DESIGN DOCUMENT

h4$h3d - Your Digital Fortress

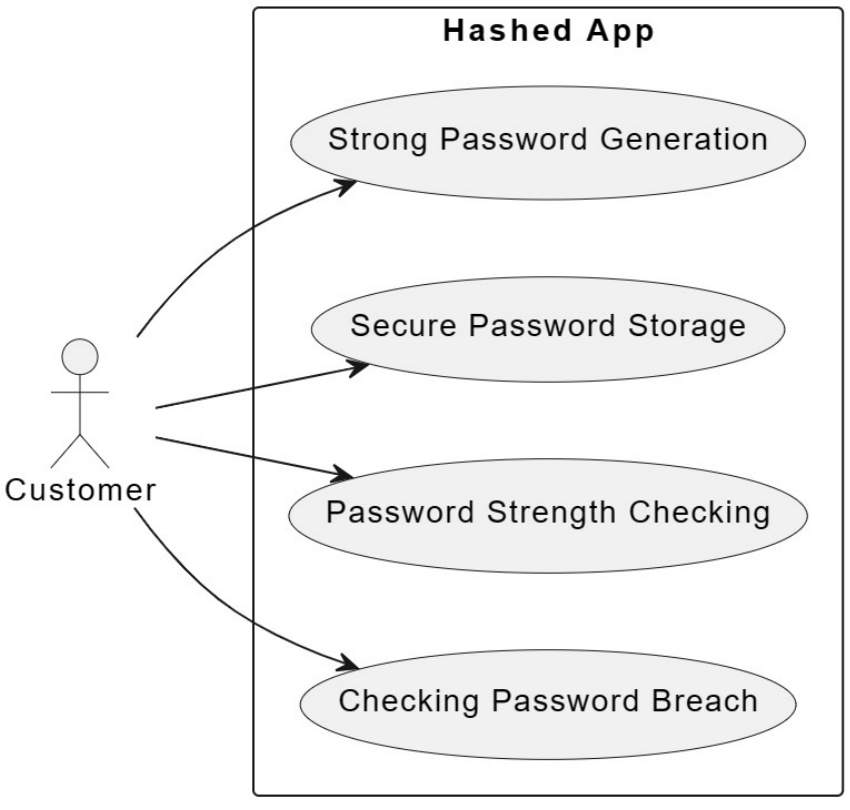
Date: 8 October, 2023

Version: 1.0

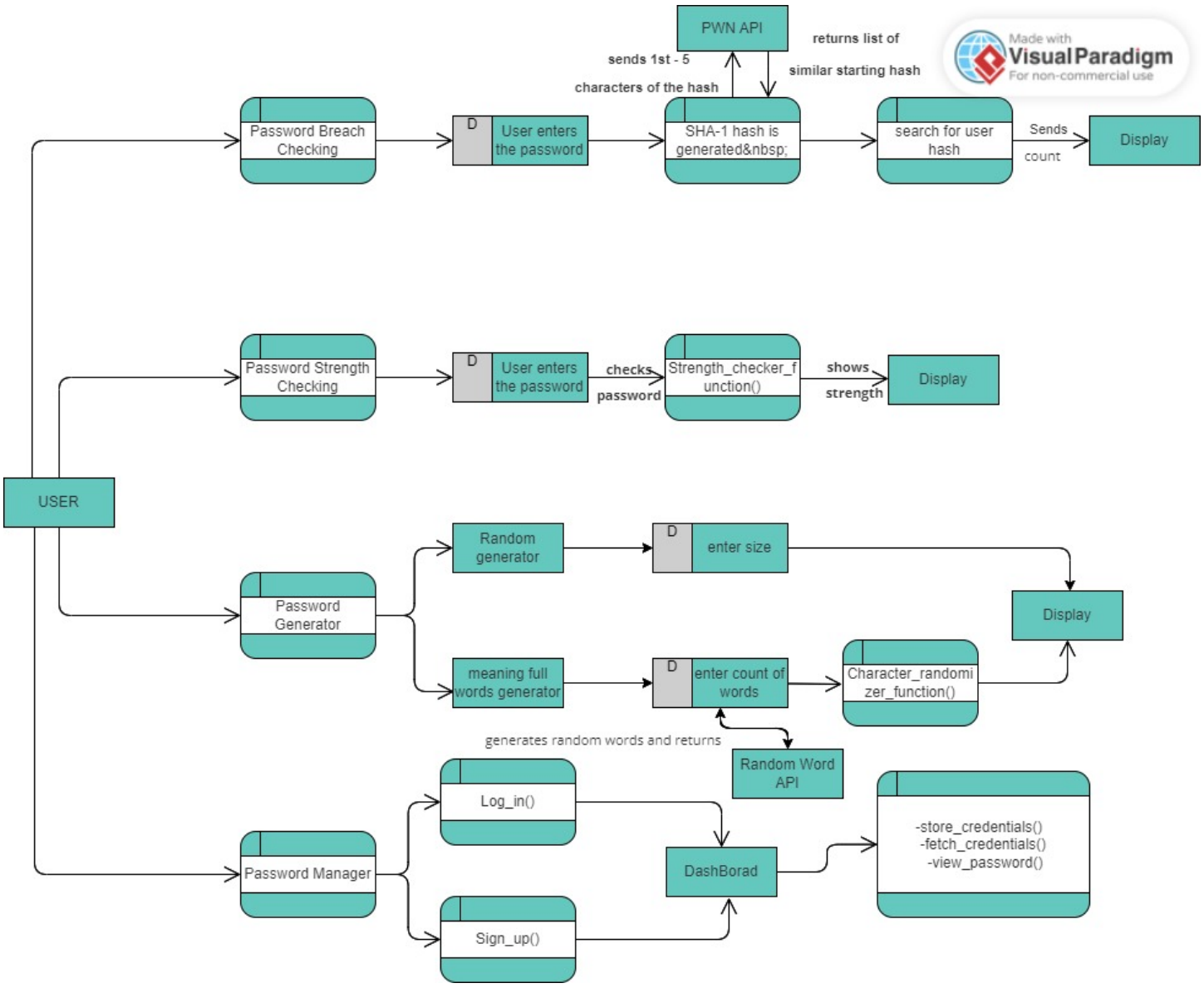
Name	Enrollment No
Anant Jain	22114005
Dhas Aryan Satish	22117046
Divij Rawal	22114031
Parit Gupta	22117100
Pratyaksh Bhalla	22115119
Roopam Taneja	22115030

