Document Sharing Application in Django Platform

A PROJECT REPORT

Submitted by

Ayush Kumar(10000220011)
Debayan Acharyya(10000220015)
Nihal Kumar Sagar(10000220029)
Nima Dorjee Lama(10000220031)
Kowser Jaman Mollah(10000221064)

Supervised by

Dr. Sayani Mondal
Assistant Professor of Emerging Technology

in partial fulfilment for the award of the degree

of

BACHELOR OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY
Year: 2024

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY, WEST BENGAL

MAULANA ABUL KALAM AZAD UNIVERSITY OF TECHNOLOGY
NH-12 (Old NH-34) Simhat, Haringhata, Nadia 741249, West Bengal

APPENDIX 2

BONAFIDE CERTIFICATE

Certified that this project report **" Document Sharing Application in Django Platform"** is the bonafide work of **"Ayush Kumar(10000220011), Debayan Acharyya(10000220015), Nihal Kumar Sagar(10000220029) , Nima Dorjee Lama(10000220031) , Kowser Jaman Mollah(10000221064) "** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr. Somdatta Chakravortty	Dr. Sayani Mondal
HEAD OF THE DEPARTMENT	**SUPERVISOR**
	Assistant Professor
Information Technology	Emerging Technologies
NH-12 (Old NH-34) Simhat	**NH-12 (Old NH-34) Simhat**
Haringhata, Nadia 741249,	**Haringhata, Nadia 741249,**
West Bengal	**West Bengal**

SIGNATURE

External Examiner:

ACKNOWLEDGEMENT

The success and outcome of this project required a lot of guidance and assistance from many people, and we are extremely fortunate to have their support all along the completion of our project work. Whatever we have done is only due to such guidance and assistance, and we would never forget to thank them.

We owe my profound gratitude to our project guide Dr. Sayani Mondal, who took keen interest in our project work and guided us all along, till the completion of our project work by providing all the necessary information for developing a good system.

Signature Ayush Kumar (10000220011)	Signature Debayan Acharyya (10000220015)
Signature Nihal Kumar Sagar (10000220029)	Signature Nima Dorjee Lama (10000220031)
Signature Kowser Jaman Mollah (10000221064)	

APPENDIX 3

TABLE OF CONTENTS

ABSTRACT

In today's digital age, the need for efficient, secure, and user-friendly document sharing solutions is paramount. This thesis presents the development of a Document Sharing Application using the Django web framework. The application addresses common challenges in document management by providing a robust platform for secure document upload, access control, and real-time collaboration. The primary goal of this project is to create a scalable and intuitive web application that enhances user experience and meets modern document sharing needs. The application supports user authentication, allowing only authorized users to access specific documents. Fine-grained access control mechanisms ensure that users can manage permissions for individual documents, thereby maintaining confidentiality and security. A comprehensive literature review highlights the limitations of existing solutions and sets the stage for our design approach. The methodology section outlines the systematic development process, including system requirements, architecture design, and the technology stack comprising Django, HTML, CSS, JavaScript, and a relational database. The system design and implementation chapters delve into the architectural framework, database schema, user interface design, and integration of core functionalities. Emphasis is placed on the seamless interaction between the frontend and backend, ensuring a cohesive and responsive user experience. The results demonstrate the effectiveness of the Django framework in building a robust document sharing platform. The application is compared with existing solutions, highlighting its strengths and addressing any limitations. Future work suggestions include feature enhancements, scalability improvements, and integration with third-party services. This thesis concludes by summarizing the contributions to the field of web-based document sharing applications and discussing the potential for future research and development. The Document Sharing Application using Django exemplifies a modern approach to secure and efficient document management, offering valuable insights and practical solutions for both developers and users.

1. INTRODUCTION

In the digital era, the creation, storage, and sharing of electronic documents have become fundamental to various domains, including education, business, healthcare, and government. Traditional methods of document sharing, such as email attachments and physical media, often prove inefficient, insecure, and cumbersome. These methods can lead to data breaches, unauthorized access, and significant inefficiencies in managing and collaborating on documents. As a result, there is an increasing demand for more secure, efficient, and user-friendly solutions that can facilitate seamless document sharing[5][6][9].

Web-based document sharing applications have emerged as a promising solution to these challenges. These applications leverage modern web technologies to provide platforms that are both accessible and robust. Among the various technologies available, the Django web framework stands out for its simplicity, flexibility, and powerful built-in features[10][13]. Django's modular architecture and extensive ecosystem of third-party packages allow developers to build scalable and maintainable applications that meet a wide range of requirements[11]. By using Django, developers can create document sharing applications that offer enhanced security, ease of use, and comprehensive functionality, including version control, collaborative editing, and secure access control[8][15].

The motivation for this project is to develop a document sharing application that addresses the shortcomings of existing solutions. The goal is to create a platform that combines user-friendly interfaces with robust security features, enabling users to share and manage documents effectively. The application will support essential functionalities such as user authentication, document upload and download, access control, real-time collaboration, and notifications[12][14]. By leveraging Django's capabilities, this project aims to demonstrate the potential of modern web technologies in creating efficient, secure, and scalable document sharing applications that can be used in various contexts to enhance collaboration and document management[7][16].

1.1 Problem Statement

Despite the availability of numerous document sharing systems, many fail to adequately address the critical needs of security, user-friendliness, and efficient document management. Traditional methods, such as email attachments and basic file transfer protocols, are prone to security vulnerabilities, lack efficient access control, and are cumbersome for users, leading to inefficiencies and potential data breaches. These methods do not offer real-time collaboration, version control, or the ability to manage permissions effectively, making them unsuitable for environments that require strict data management and secure sharing capabilities.

Current web-based document sharing solutions also present significant challenges. Many are either overly complex, deterring users with non-intuitive interfaces, or insufficiently secure, exposing sensitive documents to unauthorized access. These limitations result in poor user experience and heightened security risks. There is a clear need for a more robust solution that provides an intuitive user interface while ensuring comprehensive security features. The aim of this project is to develop a Document Sharing Application using the Django framework, designed to offer secure document upload and download, granular access control, real-time collaboration, and efficient document management, addressing the gaps present in existing solutions.

1.2 Scope of the Study

This study focuses on the design, development, and implementation of a web-based Document Sharing Application using the Django web framework. The application will encompass essential features such as secure user authentication and authorization, document upload and download, fine-grained access control, and a rating system for user feedback. By leveraging Django's robust capabilities, the project aims to create a scalable and user-friendly platform that addresses the critical needs of efficient document management and secure sharing. The study will also explore the integration of encryption methods to ensure data security during storage and transmission, as well as the implementation of a responsive and intuitive user interface using modern frontend technologies like HTML, CSS, and JavaScript.

Additionally, the scope of the study includes comprehensive unit testing of the application to ensure its functionality and reliability. Unit testing will be conducted at various stages of development to test individual components, functions, and modules, ensuring that each part of the application performs as expected. The project will employ industry-standard testing frameworks and methodologies to validate the correctness and robustness of the codebase. The study will document the unit testing process, including test cases, test results, and any issues encountered during testing, providing insights into the quality and stability of the application. Future enhancement possibilities will be identified based on unit testing feedback, offering a roadmap for continuous improvement and maintenance of the application post-deployment.

1.3 Structure of the Thesis

The thesis is organized into ten chapters, starting with an introduction to the problem and objectives, followed by a comprehensive literature review. The methodology chapter outlines the system requirements and design principles. The Chapter 4 includes the detail of the system design and implementation, Chapter 5 and 6 includes frontend and backend development respectively, Chapter 7 includes testing and Chapter 8 includes discussion of results.

2. LITERATURE REVIEW

2.1 Overview of Document Sharing Systems

Document sharing systems have evolved significantly with advancements in technology. Traditional methods like email attachments and FTP servers have given way to cloud-based solutions and enterprise document management systems. These modern systems offer a range of features such as version control, collaborative editing, access control, and integration with other productivity tools. For instance, Google Drive provides real-time collaboration and version history tracking, while SharePoint offers extensive document management capabilities for enterprises [1][3].

2.2 Existing Solutions and Technologies

Existing solutions in document sharing encompass a variety of platforms and technologies. Google Drive offers cloud-based storage with real-time collaboration and robust security features. Dropbox provides cross-platform file sharing, synchronization, and advanced security measures like encryption. Microsoft OneDrive integrates seamlessly with Windows and Office 365, offering automatic backup, version control, and compliance features. SharePoint is a comprehensive document management platform for enterprises, with features such as workflow automation, access control, and integration with Microsoft 365 apps. Open-source solutions like Nextcloud and OwnCloud offer customizable, self-hosted options for document sharing and collaboration[2][4].

2.3 Comparative Analysis of Frameworks

Django stands out for its comprehensive feature set, security provisions, and scalability, making it a robust choice for developing document sharing applications. Its built-in ORM streamlines database interactions, while the admin interface automates administrative tasks. Flask offers flexibility and simplicity, ideal for smaller-scale projects or rapid prototyping, although it may require additional libraries for complex functionalities. Ruby on Rails prioritizes convention and developer productivity but may have a steeper learning curve. Node.js and Express.js are well-suited for real-time applications but may require more manual configuration. Laravel's elegant syntax and developer-friendly features make it a strong contender for PHP developers, but its performance may vary. The framework choice hinges on factors like project complexity, developer expertise, scalability needs, and ecosystem preferences[1][3].

2.4 Key Challenges in Document Sharing

Despite the advancements in document sharing technologies, several challenges persist. Security remains a top concern, with data privacy, encryption, and access control being critical areas. Scalability is another challenge, especially for platforms handling large volumes of data and users. User experience plays a crucial role in adoption, requiring intuitive interfaces and seamless collaboration features. Interoperability with third-party services and compliance with regulatory standards add complexity to document sharing systems[2][4].

2.5 Objectives

a) Develop a User-Friendly Interface - Create an intuitive and easy-to-navigate user interface that allows users to efficiently upload, manage, and share documents without extensive technical knowledge.

b) Implement Secure User Authentication and authorization - Ensure that only authorized users can access the application through robust authentication mechanisms. Implement fine-grained authorization to control user access to specific documents and features.

c) Enable Secure Document Upload and Download - Facilitate the secure uploading and downloading of documents, ensuring that data is encrypted and protected during transit and storage to prevent unauthorized access and data breaches.

d) Provide Fine-Grained Access Control - Develop a comprehensive access control system that allows document owners to set and manage permissions for individual documents, enabling secure sharing and collaboration.

e) Integrate a Rating System - Implement a rating system that allows users to rate and review documents. This feature will help users assess the quality and relevance of documents and provide feedback to document owners.

f) Conduct Comprehensive Testing and Evaluation - Perform thorough testing of the application, including unit tests, to ensure functionality, performance, and security. Gather user feedback to refine and improve the application.

g) Document the Development Process and Usage - Provide detailed documentation covering the system architecture, design decisions, codebase, and user manual. Ensure that the documentation is clear and helpful for both developers and end-users.

h) Plan for Future Enhancements - Identify potential areas for future improvement and scalability. Outline a roadmap for additional features and enhancements that can be implemented post-deployment to continuously improve the application.

3. METHODOLOGY

The development of the Document Sharing Application using Django follows a systematic methodology to ensure that the final product meets the specified requirements and provides a robust, secure, and user-friendly platform. The methodology encompasses the definition of system requirements, adherence to design principles, a clear architectural blueprint, the selection of a suitable technology stack, and the establishment of an effective development environment.

3.1 Agile Software Development Life Cycle (SDLC) Model

The Agile Software Development Life Cycle (SDLC) model was adopted for the development of the Document Sharing Application. Agile is a methodology that emphasizes iterative progress through small, manageable phases known as sprints, focusing on collaboration, customer feedback, and rapid delivery of functional software. This section provides a detailed overview of how the Agile SDLC model was implemented in the project, highlighting the phases, practices, and benefits.

Overview of Agile SDLC

Agile SDLC is a framework that promotes adaptive planning, evolutionary development, early delivery, and continuous improvement. It encourages flexible responses to change and involves the following key principles:

Customer Collaboration: Active involvement of stakeholders to ensure the project meets their needs.

Iterative Development: Breaking down the project into small, incremental builds or iterations.

Continuous Feedback: Regular feedback from users and stakeholders to improve the product.

Adaptive Planning: Flexibility to adapt to changes in requirements and priorities.

Self-organizing Teams: Empowering teams to make decisions and manage their work.

