SECURE PASSWORD GENERATOR

CTEC3451D: Development Project

Final Deliverable

Name: Bebin Regi

Student Number: P2703826

Word Count: 8261

Submission Date: 03/05/2024

Table of Contents

1. Abstract 6
2. Introduction 7
3. Literature review 7
3.1 Abstract 7
3.2 Introduction 8
3.3 The Achilles' Heel of Passwords: Inherent Weaknesses 8
3. 4 Automated Password Generators: Striking a Balance 8
3.5 Beyond Passwords: Exploring Multi-Factor Authentication 9
3.6 Knowledge-Based Authentication: Leveraging User Memory 9
3.7 Graphical Passwords: Beyond Text-Based Authentication 9
3.8 Browser Password Managers: Convenience with Caution 10
3.9 Beyond Traditional Guessing: Machine Learning and Password Security 10
3.10 A Behavioural Approach: Context-Aware Password Generation 11
3.11 Conclusion 11
3.12 Recommendations 12
4. Aim 13
4.1 User Registration and Authentication 13
4.2 Password Generation Customization 13
4.3 Password Saving and Tagging 13
4.4 Password Viewing 14
4.5 Frontend Development 14
4.6 Backend Development 14
4.7 Security and Data Protection 14
5 Problem 15
5.1 Security Vulnerabilities 15
5.2 Scalability Challenges 15
5.3 Usability Issues 15
5.4 Compatibility Challenges 16
5.5 Performance Optimization 16
5.6 Regulatory Compliance 16
6. Goal 17
6.1 Enhance Security Measures 17
6.2 Improve User Experience 17
6.3 Ensure Scalability and Performance 17
6.4 Ensure Compatibility and Accessibility 18

6.5 Maintain Regulatory Compliance 18
7. Functional Requirements Analysis 19
7.1 User Types 19
7.2 User Permissions and Functionality 19
7.2.1 Sign Up 19
7.2.2 Log In 19
7.2.3. Log Out 19
7.2.4 Generate Password 19
7.2.5 Password Strength: 20
7.2.6. View Generated Passwords: 20
7.3 System Use Cases 20
8. Non-functional Requirements Analysis 24
8.1 Security Requirements 24
8.2 Usability Requirements 25
8.3 Performance Requirements 25
9. Design 26
9.1 Software Development Methodology 26
9.1.1 Development Lifecycle 26
9.1.2 Application of Validation and Verification 26
9.2 Project Folder Structure 27
9.2.1 Backend Structure (backend-main) 28
9.2.2 Frontend Structure (frontend-main) 28
9.2.3 Additional Components 28
9.2.4 Version Control 28
9.3 Database Design 29
9.3.1 UserCredentials Table 29
9.3.2 PasswordData Table 29
9.3.3 Entity Relationship Diagram (ERD): 29
9.4 Data Flow Diagram (DFD) 30
10 System(implementation) 31
10.1 Backend System Implementation 31
10.1.2 API Endpoints 33
10.1.3 Password Generation Algorithm 35

10.1.4 AES Encryption and Decryption 36
10.2 Streamlit Implementation 37
10.3 System Usage 38
10.3.1 Navigation 38
10.3.2 Sign Up and Log In 38
..... 40
10.3.3 Password Generator: 40
10.3.4 View Passwords: 43
10.3.5 Logout: 43
11 Testing 44
11.1 Methodology 44
12. Critical evaluation 46
13. Conclusion 47
14 References 48
15. GitHub repository 50

Table of Figures

Figure 1: Git Repository Overview, ... 29
Figure 2: Database Tables Overview 31
Figure 3: Entity Relationship Diagram (ERD) 32
Figure 4: Data Flow Diagram (DFD) illustrating the flow of data within the application's system. 32
Figure 5: Swagger UI displaying the API endpoints and documentation 33
Figure 6: app.main.py initalizing FastAPI 33
Figure 7: User Signup Endpoint - Allows users to create a new account by providing their email address and password. 34
Figure 8: User Login Endpoint - Enables users to authenticate themselves and obtain an access token for accessing protected resources. 35
Figure 9: Password Generation Endpoint - Offers users the capability to generate strong, randomized passwords for their accounts. 35
Figure 10: Password Data Retrieval Endpoint - Enables users to retrieve their stored password data from the backend system. 36
Figure 11: Password Generation Algorithm 37

Figure 12: Successful Sign Up Notification 39
Figure 13: Error Message for Existing Email 39
Figure 14: Error Message for Invalid Credentials 40
Figure 15: Successful Log In Notification 40
Figure 16: Password Generation Interface: Weak Password 42
Figure 17: Password Generation Interface : Updating With Strong Password 42
Figure 18: Passwords Viewing Interface 43
Figure 19: Clicking Logout 44
Figure 20: Session Closed 44

Index of Tables

Table 1: Use-Case Look-up Table 23
Table 2: Testing Results Table 45

1. Abstract

Passwords remain the prevalent authentication method across digital landscapes, yet they suffer from inherent weaknesses. This review explores these shortcomings, including predictability, susceptibility to brute-force attacks, and ease of interception. Exploring the consequences of weak password selection and credential reuse underscores the importance of robust authentication mechanisms for individuals. The review then analyzes research on promising solutions, including automated password generators that balance complexity with memorability. Examining alternative authentication methods like multi-factor authentication and knowledge-based authentication. Also assessing the security features of browser password managers and explore potential improvements.

2. Introduction

Protecting sensitive data is crucial in the connected digital world of today. It is impossible to overestimate the significance of strong password security given the ongoing proliferation of cyber threats. Still, it can be difficult to come up with and maintain strong passwords, which frequently results in the adoption of poor practices that jeopardize security.

I've created an advanced password generator application with the goal of enhancing user experience and security to tackle this challenge. By enabling users to generate strong, one-of-a-kind passwords that are customized to their preferences, this application acts as a cornerstone in strengthening digital defences.

With the extensive feature set provided by the password generator, customers may alter characteristics like length, character kinds, and the addition of custom phrases to their passwords. This degree of personalization guarantees that generated passwords satisfy strict security specifications while still being easily remembered by users. Furthermore, the program puts a high priority on data security by utilizing cutting-edge encryption methods and safe storage procedures. The users' confidential credentials are protected from unwanted access by upholding the highest industry security standards.

Essentially, the password generator is an amalgam of state-of-the-art technology and user-focused design, with the goal of transforming password management for the modern digital era, hoping to redefine password security via painstaking attention to detail and steadfast dedication to security.

3. Literature review

3.1 Abstract

Passwords continue to dominate as the prevailing authentication method across digital landscapes, despite their inherent weaknesses. This review examines these shortcomings, such as predictability, vulnerability to brute-force attacks, and susceptibility to interception. The consequences of weak password selection and credential reuse are explored, emphasizing the necessity for robust authentication mechanisms. Additionally, the review scrutinizes research on promising solutions, including automated password generators that strike a balance between complexity and memorability. Alternative authentication methods, such as multi-factor authentication and knowledge-based authentication, are also analyzed. Finally, the

security features of browser password managers are assessed, along with potential avenues for improvement.

3.2 Introduction

Passwords serve as the cornerstone of user authentication in computer systems, acting as the primary gateway for user identification (Memon & Zhang, 2013). However, their vulnerability to various attack vectors jeopardizes system security, potentially leading to data breaches and other serious consequences. A significant challenge with traditional passwords lies in their susceptibility to brute-force attacks due to weak and predictable user choices (Wang et al., 2017). Phishing attacks further exacerbate the issue, with social engineering tactics easily tricking users into surrendering credentials through keyloggers or shoulder surfing (Adams & Aday, 2010). Therefore, employing strong and unique passwords is crucial to ensure adequate protection against these risks.

This literature review investigates current research on password security and explores promising solutions for the future of user authentication.

3.3 The Achilles' Heel of Passwords: Inherent Weaknesses

Text-based passwords possess several critical flaws that can jeopardize user security. The most prominent vulnerability is guessability. Users frequently gravitate towards predictable patterns like "123456," "password," or keyboard sequences like "qwerty," significantly enhancing the attacker's success rate, especially with automated tools that can attempt millions of combinations within seconds (Jermyn et al., 2017; Urbanski et al., 2017).

Another critical weakness is replayability. Since compromised password hashes can be effortlessly replayed, attackers can launch offline attacks even after a password change (Turner et al., 2020). Text-based passwords are also susceptible to eavesdropping. Unencrypted passwords can be overheard with ease. If an attacker intercepts the password transmission during login, they can view it in plain text (Sun et al., 2003). Phishing scams exploit this vulnerability through social engineering tactics, tricking users into revealing their credentials on fraudulent websites (Jagatic et al., 2007).

3. 4 Automated Password Generators: Striking a Balance

For many users, manually creating secure passwords poses a significant challenge. The password needs to be both difficult to guess and complex enough to be secure. Automated password generators offer a solution to this problem. These tools function by randomly combining sets of characters to form passwords. However, advanced generators leverage more sophisticated methods to create passwords that are both secure and memorable.

Some generators, like AutoPass, achieve this by employing pronounceable words and mnemonic phrases, making passwords easier to recall (Al-Maqbali & Mitchell, 2017). Another approach involves incorporating configurable user input. This technique allows users to personalize their passwords, resulting in unique and non-guessable credentials. This flexibility empowers users to strike a balance between password strength and memorability (Glory et al., 2019). Additionally, research by Reynolds et al. (2018) explores the concept of context-aware password generation, where the system suggests passwords based on the website or service being accessed.

3.5 Beyond Passwords: Exploring Multi-Factor Authentication

The limitations of traditional passwords have led to the development of more robust authentication methods. Multi-factor authentication (MFA) adds an extra layer of security by requiring users to provide two or more verification factors in addition to their password (Rahman et al., 2019). These factors can include:

- **Something you know:** This could be a PIN, a security question, or a passphrase.
- **Something you have:** This could be a smartphone, a security token, or a hardware key.
- **Something you are:** This could be fingerprint recognition, facial recognition, or iris recognition.

