A Micro project on

## PASSWORD SECURITY ANALYSIS

Submitted in partial fulfillment of the **Full Stack Web Development Lab GRIET Lab On Board (G-LOB)**

|  |  |
| --- | --- |
| **P. Koushik Reddy** | **22241A05W1** |
| **S.S. Goutham Reddy** | **22241A05X4** |
| **Sumedh Murakonda** | **22241A05X7** |
|  |  |
|  |  |
|  |  |

Under the esteemed guidance of

**Y. Lakshmi Prasanna**

**Assistant Professor**

**Department of Computer Science and Engineering GOKARAJU RANGARAJU**

**INSTITUTE OF ENGINEERING AND TECHNOLOGY**

**(Approved by AICTE, Autonomous under JNTUH) Bachupally, Kukatpally, Hyderabad-500090.**



**GOKARAJU RANGARAJU**

**INSTITUTE OF ENGINEERING AND TECHNOLOGY DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

This is to certify that the micro project titled “**PASSWORD SECURITY ANALYSIS**” is a bonafide work done by **P. Koushik Reddy (22241A05W1), S.S. Goutham Reddy (22241A05X4), Sumedh Murakonda (22241A05X7)** under Full Stack Web Development Lab- GRIET Lab On Board (G-LOB) practice of our institute and that this work has not been submitted for the award of any other Degree/Diploma of any Institution/University.

#### Project Guide

Y.Lakshmi Prasanna

Assistant Professor

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| Ch. No. | Chapter Name | Pg. No. |
| 1 | Introduction | 4 |
|  | 1.1. Need for the Project | 4 |
|  | 1.2. Project Description | 4 |
|  | 1.3 Components of the Projects | 5 |
| 2 | Requirement Analysis | 7 |
| 3 | System Design | 9 |
|  | 3.1. Architecture Diagram | 9 |
| 4 | Implementation | 10 |
| 5 | Results and Discussion | 14 |
| 6 | Conclusion | 19 |
| 7 | References | 19 |
|  |  |  |

# Introduction

### Need for the Project

In today's digital age, the increasing frequency and sophistication of data breaches and cyber-attacks have underscored the critical importance of robust password security. Users often fall into the habit of reusing passwords across multiple online platforms. This practice significantly elevates the risk of unauthorized access if even one site suffers a security compromise. A single exposed password can lead to a domino effect, granting cybercriminals access to multiple accounts belonging to the same user.

This project aims to address the pressing need for verifying the strength and security of passwords by checking if they have been exposed in known data breaches. By leveraging databases of compromised passwords, we can provide users with real-time feedback on the vulnerability of their passwords. This proactive approach not only helps in identifying weak and reused passwords but also promotes better password hygiene among users.

### Project Description

This project is a web application designed to help users determine if their passwords have been compromised in known data breaches. By integrating with the Have I Been Pwned (HIBP) API, the application provides a secure and efficient way to check the security of passwords. Users can input their passwords, which are then hashed and checked against a database of breached passwords. The application informs users if their passwords have been found in any data breaches and how many times they have appeared.

### Components of the Project

### The "Password Security Analysis" project comprises several key components that work together to provide a seamless user experience for checking password security. Below are the primary components of the project:

### HTML Structure:

**i**. **Document Structure**: Includes the title, meta tags for charset, and viewport settings.

**ii. Main Sections**:

**Navigation**: Contains links to different sections like the password check page.

**Main Content Area**: Divided into sections for user input (password field), result display, and buttons for submitting the form.

**Footer**: Includes links to additional resources and information about the project.

### CSS Styles:

* 1. **Global Styling**: Sets font families, colors, and spacing for consistent design across the site.
  2. **Navigation Styling**: Aligns links horizontally, adjusts padding and margins for spacing, and applies hover effects for better user interaction.
  3. **Form Styling**: Ensures banners are responsive, with appropriate image scaling and text overlay.
  4. **Result Styling**: Formats the result display area, aligning text and adjusting spacing for readability.

