A Project Report On

**Phishing URL Detector**

Submitted in partial fulfillment of the requirement for the award of the degree

MASTER OF SCIENCE (CS & CL)

from

Marwadi University

Academic Year 2025 – 26

**GANDHAR REEZWAN -92400565020**

|  |
| --- |
| **Internal Guide** |
| PANKAJ MUDHOLKAR |



Rajkot-Morbi Road, At &PO :Gauridad, Rajkot 360 003. Gujarat. India.



**Faculty of Computer Applications (FoCA)**

****

**This is to certify that the project work entitled**

**Phishing URL Detector**

**submitted in partial fulfillment of the requirement for**

**the award of the degree of**

**Master of Science (CS & CL)**

**of the**

**Marwadi University**

**is a result of the bonafide work carried out by**

**GANDHAR REEZWAN -92400565020**

**during the academic year 2025 – 2026**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Faculty Guide** |  | **HOD** |  | **Dean** |

**DECLARATION**

We hereby declare that this project work entitled **Phishing URL Detector** is a record done by us.

We also declare that the matter embodied in this project is genuine work done by us and has not been submitted whether to this University or to any other University / Institute for the fulfillment of the requirement of any course of study.

Place:

Date:

GANDHAR REEZWAN - 92400565020 Signature:\_\_\_\_\_\_\_\_\_\_\_\_

**ACKNOWLEDGEMENT**

It is indeed a great pleasure to express our thanks and gratitude to all those who helped us. No serious and lasting achievement or success one can ever achieve without the help of friendly guidance and co-operation of so many people involved in the work.

We are very thankful to our guide **Dr. Pankaj Mudholkar,** the person who makes us to follow the right steps during our project work. We express our deep sense of gratitude to for his /her guidance, suggestions and expertise at every stage. A part from that his/her valuable and expertise suggestion during documentation of our report indeed help us a lot.

Thanks to our friend and colleague who have been a source of inspiration and motivation that helped to us during our project work.

We are heartily thankful to the Dean of our department **Dr. R. Sridaran** sir and HoD **Dr. Sunil Bajeja** sir for giving us an opportunity to work over this project and for their end-less and great support to all other people who directly or indirectly supported and help us to fulfil our task.

GANDHAR REEZWAN - 92400565020 Signature:\_\_\_\_\_\_\_\_\_\_\_

**CONTENTS**

|  |  |  |
| --- | --- | --- |
| **Chapters** | **Particulars** | **Page No.** |
| 1 | **Introduction**   * 1. Objective of the New System   2. Problem Definition   3. Core Components   4. Project Profile   5. Assumptions and Constraints   6. Advantages and Limitations of the Proposed System | 6-7 |
| 2 | **Requirement Determination & Analysis**   * 1. Requirement Determination   2. Targeted Users   3. Details of tools and techniques used / implemented   4. Advantages and Limitations of the used security tools | 8-9 |
| 3 | **System Design**   * 1. Flow chart   2. Use Case Diagram   3. Class Diagram   4. Data Flow Diagram   5. Activity Diagram | 10-13 |
| 4 | **Development**  4.1 Script details / Source code  4.2. Screen Shots / UI Design of simulation (if applicable)  4.3. Test reports | 14-23 |
| 5 | **Proposed Enhancements** | 24 |
| 6 | **Conclusion** | 25 |
| 7 | **Bibliography** | 26-27 |

# Chapter 1

# Introduction

The Phishing URL Detector is a web-based application built to help users identify potentially malicious links. The project uses Python Flask for the backend, an SQLite database for data storage, and a dynamic frontend with HTML, CSS, and JavaScript. Its core functionality is a heuristics-based detection engine that analyzes URLs for common phishing indicators. The system also includes a secure user authentication system, a comprehensive logging system for monitoring, and a full-featured admin dashboard to manage users, view all URL checks, and handle user feedback. The project demonstrates practical skills in full-stack development and cybersecurity.

### **Objective of the New System**

The primary objective of this project is to develop a web-based **Phishing URL Detector** that serves as a practical, accessible tool for identifying malicious Uniform Resource Locators (URLs). The system is designed to provide users with a quick and clear threat analysis, including a confidence score and detailed reasoning, thereby enhancing their personal cybersecurity posture. This project also serves as a comprehensive demonstration of full-stack development principles, secure user management, and system monitoring.

### **Problem Definition**

Phishing remains a prevalent and evolving cybersecurity threat, often succeeding by leveraging deceptive URLs that trick users into revealing sensitive information. The challenge lies in creating an effective and user-friendly solution that can proactively identify these malicious links. This project addresses this by building a tool that moves beyond simple blacklisting, utilizing a multi-layered detection approach to provide more nuanced and reliable threat verdicts.

### **Core Components**

The system is built on a modular, three-tier architecture:

* **Backend:** Developed in **Python** using the **Flask** web framework, responsible for application logic, routing, authentication, and data management.
* **Database:** A local **SQLite** database, managed by **Flask-SQLAlchemy**, for persistent storage of user accounts, URL check history, and feedback.
* **Frontend:** A dynamic web interface created with **HTML**, **CSS**, and **JavaScript** for user interaction and real-time display of analysis results.

### **Project Profile**

This project is a solo development effort that simulates a real-world application. It focuses on the core functionalities of a **Security Operations Center (SOC)**, specifically in threat detection and user support. The project demonstrates an understanding of secure coding practices, API integration, and comprehensive system logging, all of which are critical for a professional security role.

### **Assumptions and Constraints**

* **Assumptions:** The system assumes that users will have a working web browser and a stable internet connection. It also assumes that its heuristic rules are a reliable representation of common phishing tactics for this scale of project.
* **Constraints:** To manage the project scope within the given timeframe, the system does not use a complex, enterprise-grade database or a trained machine learning model. It relies on a free-tier external API, which may have usage limitations.

### **Advantages and Limitations of the Proposed System**

* **Advantages:** The system offers a clear, intuitive user experience, transparent detection logic with specific reasons, secure user authentication with distinct roles, and a robust logging system for monitoring.
* **Limitations:** The heuristic rules, while effective, can be bypassed by novel phishing techniques. The application is a proof-of-concept and not built for large-scale production environments.

# Chapter 2:

# Requirement Determination & Analysis

### **2.1 Requirement Determination**

The project's requirements were determined by defining the essential functions of a phishing detection tool. This included the need for user accounts, secure authentication, a mechanism to submit URLs, an intelligent analysis engine, a method for storing historical data, and a system for administrative oversight.

### **2.2 Targeted Users**

* **Client User:** This is the primary end-user who interacts with the URL detection tool. They need an easy-to-use interface to check URLs, view their personal history, and submit feedback.
* **Admin User:** This user has elevated privileges to manage the entire system. They require access to user management tools, feedback review, and a comprehensive view of all system activities.

