



ISTANBUL KÜLTÜR UNIVERSITY



< E
X
P
L
O
R
I
N
G
Y

N
O
V
E
L

C
H
C
R
Y
P
T
O
G
R
A
P
H
I
C

M
E
T
H
O
D
S

F
O
R

S
E
C
U
R
E

K
E
Y

G
E
N
E
R
A
T
I
O
N
>

Capstone Project
EXPLORING NOVEL CRYPTOGRAPHIC METHODS
FOR SECURE KEY GENERATION

Submitted By

Hayal İldeniz İnanç 2100005804

Özge Küçükbayram 2000004664

Bilge Demir 2000003445

Project Advisor
Assis. Prof. Dr. Öznur Şengel

Department of Computer Engineering
İstanbul Kültür University

2024-2025 Spring



ISTANBUL KÜLTÜR UNIVERSITY

Capstone Project
EXPLORING NOVEL CRYPTOGRAPHIC METHODS
FOR SECURE KEY GENERATION

Submitted By

Hayal İldeniz İnanç 2100005804

Özge Küçükbayram 2000004664

Bilge Demir 2000003445

Advisor

Assis. Prof. Dr. Öznur Şengel

THE EXAMINATION COMMITTEE

Jury Member

1.
2.
3.

Signature

.....
.....
.....

ABSTRACT

In this project, a semantically emoji-assisted password generation system based on a natural language processing (NLP)-enabled system has been developed to help users generate secure but simple-to-remember passwords. The system performs grammatical analysis on the input provided by the user to and identifies the subject, verb, and object and forms a password based on semantically meaningful emojis for these components.. In addition, passwords that do not include emojis can also be generated using only the classical character set (letters, uppercase letters, numbers, symbols). Password length and content types can be specified by the user. When the total number of selected character types is less than the desired password length, the system automatically completes the remaining characters either with emojis or only with classical characters, depending on the emoji selection. In such cases, both emoji-supported and non-emoji alternative password suggestions are presented. The security of the generated passwords is evaluated using Monobit, Runs, Cumulative Sums, and Serial Tests; in addition, an estimated cracking time against brute-force attacks is calculated to provide feedback to the user. The system is accessible through a web-based user interface and offers flexible password generation according to different security requirements.

ÖZET

Bu projede, kullanıcıların güvenli ancak hatırlaması kolay şifreler oluşturmalarına yardımcı olmak amacıyla doğal dil işleme (NLP) destekli, anlamsal olarak emoji yardımlı bir şifre üretim sistemi geliştirilmiştir. Sistem, kullanıcının verdiği girdi üzerinde dilbilgisel analiz gerçekleştirerek özne, fiil ve nesneyi belirler ve bu bileşenler için anlamsal olarak uygun emojilerle bir şifre oluşturur. Ayrıca, yalnızca klasik karakter seti (harfler, büyük harfler, sayılar, semboller) kullanılarak emojişiz şifreler de üretilebilir. Şifre uzunluğu ve içerik türleri kullanıcı tarafından belirtilebilir. Seçilen karakter türlerinin toplam sayısı istenen şifre uzunluğundan az olduğunda, sistem kalan karakterleri emoji seçimine bağılı olarak ya emojilerle ya da sadece klasik karakterlerle otomatik olarak tamamlar. Bu tür durumlarda, hem emoji destekli hem de emojişiz alternatif şifre önerileri sunulur. Oluşturulan şifrelerin güvenliği Monobit, Runs, Cumulative Sums ve Serial Testleri kullanılarak değerlendirilir; ayrıca kaba kuvvet saldırılarına karşı tahmini kırılma süresi hesaplanarak kullanıcıya geri bildirim sağlanır. Sistem, web tabanlı bir kullanıcı arayüzü aracılığıyla erişilebilir olup, farklı güvenlik gereksinimlerine göre esnek şifre üretimi sunar.

ACKNOWLEDGEMENTS

We would like to express our sincere gratitude to Assis. Prof. Dr. Öznur ŞENGEL for her guidance and valuable insights throughout the preparation of this project. She played a significant role in shaping this study with her interest, patience, and academic support at every stage of the project process.

We also thank the Department of Computer Engineering at Istanbul Kültür University for the infrastructure, resources, and academic support provided throughout the project.

We extend our thanks to our team members for their cooperation, dedication, and contributions through idea exchange during the project development process. The project was successfully completed thanks to the harmonious combination of the team members' diverse knowledge and skills.

TABLE OF CONTENTS

<i>ABSTRACT</i>	<i>I</i>
<i>ÖZET</i>	<i>II</i>
<i>ACKNOWLEDGEMENTS</i>	<i>III</i>
<i>TABLE OF CONTENTS</i>	<i>IV</i>
<i>LIST OF TABLES</i>	<i>VI</i>
<i>LIST OF FIGURES</i>	<i>VII</i>
<i>SYMBOLS & ABBREVIATIONS</i>	<i>VIII</i>
<i>1. INTRODUCTION</i>	<i>1</i>
1.1. Problem Statement	1
1.2. Project Purpose	1
1.3. Project Scope	1
1.4. Objectives and Success Criteria of the Project	1
1.5. Report Outline	1
<i>2. RELATED WORK</i>	<i>2</i>
2.1. Existing Systems	2
2.2. Overall Problems of Existing Systems	2
2.3. Comparison Between Existing and Proposed Method	2
<i>3. METHODOLOGY</i>	<i>3</i>
3.1. Requirement Analysis	3
3.2. Design	3
3.3. Implementation	3
3.4. Testing	3
<i>4. EXPERIMENTAL RESULTS</i>	<i>4</i>
<i>5. DISCUSSION</i>	<i>5</i>
<i>6. CONCLUSIONS</i>	<i>6</i>
<i>REFERENCES</i>	<i>7</i>
<i>APPENDIX</i>	<i>8</i>

LIST OF TABLES

Table 2.1: Comparison of methods.....	12
Table 4.1: Comparison of Brute-Force Test Results.....	47
Table 4.2: Comparison of Test Case Scenarios and Success Evaluation	51
Table 4.3: Comparison of Test Case Scenarios and Success Evaluation Continued.....	52
Table 4.4: Comparison of Randomness Test Results.....	54

LIST OF FIGURES

Figure 3.1: Use Case Diagram	20
Figure 3.2: Activity Diagram	22
Figure 3.3: Class Diagram.....	25
Figure 3.4: Deployment Diagram	26
Figure 3.5: NLP-based grammatical role extraction flow using spaCy.....	29
Figure 3.6: Semantic Matching with Emojis.....	31
Figure 3.7: Dynamic Brute Force Simulation.....	35
Figure 3.8: Recommended Password Feature.....	37
Figure 3.9: Frontend Features.....	38
Figure 3.10: Frontend Alert Feature.....	39
Figure 3.11: Test Features	43
Figure 4.1: Comparison of password length and similarity score for finding the password.....	49
Figure 4.2: Comparison of the effect of using emoji and without emoji on similarity rate.....	49

SYMBOLS & ABBREVIATIONS

API: Application Programming

CPU: Central Processing Unit

CSS: Cascading Style Sheets

DOM: Document Object Model

GUI: Graphical User Interface

HTML: HyperText Markup Language

IEEE: Institute of Electrical and Electronics Engineers

JS: JavaScript

JSON: JavaScript Object Notation

NIST: National Institute of Standards and Technology

NIST SP 800-22: Statistical test suite for randomness (by NIST)

NLP: Natural Language Processing

NLTK: Natural Language Toolkit (Python library for text processing)

SHA-256: Secure Hash Algorithm 256-bit

SPA: Single Page Application

SSRN: Social Science Research Network

UI: User Interface

UML: Unified Modeling Language

Unicode: Universal character encoding standard

Xor: Exclusive OR

1. INTRODUCTION

1.1. Problem Statement

As the world continues to see its internet usage spread and its web-based applications become increasingly sophisticated, it has become increasingly daunting to secure digital identities. Ordinary activities such as banking, commerce, health care services, e-learning platforms, and personal communication are now being conducted via web-based platforms. As a side effect of this growing reliance on the digital world, there has also been a severe spike in cyber threats, most significantly the ones based on weak and poorly managed passwords.

Despite the importance of password protection, most users are still relying on insecure practices. Users construct passwords using easily accessible personal information such as their name, date of birth, or telephone number. These are simple to predict and thereby make it easier for attackers to employ social engineering or even basic brute-force attempts to attack credentials. Moreover, it is common practice that the same password is applied on numerous different websites, and this threat of large-scale data breaches gets out of hand when a single account is compromised.

Another critical issue is the challenge for humans to create strong, yet easy-to-remember passwords. Strong passwords with complex characters typically including upper and lower case letters, numbers, and symbols are hard to memorize. As a result, users will write them down, store them in an insecure way, or use weaker alternatives. These behaviors significantly reduce system security overall.

