

ATMIYA UNIVERSITY

RAJKOT



A

Report On

Secure File Vault

Under subject of

MINI PROJECT

B.TECH, Semester – VII

(Computer Engineering)

Submitted by:

Sakshi Parekh (220002050)

Mr. Janak Maru

(Faculty Guide)

Ms. Tosai M. Bhalodia

(Head of the Department)

Academic Year

(2025-26)

CANDIDATE'S DECLARATION

We hereby declare that the work presented in this project entitled “**Secure File Vault**” submitted towards completion of project in **7th Semester** of B. Tech. (Computer Engineering) is an authentic record of our original work carried out under the guidance of “**Mr. Janak Maru**”.

We have not submitted the matter embodied in this project for the award of any other degree.

Semester: 7th

Place: Rajkot

Signature:

Sakshi Parekh (220002050)

ATMIYA UNIVERSITY
RAJKOT



CERTIFICATE

Date:

This is to certify that the “**Secure file Vault**” has been carried out by **Sakshi Parekh** under my guidance in fulfillment of the subject Mini Project in COMPUTER ENGINEERING (7th Semester) of Atmiya University, Rajkot during the academic year 2024.

Mr. Jank Maru

(Project Guide)

Ms. Tosai M. Bhalodia

(Head of the Department)

ACKNOWLEDGEMENT

I have taken many efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organizations. I would like to extend our sincere thanks to all of them.

I am highly indebted to **Janak Maru** for their guidance and constant supervision as well as for providing necessary information regarding the Project titled “**Secure file Vault**”. I would like to express our gratitude towards staff members of the Computer Engineering Department, Atmiya University for their kind co-operation and encouragement which helped us in completion of this project.

I even thank and appreciate my colleague in developing the project and people who have willingly helped us out with their abilities.

Sakshi Parekh (220002050)

ABSTRACT

The Secure File Encryption & Decryption Portal is a robust, web-based application designed to provide military-grade encryption for sensitive user files. The system is built on a zero-knowledge architecture, ensuring that all encryption and decryption processes occur client-side, meaning only the user possesses the keys to access their data. The platform supports multiple industry-standard cryptographic algorithms, including AES-256, DES, and RSA, catering to various security needs. Users can securely register and log in to manage their files. The application features an intuitive interface for selecting files, choosing encryption algorithms, and optionally using a passphrase for an additional layer of security. All user and file metadata is securely stored in a database. The core purpose of this project is to offer a secure, reliable, and user-friendly solution for protecting digital assets from unauthorized access, leveraging modern web technologies to ensure both security and performance.

INDEX

Sr. No.	TITLES		Page No.
	Acknowledgement		3
	Abstract		4
	List of Figures		7
	List of Tables		8
1.	Introduction		9
	1.1	Purpose	9
	1.2	Scope	9
	1.3	Technology and tool	10
2.	Project Management		11
	2.1	Project Planning	11
	2.2	Project Scheduling	11
	2.2.1	Gnatt Chart	11
	2.3	Risk Management	12
	2.3.1	Risk Identification	12
	2.3.2	Risk Analysis	13
3.	System Requirements Study		14
	3.1	Hardware and Software Requirements	14
	3.1.1	Server side hardware requirement	14
	3.1.2	Software requirement	14
	3.1.3	Client Side requirement	14
	3.2	Constraints	15
	3.2.1	Hardware Limitation	15
	3.2.2	Reliability requirements	15
	3.2.3	Safety and Security Consideration	15
4.	System Analysis		16
	4.1	Study of Current System	16
	4.2	Problem and Weaknesses of Current System	16
	4.3	Requirements of New System	16
	4.3.1	User Requirements	16

		4.3.2	System Requirements	16
	4.4	Feasibility Study		17
	4.5	Feature Of New System		18
5	System Design			19
	5.1	Input /output interface		19
		5.1.1	Personal Information Screen	19
		5.1.2	Sign-up Page	20
		5.1.3	Main Page	21
	5.2	Interface Design		22
		5.2.1	Class Diagram	22
		5.2.2	Use Case Diagram	22
		5.2.3	Activity Diagram	23
		5.2.4	Data Flow Diagram	23
		5.2.5	State Diagram	24
		5.2.6	E-R Diagram	25
		5.2.7	Sequence Diagram	26
6	Code Implementation			27
	6.1	Implementation Environment		27
	6.2	Program/Module Specification		27
	6.3	Coding Standards		27
7	Testing			28
	7.1	Testing Strategy		28
	7.2	Testing Method		28
		7.2.1	Unit Testing	28
		7.2.2	Integration Testing	28
		7.2.3	Validation Testing	28
	7.3	Test Cases		29
		7.3.1	Test Suite	29
8	Limitations and Future Enhancement			30
	8.1	Limitations		30
	8.2	Future Enhancement		30
9	Conclusion			31
10	References			32

LIST OF FIGURES

Fig. No.	Figure Title	Page No.
5.1.1	User Registration Interface	19
5.1.2	User Login Interface	20
5.1.3	Main Page	21
5.2.1	Class Diagram	22
5.2.2	Use Case Diagram	22
5.2.3	Activity Diagram	23
5.2.4	Data Flow Diagram	23
5.2.5	State Diagram	24
5.2.6	E-R Diagram	25
5.2.7	Sequence Diagram	26

LIST OF TABLES

Table No.	Table Title	Page No.
2.3.2	Risk Analysis Table	13
3.1.1	Server-side Hardware Requirement	14
3.1.2	Software Requirement	14
3.1.3	Client-side Requirement	14
7.3.1	Test Cases & Expected Results	29

CHAPTER – 1

INTRODUCTION

1.1. Purpose

The purpose of the project entitled "**Secure File Encryption & Decryption Portal**" is to develop a highly secure and user-friendly web application that empowers users to protect their sensitive files using advanced cryptographic techniques. The system aims to automate the process of file encryption and decryption, providing a zero-knowledge architecture where all cryptographic operations are performed on the client's browser, ensuring that the server never has access to unencrypted files or the keys used to protect them. This project serves to address the growing need for personal data security in the digital age by offering a reliable, accessible, and trustworthy tool for data protection.

