**DEBUGGERS**

**RANDOM PASSWORD GENERATOR**

**Presented by:**

**NIRAIMATHI S**

**SANJEEV V**

**HARSHINI J**

**DHARANEESHWARAN G**

**NIRSHIKHA K B**

**Problem Statement: Random Password Generator**

In today's digital world, strong, secure passwords are crucial to ensuring the safety and privacy of users' personal information and accounts. Many people use weak or repetitive passwords that can easily be guessed or cracked, leading to security breaches.

The goal of this project is to develop a **Random Password Generator** that can create strong, unique, and unpredictable passwords for users. The generator will provide users with the ability to customize password length, complexity, and the inclusion of various character types (e.g., uppercase letters, lowercase letters, numbers, and special symbols).

**BRIEF OVERVIEW:**

A password generator is a tool or software that

creates strong and random passwords for various online accounts and services. These passwords are typically difficult for hackers to guess, enhancing the security of your accounts. Password generators can usually be customized to include a specific length, character types (letters, numbers, symbols), and other criteria to meet your security needs. It’s an essential tool for maintaining online security and protecting sensitive information.

**NEED OF COMPUTERIZATION:**

Computerization of password generators offers several advantages:

1.**Efficiency:** Computerized password generators can quickly generate large numbers of complex passwords, saving time and effort compared to manual generation.

2.**Randomization:** Computers can generate truly random passwords, making them more secure than human-created passwords, which tend to follow patterns or use easily guessable combinations.

3.**Customization:** Users can easily customize computerized generators to meet specific security requirements, such as password length, character types, and restrictions.

4.**Storage and Management:** Computerized systems can securely store and manage generated passwords, making it easier to retrieve them when needed and reducing the risk of forgetting or losing passwords.

5.**Integration:** Password generators can be integrated into various software applications, websites, and security protocols, streamlining the password creation process and enhancing overall security.

6.**Password Strength Assessment:** Some computerized password generators include password strength assessment tools to ensure that the generated passwords meet security standards.

7.**Accessibility:** Computerized password generators are readily accessible on a variety of devices, making it convenient for users to create strong passwords wherever they are.

8.**Updates and Maintenance:** Software-based password generators can receive updates and improvements to stay ahead of evolving security threats and provide better password generation capabilities.

Overall, computerization enhances the effectiveness and convenience of password generation, contributing to improved cybersecurity practices.

**SOFTWARE AND HARDWARE REQUIREMENTS:**

The software and hardware requirements for a password generator can vary depending on its complexity and the platform it’s built for. Here are some general guidelines:

**Software Requirements:**

1.**Programming Language:** You’ll need a programming language to develop the generator. Common choices include Python, Java, JavaScript, or C++.

2.**Operating System:** Specify the target OS for your generator, whether it’s Windows, macOS, Linux, or web-based.

3.**Development Environment:** Depending on your chosen programming language, you’ll need a compatible development environment such as Visual Studio Code, PyCharm, Eclipse, etc.

4.**Random Number Generator Library:** If your password generator relies on randomization, you might need a library or built-in functions for generating random numbers securely.

**Hardware Requirements:**

1.**Processor:** The processor requirement depends on the complexity of your password generator. For basic generators, most modern CPUs should suffice.

2.**Memory (RAM):** Again, this depends on the complexity and scale of your application. For small-scale password generators, 2GB to 4GB of RAM should be adequate.

3.**Storage:** You won’t need much storage space for a password generator application itself. However, you might need some space to store configuration files or logs.

4.**Graphics:** If your password generator has a graphical user interface (GUI), you’ll need a display and graphics capabilities compatible with your chosen GUI framework.

5.**Internet Connection:** If your generator relies on online resources, such as fetching password policies or checking against online databases for password strength, you’ll need an internet connection.

6.**Additional Hardware:** If you’re developing specialized hardware for generating passwords (e.g., dedicated hardware security modules), specific hardware requirements will depend on the hardware chosen.

