LOCKZILLA – The Ultimate Password Protector

A

Mini Project Report

Submitted in partial fulfilment of the Requirements for the award of the Degree of

BACHELOR OF ENGINEERING

IN

INFORMATION TECHNOLOGY

By

T. SUHAAS RAO : 1602-23-737-181
B. SAI SHASHANK : 1602-23-737-165
S. SINDHU: 1602-23-737-317

Department of Information Technology

Vasavi College of Engineering(Autonomous)

ACCREDITED BY NAAC WITH 'A++'GRADE

(Affiliated to Osmania University and Approved by AICTE)

Ibrahimbagh,Hyderabad-31

2024

Vasavi College of Engineering (Autonomous)

ACCREDITED BY NAAC WITH 'A++' GRADE

(Affiliated to Osmania University and Approved by AICTE)Hyderabad-500031

Department of Information Technology

DECLARATION BY THE CANDIDATE

We T. SUHAAS RAO, B. SAI SHASHANK and S. SINDHU Bearing hall ticket numbers, **1602-23-737-181, 1602-23-737-165** and**1602-23-737-317,** hereby declare that the project report entitled **” LOCZILLA – Ultimate Password Protector” is** submitted in partial fulfilment of the requirement for the award of the degree of **Bachelor of Engineering** in **Information Technology**

This is a record of Bonafide work carried out by us and the results embodied in this project report have not been submitted to any other university or institute for the award of any other degree or diploma.

T. SUHAAS RAO : 1602-23-737-181
B. SAI SHASHANK: 1602-23-737-165
S. SINDHU: 1602-23-737-317

(FacultyIn-Charge) (External Examiner) (Head,DeptofIT)

ACKNOWLEDGMENT

We would like to extend our sincere thanks and appreciation to the following individuals who played an integral role in the successful completion of our Mini Project.

Firstly, we express our deepest gratitude to **Dr. S. V. Ramana**, Principal of Vasavi College of Engineering, for his constant encouragement and support throughout the duration of this project. His vision and commitment to excellence in education have inspired us to strive for success.

We are also extremely grateful to **Dr. K. Ram Mohan Rao**, Professor and Head, Department of Information Technology, Vasavi College of Engineering, for introducing the Mini-Project module as a part of our curriculum. His invaluable suggestions, motivation, and constant cooperation provided us with the necessary guidance and confidence to carry out this project successfully.

Our heartfelt thanks also go to our mini-project coordinators, **Dr. S.K. Prashanth**, Department of Information Technology, and **Mrs. D.R.L. Prasanna**, Department of Information Technology, for their guidance and expertise. Their mentorship throughout the project development process helped us understand the practical aspects of project implementation and made it possible for us to complete the project on time. Their timely suggestions and constructive feedback at every phase of the project were essential in ensuring its quality and relevance.

Finally, we would like to sincerely thank the project reviewers for their valuable inputs and suggestions. Their thoughtful feedback helped us refine the project and improve its scope and impact.

Abstract

Lockzilla is a sophisticated and secure password manager designed to help users efficiently store, manage, and protect their passwords. In today's digital age, maintaining strong and unique passwords for multiple online accounts is crucial for cybersecurity. However, many users struggle with remembering complex passwords, which often leads to poor password practices. Existing password managers can sometimes lack comprehensive security features or fail to offer an intuitive user experience. Lockzilla addresses these challenges by providing a robust, user-friendly solution that allows users to easily add, update, and delete passwords, while ensuring their security through advanced encryption and management features.

Lockzilla's key functionalities include the ability to securely add, update, and delete passwords for various accounts. With a simple and intuitive interface, users can quickly store their credentials while ensuring that sensitive information is encrypted. The platform is designed to support multiple users, making it ideal for teams or families who need to share passwords securely. The backend of Lockzilla is powered by Flask, a lightweight web framework that handles user authentication, session management, and secure interactions. SQLAlchemy is used to manage the SQLite3 database, storing user information, encrypted passwords, and other necessary data efficiently.

To further enhance security, Lockzilla leverages Werkzeug, a security library for Flask, to safeguard user data with various cryptographic techniques. Password health checks are integrated into the system using the "pwned" - API, providing users with insights into the strength of their passwords and recommending improvements where necessary. Additionally, Lockzilla features a password generator tool that creates strong, random passwords, reducing the risk of weak or reused passwords. The platform uses SHA encryption to protect all sensitive information, ensuring that passwords are never stored in plaintext. Users can also take advantage of the "copy to clipboard" functionality, making it easy to transfer passwords securely without exposing them to unauthorized parties.

With Lockzilla, users can enjoy the convenience of managing their passwords securely and effortlessly, with peace of mind knowing that their sensitive information is protected by industry-standard security measures.

Whether you're a single user or part of a team, Lockzilla offers a personalized, scalable solution to meet your password management needs.

Table Of Contents

Abstract & Introduction

Information about the Project Domain

Lockzilla: Secure Password Manager

Lockzilla is a secure password manager designed to help users store, manage, and protect their sensitive credentials with ease. In today's digital landscape, where online accounts are an essential part of daily life, users often struggle to remember complex passwords for multiple platforms. Many existing password managers offer basic password storage and retrieval features but lack advanced security measures and user-friendly functionality to fully address users' needs. These systems typically offer limited options for password management, often neglecting crucial features like password strength analysis, password generation, and multi-user support.

Lockzilla addresses these challenges by offering a comprehensive, secure, and user-centric solution for password management. Lockzilla's advanced security features ensure that passwords are always protected through robust encryption methods, such as SHA encryption, and never stored in plaintext. Users can also generate strong, random passwords directly within the system, reducing the risk of weak or reused passwords. Furthermore, Lockzilla enables password health checks, providing valuable insights into the strength of passwords and offering recommendations for improvement where needed.

The underlying architecture of Lockzilla is built using Flask, a lightweight and flexible web framework that handles user authentication, secure sessions, and interaction with the database. SQLAlchemy is used to manage the SQLite3 database, securely storing user credentials, encrypted passwords, and associated metadata. To protect sensitive data, Lockzilla leverages Werkzeug's security features, which provide a suite of cryptographic techniques to safeguard user information. The platform also supports multiple users, making it an ideal solution for families or teams who need to share passwords securely.

In addition to password management, Lockzilla includes a "copy to clipboard" feature, allowing users to securely copy passwords to their clipboard for easy use without exposing them to unauthorized access. This ensures seamless and secure password management across various devices. With Lockzilla, users can confidently manage their digital security, knowing that their passwords are protected by the latest encryption techniques and that they have complete control over their password health and security.

Lockzilla's intuitive interface and powerful security features make it the ideal tool for anyone looking to take control of their digital identity. Whether for individual use or as a collaborative solution for teams, Lockzilla ensures that passwords are stored and managed in a secure, efficient, and user-friendly manner. As the platform evolves, it continues to improve its security features and functionality, adapting to the ever-changing landscape of cybersecurity and user needs.

Technology

a. Software Requirements

Lockzilla is developed using a combination of reliable tools, libraries, and frameworks designed to ensure secure password management, seamless user interaction, and effective encryption. Below are the software requirements for the Lockzilla project:

Operating System:

- Windows, macOS, or Linux.

Programming Languages:

- **Python**: Used for server-side logic, handling user requests, encryption, and database integration. Python ensures the functionality and security of the system.
- **HTML/CSS**: For designing the front-end interface, creating a clean, responsive, and user-friendly experience for users.

Frameworks and Libraries:

- **Flask**: A lightweight and flexible Python web framework used to handle the routing, server-side logic, and user authentication. It serves as the foundation for the web application.
- **Werkzeug**: A comprehensive security library for Flask that offers secure hashing and encryption features, ensuring data safety and protecting sensitive information.
- **SQLAlchemy**: An Object Relational Mapper (ORM) that facilitates interaction with the SQLite3 database, making it easier to manage and store encrypted passwords and other user data.
- **Cryptography**: A Python library used for secure encryption and hashing, such as SHA encryption, to protect stored passwords and enhance overall security.

Database:

- **SQLite3**: A lightweight, serverless, relational database management system that stores user credentials, encrypted passwords, password health data, and other relevant information securely.

Web Technologies:

- **HTML/CSS**: For the structure and layout of the web application interface, ensuring a responsive and aesthetically pleasing design.
- **JavaScript (optional)**: Used for enhancing client-side interactivity, such as handling password visibility toggles or validating password strength in real time.

Development Tools:

- **Visual Studio Code or any text editor of choice**: For writing and editing code, offering features like code suggestions, version control, and debugging.
- **Git**: For version control, collaboration, and project management, allowing easy tracking of changes and coordination among team members.
- **Browser**: For testing the application in a live environment, debugging, and ensuring cross-browser compatibility during deployment.

These software requirements provide the necessary tools to build a secure, user-friendly, and efficient password management system, ensuring that Lockzilla meets the highest standards of cybersecurity while offering a seamless experience for users.

b. Hardware Requirements

The hardware requirements for running and deploying Lockzilla are modest, as the application is designed to be lightweight and secure. Below are the necessary hardware specifications:

Processor:

- A multi-core processor (Intel i3 or higher) is recommended for handling server-side operations and encryption tasks efficiently.

Memory (RAM):

- A minimum of 4GB of RAM is recommended for smooth development, password encryption, and user interaction. More RAM may be required if handling large amounts of user data or concurrent requests.

Storage:

- **Disk Space**: At least 5GB of available disk space is recommended for storing the Lockzilla application, database, encryption files, and dependencies.
- **Database Storage**: Since Lockzilla uses SQLite3, it requires minimal storage for encrypted password data. However, the storage size will grow depending on the number of users, stored passwords, and password health data.