### JavaScript Functionality:

* 1. **Password Visibility Toggle**: Allows users to show or hide their password input.
  2. **Responsive Design**: Ensures elements resize and reflow appropriately on different screen sizes, optimizing layout and usability.

**4. Flask:**

* A lightweight web framework used to create the web application, handle routing, and manage HTTP requests and responses.

**5. Have I Been Pwned API:**

* An external service that provides a list of passwords that have been compromised in data breaches, used to check the security status of user passwords.

# Requirement Analysis:

### Functional Requirements:

* 1. **Password Checking Functionality:**

 Display a form where users can enter their password securely.

 Upon submission, retrieve and analyze data from the Have I Been Pwned API to determine if the password has been previously exposed in data breaches.

### 2. User Feedback:

i. Provide clear and immediate feedback to users based on the results of the password check.

ii. Display messages indicating the security status of the password

### Non-Functional Requirements:

1. **Usability:**
   * Design Create a clean and intuitive user interface for entering passwords and viewing results.
   * Error handling and validation should provide helpful feedback to users during product selection and checkout.

### Performance:

* + Optimize loading times for product listings and details to enhance user experience
  + Ensure the checkout process is efficient and responsive to prevent delays or timeouts.

### Security:

* + Implement secure HTTPS protocols for data transmission during checkout to protect user information.
  + Safeguard user data and payment details with robust security measures.

### Compatibility:

* + Support major web browsers and ensure compatibility with HTML5, CSS3, and JavaScript standards.
  + Test across different browsers and devices to ensure consistent functionality and appearance.

### Technical Requirements:

1. **Frontend Development:**
   * Utilize HTML, CSS, and JavaScript to build a responsive and interactive user interface.
   * Implement AJAX for seamless data updates and interactions without page reloads.

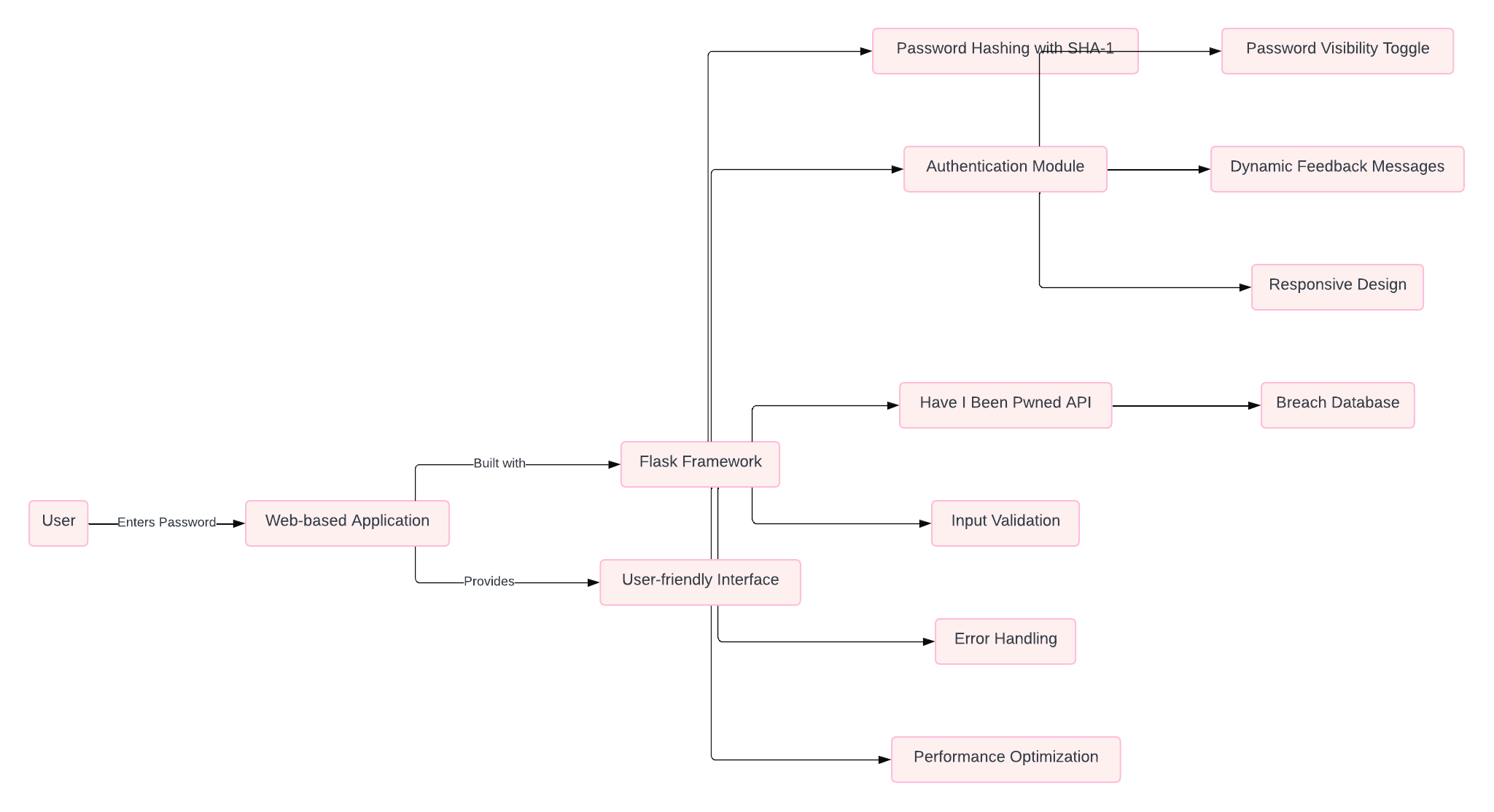
### Deployment and Testing:

* + Deploy the application on a reliable web hosting service with scalable resources.
  + Conduct thorough testing, including functionality testing, usability testing, and performance testing, before deployment.

### Maintenance and Support:

* + Plan for regular updates and maintenance to address security patches, bug fixes, and feature enhancements.
  + Provide customer support channels for addressing user inquiries and issues promptly.

**3. System Design**



**Fig: System Architecture**

The architecture diagram illustrates a system for the detecting the no. of times the given password is breached. User passwords are hashed for secure storage and hidden during typing. Input validation ensures strong passwords, while error handling guides users. An authentication module verifies credentials, and data breach checks via external APIs and internal databases add another layer of security. A user-friendly interface with dynamic feedback and responsive design prioritizes a smooth user experience, while performance optimization ensures fast loading times.

1. **Implementation**

**Server.py**

from flask import Flask, render\_template, request

import hashlib

import requests

app = Flask(\_\_name\_\_)

def request\_api\_data(query\_char):

    url = 'https://api.pwnedpasswords.com/range/' + query\_char

    res = requests.get(url)

    if res.status\_code != 200:

        raise RuntimeError(f'Error fetching: {res.status\_code}, check the api and try again')

    return res

def get\_password\_leaks\_count(hashes, hash\_to\_check):

    hashes = (line.split(':') for line in hashes.text.splitlines())

    for h, count in hashes:

        if h == hash\_to\_check:

            return count

    return 0

def pwned\_api\_check(password):

    sha1password = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()

    first5\_char, tail = sha1password[:5], sha1password[5:]

    response = request\_api\_data(first5\_char)

    return get\_password\_leaks\_count(response, tail)

@app.route('/', methods=['GET', 'POST'])

def index():

    if request.method == 'POST':

        password = request.form['password']

        count = pwned\_api\_check(password)

        if count:

            result = f'{password} was found {count} times... you should probably change your password!'

        else:

            result = f'{password} was NOT found. Carry on!'

        return render\_template('index.html', result=result)

    return render\_template('index.html')

if \_\_name\_\_ == '\_\_main\_\_':

    app.run(debug=True)

**Index.html**

<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <meta name="viewport" content="width=device-width, initial-scale=1.0">

    <title>Password Check</title>

    <style>

        body {

            font-family: 'Segoe UI', Tahoma, Geneva, Verdana, sans-serif;

            margin: 0;

            padding: 0;

            display: flex;

            justify-content: center;

            align-items: center;

            height: 100vh;

            background: linear-gradient(135deg, #71b7e6, #9b59b6);

            color: #333;

