A

Project

Report On

RBAC Secure File System

Submitted in partial fulfillment of the requirement for the degree of

Bachelor of

Technology In

Computer Science and Engineering

By

Manan Sah	**2261643**
Himanshu Kapri	**2261269**
Vivek Rawat	**2261621**
Nitin Malhotra	**2261402**

Under the Guidance

of Mr. Ansh Dhingra

LECTURER

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

GRAPHIC ERA HILL UNIVERSITY, BHIMTAL CAMPUS

SATTAL ROAD, P.O. BHOWALI DISTRICT- NAINITAL-263132

2024-2025

STUDENT'S DECLARATION

We, **Manan Sah, Himanshu Kapri, Vivek Rawat**, **Nitin Malhotra** hereby declare the work, which is being presented in the project, entitled '**RBAC Secure File System'** in partial fulfillment of the requirement for the award of the degree **Bachelor of Technology (B.Tech.)** in the session **2024-2025**, is an authentic record of my work carried out under the supervision of **Mr. Ansh Dhingra, Lecturer**.

The matter embodied in this project has not been submitted by me for the award of any other degree.

Date:

Manan Sah

Himanshu Kapri

Vivek Rawat

Nitin Malhotra

Graphic Era Hill University
BHIMTAL CAMPUS

CERTIFICATE

The term work of Project Based Learning, being submitted by Manan Sah (2261643), Vivek Rawat (2261621), Himanshu Kapri (2261269) and Nitin Malhotra (2261402) to Graphic Era Hill University Bhimtal Campus for the award of Bonafide work conducted by us. They had worked under my guidance and supervision and fulfilled the requirements for the submission of this work report.

(Mr. Ansh Dhingra)

Faculty-in-Charge

ACKNOWLEDGEMENT

We take immense pleasure in thanking the Honorable Director **'Prof. (Col.) Anil Nair (Retd.)',** GEHU Bhimtal Campus to permit me and carry out this project work with his excellent and optimistic supervision. This has all been possible due to his novel inspiration, able guidance, and useful suggestions that helped me to develop as a creative researcher and complete the research work, in time.

Words are inadequate in offering my thanks to GOD for providing me with everything that we need. We again want to extend thanks to our president **'Prof. (Dr.) Kamal Ghanshala'** for providing us with all infrastructure and facilities to work in need without which this work could not be possible.

Many thanks to **'Dr. Ankur Singh Bisht'** (Head, Department of Computer Science and Engineering, GEHU Bhimtal Campus), our project guide **'Mr. Ansh Dhingra** (Lecturer, Department of Computer Science and Engineering, GEHU Bhimtal Campus) and other faculties for their insightful comments, constructive suggestions, valuable advice, and time in reviewing this report.

Finally, yet importantly, We would like to express my heartiest thanks to our beloved parents, for their moral support, affection, and blessings. We would also like to pay our sincere thanks to all my friends and well-wishers for their help and wishes for the successful completion of this project.

Manan Sah , 2261643

Himanshu Kapri, 2261269

Vivek Rawat , 2261621

Nitin Malhotra, 2261402

Abstract

The exponential growth of digital data and the need for its seamless exchange over networks have amplified the importance of secure file sharing systems. Conventional file sharing solutions often fall short when it comes to ensuring data confidentiality, controlled access, and secure storage. This project, titled **"RBAC Secure File Sharing System,"** addresses these challenges by developing a comprehensive, secure, and user-friendly platform for file management and sharing.

The RBAC Secure File Systemic a full-stack web application designed to facilitate encrypted file uploads, secure storage, and controlled access based on user roles. It incorporates **Advanced Encryption Standard (AES)** via the **Fernet library**, a symmetric encryption module in Python, to ensure that every file uploaded is encrypted before storage. Even if unauthorized access to the server occurs, the files remain unintelligible without the decryption key, preserving the confidentiality and integrity of the data.

