PROJECT CATEGORY

This project falls under the category of a **Web Application** with a specialization in **Cryptographic Security Systems.**

The project titled **Encodex** is built using the **MERN + Flask stack**, integrating modern web development with advanced encryption techniques. It uses **React**.js and **Tailwind-CSS** for the frontend, **Flask (Python)** for the backend server, and **Cryptographic Algorithms** (AES + RSA) to implement secure image encryption and decryption.

The project is designed as a single-page application (SPA) and is focused on solving real-world challenges related to data confidentiality, secure transmission, and digital privacy. It incorporates file handling, access control via JWT authentication, and encryption key management for end-to-end security.

As such, Encodex fits within the domain of Web Security Applications, combining Frontend Technologies, Backend APIs, and Cryptographic Systems into a cohesive full-stack solution.

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to our project guide, **Prof. Achal Tomar,** for his invaluable guidance, support, and encouragement throughout the duration of this project. His insights and expertise were pivotal in the successful completion of this work.

We would also like to extend our thanks to the **Department of Computer Science and Engineering** at **Anand Engineering College, Agra**, for providing us with the resources and environment needed for this project.

Our heartfelt thanks to our families and friends for their unwavering support and encouragement throughout our B.Tech journey.

Finally, we acknowledge the collaborative efforts and contributions of our team members, without whom this project would not have been possible.

Team Members:

Sachin Kumar Gola (2100010100045)

Vinay Chhabra (2100010100058)

Money Singh (2100010100036)

B.Tech CSE, IV Year

ABSTRACT

Encodex is a secure web application designed for encrypting and decrypting image files using the AES encryption algorithm. It ensures the confidentiality and integrity of image data by securely encrypting it, then encoding the data in Base64 format for safe storage and transmission. The encrypted image data and its associated key can be safely transmitted between parties. The system also supports a hybrid encryption model where the AES key itself is encrypted using RSA for enhanced security during transmission.

The frontend is developed using React.js and Tailwind CSS, while the backend utilizes Flask in Python. Secure file handling, key generation, and decryption are all handled using standard cryptographic libraries. The aim of this project is to provide a practical, educational, and secure approach to understanding modern-day image encryption techniques.

Keywords: AES, RSA, Encryption, Flask, React.js, Base64, Image Security, for giving me the opportunity to work on this topic. Their continuous support, valuable guidance, and constructive feedback helped me to complete this project successfully. We also thank the Head of Department and all faculty members of the Department of Computer Science and Engineering for their encouragement throughout this work.

Lastly, We acknowledge the moral support of my family and friends during the development of this project.

TABLE OF CONTENTS

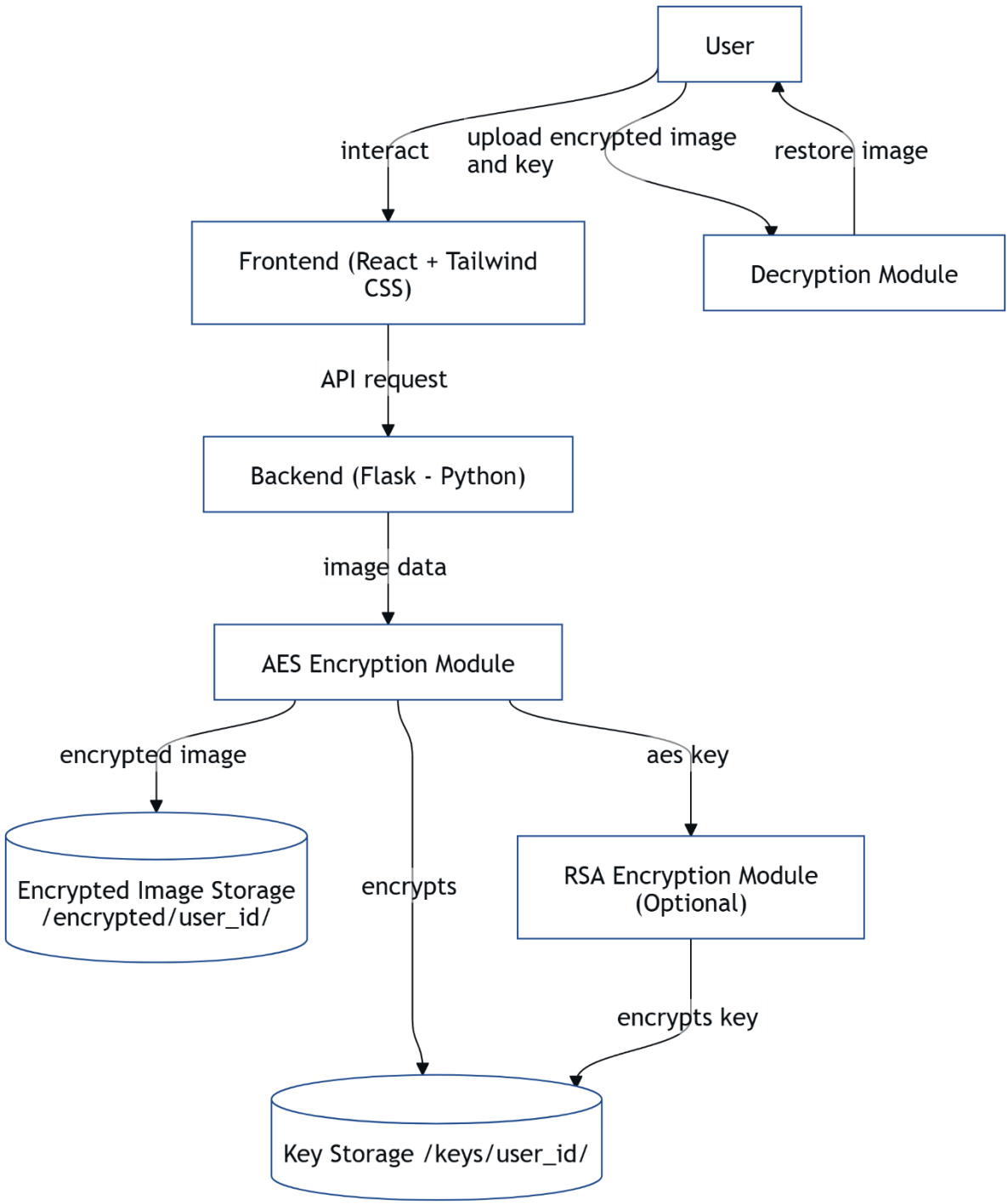
CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The EncodeX Image Encryption Project is designed to secure image data using advanced cryptographic techniques to protect sensitive visual information during storage and transmission. By leveraging the Advanced Encryption Standard (AES), the system offers a reliable and efficient method to prevent unauthorized access to images.

Built using Python with Flask for backend operations and React for the front-end interface, the application provides a secure, responsive, and user-friendly experience for image encryption and decryption.


User
interact
upload encrypted image and key
restore image
Frontend (React + Tailwind CSS)
Decryption Module
API request
Backend (Flask - Python)
image data
AES Encryption Module
encrypted image
aes key
Encrypted Image Storage /encrypted/user_id/
encrypts
RSA Encryption Module (Optional)
encrypts key
Key Storage /keys/user_id/

1.2 MOTIVATION

With the increasing use of digital platforms for sharing and storing images, the risk of unauthorized access to sensitive visual data is growing rapidly. There is a critical need for systems that can protect image data in personal, corporate, and institutional contexts. This project addresses that need by providing a practical tool for secure image encryption.

1.3 GOAL AND OBJECTIVES

Goal:

To design and develop a secure image encryption system that utilizes AES encryption and provides real-time web-based access for users to encrypt and decrypt image files.

Objectives:

- To implement AES encryption for images using Python's cryptography library.
- To develop a user-friendly interface using Flask and React.js.
- To ensure secure image handling with no data retention.
- To offer controlled decryption via unique encryption keys.

1.4 SCOPE

EncodeX is applicable in several domains including:

- Healthcare: Secure transmission of patient scans or reports.
- Legal: Protection of sensitive legal documents in image form.
- Corporate: Secure sharing of confidential business images.

- Education: Demonstrating cryptographic principles in coursework.

1.5 GENERAL OVERVIEW

EncodeX allows users to upload an image, encrypt it using AES and a unique key, and later decrypt it using the same key. The system ensures the confidentiality and integrity of image data and demonstrates how cryptographic security can be applied in web applications.

Industry	**Use Case Example**	**Benefit of EncodeX**
Healthcare	Encrypting patient reports, scans (X-ray, MRI)	Ensures privacy, HIPAA compliance
Legal Sector	Secure sharing of signed documents/images	Prevents tampering and unauthorized access
Corporate	Protecting visual presentations or prototypes	Safeguards intellectual property
Education	Demonstrating cryptography to students	Enhances hands-on learning in security
Government	Encrypting confidential IDs or reports	Maintains national security and confidentiality

CHAPTER 2

MODULES

2.1 User Authentication

Users need to sign up and log in to securely access the encryption and decryption functionalities. This is implemented using JWT (JSON Web Tokens).

2.2 Image Upload

Allows the user to upload an image file that will be encrypted using AES encryption.

2.3 Aes-Rsa Encryption Module

Encrypts the uploaded image data using the AES algorithm. The AES key is generated dynamically and securely saved. To enhance security, the AES key can be encrypted using the RSA algorithm, allowing safe key sharing over public channels.

2.5 Download Module

Provides the encrypted image as a downloadable Base64-encoded file.

2.6 Image Decryption

Accepts the AES-encrypted image and key from the user to decrypt and retrieve the original image.

2.7 History Management

Users can view lists of uploaded, encrypted, and decrypted images, sorted by session, stored in user-specific directories

CHAPTER 3

METHODOLOGY

The development of Encodex followed an agile, iterative approach broken down into weekly sprints. Each week focused on a specific component of the system, from learning fundamentals to integrating encryption logic.

Week 1: Learning Phase

- Understood basic concepts of Python and Flask
- Learned about symmetric and asymmetric encryption
- Set up the local development environment and installed required tools

Week 2: Frontend & Backend Basics

- Built the basic UI using React.js and Tailwind CSS
- Created backend server with Flask
- Developed routes for image upload and handling using Flask

Week 3: Aes Encryption Module

- Implemented AES algorithm using Python's cryptography libraries
- Encoded encrypted image using Base64
- Saved encryption key separately in a secure directory

Week 4: Rsa Hybrid Integration

- Added optional RSA encryption to secure the AES key
- Generated RSA key pairs for simulation
- Encrypted and decrypted the AES key using RSA

Week 5: User Interface + Routing

- Integrated frontend and backend using Axios
- Created tabs for Uploads, Encrypted Images, and Decrypted Images
- Verified secure transmission of keys and image data

Week 6: Testing And Deployment

- Tested complete encryption and decryption pipeline
- Validated Base64 encoded output and key handling
- Deployed locally and documented usage flow

CHAPTER 4

LEARNING OBJECTIVES

The project helped build both short-term and long-term technical competencies related to encryption, web development, and system design.