Due to such kinds of challenges, there has been an awareness for novel methods of password creation. A good solution would not only generate strong and unpredictable passwords but also ensure memorability and user engagement. This creates an opportunity for interdisciplinary solutions that integrate techniques from natural language processing (NLP), usability engineering, and cybersecurity. A system that can semantically interpret user input and generate secure, personalized, and easy to remember

passwords may help bridge the gap between usability and security, reducing user errors and strengthening digital protection.

1.2. Project Purpose

The main purpose of this project is to develop a password generation system that addresses the limitations of traditional password practices by producing strong, memorable, and user-friendly passwords through the use of semantic emoji matching and natural language processing (NLP) techniques. The system is designed to process user-provided text, identify key linguistic elements like the subject, verb, and object, and generate unique and meaningful password combinations by semantically associating these elements with appropriate emojis.

By leveraging NLP, the system aims to transform plain user input into passwords that reflect both contextual understanding and visual memorability. Unlike conventional generators, this model provides variability even when identical input is entered multiple times, ensuring that each password remains unpredictable and secure. This approach not only increases password entropy but also supports user recall through semantically relevant and visually distinct content.

In addition to supporting emoji-based elements, the system incorporates traditional security components such as uppercase and lowercase letters, numbers, and symbols, ensuring compliance with current password complexity standards. It is also built with a strong emphasis on accessibility and user-friendliness, allowing individuals with varying levels of technical expertise to generate secure passwords effortlessly. Ultimately, the project seeks to close the gap between security and usability by delivering a customizable, intelligent, and user-centered solution to modern password generation challenges.

1.3. Project Scope

This project covers the design and implementation of a web-based password generation system that converts user-entered text into secure and personalized passwords through meaningful emoji matching. The scope aims to create an end-to-end processing system that includes both back-end processing logic and front-end user interface.

The core of the proposed system is a Natural Language Processing (NLP) algorithm built on the spaCy library that performs syntactic analysis on user-provided input to determine basic grammatical components such as subject, verb, and object. To ensure semantic consistency in the generated passwords, a comprehensive emoji dataset structured in JSON (JavaScript Object Notation) format is used to match meaningful emojis with words in the entered text. A weighted selection algorithm was developed to increase both semantic consistency and password uniqueness. With this algorithm, emoji assignments were prioritized in the order of object, verb, and subject.

The back-end of the system was implemented using the Python programming language and the Flask web framework. This architecture facilitates real-time password generation and evaluation processes through an API-based design, and enables efficient interaction between front-end and back-end components. The front-end is designed as a user-friendly browser interface that allows users to enter free text, instantly view emoji-based password suggestions, and visually experience the output. The system also supports classic character types such as uppercase letters, lowercase letters, numbers, and symbols, allowing the creation of hybrid passwords that comply with modern password complexity standards.

Our system also includes a comprehensive evaluation process for password security. This process involves subjecting the generated passwords to a series of statistical randomness tests, including Monobit Test, Run Test, Cumulative Sums Test, and Serial Test. Additionally, brute-force attack simulations are run to estimate the time required for possible password cracking attempts and to evaluate the overall resilience of the system. These integrated components enable the system to deliver not only functional outputs but also measurable security performance measurements.

The current version of the project only works with English language input and does not include database integration or user authentication modules. The primary focus of the project is real-time emoji password generation based on natural language input and evaluation of their randomness and resistance to brute force attacks. All these processes are integrated into a web-based interface that allows users to access and use the system effectively and interactively.

1.4. Objectives and Success Criteria of the Project

The primary objective of this project is to develop a secure, semantically-aware password generation system that enhances both usability and security through the use of natural language processing and emoji-based representations. To achieve this, the system must accurately process and analyze user input text by applying linguistic techniques such as part-of-speech tagging and lemmatization. This grammatical parsing step is essential to identify the subject, verb, and object components of each sentence, which are then used to semantically match relevant emojis from a curated emoji dataset. The success of this mechanism is measured by the system's ability to retrieve and assign at least one appropriate emoji to each grammatical role. Furthermore, the project aims to implement a weighted distribution logic that prioritizes emojis representing objects as the most important, followed by verbs and finally subjects, in terms of their contribution to password construction. This ordering is based on the assumption that objects tend to have more visually distinctive and meaningful emoji representations compared to verbs and subjects.

Another key objective is to ensure password variability and unpredictability even when the same sentence is entered multiple times. This is achieved by incorporating randomization in emoji selection and password construction, ensuring that each generated password is unique while still semantically connected to the original input. The system must also seamlessly support additional character types such as numbers, uppercase letters, lowercase letters, and symbols to comply with modern security standards and to enhance brute-force resistance.

Beyond functional correctness, the system must demonstrate high performance and responsiveness through an intuitive web-based interface. Users should be able to input their text, define password constraints, and instantly receive a secure password along with visual feedback regarding its strength and randomness. The final indicator of success is the system's ability to integrate advanced randomness tests and brute-force simulations, offering transparency into password quality and attack resilience. Meeting all of these objectives will confirm that the project has not only achieved its functional goals but has also provided a novel, practical, and secure solution to the problem of password generation.

1.5. Report Outline

This thesis is structured into six chapters, each addressing a key aspect of the project to ensure clarity, consistency, and academic rigor. It begins with the Introduction chapter, which presents the background of the study, clearly states the problem, and outlines the objectives, motivation, and scope—emphasizing the importance of secure password generation in the context of increasing cyber threats.

The Literature Review chapter explores previous academic and technological research related to password generation, Natural Language Processing (NLP), and emoji-based semantic models. It identifies the limitations of existing systems and establishes the theoretical framework for the proposed approach.

The Methodology and System Architecture chapter provides a detailed explanation of the system's design, including the tools, technologies, and development decisions. Particular emphasis is placed on the use of Python, Flask, and spaCy, with clear descriptions of processes such as grammatical role detection, semantic similarity analysis, and emoji weighting.

In the Implementation and Development chapter, the focus shifts from design to practical realization. This section covers the backend logic, the creation of the emoji dataset, NLP

integration, and the development of the responsive web interface. It also addresses the challenges encountered during implementation and the solutions applied.

The Experimental Results chapter presents the testing outcomes of the system. It includes examples of generated passwords, security assessments, and feedback collected through usability testing. This chapter evaluates the system's performance based on criteria such as diversity, memorability, and robustness.

Finally, the Conclusion and Discussion chapter summarizes the project's main contributions, acknowledges its limitations, and proposes directions for future enhancements, including user personalization, multilingual support, and adaptive learning models to further improve password strength and user experience.

2. RELATED WORK

2.1. Existing Systems

2.1.1 Password Generation Techniques and User Behavior

In the study “Evaluating Passwords: User Behavior and the Psychology Behind Password Choice,” Allothman highlights that users often prioritize memorability over entropy, leading to the reuse of weak, predictable passwords. Their analysis, supported by behavioral metrics, shows that users frequently incorporate personal or common words into passwords despite being aware of security risks. This behavioral tendency inspired our system to retain user familiarity but enhanced security by transforming textual input into structurally varied semantically enriched passwords. Our project generates passwords that are still meaningful to users while introducing randomized layers such as character substitutions, casing variations, and emoji encoding. This directly addresses the cognitive trade-off between memorability and unpredictability [1].

In addition, the methodology described in “Strong Password Generation Based on User Inputs” parallels our own system's logic by taking user preferences as inputs but applying transformations to improve password robustness. While the cited work applies

character-level transformations like Leetspeak and randomized insertions, our approach goes beyond by incorporating a weighted grammatical structure and emoji-based semantic matching, thereby extending the personalization to visual-symbolic language. Lastly, the article “Passwords and User Behaviors” supports the idea that users respond more positively to systems that reflect their linguistic or cognitive patterns. Our system operationalizes this by maintaining user familiarity through grammatical parsing while embedding security-enhancing randomness in the form of shuffled components and emoji substitutions. In doing so, we combine technical password-generation algorithms with insights from psychological user behavior studies to enhance both security and usability [2] [3].

2.1.2 NLP and Emoji Matching

The integration of Natural Language Processing (NLP) with emoji selection has been explored primarily in areas such as sentiment analysis and digital communication; however, its application in the domain of secure password generation remains significantly underutilized. The proposed project presents an NLP-based system that syntactically parses user input to extract grammatical components—namely, subject, verb, and object—and assigns semantically appropriate emojis to each. This approach aims to enhance both the entropy and memorability of the generated passwords.

A similar approach is presented in EmojiAuth by Golla and Dürmuth (2017), where users create emoji-based passwords by selecting symbols from a static grid. Although this method improves usability, the study revealed that “users often favored emojis with personal or emotional significance, leading to non-uniform distributions and increased vulnerability” (p. 4). This shows that allowing users full control over emoji selection can introduce predictable patterns. In contrast, our system addresses this issue by automatically assigning emojis based on grammatical roles (subject, verb, object) extracted from user input, reducing selection bias and avoiding commonly chosen emoji combinations [4].