Network:

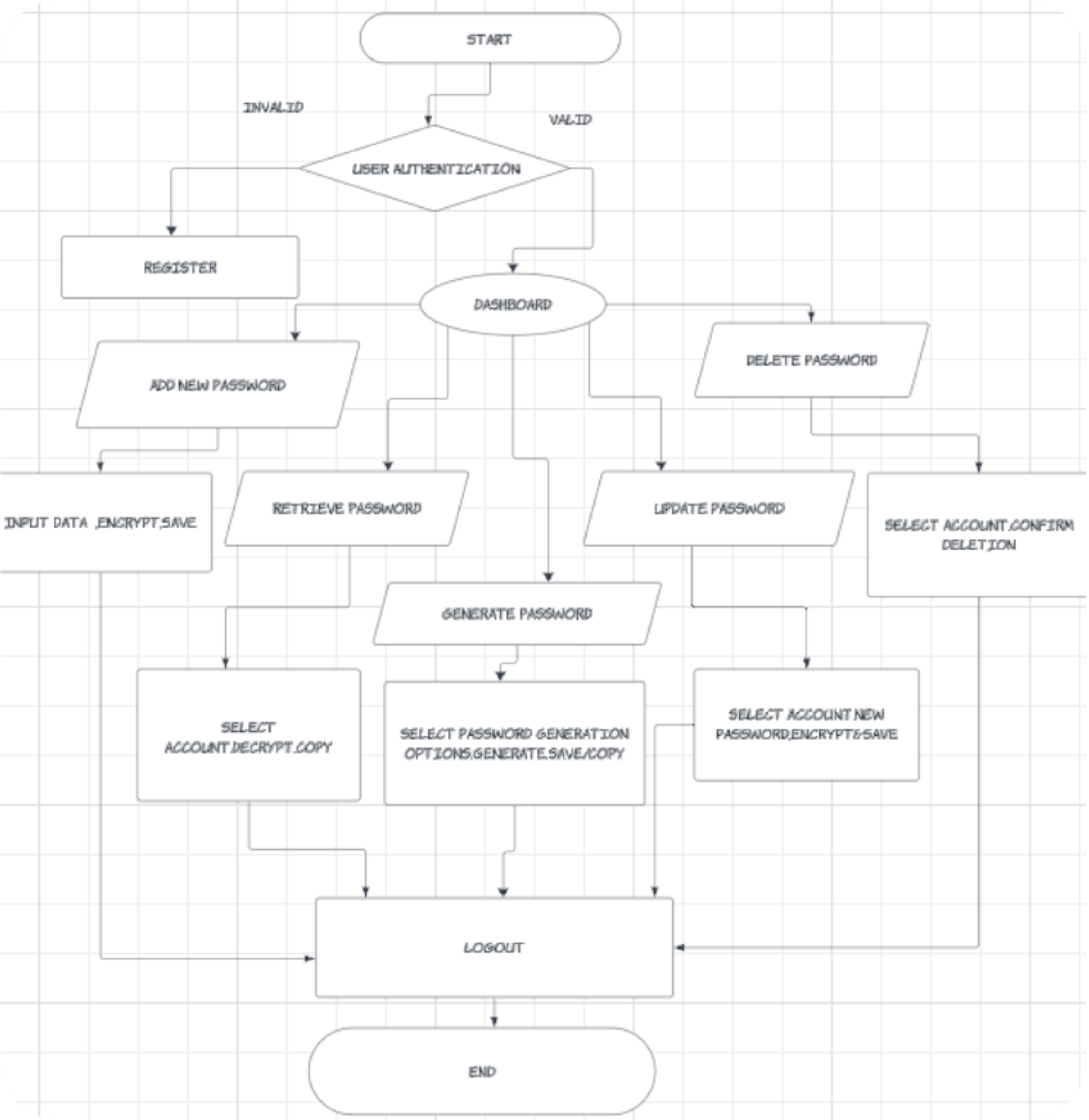
- A stable internet connection is essential for package installations, secure user authentication, and deploying the web application to cloud hosting services (if required).

Optional:

- **Web Hosting/Server**: If deploying the application for public or team use, access to a cloud hosting service (e.g., Heroku, AWS, or Google Cloud) or a dedicated server will be necessary for live deployment.
- **Browser**: Any modern web browser (Chrome, Firefox, Safari, etc.) is required for testing the application and interacting with the user interface.

These hardware requirements ensure that Lockzilla runs efficiently, offering users a secure and responsive password management solution with minimal resource usage. The system can be deployed on various platforms, making it suitable for both development environments and live production use.

Architecture & Activity Diagram:



i. Architecture

The Lockzilla architecture follows a modular design, focusing on security, ease of use, and scalability. Below is the high-level flow of data and processes:

User Interaction Layer:

- Users interact with the Lockzilla web interface to register, log in, and manage their passwords.
- Users can securely add, update, and delete passwords. They can also generate strong passwords and run password health checks.
- Inputs such as passwords and preferences are validated before being securely stored in the database.

Data Management Layer:

- An SQLite3 database is used to store user information, encrypted passwords, password health data, and preferences.
- SQLAlchemy ORM is used for smooth interaction with the database, ensuring data integrity and ease of management.

Security Layer:

- **Encryption**: Passwords are securely encrypted using SHA encryption, ensuring they are never stored in plaintext.
- **Password Health Check**: The system checks the strength of passwords and offers suggestions for improvement.
- **Password Generation**: Lockzilla provides a tool for generating strong, random passwords to enhance security.

Backend Logic Layer:

- The Flask framework powers the backend, handling user requests such as adding, updating, deleting passwords, and managing sessions securely.
- Werkzeug security features are used to ensure proper encryption and hashing of passwords.

Output Generation Layer:

- The Flask backend fetches encrypted passwords and password health information and sends them to the front-end.
- Results, including password strength reports and other relevant details, are displayed in a clean, user-friendly interface.

ii. Activity Diagram

Below is the workflow represented as an activity diagram:

- **Start**
- **User Registration** → Store User Data (Database)
- **User Login** → Session Management
- **Add/Update/Delete Password** → Store Encrypted Password (Database)
- **Generate Password** → Create Strong Password
- **Run Password Health Check** → Analyze Password Strength
- **Display Recommendations/Results** → End

