CryptoCamGuard-Elevating Image Security

Final Year Project Report

Submitted by

Sardar Nazeer (2630-2021)
Ali Sher Siyal (2201-2021)

Supervisor

Mr. Saifullah Adnan

In partial fulfilment of the requirements for the degree of
Bachelor of Science in Software Engineering
2025

Faculty of Engineering Sciences and Technology

Hamdard Institute of Engineering and Technology

Hamdard University, Main Campus, Karachi, Pakistan

Certificate of Approval

Faculty of Engineering Sciences and Technology

Hamdard Institute of Engineering and Technology
Hamdard University, Karachi, Pakistan

This project "**CryptoCamGuard – Elevating Image Security App**" is presented by Sardar Nazeer, Ali Sher Siyal under the supervision of their project advisor and approved by the project examination committee, and acknowledged by the Hamdard Institute of Engineering and Technology, in the fulfillment of the requirements for the Bachelor degree of Software Engineering.

_____
Mr. Saifullah Adnan
(Project Supervisor)

_____
In-charge FYP-Committee

_____
(Project Co-Supervisor)

_____
Chairman
(Department of Computing)

_____
(Dean, FEST)

Authors' Declaration

We declare that this project report was carried out in accordance with the rules and regulations of Hamdard University. The work is original except where indicated by special references in the text and no part of the report has been submitted for any other degree. The report has not been presented to any other University for examination.

Dated:

Authors Signatures:

_____

Sardar Nazeer

_____

Ali Sher Siyal

Plagiarism Undertaking

We, Sardar Nazeer, Ali Sher Siyal solemnly declare that the work presented in the Final Year Project Report titled **CryptoCamGuard – Elevating Image Security** has been carried out solely by ourselves with no significant help from any other person except few of those which are duly acknowledged. We confirm that no portion of our report has been plagiarized and any material used in the report from other sources is properly referenced.

Dated:

Authors Signatures:

_____

Sardar Nazeer

_____

Ali Sher Siyal

Acknowledgments

We are deeply grateful to our supervisor, **Mr. Saifullah Adnan**, for his invaluable guidance, support, and encouragement throughout this project. We also extend our gratitude to the faculty members of **Hamdard Institute of Engineering and Technology** for their continuous support. Finally, we would like to thank our families and friends for their patience and encouragement during this journey.

Document Information

Table 1: Document Information

Customer	
Project Title	CryptoCamGuard – Elevating Image Security
Document	Final Year Project Report
Document Version	1.0
Identifier	FYP-008/FL24 Final Report
Status	Final
Author(s)	Sardar Nazeer, Ali Sher Siyal
Approver(s)	Saifullah Adnan
Issue Date	

Definition of Terms, Acronyms, and Abbreviations

This section should provide the definitions of all terms, acronyms, and abbreviations required to interpret the terms used in the document properly.

Table 2: Definition of Terms, Acronyms, and Abbreviations

AES	Advanced Encryption Standard
FYP	Final Year Project
UI/UX	User Interface / User Experience
SDK	Software Development Kit
API	Application Programming Interface
DOC	Document
PDF	Portable Document Format
DB	Database

Abstract

The widespread use of digital images in modern communication and sharing platforms often exposes users to privacy and security risks. Unauthorized access, data breaches, and misuse of sensitive images are becoming increasingly prevalent. To address these challenges, **CryptoCamGuard** provides a robust mobile application that ensures the security of images through **AES-256 encryption**, secure sharing with customizable access controls, and real-time monitoring of image usage. This app prioritizes user-friendliness, scalability, and high-level security to redefine how images are shared and protected in a digital ecosystem.

Keywords: Image Security, AES-256 Encryption, Secure Sharing, Digital Privacy

Table of Contents

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

1.1 Motivation

The CryptoCamGuard project is an innovative mobile application designed to secure personal photos through advanced encryption techniques. This application empowers users to take control of their digital privacy by providing a safe and private platform to capture, encrypt, and store images. By prioritizing security, user-friendliness, and performance, the project addresses growing concerns about unauthorized access, privacy breaches, and image theft in the digital age.

1.2 Problem Statement

In the digital era, mobile devices have become an integral part of everyday life, serving as repositories for personal memories and sensitive information. Among the most frequently stored data are images, which often include private moments, family photos, and confidential documents. Unfortunately, the security measures in place for managing these images on mobile devices are inadequate.

1.3 Goals and Objectives

The primary objective of CryptoCamGuard is to provide users with a secure platform for capturing, encrypting, and storing personal images, thereby ensuring their privacy and protection from unauthorized access. The application aims to streamline the photo management process while empowering users to maintain full control over their digital memories.

1.4 Project Scope

The following significant elements will be a part of the project.

- **In Scope**:

    1. **Advanced Encryption:** Secures images upon capture with AES encryption algorithms.
    2. **User-Friendly Interface:** Intuitive design for easy navigation by all users.
    3. **Secure Image Storage:** Safely stores images within the app, away from traditional galleries.
    4. **Authentication Mechanisms:** Ensures only authorized users can access their images.
    5. **Optimized Performance:** Fast and reliable handling of photos for efficient uploads and downloads.

CHAPTER 2

RELEVANT BACKGROUND & DEFINITIONS

Traditional photo management solutions, such as mobile galleries or cloud services, do not offer sufficient protection against these risks. Many of these platforms store images in unencrypted formats, making them easily accessible to unauthorized users if a device is lost or hacked. Even when encryption is applied, it is often implemented only during data transmission, while the images themselves remain exposed during storage. This lack of comprehensive security measures has led to a growing demand for more secure solutions that can guarantee the privacy and protection of personal images. In response to these challenges, CryptoCamGuard was conceived as a mobile application that offers a secure platform for capturing and storing images. By employing advanced encryption techniques, the app ensures that personal photos are not only protected during transmission but also stored in a fully encrypted format. Unlike traditional solutions, CryptoCamGuard enables users to take photos directly through the app and encrypt them immediately, ensuring that these images remain secure throughout their lifecycle.

MANUAL PROCUREMENT: Manual procurement refers to the traditional process of sourcing goods and services through manual paperwork, phone calls, or in-person meetings. This method is often inefficient, time consuming, and prone to errors due to a lack of digital automation.

LIMITED TRANSPARENCY: In conventional image storage solutions, users often face limited transparency regarding who can access their photos and how their data is managed. This lack of visibility increases the risk of unauthorized access and data misuse, as users are unaware of potential privacy breaches.

INEFFICIENCY: Traditional image management methods often involve manual processes that are time consuming and prone to errors, leading to delays in accessing or organizing photos. This inefficiency can frustrate users, making it difficult to manage and retrieve their personal images quickly and securely.

Chapter 3: Literature Review and Related Work

Introduction

The digital age has brought forth a surge in image sharing and online data storage. While this has improved convenience, it has also exposed users to significant risks regarding data privacy and security. The need for secure image management systems has never been more urgent, especially with the rise of hacking incidents, data leaks, and unauthorized access to private information.

This chapter explores existing research, technologies, and applications relevant to CryptoCamGuard. It also identifies gaps in current systems and justifies the need for a secure, performance-optimized, and user-friendly solution.

Related Technologies and Security Protocols

AES (Advanced Encryption Standard)

AES is a widely adopted symmetric encryption algorithm recognized for its speed and security. AES-256, used in CryptoCamGuard, provides robust protection for digital data and is recommended by NIST for encrypting sensitive information. It offers a strong security foundation against brute-force and dictionary attacks.

JWT (JSON Web Tokens)

JWT is an open standard used for securely transmitting information between parties. In CryptoCamGuard, JWT is used for user session management, ensuring secure and stateless authentication.

HTTPS & TLS

To ensure secure communication between the client and server, CryptoCamGuard uses HTTPS protocol layered with TLS (Transport Layer Security). This prevents data tampering and eavesdropping during image metadata transmission.

Review of Existing Applications

SecureCam (2020)

A mobile app focused on live camera feeds with basic encryption features. However, it lacked strong local storage encryption and did not support offline functionality.

CrypPic (2022)

An image encryption app that used AES but lacked user experience enhancements. It had performance issues on older devices and didn't offer responsive design.

Tresorit

A cloud-based file encryption platform that provides end-to-end security. While powerful, it is enterprise-focused and not tailored to image-specific mobile experiences.

PixelKnot

An Android app that uses steganography to hide messages within images. While creative, it is intended for message secrecy, not full image encryption or secure storage.

Gap Analysis

After reviewing several existing solutions, we observed key shortcomings. Apps like SecureCam and CrypPic offer encryption features but either lack a proper user experience or fail to store images securely. Tresorit is focused on enterprise-level cloud encryption and doesn't cater specifically to mobile image protection. PixelKnot offers a different concept (steganography) but lacks full encryption and storage functionality.

In contrast, CryptoCamGuard stands out by combining multiple essential features — including AES-256 encryption, offline functionality, secure local storage, optimized performance, and a clean Figma-based UI — in one mobile application. It also introduces an admin dashboard for monitoring, which is missing in all compared solutions.

These comparisons clearly show the gap in current offerings, highlighting the need for an all-in-one secure image application like CryptoCamGuard. either focus on one aspect (e.g., encryption or cloud sync) but fail to offer a holistic, user-centered, and secure environment like CryptoCamGuard does.

Research Insights

Recent academic research has stressed the importance of local image encryption to prevent data interception. Studies also highlight the role of usability in encouraging users to adopt secure apps. CryptoCamGuard bridges this divide by integrating strong encryption with a seamless UI.

A review of NIST guidelines for mobile data protection further validated our choice of AES-256 and secure local storage over reliance on cloud-based options.

Summary

The literature and tools reviewed demonstrate that while partial solutions exist, a gap remains in providing an all-in-one secure image management tool for everyday users. CryptoCamGuard emerges as a unique solution that combines military-grade encryption, intuitive design, and robust backend architecture.

Chapter 4: Project Discussion

Methodology

PROJECT DESIGN AND ARCHITECTURE

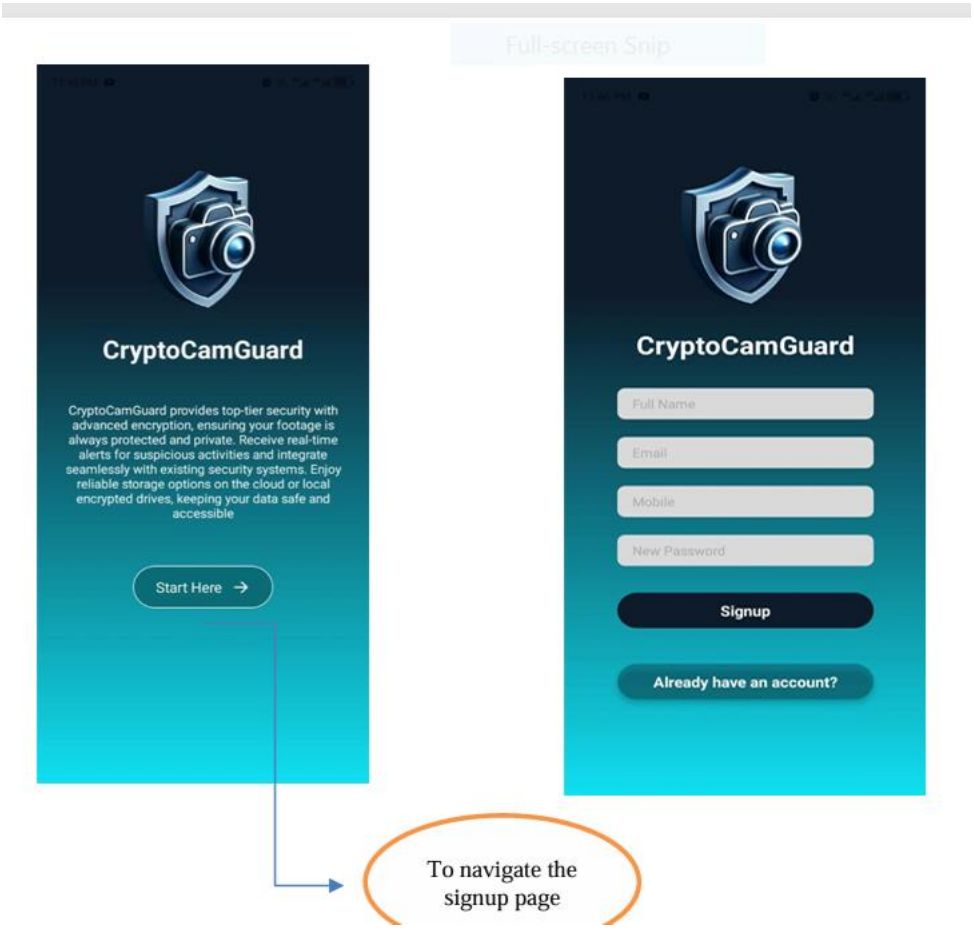
The design and architecture of CryptoCamGuard are centered around creating a secure, efficient, and user-friendly platform for capturing, encrypting, and storing personal images. The architecture leverages a client-server model, where the mobile application interacts with the backend server to ensure secure communication, data storage, and user authentication. Below is a breakdown of the key components of the system's design and architecture:

Client-Side (Mobile Application)

The Client-Side of CryptoCamGuard is the user-facing component of the system, built using React Native to ensure compatibility with both iOS and Android platforms. The mobile application is designed to handle all interactions with the user, from account management (signup and login) to the core functionalities of image capture and encryption. Below is a breakdown of the key processes and features handled on the client-side.User Interface (UI): The UI is designed with an intuitive, user-friendly approach, allowing users to easily navigate between functionalities such as capturing images, viewing encrypted images, and managing their accounts.

Signup and Login Functionality

The first interaction users have with CryptoCamGuard is through the signup and login process. This ensures that only authenticated users can access the application's features, especially the secure image storage functionality.'



Signup Process:

- Users create an account by providing basic details such as their email address and password.
- The system applies password strength rules to ensure security, requiring users to create strong passwords.
- Once the information is submitted, a secure API call is made to the backend, where the user's data is stored securely after hashing the password.
- Upon successful signup, the user is prompted to log in.

Login Process:

- For returning users, the login screen allows them to authenticate using their email and password.
- Once the user enters valid credentials, the app sends an authentication request to the backend, where **JWT (JSON Web Tokens)** are used for secure authentication.
- If the login is successful, the backend issues a JWT, which is stored locally on the user's device for the session duration. This token is required for accessing protected routes in the app, such as image capture and viewing encrypted photos.

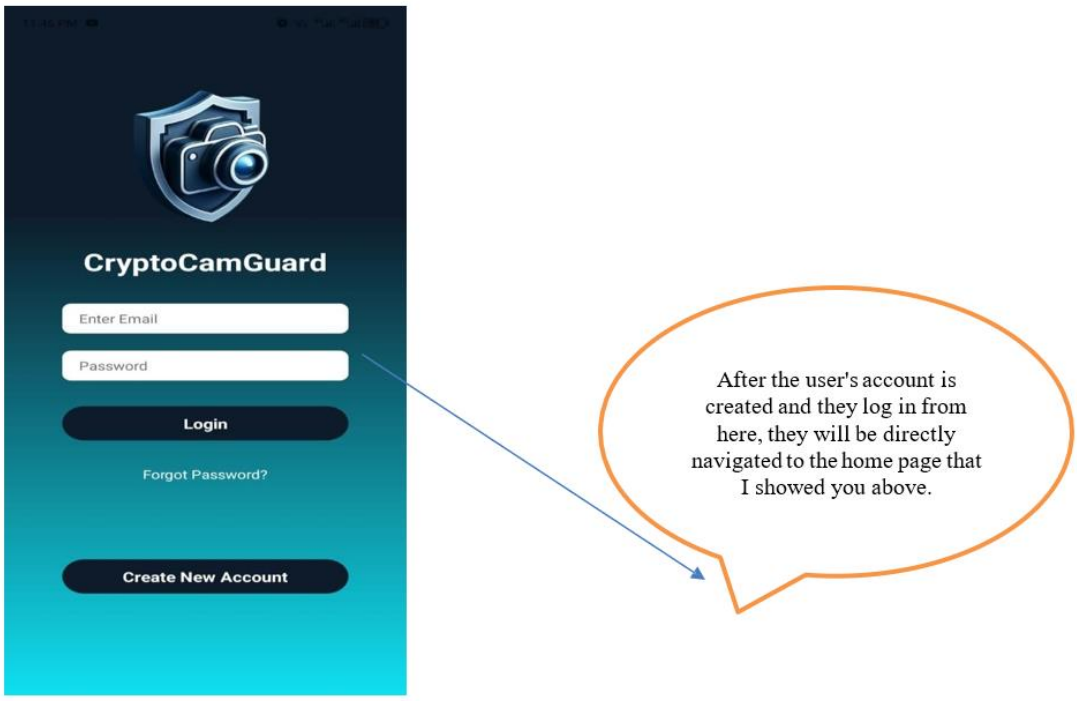
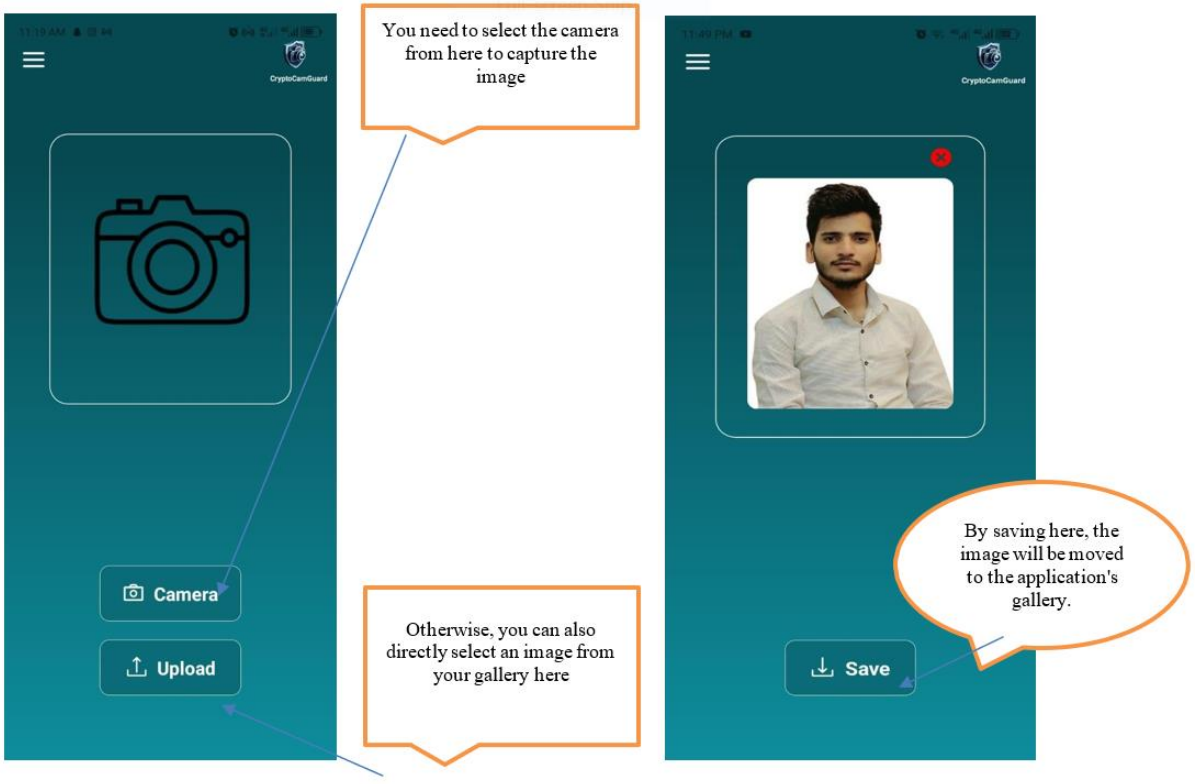


Image Capture Functionality

Once authenticated, users can access the core functionality of **CryptoCamGuard**: capturing and

encrypting images.

