COLLEGE MANAGEMENT SYSTEM

A PROJECT REPORT

Submitted by

ROSHAN BALA J

in partial fulfillment for the award of the degree of

BACHELOR OF TECHNOLOGY

in

COMPUTER SCIENCE AND BUSINESS SYSTEMS

KGiSL INSTITUTE OF TECHNOLOGY
ANNA UNIVERSITY: CHENNAI-600025

AUGUST 2025

KGiSL INSTITUTE OF TECHNOLOGY

BONAFIDE CERTIFICATE

Certified that this project report **"COLLEGE MANAGEMENT SYSTEM"**is the bonafide work of **"ROSHAN BALA J"** who carried out the project work under my supervision.

Ms . KEREN LOIS DANIEL **Dr. M . ANANTHI**

FACULTY INCHARGE **HEAD OF THE DEPARTMENT**

ABSTRACT

The College Document Management System is a full-stack web application developed using the Django framework with SQLite as the database, aimed at simplifying the management of academic documents between students and faculty within an educational institution. The system is designed to provide a secure, user-friendly platform that eliminates the inefficiencies of manual document handling and allows digital organization, submission, and retrieval of student documents under categorized headings. With role-based access controls, the platform distinctly separates functionality for students and teachers—ensuring that each user type can only perform actions relevant to their role.

Students can register and log in to their personalized dashboard where they are able to upload academic documents (in PDF format only), categorized under predefined or dynamically managed categories. Each student can manage their own submissions, including uploading, updating, and annotating documents with optional notes. Teachers, on the other hand, can access a dashboard where they can view all registered students, search based on roll numbers or usernames, and inspect individual document submissions. Teachers are also empowered to manage document categories, enabling them to create, activate, or deactivate categories as needed.

All user data, documents, and categories are securely handled using Django's built-in ORM and form validation mechanisms to ensure consistent and reliable operations. Uploaded files are stored in a structured directory and linked to user accounts, with proper error handling and permission checks to prevent unauthorized access. The platform also supports flash messaging and feedback alerts to improve interaction and usability. This project demonstrates effective use of Django's model-view-template (MVT) architecture, form handling, file management, authentication, and admin panel integration—delivering a practical and scalable solution for academic document workflows in colleges or universities.

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to all those who have supported and guided me throughout the development of the *College Document Management System* project. First and foremost, I thank my project guide and faculty mentor for their continuous encouragement, constructive feedback, and valuable insights which helped shape the direction of this project. Their expertise and support played a pivotal role in overcoming technical challenges and improving the overall quality of the application. I am also grateful to my institution and the Department of Computer Science and Business System for providing the resources and academic environment necessary to explore and apply my learning in a real-world scenario. Special thanks to my friends and peers who offered timely suggestions and helped test the application during its various stages of development. Lastly, I extend my heartfelt appreciation to my family for their unwavering support and motivation throughout this journey. This project stands as a testament to all the guidance, knowledge, and inspiration I have received along the way.

TABLE OF CONTENTS

INTRODUCTION

1.1 OBJECTIVE OF THE PROJECT

The primary objective of the *College Document Management System* is to create a secure, organized, and user-friendly digital platform that simplifies the process of managing academic documents within a college or university setting. Traditional methods of document submission and verification are often manual, time-consuming, and prone to errors or loss. This project aims to address those challenges by providing a centralized system where students can upload, view, and manage their academic documents in categorized formats, while faculty members can access, verify, and oversee submissions with minimal effort.

A key goal of the system is to implement a role-based structure, allowing different levels of access and control for students and teachers. Students are empowered with a dashboard where they can upload documents, update notes, and review submissions. Teachers, on the other hand, are granted tools to manage document categories, activate or deactivate them, and access student uploads with search and filtering capabilities.

The system also emphasizes data integrity, PDF-only uploads, and security through user authentication and permission checks. By streamlining administrative tasks and improving transparency, the project aims to enhance overall efficiency in academic workflows and promote the adoption of digital solutions in educational institutions.

1.2 COMPANY PROFILE

Accent Techno Soft (ATS) is an ISO 9001:2015 certified IT services and consulting company established in 2017 and headquartered in Coimbatore, Tamil Nadu. The company specializes in delivering custom web, mobile, ERP, and cloud-based solutions that streamline business operations, improve productivity, and support long-term growth. With a focus on innovation and client-centric development, ATS serves businesses across diverse industries, offering scalable and reliable digital transformation services.

Vision:

To be a leading provider of innovative and customized IT solutions that empower businesses to achieve operational excellence and sustainable growth.

Mission:

To deliver high-quality, cost-effective, and reliable technology solutions that meet the evolving needs of our clients, fostering long-term partnerships and driving mutual success.

Accent Techno Soft is committed to delivering superior IT solutions that adhere to the highest quality standards. In alignment with ISO 9001:2015, the company ensures:

- Client-focused development and support
- Continuous process improvement and innovation
- On-time delivery of reliable and scalable software products
- Transparent communication and agile methodologies

• A collaborative work culture that promotes skill development and quality awareness across departments

ATS views quality as a continuous journey and a core driver of client satisfaction and business success

Product / Service Analysis:

ATS delivers high-value digital solutions tailored to client goals, ensuring agility, scalability, and long-term impact.

Key offerings include:

• Custom Software Development: Tailored to unique client needs with robust architecture and clean UI/UX.

• ERP Solutions: Optimizes enterprise workflows with modular, real-time dashboards.

• Mobile & Web Apps: Cross-platform and native apps that enhance user engagement.

• Cloud Services: Efficient and secure cloud migration and infrastructure management.

• Digital Marketing: Result-driven strategies for brand growth and lead generation.

These services collectively support digital transformation and operational excellence for clients ranging from startups to enterprises.

1.3 SCOPE OF THE PROJECT

The *College Document Management System* is designed to provide a comprehensive and secure digital solution for managing academic documents within an educational institution. Its scope extends to solving the limitations and inefficiencies of manual, paper-based document handling by introducing a structured and user-friendly platform for document submission, organization, and retrieval.

This system is primarily intended for use by students and teachers, each assigned specific roles and permissions within the application. Students are allowed to upload their academic documents—such as assignments, project reports, certificates, and other departmental records—under defined categories, while teachers are provided with administrative controls to manage these categories and oversee the document submissions of students. This role-based system ensures that access is restricted according to the user's designation, thus maintaining data security and preventing unauthorized access.

The scope includes features such as user registration and login, profile creation through an extended user model, categorized document uploads (restricted to PDF files), note attachments to submissions, document updates, and role-specific dashboards. Teachers can create new document categories, activate or deactivate them, and view all submissions made by students. They can also use a built-in search functionality to quickly locate specific students by username or roll number. Students, on the other hand, have access only to their own data and can manage documents efficiently through a dedicated interface.

Another key part of the project scope is the secure handling of data and file uploads. All documents are stored in a structured media directory with proper file validation to ensure only valid file types are accepted. Form validations and permission checks are enforced throughout the system to protect against malicious activity and data misuse. Additionally, feedback messages, error handling, and success confirmations are incorporated for better user experience and interaction.

While the current implementation supports a single-institution setup, the architecture of the system is

designed to be modular and extensible. This allows for future upgrades such as multi-user role expansion (admin-level roles, reviewers, etc.), integration with other academic systems (like student portals or attendance systems), email notifications, and report generation capabilities. The system can also be scaled to support multiple departments or institutions through minor modifications in the user and category management logic.

In summary, the scope of this project is not only limited to providing basic document upload functionality, but also focuses on delivering a structured, secure, and scalable digital platform for academic document management. It demonstrates a practical application of web technologies to replace redundant offline processes and establish a digital workflow that benefits both students and faculty members alike.

1.4 METHODOLOGY

The development of the *College Document Management System* followed a systematic and iterative methodology to ensure that the final application meets both functional and user-centric requirements. The project was approached in multiple well-defined phases: requirement analysis, system design, development, testing, and deployment. Each phase was carefully planned and executed to build a robust and scalable web application that supports the streamlined handling of academic documents between students and teachers.

1.Requirement Analysis

The first phase involved a **requirement analysis**, during which both technical and user-specific needs were identified. Interviews with students and faculty members helped uncover common challenges in document handling, such as delays in submission, lost paperwork, and lack of centralized access. Based on these findings, the project scope was defined, and the necessary features were outlined—including secure user authentication, role-based access control, categorized uploads, and document retrieval mechanisms.

2. Design Phase

In the **design phase**, the system architecture was laid out using the Model-View-Template (MVT) pattern provided by the Django framework. Entity-relationship diagrams (ERDs) were created to visualize the relationship between users, documents, and categories. The user interface was designed with Bootstrap 5 to ensure a responsive and consistent user experience across all devices. Templates for different user roles (student and teacher) were planned to offer tailored dashboards and controls.

3.Development Phase

In the **design phase**, the system architecture was laid out using the Model-View-Template (MVT) pattern provided by the Django framework. Entity-relationship diagrams (ERDs) were created to visualize the relationship between users, documents, and categories. The user interface was designed with Bootstrap 5 to ensure a responsive and consistent user experience across all devices. Templates for different user roles

(student and teacher) were planned to offer tailored dashboards and controls.

4.Testing Phase

In the **testing phase**, the system was subjected to functional and usability testing. Test cases were written to verify the correctness of user authentication, form validation, document uploads, category management, and file download operations. Edge cases, such as duplicate uploads or invalid file formats, were tested to ensure the system handled exceptions gracefully. Feedback was collected from peer users to refine the interface and improve flow.