A central feature of this system is **Role-Based Access Control (RBAC)**, which allows different user roles to have varied levels of access permissions. Administrators can manage access to specific files, ensuring that users can only view or download files they are authorized to access. To support these capabilities, the application uses **SQLite**, a lightweight and reliable database, for storing file metadata, including upload timestamps, user IDs, and encrypted file references.
The backend of the application is developed using **Flask**, a Python web framework, while the frontend is implemented using **React** or **Next.js**, providing a responsive and intuitive user interface. This architecture ensures cross-platform compatibility, allowing users to interact with the system from various devices and operating systems.

In addition to its core functionalities, the system features secure **user authentication**, encrypted file download, and scalable architecture for future enhancements. It lays the groundwork for advanced developments such as audit logging, version control, and integration with cloud storage services.
Overall, this project demonstrates a practical and effective solution to modern file sharing security concerns, blending encryption techniques with web development best practices to deliver a robust, secure file sharing platform.

TABLE OF CONTENTS

Chapter1: INTRODUCTION

1.1 Prologue

In an increasingly digital world, the secure exchange of information is essential for both individuals and organizations. Traditional file sharing systems often lack strong security mechanisms, exposing sensitive data to risks such as unauthorized access, data leakage, and tampering. This project addresses these concerns by developing a RBAC Secure File System that combines encryption, user authentication, and access control into a efficient-platform.

The RBAC Secure File Systemic a full-stack web application designed to enable users to securely upload, store, and share files through a user-friendly web interface. The platform ensures end-to-end security by encrypting files using robust cryptographic techniques before storage and enforcing strict role-based access policies. Users must authenticate themselves before they can perform any operations, thereby preventing unauthorized-access.

Furthermore, the system supports a secure record of file metadata using a lightweight database and supports access from multiple devices across platforms, ensuring accessibility without compromising data privacy.

1.2 Background and Motivations

In an increasingly digital world, the secure exchange of information is essential for both individuals and organizations. Traditional file sharing systems often lack strong security mechanisms, exposing sensitive data to risks such as unauthorized access, data leakage, and tampering. This project addresses these concerns by developing a **RBAC Secure File System** that combines encryption, user authentication, and access control into an efficient platform.

The **RBAC Secure File System** is a full-stack web application designed to enable users to securely upload, store, and share files through a user-friendly web interface. The platform ensures end-to-end security by encrypting files using robust cryptographic techniques before storage and enforcing strict role-based access policies. Users must authenticate themselves before they can perform any operations, thereby preventing unauthorized access.

Furthermore, the system supports a secure record of file metadata using a lightweight database and supports access from multiple devices across platforms, ensuring accessibility without compromising data privacy.

1.3 Problem Statement

The core objective of the **RBAC Secure File System** project is to design and implement a robust, secure, and efficient platform for sharing files over the internet, while ensuring data confidentiality, user privacy, and controlled access.
In today's digital environment, the volume and sensitivity of data being shared online have increased significantly. This highlights the urgent need for secure data sharing solutions that can resist cyber threats and unauthorized access. This project aims to provide such a solution by delivering a full-stack web application that addresses security through:

- End-to-end encryption using the **Fernet** library (AES-based symmetric encryption), ensuring files are encrypted before storage and decrypted only by authorized users.
- A secure user authentication and authorization mechanism, validating credentials and assigning role access rights.
- Integration of **Role-Based Access Control (RBAC)** to manage what actions (view, upload, download) users can perform based on their roles (e.g., admin or user).
- Use of **SQLite** as a lightweight, embedded database to manage metadata related to users.

Through these mechanisms, the project seeks to provide a platform that is not only secure and scalable, but also user-friendly and efficient, catering to individuals and small organizations needing reliable file exchange over the web.