**ADVANTAGES ON PASSWORD GENERATOR:**

Password generators offer several advantages, especially when used responsibly in the context of online security:

1.**Stronger Passwords:** Password generators can create complex and random passwords that are difficult for attackers to guess or crack. This significantly improves the security of your online accounts.

2.**Avoid Common Patterns:** Password generators can ensure that passwords don’t follow common patterns like “123456” or “password,” which are often targeted by attackers.

3.**Customization:** Many password generators allow you to customize password requirements, such as length, character types (uppercase, lowercase, numbers, special characters), and exclusions, giving you control over the level of security. 4.**No Need to Remember Complex Passwords:** Users don’t need to remember intricate passwords generated by these tools, reducing the risk of weak passwords due to human memory limitations.

5.**Time-Saving:** Generating passwords on the fly is quick and convenient, saving users the effort of coming up with their passwords.

**LIMITATIONS:**

Password generators are valuable tools for enhancing online security, but they do have certain limitations:

1.**Lack of Memorability:** Generated passwords are typically complex and random, making them difficult for users to remember. This can lead to users writing down or storing passwords insecurely, which could compromise security. 2.**Dependency on a Generator:** Users may become reliant on password generators and may not know how to create strong passwords manually, which can be a disadvantage if they don’t have access to the generator.

3.**Risk of Loss:** If you lose access to your password generator or it’s compromised, you may have trouble accessing your accounts. This emphasizes the importance of securely managing and storing generated passwords.

4.**Limited Use in Offline Scenarios:** Some password generators require an internet connection or specific software, which can be limiting in offline or restricted environments.

5.**Not Ideal for All Situations:** Password generators are best suited for online accounts. They may not be suitable for physical security or systems where password entry is infrequent or where other forms of authentication are required.

**SOURCE CODE:**

import random import time def anyrandompass():

len=int(input("Enter the length of the password:")) lower="abcdefghijklmnopqrstuvwxyz"

upper="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

n="01234456789" s="!@#$%^&\*()." str=upper+lower+n+s passwd="".join(random.sample(str,len)) print(passwd) s=input('Do you want to save the password(y/n):')

if s=='y':

import mysql.connector as a

b=a.connect(host="localhost",user="root",passwd="a7$phDdj0@",databa se="password\_gene")

c=b.cursor() q='insert into password (pass,len) values("{0}",{1})'.format(passwd,len) c.execute(q)

b.commit()

b.close() print('Successfully saved') elif s=='n':

time.sleep(1) print('Remember your password')

else:

print('Wrong input') def custompassword1():

len=int(input("Enter the length of the password:")) a=input("Enter the starting word:") b=input("Enter the ending word:") lower="abcdefghijklmnopqrstuvwxyz"

upper="ABCDEFGHIJKLMNOPQRSTUVWXYZ"

n="01234456789" s="!@#$%^&\*()." str=upper+lower+n+s passwd="".join(random.sample(str,len-2)) p=a+passwd+b

print(p) s=input('Do you want to save the password(y/n):')

if s=='y':

import mysql.connector as a

b=a.connect(host="localhost",user="root",passwd="a7$phDdj0@",databa se="password\_gene")

c=b.cursor()

q='insert into password (pass,len) values("{0}",{1})'.format(p,len) c.execute(q)

b.commit()

b.close() time.sleep(1) print('Successfully saved') elif s=='n':

print('Remember your password')

else:

print('Wrong input') def custompassword2():

leng=int(input("Enter the length of the password:")) print('\*\*\*\*\*\*\*\*\*Enter',leng,'letters\*\*\*\*\*\*\*\*\*\*') time.sleep(1) strg=input("Enter the letters to used in your password:") while len(strg)!=leng and len(strg)<leng:

print("Enter",leng-len(strg),"more letters:") a=input("Enter here:") if len(strg)==leng or len(strg)>leng:

break else:

strg=strg+a passwd="".join(random.sample(strg,leng)) print(passwd)

s=input('Do you want to save the password(y/n):')

