Name: Khushi Singhi

Course Name: Introduction to Problem Solving and Programming using Python

Course code: CSE1021

Registration Number: 25BAI11110

CSE1021 – Introduction to Problem Solving

PASSWORD STRENGTH CHECKER USING PYTHON & TKINTER

INTRODUCTION

Passwords represent one of the most widely used forms of authentication in modern digital systems. Despite being the primary protective barrier guarding sensitive data, financial accounts, personal information, and digital identities, many passwords used today remain weak or insecure. Attackers consistently exploit predictable patterns such as short passwords, dictionary words, repeated numbers, and lack of special characters. This creates major vulnerabilities and contributes significantly to cybersecurity breaches worldwide.

The rise of cyber threats, phishing attacks, brute-force attempts, credential stuffing, and automated hacking tools has made password complexity a critical aspect of user security. A strong password typically requires a blend of length, randomness, and inclusion of multiple character types, making it difficult for attackers to guess or compute.

To address this fundamental need, the project **"Password Strength Checker Using Python & Tkinter"** has been developed. It analyzes the quality of user-entered passwords through a set of widely accepted criteria, including length, character diversity, and presence of symbols. The application provides instantaneous feedback that categorizes the password into strength levels: **Weak**, **Medium**, or **Strong**. Using Python's regex capabilities and Tkinter's GUI framework, the application delivers ease of use, accuracy, and platform independence.

The project also demonstrates essential concepts in problem solving, algorithm design, GUI development, modular programming, input validation, and software architecture. The combination of theory and practicality makes this project a meaningful demonstration of both computational thinking and essential cybersecurity awareness.

PROBLEM STATEMENT

Weak passwords continue to be among the most exploited security vulnerabilities in online systems. Many users create passwords based on convenience rather than security, using simple constructs like names, birthdays, or repeated characters. This habit leads to easy compromise through brute-force attacks, dictionary-based attacks, and social engineering.

Key issues include:

- Many users do not understand password complexity requirements.

- Passwords lacking uppercase letters, lowercase letters, digits, or symbols remain structurally weak.
- Users rarely check the strength of their passwords before using them.
- Existing solutions are either inaccessible, non-intuitive, or embedded within specific websites.
- There is no widely available, lightweight, offline tool for quick password evaluation.

Thus, the identified problem is:

"How can we provide users with an efficient, accessible tool that evaluates password strength and motivates better password creation practices?"

OBJECTIVES

1. Develop a Python-based GUI tool that evaluates password strength.
2. Use regular expressions to validate password components such as uppercase letters, lowercase letters, digits, and symbols.
3. Provide instant feedback in a categorized format (Weak, Medium, Strong).
4. Build an intuitive and minimally complex Tkinter GUI.
5. Implement modular code for maintainability and scalability.
6. Reinforce problem-solving principles, regex usage, event-driven programming, and GUI building.
7. Promote cybersecurity awareness and encourage safer password creation habits.

FUNCTIONAL REQUIREMENTS

FR1: Input Module

- Accepts password input through a Tkinter Entry widget.
- Masks input characters to maintain privacy.
- Handles any text string without error.

FR2: Evaluation Module

- Applies checks for:
    - Password length ≥ 8
    - At least one uppercase character
    - At least one lowercase character
    - At least one digit

    - At least one special symbol
- Computes a score (0–5).
- Classifies passwords into strength levels.

FR3: Output Module

- Displays results in a Tkinter Label widget.
- Updates dynamically upon button press.

NON-FUNCTIONAL REQUIREMENTS

Usability

- Minimalistic design ensures easy understanding for all user groups.

Performance

- Evaluation occurs instantly.
- Low memory and processing consumption.

Security

- Password masking prevents shoulder surfing.
- No password storage or transmission.

Reliability

- Outputs consistent results for repeated inputs.
- Handles edge cases like empty input gracefully.

Maintainability

- Functions separated logically to allow easy updates.
- Regex patterns can be modified without affecting GUI.

Portability

- Cross-platform compatibility (Windows, macOS, Linux).