4.1 SHORT-TERM OBJECTIVES

Objective 1: Learn AES Encryption

- Gained practical experience using the Advanced Encryption Standard (AES)
- Understood modes of operation, padding, key size, and security relevance

Objective 2: Build Full Stack Integration

- Developed a frontend using React and backend using Flask
- Connected the two using secure APIs and file handling techniques

Objective 3: Apply Base64 and File Handling

- Learned to encode encrypted binary data using Base64 for web transmission
- Implemented logic to handle large image files without breaking encryption

4.2 LONG-TERM OBJECTIVES

Objective 1: Understand Hybrid Encryption

- Implemented AES + RSA hybrid to simulate real-world cryptographic practices
- Explored secure key exchange protocols

Objective 2: System Security Awareness

- Developed understanding of secure folder management and key storage
- Recognized common threats in file transfer and user data handling

Objective 3: Professional Growth

- Gained confidence in full-stack development and secure communication design
- Prepared for advanced roles involving cybersecurity or backend engineering

CHAPTER 5

HARDWARE AND SOFTWARE REQUIREMENTS

5.1 Developer System Requirements

- Intel Core i3 or i5 processor or higher
- Minimum 8 GB RAM
- Windows 10 / Linux operating system
- Web browser (Google Chrome / Firefox)
- Internet connection for installing project dependencies
- Visual Studio Code (VS Code) for development

5.2 Software Requirements

- React.js for frontend user interface
- Tailwind CSS for responsive UI styling
- Flask (Python framework) for backend server
- Python 3.x installed in system
- Required Python libraries: cryptography, base64, pycrypto
- JWT for user authentication
- Axios for frontend-backend API calls
- Postman for testing APIs

5.3 Optional Tools

- Git for version control and team collaboration
- Figma for UI/UX wireframes and prototyping
- RSA key generation tools for optional hybrid encryption simulation
- PyCryptodome for additional encryption testing

CHAPTER 6

TECHNOLOGIES USED

6.1 FRONTEND TOOLS

- React.js: A JavaScript library for building interactive and component-based user interfaces. It provides fast updates through a virtual DOM and modular architecture.

- Tailwind CSS: A utility-first CSS framework used to create modern and responsive designs efficiently.

- Axios: A promise-based HTTP client for making API requests from the frontend to the backend.

6.2 BACKEND TECHNOLOGIES

- Flask: A lightweight and flexible Python web framework that handles backend routes, request processing, and integration with cryptographic modules.

- Python 3.x: The main programming language used for backend development, encryption, and server logic.

- JWT (JSON Web Tokens): Used for user authentication, ensuring secure access control within the application.

6.3 CRYPTOGRAPHY & SECURITY MODULES

- AES (Advanced Encryption Standard): Symmetric block cipher used for encrypting image files securely.

- RSA (Rivest–Shamir–Adleman): Asymmetric cryptographic algorithm used optionally for encrypting the AES key.

- Base64 Encoding: Encodes encrypted binary image data into a textual format for safer web transmission.

- PyCryptodome / Cryptography Module: Libraries used to implement encryption, key generation, and secure decryption.

6.4 DEVELOPMENT & TESTING TOOLS

- Visual Studio Code: Integrated development environment used for writing and debugging both frontend and backend code.

- Postman: API testing tool used for validating Flask routes and testing encrypted file uploads.

- Git: Version control system for managing project progress and collaboration.

- Figma: A design tool used to prototype and visualize the UI layout before implementation.

CHAPTER 7

FEASIBILITY STUDY

7.1 TECHNICAL FEASIBILITY

The EncodeX project is technically sound and feasible due to the use of reliable, widely adopted technologies. It utilizes Flask as the backend framework, React.js for the frontend, and the Python Cryptography library to handle AES encryption. All these technologies are open-source, platform-independent, and well-documented, ensuring smooth implementation.

7.2 ECONOMIC FEASIBILITY

The development of EncodeX is economically feasible. No proprietary software or expensive hardware is needed. All frameworks and libraries used (Flask, React, Python, Tailwind CSS) are open-source. The system can be hosted locally or on free-tier servers for testing, resulting in minimal to no cost.

7.3 OPERATIONAL FEASIBILITY

EncodeX is designed to be user-friendly, with an intuitive interface that does not require prior technical knowledge. Features like key-based decryption, real-time image processing, and simple upload/download actions ensure that users can operate the system efficiently without extensive training.

7.4 LEGAL AND ETHICAL FEASIBILITY

Since the application does not store user data and gives complete control to users over encryption keys, it aligns with data privacy best practices and regulations like GDPR. There

are no ethical conflicts as the system is intended to secure data and prevent misuse, not to access or manipulate it.

Overall, the EncodeX Image Encryption System is highly feasible from technical, economic, operational, and legal standpoints.

CHAPTER 8

SYSTEM DESIGN

8.1 SYSTEM ARCHITECTURE

The system architecture of EncodeX is designed to enable secure image encryption and decryption over a web interface using AES. It consists of three main layers:

- Presentation Layer: Built using React.js and Tailwind CSS, it allows users to upload, encrypt, and decrypt images.

- Application Layer: Developed using Flask, this layer handles encryption logic, key management, and file processing.

- Security Layer: Uses the Cryptography library to perform AES encryption and decryption on image data.

This layered architecture ensures modularity, easy debugging, and secure operations across all endpoints.

8.2 PROCESS FLOW

The process flow explains how user interactions trigger encryption and decryption operations:

1. User uploads an image through the frontend.

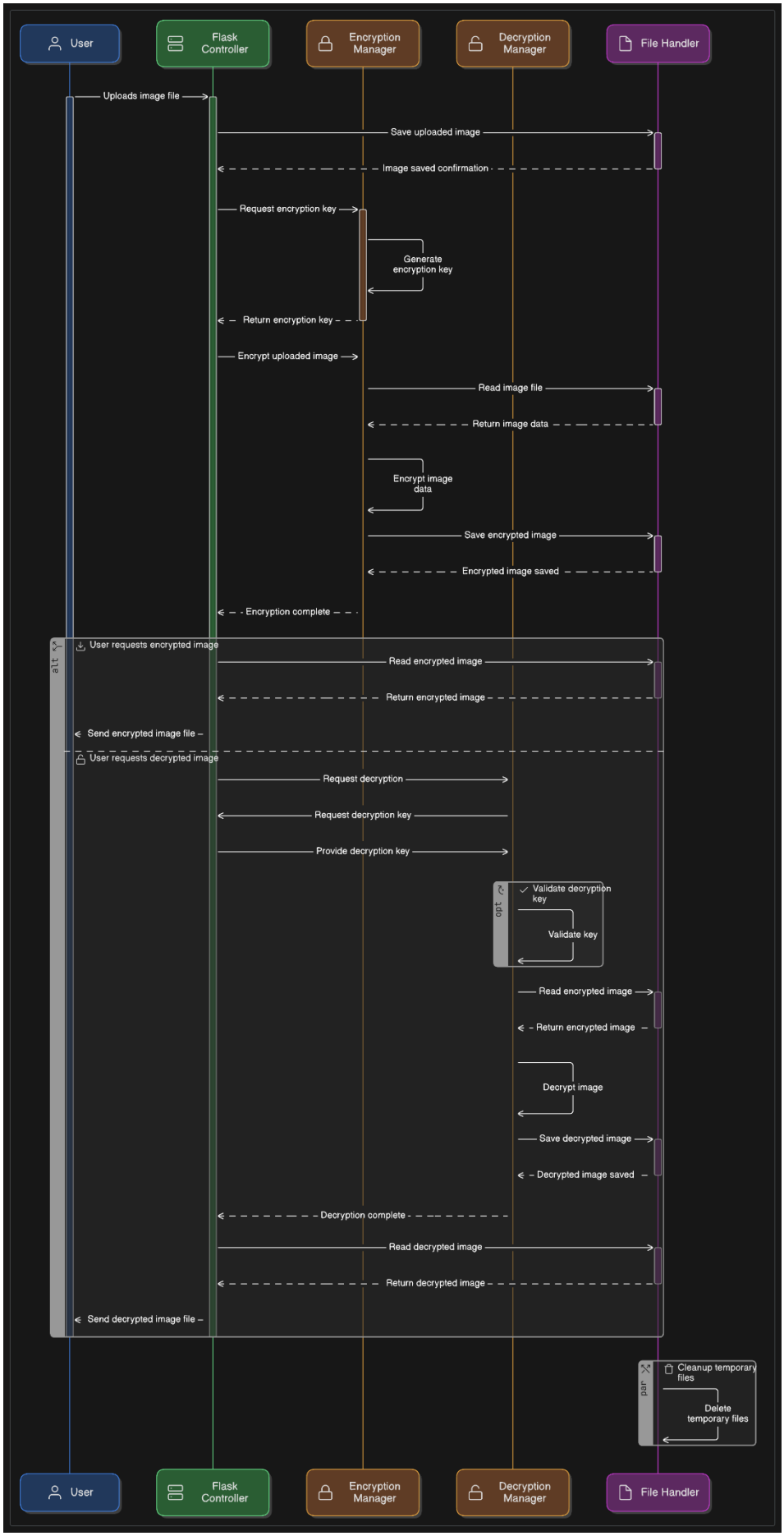
2. Image is sent to the Flask backend in Base64 format.

3. The server performs AES encryption and generates a unique key.

4. Encrypted image is returned for download and the key is displayed to the user.

5. During decryption, the same key must be provided by the user.

6. If the key is correct, the original image is restored and sent back to the user.


User
Flask Controller
Encryption Manager
Decryption Manager
File Handler
Uploads image file
Save uploaded image
Image saved confirmation
Request encryption key
Generate encryption key
Return encryption key
Encrypt uploaded image
Read image file
Return image data
Encrypt image data
Save encrypted image
Encrypted image saved
Encryption complete
alt
User requests encrypted image
Read encrypted image
Return encrypted image
Send encrypted image file
User requests decrypted image
Request decryption
Request decryption key
Provide decryption key
opt
Validate decryption key
Validate key
Read encrypted image
Return encrypted image
Decrypt image
Save decrypted image
Decrypted image saved
Decryption complete
Read decrypted image
Return decrypted image
Send decrypted image file
par
Cleanup temporary files
Delete temporary files
User
Flask Controller
Encryption Manager
Decryption Manager
File Handler

8.3 SECURITY DESIGN

Security is ensured through the following mechanisms:

- **AES Algorithm:** Industry-standard encryption with 128/256-bit keys.

- **Key-Based Access:** No encryption or decryption is possible without the correct key.

- **No Data Retention:** Encrypted images and keys are not stored after the session ends.

