ABSTRACT

In an era where digital privacy is paramount, The “Secure Photo Vault” offers a secure, user centric solution for managing personal photos through a cloud based web application focused on confidentiality and data integrity. By integrating advanced encryption methods like Elliptic Curve Cryptography and Advanced Encryption Standard, alongside Two-Factor Authentication, the platform ensures robust access control and data protection. It combines ease of use with high-security standards, utilizing HTML, CSS, and JavaScript for a responsive interface, while Django and PostgreSQL handle backend data processing and storage. Features such as automatic image tagging and face recognition enhance organization, enabling users to efficiently locate images based on tags, dates, or faces. An intuitive search and filter system streamlines photo retrieval, making the platform powerful yet user-friendly. By merging innovative photo management tools with cutting-edge security, The project sets a new standard in protecting personal media in the digital age.

Keywords: Elliptic Curve Cryptography, Advanced Encryption Standard, Two-Factor Authentication, Automatic Image Tagging, Face Recognition

Contents

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

2FA	:	Two-Factor Authentication
AES	:	Advanced Encryption Standard
CPU	:	Central Processing Unit
CSS	:	Cascading Style Sheets
DFD	:	Data Flow Diagram
ECC	:	Elliptic Curve Cryptography
HTML	:	Hypertext Markup Language
ORM	:	Object-Relational Mapping
OS	:	Operating System
OTP	:	One-Time Password
RAM	:	Random Access Memory
UI	:	User Interface

CHAPTER 1

INTRODUCTION

In an era where digital privacy is of increasing concern, safeguarding personal photos and sensitive information has become a top priority for individuals and organizations alike. Traditional cloud storage services often fail to provide the level of security required to protect against unauthorized access, data breaches, or surveillance. The "Secure Photo Vault" is designed to address these challenges by offering a highly secure, cloud-based platform for managing and storing photos. By combining advanced encryption techniques such as Elliptic Curve Cryptography (ECC) and Advanced Encryption Standard (AES), the system ensures that user data is fully protected with end-to-end encryption (E2EE). The platform also incorporates two-factor authentication (2FA) for an additional layer of security during the login process. Beyond its focus on security, The project provides users with a host of intelligent features like automatic image tagging, face recognition, and tag-based sorting, streamlining the process of organizing and retrieving photos. With its user-friendly interface and robust security framework, this project aims to offer a seamless and protected photo management experience, especially in a world where digital privacy is under constant threat.

1.1 SCOPE OF THE PROJECT

This project ambitiously focuses on developing a secure and user-friendly online platform meticulously tailored for the management and storage of personal images, with the overarching goal of creating a space that prioritizes privacy and data protection. The emphasis is unreservedly on security, where users are equipped with intuitive tools that facilitate the safe handling of their digital memories, ensuring that their personal photos remain protected from unauthorized access.

Going beyond traditional image storage solutions, the project aspires to be a leader in redefining how users interact with their visual content. The project seeks not just to provide a repository for images but to create an environment that encourages users to take control of their digital assets. By

implementing advanced encryption techniques, such as Elliptic Curve Cryptography (ECC) and Advanced Encryption Standard (AES), the platform ensures that user's images are safeguarded, fostering a sense of trust and reliability.

The project's uniqueness lies in its focus on advanced features designed to enhance user experience and streamline photo organization. From automatic tagging using a fine-tuned VGG16 model to efficient sorting through face recognition and tag-based functionalities, every aspect is geared towards improving the way individuals manage their digital memories.

In summary, the project represents a harmonious blend of technology and user-centric design, significantly contributing to the evolution of digital photo management. It is a bold step towards ensuring the privacy and security of personal images in an increasingly digital world, addressing the critical need for reliable and user-friendly solutions that prioritize the protection of personal data.

1.2 RELEVANCE OF THE PROJECT

In an age where digital privacy and data security are paramount, " Secure Photo Vault" emerges as a crucial solution for individuals seeking to protect their personal images. As the volume of digital content shared online continues to grow, so do the risks associated with unauthorized access and data breaches. This project addresses these pressing concerns by offering users a secure platform for managing their photos, emphasizing the importance of safeguarding personal memories from potential threats.

The relevance of this project extends beyond individual users; it is particularly significant for professionals and organizations that handle sensitive visual content. With the increasing prevalence of cyberattacks and privacy violations, ensuring the integrity of personal and proprietary images has become essential. By combining advanced encryption techniques with user-friendly organizational tools, The project not only enhances the security of personal images but also fosters a sense of trust among users. This project aligns with the growing demand for privacy-focused solutions in

today's digital landscape, making it a timely and necessary contribution to the field of digital data management.

1.3 ORGANISATION OF THE REPORT

This report is organized into several chapters, each covering distinct aspects of the "Secure Photo Vault". Chapter 1 introduces the project, providing an overview of its objectives, significance, and an outline of the report structure. Chapter 2 focuses on system study, discussing existing technologies and systems relevant to secure photo management and cloud storage, along with the proposed solution. It emphasizes advanced encryption, authentication methods, and the development methodology, and includes a comprehensive requirements analysis. Chapter 3 details the system design, covering both high-level architecture (such as workflows and user interaction) and low-level design (database schema and core functionalities), with particular attention to encryption techniques like ECC and AES, as well as two-factor authentication (2FA). Chapter 4 provides an in-depth look at the system's implementation, describing modules including user authentication, image encryption and decryption, face recognition, and automatic tagging, along with screenshots of the user interface and key code snippets. Chapter 5 outlines the testing strategies to ensure system robustness, covering unit, integration, and performance testing, with detailed test cases and results that demonstrate the system's reliability. Finally, Chapter 6 summarizes the project's outcomes and discusses potential future enhancements, such as refining the face recognition system or adding new photo management features to further enhance The Secure Photo Vault.

CHAPTER 2

SYSTEM STUDY

In this chapter, we delve into the comprehensive analysis of the "Secure Photo Vault" system, examining its fundamental architecture, design principles, and the technological frameworks that underpin its functionality. A thorough understanding of the system's requirements and components is crucial for ensuring that the platform effectively meets the needs of its users while maintaining high standards of security and usability. This study encompasses an exploration of the key features, the rationale behind the chosen technologies, and an assessment of how these elements work together to create a robust and reliable solution for managing personal photos. By evaluating both the current digital landscape and user expectations, this chapter sets the stage for the subsequent design and implementation phases, guiding the development process toward achieving a secure, efficient, and user-friendly photo management experience.

2.1 EXISTING SYSTEM

The current solutions for managing and storing personal photos are largely reliant on cloud-based platforms, which offer users basic functionality such as image uploading, storage, and sharing. These systems typically include features like automated sorting and basic organizational tools. While convenient, they often fail to address key concerns around data security, privacy, and secure access control.

Most existing platforms do not implement strong end-to-end encryption, leaving user data vulnerable to potential breaches or unauthorized access. Even when encryption is used during transmission or storage, it is often insufficient to ensure complete privacy, as service providers may still have access to unencrypted data. This exposes users to potential risks, such as data theft or misuse.

In addition to security concerns, these platforms can be overly complex and difficult to navigate. Users frequently encounter challenges in managing and organizing their image libraries due to a

cluttered interface and limited tools for efficiently categorizing and tagging photos. The lack of robust organizational features makes it hard to maintain an orderly and secure photo archive.

The reliance on centralized cloud infrastructure also presents risks related to data sovereignty and control. Users must trust that their data will not be misused or improperly accessed by external parties, which has become a growing concern in light of recent data breaches and security incidents involving cloud-based storage solutions.

2.2 PROPOSED SYSTEM

The proposed system is designed to provide users with a secure and intuitive platform for the storage and management of personal photos. The system emphasizes privacy and data security by encrypting all images before storage, ensuring that personal photos are safeguarded from unauthorized access. The key objectives of this system include providing robust encryption, efficient organization, and a seamless user experience, all while maintaining simplicity.

Key Features of the Proposed System

- **Encrypted Image Storage:** The platform will use Elliptic Curve Cryptography (ECC) and the Advanced Encryption Standard (AES) to encrypt images before they are stored. This encryption ensures that user photos are stored securely, preventing unauthorized individuals from accessing or viewing the images. Although the system does not implement end-to-end encryption, encryption is applied to all images before they are stored in the cloud.

- **Secure Image Upload and Retrieval:** Users can upload and retrieve encrypted images through a user-friendly interface. The system will automatically encrypt each image upon upload, ensuring that only the user can view or download the original image after decryption.

- **Face Recognition for Organization:** To enhance organization, the platform will incorporate face recognition technology to allow users to sort and search their photos based on identified faces. This feature will improve the user's ability to manage their image collection efficiently.

- **Automatic Image Tagging:** The system will automatically assign tags to uploaded images based on predefined categories such as *human, landscape, animal,* and *other*. This feature will make it easier for users to search and organize their images according to these tags.

- **User Authentication and Authorization:** The system will feature strong authentication methods, including two-factor authentication (2FA), to ensure secure access to user accounts and prevent unauthorized access to photos and personal data.

- **Cross-Platform Compatibility:** Designed to work seamlessly across desktop and mobile browsers, the platform provides users with the convenience of accessing their secure photo vault from any device, ensuring flexibility and accessibility.

2.2.1 Development Methodology

The development of "Secure Photo Vault" follows the agile methodology with a focus on iterative development, ensuring flexibility and responsiveness to evolving user needs. This approach promotes collaboration, adaptability, and continuous improvement throughout the development lifecycle. Key aspects of the agile methodology employed in this project include:

- **Iterative Development:** The project is broken down into small, manageable iterations, each focusing on implementing specific features such as encryption, user authentication, and image tagging. This iterative approach allows for regular releases and continuous user feedback, ensuring that the final product aligns with user expectations and security requirements.

