INTRODUCTION
OF
PROJECT

“Password Generator”

1.1 **BACKGROUND**

Passwords protect our online accounts and data. Many people choose weak or easy-to-guess passwords because it’s faster. This makes accounts unsafe. A simple tool that creates strong, random passwords helps everyone stay secure without extra effort.

- In the digital world, we use passwords to protect our accounts and information.
- Many people create weak or easy passwords like "123456" or "password".
- Such passwords can be easily guessed or hacked by attackers.
- Strong passwords should be long, random, and use different types of characters.
- Remembering strong passwords is hard for most people.
- That’s why tools that generate passwords are very useful.
- These tools save time and create secure, unpredictable passwords.
- This project aims to make a simple password generator using Python.
- It helps users choose password length and character types (like A-Z, 0-9, symbols).
- The password is shown on screen and can be copied easily.
- The system also shows how strong the password is (weak, medium, strong).
- This tool is easy to use and makes password creation quick and safe.

1.2 OBJECTIVE

1. **To develop a desktop-based password generator using Python and Tkinter**: The main goal is to create a simple but powerful application that can run on any computer without requiring internet access or heavy software installations. The use of Tkinter ensures a clean and user-friendly graphical interface.
2. **To allow users to customize password features according to their needs**: Users should be able to select the desired password length (between 6 and 32 characters) and choose which character types to include, such as uppercase letters, lowercase letters, digits, and special symbols. This flexibility allows users to generate passwords for different platforms and requirements.
3. **To generate completely random and secure passwords**: The password generation logic must ensure randomness using Python's random module, helping prevent patterns that attackers can predict. Each password should be unique, strong, and difficult to crack using brute-force or dictionary attacks.
4. **To assess the strength of the generated password and provide feedback**: Once a password is created, the system evaluates it based on length and character diversity. The strength is then displayed as "Weak," "Medium," or "Strong," helping users understand the quality of their passwords.
5. **To provide an easy way to copy the generated password to the clipboard**: The application includes a copy button that lets users transfer the password with a single click. This improves usability and allows users to paste the password directly into forms or password managers without manual typing.
6. **To create an application that is lightweight, fast, and works on multiple platforms**: The program should work on Windows, macOS, and Linux systems, as long as Python is installed. It must run smoothly without requiring high-end hardware or additional libraries.
7. **To promote better password practices among users**: By making strong password creation easier, the tool helps users move away from weak or reused passwords and encourages better security habits.
8. **To build a solution that is easy to maintain and extend**: The code should be modular and well-organized so future features like dark mode, password saving, or encryption can be added without rewriting the entire application.

1.3 Proposed System

1.3.1 Purpose

The purpose of this project is to create a secure, user-friendly, and customizable password generator that helps users generate strong passwords instantly.

1. **To improve password security through automation**: Many users create weak or repeated passwords due to difficulty remembering complex ones. This project aims to eliminate that problem by generating strong and random passwords automatically.
2. **To simplify the process of password creation**: Instead of thinking of new passwords every time, users can easily set a few preferences and click a button to get a secure password instantly.
3. **To encourage the use of secure practices**: With clear password strength feedback, the system teaches users to prefer longer and more diverse passwords.
4. **To reduce dependency on online password generators**: This tool works offline, so users can generate secure passwords without worrying about privacy or data exposure.
5. **To create a lightweight solution that anyone can use**: Unlike some password managers that require installation and registration, this simple Python application works with minimal setup and runs on almost any computer.

1.3.2 Scope

The scope of the project defines the extent of features and functionalities this system offers.

1. **To provide a customizable interface for password generation**: Users can choose the length of the password (between 6 and 32 characters) and select which types of characters to include—uppercase letters, lowercase letters, digits, and symbols—based on their needs.
2. **To generate random passwords with high security**: The application uses built-in Python functions to create passwords that are random and unpredictable, ensuring that the generated credentials are secure.
3. **To offer immediate strength evaluation**: The application checks each generated password and shows a strength label (Weak, Medium, Strong) based on length and character variety, helping users understand the quality of their password.
4. **To provide simple clipboard integration**: The generated password can be copied to the system clipboard with one click, allowing users to paste it wherever required without typing errors.
5. **To ensure platform independence**: Since the application is built with Python and Tkinter, it can run on Windows, macOS, and Linux systems that have Python installed.
6. **To exclude long-term storage or account management**: The tool only focuses on password generation and does not store any passwords or sensitive data, which keeps the application lightweight and reduces security risks.
7. **To serve as a base for future improvements**: The system is designed in a modular way, making it easy to add advanced features like password history, dark mode, and secure storage in future versions.

2. SURVEY OF TECHNOLOGY

This project uses several well-known and reliable technologies that are widely supported and easy to implement. Each component was carefully chosen to keep the application lightweight, cross-platform, and user-friendly.

1. **Python Programming Language**: Python is a high-level, interpreted language known for its simplicity and readability. It is widely used in both academic and professional environments. Python's large standard library and active community make it ideal for rapid application development. For this project, Python is used for both the logic (backend) and the graphical user interface (frontend).
2. **Tkinter Library**: Tkinter is Python's standard GUI library. It is built-in, meaning no additional installations are required, and it works across platforms like Windows, macOS, and Linux. Tkinter allows the creation of graphical components such as buttons, sliders, labels, and input boxes. This library was selected because it provides all the tools needed to create a functional and responsive desktop interface.
3. **Random Module**: Python's random module is used to generate passwords by randomly selecting characters from user-selected character sets (uppercase, lowercase, digits, symbols). This module ensures that the generated passwords are unpredictable, which improves security. The random.choices() function makes it easy to create strings of any desired length with random elements.
4. **String Module**: This module provides convenient constants such as string.ascii_letters, string.digits, and string.punctuation. These constants are used to build a custom character set based on the user's selection. It helps reduce errors and makes the code cleaner and more readable.
5. **Clipboard Access (via Tkinter)**: To provide the user with a convenient way to use the generated password, Tkinter's clipboard functions (clipboard_clear() and clipboard_append()) are used. These allow the password to be copied directly into the clipboard with one click, making it easy to paste into other applications without typing errors.
6. **Message Boxes and Notifications**: Tkinter's messagebox module is used to show alerts and confirmations, such as warnings when no character types are selected, or notifications that the password has been successfully copied. These messages improve the user experience by providing clear feedback during interactions.

Together, these technologies allow the creation of a secure, simple, and efficient password generator. They require minimal resources and ensure the application remains accessible to all types of users, even those with little technical background.

3. REQUIREMENTS AND ANALYSIS

3.1 Problem Definition

People need an easy way to get strong passwords without thinking too much.

Many users create weak, easy-to-guess passwords or reuse passwords across multiple sites, making them vulnerable to cyber attacks. There is a need for an application that can instantly create strong passwords.

