Chapter 1 Introduction

1.1 Introduction to Project

In today's digital age, small businesses are increasingly vulnerable to cyber threats that can compromise sensitive data, damage their reputation, and result in financial losses. Many small business owners lack the technical expertise and resources to implement robust cybersecurity measures, leaving their websites exposed to risks such as data breaches, hacking, and fraud. As cyberattacks become more sophisticated, the need for accessible, affordable, and effective security solutions has never been more critical.

Cyberwise aims to address this gap by offering a comprehensive platform that combines essential cybersecurity features with user-friendly educational resources. This platform is designed to help small business owners secure their websites, protect sensitive data, and reduce the risk of cyberattacks, all while providing valuable cybersecurity education. Cyberwise empowers businesses to take control of their online security and build trust with their customers by offering simple, scalable security tools and educational materials.

The project focuses on three core objectives: developing and implementing fundamental security features for websites, creating a modular and scalable cybersecurity framework, and offering an accessible web-based scanner to identify vulnerabilities. Ultimately, Cyberwise strives to make cybersecurity both approachable and actionable for small business owners, ensuring they can operate confidently in an increasingly interconnected digital world.

1.2 Features of this project are as follows:

Our project incorporates four core functionalities designed to enhance cybersecurity and streamline threat management processes:

1.2.1 Log Page for Daily Updates:

A dedicated log page is implemented to record and present daily updates in the realm of cybersecurity. This feature ensures users have real-time access to recent developments, aiding in informed decision-making.

1.2.2 IP Scanner for Malicious IPs:

This module performs automated scans of IP addresses to detect and flag potentially malicious activities. By identifying threats early, it enhances the system's defense against unauthorized access and attacks. An IP scanner is a tool that systematically scans a range of IP addresses within a network to detect active devices, services, or potential security risks. It checks which IP addresses are in use and identifies any malicious or unauthorized devices that could pose a threat to the network. In the context of your project, you aim to build an IP scanner that detects malicious IP addresses to help protect against cyberattacks.

A. **Functionality**:

The IP scanner you've built focuses on detecting and scanning IP addresses that could be associated with malicious activity. It operates by

- **Scanning a specified range of IP addresses**: The user or system administrator provides a range of IP addresses, and the scanner checks all of them for activity.
- **Detecting Malicious IPs**: It identifies if an IP address is flagged as malicious, such as from known blacklists (e.g., IPs involved in spamming, hacking, or other attacks).

- **Reporting Vulnerabilities**: It generates reports on which IP addresses are safe and which might be flagged as threats, offering insights into suspicious or dangerous activity on the network.

B. **Key Features of Your IP Scanner**:
   - **Malicious IP Detection**: Identifies known malicious IPs by checking against threat intelligence feeds and blacklists.
   - **Port Scanning**: Optionally checks for open ports on the target IPs to identify potentially vulnerable services.
   - **User-Friendly Interface**: A web-based interface where users can input IP ranges, start scans, and view results.
   - **Real-time Alerts**: Sends alerts or notifications when a malicious IP is detected.
   - **Logging & History**: Keeps a log of past scans and results for auditing and future reference.

C. **Benefits**:
   - **Cybersecurity**: The primary purpose is to help secure networks by identifying malicious IP addresses before they can cause damage.
   - **Efficiency**: Automates the scanning process, allowing for periodic scans without requiring constant manual oversight.
   - **Ease of Use**: Your web application enables users to easily initiate scans and understand the results, making it accessible for small businesses or individuals who are not necessarily cybersecurity experts.
   - **Early Detection**: Identifies potential threats at the IP level before they can move laterally within the network, giving network administrators a chance to block or mitigate attacks early.

D. **Technical Aspects of Building the IP Scanner**:

- **Backend Development**: The backend of the application might use scripting languages (like Python or Node.js) to perform the scanning and make network requests.
- **Integration with Threat Intelligence Services**: You would integrate with services that maintain blacklists or threat intelligence feeds (e.g., abuse.ch, Spamhaus, etc.) to flag IPs as malicious.
- **Real-time Scanning**: For better security, the IP scanner can run in real time, continuously checking for new threats and notifying the user of potential issues.
- **Visualization**: The results are displayed on a simple dashboard, showing which IPs are potentially harmful and which are safe to continue interacting with.

E. **Example of Use**:

- A **small business** could use the IP scanner to scan their network's range of IP addresses for any malicious activity, such as attempts to exploit vulnerabilities or conduct DDoS attacks.
- **Network administrators** in larger enterprises can use the scanner to monitor devices connected to the network, ensuring only trusted entities have access.

1.2.3 Vulnerability Scanner:

The vulnerability scanner developed for your project is a tool specifically designed to scan URLs and identify security vulnerabilities within web applications. This scanner is an essential part of any cybersecurity strategy, as it helps detect weaknesses that could potentially be exploited by attackers. In your case, the scanner focuses on scanning URLs and reporting back with potential vulnerabilities, such as common web application flaws.

Working of Vulnerability Scanner as follows

A. **URL Input**: The user provides a URL (typically a web application or website) that they want to scan for vulnerabilities.

B. **Scanning Process**:

   - The scanner interacts with the provided URL by sending HTTP requests (GET, POST, etc.) to gather information about the web application.
   - It attempts to detect common vulnerabilities by analyzing responses and behaviors. The vulnerabilities could be associated with issues like weak encryption, outdated software, insecure headers, input validation problems, or misconfigurations.

C. **Vulnerability Detection**:

   The scanner checks for well-known vulnerabilities, such as:

   - **SQL Injection**: This occurs when an application improperly processes user input in a query to the database, enabling attackers to run arbitrary SQL code.
   - **Cross-Site Scripting (XSS)**: XSS vulnerabilities allow attackers to inject malicious scripts into web pages viewed by other users.
   - **Cross-Site Request Forgery (CSRF)**: This type of attack tricks the user into performing unwanted actions on a web application in which they are authenticated.
   - **File Upload Vulnerabilities**: Weaknesses that allow attackers to upload harmful files (e.g., executable files).

- **Security Misconfigurations**: These occur when the web application, server, or database is not securely configured, exposing sensitive data.
- **Outdated Software**: Using unpatched software versions with known vulnerabilities.

D. **Reporting Results**:
   - After the scan completes, the vulnerability scanner generates a report that lists all the identified vulnerabilities.
   - The report is categorized based on the severity of the vulnerabilities (e.g., critical, high, medium, low).
   - Each detected issue includes a description of the vulnerability, its potential impact, and possible solutions or remediation steps.
   -

E. **User Interface**:
   - The user interacts with the scanner through a simple web interface where they can enter the target URL for scanning.
   - After submitting the URL, the scanner starts the process and provides feedback once the scan is complete, typically in the form of a downloadable report or a summary shown directly in the UI.

F. **URL Scanning**: Focused solely on scanning the security of URLs, identifying security risks in web applications.
   - **Comprehensive Vulnerability Detection**: Scans for a wide range of vulnerabilities, including SQL injection, XSS, CSRF, and more.
   - **Automated Reporting**: Automatically generates detailed vulnerability reports for users to review and act upon.

- **User-Friendly Interface**: Simple and intuitive UI for users to submit URLs and view results.
- **Customizable**: Potential for future expansions, such as adding more specific types of tests or integrating it with other cybersecurity tools.

G. Use Cases:

- **Web developers** can use this tool to ensure that their web applications are secure before going live.
- **Security Teams**: Helps penetration testers and cybersecurity professionals quickly identify vulnerabilities in websites or web apps.
- **Small Businesses**: Especially useful for small businesses with limited cybersecurity expertise, as they can use the scanner to ensure their web applications are protected against common vulnerabilities.

H. Example of a Use Case:

- A small e-commerce website provides a URL for the scanner.
- The scanner checks the website for common vulnerabilities like SQL injection, weak password policies, and outdated plugins.
- The scanner detects a high-risk SQL injection vulnerability and provides recommendations on how to patch it.
- The business receives the report and takes the necessary steps to mitigate the vulnerabilities before the website goes live.

1.2.4 Encrypter and Decrypter:

A robust encryption and decryption module secures sensitive information, enabling secure data transmission and storage. It supports various cryptographic standards, ensuring adaptability to diverse security requirements.

Each of these features contributes to a comprehensive cybersecurity toolkit, addressing critical aspects of system integrity, threat detection, and data protection.

1.2 Project Category

This project comes under the category of web application as follows

A web application is a software program that runs on a web server and can be accessed through a web browser. Unlike traditional desktop applications, web applications require no installation on the user's device, making them highly accessible and platform-independent. They are often used for tasks ranging from simple information sharing to complex operations like managing cybersecurity systems, which is the focus of our project. Our project leverages the web application framework to deliver a user-friendly and accessible interface for cybersecurity management.

A. **Key aspects include:**
    - **Browser Accessibility:**

      Users can access the application from any internet-enabled device, ensuring flexibility in monitoring and managing cybersecurity tasks.
    - **Centralized Operation:**

      The application hosts all features—daily update logs, IP scanning, vulnerability scanning, and encryption/decryption—on a centralized web platform. This approach enhances collaboration and ensures data consistency.
    - **Real-Time Functionality:**

      Through seamless integration of tools, the web application allows real-time updates, scans, and security measures, keeping users ahead of potential threats.

1.3 Problem Formulation

Cybersecurity has grown to be a major issue in the connected world of today. Malicious actors are posing a growing threat to organizations and individuals by taking advantage of weaknesses in networks, data, and systems. Our project's goal is to provide a comprehensive online application that simplifies crucial cybersecurity chores in order to overcome these obstacles. These are the main issues that our initiative attempts to solve:

A. **Lack of Centralized Tools:**

   Many cybersecurity tasks, such as IP scanning, vulnerability detection, and encryption, often require multiple standalone tools. This fragmentation leads to inefficiencies and increased operational complexity.

B. **Real-Time Threat Detection:**

   Existing solutions may not provide timely updates or alerts about emerging cybersecurity threats, leaving systems vulnerable to attacks.

C. **Identifying Malicious IPs:**

   Detecting and flagging malicious IP addresses is essential for preventing unauthorized access and breaches. However, manual or disjointed tools are often slow and error-prone.

D. **Unaddressed Vulnerabilities:**

   Security gaps in systems and applications can remain undetected without proper scanning, increasing the risk of exploitation.

E. **Data Security Challenges:**

   Ensuring secure communication and storage of sensitive information is vital. Without robust encryption and decryption mechanisms, data integrity and confidentiality are compromised.

By formulating these problems, this project highlights the need for an integrated, web-based solution that centralizes cybersecurity tools, provides real-time updates, and ensures robust data protection.

1.4 Identification/Recognition of Need

The need for our project arises from the growing demand for efficient and accessible cybersecurity solutions. Below are the key reasons justifying the necessity of our project:

A. **Increasing Cyber Threats:**

With the rise of sophisticated cyberattacks, there is an urgent need for tools that can proactively detect and mitigate potential threats to safeguard systems and sensitive information.

B. **Centralized Cybersecurity Management:**

Organizations often struggle with fragmented tools for logging updates, scanning IPs, detecting vulnerabilities, and managing encryption. A unified platform simplifies operations and enhances productivity.

C. **Accessibility Through a Web Application:**

Cybersecurity tools need to be available across multiple devices and locations. A web application ensures accessibility without the need for complex installations or maintenance.

D. **Efficiency in Threat Detection and Response:**

Manual threat detection is time-consuming and error-prone. An automated system with real-time updates and scanning capabilities is essential for effective cybersecurity management.

E. **Data Security and Privacy Compliance:**

With stricter regulations around data protection (e.g., GDPR, HIPAA), there is a

pressing need for reliable encryption and decryption tools to ensure compliance and protect sensitive data.

These factors underline the importance of our project as a comprehensive, web-based cybersecurity solution addressing critical needs in the current digital landscape.

1.5 Objectives

The objectives of Cyber-Wise (a comprehensive vulnerability scanner) are

1. To develop and implement core security modules for foundational functionality.
2. To integrate and thoroughly test a modular and scalable cybersecurity framework.
3. To develop an accessible, user-friendly web cyber scanner to help small owners safeguard sensitive data and build customer trust.