8.4 USER INTERFACE DESIGN

The UI is built with usability in mind. It allows:

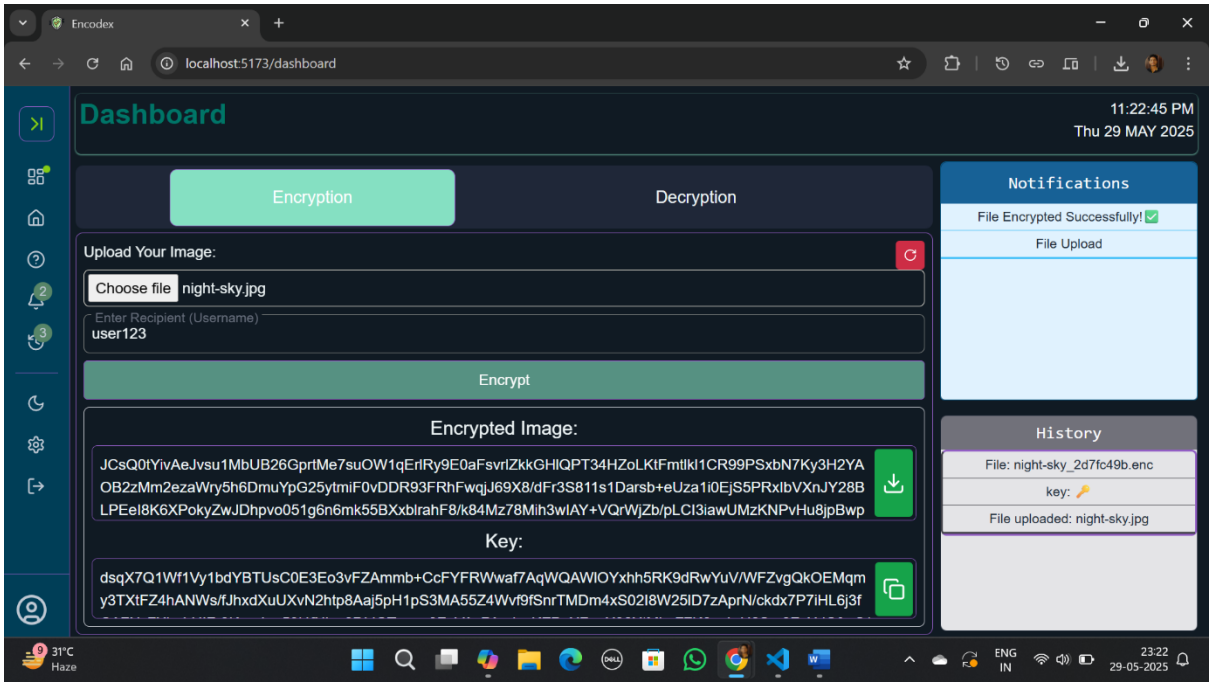
- Drag-and-drop image upload

- One-click encryption

- Display of generated key

- Decryption input field

- Real-time error messages on invalid key



The UI design ensures ease of use for both technical and non-technical users.

CHAPTER 11

DATA FLOW DIAGRAM (LEVEL 0 & LEVEL 1)

A Data Flow Diagram (DFD) illustrates the flow of data within a system, showing how input is transformed into output through various processes. Below are the Level 0 (Context-Level) and Level 1 DFDs for the EncodeX Image Encryption System.

11.1 LEVEL 0 – CONTEXT DIAGRAM

The Level 0 DFD provides a high-level overview of the entire EncodeX system, showing the interaction between the system and external entities.

External Entities:

- User

Processes:

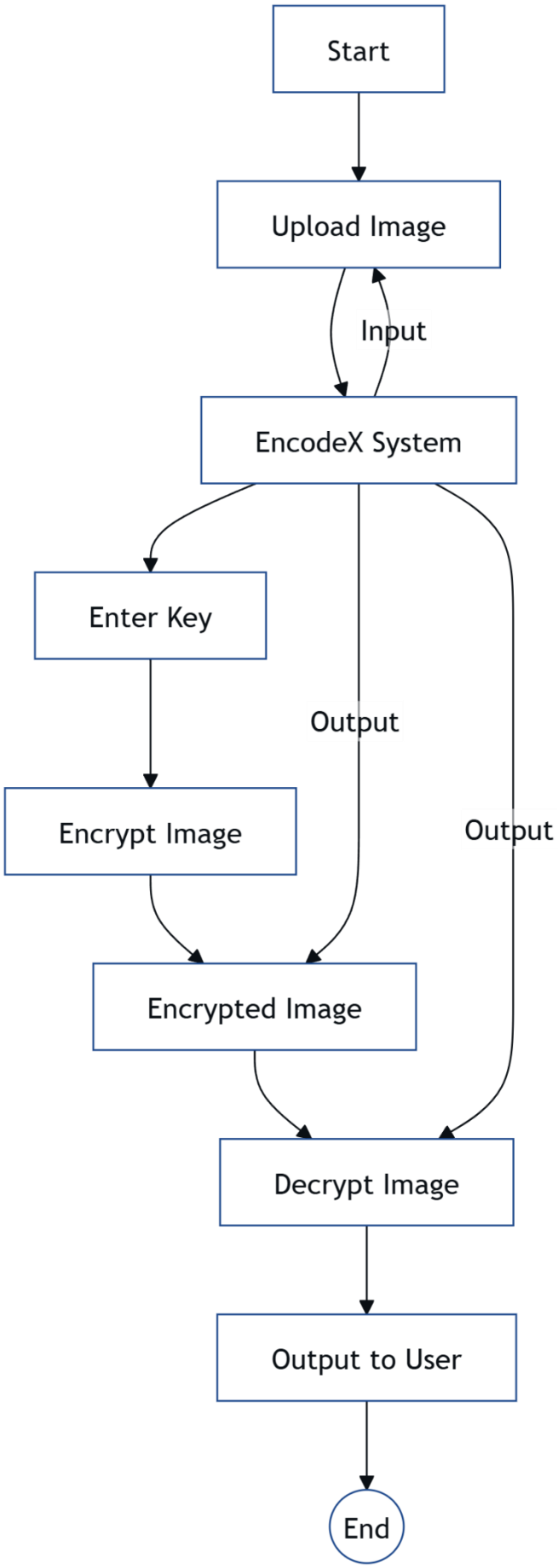
- Image Encryption & Decryption System

Data Stores:

- None (since EncodeX uses session-based temporary storage)

Data Flows:

- Image Upload
- Encrypted Image
- Key Input
- Decrypted Image


Start
Upload Image
Input
EncodeX System
Enter Key
Output
Encrypt Image
Output
Encrypted Image
Decrypt Image
Output to User
End

11.2 LEVEL 1 – DETAILED SYSTEM FLOW

The Level 1 DFD breaks down the system into detailed processes and shows internal data flows.

Processes:

1. Upload Image
2. Encrypt Image
3. Generate Key
4. Enter Key
5. Decrypt Image

Data Stores:

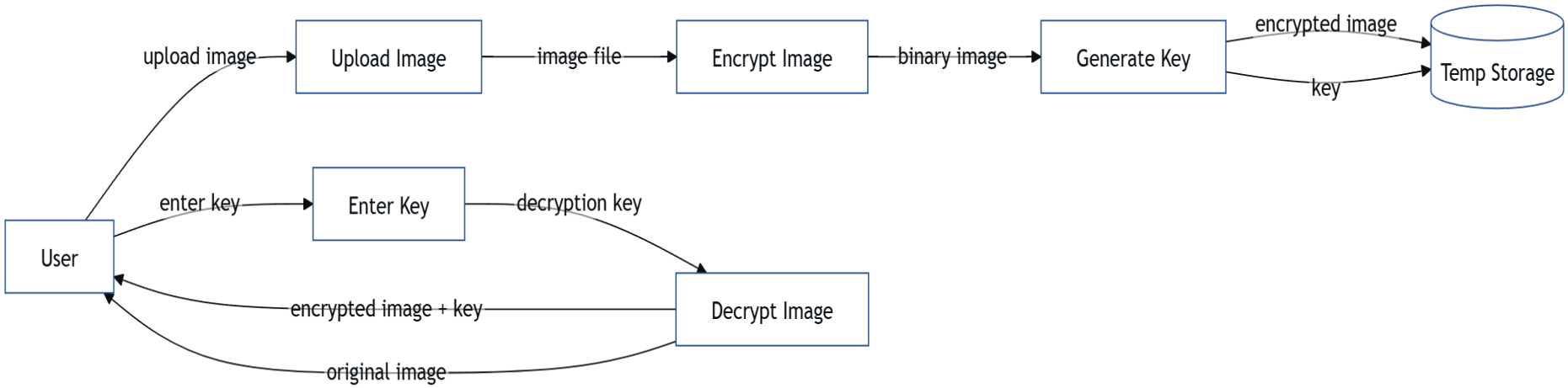
- Temporary File Storage

Data Flows:

- Raw image data
- Encrypted image output
- Generated AES key
- Encrypted file input for decryption
- Original image output

External Entity:

- User


User
upload image
Upload Image
image file
Encrypt Image
binary image
Generate Key
encrypted image
key
Temp Storage
enter key
Enter Key
decryption key
Decrypt Image
encrypted image + key
original image

CHAPTER 12

ARCHITECTURAL DIAGRAM

The architectural diagram of the EncodeX system represents the high-level structure and interactions between the different components of the application. It outlines how data moves from the user interface through the backend encryption logic and finally to the output.

The architecture is divided into the following primary layers:

1. Presentation Layer (Frontend):

- Built using React.js and styled with Tailwind CSS.
- Allows users to upload images, enter encryption keys, and download output files.
- Sends requests to the Flask backend via API endpoints.

2. Application Layer (Backend):

- Implemented using Flask (Python framework).
- Receives API requests for encryption/decryption.
- Interfaces with the encryption and file handling modules.

3. Security Layer:

- Uses Python's Cryptography library for AES encryption and decryption.
- Generates and applies a secure, random symmetric key for image data.

4. Storage Layer:

- Handles temporary storage of uploaded and encrypted files during processing.
- Files are deleted or cleared after user actions to prevent data leakage.

This modular structure ensures separation of concerns, improved maintainability, and secure handling of sensitive data.

CHAPTER 13

TESTING & TEST CASES

13.1 TESTING STRATEGY

Testing plays a crucial role in validating the functional and security aspects of the EncodeX Image Encryption system. The testing strategy includes:

- **Unit Testing:** To test encryption and decryption logic functions individually.
- **Integration Testing:** To ensure correct interaction between frontend and backend.
- **System Testing:** To validate end-to-end encryption and decryption functionality.
- **Security Testing:** To ensure unauthorized access and key mismatch scenarios are handled securely.

13.2 TEST OBJECTIVES

- Verify correct encryption of image files using AES.
- Confirm successful decryption only with the correct key.
- Ensure that invalid or missing keys produce appropriate error messages.
- Validate the download and file-handling operations.
- Confirm smooth UI operation and input validation in the frontend.