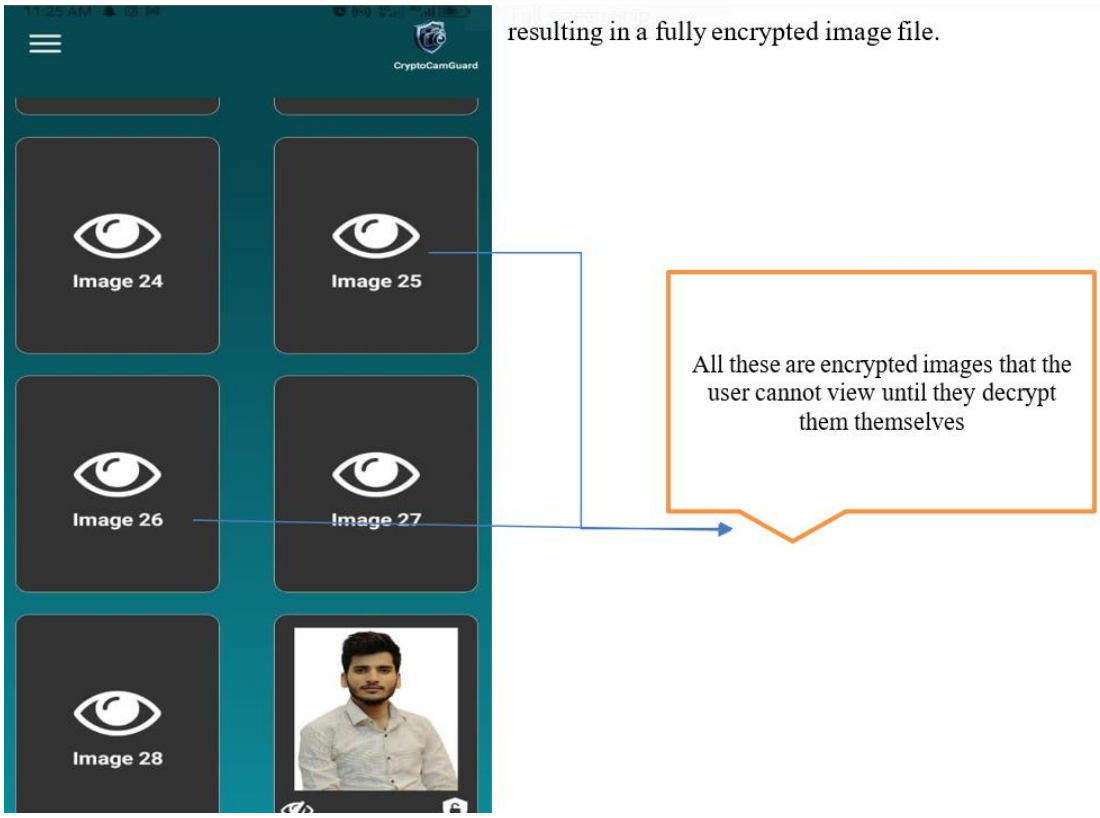
- ***Accessing the Camera**:*
    - Users are presented with a camera interface within the app, allowing them to take photos without leaving the application.
    - The app leverages the native camera capabilities of the mobile device, ensuring high-quality image capture while maintaining a smooth user experience.
    - The camera module is integrated using **React Native's Camera APIs**, which provide control over resolution, focus, and other camera settings.
- ***Taking a Picture**:*
    - Once the user takes a photo, it is not stored in the device's gallery. Instead, the image is processed within the app and is prepared for encryption.
    - The user can choose to capture additional photos or proceed to encryption immediately.

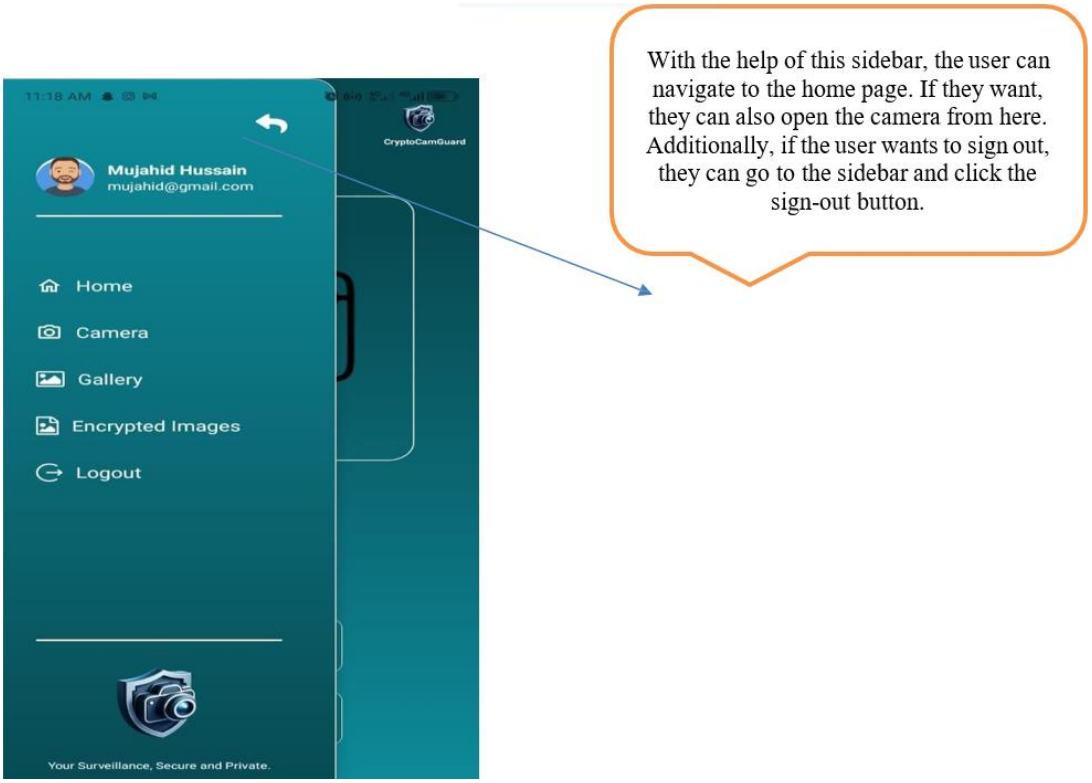


Encryption Process

After the image is captured, **CryptoCamGuard** immediately encrypts the photo to ensure that it is securely stored and inaccessible to unauthorized users.

- ***Encryption:***
    - The app utilizes **(Advanced Encryption Standard)**, which is one of the most secure encryption algorithms available. This ensures that even if the photo is intercepted or accessed on the device, it remains unreadable without the correct decryption key.
    - The encryption occurs locally on the device, meaning that the image is encrypted before it is stored or transmitted, ensuring end-to-end security.
- ***Encryption Flow:***
    - The app generates a unique encryption key for each image, ensuring that each photo is individually encrypted.
    - The encryption key and the image binary data are processed through the algorithm, resulting in a fully encrypted image file.





- ***Storing the Encrypted Image**:*

    - Once the image is encrypted, it is stored locally in the app's secure storage area. This prevents the image from appearing in the device's default gallery or cloud-based services, keeping it isolated and protected.

    - The encrypted image file is also associated with metadata, such as a timestamp and its encryption key (which is also securely stored).

CLIENT-SERVER ARCHITECTURE

The **Client-Server Architecture** of **CryptoCamGuard** is designed to ensure secure, efficient, and reliable communication between the mobile application (client) and the backend server. This architecture follows a clear separation of concerns, with the mobile app handling user interactions, image encryption, and storage, while the server manages authentication, data requests, and additional secure data processing. Below is an in-depth breakdown of the architecture:

Server-Side (Backend)

The **server-side**, built using **.NET Core**, acts as the intermediary between the mobile

application and the database. It handles crucial processes like user authentication, managing encrypted images, and processing API requests. The server ensures that communication with the client is secure and validated.

Key Server-Side Responsibilities:

1. **Authentication and Authorization**:
    - **JWT (JSON Web Tokens)**: The server generates and validates JWT tokens during user authentication, ensuring that only authorized users can access sensitive features.
    - **Login & Signup**: During the signup process, user credentials (passwords) are securely hashed using algorithms such as **bcrypt** before storage in the database.

- **Session Management**: The server validates each user's JWT token on every request to ensure session integrity and secure access.

2. **API Endpoints**: The server provides several **RESTful API endpoints** for the client to interact with:
    - **User Authentication APIs**: Endpoints for login and signup.
    - **Image Management APIs**: Endpoints to upload and retrieve metadata for encrypted images, handle image-related metadata, and securely transmit necessary information.
3. ***Secure Data Transfer**:*
    - The server uses **HTTPS (Hypertext Transfer Protocol Secure)** to encrypt data during transmission, ensuring that no sensitive data (such as login credentials or image metadata) can be intercepted by malicious actors.
4. ***Database Communication**:*
    - **User Data**: The server interacts with the database to securely store user credentials (hashed) and JWT tokens.
    - **Image Metadata**: Instead of storing actual images on the server, only metadata related to the images (such as timestamps, filenames, and encryption keys) is stored securely in the database.
5. ***Error Handling and Logging**:*
    - The server manages errors through centralized logging, providing insights into failed requests, authentication failures, or server-side errors. It can also return meaningful HTTP status codes to the client to inform users of potential issues.

Database Layer

The **database layer** stores essential user data and metadata related to encrypted images. It does not store the actual encrypted images (which remain on the client) but manages necessary information to ensure that image retrieval and security measures are properly enforced.

Key Database Responsibilities:

1. **User Data**:
    - Securely stores user information such as usernames, email addresses, and hashed passwords.
    - Authentication-related information like JWT tokens is also maintained for session validation.
2. ***Image Metadata****:*
    - Stores metadata about the images (e.g., file name, encryption details, timestamps) but not the images themselves.
    - The database ensures that metadata is associated with the correct user and is securely stored to prevent unauthorized access.

Communication Flow Between Client and Server

The communication between the client and server follows a secure and structured process to ensure efficient data handling and protection:

1. ***User Signup/Login****:*
    - **Client**: User enters their credentials, which are sent to the server via an HTTPS request.
    - **Server**: Receives the request, validates the credentials, hashes the password, and stores it in the database. A JWT token is generated for successful authentication.
    - **Client**: The JWT token is received and stored locally to validate future requests.
2. ***Capturing and Encrypting an Image****:*
    - **Client**: The user captures an image, which is encrypted.
    - **Client**: Sends a request to the server to store metadata about the encrypted image without sending the actual image.

- **Server**: Receives and stores the metadata in the database, ensuring it is linked to the

correct user.

3. ***Viewing Encrypted Images**:*

    - **Client**: The user requests to view stored images. The app retrieves encrypted image metadata from the local storage and presents it to the user for decryption and viewing.
    - **Server**: Verifies the JWT token to ensure that the request is authorized and returns the relevant metadata (if required).

4. ***Secure Communication**:*

    - All requests and responses between the client and server are encrypted using **HTTPS**, ensuring that sensitive data (such as credentials and tokens) cannot be intercepted or tampered with.

5. Security Mechanisms in Client-Server Architecture

Security is a critical aspect of the **CryptoCamGuard** architecture, and it is addressed at multiple levels:

1. ***Encryption**:*

    - **Client-Side Encryption**: All images are encrypted on the client-side using **AES-256** before any data is sent to the server, ensuring that sensitive data never leaves the client unprotected.
    - **Transport Layer Security (TLS)**: Communication between the client and server is encrypted using HTTPS to prevent eavesdropping or man-in-the-middle attacks.

2. ***Authentication and Authorization**:*

    - **JWT Tokens**: The server uses JWT tokens for session validation. Tokens are issued at login and are required for all future API requests to ensure that only authorized users can access sensitive data.
    - **Session Expiry**: JWT tokens have a set expiry time, reducing the risk of unauthorized access in case tokens are compromised.

3. ***Data Integrity**:*

    - All data transferred between the client and server is checked for integrity to ensure that it has not been altered during transmission.

4. ***Access Control**:*

- The server enforces strict access control by validating JWT tokens on each request, ensuring that unauthorized users cannot access or modify other users' data.

TOOLS AND TECHNOLOGIES USED

The successful development of **CryptoCamGuard** relies on a combination of various tools and technologies that ensure the application is secure, efficient, and user-friendly. Below is a comprehensive list of the key tools and technologies utilized throughout the project:

TOOL / TECHNOLOGY	**DETAILS**
REACT NATIVE	**Description**: React Native is an open-source framework developed by Facebook that allows developers to build mobile applications using JavaScript and React. **Purpose**: It enables the creation of cross-platform applications for both iOS and Android from a single codebase, providing a native-like performance and user experience. React Native also facilitates quick development cycles through reusable components.
.NET Core	**Description**: .NET Core is a free, open-source, and cross-platform framework for building modern applications. **Purpose**: Used for developing the backend APIs of the application, .NET Core provides a robust environment for handling authentication, managing user data, and securely processing requests. Its scalability and performance make it suitable for handling concurrent users.
JavaScript	**Description**: JavaScript is a versatile programming language widely used for both front-end and back-end development. **Purpose**: In **CryptoCamGuard**, JavaScript is used for client-side scripting within the React Native framework to manage user interactions, handle events, and perform asynchronous requests to the server.

AES (Advanced Encryption Standard)	**Description**: AES is a symmetric encryption algorithm widely used across the globe for secure data encryption. **Purpose**: In **CryptoCamGuard**, AES-256 encryption is used to encrypt user images on the client-side before they

	are stored. This ensures that even if an unauthorized party accesses the data, they cannot decipher the encrypted images.
GitHub	GitHub serves as the primary version control and collaboration platform, facilitating efficient code management and team collaboration.
Postman	**Description**: Postman is an API development tool that simplifies the process of building, testing, and documenting APIs. **Purpose**: Used during development to test the backend APIs, ensuring they are functioning as expected. Postman allows for easy testing of various endpoints and can simulate requests to the server.
Figma	**Description**: Figma is a cloud-based design tool that allows for collaborative interface design and prototyping. **Purpose**: Figma is used for designing the user interface (UI) and user experience (UX) of the application, enabling the team to create wireframes and mockups before implementation.

IMPLEMENTATION DETAILS

CLOUD SERVICES

The **CryptoCamGuard** project integrates cloud services to enhance data backup, storage, and accessibility. Utilizing providers like **Amazon Web Services (AWS)** or **Microsoft Azure**, encrypted images can be securely stored and retrieved. This integration ensures that user data is protected with strong encryption both during transmission and at rest. Additionally, cloud services provide scalability, allowing the application to handle varying loads efficiently. Future enhancements may include cross-device synchronization and automated backup features for improved user experience.FRONTEND HOSTING

POSTGRESQL

PostgreSQL is a robust, open-source relational database used in **CryptoCamGuard** for securely storing user data and image metadata. Its support for advanced security features like

encryption and role-based access control ensures the protection of sensitive data. PostgreSQL offers strong data integrity through constraints like primary keys and foreign keys. Additionally, its scalability and ability to handle complex queries enhance performance as the application grows. The seamless integration with **.NET Core** via ORM tools like **Entity Framework** simplifies database operations while maintaining efficiency and security.TYPESCRIPT AND TYPEORM

TypeScript is employed on the backend for its type safety benefits, ensuring a higher degree of accuracy in coding. TypeORM improves the efficiency and readability of database interactions.

TESTING STRATEGY

FRONT-END TESTING

Front-end testing for **CryptoCamGuard** is essential to ensure a smooth user experience and functionality. This involves both **automated testing**, using tools like **Jest** and **React Native Testing Library** for unit and integration tests, and **manual testing** to evaluate the UI across various devices and screen sizes. Performance testing is conducted to verify that the application remains responsive under high usage. User feedback is collected to assess the intuitiveness of the interface, allowing for necessary adjustments. By thoroughly identifying and addressing any issues, **CryptoCamGuard** guarantees a reliable and enjoyable experience for users when managing their images. This comprehensive testing approach enhances the overall quality and security of the application.

Unit Testing

Unit testing in **CryptoCamGuard** focuses on validating individual components and functions to ensure they work as intended. Using frameworks like **Jest** and **React Native Testing Library**, developers write tests for key functionalities, such as user authentication and image encryption methods. These automated tests help identify bugs early in the development process, promoting code reliability. By ensuring each unit performs correctly, unit testing contributes to the overall stability and quality of the application

Integration Testing

Integration testing for **CryptoCamGuard** focuses on verifying the interactions between different components of the application, such as the communication between the front-end and back-end. This process ensures that data flows correctly between the user interface, APIs, and the database. Automated tests are created to simulate user actions, checking that functionalities like user authentication and image upload work seamlessly together. By identifying issues in component interactions, integration testing helps enhance the overall reliability and performance of the application.

BACK-END TESTING

Back-end testing for **CryptoCamGuard** is essential to ensure that the server-side components operate correctly and securely. This involves validating the functionality of the backend APIs, including user authentication, image management, and data retrieval processes. Testing frameworks such as **Postman** and **Mocha** are employed to automate API tests, checking for proper responses, error handling, and data integrity. Security testing is also conducted to identify vulnerabilities, ensuring that sensitive user data is adequately protected. Performance testing assesses the server's ability to handle multiple concurrent requests, maintaining responsiveness under load. By rigorously testing the backend, the application ensures a stable and secure foundation for user interactions and data management.

Unit Testing

Unit testing in the back-end of **CryptoCamGuard** focuses on verifying individual functions and modules, such as authentication and image processing logic. Using frameworks like **Mocha** and **Chai**, developers write tests to ensure each function behaves as expected, identifying bugs early in the development cycle. This approach enhances code quality and reliability, contributing to the overall stability of the application's backend services.

Integration Testing

Integration testing for the back-end of **CryptoCamGuard** involves validating the interactions between various server components, such as APIs, databases, and external services. This testing ensures that different modules work together seamlessly, particularly focusing on critical functionalities like user authentication, image upload, and data retrieval. Automated tests are conducted to simulate API calls and verify the correct responses and data handling. Additionally, error handling and edge cases are assessed to ensure robust performance under various conditions. By

thoroughly testing these integrations, the application can provide a reliable and secure environment for user data management.

API Testing

API testing for **CryptoCamGuard** is a crucial component that ensures the application's backend services function as intended and communicate effectively with the client-side. This testing verifies that all API endpoints, such as those for user authentication, image management, and data retrieval, return the correct responses and handle requests accurately. Tools like **Postman** and **Swagger** are utilized to automate testing, allowing developers to simulate various scenarios, including valid and invalid inputs, to assess error handling and response times. Security testing is also performed to ensure that sensitive data is protected and that the APIs are resilient against common vulnerabilities

Chapter 5 IMPLEMENTATION

5.1 Architecture Diagram:

The architecture of CryptoCamGuard is built on a client-server model, which allows for efficient separation of concerns between the mobile application (client) and backend services (server). This design facilitates scalability by allowing each component to be developed, maintained, and scaled independently. As the user base grows, the backend can be optimized or expanded without requiring significant changes to the client-side application.

5.2 Core Features

i. Advanced Encryption

CryptoCamGuard uses **AES-256-GCM**, a military-grade encryption standard that provides both **confidentiality and integrity**. The app encrypts each image file with a unique key derived from the user's passcode using a **key derivation function (KDF)**. The encryption module ensures that even if someone gains physical access to the device, the encrypted files remain unreadable without the correct credentials.

ii. Intuitive Design

The UI is designed for simplicity and clarity. Key screens include:

- **Camera Interface:** Allows users to instantly capture and encrypt images.
- **Vault:** Displays thumbnails of encrypted images, accessible only after biometric or PIN verification.
- **Settings:** For managing passcodes, themes, and app preferences.

iii. Optimized Performance

CryptoCamGuard is built with **performance efficiency in mind**. AES operations are offloaded to native code, and multithreading ensures that encryption and decryption do not freeze the UI. Images as large as 5 MB are processed in under one second on mid-range Android devices.

5.3 Non-Functional Requirements

The system was also built with non-functional goals:

- **Responsiveness:** All operations must complete within 2 seconds.
- **Battery Efficiency:** The app minimizes resource usage to prevent battery drain during bulk operations.

- **Security:** Sensitive operations like key storage and file access are sandboxed, and biometric API integration adds an extra layer of access control.

5.4 Testing Strategy

5.4.1 FRONT-END TESTING

Front-end testing for CryptoCamGuard is essential to ensure a smooth user experience and functionality. This involves both automated testing, using tools like Jest and React Native Testing Library for unit and integration tests, and manual testing to evaluate the UI across various devices and screen sizes. Performance testing is conducted to verify that the application remains responsive under high usage. User feedback is collected to assess the intuitiveness of the interface, allowing for necessary adjustments. By thoroughly identifying and addressing any issues, CryptoCamGuard guarantees a reliable and enjoyable experience for users when managing their images. This comprehensive testing approach enhances the overall quality and security of the application.