1.6 Proposed System

The proposed system is a web-based cybersecurity platform designed to help small businesses protect themselves from cyberattacks. Many small businesses lack the resources or expertise to implement advanced cybersecurity measures, making them vulnerable to threats. This system provides an accessible, user-friendly, and cost-effective solution tailored to their needs.

1.7 Unique features of the proposed system

A. **Centralized Daily Cybersecurity Updates:**

Unlike other platforms, our system includes a dedicated log page that consolidates the latest cybersecurity updates, making it easy for small businesses to stay informed about emerging threats.

B. **Integrated Malicious IP Scanner:**

The platform features a built-in IP scanning tool to identify and flag harmful IP addresses, providing an automated, real-time solution for preventing unauthorized access.

C. **Customizable Vulnerability Scanner:**

Our vulnerability scanner is designed to be user-friendly, allowing small businesses to easily identify and address potential security gaps without requiring advanced technical knowledge.

D. **Simplified Encryption and Decryption Tools:**

The system offers secure, straightforward tools for encrypting and decrypting data, ensuring that sensitive information is protected in compliance with data security standards.

E. **Web-Based Accessibility for Small Businesses:**

As a web application, the platform is accessible from any device with an internet connection, eliminating the need for specialized hardware or software installations. This design ensures that even businesses with limited technical resources can benefit from enhanced cybersecurity.

This system stands out by combining critical cybersecurity functionalities into a single, easy-to-use web platform, addressing the unique challenges faced by small businesses in safeguarding their digital assets.

Chapter 2. Requirement Analysis and System Specification

2.1 Feasibility study

A feasibility study for our web-based cybersecurity platform, designed to help small businesses protect themselves from cyberattacks, involves assessing the proposal from technical, economic, and operational perspectives. This comprehensive evaluation ensures that the system is viable, cost-effective, and operationally sustainable. Let's delve into each aspect:

2.1.1 Technical Feasibility:

The proposed system is based on modern web technologies, leveraging widely used frameworks such as Node.js for the back-end and Tailwind CSS for front-end styling. It integrates readily available libraries and APIs for key functionalities, including IP scanning, vulnerability detection, and encryption/decryption. Cloud hosting platforms provide scalability, ensuring the system can handle growing user demand without significant infrastructure investments. The technical foundation ensures the system is achievable using existing expertise and tools.

2.1.2 Economic Feasibility:

Developing the platform is cost-effective due to the utilization of open-source technologies and cloud services with flexible pricing models. Small businesses benefit from an affordable cybersecurity solution compared to costly enterprise-level alternatives or hiring dedicated security personnel. The system's affordability ensures it meets the financial constraints of its target audience while delivering significant value.

2.1.3 Operational Feasibility:

The platform is designed with simplicity and user-friendliness as core principles. Small business owners, even without technical expertise, can easily navigate and utilize the system for logging updates, scanning IPs, and securing data. Automated tools reduce

manual effort, enabling businesses to seamlessly integrate the platform into their daily operations without requiring extensive training or resources.

2.2 Software Requirement Specification Document of "Cyberwise"

Below is an outline for a Software Requirements Specification (SRS) document for Cyberwise. This document covers various aspects of requirements, including data, functionality, performance, dependability, maintainability, security, and look and feel.

2.2.1 Product Perspective

This platform will operate as a standalone web application accessible from any internet-enabled device. It will combine multiple tools into a unified interface to streamline cybersecurity management for small businesses.

2.2.2 Product Functions

- Daily updates on emerging cybersecurity threats.
- Scanning and identification of malicious IPs.
- Vulnerability analysis to detect and report security gaps.
- Encryption and decryption of sensitive data.

2.2.3 User Characteristics

The primary users are small business owners and non-technical staff. The system is designed to be intuitive and require minimal technical expertise.

2.2.4 Functional Requirements

- Ability to perform real-time IP and vulnerability scans.
- Automatic updates to the log page with cybersecurity news.

- Data encryption/decryption tools for file uploads.

2.2.5 Performance Requirements

- System response time should not exceed 2 seconds for basic operations.
- The platform must support at least 100 concurrent users.

2.2.6 Dependability Requirements

- 99.9% uptime reliability for web services.
- Automatic backups of logs and reports every 24 hours.

2.2.7 Maintainability Requirements

- Modular design for easy updates and feature additions.
- Clear documentation for future maintenance and troubleshooting.

2.2.8 Security Requirements

- End-to-end encryption for all data transmissions.
- Secure user authentication with role-based access control.
- Compliance with GDPR and other relevant data protection regulations.

2.2.9 Look and Feel Requirements

- Clean, minimalist user interface with an intuitive layout.
- Consistent use of fonts, colors, and icons across the platform.
- Mobile-responsive design for seamless use on smartphones and tablets.

2.3 SD model to be used

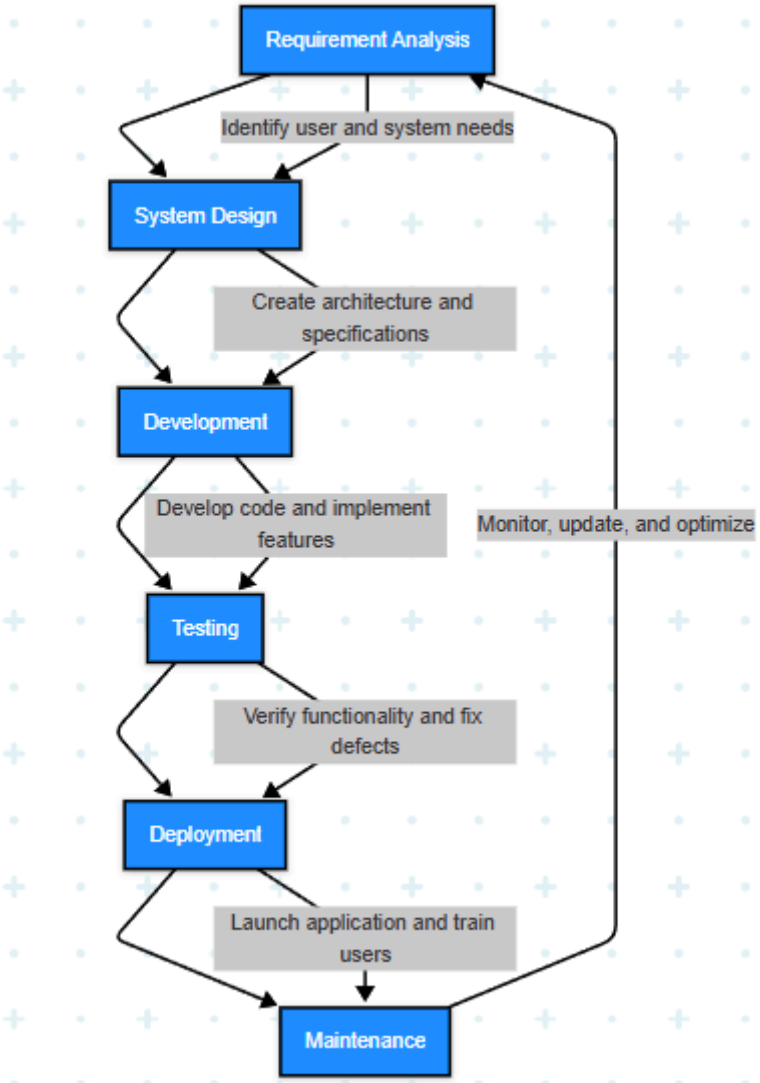

Figure 2.1 SDLC of this project

Chapter 3. System Design

3.1 Design Approach

We take a function-oriented approach to design. This indicates that the main goal of our project is to plan activities according to the precise roles and duties required to meet the cybersecurity goals for small enterprises. A systematic and effective development process can be ensured by segmenting the project into smaller, more manageable tasks.

3.2 Detail Design System Design

The system design phase defines the overall architecture and structure of the web-based cybersecurity platform. It outlines how the various components of the system interact with each other, ensuring scalability, security, and optimal performance. The design focuses on creating a modular, efficient, and user-friendly system that addresses the cybersecurity needs of small businesses.