if s=='y':

import mysql.connector as a

b=a.connect(host="localhost",user="root",passwd="a7$phDdj0@",databa se="password\_gene")

c=b.cursor() q='insert into password (pass,len) values("{0}",{1})'.format(passwd,leng) c.execute(q)

b.commit()

b.close() time.sleep(1) print('Successfully saved') elif s=='n':

time.sleep(1) print('Remember your password')

else:

print('Wrong input') def shuffling(): strg=input("Enter the words to shuffled:") word="".join(random.sample(strg,len(strg))) print(word) def menu1(): print("\*\*\*\*\*Welcome to password generator prototype\*\*\*\*\*\*") print("1.Random password") print("2.Custom password")

print("3.Shuffler") time.sleep(1) a=int(input("Enter the choice(1/2/3):")) print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

if a==1:

anyrandompass() elif a==2:

print("1.Custom1") print("2.Custom2") b=int(input("Enter the choice(1/2):")) if b==1: print("Helps to create your password with your willing first and last letter") time.sleep(1) c=input("Do you want to continue(y/n):")

if c=='y':

custompassword1()

else: pass elif b==2:

print("Helps to create your password with your favourable words") time.sleep(1) c=input("Do you want to continue(y/n):")

if c=='y':

custompassword2()

else: pass else:

print("\*\*\*Helps to shuffle your words\*\*\*") shuffling() def menu2():

d='y' while d=='y':

print("1.Add a record") print("2.Update a record") print("3.Delete a record") print("4.Display a record") print("5.Exiting") time.sleep(1) a=int(input("Enter your choice(1/2/3/4/5):")) print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

if a==1: add() time.sleep(1) print('\*\*\*\*Successfully added\*\*\*\*') elif a==2: update() time.sleep(1) print('\*\*\*\*Successfully updated\*\*\*\*') elif a==3: delete() time.sleep(1) print('\*\*\*\*Successfully deleted\*\*\*\*') elif a==4: display() time.sleep(1) print('\*\*\*\*Successfully displayed\*\*\*\*') elif a==5: print('####Exiting####') break else:

print("Wrong input") d=input('Do you want to continue viewing and modifying(y/n):') def add():

import mysql.connector as a

b=a.connect(host="localhost",user="root",passwd="a7$phDdj0@",databa se="password\_gene")

c=b.cursor() i=int(input('Enter no.:')) n=input('Enter describe about password:') p=input('Enter password:') l=int(input('Enter length:')) q='insert into password values({0},"{1}","{2}",{3})'.format(i,n,p,l)

c.execute(q)

b.commit()

b.close() def update():

import mysql.connector as a

b=a.connect(host="localhost",user="root",passwd="a7$phDdj0@",databa se="password\_gene")

c=b.cursor() try: n=input('Enter description where to be updated:') p=input('Enter new password:') q="""update password set pass='%s' where pname='%s'"""%(p,n)

c.execute(q)

b.commit()

b.close() except Expection as e:

print(e) def delete():

import mysql.connector as a

b=a.connect(host="localhost",user="root",passwd="a7$phDdj0@",databa se="password\_gene")

c=b.cursor()

n=input('Enter what record to be deleted:') q='delete from password where pname="{0}"'.format(n)

c.execute

b.commit()

b.close() def display():

import mysql.connector as a

b=a.connect(host="localhost",user="root",passwd="a7$phDdj0@",databa se="password\_gene")

c=b.cursor() try:

print('1.Display all') print('2.With conditions') time.sleep(1) k=int(input('Enter choice(1/2):')) print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

if k==1:

c.execute('select

pid,"Description:",pname,"password:",pass,"Length:",len from password') else:

n=input('Enter the description to be displayed:')

q='select pid,"Description:",pname,"password:",pass,"Length:",len from password where pname="{0}"'.format(n,) c.execute(q) g=c.fetchall() if g!='Empty set': for i in g:

print(i) else:

print("Error:unable to fetch data")

b.close() except:

print("Error:unable to fetch data") def menu():

c='y' while c=='y':

print('1.Create password') print('2.Viewing and modifying your password') time.sleep(1) l=int(input('Enter your choice(1/2):')) print('\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*')

if l==1: menu1() elif l==2: menu2() else:

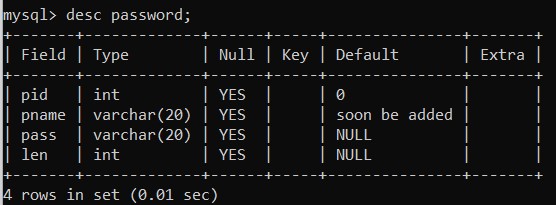
print('Wrong input') time.sleep(1) c=input('Do you want to continue the process again(y/n):') menu()

**Sample output:**

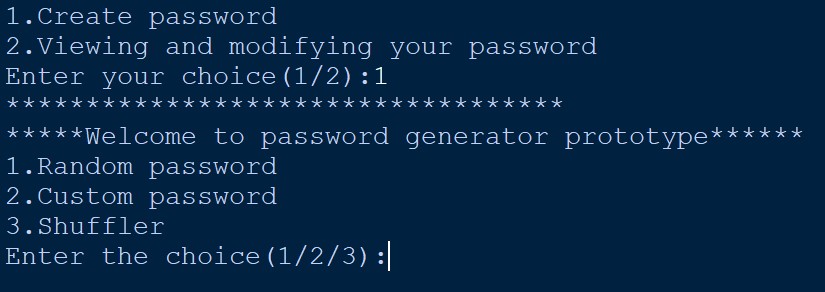
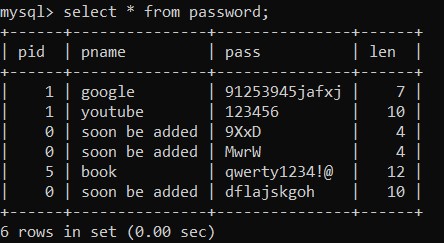
**Table password:**