### **2.3 Details of tools and techniques used / implemented**

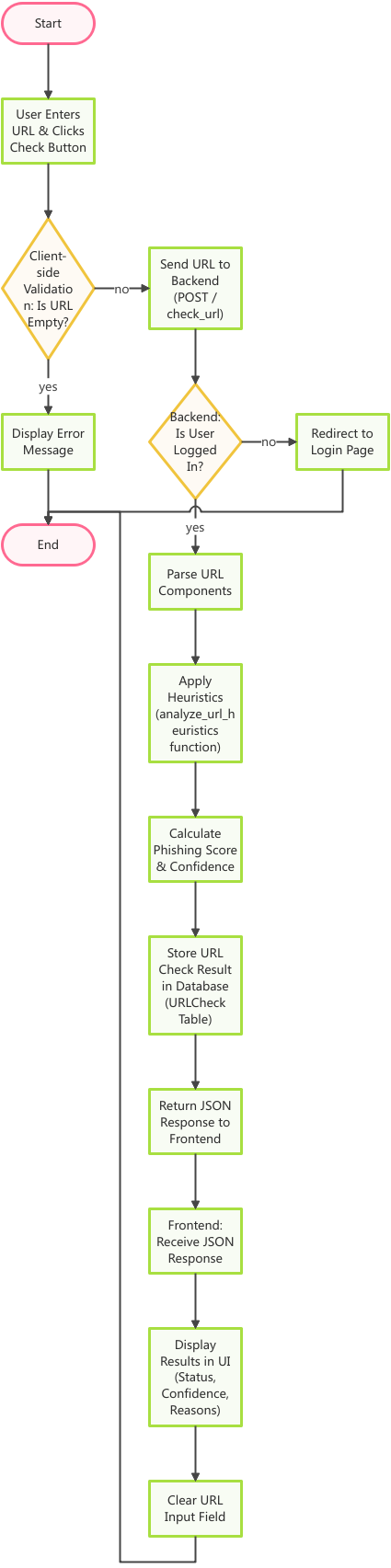
* **Python Libraries:** **Flask**, **Flask-SQLAlchemy**, **Flask-Login**, werkzeug.security (for password hashing), requests (for external API calls), and the native logging module.
* **Techniques:**
  + **Heuristics-Based Detection:** The system's primary detection method, analyzing URL components for known phishing indicators.
  + **External API Integration:** The application is architected to integrate with third-party threat intelligence APIs (e.g., Google Safe Browsing).
  + **Role-Based Access Control:** Protects routes and functionalities based on whether the user is a client or admin.
  + **Secure Authentication:** User passwords are not stored in plaintext; instead, they are salted and hashed.
  + **Centralized Logging:** Events such as successful/failed logins, user registration are automatically recorded with timestamps.

### **2.4 Advantages and Limitations of the used security tools**

* **Flask:** Lightweight, flexible, and fast for development. Its modular nature allows for easy integration of extensions.
* **SQLite:** Zero-configuration and easy to use, making it perfect for a project where a full database server is not needed.
* **Flask-Login:** Simplifies user session management, preventing common authentication vulnerabilities.
* **Limitations:** The Flask development server is not secure for public deployment. SQLite's performance can degrade with high concurrency.

# Chapter 3: System Design

### **3.1 Flow Chart Diagram**



The flow chart for the URL check process visually maps the logical steps from user input to the final result.

* **Start/End:** Represented by oval shapes.
* **Processes:** Rectangles represent actions like "User Enters URL," "Apply Heuristics," and "Display Result."
* **Decisions:** Diamonds represent conditional checks like "Is User Logged In?"
* **Data Stores:** A cylinder represents the database where data is read from and written to.