Complementing this technical perspective, the SSRN study (2020) emphasizes that emojis represent “an evolution of older visual language systems” and serve as “non-verbal surrogates” that convey meaning within specific contextual frameworks. This supports our approach of treating emojis as structured semantic units rather than arbitrary symbols. In contrast to previous work, our system uniquely combines grammatical parsing, semantic matching, and entropy-aware emoji prioritization. While EmojiAuth focuses on authentication and SSRN offers a socio-linguistic view, our project applies emoji logic to password generation, offering a secure and semantically coherent alternative [5].

2.1.3 Randomness Testing and Cryptographic Quality

In the study conducted by Rukhin et al. (2010), the NIST SP 800-22 Rev1a framework is presented as a comprehensive suite of statistical tests aimed at evaluating the randomness quality of binary sequences employed in cryptographic systems. Specifically, tests such as Monobit, Runs, Additive Sums, and Series are utilized to detect potential biases and structural irregularities within pseudo-random outputs.. In our project, a direct methodological parallel is established: these four tests are reimplemented in Python and applied to SHA-256 hash representations of generated passwords, including those containing emoji characters. By hashing each output before evaluation, our system preserves encoding uniformity; this is especially important when dealing with multibyte Unicode emojis. Similar to the NIST test suite, the objective of this study is not only to generate outputs with high entropy but also to rigorously assess their statistical properties in accordance with established measures of randomness [6].

Complementing this perspective, Bishoi and Maharana (2017) highlight the significance of mathematically grounded random number generators, with particular emphasis on Xorshift algorithms, which offer lightweight yet high-quality randomness well-suited for resource-constrained systems. We take a comparable approach by ensuring that post-generation operations such as component arrangement in our system follow strict

but entropy-preserving algorithms. This is evident in our use of the Fisher-Yates shuffling algorithm, which also plays a central role in the study by Kirana et al. (2021), where it was used to randomize exam questions in educational settings. Sharing the same concern about pattern predictability, our system applies Fisher-Yates to shuffle password components—including letters, symbols, and emojis—to reduce sequential bias. In both systems, the goal is the same: to preserve randomness and output integrity through controlled shuffling, even when inputs are partially structured or semantically generated [7][8].

2.1.4 Brute Force Attacks and Resistance Estimation

The Markov chain–based password generation model developed by Vaithyasubramanian et al. (2014) ensures structural randomness against brute-force attacks by determining character transitions through probabilistic rules. Similarly, the system proposed in this study applies natural language processing (NLP) techniques to parse user input into grammatical roles—subject, verb, and object—and assigns relevant emojis to introduce controlled variability. However, unlike the purely mathematical approach of the Markov model, our system enhances both memorability and entropy by incorporating semantic context. Additionally, the use of emojis, which draw from the extensive Unicode set, significantly increases brute-force resistance. With its user-centered design and semantic security layer, the proposed system offers a more robust and adaptable solution compared to the Markov-based approach. [9].

Kjellevand and Rauhut (2018) assessed brute-force resistance in the EmojiStory system by analyzing entropy and the combinatorial space of emoji sequences. Their study demonstrates that increasing the length of emoji passwords significantly enhances resistance against brute-force attacks. Similarly, our system estimates brute-force resistance using Unicode-aware character length and symbol diversity and further simulates attack scenarios where the attacker is unaware of emoji presence. This modeling assumes that a brute-force attacker uses a reduced character set, which greatly underestimates the actual key space when emojis are included. Both systems reflect the

importance of attacker assumptions in calculating resistance, though our system extends the idea by incorporating dynamic simulation and SHA-256 hashing prior to resistance estimation, providing a more adaptable and realistic security evaluation [10].

2.1.5 UI Design and Application Integration in Emoji-Based Security Systems

Although the SEntiMojl study by Chen et al. (2019) does not include a traditional user-facing interface, its architecture implicitly demonstrates a modular design and outlines the data flow between NLP components and emoji embeddings. The system primarily operates as a backend pipeline focused on semantic emoji classification and embedding, without incorporating a graphical user interface. In contrast, the system developed in this project features an explicitly designed, web-based user interface that not only displays emoji outputs but also visualizes them interactively alongside password strength metrics. Key enhancements include dynamic feedback mechanisms, real-time test results, and input-based emoji previews—functionalities that are not present in the SEntiMojl prototype. Thus, while both systems process emoji-text relationships, only our implementation translates these associations into an engaging and security-aware UI experience aimed at end users[11].

In contrast, the BITREST password management system (2021, Strathmore University) emphasizes user interface design and local storage of emoji-based credentials. It features a graphical interface developed in C#, enabling users to manually select and assign emojis to specific services. While this approach aligns with our project in treating emojis as core password components, the underlying methodology differs significantly. BITREST depends on manual selection from a fixed emoji set, whereas our system leverages natural language processing to automatically generate semantically relevant emojis based on user input.

Furthermore, BITREST lacks advanced functionalities such as real-time feedback, password strength indicators, and statistical randomness evaluation. In comparison, our system enhances usability and security by offering interactive feedback mechanisms, entropy-based strength metrics, and visual indicators. Despite these methodological

differences, both systems share the view that emoji integration can improve memorability and user engagement, highlighting the value of user-centered design in password systems [12].

2.2. Overall Problems of Existing Systems

Current password security systems often struggle to find a good balance between being easy to use and producing unpredictable passwords. Many depend on users to choose their own password parts, which limits how varied the passwords can be and makes them more predictable, lowering their overall strength. Also, most of these systems don't use standardized statistical tests to check how strong a password really is while it's being created, so users don't get clear feedback about the security of their passwords. Systems that include emojis usually require users to pick them manually, without considering the meaning or context, which makes it easier for attackers to guess common emoji choices. On top of that, very few of these systems have brute-force attack simulations to help users understand how long their passwords might last under attack. Our project tries to solve these issues by automatically assigning emojis using natural language processing, testing randomness at the bit level, running real-time brute-force resistance simulations, and offering an interactive frontend that gives visual feedback and encourages safer password habits. Many existing solutions also lack simple and clear user interfaces that help people understand the impact of their password choices. Overall, these gaps show the importance of creating systems that combine smart automatic processing, strong testing methods, and user-friendly design.

2.3. Comparison Between Existing and Proposed Method

To systematically evaluate the strengths and weaknesses of different password generation approaches, Table 2.1 compares four existing methods (Methods A–D) with the proposed system. The comparison is based on seven core criteria: user familiarity, entropy, emoji integration, presence of security evaluation, user interface support, resistance to brute-force attacks, and system adaptability. The results show that traditional methods have limitations as they often use fixed patterns and do not provide feedback when

generating passwords. On the other hand, our project uses natural language processing (NLP), matches emojis according to meaning, and applies statistical tests to ensure that passwords are strong. In this way, it offers better security while being easy for users to use.

Table 2.1: Comparison of methods

Feature	Method A[1]	Method B[2]	Method C[4]	Method D[11]	Our Method
User Familiarity	✓ Retains personal patterns	✓ Based on user input	✓ Emotional emoji selection	✗ Backend only, no UI	✓ Semantic + visual match
Entropy / Randomness	✗ Low entropy (predictable patterns)	✓ Leetspeak + random inserts	✗ Skewed due to bias	✓ High randomness in embedding	✓ Emoji + char mix + shuffle + SHA-256
Emoji Integration	✗ None	✗ None	✓ Static emoji grid	✓ Semantic vector model	✓ Dynamic NLP-based semantic emoji selection
Security Evaluation	✗ No formal tests	✗ No randomness tests	✗ No brute-force metrics	✗ No security layer	✓ NIST-style randomness tests + brute-force sim
Usability & Feedback	✗ No UI interaction	✗ Command-line/functional	✗ Manual emoji choice	✗ Backend only	✓ Web-based UI, dynamic feedback, interactivity
Brute-force Resistance	✗ Easily cracked	✓ Moderate	✗ Emoji patterns guessable	✗ Not tested	✓ Unicode-aware entropy + attacker-unaware emoji
Adaptability & Flexibility	✗ Fixed structure	✗ Rigid character rules	✗ Predefined emoji list	✗ Static design	✓ Dual mode, fallback logic, scalable architecture

3. METHODOLOGY

This project was executed using a structured, software-oriented methodology that integrates Natural Language Processing (NLP), emoji-based semantic matching, and randomness evaluation techniques. The development process adopted a modular

framework, beginning with a comprehensive requirements analysis, and progressing through systematic stages of design, implementation, and testing. To ensure transparency and reproducibility, each phase of the project was thoroughly documented, including detailed algorithmic steps and defined use cases. The backend was developed using Python and the Flask framework, while the frontend was implemented with HTML, CSS, and JavaScript technologies. The development emphasized creating an intuitive user interface, ensuring cryptographic strength, and building a system architecture that supports future scalability and enhancement.