**Structure of password:**

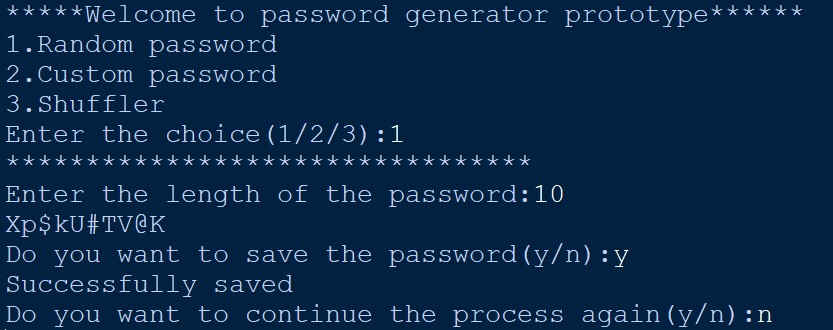
**Data in password:**



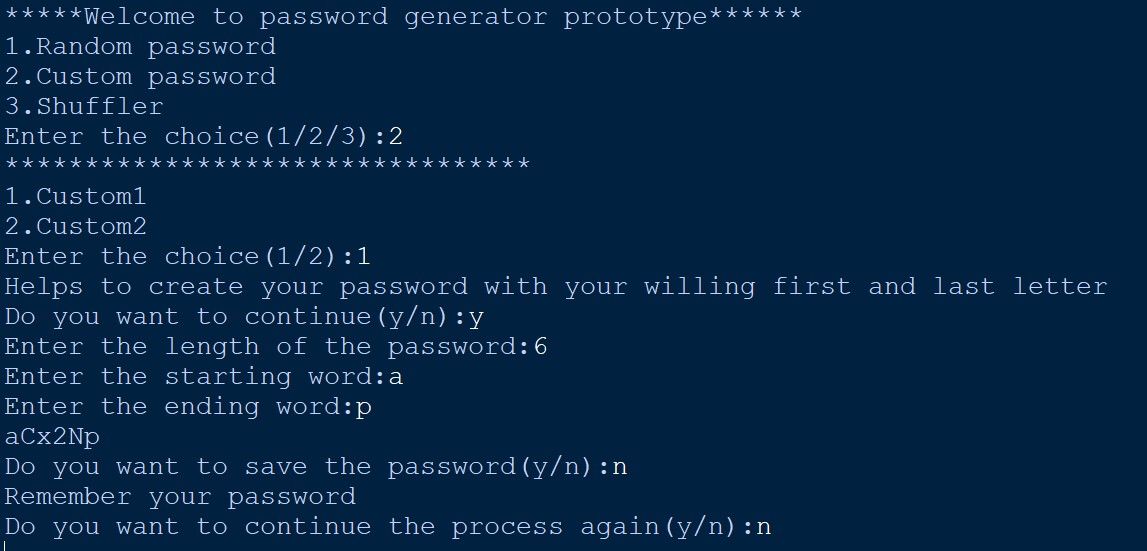
**Create password portal:**



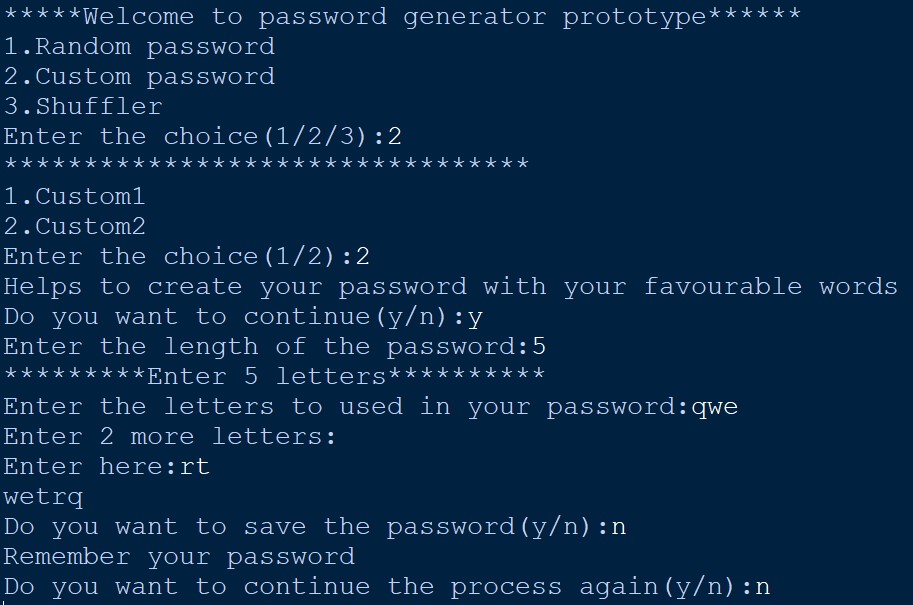
**Random password portal:**



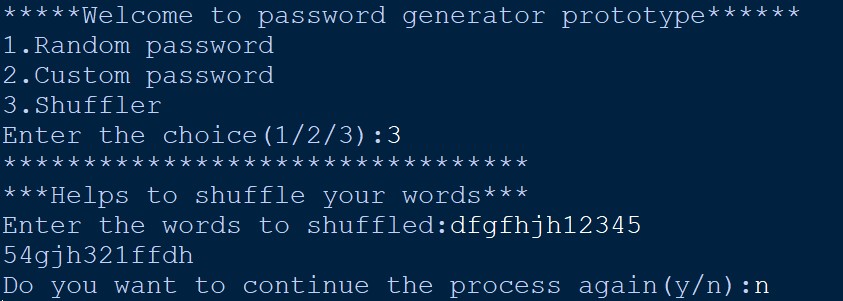