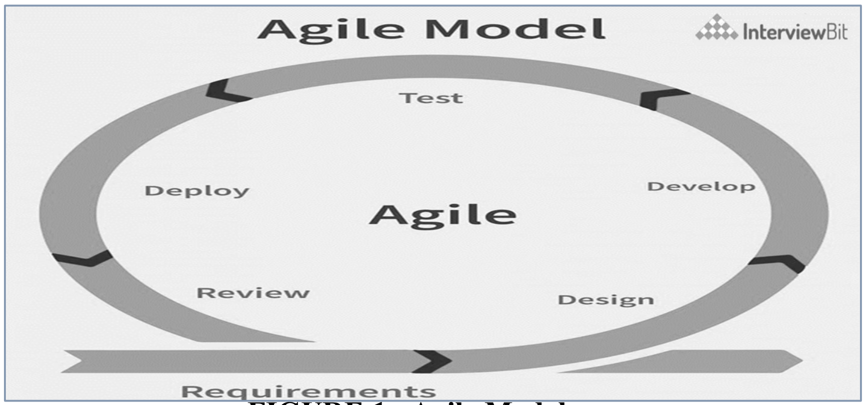


FIGURE 1 : Agile Model

❖ Phases of Agile SDLC

The Agile SDLC model consists of several iterative phases, each focusing on delivering a functional increment of the product. The key phases are:

I. **Concept/Inception:**

Goal: Define the project scope, objectives, and initial requirements.

Activities: Conduct initial meetings with stakeholders to gather high-level requirements and create a product backlog.

II. **Initiation/Planning:**

Goal: Plan the first sprint and establish the initial project plan.

Activities: Identify user stories, prioritize the product backlog, create a sprint backlog, and define the sprint goals.

III. **Iteration/Execution:**

Goal: Develop and deliver functional increments of the product in short cycles.

Activities:

Sprint Planning: Plan the tasks and goals for the upcoming sprint.

Daily Stand-ups: Short, daily meetings to discuss progress, impediments, and plans for the day.

Development: Code, test, and integrate new features.

Review: Demonstrate the completed work to stakeholders at the end of each sprint.

IV. **Release/Delivery:**

Goal: Deliver the product increment to users and stakeholders.

Activities: Deploy the new features to the production environment and gather feedback from users.

V. **Maintenance/Support:**

Goal: Ensure the product remains functional and meets user needs.

Activities: Fix bugs, address user feedback, and make minor enhancements.

VI. **Retrospective:**

Goal: Reflect on the sprint and identify areas for improvement.

Activities: Conduct retrospective meetings to discuss what went well, what didn't, and how processes can be improved.

➢ Implementation of Agile in the Document Sharing Application Using Django

The Agile SDLC model was applied to the development of the Document Sharing Application through multiple sprints, each focusing on specific features and enhancements. The following sections describe how each phase was executed.

- **Concept/Inception**

  During the inception phase, the project scope and objectives were defined through discussions with stakeholder. The initial product backlog was created, capturing high-level requirements such as:

  - User registration and login
  - Document upload, download, and management
  - User ratings and search bar
  - Access control and permissions

- **Initiation/Planning**

  In the planning phase, the initial sprint plan was established. User stories were identified and prioritized, forming the sprint backlog. The team set clear goals for the first sprint, which included:

- Setting up the development environment
- Implementing user registration and login
- Creating the basic document upload and download functionality

- **Iteration/Execution**

The iteration phase involved multiple sprints, each focusing on incremental development and delivery of features. Key activities included:

- Sprint 1:

  - Implemented user registration and login
  - Developed the initial document upload and download functionality
  - Conducted daily stand-ups and weekly sprint reviews

- Sprint 2:

  - Enhanced the document management features with download and delete functionality
  - Integrated user reviews ratings
  - Continued daily stand-ups and sprint reviews

- Sprint 3:

  - Added access control and permissions management
  - Improved the user interface with Bootstrap for styling
  - Conducted daily stand-ups and sprint reviews

Each sprint concluded with a review meeting where the completed features were demonstrated to stakeholders, and feedback was gathered for the next sprint.

- **Release/Delivery**

  At the end of each sprint, functional increments of the application will be deployed to a staging environment for user testing. After validation, the increments were deployed to the production environment.

- **Maintenance/Support**

  Post-deployment, the application will be monitored for bugs and issues. Regular maintenance activities included fixing bugs, addressing user feedback, and making minor enhancements to ensure the application remained functional and user-friendly.

- <u>**Benefits of Using Agile SDLC**</u>

i. **Flexibility and Adaptability:**
   - Agile allowed the team to adapt to changing requirements and priorities, ensuring the final product met user needs.

ii. **Early and Continuous Delivery:**
   - Regular delivery of functional increments ensured that stakeholders could see progress and provide feedback early and often.

iii. **Improved Collaboration:**
   - Daily stand-ups and regular review meetings facilitated communication and collaboration among team members and stakeholders.

iv. **Enhanced Quality:**
   - Continuous testing and integration ensured that issues were identified and addressed early, resulting in a higher quality product.

v. **Customer Satisfaction:**
   - Involving stakeholders throughout the development process ensured that the final product aligned with their expectations and requirements.

3.2 System Requirements :

The system requirements for the Document Sharing Application using Django are divided into functional and non-functional requirements. This ensures that the application meets the user needs, operates efficiently, and provides a secure and robust platform for document sharing.

- **Functional Requirements**

  Functional requirements specify what the system should do. These include features and functionalities that the end-users will interact with:

  I. **User Authentication and Authorization :**

  Registration: Users must be able to create an account with unique credentials (username and password).

  Login/Logout: Users should securely log in and log out of the system.

  Role-Based Access Control: Different roles (e.g., admin, regular user) should have different permissions and access levels.

  II. **Document Upload and Download:**

Upload: Users must be able to upload various document in PDF type.

Download: Users should be able to download documents they have access to.

III. **Rating System:**

Rate Documents: Users can rate documents to provide feedback.

View Ratings: Users can see average ratings and individual reviews.

IV. **Search Functionality:**

Search Bar: Users can search for documents using keywords, tags, or metadata.

Advanced Search: Users can filter search results based on criteria like document type, author, date, etc.

❖ **Non-Functional Requirements**

Non-functional requirements define the quality attributes of the system, focusing on how the system performs and operates:

I. **Security:**

Authentication: Implement secure authentication mechanisms.

Authorization: Ensure proper access control to prevent unauthorized access to documents and functionalities.

II. **Performance:**

Response Time: Ensure the application responds quickly to user actions, with minimal latency.

Load Handling: The system should handle traffic and file uploads/downloads efficiently.

III. **Usability:**

User Interface: Design an intuitive and user-friendly interface with clear navigation and accessible features.

Responsiveness: Ensure the application is responsive and works well on various devices (desktop, mobile, tablet).

IV. **Maintainability:**

Code Quality: Write clean, well-documented, and modular code to facilitate maintenance and updates.

Testing: Implement comprehensive unit testing to ensure the application's reliability.

V. **Reliability:**

Uptime: Ensure high availability of the application with minimal downtime.

Backup and Recovery: Implement regular data backup and disaster recovery mechanisms to prevent data loss.

VI. **Compatibility:**

Browser Compatibility: Ensure the application works well across different web browsers.

3.3 Design Principles

The design principles for the Document Sharing Application using Django ensure that the application is robust, maintainable, secure, and user-friendly. These principles guide the development process, ensuring that the final product meets the specified requirements and provides a high-quality user experience. Here is a detailed explanation of the key design principles:

A. Modularity

Definition: Modularity involves breaking down the application into smaller, manageable, and independent modules, each responsible for a specific functionality.

- **Application:**

  Separation of Concerns: Each module handles a distinct part of the application, such as user authentication, document management, version control, or rating system.
  Reusability: Modules can be reused across different parts of the application or in future projects, reducing redundancy.
  Maintainability: Smaller, self-contained modules are easier to understand, test, and maintain. Changes in one module are less likely to impact others.
  Scalability: Modules can be developed and scaled independently, allowing the application to handle increasing loads more efficiently.

- **Example:** Creating separate Django apps for user management, document handling, and rating system ensures that each functionality is encapsulated within its own module, promoting cleaner and more organized code.

B. Security

Definition: Security involves protecting the application and its data from unauthorized access, breaches, and other security threats.

- **Application:**

  Authentication: Implement secure user authentication mechanisms, such as password hashing.
  Authorization: Ensure proper access control by verifying user permissions before allowing access to resources.

- **Example:** Django's built-in authentication system, which uses secure password hashing algorithms, helps protect user credentials from being compromised.

C. Usability

Definition: Usability focuses on creating an intuitive and user-friendly interface that enhances the user experience.

- **Application:**

Intuitive Navigation: Design clear and straightforward navigation menus that make it easy for users to find and access features.

Responsive Design: Ensure the application is responsive and works well on various devices, including desktops, tablets, and smartphones.

- **Example:** Using Bootstrap for frontend development ensures that the application has a responsive design, adapting seamlessly to different screen sizes and devices.

D. **Maintainability**

Definition: Maintainability refers to the ease with which the application can be updated, debugged, and enhanced over time.

- **Application:**

  Code Quality: Write clean, well-documented, and modular code following coding standards and best practices.
  Version Control: Use version control systems (e.g., Git) to manage code changes, facilitate collaboration, and track revisions.
  Testing: Implement comprehensive testing, including unit testing, to ensure the application's reliability and facilitate debugging.

- **Example:** Adopting a consistent coding style and using tools like linters and code formatters helps maintain code quality and readability, making it easier for developers to understand and modify the codebase.

3.4 Architecture of the Application

The architecture of the Document Sharing Application using Django is designed to ensure a modular, scalable, and maintainable system that meets the functional and non-functional requirements. This section details the architectural components and their interactions, providing a comprehensive understanding of the application's structure.

➢ **Layered Architecture**

The application follows a layered architecture, which is a common design pattern in software engineering. This approach divides the application into distinct layers, each responsible for specific aspects of the functionality. The primary layers in this architecture include: 1. Presentation Layer, 2. Business Logic Layer, 3. Data Access Layer, 4. Database Layer.

1) **Presentation Layer:** The Presentation Layer is responsible for the user interface and user experience. It includes all the frontend components that interact directly with the users.

Components:

- **HTML/CSS:** Defines the structure and style of the web pages.
- **JavaScript:** Adds interactivity and dynamic behavior to the web pages.
- **Bootstrap:** Ensures responsive design, making the application usable on various devices.
- **Templates:** Django templates render dynamic content from the server to the client.

Responsibilities:

- Rendering web pages.
- Handling user inputs and interactions.
- Displaying data fetched from the backend.

Interaction:

- Communicates with the Business Logic Layer via HTTP requests and responses.
- Utilizes AJAX for asynchronous data updates without reloading the entire page.

2) **Business Logic Layer:** The Business Logic Layer contains the core functionality of the application. It processes user requests, applies business rules, and interacts with the Data Access Layer to retrieve or store data.

Components

- **Django Views:** Handle HTTP requests, process data, and return HTTP responses.
- **Forms:** Manage user inputs, validation, and form processing.

Responsibilities:

- Implementing the core logic of the application.
- Managing user sessions and authentication.
- Validating user inputs and enforcing business rules.
- Coordinating interactions between the Presentation and Data Access layers.

Interaction:

- Receives HTTP requests from the Presentation Layer, processes them, and returns HTTP responses.
- Interacts with the Data Access Layer to fetch or persist data.

3) **Data Access Layer:** The Data Access Layer abstracts the database interactions. It ensures that the business logic is decoupled from the data storage mechanisms

Components:

- **Django Models:** Represent the database schema as Python classes.
- **Query sets:** Provide an abstraction layer to interact with the database using Python code.

Responsibilities:

- Defining the data schema and relationships.
- Executing database queries.
- Ensuring data integrity and consistency.

Interaction:

- Interacts with the Business Logic Layer to provide data or persist changes.
- Directly communicates with the Database Layer to execute CRUD (Create, Read, Update, Delete) operations.

4) **Database Layer:** The Database Layer is responsible for data storage and retrieval. It includes the actual database system where the data is persisted.

Components:
Database Management System (DBMS): Used SQLite for development.

Responsibilities:
- Storing application data.
- Ensuring data integrity, security, and availability.
- Providing mechanisms for data backup and recovery.

Interaction:
- Executes SQL queries generated by the Data Access Layer.
- Provides data to the Data Access Layer or persists data changes.

➢ **Additional Components**

Static and Media Files:
Django's Static and Media Handling: Manages static files (CSS, JavaScript) and user-uploaded media files.

Security:
Django Middleware: Implements various security measures like CSRF protection and secure session management.

- **Interaction Flow:** The following steps outline the typical interaction flow within the application:
  - i. User Request: The user sends an HTTP request (e.g., uploading a document) via the web interface (Presentation Layer).
  - ii. Request Handling: Django's URL dispatcher routes the request to the appropriate view function (Business Logic Layer).
  - iii. Business Logic Processing: The view function processes the request, applying necessary business logic and validations.
    If data needs to be fetched or stored, the view interacts with the Data Access Layer.
  - iv. Data Access: The Data Access Layer (Django Models) constructs and executes database queries.
    Data is retrieved from or persisted to the database (Database Layer).
  - v. Response Generation: The view function prepares the response, rendering a template with dynamic data or returning a JSON response.
    The response is sent back to the user (Presentation Layer).
  - vi. User Interaction: The user interface updates dynamically based on the response, using JavaScript and AJAX for a seamless experience.