- **Flow charts or Block Diagrams**

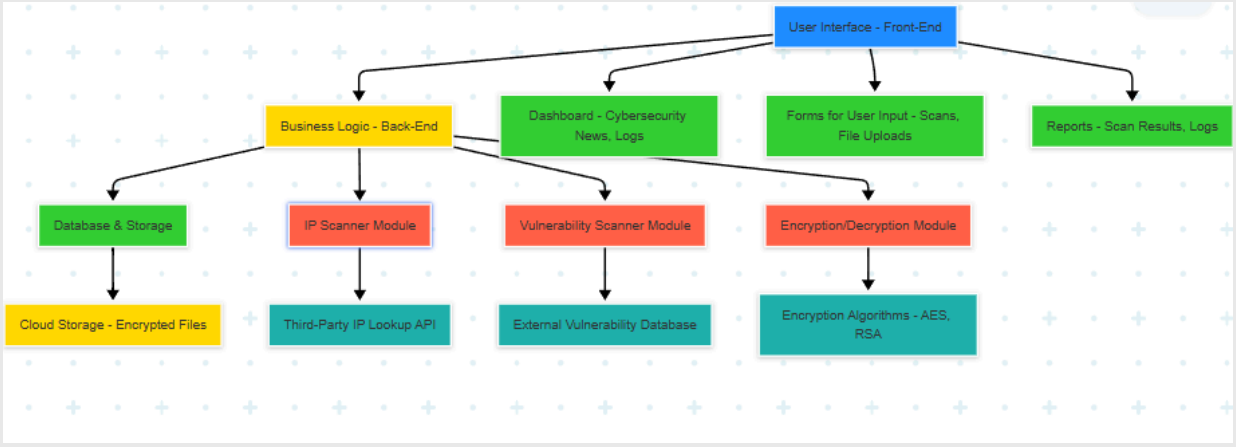

Figure 3.1 Main structure of this project

▪ **UML diagram**

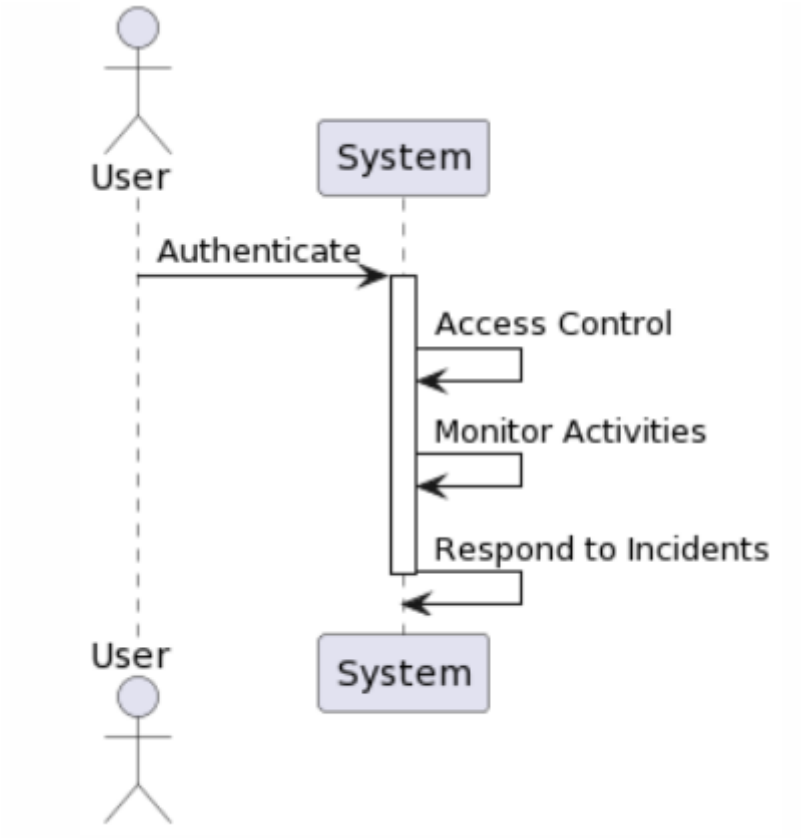

Figure 3.2 UML of this project

▪ **DFDs**

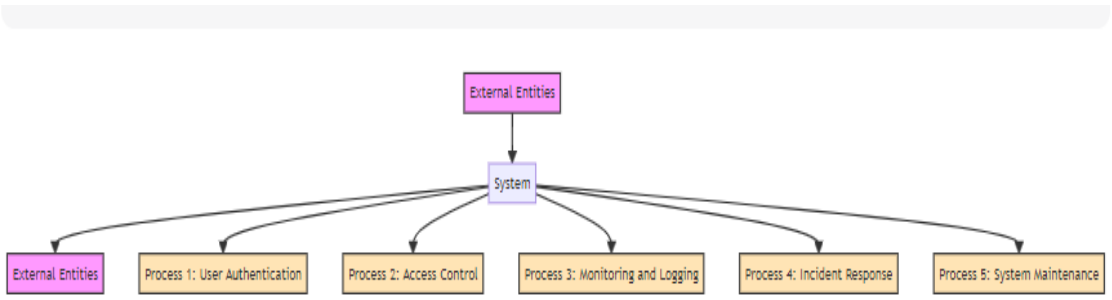

Figure 3.3 DFDs of this project

▪ **Sequence Diagram**

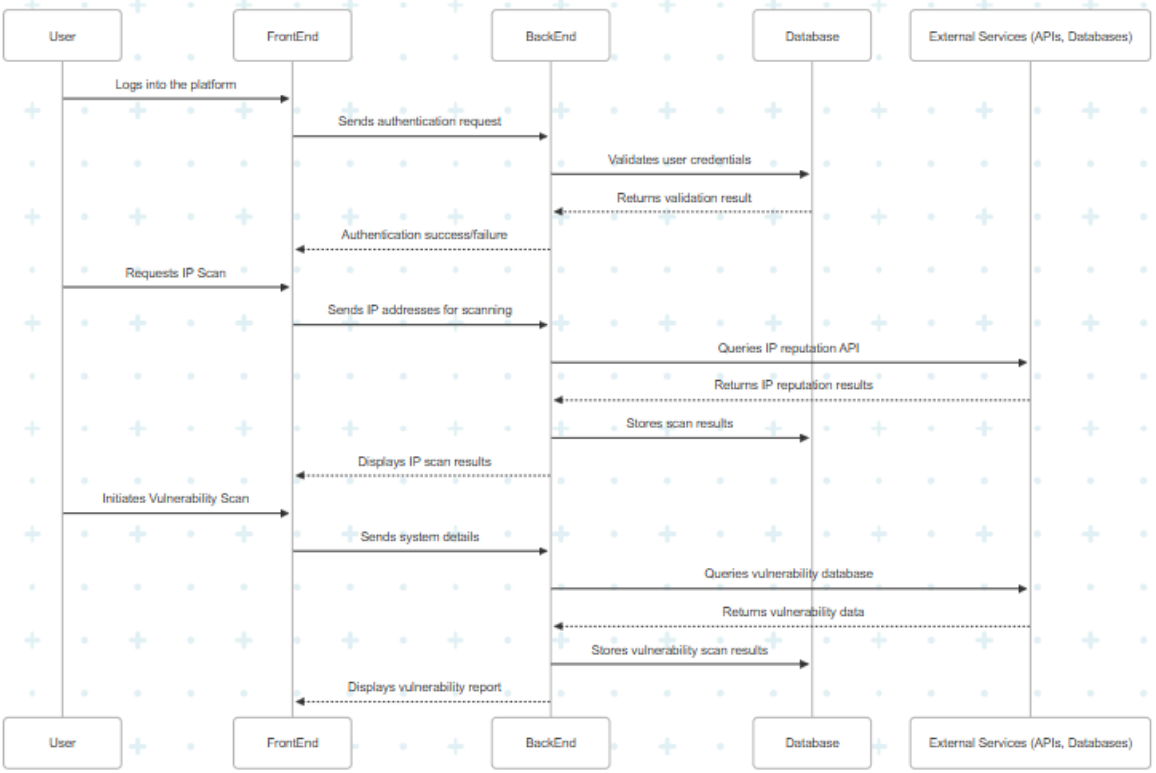

Figure 3.4 Proper Order of this project

▪ **Use case**

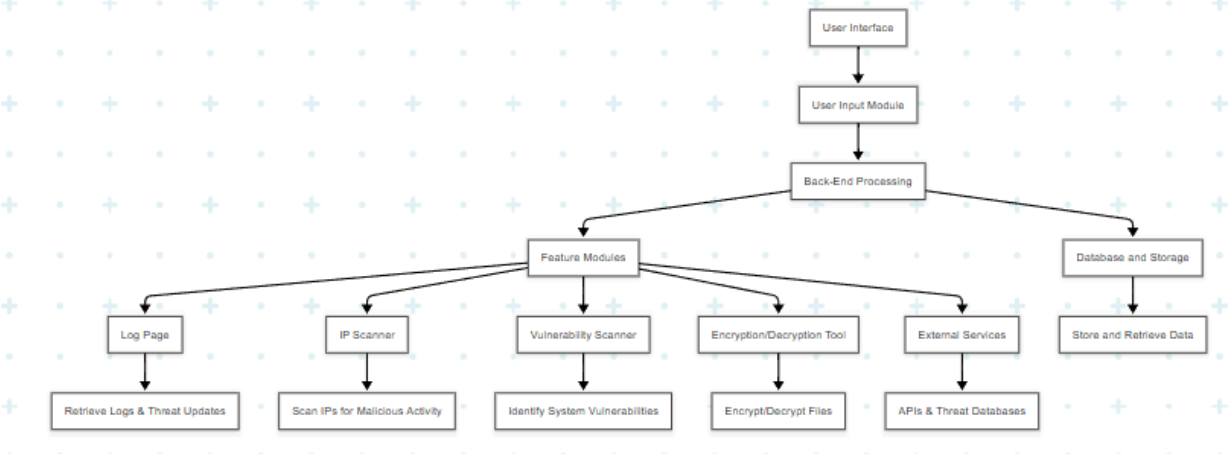

Figure 3.5 Basic working of this project

▪ **State Diagrams**

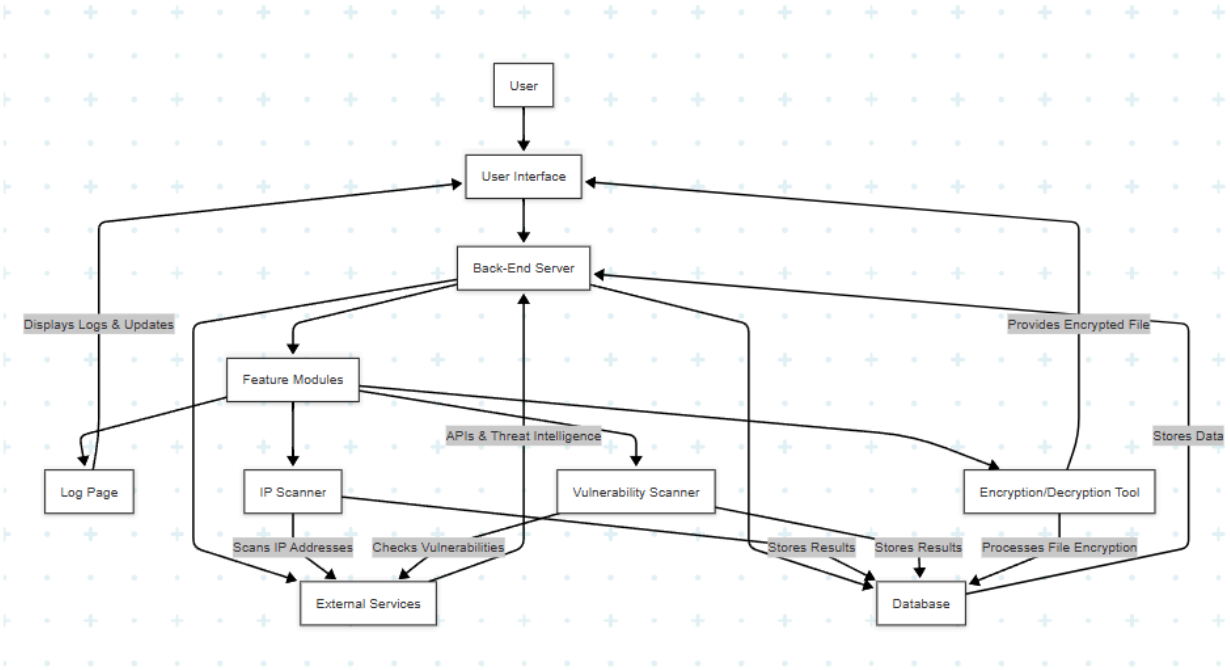

Figure 3.6 Basic interaction of this project

▪ **Deployment Diagrams**

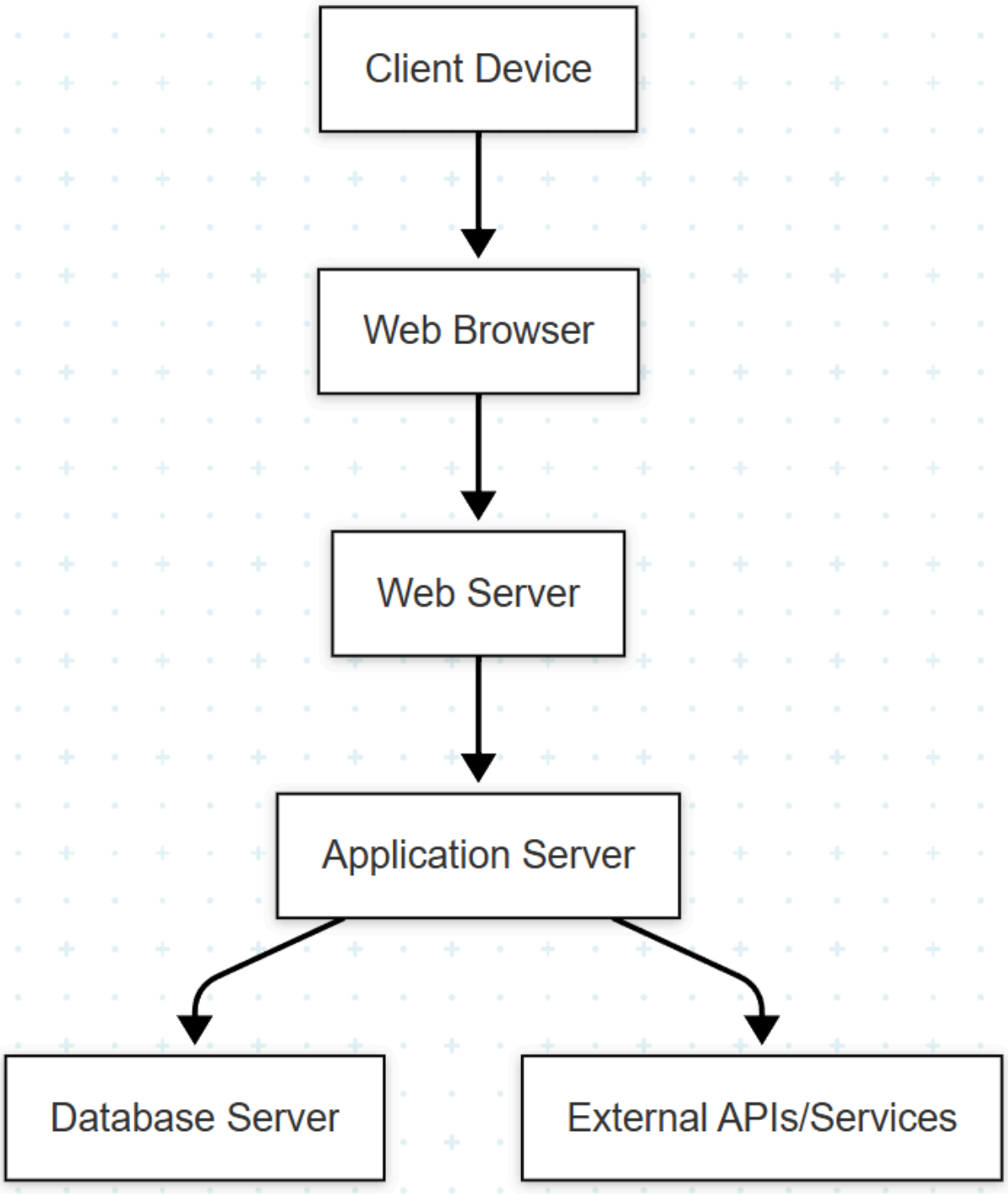

Figure 3.7 Data security of this project

