	PDF edited	pass
T-07	Edit Own Video	Edit title, description, thumbnail, category		Video edited	Video edited	pass
T-08	Edit Own Post	Edit title, description, thumbnail, category, post		Post edited	Post edited	pass
T-09	Delete Own PDF	Click to the delete icon in the PDF content		PDF deleted	PDF deleted	pass
T-10	Delete Own Video	Click to the delete icon in the video content		Video deleted	Video deleted	pass
T-11	Delete Own Post	Click to the delete icon in the post content		Post deleted	Post deleted	pass

T-12	Add Category	Enter category and click save button		Add category	Category added	Pass
T-13	Edit Own Category	Enter category and click save button		Edit category	Category edited	Pass
T-14	Delete Own Category	Click to the delete icon in the category		Delete category	Category deleted	Pass
T-15	Add comment on content	Enter comment and hit post button	Very helpful Content	Display the comment on the comment	Display the comment on the comment	Pass
T-16	Delete Own comment on content	Click on comment in the side bar and click delete icon on comment		Delete the comment	Comment is deleted	Pass
T-17	Reply to comment	Click on reply text on the comment of the content	Thank You	Reply to the comment successfully	Replied to the comment successfully	Pass
T-18	View Own Content	Click on the content in the sidebar		Display content table successfully	Displayed content table successfully	Pass
T-19	View Category	Click on the Category in the sidebar		Display category table successfully	Displayed category table successfully	Pass
T-20	View Comments	Navigate to the content		Display comment	Displayed comment	Pass

T-21	Logout	Click on logout in the sidebar and click yes		Logout of the author's page successfully	Logged out of the author's page successfully	Pass
------	--------	----------------------------------------------	--	------------------------------------------	----------------------------------------------	------

Table 4.13: System Testing of Registered Viewer

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-01	Login to System	Enter username and password	author, author123	Log into author's dashboard	Logged into author's dashboard	Pass
T-02	Request to become Author	Click on Apply as Author button on the homepage		Send request as notification to admin and editor	Sent request as notification to admin and editor	Pass
T-03	Notify viewer in email if they are accepted to become author	Open the email that you provided while signing up into the system		Send notification of viewer accepted as author	Sent notification of viewer accepted as author	Pass
T-04	View Content	Navigate to the homepage and view content section		Display contents	Displayed contents	Pass
T-05	View Comments on the content	Click on the post and view comment section		Display comment	Displayed comment	Pass
T-06	Add comment on	Enter comment and hit post	Very helpful Content	Display the comment on the comment	Display the comment on the	Pass

	content	button			comment	
T-07	Delete Own comment on content	Click on comment in the side bar and click delete icon on comment		Delete the comment	Comment is deleted	Pass
T-08	Reply to comment	Click on reply text on the comment of the content	Thank You	Reply to the comment successfully	Replied to the comment successfully	Pass
T-09	View Category	Navigate to the homepage and view category section		Display category	Displayed category	Pass
T-10	Logout	Click on logout in the sidebar and click yes		Logout of the author's page successfully	Logged out of the author's page successfully	Pass

Table 4.14: System Testing of Non-registered Viewer

Test Case Id	Test Case	Test Description	Input Test Data	Expected Result	Actual Result	Remarks
T-01	View Content	Navigate to the content section in the homepage		Display Content	Displayed Content	Pass
T-02	View Category	Navigate to the category section in the homepage		Display Category	Displayed Category	Pass

T-03	Cannot comment to the content	Enter comment and hit post button	Very helpful content	Show prompt to register	Showed prompt to register	Pass
T-04	Sign Up to the system	Enter full name, username, email, contact number, address password, confirm password and hit submit button and type otp and submit	Anita Shrestha, anita, anita@gmail.com, 9807654321, damauli, anita123, anita123,	Register as Viewer	Registered as viewer	Pass

CHAPTER V: CONCLUSION AND RECOMMENDATION

5.1. Conclusion

In conclusion, the content management system (Gyanshristi) solves the problems of traditional content creation, sharing, and teamwork by providing an easy-to-use, efficient, and accessible digital platform. This system makes managing content simpler, reduces the workload for educators, and ensures students can quickly access relevant learning materials. Although there are challenges like relying on technology and internet access, Gyanshristi overcomes these with features such as offline access, voice-based input, and tools for better collaboration.