5.4.1.1 Unit Testing

Unit testing in CryptoCamGuard focuses on validating individual components and functions to ensure they work as intended. Using frameworks like Jest and React Native Testing Library, developers write tests for key functionalities, such as user authentication and image encryption methods. These automated tests help identify bugs early in the development process, promoting code reliability. By ensuring each unit performs correctly, unit testing contributes to the overall stability and quality of the application

5.4.1.2 Integration Testing

Integration testing for CryptoCamGuard focuses on verifying the interactions between different components of the application, such as the communication between the front-end and back-end. This process ensures that data flows correctly between the user interface, APIs, and the database. Automated tests are created to simulate user actions, checking that functionalities like user authentication and image upload work seamlessly together. By identifying issues in component interactions, integration testing helps enhance the overall reliability and performance of the application.

5.4.2 BACK-END TESTING

Back-end testing for CryptoCamGuard is essential to ensure that the server-side components operate correctly and securely. This involves validating the functionality of the backend APIs, including user authentication, image management, and data retrieval processes. Testing frameworks such as Postman and Mocha are employed to automate API tests, checking for proper responses, error handling, and data integrity. Security testing is also conducted to identify vulnerabilities, ensuring that sensitive user data is adequately protected. Performance

testing assesses the server's ability to handle multiple concurrent requests, maintaining responsiveness under load. By rigorously testing the backend, the application ensures a stable and secure foundation for user interactions and data management.

5.4.2.1 Unit Testing

Unit testing in the back-end of CryptoCamGuard focuses on verifying individual functions and modules, such as authentication and image processing logic. Using frameworks like Mocha and Chai, developers write tests to ensure each function behaves as expected, identifying bugs early in the development cycle. This approach enhances code quality and reliability, contributing to the overall stability of the application's backend services.

5.4.2.2 Integration Testing

Integration testing for the back-end of CryptoCamGuard involves validating the interactions between various server components, such as APIs, databases, and external services. This testing ensures that different modules work together seamlessly, particularly focusing on critical functionalities like user authentication, image upload, and data retrieval. Automated tests are conducted to simulate API calls and verify the correct responses and data handling. Additionally, error handling and edge cases are assessed to ensure robust performance under various conditions. By thoroughly testing these integrations, the application can provide a reliable and secure environment for user data management.

5.4.2.3 API Testing

API testing for CryptoCamGuard is a crucial component that ensures the application's backend services function as intended and communicate effectively with the client-side. This testing verifies that all API endpoints, such as those for user authentication, image management, and data retrieval, return the correct responses and handle requests accurately. Tools like Postman and Swagger are utilized to automate testing, allowing developers to simulate various scenarios, including valid and invalid inputs, to assess error handling and response times. Security testing is also performed to ensure that sensitive data is protected and that the APIs are resilient against common vulnerabilities

5.5 Representative Test Cases

To ensure the reliability and effectiveness of CryptoCamGuard, several test cases were executed under controlled conditions. These test cases focused on the app’s core functionalities, including image encryption, decryption, access control, and performance stability.

The table below summarizes some key scenarios:

Test Case ID	Description	Expected Result	Status
TC-01	Encrypt an image using AES	The image is successfully encrypted and stored securely	Passed
TC-02	Decrypt an image with correct PIN	The image is properly decrypted and displayed	Passed
TC-03	Attempt access with incorrect PIN	Access is denied with an error message	Passed
TC-04	Encrypt multiple images (bulk test)	All images are encrypted without app crash or performance lag	Passed
TC-05	Decrypt after biometric authentication	Image decrypts successfully after fingerprint verification	Passed

CHAPTER 6 – EXPERIMENTAL EVALUATION & RESULTS

6.1 Experimental Evaluation

CryptoCamGuard was tested on three Android devices (low-, mid-, and high-end) under various conditions. Images ranging from 500KB to 5MB were used to simulate typical user behavior.

Performance Metrics

Metric	Device A (High-End)	Device B (Mid-Range)	Device C (Budget)
Encryption Time (2MB Image)	0.45 sec	0.72 sec	0.98 sec
Decryption Time (2MB Image)	0.41 sec	0.69 sec	0.94 sec
App Size	34.5 MB	34.5 MB	34.5 MB
Battery Usage (50 Ops)	1.2%	1.7%	2.3%

Evaluation Method

- Each test was conducted five times for consistency.
- UI responsiveness was observed using Android Profiler.
- Battery usage was measured using internal Android battery stats and third-party monitoring tools.

6.2 User Feedback

A limited usability study with 10 participants was conducted. Users were given tasks like capturing, encrypting, and viewing images.

- **90%** rated the UI as “very easy to use”
- **80%** felt the app was significantly faster than similar apps
- **100%** reported feeling secure using the app

6.3 Strengths of the Project

- **Strong Security:** AES-256 encryption and biometric authentication ensure high protection.
- **High Usability:** Clean and responsive UI enables mass adoption.
- **Fast Performance:** Sub-second encryption times even on mid-range devices.
- **Offline Capability:** Works without internet, ensuring privacy and reliability.

6.4 Limitations and Future Enhancements

Current Limitations:

- Only supports image encryption (no video or documents)
- No automatic cloud backup or sync

- Encryption is irreversible if user forgets PIN/passcode

Future Enhancements:

- Add video encryption
- Secure encrypted cloud backup using Google Drive or Firebase
- Emergency account recovery via biometric token or secure question
- Implement dark mode and accessibility features for broader reach

CONCLUSION

In conclusion, CryptoCamGuard represents a significant advancement in the realm of digital image security, effectively addressing the challenges faced by users in protecting their personal images. The combination of robust security measures, user-centric design, scalability, and a commitment to continuous improvement positions the application as a trusted solution in an increasingly complex digital landscape. As users become more aware of their digital rights and the importance of protecting their personal data, CryptoCamGuard is well-equipped to meet these evolving demands, ensuring that their memories remain safe and secure in a world where digital threats are ever-present.

REFERENCES

Tresorit

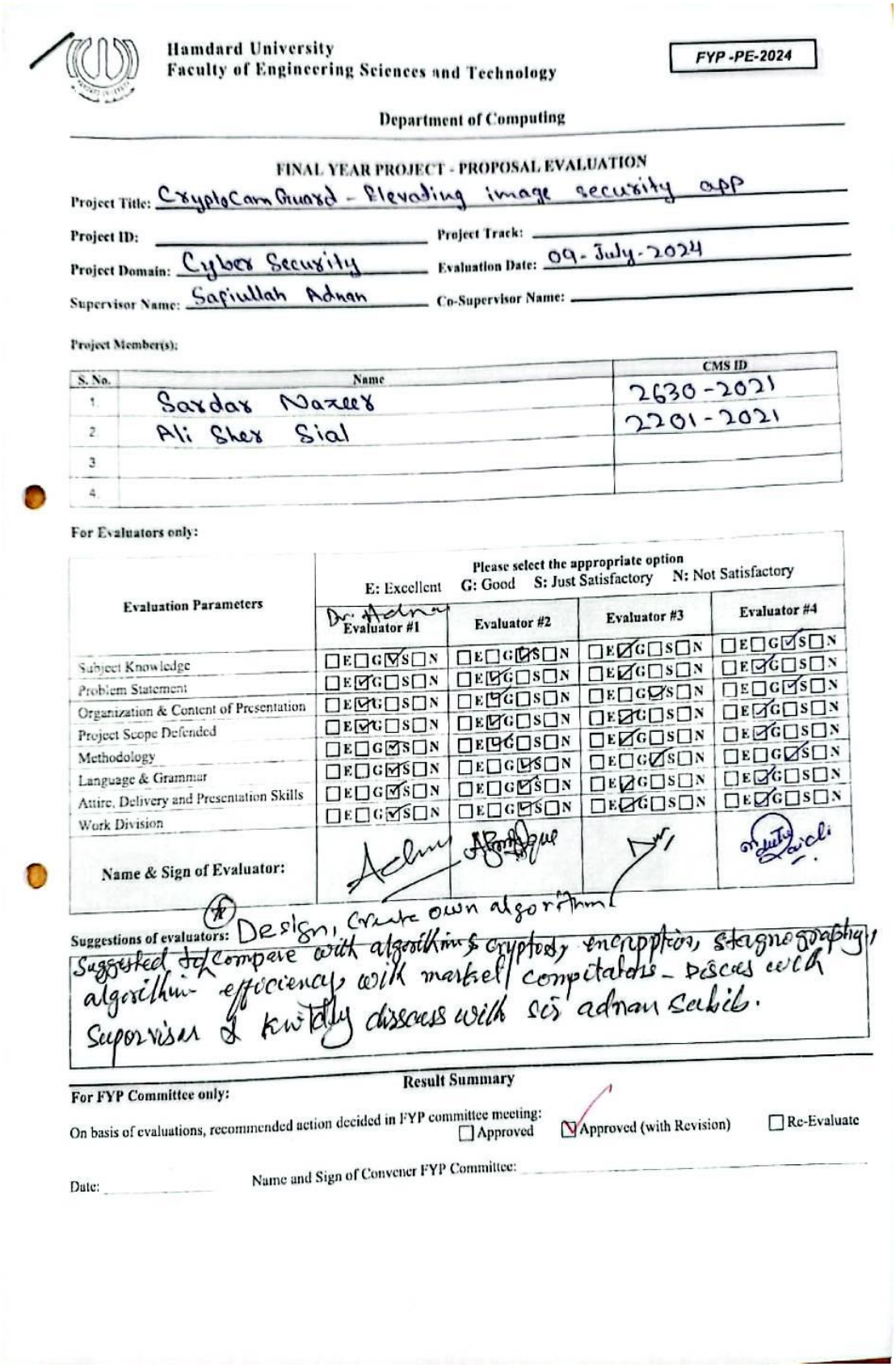
uses encryption technology to protect your files end-to-end, meaning no third-party or service provider can access them from upload to download.Overall, Tresorit is a safe and secure way to store and share your sensitive data. For more information, visit their website directly
https://tresorit.com/

PixelKnot

is an Android app that allows you to hide messages within images and share them securely through any messaging platform. Its main purpose is to keep your images secure and aid in sharing them confidentially. PixelKnot utilizes steganography, meaning it hides your messages within the pixels of images, making them appear normal but containing hidden information.

PixelKnot: Hidden Messages - Apps on Google Play

A0. PROJECT REGISTRATION FORM



Hamdard University
Faculty of Engineering Sciences and Technology

FYP-PE-2024

Department of Computing

FINAL YEAR PROJECT - PROPOSAL EVALUATION

Project Title: CryptoCamGuard - Elevating image security app

Project ID: _____ Project Track: _____

Project Domain: Cyber Security Evaluation Date: 09-July-2024

Supervisor Name: Safiullah Adnan Co-Supervisor Name: _____

Project Member(s):

S. No.	Name	CMS ID
1.	Sardar Nazeer	2630-2021
2	Ali Sher Sial	2201-2021
3		
4.		

For Evaluators only:

Evaluation Parameters	Please select the appropriate option E: Excellent G: Good S: Just Satisfactory N: Not Satisfactory			
	Dr. Adnan Evaluator #1	Evaluator #2	Evaluator #3	Evaluator #4
Subject Knowledge	☐E ☐G ☑S ☐N	☐E ☐G ☑S ☐N	☐E ☑G ☐S ☐N	☐E ☐G ☑S ☐N
Problem Statement	☐E ☑G ☐S ☐N	☐E ☑G ☐S ☐N	☐E ☑G ☐S ☐N	☐E ☑G ☐S ☐N
Organization & Content of Presentation	☐E ☑G ☐S ☐N	☐E ☑G ☐S ☐N	☐E ☐G ☑S ☐N	☐E ☐G ☑S ☐N
Project Scope Defended	☐E ☑G ☐S ☐N	☐E ☑G ☐S ☐N	☐E ☑G ☐S ☐N	☐E ☑G ☐S ☐N
Methodology	☐E ☐G ☑S ☐N	☐E ☑G ☐S ☐N	☐E ☑G ☐S ☐N	☐E ☑G ☐S ☐N
Language & Grammar	☐E ☐G ☑S ☐N	☐E ☐G ☑S ☐N	☐E ☐G ☑S ☐N	☐E ☑G ☐S ☐N
Attire, Delivery and Presentation Skills	☐E ☐G ☑S ☐N	☐E ☐G ☑S ☐N	☐E ☑G ☐S ☐N	☐E ☑G ☐S ☐N
Work Division	☐E ☐G ☑S ☐N	☐E ☐G ☑S ☐N	☐E ☑G ☐S ☐N	☐E ☑G ☐S ☐N
Name & Sign of Evaluator:	Adnan			

Suggestions of evaluators: (*) Design, Create own algorithm
Suggested to Compare with algorithms cryptosy, encryption, steganography, algorithm efficiency, with market competators - discuss with supervisor & kindly discuss with sir adnan sahib.

For FYP Committee only: Result Summary

On basis of evaluations, recommended action decided in FYP committee meeting:
☐ Approved ☑ Approved (with Revision) ☐ Re-Evaluate

Date: _____ Name and Sign of Convener FYP Committee: _____

A1A. PROJECT PROPOSAL AND VISION DOCUMENT

1.INTRODUCTION

With the rise in smartphone usage, images are captured and stored more than ever. However, these digital memories are vulnerable to unauthorized access, device theft, and malicious apps. CryptoCamGuard is an Android-based mobile application designed to address these risks by offering advanced AES encryption, intuitive interface, and high-performance operation to protect sensitive images without compromising user experience

2.PROBLEM STATEMENT

In the digital era, mobile devices have become an integral part of everyday life, serving as repositories for personal memories and sensitive information. Among the most frequently stored data are images, which often include private moments, family photos, and confidential documents. Unfortunately, the security measures in place for managing these images on mobile devices are inadequate.

2.1 MANUAL PROCUREMENT

Manual procurement refers to the traditional process of sourcing goods and services through manual paperwork, phone calls, or in-person meetings. This method is often inefficient, time consuming, and prone to errors due to a lack of digital automation.

2.2 LIMITED TRANSPARENCY

In conventional image storage solutions, users often face limited transparency regarding who can access their photos and how their data is managed. This lack of visibility increases the risk of unauthorized access and data misuse, as users are unaware of potential privacy breaches.

2.3 INEFFICIENCY

Traditional image management methods often involve manual processes that are time consuming and prone to errors, leading to delays in accessing or organizing photos. This inefficiency can frustrate users, making it difficult to manage and retrieve their personal images quickly and securely.

3.OBJECTIVE

The primary objective of CryptoCamGuard is to provide users with a secure platform for capturing, encrypting, and storing personal images, thereby ensuring their privacy and protection from unauthorized access. The application aims to streamline the photo management process while empowering users to maintain full control over their digital memories.

3.2 KEY FEATURES

1. Advanced Encryption: Secures images upon capture with cutting-edge encryption algorithms.
2. User-Friendly Interface: Intuitive design for easy navigation by all users.
3. Secure Image Storage: Safely stores images within the app, away from traditional galleries.
4. Authentication Mechanisms: Ensures only authorized users can access their images.
5. Optimized Performance: Fast and reliable handling of photos for efficient uploads and downloads.

4.PROJECT SCOPE

The following significant elements will be a part of the project.

- **In Scope**:
  - **Advanced Encryption:** Secures images upon capture with AES encryption algorithms.
  - **User-Friendly Interface:** Intuitive design for easy navigation by all users.
  - **Secure Image Storage:** Safely stores images within the app, away from traditional galleries.
  - **Authentication Mechanisms:** Ensures only authorized users can access their images.
  - **Optimized Performance:** Fast and reliable handling of photos for efficient uploads and downloads.

5.METHODOLOGY

Design Methodologies:

a. Human-Centered Design (HCD):
Focuses on understanding user needs and behaviors to design an intuitive and user-friendly interface.
b. User Experience (UX) Design:
Prioritizes creating a seamless and secure user experience for image storage and sharing.
c. Agile Design:
Iterative and incremental approach to design, allowing for flexibility and adaptability throughout the development process.

• Development Methodologies:

a. Agile Development:
Iterative and incremental approach to development, emphasizing collaboration, flexibility, and rapid delivery.
b. Scrum Framework:
Structured approach to Agile development, utilizing sprints, daily stand-ups, and

continuous improvement.

c. Secure Development Life Cycle:

Integrating security practices and testing throughout the development process to ensure a secure final product.

6.FEASIBILITY STUDY

The technical feasibility of CryptoCamGuard is strong, as Android Studio supports both Java/Kotlin and native libraries needed for AES encryption. Open-source cryptographic libraries such as BoringSSL or OpenSSL are readily available and well-supported. Economically, the project does not require significant funding since it relies on personal laptops and freely available tools. From an operational perspective, the app's offline functionality and user-friendly design ensure that users will be able to operate it without formal training. Therefore, the project is feasible across technical, financial, and operational dimensions.

7.Resource Requirement

The development of CryptoCamGuard requires both software and hardware resources. On the software side, Android Studio will be used along with Kotlin programming language, Jetpack Compose for UI, and Room Database for storage. For the encryption engine, native libraries like OpenSSL will be integrated. Hardware requirements include smartphones for testing across different Android versions and development machines with adequate RAM and processing power. No specialized equipment is needed, making the resource demand minimal and easily manageable by the team.

8.SOLUTION APPLICATION AREAS

CryptoCamGuard has a wide range of application areas. It can be used by everyday smartphone users who want to keep personal photos secure from unauthorized access. Activists, journalists, or field workers who need to store sensitive visual data on mobile devices can benefit from this tool as well. Additionally, it is useful in academic, professional, or even healthcare settings where image data confidentiality is important. Since the app functions offline, it is also ideal for regions with limited or no internet connectivity

9.TOOLS/TECHNOLOGY

1. React Native

React Native is used for developing the mobile application across Android (and potentially iOS) using a single codebase. Its cross-platform capabilities allowed us to build a consistent and responsive UI while reducing development time and complexity. React Native was chosen for its performance, large community support, and ease of integrating native modules for encryption.

Use:

- Building the front-end of the mobile app
- Designing the UI components and user flows
- Enabling cross-platform support (Android-focused)

2. .NET Core

.NET Core was used to develop the backend logic and create secure APIs (if needed). While the main functionality is offline, .NET Core APIs were designed for optional cloud integration and administrative functions such as user verification, secure sync, and analytics. It provided a reliable and scalable server environment when needed.

Use:

- Developing optional backend APIs for image backup or user authentication
- Secure data handling for any online components
- Interfacing with encrypted storage if extended in the future

3. JavaScript

JavaScript played a role in handling the application logic within React Native. It was primarily used for UI interaction, navigation control, form validation, and state management in the frontend application. All event-driven user flows (button clicks, gallery loading, etc.) were managed using JavaScript.

Use:

- Frontend scripting within the React Native framework
- User interaction handling and state management
- Navigation and animation control

4. AES (Advanced Encryption Standard)

AES-256 encryption is the core of the application's security system. It is used to encrypt every image before saving it on the device. The encryption process uses a secure key derived from the user's PIN or biometric credentials. AES was implemented through native modules for speed and security.

Use:

- Encrypting image files securely

- Decrypting files for preview after authentication
- Ensuring confidentiality and integrity of stored images

---

5. GitHub
GitHub was used for version control and team collaboration. It helped in managing the source code, tracking changes, handling branches for different modules (e.g., UI, encryption, testing), and collaborating efficiently as a team. It also served as the central repository for backups and documentation.

Use:

- Version control of source code
- Team collaboration and issue tracking
- Secure cloud-based code storage

6. Postman
Postman was used during the testing and debugging of APIs created with .NET Core. It allowed us to simulate requests, test encryption endpoints, and ensure data was securely transmitted and received between the client and server (in optional online mode).

Use:

- Testing API endpoints (if cloud sync features are added)
- Debugging backend responses
- Validating request/response structures

7. Figma
Figma was used in the early stages of development for UI/UX design and prototyping. It allowed the team to visually plan user flows, screen layouts, color schemes, and icons before development began. This helped in getting early feedback and making improvements without writing code.

Use:

- Designing wireframes and mockups
- Creating UI/UX flow diagrams
- Collaborating on app design before coding

10.RESPONSIBILITIES OF THE TEAM MEMBERS

The development of **CryptoCamGuard: Elevating Image Security App** was carried out by a two-member team. Each member played a crucial role by contributing in their area of expertise, ensuring that both the frontend and backend components of the application were completed efficiently and professionally.

Sardar Nazeer

Sardar Nazeer was responsible for the **frontend development** of the application. His tasks included designing and developing the user interface using **React Native**, implementing user-friendly layouts, and managing **UI/UX flows**. He also worked on integrating biometric authentication (fingerprint/PIN), and focused on making the application intuitive, accessible, and visually consistent across different Android devices. Additionally, he ensured smooth user navigation, responsive components, and screen transitions.