▪ **Communication Diagram**

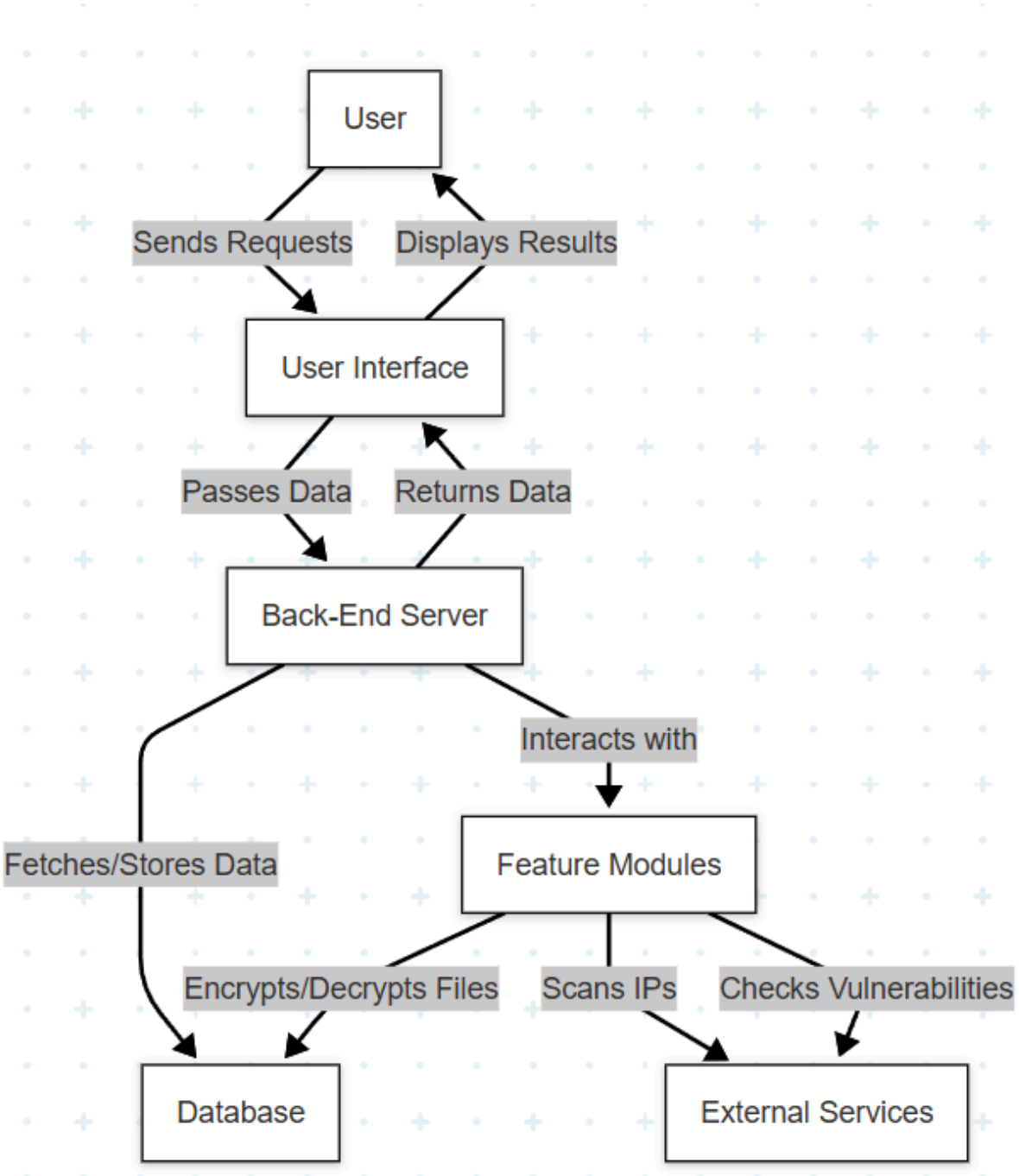

Figure 3.8 Backend interaction of this project

o E-R Diagram

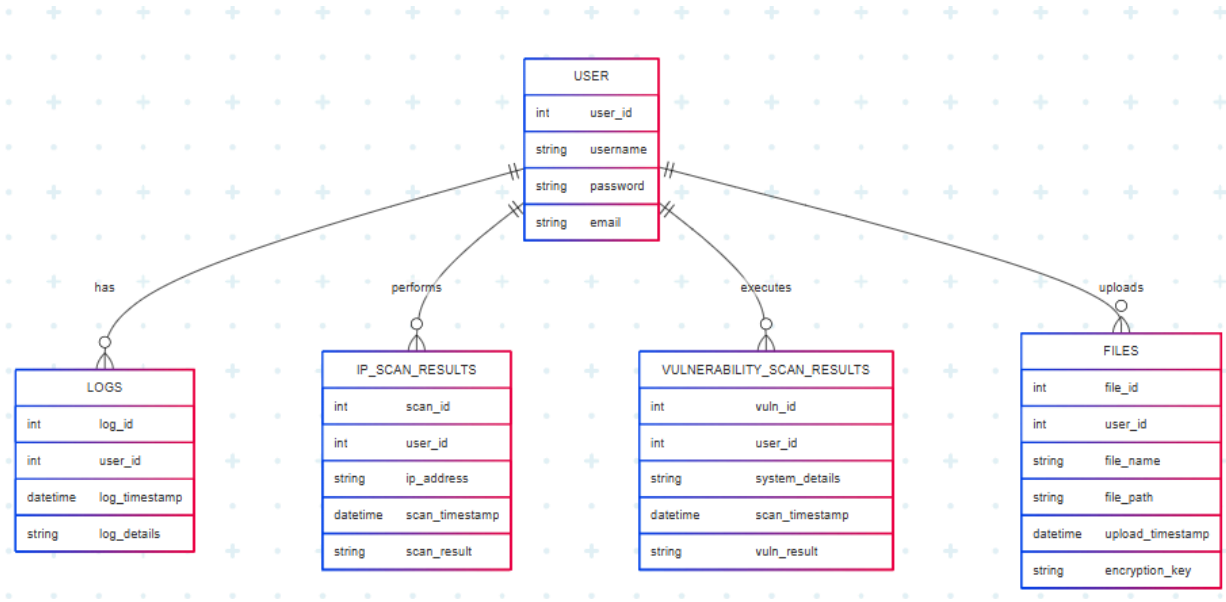

Figure 3.9: Workflow of this project

3.3 User Interface Design

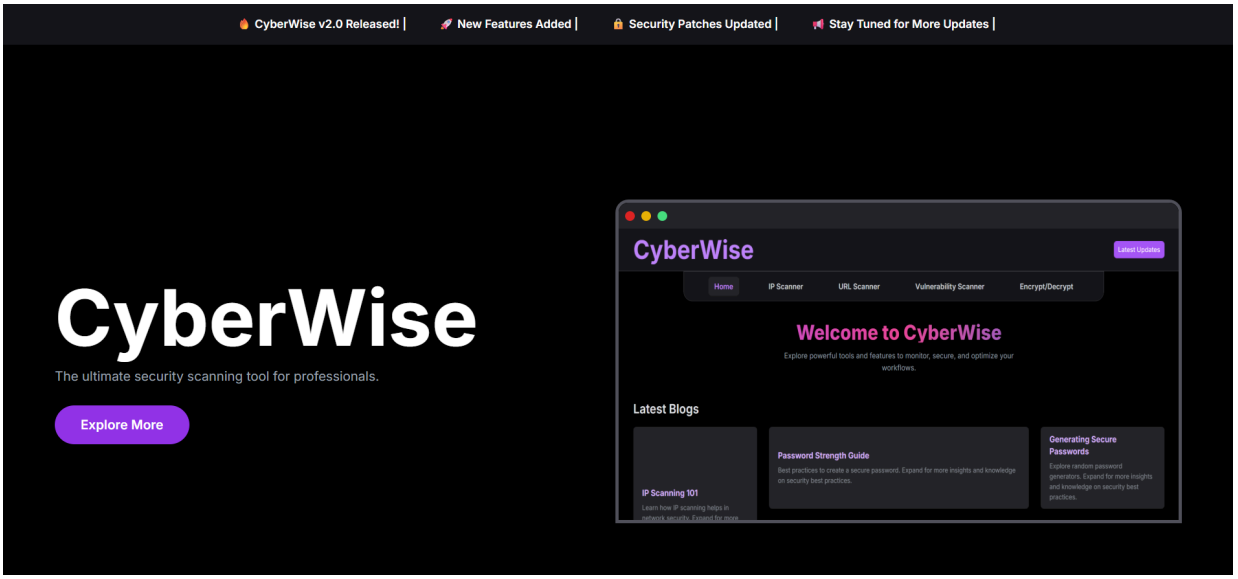

Figure 3.10 User view of this project

3.4 Methodology

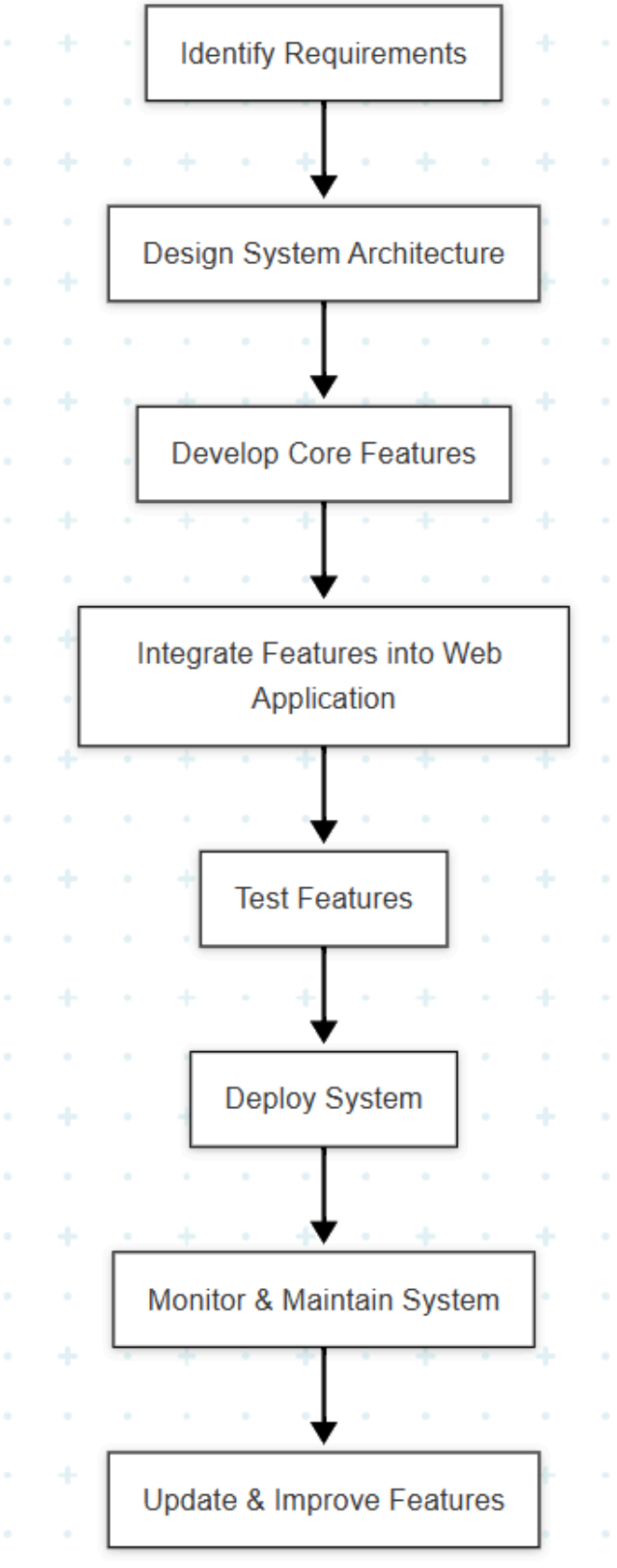

Figure 3.11 Block Diagram of this project

Chapter 4. Implementation and Testing

4.1 Introduction to Languages, IDEs, Tools, and Technologies used for Project work

The project was developed using a carefully selected combination of modern web development languages, tools, and technologies to ensure a responsive, efficient, and user-friendly application.