With advanced features like speech-to-text and text-to-speech, Gyanshristi makes content more accessible, especially for people with different learning needs. Its tools for organizing content, managing users, and supporting different roles make it useful for many educational and professional needs. By focusing on inclusive design and user engagement, Gyanshristi helps foster collaboration and self-paced learning. Gyanshristi is set to improve how content is managed by promoting inclusivity, interaction, and accessibility, creating a better learning experience for people from all backgrounds.

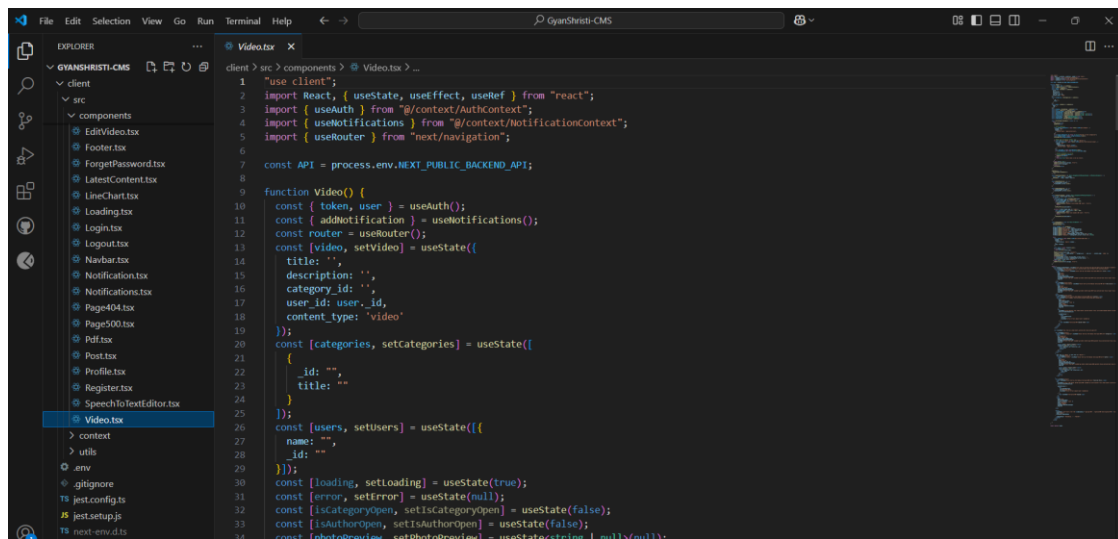
5.2. Recommendation

Once system is fully developed, it will offer powerful tools to manage content creation, organization, and collaboration while being easy to access for all users. We highly recommend that educators and institutions use system to simplify content management and make learning more engaging and inclusive. Teachers can save time by using features like voice-based content creation and speech-to-text tools, allowing them to focus on delivering quality educational materials. Administrators should regularly update user roles and permissions to keep the platform secure and well-organized. Authors and contributors are encouraged to use the platform's collaboration tools to interact with users, boost engagement, and gather valuable feedback. To make the most of the platform, users should explore features like offline access and accessibility tools, such as the text-to-speech converter. Institutions should also provide training for teachers and students to help them understand and effectively use the system. By using Gyanshristi's innovative features, educators, students, and contributors can create a more interactive and accessible learning experience, supporting the system's goal of improving content management in modern education.