5.Deployment Phase

In the **deployment phase**, the project was organized for local deployment using Django's built-in development server. Static and media files were served properly, and test data was inserted to demonstrate the complete document management cycle. Although not deployed on a cloud server in this version, the system is ready for hosting on platforms such as PythonAnywhere or Heroku for broader access.

1.5 LITERATURE REVIEW

1. A. Sharma et al., "Design and Implementation of an Online Document Submission System for University Projects", IJCSMC, Vol. 9, Issue 3, 2020.
2. R. Gupta and P. Nair, "Academic Document Management System Using Django Framework", International Journal of Scientific Research in Computer Science, 2021.
3. S. Khan and M. Shaikh, "A Secure Role-Based Access Control Mechanism for Educational Web Applications", International Journal of Web & Semantic Technology, 2019.
4. N. Patel and A. Mehta, "Web-Based Document Handling System for Universities", IEEE Xplore, 2022.
5. T. Lee and Y. Zhang, "Enhancing Educational Workflow through Digital Document Platforms", International Journal of Educational Technology, 2021.

LITERATURE REVIEW

2.1 EXISTING TECHNOLOGICS

The development of the *College Document Management System* leverages several modern web technologies to ensure security, scalability, and ease of use.The backend of the system is built using Django, a high-level Python web framework known for its rapid development capabilities and clean design.

Django's Model-View-Template (MVT) architecture promotes a structured and modular development approach, while its Object Relational Mapper (ORM) simplifies database operations. Authentication, session handling, and form validation are handled efficiently using Django's built-in features.

For the frontend, the system utilizes HTML5, CSS3, and Bootstrap 5, providing a responsive and user-friendly interface that works seamlessly across desktops, tablets, and mobile devices.

Bootstrap components such as tables, forms, buttons, and alerts enhance visual consistency and usability.

User interactions are managed through Django templates, allowing dynamic rendering of personalized content for students and teachers.

For data storage, the system uses SQLite, a lightweight relational database ideal for development and small-scale deployment. File handling is integrated into Django using the FileField, and uploaded documents (PDFs) are stored in the media directory, with validation using Django's FileExtensionValidator. This ensures only appropriate and safe file formats are accepted. Role-based access control is implemented through a custom UserProfile model that extends Django's default user authentication system.

2.2 REVIEW OF RELATED WORK

1. **Sharma et al., 2020** – Proposed a document submission portal for project work using ASP.NET. While the system had document upload and grading features, it lacked category management and fine-grained access control, which our system improves using Django models and role-based views.
2. **Gupta & Nair, 2021** – Built an academic document portal using Django and emphasized its modular structure, built-in authentication, and admin panel. Their system inspired the use of Django's User model and template-based rendering in our project.
3. **Khan & Shaikh, 2019** – Explored role-based access control for educational web applications. Their study showed that user roles must be enforced both at the view and model levels—an approach we adopted through user_type checking in views and templates.
4. **Patel & Mehta, 2022** – Discussed a university document system with category filtering and search features. Their UI concepts guided the layout of teacher dashboards and search functionalities in our system.
5. **Lee & Zhang, 2021** – Showed the impact of digital documentation on academic workflows. Their findings validated the purpose of our system in reducing paper usage and administrative burden by enabling digital submissions.

2.3 SUMMARY

The literature reviewed in the previous sections demonstrates a growing global emphasis on digitalizing academic processes, especially in areas such as document submission, evaluation, and archival. With the increase in student enrollment and the growing complexity of educational administration, manual document handling has become inefficient and error-prone.

Various researchers and developers have proposed and implemented systems that partially automate document workflows within educational institutions. However, most of these systems either focus on specific functionalities—such as assignment upload or project submission—or lack comprehensive control mechanisms for managing user roles and document categories. This creates a gap between what is needed for complete academic documentation workflows and what is currently being delivered in existing solutions.

Most related works confirm the importance of integrating authentication, file validation, and secure access into educational systems. Studies also stress the need for responsive design and intuitive user

experience, especially in systems that serve both students and teachers. However, many platforms reviewed tend to be either too lightweight—lacking administrative control—or overly complex, requiring significant resources and technical expertise to deploy and maintain.

A common limitation observed is the lack of dynamic category creation and management by academic staff, a feature that greatly improves the adaptability and usability of the system in real-world academic environments.The *College Document Management System* developed in this project seeks to overcome these limitations by providing a robust, full-stack web application built on the Django framework.

It integrates essential features such as user registration, role-based access control, PDF-only document uploads, note attachments, secure download functionality, and real-time category management. Django's built-in admin panel, ORM, and authentication system allow for quick development without compromising security or data integrity.

The use of Bootstrap ensures the interface is visually appealing and responsive, while SQLite provides lightweight yet reliable storage during development.

In conclusion, the review of existing literature has reinforced the relevance and necessity of developing a centralized document management system specifically tailored for academic institutions. The findings have guided the technical choices and feature design of this project, ensuring that it not only meets the immediate needs of digital documentation but also lays the foundation for future scalability, modularity, and institutional adoption. This project builds upon the strengths and limitations observed in previous systems and demonstrates a practical, secure, and efficient approach to modern academic document management.

SYSTEM ANALYSIS

3.1 PROBLEM STATEMENT

In the current academic landscape, document handling in many educational institutions is still dependent on outdated methods such as physical submissions or unstructured email-based uploads. Students often submit handwritten or printed copies of assignments, project reports, and certificates, which are stored manually or in inconsistent digital formats. This process is not only time-consuming but also vulnerable to misplacement, duplication, and lack of accountability. Teachers, on the other hand, face challenges in organizing and tracking large volumes of student submissions, especially without a structured system to categorize, retrieve, and validate documents efficiently.

The absence of a centralized and digital document management platform creates several problems. There is no standard format for submissions, no control over accepted file types, and no efficient way to track who submitted what, when, and under which category. Students do not get immediate feedback or confirmation of their uploads, while teachers lack the ability to control categories dynamically based on academic timelines. Moreover, document storage becomes chaotic without a defined folder structure or naming convention, leading to confusion and administrative burden during audits or internal reviews. Existing Learning Management Systems (LMS) or cloud drives are either too generic or costly, and they rarely offer role-based access or category-level control specific to academic documentation. They are also

not tailored to the needs of educational workflows where both students and faculty need distinct but interconnected functionality.
To solve these issues, the *College Document Management System* provides a focused, secure, and user-friendly solution. It allows students to upload categorized PDF documents through their dashboard, and teachers to manage document categories, access submissions, and control academic document flow. It features role-based access control, format validation, PDF-only enforcement, and custom dashboards. This project addresses real problems faced in academic institutions and provides a practical solution using modern web technologies.

3.2 REQUIREMENT SPECIFICATION

The functional and non-functional requirements of the *College Document Management System* are outlined to guide the complete system design and development process. These requirements help ensure the application is robust, reliable, and fit for its academic use case.

Functional Requirements

1. User Registration and Login
    - Students must be able to register and log in using their credentials.
    - Teachers must log in using predefined accounts.
2. Role-Based Dashboard Views
    - Students access their own dashboard to upload documents.
    - Teachers can manage categories and view all student submissions.
3. Document Upload with Notes
    - Students upload only PDF files under selected categories.
    - Optional notes can be added with each document.
4. Category Management by Teachers
    - Teachers can create, activate, or deactivate document categories.
5. Student Search and Document Filtering
    - Teachers can search by roll number or username to filter submissions.
6. Document Download Access
    - Teachers and students can download documents securely.

Non-Functional Requirements

1. Performance and Speed
    - Pages should load quickly and respond within acceptable limits.
2. Security and Permissions
    - User roles must be strictly enforced using Django's authentication system.
    - Only authorized users should access protected routes.
3. Cross-Browser and Cross-Device Support
    - The platform must work consistently on Chrome, Firefox, Edge, and mobile devices.
4. File Format Control
    - The system must accept only valid PDF files and reject others.
5. User-Friendly UI
    - Navigation, feedback messages, and form handling should be intuitive.
6. Future Maintainability

- Code should be modular, with room for expansion like department filters, upload deadlines, or notification systems.

3.3 TOOLS AND TECHNOLOGIES USED

Django(Python Web Framework)

Django serves as the backend foundation for the application. It is used for managing authentication, user sessions, database operations, and routing. Django's Model-View-Template (MVT) structure separates the logic, interface, and data handling, enabling clean and scalable code organization. Models are defined for users, documents, and categories. Django's built-in admin panel helps teachers or administrators manage data with ease, while views are protected by decorators like @login_required and role checks for added security.

SQLite(Database)

SQLite is used as the database engine during development. It stores user data, uploaded document metadata, and category records. It integrates directly with Django's ORM, requiring no additional configuration and supporting fast read/write operations. It is ideal for lightweight deployment and testing and can be replaced with PostgreSQL or MySQL in production.

HTML & Django Templates

HTML is used along with Django's templating language to create dynamic content based on user roles. Templates are responsible for rendering forms, displaying flash messages, and showing uploaded documents. They are conditionally rendered depending on whether the logged-in user is a student or teacher.

CSS & Bootstrap 5

CSS and Bootstrap 5 are used for styling the application. Bootstrap components help create responsive layouts and user-friendly interfaces. Cards, tables, buttons, and modals are styled using Bootstrap classes to keep the UI clean and consistent.