- **Technologies used in this project:** The main technologies used are as follows:
    - **Next.js**: A powerful React-based framework that allows building fast, scalable, and SEO-friendly web applications with server-side rendering and static site generation.

      Tailwind CSS: A utility-first CSS framework that provides a highly flexible and efficient approach to designing and styling user interfaces using pre-built classes.
    - **Framer Motion**: A leading animation library that enables smooth and visually appealing animations and transitions in React applications, enhancing the overall user experience.
    - **NextUI**: A modern and customizable component library designed for Next.js, offering pre-styled and reusable UI components that accelerate development and maintain a consistent design language.
    - **JavaScript (ES6+)**: The core programming language used to build interactive, dynamic, and functional web application components and implement project logic.

    - **Node.js**: A runtime environment used to handle server-side operations and manage backend processes efficiently.
    - **AES-GCM Encryption**: An advanced cryptographic method implemented to provide secure and reliable data encryption and decryption functionalities within the application.
    - **Visual Studio Code (VS Code)**: The primary Integrated Development Environment (IDE) used for writing, editing, debugging, and managing the project's codebase, offering a wide range of extensions and development tools.
    - **Git and GitHub**: Essential version control tools employed for managing code versions, collaborating on development tasks, and maintaining the project repository.
- **The languages used in this project are:** The languages used in this project as follows:
    - TypeScript
    - JavaScript (ESNext)
    - JSON
    - CSS (Tailwind CSS)
    - JSX / TSX

The introduction to these languages is as follows:

- **TypeScript**

TypeScript is an open-source programming language developed and maintained by Microsoft. It is a syntactic superset of JavaScript, meaning that any valid JavaScript code is also valid TypeScript code. TypeScript introduces optional static typing to JavaScript, allowing developers to specify types for variables, function parameters, and object properties. This

feature enables early detection of errors during development, enhancing code reliability and maintainability.

One of the primary motivations behind TypeScript is to facilitate the development of large-scale applications. By providing a robust type system, TypeScript helps in organizing and structuring code more effectively. It supports modern JavaScript features, including classes, interfaces, and modules, aligning with the ECMAScript 6 specification.

TypeScript's compatibility with existing JavaScript codebases allows for incremental adoption. Developers can gradually introduce TypeScript into their projects, benefiting from its features without the need for a complete rewrite. The TypeScript compiler transpiles TypeScript code into plain JavaScript, ensuring compatibility with all JavaScript environments.

In addition to its type system, TypeScript offers enhanced tooling support. Integrated Development Environments (IDEs) like Visual Studio Code leverage TypeScript's type information to provide features such as autocompletion, code navigation, and refactoring tools. These capabilities contribute to a more efficient and productive development experience.

Overall, TypeScript serves as a powerful tool for developers seeking to build scalable, maintainable, and robust applications. Its integration of static typing with modern JavaScript features makes it a valuable asset in contemporary web development.

- **JavaScript (ES Next)**

JavaScript is a versatile, high-level programming language that serves as the backbone of modern web development. Over the years, it has undergone significant evolution through the ECMAScript (ES) specifications, introducing new features and enhancements to meet the growing demands of developers. The term "ESNext" refers to the latest ECMAScript features that are either currently available or in the process of being standardized.

The evolution from ES6 (ECMAScript 2015) to ESNext has brought about numerous improvements in syntax, functionality, and performance. These enhancements aim to make JavaScript more expressive, concise, and efficient. Some notable features introduced in recent ECMAScript versions include:

- **Optional Chaining (?.)**: Allows developers to safely access deeply nested object properties without having to check each level for null or undefined.

- **Nullish Coalescing Operator (??)**: Provides a default value when dealing with null or undefined, offering a more precise alternative to the logical OR (||) operator.

- **Logical Assignment Operators (&&=, ||=, ??=)**: Combine logical operations with assignment, simplifying common patterns in code.

- **BigInt**: Introduces a new primitive type for representing integers with arbitrary precision, enabling accurate representation of large numbers beyond the safe integer limit.

- **Dynamic Import (import())**: Allows modules to be loaded dynamically at runtime, facilitating code splitting and on-demand loading.

- **Promise.any()**: Returns the first fulfilled promise from a list of promises, ignoring rejected ones, which is useful in scenarios where only one successful result is needed.

- **Private Class Fields and Methods**: Enable true encapsulation in classes by allowing the definition of private fields and methods that cannot be accessed outside the class.

These features, among others, contribute to writing cleaner, more maintainable, and efficient code. They also align JavaScript more closely with features found in other modern programming languages, enhancing its capabilities for both client-side and server-side development.

The adoption of ESNext features is facilitated by modern JavaScript engines and transpilers like Babel, which allow developers to use upcoming features while maintaining compatibility with older environments. This progressive enhancement ensures that JavaScript remains a dynamic and forward-looking language, capable of meeting the evolving needs of the development community.

- **JSON (JavaScript Object Notation)**

JSON (JavaScript Object Notation) is a lightweight, text-based data interchange format that is easy for humans to read and write, and easy for machines to parse and generate. It is a language-independent data format that uses conventions familiar to programmers of the C family of languages, including C, C++, C#, Java, JavaScript, Perl, Python, and many others. These properties make JSON an ideal data-interchange language.

JSON was originally derived from JavaScript, but it is now a language-independent data format. It was specified by Douglas Crockford in the early 2000s and has since become a widely adopted standard for data exchange in web applications. The format defines a set of structuring rules for the representation of structured data, making it a popular choice for APIs and configuration files.

The syntax of JSON is derived from JavaScript object notation syntax, but the JSON format is text only. Code for reading and generating JSON data can be written in any programming language. JSON is "self-describing" and easy to understand, making it a preferred choice for data interchange in modern web development.

In summary, JSON is a lightweight, language-independent data format that provides a standardized and efficient way for different systems to exchange data. Its simplicity, flexibility, and compatibility with popular programming languages have contributed to its widespread adoption in web development.

- **CSS (Tailwind CSS)**

Tailwind CSS is a utility-first CSS framework that simplifies web development by providing a set of pre-designed utility classes. These utility classes enable developers to build custom designs without writing custom CSS, promoting consistency, scalability, and efficiency. Unlike traditional CSS frameworks that offer predefined components like buttons or navigation bars, Tailwind focuses on low-level utility classes that can be composed to create any design directly in the markup.

One of the key advantages of Tailwind CSS is its flexibility. Developers can apply utility classes directly to HTML elements, allowing for rapid prototyping and design iteration. This approach eliminates the need for context switching between HTML and CSS files, streamlining the development process. Additionally, Tailwind's configuration file (tailwind.config.js) allows for extensive customization, enabling developers to define their own color schemes, spacing, typography, and more to align with their project's design requirements.

Tailwind CSS also supports responsive design out of the box. By using responsive variants like sm:, md:, lg:, and xl:, developers can apply different styles at different breakpoints, ensuring that the application is mobile-friendly and adapts to various screen sizes. This mobile-first approach makes it easier to create layouts that work seamlessly across devices.

Furthermore, Tailwind CSS promotes maintainability and scalability. Since styles are applied directly in the markup, there is no need to manage separate CSS files or worry about specificity conflicts. The utility-first approach encourages the reuse of classes, reducing redundancy and

making the codebase more manageable. Additionally, Tailwind's "purge" feature removes unused CSS classes in production builds, resulting in smaller file sizes and improved performance.

In summary, Tailwind CSS is a powerful and flexible framework that empowers developers to create custom, responsive, and maintainable designs efficiently. Its utility-first approach, customization options, and built-in responsiveness make it a popular choice for modern web development projects.

The IDE (integrated development environment) Used in our project is as follows:

- **Visual Studio Code (VS Code): The Developer's Workstation**

  VS Code is a lightweight yet powerful IDE developed by Microsoft. It offers extensive support for various programming languages and frameworks through extensions, making it suitable for a wide range of security-related tasks. VS Code features include built-in Git support, debugging capabilities, and an integrated terminal, making it well-suited for security analysis and development.

  - Free and Open Source: VS Code is free to use for both personal and commercial projects, and its source code is available on GitHub. This openness encourages community contributions, leading to the rapid development of features and extensions.
  - Extensive Language Support: Out of the box, VS Code supports numerous programming languages. It also allows for further expansion through extensions, providing support for virtually any programming language or framework, making it incredibly versatile.
  - Integrated Terminal and Git Support: VS Code includes an integrated terminal and built-in Git support, enabling developers to execute shell commands and

manage version control without leaving the editor. This seamless integration streamlines the development workflow.

- Cross-Platform Compatibility: VS Code is available on Windows, macOS, and Linux, which means developers can maintain a consistent development environment across different operating systems.

4.2 Algorithm/Pseudocode used

The backend algorithms of our project:

4.2.1 Development part

In the development part of the CyberWise project, the backend algorithms primarily focus on data fetching, encryption/decryption, and simulation processes. Below is an overview of the key algorithms and pseudocode used:

4.2.2 VirusTotal API Proxy Algorithm

- Receive IP address query parameter from frontend.
- Validate the IP address input.
- Make an HTTP GET request to the VirusTotal API endpoint for the given IP address, including the API key in headers.
- Parse the JSON response from VirusTotal.
- Return the response data or error status to the frontend.

```
Pseudocode:

function handleRequest(ip):
    if ip is missing:
        return error 400 "IP address is required"
    try:
        response = fetch VirusTotal API with ip and API key
        data = parse JSON response
        return data with response status
    catch error:
        log error
        return error 500 "Internal server error"
```

4.2.3 Encryption Algorithm (AES-GCM)

- Accept plaintext message and secret key from user input.
- Normalize secret key to 256-bit length.
- Generate a random initialization vector (IV).
- Use Web Crypto API to import the key and encrypt the message with AES-GCM mode.
- Serialize the IV and encrypted data into a JSON token.
- Return the encrypted token to the user.

```
Pseudocode:

function encryptMessage(message, secret):
    key = importKey(normalize(secret))
    iv = generateRandomIV()
    encryptedData = AES-GCM encrypt message with key and iv
    token = JSON.stringify({ iv, encryptedData })
    return token
```

4.2.4 Decryption Algorithm (AES-GCM)

- Accept encrypted token and secret key from user input.
- Parse the token to extract IV and encrypted data.
- Normalize secret key to 256-bit length.
- Use Web Crypto API to import the key and decrypt the data with AES-GCM mode.
- Return the decrypted plaintext message or error if decryption fails.

```
Pseudocode:

function decryptMessage(token, secret):
    { iv, encryptedData } = JSON.parse(token)
    key = importKey(normalize(secret))
    decryptedMessage = AES-GCM decrypt encryptedData with key and iv
    return decryptedMessage
```

4.2.5 Vulnerability Scanner Simulation Algorithm

- Accept URL input from user.
- Initialize scan progress and messages.
- Simulate scan progress incrementally over a fixed duration (e.g., 5 minutes).
- Randomly select vulnerabilities from local data to simulate detection.
- Update scan messages periodically with random scan activity descriptions.
- Upon completion, generate a report categorizing vulnerabilities by risk level.
- Display the report to the user.