  MFA significantly reduces the risk of unauthorized access, even if an attacker obtains a user's password. Research by Bao et al. (2014) demonstrates the effectiveness of MFA in mitigating phishing attacks.

3.6 Knowledge-Based Authentication: Leveraging User Memory

Knowledge-based authentication (KBA) is another alternative to traditional passwords. This approach relies on users answering pre-defined questions based on personal information (Brostoff & Sasse, 2000). However, KBA suffers from limitations. Users may forget answers to security questions, and the information used can be vulnerable to social engineering attacks. Additionally, research by Ayoub et al. (2018) suggests that attackers can leverage social media profiles and other online footprints to answer KBA questions successfully. Consequently, KBA should be used cautiously and combined with other authentication methods for enhanced security.

3.7 Graphical Passwords: Beyond Text-Based Authentication

Graphical passwords represent an alternative authentication method that relies on images or graphics instead of text-based passwords. There are several techniques within graphical passwords, offering increased resistance against brute-force attacks:

- **Click-based methods:** Users select specific points within an image or click a sequence of images in the correct order (De Luca et al., 2004; Santoso et al., 2023).
- **Drawing methods:** Users recreate complex drawings from memory as their passwords (De Luca et al., 2004).
- **Passphrases with image selection:** Users choose a memorable phrase and associate it with specific images in a predefined order (Cho et al., 2013).

These visual approaches offer a more intuitive authentication experience compared to text-based passwords. However, concerns regarding user shoulder surfing and the potential for smudge attacks on touchscreens necessitate further investigation (Man et al., 2020).

3.8 Browser Password Managers: Convenience with Caution

Modern web browsers offer integrated password managers that securely store login credentials for various websites. While convenient, recent research by Oesch & Ruoti (2019) has revealed vulnerabilities in their autofill functionality. These managers may populate login credentials into phishing websites that mimic legitimate ones, bypassing authentication checks. Disabling autofill can mitigate this risk, but it diminishes usability and may encourage unsafe practices like writing down passwords. Implementing mechanisms that verify domain legitimacy before autofilling credentials can significantly enhance the security of password managers.

3.9 Beyond Traditional Guessing: Machine Learning and Password Security

The realm of password security extends beyond traditional dictionary attacks and brute-force methods. With advancements in machine learning, attackers are exploring sophisticated techniques to predict user-chosen passwords. One such approach leverages Long Short-Term Memory (LSTM) networks, a type of deep neural network adept at recognizing patterns in sequential data (Pasquini et al., 2021).

LSTM networks excel in language processing tasks, making them well-suited for analyzing password character sequences. Research by Pasquini et al. (2021) demonstrates the effectiveness of LSTM-based models in predicting password characters. This technique surpasses traditional Markov models in accuracy by a significant margin. Furthermore, it requires minimal data preparation and outperforms baseline machine learning classifiers. The successful application of LSTMs underscores the critical need for continuously evolving password security measures. Further exploration with diverse keyboard layouts and password representations could yield valuable insights into user behaviour and password selection patterns (Pasquini et al., 2021).

3.10 A Behavioural Approach: Context-Aware Password Generation

While robust password creation is essential, user behaviour plays a crucial role in password security. Traditional password managers often struggle to strike a balance between usability and security. Users may prioritize convenience and opt for weaker, more memorable passwords, jeopardizing account safety.

A recent study by Reynolds et al. (2018) proposes a context-aware password generation system that incorporates user preferences and behaviours. This system analyzes user interactions with various websites and services and suggests contextually relevant passwords during the registration process. For instance, the system might recommend a more complex password for a financial institution compared to a social media platform. This approach empowers users with passwords tailored to specific security needs, potentially mitigating the trade-off between memorability and strength.

By understanding user behaviour and adapting password suggestions accordingly, context-aware generation offers a promising avenue for enhancing overall password security (Reynolds et al., 2018).

3.11 Conclusion

Despite inherent limitations, passwords remain the frontline defence for user authentication in the digital realm. This review has examined the vulnerabilities of traditional password schemes, including susceptibility to brute-force attacks, predictability, and ease of interception. I've emphasized the consequences of weak password selection and credential reuse, highlighting the need for more robust authentication mechanisms. The review has also explored promising solutions that hold the potential to revolutionize user authentication.

The quest for robust and user-friendly authentication solutions remains an ongoing pursuit in the ever-evolving technological landscape. While passwords continue to play a crucial role, advancements in various areas offer promising avenues for a future where user convenience and robust security coexist.

Automated password generators offer a practical solution for users struggling to create strong yet memorable passwords. These tools leverage advanced algorithms to generate complex combinations while prioritizing user recall. Sophisticated generators may incorporate user input or employ pronounceable words and mnemonic phrases, achieving a balance between security and usability. Context-aware password generation, explored by Reynolds et al. (2018), suggests tailoring password suggestions to specific online services, further enhancing overall security.

Multi-factor authentication (MFA) transcends passwords, adding an extra layer of defence by requiring additional verification factors beyond a password (Rahman et al., 2019). These factors can encompass something a user knows (PIN, security question),

something they have (smartphone, security token), or something they are (fingerprint, facial recognition). By requiring more than just a password, MFA significantly reduces the risk of unauthorized access, even if an attacker manages to steal a user's password.

Graphical passwords offer an alternative authentication method that utilizes images or graphics instead of text-based passwords. While offering a more intuitive user experience, concerns regarding shoulder surfing and smudge attacks necessitate further investigation (Man et al., 2020). Click-based methods involve selecting specific points within an image or clicking a sequence of images (De Luca et al., 2004; Santoso et al., 2023). Drawing methods require recreating complex drawings from memory (De Luca et al., 2004). Passphrases with image selection combine memorable phrases with the selection of specific images (Cho et al., 2013).

Browser password managers, a ubiquitous feature in modern web browsers, offer convenient storage for login credentials across various websites. However, recent research by Oesch & Ruoti (2019) has revealed vulnerabilities in their autofill functionality. These managers may populate login credentials into phishing websites that mimic legitimate ones, bypassing authentication checks. Disabling autofill can mitigate this risk, but it comes at the cost of reduced usability and might encourage unsafe practices like writing down passwords. Implementing mechanisms that verify domain legitimacy before autofilling credentials can significantly enhance the security of password managers.

As it move forward, continuous research and development are imperative to ensure the security of the digital identities in a world increasingly reliant on online interactions. The ideal future authentication solution will seamlessly integrate robust security measures with a user-friendly experience, empowering users to navigate the digital landscape with confidence.

3.12 Recommendations

- **Embrace Automated Password Generation:** Leverage password generators to create strong, unique passwords for each online account. Prioritize tools that balance security with memorability through features like user input or pronounceable words.
- **Fortify with Multi-Factor Authentication:** Whenever possible, enable multi-factor authentication (MFA) to add an extra layer of security beyond just a password. Consider options like fingerprint scanners, security tokens, or verification codes sent to your phone.
- **Explore Graphical Passwords (with Caution):** For specific accounts, graphical passwords offer a potentially more intuitive user experience. However, be mindful of surrounding privacy concerns and choose platforms that address smudge attack vulnerabilities.

- **Use Password Managers Cautiously:** Browser password managers offer convenience, but prioritize security. Disable autofill on untrusted websites and consider using managers with domain verification features before autofilling credentials.

4. Aim

The project's objective is to offer a web application that lets users securely and conveniently manage their passwords. The program seeks to improve user experience and expedite password management through the use of strong security mechanisms and intuitive features. In particular, the goals consist of:

4.1 User Registration and Authentication

- Develop a secure user registration system allowing users to create accounts.
- Implement JWT authentication to ensure secure verification of user identity.
- Utilize hashing algorithms to protect sensitive user information such as passwords.
- Employ token-based authentication for seamless and secure user login sessions.

4.2 Password Generation Customization

- Create functionality for users to customize password parameters.
- Allow users to specify password length, inclusion of uppercase letters, special characters, and the option to add a custom word.
- Design a user-friendly interface for effortless customization of password settings.
- Implement validation checks to ensure the generated passwords meet specified criteria for strength and complexity.

4.3 Password Saving and Tagging

- Enable users to save generated passwords securely within the application.
- Implement tagging functionality to allow users to categorize and identify saved passwords easily.

- Develop storage mechanisms using PostgreSQL to securely store passwords and associated tags.
- Ensure data integrity and confidentiality through encryption and access control measures.

4.4 Password Viewing

- Create a feature that allows users to view their saved passwords within the application.
- Implement data retrieval mechanisms to fetch stored passwords from the database.
- Display passwords to the user in a user-friendly and organized manner, providing options for sorting and filtering.

4.5 Frontend Development

- Utilize Streamlit framework to develop an intuitive and visually appealing frontend interface.
- Design the frontend interface to enhance user interaction and experience.
- Ensure responsiveness and accessibility across various devices and screen sizes.
- Implement smooth transitions and animations to improve user engagement.

4.6 Backend Development

- Utilize FastAPI framework to develop a strong backend server.
- Implement endpoints for user authentication, password generation, and database interactions.
- Develop RESTful APIs to facilitate communication between the frontend and backend components.
- Optimize backend code for efficiency and scalability, ensuring smooth application performance.

4.7 Security and Data Protection

- Prioritize encryption methods to safeguard private data, including user credentials and passwords.

- Implement secure transmission protocols (HTTPS) to protect data during transfer.
- Adhere to recommended security procedures for data storage, including encryption-at-rest and access control measures.
- Regularly audit and update security protocols to address emerging threats and vulnerabilities.

5 Problem

5.1 Security Vulnerabilities

- Identify potential security vulnerabilities within the application's architecture and codebase.
- Assess risks associated with user authentication, password storage, and data transmission.
- Address common security threats such as SQL injection, cross-site scripting (XSS), and authentication bypass.
- Implement security best practices to mitigate vulnerabilities and protect user data from unauthorized access or manipulation.