Figure :- 1.1

### **3.2 Use Case Diagram**

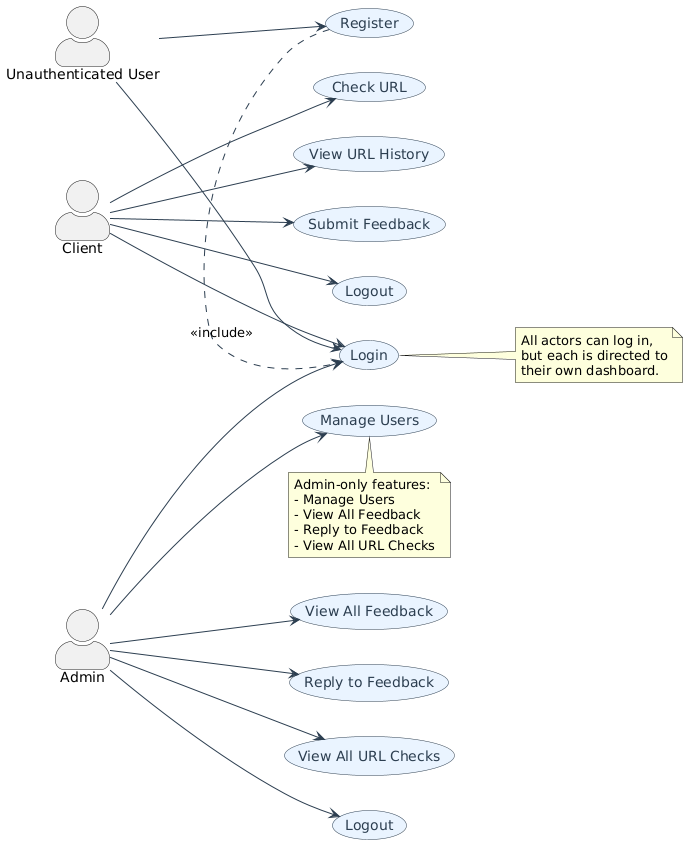


Figure :- 1.2

### **3.3 Class Diagram**

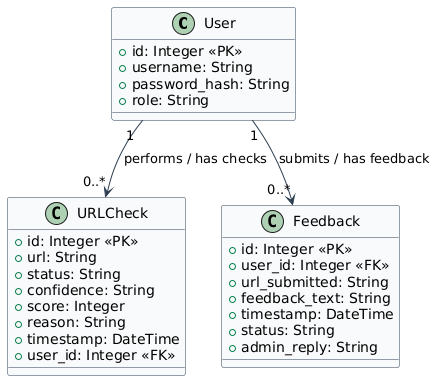


Figure :- 1.3

### **3.4 Data Flow Diagram (DFD)**

**DFD LEVEL - 0**

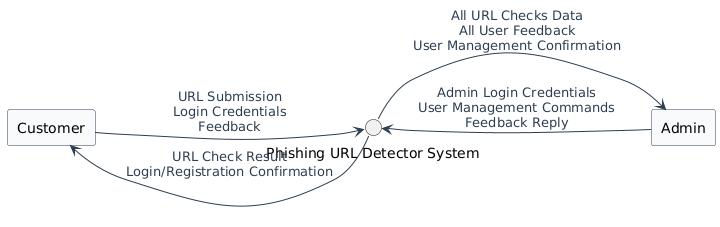


Figure :- 1.4

**DFD LEVEL – 1**

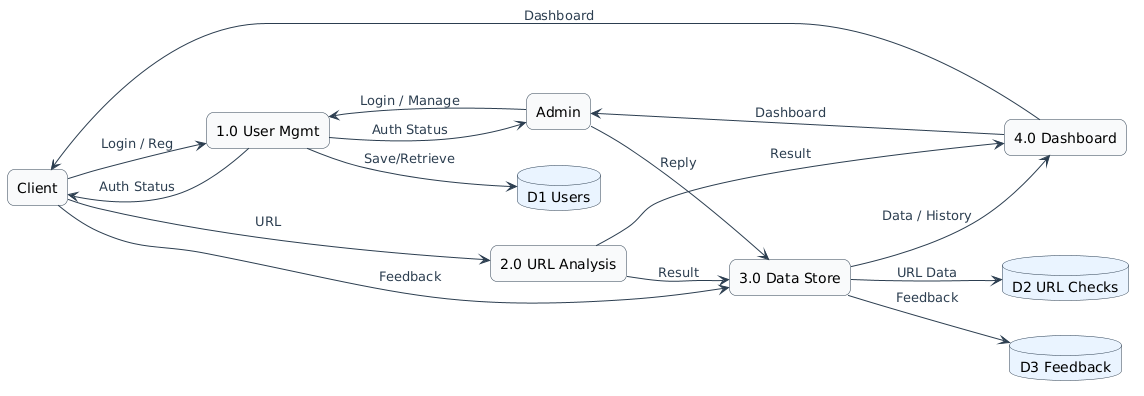


Figure :- 1.5

### **3.5 Activity Diagram**

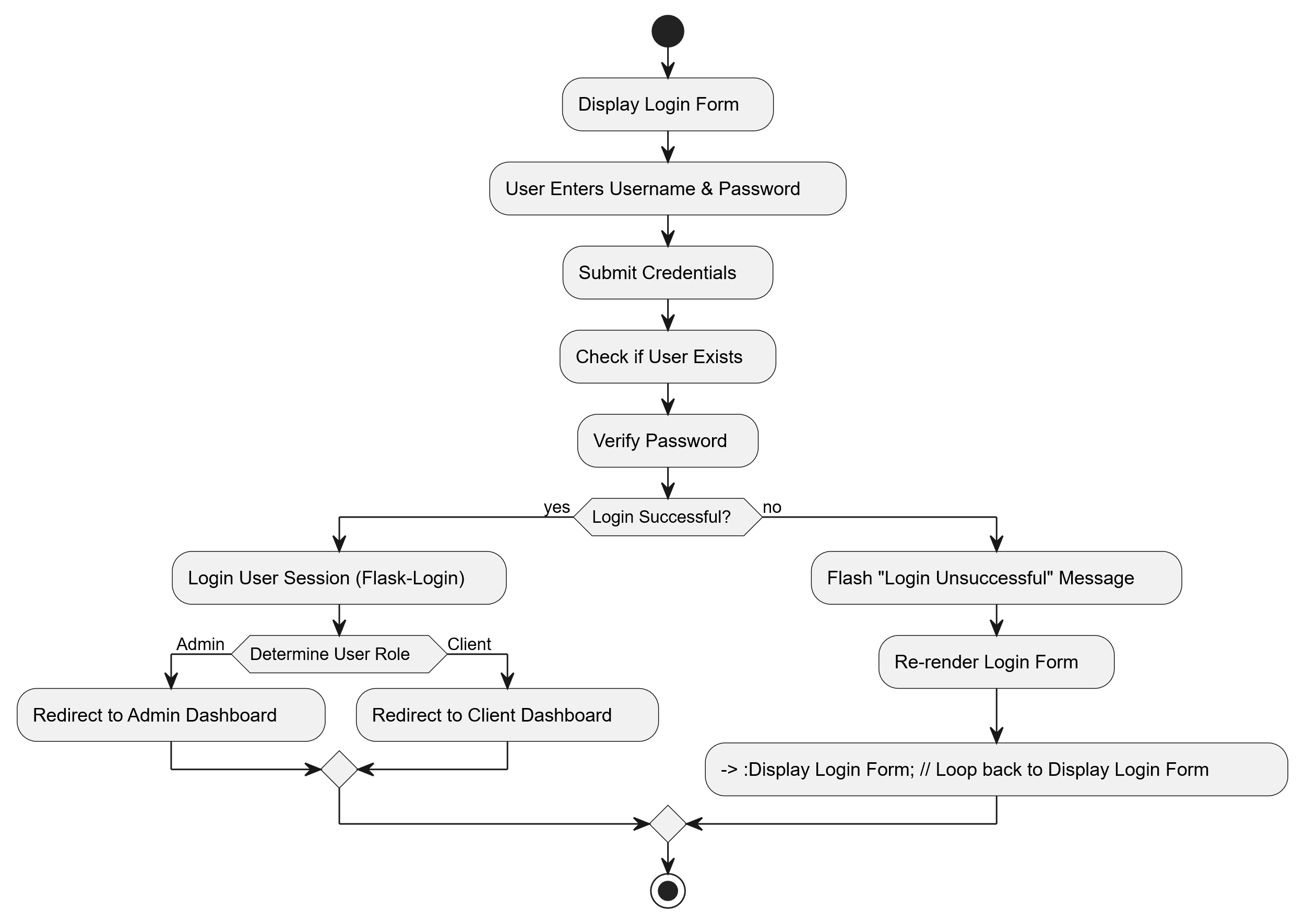
****

Figure :- 1.6

# Chapter 4:

# Development

### **4.1 Script details / Source code**

**app.py - Core Detection Logic (analyze\_url\_heuristics)**

This function is the brain of my application. It demonstrates the multi-layered detection engine that assigns a score, determines confidence, and provides the specific reasons for its verdict.

def analyze\_url\_heuristics(url):  
    is\_phishing = False  
    score = 0  
    reasons = []  
  
    # --- Step 1: Google Safe Browsing API Check (Highest Priority) ---  
    google\_safe\_browsing\_result = check\_with\_google\_safe\_browsing(url)  
  
    if google\_safe\_browsing\_result == "SOCIAL\_ENGINEERING":  
        is\_phishing = True  
        score += 15  
        reasons.append("Google Safe Browsing: Identified as Social Engineering (Phishing)")  
    elif google\_safe\_browsing\_result == "MALWARE":  
        is\_phishing = True  
        score += 20  
        reasons.append("Google Safe Browsing: Identified as Malware Site")  
    elif google\_safe\_browsing\_result == "UNWANTED\_SOFTWARE":  
        is\_phishing = True  
        score += 10  
        reasons.append("Google Safe Browsing: Identified as Unwanted Software Site")  
    elif google\_safe\_browsing\_result == "POTENTIALLY\_UNWANTED\_APPLICATION":  
        is\_phishing = True  
        score += 8  
        reasons.append("Google Safe Browsing: Identified as Potentially Unwanted Application Site")  
    elif google\_safe\_browsing\_result in ["API\_TIMEOUT", "API\_ERROR", None]:  
        if google\_safe\_browsing\_result is not None:  
            app.logger.warning(f"Safe Browsing API issue or skip. Proceeding with heuristics only for URL: {url}")  
    elif google\_safe\_browsing\_result == "SAFE":  
        pass  
  