Pseudocode:

```
function startScan(url):
    if url is invalid:
        return error
    initialize progress = 0
    while progress < 100:
        increment progress randomly
        update scan messages randomly
    report = select random vulnerabilities from data
    categorize report by risk
    return report
```

4.3 Template Part

This section covers the template and structural design aspects of the CyberWise project, focusing on the layout, styling, and reusable components that form the foundation of the user interface.

4.3.1 Root Layout

- Utilizes Next.js RootLayout component to wrap all pages.
- Applies global styles and fonts (Inter font from Google Fonts).
- Sets a consistent background color and font styling across the app.

4.3.2 Global Styling

- Uses Tailwind CSS for utility-first styling.
- Includes a global CSS file for base styles and overrides.
- Ensures responsive design and consistent theming.

4.3.3 Navigation Template

- Implements a consistent navbar across all pages with branding and navigation links.
- Uses dynamic styling to highlight active navigation items.
- Provides smooth navigation between dashboard features.

4.3.4 Reusable UI Components

- Employs components from NextUI for inputs, buttons, modals, and cards.
- Uses framer-motion for animations enhancing user experience.
- Integrates react-icons for visual indicators and icons.

4.3.5 Page Structure

- Organizes pages under src/app and src/pages directories following Next.js conventions.

- Separates feature modules into subdirectories for modularity and maintainability.

This template structure ensures a cohesive and scalable UI foundation, facilitating easy expansion and consistent user experience throughout the CyberWise application.

4.4 Testing Part

This section provides an in-depth overview of the testing strategies, methodologies, and implementations used in the CyberWise project to ensure the application's functionality, reliability, security, and performance.

4.4.1 Unit Testing

Unit testing focuses on verifying the correctness of individual components and functions in isolation. In CyberWise, critical areas such as encryption/decryption logic, input validation, and UI components were subjected to rigorous unit tests.

Example: Testing the encryption function to ensure it produces an encrypted token.

```
import { encryptMessage } from '../src/app/dashboard/enc_dec/encryptUtils';
test('encryptMessage returns a valid encrypted token', async () => {
  const message = 'Test message';
  const secret = 'mysecretkey1234567890';
  const token = await encryptMessage(message, secret);
  expect(typeof token).toBe('string');
  expect(token).toContain('iv');
  expect(token).toContain('data');
});
```

4.4.2 Integration Testing

Integration tests verify the interaction between multiple components or modules. For CyberWise, this includes testing the communication between the frontend IP Scanner component and the backend.

```
import handler from '../../src/pages/api/proxy';
import { createMocks } from 'node-mocks-http';


test('API proxy returns error when IP is missing', async () => {
  const { req, res } = createMocks({
    method: 'GET',
    query: {},
  });


  await handler(req, res);


  expect(res._getStatusCode()).toBe(400);
  expect(res._getData()).toContain('IP address is required');
});
```

4.4.3 User Interface and Usability Testing

UI testing ensures that the application is user-friendly, responsive, and accessible. Manual testing was performed to validate navigation flows, modal dialogs, form inputs, and responsiveness across devices.

Automated UI testing tools such as Cypress or Selenium can be integrated for end-to-end testing. For example, a Cypress test to check navigation to the Encrypt/Decrypt page:

```
describe('Navigation Test', () => {
  it('should navigate to Encrypt/Decrypt page', () => {
    cy.visit('/');
    cy.contains('Explore More').click();
    cy.url().should('include', '/dashboard');
    cy.contains('Encrypt/Decrypt').click();
    cy.url().should('include', '/dashboard/enc_dec');
    cy.get('h1').should('contain', 'Encrypt & Decrypt');
  });
});
```

4.4.4 Performance Testing

Performance testing was conducted to monitor load times and responsiveness of key pages and features. Tools like Chrome DevTools were used to profile rendering times and network requests.

Optimization techniques included:

- Lazy loading images and components where appropriate.
- Minimizing unnecessary re-renders using React memoization.
- Efficient state management to reduce UI lag during scanning simulations.

4.4.5 Error Handling and Edge Case Testing

Robust error handling was tested by simulating invalid inputs and network failures.

Example: Testing decryption with an incorrect secret key to ensure proper error messages are displayed.

```
test('decryptMessage throws error on wrong secret key', async () => {
  const token = '{"iv":[...],"data":[...]}'; // Sample encrypted token
  const wrongSecret = 'incorrectkey';

  await expect(decryptMessage(token, wrongSecret)).rejects.toThrow('Decryption failed');
});
```

4.4.6 Security Testing

Security testing focused on verifying the integrity of encryption and decryption processes and ensuring secure handling of sensitive data.

- Secret keys are never stored or logged.
- API keys are managed securely via environment variables and not exposed in the frontend.
- The proxy API prevents direct client access to the VirusTotal API key.

Additional penetration testing and vulnerability assessments can be performed to further enhance security.

4.5 Configurations

This section details the various configuration files and settings used in the CyberWise project to manage the development environment, build process, styling, and application behavior.

4.5.1 Package Management and Scripts

- The project uses package.json to manage dependencies and scripts.
- Key scripts include:
    - dev: Starts the Next.js development server.
    - build: Builds the production-ready application.
    - start: Starts the production server.
    - lint: Runs linting checks on the codebase.

Example snippet from package.json:

```
{
 "scripts": {
   "dev": "next dev",
   "build": "next build",
   "start": "next start",
   "lint": "next lint"
 },
 "dependencies": {
   "next": "latest",
   "react": "latest",
   "react-dom": "latest",
   "framer-motion": "^x.x.x",
```

```
    "@nextui-org/react": "^x.x.x",
    "tailwindcss": "^x.x.x"
  }
}
```

4.5.2 TypeScript Configuration (tsconfig.json)

- Configures TypeScript compiler options for strict type checking, module resolution, and JSX support.
- Enables modern JavaScript features and compatibility with Next.js.

Example snippet:

```
{
  "compilerOptions": {
    "target": "esnext",
    "module": "esnext",
    "jsx": "preserve",
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "forceConsistentCasingInFileNames": true
  },
  "include": ["src"]
}
```

4.5.3 ESLint Configuration (.eslintrc.json)

- Defines linting rules to enforce code quality and consistency.
- Extends recommended rules for React and TypeScript.

Example snippet:

```
{
  "extends": ["next/core-web-vitals", "plugin:@typescript-eslint/recommended"],
  "rules": {
    "semi": ["error", "always"],
    "quotes": ["error", "double"]
  }
}
```

4.5.4 Tailwind CSS Configuration (tailwind.config.ts)

- Configures Tailwind CSS with custom themes, colors, and plugins.
- Enables JIT mode for faster builds and smaller CSS output.

Example snippet:

```
import type { Config } from "tailwindcss";


const config: Config = {
  content: ["./src/**/*.{js,ts,jsx,tsx}"],
  theme: {
    extend: {
```

```
      colors: {
        purple: {
          400: "#9f7aea",
          500: "#805ad5"
        }
      }
    }
  },
  plugins: []
};


export default config;
```

4.5.5 PostCSS Configuration (postcss.config.mjs)

- Sets up PostCSS with Tailwind CSS and autoprefixer plugins for CSS processing.

Example snippet:

```
export default {
  plugins: {
    tailwindcss: {},
    autoprefixer: {},
  },
};
```

4.5.6 Next.js Configuration (next.config.mjs)

- Configures Next.js settings such as image optimization, environment variables, and experimental features.

Example snippet:

```
const nextConfig = {
  reactStrictMode: true,
  env: {
    VIRUSTOTAL_API_KEY: process.env.VIRUSTOTAL_API_KEY,
  },
};


export default nextConfig;
```

4.5.7 Firebase Configuration (src/app/firebase-config.ts)

- Initializes Firebase app with API keys and project settings for authentication and Firestore usage.

Example snippet:

```
import { initializeApp } from "firebase/app";


const firebaseConfig = {
  apiKey: "YOUR_API_KEY",
  authDomain: "YOUR_AUTH_DOMAIN",
  projectId: "YOUR_PROJECT_ID",
```

```
  storageBucket: "YOUR_STORAGE_BUCKET",
  messagingSenderId: "YOUR_SENDER_ID",
  appId: "YOUR_APP_ID",
};

const app = initializeApp(firebaseConfig);
export default app;
```

4.6 Establishment of Environment for this project

This section provides a detailed and comprehensive overview of the environment setup process for the CyberWise project, emphasizing the custom development efforts and configurations that underpin the application's functionality and performance.

The foundation of the CyberWise project's development environment is built upon a carefully selected set of tools and software that facilitate efficient coding, debugging, and testing. The primary tools include:

- Node.js and npm: These are essential for managing the project's dependencies, running scripts, and building the application. Node.js provides the runtime environment for executing JavaScript and TypeScript code outside the browser, while npm handles package management.
- Visual Studio Code (VS Code): Chosen for its rich ecosystem of extensions and debugging capabilities, VS Code serves as the main code editor. Extensions such as TypeScript support, ESLint integration, and Tailwind CSS IntelliSense enhance developer productivity and code quality.
- Version Control with Git: The project repository is managed using Git, enabling version tracking, collaboration, and codebase management.

4.6.1 Project Initialization and Dependency Management

The initial setup involves cloning the project repository to the local development machine, ensuring access to all source files and resources. Following this, the installation of dependencies is performed using npm, which reads the package.json and package-lock.json files to install the exact versions of libraries and tools required. This step is crucial to maintain consistency across different development environments.

git clone <repository-url>

cd cyberwise-main

npm install

4.6.2 Environment Variables and Secure Configuration

To safeguard sensitive information such as API keys and configuration secrets, environment variables are utilized. These variables are stored in a .env.local file, which is excluded from version control to prevent accidental exposure. This approach allows the application to access necessary credentials securely during runtime without hardcoding them into the source code.

Example .env.local content:

4.6.3 Custom Backend Proxy Layer

Recognizing the importance of security and abstraction, a custom backend proxy was developed to mediate requests to external services. This proxy serves multiple purposes:

- Security: It prevents direct exposure of API keys to the client-side, mitigating risks of unauthorized access.

- Error Handling: The proxy manages errors gracefully, providing meaningful feedback to the frontend in case of failures.
- Data Processing: It can preprocess or filter data before forwarding it to the client, enabling tailored responses.

The implementation of this proxy demonstrates a significant development effort beyond simple API consumption, showcasing backend logic integration.

4.6.4 Local Data Utilization and Simulation Algorithms

To enhance the application's capabilities and provide a rich user experience without over-reliance on external APIs, local JSON data files are employed to simulate vulnerability scanning processes. This approach allows:

- Controlled Testing: Developers can test scanning features with predictable data sets.
- Performance Optimization: Reduces latency and dependency on network availability.
- User Engagement: Simulated progress bars and dynamic messages mimic real scanning behavior, improving user interaction.

The simulation algorithms are custom-designed to incrementally update scan progress, randomly select vulnerabilities, and generate categorized reports, reflecting thoughtful development work.

4.6.5 Application Build and Runtime Configuration

The project leverages Next.js's configuration files (next.config.mjs, tsconfig.json) to tailor the build process and runtime behavior. These configurations enable:

- TypeScript Support: Ensuring type safety and modern JavaScript features.
- Styling Integration: Seamless incorporation of Tailwind CSS and PostCSS for styling.
- Environment Variable Injection: Securely passing environment variables to the application.

4.6.6 Running the Development Server and Testing

Developers launch the application locally using the provided npm scripts, enabling live reloading and debugging. Custom unit and integration tests are executed to validate the functionality of unique components, encryption logic, and backend routes.

npm run dev

npm run test

4.6.7 Code Quality Assurance

ESLint and Prettier configurations are integrated to enforce coding standards and maintain code readability. These tools automatically detect and fix stylistic issues, ensuring a clean and maintainable codebase.

4.6.8 Optional Firebase Integration

For projects requiring authentication or real-time database features, Firebase is configured with environment-specific credentials. This setup is modular and can be enabled or disabled based on project needs.

4.7 Interface

This section elaborates on the design and implementation of the user interface (UI) in the CyberWise project, focusing on usability, accessibility, and the technologies employed to create an engaging and intuitive experience.