1.2 Scope

The scope of this project encompasses the following key areas:

- **User Management:** Secure user registration and authentication system.
- **File Handling:** Allows users to upload files for encryption and download files for decryption.
- **Cryptographic Operations:** Implementation of three major encryption algorithms: AES-256 (symmetric), DES (symmetric), and RSA (asymmetric).
- **Key Management:** Automatic generation of encryption keys. Optional passphrase-based key derivation for enhanced security.
- **Client-Side Processing:** All cryptographic operations (encryption/decryption) are performed within the user's browser to uphold the zero-knowledge principle.
- **Data Storage:** Only encrypted files and encrypted metadata are stored on the server. Original file contents and plaintext keys are never stored.
- **User Interface:** An intuitive web interface that guides the user through the process of securing their files.

1.3 Technology and Tools

Front End technology used in the website:

- **HTML:** Used for creating the structure and content of the web pages.
- **CSS:** Used for styling the web pages, ensuring a responsive and visually appealing layout that works on various device screens.
- **JavaScript (ES6+):** Used to create dynamic and interactive elements on the client side. It handles the core logic for client-side file processing, cryptographic operations via Web Crypto API or similar libraries, and DOM manipulation.

Back End technology used in the website:

- **Node.js:** A JavaScript runtime environment used to build the server-side application. It handles HTTP requests, user authentication, and database interactions.
- **Express.js:** A web application framework for Node.js that simplifies the process of building robust APIs and handling server routes.
- **MongoDB:** A NoSQL database used to store user credentials (hashed passwords), metadata about uploaded files (e.g., filename, encryption algorithm used), and the encrypted files themselves or links to their storage location.
- **Other Libraries:** Various npm packages such as bcryptjs for password hashing, jsonwebtoken for authentication, and crypto-js or the native crypto module for server-side cryptographic functions if needed

CHAPTER – 2

PROJECT MANAGEMENT

2.1 Project Planning

Project planning for the "Secure File Encryption & Decryption Portal" involved defining the project's scope, objectives, and deliverables. The plan outlined the development phases: requirement analysis, system design, implementation, testing, and deployment. Key milestones included completing the user authentication module, implementing each encryption algorithm, and achieving full client-side processing.

2.2 Project Scheduling

The project was scheduled over a period of 12 weeks. The schedule allocated time for research on cryptography, core development of front-end and back-end components, integration testing, and documentation. Agile methodology was adopted, with weekly sprints to track progress and adapt to challenge

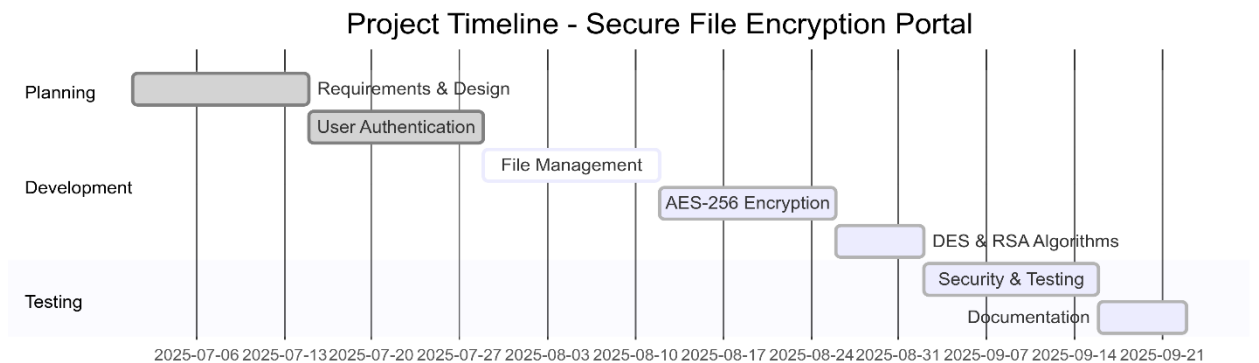


Figure 2.2.1 Gantt Chart

- **July 15-28:** User Authentication System
- **July 29-Aug 11:** File Upload/Download Features
- **Aug 12-25:** AES-256 Encryption Core
- **Aug 26-Sep 1:** DES & RSA Algorithms
- **Sep 2-15:** Security Testing & Optimization
- **Sep 16-22:** Documentation & Deployment

2.3 Risk Management

Risk management consists of a series of steps that help a software development team to understand and manage uncertain problems that may arise during software development and can plague a software project.

2.3.1 Risk Identification :

We identified potential problems that could impact the project's success, security, and functionality, categorizing them as follows:

➤ **Technical Risks:**

- **Faulty Encryption Implementation:** The core risk was making mistakes while integrating the complex AES, DES, and RSA algorithms. An error here wouldn't be obvious but would completely break the system's security, leaving files unprotected.
- **Client-Side Performance:** Encrypting large files (like videos) entirely in the web browser could consume a lot of the user's computer memory (RAM), potentially causing the website to freeze or crash, leading to a poor user experience.
- **Data Integrity:** While the encrypted files are secure, issues with our database (MongoDB) could lead to corruption of user metadata (like which key was used for which file), making it impossible to retrieve the correct information for decryption.

➤ **People Risks:**

- **Skill Gap:** Our team's practical experience with implementing advanced cryptography was limited. This meant we had to be extremely careful to use trusted code libraries and not make custom modifications we couldn't fully vet.
- **Communication Gaps:** A misunderstanding between team members about how a security feature should work could lead to inconsistencies in the code, creating unintended vulnerabilities.

➤ **Tools Risks:**

- **Dependency Vulnerabilities:** Our project relied on many external npm packages (like crypto-js or bcrypt). If any of these pre-written code libraries had a hidden security flaw, it could directly compromise our entire application.

➤ **General Risks:**

- **Shifting Requirements:** Mid-project requests to add new features or change security protocols could disrupt our timeline and introduce new, unanticipated bugs.
- **Inadequate Testing:** If our testing wasn't thorough enough, a subtle bug related to how encryption keys are handled could slip through, potentially leading to data being leaked or becoming permanently inaccessible.

2.3.2 Risk Analysis :

“Risk analysis = risk assessment + risk management + risk communication.” Risk analysis is employed in its broadest sense to include:

Risk	Impact	Mitigation
Encryption Implementation Errors	High	Use tested Web Crypto API libraries
Data Leakage	High	Strict client-side only processing
Key Management Issues	Medium	Clear backup instructions & passphrase hints
Browser Compatibility	Medium	Multi-browser testing & fallbacks
Large File Performance	Low	File chunking & progress indicators

Table 2.3.2 Risk Analysis

CHAPTER – 3

SYSTEM REQUIREMENTS STUDY

3.1 Hardware and Software Requirement

This shows minimum requirements to carry on to run this system efficiently.