3.1. Requirement Analysis

The Emoji Password Generator System developed in this project is an application that generates passwords according to the criteria specified by the users. When the user selects only the emoji character, the system expects them to enter a word or sentence; in other cases, it is sufficient to select only the character types (letters, numbers, symbols) and the password length. The system generates passwords in accordance with the selected criteria, and if an emoji is selected, it enriches the password by establishing a relationship with meaningful emojis. In addition, the user can copy the password, recreate it, and view security analysis.

3.1.1. Textual Requirements / Use Case Diagram

Emoji Password Generator System Requirements:

1. The user must be able to enter a text (word or sentence)
 - The system processes the text for password generation according to this input.
2. The user must be able to determine the password length.
3. The user must be able to select the password character types.
 - Letters
 - Capital letters
 - Numbers
 - Symbols

- Emojis
4. The system must generate a password that meets the selected criteria.
 5. The system must establish a relationship between the text and emojis.
 - (Indicated as "Emoji relationship" in the Use Case diagram.)
 6. The user must be able to see the password generated.
 7. The user must be able to copy the password to the clipboard.
 - This operation is provided by the system.
 - Extends the "Generate password" operation (<<extend>>).
 8. The user must be able to re-generate the password.

3.1.2. Use Case Scenarios

Use Case 1: Enter Text (Sentence or Word)

Actor: User

Precondition: The application must be running.

Main Flow:

1. The user enters a word or sentence.
2. The system receives and processes the input.

Postcondition: The text is stored for use in password generation.

Use Case 2: Set Password Length

Actor: User

Precondition: The user must have entered text.

Main Flow:

1. The user selects the desired password length.

Postcondition: The chosen length is saved by the system.

Use Case 3: Select Password Character Type

Actor: User

Precondition: The user must have entered the input text and selected the desired password length.

Main Flow: The user selects one or more-character types:

- lowercase letters
- uppercase letters
- numbers
- symbols
- emojis

Postcondition: The selected character types are stored by the system.

Use Case 4: Generate Password

Actor: System

Precondition: The user must have provided all required input.

Main Flow:

1. The system generates a password based on user inputs.
2. The "Emoji Relationship" use case is included. (<<include>>)
3. The generated password is displayed to the user.

Extensions:

- Optionally, the user can choose to copy the password to the clipboard.
(<<extend>> Copy to Clipboard)

Postcondition: A password is successfully generated and presented to the user.

Use Case 5: Emoji Relationship

Actor: System

Included By: Generate Password

Precondition: The user must have entered input text and selected emoji as one of the character types.

Main Flow:

1. The system analyzes the input text for semantic meaning.
2. Relevant emojis are selected and incorporated into the password.

Postcondition: A semantically meaningful password containing emojis is created.

Use Case 6: Copy to Clipboard

Actor: User, System

Extended From: Generate Password

Precondition: A password must have been successfully generated and displayed to the user.

Main Flow:

1. The user selects the option to copy the password.
2. The system copies the generated password to the clipboard.

Postcondition: The password is stored in the user's clipboard.

Use Case 7: Show/Hide Text Field Based on Emoji Selection

Actor: System

Precondition:

- The application is running, and the UI is loaded.
- The emoji character type checkbox is initially **unselected**.

Main Flow:

1. The user selects the "Emojis" checkbox.
2. The system detects the change in checkbox state.
3. The system displays the text input field if it is hidden.

Postcondition: The text input field becomes visible when emoji selection is active.

Use Case 8: Validate Character Type Selection Before Password Generation

Actor: System

Precondition:

- The application is running.
- The user has entered a password length.
- No character types (letters, numbers, symbols, emojis) are selected.

Main Flow:

1. The user clicks the "Generate Password" button.
2. The system checks if at least one character type is selected.
3. If none are selected, an error or warning is displayed.

Postcondition: The system prevents password generation and informs the user to select at least one character type.

Use Case 9: Handle Misspelled Words for Emoji Matching

Actor: System

Precondition:

- Emoji character type is selected.
- User has entered an input word that may be misspelled (e.g., “winer” instead of “winter”).

Main Flow:

1. The system attempts to semantically interpret the entered word.

2. Based on partial matches or similarity, the most relevant emoji is identified.
3. The selected emoji is included in the generated password.

Postcondition: A password containing a relevant emoji is generated even if the input word has minor spelling errors.

Use Case 10: Perform Brute Force Simulation Test

Actor: User

Precondition:

- A password must have been successfully generated and is visible on the screen.

Main Flow:

1. The user clicks the "Start Brute Force Test" button.
2. The system begins simulating brute-force attacks.
3. Time to crack and the complexity metrics are computed.

Postcondition: Brute-force test results are shown to the user.

Use Case 11: View Password Randomness Analysis

Actor: User

Precondition:

- A password must have been generated.

Main Flow:

1. The user requests to view randomness results.
2. The system analyzes the password for entropy and pattern variety.
3. The analysis is displayed in a readable format.

Postcondition: The user receives feedback on the randomness and strength of the generated password.

3.2. Design

3.2.1. Use Case Diagram

Figure 3.1 presents the Use Case diagram, which describes the functions of the password generation and evaluation system, and describes the roles of two important actors, User and Administrator. The user can enter a basic text, select the desired password length, and select character types such as lowercase, uppercase, number, symbol, and emoji. When the user provides these inputs, the Generate Password function is triggered, which causes the system to generate a random password. Once the password is generated, it automatically passes through various statistical tests, including Monobit Test, Serial Test, Cumulative Sum Test, and Run Test. If the password does not meet the required standards according to the tests, the system enables the Suggest Stronger Password feature without direct user input. The user can then view the suggested stronger password through this option. In addition, the User has the ability to copy the generated password using the Show Brute Force feature, which displays information such as cracking time, number of attempts, and the current status of the transaction. Meanwhile, the Administrator manages the system settings, including configuring brute force limits, reviewing user statistics, setting password policies, and controlling randomness testing algorithms. This design ensures that the system not only generates secure passwords, but also provides a user-friendly experience and provides the necessary administrative controls.