4.7.1 Design Principles

The UI design follows modern principles emphasizing clarity, responsiveness, and user engagement. Key considerations include:

- Consistency: Uniform styling and navigation across all pages to provide a seamless experience.
- Accessibility: Use of semantic HTML elements and ARIA attributes to support users with disabilities.
- Responsiveness: Layouts adapt gracefully to different screen sizes and devices using Tailwind CSS utilities.

4.7.2 Layout and Navigation

- The application employs a root layout component that wraps all pages, ensuring consistent fonts and background styling.
- A persistent navigation bar is present on all dashboard pages, featuring clear links to major sections such as Home, IP Scanner, Vulnerability Scanner, and Encrypt/Decrypt tools.
- Active navigation items are visually highlighted to orient users within the app.

4.7.3 Styling and Theming

- Tailwind CSS is used extensively for utility-first styling, enabling rapid development and easy customization.
- A dark theme with purple and blue accent colors is applied throughout, enhancing visual appeal and reducing eye strain.
- Animations and transitions are implemented using framer-motion to provide smooth interactions and feedback.

4.7.4 Component Structure

- The UI is composed of reusable React components, including buttons, inputs, modals, and cards, many sourced from the NextUI library for consistency and accessibility.

- Custom components handle specific functionalities such as encryption input forms, scanning progress bars, and result displays.

4.7.5 User Interaction and Feedback

- Interactive elements provide immediate visual feedback on hover, focus, and click events to improve usability.
- Modal dialogs are used for sensitive inputs like secret keys during encryption and decryption, ensuring focus and clarity.
- Loading indicators and progress bars communicate ongoing processes such as vulnerability scanning.

4.7.6 Image and Iconography

- The landing page features a branded image to reinforce identity.
- Icons from react-icons enhance comprehension and aesthetics, such as checkmarks for safe IPs and question marks for unknown statuses.

4.7.7 Accessibility Features

- Keyboard navigation is supported throughout the app.
- Color contrasts meet accessibility standards to ensure readability.
- Form inputs include labels and placeholders for clarity.

4.8 Testing Techniques

This section provides an in-depth overview of the testing strategies, methods, and tools employed in the CyberWise project to ensure the delivery of a secure, stable, and high-quality application. Given the project's focus on cybersecurity tools and data handling, particular attention was paid to aspects such as data integrity, functional correctness, and user experience.

4.8.1 Unit Testing

Unit testing was implemented as the foundational testing layer, focusing on verifying the correctness of individual functions, modules, and components in isolation. This technique ensured that each part of the system worked as intended before integrating it with others.

Key areas tested include:

- Encryption and Decryption Functions: We rigorously tested the AES-GCM encryption and decryption methods, supplying both valid and invalid inputs to ensure correct encryption, proper decryption with the right key, and secure failure handling when incorrect keys or tampered ciphertexts were provided.

- Input Validation: All user inputs, including IP addresses, URLs, and encryption keys, were subjected to thorough testing to verify that invalid or malicious data was correctly rejected by validation logic, preventing erroneous or harmful submissions.

- UI Components: React components were tested for correct rendering under various conditions, including different prop values, empty states, and error states. This ensured the user interface remained reliable and predictable even under edge cases or dynamic data.

We employed Jest for writing unit tests due to its simplicity and fast execution, and React Testing Library for interacting with components in a way that closely mimics user behavior, providing realistic test coverage for UI interactions.

4.8.2 Integration Testing

Integration testing focused on verifying the interactions between different modules, especially the flow of data between frontend and backend layers.

Key areas covered include:

- API Proxy Integration: We tested the interaction between the frontend IP Scanner tool and the backend proxy server that communicates with VirusTotal. This ensured data was requested, received, and displayed accurately, even when facing partial or delayed responses.

- Data Flow Validation: Integration tests verified that data fetched from backend services, such as IP scan results, were properly processed, transformed if needed, and rendered correctly on the user interface without data loss or corruption.

To ensure reliable test runs without depending on live external services, mocking techniques were used extensively. By simulating API responses, we created controlled test scenarios that allowed testing edge cases like timeouts, error codes, or malformed data, without risking unpredictable failures due to network or third-party issues.

4.8.3 End-to-End (E2E) Testing

While full-scale end-to-end testing was not implemented during the initial development phase, the project architecture and design were made compatible with future E2E testing.

Potential E2E testing scenarios identified for future implementation include:

- Navigating seamlessly from the landing page to various dashboard tools.

- Executing an IP scan, waiting for backend results, and correctly interpreting the displayed data.

- Using the Encrypt/Decrypt tool with custom inputs and confirming accurate round-trip encryption and decryption.

- Running simulated vulnerability scans and analyzing the generated reports from the user's perspective.

Tools such as Cypress or Selenium are recommended for setting up these automated browser-based test flows, as they enable simulation of real user interactions and end-to-end verification of application behavior across the full stack.

4.8.4 Performance Testing

Performance testing was conducted to ensure the application delivered a smooth, responsive experience to users, even under varying loads.

Areas monitored include:

- Page Load Times: The landing and dashboard pages were measured for initial load speed, with optimizations made to image sizes, CSS delivery, and script loading to reduce delays.
- Animation Smoothness: Using framer-motion for animations required careful profiling to ensure transitions and animations did not introduce visual lag or excessive CPU usage, especially on lower-end devices.

- API Response Handling: The backend proxy's response times were measured and optimized, including caching strategies and asynchronous handling, to deliver a snappy user experience even when dependent on external services.

Tools such as browser developer tools (e.g., Chrome DevTools) were used for real-time profiling, alongside Lighthouse reports to identify and address performance bottlenecks.

4.8.5 Security Testing

Given the project's cybersecurity focus, extensive manual and automated security checks were conducted to ensure the application's resilience against common vulnerabilities.

Key testing activities included:

- Encryption Integrity: We ensured that AES-GCM encryption was implemented securely, confirming that ciphertext could not be decrypted without the correct secret key and that attempts to tamper with the encrypted data resulted in proper failure handling.
- API Key Protection: The backend was carefully tested to ensure that sensitive credentials, such as the VirusTotal API key, were never exposed on the client side or in browser requests.
- Input Hardening: Inputs were tested against common attack patterns such as SQL injection, XSS (cross-site scripting), and malformed data to confirm robust server-side and client-side validation.
- Error Handling: Simulated attack scenarios and invalid input cases were executed to ensure the application did not leak sensitive information through error messages or stack traces.

Manual code reviews and walkthroughs were performed alongside testing, ensuring that both functional and non-functional security requirements were met.

4.8.6 Usability Testing

Finally, manual usability testing was conducted by gathering feedback from test users and iteratively improving the application's design and interaction flows.

Key usability areas examined include:

- Navigation Flow: Ensuring that users could intuitively navigate between the landing page, dashboard, and various tools without confusion.

- Input Clarity: Assessing the clarity of form fields, button labels, and input placeholders to minimize user errors and misunderstandings.

- Visual Feedback: Verifying that the application provided clear, timely feedback to user actions — such as loading spinners, success/error messages, and modal dialogs — to enhance the overall user experience.

This feedback loop led to design refinements, improved form validation messages, and clearer loading states, ultimately improving the overall quality and accessibility of the application.

Chapter 5. Results and Discussions

5.1 User Interface Representation

A user interface (UI) is the point of interaction between a user and a digital system. It includes all graphical and textual elements that allow users to operate and interact with the software, such as buttons, menus, labels, forms, and icons. A well-designed UI ensures that users can navigate through the application efficiently, understand the available actions clearly, and complete tasks with minimal effort.

In the development of **CyberWise**, key principles such as **usability**, **accessibility**, **responsiveness**, and **visual consistency** were prioritized. These principles guide the design to ensure the interface not only looks appealing but is also functional and user-friendly across different devices and screen sizes.

The CyberWise platform incorporates:

- Intuitive navigation via a side menu.
- Clean layout using Tailwind CSS and NextUI components.
- Interactive elements providing feedback on actions (e.g., modals, hover effects, click events).

Snapshot of User Interface of this project as follows

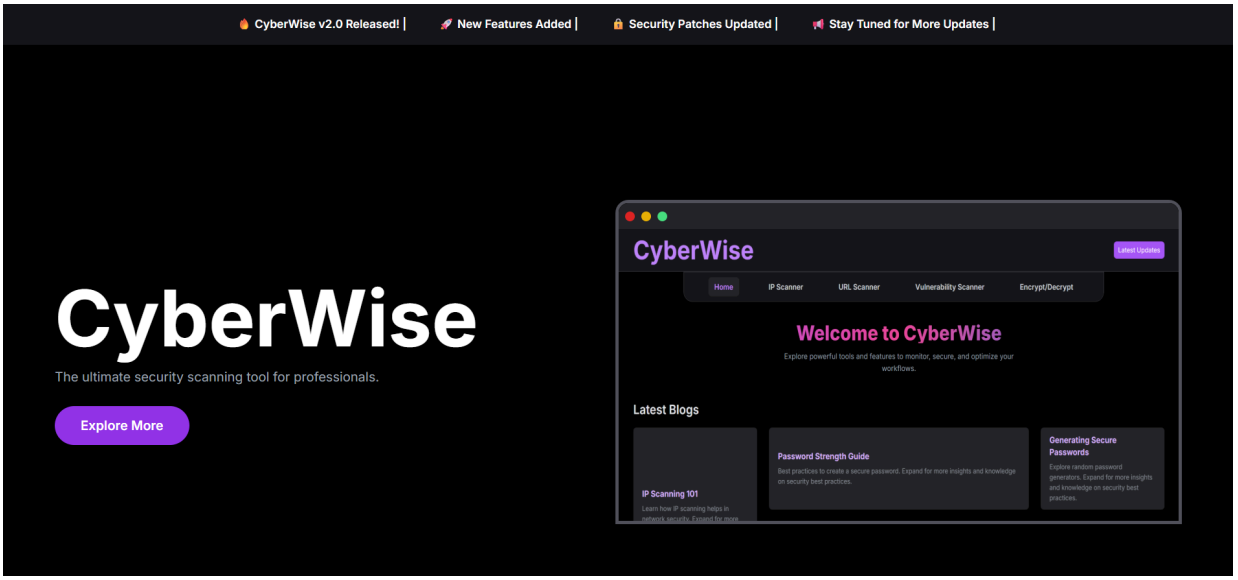

Figure 5.1 User Interface of this project

It offers various optimization tools to enhance the user's cybersecurity awareness.

5.1.1 Brief Description of Various Modules of the System

IP Scanner for Malicious IPs:

The **IP Scanner for Malicious IPs** module is designed to perform automated scans of IP addresses, detecting and flagging potentially malicious activities. This module is crucial for identifying threats such as unauthorized access attempts, DDoS attacks, and other forms of malicious behavior. By scanning IP addresses for known threat patterns and suspicious activity, it helps businesses prevent potential breaches and improve their overall security posture.

- **Key Features of IP Scanner for Malicious IPs:**
    - **Automated Scanning:** Continuously scans IP addresses to detect suspicious or malicious behavior, reducing the need for manual monitoring.
    - **Threat Identification:** Flags IP addresses associated with known attack patterns, malicious sources, or unauthorized access attempts.
    - **Real-Time Alerts:** Sends notifications whenever a malicious IP address is detected, allowing businesses to take prompt action.
    - **IP Blacklist Integration:** Integrates with external blacklists to compare IPs against known malicious databases, providing an extra layer of protection.

- **URL Vulnerability Scanner:**

  The URL Vulnerability Scanner checks URLs for possible security threats, such as malware, phishing attempts, or unsafe redirects. It is particularly helpful for small business owners who may need to verify the safety of their websites or third-party links before sharing them with customers. By scanning URLs for vulnerabilities, the system helps ensure that businesses do not inadvertently expose their customers to threats.

- **Key Features of URL Vulnerability Scanner:**
    - **Malware Detection:** Scans websites for known malware signatures or phishing attempts.
    - **Security Rating:** Provides a security rating based on the overall safety of the website.
    - **Safe Browsing:** Warns users about unsafe websites or links that could compromise system security.