    # --- Step 2: Existing Heuristics ---  
    parsed\_url = urlparse(url)  
    hostname = parsed\_url.hostname if parsed\_url.hostname else ''  
    path = parsed\_url.path if parsed\_url.path else ''  
    query = parsed\_url.query if parsed\_url.query else ''  
  
    if re.match(r'^\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3}$', hostname):  
        score += 3  
        reasons.append("IP address used in hostname")  
  
    if len(url) > 75:  
        score += 2  
        reasons.append(f"Excessive URL length ({len(url)} characters)")  
  
    if "@" in url:  
        score += 4  
        reasons.append("Contains '@' symbol (potential obfuscation)")  
    suspicious\_keywords = ["login", "signin", "verify", "account", "bank", "secure", "webscr", "update", "confirm", "service", "support", "billing", "ebay", "paypal", "amazon", "appleid", "icloud"]  
    url\_lower = url.lower()  
    for keyword in suspicious\_keywords:  
        if keyword in url\_lower:  
            if f".{keyword}." not in url\_lower and f"//{keyword}." not in url\_lower:  
                score += 1  
                reasons.append(f"Contains suspicious keyword: '{keyword}'")  
    common\_brands\_for\_mimicry = ["paypal", "google", "microsoft", "apple", "amazon", "ebay", "facebook", "whatsapp", "netflix", "bank", "secure"]  
    domain\_parts = hostname.split('.')  
    if len(domain\_parts) > 2:  
        main\_domain = ".".join(domain\_parts[-2:])  
        for brand in common\_brands\_for\_mimicry:  
            if brand in hostname.lower() and brand not in main\_domain.lower():  
                score += 3  
                reasons.append(f"Mismatched/suspicious brand '{brand}' in subdomain or path")  
                break  
  
    if len(domain\_parts) > 4:  
        score += 2  
        reasons.append(f"Excessive number of subdomains ({len(domain\_parts) - 2} beyond primary domain)")  
  
    shortening\_services = ["bit.ly", "tinyurl.com", "ow.ly", "goo.gl", "t.co", "rebrand.ly", "is.gd", "cli.gs"]  
    if any(service in hostname for service in shortening\_services):  
        score += 5  
        reasons.append("Uses a known URL shortening service")  
  
    if "Google Safe Browsing" in "".join(reasons):  
        is\_phishing = True  
        confidence = "High"  
        if google\_safe\_browsing\_result == "SOCIAL\_ENGINEERING":  
            status = "Phishing (High - GSB)"  
        elif google\_safe\_browsing\_result == "MALWARE":  
            status = "Malware (High - GSB)"  
        elif google\_safe\_browsing\_result == "UNWANTED\_SOFTWARE":  
            status = "Unwanted Software (High - GSB)"  
        elif google\_safe\_browsing\_result == "POTENTIALLY\_UNWANTED\_APPLICATION":  
            status = "PUA (High - GSB)"  
        else:  
            status = "Phishing (High - GSB & Heuristics)"  
        if score >= 10:

**app.py - Database Models**

These Python classes define the database schema and the relationships between your tables.

class User(db.Model, UserMixin):  
    id = db.Column(db.Integer, primary\_key=True)  
    username = db.Column(db.String(20), unique=True, nullable=False)  
    password\_hash = db.Column(db.String(128), nullable=False)  
    role = db.Column(db.String(10), default='client', nullable=False)  
      
    url\_checks = db.relationship('URLCheck', back\_populates='checker', lazy=True)  
    feedback\_entries = db.relationship('Feedback', back\_populates='author', lazy=True)  
  
    def set\_password(self, password):  
        self.password\_hash = generate\_password\_hash(password)  
  
    def check\_password(self, password):  
        return check\_password\_hash(self.password\_hash, password)  
  
    def \_\_repr\_\_(self):  
        return f"User('{self.username}', '{self.role}')"  
  
class URLCheck(db.Model):  
    id = db.Column(db.Integer, primary\_key=True)  
    url = db.Column(db.String(500), nullable=False)  
    status = db.Column(db.String(50), nullable=False)  
    confidence = db.Column(db.String(10), nullable=True)  
    score = db.Column(db.Integer, nullable=True)  
    reason = db.Column(db.String(500), nullable=True)  
    timestamp = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)  
    user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)  
      
    checker = db.relationship('User', back\_populates='url\_checks', lazy=True)  
  
    def \_\_repr\_\_(self):  
        return f"URLCheck('{self.url}', '{self.status}', '{self.timestamp}')"  
  
class Feedback(db.Model):  
    id = db.Column(db.Integer, primary\_key=True)  
    user\_id = db.Column(db.Integer, db.ForeignKey('user.id'), nullable=False)  
    url\_submitted = db.Column(db.String(500), nullable=False)  
    feedback\_text = db.Column(db.Text, nullable=False)  
    timestamp = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)  
    status = db.Column(db.String(20), default='pending', nullable=False)  
    admin\_reply = db.Column(db.Text, nullable=True)  
      
    author = db.relationship('User', back\_populates='feedback\_entries', lazy=True)  
  
    def \_\_repr\_\_(self):  
        return f"Feedback('{self.url\_submitted}', '{self.status}')"

**templates/index.html - Jinja2 History Loop**

This is the specific snippet from your client dashboard that dynamically generates the URL check history table.

{% if user\_checks %}  
    <table class="history-table">  
        <thead>  
            <tr>  
                <th>Date</th>  
                <th>URL</th>  
                <th>Status</th>  
                <th>Confidence</th>  
                <th>Score</th>  
                <th>Reason</th>  
            </tr>  
        </thead>  
        <tbody>  
            {% for check in user\_checks %}  
                <tr>  
                    <td>{{ check.timestamp.strftime('%Y-%m-%d %H:%M:%S') }}</td>  
                    <td class="url-cell">{{ check.url }}</td>  
                    <td><span class="{{ 'phishing' if 'Phishing' in check.status else 'legitimate' }}">{{ check.status }}</span></td>  
                    <td>{{ check.confidence }}</td>  
                    <td>{{ check.score }}</td>  
                    <td>{{ check.reason }}</td>  
                </tr>  
            {% endfor %}  
        </tbody>  
    </table>  
{% else %}  
    <p>No history yet. Check a URL to see it appear here.</p>  
{% endif %}

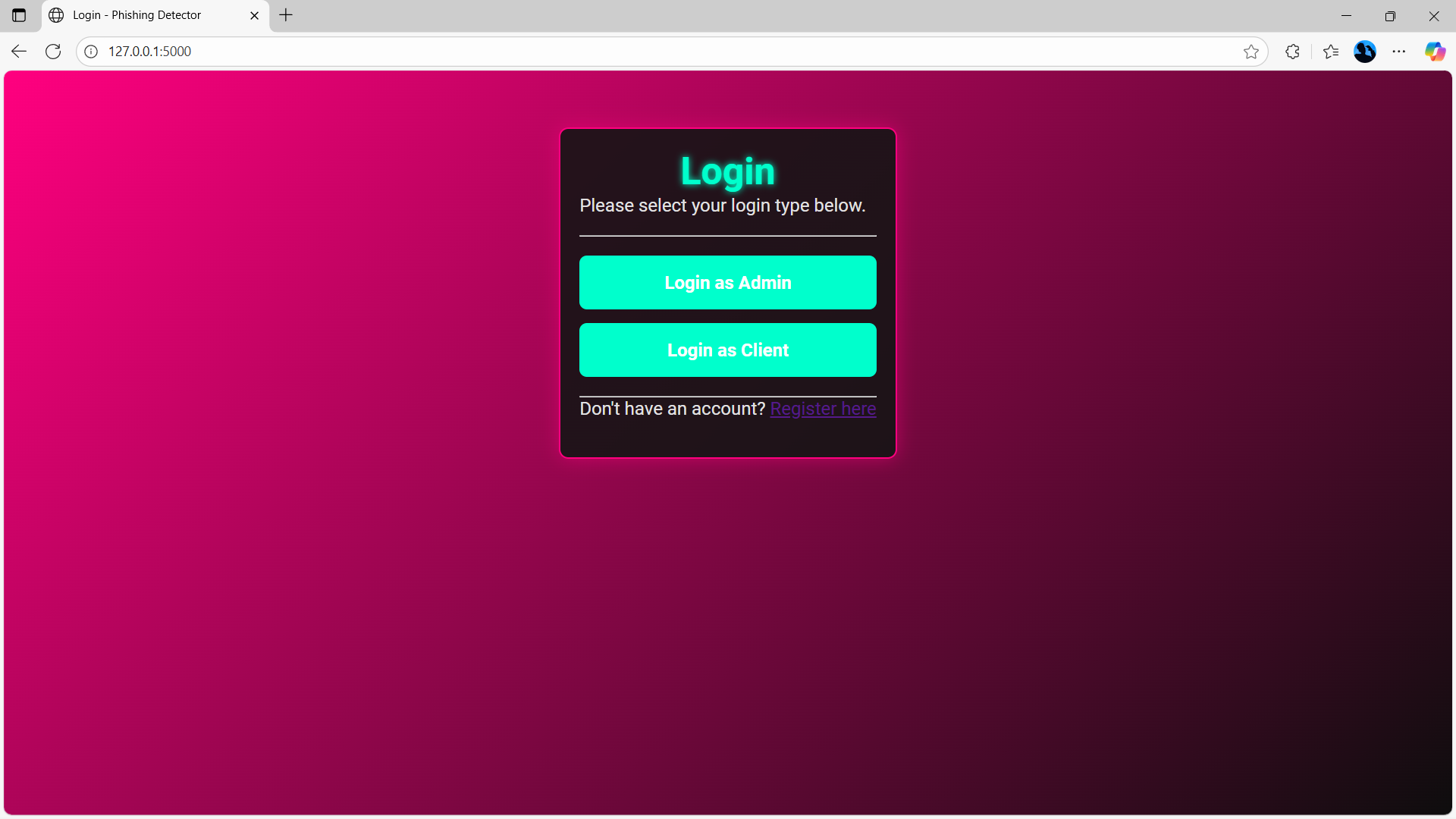
**static/script.js - fetch Call for URL Checks**