- **Continuous Integration and Testing:** Code changes are regularly integrated, followed by automated and manual testing to identify and resolve issues early in the development process. Continuous integration ensures that new features, such as encryption and face recognition, are thoroughly tested and do not introduce bugs or security vulnerabilities.

- **User-Centric Design:** User feedback is a priority throughout the development process. Regular demonstrations and user testing sessions are conducted to gather insights, helping to

fine-tune features like image tagging, search, and encryption. This ensures that the platform remains user-friendly and meets the privacy needs of its users.

- **Flexibility and Adaptability:** The agile methodology allows for adjustments in project scope and requirements based on ongoing feedback and changing user needs. This ensures that the system remains relevant and up-to-date with the latest security and usability requirements.

By adopting agile, the project aims to deliver a secure, efficient, and user-friendly platform that addresses the critical need for privacy and data protection in personal photo management.

2.3 REQUIREMENT ANALYSIS

The requirement analysis for "Secure Photo Vault" focuses on identifying the key software and hardware needs to ensure the successful implementation of the system's core functionalities. This section outlines the essential requirements to build a secure, user-friendly, and scalable platform for photo storage and management.

2.3.1 Software Requirements

- **Web Framework:**
  - **Django:** It is a high-level python web framework designed to facilitate the development of secure and scalable web applications. It streamlines backend operations, including user authentication and database interactions, allowing developers to focus on core functionalities. Django's extensive library and built-in security features make it ideal for managing complex applications.

- **Frontend Technologies:**
  - **HTML/CSS:** For structuring and styling the web application's user interface, HTML and CSS are used in tandem. HTML provides the foundational structure of the website, defining elements like headings, images, buttons, and forms. CSS enhances the visual presentation by applying styles such as colors, fonts, and layouts, ensuring a clean and intuitive design. Together, they create a cohesive and responsive layout that adapts smoothly to different devices and screen sizes, offering an engaging and user-friendly experience across various platforms.
  - **JavaScript:** It is used to add interactivity and dynamic features to the web application, enhancing the user experience. By enabling features like form validation, interactive galleries, and animated transitions, JavaScript contributes to a more engaging and intuitive application. It is also crucial for integrating frontend logic with backend operations seamlessly.
- **Database:**
  - **PostgreSQL:** A powerful open-source relational database known for its reliability, flexibility, and data integrity features. It securely stores user data, including encrypted images, login credentials, and tags, with robust support for concurrent transactions and complex queries. PostgreSQL's security features, like data encryption at rest, help protect sensitive user data from unauthorized access.
- **Encryption Libraries:**
  - **PyCryptodome:** It is a python library for cryptographic functions, used here to implement Elliptic Curve Cryptography (ECC) and Advanced Encryption Standard (AES). ECC and AES are crucial for securing data transmission and storage, ensuring that sensitive information remains confidential. By leveraging PyCryptodome, the system guarantees secure encryption and decryption processes that meet high-security standards.

- **Image Processing Libraries:**
  - **Pillow:** An image processing library in Python that facilitates operations like reading, modifying, and saving image files. It is essential for tasks such as resizing, compressing, and encoding images during encryption and decryption. Pillow provides a versatile toolkit for managing images efficiently, enabling seamless image processing within the application.
- **Machine Learning Libraries:**
  - **TensorFlow/Keras:** Popular machine learning libraries that support training and fine-tuning of models, including the VGG16 model used for image tagging. This setup enables automatic categorization of uploaded images into predefined categories, enhancing the application's functionality. The integration of machine learning makes image management more intelligent and user-friendly by adding automated tagging.
- **Testing Frameworks:**
  - **Pytest/Unittest:** Frameworks for running unit and integration tests to validate the system's functionality, performance, and security. They allow developers to systematically test individual modules and their interactions, ensuring reliable performance and identifying any issues early in development. Regular testing ensures the application is robust, secure, and performs as expected across various scenarios.
- **Version Control System:**
  - **Git:** A distributed version control system that tracks code changes, facilitating collaboration among team members during development. It allows developers to create branches, experiment with new features, and merge changes without affecting the main codebase. By providing a complete history of changes, Git makes it easy to roll back to previous versions, ensuring code stability and efficient issue resolution throughout the project's lifecycle.

2.3.2 Hardware Requirements

- **Server Specifications:**
  - **Processor:** Multi-core processor (e.g., Intel ® Core i5 or equivalent) to handle concurrent user requests and operations efficiently.
  - **RAM:** Minimum of 8GB RAM to support the web application and database operations simultaneously.
  - **Storage:** At least 100GB of storage, SSD preferred for faster data access and application performance.
  - **Network:** Reliable high-speed internet connection to facilitate smooth communication and data transfer.
- **Client Requirements:**
  - **User Devices:** Users can access the system from any device with internet access, including smartphones, tablets, and computers.
  - **Browser Compatibility:** The application should be compatible with major web browsers (Chrome, Firefox, Safari, Edge) to ensure accessibility for all users.

The chapter analyzes highlights the limitations of current photo storage platforms and how this project addresses them. The proposed system focuses on data security through ECC and AES encryption, two-factor authentication, and secure storage, while also enhancing usability with features like face recognition and automatic tagging. Developed using the agile methodology, the project emphasizes user feedback and continuous improvement. The chapter also outlines the software and hardware requirements necessary for a secure and efficient photo management solution. Next chapter outlines the architectural design of the project, focusing on secure data handling, user interaction, and system scalability.

CHAPTER 3

DESIGN

The design phase of "Secure Photo Vault" project aims to create a robust and secure architecture for managing and protecting user images. This chapter outlines the system's design components, including the overall architecture, data flow, encryption processes, and user interactions. By focusing on modular design and secure handling of sensitive information, this system ensures data confidentiality, integrity, and usability. The diagrams provided illustrate the logical flow, data exchanges, and functional modules that enable secure image management and retrieval, including features like registration, authentication, encryption, decryption, and face search. The design prioritizes user-friendly interaction while maintaining high standards of security. Additionally, the system architecture is flexible and scalable, paving the way for potential integration of advanced security features, such as cloud storage solutions or enhanced multi-factor authentication. Every design decision is made to ensure that components interact efficiently while isolating critical functionalities for maximum security.

3.1 BLOCK DIAGRAM

The block diagram Fig. 3.1 represents the high-level architecture of the project. It illustrates the key components and interactions, including registration, login, OTP validation, encryption, decryption, deletion, and face search functionalities. This diagram helps in visualizing the flow of information and the primary functionalities within the system. Each component interacts with user inputs and system processes to ensure secure and efficient management of image data. The architecture emphasizes a seamless user experience while maintaining strong encryption protocols to safeguard sensitive information. Furthermore, the design ensures that each function, from registration to face search, operates independently but integrates smoothly with the overall system to enhance security and usability. This modular approach also simplifies the process of scaling and adding new features in the future.

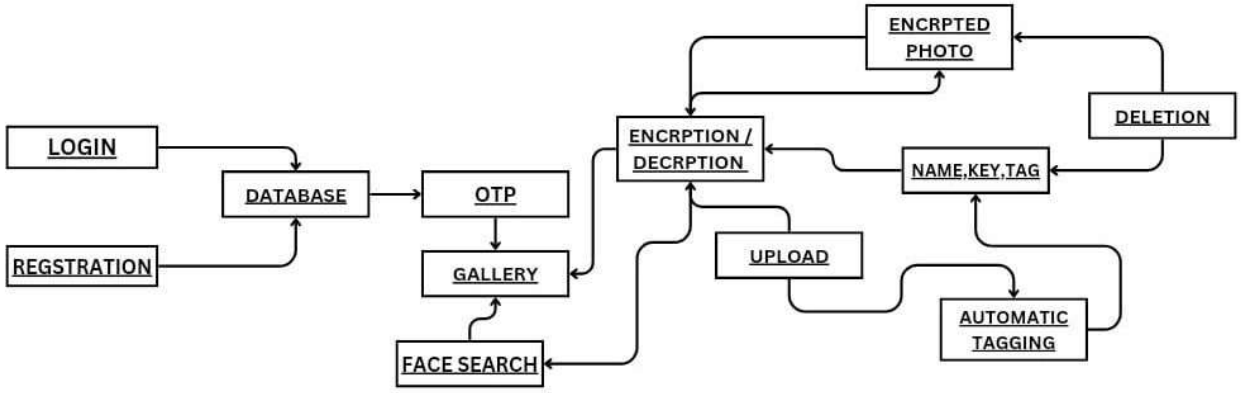

Figure 3.1: Block Diagram

3.2 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is a graphical representation illustrating how data flows within a system. It is a valuable tool in software engineering and systems analysis, providing insights into the data movement among various components or processes. DFDs are particularly useful for visualizing interactions within a system and understanding data flow. They help break down complex systems into simpler, manageable processes by showing how data is processed and transferred. Additionally, DFDs are beneficial in identifying potential constraints or limitations, enabling developers to optimize data handling and improve system performance. DFDs also aid in pinpointing areas where data security and integrity need to be reinforced, making them crucial for designing robust systems. By providing a clear overview of the system's data processes, DFDs facilitate better communication among stakeholders, ensuring a shared understanding of system functionality and requirements.