IMPLEMENTATION

Code

PYTHON

```
from flask import Flask, render_template, request, redirect, url_for, session, flash, jsonify
from werkzeug.security import generate_password_hash, check_password_hash
import sqlite3
import hashlib
import requests

# Database Manager for handling SQLite operations
class DatabaseManager:
  @staticmethod
  def get_db_connection():
    conn = sqlite3.connect('lockzilla.db', timeout=10)  # Timeout to prevent immediate lock errors
    conn.row_factory = sqlite3.Row
    conn.execute('PRAGMA journal_mode = WAL')  # Enable Write-Ahead Logging (WAL) for
concurrency
    return conn

  @staticmethod
  def get_user_by_username(username):
    with DatabaseManager.get_db_connection() as conn:
      user = conn.execute('SELECT * FROM users WHERE username = ?', (username,)).fetchone()
    return user

  @staticmethod
  def insert_user(username, password, email):
    with DatabaseManager.get_db_connection() as conn:
      conn.execute('INSERT INTO users (username, password, email) VALUES (?, ?, ?)',
             (username, generate_password_hash(password), email))
      conn.commit()

  @staticmethod
  def validate_user(username, password):
    user = DatabaseManager.get_user_by_username(username)
    if user and check_password_hash(user['password'], password):
      return user
    return None

  @staticmethod
  def get_passwords(user_id):
    with DatabaseManager.get_db_connection() as conn:
      passwords = conn.execute('SELECT * FROM passwords WHERE user_id = ?',
(user_id,)).fetchall()
    return passwords
    @staticmethod
    def insert_password(user_id, service, password):
        with DatabaseManager.get_db_connection() as conn:
            conn.execute('INSERT INTO passwords (user_id, service, password) VALUES (?, ?, ?)',
                         (user_id, service, password))
            conn.commit()

    @staticmethod
    def update_password(user_id, service, password):
        with DatabaseManager.get_db_connection() as conn:
            conn.execute('UPDATE passwords SET password = ? WHERE user_id = ? AND service = ?',
                         (password, user_id, service))
            conn.commit()

    @staticmethod
    def delete_password(user_id, service):
        with DatabaseManager.get_db_connection() as conn:
            conn.execute('DELETE FROM passwords WHERE user_id = ? AND service = ?',
                         (user_id, service))
            conn.commit()

# Flask app to handle routes and logic
class AppRoutes:
    def __init__(self):
        self.app = Flask(__name__)
        self.app.secret_key = 'your_secret_key'  # Secret key for session management
        self.setup_routes()

    def setup_routes(self):
        self.app.add_url_rule('/', 'index', self.index)
        self.app.add_url_rule('/login', 'login', self.login, methods=['GET', 'POST'])
        self.app.add_url_rule('/logout', 'logout', self.logout)
        self.app.add_url_rule('/add', 'add_password', self.add_password, methods=['GET', 'POST'])
        self.app.add_url_rule('/update/<service>', 'update_password', self.update_password,
methods=['GET', 'POST'])
        self.app.add_url_rule('/delete/<service>', 'delete_password', self.delete_password,
methods=['POST'])
        self.app.add_url_rule('/register', 'register', self.register, methods=['GET', 'POST'])
        self.app.add_url_rule('/get_password', 'get_password', self.get_password, methods=['GET'])

    def check_password_breach(self, password):
        sha1_hash = hashlib.sha1(password.encode('utf-8')).hexdigest().upper()
        prefix = sha1_hash[:5]
        suffix = sha1_hash[5:]
        url = f"https://api.pwnedpasswords.com/range/{prefix}"
        response = requests.get(url)

        if response.status_code == 200:
```

```
        hashes = response.text.splitlines()
        for hash_entry in hashes:
            hash_suffix, count = hash_entry.split(':')
            if hash_suffix == suffix:
                print(f"Password has been exposed {count} times in data breaches.")
                return True
        print("Password is safe (no breaches found).")
        return False
    else:
        print("Error querying the API.")
        return False

def index(self):
    if 'username' not in session:
        return redirect(url_for('login'))
    user_id = session['user_id']
    passwords = DatabaseManager.get_passwords(user_id)
    return render_template('index.html', passwords=passwords, username=session['username'])

def login(self):
    if 'username' in session:
        return redirect(url_for('index'))

    if request.method == 'POST':
        username = request.form['username']
        password = request.form['password']
        user = DatabaseManager.validate_user(username, password)
        if user:
            session['username'] = username
            session['user_id'] = user['id']
            flash('Logged in successfully!', 'success')
            return redirect(url_for('index'))
        else:
            flash('Invalid username or password.', 'danger')

    return render_template('login.html')

def logout(self):
    session.clear()
    flash('Logged out successfully!', 'success')
    return redirect(url_for('login'))

def register(self):
    if 'username' in session:
        return redirect(url_for('index'))

    if request.method == 'POST':
        username = request.form['username']
```

```
            password = request.form['password']
            confirm_password = request.form['confirm_password']
            email = request.form['email']

            if password != confirm_password:
                flash('Passwords do not match, please try again.', 'danger')
                return redirect(url_for('register'))

            if DatabaseManager.get_user_by_username(username):
                flash('Username already exists.', 'danger')
                return redirect(url_for('register'))

            DatabaseManager.insert_user(username, password, email)
            flash('Registration successful! Please log in.', 'success')
            return redirect(url_for('login'))

        return render_template('register.html')

    def add_password(self):
        if 'username' not in session:
            return redirect(url_for('login'))

        if request.method == 'POST':
            service = request.form['service_name']
            password = request.form['password']
            user_id = session['user_id']
            self.check_password_breach(password)

            DatabaseManager.insert_password(user_id, service, password)
            flash(f'Password for {service} added successfully!', 'success')
            return redirect(url_for('index'))

        return render_template('add_password.html')

    def update_password(self, service):
        if 'username' not in session:
            return redirect(url_for('login'))

        user_id = session['user_id']
        existing_password = DatabaseManager.get_passwords(user_id)
        service_password = next((pw for pw in existing_password if pw['service'] == service), None)

        if request.method == 'POST':
            password = request.form['password']
            DatabaseManager.update_password(user_id, service, password)
            flash(f'Password for {service} updated successfully!', 'success')
            return redirect(url_for('index'))
        return render_template('update_password.html', service_password=service_password)

    def delete_password(self, service):
        if 'username' not in session:
            return redirect(url_for('login'))

        user_id = session['user_id']
        DatabaseManager.delete_password(user_id, service)
        flash(f'Password for {service} deleted successfully!', 'success')
        return redirect(url_for('index'))

    # API endpoint for retrieving stored password for autofill
    def get_password(self):
        if 'username' not in session:
            return jsonify({"error": "User not logged in"}), 401

        user_id = session['user_id']
        domain = request.args.get('domain')

        if not domain:
            return jsonify({"error": "Domain parameter missing"}), 400

        with DatabaseManager.get_db_connection() as conn:
            passwords = conn.execute('SELECT * FROM passwords WHERE user_id = ? AND service LIKE
?',
                                    (user_id, f'%{domain}%')).fetchall()

        if passwords:
            password_data = []
            for password in passwords:
                password_data.append({
                    "service": password['service'],
                    "password": password['password']
                })
            return jsonify(password_data)
        else:
            return jsonify({"error": "No passwords found for this domain"}), 404

    def run(self):
        self.app.run(debug=True)

# Run the app
if __name__ == '__main__':
    app = AppRoutes()
    app.run()
```

HTML

Login page

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Login - Lockzilla</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <style>
    /* General body styling */
    body {
      font-family: 'Roboto', sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f4f4f4;
      color: #333;
    }

    .container {
      max-width: 500px;
      margin: 40px auto;
      padding: 30px;
      background: #ffffff;
      border-radius: 15px;
      border: 1px solid #e0e0e0;
      box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);
      animation: fadeIn 1s ease-out;
    }

    header {
      text-align: center;
      margin-bottom: 30px;
      position: relative;
    }

    header h1 {
      font-size: 3rem;
      color: #222;
      text-transform: uppercase;
      letter-spacing: 3px;
      margin: 0;
      font-weight: bold;
      position: relative;
      animation: slideInUp 1s ease-out;
      text-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
    }

    h2 {
      text-align: center;
      margin: 20px 0;
      font-size: 2.2rem;
      color: #222;
      text-transform: uppercase;
      font-weight: bold;
      letter-spacing: 1px;
    }

    form {
      display: flex;
      flex-direction: column;
      gap: 20px;
    }

    form label {
      font-size: 1.1rem;
      color: #444;
      font-weight: 600;
    }

    form input {
      padding: 12px;
      font-size: 1rem;
      border: 1px solid #ddd;
      border-radius: 8px;
      background-color: #fafafa;
      transition: border-color 0.3s ease-in-out;
    }

    form input:focus {
      border-color: #007BFF;
      outline: none;
      box-shadow: 0 0 8px rgba(0, 123, 255, 0.4);
    }

    form .btn {
      background: #333;
      color: #fff;
      padding: 12px 25px;
      border-radius: 8px;
      font-size: 1.1rem;
      font-weight: bold;
      text-decoration: none;
      text-align: center;
      transition: all 0.3s ease-in-out;
      box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
    }

    form .btn:hover {
      background: #007BFF;
      transform: translateY(-5px);
      box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);
    }

    a.btn {
      display: inline-block;
      margin-top: 20px;
      text-align: center;
      background: #333;
      color: #fff;
      padding: 12px 25px;
      border-radius: 8px;
      font-size: 1.1rem;
      font-weight: bold;
      letter-spacing: 1px;
      text-decoration: none;
      transition: all 0.3s ease-in-out;
      box-shadow: 0 8px 30px rgba(0, 0, 0, 0.15);
    }

    a.btn:hover {
      background: #007BFF;
      transform: translateY(-5px);
      box-shadow: 0 15px 35px rgba(0, 0, 0, 0.2);
    }

    @keyframes fadeIn {
      from {
        opacity: 0;
        transform: scale(0.8);
      }
      to {
        opacity: 1;
        transform: scale(1);
      }
    }

    @keyframes slideInUp {
      from {
        opacity: 0;
        transform: translateY(50px);
```

```
            }
            to {
                opacity: 1;
                transform: translateY(0);
            }
        }

        @media (max-width: 768px) {
            .container {
                padding: 20px;
            }

            h1 {
                font-size: 2.5rem;
            }

            h2 {
                font-size: 1.8rem;
            }

            form {
                gap: 15px;
            }

            form input, form .btn {
                padding: 10px;
            }
        }
    </style>
    <div class="container">
        <header>
            <h1>Lockzilla</h1>
        </header>
        <h2>Login</h2>
        <form method="POST">
            <label for="username">Username:</label>
            <input type="text" id="username" name="username" required>

            <label for="password">Password:</label>
            <input type="password" id="password" name="password" required>