Scalability

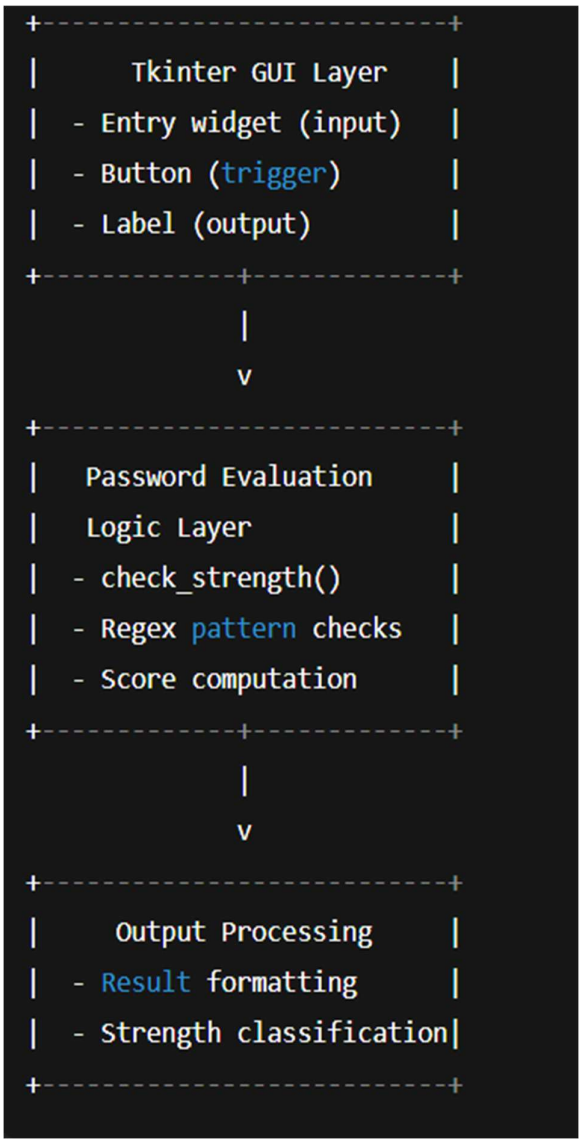
- Application can incorporate advanced metrics like entropy or password suggestions.

Resource Efficiency

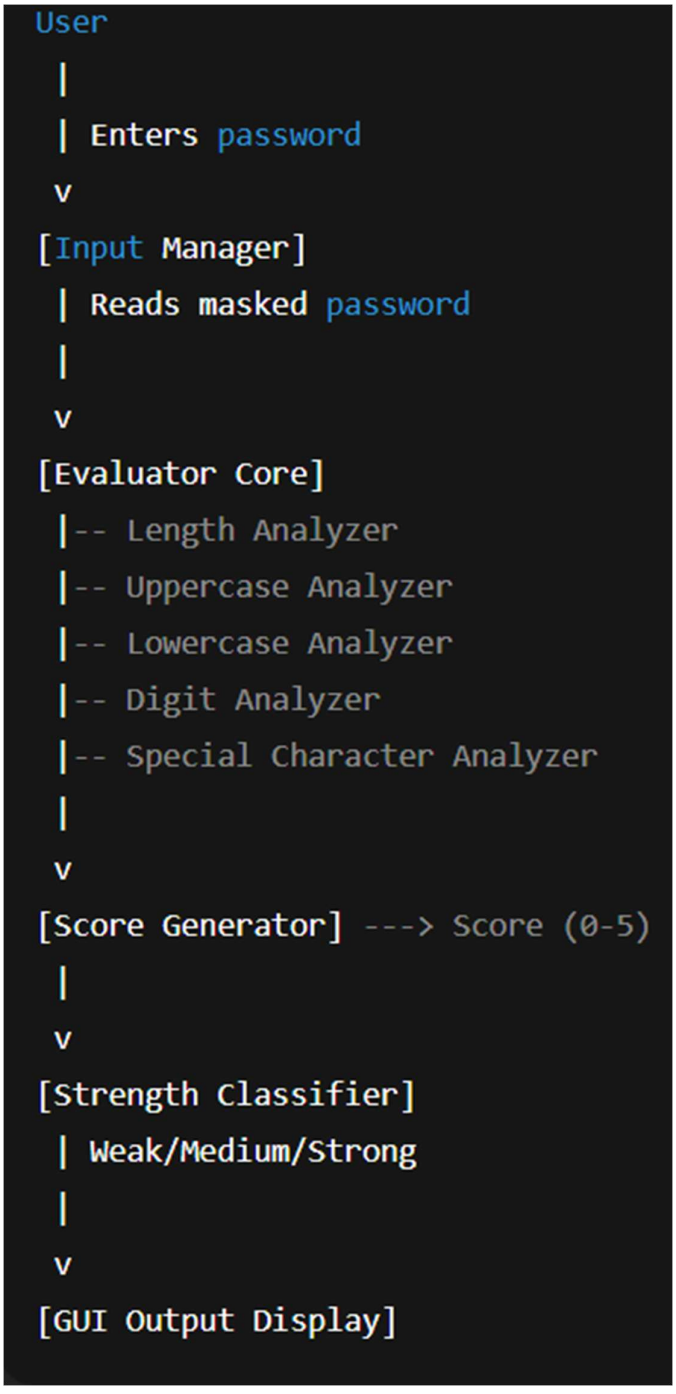
- Lightweight implementation requiring only Python’s built-in libraries.