JavaScript (Optional Enhancements)

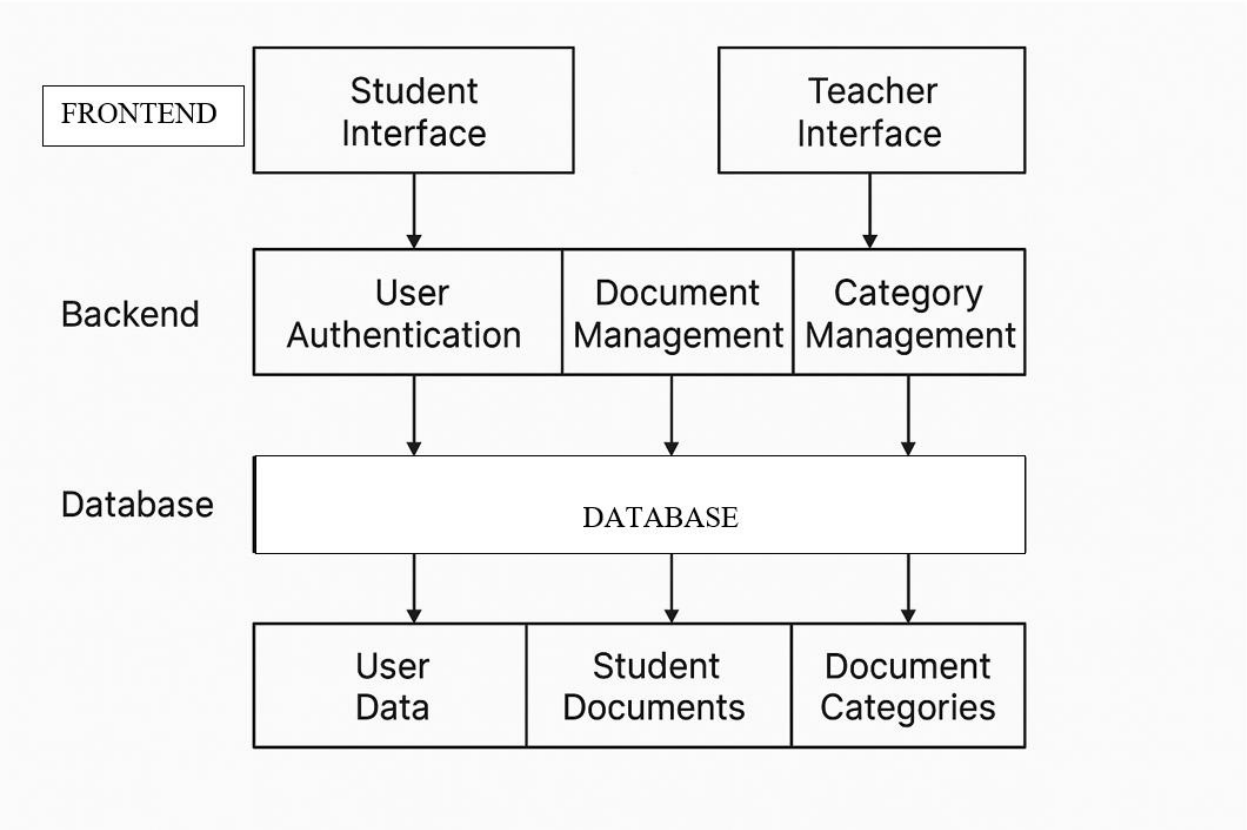
JavaScript is used for small frontend enhancements like alert dismissal, table interactions, or future real-time validation. While Django handles most functionality server-side, JS improves the overall user experience.

Visual Studio Code (IDE)

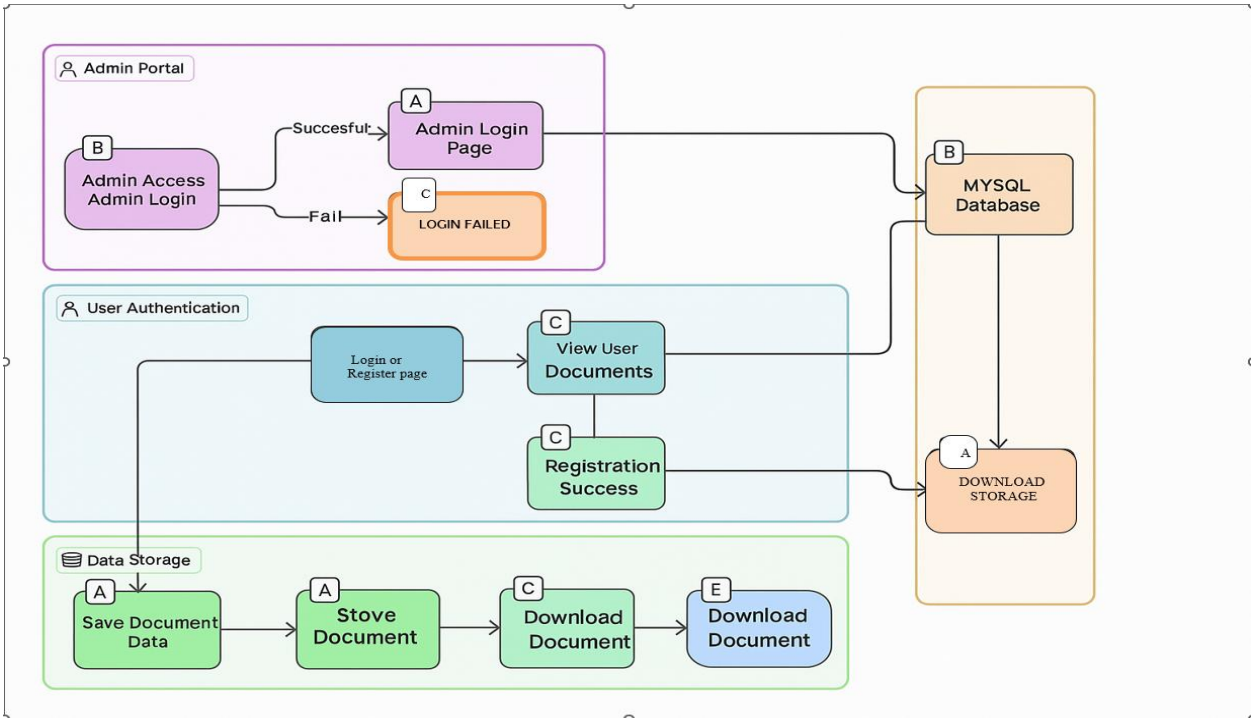
VS Code is used as the primary development environment due to its integrated terminal, Git support, syntax highlighting, and extensions for Django and Python development.

SYSTEM DESIGN

4.1 SYSTEM ARCHITECTURE



4.2 DATA FLOW DIAGRAM



4.3 DATABASE DESIGN

The database design serves as the backbone of the College Document Management System, enabling secure storage and organized access to user profiles, categorized documents, and administrative metadata. A relational model was chosen using SQLite (or MySQL in production) to ensure consistency, reduce redundancy, and streamline data retrieval for both students and teachers.

This design supports multiple core features such as role-based user access, document uploads by students, category management by teachers, and secure document downloads.

The system consists primarily of four tables: User, UserProfile, DocumentCategory, and StudentDocument. Each table is defined with specific fields tailored to the needs of the application. For instance, the StudentDocument table stores uploaded PDF files with metadata, while the DocumentCategory table manages various types of documents such as assignments, ID cards, and mark sheets. Relationships are established using foreign keys, ensuring data normalization and referential integrity. The following is a detailed breakdown of the StudentDocument table structure.

FIELD NAME	DATA TYPE	DESCRIPTION
id	INTEGER	Unique identifier for each document entry in the database.
student_id	INTEGER	Foreign key referencing the user who uploaded the document.
category_id	INTEGER (FK)	Foreign key referencing the type/category of the document.
document	FILE (PDF)	Path to the uploaded PDF file stored in the server directory.

FIELD NAME DATA TYPE DESCRIPTION

id	INTEGER	Unique identifier for each document entry in the database.
uploaded_at	DATETIME	Timestamp of when the document was uploaded.
notes	TEXT	Additional information or remarks about the uploaded document.

ADMIN TABLE

FIELD NAME	DATA TYPE	DESCRIPTION
Id	INTEGER	Unique identifier for each admin user in the database.
Full_name	VARCHAR	The admin's complete name as recorded in the system.

FIELD NAME	DATA TYPE	DESCRIPTION
Job_title	VARCHAR	The official designation of the admin (e.g., Faculty, HOD).
Email	VARCHAR	The admin's official email for login and communication.
Phone	INTEGER	The admin's contact number for verification and messaging.

USER TABLE

FIELD NAME	DATA TYPE	DESCRIPTION
Id	INTEGER	Unique identifier for each student user.
Username	VARCHAR	The student's login username for accessing the system.
Password	VARCHAR	The encrypted password used for authentication.

4.4. USER INTERFACE DESIGN

The user interface (UI) of the *College Document Management System* is designed to be clean, responsive, and role-specific, offering tailored experiences for students and teachers. Built using HTML, CSS, Bootstrap 5, and Django templating, the interface ensures ease of navigation, accessibility, and functionality across all devices, including desktops, laptops, tablets, and mobile phones.

Login and Registration Pages

The landing interface presents users with options to either log in or register. The login form accepts valid credentials and uses secure Django authentication methods. The student registration form includes additional fields like roll number and email, with real-time validation for existing entries. The UI components are styled with Bootstrap to offer immediate feedback (like success/error messages) and

smooth transitions.