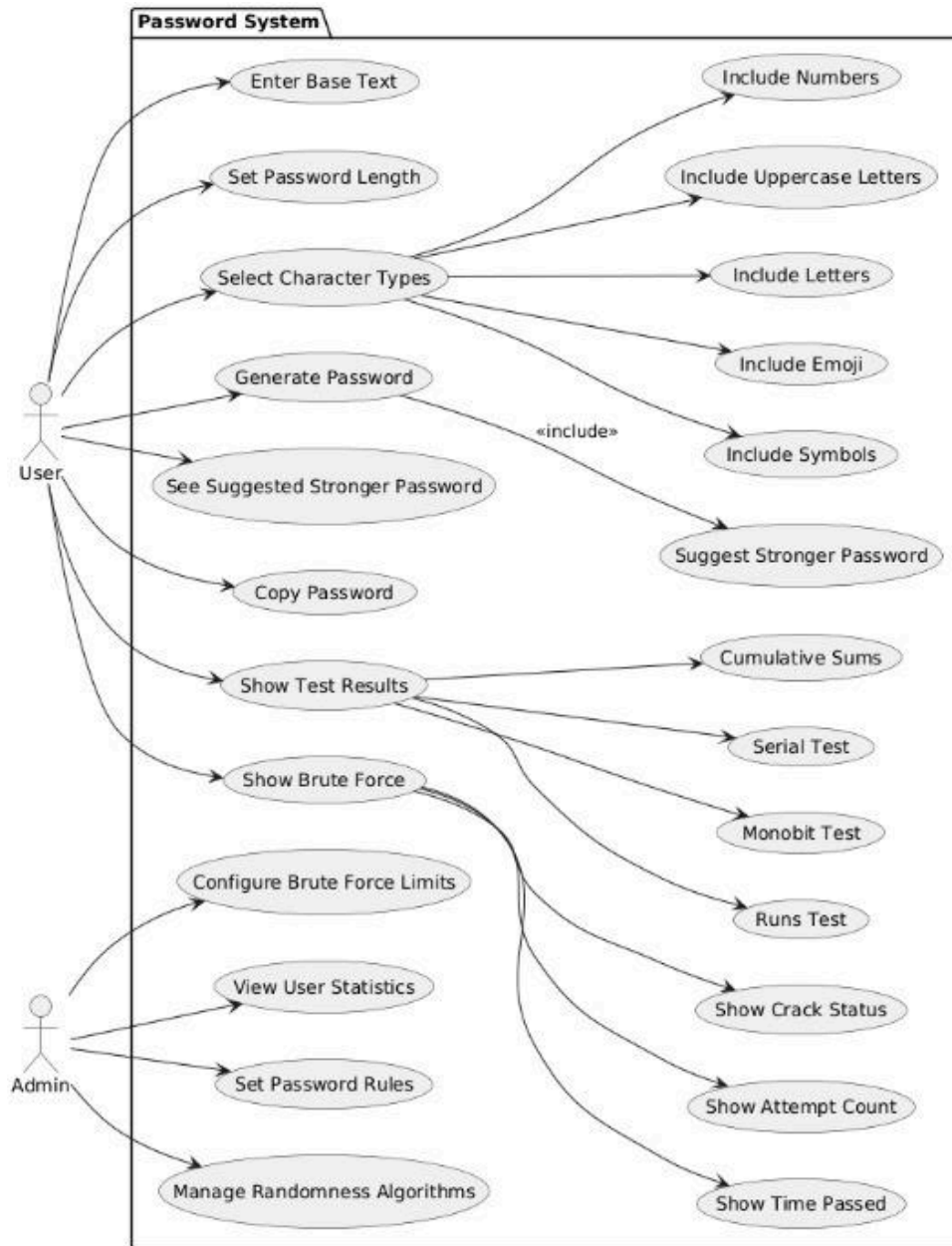


Figure 3.1: Use Case Diagram

3.2.2 Activity Diagram

Figure 3.2 shows the step-by-step flow of the password generation system, focusing on how decisions are made based on user choices. Initially, the user decides whether to include emojis in their password. If emojis are selected, the user can create a password consisting of only emojis or can mix emojis with other character types such as lowercase, uppercase, numbers, and symbols. If emojis are not selected, the system continues with a standard character set. After the user selects the character types and the desired password length, the system generates the password and presents it to the user. Following this, the user can optionally run statistical randomness tests (including Monobit, Series, Cumulative Sums, and Runs tests) or perform a brute force attack simulation to evaluate the password's security level. If the password length exceeds the selected number of characters, the system suggests a stronger password to improve both security and user experience. This diagram illustrates the adaptive and user-friendly process involved in password generation and evaluation.

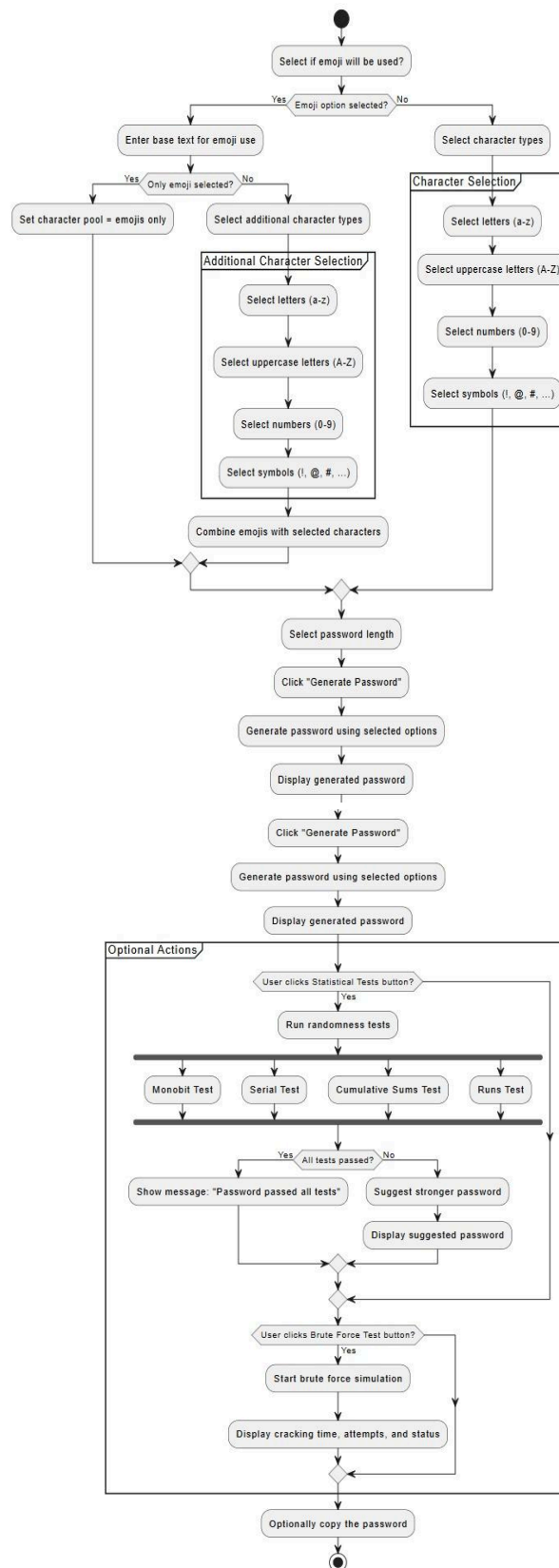


Figure 3.2: Activity Diagram

3.2.3. Class Diagram

The UML class diagram shown in Figure 3.3 depicts the architecture of a Flask-based web application designed to generate emojis from user-provided text input. It also performs randomness evaluations on the generated passwords and returns a response that includes both the selected emojis and the results of the statistical tests. This diagram highlights the key classes, their attributes, and methods, illustrating how the system's components interact to support password creation and analysis.

1. FlaskApp::App

This is the main application controller built with Flask. It defines the route `/get-emojis`, which handles POST requests containing input text and emoji count. It coordinates the flow between the input, processing modules, and the final output response.

- Responsibilities:
 - Accept client requests.
 - Delegate processing to service classes.
 - Return structured JSON responses.

2. EmojiService

This service layer handles core processing tasks, including:

- Extracting or matching emojis based on input text.
- Parsing and tokenizing input.
- Generating a bitstring representation of the emojis.
- Hashing the resulting string using SHA-256.
- Key Methods:
 - `get_random_emojis(text, count)`
 - `get_emojis_from_word(word, limit, used_emojis)`
 - `parse_sentence(text)`
 - `text_to_bitstring(text)`
 - `hash_password(password)`

3. RandomnessTests

This module implements several statistical randomness tests inspired by NIST SP800-22 standards:

- Monobit Test: Analyzes the balance of 0s and 1s.
- Runs Test: Examines patterns and their frequency.
- Cumulative Sums Test: Checks for statistical deviation over time.
- Serial Test: Looks for repeating binary patterns.

These tests are applied to the emoji-derived bitstring to assess its entropy and distribution.

4. JSONEmojiLoader

This component is responsible for loading and maintaining the emoji dataset. It reads a JSON file and populates two main data structures:

- emoji_dict: A dictionary mapping words to emojis.
- all_words: A list of all available keys for fuzzy matching.

5. Request and Response

The Request class models the incoming JSON body with text and count. The Response class formats the returned data with selected emojis, the SHA-256 hash, and the output of all randomness tests.

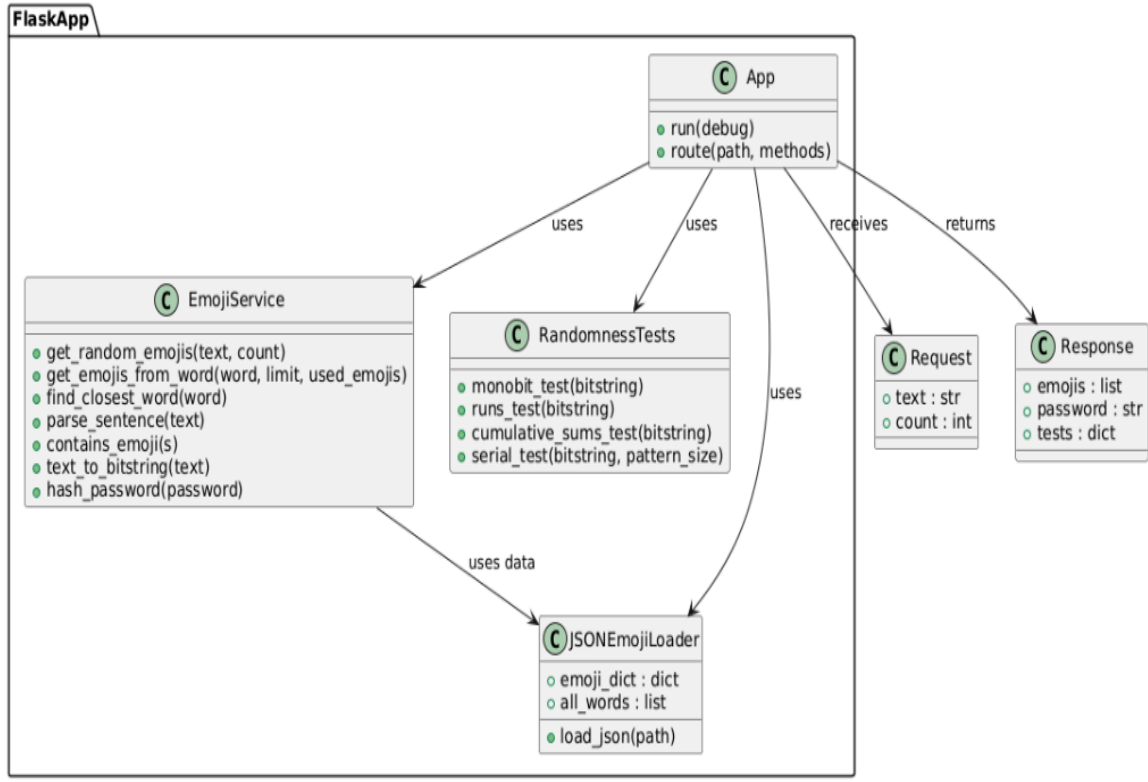


Figure 3.3: Class Diagram

3.2.4. Deployment Diagram

Figure 3.4 illustrates the client-side architecture, which is composed of three main files: `index.html`, `script.js`, and `style.css`. These files operate within the user's browser, where the HTML file defines the page structure, JavaScript manages communication with the backend through API calls, and the CSS file applies styling to ensure a clean and accessible user interface. Collectively, these elements create the front-end environment that allows users to interact seamlessly with the Flask-based backend system.