3.2.1 Level-0 DFD

The Level 0 DFD for "Secure Photo Vault" provides a high-level view of the system's interaction with users and its core functionality. At this level, the Fig. 3.2 shows that the user interacts with the central system by providing inputs and requests. The system processes these inputs, managing essential tasks such as handling user authentication, storing secure information, and enabling

access to secure photo storage and management features. The system database for storing user-specific information, such as login credentials and other identifying details, which are essential for secure access and personalized interactions. This Level 0 DFD illustrates the main flow of data and high-level processes, setting the foundation for more detailed diagrams that expand on specific functionalities within the project.

Figure 3.2: Level 0 DFD

3.2.2 Level-1 DFD

The Level 1 DFD Fig. 3.3 illustrates how users interact with the system through a series of secure and efficient processes. First, in the registration phase, users provide email, name, and password, which are stored as user details. When logging in, users enter their credentials, triggering the OTP authentication process, where an OTP is generated and verified for additional security. Once authenticated, users can upload images, which are then encrypted and stored securely in the encrypted file store. For viewing, the decryption module retrieves and decrypts encrypted images, providing accessible decrypted images to the user. The deletion process allows users to permanently remove both encrypted and decrypted versions of selected images from the system. Additionally, users can perform face search to filter and retrieve images based on facial recognition. This DFD emphasizes data security with OTP verification and encryption, while also offering efficient image management and search functionalities.

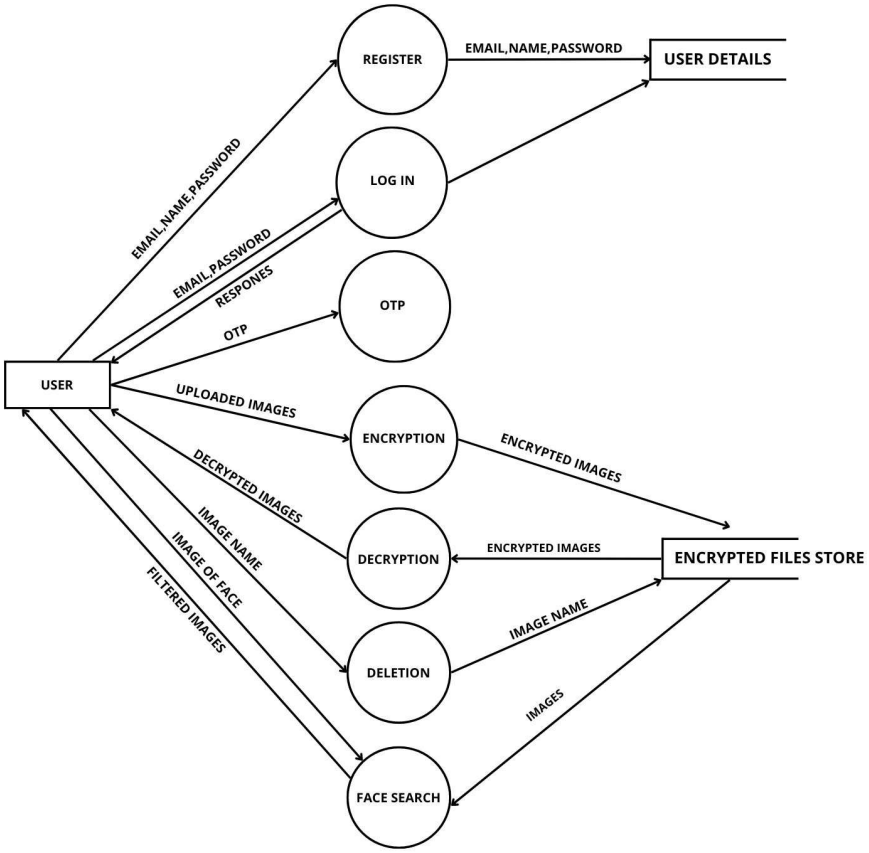

Figure 3.3: Level 1 DFD

3.3 DATABASE DESIGN

Database design encompasses processes facilitating the design, development, implementation, and maintenance of enterprise data management systems. Well-designed databases are easy to maintain, enhance data consistency, and are cost-effective in terms of disk storage space. The database designer determines how data elements correlate and what data must be stored. By carefully structuring the data and defining relationships, designers ensure that the database can efficiently handle complex queries and large volumes of information. Additionally, a strong database design includes considerations for data security and user access controls, safeguarding sensitive information from unauthorized access. Ultimately, effective database design lays the foundation for reliable data management, supporting the organization's strategic goals and operational efficiency.

3.3.1 Table

Table 3.1 shows the schema of the User, designed to store essential user information in the application, acting as a central repository for user-related data. It includes four fields: id, which is of type ObjectId and serves as the primary key to uniquely identify each user; name, which stores the user's name in text format; email, a text field with a unique constraint to ensure each email is distinct within the system; and hashedPassword, which holds the securely hashed password for authentication. This schema enables efficient management and retrieval of user information, supporting secure access to the application.

Table 3.1: User

Field Name	**Datatype**	**Constraint**
ID	ObjectId	Primary Key
Name	Text	
Email	Text	Unique
Hashed Password	Text	

This chapter details the design phase of the project focusing on creating a secure and scalable architecture for managing user images. This includes designing the system's core components, data flow, encryption, and user interactions to ensure data security and usability. Block and data flow diagrams illustrate the processes involved in registration, login, OTP validation, encryption, decryption, deletion, and face search, emphasizing modularity and security. The database design defines essential tables like the User table, ensuring efficient data management and secure storage of user information. The system's design is user-friendly and adaptable, supporting potential future enhancements such as cloud integration and advanced authentication methods. Next chapter details the process of transforming the design of the project into a functional, secure web application with user-centric features.

CHAPTER 4

IMPLEMENTATION

The implementation phase of the "Secure Photo Vault" involves translating the design and functionalities into a fully operational web application. This stage focuses on the integration of advanced encryption techniques, such as ECC and AES, to ensure the secure storage and management of user images. Utilizing the Django framework, the application will facilitate user registration, two-factor authentication, and efficient image uploading and tagging through a user-friendly interface.

4.1 FRONTEND

The frontend of this project is built using HTML, CSS, and JavaScript to create an intuitive and responsive user interface. By focusing on simplicity and user experience, the design ensures seamless navigation and easy access to essential features like image upload, encryption, and face search. The frontend also supports various interactive elements that allow users to manage their images securely, maintaining high usability while integrating advanced security functionalities.

4.2 BACKEND

The backend of the project is developed using Django, a powerful Python web framework, to handle the core logic and processing. It manages API requests, user authentication, and data encryption, ensuring secure interactions with the database. The backend is responsible for key functionalities like image encryption and decryption, face recognition, and automatic image tagging. It also supports user authentication through two-factor authentication (2FA) and handles the secure storage of encrypted images. The use of Django provides a robust framework for maintaining security, scalability, and efficient data management, ensuring the seamless execution of the platform's features.

4.3 DATABASE

The database for the project is built using PostgreSQL, a powerful relational database system. PostgreSQL offers strong data integrity and security, making it ideal for storing sensitive information such as user profiles and authentication data. The database schema is structured to accommodate efficient storage and retrieval of data. Django's ORM seamlessly integrates with PostgreSQL, simplifying database interactions, ensuring data consistency, and supporting the platform's scalability as user data grows over time.

4.4 KEY COMPONENTS

The implementation includes several key components and modules to ensure the platform's functionality:

- **User Authentication and Authorization:** The system includes a robust user authentication mechanism with two-factor authentication (2FA), requiring users to verify their identity through an OTP sent to their registered email. This additional layer of security ensures that only authorized users can access the platform, helping safeguard user accounts from unauthorized access and account breaches.

- **Image Encryption and Decryption:** The platform employs advanced cryptographic techniques, utilizing Elliptic Curve Cryptography (ECC) for key generation and AES encryption for data encryption. Each image uploaded by the user is encrypted with a unique key pair generated through ECC, ensuring it is securely stored in an encrypted format. Decryption occurs when users wish to view or manage their images, where the platform uses the associated private key to decrypt the data. This secure encryption and decryption process guarantees the confidentiality and integrity of user images while stored on the platform.

- **Automatic Image Tagging:** A fine-tuned VGG16 deep learning model has been integrated into the system to automatically analyze and categorize uploaded images. The model assigns predefined tags such as *'fruit', 'animal', 'landscape', 'human'* and *'vehicle'* based on the

image's content, making it easier for users to search and organize their collections. The tagging process enhances efficiency by minimizing the need for manual sorting, especially when managing large libraries of images.

- **Face Recognition:** The platform features a sophisticated face recognition system that identifies and groups images containing faces. This feature allows users to quickly locate and organize images of specific individuals or groups, enhancing the user experience by enabling accurate access to desired images. By comparing face embeddings directly, the system achieves high accuracy without needing to store specific facial identities, thus maintaining user privacy.

- **Tag-based Image Sorting:** Users can utilize tags assigned to each image during the upload process to sort and filter their photo collections. This tag-based sorting feature enables efficient browsing, making it easy for users to locate images of a particular type or subject, such as landscapes or portraits. It improves usability, particularly for large image libraries, by providing a structured and intuitive way to access images.

- **Secure Image Storage:** All encrypted images are stored in designated directories that are accessible only through authorized requests. When users access or download an image, the platform creates a decrypted version in a temporary and secured location, maintaining separate directories for encrypted and decrypted files. This secure storage approach, combined with the encryption protocols, ensures that sensitive data, including encryption keys and image files, remain protected from unauthorized access or data breaches.