5.2 Scalability Challenges

- Evaluate the application's scalability to handle increasing user loads and data volumes.
- Identify potential bottlenecks in the system architecture that may hinder scalability.
- Consider strategies for horizontal and vertical scaling to accommodate future growth.
- Implement caching mechanisms, load balancing, and database optimization techniques to improve scalability and performance.

5.3 Usability Issues

- Identify usability issues that may impact user experience and adoption of the application.
- Evaluate the clarity and intuitiveness of the user interface for password generation and management.

- Gather feedback from users through usability testing and surveys to identify pain points and areas for improvement.
- Iterate on the design and functionality of the application to enhance usability and user satisfaction.

5.4 Compatibility Challenges

- Assess compatibility issues that may arise across different devices, browsers, and operating systems.
- Test the application on various platforms to identify compatibility issues and inconsistencies.
- Address compatibility issues through responsive design, cross-browser testing, and compatibility libraries.
- Ensure a consistent user experience across all supported platforms to maximize accessibility and usability.

5.5 Performance Optimization

- Evaluate the performance of the application under various conditions, including peak usage and heavy loads.
- Identify performance bottlenecks in the frontend and backend components of the application.
- Implement optimization techniques such as code refactoring, database indexing, and caching to improve performance.
- Monitor application performance metrics and conduct regular performance testing to ensure optimal performance.

5.6 Regulatory Compliance

- Assess compliance requirements related to data privacy and security regulations, such as GDPR, CCPA, and HIPAA.
- Implement measures to ensure compliance with relevant regulatory standards and guidelines.
- Develop data protection policies and procedures to safeguard user privacy and confidentiality.

- Conduct regular audits and assessments to verify compliance and address any non-compliance issues promptly.

6. Goal

6.1 Enhance Security Measures

- Implement robust security features to protect user data, including secure user authentication, encrypted password storage, and secure data transmission.
- Integrate JWT authentication to verify user identity and prevent unauthorized access to sensitive information.
- Employ encryption techniques to safeguard passwords stored in the database and ensure confidentiality.
- Implement security best practices to mitigate common vulnerabilities such as SQL injection, cross-site scripting (XSS), and authentication bypass.

6.2 Improve User Experience

- Enhance the user interface and user experience (UX) of the application to make password generation and management intuitive and user-friendly.
- Streamline the password generation process by providing customizable options for password parameters such as length, character types, and inclusion of custom words.
- Implement features for saving and tagging generated passwords to facilitate easy retrieval and organization.
- Conduct usability testing and gather user feedback to identify areas for improvement and optimize the overall user experience.

6.3 Ensure Scalability and Performance

- Design the application architecture to be scalable and capable of handling increasing user loads and data volumes.
- Implement performance optimization techniques to improve the responsiveness and efficiency of the application.

- Utilize caching mechanisms, load balancing, and database optimization strategies to enhance scalability and performance.
- Monitor application performance metrics and conduct regular performance testing to identify and address performance bottlenecks.

6.4 Ensure Compatibility and Accessibility

- Ensure compatibility across various devices, browsers, and operating systems to maximize accessibility and reach.
- Implement responsive design principles to ensure the application adapts seamlessly to different screen sizes and resolutions.
- Conduct thorough testing across multiple platforms to identify and address compatibility issues.
- Ensure adherence to accessibility standards and guidelines to make the application accessible to users with disabilities.

6.5 Maintain Regulatory Compliance

- Ensure compliance with relevant data privacy and security regulations, such as GDPR, CCPA, and HIPAA.
- Develop and implement data protection policies and procedures to safeguard user privacy and comply with regulatory requirements.
- Conduct regular audits and assessments to verify compliance and address any non-compliance issues promptly.
- Stay updated on changes to regulations and standards and adapt the application accordingly to maintain compliance.

7. Functional Requirements Analysis

This document outlines the functional requirements for a password generator application. The application aims to provide users with a secure and convenient way to generate strong passwords.

7.1 User Types

The application will utilize a single user type: **Bas**e **User**.

7.2 User Permissions and Functionality

7.2.1 Sign Up

- Users can create a new account by providing an email address and password.
- The system should validate email format and password strength during registration.
- Upon successful registration, the system should generate a JWT containing user information (e.g., user ID) and a secret key.
- The JWT token is sent back to the user for secure storage.

7.2.2 Log In

- Users can log in to their accounts using their registered email address and password.
- The system validates the credentials against stored data.
- Upon successful login, the system generates a new JWT and sends it back to the user.

7.2.3. Log Out

- Users can log out of their accounts to terminate their session.
- The application should clear the JWT from the user's storage.

7.2.4 Generate Password

- Users can access password generation functionalities only after successful authentication (valid JWT).
- Users can initiate password generation by specifying the desired password length.
- Users can optionally choose to include:
    - Uppercase letters
    - Lowercase letters
    - Digits

        - Special characters
        - A custom word
- Users can add a tag to the generated password for easy identification (optional).
- The application generates a random password based on the user's selections.
- The generated password is displayed to the user.

7.2.5 Password Strength:

- The application calculates and displays the strength of the generated password.

7.2.6. View Generated Passwords:

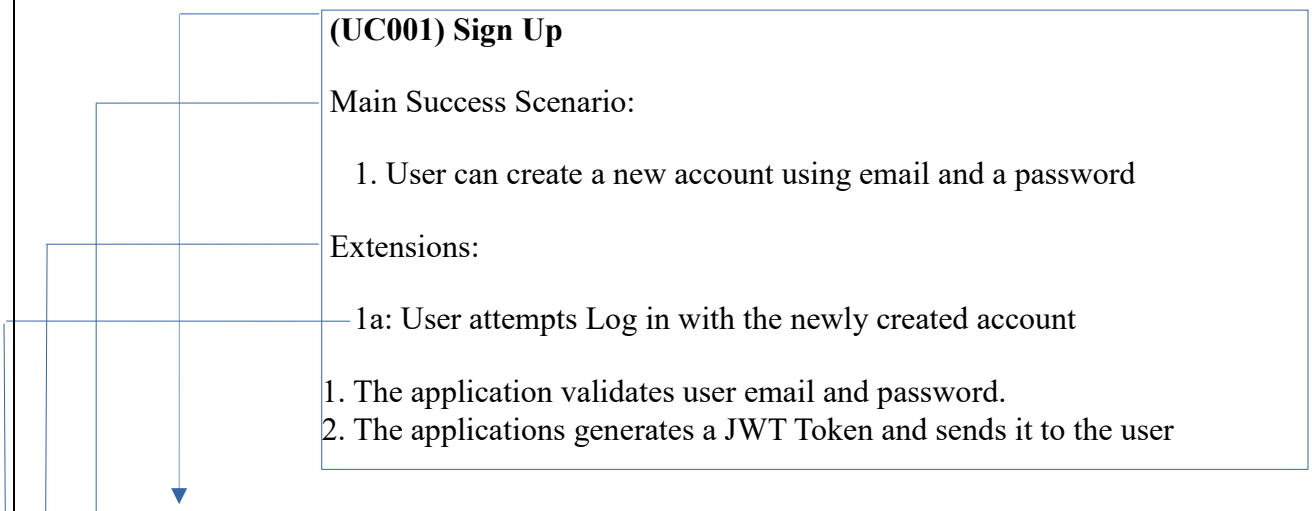
- Users can view a history of all generated passwords within their account.
- The application sends a request to the server with the user's JWT for authorization.
- The server validates the JWT before allowing access to user data.
- The password list displays each password, its corresponding strength, and any associated tag.

7.3 System Use Cases

This section will start by providing a breakdown of each user's use-cases via tables. These tables will also include an evaluation of the priority of each use-case for every user type. These tables will be followed by a compilation of use-cases documented in a uniform format and separated via user-type.

A sample annotated use-case can be seen below which is aimed to provides an overview of the

Structure of how these are documented, how to read them, and what information they contain.



Numeric identifier key that is unique to each use-case and descriptive title which outlines its purpose.

The **main success scenario** is the main path of events that should occur for the action in the
use-case to succeed.

The **extensions are alternative pathways** that may occur if the main path was to encounter
a failure at any stage. A description of the alternative route and a granular breakdown of
what occurs in the scenario.

The extension identifier's number relates to the step within the main success scenario at
which the alternative extension route can occur. In this example, extension '1a' is the first
alternative extension of the main success scenario's step 2. The letter 'a' in this identifier is
incremented as more extensions for step 1 are added

Use-case Lookup Table

The below table displays an overview of all the use-cases in the system and can be used as a look-up guide.

Table 1: Use-Case Look-up Table

Use-Case Look-up Table			
	Use-Case ID	**Use-Case Description**	**Priority**
Base User	UC001	Sign Up	High
	UC002	Log In	High
	UC003	Generate Password	Medium
	UC004	View Generated Passwords	Low
	UC005	Log Out	Low

Base User

UC001: Sign Up

Main Success Scenario:

- User opens the password generator application.
- User clicks the "Sign Up" radio option.
- User enters a valid email address.
- User creates a strong password.
- User clicks the "Sign Up" button.
- The application validates the user's input (email, password).
- Upon successful validation, the application creates a new user account and stores the user's credentials securely using hashing or encryption techniques.
- The application generates a JWT (JSON Web Token) containing user information
- The JWT is sent back to the user's device for secure storage.
- The application logs the user in and redirects them to the password generation interface.

Extensions:

3a: Email address is already in use.

- System notifies the user that the email address is already associated with an existing account.
- User can choose to enter a different email address or proceed to log in if they already have an account.

3b: Invalid user input (e.g. invalid email format).

- System prompts the user to correct the invalid input fields.
- User can edit their input and try registering again.