            <button type="submit" name="login" class="btn">Login</button>
        </form>

        <a href="{{ url_for('register') }}" class="btn">Back to Register</a>
    </div>
</body>
</html>
```

Index page

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Lockzilla - Password Manager</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
    <style>
        /* Keep your original CSS styles intact */

body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background: #f9f9f9;
  color: #333333;
  overflow-x: hidden;
}

.container {
  max-width: 900px; /* Reduced max width */
  margin: 40px auto;
  padding: 15px 20px; /* Reduced padding to decrease height */
  background: #ffffff;
  border-radius: 15px;
  border: 1px solid #e0e0e0;
  box-shadow: 0 15px 45px rgba(0, 0, 0, 0.1);
  animation: fadeIn 1s ease-out;
}

header {
  text-align: center;
  margin-bottom: 30px; /* Reduced margin */
  position: relative;
  z-index: 2;
}

header h1 {
  font-size: 3rem; /* Slightly reduced font size */
  color: #2c2c2c;
  text-transform: uppercase;
  letter-spacing: 8px;
  margin: 0;
  font-weight: bold;
  position: relative;
```

```
 animation: slideInUp 1s ease-out;
 text-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
}

header p {
 margin: 15px 0;
 font-size: 1.4rem;
 color: #777777;
}

header .btn {
 background: #4A90E2;
 color: #ffffff;
 padding: 10px 20px; /* Reduced padding */
 border-radius: 10px;
 font-weight: bold;
 font-size: 1rem; /* Reduced font size */
 text-decoration: none;
 transition: all 0.3s ease-in-out;
 box-shadow: 0 6px 20px rgba(72, 144, 226, 0.4);
 display: inline-block;
}

header .btn:hover {
 background: #0066cc;
 transform: translateY(-3px);
 box-shadow: 0 10px 30px rgba(0, 102, 204, 0.4);
}

h2 {
 text-align: center;
 margin: 20px 0;
 font-size: 2.3rem; /* Reduced font size */
 color: #333333;
 text-transform: uppercase;
 font-weight: bold;
 letter-spacing: 2px;
 position: relative;
 animation: fadeInUp 1s ease-out;
}

h2::after {
 content: '';
 display: block;
 width: 180px;
 height: 4px;
 background: #4A90E2;
 margin: 10px auto 0;
```

```
  border-radius: 2px;
}

table {
  width: 100%;
  border-collapse: collapse;
  background: #f9f9f9;
  margin-top: 20px;
  border-radius: 10px;
  overflow: hidden;
  animation: fadeInUp 1.5s ease-out;
}

table thead {
  background: #4A90E2;
  color: #ffffff;
  text-transform: uppercase;
}

table th, table td {
  padding: 15px; /* Reduced padding */
  text-align: center;
  font-size: 1.1rem;
  font-weight: 500;
  border-bottom: 1px solid rgba(0, 0, 0, 0.1);
}

table tbody tr:hover {
  background: rgba(0, 102, 204, 0.1);
  cursor: pointer;
  transition: background 0.3s ease-in-out;
}

table .btn {
  background: #4A90E2;
  color: #ffffff;
  padding: 8px 20px; /* Reduced padding */
  text-decoration: none;
  border-radius: 5px;
  font-size: 0.9rem; /* Slightly smaller font size */
  font-weight: 600;
  transition: background-color 0.3s ease, transform 0.3s ease;
  box-shadow: 0 5px 15px rgba(72, 144, 226, 0.3);
}

table .btn:hover {
  background: #0066cc;
  transform: scale(1.05);
```

```
 box-shadow: 0 8px 25px rgba(0, 102, 204, 0.5);
}

a.btn {
 display: inline-block;
 margin: 30px 0; /* Reduced margin */
 text-align: center;
 background: #4A90E2;
 color: #ffffff;
 padding: 12px 25px; /* Reduced padding */
 border-radius: 8px;
 font-size: 1.1rem; /* Reduced font size */
 font-weight: bold;
 letter-spacing: 1px;
 text-decoration: none;
 transition: all 0.3s ease-in-out;
 box-shadow: 0 8px 30px rgba(72, 144, 226, 0.4);
 display: inline-block;
}

a.btn:hover {
 background: #ff6ec7;
 transform: translateY(-5px);
 box-shadow: 0 15px 35px rgba(255, 110, 199, 0.5);
}

@keyframes fadeIn {
 from {
   opacity: 0;
   transform: scale(0.8);
 }
 to {
   opacity: 1;
   transform: scale(1);
 }
}

@keyframes fadeInUp {
 from {
   opacity: 0;
   transform: translateY(30px);
 }
 to {
   opacity: 1;
   transform: translateY(0);
 }
}
```

```
@keyframes slideInUp {
  from {
    opacity: 0;
    transform: translateY(50px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

@media (max-width: 768px) {
  header h1 {
    font-size: 2.8rem;
  }

  table thead {
    display: none;
  }

  table tbody tr {
    display: block;
    margin-bottom: 15px;
    border: 1px solid rgba(0, 0, 0, 0.1);
    border-radius: 8px;
    padding: 15px;
  }

  table tbody td {
    display: block;
    text-align: right;
    font-size: 1rem;
    padding: 15px 0;
  }

  table tbody td::before {
    content: attr(data-label);
    float: left;
    font-weight: bold;
    color: #4A90E2;
    text-transform: uppercase;
  }

  a.btn {
    width: 100%;
    text-align: center;
  }
}
```

```
  </style>
  <div class="container">
    <header>
      <h1>Lockzilla</h1>
      <p>Welcome, {{ username }}!</p> <!-- Show username -->
      <a href="{{ url_for('logout') }}" class="btn">Logout</a>
    </header>

    <h2>Your Saved Passwords</h2>

    {% if passwords %}
      <table>
        <thead>
          <tr>
            <th>Service</th>
            <th>Password</th>
            <th>Actions</th>
          </tr>
        </thead>
        <tbody>
          {% for password in passwords %}
            <tr>
              <td>{{ password['service'] }}</td>
              <td>{{ password['password'] }}  - Safe</td>
              <td>
                <a href="{{ url_for('update_password', service=password['service']) }}"
class="btn">Update</a>
                <!-- Form to delete password -->
                <form method="POST" action="{{ url_for('delete_password',
service=password['service']) }}">
                  <button type="submit" class="btn btn-danger">Delete</button>
                </form>
              </td>
            </tr>
          {% endfor %}
        </tbody>
      </table>
    {% else %}
      <p>No passwords saved.</p>
    {% endif %}

    <a href="{{ url_for('add_password') }}" class="btn">Add New Password</a>

  </div>
</body>
</html>
```

Add Password

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Add Password - Lockzilla</title>
    <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">

    <script>
        // Function to generate a random password
        function generatePassword(length = 12) {
            const charset =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()_+[]{}|;:
,.<>?";
            let password = "";
            for (let i = 0; i < length; i++) {
                const randomIndex = Math.floor(Math.random() * charset.length);
                password += charset[randomIndex];
            }
            return password;
        }

        // Function to copy password to clipboard
        function copyToClipboard() {
            const passwordField = document.getElementById("password");
            passwordField.select();
            passwordField.setSelectionRange(0, 99999); // For mobile devices
            document.execCommand("copy"); // Copy the password to clipboard
            alert("Password copied to clipboard!"); // Alert for the user
        }

        // Generate and set password in the input field when the button is clicked
        function onGenerateClick() {
            const generatedPassword = generatePassword();
            document.getElementById("password").value = generatedPassword;
            copyToClipboard(); // Automatically copy password to clipboard
        }
    </script>
</head>
<body>
    <style>
        /* styles.css */

/* General body styling */
body {
    font-family: 'Roboto', sans-serif;
    margin: 0;
```

```
  padding: 0;
  background-color: #f4f4f4;
  color: #333;
}

.container {
  max-width: 500px; /* Container width for Update Password */
  margin: 40px auto;
  padding: 30px;
  background: #ffffff;
  border-radius: 15px;
  border: 1px solid #e0e0e0;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);
  animation: fadeIn 1s ease-out;
}

header {
  text-align: center;
  margin-bottom: 30px;
  position: relative;
}

header h1 {
  font-size: 3rem;
  color: #222;
  text-transform: uppercase;
  letter-spacing: 3px;
  margin: 0;
  font-weight: bold;
  position: relative;
  animation: slideInUp 1s ease-out;
  text-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
}

/* Form Section */
h2 {
  text-align: center;
  margin: 20px 0;
  font-size: 2.2rem;
  color: #222;
  text-transform: uppercase;
  font-weight: bold;
  letter-spacing: 1px;
}

form {
  display: flex;
  flex-direction: column;
  gap: 20px;
}

form label {
  font-size: 1.1rem;
  color: #444;
  font-weight: 600;
}

form input {
  padding: 12px;
  font-size: 1rem;
  border: 1px solid #ddd;
  border-radius: 8px;
  background-color: #fafafa;
  transition: border-color 0.3s ease-in-out;
}

form input:focus {
  border-color: #007BFF;
  outline: none;
  box-shadow: 0 0 8px rgba(0, 123, 255, 0.4);
}

form .btn {
  background: #333;
  color: #fff;
  padding: 12px 25px;
  border-radius: 8px;
  font-size: 1.1rem;
  font-weight: bold;
  text-decoration: none;
  text-align: center;
  transition: all 0.3s ease-in-out;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
}

form .btn:hover {
  background: #007BFF;
  transform: translateY(-5px);
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);
}

/* Back to List button */
a.btn {
  display: inline-block;
  margin-top: 20px;
  text-align: center;
  background: #333;
  color: #fff;
  padding: 12px 25px;
  border-radius: 8px;
  font-size: 1.1rem;
  font-weight: bold;
  letter-spacing: 1px;
  text-decoration: none;
  transition: all 0.3s ease-in-out;
  box-shadow: 0 8px 30px rgba(0, 0, 0, 0.15);
}

a.btn:hover {
  background: #007BFF;
  transform: translateY(-5px);
  box-shadow: 0 15px 35px rgba(0, 0, 0, 0.2);
}

/* Animation Effects */
@keyframes fadeIn {
  from {
    opacity: 0;
    transform: scale(0.8);
  }
  to {
    opacity: 1;
    transform: scale(1);
  }
}

@keyframes slideInUp {
  from {
    opacity: 0;
    transform: translateY(50px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

/* Responsive Design */
@media (max-width: 768px) {
  .container {
    padding: 20px;
  }

  h1 {
    font-size: 2.5rem;
  }

  h2 {
    font-size: 1.8rem;
  }

  form {
    gap: 15px;
  }

  form input, form .btn {
    padding: 10px;
  }
}

  </style>
  <div class="container">
    <header>
      <h1>Lockzilla</h1>
      <a href="{{ url_for('logout') }}" class="btn">Logout</a>
    </header>
    <h2>Add New Password</h2>
    <form method="POST">
      <label for="username">Username:</label>
      <input type="text" id="username" name="username" value="{{ session['username'] }}"
readonly required>

      <label for="service_name">Service Name:</label>
      <input type="text" id="service_name" name="service_name" required>

      <label for="password">Password:</label>
      <input type="text" id="password" name="password" value="{{ generated_password }}"
readonly required>