**Custom password1 portal:**



**Custom password2 portal:**

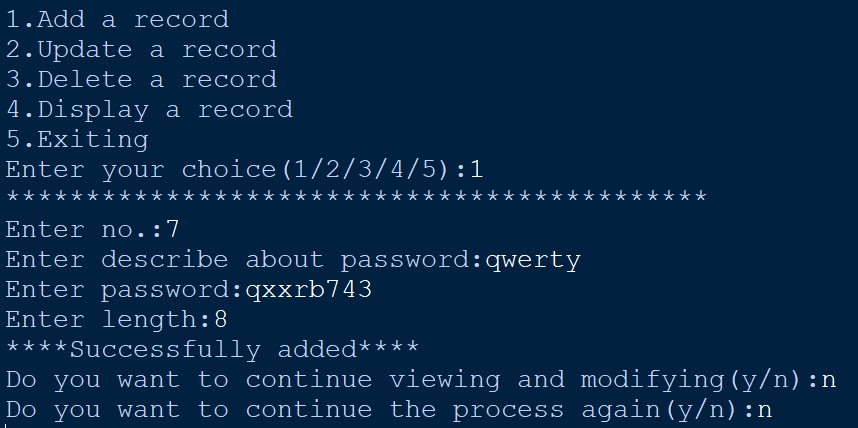
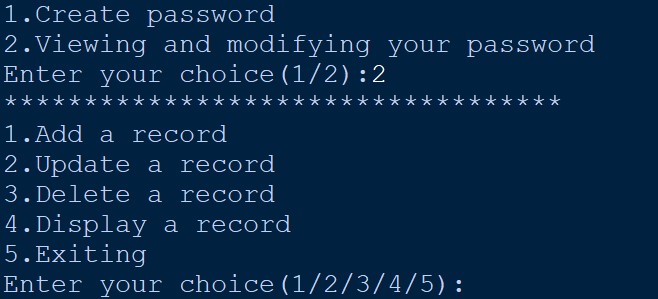


**Shuffler portal:**



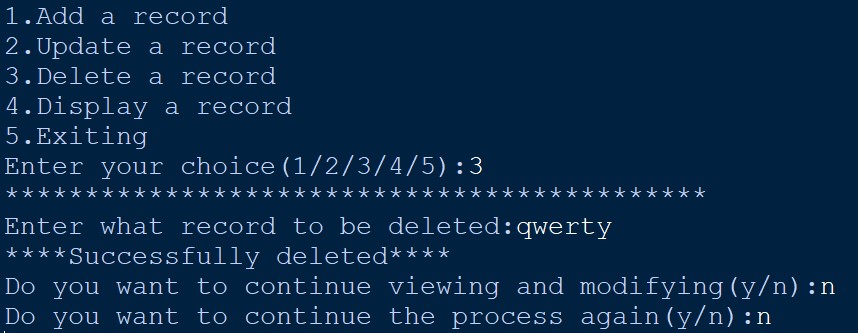
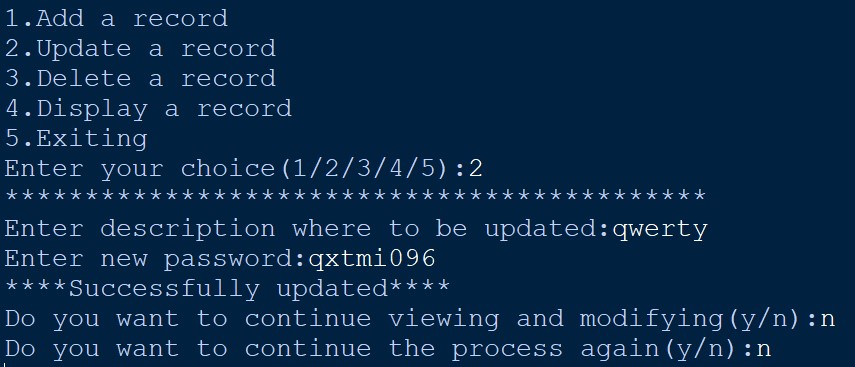
**Viewing and Modifying portal:**