High Level Design

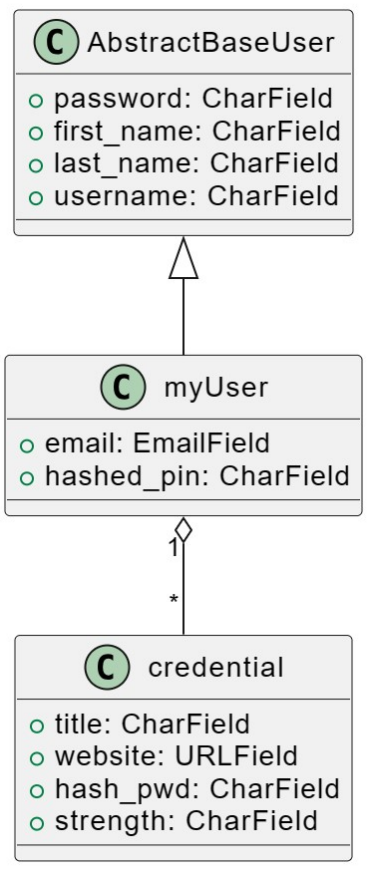
Use Case Diagram



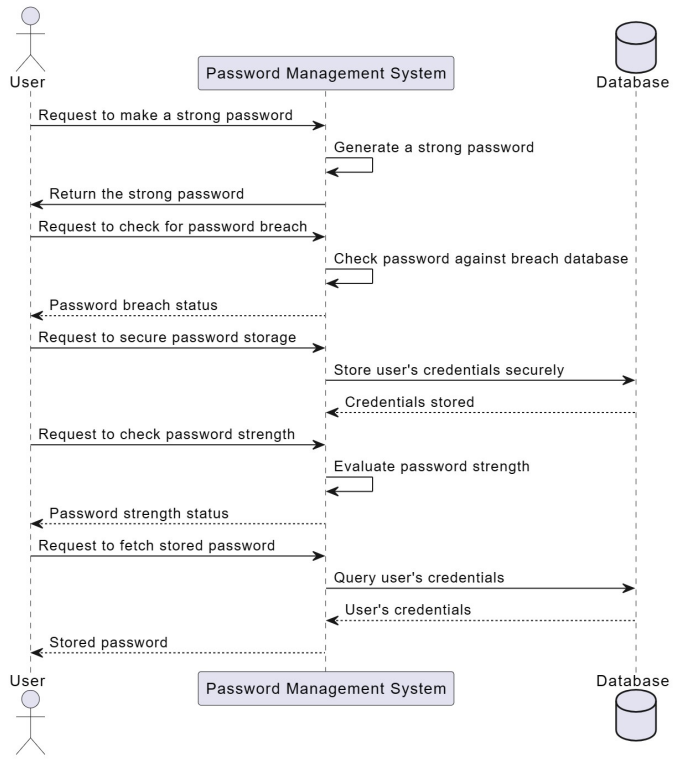
Data Flow Diagram



Class Diagram



Sequence Diagram



System overview:

Description:

The "h4$h3d - Your Digital Fortress" project aims to develop a comprehensive password manager application that provides following functionalities:

i) Password generation.

ii) Password breach checking.

iii) Password strength assessment.

iv) Password storage management.

Low Level Design:

i) Password Generation:

1. Secure Random Password:

Input: Length of the desired password.

Implementation:

- Within the Python script, we utilize the 'secrets' module, known for its cryptographic strength, to guarantee the security of randomly generated characters. As opposed to using ‘random’ module, known for it’s pseudo-random characteristics.
- To enforce specific character restrictions on the password, we maintain a predefined set of permissible characters.
- Using the 'secrets' module, we randomly select characters from this set, which will ultimately compose the final password.

Output:

- The output is a randomly generated password that adheres to the specified character restrictions, ensuring a high level of security.

2. Pass-phrase generation:

Input: Specify the desired number of words for the passphrase.

Implementation:

- Our implementation leverages a secure random word API (https://random-word-api.herokuapp.com/home), ensuring the generation of truly random words of variable lengths.
- These random words are then concatenated, with each word separated by a hyphen ('-').
- To further enhance security, a pass-checker function is applied to replace certain letters with special characters, and word length restrictions are imposed, resulting in the pass-phrase with desired properties.

Output:

- The output is a randomly generated, secure passphrase. For example, an output might look like: "B@y-R3stricts-root." This passphrase is designed to offer a high level of security and is a valid password for all sites.

Psuedocode:

```
FUNCTION get_pass(num):
    api = "https://random-word-api.herokuapp.com/word"
    parameters = {"number": num}
    response = GET(api, parameters)

    IF response.status_code != 200:
        RETURN "Error in fetching data"
    ELSE:
        result = PARSE_JSON(response.body)
        s = ""

        FOR i FROM 0 TO num - 2:
            s += result[i] + "-"

        s += result[num - 1]
        RETURN s
```

ii) Password Breach Checking:

Input: Enter the password string that you want to evaluate for security.

Source/API:

- We utilize the Have I Been Pwned website's API to assess the password's security. This API allows us to determine how frequently the given password has been compromised in previous data breaches, thereby indicating the potential risk of using it.

API Privacy Protection:

- The API safeguards user privacy through a technique called k-anonymity. Instead of sending the complete password, a prefix of the SHA-1 hash of the password string is transmitted to the API. The API then returns the count for each possible hash with same prefix. The user then can locally search the hash of password. This ensures that the user's identity remains anonymous, and the API cannot trace the password back to the user.

Implementation:

- Our implementation involves interfacing with the Have I Been Pwned API to retrieve the count of data breaches associated with the provided password.

Output:

- The output is the number of data breaches associated with the input password. This count serves as an indicator of the password's vulnerability based on its history of exposure in breaches.

Psuedocode:

```
FUNCTION check_password_pwned(password):
sha1_password = HASH_SHA1(password)
prefix = FIRST_FIVE_CHARACTERS(sha1_password)
suffix = REMAINING_CHARACTERS(sha1_password)
api_url = "https://api.pwnedpasswords.com/range/" + prefix
response = GET(api_url)
IF response.status_code == 200:
    FOR EACH line IN SPLIT_LINES(response.body):
        IF line STARTS_WITH suffix:
            count = GET_COUNT_FROM_LINE(line)
            RETURN "The password has been found " + count + " times and is compromised."
    RETURN "The password has not been found in any known breaches."
ELSE IF response.status_code == 404:
    RETURN "Invalid API endpoint or no data found."
ELSE:
    RETURN "Error occurred while checking the password."
```

iii) Password strength assessment:

Input:

- Provide the password string that you want to evaluate for its strength.