SYSTEM ARCHITECTURE

HIGH LEVEL SYSTEM ARCHITECTURE

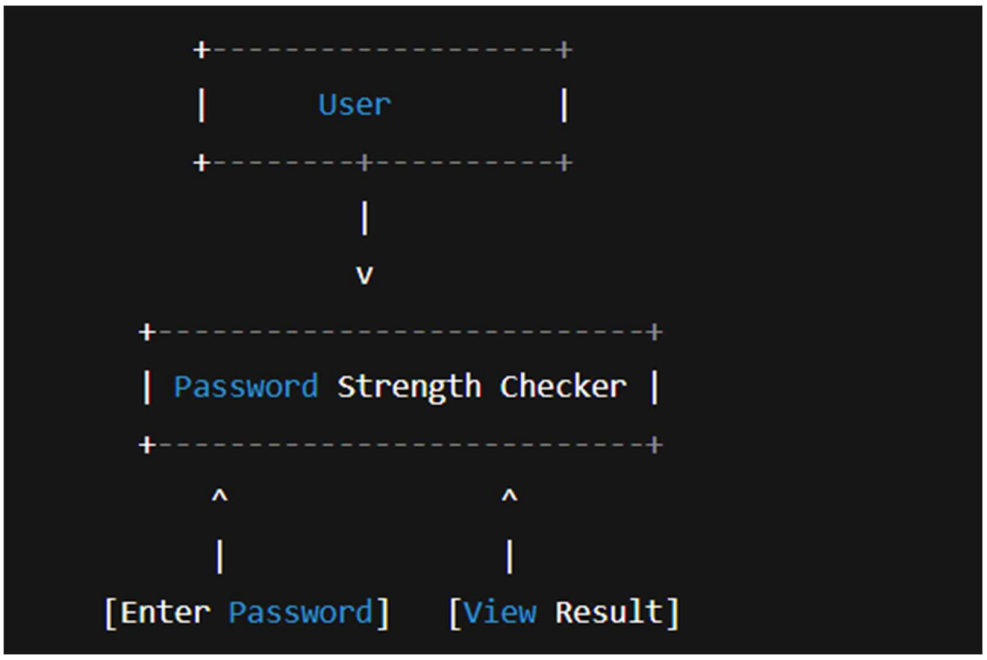


DETAILED ARCHITECTURAL DIAGRAM (TEXT FORM)