5.3. References

- [1] Patel, S. K., Rathod, V. R., & Parikh, S. (2011, December 9). *A statistical comparison of Open Source CMS - IEEE Xplore*. Joomla, Drupal, and WordPress - a statistical comparison of open source CMS. <https://ieeexplore.ieee.org/abstract/document/6169111>
- [2] Patel, S. K., Rathod, V. R., & Parikh, S. (2011, December 9). *A statistical comparison of Open Source CMS - IEEE Xplore*. Joomla, Drupal, and WordPress - a statistical comparison of open source CMS. <https://ieeexplore.ieee.org/abstract/document/6169111>
- [3] Mohammed Ali Mohammed, Dr. Karim Q. Hussein, & Dr. Mustafa Dhiaa Al-Hassani. (2021, August 7). Real-Time Mobile Cloud Audio Reading System for Blind Persons. <https://www.webology.org/data-cms/articles/20220122114149amWEB19024.pdf>
- [4] ComeConnect. (2023, November 8). *Reasons why scholar's, researchers and students need listening.io*. Medium. <https://medium.com/@enahjayjohn/reasons-why-scholars-researchers-and-students-need-listening-io-2c4ccc28ee3c>
- [5] ForbesIndia. (n.d.). *Paving the way for growth: GeeksforGeeks is making learning accessible and affordable for programmers*. Forbes India. <https://www.forbesindia.com/article/brand-connect/paving-the-way-for-growth-geeksforgeeks-is-making-learning-accessible-and-affordable-for-programmers/75199/1>
- [6] Peetz, C. (2023, August 11). *Creating inclusive classrooms for blind students can benefit everyone. here's how*. Education Week. <https://www.edweek.org/teaching-learning/creating-inclusive-classrooms-for-blind-students-can-benefit-everyone-heres-how/2023/08>
- [7] *What is Data Flow Diagram?* (n.d.). <https://www.visualparadigm.com/guide/data-flow-diagram/what-is-data-flow-diagram/>;WWWSESSIONID=1715BFE147C14CA986ECC278C44815E7.www1

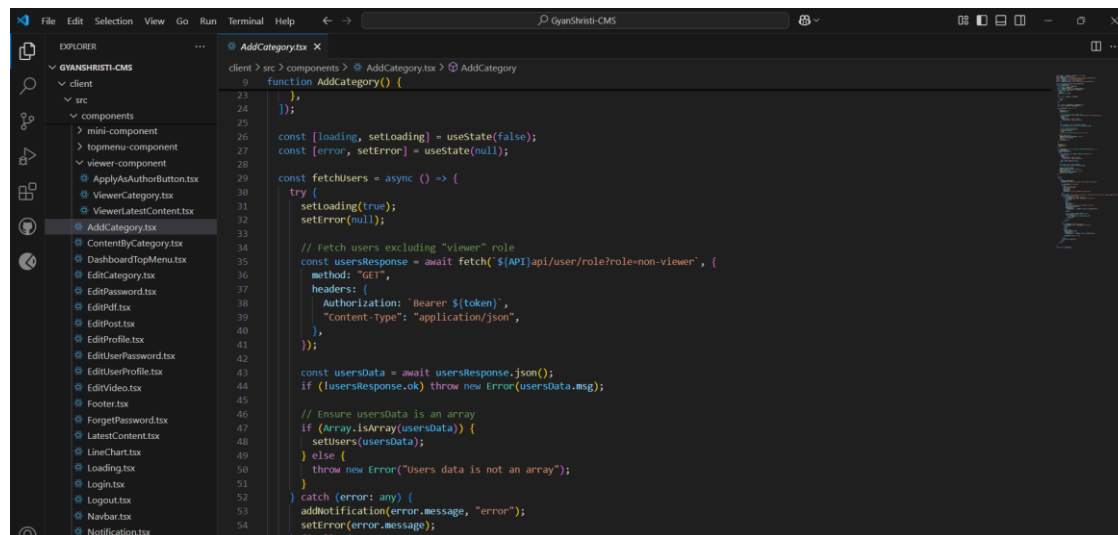
5.4. Appendix



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'client' directory containing 'components' and 'context'. The 'components' directory contains various files, including 'Video.tsx' which is selected. The code editor shows the following code:

```
1 "use client";
2 import React, { useState, useEffect, useRef } from "react";
3 import { useAuth } from "@context/AuthContext";
4 import { useNotifications } from "@context/NotificationContext";
5 import { useRouter } from "next/navigation";
6
7 const API = process.env.NEXT_PUBLIC_BACKEND_API;
8
9 function Video() {
10   const { token, user } = useAuth();
11   const [ addNotification ] = useNotifications();
12   const router = useRouter();
13   const [video, setVideo] = useState({
14     title: "",
15     description: "",
16     category_id: "",
17     user_id: user._id,
18     content_type: "video"
19   });
20   const [categories, setCategories] = useState([
21     {
22       id: "",
23       title: ""
24     }
25   ]);
26   const [users, setUsers] = useState([
27     {
28       name: "",
29       id: ""
30     }
31   ]);
32   const [loading, setLoading] = useState(true);
33   const [error, setError] = useState(null);
34   const [isCategoryOpen, setIsCategoryOpen] = useState(false);
35   const [isAuthOpen, setIsAuthOpen] = useState(false);
36   const [photoPreview, setPhotoPreview] = useState(null);
```