This is the core JavaScript snippet that sends a URL to the backend and handles the JSON response to update the page.

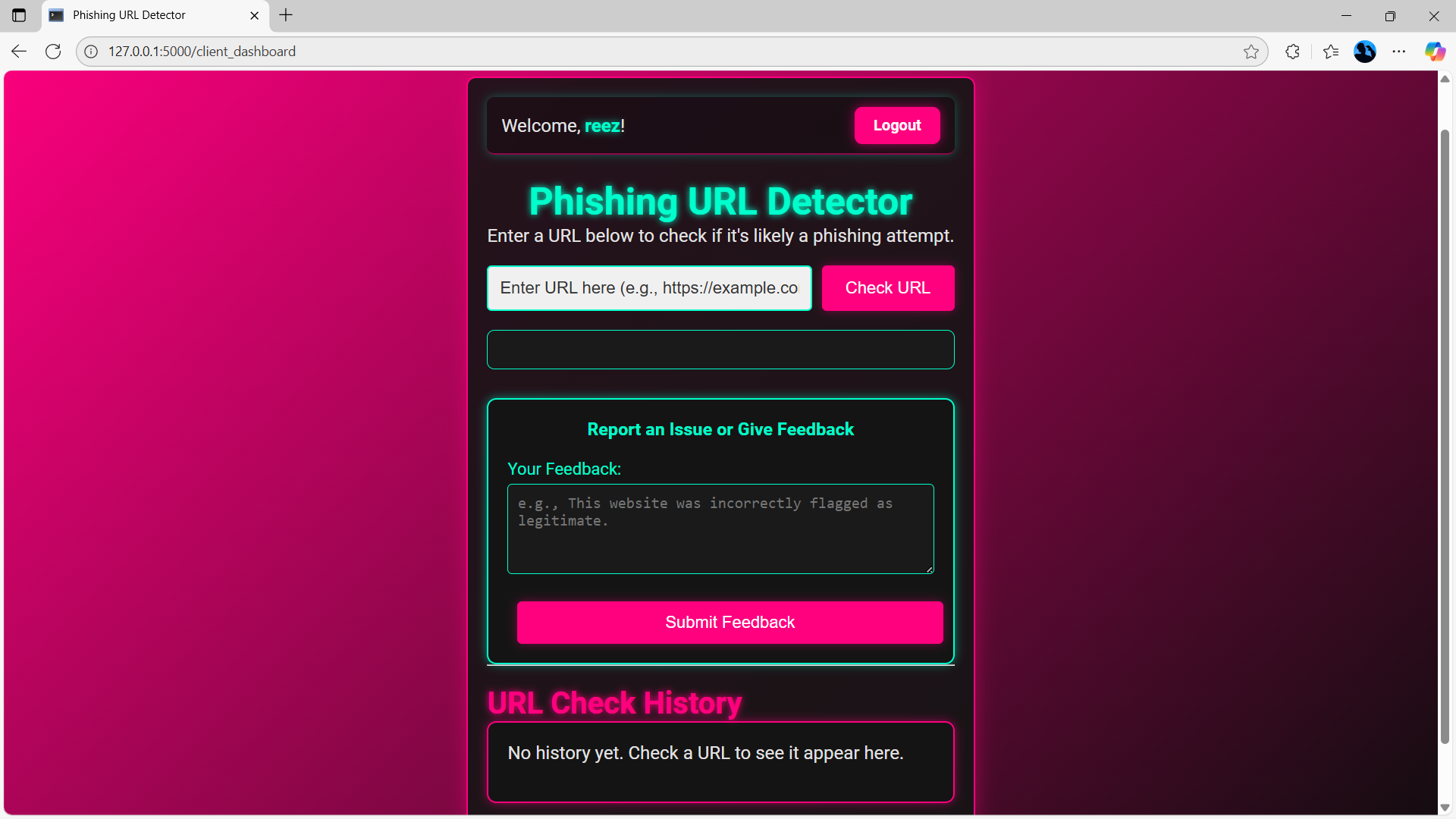
// --- PART 1: URL Check Submission ---  
const urlInput = document.getElementById('urlInput');  
const checkUrlBtn = document.getElementById('checkUrlBtn');  
const resultSection = document.getElementById('resultSection');  
  
if (checkUrlBtn) { // Check if the button exists on the page  
    checkUrlBtn.addEventListener('click', async () => {  
        const urlToCheck = urlInput.value.trim();  
        if (!urlToCheck) {  
            resultSection.innerHTML = '<p class="error">Please enter a URL.</p>';  
            return;  
        }  
        resultSection.innerHTML = '<p>Checking...</p>';  
        try {  
            const response = await fetch('/check\_url', {  
                method: 'POST',  
                headers: {  
                    'Content-Type': 'application/json',  
                },  
                body: JSON.stringify({ url: urlToCheck }),  
            });  
  
            if (!response.ok) {  
                const err = await response.json();  
                throw new Error(err.error || 'Server error');  
            }  
              
            const data = await response.json();  
            console.log('Detection result:', data);  
  
            const statusClass = data.is\_phishing ? 'phishing' : 'legitimate';  
            const statusText = data.is\_phishing ? 'Likely Phishing!' : 'Looks Legitimate!';  
  
            let resultHtml = `  
                <p><strong>URL:</strong> ${data.url}</p>  
                <p><strong>Status:</strong> <span class="${statusClass}">${statusText}</span></p>  
                <p><strong>Confidence:</strong> ${data.confidence}</p>  
                <p><strong>Heuristic Score:</strong> ${data.score}</p>  
                <p><strong>Reasons:</strong></p>  
                <ul>  
            `;  
            if (data.reasons && data.reasons.length > 0) {  
                data.reasons.forEach(reason => {  
                    resultHtml += `<li>${reason}</li>`;  
                });  
            } else {  
                resultHtml += `<li>No specific reasons provided.</li>`;  
            }