UC002: Log In

Main Success Scenario:

- User opens the password generator application.
- User clicks the "Log In" radio option.
- User enters their registered email address.
- User enters their password.
- User clicks the "Log In" button.
- The application validates the user's credentials against stored data.
- Upon successful validation, the application generates a new JWT.
- The JWT is sent back to the user's device for secure storage.
- The application redirects the user to the password generation interface.

Extensions:

3a: Invalid login credentials.

- System notifies the user that the email address or password is incorrect.
- User can retry logging in with the correct credentials.

UC003: Generate Password

Main Success Scenario:

- User is logged in (valid JWT).
- User choses a desired password length.
- User selects options for password composition (uppercase letters, lowercase letters, digits, symbols, custom word).
- (Optional) User enters a custom word to include in the password.
- (Optional) User enters a tag to associate with the generated password.
- User clicks the "Generate Password" button.
- The application generates a random password based on the user's selections.
- The generated password is displayed to the user.
- The application calculates and displays the password strength.

UC004: View Generated Passwords (Optional)

- User is logged in (valid JWT).
- User clicks the "View Passwords" option (if available).
- The application sends a request to the server with the user's JWT for authorization.
- The server validates the JWT and retrieves the user's password history if authorized.
- The application displays a list of all generated passwords, their corresponding strengths, and any associated tags.

UC005: Log Out

- User clicks the "Log Out" button.
- The application clears the user's JWT from storage.
- The application redirects the user to the application's main page or login interface.

8. Non-functional Requirements Analysis

8.1 Security Requirements

To ensure the security of the password generation system, the following requirements must be met:

- **Encryption Mechanisms**: Implement strong encryption techniques to protect sensitive data, including user credentials and generated passwords. Utilize industry-standard encryption algorithms such as AES (Advanced Encryption Standard) for data storage and transmission.
- **Secure Authentication**: Implement secure authentication mechanisms, such as JWT (JSON Web Tokens), to verify user identities and prevent unauthorized access to the system. Employ techniques like password hashing and salting to securely store user passwords in the database.
- **Protection against Attacks**: Incorporate defences against common security threats, including brute-force attacks, SQL injection, and cross-site scripting (XSS). Implement input validation and sanitization measures to mitigate the risk of injection attacks. Utilize rate limiting and CAPTCHA mechanisms to prevent automated attacks.

8.2 Usability Requirements

To ensure a seamless and intuitive user experience, the following usability requirements must be considered:

- **Intuitive Interface Design**: Design a user-friendly interface that guides users through the password generation process smoothly. Use clear instructions, visual cues, and feedback messages to assist users in customizing and generating passwords.
- **Accessible Features**: Ensure that the system is accessible to users with disabilities by adhering to web accessibility standards (WCAG). Provide alternative text for images, keyboard navigation support, and high contrast modes to accommodate users with visual impairments or motor disabilities.
- **Responsive Design**: Develop a responsive design that adapts to various screen sizes and devices, including desktops, laptops, tablets, and smartphones. Ensure that all features and functionalities are accessible and optimized across different platforms.

8.3 Performance Requirements

To maintain optimal performance under varying conditions, the following performance requirements must be met:

- **Efficient Password Generation**: Optimize the password generation algorithm to ensure fast and efficient generation of strong passwords. Minimize computational overhead and resource usage to deliver quick response times even during peak usage periods.
- **Scalability**: Design the system to be scalable to accommodate growing user demands and data volumes. Implement horizontal scaling techniques such as load balancing and clustering to distribute workload across multiple servers and handle increased traffic effectively.
- **Database Optimization**: Optimize database performance by employing indexing, query optimization, and caching strategies. Ensure that database queries are optimized for speed and efficiency to minimize latency and response times.

9. Design

9.1 Software Development Methodology

For the development of the secure password generator application, I adopted an agile software development methodology. Agile was chosen due to its flexibility, allowing us to adapt to changing requirements and priorities throughout the development lifecycle, which is essential in a dynamic domain like cybersecurity. The Agile approach was further refined with elements of the Kanban framework, enabling us to efficiently manage tasks and ensure steady progress.

9.1.1 Development Lifecycle

The development lifecycle consisted of iterative stages, each focusing on specific aspects of the project while incorporating validation and verification processes to maintain quality and security standards.

1. **Planning and Requirements Gathering**: At the outset, I conducted comprehensive research to identify user requirements and security needs. This phase involved collaborating with stakeholders to define project objectives, user stories, and acceptance criteria. Validation occurred through continuous feedback loops to ensure alignment with user expectations and security standards.
2. **Design and Architecture**: With a clear understanding of requirements, I proceeded to design the system architecture, database schema, and API endpoints. Verification processes, including code reviews and architectural assessments, were conducted to validate the design against security best practices and scalability requirements.
3. **Implementation**: The implementation phase involved the actual coding of backend systems using FastAPI and frontend interfaces using Streamlit. Throughout this stage, rigorous testing, including unit tests and integration tests, was performed to verify the functionality and security of each component.

9.1.2 Application of Validation and Verification

Validation and verification were integral components at each stage of the development lifecycle:

- **Requirements Validation**: User stories and acceptance criteria were validated with user needs and cybersecurity requirements.
- **Design Verification**: Code reviews, architectural assessments, and threat modelling were conducted to verify the security and scalability of the system design.

- **Implementation Testing**: Unit tests, integration tests, and security tests were performed during implementation to verify the correctness and security of the codebase.
- **Quality Assurance**: Functional testing were carried out to validate the application's functionality.

By applying rigorous validation and verification practices at each stage, I ensured the development of a secure and reliable password generator application that also meets cybersecurity standards

9.2 Project Folder Structure

Incorporating version control with Git is fundamental for efficient collaboration and project management. Below is an outline of the version control setup and an image illustrating the project's Git repository:

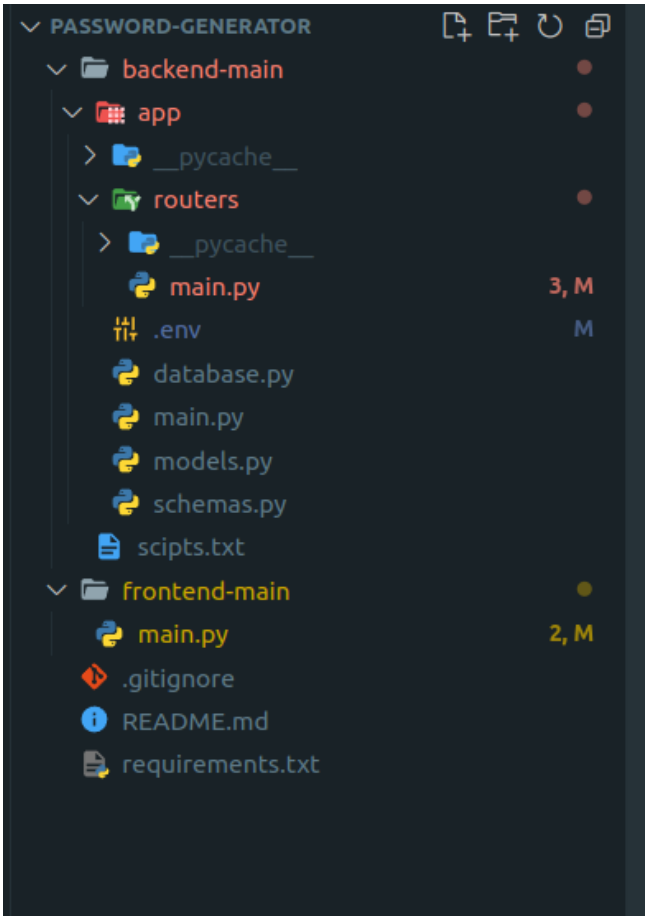


Figure 1: Git Repository Overview,

9.2.1 Backend Structure (backend-main)

The backend structure follows the FastAPI framework for building APIs, ensuring efficient and scalable development. Below is a breakdown of the folders and files within the backend-main directory:

app: This directory contains the core logic and components of the backend application.

routers: Subdirectory housing route definitions for different API endpoints.

main.py: Entry point for the backend application, where the FastAPI instance is created and configured.

database.py: File containing code for interacting with the PostgreSQL database, including database engine setup.

models.py: Definitions for data models representing tables in the database.

schemas.py: Data schema definitions used for data validation and serialization.

scripts.txt: Additional scripts or notes relevant to backend development.

.env: Environment file containing sensitive information such as PostgreSQL credentials.

9.2.2 Frontend Structure (frontend-main)

The frontend structure currently consists of a single file made in streamlit framework.

main.py: The frontend application, handling client-side rendering and interactions.

9.2.3 Additional Components

requirements.txt: This file enumerates all Python dependencies necessary to run the project. It ensures consistency in development environments and facilitates dependency management.

.gitignore: Configuration file specifying files and directories to be ignored by version control systems such as Git. The .env file containing sensitive credentials is typically added to .gitignore to prevent accidental exposure.

9.2.4 Version Control

The project is maintained within a Git repository, allowing for version control. The .gitignore file ensures that sensitive data, such as PostgreSQL credentials in the .env file, is not included in the repository.

9.3 Database Design

The database design for the application consists of two main tables: "user_credentials" and "password_data".

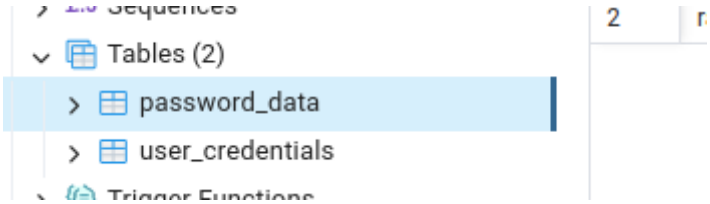


Figure 2: Database Tables Overview

9.3.1 UserCredentials Table

This table stores user credentials, with each entry identified by a unique email address (primary key). It contains columns for the user's email and password.