Figure 1: Code for Uploading Video



The screenshot shows a VS Code editor with a file explorer on the left and a code editor on the right. The file explorer shows a project structure with a 'client' directory containing 'components' and 'context'. The 'components' directory contains various files, including 'AddCategory.tsx' which is selected. The code editor shows the following code:

```
1 "use client";
2 import React, { useState } from "react";
3 import { useAuth } from "@context/AuthContext";
4 import { useNotifications } from "@context/NotificationContext";
5 import { useRouter } from "next/navigation";
6
7 const API = process.env.NEXT_PUBLIC_BACKEND_API;
8
9 function AddCategory() {
10   const { token, user } = useAuth();
11   const [ addNotification ] = useNotifications();
12   const router = useRouter();
13   const [loading, setLoading] = useState(true);
14   const [error, setError] = useState(null);
15
16   const fetchUsers = async () => {
17     try {
18       setLoading(true);
19       setError(null);
20
21       // fetch users excluding "viewer" role
22       const usersResponse = await fetch(`${API}api/user/role?role=non-viewer`, {
23         method: "GET",
24         headers: {
25           Authorization: `Bearer ${token}`,
26           "Content-Type": "application/json",
27         },
28       });
29
30       const usersData = await usersResponse.json();
31       if (!usersResponse.ok) throw new Error(usersData.msg);
32
33       // Ensure usersData is an array
34       if (Array.isArray(usersData)) {
35         setUsers(usersData);
36       } else {
37         throw new Error("Users data is not an array");
38       }
39     } catch (error: any) {
40       addNotification(error.message, "error");
41       setError(error.message);
42     }
43   };
44
45   return (
46     <div>
47       <h3>Add Category</h3>
48       <input type="text" value={video.title} />
49       <input type="text" value={video.description} />
50       <input type="text" value={video.category_id} />
51       <input type="text" value={video.user_id} />
52       <input type="text" value={video.content_type} />
53       <button type="button" value="Add Category" />
54     </div>
55   );
56 }
```

Figure 2: Code for Adding Category

```

server > utils > otp > JS sendOTP.js > sendOTP
1  const nodemailer = require('nodemailer');
2  const generateOTP = require('./generateOTP');
3  const generateOTPEmail = require('./htmlForOTP');
4  require('dotenv').config();
5
6
7  async function sendOTP(email){
8      const otp = generateOTP();
9
10     // Create a transporter object using the gmail SMTP
11     const transporter = nodemailer.createTransport({
12         service: 'gmail',
13         host: 'smtp.gmail.com',
14         secure: false,
15         auth: {
16             user: process.env.EMAIL_USER,
17             pass: process.env.EMAIL_PASS
18         }
19     });
20
21     // Setup email data with unicode symbols
22     const mailOptions = {
23         from: {
24             name: 'Gyanshruti CMS',
25             address: process.env.EMAIL_USER
26         },
27         to: email,
28         subject: 'Keep this OTP confidential',
29         text: 'Your OTP is ${otp}',
30         html: '
31             ${generateOTPEmail(otp)}
32         '
33     };

```

Fig 3: Code for Sending OTP

```

client > src > _tests_ > Login.test.tsx > describe(Login Component) callback > test('updates form state on input change') callback
1  import { render, act } from '@testing-library/react';
2  import { fireEvent } from '@testing-library/dom';
3  import { screen } from '@testing-library/dom';
4  import Login from '@components/Login';
5  import { useNotifications } from '@context/NotificationContext';
6  import { useAuth } from '@context/AuthContext';
7
8
9  jest.mock('@context/navigation', () => ({
10      useRouter: () => ({
11          push: jest.fn(),
12      }),
13  }));
14
15  jest.mock('@context/NotificationContext');
16  jest.mock('@context/AuthContext');
17
18  describe('Login Component', () => {
19      const mockAddNotification = jest.fn();
20      const mockLogin = jest.fn();
21
22      beforeEach(() => {
23          (useNotifications as jest.Mock).mockReturnValue({
24              addNotification: mockAddNotification,
25          });
26          (useAuth as jest.Mock).mockReturnValue({
27              login: mockLogin,
28          });
29      });
30
31      afterEach(() => {
32          jest.clearAllMocks();
33      });

```

Fig 4: Test Code for Login Component