### **4.2 Screen Shots / UI Design of simulation**

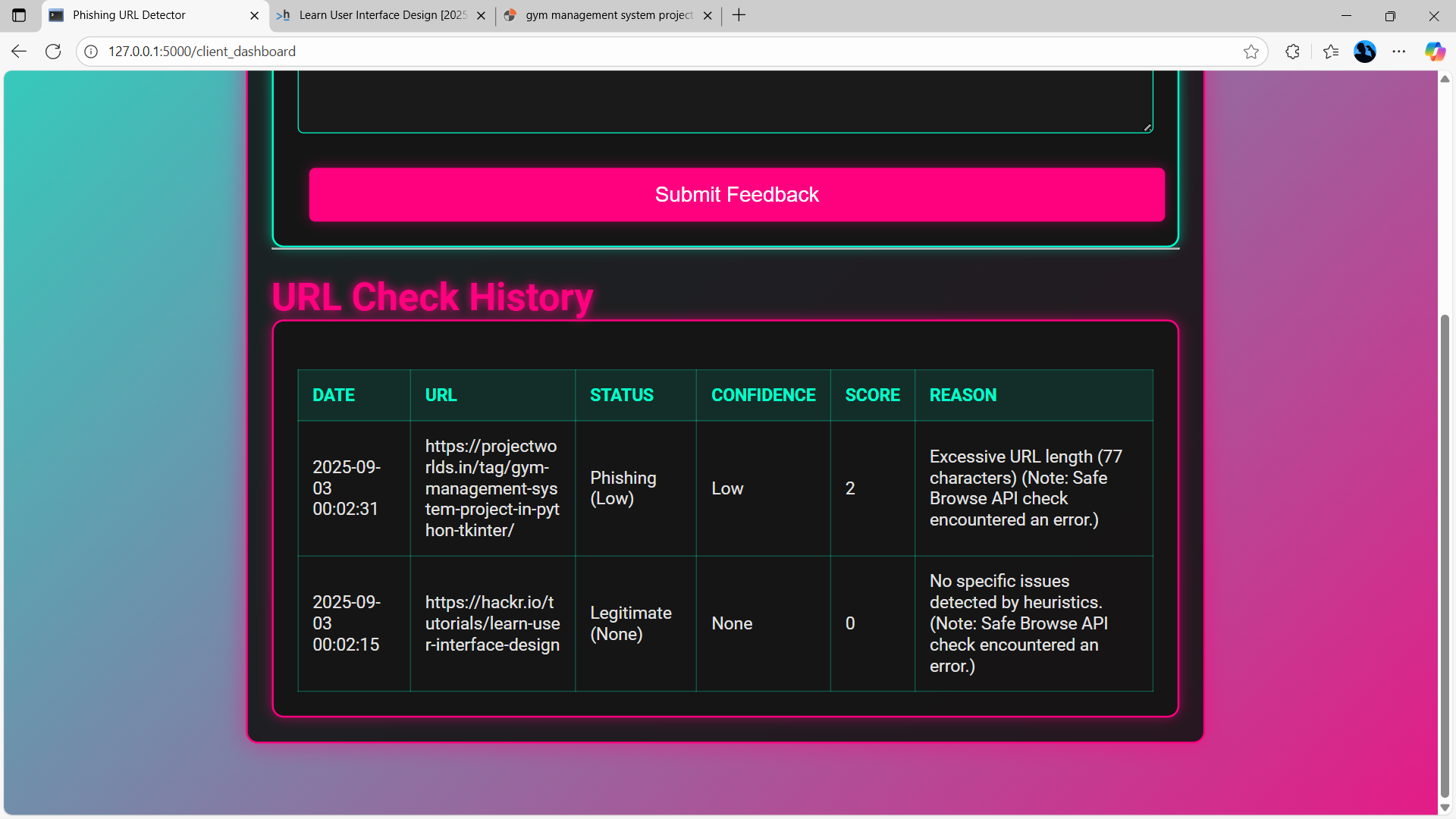
* **Login Page:**



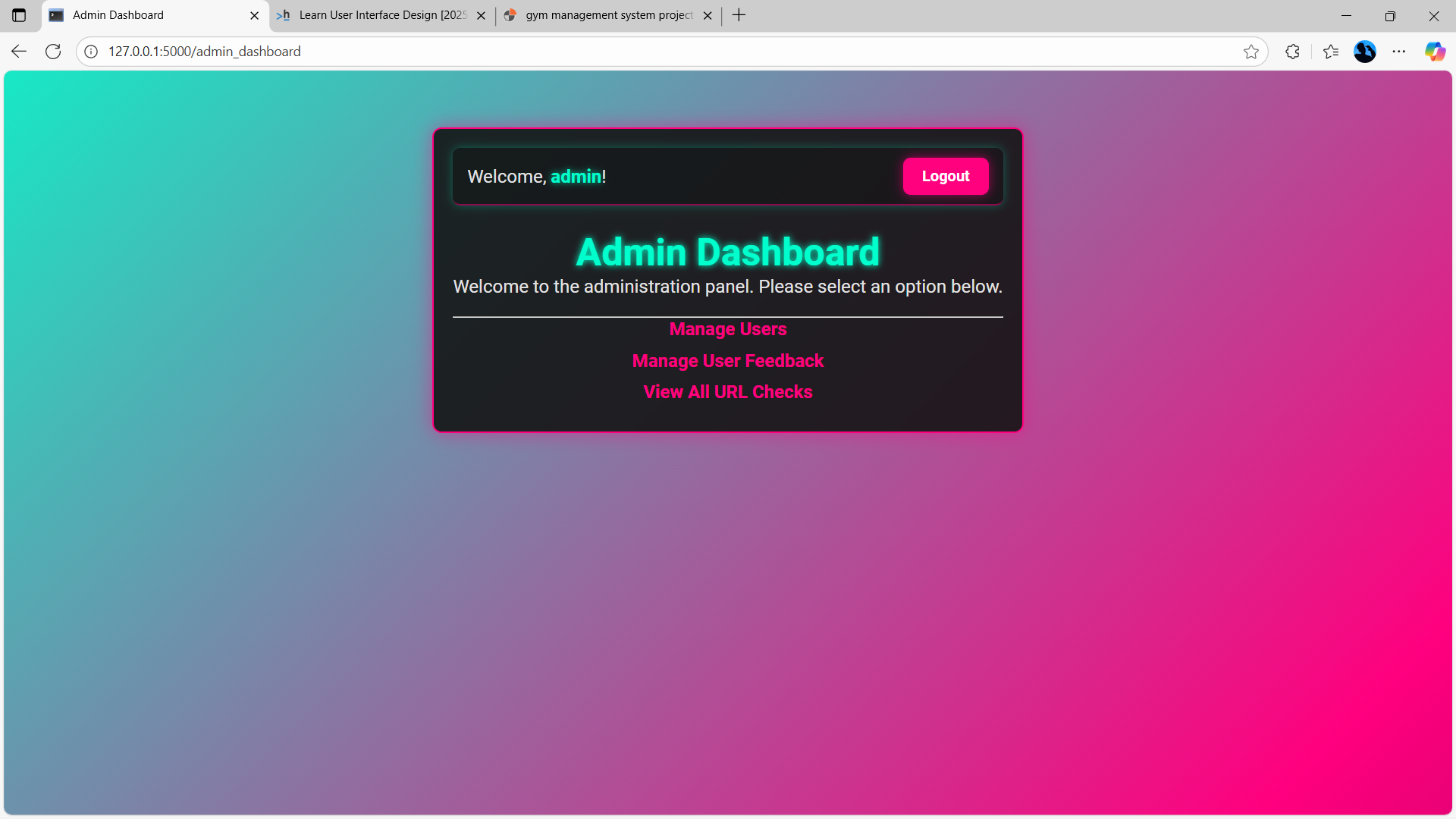
* **Client Dashboard:**



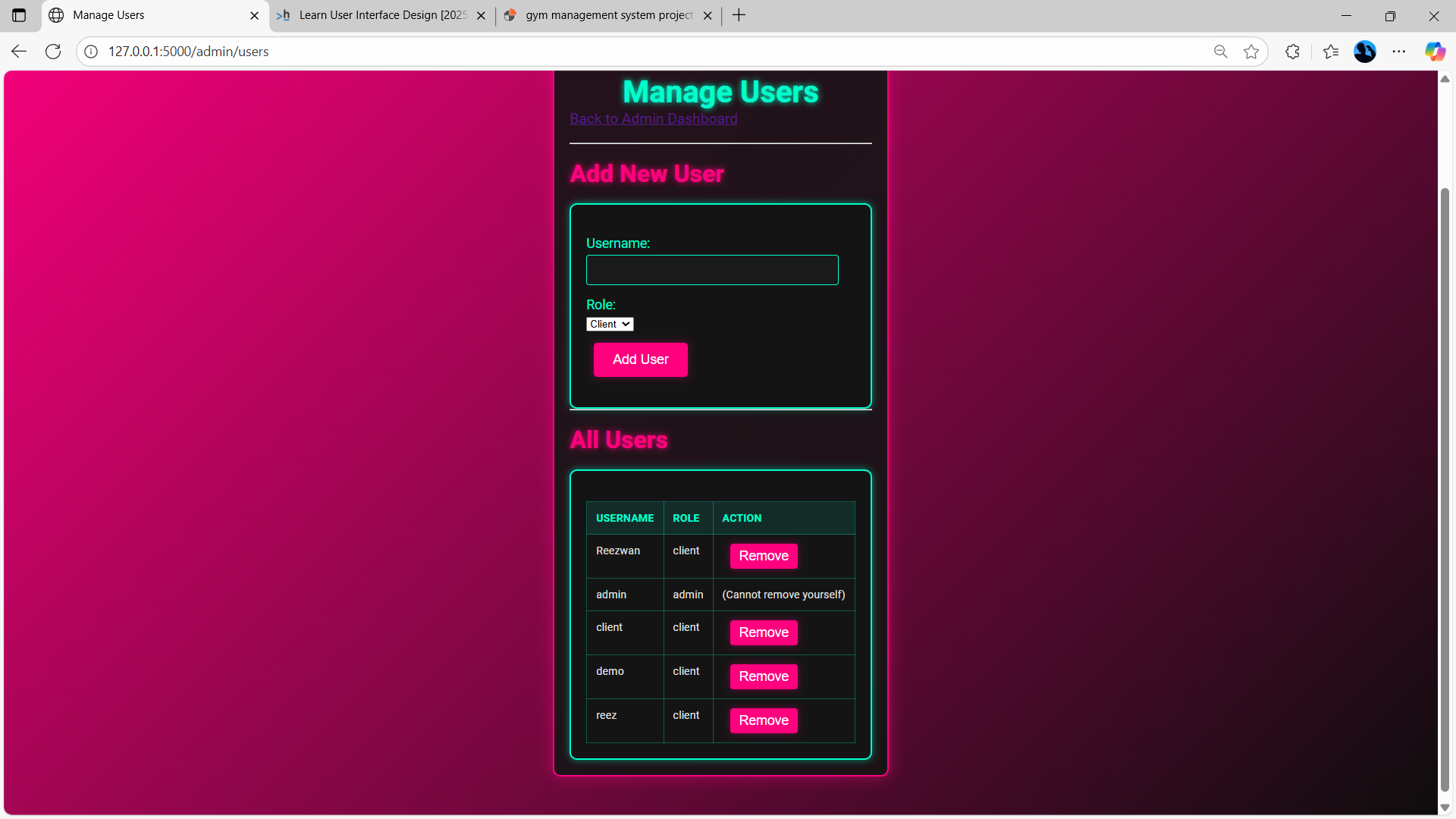
* **URL Check History:**



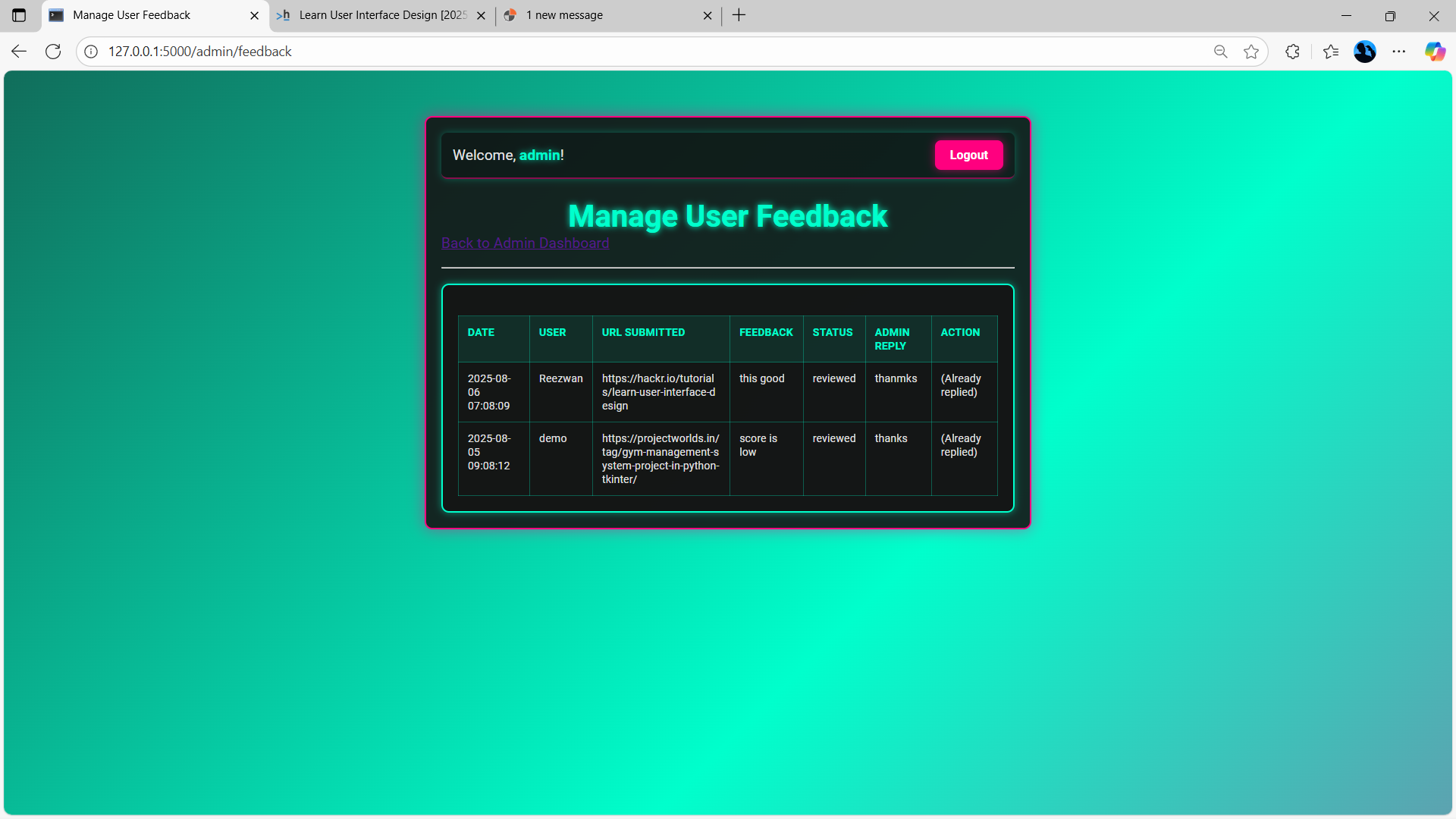
* **Admin Dashboard:**



* **User Management Page:**



* **Feedback Management Page:**



### **4.3 Test reports**

A simple test report demonstrating the system's ability to correctly classify different types of URLs.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case | URL | Expected Result | Actual Result | Status |
| 1 | http://192.168.1.100/login | Phishing (IP in Hostname) | Phishing (Low) | Pass |
| 2 | http://bit.ly/2K4o8z | Phishing (URL Shortener) | Phishing (Medium) | Pass |
| 3 | http://paypal.com-verify.com | Phishing (Mismatched Brand) | Phishing (High) | Pass |
| 4 | https://www.google.com | Legitimate | Legitimate (None) | Pass |
| 5 | http://valimail.com/phishing-guide-for-blog-post-of-examples | Legitimate | Legitimate (None) | Pass |

**Chapter 5 :**

**Proposed Enhancements**

This chapter outlines key enhancements to the Phishing URL Detector project, moving it from a foundational proof-of-concept to a more advanced and robust application. These enhancements represent a clear roadmap for future development, demonstrating an understanding of modern security and software engineering practices.

**1. Implement a Full-Featured Admin Reply System**

* **Email Notifications to Users:** The current system only saves the admin's reply to the database. A full-featured system would integrate with an email service (such as using a library like **Flask-Mail**). After an admin submits a reply, the system would automatically send an email notification to the user who submitted the feedback, creating a direct and professional communication channel. This would require configuring an SMTP server and creating email templates.
* **Enhanced Feedback Status Tracker:** The current status is a simple pending or reviewed. A more detailed system would introduce a lifecycle for each feedback entry. The status could be updated by the admin to include:
* pending: The feedback has been received but not yet reviewed.
* in-progress: An admin is actively investigating the reported URL.
* resolved: The issue has been addressed and a reply has been sent.
* This status would be visible on both the admin and client dashboards, providing transparency and demonstrating a more mature incident-handling process.