9.3.2 PasswordData Table

The "PasswordData" table holds password-related data, including a unique identifier (id) for each record. It includes a foreign key (email) referencing the email column in the "UserCredentials" table to establish the relationship between users and their password data. The "tag_password" column stores password information in JSON format, allowing flexibility in storing various attributes associated with each password entry. Each encrypted password is encrypted using AES.

This database design facilitates efficient storage and retrieval of user credentials and associated password data, supporting the functionality of the application.

9.3.3 Entity Relationship Diagram (ERD):

The ER diagram below illustrates the relationship between the "user_credentials" and "password_data" tables:

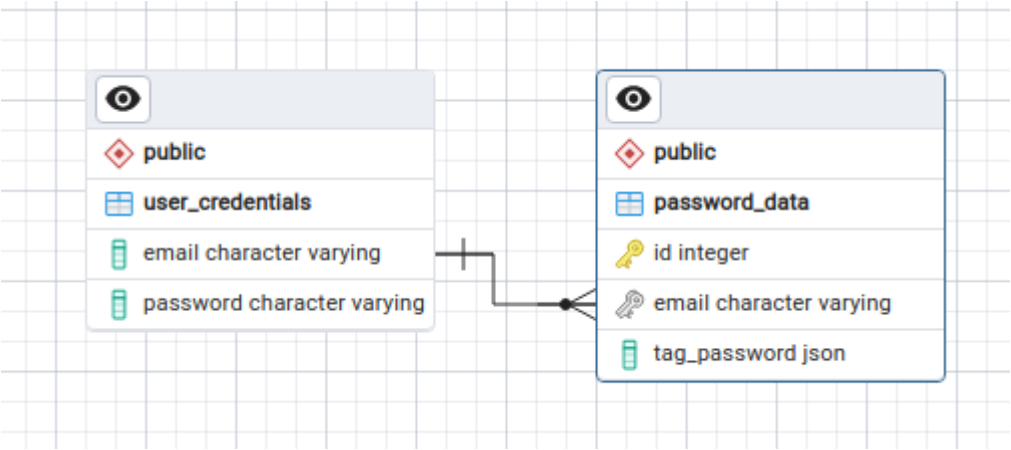


Figure 3: Entity Relationship Diagram (ERD)

9.4 Data Flow Diagram (DFD)

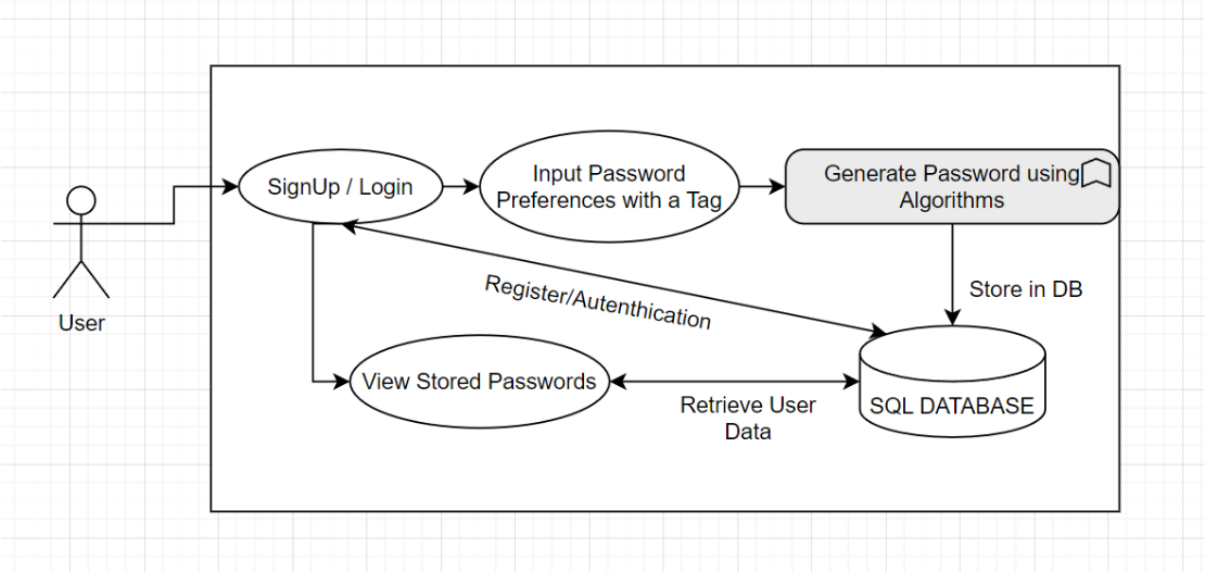


Figure 4: Data Flow Diagram (DFD) illustrating the flow of data within the application's system.

10 System(implementation)

10.1 Backend System Implementation

The backend system of the application is built using FastAPI, a modern, fast (high-performance), web framework for building APIs with Python 3.7+.

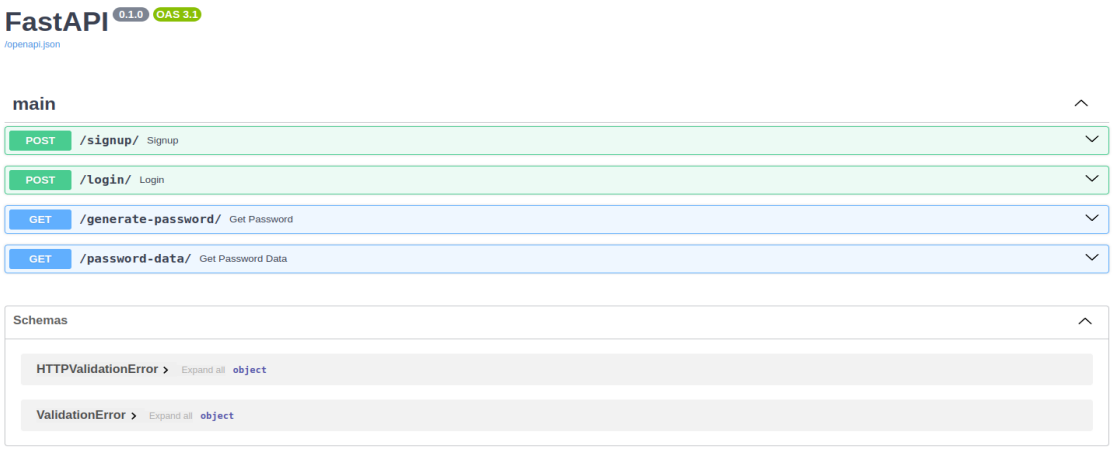


Figure 5: Swagger UI displaying the API endpoints and documentation

10.1.1 FastAPI Initialization: Initializes a FastAPI instance.

- **CORS Middleware**: Configures Cross-Origin Resource Sharing (CORS) middleware to allow requests from any origin.
- **Database Initialization**: Creates database tables defined in the models.
- **Router Inclusion**: Includes the main router from the routers module to handle API endpoints.

```
You, last week | 1 author (You)
from fastapi import FastAPI
from . import models
from .database import engine
from .routers import main      You, last week • first

from fastapi.middleware.cors import CORSMiddleware


app = FastAPI()

app.add_middleware(
    CORSMiddleware,
    allow_origins=['*'],
    allow_credentials=True,
    allow_methods=['*'],
    allow_headers=['*'],
)

models.Base.metadata.create_all(engine)

app.include_router(main.router)
```

Figure 6: app.main.py initializing FastAPI

10.1.2 API Endpoints

This section describes the four main API endpoints implemented in the backend system, each serving a specific purpose.

User Signup Endpoint

The user signup endpoint (/signup/) allows users to create a new account by providing their email address and password. Upon receiving the signup request, the backend system checks if the user already exists in the database. If the user does not exist, their password is securely hashed and stored in the database, along with their email address. This endpoint ensures that each user has a unique account within the application.

```python
@router.post("/signup/")
def signup(email: str, password: str, db: Session = Depends(get_db)):
    if db.query(models.UserCredentials).filter(models.UserCredentials.email == email).first():
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="User already exists")
    hashed_password = pwd_context.hash(password)
    new_user = models.UserCredentials(email=email, password=hashed_password)
    db.add(new_user)
    db.commit()      You, last week • first
    return {"message": True}
```

Figure 7: User Signup Endpoint - Allows users to create a new account by providing their email address and password.

User Login Endpoint

The user login endpoint (/login/) enables users to authenticate themselves and obtain an access token for accessing protected resources within the application. Users provide their email address and password, which are then validated against the stored credentials in the database. If the provided credentials are valid, the backend system generates a JSON Web Token (JWT) as an access token, which is returned to the user. This token can be used for subsequent requests to authorized endpoints, allowing the user to access protected resources securely.

```python
@router.post("/login/")
def login(email: str, password: str, db: Session = Depends(get_db)):

    user = db.query(models.UserCredentials).filter(models.UserCredentials.email == email).first()
    if user is None or not pwd_context.verify(password, user.password):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid credentials")

    # Generate JWT token
    access_token_expires = timedelta(minutes=ACCESS_TOKEN_EXPIRE_MINUTES)
    access_token = create_access_token(
        data={"sub": user.email}, expires_delta=access_token_expires
    )

    return {"message": True,"access_token": access_token, "token_type": "bearer"}
```

Figure 8: User Login Endpoint - Enables users to authenticate themselves and obtain an access token for accessing protected resources.