In today’s digital age, users are expected to create and remember strong passwords for multiple platforms such as email, banking, social media, and more. However, most people end up using weak passwords or the same password across multiple accounts due to convenience. This makes their accounts vulnerable to hacking, especially through brute-force or dictionary attacks. The problem is the lack of a simple, offline, and user-controlled tool to generate strong and secure passwords. The aim of this project is to solve this problem by providing an easy-to-use desktop application that helps users create random and secure passwords quickly, with

3.2 Planning and Scheduling

1. **Requirement Analysis (Week 1)**: The first step was to clearly understand what the project needed to do. This included identifying the features of a password generator, such as selecting password length, choosing character types, and generating random passwords. We also decided that the app should work offline and be easy for anyone to use.
2. **Design Phase (Week 2)**: After the requirements were clear, we planned how the software would look and work. The design included the layout of buttons, labels, and text boxes using Tkinter. We also prepared a basic flow of how the user would interact with the application from start to finish.
3. **Implementation Phase (Week 3 and Week 4)**: In this stage, we started writing the Python code for the application. First, we developed the main logic for generating random passwords using the random and string modules. Then, we created the GUI (Graphical User Interface) using Tkinter. All features such as password strength checking and the copy-to-clipboard button were also added in this stage.
4. **Testing Phase (Week 5)**: After coding was complete, we tested the application to find and fix any bugs or errors. We checked if the password length and character types worked correctly, and whether the copy button and strength meter were accurate. Testing ensured the app works smoothly without crashing.
5. **Final Review and Report Submission (Week 6)**: In the last week, we reviewed the full project and prepared the documentation. Screenshots were taken, the report was written, and final corrections were made before submission.

3.3 Software and Hardware Requirements

Software Requirements

1. **Python 3.7 or above**: Python is the programming language used for this project. It is free, open-source, and works on all operating systems.
2. **Tkinter Library**: Tkinter is the built-in GUI (Graphical User Interface) toolkit in Python. It is used to create the visual part of the application like buttons, labels, and input fields.
3. **Text Editor or IDE**: IDLE (Python 3.13 64 bit) or even Notepad++ can be used to write and run the Python code.
4. **Operating System**: Windows, Linux,the program runs smoothly on any OS that supports Python.
5. **Python Libraries :** random, string, tkinter, pyperclip *(optional for clipboard copy)*

Hardware Requirements

1. **Processor**: Any basic dual-core processor or higher is sufficient.
2. **RAM**: Minimum 1 GB of RAM is enough to run the application without any lag.
3. **Storage**: Less than 50 MB of space is needed to save the Python files and run the project.
4. **Input Devices**: A keyboard and mouse are required to interact with the application.
5. **Display**: A monitor or laptop screen to view the GUI of the application.

This project is designed to be lightweight and does not require any high-end system, making it suitable for students, beginners, and general users.

3.4 Feasibility Study

A feasibility study is done to check if the project is possible and practical. It helps in understanding whether the project can be developed successfully with the available resources and skills. Below are the three main types of feasibility checks done for this project:

3.4.1 Economic Feasibility

1. The project is cost-effective because it uses free and open-source tools like Python and Tkinter.
2. No extra cost—Python and Tkinter are free
3. No need to buy any licenses or expensive software.
4. The hardware needed is very basic and already available with most students or users.
5. Since no server or cloud services are used, there are no running costs. **Conclusion**: The project is economically feasible and can be developed with almost zero cost.

3.4.2 Technical Feasibility

1. Uses stable, built-in Python tools. Easy to set up.
2. The technology used (Python and Tkinter) is simple and easy to learn.
3. These tools are widely supported and work on multiple platforms (Windows, Linux, macOS).
4. The development tools and environment needed are freely available online.
5. Any student with basic programming knowledge can build and maintain the project.
6. **Conclusion**: The project is technically feasible because it does not require advanced skills or special tools.

3.4.3 Operational Feasibility

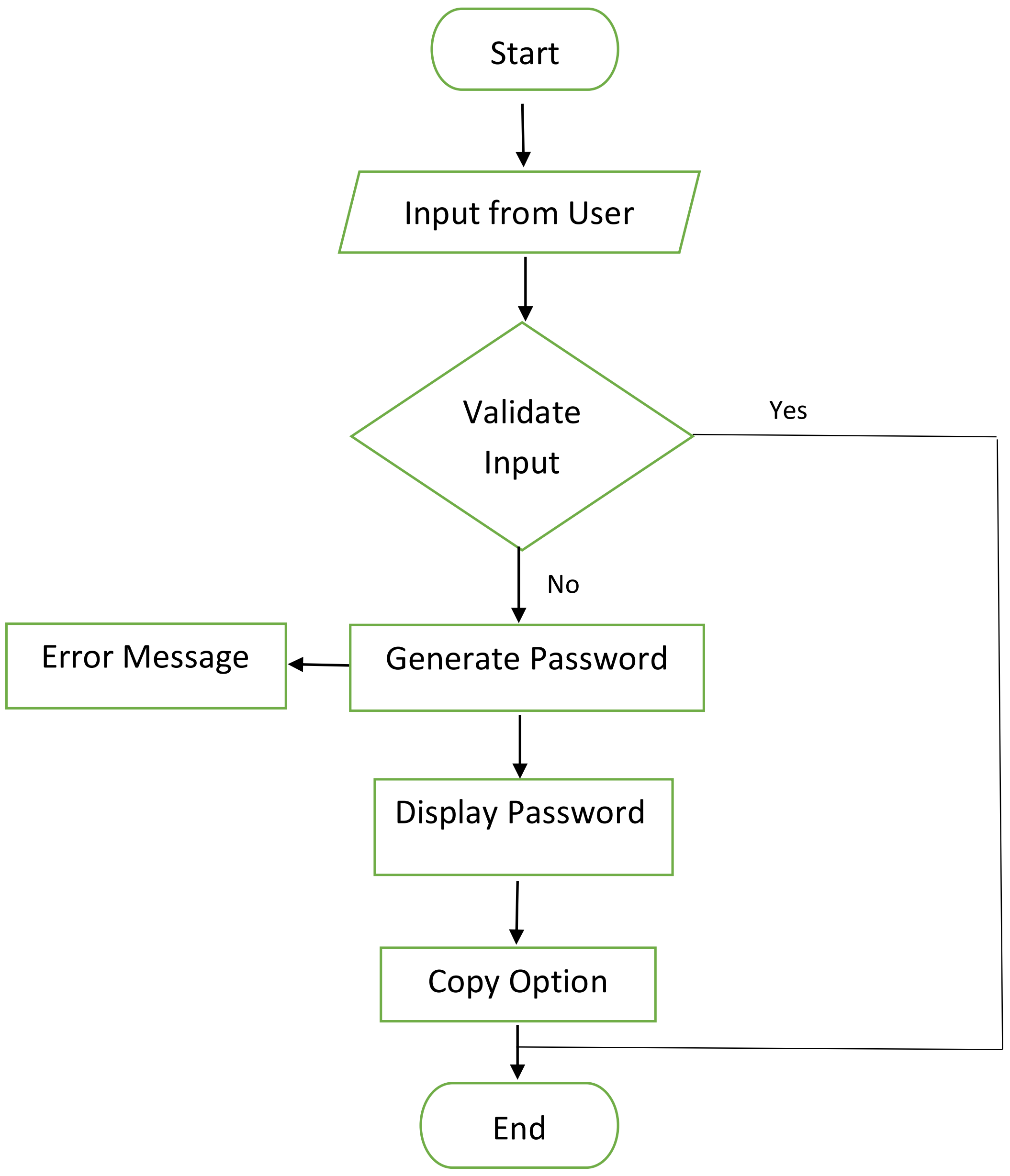
1. The application is very simple to use with buttons and labels clearly shown.
2. Users can generate a password in just a few clicks.
3. There is no need for user login or internet connection.
4. Simple interface. Anyone familiar with basic desktop apps can use it
5. Error messages and confirmations guide the user throughout the process. **Conclusion**: The project is operationally feasible because it is easy to use, runs offline, and needs no special training.