These components collectively contribute to the seamless functioning of the "Secure Photo Vault", providing an integrated system for secure image management, efficient user interaction, and robust data protection. Each element is designed to work in harmony, balancing user-friendliness with high levels of data security, ensuring a safe and enjoyable experience for the user.

4.5 TESTING AND DEBUGGING

Extensive testing has been conducted to ensure the reliability and security of the system. Unit tests were applied to validate individual components, including encryption, authentication, and image tagging features. Integration tests ensured seamless interaction between modules, verifying that encryption processes, face recognition, and tagging functionalities operate correctly within the larger system. Additionally, user acceptance testing was conducted with a sample group to gather feedback on usability and performance, leading to adjustments that enhance user experience and secure data handling before final deployment.

4.6 DEPLOYMENT

Following comprehensive testing, the system is deployed on a secure and reliable server, making the application accessible to users. The deployment process involves setting up the production environment, configuring the PostgreSQL database, and securing connections to protect data during transmission. All features, including encryption, tagging, and face recognition, are thoroughly verified in the production setup to ensure they function seamlessly. Documentation is provided to guide the users, offering instructions on managing accounts, uploading images, and using the platform's security features for a smooth and secure experience.

In this phase of the project brings together various components and technologies to build a functional, secure, and user-friendly platform for managing personal images. Through the integration of advanced encryption techniques like ECC and AES, robust two-factor authentication, and Django-based backend processing, the platform achieves high standards of data protection and secure user interactions. The frontend, crafted with HTML, CSS, and JavaScript, provides an intuitive interface that allows users to navigate easily and utilize features like image upload, tagging, and face recognition. The PostgreSQL database, managed through Django's ORM, supports efficient data storage and retrieval, ensuring data integrity and scalability. Extensive testing and debugging further solidify the platform's reliability, while secure deployment practices guarantee smooth and safe access for users. Next chapter presents the outcomes and evaluates the performance of project, focusing on its core functionalities, security, and user experience.

CHAPTER 5

RESULT AND ANALYSIS

This chapter presents the findings from the "Secure Photo Vault" system, focusing on the performance of core features like user authentication, image encryption/decryption, automatic tagging, and face-recognized sorting. The analysis evaluates these results in terms of security, scalability, and efficiency, offering insights into the system's effectiveness and areas for potential improvement.

5.1 RESULT

Testing of the project confirms successful implementation of key features, including secure authentication, encryption, automatic tagging, and face-recognized sorting. The system effectively protects data while enabling efficient organization and retrieval of images, demonstrating reliability in data security and usability.

Figure 5.1: Gallery

The gallery shown in Fig. 5.1 is a feature in the project provides an organized, user-friendly interface, allowing users to easily browse, filter, and locate photos based on tags like 'animal,' 'landscape,' 'human,' and 'vehicle,' or through face recognition. This structured layout enhances user experience by streamlining access to specific images. Images are encrypted upon upload to protect against unauthorized access, and are seamlessly decrypted for viewing in gallery.

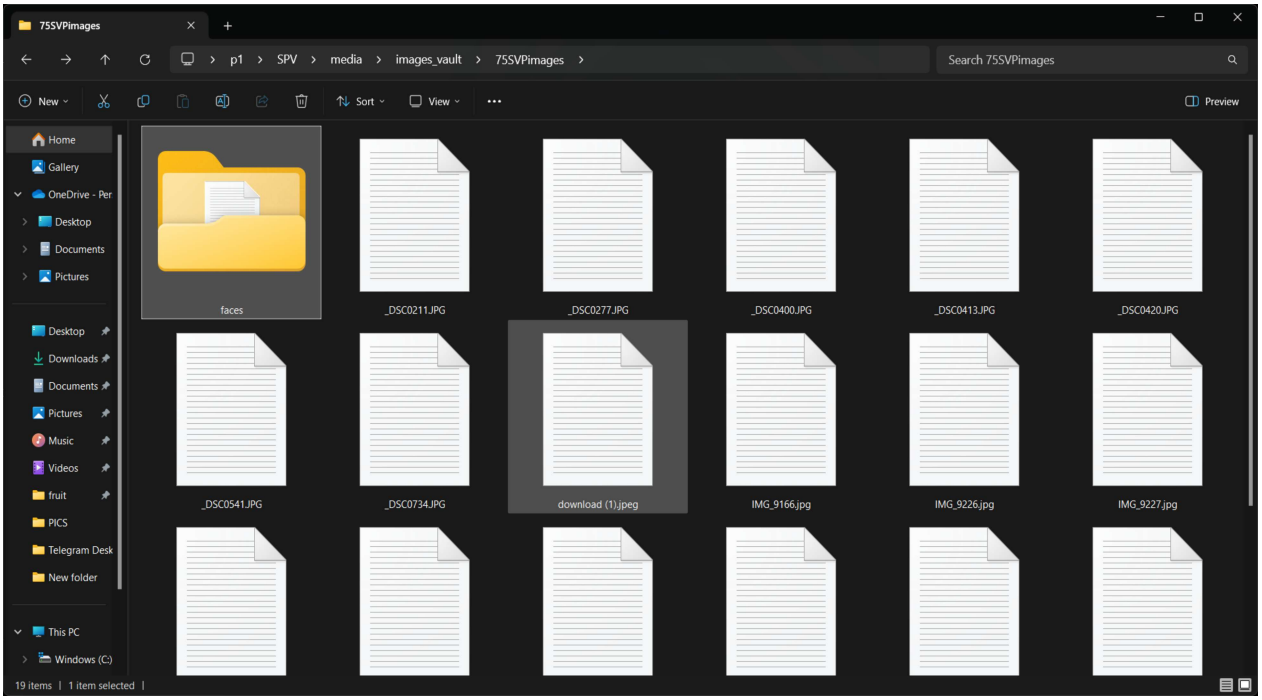

Figure 5.2: Encrypted Image Files

In this system, images are encrypted immediately upon upload to safeguard against unauthorized access. and protect sensitive data. The system employs elliptic curve cryptography for secure key exchange and Advanced Encryption Standard for robust encryption of images. Encrypted images are stored in a secure ".bin" file format Fig. 5.2, accessible only through decryption with the correct keys. This approach enables users to store their sensitive images confidently.

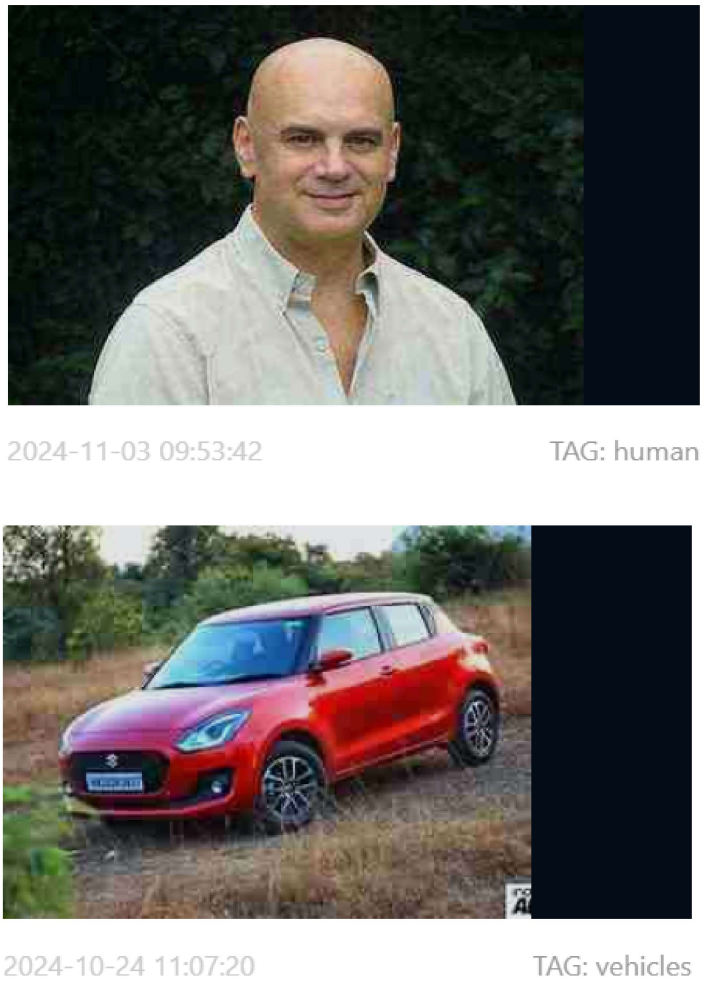

Figure 5.3: Auto Tagging

The auto-tagging Fig. 5.3 feature in the project enhances image organization by categorizing photos based on predefined tags. Using a fine-tuned VGG16 model, the system accurately assigns tags to each uploaded image, streamlining search and retrieval. This automated process reduces manual effort, allowing users to quickly locate specific types of photos, thereby improving overall usability and efficiency of the platform.

Figure 5.4: Face Sorted Images

Face-based sorting Fig. 5.4 feature in the project enables efficient organization of images containing human faces. Using face recognition technology, the system identifies and groups similar faces, allowing users to quickly locate photos with specific individuals without manually tagging each image. This automated sorting not only enhances search operation but also simplifies the photo management process, making it easier for users to access images based on face-based groupings. This feature further enhances the usability and personalization of the gallery.