        }

        .container {

            background-color: white;

            padding: 30px;

            border-radius: 12px;

            box-shadow: 0 10px 20px rgba(0, 0, 0, 0.15);

            max-width: 400px;

            width: 100%;

            text-align: center;

        }

        .container h1 {

            margin-bottom: 20px;

            font-size: 24px;

            color: #4a4a4a;

        }

        .form-group {

            margin-bottom: 20px;

        }

        .form-group label {

            display: block;

            margin-bottom: 8px;

            font-size: 16px;

            color: #4a4a4a;

        }

        .form-group input[type="password"],

        .form-group input[type="text"] {

            width: 100%;

            padding: 12px;

            border: 1px solid #ccc;

            border-radius: 8px;

            box-sizing: border-box;

            transition: border 0.3s ease;

        }

        .form-group input[type="password"]:focus,

        .form-group input[type="text"]:focus {

            border-color: #007BFF;

            outline: none;

        }

        .form-group input[type="checkbox"] {

            margin-top: 10px;

        }

        .btn {

            background-color: #007BFF;

            color: white;

            padding: 12px 20px;

            border: none;

            border-radius: 8px;

            cursor: pointer;

            font-size: 16px;

            transition: background-color 0.3s ease;

        }

        .btn:hover {

            background-color: #0056b3;

        }

        .result {

            margin-top: 20px;

            font-size: 18px;

            color: #333;

        }

    </style>

</head>

<body>

    <div class="container">

        <h1>Password Check</h1>

        <form method="POST">

            <div class="form-group">

                <label for="password">Password</label>

                <input type="password" id="password" name="password" required>

                <div style="text-align: left; margin-top: 10px;">

                    <input type="checkbox" onclick="togglePassword()"> Show Password

                </div>

            </div>

            <button type="submit" class="btn">Check</button>

        </form>

        {% if result %}

        <div class="result">

            {{ result }}

        </div>

        {% endif %}

    </div>

    <script>

        function togglePassword() {

            var passwordField = document.getElementById("password");

            if (passwordField.type === "password") {

                passwordField.type = "text";

            } else {

                passwordField.type = "password";