3.1.1 Hardware Requirements :

Server-side Hardware Requirement :

Devices	Description
Processor	Intel Core i3 or equivalent
RAM	4 GB or More
Hard Disk	50 GB or More (SSD recommended)

Table 3.1.1 Server-side Hardware Requirement

3.1.2 Software Requirements :

For Which	Software
Operating System	Windows 10/11, Linux
Runtime	Node.js
Database	MongoDB
Development Tool	Visual Studio Code

Table 3.1.1 Software Requirements

3.1.3 Client-side Requirements :

For Which	Requirement*
Browser	Chrome 80+, Firefox 75+, Safari 13+
JavaScript	Enabled

Table 3.1.3 Client-side Requirement

3.2 Constraints

3.2.1 Hardware Limitations:

Think of this like trying to edit a massive video on an old phone—it will struggle. Similarly, our app's ability to encrypt a huge video file depends entirely on the user's own computer. If their device isn't very powerful, the process will be slow or might even cause the web browser to freeze. This is a natural trade-off for keeping their file private, as all the hard work is done on their machine instead of ours.

3.2.2 Reliability Requirements:

This is about being a dependable service. The website needs to be up and running whenever a user needs it. More importantly, we must be incredibly careful with the data we *do* store (like user account info). We have to have backup systems in place. Losing this data because of a server crash would be like a bank losing its record of who has a safety deposit box—even if the box is safe, no one can access it.

3.2.3 Safety and Security Consideration:

This isn't just a feature; it's the entire foundation of the project. We have to build it with a "security-first" mindset from the ground up. This means:

- **Passwords:** We don't store actual passwords. Instead, we store a scrambled, irreversible version (like a unique fingerprint for the password) so even we can't see it.
- **Communication:** Every bit of data sent between the user and our server must be sent through a secure, encrypted tunnel (HTTPS) to prevent eavesdropping.
- **The Golden Rule (Zero-Knowledge):** We designed the system so that we, the developers, are technically unable to see the user's files or their encryption keys. It's like designing a vault where we don't have a master key; only the user does. This is the core promise of the project.

CHAPTER – 4

SYSTEM ANALYSIS

4.1 Study Current System

Today, people who want to protect their files mainly have two options:

1. **Online Encryption Websites:** These are easy to find but require you to upload your private files to someone else's computer (a server). You have to hope they are honest and will delete your data after encrypting it.
2. **Software Installed on Your Computer (e.g., GPG):** These are very secure but are often complicated. They usually require typing commands, which can be confusing if you're not a tech expert.

4.2 Problem and weakness of current system

- **The Trust Issue:** It's risky to send your personal files to a company online. You can't be sure what they do with your data once you upload it.
- **The Complexity Issue:** The most secure tools are often difficult for regular people to use. If something is too complicated, most people won't use it, leaving their files unprotected.
- **The Lack of Choice:** Some tools only offer one kind of encryption. Different files might need different levels of security, and users should have options

4.3 Requirements of New System

4.3.1 User Requirements :

Users need a tool that is:

- **Simple:** As straightforward to use as any popular website.
- **Secure:** They should feel confident that their files are truly private.
- **Under Their Control:** They want to select how their data is protected and know that they are the only ones with the key.

4.3.2 System Requirements :

➤ Functional System Requirement :

This system must support the following core functions:

- User registration and secure login.

- File upload and download functionality.
- Selection of encryption algorithms (AES-256, DES, RSA).
- Client-side encryption and decryption processes.
- Optional passphrase-based encryption for additional security.

➤ **Non-Functional System Requirements :**

- **Efficiency Requirement:** The system must process files of common sizes (e.g., documents, images) quickly without overloading the user's browser or device.
- **Reliability Requirement:** The system must perform consistently and accurately. File encryption and decryption must work error-free every time.
- **Usability Requirement:** The interface must be clean, intuitive, and easy to navigate, ensuring a smooth experience even for non-technical users.
- **Implementation Requirement:** The system is built using HTML, CSS, and JavaScript for the frontend, with Node.js and Express.js for the backend server. MongoDB is used for database management to store user accounts and file metadata securely.
- **Delivery Requirement:** The project is scheduled for completion within a 12-week timeframe, with regular progress evaluations and guidance from the project supervisor.

4.4 Feasibility Study

A feasibility study was conducted to determine whether the proposed system could be successfully developed and deployed. The study focused on technical, operational, and economic aspects, confirming that the project is achievable with available technologies and resources.

➤ **Technical Feasibility :**

The proposed system relies on well-established and widely supported technologies:

- JavaScript and modern web standards enable client-side encryption.
- Node.js and Express.js provide a robust and scalable backend framework.
- MongoDB offers flexibility and efficiency for storing user and file data.
- Libraries such as CryptoJS and the Web Crypto API support reliable encryption implementation.
- The system can operate effectively on standard hardware, and encryption processes are designed to run efficiently in modern web browsers.

4.5 Selection of Hardware and Software and Justification

➤ **Hardware Configuration:**

- **Processor:** Intel Core i3 or higher (for smooth server operation and testing).
- **Memory:** 4 GB RAM or higher (to handle server processes and database operations efficiently).
- **Storage:** 50 GB or higher (to accommodate the database, application files, and backups).

➤ **Software Configuration:**

- **Operating System:** Windows 10/11 or Linux (for stability and development compatibility).
- **Development Tools:**
 - Frontend: HTML, CSS, JavaScript.
 - Backend: Node.js, Express.js.
 - Database: MongoDB.
 - Development Environment: Visual Studio Code.
- **Documentation Tool:** Microsoft Word or Google Docs for report drafting and presentation.

Justification:


This hardware and software stack was chosen for its reliability, scalability, and compatibility with modern web development practices. Node.js and JavaScript allow seamless full-stack development, while MongoDB provides a flexible and efficient solution for handling user data and metadata. The selected tools are widely used in the industry, ensuring ample resources and community support for development and troubleshooting.