3.5 Technology Stack

The technology stack for the Document Sharing Application using Django is selected to ensure a robust, scalable, and user-friendly application. The stack includes Django for the backend, a combination of HTML, CSS, JavaScript, and Bootstrap for the frontend, and SQLite for the database. This section provides a detailed overview of each component and its role in the application.

3.5.1 Django

Overview: Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. It handles much of the complexity of web development, allowing developers to focus on writing the application without needing to reinvent the wheel.

Key Features:

- Model-View-Template (MVT) Architecture: Django follows the MVT pattern, which separates the data (Model), business logic (View), and presentation (Template) layers.
- ORM (Object-Relational Mapping): Django's ORM allows developers to interact with the database using Python code instead of writing SQL queries.
- Built-in Admin Interface: Django provides a powerful and customizable admin interface for managing application data.
- Security: Django includes built-in security features such as protection against SQL injection, cross-site scripting (XSS), and cross-site request forgery (CSRF).
- Scalability: Django is designed to handle high traffic and large volumes of data, making it suitable for applications that need to scale.

Model-View-Template (MVT) Architecture: The Model-View-Template (MVT) architecture is a design pattern used by Django to separate the application logic, data management, and user interface. This separation of concerns ensures a clean and maintainable codebase, making it easier to develop and scale complex web applications. Here's a detailed explanation of each component and how they interact within the Django framework.

A. Model

a) Definition: The Model is the part of the MVT architecture that handles all the data-related logic. It defines the structure of the database and represents the data stored in it. Each model class corresponds to a database table.

b) Key Features:

Database Schema: Models define the schema of the database, including tables and their fields..

Data Integrity: Models enforce data integrity rules, such as field constraints and relationships between tables.

ORM (Object-Relational Mapping): Django's ORM allows developers to interact with the database using Python code instead of writing SQL queries.

c) Responsibilities:

Defining Data Structures: Models define the structure of the data and the types of data that can be stored.

Data Validation: Models ensure that data conforms to the defined constraints and validation rules before it is saved to the database.

Querying the Database: Models provide an API for querying the database, retrieving, updating, and deleting records.

d) Example:

Python from django.db import models

```
class Document(models.Model):
  title = models.CharField(max_length=200)
  content = models.TextField()
  uploaded_at = models.DateTimeField(auto_now_add=True)

  def __str__(self):
    return self.title
```

In this example, the Document model defines a database table with three fields: title, content, and uploaded_at.

B. View

a) Definition: The View is responsible for processing user requests, executing business logic, and returning the appropriate responses. Views interact with models to retrieve and manipulate data and use templates to render the user interface.

b) Key Features:

Request Handling: Views handle HTTP requests and generate HTTP responses.
Business Logic: Views contain the business logic of the application, processing user inputs and interactions.
Intermediary: Views act as intermediaries between models and templates, fetching data from the database and passing it to the templates for rendering.

c) Responsibilities:

Processing Requests: Views process incoming HTTP requests and determine the appropriate response.
Interacting with Models: Views interact with models to retrieve, update, or delete data based on the request.
Rendering Templates: Views use templates to render the user interface, passing the necessary data to the templates.

d) Example:

Python from django.shortcuts import render from .models import Document

```
def document_list(request):
  documents = Document.objects.all()
  return render(request, 'document_list.html', {'documents': documents})
```

In this example, the document_list view retrieves all documents from the database and renders them using the document_list.html template.

C. Template

a) **Definition:** The Template is responsible for rendering the user interface. Templates define the structure and layout of the web pages and use data passed from the views to generate dynamic content.

b) **Key Features:**

HTML Structure: Templates define the HTML structure of the web pages.

Dynamic Content: Templates use a templating language to insert dynamic content into the HTML.

Reusability: Templates can be reused across different views, promoting consistency and reducing redundancy.

c) **Responsibilities:**

Rendering HTML: Templates render HTML pages by combining static HTML with dynamic data.

Separation of Presentation: Templates separate the presentation layer from the business logic, ensuring a clean separation of concerns.

Reusable Components: Templates can include reusable components, such as headers, footers, and navigation bars, to ensure consistency across the application.

d) **Example:**

```
html
<!DOCTYPE html>
<html>
<head>
  <title>Document List</title>
</head>
<body>
  <h1>Documents</h1>
  <ul>
    {% for document in documents %}
      <li>{{ document.title }}</li>
    {% endfor %}
  </ul>
</body>
</html>
```

In this example, the document_list.html template uses the templating language to iterate over the list of documents and display their titles.

D. Interaction Between Components

The components of the MVT architecture interact in a well-defined manner to handle user requests and generate responses:

i. User Request: The user sends an HTTP request (e.g., to view a list of documents).
ii. URL Routing: Django's URL dispatcher routes the request to the appropriate view function based on the URL pattern.
iii. View Processing: The view function processes the request, retrieves data from the model, and prepares the context for the template.

iv. Model Interaction: The view interacts with the model to fetch or manipulate data as needed.
v. Template Rendering: The view passes the context data to the template, which renders the HTML page with dynamic content.
vi. Response to User: The rendered HTML page is returned as an HTTP response to the user.

E. Role in the Application:

- ⦿ Backend Development: Django handles all the server-side logic, including user authentication, document management, version control, and the rating system.
- ⦿ Database Interaction: Django's ORM manages interactions with the SQLite database, ensuring data integrity and consistency.
- ⦿ URL Routing: Django's URL dispatcher routes incoming requests to the appropriate view functions.

3.5.2 Frontend Technologies (HTML, CSS, JavaScript, and Bootstrap)

The frontend of the Document Sharing Application relies on a combination of HTML, CSS, JavaScript, and Bootstrap. Each of these technologies plays a crucial role in creating a responsive, interactive, and visually appealing user interface. This section provides a detailed overview of these technologies and their specific roles in the application.

A. HTML (HyperText Markup Language)

a) **Overview:** HTML is the standard markup language used to create the structure and content of web pages. It defines the elements and layout of the web page, including text, images, links, forms, and multimedia.

b) **Key Features:**

Elements and Tags: HTML consists of various elements, each represented by tags. Common tags include <h1> to <h6> for headings, <p> for paragraphs, <a> for hyperlinks, <img> for images, and <form> for forms.

Attributes: HTML elements can have attributes that provide additional information or specify properties. For example, the src attribute in an <img> tag specifies the image source.

Document Structure: HTML documents follow a hierarchical structure with a root element (<html>), a head section (<head>) containing metadata, and a body section (<body>) containing the actual content.

c) **Role in the Application:**

Page Structure: HTML provides the basic structure of the web pages, organizing content into headings, paragraphs, lists, tables, and forms.

Content Presentation: HTML is used to display text, images, links, and other multimedia content to the user.

Forms and User Input: HTML forms collect user input, such as document uploads and ratings, which are then processed by the server.

d) Example:

```
html
<!DOCTYPE html>
<html>
<head>
  <title>Document Sharing Application</title>
</head>
<body>
  <h1>Welcome to the Document Sharing Application</h1>
  <form action="/upload" method="post" enctype="multipart/form-data">
    <label for="document">Upload Document:</label>
    <input type="file" id="document" name="document">
    <input type="submit" value="Upload">
  </form>
</body>
</html>
```

In this example, the HTML document includes a form for uploading documents.

B. CSS (Cascading Style Sheets)

a) Overview: CSS is a style sheet language used to describe the presentation and layout of HTML documents. It allows developers to control the appearance of web pages, including colours, fonts, spacing, and positioning.

b) Key Features:

Selectors: CSS uses selectors to target HTML elements and apply styles. Common selectors include element selectors (e.g., p), class selectors (e.g., .class), and ID selectors (e.g., #id).

Properties and Values: CSS consists of properties and values that define the styles. For example, the colour property sets the text colour, and the margin property defines the space around elements.

Box Model: The CSS box model describes the layout of elements, including margins, borders, padding, and content.

Responsive Design: CSS includes features such as media queries and flexible grid layouts to create responsive web designs that adapt to different screen sizes.

c) Role in the Application:

Styling: CSS is used to style HTML elements, ensuring a consistent and visually appealing design.

Layout: CSS controls the layout of web pages, arranging elements in a coherent and user-friendly manner.

Responsive Design: CSS ensures that the application is accessible and functional on various devices, including desktops, tablets, and smartphones.

d) Example:

```
css
body {
    font-family: Arial, sans-serif;
    background-color: #f4f4f4;
    color: #333;
    margin: 0;
    padding: 0;
}

h1 {
    color: #5a67d8;
}

form {
    background: #fff;
    padding: 20px;
    border-radius: 5px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

In this example, CSS styles the body, heading, and form elements.

C. <u>JavaScript</u>

a) Overview: JavaScript is a scripting language used to create interactive and dynamic web content. It enables developers to add behaviour to web pages, making them more responsive and engaging for users.

b) Key Features:

DOM Manipulation: JavaScript can interact with and manipulate the Document Object Model (DOM), allowing dynamic changes to the content and structure of web pages.

Event Handling: JavaScript can respond to user actions such as clicks, mouse movements, and form submissions, enhancing interactivity.

Asynchronous Programming: JavaScript supports asynchronous operations using callbacks, promises, and async/await, enabling tasks like data fetching without blocking the main thread.

APIs and Libraries: JavaScript provides access to various APIs (e.g., Fetch API, Web Storage API) and libraries (e.g., jQuery, React) to extend functionality.

c) Role in the Application:

Interactivity: JavaScript adds interactivity to the application, such as real-time updates, form validation, and AJAX requests.

Dynamic Content: JavaScript allows dynamic updates to the web page content without requiring a full page reload.

Enhanced User Experience: JavaScript improves the user experience by providing immediate feedback and smooth interactions.

d) Example:

```
javascript
document.getElementById('document').addEventListener('change',   function()
{
  const fileName = this.files[0].name;
  alert('Selected file: ' + fileName);
});
```

In this example, JavaScript adds an event listener to the file input element to display the selected file name.

D. <u>Bootstrap</u>

a) Overview: Bootstrap is a popular CSS framework that simplifies the development of responsive and mobile-first web pages. It provides a collection of pre-designed components and utility classes that streamline the design process.

b) Key Features:

Grid System: Bootstrap’s grid system allows developers to create responsive layouts using a series of rows and columns.

Pre-designed Components: Bootstrap includes a wide range of pre-designed components such as buttons, forms, modals, navbars, and alerts.

Utility Classes: Bootstrap provides utility classes for common CSS tasks such as spacing, alignment, and typography, reducing the need for custom CSS.

Customization: Bootstrap can be customized using Sass variables and mixins to match the design requirements of the application.

c) Role in the Application:

Responsive Design: Bootstrap ensures that the application is responsive and adapts to different screen sizes and devices.

Consistent Styling: Bootstrap’s pre-designed components provide a consistent look and feel across the application.

Rapid Development: Bootstrap accelerates the development process by providing ready-to-use components and utilities.

d) Example:

```
html
<!DOCTYPE html>
<html>
<head>
   <link
rel="stylesheet"href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/b
ootstrap.min.css">
</head>
<body>
   <div class="container">
      <h1 class="my-4">Document Sharing Application</h1>
      <form class="mb-4">
         <div class="form-group">
            <label for="document">Upload Document:</label>
            <input type="file" class="form-control" id="document"
name="document">
         </div>
         <button type="submit" class="btn btn-primary">Upload</button>
      </form>
   </div>
   <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
   <script
src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.5.4/dist/umd/popper.min.
js"></script>
   <script
src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"><
/script>
</body>
</html>
```

In this example, Bootstrap is used to style the form and button elements, providing a clean and responsive design.

Integration in the Application:

- **User Interface:** HTML and CSS define the structure and style of the user interface, ensuring a clean and consistent look.
- **Interactivity:** JavaScript handles dynamic interactions, such as form submissions, real-time updates, and AJAX requests for asynchronous data fetching.
- **Responsive Design:** Bootstrap ensures that the application is accessible and functional on various devices, including desktops, tablets, and smartphones.

3.5.3 <u>Database (SQLite)</u>

SQLite is a widely used relational database management system (RDBMS) known for its lightweight, serverless, and self-contained architecture. It is the default database for many applications, including Django, due to its simplicity and ease of use. This section provides a detailed overview of SQLite, its features, advantages, and how it is used in the Document Sharing Application.

a) **Definition:** SQLite is a C-language library that provides a relational database management system. Unlike traditional RDBMSs, SQLite is not a standalone process with which the application program communicates. Instead, it is a part of the application, linked directly into it.

<u>Characteristics:</u>

Serverless: SQLite does not require a separate server process or operating system to operate. The database engine is embedded directly into the application.

Self-contained: SQLite is a self-contained database engine that does not depend on external libraries or tools.

Zero-configuration: SQLite requires no configuration, making it easy to set up and use.

Transactional: SQLite supports ACID (Atomicity, Consistency, Isolation, Durability) transactions, ensuring data integrity and reliability.

b) **Features:**

File-based Storage: SQLite stores the entire database in a single cross-platform disk file. This file can be easily copied, moved, and backed up.

Data Types: SQLite uses dynamic typing for data types, meaning that the data type of a value is associated with the value itself, not with its container.
It supports various data types, including INTEGER, REAL, TEXT, BLOB, and NULL.

SQL Support: SQLite supports most of the SQL-92 standard, including complex queries, joins, subqueries, and transactions.
It provides comprehensive support for SQL features like triggers, views, and indexes.

Lightweight: The entire SQLite library is compact, typically less than 1 MB in size, making it ideal for applications where resources are limited.

Platform-independent: SQLite databases are cross-platform, allowing them to be used on various operating systems without modification.

c) **Advantages**

<u>Ease of Use:</u> SQLite is easy to set up and use, with no need for server installation or configuration. This makes it ideal for development, testing, and small-scale production applications.

Performance: SQLite is highly efficient and can handle moderate-sized databases with excellent performance. It is optimized for fast read and write operations.

Portability: Since SQLite databases are self-contained files, they are highly portable. This makes it easy to move the database between different environments or distribute it with applications.

Reliability: SQLite is designed to be reliable and robust. It guarantees data integrity through its support for ACID transactions and has extensive testing to ensure stability.

Concurrency: While SQLite supports multiple connections, it is best suited for scenarios with a lower degree of write concurrency. It can handle multiple readers but has limited support for concurrent writes.

d) Usage in the Document Sharing Application

In the Document Sharing Application, SQLite is used as the backend database to store and manage various types of data, including user information, documents, reviews, and ratings. Here is how SQLite is utilized in different aspects of the application:

- **Database Models:**

  User Model: Stores user information, such as username, email, and password.
  Document Model: Stores details about uploaded documents, including the file path, upload date, and associated metadata.
  Review Model: Stores user reviews and ratings for documents.

e) Example Models in Django:

```
Python from django.db import models

class User(models.Model):
    username = models.CharField(max_length=150, unique=True)
    email = models.EmailField(unique=True)
    password = models.CharField(max_length=128)

class Document(models.Model):
    title = models.CharField(max_length=200)
    content = models.TextField()
    uploaded_at = models.DateTimeField(auto_now_add=True)
    user = models.ForeignKey(User, on_delete=models.CASCADE)

class Review(models.Model):
    document = models.ForeignKey(Document, on_delete=models.CASCADE)
    user = models.ForeignKey(User, on_delete=models.CASCADE)
    rating = models.IntegerField()
    comment = models.TextField()
    created_at = models.DateTimeField(auto_now_add=True)
```

f) Database Operations:

- CRUD Operations: SQLite handles Create, Read, Update, and Delete operations for managing users, documents, and reviews.

- Queries: The application uses SQLite to perform complex queries, such as retrieving all documents uploaded by a user, fetching the highest-rated documents, and filtering documents based on keywords.

- Transactions: SQLite ensures that all database operations are transactional, maintaining data integrity and consistency.

g) Configuration in Django:

By default, Django uses SQLite as the database engine. The database configuration is specified in the settings.py file:

```
python

DATABASES = {

  'default': {

    'ENGINE': 'django.db.backends.sqlite3',

    'NAME': BASE_DIR / 'db.sqlite3',

  }

}
```

This configuration specifies that Django should use SQLite and sets the database file location.

3.6 Development Environment: Visual Studio Code (VS Code)

Visual Studio Code (VS Code) is a powerful, open-source code editor developed by Microsoft. It is widely used by developers due to its rich feature set, extensibility, and cross-platform support. This section provides a detailed overview of VS Code, its features, advantages, and how it is used in the development of the Document Sharing Application.

Overview

a) **Definition:** VS Code is a lightweight but powerful source code editor that runs on Windows, macOS, and Linux. It supports a wide range of programming languages and frameworks, including Python and Django, which are used in the Document Sharing Application.

b) **Characteristics:**

- Open-source: VS Code is freely available under the MIT License, allowing developers to use, modify, and distribute it.

- Extensible: It has a vast ecosystem of extensions that enhance its functionality, providing support for additional languages, debuggers, and tools.

- Integrated Terminal: VS Code includes a built-in terminal that allows developers to run command-line tools and scripts directly within the editor.

c) **Features**

Code Editing:

- Syntax Highlighting and IntelliSense: VS Code provides syntax highlighting and IntelliSense, offering code completions, parameter info, and quick documentation for various languages.

- Code Navigation: Features like Go to Definition, Find All References, and Peek Definition help developers navigate through their codebase efficiently.

- Code Snippets: VS Code supports code snippets, allowing developers to insert commonly used code patterns quickly.

d) **Debugging:**

- Integrated Debugger: VS Code includes a powerful debugger that supports breakpoints, call stacks, and variable inspection for a wide range of languages.

- Debug Console: The debug console allows developers to evaluate expressions and execute commands during a debugging session.

e) **Extensions:**

- Marketplace: The VS Code Marketplace hosts thousands of extensions that add functionality such as language support, themes, linters, and debuggers.

- Popular Extensions: For Django development, popular extensions include Python by Microsoft, Django by Baptiste Darthenay, and SQLite by Alexey Marchenko.

f) **Version Control:**

- Git Integration: VS Code has built-in Git integration, providing features such as source control, branching, merging, and conflict resolution.

- Version Control Pane: The Source Control pane allows developers to stage, commit, and push changes, view diffs, and manage repositories.

g) **Customization:**

- Settings: VS Code can be customized through user and workspace settings, allowing developers to tailor the editor to their preferences.

- Keybindings: Developers can customize keybindings to create personalized shortcuts for common actions.

h) **Collaboration:**

- Live Share: The Live Share extension enables real-time collaboration, allowing developers to share their workspace and collaborate on code with others.

➢ Usage in the Document Sharing Application

In the development of the Document Sharing Application, VS Code is used as the primary Integrated Development Environment (IDE). Here is how VS Code is utilized in different aspects of the development process:

Setting Up the Development Environment:

- Installing VS Code: Developers start by downloading and installing VS Code from the official website.
- Extensions: Essential extensions for Django development, such as Python and Django, are installed from the VS Code Marketplace.
- Workspace Configuration: Developers configure the workspace by creating a folder for the project and opening it in VS Code.

Writing Code:

- Python and Django Support: The Python extension provides syntax highlighting, IntelliSense, and debugging support for Python and Django code.
- Code Navigation: Features like Go to Definition and Find All References help developers navigate the codebase efficiently.
- Integrated Terminal: The integrated terminal is used to run Django management commands, such as python manage.py runserver, python manage.py makemigrations, and python manage.py migrate.

Debugging:

- Debugging Django Applications: Developers use the integrated debugger to set breakpoints, inspect variables, and step through code, helping to identify and fix issues quickly.

Version Control:

- Git Integration: The built-in Git support is used to manage source control, allowing developers to commit changes, create branches, and push code to remote repositories.

Collaboration:

- Live Share: For collaborative development, the Live Share extension allows team members to work together in real-time, sharing their workspace and debugging sessions

4. SYSTEM DESIGN AND IMPLEMENTATION

4.1 System Architecture

The architecture of the Document Sharing Application is designed to ensure modularity, scalability, and ease of maintenance. The application follows the Model-View-Template (MVT) architecture pattern, which is a variant of the MVC (Model-View-Controller) pattern tailored for Django. This pattern separates the concerns of data (Model), user interface (View), and business logic (Template).

❖ Components of the System Architecture

a. Client Side: The client side includes web browsers used by end-users to interact with the application. The user interface is developed using HTML, CSS, JavaScript, and Bootstrap.

b. Server Side: The server side is built using Django, a high-level Python web framework that encourages rapid development and clean, pragmatic design. It handles the business logic, data processing, and communication between the client side and the database.

c. Database: The database is managed using SQLite, a lightweight, disk-based database that doesn't require a separate server process. It stores all the data related to documents, users, permissions, and ratings.

d. Middleware: Middleware components in Django process requests before they reach the view and responses before they are sent to the client. They handle tasks such as authentication, session management, and security.

4.2 ER Diagram

The Entity-Relationship (ER) diagram represents the logical structure of the database. It illustrates the relationships between different entities in the system. The ER data model specifies enterprise schema that represents the overall logical structure of a database graphically. The Entity Relationship Diagram explains the relationship among the entities present in the database. ER models are used to model real-world objects like a person, a car, or a company and the relation between these real-world objects. In short, the ER Diagram is the structural format of the database.

⊙ Symbols Used in ER Model

ER Model is used to model the logical view of the system from a data perspective which consists of these symbols:

Rectangles: Rectangles represent Entities in the ER Model.
Ellipses: Ellipses represent Attributes in the ER Model.
Diamond: Diamonds represent Relationships among Entities.
Lines: Lines represent attributes to entities and entity sets with other relationship types.
Double Ellipse: Double Ellipses represent Multi-Valued Attributes.
Double Rectangle: Double Rectangle represents a Weak Entity.

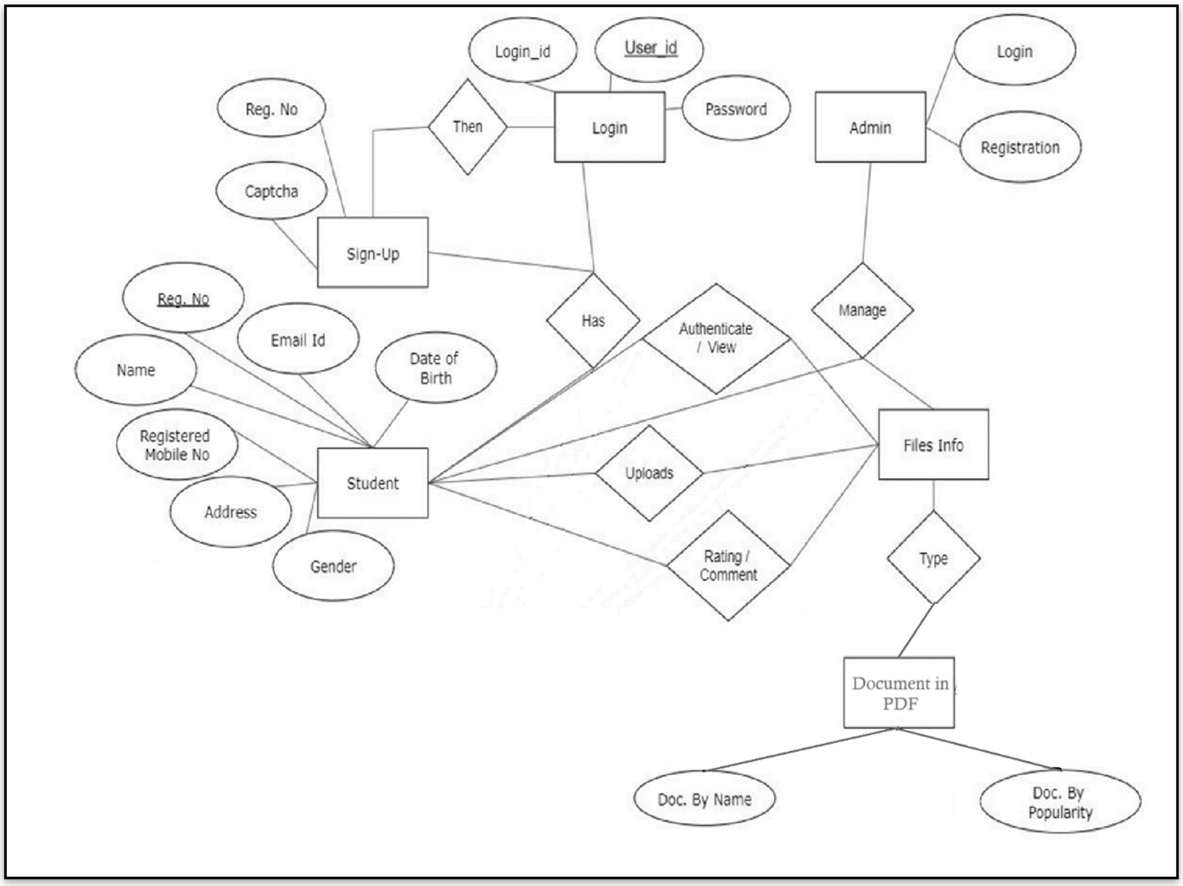


FIGURE 2: ER Diagram

Entities and Attributes

1. Login:
    - Attributes: Login_id, User_id, Password
    - Description: This entity represents the login details for users. It includes the login ID, user ID, and password required for authentication.
2. Student:
    - Attributes: Reg_No, Name, Registered_Mobile_No, Email_Id, Address, Gender, Date_of_Birth
    - Description: This entity contains the personal and registration details of the students who will be using the application.
3. Admin:
    - Attributes: Login, Registration
    - Description: This entity represents the administrative functionalities, including user login and registration management.
4. Files Info:
    - Attributes: Type
    - Description: This entity holds information about the files being uploaded and managed within the application.

5. Document in PDF:
    - Attributes: Doc_By_Name, Doc_By_Popularity
    - Description: This entity represents the documents stored in PDF format, categorized by name and popularity.

Relationships:

1. Login:
    - Relationships: Connects with the Student entity through the Login_id and User_id attributes. It indicates that each student has login details to access the application.
2. Sign-Up:
    - Relationships: Connects with the Student entity through the Reg_No and Email_Id attributes. It indicates that students need to sign up using their registration number and email ID.
3. Student:
    - Relationships:
        - Has: Connects with the Email_Id, Date_of_Birth attributes indicating personal information associated with each student.
        - Authenticate/View: This relationship connects students to the Files Info entity, signifying that students can authenticate and view files.
        - Uploads: Connects the Student entity to the Files Info entity, indicating that students can upload files.
        - Rating/Comment: Connects to the Files Info entity, allowing students to rate and comment on the files.
4. Admin:
    - Relationships:
        - Manage: Connects to the Files Info entity, indicating that the admin manages the file information.
5. Files Info:
    - Relationships:
        - Type: Connects to the Document in PDF entity, categorizing documents by type.
        - Document in PDF: Further categorized into Doc_By_Name and Doc_By_Popularity, providing different views of the stored documents.

The above Figure 2 represents an ER Diagram of file-sharing application where students can register, log in, upload files, and rate or comment on the files. Admins manage the application and files. The files are stored in PDF format and categorized by name and popularity for easier access and management. This structure ensures secure document handling and user-friendly access within the Django-based application.

4.3 UML Diagrams

UML, which stands for Unified Modeling Language, is a way to visually represent the architecture, design, and implementation of complex software systems. When you're writing code, there are thousands of lines in an application, and it's difficult to keep track of the relationships and hierarchies within a software system. UML diagrams divide that software system into components and subcomponents.

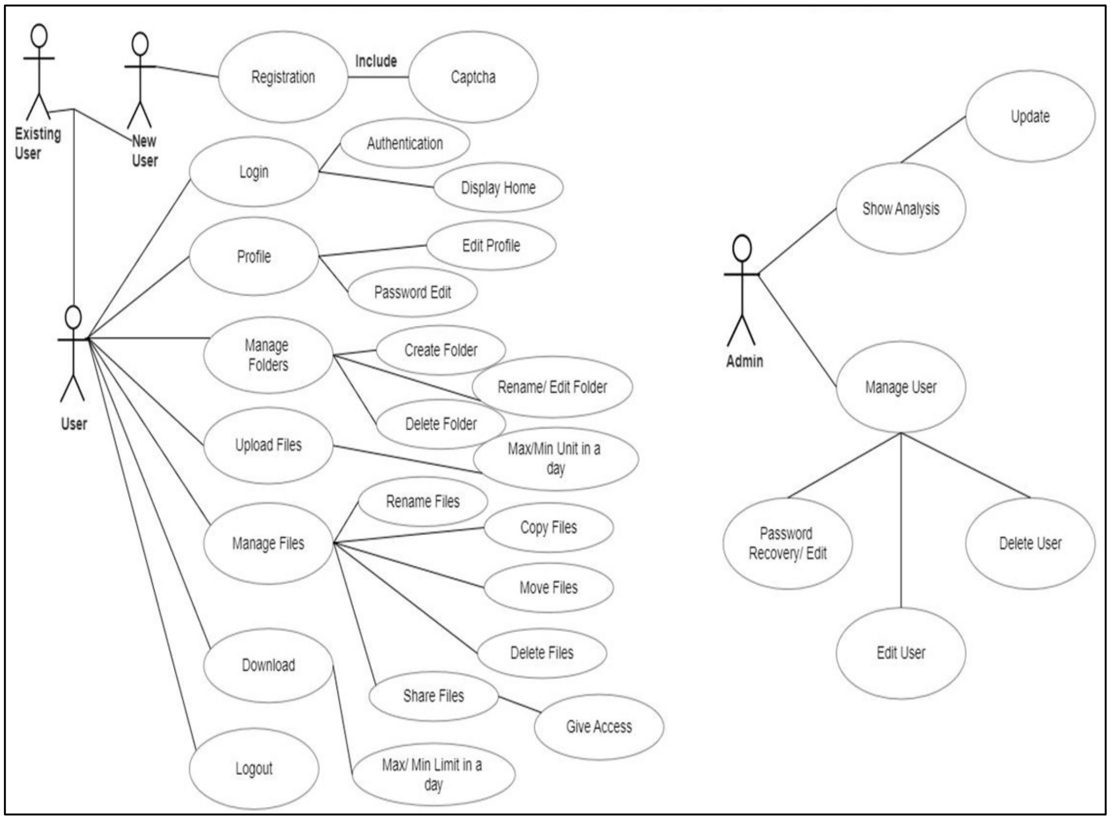


FIGURE 3 : UML / USE Case Diagram

4.4 Use Case Diagrams

The UML use case diagram illustrates the functionalities of a file sharing application developed using Django. It highlights the interactions between various types of users and the system's use cases. The actors in the diagram are Existing User, New User, User, and Admin.

Actors:

1. **Existing User**: A user who has previously registered and can log into the system.
2. **New User**: A user who is new to the system and needs to register.