13.3 SAMPLE TEST CASES

Test Case ID	Test Description	Input	Expected Result	Actual Result	Status
TC_01	Upload valid image	PNG/JPEG image	Image uploaded successfully	As expected	Pass
TC_02	Encrypt image	Uploaded image	Encrypted file generated	As expected	Pass
TC_03	Download encrypted file	After encryption	File downloads in .enc format	As expected	Pass
TC_04	Decrypt image with correct key	Encrypted image + correct key	Original image restored	As expected	Pass
TC_05	Decrypt image with wrong key	Encrypted image + wrong key	Error message shown	As expected	Pass
TC_06	Attempt upload with unsupported format	.txt file	Error message displayed	As expected	Pass
TC_07	Encrypt without uploading image	No image	Warning or disabled button	As expected	Pass

13.4 TESTING RESULTS

The application passed all major functional and security test cases. User interactions including file upload, encryption, decryption, and file download were handled smoothly. Invalid input and edge cases such as wrong keys or unsupported formats were also tested and handled gracefully.

13.5 USER ACCEPTANCE TESTING

A small group of users tested the EncodeX system. Feedback was collected on:

- Ease of use
- Speed of encryption/decryption
- Clarity of error messages
- Design and responsiveness

The response was positive with users finding the application intuitive and reliable. Suggestions for improvements included adding file preview before encryption and storing key history (which were intentionally omitted for security).

CHAPTER 14

SCREENSHOTS

The following screenshots illustrate the key interfaces and workflows of the EncodeX Image Encryption System. These include the user experience during image upload, encryption, key display, decryption, and error handling.

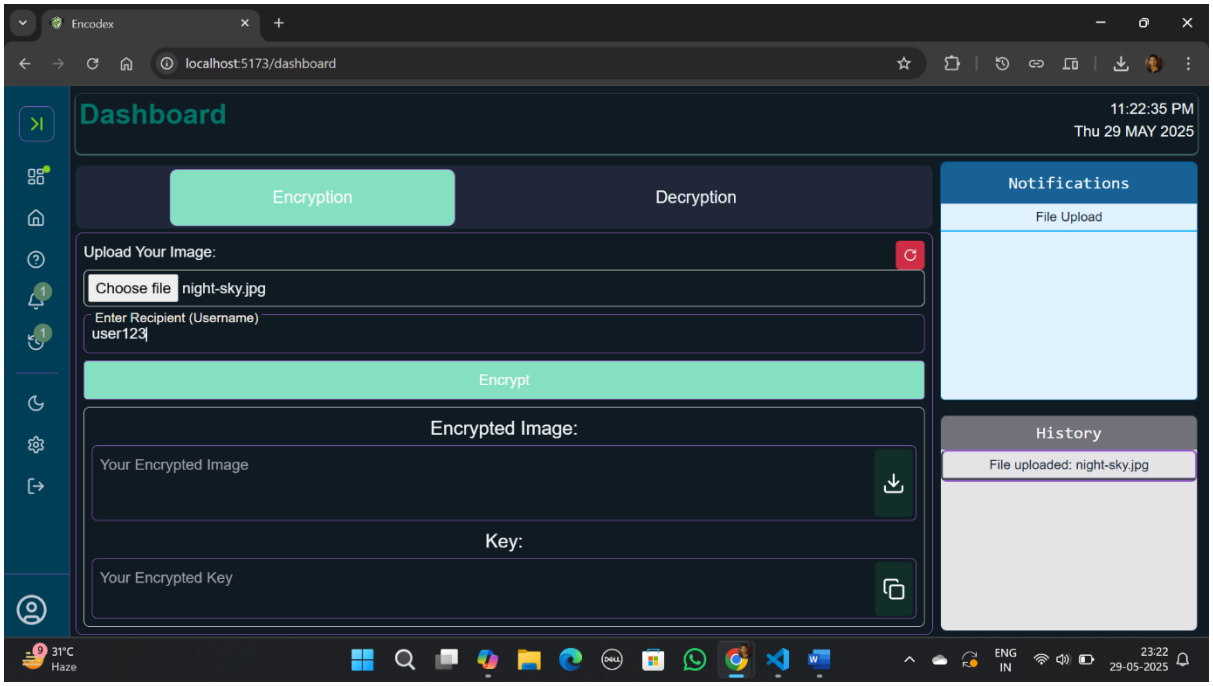


Fig. 14.1 – Upload Image Interface

This screen allows the user to upload an image file in PNG or JPEG format for encryption.

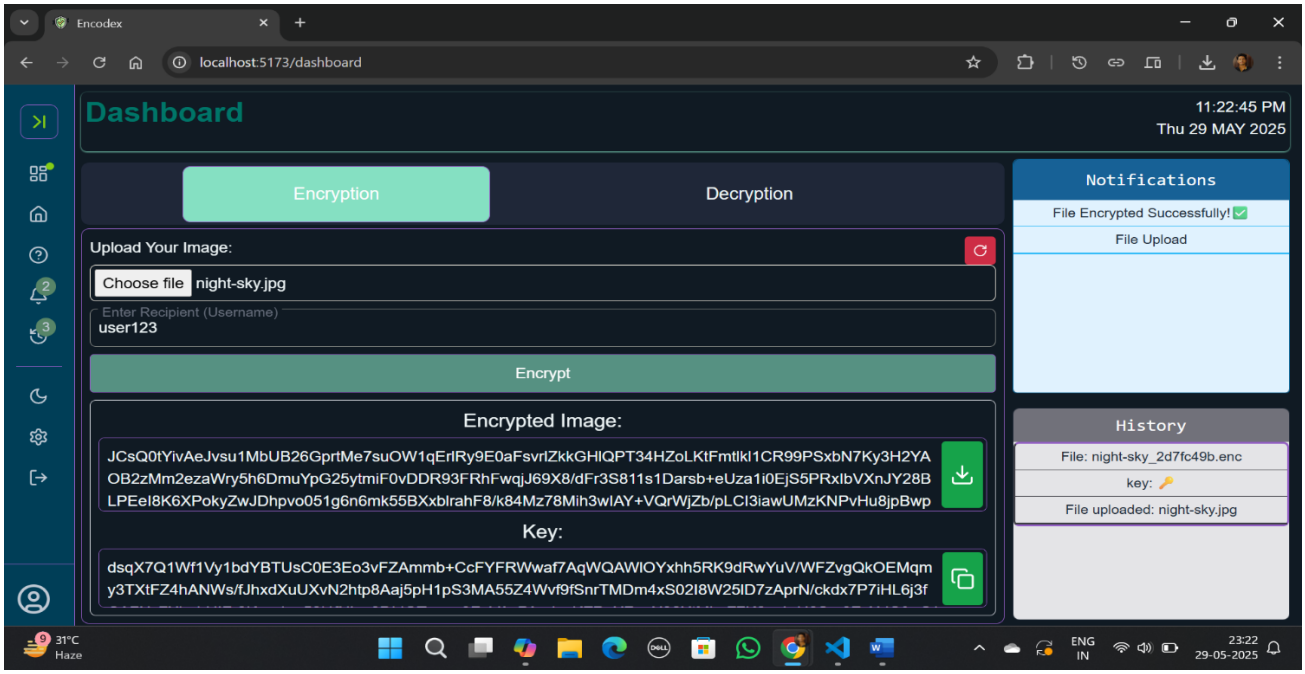


Fig. 14.2 – Image Encryption with Key Display

After uploading an image, the user receives a unique key upon successful encryption.

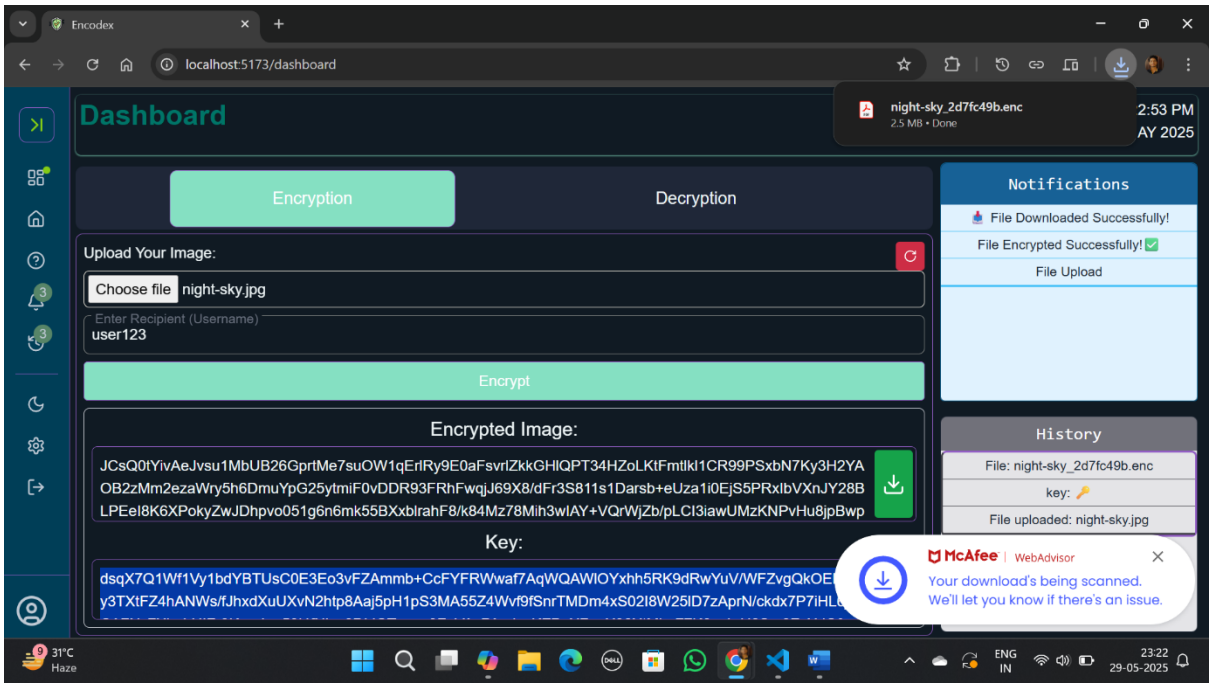


Fig. 14.3 – Download Encrypted File

The system provides the option to download the encrypted image securely to the local device.