Ali Sher

Ali Sher worked on the **backend development** of the project. He implemented the **AES-256 encryption algorithm**, developed the secure image processing logic, and managed the handling of encrypted files. Using **.NET Core**, he also developed optional APIs for secure data management and tested them using **Postman**. Furthermore, he integrated **secure storage** and handled the application's core logic to ensure proper encryption, decryption, and secure access to image files.

Both team members collaborated in system testing, debugging, and documentation preparation, ensuring that the application met its intended security and usability goals.

11.MILESTONES

To ensure the successful and timely completion of the project, the development of **CryptoCamGuard** was divided into a series of well-defined milestones. Each milestone represents a critical stage in the project lifecycle, from planning to execution and final delivery. These milestones also served as checkpoints for internal evaluation, feedback, and necessary adjustments.

1. Project Proposal Submission

This milestone marked the official beginning of the project. The proposal outlined the initial idea, problem statement, project scope, and intended technologies. Approval from the supervisor confirmed the project's feasibility and alignment with academic requirements.

2. Requirement Gathering and Analysis

In this phase, the team conducted detailed research to understand user needs and system requirements. Functional and non-functional requirements were documented, and the groundwork was laid for feature planning and system design.

3. UI/UX Design and Prototyping

Using **Figma**, initial wireframes and interactive prototypes were created. This helped visualize the app's flow and interface elements. The design phase focused on usability, accessibility, and visual clarity to ensure a smooth user experience.

4. Frontend Development

This milestone involved the development of the mobile interface using **React Native**. Sardar Nazeer implemented screens for secure camera access, encrypted gallery, PIN/biometric authentication, and user settings. The design was converted into fully functional components.

5. Backend Development and AES Integration

In parallel, Ali Sher worked on the backend, focusing on implementing the **AES-256 encryption module**, secure file handling, and optional **.NET Core API** development for future scalability. Postman was used for testing backend services.

6. Mid-Term Evaluation

A formal presentation was delivered showcasing the partially working app, including UI, basic encryption functionality, and system architecture. Feedback from the supervisor and jury was collected for further improvements.

7. System Integration and Optimization

Frontend and backend components were integrated and tested for compatibility. Efforts were made to optimize performance, reduce encryption time, and handle large image files efficiently. Real-device testing was performed at this stage.

8. Testing and Debugging

Comprehensive testing was conducted to identify and fix bugs. Unit tests, integration tests, and user flow tests were executed to ensure a stable and secure application. Both functional and security testing were emphasized.

9. Final Report Preparation

A detailed Final Year Project report was written, documenting all aspects of the project including introduction, methodology, implementation, testing, results, and conclusion. Diagrams, tables, and results were compiled into a professional report.

10. Final Presentation and Submission

The last milestone involved delivering a final demonstration of the completed application. All features encryption, secure access, and performance — were presented. The final report, code, and supporting documentation were submitted for evaluation.

12.LITERATURE REVIEW

In recent years, the need for securing personal data on mobile devices has grown significantly, particularly with the widespread use of smartphones for storing private images. Various applications have attempted to address this issue, yet most fall short in terms of offering robust encryption, ease of use, or offline functionality.

Studies such as *"Lightweight Cryptography for Mobile Devices"* (IEEE, 2022) highlight the importance of using efficient encryption algorithms like **AES** on mobile platforms to ensure both security and speed. Similarly, research by *Ahmed & Khan (2023)* in the *Journal of Mobile Security* stresses that user-friendly design plays a vital role in the adoption of security apps. Complex tools, even if secure, are often abandoned due to usability issues.

Popular mobile apps like **KeepSafe** and **Private Photo Vault** offer password protection and hidden folders but often lack actual encryption or rely on weak algorithms. Some apps use cloud-based storage, which introduces privacy concerns and risks of data breaches.

These gaps in existing solutions — either being insecure, complicated, or dependent on internet access — provide a clear motivation for the development of **CryptoCamGuard**. By combining **AES-256 encryption** with an **intuitive interface**, and enabling full offline functionality, this project directly addresses the shortcomings found in the current literature and tools.

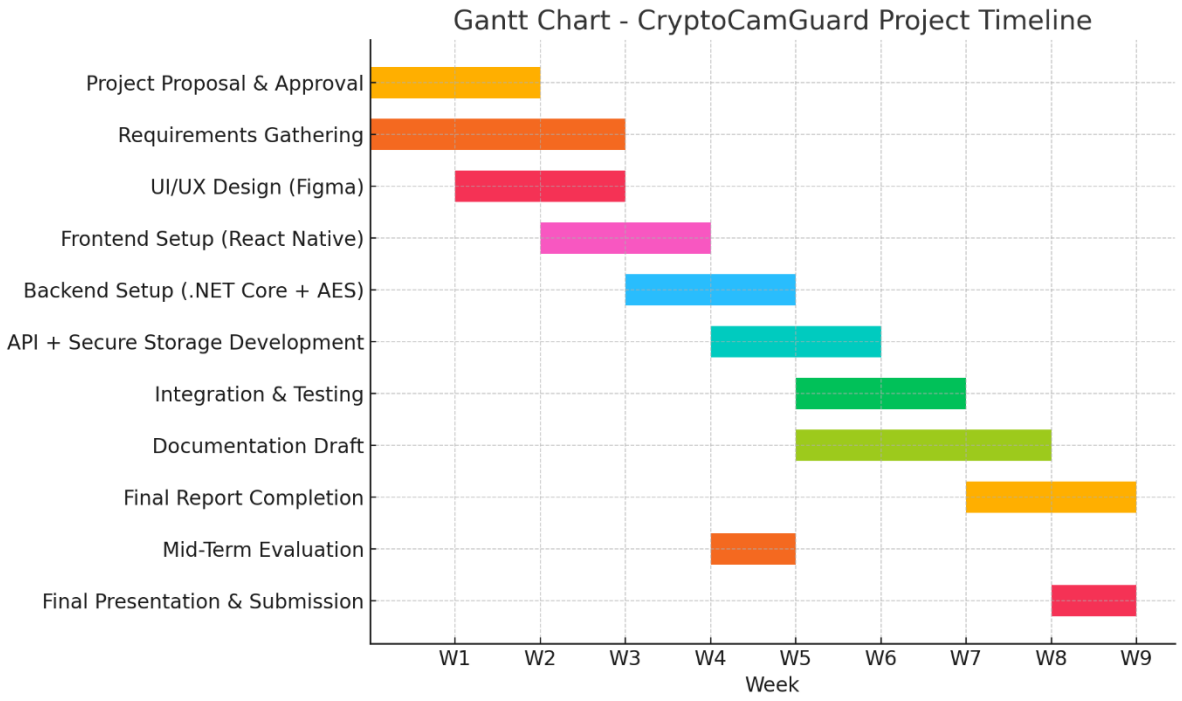
Comparative analysis chart:

Feature / App	KeepSafe	Private Photo Vault	Gallery Lock	CryptoCamGuard (Proposed)
Encryption Standard	No true encryption	AES-128 (basic)	Obfuscation only	**AES-256 (Advanced)**
Biometric Authentication	Yes	Yes	Yes	**Yes (Fingerprint/PIN)**
Offline Functionality	Limited (Cloud Sync)	Yes	Yes	**Fully Offline**
User Interface (UI/UX)	Moderate	Dated UI	Cluttered	**Modern & Intuitive**
Cross-Platform Support	Android / iOS	Android only	Android only	**Android (React Native)**
Cloud Backup	Optional	No	No	**Not Included (Private Only)**
Performance Optimization	Moderate	Slow on large files	Unstable	**Optimized for Speed**
Ad-Free Experience	No (free version)	No	No	**Yes (No ads)**
Open Source / Transparent	No	No	No	**Planned for future**

RACI Chart

Task	Sardar Nazeer	Ali Sher	Mr. Saifullah Adnan
Define Project Scope	R	R	I
Research and Development	R	R	C
System Design	R	A	C
Implementation	R	R	I
Testing	A	R	C
Documentation	R	R	I
Deployment	A	R	I
Maintenance and Support	R	R	C

GANTT CHART:



A1B. COPY OF PROPOSAL EVALUATION COMMENTS BY JURY

Dr. Khurram Iqbal	Satisfactory.	Approved (with revisions)
Engr. Maqsood Khatoon	Although most of theoretical work was complete to further enhance the quality and impact of their work, I believe the students would benefit greatly from additional guidance on the implementation phase. With your valuable mentorship, I am confident they will be able to translate their theoretical knowledge into a practical solution effectively.	
Aqsa Nadir	The student's domain knowledge has significantly improved compared to the previous evaluation. However, a deeper understanding and practical implementation of encryption algorithms are essential. Additionally, it is recommended to develop a mobile application alongside the web application for enhanced usability and accessibility. Since the project report has not been updated after the previous evaluation, it is required to update the report.	

A2. REQUIREMENT SPECIFICATIONS

CryptoCamGuard is a image security application designed to protect sensitive digital images from unauthorized access. The app leverages advanced encryption technologies to ensure secure storage, sharing, and management of visual content. With rising concerns over data breaches and image misuse, CryptoCamGuard addresses the critical need for enhanced privacy, making it an ideal solution for photographers, content creators, businesses, and privacy-conscious individuals. Its user-friendly interface and robust security features offer a reliable safeguard against digital threats while ensuring ease of use for non-technical users.

.1 Purpose of Document

This document specifies the functional and non-functional requirements for CryptoCamGuard, a secure image encryption and sharing application.

.2 Intended Audience

The purpose of this project is to develop CryptoCamGuard, an image security app designed for photographers, content creators, and privacy-conscious users. It ensures secure image storage and transmission through advanced encryption, preventing unauthorized access and safeguarding digital assets.

- Developers
- Testers
- Supervisors
- End Users

- Overall System Description

.1 Project Background

In today's digital era, image security has become a growing concern due to the rising incidents of data breaches, unauthorized access, and misuse of personal media. Content creators, photographers, and privacy-conscious users often face challenges in safeguarding their digital assets, especially when sharing images online or storing them on cloud platforms.

The **CryptoCamGuard (CCG)** project aims to address these vulnerabilities by providing a secure image management solution with advanced encryption techniques. It ensures that only authorized users can access, share, or store images, offering a robust defense against data leaks and unauthorized sharing.

CCG is designed to be user-friendly while integrating cutting-edge security measures, making it ideal for both professional and personal use cases. This project emphasizes privacy, control, and ease of access, ensuring users can protect their visual content with confidence.

.2 Problem Statement

With the increasing use of digital images across various platforms, users face significant risks of unauthorized access, data theft, and image misuse. Existing image security solutions often lack robust encryption and easy-to-use features, leaving sensitive content vulnerable. There is a need for a secure, reliable, and user-friendly application that ensures the protection of images throughout their lifecycle, from storage to sharing, especially for photographers, content creators, and privacy-conscious individuals. CryptoCamGuard aims to fill this gap by providing a comprehensive solution to safeguard digital images against security threats.

.3 Project Scope

CryptoCamGuard will focus on:

- Real-time image encryption.
- Integration with existing surveillance systems.
- User-friendly interface for encryption settings.
- Performance optimization to ensure minimal latency.

.4 Not In Scope

The project will not cover:

- Audio data encryption.
- Cloud storage solutions.
- Integration with non-digital (analog) surveillance systems.

.5 Project Objectives

The primary goal of CryptoCamGuard is to empower users to take control of their digital privacy by offering a secure solution for photo management.

.6 Stakeholders & Affected Groups

Primary Stakeholders:

1. End Users
2. App Developers
3. Product Managers

Secondary Stakeholders:

1. Marketing and Sales Teams
2. Legal and Compliance Teams
3. Security Experts

.7 Operating Environment

The system will operate in a browser-based environment and should support the following:

- **Browsers:** Chrome, Firefox, Safari.
- **Devices:** Desktop, laptop, tablet, and smartphone.

.8 System Constraints

1. **Performance**:
   - Encryption may impact performance on lower-end devices.
   - High storage requirements for encrypted images.
2. **Compatibility**:
   - Need for cross-platform compatibility (Android, iOS, Web).
   - Variability in device performance (e.g., RAM, processor).
3. **Network**:
   - Slow data transfer speeds on low-bandwidth connections.
4. **Security**:
   - Encryption can introduce delays in processing.
   - Compliance with data protection laws (e.g., GDPR, CCPA).
5. **Scalability**:
   - Handling large volumes of image data and user growth.
6. **Usability**:
   - Need for a user-friendly interface despite security features.
   - Consistent experience across platforms.
7. **Budget/Resources**:
   - High development and maintenance costs.
   - Limited time for development and updates.

8. **Legal/Compliance**:
    - Adherence to privacy laws and regulations, affecting features and data management.

.9 Assumptions & Dependencies

Assumptions:

1. Users are aware of the need for image security.
2. The app will be compatible with recent Android and iOS devices.
3. Cloud services for storage will be reliable.
4. Users will have internet access for syncing and sharing images.
5. User-uploaded images will be valid and not corrupted.
6. The app will comply with privacy regulations (e.g., GDPR, CCPA).
7. Stable encryption libraries will be available and updated.

Dependencies:

1. Cloud services (e.g., AWS, Azure) for image storage.
2. Updates to mobile OS (Android, iOS).
3. Encryption libraries for data security.
4. Third-party authentication services (e.g., OAuth, biometrics).
5. Legal compliance with data protection laws.
6. Stable internet access for syncing and sharing.

- External Interface Requirements

.1 Hardware Interfaces

- **Server Requirement**

- Processor: 2 GHz or higher.
- RAM: 8 GB minimum.
- Storage: 100 GB SSD.

- **Client Requirement**

- Any modern device that supports web browser.

.2 Software Interfaces

- **Server Requirement**

- **Backend**: Node.js, Express.js
- **Database**: MongoDB (Mongoose for connection)
- **Frontend**: React.js

.3 Communications Interfaces

- HTTPS for secure data transmission.
- REST APIs for interaction between components.

- System Functions / Functional Requirements

.1 System Functions

Ref #	Functions	Category	Attribute	Details & Boundary Constraints
R1.1	User registration and login	Evident	Response time	Registration should complete within 5 seconds.
R1.2	Image encryption using AES-256	Evident	Security	Ensures secure image storage and transfer.
R1.3	Secure sharing with access controls	Evident	Functionality	Users can specify access permissions.
R1.4	Access monitoring and logging	Hidden	Auditability	Logs access attempts and usage statistics.
R1.5	Cloud-based storage with encryption	Evident	Scalability	Supports dynamic storage allocation.

.2 Use Cases

List of Actors:

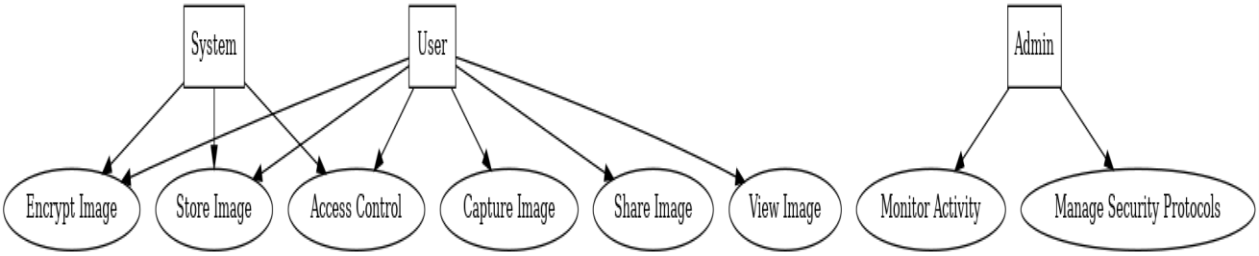
- User
- System Administrator

List of Use Cases:

- Upload and encrypt an image.
- Detect unauthorized access attempts.
- Retrieve and decrypt images.

Use Case #	Name	Brief Description
UC1	Account Registration / Login	Allows users to register securely and create unique profiles.
UC2	Upload Image	Users can upload images for encryption and secure storage.
UC3	Retrieve Image	Users can decrypt and retrieve their encrypted images.
UC4	Detect Intrusions	The system detects and alerts users of unauthorized access.
UC5	Manage Accounts	Administrators can manage user accounts and system settings.

.2.1 Use Case Diagram



.2.2 Description of Use Cases

Section: Main	
Name:	**Account Registration / Login**
Actors:	Users
Purpose:	To allow users with a secure account to access the system.

Description:	Users can create an account or log in using valid credentials.
Cross References:	Functions: R1.1
Pre-Conditions	User must have valid email and password for login or registration information.
Successful Post-Conditions	Upon a successful login, the user is taken to their dashboard.
Failure Post-Conditions	Login fails and error message is displayed.

Typical Course of Events

	Actor Action		System Response
1	User enters registration details or login credentials.		System verifies the credentials and redirects to the appropriate dashboard.
2	Incorrect credentials are provided.		System displays an error message and asks the user to retry or reset password.

Section: Main	
Name:	**Upload and Encrypt Image**
Actors:	User
Purpose:	To allow users to upload images and secure them with encryption.
Description:	Users upload images, which are encrypted and stored in the system.
Cross References:	Functions: R1.2
Pre-Conditions	User must be logged into their account.
Successful Post-Conditions	Image is encrypted and stored successfully.
Failure Post-Conditions	Image upload fails due to unsupported format or network error.

Typical Course of Events

	Actor Action		System Response
1	User selects a image file and clicks upload.		System validates the image format and stores the image in the database.
2	Invalid file format is selected.		System displays an error message and asks the candidate to re-upload.

Section: Main	
Name:	**Share Image Securely**
Actors:	Users, Third-party Viewers
Purpose:	To allow users to share images with specified permissions.
Description:	Users select images to share and set access permissions for specific viewers.
Cross References:	Functions: R1.3
Pre-Conditions	User must have encrypted images stored in the system.
Successful Post-Conditions	Image is accessible only to specified viewers.
Failure Post-Conditions	Sharing fails if permissions are invalid or there is a system error.

Typical Course of Events			
Actor Action			**System Response**
1	User select image and clicks “Share”		System validates the information and grant access for sharing.
2	If image is not stored in the system.		System displays an error message and asks the user to complete missing fields.

Section: Main	
Name:	**Monitor Access**
Actors:	User
Purpose:	To allow users to track access to their shared images.
Description:	Users can view access logs, including time and viewer details.
Cross References:	Functions: R1.4
Pre-Conditions	Images must be shared.
Successful Post-Conditions	Access logs are displayed.
Failure Post-Conditions	Logs fail to display due to a system error.

Typical Course of Events			
Actor Action			**System Response**
1	To allow users to track access to their shared images.		System will verify and displays the images.

Section: Main			
Name:	**Manage Permissions**		
Actors:	User		
Purpose:	To allow users to update or revoke access permissions for shared images.		
Description:	Users modify access settings or revoke permissions as needed.		
Cross References:	Functions: R1.5		
Pre-Conditions	Images must be shared with permissions.		
Successful Post-Conditions	Updated permissions are enforced.		
Failure Post-Conditions	Update fails due to a system error.		
Typical Course of Events			
Actor Action		**System Response**	
1	User selects a shared image and modifies permissions.		System validates and updates permissions.
2	If the update fails, the system displays an error message.		System records the user performance and saves the results.

- Non - Functional Requirements

.1 Performance Requirements

- The system should handle up to 500 concurrent users without significant performance degradation.
- Each operation, including encryption and sharing, should complete within 2 seconds.

.2 Safety Requirements

- The application must prevent unauthorized access by using industry-standard encryption methods.
- All image data transfers must occur over HTTPS.

.3 Security Requirements

- Passwords must be hashed and stored using SHA-256.
- Two-factor authentication should be available for all user accounts.
- Encrypted images must not be accessible without valid decryption keys.

.4 Reliability Requirements

- The system must have an uptime of 99.5% or higher.
- Backup services should run daily to ensure data recovery in case of a failure.

.5 Usability Requirements

- The user interface should be intuitive and accessible, with clear guidance for all primary functions.
- The application should be responsive across various devices, including desktops, tablets, and smartphones.

.6 Supportability Requirements

- The application should support future enhancements without major overhauls.
- Cloud-based architecture must allow scalability to accommodate increased user loads.

.7 User Documentation

- Comprehensive user guides must be provided for all user roles (e.g., Users, Administrators).
- FAQs and troubleshooting steps should be easily accessible within the app.