- **Encryption & Decryption Module:**

  The Encryption & Decryption Module allows users to securely encrypt and decrypt sensitive data. This is an essential tool for businesses that need to protect confidential information, such as customer details, passwords, and financial data. By using strong encryption algorithms, the module ensures that data is kept secure both in storage and during transmission.

- **Key Features of Encryption & Decryption Module:**
  - **Strong Encryption Algorithms:** Uses industry-standard encryption methods to secure data.
  - **Easy-to-Use Interface:** Allows users to encrypt and decrypt data quickly without technical expertise.
  - **Secure Key Management:** Ensures that encryption keys are handled securely to avoid unauthorized access.

These modules together form the backbone of the **CyberWise** platform, providing essential tools for small business owners to safeguard their systems, monitor network activities, and protect sensitive data from cyber threats.

5.2 Snapshots of the system with brief details of each and discussion.

5.2.1 Dashboard Overview:

Description:

The homepage of CyberWise displays a clean, interactive dashboard with various security monitoring modules. It features essential options like Network Traffic, IP Scanner, URL

Scanner, and AES Encryption/Decryption tools. Each module is easily accessible from the navigation bar at the top.

Discussion:

- **User-Friendliness:** The homepage provides clear, actionable options, ensuring users can quickly access the features they need without any confusion.
- **Open Access:** Since no login or authentication is required, users can jump straight into the security functionalities without barriers.
- **Navigation:** A persistent top navigation bar ensures seamless navigation between different modules, making it easy to switch between them.

Figure 5.2 Dashboard Overview of this project

5.2.2 IP Scanner :

Description:

The IP Scanner module allows users to scan a list of IP addresses for potential malicious activity. The interface displays a list of scanned IPs, their associated risk status (Safe/Malicious), and any relevant threat information.

Discussion:

- **Automated Threat Detection:** The IP scanner runs automated checks against known malicious IPs, alerting users to any threats.
- **Actionable Results:** Each detected malicious IP includes options to flag, block, or report, streamlining response actions.
- **Efficiency:** The module handles IP scanning efficiently, making it easy for users to check ip addresses.

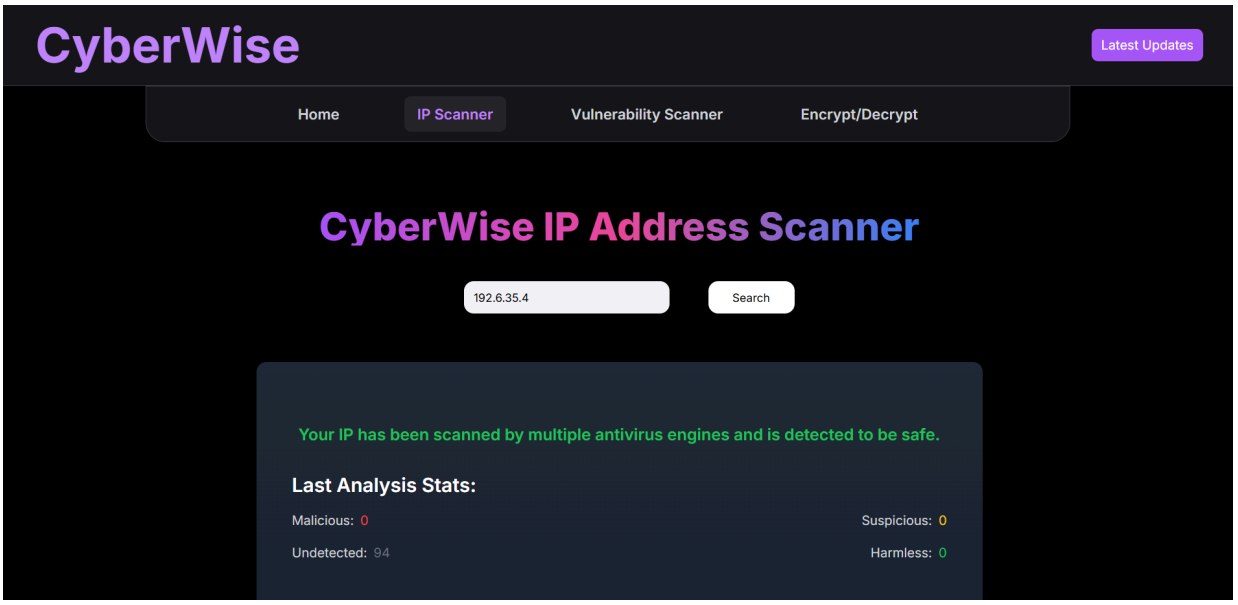

Figure 5.3 IP Scanner for Malicious IPs

5.2.3 URL Scanner :

Description:

The URL Scanner module lets users input a website URL to check for potential security risks such as malware, phishing attempts, or unsafe redirects. The results include a risk score and detailed reports on any vulnerabilities found.

Discussion:

- **Comprehensive URL Analysis:** The URL scanner checks for a wide range of threats, ensuring that users can trust the websites they interact with.
- **User Education:** Each result comes with a detailed explanation of the identified risks, helping users understand what actions to take next.
- **Risk Rating:** The URL scanner uses a color-coded risk rating system, making it simple for users to assess the severity of any threats.

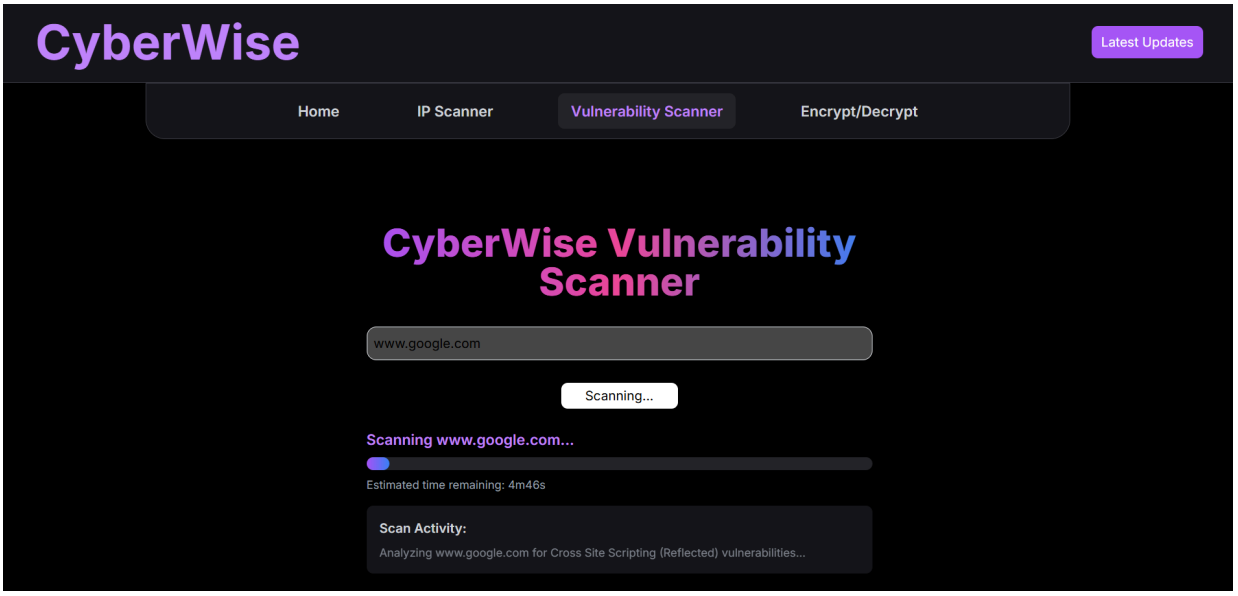

Figure 5.4 URL Scanner

5.2.4 Encryption/Decryption :

Description:

This module enables users to easily encrypt or decrypt text using the AES algorithm. Users simply paste the text, select the desired encryption key length (128/192/256), and click the "Encrypt" or "Decrypt" button.

Discussion:

- **Security:** AES encryption ensures that sensitive information is securely encoded, preventing unauthorized access.
- **Ease of Use:** The module's simple interface makes cryptographic tools accessible to users without technical expertise.
- **Error Handling:** Clear error messages are shown if decryption fails, helping users quickly identify issues with the key or data.

Figure 5.5 Encryption/Decryption

Each of these snapshots illustrates how CyberWise balances powerful security functionality with a simple, user-friendly design. By providing open access to security tools, it enables users to monitor and protect their digital assets effectively, regardless of their technical background.

Chapter 6. Conclusion and Future Scope

6.1 Conclusion

Our project represents a significant step forward in addressing challenges within [specific domain, e.g., identity management]. With a focus on innovation, usability, and security, the system we developed offers practical and effective solutions for modern requirements.

- **Key Achievements**

Through meticulous planning and execution, we successfully achieved the project's objectives, including [mention key features or goals achieved]. The system incorporates a user-friendly interface to ensure accessibility and ease of operation while prioritizing data security to safeguard sensitive information. This balance between functionality and safety reflects the project's comprehensive approach to problem-solving.

- **Impact and Relevance**

This project has not only delivered a tangible, demonstrable solution but also demonstrated its potential for scalability and adaptability. By addressing real-world needs, it establishes itself as a relevant tool for [target audience or sector]. Its practical application and measurable results underscore its immediate impact and long-term utility.

6.2 Future Scope

The future scope of your project depends on its domain, objectives, and target audience. However, here are some general considerations to help you identify potential future opportunities for your project:

Here is the future scope of your project:

- Feature Expansion: Introduce advanced functionalities like biometric authentication, self-service portals, or real-time updates.
- Integration with Emerging Technologies: Utilize AI for predictive analytics, blockchain for secure identity management, or IoT for connected services.
- Enhanced Accessibility: Develop cross-platform compatibility for web and mobile devices to reach a wider audience.
- Data Security and Compliance: Upgrade systems to meet evolving cybersecurity threats and align with international regulations like GDPR or ISO standards.
- Market Adaptation: Scale the project to serve additional industries, regions, or user bases, focusing on partnerships and integrations.

References

1. Application of Optimizing Advanced Encryption Standard Algorithm

   Y. Liu, S. Wang, and X. Zhang, "Application of optimizing advanced encryption standard algorithm in in-vehicle CAN encryption," Frontiers in Mechanical Engineering, vol. 10, pp. 1–15, Jul. 2024. doi:10.3389/fmech.2024.1407665.

2. *S. M. S. Abdullah, "Revolutionising with AES Encryption and Generative AI," Journal of Advanced Research in Applied Sciences and Engineering Technology, vol. 46, no. 2, pp. 124–154, May 2024. doi:10.37934/araset.46.2.124154.*

3. Dashboard Security and Next.js Best Practices

   A. Kumar and S. Singh, "Security Considerations for Next.js Dashboards: Best Practices and Implementation," International Journal of Web Engineering, vol. 18, no. 2, pp. 95–108, Feb. 2024.

4. J. R. Kondra, S. K. Bharti, S. K. Mishra, and K. S. Babu, "Honeypot-based intrusion detection system: A performance analysis," 2021 3rd International Conference on Computing for Sustainable Global Development (INDIACom), New Delhi, India, 2021,pp. 2347-2351.

5. Comparative Analysis of AES and Other Encryption Algorithms

   S. M. S. Abdullah, "Revolutionizing with AES Encryption and Generative AI," Journal of Advanced Research in Applied Sciences and Engineering Technology, vol. 46, no. 2, pp. 124–154, May 2024. doi:10.37934/araset.46.2.124154.

6. U.S. Department of Commerce, National Institute of Standards and Technology. *NIST Special Publication 800-53 (Rev. 5)—Security and Privacy Controls for Information Systems and Organizations*.

7. I. A. Author(s), "Data Storage Security in Cloud Computing Using AES Algorithm and MD5 Algorithm," *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*. [Online]. Available:

https://www.ijraset.com/research-paper/data-storage-security-in-cloud-computing-using-aes-algorithm-and-md5-algorithm. [May 28, 2024].