4. SYSTEM DESIGN

4. SYSTEM DESIGN

System design is the process of planning how the software will work internally. It includes how the program flows and how different parts (modules) are connected. It helps to understand the working logic clearly before writing the code.

4.1 Program Flowchart



4.2 Basic Modules

This project is divided into the following main modules to keep it simple and organized:

1. **User Interface Module**
   - Developed using Tkinter
   - Contains the layout of buttons, labels, checkboxes, slider, and display box
   - Helps user select options and interact with the app
2. **Input Handling Module**
   - Takes user selections (length, character types)
   - Checks if input is valid (e.g., at least one character type selected)
   - Shows warning messages for invalid input
3. **Password Generation Module**
   - Uses Python's random and string modules
   - Combines selected characters and generates a password of given length
   - Ensures randomness and complexity
4. **Output Module**
   - Displays the generated password
   - Provides option to copy password to clipboard using Tkinter's clipboard function
5. **Message Display Module**
   - Shows success or warning messages using messagebox
   - Improves user experience and error handling

4.3 Data Design

Since the password generator project is a simple desktop application, it does **not require a complex database**. All operations are performed in memory, and no user data or password history is stored. However, we can still explain data handling in basic terms for academic purposes.

4.3.1 Database Design

This project **does not use any external database** like MySQL or SQLite. All inputs (password length, character types) are taken from the user interface, and the output (generated password) is displayed directly on the screen. If in the future we add features like saving generated passwords, then a small local database can be used.

4.3.2 Data Dictionary and Data Views

Even though there's no formal database, we can define the main data elements used:

Data Element	Description	Type	Source
Length	Length of the password selected by the user	Integer	Slider input
include_uppercase	Checkbox to include uppercase letters	Boolean	Checkbox
include_lowercase	Checkbox to include lowercase letters	Boolean	Checkbox
include_numbers	Checkbox to include numbers	Boolean	Checkbox
include_symbols	Checkbox to include special characters	Boolean	Checkbox
Password	Final generated password	String	Output display

These values are temporarily stored in memory and cleared when the user exits or restarts the program.

4.3.3 E-R diagram and DFDs

Entity-Relationship (E-R) Diagram

Entities:

- User
- Password Generator

Relationships:

- User → inputs preferences → Password Generator
- Password Generator → outputs → password to User

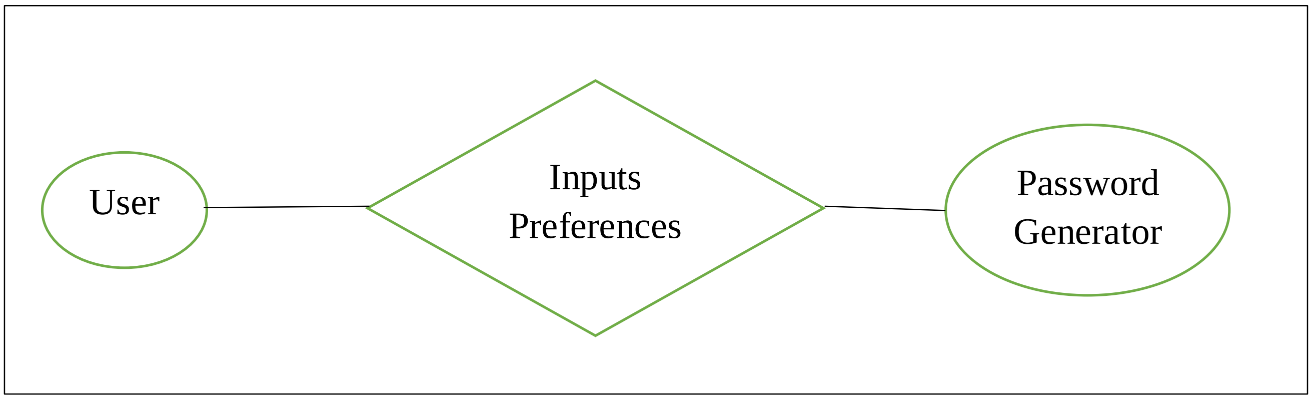
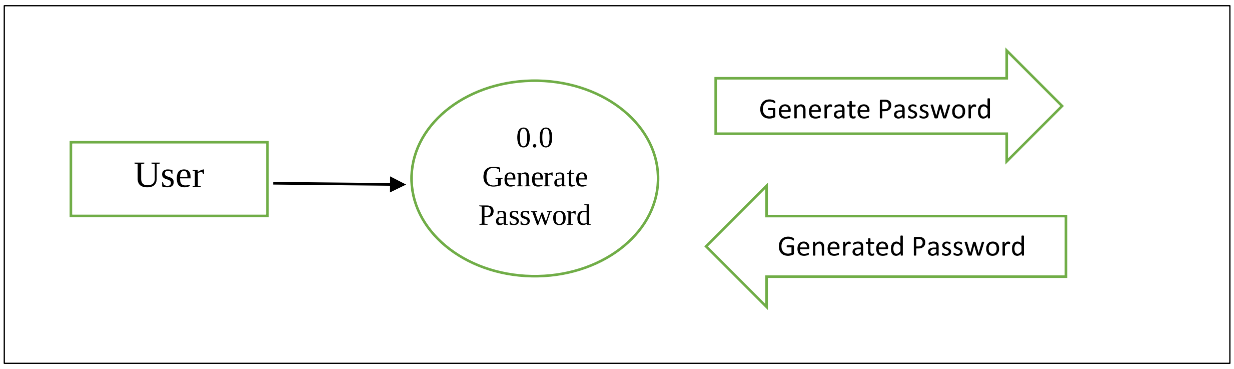