Student Dashboard

Once logged in, students are redirected to a personal dashboard. The interface includes:

- A welcoming message with the student's name and user type.
- A summary table of uploaded documents.
- Buttons for uploading new documents, selecting categories, and writing notes.
- Alerts to indicate successful uploads or errors.
  The document upload form uses dropdown menus for category selection, file pickers limited to .pdf, and text areas for additional notes.

Teacher Dashboard

Teachers, upon logging in, access a more administrative interface. It includes:

- A searchable list of registered students (by username or roll number).
- Links to view and download each student's documents.
- A category management panel to add, enable, or disable document types.

Category Management

This section allows teachers to manage the list of document categories. A form interface enables teachers to:

- Add new categories with name and description.
- Toggle their active/inactive status.
- View a table of all categories with timestamps and creator information.

IMPLEMENTATION

5.1MODULE DESCRIPTION

The College Document Management System is logically divided into multiple interrelated modules, each of which contributes to the overall functionality, usability, and robustness of the application. Below is a detailed description of the key modules:

1. User Authentication Module

This module manages user registration, login, and session handling. It differentiates between two user roles: **Student** and **Teacher**.

- **Student Registration**: Students can register with a unique roll number, username, email, and password. Upon successful registration, their profile is saved in the UserProfile model with user type marked as "student".
- **Login/Logout**: Both students and teachers can log in using Django's built-in authentication system. Logged-in users are redirected to role-based dashboards. Secure logout ensures user session ends appropriately.
- **Admin (Teacher) Login**: Teachers can log in and access administrative features such as category management and document oversight.

2. Dashboard Module

Based on user roles, the dashboard module provides customized interfaces:

- **Student Dashboard**: Displays the user's uploaded documents, available categories, and a form to

upload new documents.

- **Teacher Dashboard**: Displays a searchable list of students, links to view their documents, and access to manage document categories.

3. Document Upload and Management Module

This core module enables students to upload academic or project-related documents under predefined categories.

- **Upload Function**: Only PDF files are allowed. The form validates and stores the file along with notes and the selected category.
- **Update Logic**: If a document for the same category already exists, it is replaced with the new file.
- **Storage**: All documents are securely stored in the /media/documents/ directory and mapped in the database via the StudentDocument model.

4. Category Management Module

This module is accessible only to teachers (admins).

- **Create Category**: Teachers can add new document categories with a name and optional description.
- **Toggle Activation**: Categories can be activated or deactivated without deleting them. Inactive categories are hidden from student dashboards.
- **View & Edit**: All categories are displayed in a management interface with toggle buttons.

5. Document Viewing and Downloading Module

This module ensures that:

- **Teachers** can view and download documents submitted by students.
- **Students** can view or download only their own documents.
- Downloaded files are served in PDF format using Django's HttpResponse.

6. Admin Interface Module

Using Django's admin panel, the system offers a graphical interface for managing:

- **Users and Profiles** (UserProfile)
- **Document Categories**
- **Uploaded Documents**
  The admin can perform CRUD operations with filtering and searching capabilities.

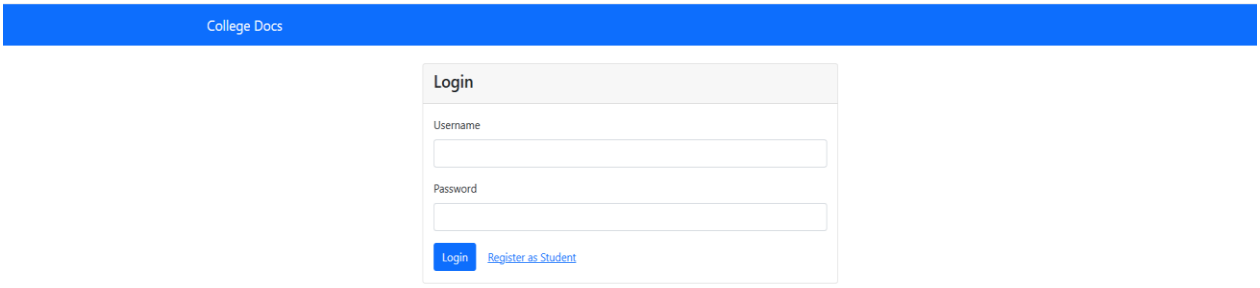
5.2 SYSTEM MODULES AND SCREENSHOTS

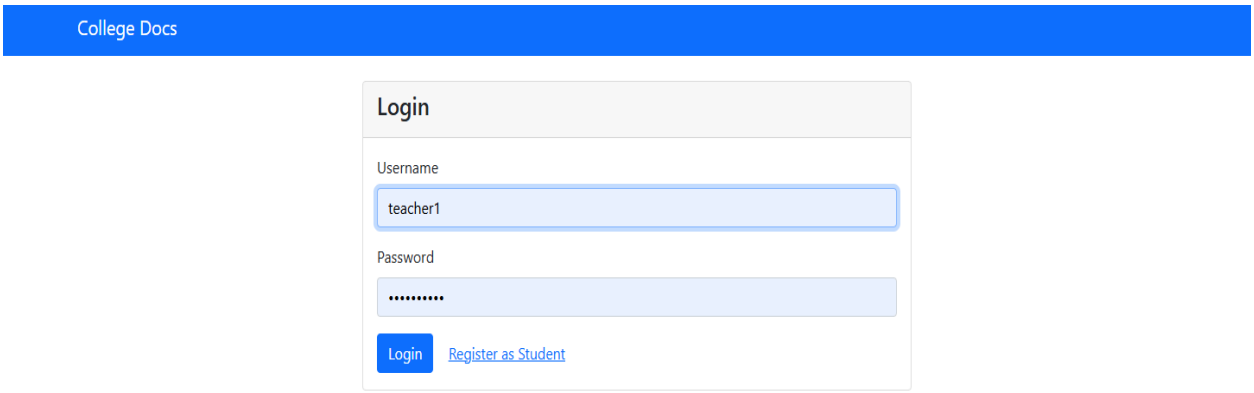
1. Authentication Module

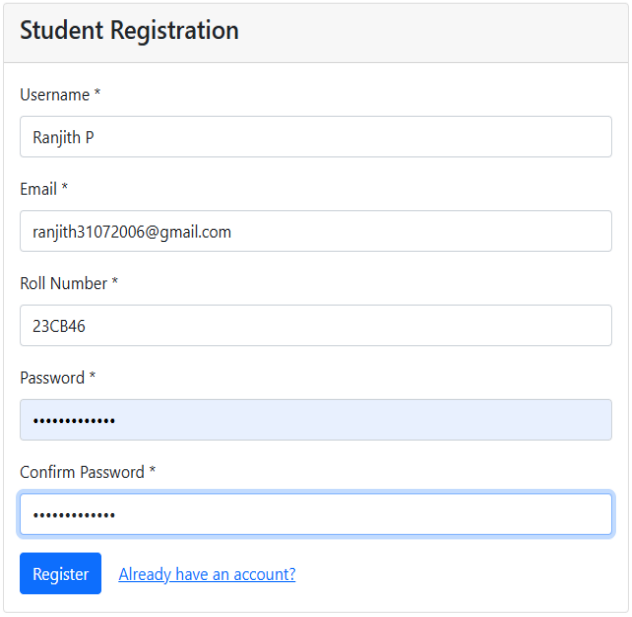
This module handles user access control. It authenticates users during login and ensures secure session management. It also enforces user role separation (Student and Teacher) and controls what each role can view or perform. Features include:

- Secure login and logout
- Student registration with unique roll numbers

- Role-based redirection after login





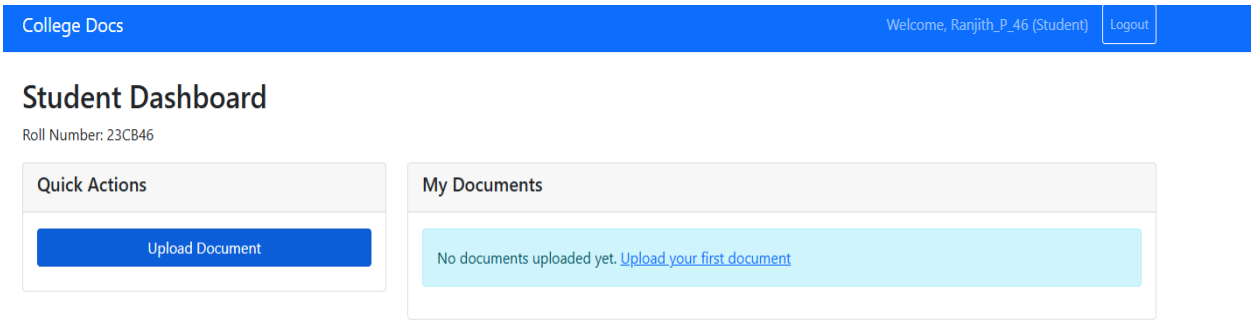


2. Dashboard Module

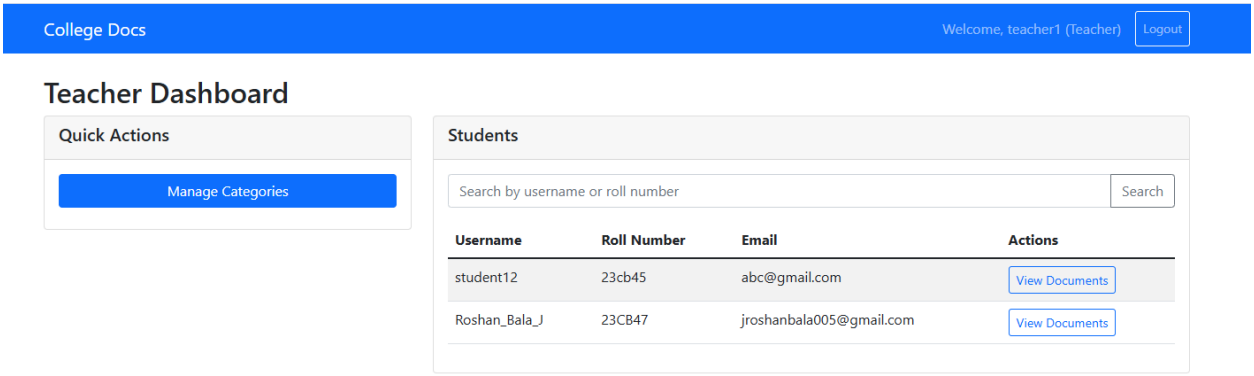
The dashboard provides role-specific functionality:

- **Student Dashboard**: View uploaded documents, choose categories, and upload files
- **Teacher Dashboard**: Search students, view their uploads, and manage categories

STUDENT DASHBOARD



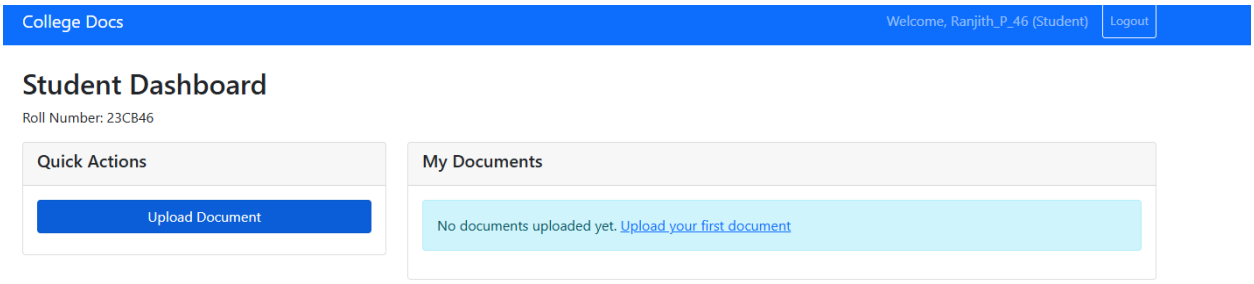
TEACHER DASHBOARD

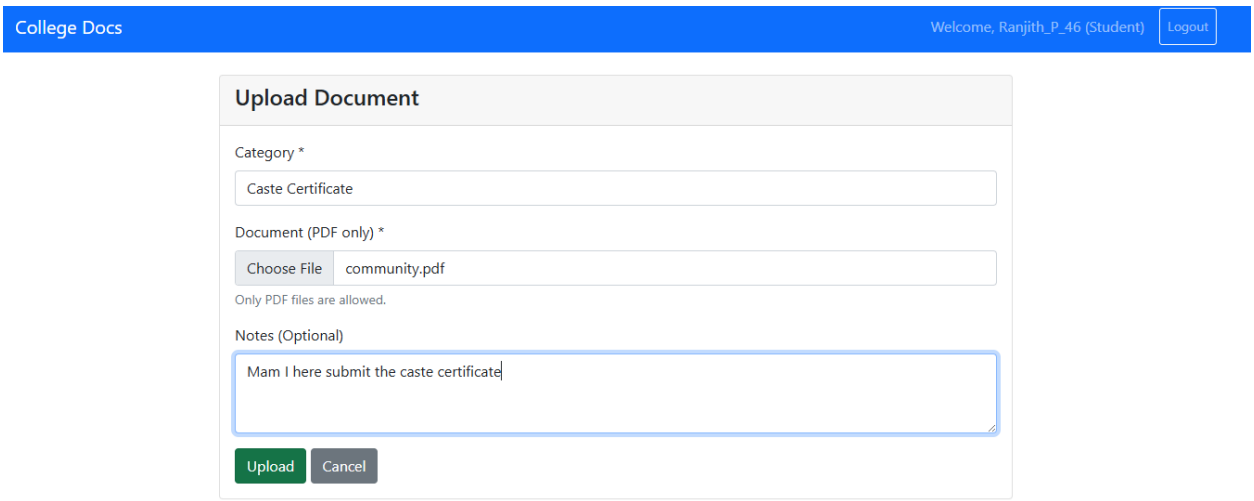


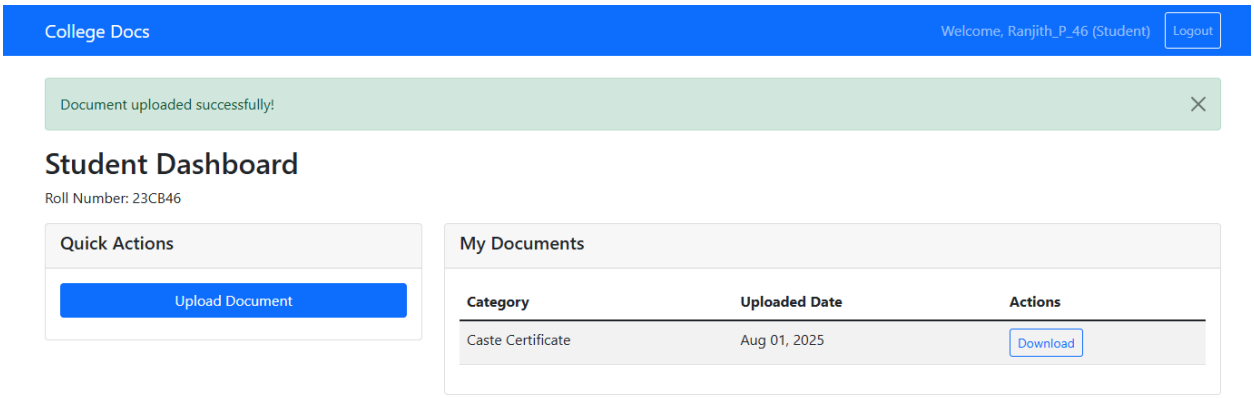
3. Document Upload Module

Students use this module to upload their academic documents. Key highlights include:

- File validation (PDF only)
- Upload per category (one document per category per student)
- Automatic replacement if a document already exists in a category
- Form inputs for optional notes and descriptions


College Docs
Welcome, Ranjith_P_46 (Student)
Logout
Student Dashboard
Roll Number: 23CB46
Quick Actions
Upload Document
My Documents
No documents uploaded yet. Upload your first document

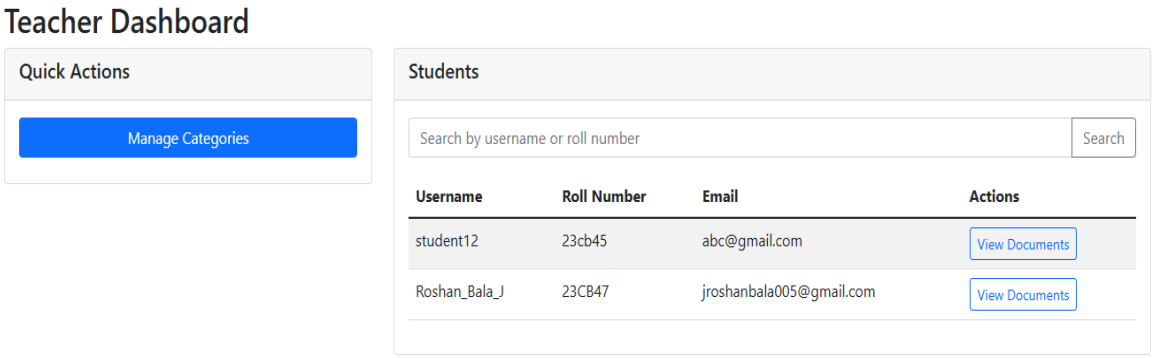

College Docs
Welcome, Ranjith_P_46 (Student)
Logout
Upload Document
Category *
Caste Certificate
Document (PDF only) *
Choose File
community.pdf
Only PDF files are allowed.
Notes (Optional)
Mam I here submit the caste certificate
Upload
Cancel


College Docs
Welcome, Ranjith_P_46 (Student)
Logout
Document uploaded successfully!
Student Dashboard
Roll Number: 23CB46
Quick Actions
Upload Document
My Documents
Category
Uploaded Date
Actions
Caste Certificate
Aug 01, 2025
Download