**2. Integrate a Live Threat Intelligence API**

While the project's internal heuristics are effective, they are a single layer of defense. A more robust solution involves a multi-layered approach by integrating with a live threat intelligence service.

* **Objective:** The goal of this enhancement is to successfully integrate and resolve any configuration challenges with an external API like the **Google Safe Browsing API**. This API maintains a massive, real-time database of known malicious URLs, providing an authoritative first layer of defense.
* **Two-Layered Detection:** A successfully integrated API would create a powerful hybrid system.

**Chapter 6:**

**Conclusion**

The Phishing URL Detector project has successfully delivered a functional, robust, and well-documented application that addresses a critical cybersecurity problem. The system's successful implementation of a multi-layered detection engine, secure user management, and an administrative oversight dashboard demonstrates a strong command of modern software development and security principles.

The project successfully met its core objectives by creating a tool that provides clear, transparent, and actionable threat analysis. Through the implementation of a heuristics-based scoring system, the application effectively identifies a wide range of phishing indicators and provides a corresponding confidence level and detailed reasoning. The secure user authentication system, with its distinct client and admin roles, establishes a solid foundation for controlled access and data integrity. Furthermore, the comprehensive logging system fulfills the crucial requirement of system monitoring, providing a timestamped record of all key application events.

Ultimately, this project is a comprehensive showcase of key skills and practices, including:

* **Full-Stack Development:** The seamless integration of a Python Flask backend with a dynamic HTML/CSS/JavaScript frontend.
* **Secure Coding:** The implementation of secure password hashing and role-based access control.
* **Database Design:** The structured use of an SQLite database with Flask-SQLAlchemy to manage complex data relationships between users, URL checks, and feedback.
* **System Monitoring:** The use of Python's logging module for centralized event tracking, which is fundamental in any professional security application.

# Chapter 7:

# Bibliography

1. **Flask Documentation:** The official documentation for the Flask micro-framework, serving as the primary reference for routing, Jinja2 templating, and configuration.