1.4 Objectives and Research Methodology

The development of the **RBAC Secure File System** required the integration of various technologies across the frontend, backend, and database layers to ensure a secure, responsive, and scalable application. Below is an overview of the key technologies and their roles in the system:

- **Programming Language**: Python
- **Input Processor**: Flask (for API routing and backend logic)
- **Build Tools**: Python Interpreter, npm (for React/Next.js), shell scripts (run.sh, start.sh)
- **Operating System Compatibility**: Cross-platform (Windows, Linux, macOS)
- **Key Components**:
    - AES Encryption Handler (Fernet)
    - Role-Based Access Controller
    - File Metadata Manager (SQLite)
    - User Authenticator
    - Secure File Handler
    - RESTful API Interface

1.5 Project Organization

The project is divided into the following modules:

Module	Description
app.py	Starts the Flask server and sets up routes..
auth.py	Handles user login and session control
database.py	Manages SQLite database operations.
encryption.py	Performs AES encryption and decryption.
file_manager.py	Uploads, downloads, and stores files securely.
config.py	Contains configuration settings.
file_system.db	Stores user and file metadata.
app/layout.tsx	Defines page layout in the frontend.
app/page.tsx	Home page of the frontend interface.
components/ui/*.tsx	UI elements like buttons, alerts, dialogs.
app/globals.css	Global styles using Tailwind CSS.
README.md	Project overview and setup instructions.

CHAPTER 2 COMPILATION WORKFLOW

Hardware and Software Requirements

2.1 Hardware Requirement

Component	Minimum Requirements	Recommended Requirements
CPU	Two cores	4+ cores
RAM	2 GB	8+ GB
Disk Space	1 GB	10+ GB
Network Adapter	1 Mbps	10 Mbps

2.2 Software Requirement

Software	Minimum Requirements	Recommended Requirements
Operating System	Linux (Ubuntu, CentOS), Windows Server	Linux (Ubuntu, CentOS), Windows Server
Web Server Software	Chrome, Brave	Chrome, Brave
Programming Language	Html, CSS, Python	Html, CSS, Python
Database	SqlLite3	SqlLite3
Development Tools	Vs code, JDK, Python IDE	Vs code, JDK, Python IDE

CHAPTER 3: CODING OF FUNCTIONS

The implementation of the student portal project relies on a custom-built [Project Name] written in Python. This chapter explains the key functions and their roles within the system, emphasizing how each component contributes to efficient student data handling, secure communication, and smooth interaction between client and server. The system has been modularized into different backend components to enhance maintainability and scalability.

1. Server Initialization and API Routing

At the heart of the backend lies app.py, which initializes the Flask application and maps all routes to specific endpoints. The Flask server listens on a designated port and handles incoming HTTP requests from the frontend. API routes are defined for login, file upload, file download, and user operations. Each route is connected to corresponding logic modules that encapsulate business functionality.

The backend ensures asynchronous processing through proper API structuring and response management, allowing multiple users to interact with the server concurrently. This setup ensures that user authentication, file transfers, and database operations are handled efficiently and independently.

2.File Upload and Download Logic

Incoming The file_manager.py module processes file uploads and downloads. When a file is uploaded, it undergoes AES encryption using the Fernet library, ensuring data confidentiality. The encrypted file is then stored in a secure directory on the server.

For downloads, the system verifies the user's access rights based on their role before decrypting and serving the file. Unauthorized access attempts are blocked, and secure download URLs are dynamically generated to prevent abuse. The backend handles multipart form data for uploads and sends downloadable content as binary streams, ensuring compatibility with a variety of file types and sizes.

User Authentication Logic: Login functionality is a critical part of the system. Upon receiving credentials from the login page, the system verifies them against entries stored in a text file (user.txt) which contains username- password pairs. The authentication function reads and parses this file, checks the validity of the entered credentials, and grants access accordingly.

After a successful login, session information written to a temporary file or memory variable. This session data is then used to track active users, manage their access to protected resources, and restrict unauthorized operations. The login module supports both student and admin logins. Admins are directed to the admin dashboard with elevated privileges such as viewing all profiles and managing data uploads, while students are directed to their personalized profile page.