- References

List References

1. Stallings, W. (2017). Cryptography and Network Security Principles and Practice. Pearson.
2. Schneier, B. (1996). Applied Cryptography: Protocols, Algorithms, and Source Code in C. Wiley.
3. IEEE Standards Association. (2020). IEEE 802.11 Wireless LAN Standards.
4. OWASP Foundation. (2023). OWASP Application Security Verification Standard (ASVS).
5. Amazon Web Services (AWS) Documentation. Retrieved from: https://docs.aws.amazon.com/

A3. DESIGN SPECIFICATIONS

1.1 Purpose of Document

The purpose of this document is to provide a detailed design specification for CryptoCamGuard, a security-focused application designed to enhance image security through advanced encryption and controlled sharing mechanisms.

1.2 Intended Audience

This document is intended for:

- Development team members
- Project managers
- System architects
- QA testers
- Stakeholders and supervisors

1.3 Project Overview

CryptoCamGuard is an innovative application focused on securing digital images using cryptographic techniques. Users can encrypt, store, and share their images securely, ensuring privacy and protection against unauthorized access.

1.4 Scope

CryptoCamGuard will focus on:

- Real-time image encryption.
- Integration with existing surveillance systems.
- User-friendly interface for encryption settings.
- Performance optimization to ensure minimal latency.

Not In Scope

The project will not cover:

- Audio data encryption.
- Cloud storage solutions.
- Integration with non-digital (analog) surveillance systems.

2 Design Considerations

The design of the CryptoCamGuard application takes into account several critical factors to ensure the system's effectiveness, scalability, and security:

2.1 Assumptions and Dependencies

- The application will be accessible via web and mobile platforms.
- Users will have access to a stable internet connection.
- The backend will utilize a secure database like PostgreSQL.
- Hosting will be done on a cloud platform like AWS.

2.2 Risks and Volatile Areas

- Evolving security threats necessitating regular updates.
- Scalability issues with a growing user base.
- User adoption challenges for non-tech-savvy individuals.

2.3 Scalability and Performance

- The application should handle a high volume of image uploads and downloads.
- Load balancing will ensure performance consistency during peak usage.
- Caching mechanisms will reduce latency.

3 System Architecture

The CryptoCamGuard application is designed using a three-tier architecture to ensure scalability, robustness, and security. This architecture decomposes the system into subsystems or components, each with specific responsibilities that work together to provide the desired functionality.

Overview of Architecture

CryptoCamGuard follows a three-tier architecture:

1. **Presentation Layer:** Web and mobile interface.
2. **Application Layer:** Business logic and processing.
3. **Data Layer:** Secure storage and encryption.

3.1 System Level Architecture

The system architecture of CryptoCamGuard is designed to ensure a secure, scalable, and robust application. This section elaborates on the decomposition of the system into elements, their relationships, and integration with external systems.

3.1.1 System Decomposition into Elements

CryptoCamGuard is divided into the following primary components:

- **Presentation Layer:** This includes the user-facing components such as web and mobile applications built with React Native. It provides a user-friendly interface for uploading, encrypting, and sharing images.
- **Application Layer:** This layer is responsible for executing the business logic of the system. It consists of REST APIs developed using Django REST Framework, which handle encryption, user authentication, and image management.
- **Data Layer:** This layer stores all the data, including encrypted images, user information, and logs. PostgreSQL serves as the primary database for structured data, while AWS S3 stores encrypted image files.

3.1.2 The Relationship between the Elements

The elements of CryptoCamGuard interact through clearly defined interfaces to ensure smooth communication:

1. **User Interaction:** Users interact with the system through the web or mobile application (presentation layer), which forwards requests to the backend (application layer).
2. **API Communication:** The application layer processes these requests and communicates with the data layer using secure APIs to retrieve or update data.
3. **Data Storage:** Encrypted images and metadata are securely stored in the data layer, ensuring data integrity and availability.

3.1.2.1 Interfaces to External Systems

CryptoCamGuard integrates with external systems to enhance functionality and security:

- **Email Service:** Used for account verification, notifications, and alerts.
- **Authentication Provider:** OAuth 2.0 is used to authenticate users securely and manage tokens.
- **Cloud Storage Service:** AWS S3 is used to store encrypted images, ensuring scalability and durability.

3.1.3 Major Physical Design Issues

Several key considerations are addressed to ensure the system's physical design aligns with its requirements:

1. **Server Load Balancing:** Multiple application servers are deployed with load balancers to handle high user traffic efficiently.
2. **Data Redundancy:** Database and storage redundancy are implemented to prevent data loss and ensure high availability.
3. **Secure Communication:** All data exchanges between the layers and external systems are encrypted using HTTPS and TLS protocols.

3.2 Software Architecture

The software architecture defines the internal design of CryptoCamGuard, emphasizing modularity, scalability, and security.

3.2.1 User Interface Layer

The user interface layer is responsible for presenting data to the user and capturing input:

- **Technologies:** Built using React Native to ensure a seamless experience across web and mobile platforms.
- **Responsibilities:**
  - Rendering user interfaces.
  - Handling user interactions like uploading images and setting permissions.
  - Sending API requests to the backend.

3.2.2 Middle Tier

The middle tier handles the application's core functionalities:

- **Components:** Django REST Framework APIs and business logic modules.
- **Responsibilities:**

- Validating user inputs and ensuring they meet security standards.
- Implementing encryption algorithms (e.g., AES-256) for image security.
- Managing session states and user authentication.

3.2.3 Data Access Layer

The data access layer is responsible for securely storing and retrieving data:

- **Technologies:** PostgreSQL for structured data and AWS S3 for encrypted image files.
- **Responsibilities:**
  - Performing CRUD operations for user data, permissions, and audit logs.
  - Ensuring data consistency and integrity.
  - Enforcing access controls to prevent unauthorized data access.

4 Design Strategy

The design strategy focuses on creating a robust, scalable, secure, and user-friendly platform through the following key principles:

4.1 Modular Architecture

Each module (encryption, authentication, sharing) is independently designed for scalability and maintainability.

4.2 Scalability and Performance

- Horizontal scaling with containerized services.
- Use of CDN for faster image delivery.

4.3 Security

- End-to-end encryption for all data.
- Regular penetration testing.

4.4 User Experience (UX)

- Intuitive design with accessible features for all user levels.
- Real-time feedback for user actions.

4.5 Maintainability

- Modular code with comprehensive documentation.
- Automated CI/CD pipelines for seamless updates.

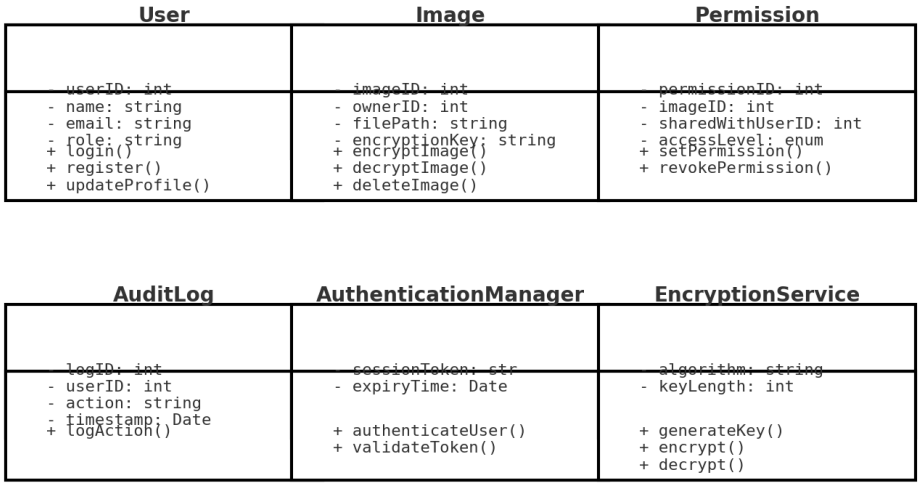
4.6 Reliability and Availability

- **Redundancy:** Prevent single points of failure with redundant servers and data replication.
- **Monitoring and Alerts:** Track performance and health, with alerts for issues.

5 Detailed System Design

[A detailed design should include the following:

5.1 Design Class Diagram

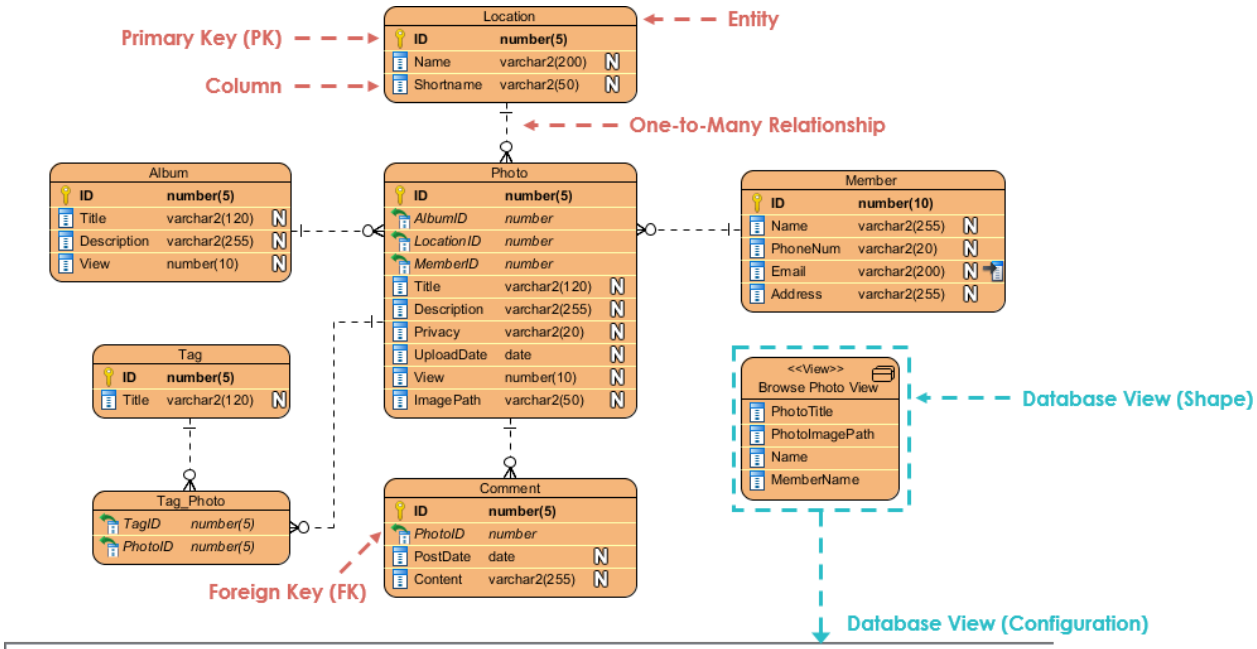


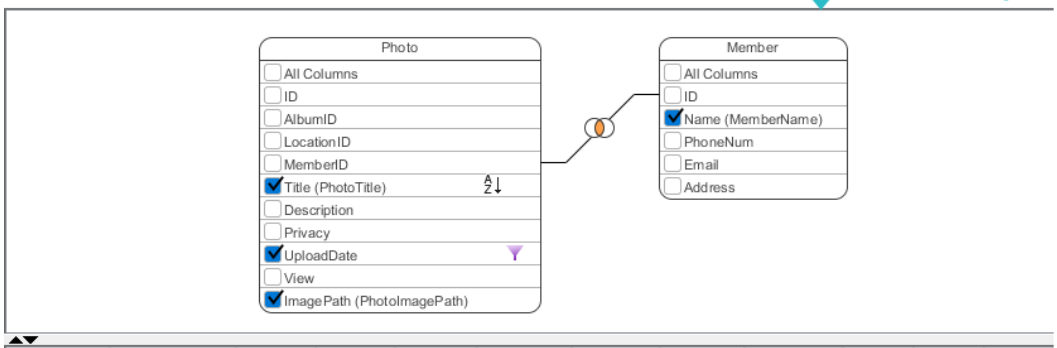
5.2 Database Design

[A detailed Database design should include the following:

- *Logical data model (E/R model)*
- *Data dictionary]*

5.3 ER Diagram



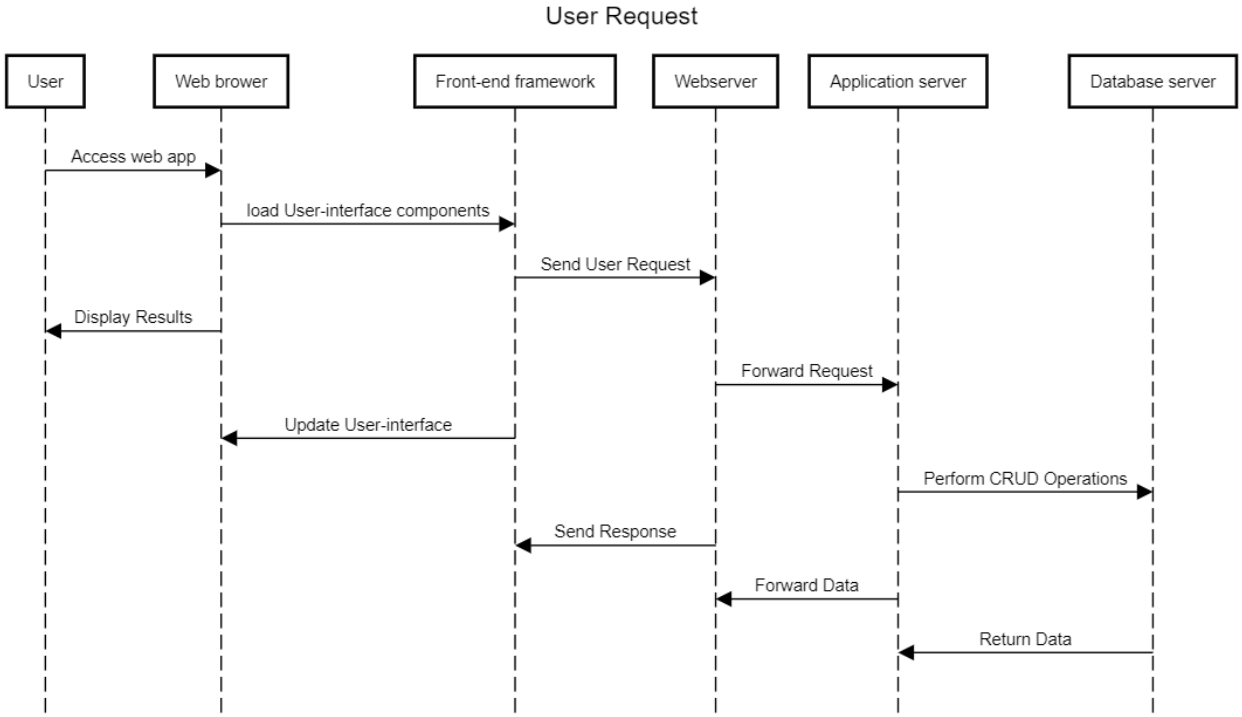


Column	Alias	Entity	Output	Sort	Sort Order	Group By	Function	Filter	Filter	Filter	Filter
Title	PhotoTitle	Photo	☑	Ascending	1	☐					
ImagePath	PhotoImagePath	Photo	☑			☐					
Name	MemberName	Member	☑			☐					
UploadDate		Photo	☑			☐		> sysdate-5			
			☐			☐					