Implementation:

- Our implementation incorporates a 'password_strength()' function designed to assess the strength of the provided password. This function employs a set of criteria, including the presence of special characters and password length, to determine whether the password falls into one of three categories: weak, medium, or strong.

Output:

- The output of the 'password_strength()' function is a qualitative assessment of the password's strength. It categorizes the password as either weak, medium, or strong based on the defined criteria. This categorization helps users gauge the level of security associated with their chosen password.

Psuedocode:

```
FUNCTION password_strength(password):
    very_weak_pattern = '^.{1,5}$'
    weak_pattern = '^(?=.*[a-zA-Z])(?=.*\d).{6,}$'
    medium_pattern = '^(?=.*[a-z])(?=.*[A-Z])(?=.*\d).{8,}$'
    strong_pattern = '^(?=.*[a-z])(?=.*[A-Z])(?=.*\d)(?=.*[@#$%^&+=]).{8,}$'

    IF MATCH(strong_pattern, password):
        RETURN "Strong"
    ELSE IF MATCH(medium_pattern, password):
        RETURN "Medium"
    ELSE IF MATCH(weak_pattern, password):
        RETURN "Weak"
    ELSE IF MATCH(very_weak_pattern, password):
        RETURN "Very Weak"
    ELSE:
        RETURN "Invalid"
```

iv) Password storage:

At the time of registration, each user has their own PIN which is not changeable.

After a user is logged in:

Input:

- Provide the title of the password (e.g., account or service name), the corresponding website or application, and the password you wish to secure.

Implementation:

- Our implementation employs the AES (Advanced Encryption Standard) algorithm to enhance password security. This process begins with a permanent PIN (Personal Identification Number), which serves as the basis for deriving the encryption key.
- When a user enters their PIN, it is converted into a hash value. If the entered PIN matches the stored one, the PIN is used to compute the encryption key.
- This encryption key is then employed to encrypt the password, resulting in an encrypted version of the password. Importantly, the key is never stored, ensuring that even in the event of a data breach, the encrypted password cannot be decrypted without the correct PIN.

Output:

- The output is an updated password list that includes the encrypted version of the password, which is securely stored. This approach significantly enhances the security of stored passwords, making them resistant to unauthorized access, even in the case of a data breach.

Questions:

Q:Can you describe your project?

A:The "h4$h3d - Your Digital Fortress" project aims to develop a comprehensive password manager application that provides following functionalities:

i) Password generation.

ii) Password breach checking.

iii) Password strength assessment.

iv) Password storage management.

Q:Can you elaborate more on password generation?

A:Within the Python script, we utilize the 'secrets' module, known for its cryptographic strength, to guarantee the security of randomly generated characters. As opposed to using ‘random’ module, known for it’s pseudo-random characteristics.

- To enforce specific character restrictions on the password, we maintain a predefined set of permissible characters.
- Using the 'secrets' module, we randomly select characters from this set, which will ultimately compose the final password.

Q:What is the process for checking for password breaches?

A:We utilize the Have I Been Pwned website's API to assess the password's security. This API allows us to determine how frequently the given password has been compromised in previous data breaches, thereby indicating the potential risk of using it.

- The API safeguards user privacy through a technique called k-anonymity. Instead of sending the complete password, a prefix of the SHA-1 hash of the password string is transmitted to the API. The API then returns the count for each possible hash with same prefix. The user then can locally search the hash of password. This ensures that the user's identity remains anonymous, and the API cannot trace the password back to the user.
- Our implementation involves interfacing with the Have I Been Pwned API to retrieve the count of data breaches associated with the provided password.

Q:How is the strength of my password evaluated?

A:Our implementation incorporates a 'password_strength()' function designed to assess the strength of the provided password. This function employs a set of criteria, including the presence of special characters and password length, to determine whether the password falls into one of three categories: weak, medium, or strong.

Q:Is my password encrypted?

A:Our implementation employs the AES (Advanced Encryption Standard) algorithm to enhance password security. This process begins with a permanent PIN (Personal Identification Number), which serves as the basis for deriving the encryption key.