   1. **URL:** <https://flask.palletsprojects.com/>
2. **Flask-SQLAlchemy Documentation:** The authoritative guide for integrating SQLAlchemy with Flask, a library instrumental in simplifying database management and utilized for the User, URLCheck, and Feedback models.
   1. **URL:** <https://flask-sqlalchemy.palletsprojects.com/>
3. **Flask-Login Documentation:** The official documentation for Flask-Login, the extension responsible for user session management, including decorators such as @login\_required and the current\_user object.
   1. **URL:** <https://flask-login.readthedocs.io/>
4. **Python logging Module Documentation:** This resource elucidates the usage of Python's standard library for event logging, which was configured to generate the app.log file.
   1. **URL:** <https://docs.python.org/3/library/logging.html>
5. **requests Library Documentation:** The documentation for the requests library, recognized as the industry standard for executing HTTP requests in Python, a critical component for external API calls.
   1. **URL:** <https://docs.python-requests.org/en/latest/>
6. **Google Safe Browsing API Documentation:** The technical documentation for the external API, the integration of which was attempted. It delineates threat types and the structure of requests and responses.
   1. **URL:** <https://developers.google.com/safe-browsing>
7. **OWASP Top 10 Web Application Security Risks:** An authoritative compilation of the most significant security vulnerabilities affecting web applications. Referencing this list demonstrates an understanding of broader security principles, including secure authentication and input validation.
   1. **URL:** <https://owasp.org/www-project-top-ten/>
8. **Python-dotenv Documentation:** This documentation clarifies the application of this library for managing configuration and sensitive data (e.g., API keys) within a .env file, representing a contemporary and secure development practice.
   1. **URL:** <https://pypi.org/project/python-dotenv/>
9. **Werkzeug Security Documentation:** A foundational component of Flask providing secure utilities such as generate\_password\_hash and check\_password\_hash, which were employed for the secure storage of user passwords.
   1. **URL:** <https://tedboy.github.io/flask/generated/werkzeug.security.html>
10. **MDN Web Docs (for HTML, CSS, JavaScript):** The definitive resource for web development languages, encompassing the Fetch API, which is utilized by the JavaScript for real-time communication with the backend.
    1. **URL:** <https://developer.mozilla.org/en-US/>