5.2 ANALYSIS

The results of the project demonstrate significant improvements in image management, security, and user experience. Here's a closer look at the analysis of these results:

- **Enhanced Security:** The platform incorporates advanced encryption techniques, including Elliptic Curve Cryptography (ECC) and Advanced Encryption Standard (AES), alongside two-factor authentication (2FA) to ensure that only authorized users can access their images. This multi-layered security approach mitigates the risk of unauthorized access, data breaches, and identity theft. By prioritizing data protection at every stage, from upload to storage and retrieval, the system builds trust among users and aligns with modern security best practices.
- **Efficient Image Management:** Automatic image tagging and face recognition significantly enhance image organization and retrieval. The system uses a fine-tuned VGG16 model to categorize images with tags such as *'animal', 'landscape', 'human'*, and more. Face recognition adds another layer of organization, enabling users to search and sort images based on identified faces. This comprehensive approach to image management improves the search efficiency, making it easier for users to locate specific photos within large libraries, and ultimately enhances the overall user experience.

- **Scalability and Performance:** Designed with scalability in mind, the system is capable of handling increased data volumes and a growing number of users without sacrificing performance. The underlying architecture is optimized to support high loads and ensure quick response times, even as the platform expands. This scalability ensures that the platform can accommodate future features and enhancements, such as additional tagging categories or more advanced image processing capabilities, without compromising system efficiency or user satisfaction.

- **User-Friendly Interface:** The platform provides a clean, intuitive, and responsive user interface that facilitates easy navigation through essential features like image uploading, tagging, and encryption. The frontend design, implemented using modern web development frameworks, ensures a smooth user experience across different devices, enabling accessibility for users of varying technical abilities. This design approach prioritizes simplicity without sacrificing functionality, making advanced security features accessible to all users, regardless of their technical background.

- **Reliable Data Integrity:** The platform's secure image storage strategy emphasizes data integrity, ensuring that both image files and associated metadata are preserved accurately throughout all operations. By securely handling encryption keys, image metadata, and user data, the system protects against data corruption and unauthorized alterations. This level of data integrity reassures users that their images remain safe, unaltered, and accessible as intended, reinforcing confidence in the platform's reliability.

- **Optimized Resource Utilization:** The platform employs efficient resource management techniques to ensure optimal use of memory, processing power, and storage space. This is especially relevant for high-resolution images and complex encryption processes, where efficient use of resources is critical to maintaining performance. By balancing computational demands with system responsiveness, the platform can provide a seamless experience even for resource-intensive operations, such as large-scale image uploads and face recognition tasks.

- **Comprehensive Logging and Monitoring:** Although the platform does not implement log-based security monitoring, it incorporates basic logging mechanisms for tracking key system events and processes. These logs help in identifying any issues during encryption, decryption, and tagging processes, allowing for quick troubleshooting and enhancing the reliability of the system.

This thorough analysis illustrates how the "Secure Photo Vault" achieves a balance between security, functionality, and user convenience, demonstrating its potential as a secure, scalable, and user-friendly platform for managing personal image libraries.Next chapter reviews the platform's success in providing a secure image management solution and explores potential enhancements for future development.

CHAPTER 6

CONCLUSION AND FUTURE SCOPE

This chapter, we review how the platform successfully fulfills the need for a secure and user-friendly image management solution, combining advanced encryption techniques and authentication mechanisms with features that streamline photo organization. Additionally, we discuss possible improvements that could further elevate the platform's capabilities. That aim to refine the user experience and strengthen the platform's security, making the "Secure Photo Vault" an even more versatile and robust tool for personal media management.

6.1 CONCLUSION

The "Secure Photo Vault" project successfully meets the demand for a secure and efficient platform to manage personal images, focusing on privacy and data protection. By implementing advanced encryption techniques, such as ECC and AES, alongside two-factor authentication (2FA), the platform ensures robust data security and strict access control, protecting user images from unauthorized access. Secure storage solutions further reinforce this protection, creating a multi-layered approach to safeguard sensitive data from potential threats, while maintaining the confidentiality and integrity of user content.

In addition to its strong security measures, the platform includes user-friendly features like automatic image tagging, face recognition, and tag-based sorting. These functionalities make it easy for users to organize, search, and retrieve images quickly, even in large collections. The intuitive interface promotes a smooth user experience, while the system's scalability ensures it can handle increasing amounts of data without compromising performance. By balancing data security with usability, The "Secure Photo Vault" provides a comprehensive, scalable, and intuitive solution for managing personal photo collections, making it a valuable tool in an era where privacy and data management are essential.

6.2 FUTURE SCOPE

Several enhancements could further improve the functionality and user experience of the project:

- **Biometric Authentication:** Explore the integration of biometric authentication methods, such as fingerprint or facial recognition, to add an extra layer of security. This would ensure that only authorized users can access their data, enhancing the platform's overall protection.
- **Support for Additional Media Formats:** Extend the platform's capabilities to support other media types, such as video. This enhancement would enable users to securely store, organize, and manage various media files beyond just images.
- **Optimized Processing for Core Functions:** Refine encryption, face detection, and sorting algorithms to improve processing speed and reduce response times. This optimization would make the platform more efficient, particularly for users managing large collections.
- **Enhanced Face Recognition Accuracy:** Optimizing the face recognition model to support larger datasets and improve recognition speed and accuracy.

REFERENCES

[1] Rameela Ravindran K and Silpa K. S, *Image Encryption using Elliptic Curve Cryptography with Data Security*, International Journal of Engineering Research and Technology (IJERT), ISSN: 2278-0181, Vol. 11 Issue 05, May-2022.

[2] Django Documentation, `https://docs.djangoproject.com/en/5.1/`, Accessed on: October 27, 2024.

[3] PostgreSQL Documentation, `https://www.postgresql.org/docs/`, Accessed on: October 27, 2024.

[4] OpenCV Documentation, `https://docs.opencv.org/4.x/index.html`, Accessed on: October 27, 2024.