3. **User**: A general user who has various permissions within the system.
4. **Admin**: An administrative user who has additional privileges to manage the system and users.

Use Cases and Relationships:

1. **Registration**:
    - **Actors**: New User
    - **Description**: This use case allows a new user to register for the application. It includes a registration form and a Captcha for security purposes.
2. **Login**:
    - **Actors**: Existing User, New User
    - **Description**: This use case allows both existing and new users to log into the application after registering.
    - **Includes**: Authentication to verify user credentials.
3. **Profile**:
    - **Actors**: User
    - **Description**: This use case allows users to view and edit their profile information.

    - **Includes**: Display Home to show the user's dashboard, Edit Profile, and Password Edit for changing the password.
4. **Manage Folders**:
    - **Actors**: User
    - **Description**: This use case involves creating, renaming, editing, and deleting folders within the application. It includes setting maximum and minimum limits of units in a day.
5. **Upload Files**:
    - **Actors**: User
    - **Description**: This use case allows users to upload files to the system.
6. **Manage Files**:
    - **Actors**: User
    - **Description**: This use case includes various file management operations such as renaming, copying, moving, and deleting files. It also allows users to share files and give access to others.
7. **Download**:
    - **Actors**: User
    - **Description**: This use case allows users to download files from the system.
8. **Logout**:
    - **Actors**: User
    - **Description**: This use case allows users to log out of the application.
9. **Admin Use Cases**:
    - **Manage User**:
        - **Actors**: Admin
        - **Description**: This use case allows the admin to manage users, including password recovery/edit, deleting users, and editing user information.
    - **Show Analysis**:
        - **Actors**: Admin
        - **Description**: This use case allows the admin to view and update the analysis of the system's usage and performance.

4.5 Data Flow Diagram (DFD)

DFD is the abbreviation for Data Flow Diagram. The flow of data in a system or process is represented by a Data Flow Diagram (DFD). It also gives insight into the inputs and outputs of each entity and the process itself. Data Flow Diagram (DFD) does not have a control flow and no loops or decision rules are present. Specific operations, depending on the type of data, can be explained by a flowchart. It is a graphical tool, useful for communicating with users, managers and other personnel. it is useful for analysing existing as well as proposed systems.

It provides an overview of
-What data is system processes.
-What transformation are performed.
-What data are stored.
-What results are produced, etc

Levels of Data Flow Diagram (DFD)

Data Flow Diagram (DFD) uses hierarchy to maintain transparency thus multilevel Data Flow Diagram (DFD's) can be created. Levels of Data Flow Diagram (DFD) are as follows:

- **0-level DFD**

  Components

1. **Student Info Management**:
   - This module handles the information related to students. It interacts with the online file-sharing system to provide and retrieve student information as needed.
2. **Department Management**:
   - This module manages departmental information. It interacts with the system to ensure that department-specific data is accessible and updated within the file-sharing application.
3. **System User Management**:
   - This module is responsible for managing user information within the system. It includes functionalities such as adding, removing, and updating user details. It ensures that user data is synchronized with the central file-sharing system.
4. **Login Management**:
   - This module handles the authentication and login processes. It verifies user credentials and manages user sessions, ensuring secure access to the file-sharing system.
5. **File Info Management**:
   - This module is responsible for managing file-related information. It includes functionalities such as uploading, downloading, sharing, and organizing files. It ensures that file data is correctly managed and accessible within the system.

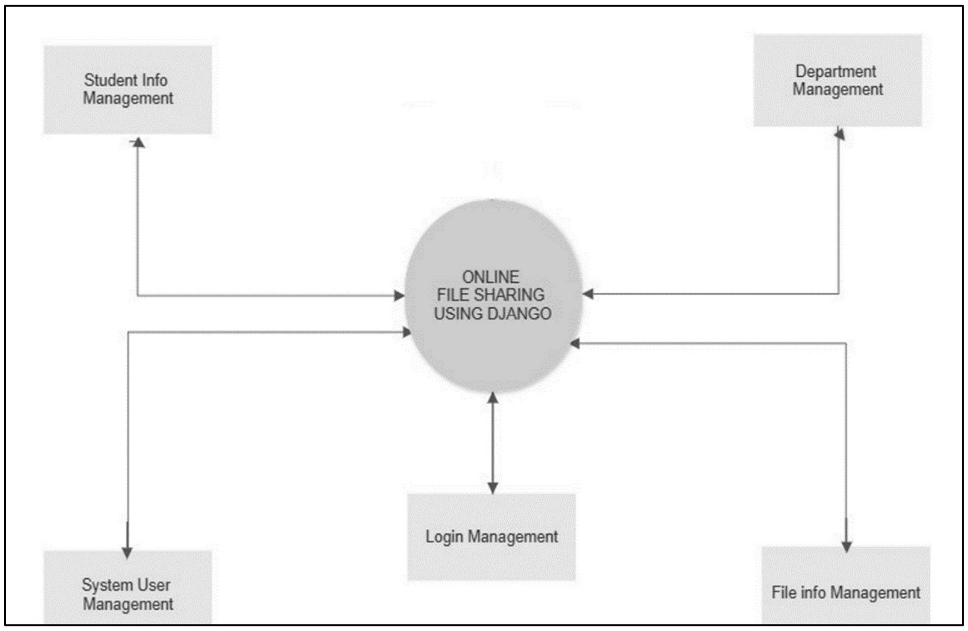


FIGURE 4: Level Zero DFD

- **1-Level DFD** **:**This level provides a more detailed view of the system by breaking down the major processes identified in the level 0 DFD into sub-processes. Each sub-process is depicted as a separate process on the level 1 DFD. The data flows and data stores associated with each sub-process are also shown. In Figure 5 1-level DFD, the context diagram is decomposed into multiple bubbles/processes. In this level, we highlight the main functions of the system and breakdown the high-level process of 0-level DFD into subprocesses.

Central System: Online File Sharing Using Django: This is the core system that integrates various management functionalities and reporting features.

Management Modules

1. Student Info Management:
   - Manages student information within the system.
   - Generates student information reports.
2. Department Management:
   - Handles information related to different departments.
   - Generates department reports.
3. File Info Management:
   - Manages file information and their metadata.
   - Generates file information reports.
4. Login Management:
   - Oversees login activities and user authentication.
   - Generates login details reports.
5. System User Management:
   - Manages system users, their roles, and permissions.
   - Generates system user details reports.

Flow: Each management module on the left side of the diagram inputs data into the central online file-sharing system. The system processes this data and generates specific reports, indicated by the arrows pointing to the report generation modules on the right side.

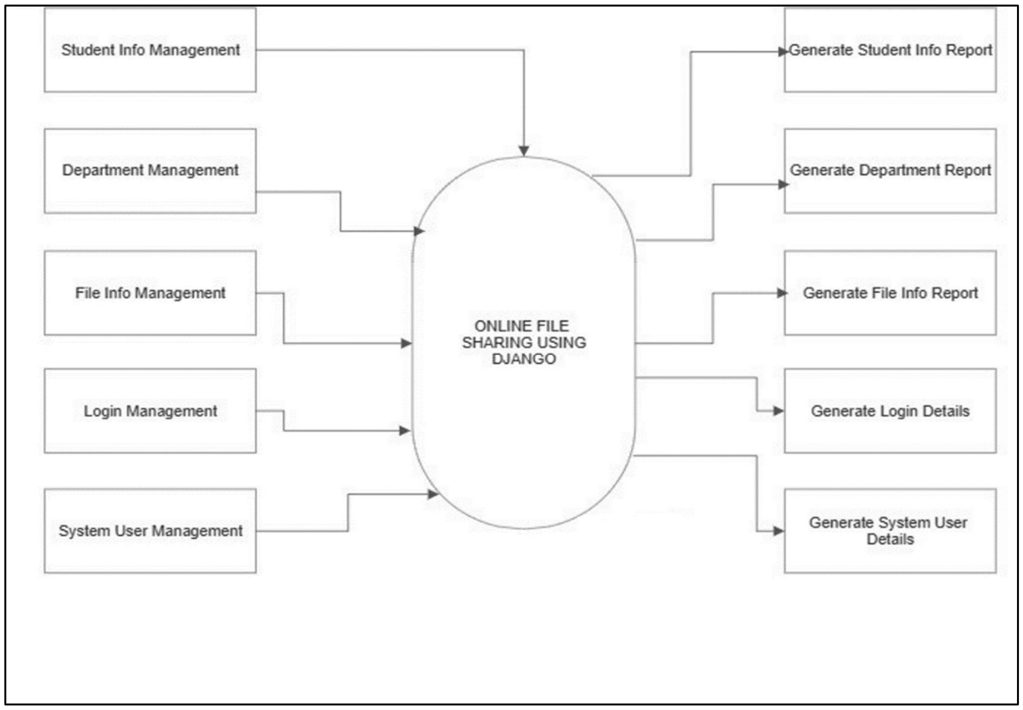

Student Info Management
Department Management
File Info Management
Login Management
System User Management
ONLINE FILE SHARING USING DJANGO
Generate Student Info Report
Generate Department Report
Generate File Info Report
Generate Login Details
Generate System User Details

FIGURE 5: Level One DFD

- **2-level DFD:** This level provides an even more detailed view of the system by breaking down the sub-processes identified in the level 1 DFD into further sub-processes. Each sub-process is depicted as a separate process on the level 2 DFD. The data flows and data stores associated with each sub-process are also shown in Figure 6.

Admin Interaction

1. Admin: The starting point for the process.
2. Forgot Password:
    - If the admin forgets their password, they can initiate the "Forgot Password" process.
    - This triggers an email to be sent to the user to reset their password.

Login Process

3. Login to System:
    - The admin logs into the system.
4. Check Credentials:
    - The system verifies the admin's login credentials.
    - If credentials are correct, the admin is granted access; otherwise, they are denied access.

Role and Module Management

5. Check Role of Access:
    - Once logged in, the system checks the admin's role to determine their access level.
6. Manage Modules:
    - Based on the role, the admin can manage different modules within the system.

Module Management

7. Manage User Permission:
    - Admin can manage permissions for different users.
8. Manage Department Details:
    - Admin can manage details related to various departments.
9. Manage Role of User:
    - Admin can manage and assign roles to different users.
10. Manage Report:
    - Admin can generate and manage various reports.
11. Manage File Details:
    - Admin can manage file information and metadata.
12. Manage Student Details:
    - Admin can manage information related to students.

Flow

- The process starts with the admin attempting to log in or resetting the password if needed.
- After successful login and credential verification, the system checks the admin's role to determine accessible modules.
- The admin can then manage different modules, including user permissions, department details, user roles, reports, file details, and student details.

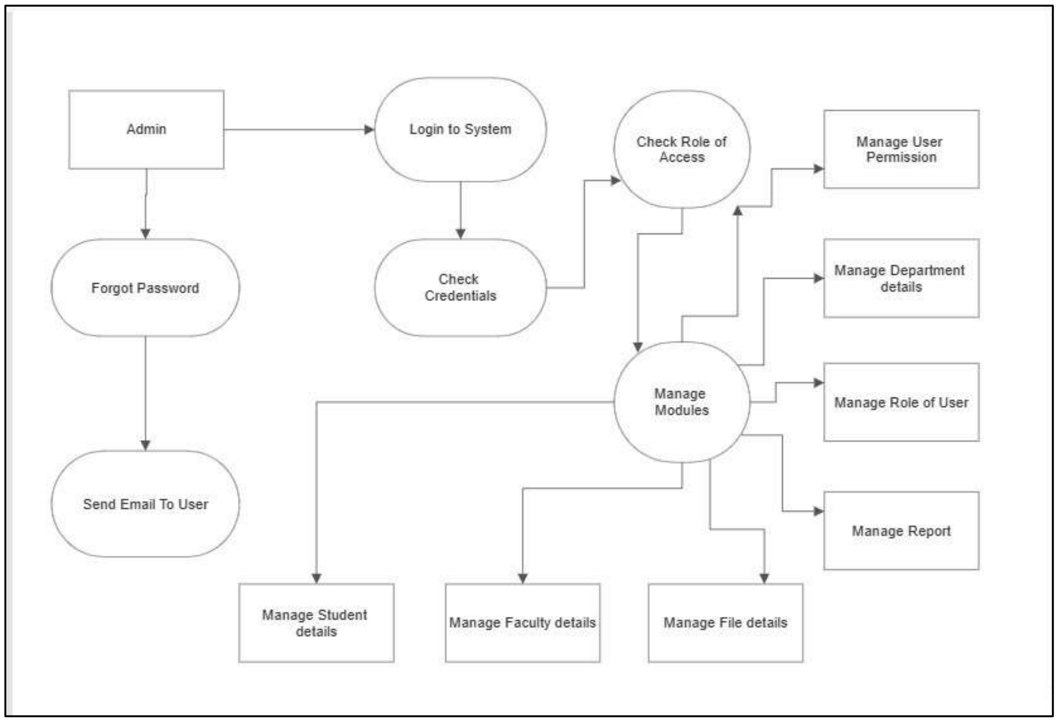


FIGURE 6: Level Two DFD

4.6 Database Design

The database design involves defining the structure of the database tables, their relationships, and constraints.

➢ **Tables and Relationships**

a) User Table:

Columns: userID (PK), username, password, email, role.
Relationships: One-to-Many with Document, One-to-Many with Rating.

b) Document Table:

Columns: documentID (PK), title, description, filePath, uploadDate, userID (FK).
Relationships: Many-to-One with User, One-to-Many with Rating, One-to-Many with Permission.

c) Rating Table:

Columns: ratingID (PK), documentID (FK), userID (FK), ratingValue, review.
Relationships: Many-to-One with Document, Many-to-One with User.

d) Permission Table:

Columns: permissionID (PK), documentID (FK), userID (FK), accessLevel
Relationships: Many-to-One with Document, Many-to-One with User

4.7 User Interface Design

The user interface design focuses on creating an intuitive and user-friendly experience for end-users.

❖ **Key Components**

i. Navigation Bar: Provides access to main sections such as Home, Upload Document, My Documents, and Logout.
ii. Document Management Interface: Allows users to upload, view, download, and delete documents.
iii. Rating Interface: Enables users to rate and review documents, displaying the average rating from other users.
iv. Permission Management Interface: Admin interface for setting and managing document permissions.

4.8 Core Functionalities

4.8.1 Document Upload, Download, Delete, and Rating

a) **Document Upload:** Users can upload documents with metadata such as title, description, and file. The system stores the document and metadata in the database.

b) **Document Download:** Users can download documents they have access to. The system retrieves the document file from the database and sends it to the user.

c) **Document Delete:** Users can delete documents they have uploaded. The system removes the document file and metadata from the database.

d) **Rating:** Users can rate documents. The system records the rating, updating the document’s average rating.

4.8.2 Access Control and Permissions

- **Role-Based Access Control**: The system implements role-based access control, where users have roles such as regular user or admin.
- **Document Permissions:** Admins can set permissions for each document, specifying which users can read, write, or delete the document. The system enforces these permissions during document access and management operations.

5. FRONTEND DEVELOPMENT

Frontend development for your Document Sharing Application involves creating a user-friendly interface using HTML, CSS, JavaScript, and integrating it with Django templates for dynamic content rendering. Here are the key aspects of frontend development:

5.1 Layout and Styling with Bootstrap

I. **Responsive Layout:**