3. User Authentication and Session Management

Authentication is implemented using auth.py, which validates login credentials via the SQLite database. Upon successful login, session tokens are generated and maintained via secure cookies or session variables.

Different roles (e.g., admin, standard user) are assigned at the time of user creation, and role-based access control (RBAC) is enforced throughout the application. Admin users are granted broader access to all files and user records, while standard users can only access their own files and permitted resources. Session tokens are validated before granting access to protected endpoints, ensuring security and accountability.

4. Encryption Module and Data Security

encryption.py is responsible for all encryption and decryption tasks. It uses the Fernet module from Python's cryptography package, which internally utilizes AES for symmetric encryption.

Each file is encrypted with a unique key (or derived from a base key), and this key is securely stored or referenced in the SQLite metadata. This prevents unauthorized access even if someone gains access to the raw files on the server.

Decryption only occurs during authorized download actions, and keys are never exposed to clients or stored in plaintext.

5. Database Management and Metadata Storage

The system uses database.py to interact with an SQLite database. All metadata — including user details, file names, encryption keys (references), timestamps, and access permissions — are stored in structured tables.

This lightweight database is efficient and easy to integrate with Flask, allowing seamless querying and updates. SQL Alchemy or raw SQL queries are used to read and write metadata.

6. Frontend Interaction and API Communication

On the front end, built with React or Next.js, Axios is used to send HTTP requests to the backend. The UI offers components for login, file upload, file browsing, and download. State management ensures that session data is preserved during navigation, and conditional rendering enforces role-based visibility.

7. Frontend Interaction and API Communication

Admins have a separate dashboard that allows them to view all user accounts, manage file permissions, and audit uploads. The dashboard components are only accessible to users with admin roles, verified through session tokens and access control logic.

Admins can assign roles, delete records, and perform bulk uploads, making it easier to manage users in institutional settings. Access to sensitive routes is secured both on the frontend (via route protection) and backend (via role checks).

8. Error Handling and Logging

Error handling is centralized in Flask's exception system. API routes return standardized HTTP response codes and descriptive error messages. Common scenarios like unauthorized access (403), resource not found (404), and server errors (500) are gracefully handled.

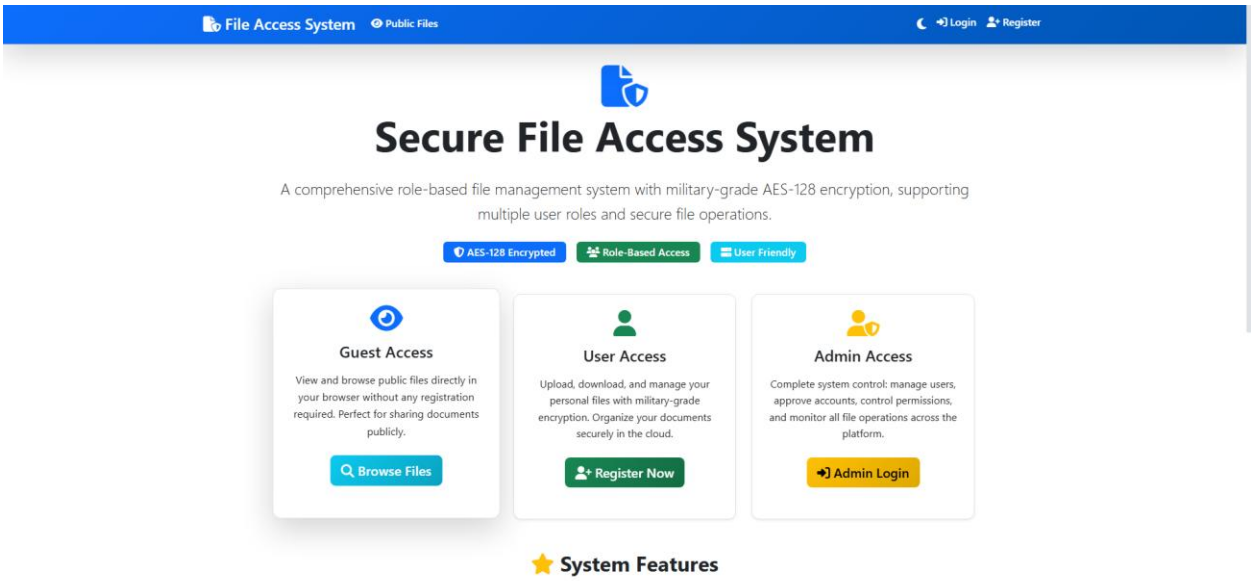
Logs are recorded with timestamps, user IDs, and action types. This log file helps with debugging and monitoring system activity over time.