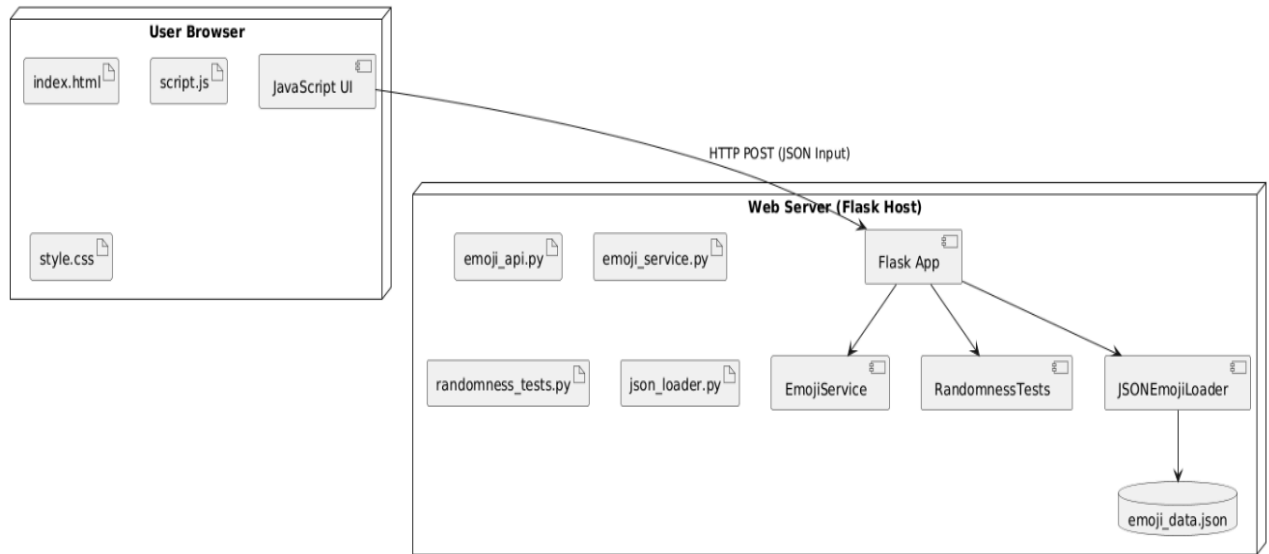


Figure 3.4: Deployment Diagram

3.3. Implementation

3.3.1. Creating JSON Dataset

The main data source of the semantic code generation system developed in the project is a .md (Markdown) file containing a comprehensive list of all emojis supported by the Windows 11 operating system. This file contains each emoji character along with their corresponding short names.

In the first stage, this markdown file was processed and the data it contained was converted to a computer-processable format, namely the JSON structure. This basic JSON output, which does not have a categorical or semantic structure, was not found useful in terms of use because it was not presented in a structure suitable for our project.

In the second stage, the content of the markdown file was examined, the title structure in the original document was used, and emojis were systematically classified according to the main and subcategories they belong to. The emoji category and subcategory structures determined by Unicode were used during this classification. The first-level headings in the file (such as Smileys & Emotion, Animals & Nature, Food & Drink) were defined as the main category; and the second-level headings (such as Face Smiling, Face Affection) were

defined as the subcategories. As a result, emojis related to each subcategory were grouped and a hierarchical JSON data structure was obtained.

The emoji dataset that was created initially contained only emoji characters and their short names. However, based on one-to-one name matching for the matches to be made with the texts received from the user would be insufficient both for future processes and in terms of usage. For this reason, the semantic scope of the names corresponding to each emoji was expanded and enriched with synonyms. The "synonyms" list included in each emoji was used, and the words in this list were added to the dictionary as a key.

In this process, the open-source WordNet dictionary database, which contains the semantic relationships of words in the English language, was used. Synonyms for each emoji name were determined and systematically added to the relevant emoji object. Thus, for example, the name "pizza" could be matched not only in direct matches, but also with related words such as "pizza slice", "food".

The added synonyms were converted to lowercase letters with the `.lower()` function for normalization to be uniform throughout the dataset, and duplicate records were removed and simplified.

In addition, all emoji names and their synonyms are collected in a central dictionary repository and optimized using the system's fuzzy matching algorithm. This ensures that the system provides correct matches even when users enter entries with spelling errors or different word choices.

3.3.2. Backend Architecture: Python + Flask

The backend of the system is developed using the Python Flask framework, which offers a simple and effective environment for building lightweight APIs. Flask is chosen specifically for its modularity, ease of use, and fast integration capabilities with data-processing libraries in Python. Since the project requires dynamic interactions—such as analyzing user input, performing linguistic processing, generating a password, simulating brute-force attacks, and executing real-time randomness tests—Flask provides a flexible foundation that can accommodate all these operations within a coherent and RESTful API structure. Moreover, Flask's simplicity reduces overhead and allows rapid prototyping, which is ideal for academic research and iterative development. All backend

functionalities—from receiving and parsing the input to password generation and result transmission—are handled through this framework.

3.3.3. NLP-Based Input Processing: spaCy

To identify the basic components such as subject, verb, and object from the text entered by the user, the system uses the spaCy library, a widely recognized tool for Natural Language Processing (NLP) in Python. The `en_core_web_sm` model was chosen due to its efficient balance between computational speed and accuracy to enable real-time processing in the application. After receiving the text input, spaCy breaks the sentence into its elements and assigns appropriate tags to each word. This allows for the identification of syntactic roles such as `nsubj` (subject), `ROOT` (main verb), and `dobj` (direct object). This process is important as it determines how emojis are semantically mapped to words. Lemmatization helps by converting words to their roots to maintain consistency during matching (e.g., changing “playing” to “play”). spaCy is a useful tool for the project because it works well with Python and has many built-in language features. Figure 3.5 shows the detailed steps of how the system extracts the subject, predicate, and object from a sentence written by the user.

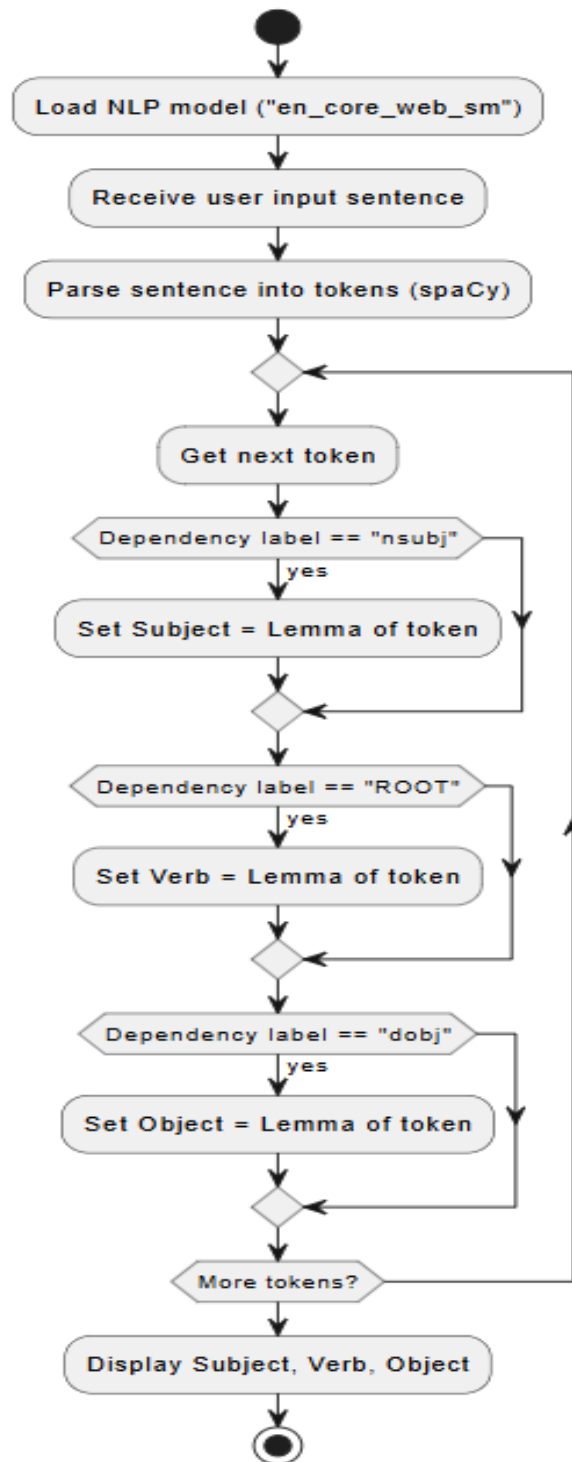


Figure 3.5: NLP-based grammatical role extraction flow using spaCy

3.3.4. Semantic Matching with Emojis

After the system finds the grammatical parts of the sentence, it matches the words with emojis in the large JSON dataset we prepared. Each emoji in this dataset contains its name and a list of synonyms. To make the matching more accurate and flexible, PorterStemmer and RapidFuzz NLP tools from the NLTK library are used. PorterStemmer separates the words into their stems. This helps to cope with different word endings or forms. RapidFuzz performs fuzzy matching by checking how similar the user's words are to the emoji names and synonyms. In this way, even if the user types something like "joyful" instead of "happy", the system can still choose a matching emoji. Combining lemmatization, stemming and fuzzy matching makes the emoji selection process more powerful and better reflects what the user means. This process is illustrated in Figure 3.6.