Password Generation Endpoint

The password generation endpoint (/generate-password/) offers users the capability to generate strong, randomized passwords for their accounts. Users can specify the desired length of the password, along with any custom words or character types (uppercase letters, lowercase letters, digits, special characters) to include or exclude. Upon successful generation of the password, it is associated with a user's email address and a specified tag, allowing users to organize and manage their passwords efficiently.

```python
@router.get("/generate-password/")
def get_password(token: str = '', length: int = 8, custom_word: str = '',
                 include_uppercase: bool = True, include_lowercase: bool = True,
                 include_digits: bool = True, include_special: bool = True,tag: str = '',db: Session = Depends(get_db)):
    if not verify_token(token):
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Invalid token")
    password = generate_password(length, custom_word, include_uppercase, include_lowercase, include_digits, include_special)
    e_password=encrypt_sp(str(password),key)
    email = decode_token(token)

    existing_password_data = db.query(models.PasswordData).filter(models.PasswordData.email == email).first()

    if existing_password_data:
        existing_tag_password = existing_password_data.tag_password
        #convert to normal json
        existing_tag_password[tag] = e_password
        # Update the tag_password attribute of the PasswordData instance
        existing_password_data.tag_password = existing_tag_password

        stmt = (
            update(models.PasswordData).
            where(models.PasswordData.email == email).
            values(tag_password=existing_tag_password)
        )
        db.execute(stmt)
        db.commit()
        return {"tag":tag,"password":password}
    # If there is no existing data, create a new entry
    new_password_data = models.PasswordData(email=email, tag_password={tag: e_password})
    db.add(new_password_data)
    db.commit()
    db.refresh(new_password_data)
    return {"tag":tag,"password":password}
```

Figure 9: Password Generation Endpoint - Offers users the capability to generate strong, randomized passwords for their accounts.

Password Data Retrieval Endpoint

The password data retrieval endpoint (/password-data/) enables users to retrieve their stored password data from the backend system. Users provide a valid JWT token as authentication to access their password data. Once authenticated, the backend system retrieves the password data associated with the user's email address and returns it in JSON format. This endpoint facilitates users in accessing and managing their stored passwords securely.

```python
@router.get("/password-data/")
def get_password_data(token: str, db: Session = Depends(get_db)):

    email = decode_token(token)

    password_data = db.query(models.PasswordData).filter(models.PasswordData.email == email).first()

    if not password_data:
        raise HTTPException(status_code=404, detail="No password data found for the user")
    data=password_data.tag_password
    for i in data:
        data[i]=decrypt_sp(data[i],key)
    return data
```

Figure 10: Password Data Retrieval Endpoint - Enables users to retrieve their stored password data from the backend system.

10.1.3 Password Generation Algorithm

The generate_password function is responsible for creating strong and randomized passwords based on specified parameters. Here's an overview of how the algorithm works:

1. **Character Set Construction**: The function constructs a character set based on the specified parameters:
    - include_uppercase: Determines whether uppercase letters are included.
    - include_lowercase: Determines whether lowercase letters are included.
    - include_digits: Determines whether digits (0-9) are included.
    - include_special: Determines whether special characters (e.g., punctuation marks) are included.
2. **Custom Word Handling**: If a custom word is provided, the function removes its characters from the character set to prevent conflicts.
3. **Password Generation**:
    - If the desired length of the password is less than or equal to the length of the custom word, the custom word is returned as the password.

- Otherwise, a random index is selected within the length of the password minus the length of the custom word. The custom word is inserted at this index to ensure its inclusion in the generated password.
- The remaining characters of the password are randomly chosen from the constructed character set.

4. **Return**: The function returns the generated password.

```python
def generate_password(length: int, custom_word: str = '',
                      include_uppercase: bool = True, include_lowercase: bool = True,
                      include_digits: bool = True, include_special: bool = True):
    characters = ''
    if include_uppercase:
        characters += ascii_uppercase
    if include_lowercase:
        characters += ascii_lowercase
    if include_digits:
        characters += digits
    if include_special:
        characters += punctuation
    for char in custom_word:
        characters = characters.replace(char, '')

    password = ''
    if length <= len(custom_word):
        return custom_word[:length]
    else:
        remaining_length = length - len(custom_word)
        password += custom_word
        for i in range(remaining_length):
            password += choice(characters)
    return password[:length]
```

Figure 11: Password Generation Algorithm

10.1.4 AES Encryption and Decryption

A cryptographic approach to encrypting and decrypting user passwords using the AES algorithm in Cipher Block Chaining (CBC) mode has been implemented. The *encrypt_sp* function encrypts a user-provided password with a specified secret key, generating a unique initialization vector (IV) for each encryption process. The resulting encrypted password, along with its corresponding IV, is encoded using base64 for storage or transmission. Conversely, the *decrypt_sp* function decrypts the encrypted password using the same secret key and IV, providing a means to retrieve the original plaintext password securely. This encryption scheme aims to enhance password security by safeguarding sensitive user credentials while ensuring accessibility when required.

```python
def encrypt_sp(name: str, key: bytes) -> str:
    cipher = AES.new(key, AES.MODE_CBC, iv=os.urandom(16))
    ct_bytes = cipher.encrypt(pad(name.encode(), AES.block_size))
    iv = b64encode(cipher.iv).decode('utf-8')
    ct = b64encode(ct_bytes).decode('utf-8')
    return iv + ':' + ct

def decrypt_sp(enc_name: str, key: bytes) -> str:
    iv, ct = enc_name.split(':')
    iv = b64decode(iv)
    ct = b64decode(ct)
    cipher = AES.new(key, AES.MODE_CBC, iv)
    pt = unpad(cipher.decrypt(ct), AES.block_size)
    return pt.decode('utf-8')
```

Figure 12: AES Implementation

10.2 Streamlit Implementation

The Streamlit implementation serves as the frontend interface for interacting with the password management system. It consists of three main functionalities: **Sign Up**, **Log In**, **Password Generator**, and **View Passwords**.

Sign Up: Users can create a new account by providing their email and password. Upon submission, the system sends the user credentials to the backend API for account creation. If successful, a success message is displayed; otherwise, appropriate error messages are shown.

Log In: Existing users can log in using their email and password. Similar to the sign-up process, the user credentials are sent to the backend API for authentication. Upon successful authentication, the user receives an access token, which is stored in the session state for subsequent requests.

Password Generator: The password generator functionality allows users to generate strong passwords based on specified criteria. Users can define the length of the password, include or exclude uppercase letters, lowercase letters, digits, and special characters. Additionally, users can provide a custom word and tag for password customization. The generated password, along with its strength score, is displayed to the user.

View Passwords: Once logged in, users can view their stored passwords. The system retrieves password data associated with the user's email from the backend API. If passwords are found, they are displayed in a tabular format, showing the tag and

corresponding password. If no passwords are found, a message indicating the absence of passwords is displayed.

Logout: A logout button is provided in the sidebar for users to log out of their accounts. Upon clicking the logout button, the access token is cleared from the session state, and the user is successfully logged out.

10.3 System Usage

The implemented system offers a straightforward and intuitive interface for users to interact with various functionalities. Below is an overview of how users can effectively utilize the system:

10.3.1 Navigation

The application incorporates a navigation feature to guide users seamlessly through its various functionalities. Users can easily switch between different sections of the application using the sidebar navigation menu. This menu provides options to access essential features such as logging in, generating passwords, and viewing stored passwords. Additionally, a logout button is conveniently located within the sidebar, allowing users to securely log out of their accounts when needed. This intuitive navigation design enhances user experience by ensuring effortless interaction with the application's functionalities.

10.3.2 Sign Up and Log In

- **Sign Up:** Users can create a new account by providing their email and password. Upon submission, the system verifies the uniqueness of the email and creates the account if it's not already registered.

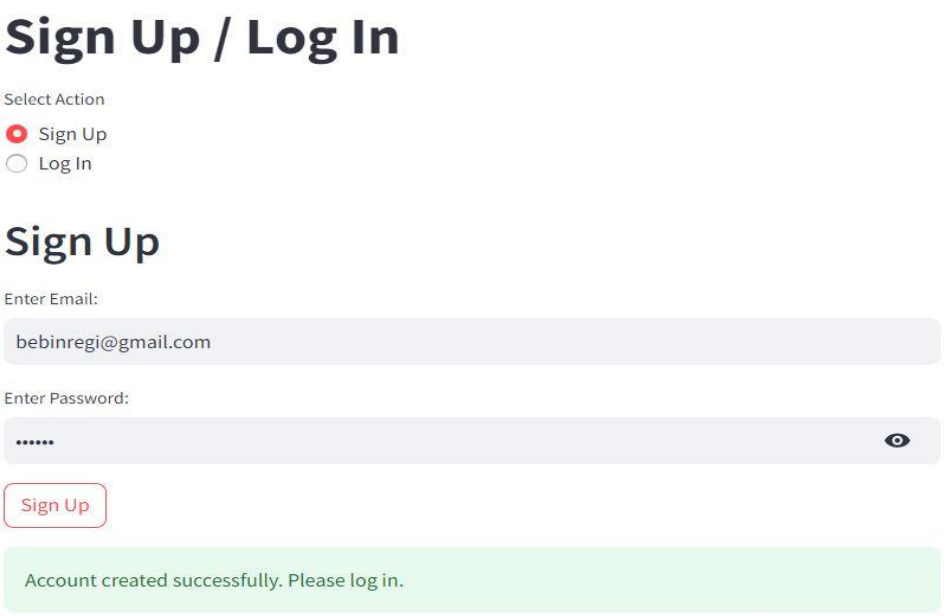


Figure 13: Successful Sign Up Notification

Figure 14: Error Message for Existing Email

- **Log In:** Existing users can log in to their accounts using their registered email and password. Upon successful authentication, users receive an access token for subsequent interactions with the system.