5.4 Application Design

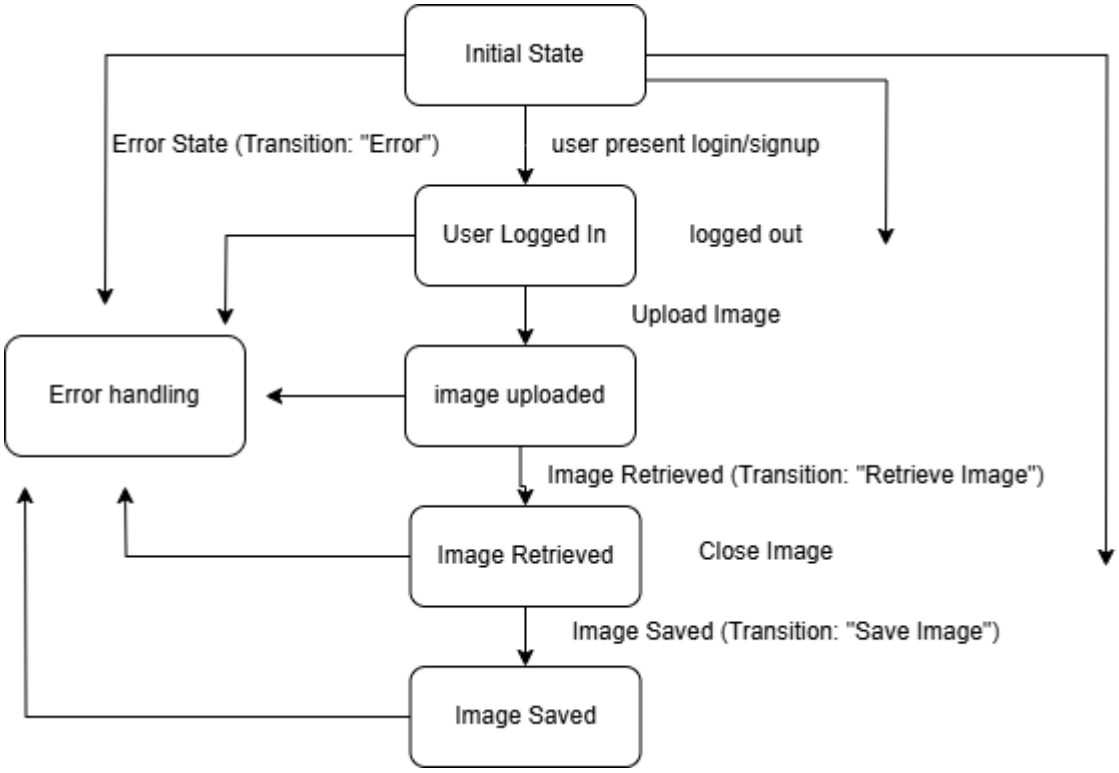
5.4.1 Sequence Diagram

5.4.1.1 <Sequence Diagram 1>

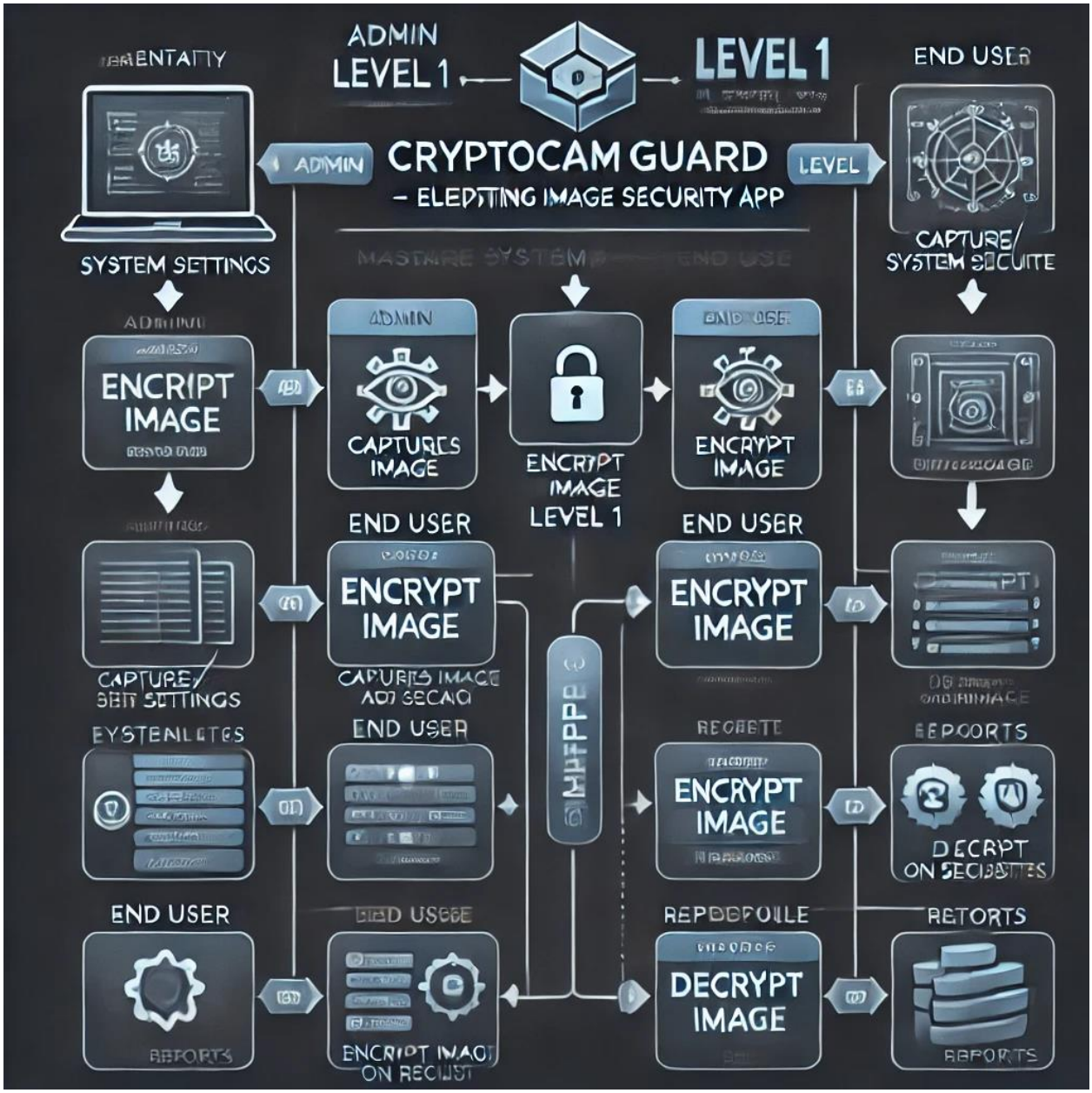


5.4.2 State Diagram

5.4.2.1 <State Diagram 1>

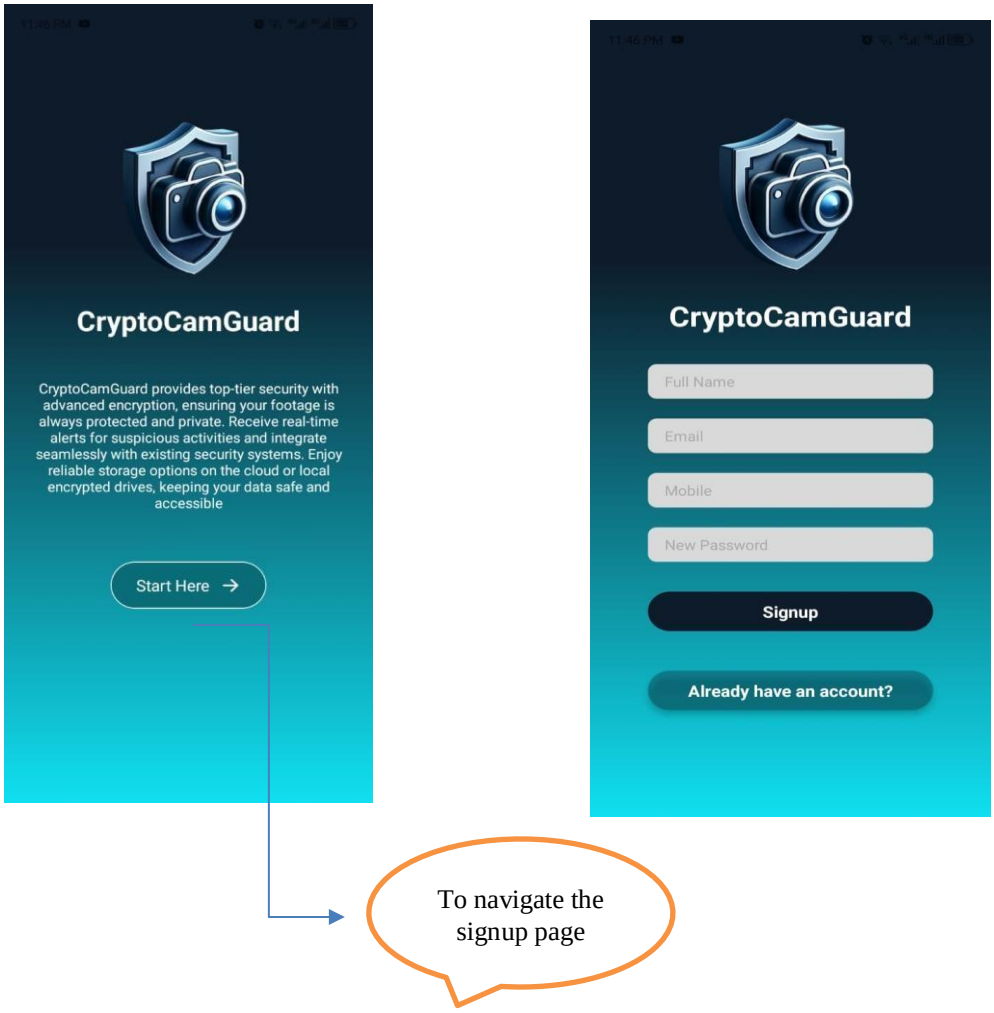


5.4.3 DFD level 1

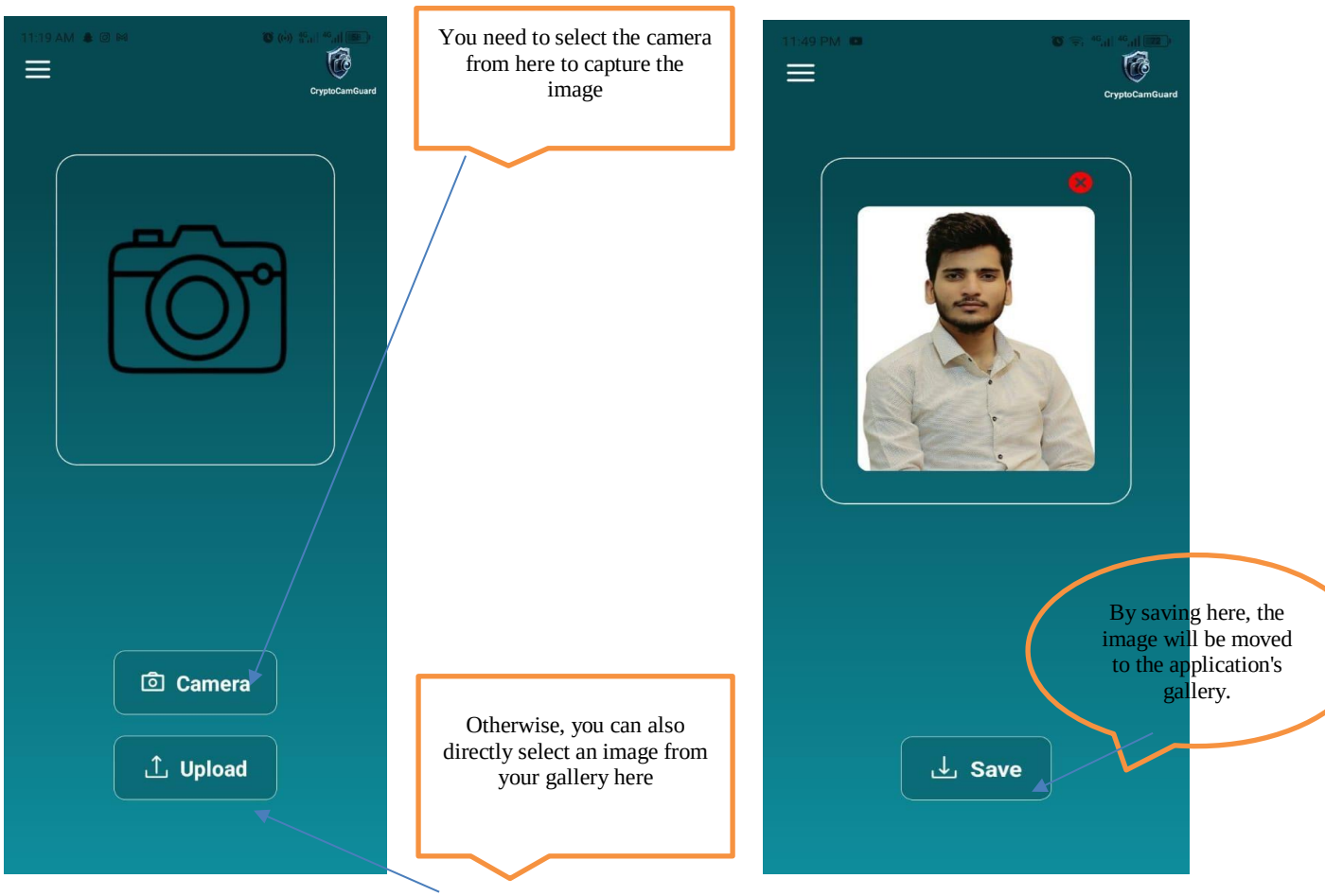


5.5 GUI Design

5.5.1 App Landing Page



5.5.2 Project Core Features



- Once the user takes a photo, it is not stored in the device's gallery. Instead, the image is processed within the app and is prepared for encryption.
- The user can choose to capture additional photos or proceed to encryption immediately.

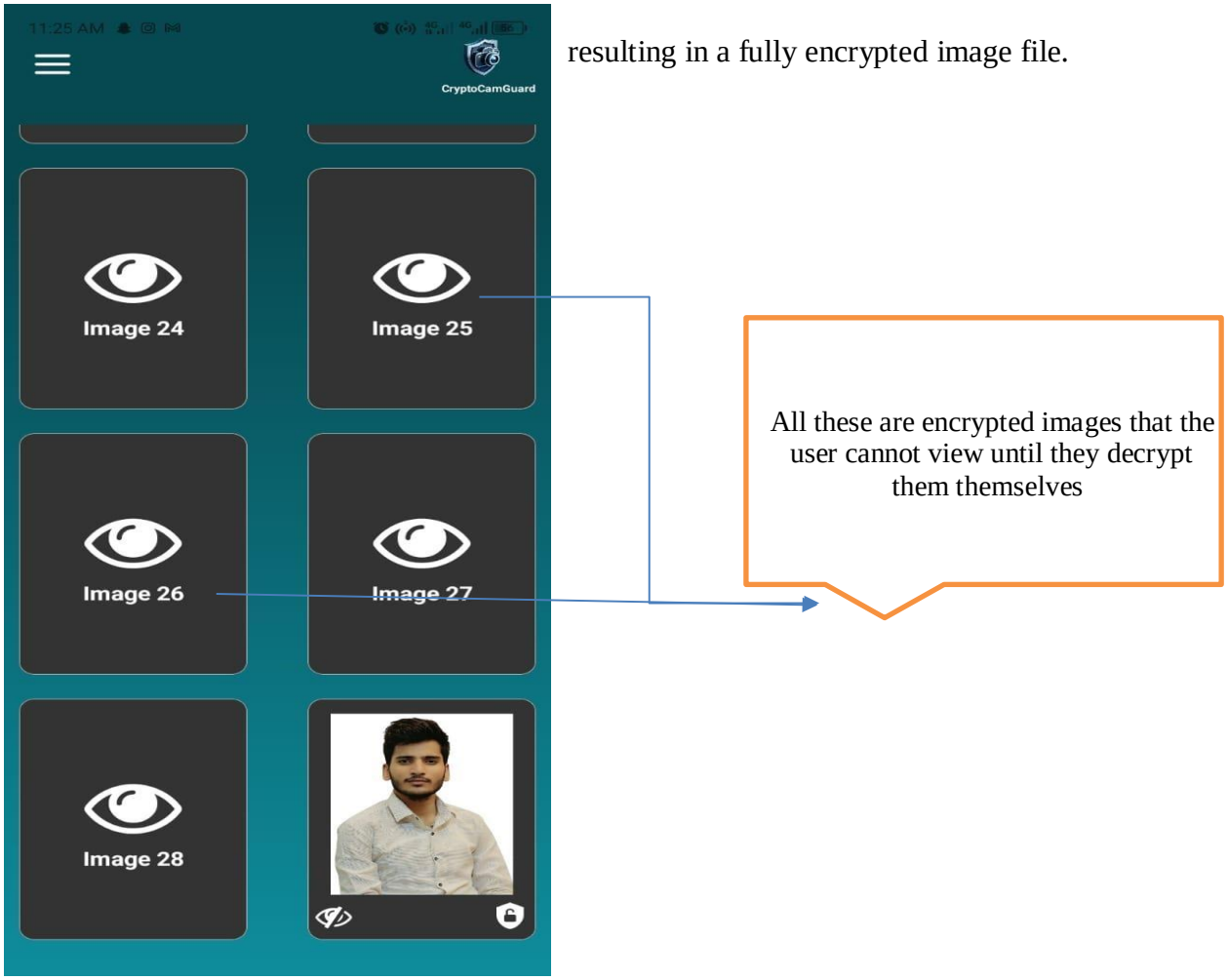
5.5.3 Encryption Process

After the image is captured, **CryptoCamGuard** immediately encrypts the photo to ensure that it is securely stored and inaccessible to unauthorized users.

- **Encryption**:
- The app utilizes **(Advanced Encryption Standard)**, which is one of the most secure encryption algorithms available. This ensures that

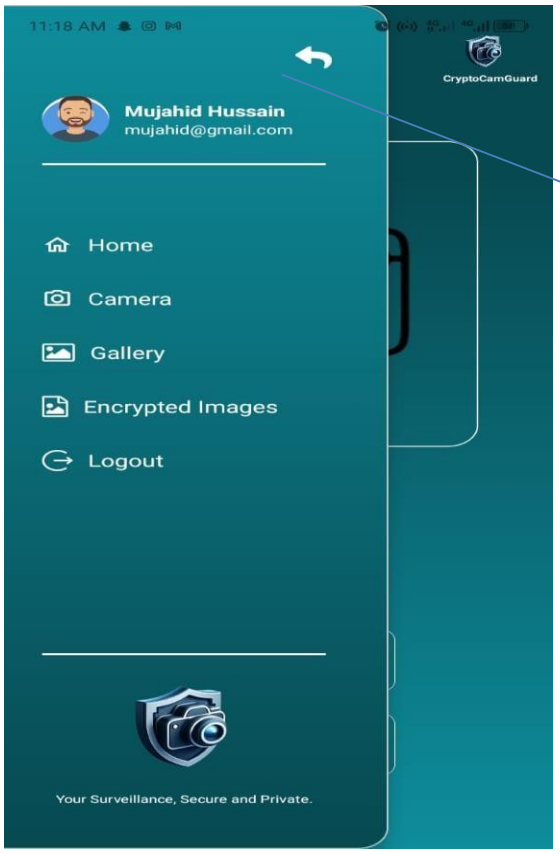
even if the photo is intercepted or accessed on the device, it remains unreadable without the correct decryption key.

- The encryption occurs locally on the device, meaning that the image is encrypted before it is stored or transmitted, ensuring end-to-end security.
- **Encryption Flow**:
- The app generates a unique encryption key for each image, ensuring that each photo is individually encrypted.
- The encryption key and the image binary data are processed through the algorithm,



resulting in a fully encrypted image file.

5.5.4 Sidebar



With the help of this sidebar, the user can navigate to the homepage. If they want, they can also open the camera from here. Additionally, if the user wants to sign out, they can go to the sidebar and click the sign-out button.

- **Storing the Encrypted Image**:

- Once the image is encrypted, it is stored locally in the app's secure storage area. This prevents the image from appearing in the device's default gallery or cloud-based services, keeping it isolated and protected.

- The encrypted image file is also associated with metadata, such as a timestamp and its encryption key (which is also securely stored).

6 References

Tresorit

uses encryption technology to protect your files end-to-end, meaning no third-party or service provider can access them from upload to download.Overall, Tresorit is a safe and secure way to store and share your sensitive data. For more information, visit their website directly https://tresorit.com/

PixelKnot

is an Android app that allows you to hide messages within images and share them securely through any messaging platform. Its main purpose is to keep your images secure and aid in sharing them confidentially. PixelKnot utilizes steganography, meaning it hides your messages within the pixels of images, making them appear normal but containing hidden information. PixelKnot: Hidden Messages - Apps on Google Play

7 Appendices

[Include supporting detail that would be too distracting to include in the main body of the document.]

A4. OTHER TECHNICAL DETAIL DOCUMENTS

1. Test Cases Document

Test Case 1: Image Encryption

Field	Details
Test Case ID	TC001
Test Objective	Verify that an image is encrypted successfully
Preconditions	User is logged in and has selected an image
Test Steps	1. Open the app 2. Select image 3. Click “Encrypt”
Expected Result	Image is encrypted and stored securely
Actual Result	*Encrypted image stored*
Status	Pass

Test Case 2: Image Decryption

Field	Details
Test Case ID	TC002
Test Objective	Verify that an encrypted image is decrypted correctly
Preconditions	Encrypted image exists
Test Steps	1. Open app 2. Select encrypted image 3. Click “Decrypt”
Expected Result	Original image is restored
Actual Result	*Image decrypted successfully*
Status	Pass

Test Case 3: Prevent Unauthorized Access

Field	Details
Test Case ID	TC003
Test Objective	Ensure unauthorized users cannot access encrypted images
Preconditions	App is launched by unauthenticated user
Test Steps	1. Open app without login 2. Try to view encrypted image
Expected Result	Access is denied or redirected to login
Actual Result	*Access denied*
Status	Pass

Test Case 4: Image Upload Limit Validation

Field	Details
Test Case ID	TC004
Test Objective	Verify that system restricts images above a certain size (e.g. 10MB)
Preconditions	App is running
Test Steps	1. Select a 20MB image 2. Click “Encrypt”
Expected Result	System shows error: “Image size too large”
Actual Result	*Error displayed as expected*
Status	Pass

Test Case 5: User Interface Responsiveness

Field	Details
Test Case ID	TC005
Test Objective	Verify that the app UI adapts properly on various screen sizes
Preconditions	App is installed on different devices
Test Steps	1. Open app on mobile, tablet, and desktop
Expected Result	UI should adapt without layout issues
Actual Result	*Responsive on all devices*
Status	Pass

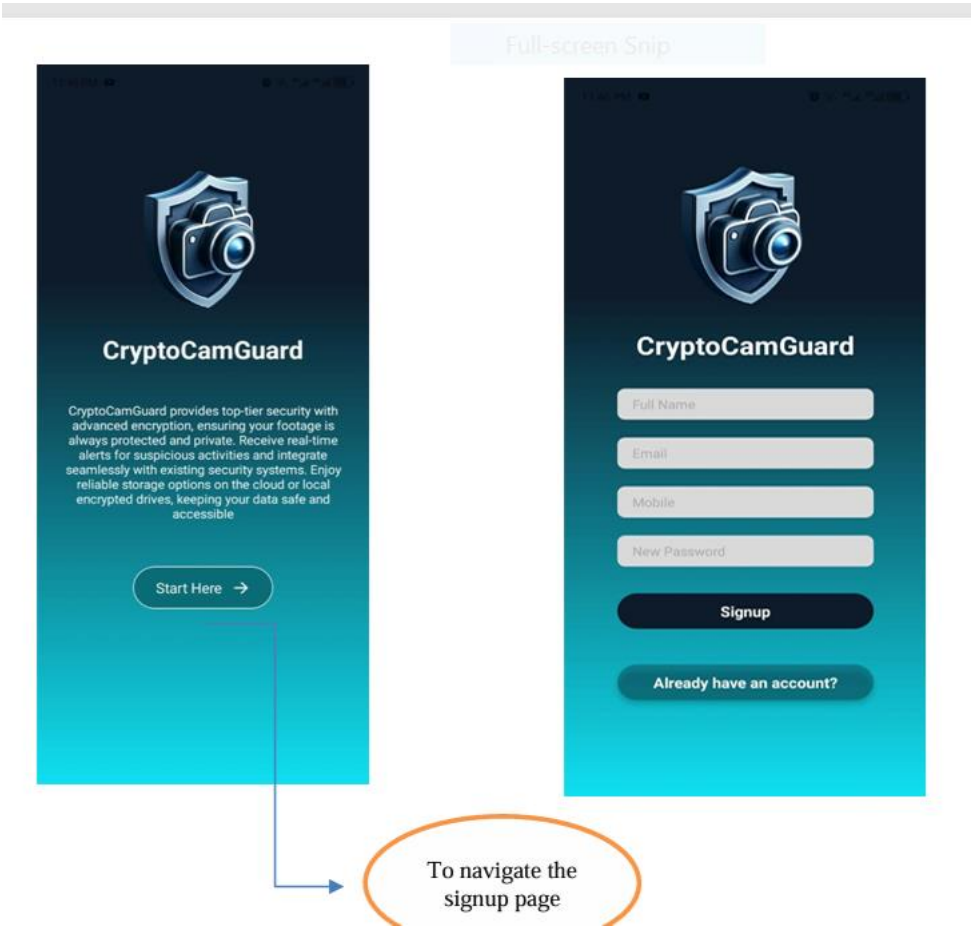
UI/UX Detail Document

Client-Side (Mobile Application)

The Client-Side of CryptoCamGuard is the user-facing component of the system, built using React Native to ensure compatibility with both iOS and Android platforms. The mobile application is designed to handle all interactions with the user, from account management (signup and login) to the core functionalities of image capture and encryption. Below is a breakdown of the key processes and features handled on the client-side.User Interface (UI): The UI is designed with an intuitive, user-friendly approach, allowing users to easily navigate between functionalities such as capturing images, viewing encrypted images, and managing their accounts.