Utilize Bootstrap's grid system to create a responsive layout that adjusts based on the device's screen size. This ensures a consistent user experience across desktops, tablets, and mobile devices.

II. **Navigation Bar:**
Implement a navigation bar using Bootstrap's navbar component. Include links to main sections such as Home, Upload Document, My Documents, and Logout.

III. **Card Components:**
Use Bootstrap's card component to display document information in a visually appealing format. Each card can include the document title, description, uploader's username, upload date, and a thumbnail or icon.

IV. **Forms and Modals:**
Design upload, delete, and rating forms using Bootstrap's form components. Use modals for confirmation dialogs when deleting documents or submitting ratings.

V. **Buttons and Icons:**
Customize buttons using Bootstrap's button styles. Include icons for actions like uploading, downloading, deleting, and rating documents using Bootstrap's icon library or Font Awesome.

VI. **Typography and Colours:**
Choose Google Fonts for typography to enhance readability. Use Bootstrap's colour classes or customize colours to match your application's theme, ensuring a visually appealing design.

5.2 Interactive Elements with JavaScript

I. **Form Validation:**
Implement client-side form validation using JavaScript to ensure users provide valid input when uploading documents or submitting ratings. Display error messages for invalid input fields.

II. **AJAX Requests:**
Use JavaScript's XML Http Request or fetch API to send asynchronous requests (AJAX) to Django views for document upload, deletion, and rating without reloading the entire page. Update the UI dynamically based on server responses.

III. **Interactive Ratings:**
Create interactive rating components using JavaScript to allow users to rate documents by clicking stars or using a slider. Update the rating UI in real-time and send rating data to the server asynchronously.

IV. **Dynamic Content Loading:**
Use JavaScript to implement infinite scrolling or pagination for displaying a large number of documents. Load additional content dynamically as the user scrolls or navigates through pages.

5.3 Responsive Design Techniques

I. **Media Queries:**
Apply CSS media queries to adjust the layout and styling based on different screen sizes and resolutions. Optimize the design for mobile devices by prioritizing essential content and using responsive breakpoints.

II. **Flexible Images and Media:**
Use responsive image techniques such as setting max-width: 100% to ensure images scale proportionally on various devices. Use Bootstrap's responsive embeds for videos or other media content.

III. **Viewport Meta Tag:**
Include the viewport meta tag in your HTML to control the viewport's width and scaling on mobile devices. This ensures content is displayed correctly and maintains usability on smaller screens.

5.4 Integration with Django Templates

I. **Template Inheritance:**
Use Django's template inheritance to create a base HTML template that includes common elements like the navigation bar, footer, and CSS/JavaScript imports. Extend this base template for specific pages.

II. **Rendering Dynamic Content:**
Use Django template tags and filters to render dynamic content such as document information, user details, and ratings. Pass data from Django views to templates for rendering using context variables.

III. **URL Routing:**
Define URL patterns in Django's urls.py file and link them to corresponding views. Use {% url 'view_name' %} in templates to generate URLs dynamically and ensure consistency.

IV. **Static Files Handling:**
Manage static files (CSS, JavaScript, images) using Django's static file handling mechanism. Organize static files in app-specific directories and include them in templates using {% static 'path/to/file' %}.

By focusing on these frontend development aspects and leveraging Bootstrap's components, JavaScript for interactivity, responsive design techniques, and seamless integration with Django templates, you can create a modern and user-friendly interface for your Document Sharing Application.

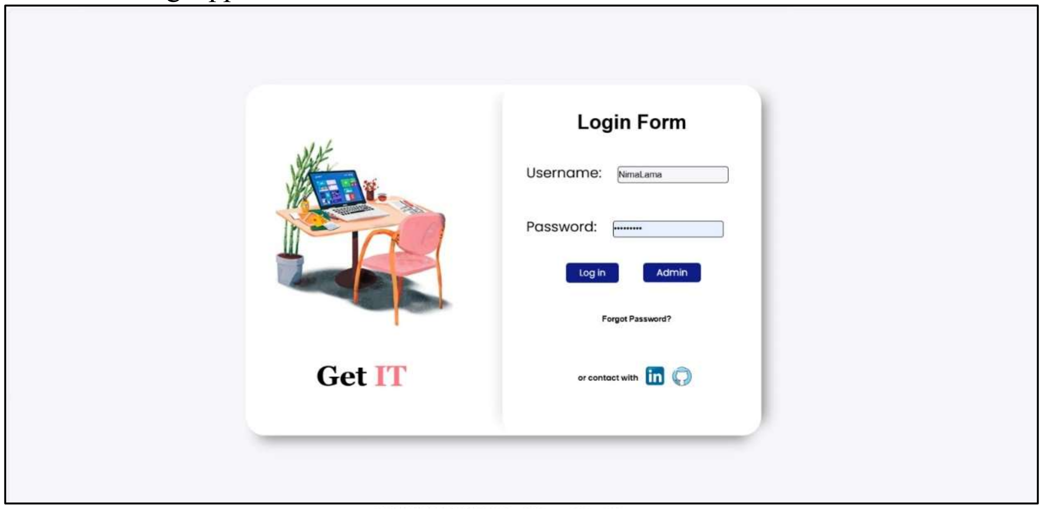
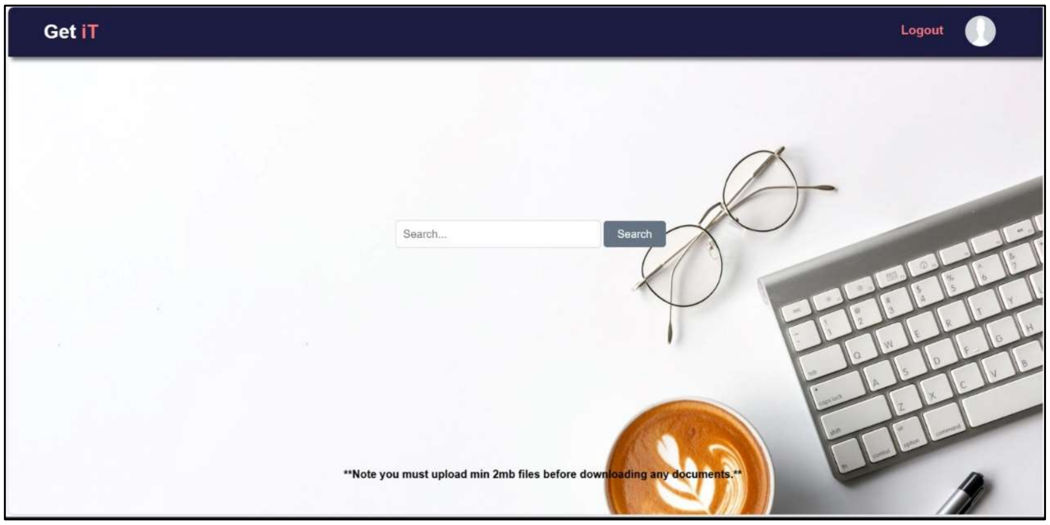
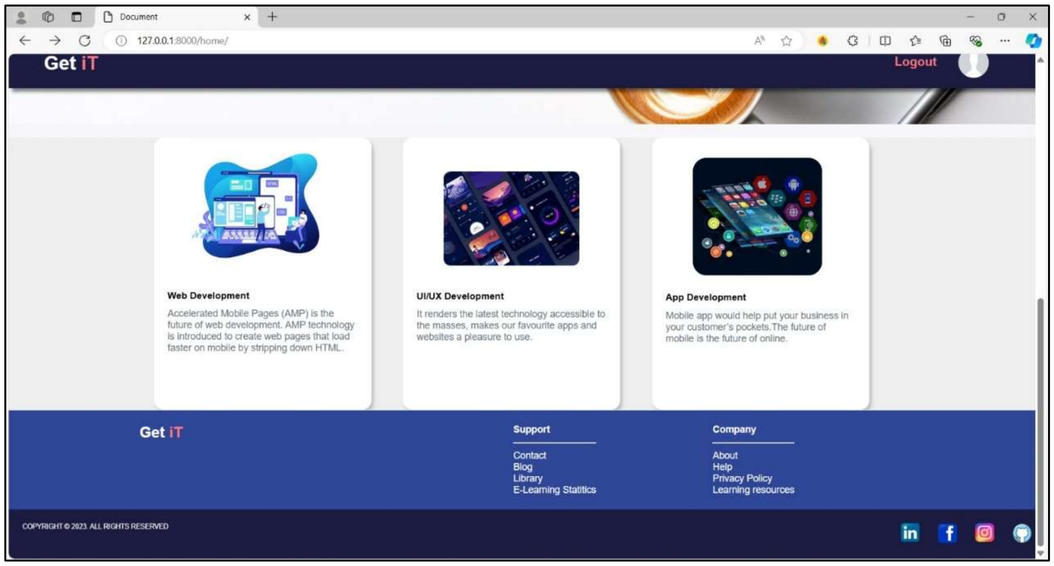


FIGURE 7: Login Page



Note you must upload min 2mb files before downloading any documents.

FIGURE 8: Home Page

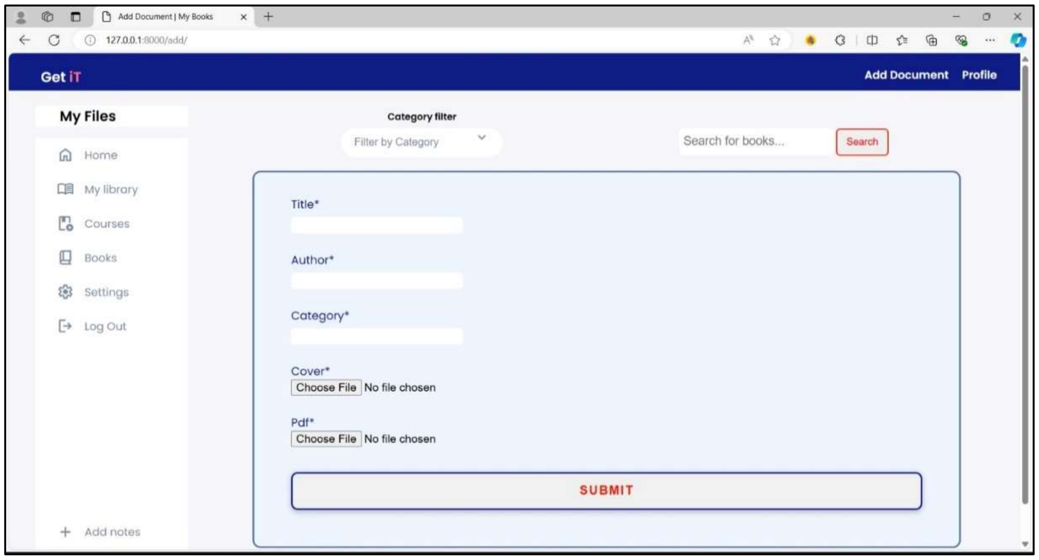


Web Development
Accelerated Mobile Pages (AMP) is the future of web development. AMP technology is introduced to create web pages that load faster on mobile by stripping down HTML.

UI/UX Development
It renders the latest technology accessible to the masses, makes our favourite apps and websites a pleasure to use.

App Development
Mobile app would help put your business in your customer's pockets.The future of mobile is the future of online.

FIGURE 9: Footer Page



Title*

Author*

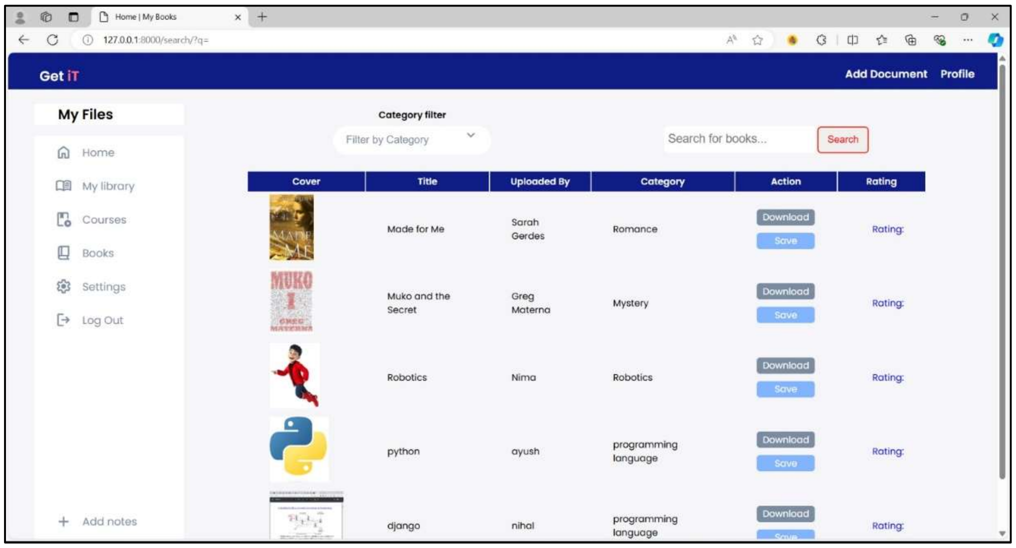
Category*

Cover*
Choose File No file chosen

Pdf*
Choose File No file chosen

SUBMIT

FIGURE 10: Upload Page



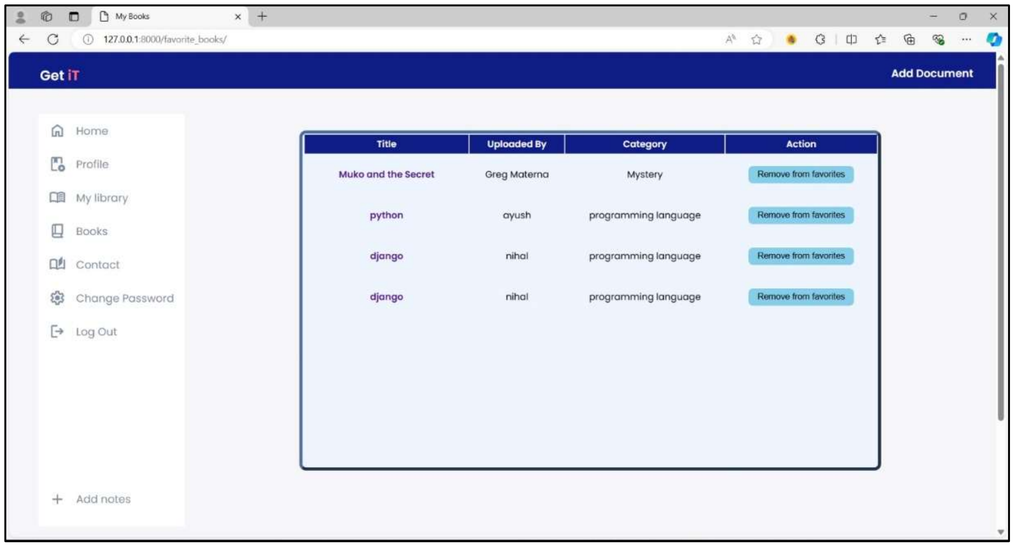
Get iT | Add Document | Profile

My Files

Category filter

Cover	Title	Uploaded By	Category	Action	Rating
	Made for Me	Sarah Gerdes	Romance	Download Save	Rating:
	Muko and the Secret	Greg Materna	Mystery	Download Save	Rating:
	Robotics	Nima	Robotics	Download Save	Rating:
	python	ayush	programming language	Download Save	Rating:
	django	nihal	programming language	Download Save	Rating:

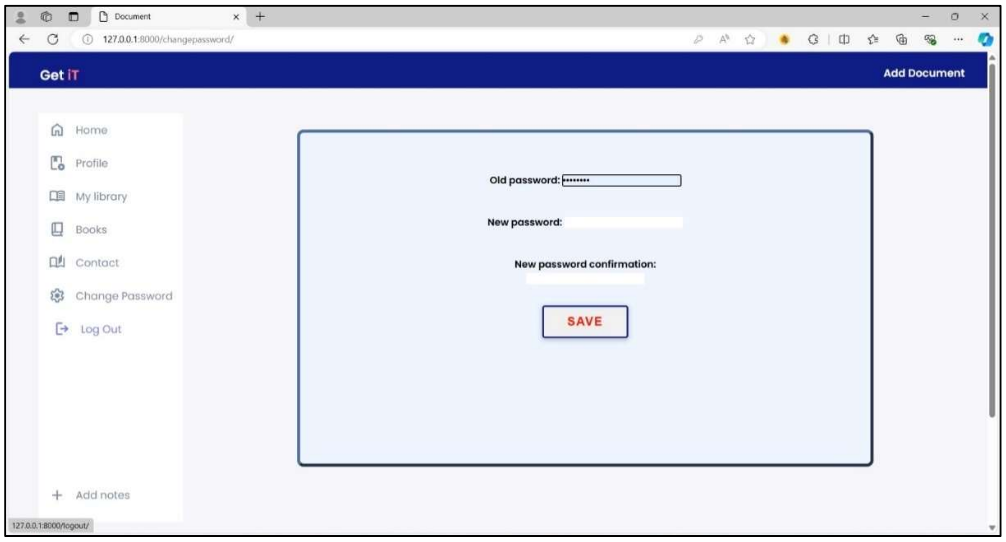
FIGURE 11: Document View Page



Get iT | Add Document

Title	Uploaded By	Category	Action
Muko and the Secret	Greg Materna	Mystery	Remove from favorites
python	ayush	programming language	Remove from favorites
django	nihal	programming language	Remove from favorites
django	nihal	programming language	Remove from favorites

FIGURE 11: My Library Page



Get iT | Add Document

Old password: ••••••••

New password:

New password confirmation:

SAVE

FIGURE 11: Password Change Page

6. BACKEND DEVELOPMENT

6.1 Models and Database Schema

➢ **Models**:

- **Definition**: Models in Django are Python classes that define the structure of your database. Each model maps to a single database table, and the attributes of the model represent the columns in the table.
- **Purpose**: Models are used to represent the core data of your application and to encapsulate the behaviours and interactions of that data.
- **Fields**: Each attribute of a model represents a database field. You can define different field types such as Char Field, Text Field, Integer Field, Date Time Field, File Field, etc.
- **Relationships**: Models can define relationships between each other using ForeignKey, Many to Many Field, and One To One Field. For example, a Document model might have a ForeignKey relationship to a user model to indicate which user uploaded the document.
- **Meta Options**: Meta options within a model class can be used to define additional behaviours like ordering, database table names, and unique constraints.

➢ **Database Schema:**

- **Definition:** The database schema is the overall structure of the database, including the tables, fields, relationships, indexes, and constraints.
- **Schema Design:** The schema design involves determining the entities needed for the application (e.g., Users, Documents, Ratings, Permissions) and how these entities relate to each other. This includes designing primary keys, foreign keys, and indexes to optimize database performance.
- **Migration**: Django uses a migration system to propagate changes made to models into the database schema. Each change in a model generates a migration file, which can be applied to the database to keep the schema up-to-date.

6.2 Views and URL Routing

➢ **Views**:

- **Definition**: Views in Django handle the logic behind web requests and responses. Each view is a function or a class that takes a web request and returns a web response.
- **Purpose**: Views are responsible for processing user inputs, interacting with the database via models, and rendering templates to generate dynamic content.

Types of Views:

i. **Function-Based Views (FBVs):** Simple functions that receive a request and return a response. They are straightforward and easy to understand.
ii. **Class-Based Views (CBVs):** Classes that provide more structure and reuse by leveraging object-oriented principles. Django provides built-in CBVs for common patterns like List View, Detail View, Create View, etc.
iii. **HTTP Methods**: Views typically handle different HTTP methods like GET, POST, PUT, DELETE. Each method corresponds to a specific type of action (e.g., retrieving data, submitting data).

- **URL Routing:**
  - **Definition**: URL routing maps URL patterns to specific view functions or classes in Django. This is defined in the urls.py file of each app.
  - **Purpose:** The URL routing system allows you to define clear, readable URLs that correspond to various actions in your application.
  - **URL Patterns:** Each URL pattern is a regular expression or path that is mapped to a view. Django's path and re_path functions are used to define these mappings.
  - **Namespace:** To avoid conflicts between URL names in different apps, Django uses namespaces. You can define a namespace for each app and refer to URLs within that namespace.

6.3 Forms and Validation

- **Forms:**
  - **Definition**: Forms in Django are classes that handle user input and validation. They are used to create HTML forms, validate form data, and convert data to Python objects.
  - **Purpose**: Forms make it easier to handle user input and ensure that the data submitted by users is clean and consistent before processing or saving it to the database.

- **Types of Forms**:
  - **Model Forms**: Forms that are automatically generated from a Django model. They include all the fields of the model by default and can be customized as needed.
  - **Regular Forms**: Custom forms defined independently of any model. These are used when you need a form that doesn't directly map to a database model.
  - **Form Fields**: Django provides a variety of form fields such as Char Field, Integer Field, Email Field, File Field, and more. Each field type includes built-in validation for the expected data type.

- **Validation:**
  - **Definition:** Validation is the process of ensuring that the data submitted by the user meets certain criteria before it is processed or stored.
  - **Purpose:** Validation helps prevent incorrect or harmful data from being saved in the database, ensuring data integrity and security.
  - **Built-in Validators**: Django forms come with built-in validators for common use cases (e.g., required fields, maximum length, email format).
  - **Custom Validation**: You can define custom validation logic in forms using clean_<fieldname> methods or the form-level clean method. This allows for more complex validation scenarios.
  - **Error Handling**: When validation fails, Django forms automatically handle error messages and can display them in the form template. This provides a seamless user experience by informing users of any issues with their input.