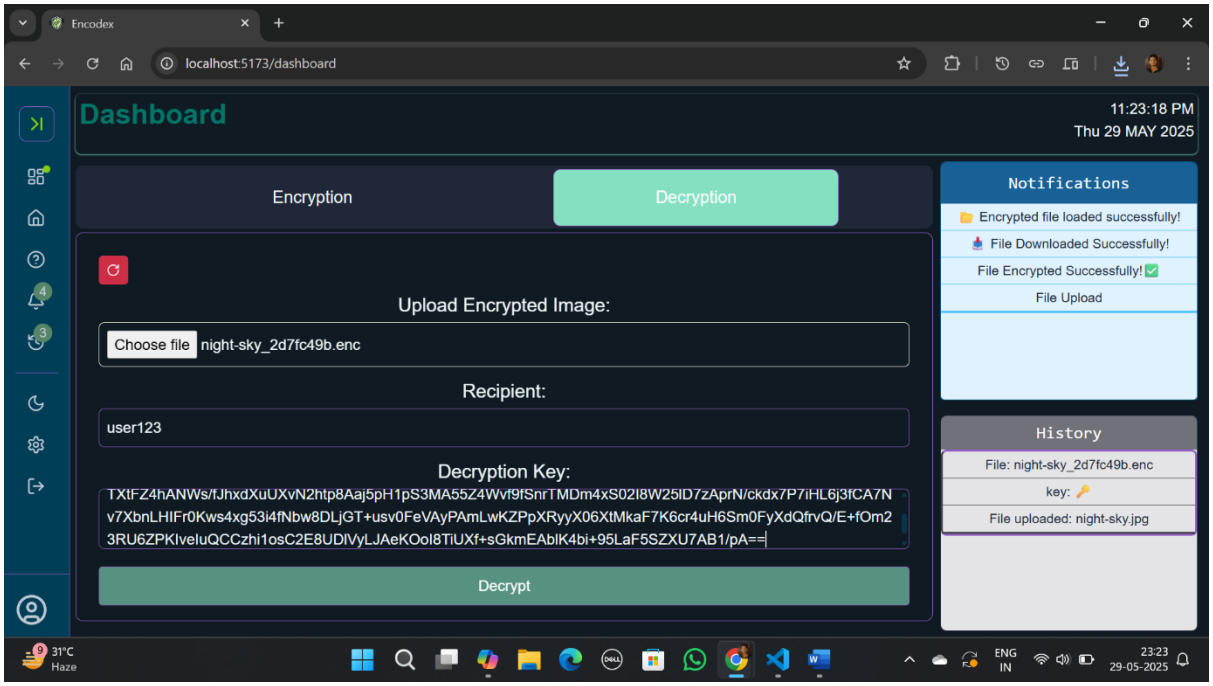


Fig. 14.4 – Decryption Input Page

This screen enables users to upload the encrypted image and enter the corresponding decryption key.

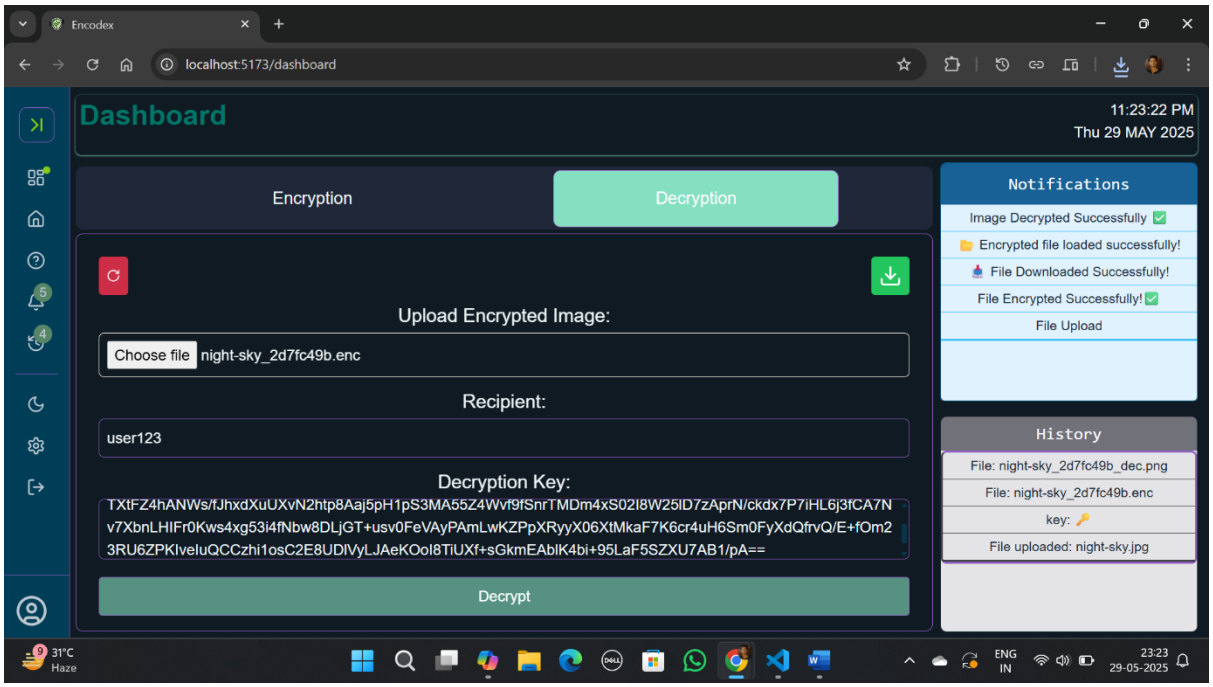


Fig. 14.5 – Decrypted Image Output

After successful key verification, the decrypted image is displayed or downloaded.

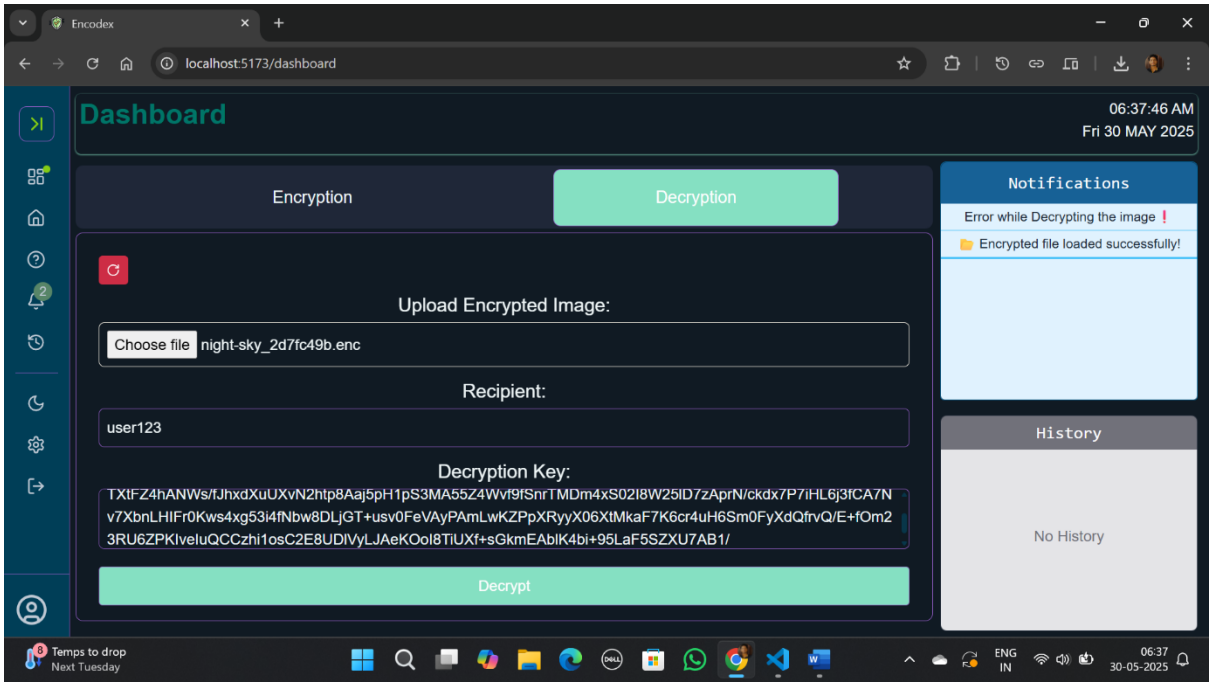


Fig. 14.6 – Error Message on Incorrect Key

The system notifies the user when an invalid decryption key is entered.

Each screenshot represents an important component of the user flow, ensuring a clear and secure image encryption/decryption experience.

CHAPTER 15

RESEARCH PAPERS

The EncodeX project was inspired and supported by several research works in the fields of cryptography, image security, and web-based application development. Below are some of the key papers and resources that contributed to the design and implementation of this project:

1. "Advanced Encryption Standard (AES) – A Survey"

- Source: GeeksforGeeks

- URL: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/

- Description: Provided a detailed explanation of the AES algorithm and its suitability for encrypting image data.

2. **"Image Encryption and Decryption using AES algorithm"**

- Source: GitHub Repository by Aditya Agrawal

- URL: https://github.com/aditya-agrawal16/Image-Encryption-and-Decryption-using-AES-algorithm

- Description: Served as a reference for basic image encryption/decryption logic in Python.

3. **"A Comparative Study of Various Image Encryption Techniques"**

- Authors: R. Ramesh, V. Sridevi

- Published in: International Journal of Computer Applications

- Description: Helped evaluate which encryption technique would be best for this project.

4. **"Web Application Security Practices"**

- Source: OWASP Foundation

- URL: https://owasp.org

- Description: Guided the security best practices followed in the EncodeX application.

5. **"Introduction to Flask for Web Development"

- Source: Flask Documentation
- URL: https://flask.palletsprojects.com/
- Description: Helped in structuring the Flask backend for handling encryption APIs.

6. **"Tailwind CSS Documentation"

- URL: https://tailwindcss.com/docs
- Description: Supported the front-end design and responsive layout for the EncodeX UI.

These resources provided the technical foundation and validation necessary for building a secure, efficient, and user-friendly image encryption system.

CHAPTER 16

REFERENCES

[1] GeeksforGeeks. "Advanced Encryption Standard (AES) – A Survey." Available at: https://www.geeksforgeeks.org/advanced-encryption-standard-aes/

[2] Aditya Agrawal, "Image Encryption and Decryption using AES Algorithm." GitHub Repository. Available at: https://github.com/aditya-agrawal16/Image-Encryption-and-Decryption-using-AES-algorithm

[3] Ramesh, R. & Sridevi, V. "A Comparative Study of Various Image Encryption Techniques." International Journal of Computer Applications, Vol. XX, No. XX, 2023.