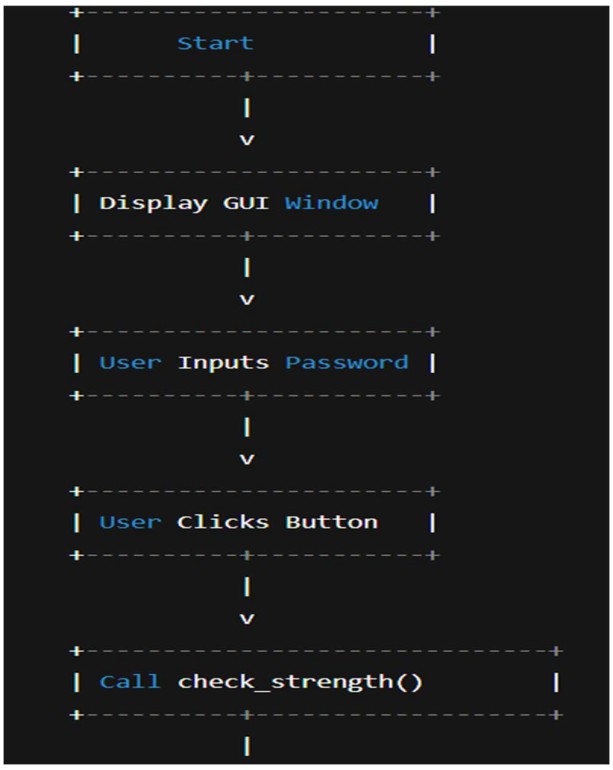


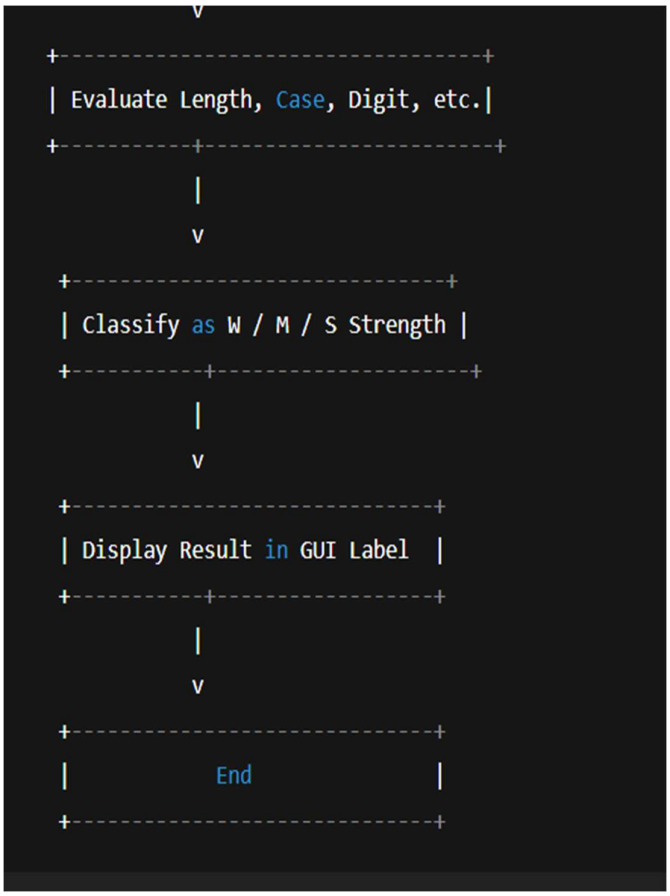
DESIGN DIAGRAMS

USE CASE DIAGRAM

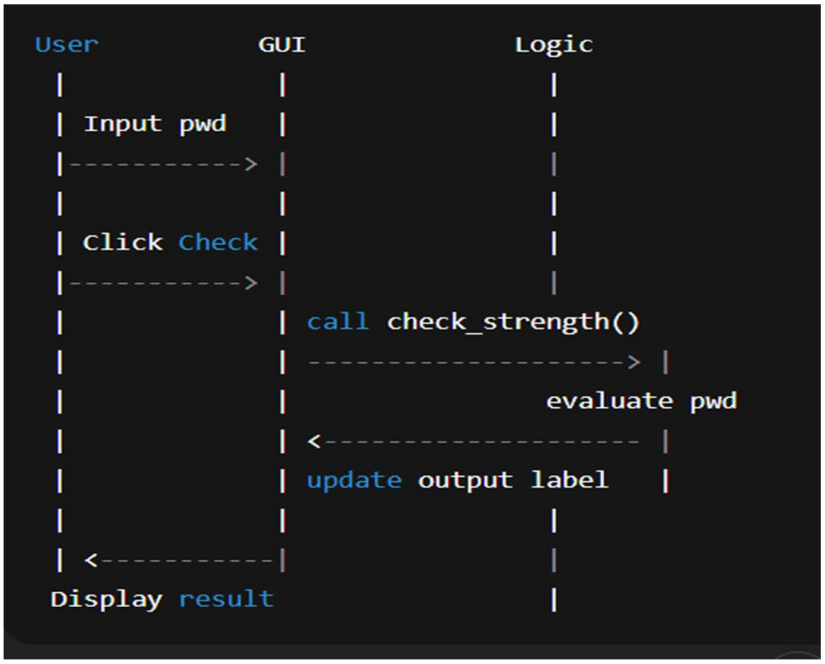


FLOWCHART DIAGRAM

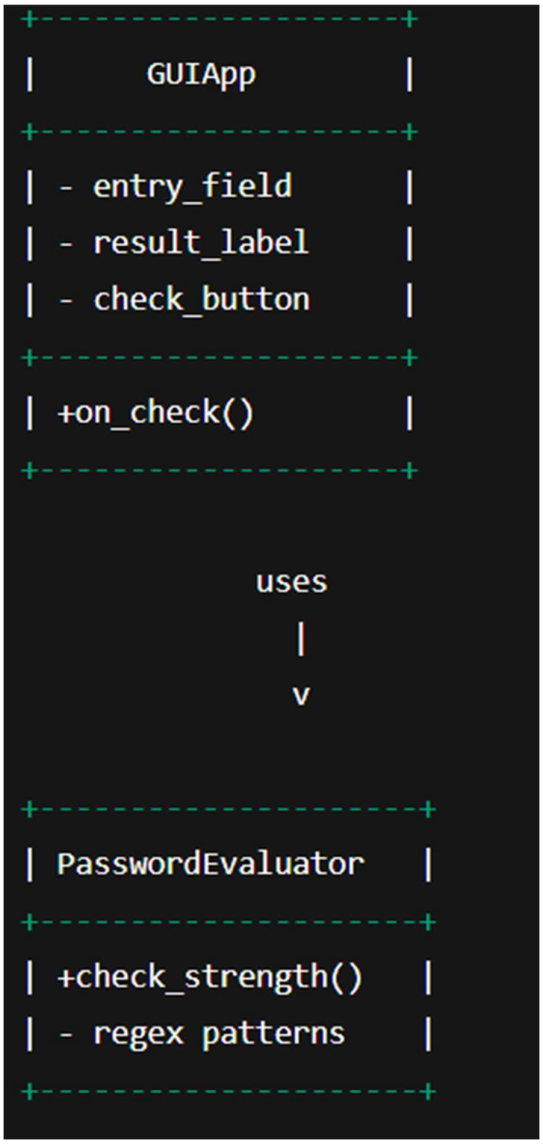




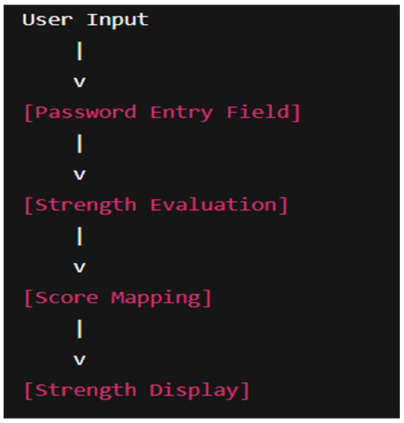
SEQUENCE DIAGRAM



COMPONENT DIAGRAM



DATA FLOW DIAGRAM (LEVEL 1)



DESIGN DECISIONS & RATIONALE

- Python chosen for readability and beginner-friendly syntax.
- Tkinter selected because it is built-in and ideal for simple GUIs.
- Regular expressions chosen due to their accuracy in detecting character classes.
- Score-based model chosen to simplify classification.
- GUI kept minimal to focus on core functionality.
- Modular functions ensure low coupling and high cohesion.