Signup and Login Functionality

The first interaction users have with CryptoCamGuard is through the signup and login process. This ensures that only authenticated users can access the application's features, especially the secure image storage functionality.'



Signup Process:

- Users create an account by providing basic details such as their email

address and password.

- The system applies password strength rules to ensure security, requiring users to create strong passwords.
- Once the information is submitted, a secure API call is made to the backend, where the user's data is stored securely after hashing the password.
- Upon successful signup, the user is prompted to log in.

Login Process:

- For returning users, the login screen allows them to authenticate using their email and password.
- Once the user enters valid credentials, the app sends an authentication request to the backend, where **JWT (JSON Web Tokens)** are used for secure authentication.
- If the login is successful, the backend issues a JWT, which is stored locally on the user's device for the session duration. This token is required for accessing protected routes in the app, such as image capture and viewing encrypted photos.

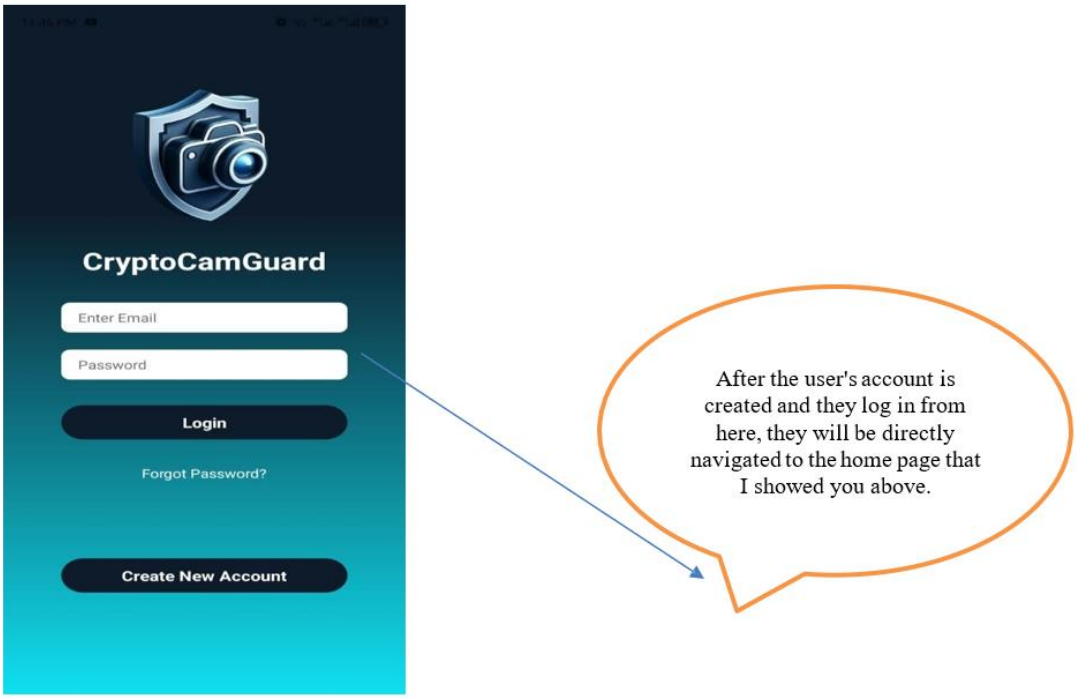


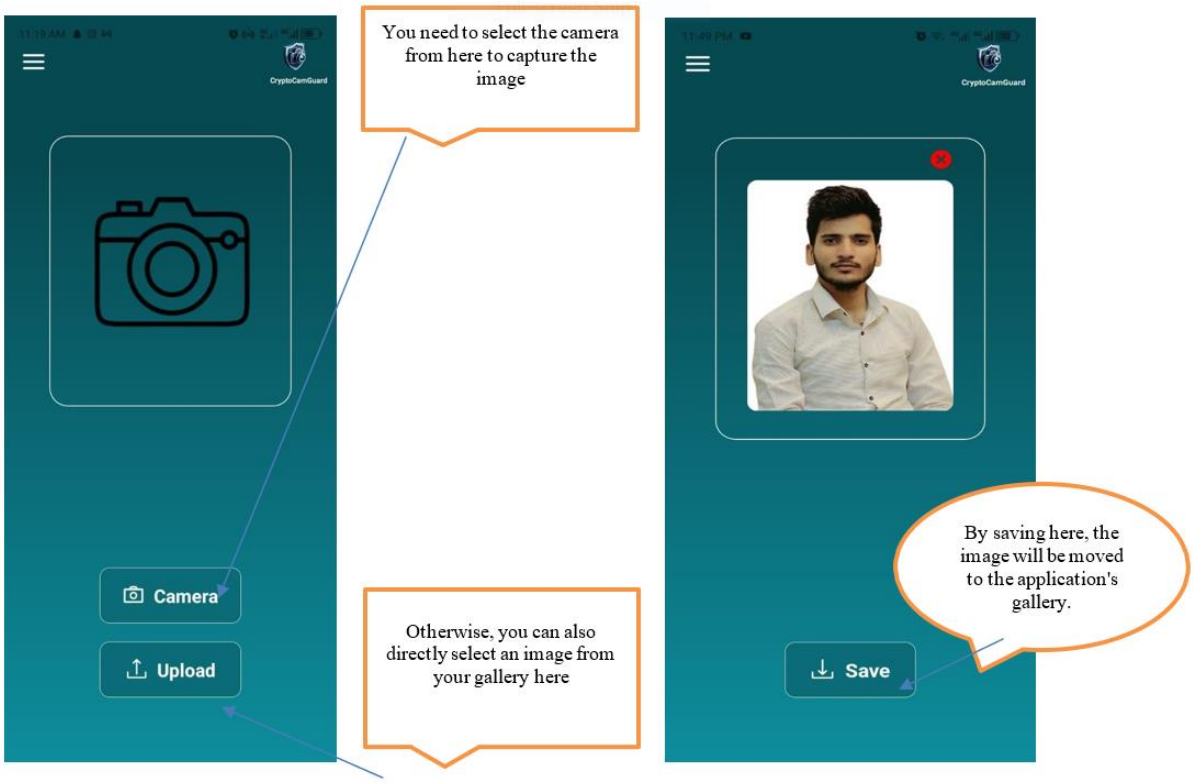
Image Capture Functionality

Once authenticated, users can access the core functionality of **CryptoCamGuard**: capturing and encrypting images.

- ***Accessing the Camera****:*
    - Users are presented with a camera interface within the app, allowing them to take photos without leaving the application.
    - The app leverages the native camera capabilities of the mobile device, ensuring high- quality image capture while maintaining a smooth user experience.
    - The camera module is integrated using **React Native's Camera APIs**, which provide control over resolution, focus, and other camera settings.
- ***Taking a Picture****:*
    - Once the user takes a photo, it is not stored in the device's gallery. Instead, the image is processed within the app and is

prepared for encryption.

- The user can choose to capture additional photos or proceed to encryption immediately.



Encryption Process

After the image is captured, **CryptoCamGuard** immediately encrypts the photo to ensure that it is securely stored and inaccessible to unauthorized users.

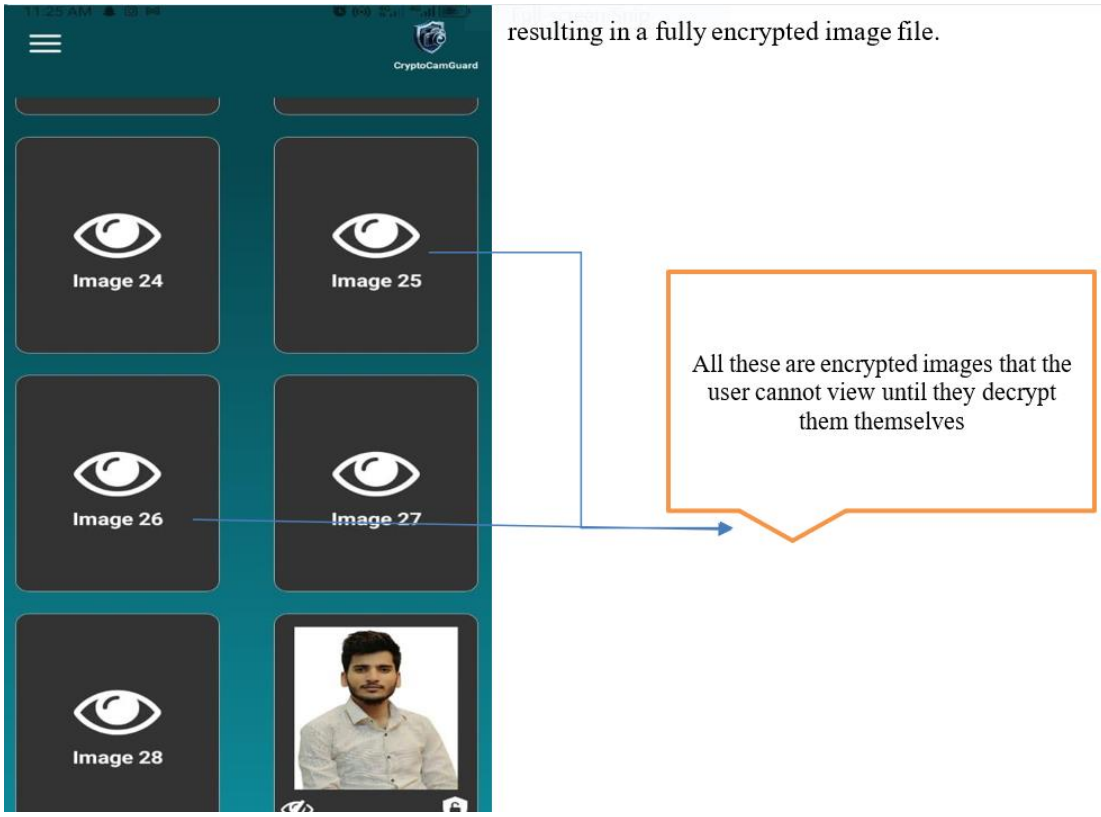
- ***Encryption***:
  - The app utilizes **(Advanced Encryption Standard)**, which is one of the most secure encryption algorithms available. This ensures that even if the photo is intercepted or accessed on the device, it remains unreadable without the correct decryption key.
  - The encryption occurs locally on the device, meaning that the image is encrypted before it is stored or transmitted, ensuring

end-to-end security.

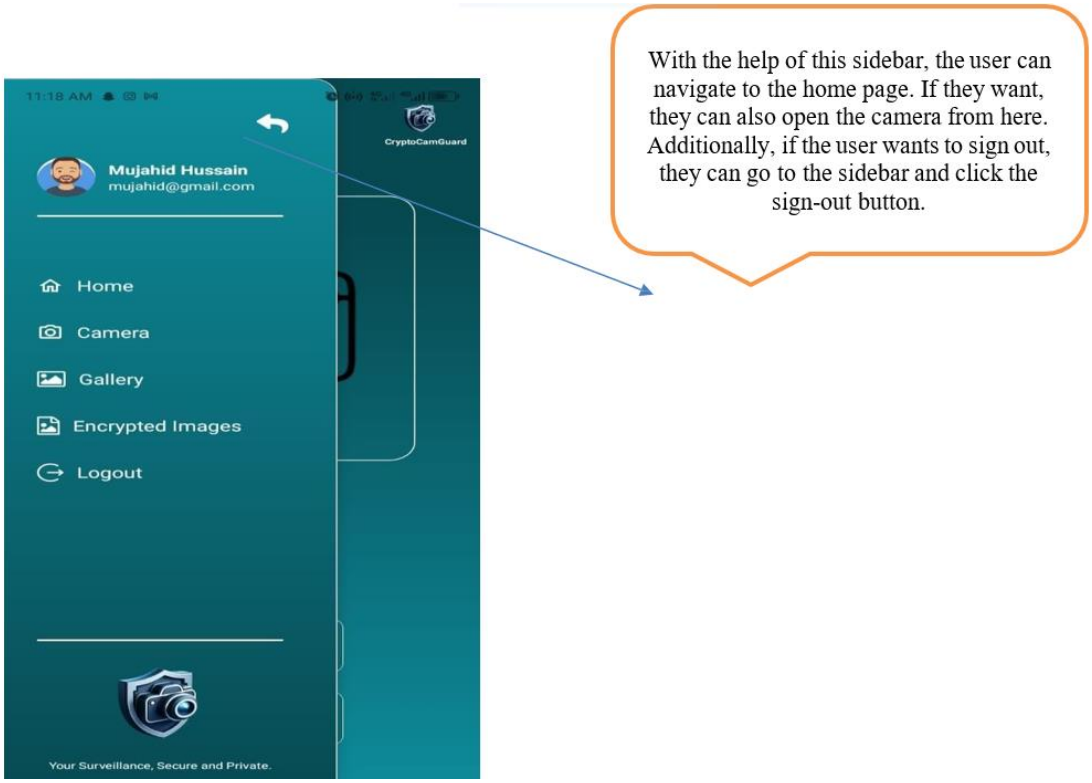
- ***Encryption Flow**:*

  - The app generates a unique encryption key for each image, ensuring that each photo is individually encrypted.

  - The encryption key and the image binary data are processed through the algorithm,



resulting in a fully encrypted image file.



- ***Storing the Encrypted Image**:*

    - Once the image is encrypted, it is stored locally in the app's secure storage area. This prevents the image from appearing in the device's default gallery or cloud-based services, keeping it isolated and protected.

    - The encrypted image file is also associated with metadata, such as a timestamp and its encryption key (which is also securely stored).

b. CODING STANDARDS

To ensure that the development of *CryptoCamGuard* is consistent, maintainable, secure, and efficient, the following coding standards were followed throughout the project lifecycle. These practices contributed to code clarity, team collaboration, and future scalability of the application.

1. Code Formatting

Consistent code formatting was maintained throughout the application to ensure readability and professional coding practices. Key formatting rules included:

- **Indentation:** 4-space indentation was used across all source files.
- **Line Length:** No line exceeded 80–100 characters to enhance readability.
- **Spacing:** Proper spacing was maintained around operators, after commas, and before braces.
- **Braces:** Braces were always used with conditional and loop blocks, even for one-liners, to avoid logical errors.
- **Commenting:** Inline and block comments were used to explain logic, especially in encryption algorithms, user input validation, and UI controls.

This standard helped reduce confusion during debugging and ensured uniformity across modules.

2. Naming Conventions

A clear and consistent naming convention was applied for variables, functions, classes, and files:

- **Variables:** Written in **camelCase** (e.g., `selectedImagePath`, `encryptionKey`).
- **Constants:** Written in **UPPER_SNAKE_CASE** (e.g., `MAX_IMAGE_SIZE`, `DEFAULT_EXTENSION`).
- **Functions/Methods:** Followed **camelCase** (e.g., `encryptImage()`, `validateInput()`).
- **Classes:** Used **PascalCase** (e.g., `ImageEncryptor`, `SecurityManager`).
- **File Names:** Used lowercase with underscores for clarity (e.g., `image_utils.py`, `user_auth.py`).

Using consistent naming made the code self-explanatory and easier to navigate during development and testing.

---

3. Error Handling

Robust error handling was implemented to ensure stability and a good user experience:

- **Try-Except Blocks:** Used throughout the application to catch exceptions like file access errors, unsupported formats, or null references.
- **Custom Messages:** Descriptive error messages were displayed to users in case of failure during encryption, decryption, or image upload.
- **Logging (Optional):** Debugging information was optionally printed to the console for developers.
- **Fail-Safe Measures:** In case of any unexpected exception, the system reverted to a safe state without crashing.

This strategy ensured the app could gracefully handle unexpected inputs or behavior.

4. File Organization

Code was modularized into logical packages such as:

- `encryption/` for encryption-related functions.
- `ui/` for user interface code.
- `auth/` for login/authentication logic.

Each file had a clear responsibility and was documented with comments.

Unused files and dead code were removed to keep the structure clean.

5. Version Control

The Git version control system was used to manage the project's source code:

- **Commits:** Regular commits were made with meaningful messages (e.g., "Implemented AES encryption", "Fixed UI bug in image preview").
- **Branching:** Feature branches were created for major modules (e.g., `encryption-module`, `login-feature`).
- **Gitignore:** A `.gitignore` file was used to exclude unnecessary files like `__pycache__`, logs, and temporary files.
- **Backup & Collaboration:** Git ensured that progress was never lost and the project could be safely restored at any stage.

Using Git also demonstrated real-world software engineering practices.

6. Performance

Performance considerations were taken seriously during development, especially because image encryption can be resource-intensive:

- **Memory Efficiency:** Image data was processed in chunks when needed to avoid memory overload.
- **Minimal Dependencies:** Only essential third-party libraries were used to keep the application lightweight.
- **Code Optimization:** Redundant computations and unnecessary UI updates were eliminated to improve response time.
- **Lazy Loading:** Images were loaded only when required.

These steps helped deliver a fast and smooth user experience even on moderate hardware.

7. Security

As CryptoCamGuard is a security-focused application, data protection was the highest priority:

- **Advanced Encryption:** AES (Advanced Encryption Standard) was used for robust image encryption.
- **Key Management:** Encryption keys were not hardcoded and were securely generated and stored.
- **Authentication:** Basic login was required to restrict unauthorized access to the app.
- **Validation:** All user inputs were validated to prevent crashes or abuse.
- **No Hardcoded Credentials:** Any sensitive data (like file paths or keys) was kept secure.

These practices ensured that images remained protected from unauthorized access.

8. Review and Testing

Code quality was maintained through regular review and testing:

- **Code Review:** Each module was manually reviewed for correctness, security, and performance.
- **Manual Testing:** All user interactions were tested on various screen sizes and usage conditions.
- **Test Cases:** A set of predefined test cases were executed to validate each function (as documented earlier).
- **Debugging Tools:** Built-in debugging tools were used to trace and resolve runtime issues.

Testing and reviews helped maintain a high standard of quality and reliability.

c. USER DOCUMENT

1. Accessing the Application

• Launch the CryptoCamGuard application by double-clicking the `main.py` file or using the command `python main.py` in the terminal.
• The application login screen will appear.
• Enter your registered username and password to access the application dashboard.

2. Exploring the Dashboard

• Once logged in, you will see the main dashboard of CryptoCamGuard.
• The dashboard contains the following core functionalities:

- Image Encryption
- Image Decryption
- Encrypted Image Directory
- User Logout

3. Encrypting an Image

• Click on the **"Encrypt Image"** button on the dashboard.
• A file browser window will open.
• Select the image you want to encrypt (supported formats: `.jpg`, `.jpeg`, `.png`).
• Click **"Open"** to upload the image.
• The system will encrypt the image using AES encryption and save it with a `.enc` extension in the encrypted folder.
• A success message will confirm the encryption.

4. Decrypting an Image

• Click on the **"Decrypt Image"** button on the dashboard.
• Select an encrypted `.enc` image file from your system.
• Click **"Open"** to proceed.
• The application will decrypt the image and restore it to its original form in a selected output directory.
• A notification will confirm that the decryption is successful.

5. Viewing Encrypted Files

• Click on the **"View Encrypted Files"** option to see all previously encrypted images.
• You can preview encrypted filenames, encryption time, and file size.
• This helps you keep track of your secure data in one place.

6. Logging Out

• Click on the **"Logout"** button located at the top right corner of the dashboard.
• This will securely end your session and return you to the login screen.

7. Error Notifications

• If an invalid file format is selected, the system will show a warning message.
• If you try to decrypt an unencrypted file, an error popup will be displayed.
• In case of login failure, an "Invalid Credentials" alert will be shown.