CHAPTER – 5

System Design

5.1 System Design Screenshot


➤ **Simple Design of Sign-Up Page :**




Create Your Account

Join us today and start your journey


Username

 Choose a username


Email Address

 Enter your email

Password

 Create a strong password

Confirm Password

 Confirm your password

Create Account

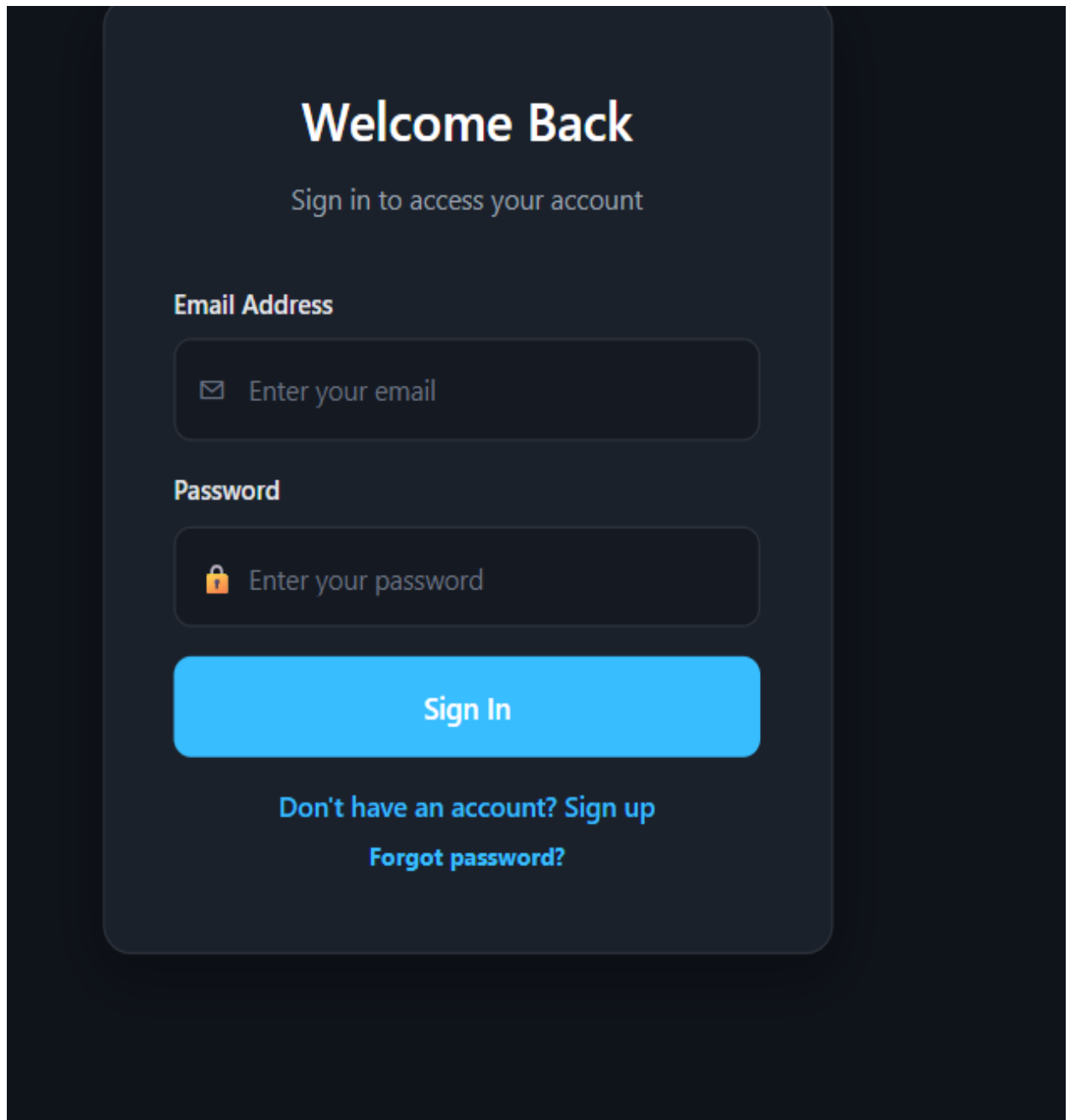
Already have an account?

[Sign in instead](#)

By creating an account, you agree to our [Terms of Service](#) and [Privacy Policy](#)

5.1.1 Personal Information Screen

Simple Design of Login Page :



Welcome Back

Sign in to access your account

Email Address

✉ Enter your email

Password

🔒 Enter your password

Sign In

[Don't have an account? Sign up](#)

[Forgot password?](#)

Figure-5.1.2 Sign-up Page

Home Page :

File Selection

Screenshot 2025-07-24 213549.png.enc
41.69 KB • application/octet-stream

Encryption Algorithm

DES Legacy Standard

DES
Data Encryption Standard - Legacy symmetric encryption algorithm

Decryption Keys

Passphrase (optional for AES/DES)
Enter passphrase if used during encryption

Security Actions

Encrypt File Decrypt File

Download Result

Ready to process with DES

Encryption Key

Security Notice
Store this key securely. It's required for decryption and cannot be recovered if lost.

Copy Key Download

Original File Content

Screenshot 2025-07-24 213549.png.enc • application/octet-stream • 42 KB

Encryption & Decryption Process - DES

DES Legacy Standard

Original Content (DES Process)

Encrypted Content (DES)

----- DES ENCRYPTED IMAGE -----
7502bf2b9cf6dacc16dd3827bb622e59cfbc8408e0213b3446a74d06e56115
461b772b12c4641241c115a9204708d3c8bbb5944baF37c50e2ub436074d641
d1f86ead0b0d487110d9c659f46eae52f34d6eb04e07f7fa3990838ccaaFaF
5f6a2b0a29a0c8e72b5333aa7b040fd43b1cbf4743a6ecd51c8b7b8e54771b
3337e1b04282f8f7ffF875a99f3509aaa7c7f233f18fa4e6be2a7c7aaa0b046
9b31ac6230150au12D04a25d5eb2898464dFa4Fa1d3c98ca4dcbF5f695a1ddd

Copy Encrypted

Decrypted Content (DES)

----- DES DECRYPTED IMAGE -----
Image successfully restored: Screenshot 2025-07-24 213549.png.enc
Size: 42688 bytes
Type: application/octet-stream
----- END DECRYPTED CONTENT -----