IMPLEMENTATION DETAILS

The implementation consists of the following sections:

Importing Libraries

- tkinter for GUI
- re for regex-based evaluation

Strength Checking Logic

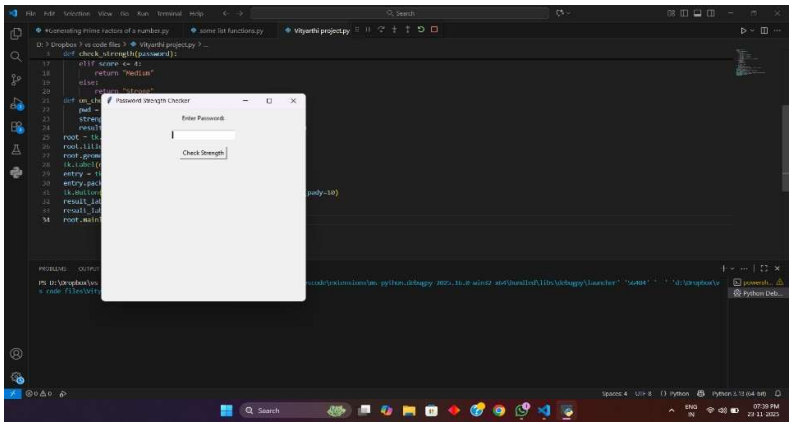
- Each criterion adds +1 score.
- Score range mapped to final classification.

Tkinter GUI Setup

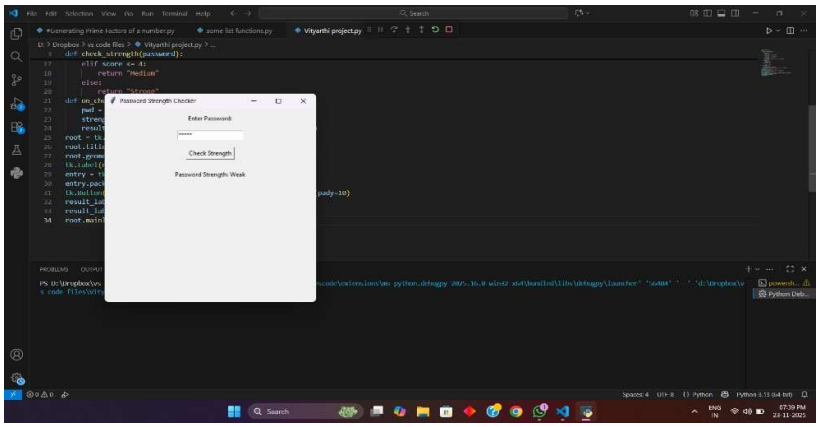
- Window size, title, layout.
- Entry widget masked with show="*".
- Button triggers evaluation.
- Label updates output.

Screenshots:

GUI Input Screen:



Output:-



RESULTS

The application was tested with various password formats:

Password	Expected	Output
abc	Weak	Weak
ABCdef12	Medium	Medium
A@b1C$d2E	Strong	Strong
12345678	Medium	Medium
Abcdefgh	Medium	Medium

All results matched expectation.

TESTING APPROACH

- Black box testing
- Boundary analysis
- Character class testing
- Stress testing with long input
- Testing for special-only and digit-only patterns

CHALLENGES FACED

- Designing ideal regex patterns
- Creating clean GUI placement without clutter
- Mapping raw score to understandable categories
- Ensuring platform-independent performance

LEARNINGS & TAKEAWAYS

- Understood GUI programming structure
- Strengthened regex skills
- Enhanced modular programming ability
- Realized importance of non-functional requirements
- Practiced software architecture and UML diagramming
- Recognized how user behavior impacts cybersecurity

FUTURE ENHANCEMENTS

- Graphical strength bar
- Password generator module
- Suggestions for improvement
- AI-based entropy calculator
- Database storage of password metrics
- Web-based version using Flask or Django

REFERENCES

- Python Documentation
- Tkinter Reference Guide
- Regular Expressions HOWTO
- CISA Cybersecurity Guidelines
- Class notes and internal lectures

ADVANCED SECURITY ANALYSIS

Password security requires understanding entropy, brute-force resistance, and attack surfaces.
This section expands analysis beyond basic regex checks.

PASSWORD ENTROPY

Entropy measures unpredictability. Stronger passwords have higher entropy.
Entropy = log2(pool_size^length). This explains why length matters as much as complexity.

THREATS & VULNERABILITY LANDSCAPE

- Brute force attacks
- Dictionary attacks
- Social engineering
- Credential stuffing
- Rainbow table attacks

IMPROVED TEST CASE TABLE

Include boundary tests, invalid inputs, unicode characters, long passwords, and symbol-heavy cases.

DEEP IMPLEMENTATION NOTES

Discuss time complexity of validation, Tkinter event loop behavior, and modularity benefits.