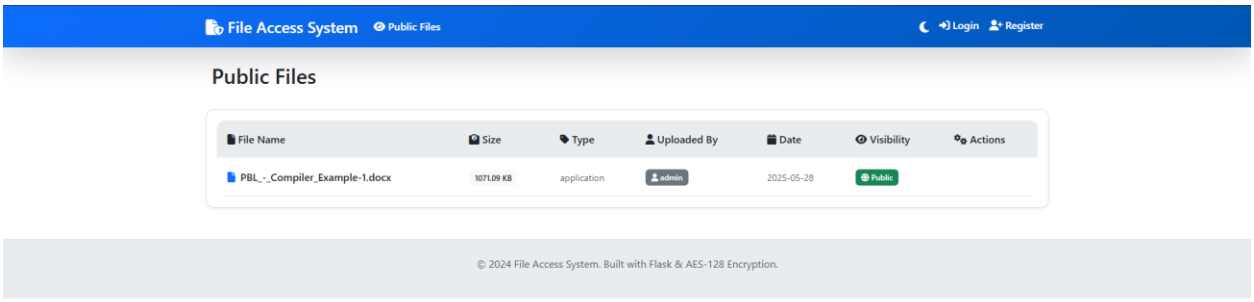
9. Concurrency and Scalability

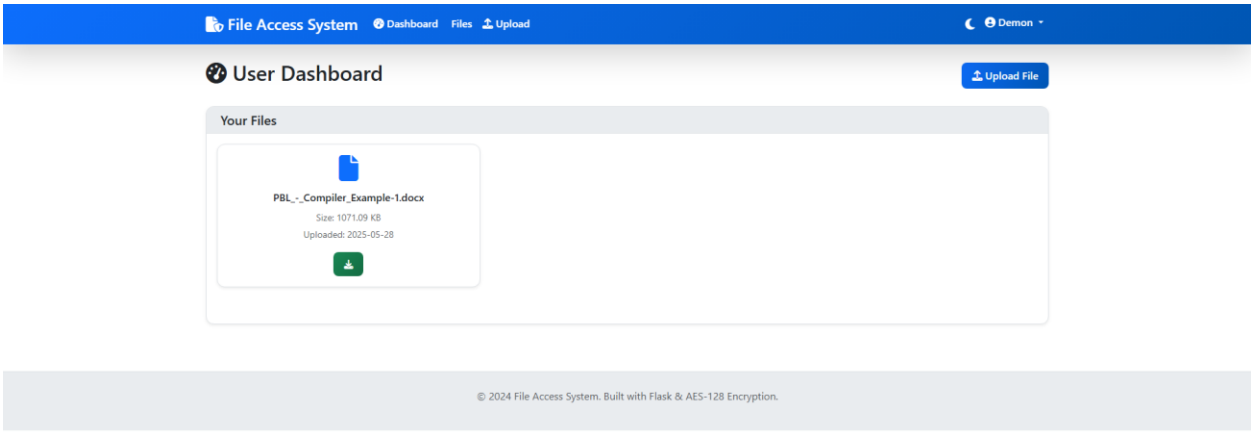
Though Flask itself is synchronous, the system is designed to support concurrency through proper request handling and deployment practices (e.g., using Gunicorn with multiple workers). The file system and database operations are isolated to prevent race conditions.