[4] OWASP Foundation. "Web Application Security Guidelines." Available at: https://owasp.org

[5] Flask Documentation. "Welcome to Flask." Available at: https://flask.palletsprojects.com/

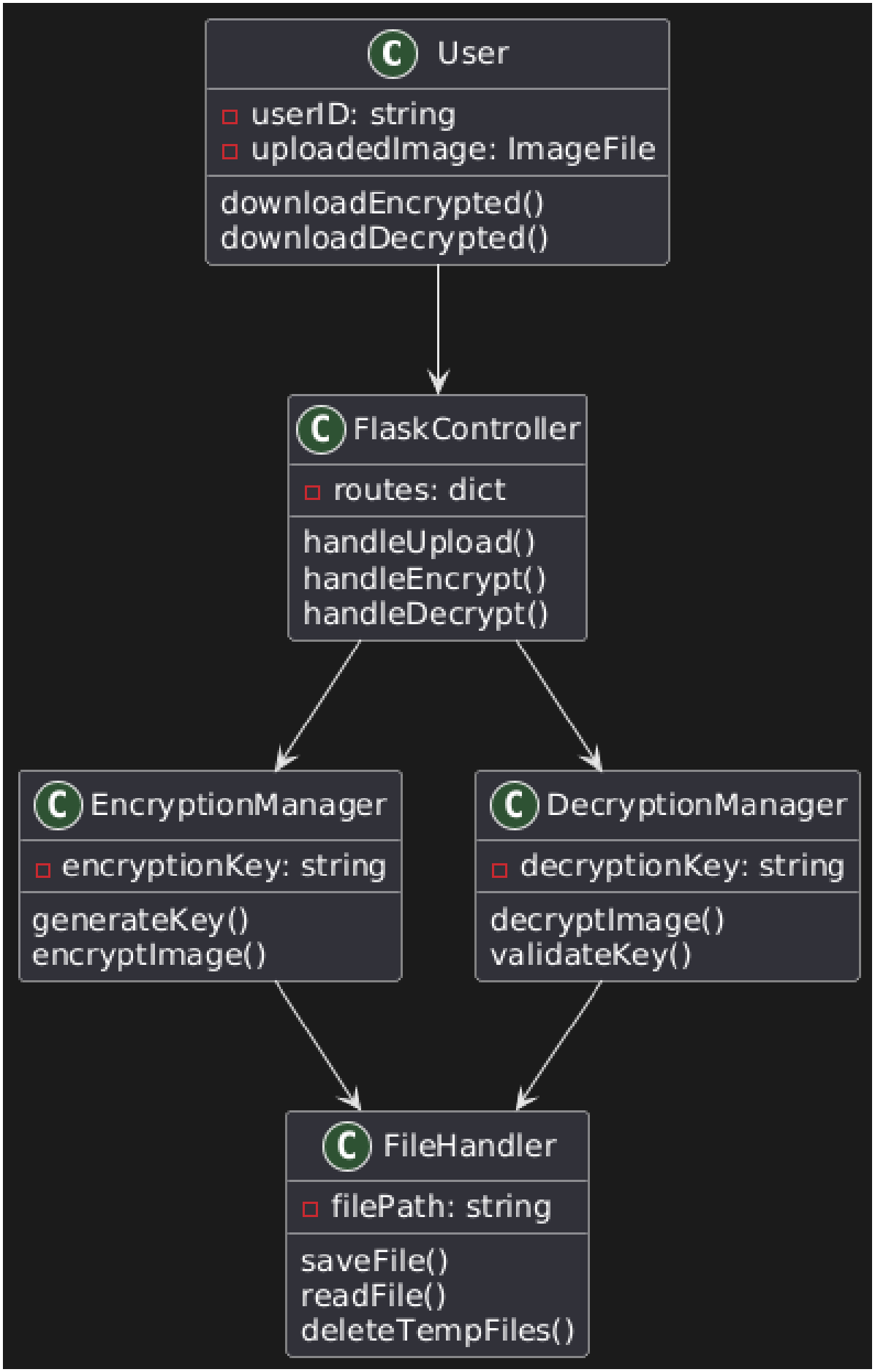
[6] Tailwind CSS Documentation. "Official Tailwind CSS Guide." Available at: https://tailwindcss.com/docs

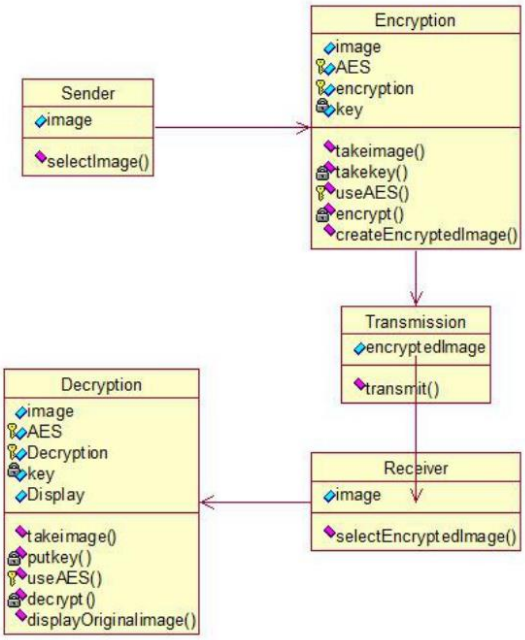
[7] Python Software Foundation. "Base64 Module — Encoding & Decoding." Available at: https://docs.python.org/3/library/base64.html

[8] ReactJS Documentation. "React – A JavaScript Library for Building User Interfaces." Available at: https://react.dev/

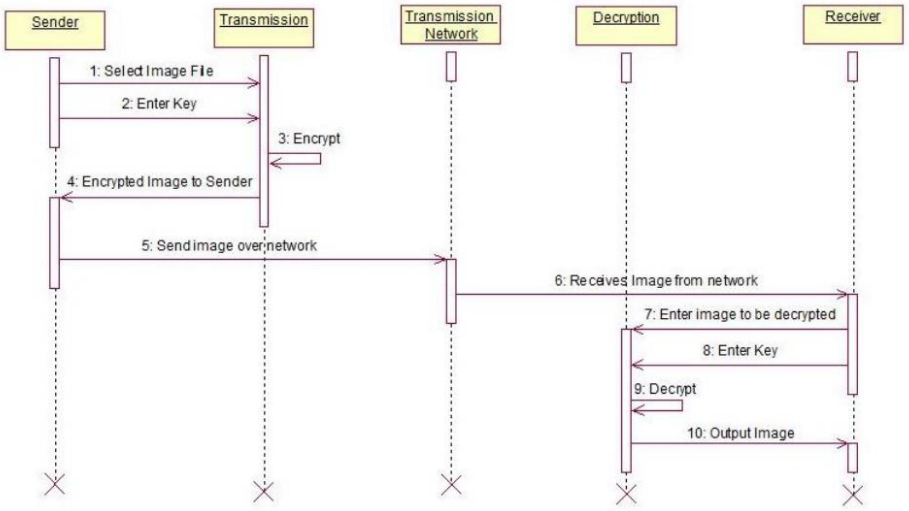
[9] Figma. "Collaborative Interface Design Tool." Available at: https://www.figma.com

[10] VS Code. "Visual Studio Code Editor by Microsoft." Available at: https://code.visualstudio.com

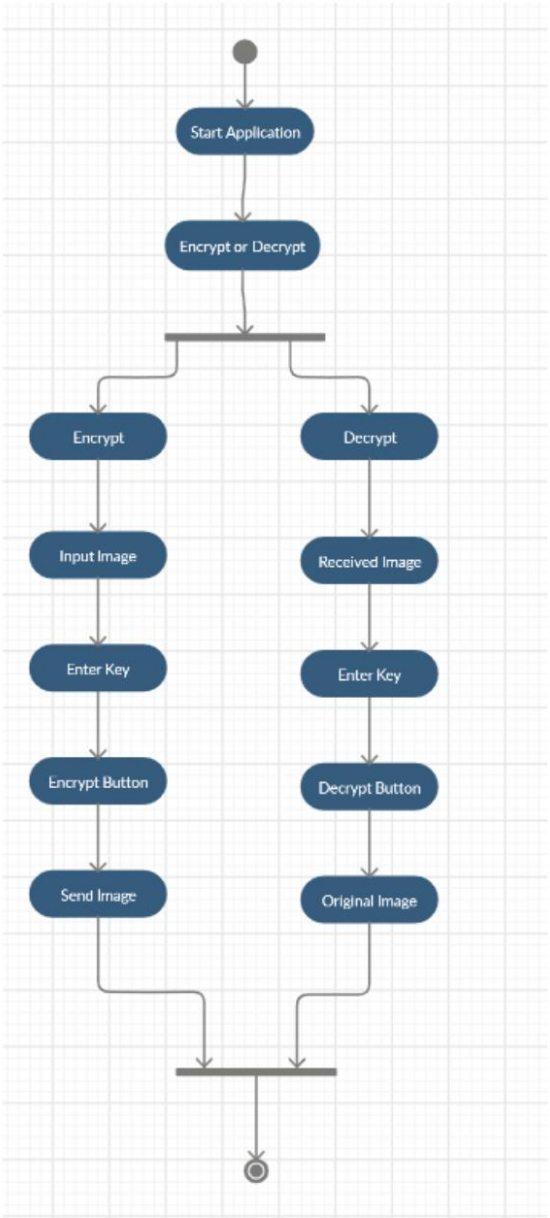

User
userID: string
uploadedImage: ImageFile
downloadEncrypted()
downloadDecrypted()
FlaskController
routes: dict
handleUpload()
handleEncrypt()
handleDecrypt()
EncryptionManager
encryptionKey: string
generateKey()
encryptImage()
DecryptionManager
decryptionKey: string
decryptImage()
validateKey()
FileHandler
filePath: string
saveFile()
readFile()
deleteTempFiles()



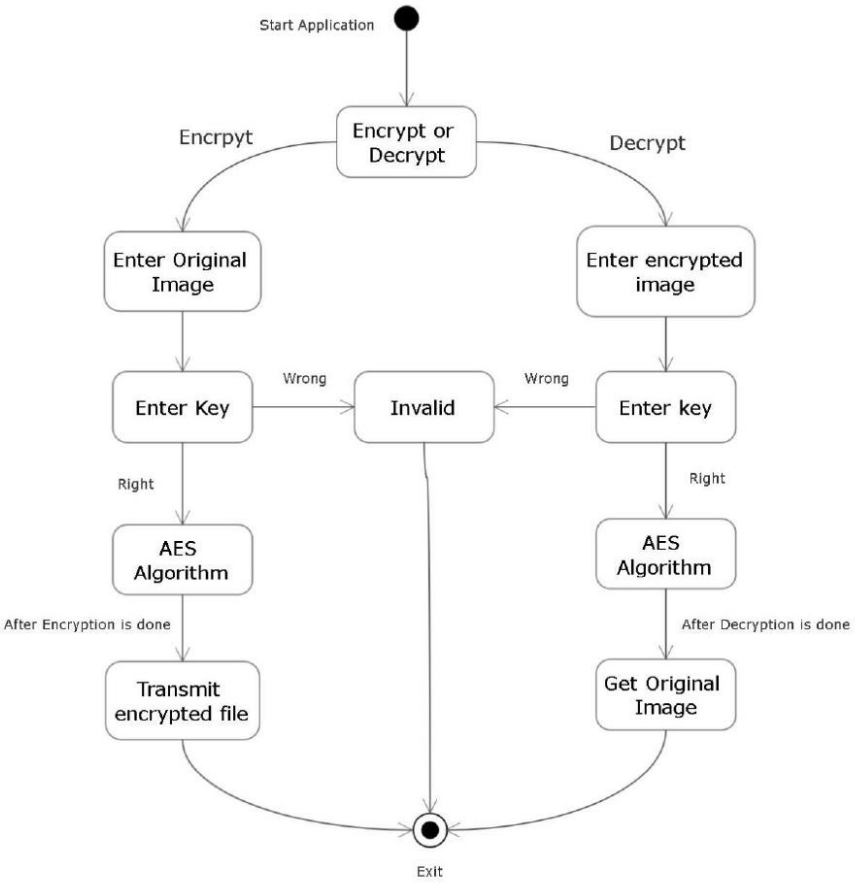
Sequence Diagram:



Activity Diagram:

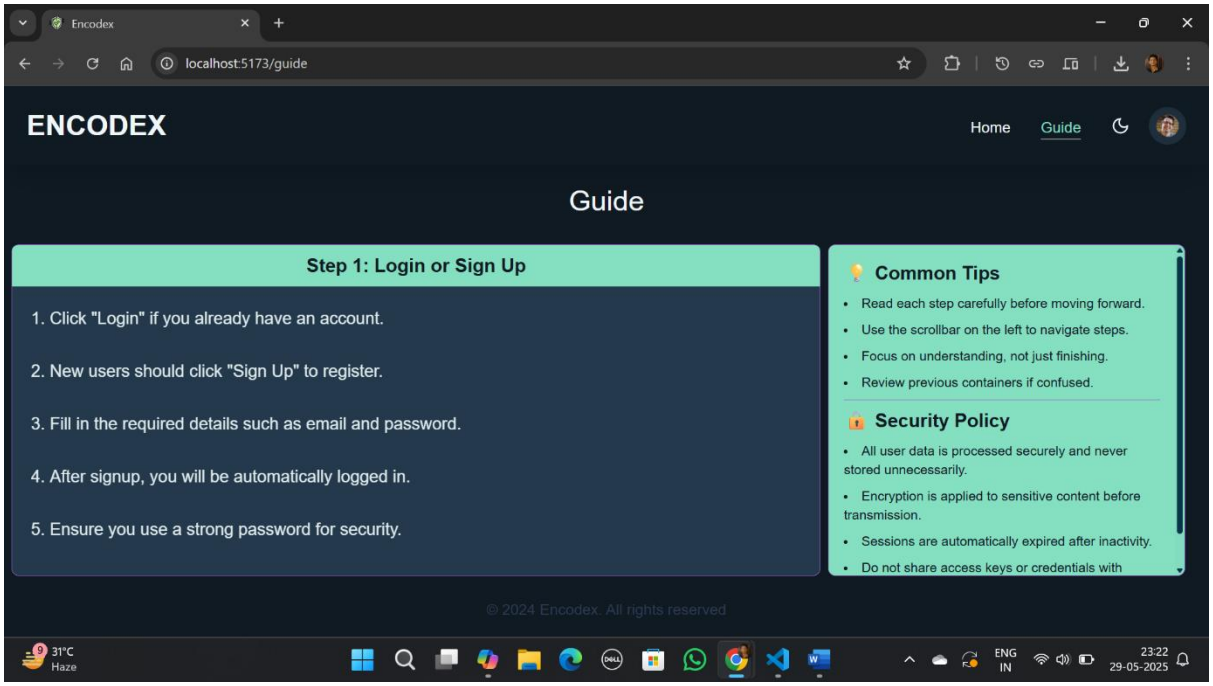


State Chart Diagram:



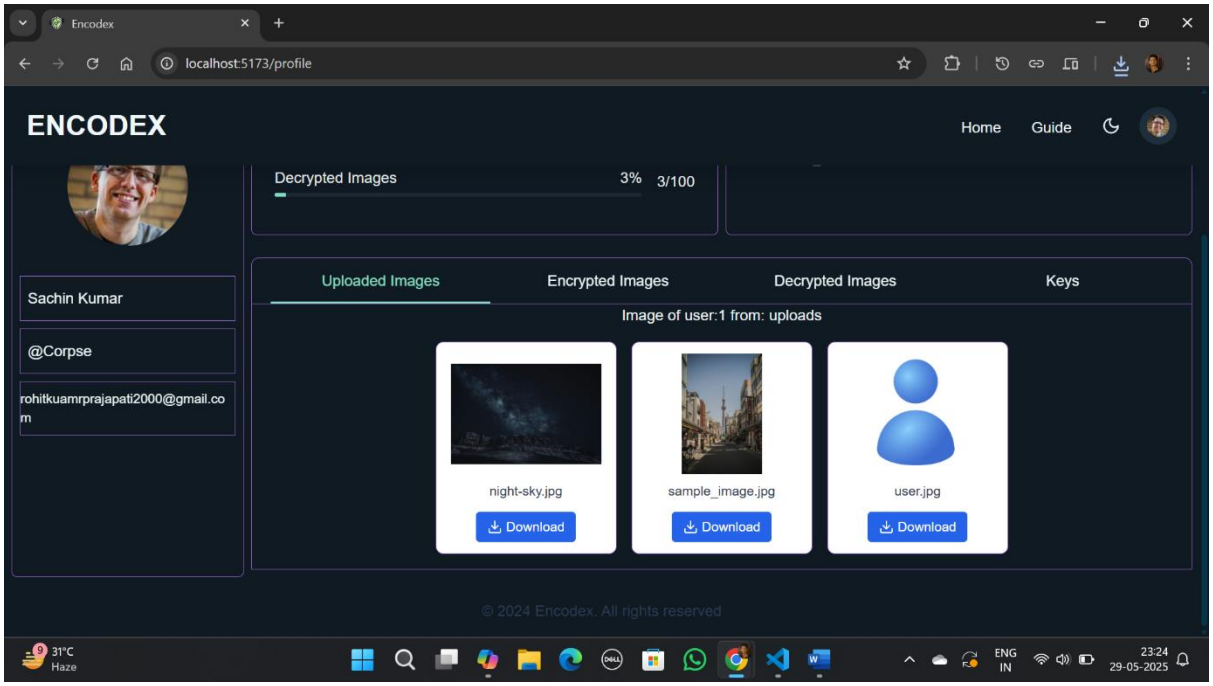
Home Page

Guide Page

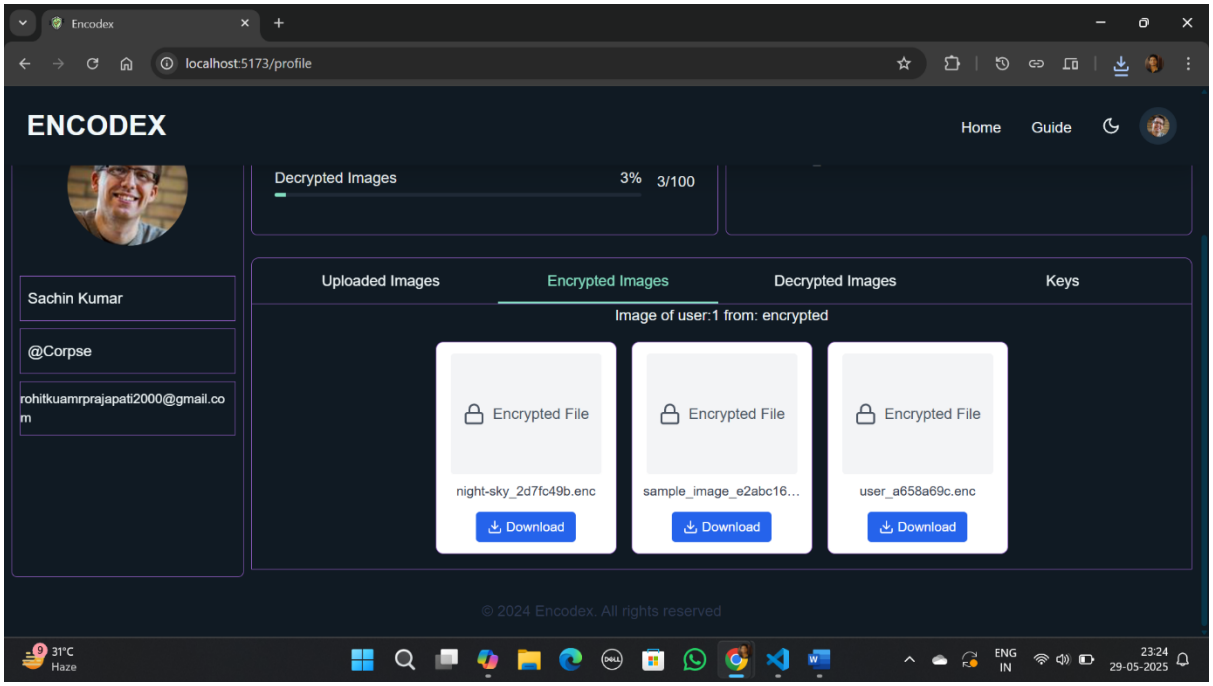


PROFILE PAGE

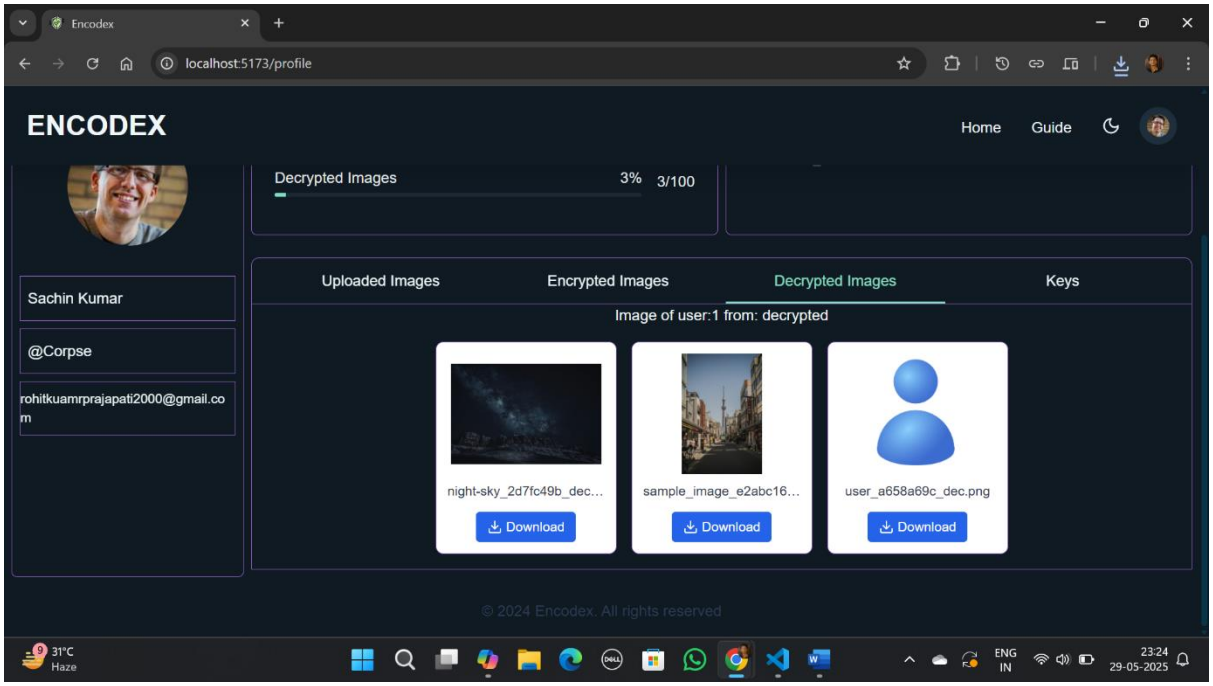
-Upload Section:



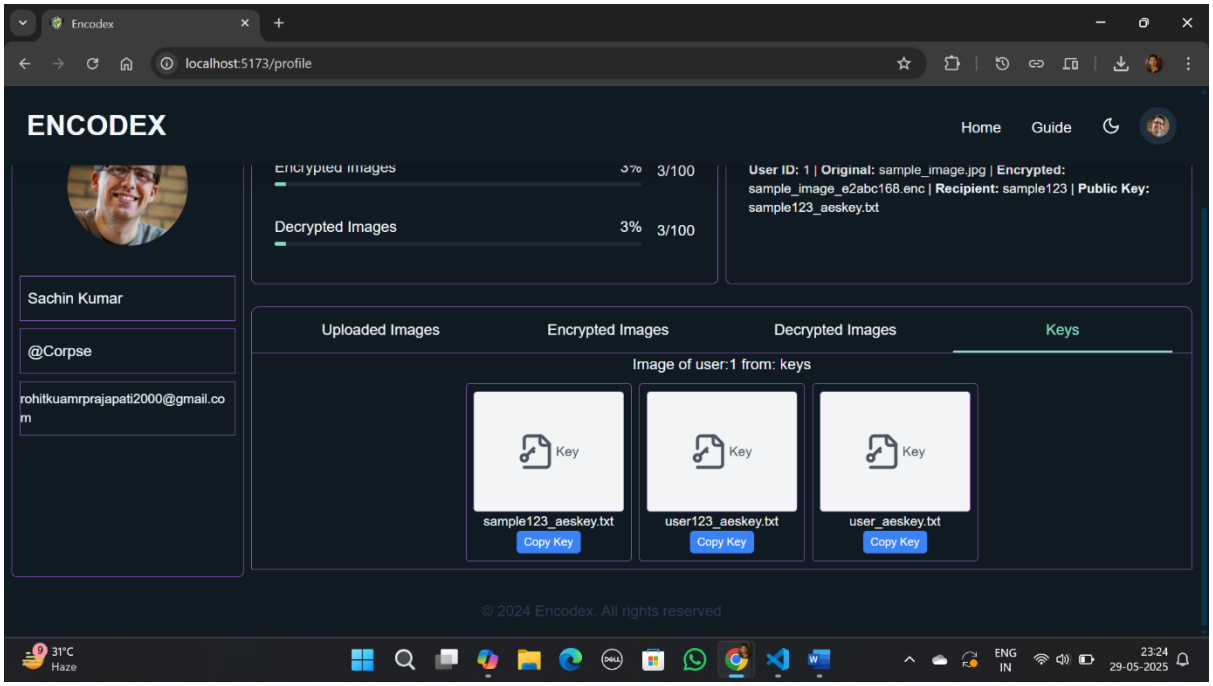
-Encrypted Image Section:



-Decrypted Image Section:



-Encrypted Public Key Section:



CERTIFICATE

This is to certify that the project report entitled "**ENCODEX – Securing Images Through Encryption"**, submitted by **Sachin Kumar Gola , Vinay Chhabra and Money Singh**, in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Computer Science and Engineering,** of **Anand Engineering College, Agra**, is a bonafide record of work carried out by them under my supervision and guidance.

This project work has been carried out at Anand Engineering College, Agra during the academic session 2024–2025.

Place: [Agra]

Date: [30/05/2025

Mr. Achal Tomar

(Project Coordinator)

CSE Department

Mr. Achal Tomar

(Head of Department)

CSE Department

CERTIFICATE (EXTERNAL GUIDE)

This is to certify that the project report entitled "ENCODEX – Securing Images Through Encryption", carried out by **Sachin Kumar Gola (**2100010100045), **Vinay Chhabra** (2100010100058), **Money Singh** (2100010100036) ,under my supervision during their final year project, is a bonafide work completed at Anand Engineering College.

The project has been found satisfactory and is submitted in partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering under Dr. A.P.J. Abdul Kalam Technical University, Lucknow, during the academic session 2024–2025.

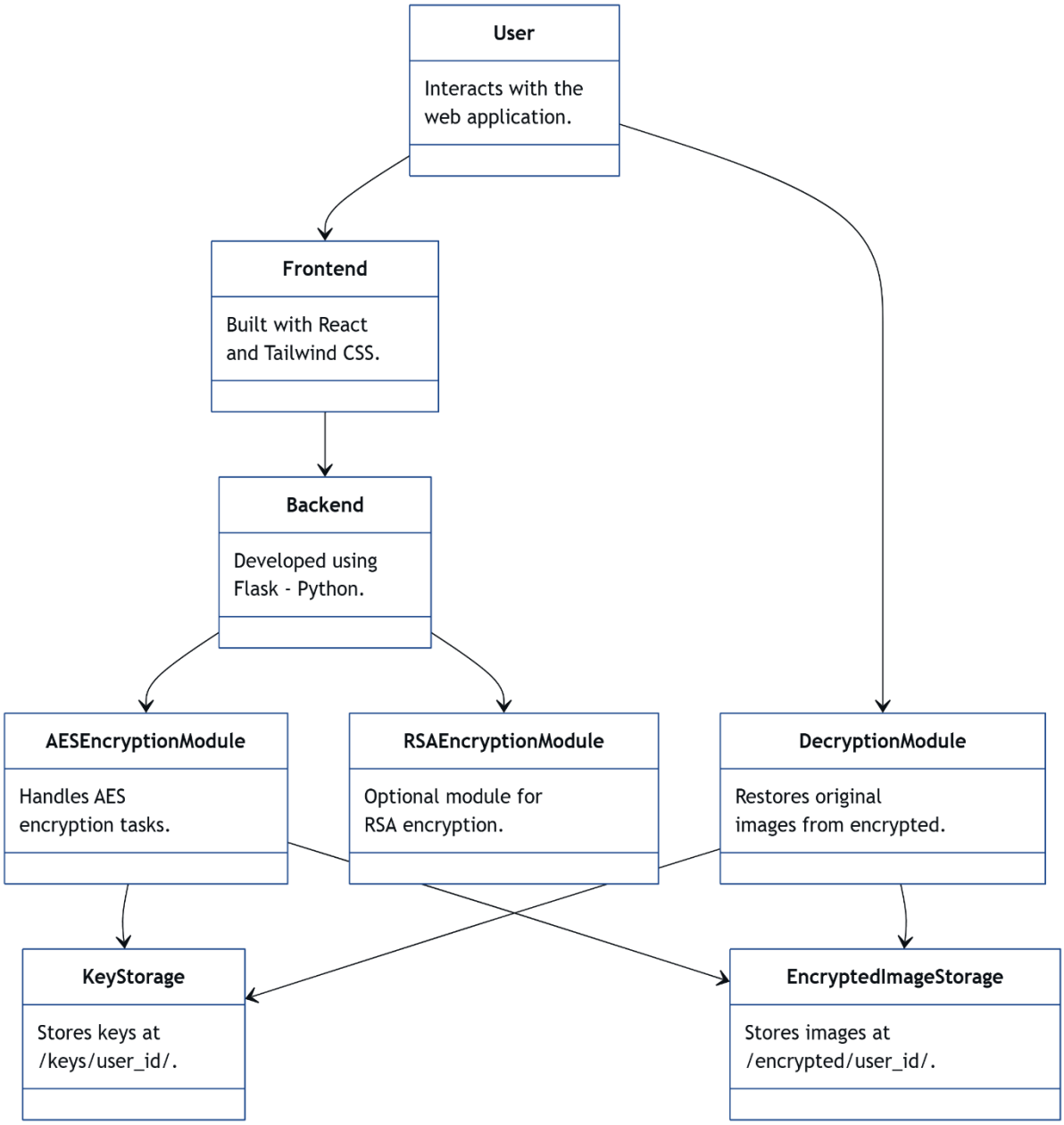
Place: [Agra]

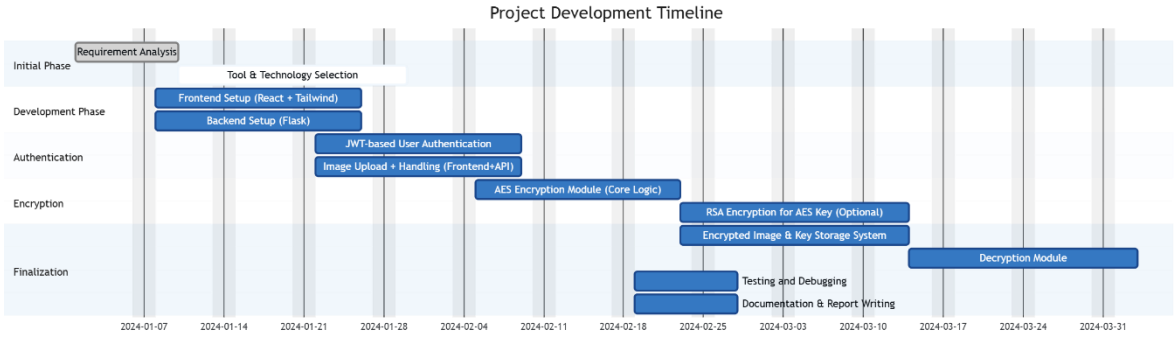
Date: [30/05/2025]

EXTERNAL EXAMINER

Dr. ……

(External Project Supervisor)


User
Interacts with the web application.
Frontend
Built with React and Tailwind CSS.
Backend
Developed using Flask - Python.
AESEncryptionModule
Handles AES encryption tasks.
RSAEncryptionModule
Optional module for RSA encryption.
DecryptionModule
Restores original images from encrypted.
KeyStorage
Stores keys at /keys/user_id/.
EncryptedImageStorage
Stores images at /encrypted/user_id/.


Project Development Timeline
Initial Phase
Development Phase
Authentication
Encryption
Finalization
Requirement Analysis
Tool & Technology Selection
Frontend Setup (React + Tailwind)
Backend Setup (Flask)
JWT-based User Authentication
Image Upload + Handling (Frontend+API)
AES Encryption Module (Core Logic)
RSA Encryption for AES Key (Optional)
Encrypted Image & Key Storage System
Decryption Module
Testing and Debugging
Documentation & Report Writing
2024-01-07
2024-01-14
2024-01-21
2024-01-28
2024-02-04
2024-02-11
2024-02-18
2024-02-25
2024-03-03
2024-03-10
2024-03-17
2024-03-24
2024-03-31