Project Statement:

Secure Password Generator and Manager

1. Introduction

This project, titled "Secure Password Generator and Manager," represents a desktop application built with Python and its standard Graphical User Interface (GUI) library, Tkinter. The primary goal is to provide users with a reliable tool for generating cryptographically strong passwords and securely managing their credentials in a localized environment. The application addresses the common security vulnerability of using weak, reused passwords across multiple services.

2. Purpose and Objectives

The core objectives of this project to:

- **Mitigate Security Risks:** To actively combat poor password practices by automating the creation of high-entropy, complex passwords that are difficult to guess or crack.
- **Develop a User-Friendly Interface:** To design an intuitive and responsive GUI that will ease the process of credential management for non-technical users.
- **Core Python Skills:** Demonstrate core Python programming competencies including:
    - Development of GUIs using Tkinter.
    - Data persistence using File I/O operations.
    - Randomization algorithms for generating strong passwords-Implementation of the random module.
    - Error handling and input validation.

3. Methodology

The application is divided into three main modules: GUI, Password Generation, and Data Persistence.

3.1. User Interface (UI) Design

The GUI is implemented by means of the tkinter library. It contains a canvas for displaying an application logo and a grid layout to arrange the input fields for Website, Email/Username, and Password. Interactive elements include:

- **Generate Password Button:** Triggers the randomization function.
- **Add Button:** Triggers input validation and the process for saving data.
- **tkinter.messagebox:** Used for user confirmation (before saving) and reporting errors (for empty fields).

3.2. Password Generation Algorithm

The password generation function uses Python's random module in order to assure non-deterministic, high-quality passwords.

1. **Character Pools:** Defines three different pools of characters: letters, including uppercase and lowercase, numbers, and symbols.
2. **Random Lengths:** Randomly determines how many characters shall be pulled from each pool, such as 8-10 letters, 2-4 symbols, 2-4 numbers.
3. **Combination:** The selected characters are combined into one list.
4. **Shuffling:** The combined list is inarguably thoroughly shuffled to maximize randomness, using the random.shuffle() function in order for the character types not to appear in groups, such as all letters followed by all numbers.
5. **Output:** The final list is joined into a string and displayed in the password field.

3.3. Data Persistence (Save Function)

The application handles data saving as follows:

1. **Validation Check:** This checks that both **Website** and **Password** fields hold content. It warns the user if validation fails.
2. **User Confirmation:** A confirmation dialog (ask ok cancel) presents the user with the list of credentials before writing to a file acting as a final check.

3. **File Write Operation:** The credentials are written to a local file upon confirmation: python project 2 updated.txt. The data is in a pipe-separated values format for better readability and possible parsing in future updates: Website | Email | Password\n.

4. Conclusion and Future Scope

With this Password Manager, it successfully provides a functional desktop utility for the generation and management of credentials. Although in its current form it uses local plaintext storage-a good enough solution for a demonstration of a beginner-level project-the next important step toward deploying the software in the real world would be to implement an **AES-256-encrypted** storage file, guarded by a master password, in order to ensure proper data confidentiality. Other possible future improvements include a search/retrieval feature and a dedicated password strength meter.