      <button type="button" class="btn" onclick="onGenerateClick()">Generate & Copy
Password</button>
      <button type="submit" name="save" class="btn">Save Password</button>
    </form>
    <a href="{{ url_for('index') }}" class="btn">Back to List</a>
  </div>
</body>
</html>
```

Update Password

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Update Password - Lockzilla</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">

  <script>
    // Function to generate a random password
    function generatePassword(length = 12) {
      const charset =
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789!@#$%^&*()_+[]{}|;:
,.<>?";
      let password = "";
      for (let i = 0; i < length; i++) {
        const randomIndex = Math.floor(Math.random() * charset.length);
        password += charset[randomIndex];
      }
      return password;
    }

    // Function to copy password to clipboard
    function copyToClipboard() {
      const passwordField = document.getElementById("password");
      passwordField.select();
      passwordField.setSelectionRange(0, 99999); // For mobile devices
      document.execCommand("copy"); // Copy the password to clipboard
      alert("Password copied to clipboard!"); // Alert for the user
    }

    // Generate and set password in the input field when the button is clicked
    function onGenerateClick() {
      const generatedPassword = generatePassword();
      document.getElementById("password").value = generatedPassword;
      copyToClipboard(); // Automatically copy password to clipboard
    }
  </script>
</head>
<body>
  <style>
    /* styles.css */

/* General body styling */
body {
  font-family: 'Roboto', sans-serif;
  margin: 0;
  padding: 0;
  background-color: #f4f4f4;
  color: #333;
}

.container {
  max-width: 500px; /* Container width for Login */
  margin: 40px auto;
  padding: 30px;
  background: #ffffff;
  border-radius: 15px;
  border: 1px solid #e0e0e0;
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);
  animation: fadeIn 1s ease-out;
}

header {
  text-align: center;
  margin-bottom: 30px;
  position: relative;
}

header h1 {
  font-size: 3rem;
  color: #222;
  text-transform: uppercase;
  letter-spacing: 3px;
  margin: 0;
  font-weight: bold;
  position: relative;
  animation: slideInUp 1s ease-out;
  text-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
}

/* Form Section */
h2 {
  text-align: center;
  margin: 20px 0;
  font-size: 2.2rem;
  color: #222;
  text-transform: uppercase;
  font-weight: bold;
  letter-spacing: 1px;
}

form {
  display: flex;
  flex-direction: column;
  gap: 20px;
}

form label {
  font-size: 1.1rem;
  color: #444;
  font-weight: 600;
}

form input {
  padding: 12px;
  font-size: 1rem;
  border: 1px solid #ddd;
  border-radius: 8px;
  background-color: #fafafa;
  transition: border-color 0.3s ease-in-out;
}

form input:focus {
  border-color: #007BFF;
  outline: none;
  box-shadow: 0 0 8px rgba(0, 123, 255, 0.4);
}

form .btn {
  background: #333;
  color: #fff;
  padding: 12px 25px;
  border-radius: 8px;
  font-size: 1.1rem;
  font-weight: bold;
  text-decoration: none;
  text-align: center;
  transition: all 0.3s ease-in-out;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
}

form .btn:hover {
  background: #007BFF;
  transform: translateY(-5px);
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);
}

/* Back to List button */
a.btn {
  display: inline-block;
  margin-top: 20px;
  text-align: center;
  background: #333;
  color: #fff;
  padding: 12px 25px;
  border-radius: 8px;
  font-size: 1.1rem;
  font-weight: bold;
  letter-spacing: 1px;
  text-decoration: none;
  transition: all 0.3s ease-in-out;
  box-shadow: 0 8px 30px rgba(0, 0, 0, 0.15);
}

a.btn:hover {
  background: #007BFF;
  transform: translateY(-5px);
  box-shadow: 0 15px 35px rgba(0, 0, 0, 0.2);
}

/* Animation Effects */
@keyframes fadeIn {
  from {
    opacity: 0;
    transform: scale(0.8);
  }
  to {
    opacity: 1;
    transform: scale(1);
  }
}

@keyframes slideInUp {
  from {
    opacity: 0;
    transform: translateY(50px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}

/* Responsive Design */
@media (max-width: 768px) {
  .container {
    padding: 20px;
  }

  h1 {
    font-size: 2.5rem;
  }

  h2 {
    font-size: 1.8rem;
  }

  form {
    gap: 15px;
  }

  form input, form .btn {
    padding: 10px;
  }
}

  </style>
  <div class="container">
    <header>
      <h1>Lockzilla</h1>
      <a href="{{ url_for('logout') }}" class="btn">Logout</a>
    </header>
    <h2>Update Password for "{{ name }}"</h2>
    <form method="POST">
      <label for="username">Username:</label>
      <input type="text" id="username" name="username" value="{{ session['username'] }}"
readonly required>

      <label for="password">New Password (current: {{ current_password }}):</label>
      <input type="text" id="password" name="password" value="{{ generated_password }}"
readonly required>

      <button type="button" class="btn" onclick="onGenerateClick()">Generate & Copy
Password</button>
      <button type="submit" name="update" class="btn">Update Password</button>
    </form>
    <a href="{{ url_for('index') }}" class="btn">Back to List</a>
  </div>
</body>
</html>
```

Register page

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Register - Lockzilla</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='styles.css') }}">
</head>
<body>
  <style>
    /* General body styling */
    body {
      font-family: 'Roboto', sans-serif;
      margin: 0;
      padding: 0;
      background-color: #f4f4f4;
      color: #333;
    }

    .container {
      max-width: 500px;
      margin: 40px auto;
      padding: 30px;
      background: #ffffff;
      border-radius: 15px;
      border: 1px solid #e0e0e0;
      box-shadow: 0 10px 30px rgba(0, 0, 0, 0.1);
      animation: fadeIn 1s ease-out;
    }

    header {
      text-align: center;
      margin-bottom: 30px;
      position: relative;
    }

    header h1 {
      font-size: 3rem;
      color: #222;
      text-transform: uppercase;
      letter-spacing: 3px;
      margin: 0;
      font-weight: bold;
      position: relative;
      animation: slideInUp 1s ease-out;
      text-shadow: 0 4px 15px rgba(0, 0, 0, 0.1);
    }
```

```
h2 {
  text-align: center;
  margin: 20px 0;
  font-size: 2.2rem;
  color: #222;
  text-transform: uppercase;
  font-weight: bold;
  letter-spacing: 1px;
}

form {
  display: flex;
  flex-direction: column;
  gap: 20px;
}

form label {
  font-size: 1.1rem;
  color: #444;
  font-weight: 600;
}

form input {
  padding: 12px;
  font-size: 1rem;
  border: 1px solid #ddd;
  border-radius: 8px;
  background-color: #fafafa;
  transition: border-color 0.3s ease-in-out;
}

form input:focus {
  border-color: #007BFF;
  outline: none;
  box-shadow: 0 0 8px rgba(0, 123, 255, 0.4);
}

form .btn {
  background: #333;
  color: #fff;
  padding: 12px 25px;
  border-radius: 8px;
  font-size: 1.1rem;
  font-weight: bold;
  text-decoration: none;
  text-align: center;
  transition: all 0.3s ease-in-out;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.1);
}
form .btn:hover {
  background: #007BFF;
  transform: translateY(-5px);
  box-shadow: 0 10px 30px rgba(0, 0, 0, 0.2);
}
a.btn {
  display: inline-block;
  margin-top: 20px;
  text-align: center;
  background: #333;
  color: #fff;
  padding: 12px 25px;
  border-radius: 8px;
  font-size: 1.1rem;
  font-weight: bold;
  letter-spacing: 1px;
  text-decoration: none;
  transition: all 0.3s ease-in-out;
  box-shadow: 0 8px 30px rgba(0, 0, 0, 0.15);
}
a.btn:hover {
  background: #007BFF;
  transform: translateY(-5px);
  box-shadow: 0 15px 35px rgba(0, 0, 0, 0.2);
}
@keyframes fadeIn {
  from {
    opacity: 0;
    transform: scale(0.8);
  }
  to {
    opacity: 1;
    transform: scale(1);
  }
}
@keyframes slideInUp {
  from {
    opacity: 0;
    transform: translateY(50px);
  }
  to {
    opacity: 1;
    transform: translateY(0);
  }
}
```

```
    @media (max-width: 768px) {
      .container {
        padding: 20px;
      }

      h1 {
        font-size: 2.5rem;
      }

      h2 {
        font-size: 1.8rem;
      }

      form {
        gap: 15px;
      }

      form input, form .btn {
        padding: 10px;
      }
    }
  </style>
  <div class="container">
    <header>
      <h1>Lockzilla</h1>
    </header>
    <h2>Register</h2>
    <form method="POST">
  <label for="username">Username:</label>
  <input type="text" id="username" name="username" required>

  <label for="email">Email:</label>
  <input type="email" id="email" name="email" required>

  <label for="password">Password:</label>
  <input type="password" id="password" name="password" required>

  <label for="confirm_password">Confirm Password:</label>
  <input type="password" id="confirm_password" name="confirm_password" required>

  <button type="submit" name="register" class="btn">Register</button>
</form>


    <a href="{{ url_for('login') }}" class="btn">Back to Login</a>
  </div>
</body>
</html>
```

CSS

```
/* styles.css */
body {
  margin: 0;
  font-family: 'Arial', sans-serif;
  background: linear-gradient(120deg, #2b2b2b, #121212);
  color: #ffffff;
  display: flex;
  justify-content: center;
  align-items: center;
  min-height: 100vh;
}