**Adding data portal:**

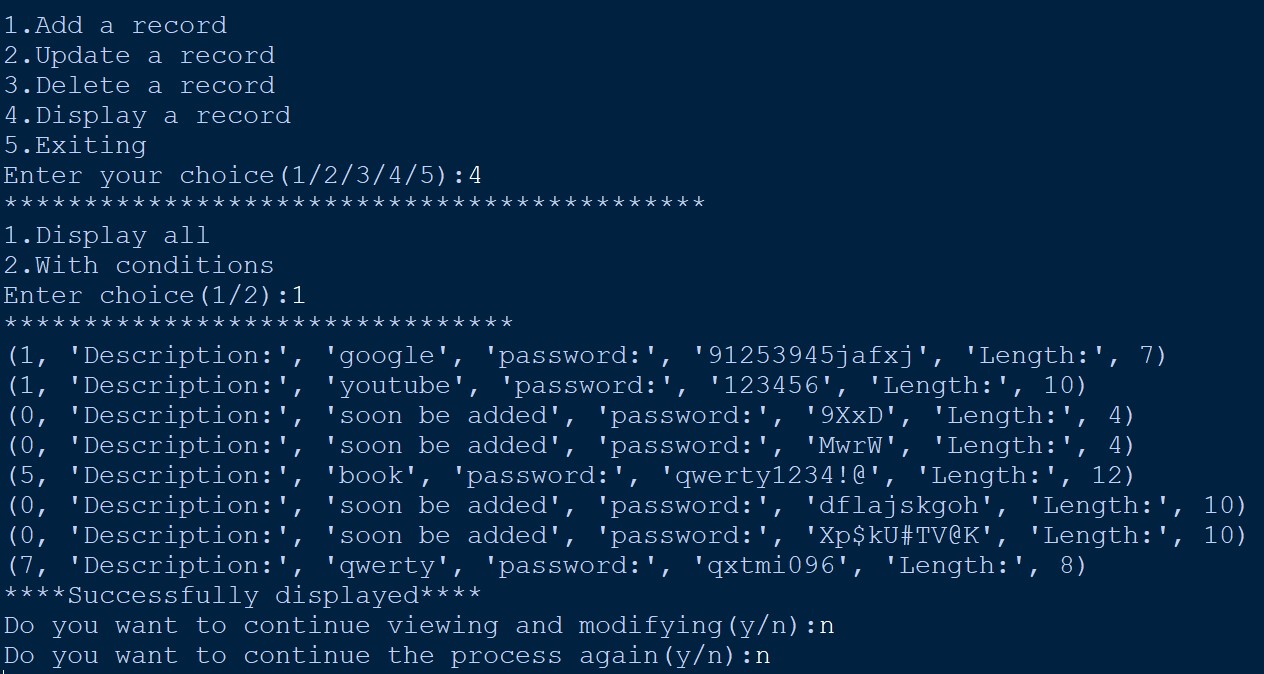


**Updating data portal:**

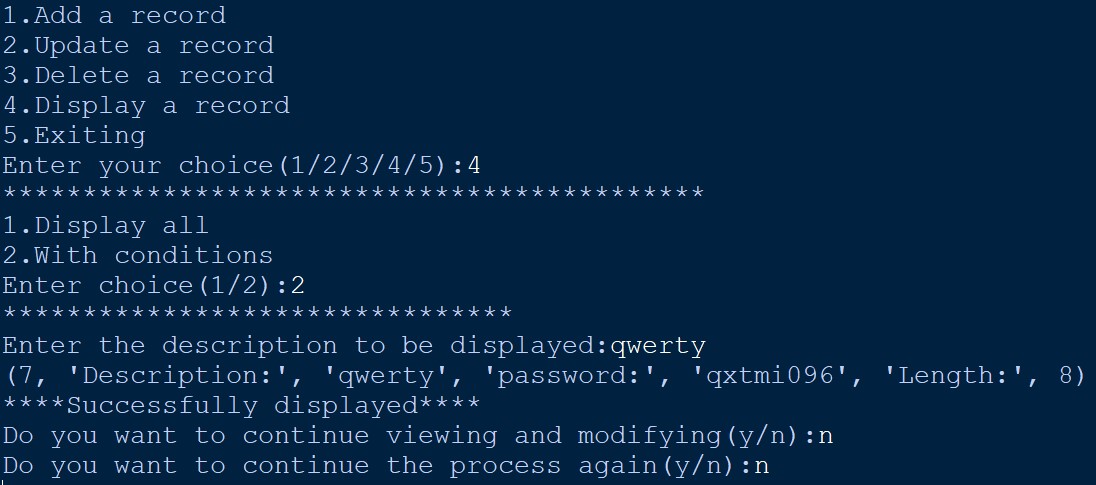
**Deleting data portal:**



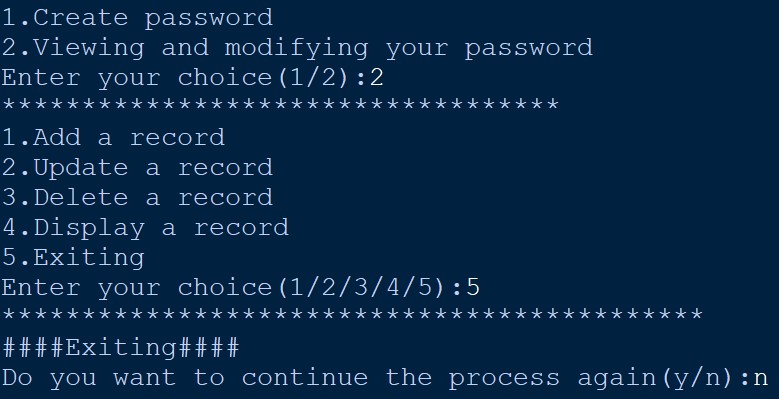
**Displaying all data portal:**



**Displaying data with conditions portal:**



**Exiting portal:**



**FUTURE ENHANCEMENT:**

The future enhancements of password generators are likely to focus on addressing emerging security challenges and improving user experience. Here are some potential directions for future development:

1.**Biometric Integration:** Password generators could integrate with biometric authentication methods such as fingerprint recognition, facial recognition, or iris scans to enhance security and simplify password input.

2.**Machine Learning and AI:** Password generators could leverage machine learning and artificial intelligence to create even more secure and context-aware passwords. They could adapt to evolving threats and user behaviors.

3.**Passwordless Authentication:** The evolution of passwordless authentication methods, like hardware tokens or smartphone-based authentication, may influence how password generators are used and integrated into authentication workflows.

4.**Improved Usability:** Future password generators should prioritize userfriendliness. This includes creating passwords that are not only secure but also easy to enter and remember.

5.**Integration with Password Managers:** Seamless integration between password generators and password managers could further streamline password management, making it easier for users to generate, store, and autofill complex passwords.

6.**Multi-Platform Support:** Password generators should be available and compatible with a wide range of devices and platforms, including mobile devices, IoT devices, and web applications.

7.**Enhanced Security Alerts:** Password generators could provide real-time security alerts to users, notifying them of potential breaches, compromised passwords, or suspicious activities related to their accounts.

8.**Customization and Personalization:** Users might be able to customize password generation rules to align with their preferences while maintaining security standards.

9.**Education and Guidance:** Password generators could include built-in educational features to guide users on password security best practices and offer explanations about the generated passwords’ strength.

10.**Integration with Security Ecosystems:** Password generators could integrate with broader security ecosystems, working alongside antivirus software, firewalls, and other security tools to provide a comprehensive defense against cyber threats.

**Bibliography:**

* Computer Science with Python-Preeti Arora
* Python - https://www.python.org/
* MySQL Documentation - https://dev.mysql.com/doc/
* CodeWithHarry.com