Copy Decrypted

Figure 5.1.3 Main Page

21

5.2 Interface Design

5.2.1 Class Diagram :

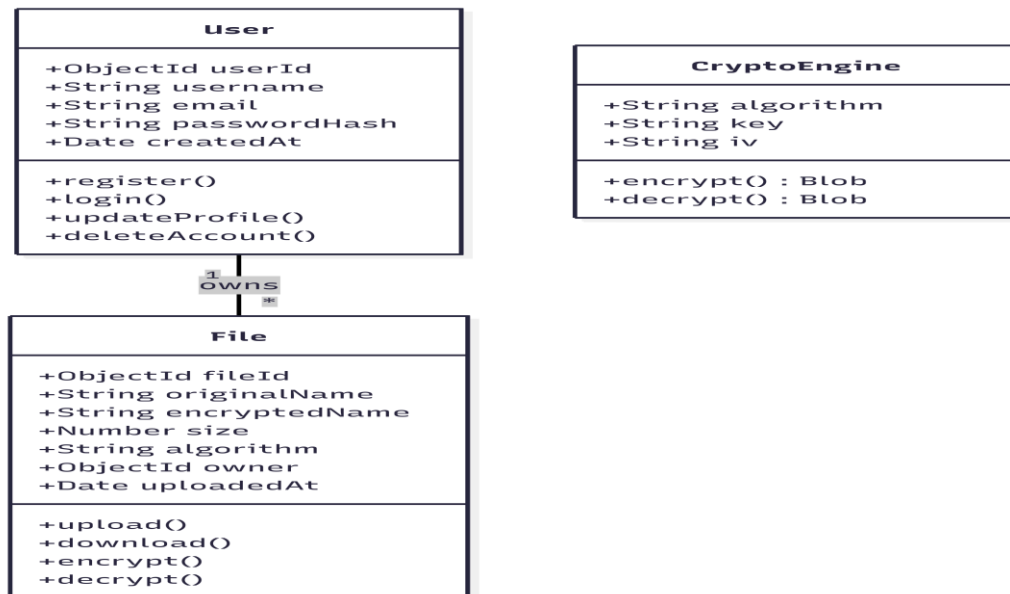


Figure 5.2.1 Class Diagram

5.2.2 Use Case Diagram

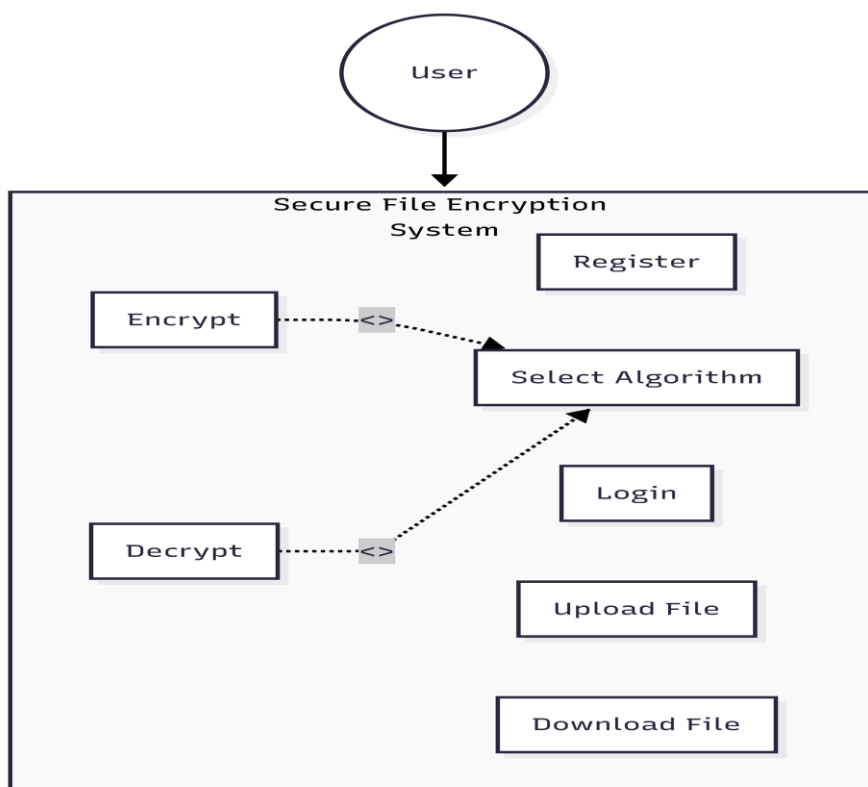


Figure 5.2.2 Use-case diagram

5.2.3 Activity Diagram :

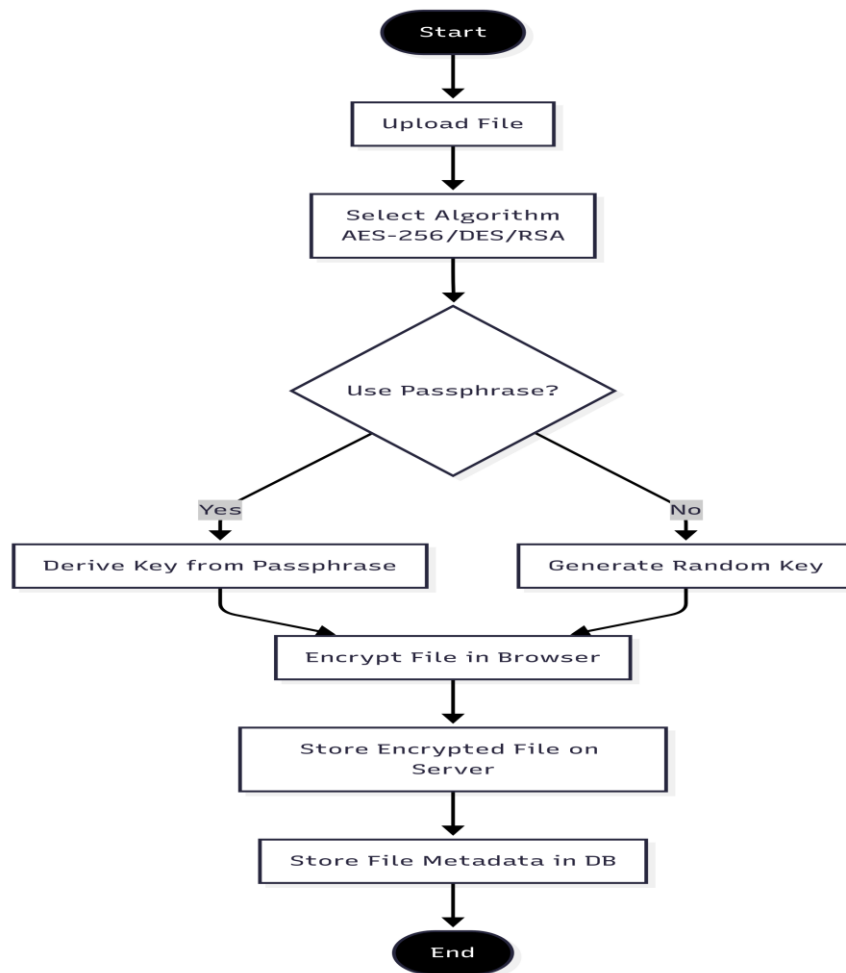


Figure 5.2.3 Activity Diagram

5.2.4 Data Flow Diagram :

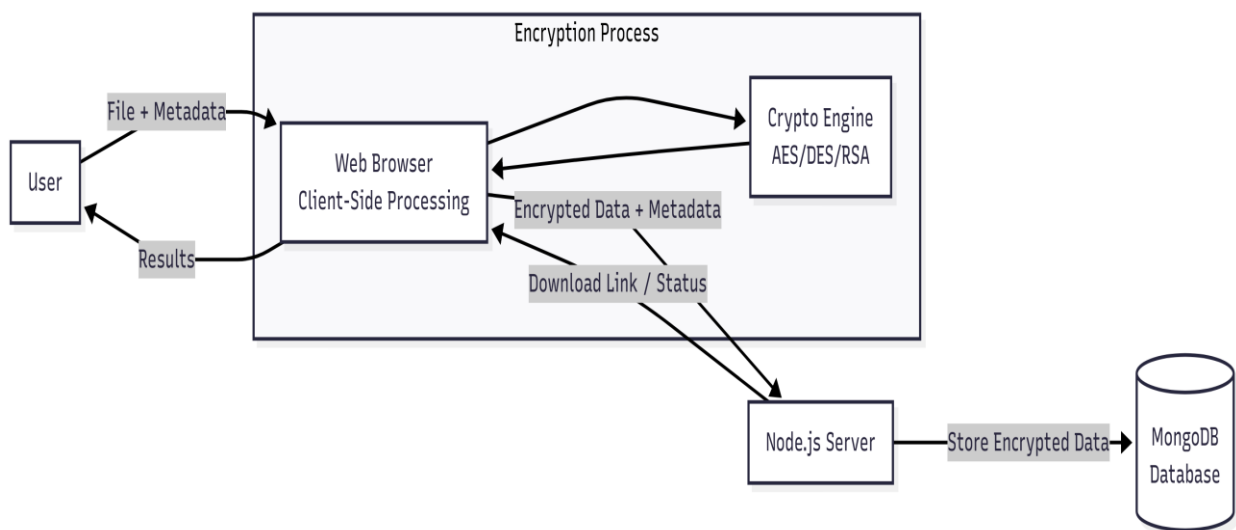


Figure 5.2.4 Data Flow Diagram

5.2.5 State Diagram :

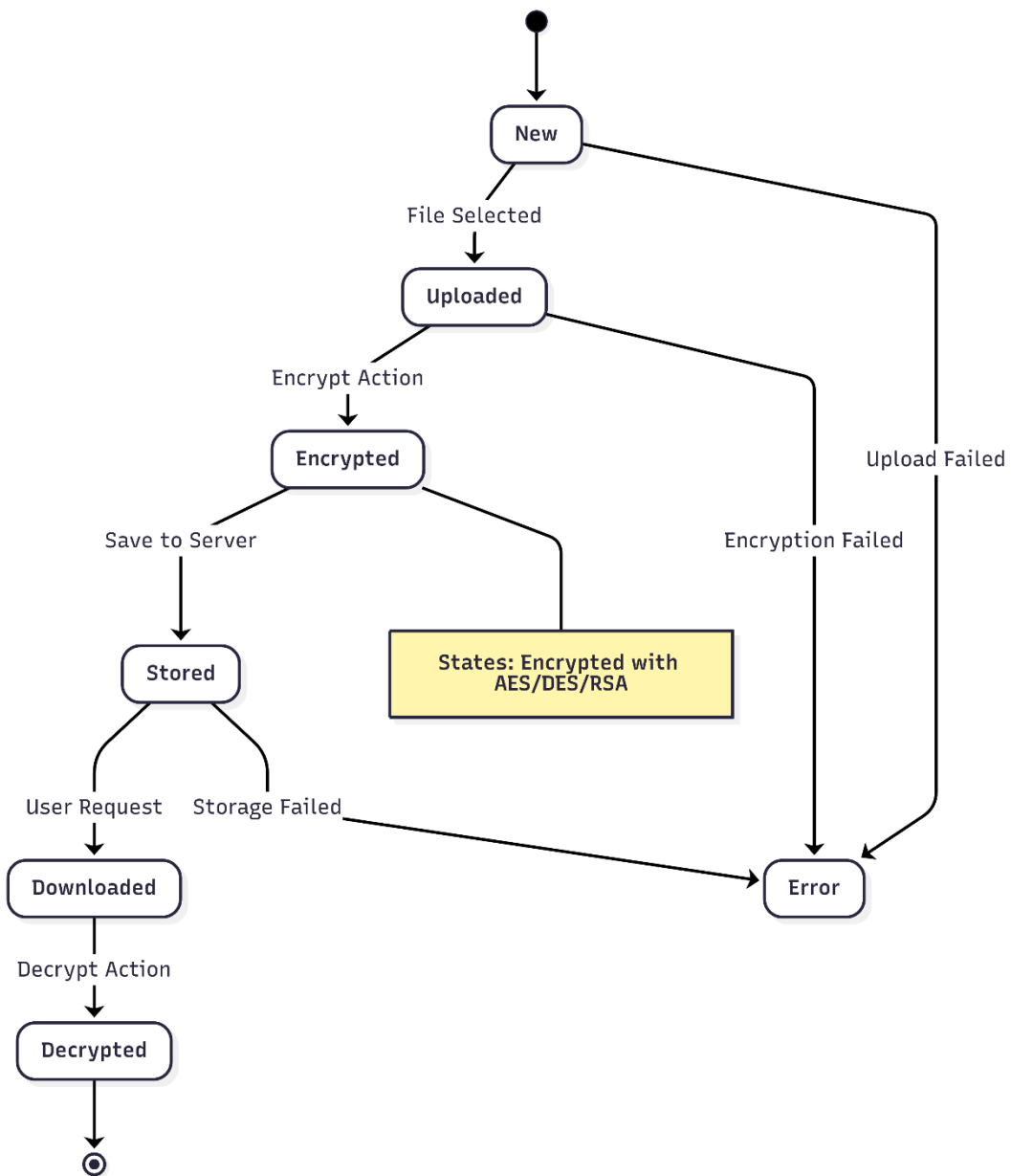


Figure 5.2.5 State Diagram

5.2.6 E-R Diagram :

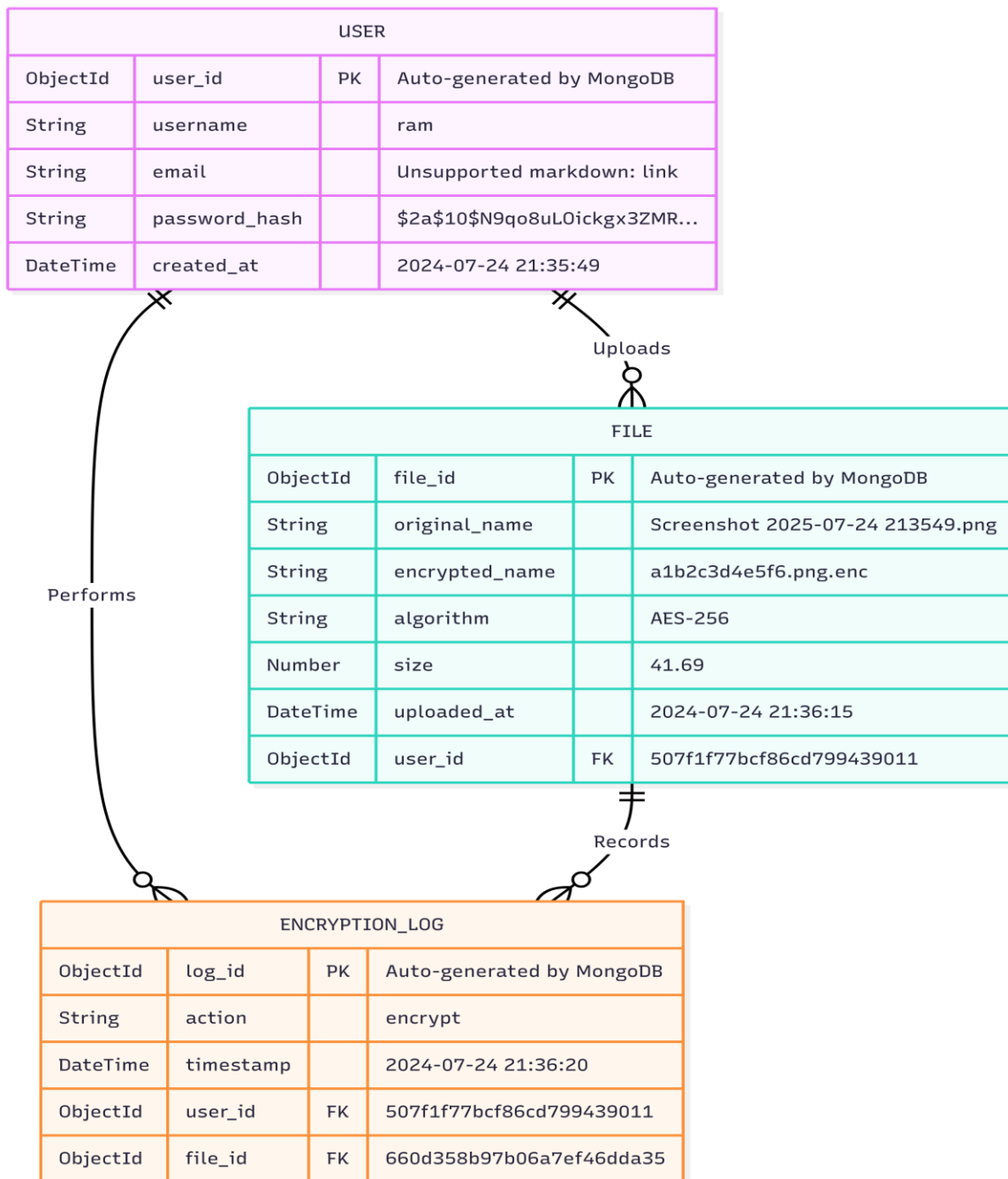


Figure 5.2.6 E-R Diagram

5.2.7 Sequence Diagram :

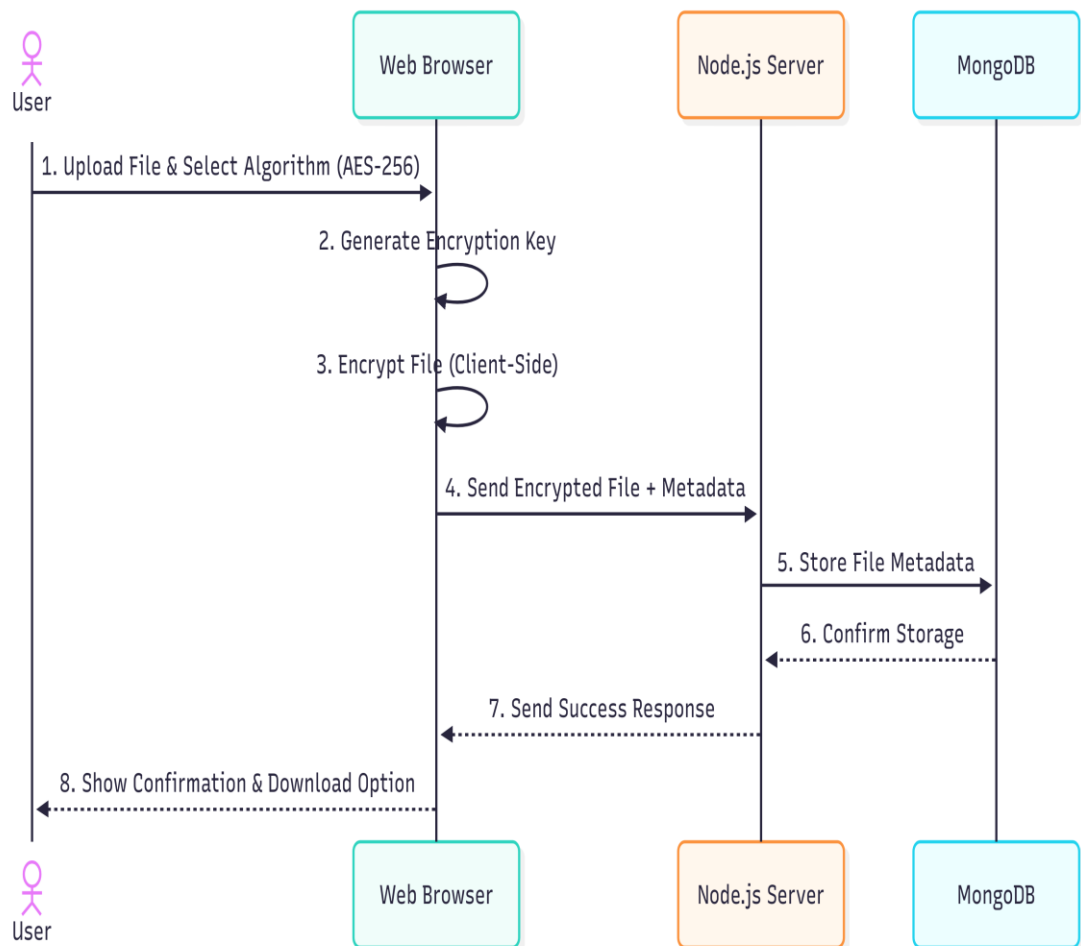


Figure 5.2.7 Sequence Diagram

CHAPTER – 6

Code Implementation

6.1 Implementation Environment

The project was implemented in a Windows/Linux development environment using Visual Studio Code. Version control was managed with Git. The team consisted of two members collaborating on front-end and back-end components.

6.2 Program/Module Specification

The system is modular:

- **Auth Module:** Handles user sign-up, login, and session management using JWT.
- **File Handler Module:** Manages file uploads to a temporary location before client-side processing.
- **Crypto Module:** The core module containing functions for AES, DES, and RSA operations, implemented to run in the browser.
- **DB Module:** Hand all interactions with the MongoDB database.

6.3 Coding Standards

Normally, good software development organization requires their programmers to maintain some well-defined and standard style of coding called coding standard.

6.3.1 Comment Standards :