.container {
  max-width: 900px;
  width: 100%;
  background: rgba(0, 0, 0, 0.8);
  border-radius: 15px;
  box-shadow: 0px 10px 30px rgba(0, 0, 0, 0.5);
  padding: 20px;
  animation: fadeIn 1s ease-in-out;
}

header {
  text-align: center;
  margin-bottom: 20px;
}

header h1 {
  font-size: 2.5em;
  background: linear-gradient(90deg, #ff00cc, #3333ff);
  -webkit-background-clip: text;
  -webkit-text-fill-color: transparent;
  animation: pulse 2s infinite;
}

header p {
  font-size: 1.2em;
  margin: 10px 0;
}

table {
  width: 100%;
  border-collapse: collapse;
  margin-bottom: 20px;
  overflow: hidden;
  border-radius: 10px;
  box-shadow: 0 5px 15px rgba(0, 0, 0, 0.3);
}

table thead {
  background: #3333ff;
}

table thead th {
  color: #fff;
  font-weight: bold;
  padding: 10px;
  text-align: left;
}

table tbody tr {
  background: #1a1a1a;
  border-bottom: 1px solid rgba(255, 255, 255, 0.1);
  transition: background 0.3s ease-in-out;
}

table tbody tr:hover {
  background: rgba(255, 0, 204, 0.2);
}

table td {
  padding: 10px;
}

table td a.btn,
table td form button {
  display: inline-block;
  background: linear-gradient(90deg, #ff00cc, #3333ff);
  border: none;
  padding: 8px 15px;
  color: #fff;
  border-radius: 5px;
  text-decoration: none;
  font-size: 0.9em;
  transition: transform 0.3s, box-shadow 0.3s;
  cursor: pointer;
}

table td a.btn:hover,
table td form button:hover {
  transform: scale(1.1);
  box-shadow: 0px 5px 15px rgba(255, 0, 204, 0.4);
}
```

```
a.btn {
  display: inline-block;
  background: linear-gradient(90deg, #ff00cc, #3333ff);
  padding: 10px 20px;
  color: #fff;
  border-radius: 5px;
  text-decoration: none;
  text-align: center;
  font-weight: bold;
  transition: transform 0.3s, box-shadow 0.3s;
  margin-top: 10px;
}

a.btn:hover {
  transform: translateY(-5px);
  box-shadow: 0 10px 20px rgba(255, 0, 204, 0.4);
}

@keyframes fadeIn {
  from {
    opacity: 0;
    transform: scale(0.9);
  }
  to {
    opacity: 1;
    transform: scale(1);
  }
}

@keyframes pulse {
  0%, 100% {
    filter: brightness(1);
  }
  50% {
    filter: brightness(1.5);
  }
}
```

Important Code snippets

Following is the code for the main functionalities of the password manager:

```
class DatabaseManager:
    @staticmethod
    def get_db_connection():
        conn = sqlite3.connect('lockzilla.db', timeout=10)  # Timeout to prevent immediate lock
errors
        conn.row_factory = sqlite3.Row
        conn.execute('PRAGMA journal_mode = WAL')  # Enable Write-Ahead Logging (WAL)
for concurrency
        return conn

    @staticmethod
    def get_user_by_username(username):
        with DatabaseManager.get_db_connection() as conn:
            user = conn.execute('SELECT * FROM users WHERE username = ?',
(username,)).fetchone()
        return user

    @staticmethod
    def insert_user(username, password, email):
        with DatabaseManager.get_db_connection() as conn:
            conn.execute('INSERT INTO users (username, password, email) VALUES (?, ?, ?)',
                         (username, generate_password_hash(password), email))
            conn.commit()

    @staticmethod
    def validate_user(username, password):
        user = DatabaseManager.get_user_by_username(username)
        if user and check_password_hash(user['password'], password):
            return user
        return None

    @staticmethod
    def get_passwords(user_id):
        with DatabaseManager.get_db_connection() as conn:
            passwords = conn.execute('SELECT * FROM passwords WHERE user_id = ?',
(user_id,)).fetchall()
        return passwords

    @staticmethod
    def insert_password(user_id, service, password):
        with DatabaseManager.get_db_connection() as conn:
            conn.execute('INSERT INTO passwords (user_id, service, password) VALUES (?, ?, ?)',
                        (user_id, service, password))
            conn.commit()

    @staticmethod
    def update_password(user_id, service, password):
        with DatabaseManager.get_db_connection() as conn:
            conn.execute('UPDATE passwords SET password = ? WHERE user_id = ? AND service
= ?',
                        (password, user_id, service))
            conn.commit()

    @staticmethod
    def delete_password(user_id, service):
        with DatabaseManager.get_db_connection() as conn:
            conn.execute('DELETE FROM passwords WHERE user_id = ? AND service = ?',
                        (user_id, service))
            conn.commit()
```

GitHub link & Folder Structure

GitHub Link: https://github.com/SuhaasRao001/LockZilla

Folder Structure:

```
Lockzilla/
│
├── main.py          # Main Flask application
├── templates/        # HTML templates
│   ├── index.html
│   ├── login.html      # Login page
│   ├── register.html   # Registration page
│   ├── update_password.html
│   ├── add_password.html    # Page to add a new password
├── static/            # CSS files
│   ├── styles.css      # Styles for the application
├── requirements.txt    # Python dependencies
└── README.md         # Project documentation
```

TESTING

Test case id: IND01		Use case id: IND
Test case title: Test the Dashboard		
Test case description: Clicking on login would direct to dashboard where the passwords are visible		
Test steps	Expected Result	Actual Result
Click on login	Go to dashboard	Directed to dashboard

Test case id: AP01		Use case id: AP
Test case title: Test the add password		
Test case description: Click on add new password button in the index page has taken us to add password page to add new password with service name and password that generates the password and copies to clipboard.		
Test steps	Expected Result	Actual Result
Click on add password	Go to add password page and create a new password	Directed to add password and created the password and copied to clipboard automatically

Test case id: AP02		Use case id: AP
Test case title: Test the dashboard to see if new password is there		
Test case description: Check the dashboard to see if the password has the encrypted password or not		
Test steps	Expected Result	Actual Result
Check the dashboard for the service name	New service and password should be seen	New service and password have been created and visible

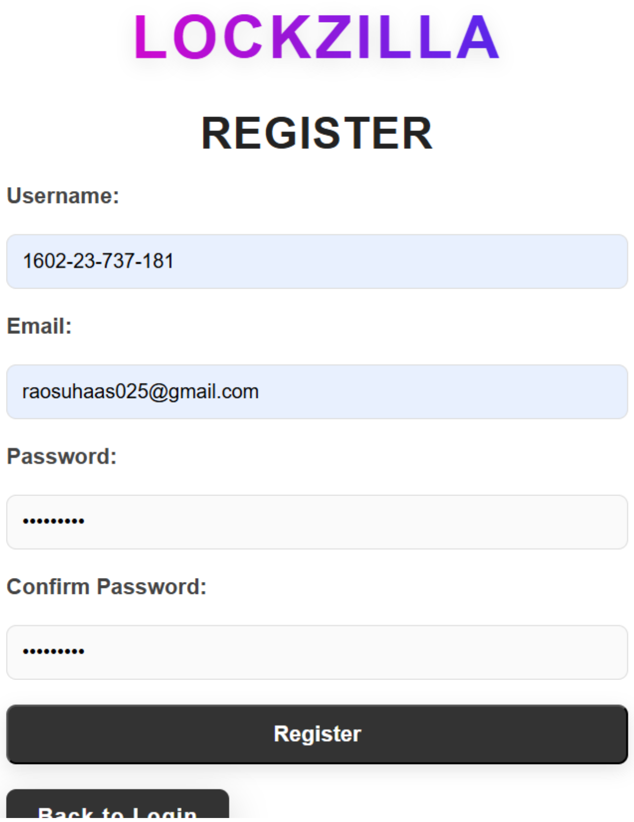
Test case id: UP01		Use case id: UP
Test case title: Test the update password		
Test case description: Click on update password button in the index page has taken us to update password page to update the existing password of service and password that generates the password and copies to clipboard.		
Test steps	Expected Result	Actual Result
Click on update password	Go to update password page and create a new password	Directed to update password and created the password and copied to clipboard automatically

Test case id: UP02		Use case id: UP
Test case title: Check dashboard for updated password		
Test case description: Check the dashboard for the new updated password		
Test steps	Expected Result	Actual Result
Check dashboard	Go to add password page and create a new password	Directed to add password and created the password and copied to clipboard automatically

Test case id: DEL01		Use case id: DEL
Test case title: Check dashboard and delete password		
Test case description: Click on delete button and see if the password is deleted or not		
Test steps	Expected Result	Actual Result
Click on delete	The created cell of password should be deleted	Password is removed

Test case id: PH01		Use case id: PH
Test case title: Check password health		
Test case description: If the generated password is safe after the API call it will show Safe beside the password		
Test steps	Expected Result	Actual Result
Check beside the Password for Safe	Safe word should be shown to validate safe password generation	Safe is shown for all passwords shown

RESULTS


LOCKZILLA

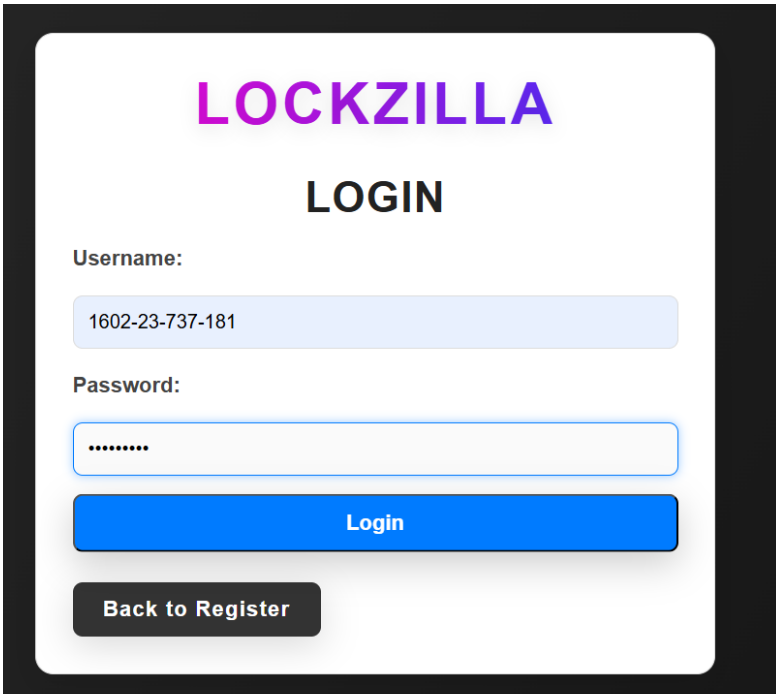
REGISTER

Username: 1602-23-737-181

Email: raosuhaas025@gmail.com

Password: •••••••••

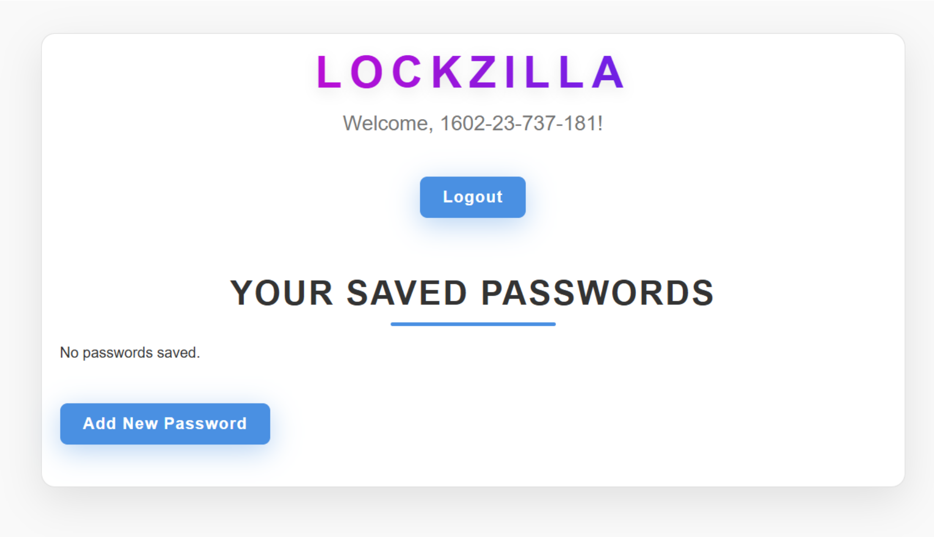
Confirm Password: •••••••••


LOCKZILLA

LOGIN

Username: 1602-23-737-181

Password: •••••••••

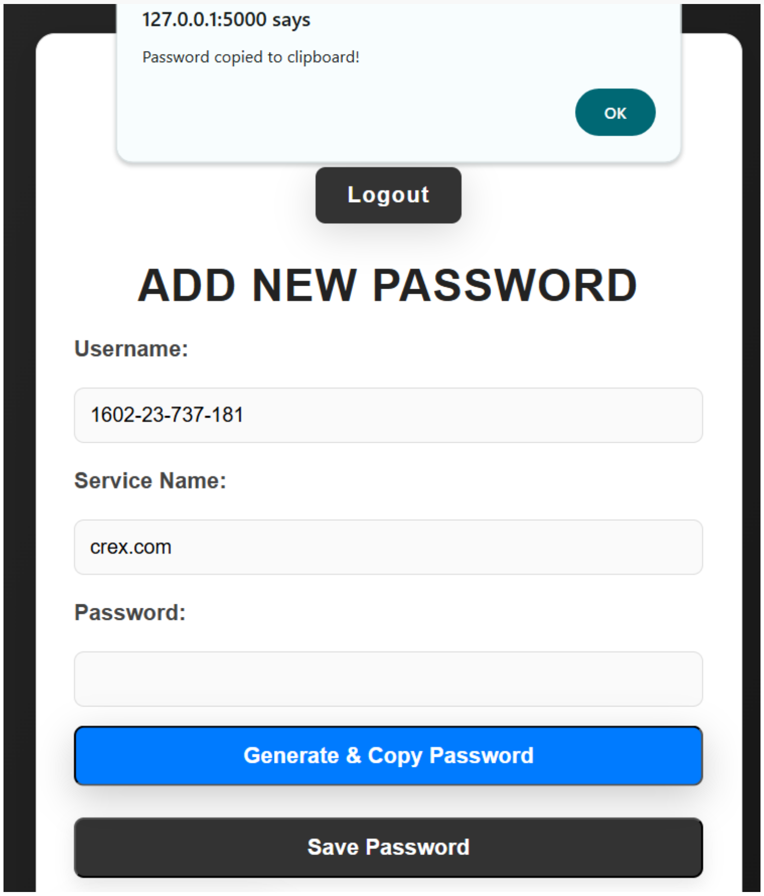


LOCKZILLA

Welcome, 1602-23-737-181!

YOUR SAVED PASSWORDS

No passwords saved.



127.0.0.1:5000 says

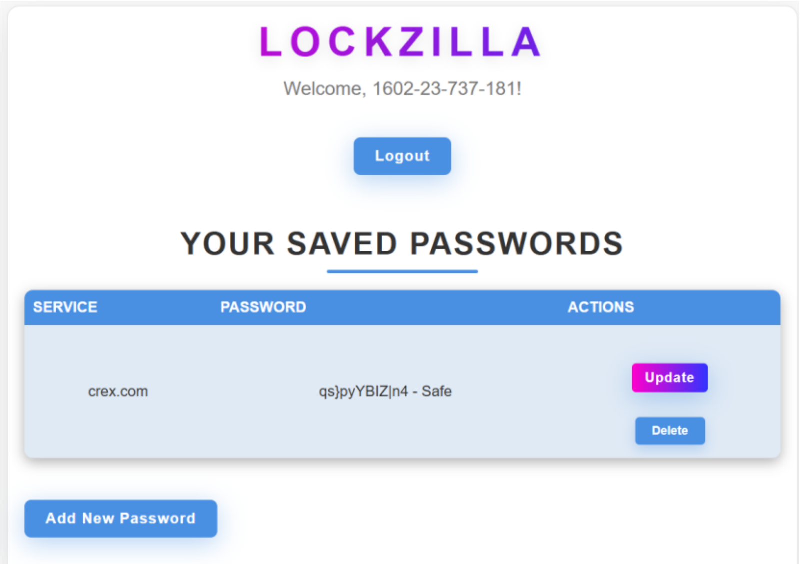
Password copied to clipboard!

ADD NEW PASSWORD

Username: 1602-23-737-181

Service Name: crex.com

Password:

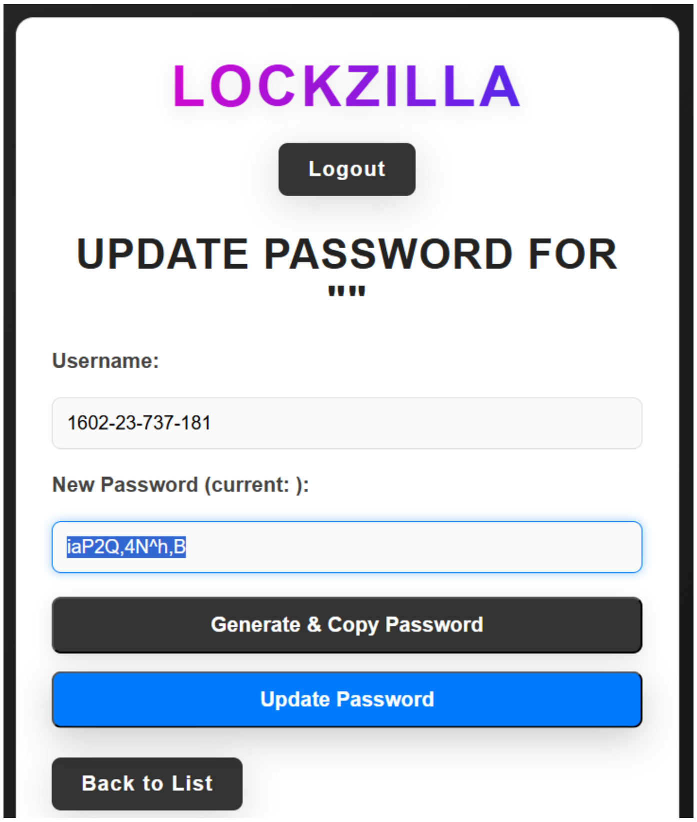


LOCKZILLA

Welcome, 1602-23-737-181!

YOUR SAVED PASSWORDS

SERVICE	PASSWORD	ACTIONS
crex.com	qs}pyYBIZ|n4 - Safe	Update Delete



LOCKZILLA

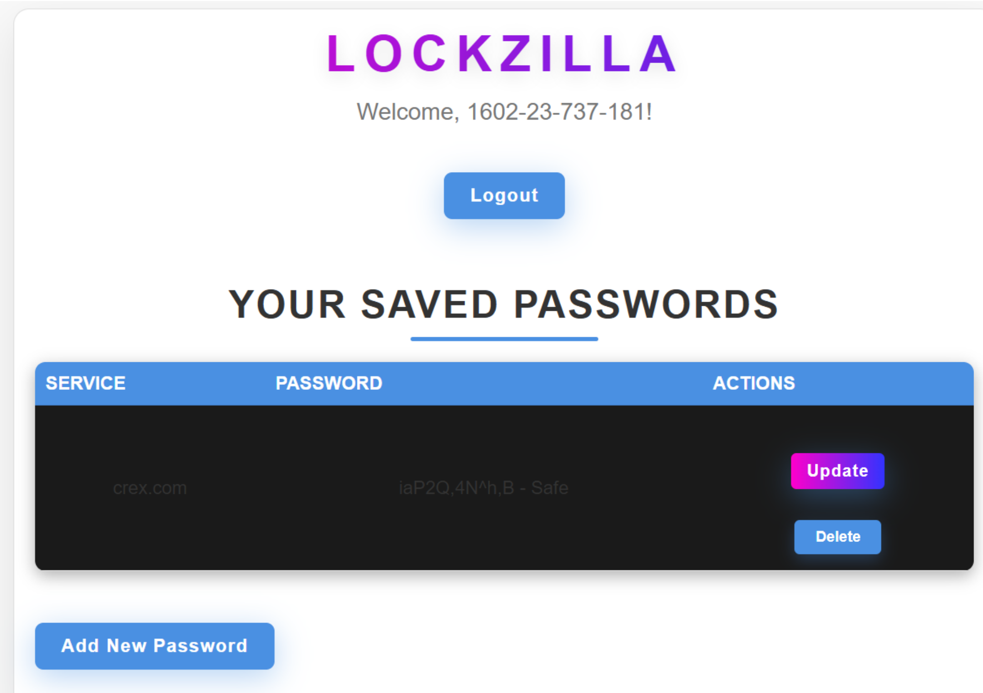
UPDATE PASSWORD FOR
""

Username:

1602-23-737-181

New Password (current:):

iaP2Q,4N^h,B

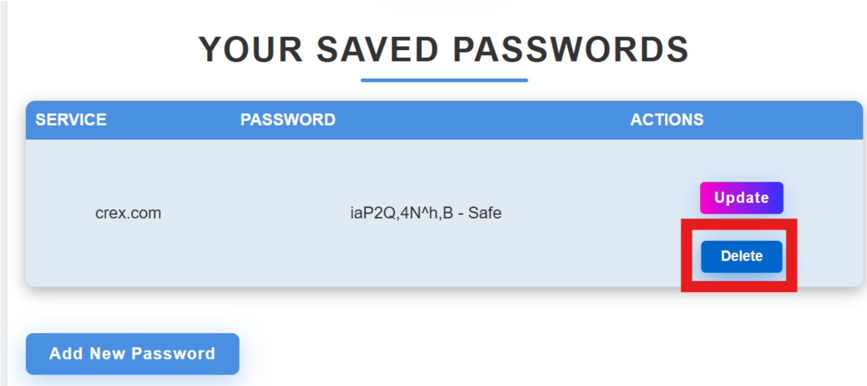


LOCKZILLA

Welcome, 1602-23-737-181!

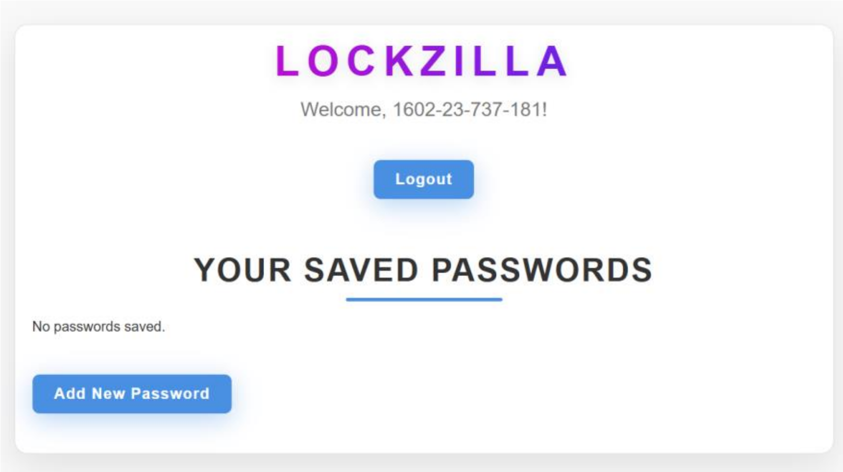
YOUR SAVED PASSWORDS

SERVICE	PASSWORD	ACTIONS
crex.com	iaP2Q,4N^h,B - Safe	Update Delete



YOUR SAVED PASSWORDS

SERVICE	PASSWORD	ACTIONS
crex.com	iaP2Q,4N^h,B - Safe	Update Delete



LOCKZILLA

Welcome, 1602-23-737-181!

YOUR SAVED PASSWORDS

No passwords saved.

ADDITIONAL KNOWLEDGE GAINED

1. **Web Development with Flask:**
   - **Learning Flask**: You gained valuable experience working with Flask, a lightweight Python web framework, to build Lockzilla's core functionality. You learned to handle HTTP requests, set up routes for different endpoints and manage user sessions securely, essential for a dynamic web application.
2. **Database Interaction with SQLite3:**
   - **SQLite3**: You developed proficiency in using SQLite3 as a lightweight database to store user information, encrypted passwords, and user preferences. You learned how to execute SQL queries directly to retrieve, store, and update password-related data.
3. **Encryption and Security:**
   - **Werkzeug Security Features**: You used the **Werkzeug** library to implement secure password hashing and session management. Werkzeug's built-in functions were essential for securely hashing passwords and maintaining safe user sessions, improving overall application security.