8. Saving and Managing Files

• Encrypted and decrypted images are automatically saved in their respective folders.
• The user does not need to manually specify the location each time.
• Files are saved with timestamped names to avoid duplication.

9. Exiting the Application

• Click on the close (`X`) icon on the top right corner of the application window.
• You can also use the `Exit` button if provided on the dashboard.

d. Project Policy Document

1. Introduction

This Project Policy Document provides a structured overview of the planning, development, and management policies followed during the creation of **CryptoCamGuard**, an image security application developed over a span of 12 months. The aim of the project was to provide users with a secure, offline method of encrypting and decrypting personal images using strong encryption algorithms. With increasing concerns over digital privacy, this tool ensures that sensitive image files are stored in a protected form on the user's own system, free from online vulnerabilities.

2. Objectives

The major objectives of the CryptoCamGuard project were:

• To develop a fully functional desktop-based application that enables image encryption and decryption.
• To implement AES encryption to protect image confidentiality.
• To design an intuitive graphical interface for user interaction using Python Tkinter.
• To provide authentication-based access for enhanced data protection.
• To allow users to manage and store their encrypted images securely in local folders.
• To ensure that the system works efficiently with commonly used image formats and runs on mid-level hardware.

3. Scope

The scope of CryptoCamGuard is limited to local usage on personal systems for securing image files:

• **Users:** Designed for students, professionals, or anyone wishing to protect personal photos offline.
• **Platform:** Desktop-based; compatible with Windows and Linux systems.
• **Supported Files:** JPG, JPEG, and PNG image formats.
• **Functionality:** Image encryption, decryption, user authentication, and file handling.
• **Security:** No cloud storage or online transfer—everything is handled offline to maintain user privacy.

4. Implementation Plan

The project was divided into four main phases across 12 months:

• Phase 1 – Research & Analysis (Month 1–3)

- Conducted research on encryption algorithms.
- Selected AES as the primary method for image security.
- Finalized project goals, tools, and technologies.

• Phase 2 – Design & Development (Month 4–7)

- Designed the UI.
- Developed encryption and decryption logic.
- Implemented user login system and file handling.

• Phase 3 – Testing & Refinement (Month 8–10)

- Performed manual testing on each module.
- Created and executed test cases.
- Collected feedback and improved UI, error handling, and responsiveness.

• Phase 4 – Documentation & Deployment (Month 11–12)

- Prepared user manual, testing reports, and project documentation.
- Finalized application version for local deployment and academic submission.

5. Project Team and Responsibilities

The CryptoCamGuard project was developed by a 2-member team:

Team Member	Role & Responsibility
Sardar Nazeer	Frontend Design (Tkinter), Backend Logic (Python), encryption & decryption module development, GUI integration, system architecture, testing, and report writing.
Ali Sher	Assisted in logic implementation, error handling, user feedback collection, documentation support, testing, and final review.

The clear division of responsibilities helped ensure that the workload was balanced and progress was steady.

6. Timeline

The project was executed over a full academic year (12 months). Here's the summarized monthly breakdown:

- **Month 1–3:** Requirement gathering, research on AES and similar algorithms, tool selection.
- **Month 4–7:** Coding core features (encryption, decryption, authentication), UI creation.
- **Month 8–10:** Testing, debugging, usability improvements, and performance checks.
- **Month 11–12:** Final documentation, deployment, and preparation for the viva and submission.

Two Weekly team check-ins and supervisor reviews ensured milestones were achieved on time.

7. Budget

The budget requirements for this project were minimal, as open-source tools and local development resources were used:

Item	Cost
.Net & Libraries	Free
Tkinter GUI Toolkit	Free
Testing & Debug Devices	Own Systems
Printing & Binding	PKR 1,000–1,500 approx.
Miscellaneous	Minor/Personal expenses

Total Cost: Under PKR 2,000 (self-funded)

8. Risk Management

Several risks were identified during the project, and appropriate measures were taken:

• **Encryption Risk:** Ensured AES encryption was properly implemented and tested with different image sizes.
• **Data Risk:** Created separate folders for encrypted and decrypted images to avoid accidental overwrites.
• **Timeline Risk:** Work was divided monthly and closely tracked using weekly plans.
• **User Confusion:** Designed a simple and clean interface to eliminate complexity for non-technical users.

9. Stakeholder Communication

Effective communication with stakeholders (especially the supervisor) was maintained throughout the year:

• Weekly status meetings were conducted to share progress and get feedback.
• Major features were reviewed before integration.
• All documents and source code were shared periodically for transparency and review.
• Supervisor feedback was applied during the development and testing phases to improve quality.

FYP POSTER

HAMDARD UNIVERSITY
جامعہ ہمدرد

S15

PROJECT NAME

CRYPTO CAM GUARD - ELEVATING IMAGE SECURITY APP

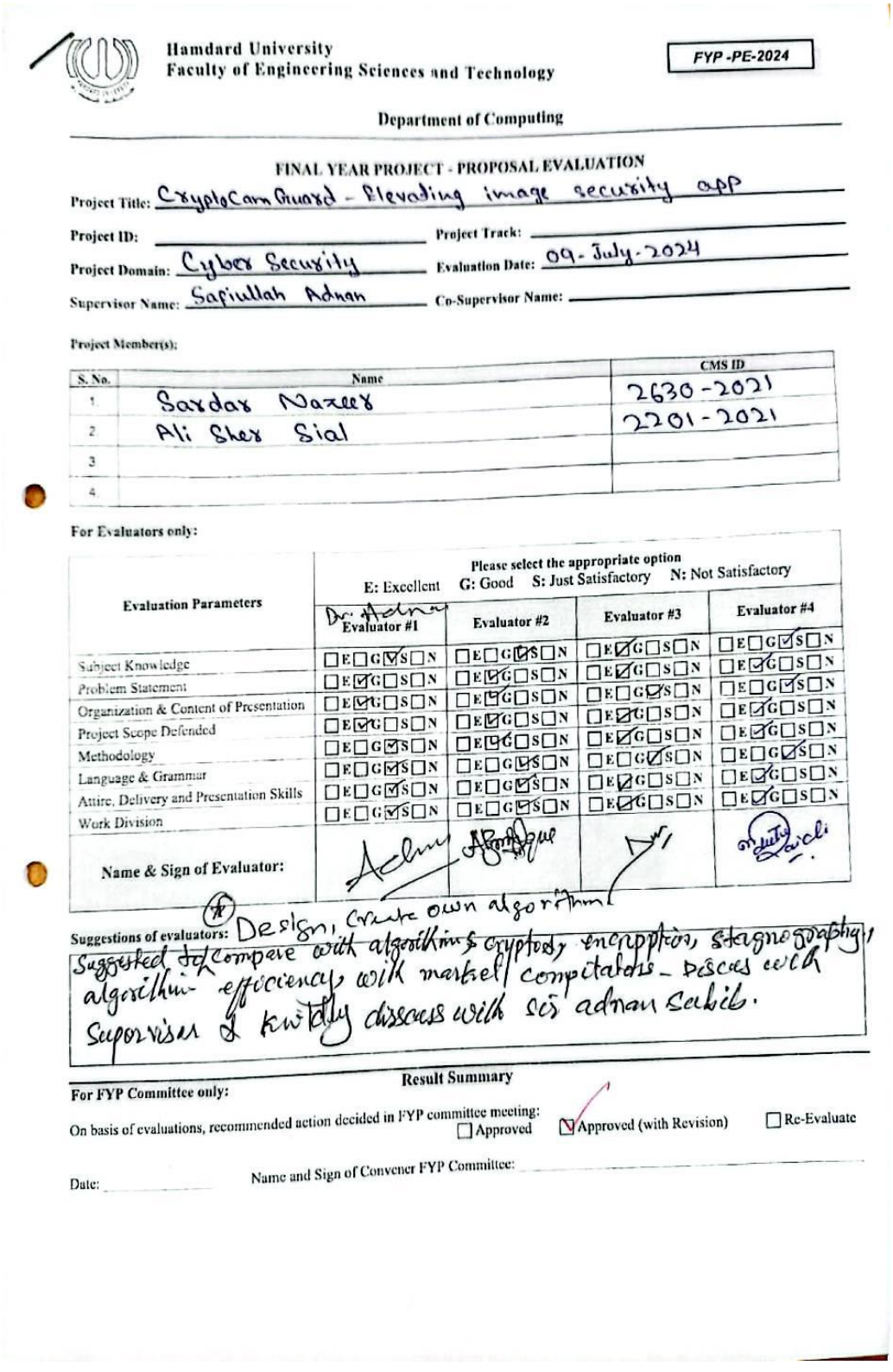
CS

INTRODUCTION

IN TODAY'S DIGITAL AGE USERS OFTEN GRANT THIRD- PARTY APPS ACCSESS TO THEIR DEVICE'S GALLERY, RAISING PRIVACY CONERNS.
THIS PROJECT INTRODUCE A MOBILE APP THAT ALLOW USERS TO SECURELY CAPTURE AND UPLOAD IMAGES USING ENCRYPTION TO ENSURE THEIR PRIVACY AUTO-GENERATED ENCRYPT AND DECRYPT THEIR IMAGES, GIVING THEM COMPLETE CONTROL OVER THEIR PERSONAL PHOTOS WITHOUT THE RISK OF THIRD-PART ACCESS.

OBJECTIVE

A6. COPY OF EVALUATION COMMENTS COPY OF EVALUATION COMMENTS BY SUPERVISOR FOR PROJECT – I MID SEMESTER EVALUATION



Hamdard University
Faculty of Engineering Sciences and Technology

FYP -PE-2024

Department of Computing

FINAL YEAR PROJECT - PROPOSAL EVALUATION

Project Title: CryptoCamGuard - Elevating image security app

Project ID: _____ Project Track: _____

Project Domain: Cyber Security Evaluation Date: 09-July-2024

Supervisor Name: Safiullah Adnan Co-Supervisor Name: _____

Project Member(s):

S. No.	Name	CMS ID
1.	Sardar Nazeer	2630-2021
2.	Ali Sher Sial	2201-2021
3.		
4.		

For Evaluators only:

Evaluation Parameters	Please select the appropriate option (E: Excellent G: Good S: Just Satisfactory N: Not Satisfactory)			
	Dr. Adnan Evaluator #1	Evaluator #2	Evaluator #3	Evaluator #4
Subject Knowledge	☐E☐G☑S☐N	☐E☐G☑S☐N	☐E☑G☐S☐N	☐E☐G☑S☐N
Problem Statement	☐E☑G☐S☐N	☐E☑G☐S☐N	☐E☑G☐S☐N	☐E☑G☐S☐N
Organization & Content of Presentation	☐E☑G☐S☐N	☐E☑G☐S☐N	☐E☐G☑S☐N	☐E☐G☑S☐N
Project Scope Defended	☐E☑G☐S☐N	☐E☑G☐S☐N	☐E☑G☐S☐N	☐E☑G☐S☐N
Methodology	☐E☐G☑S☐N	☐E☐G☐S☐N	☐E☑G☐S☐N	☐E☑G☐S☐N
Language & Grammar	☐E☐G☑S☐N	☐E☐G☑S☐N	☐E☐G☑S☐N	☐E☐G☑S☐N
Attire, Delivery and Presentation Skills	☐E☐G☑S☐N	☐E☐G☑S☐N	☐E☑G☐S☐N	☐E☑G☐S☐N
Work Division	☐E☐G☑S☐N	☐E☐G☑S☐N	☐E☑G☐S☐N	☐E☑G☐S☐N
Name & Sign of Evaluator:	Adnan	Ahmad		

Suggestions of evaluators: (*) Design, create own algorithm
Suggested to compare with algorithms cryptosy, encryption, steganography, algorithm efficiency, with market/ competators - discuss with Supervisor & kindly discuss with sir adnan sahib.

For FYP Committee only: Result Summary

On basis of evaluations, recommended action decided in FYP committee meeting:
☐ Approved ☑ Approved (with Revision) ☐ Re-Evaluate

Date: _____ Name and Sign of Convener FYP Committee: _____

A7. COPY OF EVALUATION COMMENTS BY SUPERVISOR FOR PROJECT – II MID SEMESTER EVALUATION

Dr. Khurram Iqbal	Satisfactory.	Approved (with revisions)
Engr. Maqsood Khatoon	Although most of theoretical work was complete to further enhance the quality and impact of their work, I believe the students would benefit greatly from additional guidance on the implementation phase. With your valuable mentorship, I am confident they will be able to translate their theoretical knowledge into a practical solution effectively.	
Aqsa Nadir	The student's domain knowledge has significantly improved compared to the previous evaluation. However, a deeper understanding and practical implementation of encryption algorithms are essential. Additionally, it is recommended to develop a mobile application alongside the web application for enhanced usability and accessibility. Since the project report has not been updated after the previous evaluation, it is required to update the report.	

A8. MEETINGS' MINUTES & Sign-Off Sheet

FYP Project Meeting

Minutes of Meeting

Meeting Date: 20/02/2025
Meeting Location: Room-230
Meeting Time: 2:00 – 2:30

Project Title: CryptoCamGuard - Elevating Image Security App
Project Code: FYP-008/FL24

1- List of Participants

Name	Project Role
Sardar Nazeer	Frontend Developer
Ali Sher Sial	Backend Developer
Mr. Saifullah Adnan	Supervisor

2- Meeting Agenda

UI Implementation and Algorithm

3- Agenda Points discussed in meeting

Discussed the UI design layout and selected algorithms for implementation.

4- Next Meeting for this project

Next Meeting Date: 06/03/2025

FYP Project Meeting

Minutes of Meeting

Meeting Date: 06/03/2025
Meeting Location: Room-230
Meeting Time: 2:00 – 2:30

Project Title: CryptoCamGuard - Elevating Image Security App
Project Code: FYP-008/FL24

1- List of Participants

Name	Project Role
Sardar Nazeer	Frontend Developer
Ali Sher Sial	Backend Developer
Mr. Saifullah Adnan	Supervisor

2- Meeting Agenda

Discussion on the Backend Design

3- Agenda Points discussed in meeting

Reviewed backend architecture and finalized database schema.

4- Next Meeting for this project

Next Meeting Date: 10/03/2025

FYP Project Meeting

Minutes of Meeting

Meeting Date: 10/03/2025
Meeting Location: Room-230
Meeting Time: 2:00 – 2:30

Project Title: CryptoCamGuard - Elevating Image Security App
Project Code: FYP-008/FL24

1- List of Participants

Name	Project Role
Sardar Nazeer	Frontend Developer
Ali Sher Sial	Backend Developer
Mr. Saifullah Adnan	Supervisor

2- Meeting Agenda

Application layout implementation

3- Agenda Points discussed in meeting

Implemented initial application layout and integrated front-end with back-end.

4- Next Meeting for this project

Next Meeting Date: 17/04/2025

FYP Project Meeting

Minutes of Meeting

Meeting Date: 17/04/2025
Meeting Location: Room-230
Meeting Time: 2:00 – 2:30

Project Title: CryptoCamGuard - Elevating Image Security App
Project Code: FYP-008/FL24

1- List of Participants

Name	Project Role
Sardar Nazeer	Frontend Developer
Ali Sher Sial	Backend Developer
Mr. Saifullah Adnan	Supervisor

2- Meeting Agenda

To implement database

3- Agenda Points discussed in meeting

Database tables created and connected to the application.

4- Next Meeting for this project

Next Meeting Date: 17/05/2025

FYP Project Meeting

Minutes of Meeting

Meeting Date: 17/05/2025
Meeting Location: Room-230
Meeting Time: 2:00 – 2:30

Project Title: CryptoCamGuard - Elevating Image Security App
Project Code: FYP-008/FL24

1- List of Participants

Name	Project Role
Sardar Nazeer	Frontend Developer
Ali Sher Sial	Backend Developer
Mr. Saifullah Adnan	Supervisor

2- Meeting Agenda

Review of database implementation

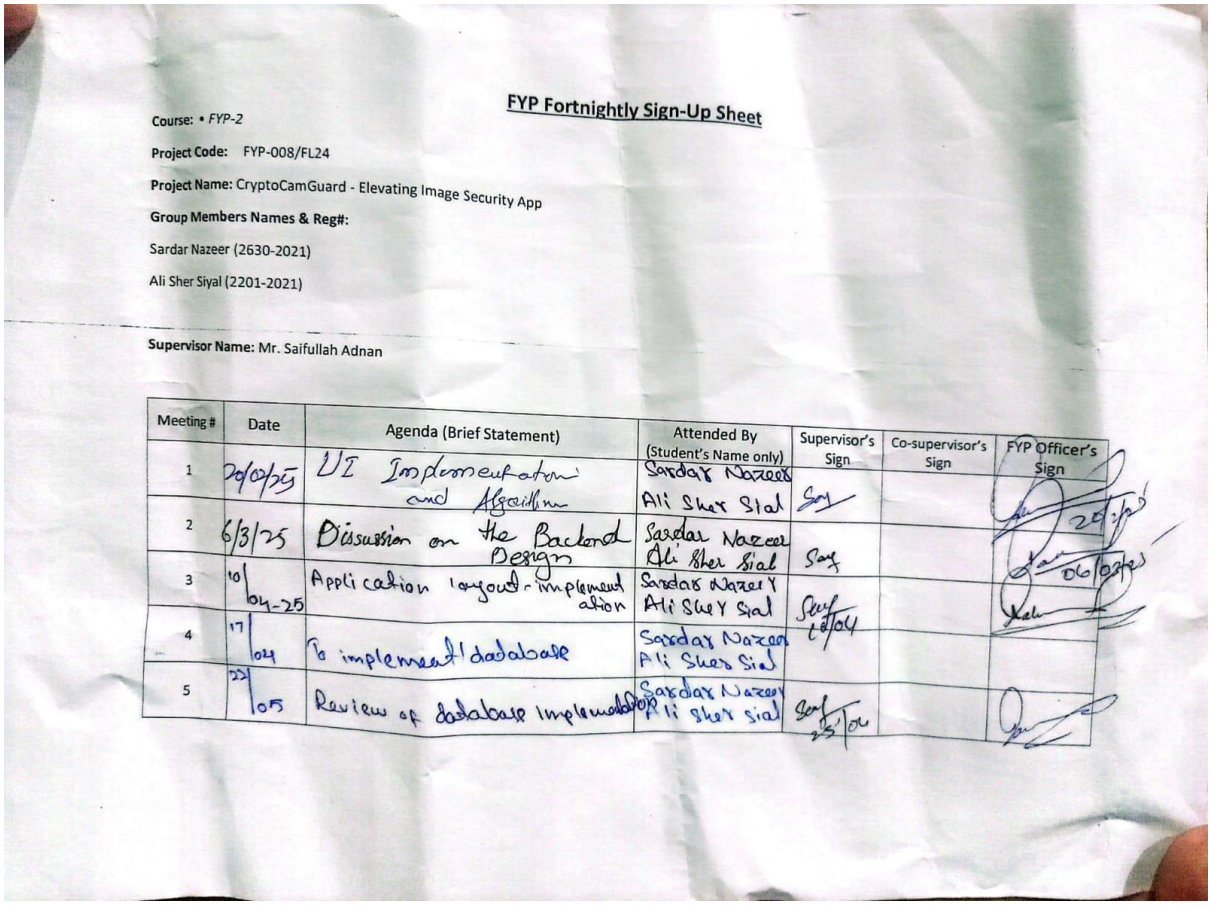
3- Agenda Points discussed in meeting

Reviewed the database implementation and made optimizations where necessary.

4- Next Meeting for this project

Next Meeting Date: 22/05/2025

A9. FYP fortnightly sign-up sheet



FYP Fortnightly Sign-Up Sheet

Course: • *FYP-2*

Project Code: FYP-008/FL24

Project Name: CryptoCamGuard - Elevating Image Security App

Group Members Names & Reg#:

Sardar Nazeer (2630-2021)

Ali Sher Siyal (2201-2021)

Supervisor Name: Mr. Saifullah Adnan

Meeting #	Date	Agenda (Brief Statement)	Attended By (Student's Name only)	Supervisor's Sign	Co-supervisor's Sign	FYP Officer's Sign
1	20/02/25	UI Implementation and Algorithm	Sardar Nazeer, Ali Sher Sial			
2	6/3/25	Discussion on the Backend Design	Sardar Nazeer, Ali Sher Sial			
3	10/04-25	Application layout-implementation	Sardar Nazeer, Ali Sher Sial			
4	17/04	To implement database	Sardar Nazeer, Ali Sher Sial			
5	22/05	Review of database implementation	Sardar Nazeer, Ali Sher Sial			