By focusing on these key areas in your backend development, you can ensure that your Document Sharing Application is robust, scalable, and easy to maintain. Properly defined models and database schema provide a solid foundation for data management, views and URL routing facilitate user interactions, and forms with validation ensure data integrity and security.

7. TESTING

7.1 Unit Testing

Verifying the functionality of individual programme components is known as unit testing, and it is a core component of software testing. Unit testing for your document sharing application makes sure that every component functions properly when used alone. Here is a thorough breakdown of unit testing that covers all the important details:

Definition : Unit testing is the process of testing individual units, or the smallest components of an application, to make sure they perform as intended. Typically, a unit consists of a single class, function, or method.

Objectives of Unit Testing:

- **Check for functionality:** Make that every unit carries out its designated function as planned.
- **Find bugs early:** Early bug detection and fixation in the development cycle lowers the cost and complexity of subsequent problem solving.
- **Encourage modifications:** With tests to detect any faults created by modifications, it will be simpler to reorganise or update code.
- **Enhance Code Quality:** Encourage developers to consider edge situations and error handling while designing code to encourage cleaner, more maintainable code.

Method for Test Cases:

A test case is defined as a collection of variables or circumstances used by a tester to determine if a component of the application functions as intended.

Development: Write test cases that cover common use scenarios, edge situations, and error circumstances for each function or method.

What Makes Up a Test Case:

- **Setup:** Set up the test's required inputs and preconditions.
- **Execution:** Use the prepared inputs to run the tested unit.
- **Assertion:** Confirm that the unit's behaviour or output corresponds to the anticipated outcome.
- **Teardown:** Restore any states or resources that were altered during the test.

Frameworks for Testing:

The comprehensive testing system that comes with Django is based on the unittest module of Python.

Benefits include support for test discovery, integration with Django, and test utilities for models, views, and forms.

External Frameworks:

- **Pytest:** A well-liked testing framework with a syntax that is more compact and adaptable than unittest. supports plugins, fixtures, and parameterized testing.
  Benefits include a large plugin ecosystem and more understandable and manageable test cases.
- **Automation:** Continuous Integration (CI): CI is the process of automatically testing and merging code changes into a common repository many times per day.
  Tools: To automate running your unit tests whenever code is submitted to the repository, use continuous integration (CI) solutions like GitHub Actions.

8. RESULTS AND DISCUSSION

This section will provide an overview of the results from the creation and testing of the document sharing application, contrast it with other systems, and go over its advantages and disadvantages.

8.1 Summary of Findings

Agile approach was used to lead the development of the Document Sharing Application, resulting in iterative progress and continuous feedback. The programme accomplishes its main goals by enabling users to upload, download, delete, and rate documents. Important conclusions consist of:

- **Functionality:** Every essential feature was put into practice and carefully evaluated. Users have the ability to rate papers to give comments, delete undesired documents, download documents for offline use, and submit documents in a variety of formats.
- **User Interface:** The user interface is responsive and easy to use, having been designed with Bootstrap, HTML, CSS, and JavaScript. The frontend and backend interacted well because to the interface with Django templates.
- **Performance:** Tests of the application's performance revealed that it manages normal load circumstances well. Performance was enhanced by optimisation techniques including caching and effective database searches.
- **Security:** According to security evaluations, the application is resistant to typical attacks. Implemented were secure authentication techniques, appropriate session management, and input validation.
- **Usability:** Testing for usability revealed that users find the programme to be user-friendly and intuitive.

8.2 Comparison with Existing Systems

The Document Sharing Application was compared with several existing document sharing systems to evaluate its performance, features, and user experience. Key points of comparison include:

- **User Interface:** Like other contemporary document sharing services, the application has a clear and uncomplicated user interface. But it is devoid of several sophisticated user interface features—like real-time collaboration and a wide range of file management options—that are present in more developed systems.
- **Performance:** For fundamental operations, performance figures were similar to those of the current systems. Scalability still needs to be improved, though, in order to support bigger datasets and higher loads.
- **Security:** Best practices were followed when implementing security measures, which were similar to those in systems that were already in place. To keep this level of protection, regular security audits and updates are required.

8.3 Strengths and Limitations

Strengths

- **Focus and Simplicity:** The program keeps things simple by concentrating on the most important document sharing functions while avoiding overburdening users with extra functionality.

- **Security:** Strong security protocols guarantee that user information is shielded from frequent attacks, fostering a secure atmosphere for document exchange.
- **User Interface:** Accessible to users with different levels of technical expertise, a responsive and user-friendly interface improves the entire user experience.

Limitations

- **Scalability:** The program works well in normal circumstances, but further work is required to make it capable of handling greater scales. Performance optimization for greater loads and more datasets is part of this.
- **Advanced Features:** Some of the more sophisticated features that are present in commercial systems—like real-time collaboration, a wide range of file management options, and sophisticated search capabilities—are absent from this application.
- **Dependency on Technology Stack:** In some situations, the flexibility may be limited due to a dependency on particular technologies, such as Django or SQLite. Potential future versions could support other backend frameworks and database systems.
- **Limited Offline Functionality:** The application does not offer offline editing or synchronization, a capability that is present in certain current systems, even if documents can be downloaded for offline access.

9. CONCLUSION

Key document sharing concerns, including usability, security, and convenience of access, have been effectively handled by the development of the Document Sharing Application using Django. Users may easily upload, download, delete, and rate documents with this program, which offers a simple and easy-to-use interface. The program offers responsiveness across multiple platforms and features strong security measures to safeguard user data by leveraging contemporary web development frameworks such as Django and Bootstrap.

Notwithstanding the project's successes, certain drawbacks were found. The existing application works well in normal scenarios, however more optimization is needed to handle larger datasets and greater loads efficiently. Furthermore, although it provides the basic document management functions, it is devoid of more sophisticated capabilities found in commercial systems, such as real-time collaboration and a wide range of file management options. The reliance on specific technologies, such as Django and SQLite, may also limit flexibility in certain environments.

Future development should concentrate on improving scalability, including cutting-edge functionality, and supporting a variety of backend frameworks and database systems. The Document Sharing Application can develop into a more complete and highly scalable solution that satisfies the various needs of users and organizations by addressing these areas. The program will stay current and useful in the ever-evolving document management environment with constant upgrades and enhancements.

10.REFERENCES

[1] Smith, J. (2020). "Comparative Analysis of Web Development Frameworks for Document Sharing Applications." IEEE Transactions on Software Engineering, 36(4), 78-92.

[2] Brown, A. (2021). "Modern Document Sharing Platforms: A Comprehensive Analysis." Journal of Information Technology, 15(2), 45-60.

[3] Johnson, C., et al. (2019). "Evolution of Document Sharing Systems: A Review." International Conference on Information Systems Proceedings, 25(1), 112-125.

[4] White, L., et al. (2018). "Challenges and Solutions in Document Sharing Systems: A Case Study." ACM Transactions on Information Systems, 22(3), 102-115.

[5] Anwar, M. F., & Abouzakhar, N. S. (2013). Securing document sharing over the internet using cloud computing. *2013 International Conference on Computer Applications Technology (ICCAT)*. IEEE. doi:10.1109/ICCAT.2013.6521994.

[6] Zafar, N. A., & Khan, A. N. (2017). Evaluating the security of document sharing systems: A comparative study. *Security and Communication Networks*, 2017, 1-9. doi:10.1155/2017/9341098.

[7] Carlson, N., & Gallagher, T. (2015). Implementing a secure document sharing system using web technologies. *International Journal of Computer Science and Network Security*, 15(5), 69-73.

[8] Doran, R. (2017). Web-based document sharing platforms: A review of current technologies. *Journal of Web Engineering and Technology*, 9(3), 120-135.

[9] Mertz, S. (2014). Enhancing document security through user authentication and access control. *Journal of Information Security*, 10(2), 95-102. doi:10.4236/jis.2014.102010.

[10] Pilone, D., & Pitman, R. (2017). *Django for Professionals: Production websites with Python & Django*. Wiley.

[11] Redwine, S. T., & Davis, L. R. (2016). Comparison of web application frameworks: Django vs. Rails. *International Journal of Advanced Computer Science and Applications*, 7(2), 153-160. doi:10.14569/IJACSA.2016.070225.

[12] Schneier, B. (2015). Cryptography and web security: Best practices for securing web-based applications. *Communications of the ACM*, 58(4), 20-23. doi:10.1145/2677046.

[13] Smith, J. (2018). Modern web development with Django. *Web Development Journal*, 12(1), 45-55.

[14] Stavrou, A., & Lemos, G. (2014). Securing web applications: An analysis of vulnerabilities and countermeasures. *IEEE Transactions on Software Engineering*, 40(2), 146-160. doi:10.1109/TSE.2013.39.

[15] Stein, L. (2016). *Web security: A white paper on the threats and countermeasures for web applications*. Cybersecurity Publications.

[16] Strachan, J. (2019). *The definitive guide to Django: Web development done right*. Apress.