            }

        }

    </script>

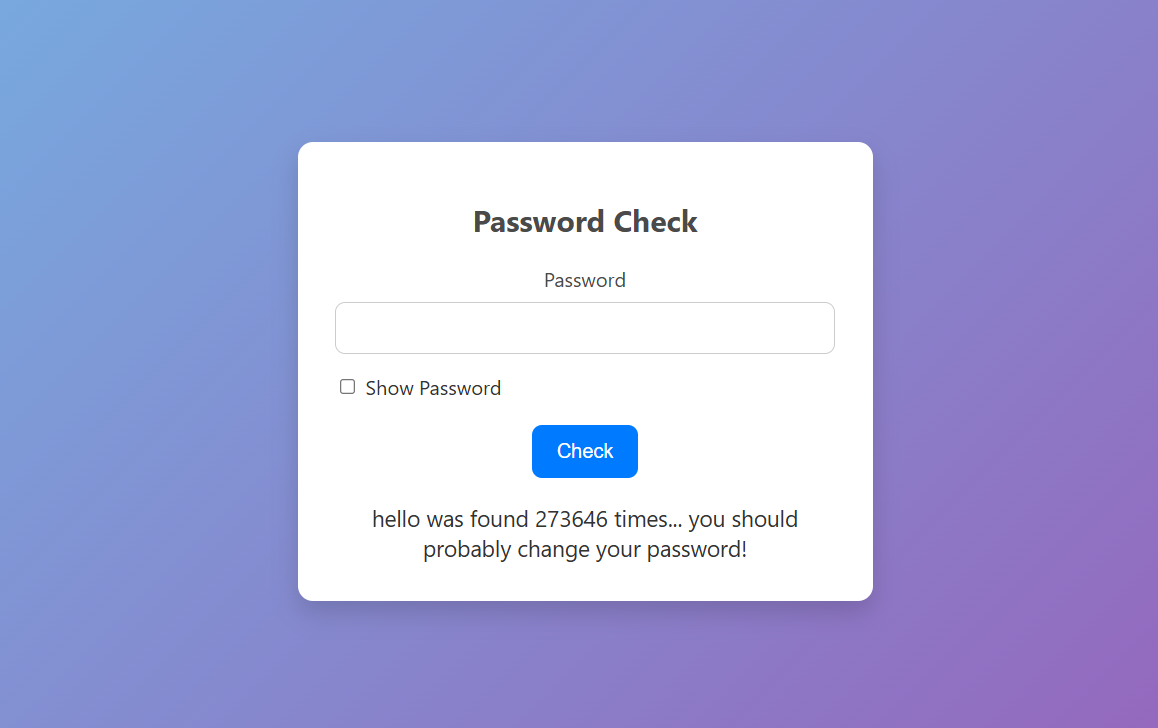
</body>

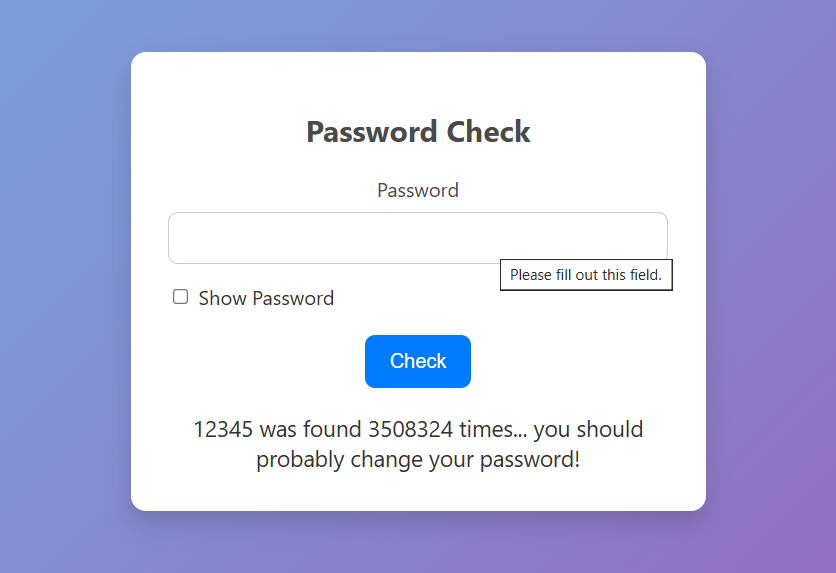
</html>

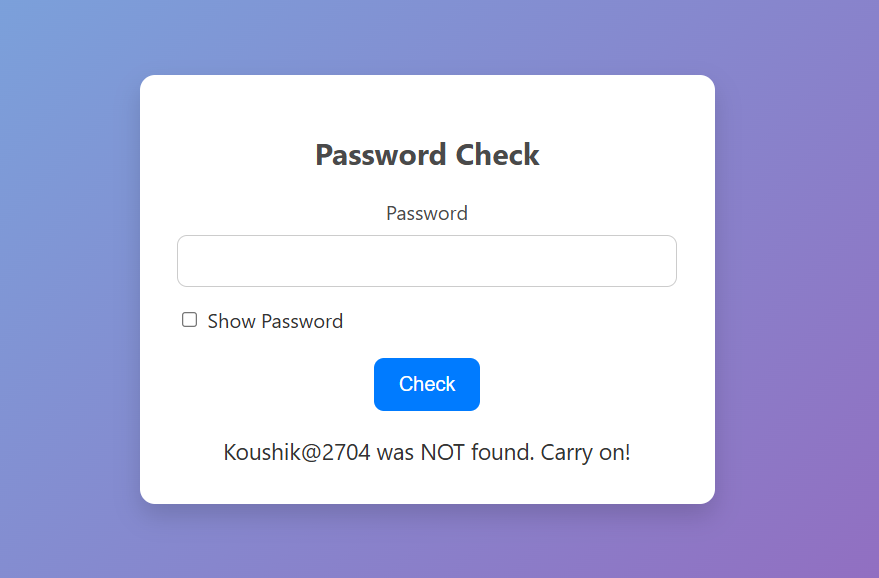
## Result and Discussion

**Result:**

****





****

**Discussion:`**

### Strengths:

* 1. **Clear Structure:**

i. The HTML is well-structured with a clear separation between the form elements and the script for toggling password visibility.

ii. The Flask application code is organized, separating API interaction logic from the route handling.