4. Document Storage and Retrieval Module

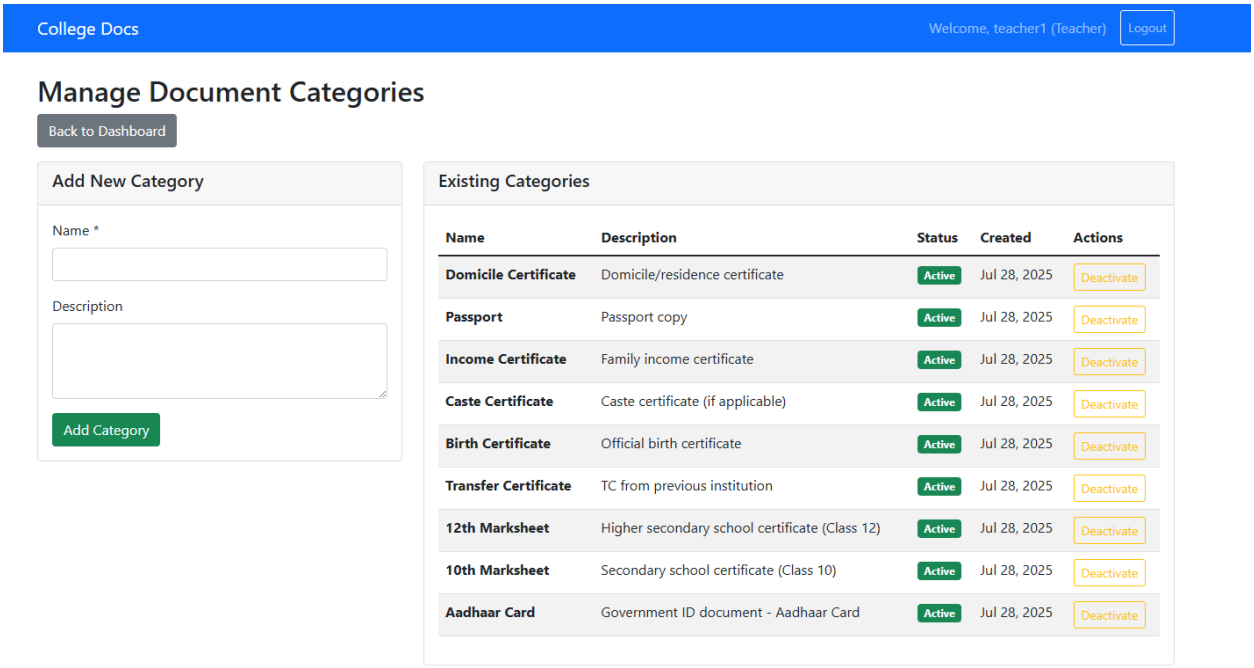
Uploaded documents are stored on the server and indexed in the database. Both students and teachers can access documents based on their permissions:

- Students: can view and download their own documents
- Teachers: can view and download documents of all students



5. Category Management Module

Accessible only by teachers, this module allows for the dynamic creation, activation, and deactivation of document categories. It helps in classifying student submissions into organized types such as Assignments, Reports, Certificates, etc.



5.4 CODE SNIPPETS

Manage,py

```
#!/usr/bin/env python
"""Django's command-line utility for administrative tasks."""
import os
import sys

def main():
    """Run administrative tasks."""
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'college_docs.settings')
    try:
        from django.core.management import execute_from_command_line
    except ImportError as exc:
        raise ImportError(
            "Couldn't import Django. Are you sure it's installed and "
            "available on your PYTHONPATH environment variable? Did you "
            "forget to activate a virtual environment?"
        ) from exc
    execute_from_command_line(sys.argv)

if __name__ == '__main__':
    main()
```

asgi.py

```
"""
ASGI config for college_docs project.

It exposes the ASGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/asgi/
"""

import os

from django.core.asgi import get_asgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'college_docs.settings')

application = get_asgi_application()
```

settings.py

```
"""
```

```
Django settings for college_docs project.

Generated by 'django-admin startproject' using Django 4.2.7.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/topics/settings/

For the full list of settings and their values, see
https://docs.djangoproject.com/en/4.2/ref/settings/
"""
import os
from pathlib import Path

# Build paths inside the project like this: BASE_DIR / 'subdir'.
BASE_DIR = Path(__file__).resolve().parent.parent

# Quick-start development settings - unsuitable for production
# See https://docs.djangoproject.com/en/4.2/howto/deployment/checklist/

# SECURITY WARNING: keep the secret key used in production secret!
SECRET_KEY = 'django-insecure-_*3p%eu008sei5f7vk4kuv!z051xbi-$n!u=gq3=(h)f_yzo-$'

# SECURITY WARNING: don't run with debug turned on in production!
DEBUG = True

ALLOWED_HOSTS = []

# Application definition

INSTALLED_APPS = [
    'django.contrib.admin',
    'django.contrib.auth',
    'django.contrib.contenttypes',
    'django.contrib.sessions',
    'django.contrib.messages',
    'django.contrib.staticfiles',
    'documents'
]

MIDDLEWARE = [
    'django.middleware.security.SecurityMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    'django.middleware.common.CommonMiddleware',
    'django.middleware.csrf.CsrfViewMiddleware',
```

```
    'django.contrib.auth.middleware.AuthenticationMiddleware',
    'django.contrib.messages.middleware.MessageMiddleware',
    'django.middleware.clickjacking.XFrameOptionsMiddleware',
]

ROOT_URLCONF = 'college_docs.urls'

TEMPLATES = [
    {
        'BACKEND': 'django.template.backends.django.DjangoTemplates',
        'DIRS': [BASE_DIR / 'templates'],
        'APP_DIRS': True,
        'OPTIONS': {
            'context_processors': [
                'django.template.context_processors.debug',
                'django.template.context_processors.request',
                'django.contrib.auth.context_processors.auth',
                'django.contrib.messages.context_processors.messages',
            ],
        },
    },
]

WSGI_APPLICATION = 'college_docs.wsgi.application'

# Database
# https://docs.djangoproject.com/en/4.2/ref/settings/#databases

DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.sqlite3',
        'NAME': BASE_DIR / 'db.sqlite3',
    }
}

# Password validation
# https://docs.djangoproject.com/en/4.2/ref/settings/#auth-password-validators

AUTH_PASSWORD_VALIDATORS = [
    {
        'NAME': 'django.contrib.auth.password_validation.UserAttributeSimilarityValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.MinimumLengthValidator',
```

```
    },
    {
        'NAME': 'django.contrib.auth.password_validation.CommonPasswordValidator',
    },
    {
        'NAME': 'django.contrib.auth.password_validation.NumericPasswordValidator',
    },
]

# Internationalization
# https://docs.djangoproject.com/en/4.2/topics/i18n/

LANGUAGE_CODE = 'en-us'

TIME_ZONE = 'UTC'

USE_I18N = True

USE_TZ = True

# Static files (CSS, JavaScript, Images)
# https://docs.djangoproject.com/en/4.2/howto/static-files/

STATIC_URL = 'static/'
STATICFILES_DIRS = [BASE_DIR / 'static']

MEDIA_URL = '/media/'
MEDIA_ROOT = BASE_DIR / 'media'
# Default primary key field type
# https://docs.djangoproject.com/en/4.2/ref/settings/#default-auto-field

DEFAULT_AUTO_FIELD = 'django.db.models.BigAutoField'

LOGIN_URL = 'login'
LOGIN_REDIRECT_URL = 'dashboard'
LOGOUT_REDIRECT_URL = 'login'
```

Urls.py

```
"""
URL configuration for college_docs project.

The `urlpatterns` list routes URLs to views. For more information please see:
    https://docs.djangoproject.com/en/4.2/topics/http/urls/
Examples:
```

```
Function views
    1. Add an import:  from my_app import views
    2. Add a URL to urlpatterns:  path('', views.home, name='home')
Class-based views
    1. Add an import:  from other_app.views import Home
    2. Add a URL to urlpatterns:  path('', Home.as_view(), name='home')
Including another URLconf
    1. Import the include() function: from django.urls import include, path
    2. Add a URL to urlpatterns:  path('blog/', include('blog.urls'))
"""
from django.contrib import admin
from django.urls import path, include
from django.conf import settings
from django.conf.urls.static import static

urlpatterns = [
    path('admin/', admin.site.urls),
    path('', include('documents.urls')),
]

if settings.DEBUG:
    urlpatterns += static(settings.MEDIA_URL, document_root=settings.MEDIA_ROOT)
```

wsgi.py

```
"""
WSGI config for college_docs project.

It exposes the WSGI callable as a module-level variable named ``application``.

For more information on this file, see
https://docs.djangoproject.com/en/4.2/howto/deployment/wsgi/
"""

import os

from django.core.wsgi import get_wsgi_application

os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'college_docs.settings')

application = get_wsgi_application()
```

forms.py

```
from django import forms
from django.contrib.auth.forms import UserCreationForm
```

```
from django.contrib.auth.models import User
from django.core.exceptions import ValidationError
from .models import UserProfile, StudentDocument, DocumentCategory

class StudentRegistrationForm(UserCreationForm):
    email = forms.EmailField(
        required=True,
        widget=forms.EmailInput(attrs={'class': 'form-control'})
    )
    roll_number = forms.CharField(
        max_length=20,
        required=True,
        widget=forms.TextInput(attrs={'class': 'form-control'})
    )

    class Meta:
        model = User
        fields = ('username', 'email', 'password1', 'password2')

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.fields['username'].widget.attrs.update({'class': 'form-control'})
        self.fields['password1'].widget.attrs.update({'class': 'form-control'})
        self.fields['password2'].widget.attrs.update({'class': 'form-control'})

    def clean_email(self):
        email = self.cleaned_data.get('email')
        if User.objects.filter(email=email).exists():
            raise ValidationError("A user with this email already exists.")
        return email

    def clean_roll_number(self):
        roll_number = self.cleaned_data.get('roll_number')
        if UserProfile.objects.filter(roll_number=roll_number).exists():
            raise ValidationError("A student with this roll number already exists.")
        return roll_number

    def save(self, commit=True):
        user = super().save(commit=False)
        user.email = self.cleaned_data['email']
        if commit:
            user.save()
            UserProfile.objects.create(
                user=user,
```

```
            user_type='student',
            roll_number=self.cleaned_data['roll_number']
        )
    return user

class DocumentUploadForm(forms.ModelForm):
    class Meta:
        model = StudentDocument
        fields = ['category', 'document', 'notes']
        widgets = {
            'category': forms.Select(attrs={'class': 'form-control'}),
            'document': forms.FileInput(attrs={
                'class': 'form-control',
                'accept': '.pdf'
            }),
            'notes': forms.Textarea(attrs={
                'class': 'form-control',
                'rows': 3
            }),
        }

class DocumentCategoryForm(forms.ModelForm):
    class Meta:
        model = DocumentCategory
        fields = ['name', 'description']
        widgets = {
            'name': forms.TextInput(attrs={'class': 'form-control'}),
            'description': forms.Textarea(attrs={
                'class': 'form-control',
                'rows': 3
            }),
        }
```