APPENDIX

SAMPLE CODE

Code for encrypt and decrypt images

```
from . csvfile  import  csv_access
import  os
import  csv
from datetime  import  datetime
from cryptography.hazmat. primitives  import  hashes,  serialization
from cryptography.hazmat. primitives .asymmetric import  ec
from cryptography.hazmat. primitives .kdf.pbkdf2 import  PBKDF2HMAC
from cryptography.hazmat. primitives . ciphers  import  Cipher,  algorithms,  modes
from cryptography.hazmat.backends import  default_backend
from cryptography.hazmat. primitives  import  padding
from django.conf  import  settings
from django. shortcuts  import  render
from django. http  import  HttpResponseBadRequest
from django. contrib  import  messages
from . face_detection  import  detect_and_crop_faces
from .tag  import  get_image_tag
from io  import  BytesIO




def  upload( request ):
    user_id  =  request . session . get (' user_id ')

    if  not  user_id :
        return  HttpResponseBadRequest("User not  authenticated ")

    if  request .method == 'POST':
```

```
images = request.FILES.getlist('images')
if not images:
    messages.error(request, 'No images were uploaded.')
    return render(request, 'upload.html')

user_images_dir = os.path.join(settings.IMAGES_VAULT, f'{user_id}SVPimages')
os.makedirs(user_images_dir, exist_ok=True)

face_images_dir = os.path.join(user_images_dir, 'faces')
os.makedirs(face_images_dir, exist_ok=True)

csv_file_path = os.path.join(settings.META_DATA, f'SPV{user_id}.csv')

try:
    with open(csv_file_path, mode='a', newline='') as csvfile:
        fieldnames = ['image_name', 'public_key', 'private_key', 'tags', 'date']
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)

        for image in images:
            image_name = os.path.basename(image.name)
            image_data = image.read()

            # Generate ECC key pair
            private_key = ec.generate_private_key(ec.SECP384R1(), default_backend())
            public_key = private_key.public_key()

            public_key_bytes = public_key.public_bytes(
                encoding=serialization.Encoding.PEM,
                format=serialization.PublicFormat.SubjectPublicKeyInfo
            )

            shared_key = private_key.exchange(ec.ECDH(), public_key)
```

```
salt = os.urandom(16)
kdf = PBKDF2HMAC(
    algorithm=hashes.SHA256(),
    length=32,
    salt=salt,
    iterations=100000,
    backend=default_backend()
)
symmetric_key = kdf.derive(shared_key)

iv = os.urandom(16)
cipher = Cipher(algorithms.AES(symmetric_key), modes.CBC(iv),
    backend=default_backend())

# Encrypt main image
main_padder = padding.PKCS7(algorithms.AES.block_size).padder()
padded_image_data = main_padder.update(image_data) + main_padder.finalize()

main_encryptor = cipher.encryptor()
ciphertext = main_encryptor.update(padded_image_data) +
    main_encryptor.finalize()

encrypted_image_path = os.path.join(user_images_dir, f'{image_name}.bin')
with open(encrypted_image_path, 'wb') as f:
    f.write(salt + iv + ciphertext)

private_key_bytes = private_key.private_bytes(
    encoding=serialization.Encoding.PEM,
    format=serialization.PrivateFormat.PKCS8,
    encryption_algorithm=serialization.NoEncryption()
)
```

```
auto_tag = get_image_tag(image)

image_metadata = {
    'image_name': f'{image_name}.bin',
    'public_key': public_key_bytes.decode('utf-8'),
    'private_key': private_key_bytes.decode('utf-8'),
    'tags': auto_tag,
    'date': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
}
writer.writerow(image_metadata)

# Detect faces and crop face_detection.py
face_images = detect_and_crop_faces(image_data)
for i, face_image in enumerate(face_images):
    face_image_name = f"{image_name}_face_{i}.bin"
    face_encrypted_image_path = os.path.join(face_images_dir,
        face_image_name)

    # Encrypt each cropped face image separately
    face_padder = padding.PKCS7(algorithms.AES.block_size).padder()
    face_padded_data = face_padder.update(face_image) +
        face_padder.finalize()

    face_encryptor = cipher.encryptor()
    face_ciphertext = face_encryptor.update(face_padded_data) +
        face_encryptor.finalize()

    with open(face_encrypted_image_path, 'wb') as f:
        f.write(salt + iv + face_ciphertext)

    face_metadata = {
```

```
                    'image_name': face_image_name,
                    'public_key': public_key_bytes.decode('utf-8'),
                    'private_key': private_key_bytes.decode('utf-8'),
                    'tags': 'face',
                    'date': datetime.now().strftime('%Y-%m-%d %H:%M:%S')
                }
                writer.writerow(face_metadata)

            messages.success(request, 'Images uploaded and encrypted successfully with face
                images extracted!')

        except IOError as e:
            messages.error(request, f"An error occurred while processing the upload: {str(e)}")
            return render(request, 'upload.html')

    return render(request, 'upload.html')


def decrypt_image(user_id, image_name, private_key_pem, public_key_pem):
    encrypted_image_path = os.path.join(settings.MEDIA_ROOT, 'images_vault',
        f'{user_id}SVPimages', image_name)

    decrypted_image_name = os.path.splitext(image_name)[0] # Remove file extension
    decrypted_image_path = os.path.join(settings.MEDIA_ROOT, 'images_vault',
        f'{user_id}SVPimages', 'decrypted', decrypted_image_name)
    decrypted_face_path = os.path.join(settings.MEDIA_ROOT, 'images_vault',
        f'{user_id}SVPimages', 'decrypted', 'faces')

    print(user_id, image_name, private_key_pem, public_key_pem)

    # Load the private key from the PEM string
    private_key = serialization.load_pem_private_key(
```

```
        private_key_pem.encode('utf-8'),
        password=None,
        backend=default_backend()
    )

    # Load the public key from the PEM string
    public_key = serialization.load_pem_public_key(
        public_key_pem.encode('utf-8'),
        backend=default_backend()
    )

    # Load the encrypted image
    with open(encrypted_image_path, 'rb') as f:
        salt = f.read(16)
        iv = f.read(16)
        ciphertext = f.read()

    # Generate the shared key using ECC
    shared_key = private_key.exchange(ec.ECDH(), public_key)

    # Derive the symmetric key from the shared key using PBKDF2
    kdf = PBKDF2HMAC(
        algorithm=hashes.SHA256(),
        length=32,
        salt=salt,
        iterations=100000,
        backend=default_backend()
    )
    symmetric_key = kdf.derive(shared_key)

    # Decrypt the image using AES
    cipher = Cipher(algorithms.AES(symmetric_key), modes.CBC(iv), backend=default_backend())
```

```
decryptor = cipher.decryptor()

try:
    # Decrypt the data and remove padding
    padded_image_data = decryptor.update(ciphertext) + decryptor.finalize()
    unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
    image_data = unpadder.update(padded_image_data) + unpadder.finalize()
except ValueError as e:
    print("Decryption failed: Invalid padding or corrupted ciphertext.")
    print(f"Error details: {e}")
    return

# Ensure the decrypted directory exists
os.makedirs(os.path.dirname(decrypted_image_path), exist_ok=True)
os.makedirs(decrypted_face_path, exist_ok=True) # Ensure the face directory exists

# Save the decrypted main image
with open(decrypted_image_path, 'wb') as f:
    f.write(image_data)

print(f"Decrypted image saved to: {decrypted_image_path}")

# Decrypt corresponding face images
face_images_dir = os.path.join(settings.MEDIA_ROOT, 'images_vault', f'{user_id}SVPimages',
    'faces')
face_images = [f for f in os.listdir(face_images_dir) if
    f.startswith(decrypted_image_name)]

for i, face_image in enumerate(face_images):
    encrypted_face_path = os.path.join(face_images_dir, face_image)
    decrypted_face_image_path = os.path.join(decrypted_face_path,
        f"{decrypted_image_name}_face_{i}.png") # Change the extension to your desired
```

```
    format

with open(encrypted_face_path, 'rb') as f:
    salt = f.read(16)
    iv = f.read(16)
    ciphertext = f.read()

cipher = Cipher(algorithms.AES(symmetric_key), modes.CBC(iv),
    backend=default_backend())
decryptor = cipher.decryptor()

try:
    padded_face_data = decryptor.update(ciphertext) + decryptor.finalize()
    unpadder = padding.PKCS7(algorithms.AES.block_size).unpadder()
    face_data = unpadder.update(padded_face_data) + unpadder.finalize()

    # Use PIL to save the face image in a specific format
    image = Image.open(io.BytesIO(face_data))
    image.save(decrypted_face_image_path, format='PNG') # Save as PNG or specify any
        other format

    print(f"Decrypted face image saved to: {decrypted_face_image_path}")

except ValueError as e:
    print(f"Decryption of face {face_image} failed: {e}")
```