### Basic Functionality:

i. Users can input a password and check if it has been compromised using the Have I Been Pwned API.

ii. The application provides feedback on whether the password was found in any data breaches.

### Interactive Elements:

i. The password visibility toggle enhances user interaction, making it easier to verify the entered password.

ii. The button provides a clear call-to-action for users to check their password.

### Responsive Design:

i. The use of flexbox and media queries ensures the form is centered and accessible on various devices and screen sizes.

### Security Considerations:

The password is hashed using SHA-1 before being sent to the API, protecting user input.

### Potential Improvements:

1. **Input Validation:**

i. Ensure the password field is not empty before submitting the form to prevent unnecessary API calls.

### Error Handling:

i. Enhance error messages and notifications to provide clear guidance when API requests fail or if the password format is invalid.

ii. Handle edge cases such as network errors or API rate limiting gracefully.

### User Feedback:

i. Implement more detailed feedback messages, especially when a password is found in breaches. Consider showing the exact count in a more user-friendly format.

### Accessibility:

i. Introduce ARIA labels and roles to enhance accessibility for users relying on assistive technologies.

ii. Ensure keyboard navigation is smooth and intuitive across all interactive elements, including form fields and buttons.

### Data Persistence:

i. Optimize the Flask application and API calls to ensure quick response times.

ii. Minimize HTTP requests and leverage caching mechanisms for frequently accessed data.

### Performance:

i. Optimize the Flask application and API calls to ensure quick response times.

ii. Minimize HTTP requests and leverage caching mechanisms

### 7. Performance:

i. Implement HTTPS protocol site-wide to protect user data during transmission.

ii. Sanitize inputs and validate data to mitigate risks of injection and cross-site scripting (XSS) attacks.

### Considerations:

1. **Browser Compatibility:**

Ensure compatibility across major web browsers (Chrome, Firefox, Safari, Edge) and validate functionality on both desktop and mobile platforms.

### Performance Scaling:

i. Plan for scalability by optimizing server-side scripts to handle increased traffic and data volume during peak periods.

ii. Monitor and analyze performance metrics regularly to identify and address potential bottlenecks proactively.

### Future Extensions:

i. Explore additional features such as user authentication and account management to enable personalized experiences.

ii. Consider integrating features like password strength meters, tips for creating strong passwords, and a history of previously checked passwords for better user engagement.

## Conclusion

The “Password Security Analysis” project aims to improve online security by enabling users to verify the safety of their passwords. By integrating the Have I Been Pwned API, the project offers a user-friendly interface for checking password integrity. It is noteworthy for its clear structure, basic functionality, interactive elements, responsive design, and attention to security.

Nevertheless, there are opportunities for improvement, including input validation, error handling, user feedback, accessibility, data persistence, performance optimization, and enhanced security measures. Addressing these areas will refine the application and ensure a robust and seamless user experience.

## Reference

* 1. Flask Documentation:

Flask, the web framework used for this project, is well-documented and provides comprehensive guides and tutorials. <https://flask.palletsprojects.com/en/2.0.x/deploying/>

* 1. Have I Been Pwned API:

The project utilizes the Have I Been Pwned API to check if passwords have been compromised in data breaches. <https://haveibeenpwned.com/API/v3>

* 1. **JavaScript MDN Web Docs**: Reference for JavaScript language features and APIs. Available at [developer.mozilla.org/en-US/docs/Web/JavaScript](https://developer.mozilla.org/en-US/docs/Web/JavaScript).
  2. **HTML5 & CSS3**: W3Schools reference for HTML5 and CSS3 features and best practices. Available at [w3schools.com](https://www.w3schools.com/).