Fig : E R Diagram

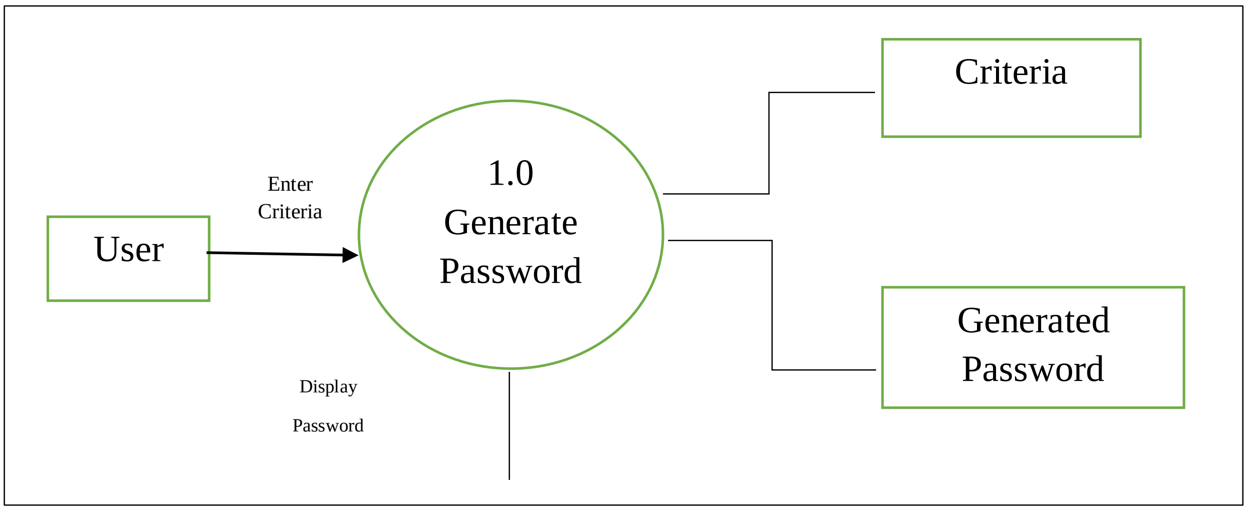
Data Flow Diagram (DFDs):