- The project followed consistent commenting practices to ensure code readability and maintainability. JSDoc-style comments were used for all major functions to document parameters, return values, and purpose. Inline comments (//) were added to explain complex logic, especially around cryptographic operations and security measures. This approach helped in maintaining clear documentation of the zero-knowledge architecture and client-side processing implementation.
- This concise description is perfect for your project report - it covers the essential points without unnecessary technical depth.

CHAPTER – 7

Testing

7.1 Testing Strategy

A multi-level testing strategy was employed, starting with unit tests for individual functions and modules, followed by integration testing to ensure modules work together correctly, and ending with user acceptance testing to validate functionality against requirements.

7.2 Testing Method

7.2.1 Unit Testing :

- We tested each part of the system separately to make sure every small piece worked correctly on its own. For example:
- We tested the password hashing function to ensure it securely converts passwords into encrypted formats
- We verified that each encryption algorithm (AES, DES, RSA) works properly with different types of files
- We checked that database operations like saving user information work correctly

7.2.2 Validation Testing :

- After unit testing, we tested how different parts work together:
- We verified that when a user logs in, they can successfully access their file encryption dashboard
- We tested that the user authentication system properly protects file routes - meaning only logged-in users can encrypt/decrypt files
- We checked that file uploads correctly connect with the encryption system

7.2.3 Integration Testing :

- This was the final testing phase where we confirmed the system meets real-world requirements:
- We encrypted a file using AES-256 with a specific password, then verified it could only be decrypted using the exact same password
- We tested that files encrypted with one algorithm (like RSA) cannot be decrypted with another algorithm (like DES)

- We verified the system works as expected from a user's perspective - the entire process from registration to file encryption works smoothly

7.3 Test Cases

Test Case ID	Module	Input	Expected Output	Result
TC-01	User Registration	Valid user details	Account created successfully	Pass
TC-02	User Login	Correct credentials	Access to dashboard	Pass
TC-03	File Upload	Select document file	File accepted for processing	Pass
TC-04	AES Encryption	File + AES-256 algorithm	File encrypted successfully	Pass
TC-05	DES Encryption	File + DES algorithm	File encrypted successfully	Pass
TC-06	RSA Encryption	File + RSA algorithm	File encrypted successfully	Pass
TC-07	File Decryption	Correct key/passphrase	Original file restored	Pass
TC-08	File Decryption	Wrong passphrase	Decryption failed error	Pass
TC-09	Download	Encrypted file	File saved to device	Pass

Table- 7.3.1 Test Cases & Expected Results

CHAPTER – 8

Limitations and Future Enhancement

8.1 Limitations

- **Client Hardware Dependency:** Performance is limited by the user's device capabilities.
- **Browser Support:** Relies on modern browser features for the Crypto API.
- **File Size:** Very large files may cause performance issues or browser timeouts.
- **Key Recovery:** There is no key recovery mechanism if a user loses their passphrase, which is a necessary feature of a zero-knowledge system.

8.2 Future Enhancement

- **Cloud Storage Integration:** Allow users to save encrypted files directly to cloud providers (Google Drive, Dropbox).