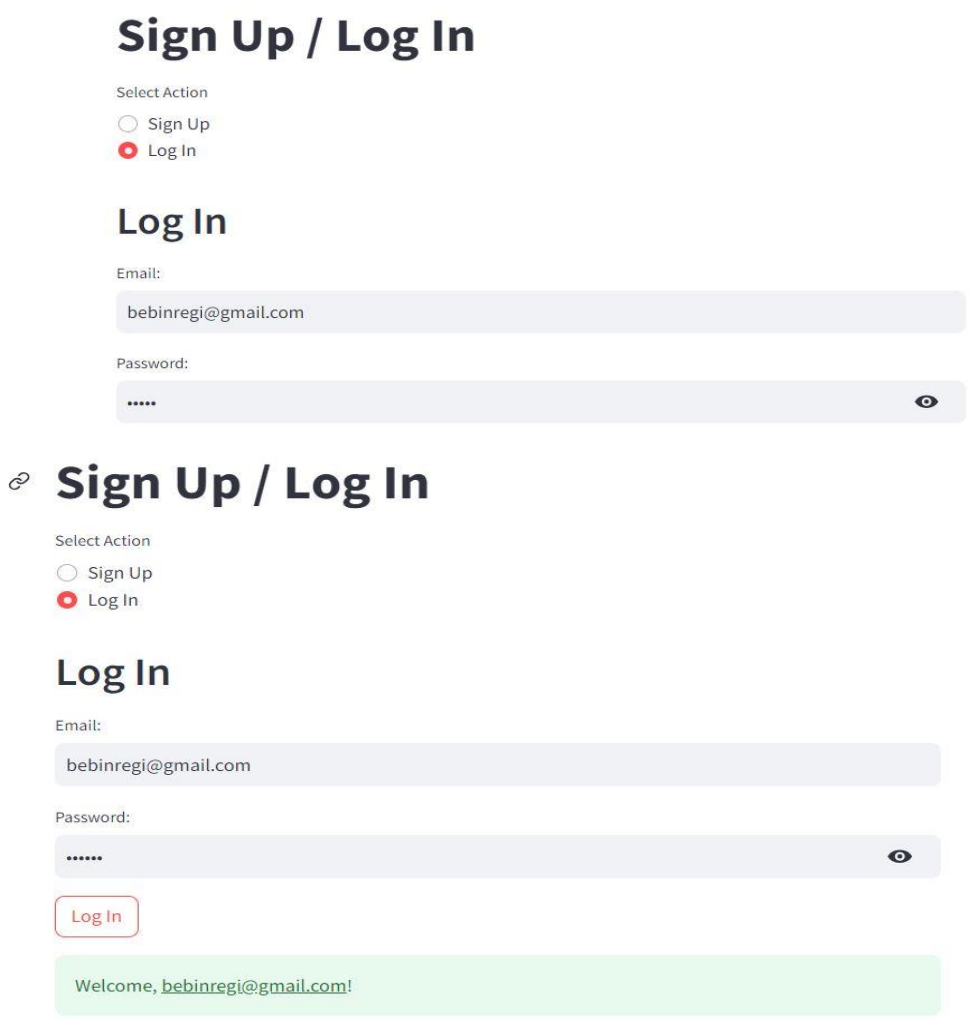


Figure 16: Successful Log In Notification

10.3.3 Password Generator:

- Users can generate secure passwords using the Password Generator feature.
- They have the flexibility to customize the password length, include or exclude uppercase letters, lowercase letters, digits, and special characters.
- Additionally, users can specify a custom word to be included in the generated password and assign a tag for identification.

- Users can also see the strength of the password, and use the same tag to update the password if needed.

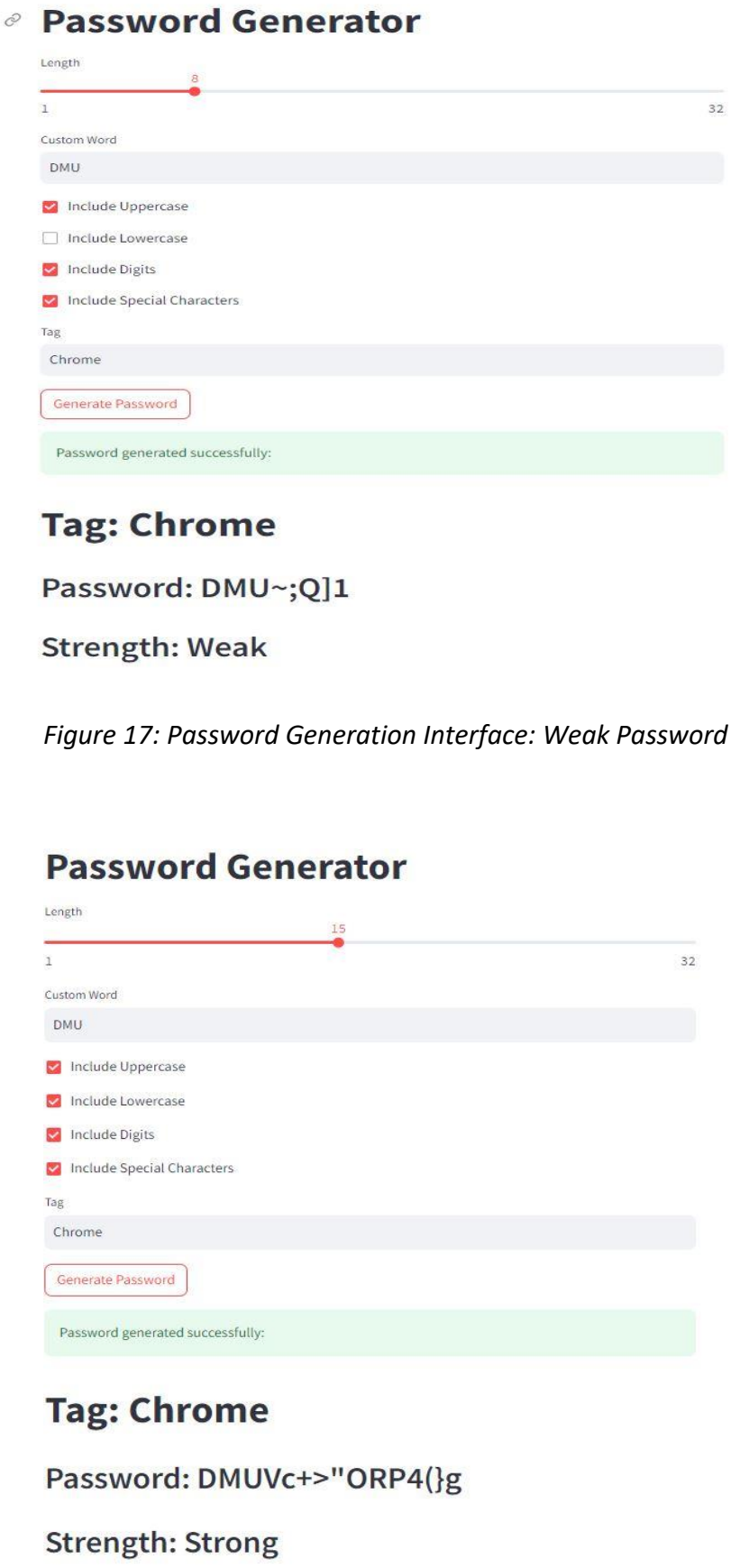


Figure 17: Password Generation Interface: Weak Password

Figure 18: Password Generation Interface: Updating With Strong Password

10.3.4 View Passwords:

- Users can view their stored passwords, organized by tags.
- Upon accessing the View Passwords feature, users are presented with a table displaying their saved passwords alongside their corresponding tags.

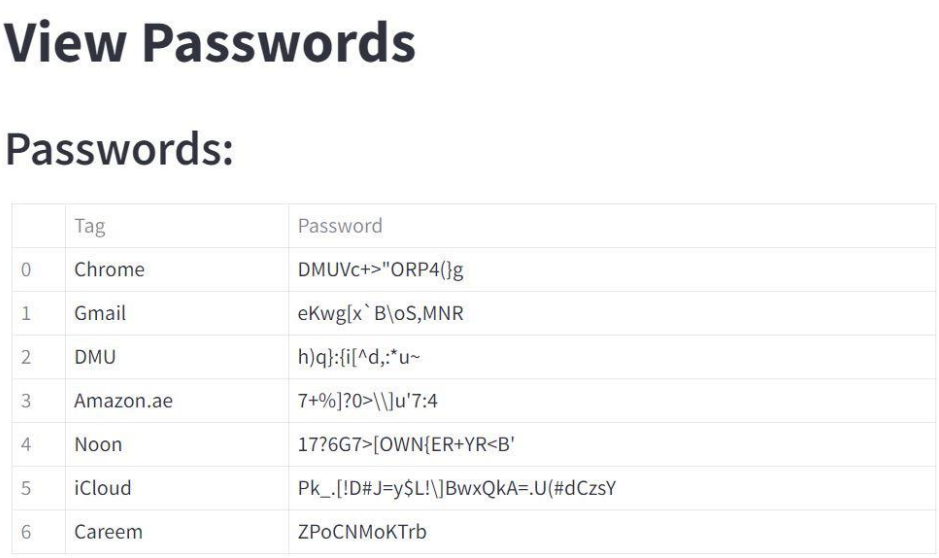


Figure 19: Passwords Viewing Interface

10.3.5 Logout:

- To securely end their session, users can log out of the system. This action invalidates the current access token, requiring users to log in again for subsequent interactions.