DFD Level 0: Context Level

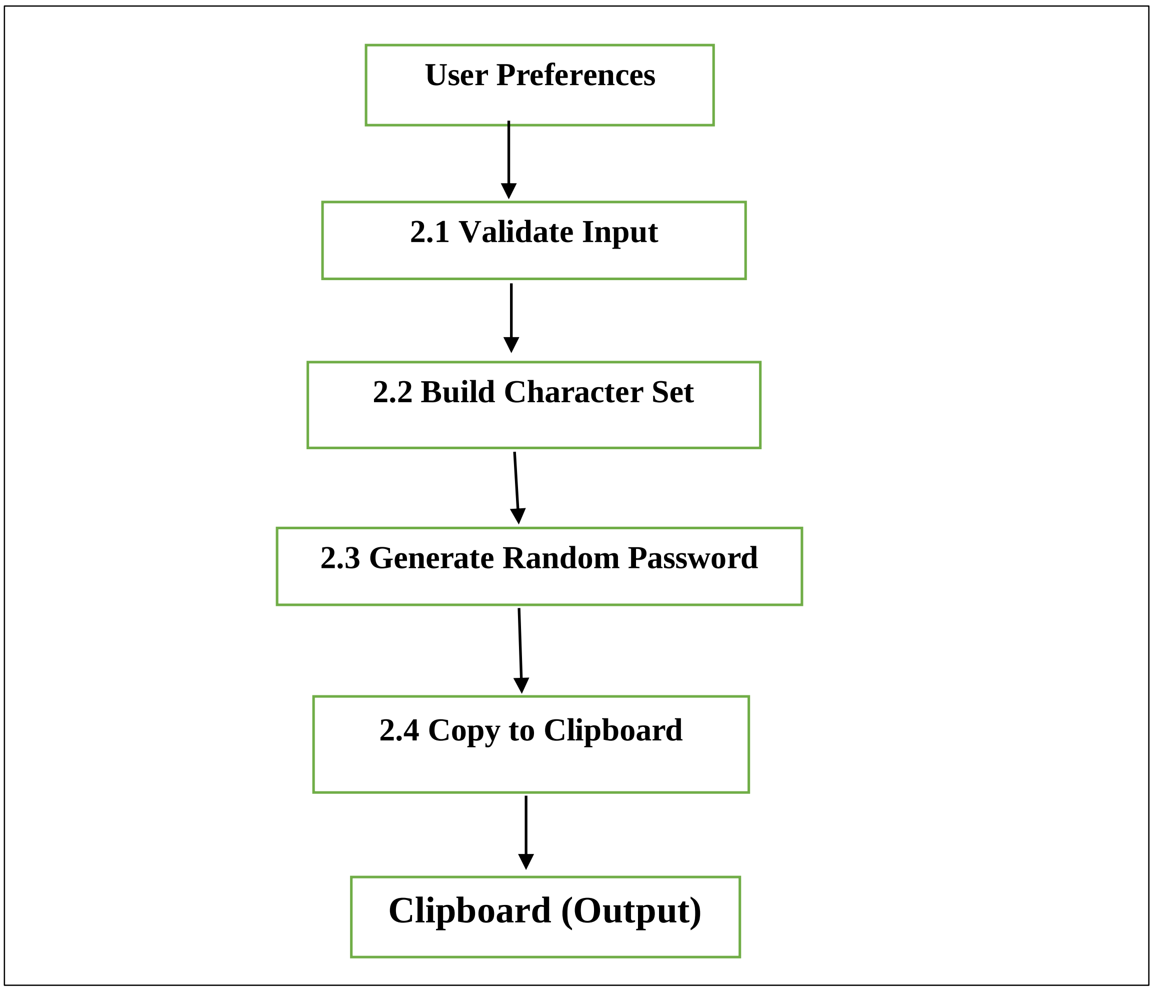


DFD Level 0: Context Diagram

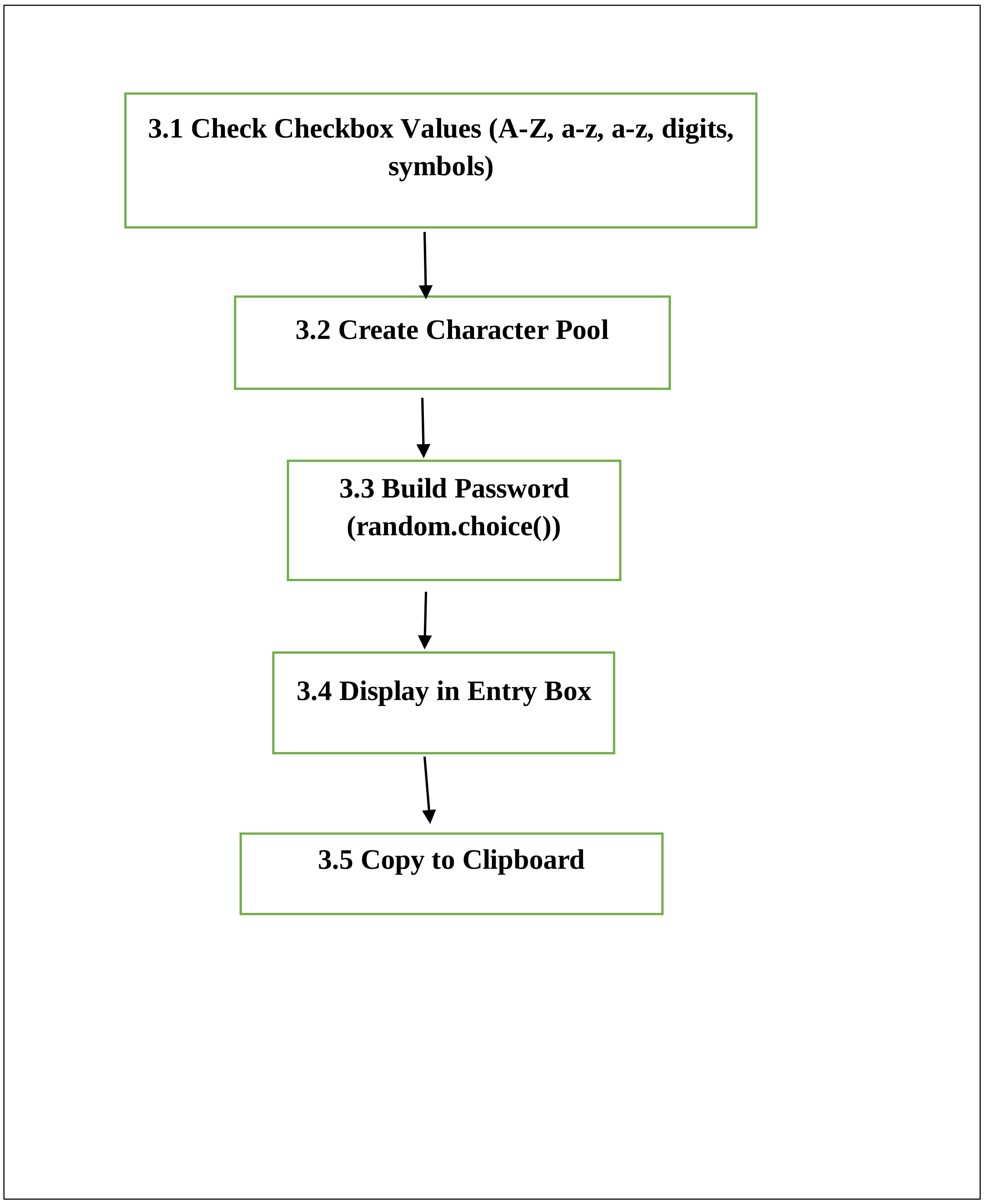
DFD Level 1: Main Subsystems



DFD Level 2: Internal Processing Details



DFD Level 3: Advanced Internal Operations


3.1 Check Checkbox Values (A-Z, a-z, a-z, digits, symbols)
3.2 Create Character Pool
3.3 Build Password (random.choice())
3.4 Display in Entry Box
3.5 Copy to Clipboard

5. IMPLEMENTATION AND TESTING

5.1 Coding Details and screenshots

1. Coding Details

The password generator project is developed using **Python** with a **Tkinter** graphical user interface. The code includes the following important components

Importing Libraries

```
import tkinter as tk
  import random
    import string
```

2. Creating the Main Window

A window is created using Tkinter, with input fields and checkboxes for user options.

3. Password Generation Logic

The password is created based on selected character types (uppercase, lowercase, digits, special characters) and the length provided by the user.

```
def generate_password():

    characters = ""

    if use_upper.get():

        characters += string.ascii_uppercase

    if use_lower.get():

        characters += string.ascii_lowercase
    if use_digits.get():
        characters += string.digits
    if use_special.get():
        characters += string.punctuation

    length = int(length_entry.get())
    password = "".join(random.choice(characters) for _ in range(length))
    result_entry.delete(0, tk.END)
    result_entry.insert(0, password)
```

4. Copy to Clipboard Functionality

```
def copy_to_clipboard():
    root.clipboard_clear()
    root.clipboard_append(result_entry.get())
```

5. GUI Components

Buttons, Labels, and Checkbuttons are arranged on the window to allow user interaction.

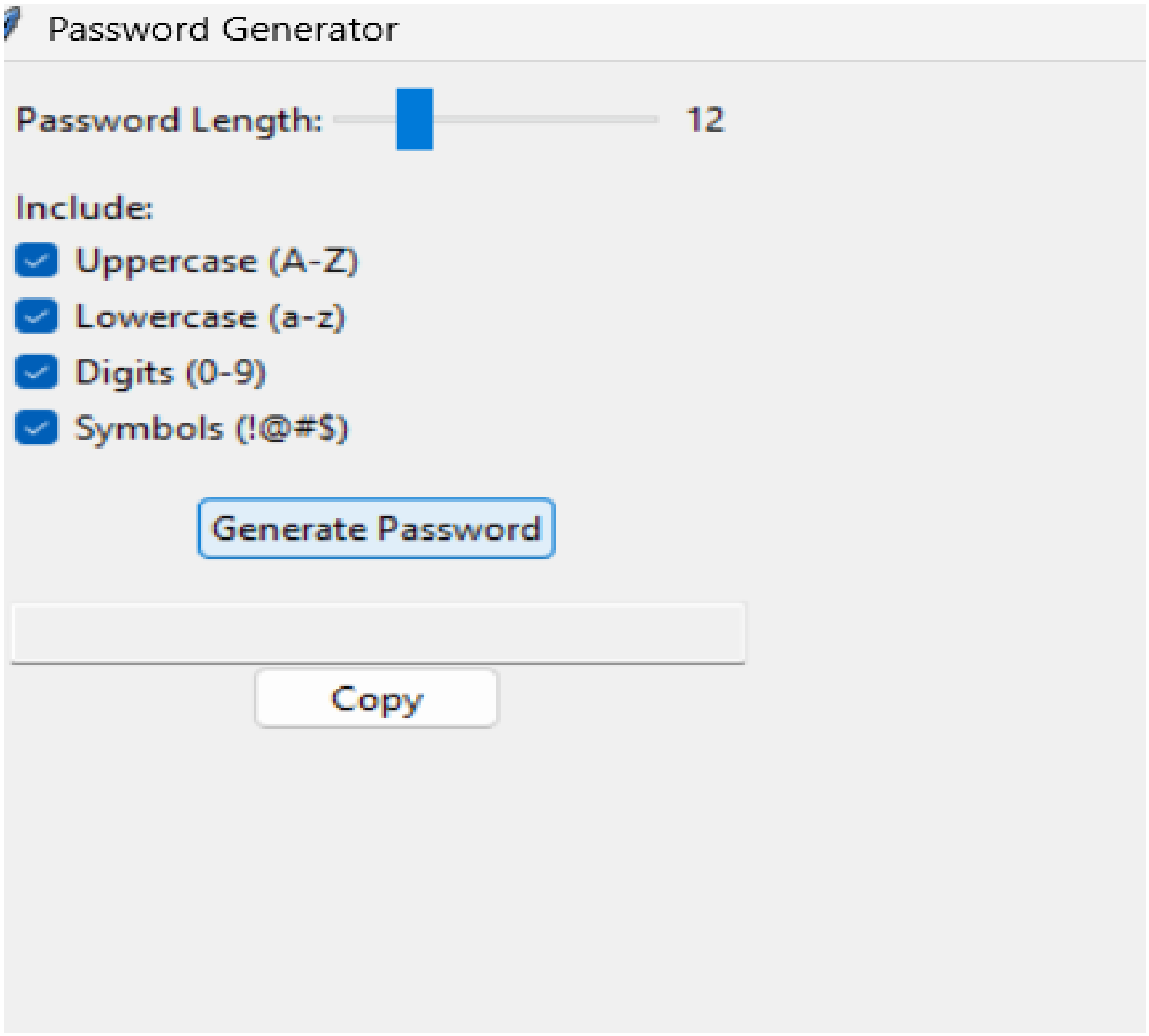
Source Code:

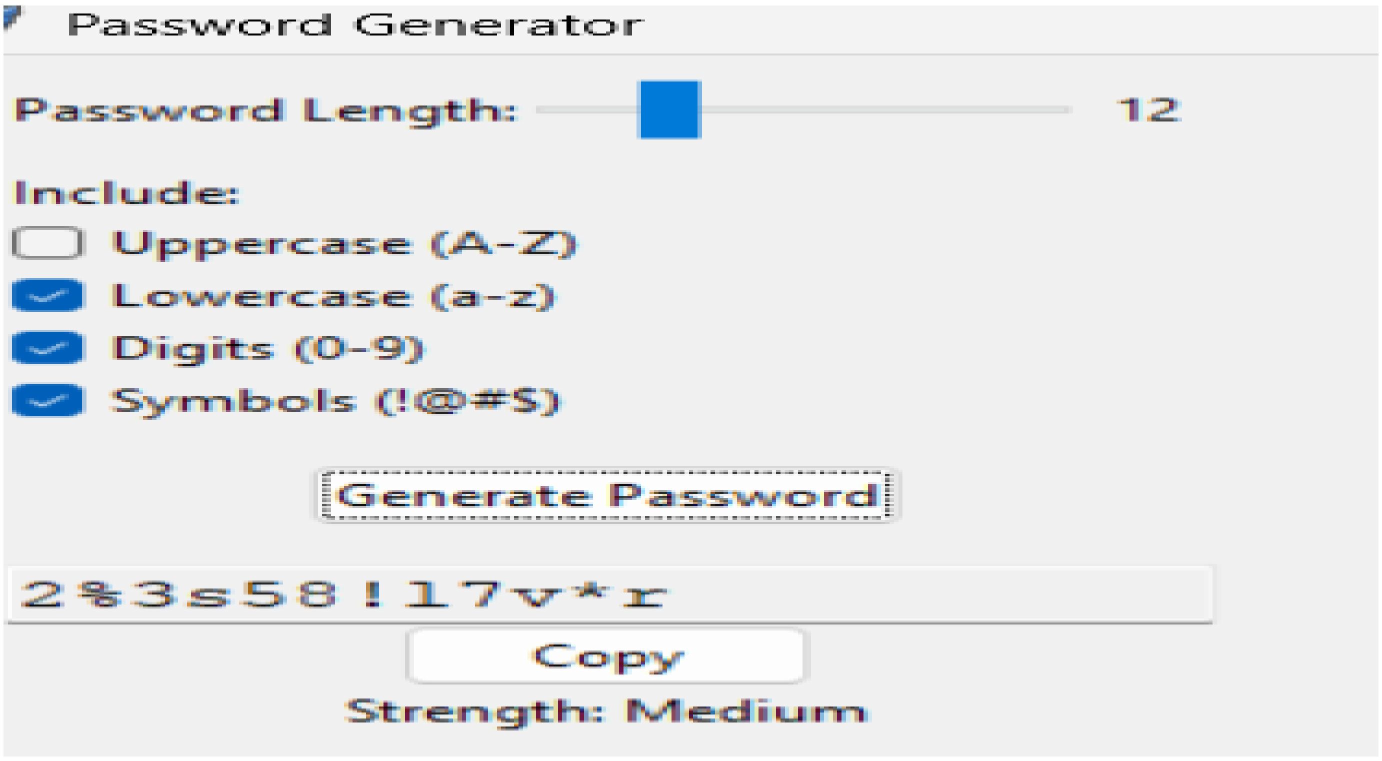
```
import tkinter as tk
from tkinter import ttk, messagebox
```

```
import random

import string

class PasswordGeneratorApp:

    def __init__(self, root):

        self.root = root

        self.root.title("Password Generator")

        self.root.geometry("400x450")



        # Variables

        self.length_var = tk.IntVar(value=12)

        self.password_var = tk.StringVar()

        self.upper_var = tk.BooleanVar(value=True)

        self.lower_var = tk.BooleanVar(value=True)

        self.digits_var = tk.BooleanVar(value=True)

        self.symbols_var = tk.BooleanVar(value=True)



        # Create UI

        self.create_widgets()



    def create_widgets(self):

        # Main container
        mainframe = ttk.Frame(self.root, padding="10")

        mainframe.pack(fill=tk.BOTH, expand=True)



        # Length control

        ttk.Label(mainframe, text="Password Length:").grid(row=0, column=0, sticky=tk.W)

        ttk.Scale(

            mainframe,

            from_=6,

            to=32,

            orient=tk.HORIZONTAL,

            variable=self.length_var,

            command=lambda x: self.length_var.set(round(float(x)))

        ).grid(row=0, column=1, sticky=tk.EW)

        ttk.Label(mainframe, textvariable=self.length_var).grid(row=0, column=2, padx=5)



        # Character types

        ttk.Label(mainframe, text="Include:").grid(row=1, column=0, sticky=tk.W, pady=(10,0))

        ttk.Checkbutton(mainframe, text="Uppercase (A-Z)", variable=self.upper_var).grid(row=2,
column=0, columnspan=3, sticky=tk.W)

        ttk.Checkbutton(mainframe, text="Lowercase (a-z)", variable=self.lower_var).grid(row=3,
column=0, columnspan=3, sticky=tk.W)
```

```
        ttk.Checkbutton(mainframe, text="Digits (0-9)", variable=self.digits_var).grid(row=4, column=0, columnspan=3, sticky=tk.W)

        ttk.Checkbutton(mainframe, text="Symbols (!@#$)", variable=self.symbols_var).grid(row=5, column=0, columnspan=3, sticky=tk.W)



        # Generate button

        ttk.Button(mainframe, text="Generate Password", command=self.generate_password).grid(row=6, column=0, columnspan=3, pady=15)



        # Password display

        ttk.Entry(mainframe, textvariable=self.password_var, font=('Courier', 12), state='readonly').grid(row=7, column=0, columnspan=3, sticky=tk.EW)



        # Copy button

        ttk.Button(mainframe, text="Copy", command=self.copy_password).grid(row=8, column=0, columnspan=3)



        # Strength indicator

        self.strength_var = tk.StringVar(value="")

        ttk.Label(mainframe, textvariable=self.strength_var).grid(row=9, column=0, columnspan=3)



    def generate_password(self):

        if not any([self.upper_var.get(), self.lower_var.get(), self.digits_var.get(), self.symbols_var.get()]):
            messagebox.showerror("Error", "Please select at least one character type!")
            return

        chars = []
        if self.upper_var.get(): chars.extend(list(string.ascii_uppercase))
        if self.lower_var.get(): chars.extend(list(string.ascii_lowercase))
        if self.digits_var.get(): chars.extend(list(string.digits))
        if self.symbols_var.get(): chars.extend(list("!@#$%^&*"))

        length = self.length_var.get()
        password = ''.join(random.choices(chars, k=length))
        self.password_var.set(password)
        self.assess_strength(password)

    def assess_strength(self, password):
        length = len(password)
        complexity = sum([
            1 if any(c in string.ascii_uppercase for c in password) else 0,
            1 if any(c in string.ascii_lowercase for c in password) else 0,
            1 if any(c in string.digits for c in password) else 0,
            1 if any(c in "!@#$%^&*" for c in password) else 0
        ])

        strength = min(100, (length * 3) + (complexity * 10))

        if strength < 40:
            self.strength_var.set("Strength: Weak")
        elif strength < 70:
            self.strength_var.set("Strength: Medium")
        else:
            self.strength_var.set("Strength: Strong")
    def copy_password(self):
        if self.password_var.get():
            self.root.clipboard_clear()
            self.root.clipboard_append(self.password_var.get())
            messagebox.showinfo("Copied", "Password copied to clipboard!")
        else:
            messagebox.showwarning("Error", "No password to copy")
if __name__ == "__main__":
    root = tk.Tk()
    app = PasswordGeneratorApp(root)
    root.mainloop()
```