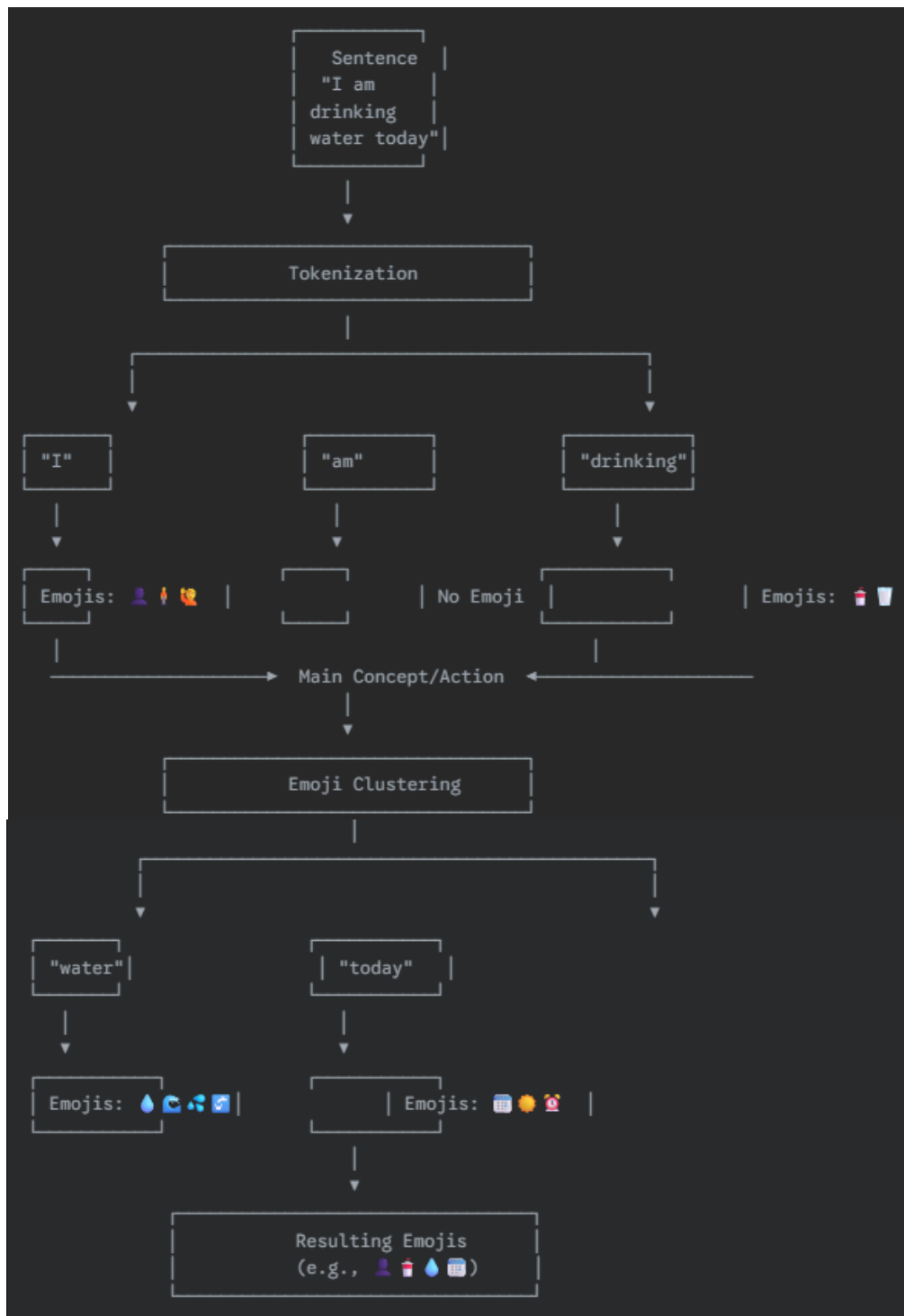


Figure 3.6: Semantic Matching with Emojis

3.3.5. Emoji Selection Strategy

The system uses a weighting selection strategy to ensure that the most semantically relevant emojis are included in the password. In this strategy, emojis representing objects are given the highest priority, followed by verbs and finally subjects. This ranking is based on the characteristics of the emoji dataset: objects such as “apple”, “car” or “book” generally have clear and recognizable emoji equivalents, while subjects such as “me” or “it” are abstract and often lack direct visual representations. Verbs occupy a middle ground, as many of them are visually supported through action-based emojis such as running or sleeping. If a direct emoji match cannot be found for a particular grammatical role, the system searches for semantically related alternatives or uses fallback categories to ensure that the desired password length is met. This approach achieves a balance between functional accuracy and visual expressiveness, resulting in passwords that are both structurally meaningful and aesthetically appealing.

3.3.6. Password Construction

Password generation is the core functionality of the system. It combines user-defined parameters with outputs derived from natural language processing and semantic emoji matching. Users can specify the total password length and choose which types of characters to include; i.e. lowercase letters, uppercase letters, digits, symbols, and emojis. Based on these selections, the system generates a password by randomly selecting elements from the selected categories and including emojis that are semantically correctly matched with the input text. In cases where the total number of certain characters (e.g., a letter, a symbol, and an emoji) is less than the desired password length, the system automatically fills the remaining positions with additional emojis. If a sufficient number of emojis cannot be obtained, substitute characters (such as letters or symbols) are used to complete the password. This ensures that each generated password strictly complies with the defined length. The password generation logic is designed with careful consideration for entropy, diversity, and uniqueness. As a result, the same inputs can produce different outputs, increasing unpredictability and minimizing the risk of repetition. A pseudocode

example showing the step-by-step generation process is provided below and shows how the system applies character type choices and emoji integration rules during password generation.

Pseudo Code Example:

GeneratePassword(length, options):

Input: length (int), options (dict with booleans for 'lowercase', 'uppercase', 'digits', 'symbols', 'emojis')

Output: password (string)

1. Initialize selectedGroups = []
2. If options["lowercase"]: add lowercase letters to selectedGroups
3. If options["uppercase"]: add uppercase letters to selectedGroups
4. If options["digits"]: add digits to selectedGroups
5. If options["symbols"]: add symbols to selectedGroups
6. If options["emojis"]:
 - a. Extract keywords or preferences using NLP
 - b. Match and select relevant emojis based on semantic context
 - c. Add selected emojis to selectedGroups
7. Initialize passwordChars = []
8. For each enabled group in selectedGroups:
 - a. Randomly pick one character from group and append to passwordChars
9. While len(passwordChars) < length:

- a. If emojis are enabled and emojis left: append more emojis
 - b. Else: randomly pick from other groups and append
10. Shuffle passwordChars to ensure randomness
 11. Return joined passwordChars as password

3.3.7. Unicode and Hashing with SHA-256

The presence of emojis in passwords poses some challenges because emojis are Unicode characters that can occupy between 2 and 4 bytes, unlike regular ASCII characters that always occupy 1 byte. This difference causes problems when measuring the bitwise randomness of passwords. To solve this problem, each password is first processed with a SHA-256 hash function before randomness tests are performed. SHA-256 is a secure cryptographic hash function that converts any input into a fixed 256-bit output. This process ensures consistency across tests and allows fair comparisons between different passwords. In addition to standardizing the input, hashing also increases security by obscuring the original password, preventing attackers from easily reversing or analyzing it. This makes hashing an important step in protecting passwords within cryptographic systems.