   - **SHA Encryption**: You implemented SHA (Secure Hash Algorithm) encryption to securely store user passwords. This approach ensured that passwords were never stored in plaintext and were protected with cryptographic hashing.
   - **Cryptography**: You worked with the **Cryptography** library to enhance security by generating strong encryption keys for password storage. This included using secure methods like hashing and salting to prevent unauthorized access to sensitive data.
4. **User Authentication and Password Management:**
   - **User Authentication**: You built a secure authentication system using Flask's session management features, enabling users to register, log in, and manage their passwords safely. You learned how to validate user credentials, maintain secure sessions, and implement user-specific password vaults.
   - **Password Management Features**: You developed functionality for users to add, update, and delete passwords securely. This involved encryption and decryption processes to ensure that passwords remained protected during these actions.
5. **Frontend Development (HTML/CSS):**
   - **HTML/CSS**: You applied HTML and CSS to create the user interface for Lockzilla. This included building responsive and visually appealing pages, such as login forms, password management dashboards, and settings interfaces.
   - **Responsive Design**: You created responsive designs that adapt to different screen sizes (desktop, tablet, mobile), ensuring a seamless experience across devices.
6. **Password Health Check and Generation:**
   - **Password Health Check**: You implemented a password strength checker, allowing users to evaluate the strength of their passwords.
   - **Password Generation**: You created functionality to generate strong, random passwords. This required implementing algorithms that create passwords with sufficient complexity and length, adhering to modern security standards.
7. **Clipboard component**
   - **Clipboard Functionality:** You integrated a clipboard functionality to allow users to easily copy their passwords to the clipboard. This feature improves usability by providing a quick way for users to retrieve their passwords.

Conclusion & Future Work

Lockzilla is a secure and user-friendly password manager that combines web development and encryption to provide a seamless experience for managing sensitive credentials. Built with Flask for the backend, SQLite3 for database management, and cryptographic libraries like SHA and Werkzeug, Lockzilla ensures passwords are securely stored and never kept in plaintext. The application allows users to register, log in, and securely manage their passwords, while the responsive frontend, built with HTML and CSS, enhances the user experience across devices.

Key features include password generation, health checks, and clipboard functionality, encouraging users to maintain strong and secure passwords. User preferences and password data are securely stored and easily retrievable from the SQLite3 database.

Future Work:

To further enhance Lockzilla's capabilities, future work could involve adding multi-factor authentication (MFA) to increase the security of user logins. Another potential improvement is incorporating password strength meters with more advanced algorithms for better health checks, as well as supporting integration with third-party password security tools. Additionally, expanding Lockzilla's functionality to support cloud synchronization could allow users to access their passwords securely across multiple devices, creating a more flexible experience.

Future versions could also include features like password sharing with end-to-end encryption, auto-fill capabilities for web forms, and the ability to securely store other sensitive data such as credit card information or notes. To increase usability, Lockzilla could implement features like password reminders or recovery options in case of lost credentials. Integrating feedback loops to collect user input and improve the interface, as well as offering additional security analytics, would make the platform even more user-centric and interactive. These improvements would make Lockzilla a more comprehensive and robust password management tool, helping users stay secure while easily managing their digital lives.

REFERENCES

1. Documentation

- **Flask Documentation**: https://flask.palletsprojects.com/
- **SQLite3 Documentation**: https://docs.python.org/3/library/sqlite3.html
- **Werkzeug Documentation**: https://werkzeug.palletsprojects.com/
- **Cryptography Documentation**: https://cryptography.io/en/latest/

2. Online Tutorials and Guides

- **Flask for Beginners**: Flask Tutorials - GeeksforGeeks
- **Password Management in Python**: Password Management Systems - Real Python
- **Python Cryptography Tutorial**: Python Cryptography Tutorial on YouTube

3. Tools and Libraries

- **Flask**: Flask - Python Web Framework
- **SQLite3**: SQLite3 Database Management
- **Werkzeug**: Werkzeug Security Documentation
- **Cryptography**: Cryptography Python Library

4. Development Tools

- **Visual Studio Code (VS Code):** https://code.visualstudio.com/
- **SQLite Database Browser**: https://sqlitebrowser.org/
- **Postman** (for API testing): https://www.postman.com/

5. Password and Security References

- **OWASP Top Ten**: OWASP - Open Web Application Security Project
- **Password Strength Guidelines**: How to Create Strong Passwords - OWASP