USER INTERFACE SCREENSHOTS

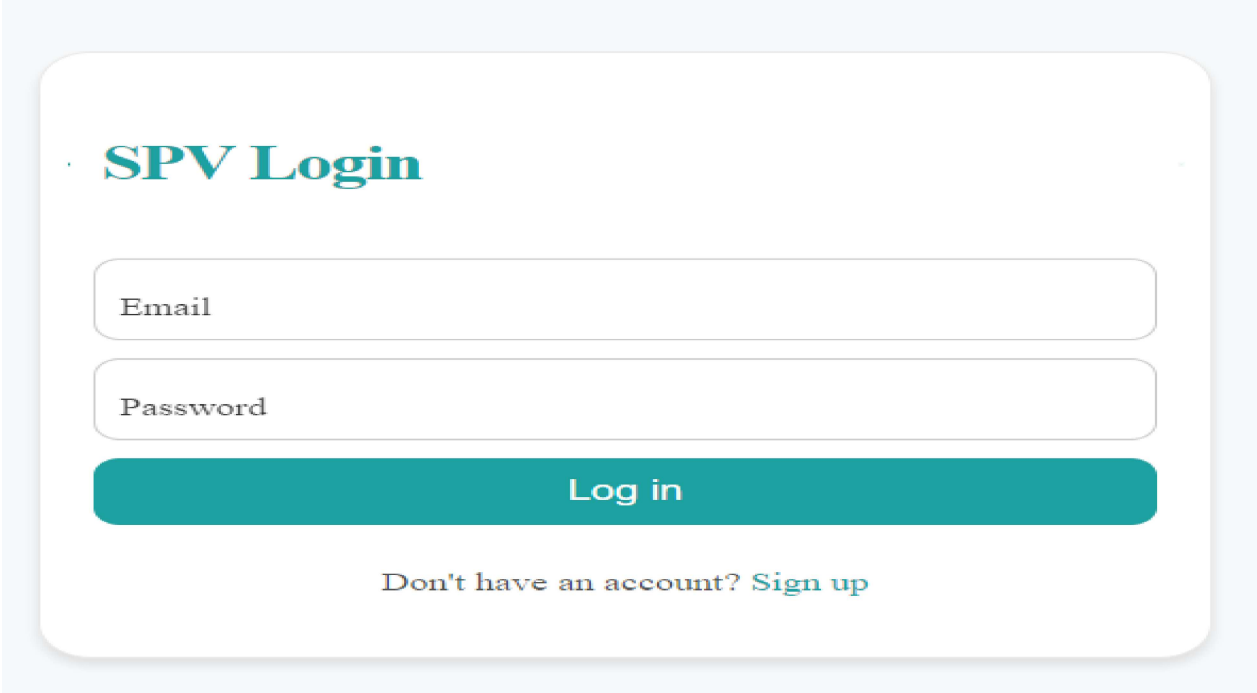

Figure 6.1: Login Page

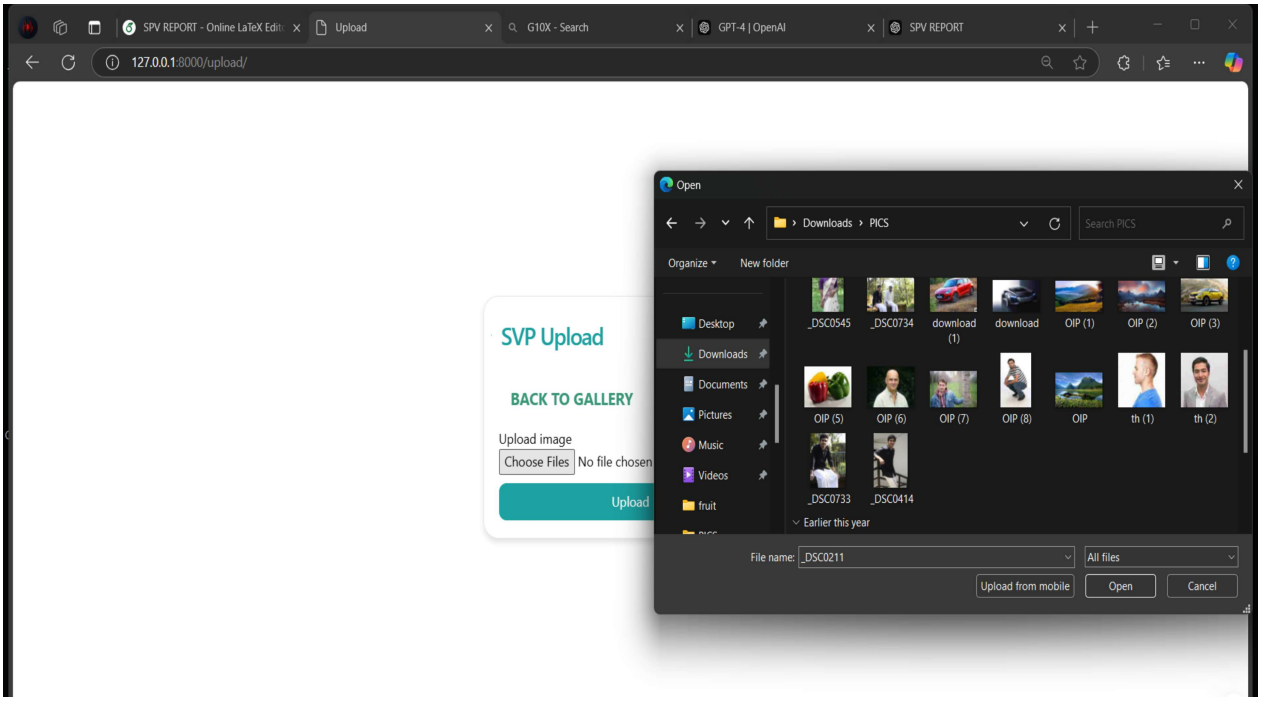

Figure 6.2: Upload Page

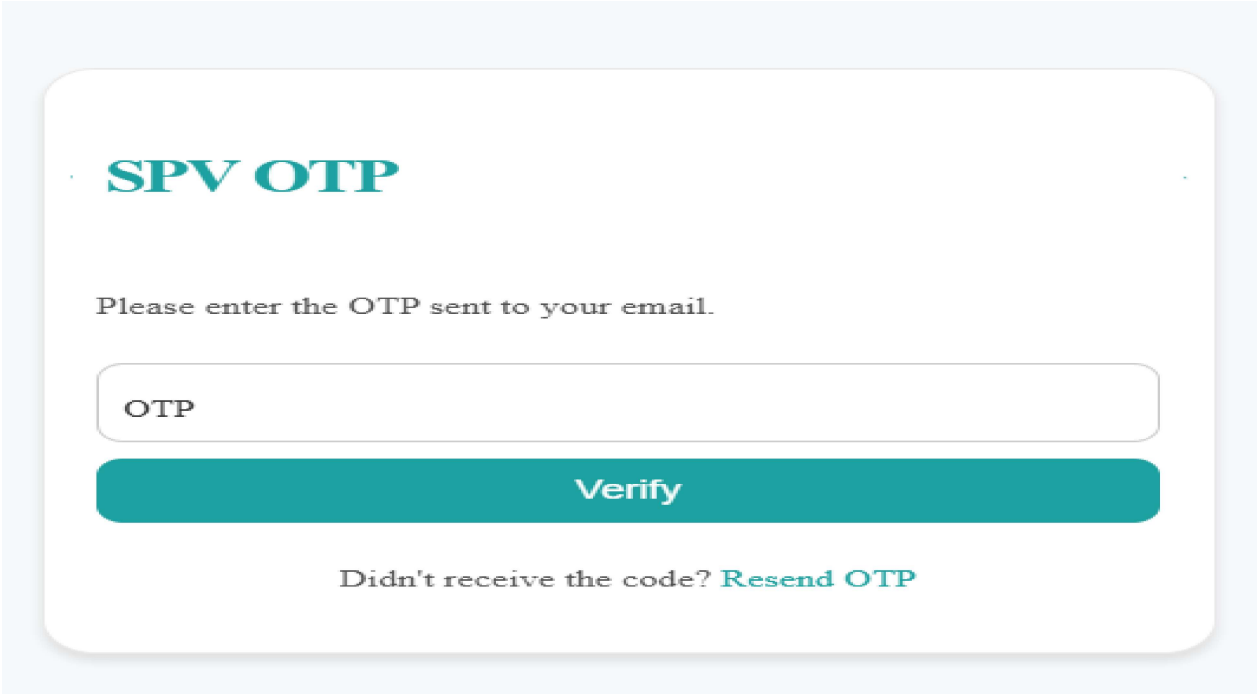

Figure 6.3: OTP Verification Page

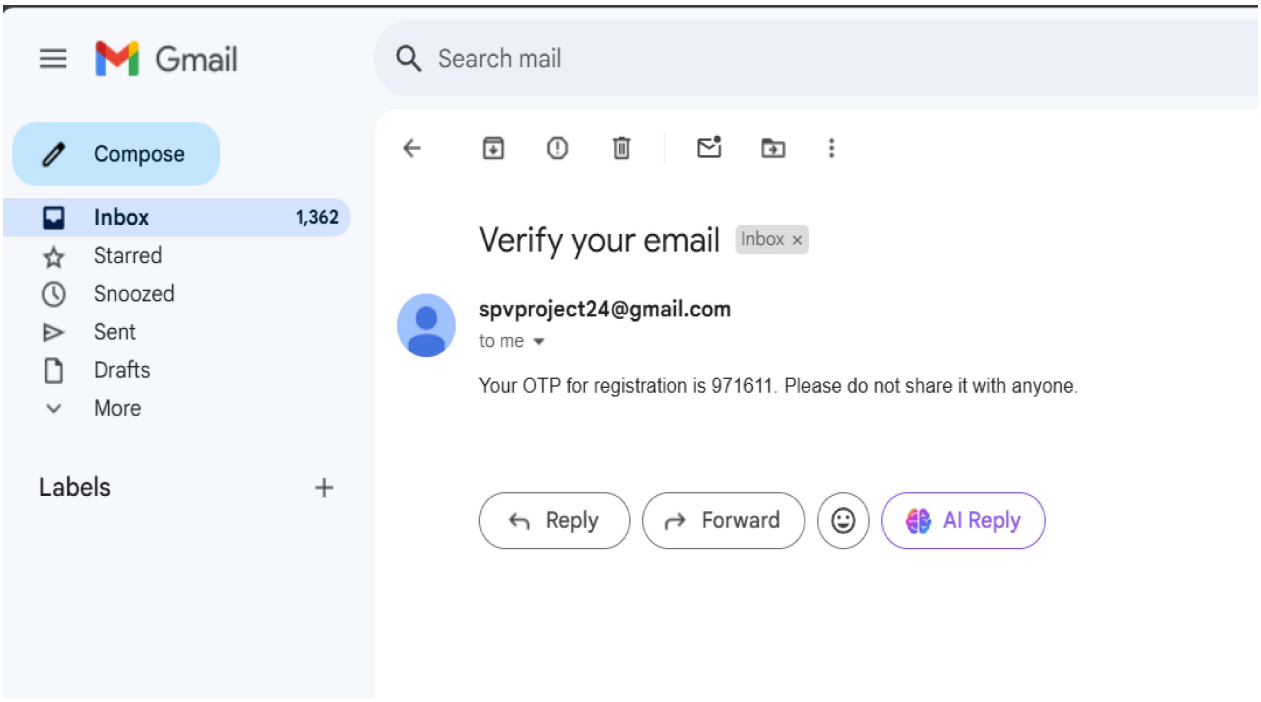