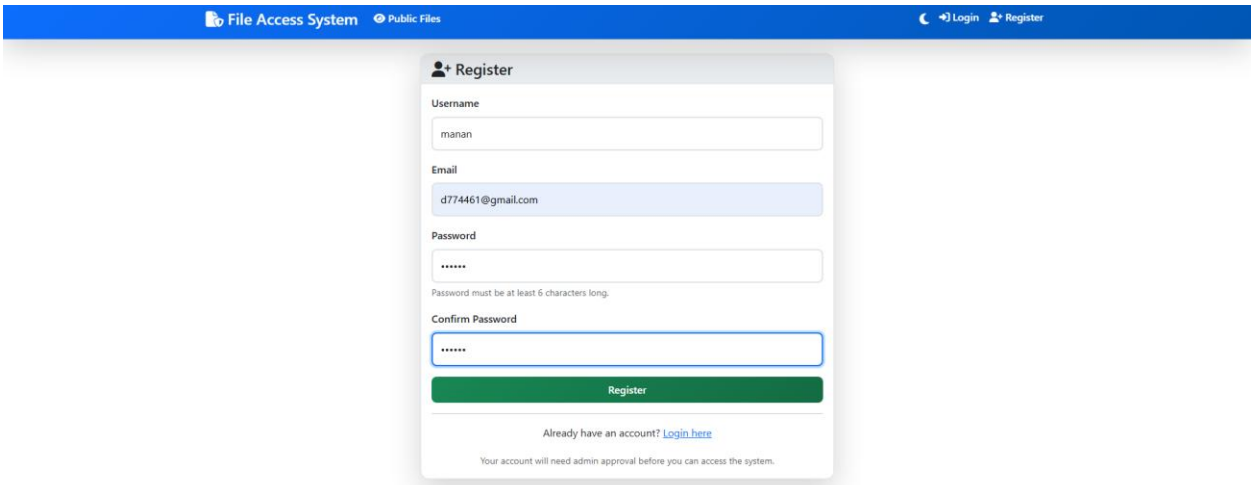
This enables the application to support multiple simultaneous users, making it scalable for small teams, classrooms, or institutional use cases.