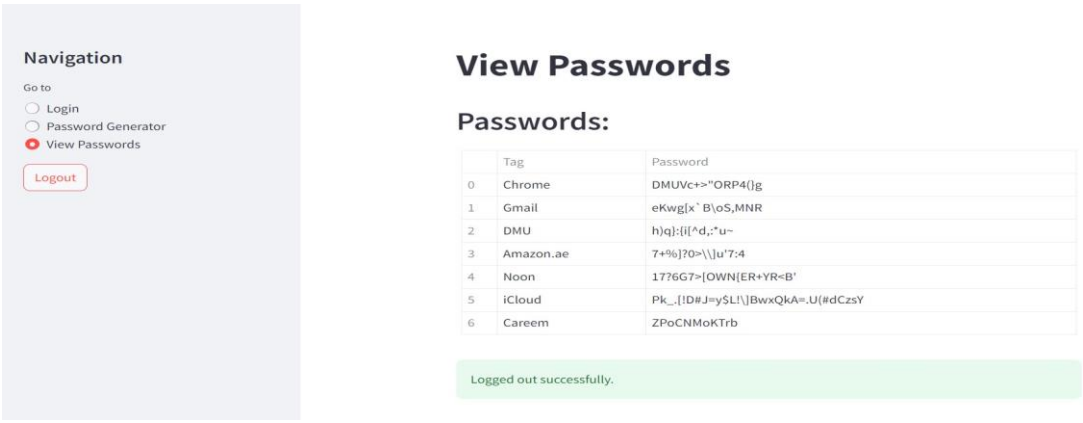


Figure 20: Clicking Logout

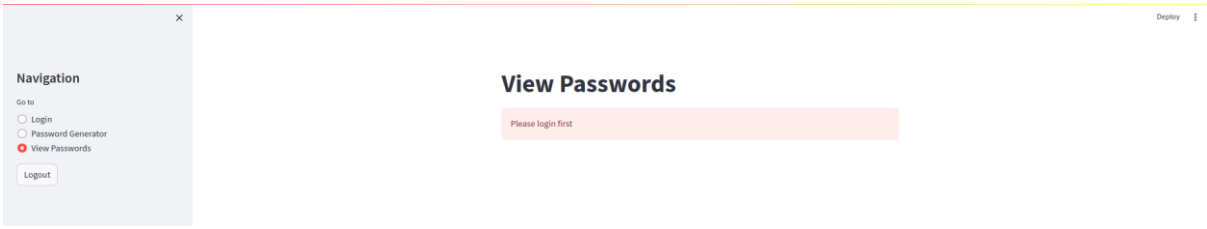


Figure 21: Session Closed

11 Testing

The testing phase of software development is critical to ensuring the reliability, functionality, and security of the application. In this section, the rigorous testing methodology was employed to verify the robustness and correctness of the system. Leveraging industry-standard practices and methodologies, the testing approach encompasses a comprehensive suite of tests designed to cover various aspects of the application, including functionality, security, and user experience.

11.1 Methodology

The testing methodology adheres to established principles of software testing, combining both manual and automated techniques to achieve thorough coverage. Drawing upon the principles of test-driven development (TDD) and behaviour-driven development (BDD), the testing strategy is iterative, with tests being continuously developed alongside the application code.

This table summarizes the results of testing conducted on various endpoints, including "Signup," "Login," "Password Generation," "View Password," and "Logout." Each row represents a specific test case, detailing the endpoint, input provided, expected outcome, and the actual result observed during testing.

Table 2: Testing Results Table

Test Case	**Endpoint**	**Input**	**Expected Outcome**	**Result**
1	/login/	Valid Credentials	Successful login	Passed
2	/login/	Invalid Password	Error message indicating invalid credentials	Passed
3	/login/	Non-existent User	Error message indicating invalid credentials	Passed
4	/signup/	New User	Successful account creation	Passed
5	/signup/	Existing User	Error message indicating email already exists	Passed
6	/generate-password/	Valid parameters	Successful password generation	Passed
7	/generate-password/	Valid parameters (custom word included)	Successful password generation with custom word included	Passed
8	/password-data/	Valid token	Password data retrieved successfully	Passed
9	/password-data/	Invalid token	Error message indicating invalid token	Passed
10	/logout/		Successful logout	Passed

12. Critical evaluation

The password generator application represents a pivotal advancement in password management technology, born out of a critical analysis of the vulnerabilities inherent in traditional password-based authentication methods. I meticulously explored the shortcomings of existing systems, identifying the pressing need for innovative solutions to bolster user security while maintaining convenience. This culminated in the development of an advanced password generator application, designed to provide users with a secure and seamless means of managing their credentials.

Implemented with a keen focus on user experience and data security, the application boasts a robust backend system built on the FastAPI framework. This backend architecture encompasses crucial components such as user registration, authentication, password generation, data retrieval, and storage mechanisms, all fortified by industry-standard encryption protocols. Complementing the backend, frontend interface, powered by Streamlit, offers users an intuitive and visually engaging platform to navigate through various functionalities effortlessly.

Every element of the application is designed with the needs of the user in mind, from the simple navigation features that facilitate seamless interaction to the meticulous testing procedures that guarantee dependability and security. The password generator application, in a sense, ushers in a new era of password management, one in which convenience and security come together to redefine the user experience when it comes it to protecting digital identities.

13. Conclusion

In conclusion, I've developed a sophisticated and user-centric solution ready to address the urgent cybersecurity issues of the digital era, stemming from my journey to revolutionize password management. My password generator application prioritizes security while enhancing user experience, as I recognize the limitations of conventional authentication methods and embrace innovative technologies.

Through meticulous attention to detail, rigorous testing, and adherence to industry best practices, I've created an efficient system that empowers users to confidently safeguard their digital identities.

Looking ahead, my dedication to adaptation and continuous improvement will drive further enhancements to the application, ensuring its position at the forefront of password management innovation. I envision my solution becoming synonymous with convenience, trustworthiness, and reliability in the cybersecurity realm. My steadfast commitment to this goal and relentless pursuit of excellence position me to revolutionize password security and instill users with the confidence to navigate the digital landscape securely.

14 References

- Murmu, M. R., Kasyap, N., & Tripathy, A. R. (2021). PassMon: A framework for password security using BiGAN, one-class SVM and LSTM.
- 2021 International Conference on Computing, Communication, and Security (ICCECS), 1-6. https://arxiv.org/pdf/2305.18324
- Adams, A., & Aday, S. (2010). Phishing scams and deception on the internet. Social Science Computer Review, 28(1), 50-62.
- Al-Maqbali, N., & Mitchell, C. J. (2017). AutoPass: Usable password generation from user input. International Journal of Human-Computer Studies, 107, 1-13.
- Ayoub, A., Tollner, R., Javed, M. U., & Boudriga, N. (2018). Security question fatigue: A field study of knowledge-based authentication and user security behavior. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (pp. 1821-1836).
- Bao, L., Li, Z., & Wang, Y. (2014). Password-based authenticated key exchange protocols: Models and security. IEEE Transactions on Dependable and Secure Computing, 11(2), 131-144.
- Brostoff, S., & Sasse, M. A. (2000). Ten usability heuristics for information security systems. Interacting with Computers, 12(2), 135-152.
- Cho, S., Han, D., & Kwon, K. (2013). Graphical passwords for multi-touch devices. Computers & Security, 39, 346-359.
- De Luca, A., Wobbrock, J., De Rose, L., & Kim, M. (2004). Graphical password authentication using touch screens: Part 1, design. In Proceedings of the 13th Annual ACM Conference on Human Factors in Computing Systems (pp. 721-728).
- Glory, J., Choi, J., Isbister, K., & Akinyokun, O. L. (2019). User-driven password generation: Striking a balance between strength and memorability. In Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (pp. 1-13).
- Pasquini, D., et al. (2021) 'Improving Password Guessing via Representation Learning,' IEEE Conference Publication | IEEE Xplore.
- Guo, Z., Zhang, Z., Liu, X., & Sun, H. (2018). Towards secure and user-friendly touch gesture passwords. IEEE Transactions on Information Forensics and Security, 13(1), 185-199.
- Jagatic, W. N., Brush, A. D., & Jakobsson, M. (2007). Phishing lure click-through rates and deception effectiveness: A field study. Information Security Journal: A Global Perspective, 16(5), 209-222.

- Jermyn, I., MacKenzie, A., Rubin, A. D., Simon, M., & Virto, M. (2017). On the security of clickable graphical passwords. ACM Transactions on Information and System Security (TISSEC), 20(1), 1-39.
- Mahdi, A., Abu Bakar, S. B., & Arof, H. (2022). A comprehensive survey on hand gesture recognition for user authentication. Journal of Network and Computer Applications, 209, 103453.
- Man, S., Engel, T., & Komssi, M. (2020). Gesture intersection attacks: Breaking graphical passwords on multi-touch devices. In Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security (pp. 1953-1967).
- Memon, N., & Zhang, P. (2013). A survey of user authentication schemes in distributed systems. ACM Computing Surveys (CSUR), 45(3), 1-29.
- Oesch, G., & Ruoti, S. (2019). Password managers: A security risk analysis. In Proceedings of the 12th Symposium on Usable Privacy and Security (pp. 301-312).
- Rahman, M. M., Islam, S. M. R., & Wasnik, M. S. (2019). A survey on multi-factor authentication for internet of things (IoT). Security and Communication Networks, 12(17), 4242-4255.
- Reynolds, C., Shay, T., Ren, J., & Bao, F. (2018). Context-aware password generation: Understanding user preferences and behaviors. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (pp. 1-12).
- Santoso, F., Widyastuti, H., & Budiarto, R. (2023). A robust click-based graphical password using user's drawing habit. Journal of Big Data, 11(1), 1-16.
- Shay, S., Ingles, A., & Orr, M. (2010). Ending reliance on memory: Unlock patterns for smartphone authentication. In Proceedings of the 18th ACM Conference on Computer and Communications Security (pp. 185-194).
- Sun, S. L., Ng, E., & Goi, B. M. (2003). Credibility of web banners with security seals. Information & Management, 40(6), 521-532.
- Turner, E., Wuyts, K., & Ashford, M. (2020). Password hashing: A study in usability. In Proceedings of the 2020 Conference on Security and Privacy in Emerging Systems (pp. 1-12).
- Urbanski, M., Stoecker, S., & Holtmanns, S. (2*17). Measuring password guessability – how well do current password meters perform? In Proceedings of the 2017 IEEE European Symposium on Security and Privacy Workshops (pp. 237-244).
- Wang, L., Zhu, Y., Liu, X., Li, J., & Sun, X. (2017). Password complexity: A tradeoff between security and usability. Computers & Security, 69, 1-15.

- Yang, M., Li, X., Zhang, Z., & Sun, H. (2015). Enabling continuous authentication using hand gesture passwords. In Proceedings of the 2015 ACM SIGSAC Conference on Computer and Communications Security (pp. 689-700).

15. GitHub Repository

https://github.com/BebRn/Password-Generator