Models.py

```
from django.db import models
from django.contrib.auth.models import User
from django.core.validators import FileExtensionValidator

class UserProfile(models.Model):
    USER_TYPES = (
        ('student', 'Student'),
        ('teacher', 'Teacher'),
    )
```

```
    user = models.OneToOneField(User, on_delete=models.CASCADE)
    user_type = models.CharField(max_length=10, choices=USER_TYPES)
    roll_number = models.CharField(max_length=20, blank=True, null=True)

    def __str__(self):
        return f"{self.user.username} - {self.user_type}"
# Create your models here.

class DocumentCategory(models.Model):
    name = models.CharField(max_length=100, unique=True)
    description = models.TextField(blank=True)
    is_active = models.BooleanField(default=True)
    created_by = models.ForeignKey(User, on_delete=models.CASCADE)
    created_at = models.DateTimeField(auto_now_add=True)

    class Meta:
        verbose_name_plural = "Document Categories"

    def __str__(self):
        return self.name

class StudentDocument(models.Model):
    student = models.ForeignKey(User, on_delete=models.CASCADE, related_name='documents')
    category = models.ForeignKey(DocumentCategory, on_delete=models.CASCADE)
    document = models.FileField(
        upload_to='documents/',
        validators=[FileExtensionValidator(allowed_extensions=['pdf'])]
    )
    uploaded_at = models.DateTimeField(auto_now_add=True)
    notes = models.TextField(blank=True)

    class Meta:
        unique_together = ['student', 'category']

    def __str__(self):
        return f"{self.student.username} - {self.category.name}"
```

Urls.py

```
from django.urls import path
from django.contrib.auth import views as auth_views
from . import views

urlpatterns = [
    path('', views.dashboard, name='dashboard'),
    path('login/', auth_views.LoginView.as_view(template_name='login.html'), name='login'),
```

```
    path('logout/', views.custom_logout, name='logout'),
    path('register/', views.register, name='register'),
    path('upload/', views.upload_document, name='upload_document'),
    path('student/<int:student_id>/', views.view_student_documents, name='view_student_documents'),
    path('download/<int:document_id>/', views.download_document, name='download_document'),
    path('categories/', views.manage_categories, name='manage_categories'),
    path('categories/toggle/<int:category_id>/', views.toggle_category, name='toggle_category'),
]
```

View.py

```
from django.shortcuts import render, redirect, get_object_or_404
from django.contrib.auth.decorators import login_required
from django.contrib.auth import login, logout
from django.contrib import messages
from django.http import HttpResponse, Http404
from django.core.exceptions import PermissionDenied
from django.db.models import Q
from django.contrib.auth.models import User
from .models import UserProfile, StudentDocument, DocumentCategory
from .forms import StudentRegistrationForm, DocumentUploadForm, DocumentCategoryForm

def register(request):
    if request.method == 'POST':
        form = StudentRegistrationForm(request.POST)
        if form.is_valid():
            try:
                user = form.save()
                login(request, user)
                messages.success(request, 'Registration successful! Welcome to the system.')
                return redirect('dashboard')
            except Exception as e:
                messages.error(request, f'Registration failed: {str(e)}')
        else:
            messages.error(request, 'Please correct the errors below.')
    else:
        form = StudentRegistrationForm()

    return render(request, 'register.html', {'form': form})

def custom_logout(request):
    logout(request)
    messages.success(request, 'You have been logged out successfully.')
    return redirect('login')
```

```
@login_required
def dashboard(request):
    try:
        profile = request.user.userprofile
    except UserProfile.DoesNotExist:
        if request.user.is_superuser:
            profile = UserProfile.objects.create(user=request.user, user_type='teacher')
        else:
            profile = UserProfile.objects.create(user=request.user, user_type='student')
        messages.info(request, 'Profile created for your account.')

    if profile.user_type == 'student':
        return student_dashboard(request)
    else:
        return teacher_dashboard(request)

@login_required
def student_dashboard(request):
    profile = get_object_or_404(UserProfile, user=request.user)
    if profile.user_type != 'student':
        raise PermissionDenied

    documents = StudentDocument.objects.filter(student=request.user)
    categories = DocumentCategory.objects.filter(is_active=True)

    context = {
        'documents': documents,
        'categories': categories,
        'profile': profile,
    }
    return render(request, 'student_dashboard.html', context)

@login_required
def teacher_dashboard(request):
    profile = get_object_or_404(UserProfile, user=request.user)
    if profile.user_type != 'teacher':
        raise PermissionDenied

    search_query = request.GET.get('search', '')
    students = User.objects.filter(userprofile__user_type='student')

    if search_query:
        students = students.filter(
            Q(username__icontains=search_query) |
```

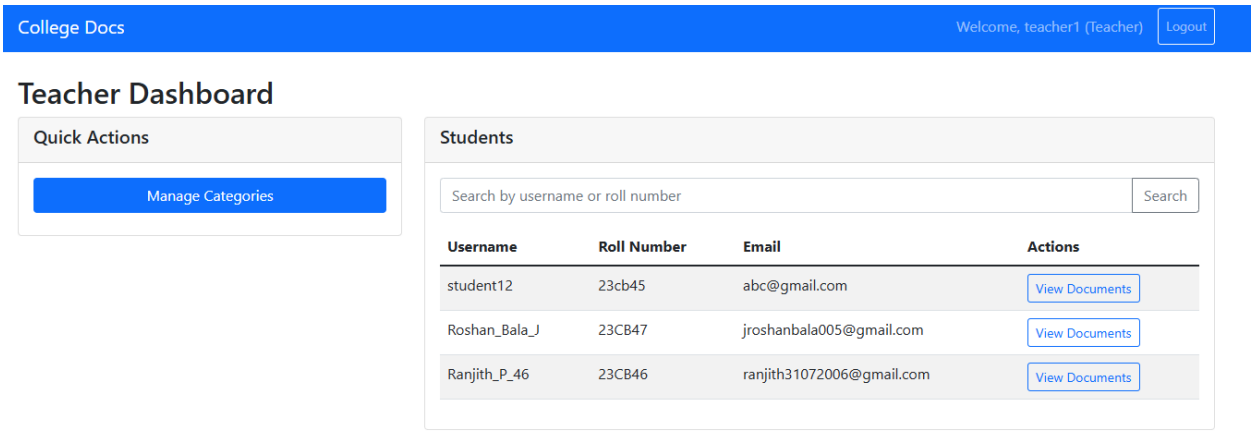
```
            Q(userprofile__roll_number__icontains=search_query)
        )

    context = {
        'students': students,
        'search_query': search_query,
    }
    return render(request, 'teacher_dashboard.html', context)

@login_required
def upload_document(request):
    profile = get_object_or_404(UserProfile, user=request.user)
    if profile.user_type != 'student':
        raise PermissionDenied

    if request.method == 'POST':
        form = DocumentUploadForm(request.POST, request.FILES)
        if form.is_valid():
            document = form.save(commit=False)
            document.student = request.user

            existing = StudentDocument.objects.filter(
                student=request.user,
                category=document.category
            ).first()

            if existing:
                existing.document.delete()
                existing.document = document.document
                existing.notes = document.notes
                existing.save()
                messages.success(request, 'Document updated successfully!')
            else:
                document.save()
                messages.success(request, 'Document uploaded successfully!')

            return redirect('dashboard')
        else:
            messages.error(request, 'Please correct the errors below.')
    else:
        form = DocumentUploadForm()

    return render(request, 'upload_documents.html', {'form': form})
```