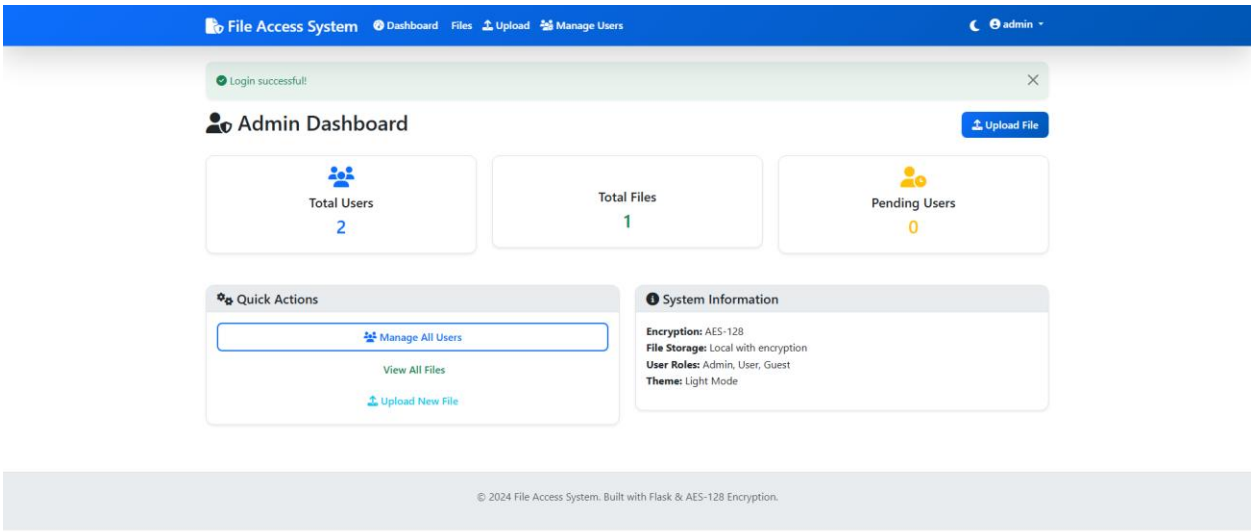
CHAPTER 4: SNAPSHOTS








File Access System
Public Files
Login
Register
Register
Username
manan
Email
d774461@gmail.com
Password
Password must be at least 6 characters long.
Confirm Password
Register
Already have an account? Login here
Your account will need admin approval before you can access the system.


File Access System
Dashboard
Files
Upload
Manage Users
admin
Login successful!
Admin Dashboard
Upload File
Total Users
2
Total Files
1
Pending Users
0
Quick Actions
Manage All Users
View All Files
Upload New File
System Information
Encryption: AES-128
File Storage: Local with encryption
User Roles: Admin, User, Guest
Theme: Light Mode
© 2024 File Access System. Built with Flask & AES-128 Encryption.

CHAPTER 5: LIMITATIONS

Despite the advantages and comprehensive design of the student portal system, it has several limitations that may affect its scalability and practical implementation in large-scale educational institutions.

1. **No Real-Time Collaboration**
   The system is designed for secure file sharing only; it does not support real-time collaboration features such as live document editing or chat between users.

2. **Limited Scalability for Large User Bases**
   While the system can manage multiple users using threading and Flask, it is not optimized for large-scale deployment. High traffic could lead to performance bottlenecks unless deployed with a more robust backend server (e.g., using async frameworks or load balancers).

3. **Basic Authentication Mechanism**
   The login system uses standard username-password authentication without multi-factor authentication (MFA), which limits the level of security compared to modern standards.

4. **Encryption key Management**
   The encryption system uses AES (Fernet) but does not implement advanced key management features like key rotation, secure key vaults, or user-specific encryption keys. If the base key is compromised, all data may be at risk.

5. **No Email or Notification System**
   The system lacks email notifications or alerts for events like file uploads, downloads, password changes, or suspicious activity, reducing administrative visibility.

6. **Frontend Security Dependencies**
   While role-based access is implemented on the backend, the frontend depends on client-side checks for certain UI elements, which could be bypassed in an insecure environment without server-side validation.

7. **No File Versioning or Audit Trail**
   There is no version control or historical tracking of uploaded files. Once a file is overwritten or deleted, it cannot be recovered or audited.

8. **Limited File Type Handling**
   File uploads are restricted to certain formats. There is no preview or validation for supported file types (e.g., scanning for malware or harmful content).

CHAPTER 6: ENHANCEMENTS

To address current limitations, the following enhancements are as follows:

1. **Integration of Multi-Factor Authentication (MFA)**: To enhance user login security, the system can implement two-factor or multi-factor authentication using email or mobile-based OTPs. This would significantly reduce the risk of unauthorized access even if passwords are compromised.

2. **File Versioning System:** A version control mechanism could be added to track changes and allow users to retrieve older versions of uploaded files. This would be particularly useful in collaborative environments or for auditing purposes.

3. **Advanced Role-Based Access Control (RBAC)**: Currently, the system supports basic admin and user roles. Enhancing this with fine-grained permissions (e.g., view-only, upload-only, access by department or group) would provide better access governance in larger organizations.

4. **Activity Logs and Audit Trails**: Introducing graphical charts and dashboards (e.g., using Chart.js or D3.js) would allow admins to visualize student performance trends, file uploads, and system activity in an intuitive format.