Figure 6.4: Email with OTP

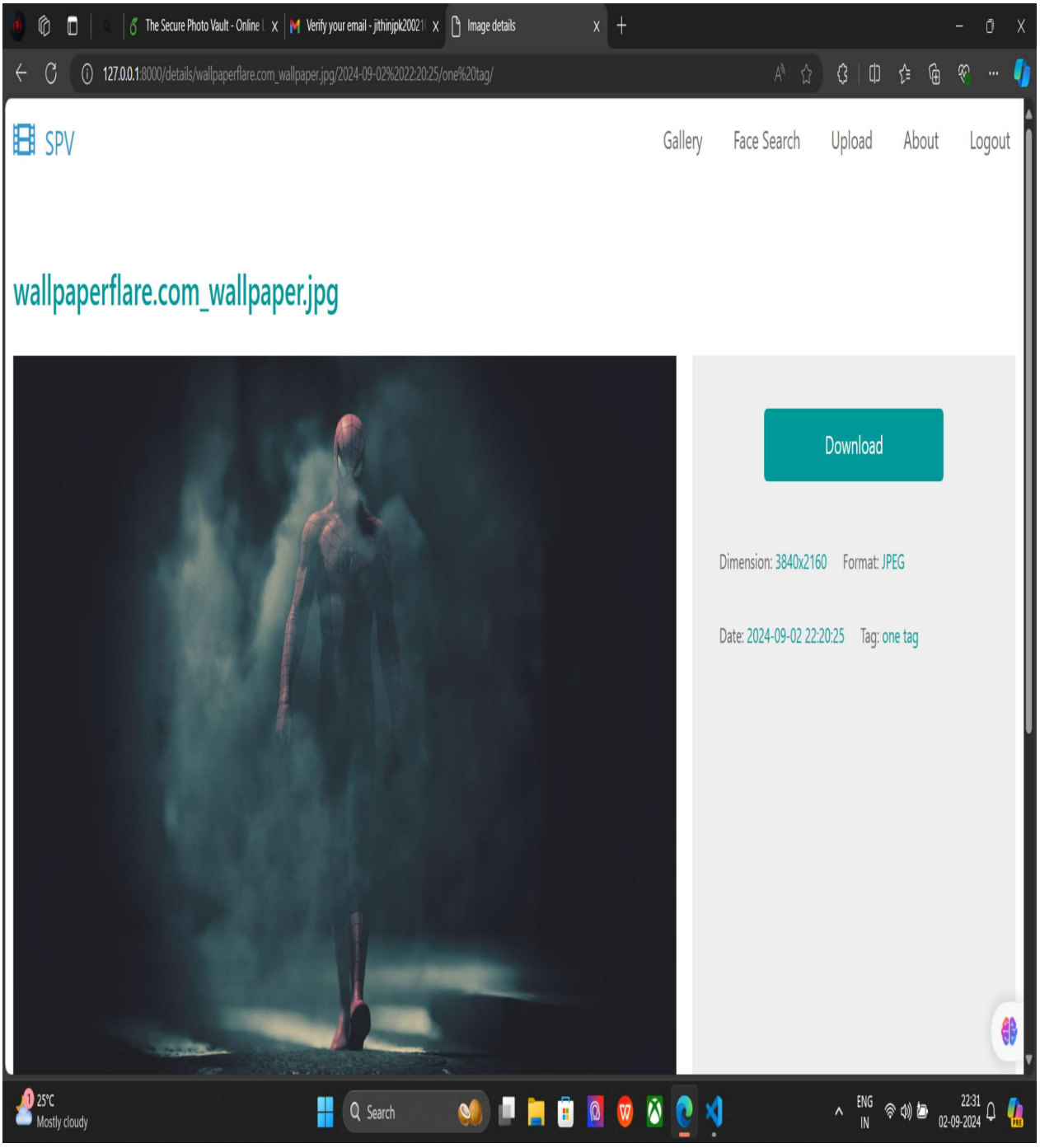

Figure 6.5: Image Details

GIT COMMITS

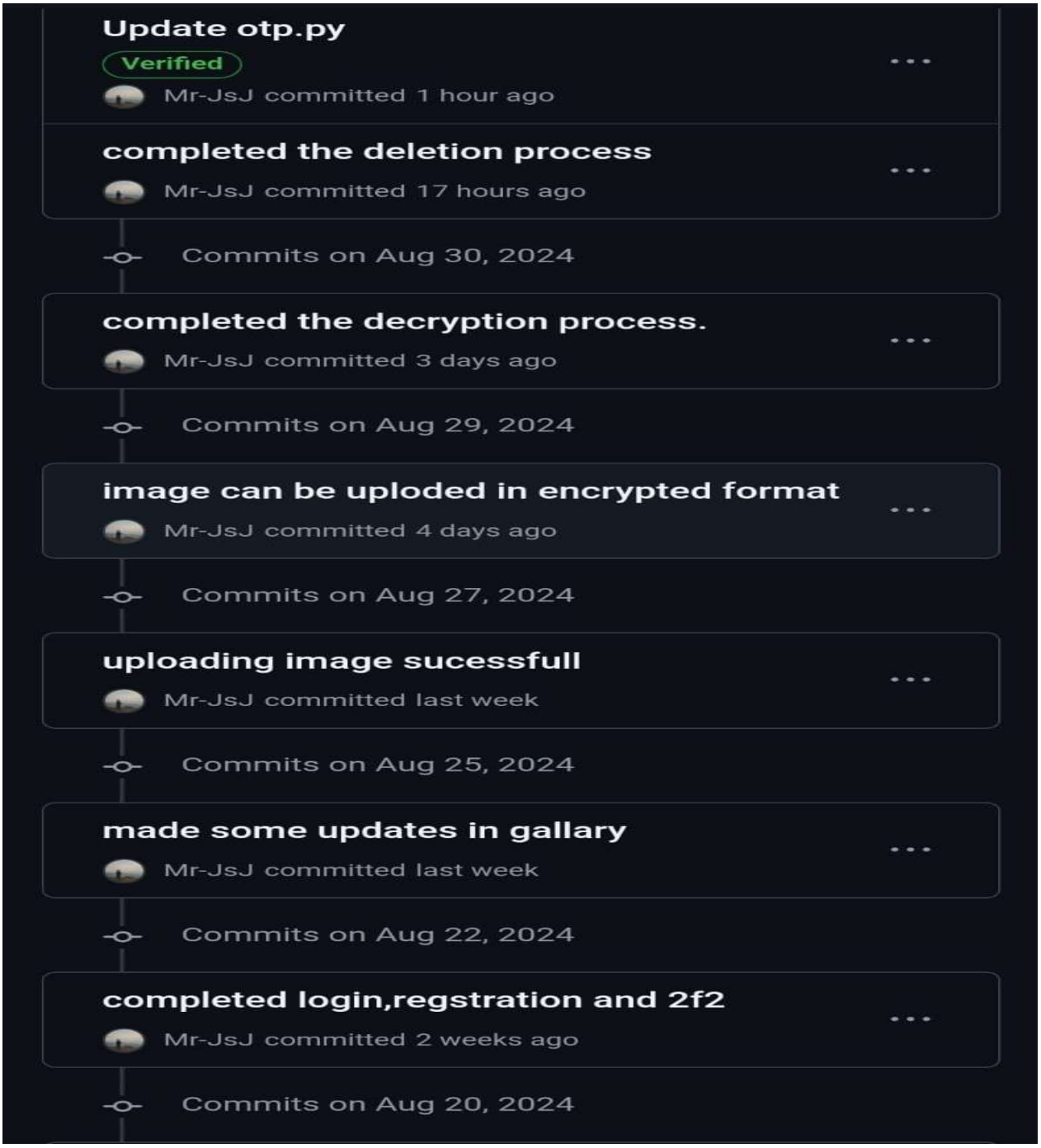

Figure 6.6: Git Commits

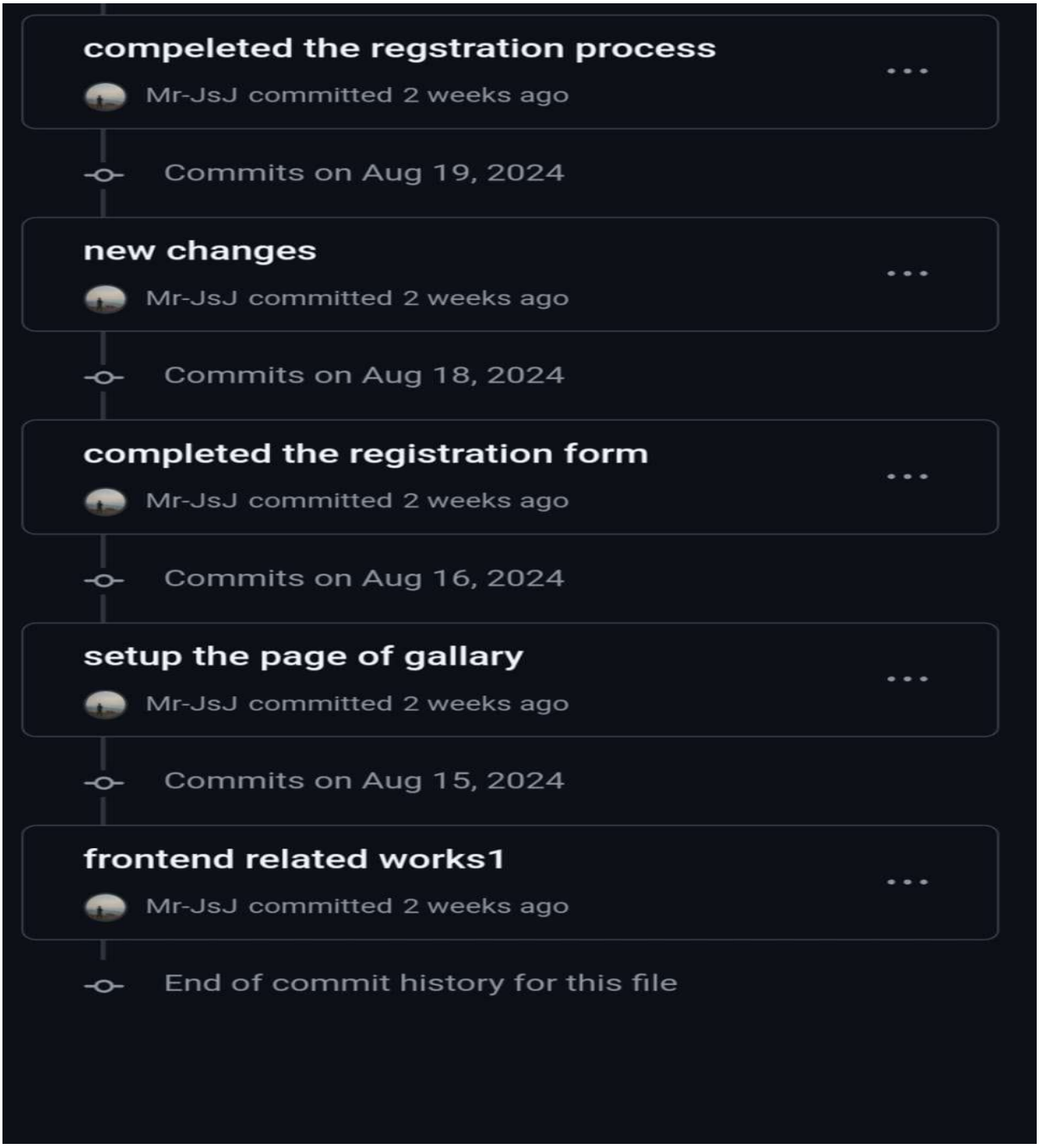

Figure 6.7: Git Commits