Screenshot


Password Generator
Password Length:
12
Include:
Uppercase (A-Z)
Lowercase (a-z)
Digits (0-9)
Symbols (!@#$)
Generate Password
Copy



Password Generator

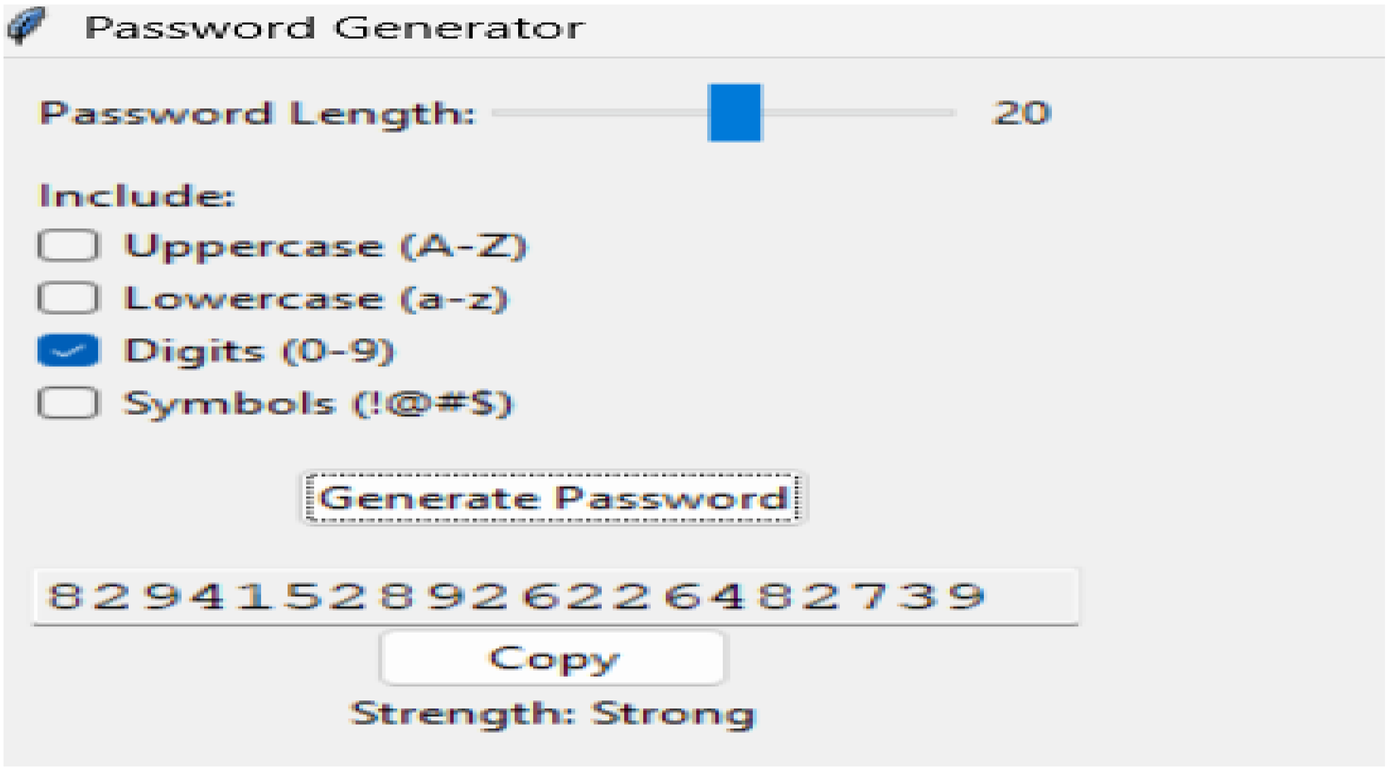
Password Length: 12

Include:

- [ ] Uppercase (A-Z)
- [✓] Lowercase (a-z)
- [✓] Digits (0-9)
- [✓] Symbols (!@#$)

```
2%3s58!17v*r
```