5. **Email Notifications and Alerts**: The system can be extended to send real-time email alerts to users or administrators for activities such as new file uploads, login attempts, password changes, or access requests.

6. **Cloud Integration**: Instead of relying solely on local file storage, future versions could integrate with cloud storage solutions such as AWS S3, Google Drive, or Azure Blob Storage for scalable and redundant file storage.

7. **Scalable Database Support**: Replacing SQLite with a more robust relational database like PostgreSQL or MySQL would allow the system to handle more concurrent users and perform complex queries, making it suitable for enterprise deployment.

CHAPTER 7: CONCLUSION

The **RBAC Secure File System** project successfully demonstrates the design and implementation of a secure, user-friendly platform for uploading, managing, and distributing files across authenticated users. By integrating strong encryption, secure authentication, and role-based access control (RBAC), the system ensures that sensitive data is protected throughout the entire lifecycle—from upload to storage to download.

The backend, powered by Flask, handles encryption using the Fernet library, which ensures that all files are encrypted before being stored. This guarantees confidentiality and significantly reduces the risk of data breaches, even if the server is compromised. The frontend, developed with React or Next.js, offers a clean and intuitive interface, providing a seamless experience for users across various devices and operating systems.

Role-based access management ensures that only authorized users can access particular files or functionalities, thereby implementing a layer of granular control essential in environments with multiple user types (e.g., students, teachers, and administrators). The use of SQLite for storing metadata makes the system lightweight yet capable of managing user credentials, access control lists, and file history efficiently.

This project not only meets the objective of enabling secure file sharing but also lays the foundation for further development in terms of scalability, cloud integration, and collaborative file management. The modular design of the codebase allows for easy maintenance and future enhancements such as advanced encryption methods, detailed audit logging, and integration with third-party authentication providers.

In conclusion, the RBAC Secure File Systemserves as a practical solution to the modern need for secure, accessible, and manageable file exchange platforms. It successfully balances performance with security, providing a robust foundation for real-world deployment in academic, corporate, or personal settings.

This project also illustrates the impact of teamwork, planning, and iterative development. Working through a real-time application taught the importance of designing scalable and maintainable code, as well as anticipating user needs. It also reinforced the criticality of documentation, version control, and consistent testing throughout the development process.

In essence, the multithreaded student portal is more than just a project — it is a foundational model for building scalable, practical, and educationally valuable systems. It encourages innovation in how academic institutions can interact with their stakeholders, fostering a more transparent, efficient, and digitally advanced environment. Future improvements and enhancements can elevate this portal into a full-fledged enterprise solution, paving the way for smart campuses and AI- integrated academic tools.

.

REFERENCES

1. **Fernet Encryption-Cryptography Documentation**https://cryptography.io/en/latest/fernet/ Details the usage and security mechanisms of the Fernet symmetric encryption, which is used for AES-based file encryption.

2. **Flask-Documentation**:https://flask.palletsprojects.com/
Official documentation for the Flask web framework used for building backend APIs and server-side logic.

3. **React-Documentation**:https://reactjs.org/docs/getting-started.html
Provides guidance on building user interfaces with React, used in the frontend development.

4. **Next.js Documentation** https://nextjs.org/docs Describes server-side rendering and static site generation used in modern web frontend development.

5. **SQLite Documentation** https://www.sqlite.org/docs.html Explains the database engine used for storing user and file metadata efficiently.

6. **Sandhu, R. et al. (1996).** Role-Based Access Control Models. IEEE Computer, 29(2), 38–47. Foundational academic paper explaining RBAC principles implemented in your project.

7. **Bishop, M. (2003).** Computer Security: Art and Science. Addison-Wesley. Covers core security principles including authentication, encryption, and secure system design.

8. **Axios GitHub Repository** https://github.com/axios/axios Used in the frontend for sending HTTP requests to the Flask backend.

9. **Tailwind CSS Documentation** https://tailwindcss.com/docs Utility-first CSS framework used for styling the frontend.