- **File Sharing:** Implement secure file sharing between users using asymmetric encryption (RS
- **Advanced Key Management:** Implement a Shamir's Secret Sharing scheme for key recovery.
- **Mobile Application:** Develop a dedicated mobile app for easier access.
- **Audit Logging:** Provide users with a log of all access and operations performed on their files.

CHAPTER – 9

Conclusion

The "Secure File Encryption & Decryption Portal" project successfully delivers a functional and secure web application for protecting sensitive files. The core objectives of implementing a zero-knowledge architecture, providing multiple encryption algorithms, and creating an intuitive user interface have been achieved. The system ensures that users remain in full control of their data and keys at all times. While there are limitations related to client-side processing, the project provides a strong foundation for a secure file management system. The technologies used, namely Node.js, MongoDB, and client-side JavaScript, proved to be highly effective for building this type of application. This project demonstrates a practical and secure approach to personal data encryption in the modern web environment

CHAPTER – 10

Reference

- Node.js Documentation: <https://nodejs.org/en/docs/>
- MongoDB Documentation: <https://www.mongodb.com/docs/>
- Express.js Framework: <https://expressjs.com/>
- Web Crypto API: https://developer.mozilla.org/en-US/docs/Web/API/Web_Crypto_API
- JavaScript Documentation: <https://developer.mozilla.org/en-US/docs/Web/JavaScript>
- HTML & CSS Standards: <https://www.w3.org/standards/webdesign/>
- Bcrypt Password Hashing: <https://www.npmjs.com/package/bcrypt>
- JWT Authentication: <https://jwt.io/>
- CryptoJS Library: <https://www.npmjs.com/package/crypto-js>
- Mongoose ODM: <https://mongoosejs.com/>