Strength: Medium



Password Generator

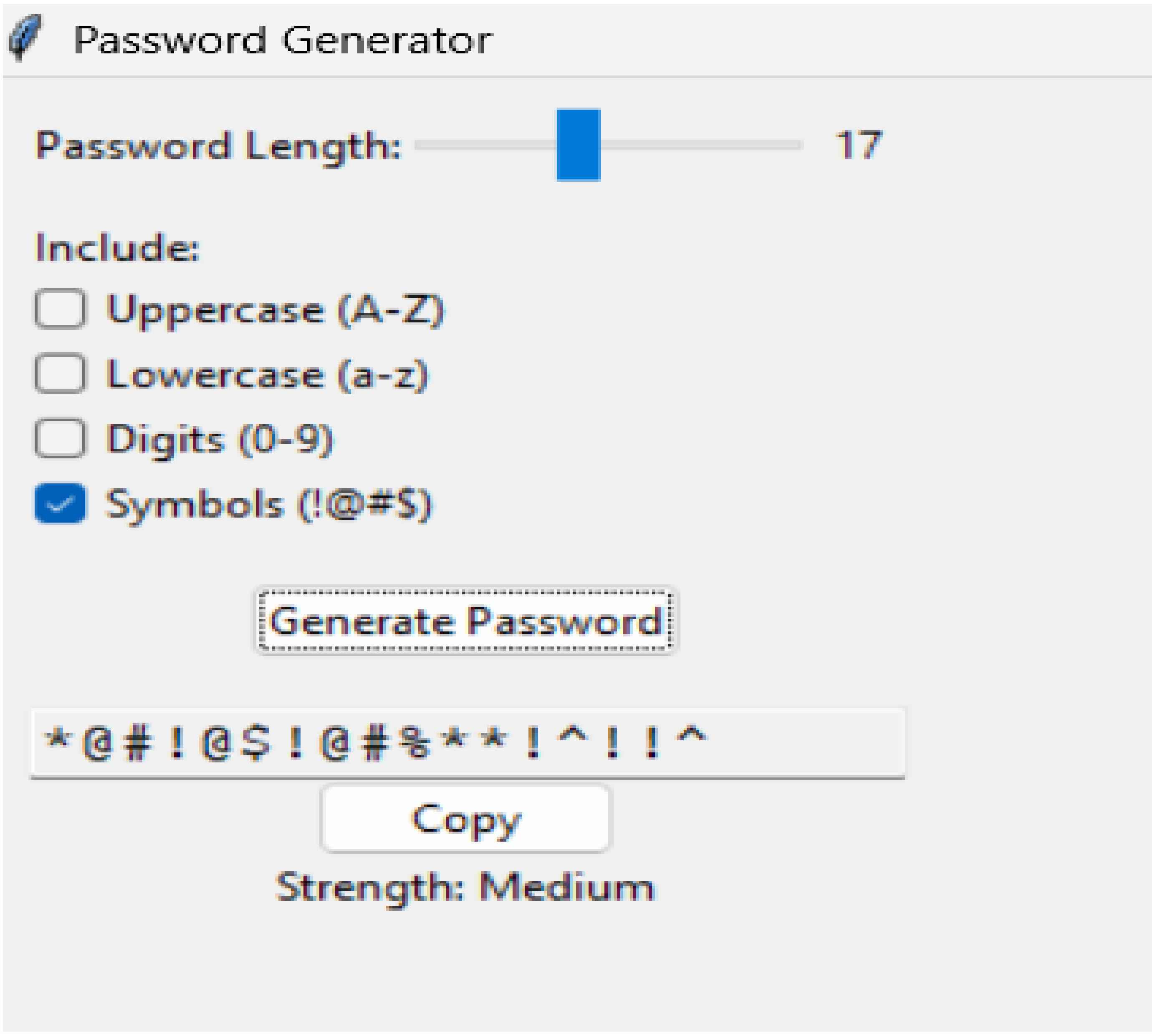
Password Length: 20

Include:

- [ ] Uppercase (A-Z)
- [ ] Lowercase (a-z)
- [✓] Digits (0-9)
- [ ] Symbols (!@#$)

```
82941528926226482739
```

Strength: Strong


Password Generator

Password Length: 17

Include:

- [ ] Uppercase (A-Z)
- [ ] Lowercase (a-z)
- [ ] Digits (0-9)
- [✓] Symbols (!@#$)

```
*@#!@$!@#%**!^!!^
```

Strength: Medium

5.2 Testing Approach:

Testing is an important phase in software development that helps ensure the application works correctly and meets the user's requirements. In this project, we used three levels of testing: **Unit Testing**, **Integrated Testing**, and **System Testing**. Each level helped verify the accuracy and reliability of different parts of the Password Generator application.

5.2.1 Unit Testing

- ✓ Each function (like password generation, strength check, copy to clipboard) was tested separately.
- ✓ The generate_password() function was tested for correct length and character mix.
- ✓ The password strength logic was checked using different password types.
- ✓ Copy feature was verified to ensure it worked correctly.

5.2.2 Integrated Testing

- ✓ Tested how GUI elements (buttons, sliders, checkboxes) interact with backend functions.
- ✓ Verified that selected options correctly affected password output.
- ✓ Confirmed that changes in length or character type immediately reflected in output.
- ✓ Ensured real-time communication between password generation and strength label.

5.2.3 System Testing

- Tested the complete application from start to end as a user would use it.
- Ensured all components (GUI + Logic) worked smoothly together.
- Verified the app works on different systems with no errors.
- Checked UI responsiveness and user-friendliness under different settings.

5.3 Implementation and Maintenance

Implementation

- The project was developed using **Python** with the **Tkinter** library for the graphical user interface (GUI).
- The GUI includes checkboxes, sliders, buttons, and entry fields to interact with the user.
- The random and string libraries were used to generate secure passwords based on user choices.
- Features like password strength checking and copy-to-clipboard were implemented for better usability.
- After coding, the application was tested to ensure all functions worked correctly and smoothly.

Maintenance

- The code is written in a simple and modular way, making it easy to update in the future.
- Maintenance includes fixing bugs, improving the user interface, or adding new features (e.g., dark mode, password saving).
- The app can be easily updated to stay compatible with newer Python versions or operating systems.
- Feedback from users can be used to make the application better over time.
- Overall, it is a lightweight, easy-to-maintain tool for secure password generation.

6 .CONCLUSIONS

6.1 Limitations of the System

- The application does not store or save passwords for future use.
- There is no option for generating passphrases or custom password patterns.
- Limited design – no dark mode or theme options.
- The app is only available in desktop GUI format and not web or mobile compatible.

6.2 Conclusion

- The Password Generator project successfully meets the need for creating strong, secure passwords.
- It allows users to select password length and character types easily through a user-friendly interface.
- The use of Python and Tkinter made development fast, flexible, and effective for beginners.
- With features like password strength display and copy-to-clipboard, it adds both security and convenience.
- The project helped us understand GUI development, random logic, and secure programming practices.

6.3 Future Scope of the Project

- Add features like password history or auto-save for convenience.
- Convert it to a web or mobile app for wider accessibility.
- Implement advanced encryption and secure storage options.
- Add voice commands or accessibility support for differently-abled users.
- Improve the design with themes, animations, and more advanced UI elements.
- Save password history securely.
- Dark mode and language support

7. BIBLIOGRAPHY

I. **Python Official Documentation**

https://docs.python.org/3/

Used to understand core Python syntax, libraries like `random`, `string`, and best practices.

II. **Tkinter GUI Documentation**

https://tkdocs.com/

Referred for building GUI elements such as buttons, labels, sliders, and input fields.

III. **GeeksforGeeks - Python Projects**

https://www.geeksforgeeks.org/python-projects/

Used for ideas and guidance on small-scale Python project structure.

IV. **W3Schools Python Tutorial**

https://www.w3schools.com/python/

Referred for basic Python functions, loops, and GUI development references.

V. **Stack Overflow**

https://stackoverflow.com/

Helped troubleshoot issues and understand user-contributed solutions during development.

VI. **Real Python – GUI Programming with Tkinter**

https://realpython.com/python-gui-tkinter/

Used to enhance knowledge on creating interactive Python desktop applications.