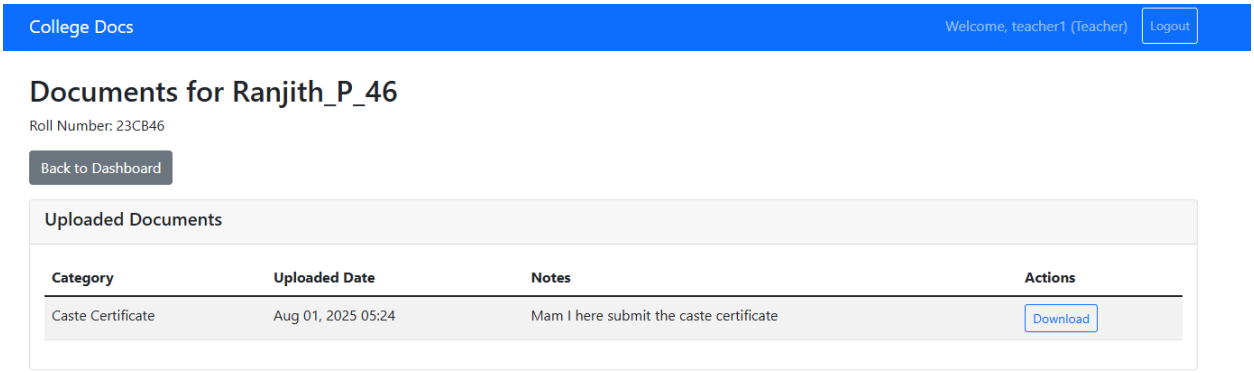
```
@login_required
def view_student_documents(request, student_id):
    profile = get_object_or_404(UserProfile, user=request.user)
    if profile.user_type != 'teacher':
        raise PermissionDenied

    student = get_object_or_404(User, id=student_id, userprofile__user_type='student')
    documents = StudentDocument.objects.filter(student=student)

    context = {
        'student': student,
        'documents': documents,
    }
    return render(request, 'view_student_documents.html', context)

@login_required
def download_document(request, document_id):
    document = get_object_or_404(StudentDocument, id=document_id)

    profile = get_object_or_404(UserProfile, user=request.user)
    if profile.user_type == 'student' and document.student != request.user:
        raise PermissionDenied
    elif profile.user_type != 'teacher' and profile.user_type != 'student':
        raise PermissionDenied

    try:
        response = HttpResponse(document.document.read(), content_type='application/pdf')
        response['Content-Disposition'] = f'attachment;
filename="{document.category.name}_{document.student.username}.pdf"'
        return response
    except FileNotFoundError:
        raise Http404("Document not found")

@login_required
def manage_categories(request):
    profile = get_object_or_404(UserProfile, user=request.user)
    if profile.user_type != 'teacher':
        messages.error(request, 'You do not have permission to access this page.')
        return redirect('dashboard')

    if request.method == 'POST':
        form = DocumentCategoryForm(request.POST)
        if form.is_valid():
            category = form.save(commit=False)
```

```
            category.created_by = request.user
            category.save()
            messages.success(request, 'Category added successfully!')
            return redirect('manage_categories')
        else:            messages.error(request, 'Please correct the errors below.')
    else:
        form = DocumentCategoryForm()

    categories = DocumentCategory.objects.all().order_by('-created_at')

    context = {
        'form': form,
        'categories': categories,
    }
    return render(request, 'manage_categories.html', context)

@login_required
def toggle_category(request, category_id):
    profile = get_object_or_404(UserProfile, user=request.user)
    if profile.user_type != 'teacher':
        raise PermissionDenied

    category = get_object_or_404(DocumentCategory, id=category_id)
    category.is_active = not category.is_active
    category.save()

    status = "activated" if category.is_active else "deactivated"
    messages.success(request, f'Category "{category.name}" {status} successfully!')
    return redirect('manage_categories')
```

RESULTS

6.1OUTPUTS RESULT





6.2 ADVANTAGES AND LIMITATIONS:

Advantages

1. **User-Friendly Interface**
   The system provides a clean and intuitive UI for both students and teachers, ensuring ease of use even for users with minimal technical knowledge.
2. **Role-Based Access**
   The system distinguishes between student and teacher roles, allowing personalized functionalities like uploading documents (students) and managing categories/viewing submissions (teachers).
3. **Efficient Document Management**
   Students can upload, view, and download PDF documents securely under predefined categories, while teachers can monitor all submissions centrally.
4. **Secure Authentication**
   User accounts are protected with Django's built-in authentication system, providing session

management and form validation.

5. **Categorization System**
   Documents are organized into categories created by teachers, allowing structured management and quick access.
6. **Responsive Design**
   The front-end, built with Bootstrap, ensures the application is responsive and works well on desktops, tablets, and smartphones.
7. **Open-Source & Extensible**
   The system is built using Python, Django, and MySQL, making it open-source, easy to maintain, and extendable for future features like approval workflows or notifications.

Limitations

1. **No Document Approval Workflow**
   Currently, documents uploaded by students are immediately visible. There is no mechanism for teachers to approve or reject files before they are finalized.
2. **Limited File Format Support**
   The system only supports PDF files. Users cannot upload other document types like Word, Excel, or images.
3. **No Email Notifications**
   There are no automated notifications for document uploads, approvals, or system alerts which could improve user engagement and system feedback.
4. **No Advanced Search or Filters**
   Document and user searches are basic. There's a lack of filters by date, category, or keyword within documents.
5. **Scalability Constraints**
   As the number of users and documents increases, performance might degrade if hosted on limited infrastructure without optimization.
6. **No Backup or Version Control**
   There is no built-in backup or versioning mechanism. Once a file is uploaded, it cannot be reverted to an earlier version.

CONCLUSION

7.1 CONCLUSION:

The *College Document Management System* provides a robust and efficient solution for managing academic documents in an educational environment. Designed using Django as the backend framework and Bootstrap for the frontend, the platform ensures seamless document submission, retrieval, and categorization. It streamlines the process of document handling by offering students a secure and simple interface to upload their academic files, while allowing teachers to manage categories and view student submissions efficiently.

This system addresses the common challenges of unorganized document storage, manual verification, and limited access to centralized data by introducing a user-friendly web-based interface. The role-based access control ensures that each user, whether a student or a teacher, has the appropriate level of access and functionality. With added features like PDF-only uploads, category management, download support, and responsive UI design, the system promotes accessibility, transparency, and usability.

In conclusion, the project not only demonstrates practical implementation of full-stack development principles but also serves as a scalable foundation that can be extended in the future to support additional functionalities like document approval workflows, real-time notifications, and analytics. It stands as a valuable academic and administrative tool that enhances document management processes in colleges.

7.2 SUGGESTIONS FOR IMPROVEMENT:

While the *College Document Management System* fulfills its core objectives effectively, several enhancements can be implemented to make the platform more robust, user-friendly, and scalable in future iterations:

1. **Document Approval Workflow:**
   Introducing a document verification and approval mechanism where teachers can approve or reject submitted documents would add an extra layer of control and ensure content validity.
2. **Search and Filter Enhancements:**
   Implementing advanced search filters (by date, category, file type, etc.) and a global search bar across the platform would significantly improve accessibility and navigation, especially as data volume grows.
3. **Version Control for Documents:**
   Allowing students to upload new versions of the same document while preserving previous versions would help in tracking updates and changes over time.
4. **Notification System:**
   Email or in-app notifications can alert users about successful uploads, approvals, deadlines, or category updates, thereby increasing engagement and communication.
5. **Teacher Feedback Module:**
   Including the option for teachers to provide comments or feedback on each uploaded document would make the system more interactive and educational.
6. **Role Expansion:**
   Adding roles like "Admin" or "Principal" for overall monitoring and report generation can broaden the platform's applicability and control.
7. **Mobile Responsiveness & App Integration:**
   Although the system is currently responsive, developing a dedicated mobile app could provide a smoother experience for users on smartphones and tablets.
8. **Security Enhancements:**
   Implementing two-factor authentication (2FA), audit logs, and file encryption would further strengthen data security, especially in an academic setup.
9. **Bulk Upload and Export Options:**
   Enabling bulk uploads or CSV exports for documents and user data would support administrative tasks and academic audits.

7.3 FUTURE ENHANCEMENTS:

As the educational environment continues to embrace digital transformation, the *College Document Management System* holds potential for several valuable enhancements to improve its scalability, usability, and impact. Some key future improvements include:

1. **Optical Character Recognition (OCR):**

By incorporating OCR technology, the system could extract text from scanned documents or images, making documents searchable and content more accessible.

2. **Mobile Application Support:**
   Developing a dedicated Android/iOS app for students and teachers will provide more convenience and real-time access to document uploads, approvals, and notifications.
3. **Cloud Storage Integration:**
   Future versions could integrate with cloud storage services like Google Drive or Dropbox, enabling users to sync their documents across platforms and devices.
4. **Multi-language Support:**
   Adding support for regional languages and multilingual interfaces would make the platform more inclusive and accessible to users from diverse backgrounds.
5. **Document Expiry and Reminder System:**
   Setting expiry dates for certain types of documents (like ID cards or certifications) and sending reminders for renewal would benefit colleges that track time-sensitive submissions.
6. **Analytics Dashboard:**
   A dashboard for teachers and administrators showing trends, usage statistics, most active students, or pending reviews could support decision-making and improve oversight.
7. **Blockchain for Document Authentication:**
   In the long term, integrating blockchain technology could help validate and authenticate uploaded academic documents, ensuring tamper-proof verification and transparency.
8. **Collaboration Features:**
   Allowing group document uploads, shared folders, or team access to certain files could support project-based academic submissions.
9. **Accessibility Improvements:**
   Ensuring compatibility with screen readers and implementing accessibility guidelines (WCAG) will make the system usable for differently-abled individuals.

REFERENCES

1. Sakshi Jadhav, *ICT Resume Builder System Using Full Stack Web Development*, International Journal for Research in Applied Science and Engineering Technology (IJRASET), Vol. 11(5), 2023.
2. Vivian Lai, Shenghua Zhong, Xiyuan Zhang, and Maosong Sun, *CareerMapper: An Automated Resume Evaluation Tool*, arXiv preprint arXiv:1605.01722, 2016.
3. Chen Zhang, Zhenzhen Li, Wei Zeng, and Xiaoru Yuan, *ResumeVis: A Visual Analytics System for Resume Data*, arXiv preprint arXiv:1705.11142, 2017.
4. Xiao, Xuefei, et al., *Prototype2Code: End-to-end Front-end Code Generation from Design Prototypes*, arXiv preprint arXiv:2405.09240, May 2024.
5. Alaina G. Levine, *Résumé Template Guidance: Strategies to Highlight Your Skills Effectively*, IEEE Computer Society.