3.3.8. Dynamic Brute Force Simulation

The system has a brute-force simulation feature that works through a special Flask endpoint. When started, it looks at the generated password to see what kinds of characters are used. Then, it builds a custom pool of characters based on those types like letters, numbers, symbols, and sometimes emojis. This setup tries to imitate how real attackers guess passwords by focusing on likely character sets. The simulation runs random guesses for up to 30 seconds to test how quickly the password could be cracked.

If the password is cracked within the given time, the system reports the number of attempts made and the elapsed time. If not, it indicates that the password was not cracked. The time limit prevents the simulation from running indefinitely and reflects realistic constraints of offline attacks. This feature offers valuable insights into the relative strength of various password types and helps users understand how emojis and password length affect resistance to brute-force attacks, as illustrated in Figure 3.7

The screenshot shows a web application titled "Password Generator" with the subtitle "Quickly and securely generate passwords". The interface includes a text input field for "Enter Base Text" containing "i love dog", a "Password Length" dropdown set to "5", and a "Remaining: 0" indicator. Below these are five checkboxes for password components: "Letters" (unchecked), "Uppercase Letters" (unchecked), "Numbers" (checked), "Symbols" (checked), and "Emojis" (checked). To the right of the checked items are input fields with values "1", "1", and "3" respectively. At the bottom of the configuration section, the "Show Brute Force" button is highlighted with a red border. Below the configuration are two buttons: "Generate" and "Copy". At the very bottom, a result box with a red "X" icon displays the message: "Password could not be cracked within the time limit. Attempts Made: 29.576.875 Time Taken: 30 seconds".

Component	Selected	Count
Letters	<input type="checkbox"/>	
Uppercase Letters	<input type="checkbox"/>	
Numbers	<input checked="" type="checkbox"/>	1
Symbols	<input checked="" type="checkbox"/>	1
Emojis	<input checked="" type="checkbox"/>	3

Generate Copy

Show Brute Force Show Test Results

Generated Password: i love dog 6< 🐶

❌ Password could not be cracked within the time limit.
⌚ Attempts Made: 29.576.875
⌚ Time Taken: 30 seconds

Figure 3.7: Dynamic Brute Force Simulation

3.3.9. Recommended Password Feature

The Recommended Password feature in the proposed system is designed to improve password strength when users select a limited or imbalanced combination of character types—particularly emojis. This feature generates a suggested password that maintains semantic relevance while offering enhanced randomness and unpredictability. To achieve this, the system ensures that the number of emojis included in the recommended password is at least one more than the user's initial selection. Although there is a general consideration that emojis should not exceed half of the total password length, this is treated as a flexible guideline. If the user selects a high number of emojis, the system prioritizes fulfilling the minimum emoji count, even if it surpasses this threshold. Once the emojis are placed, the system includes all other selected character types in at least the same quantity as specified by the user. If there are remaining character slots, they are filled with randomly chosen characters from a combined character set. All characters are then shuffled before the final password is displayed. This feature ensures that the recommended password is both contextually meaningful and resistant to predictable patterns, balancing usability and security.

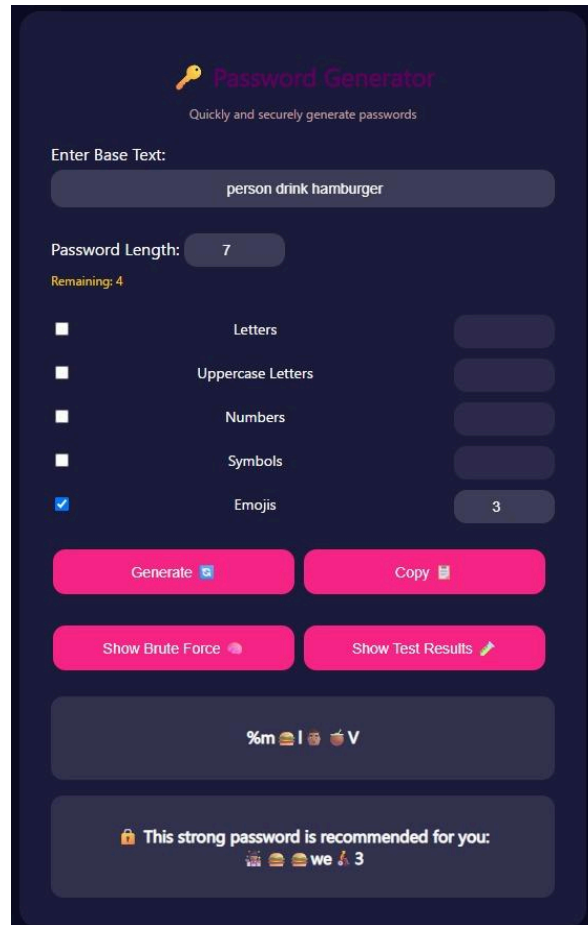


Figure 3.8: Recommended Password Feature

As shown in Figure 3.8, the user selected only 3 emojis and no other character types. In response, the system generated a recommended password containing at least 4 emojis. The remaining 3 characters were filled from the traditional character pool, and all characters were shuffled before presenting the final password suggestion.

3.3.10. Frontend (JavaScript + HTML/CSS)

The frontend of the system is implemented using standard HTML, CSS, and JavaScript, with the goal of providing intuitive, interactive, and responsive user experience. From the outset, the interface is designed to guide users through the password generation process in a controlled and informative way. The frontend is not a passive display but a logic-driven layer that enforces input correctness, controls visibility of system elements, and dynamically reacts to user actions.

When the page is first loaded, all advanced result sections—such as the password output box, brute-force result display, and randomness test panel—are hidden by default. As shown in Figure 3.9, this design helps avoid user confusion by keeping the focus on the main input area. Users can choose which character types to include—such as letters, uppercase letters, numbers, symbols, and emojis—using checkboxes. The number input fields for each type remain disabled until the corresponding checkbox is selected. This approach reduces unnecessary visual elements and prevents users from entering invalid information.

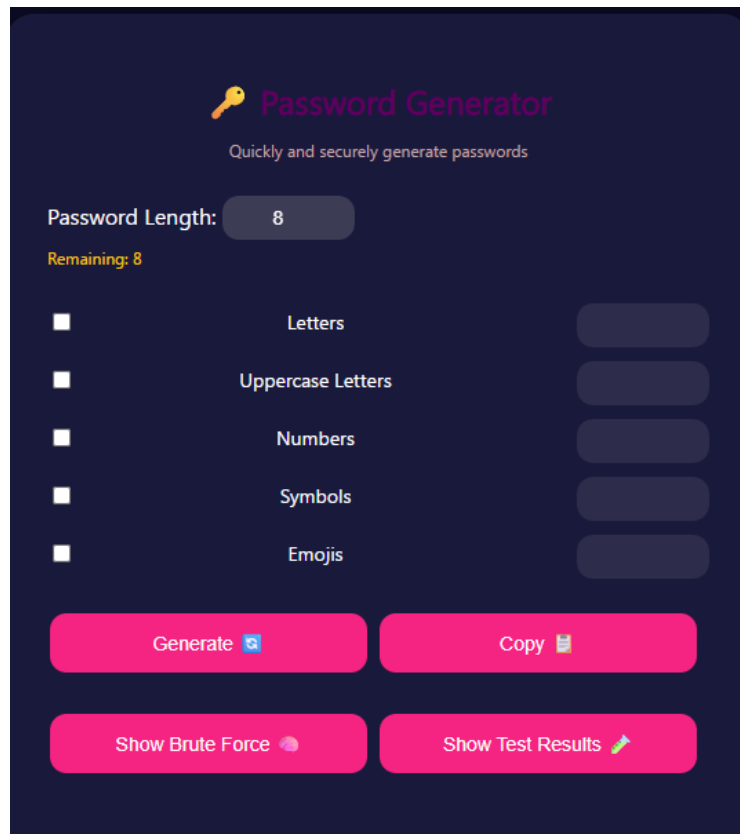
The image shows a mobile application interface for a 'Password Generator'. At the top, there is a yellow key icon and the title 'Password Generator' in pink. Below the title is the subtitle 'Quickly and securely generate passwords'. The 'Password Length' is set to '8' in a grey rounded rectangle. Below this, it says 'Remaining: 8' in yellow. There are five checkboxes, each with a label and a corresponding number input field to its right: 'Letters' (8), 'Uppercase Letters' (8), 'Numbers' (8), 'Symbols' (8), and 'Emojis' (8). All checkboxes are currently unchecked. At the bottom, there are four pink buttons: 'Generate' with a blue key icon, 'Copy' with a clipboard icon, 'Show Brute Force' with a purple bomb icon, and 'Show Test Results' with a green pencil icon.

Figure 3.9: Frontend Features

If the emoji character type is selected, a text input field is revealed for entering a base sentence. This base sentence is then sent to the backend for semantic emoji matching, but only if the user attempts to generate a password. If the text field is left empty while emojis are selected, the system notifies the user and stops the process, ensuring a clear and consistent interaction. This method guarantees that every emoji included in the

password has a meaningful linguistic connection, preventing random or irrelevant emoji use. The password generation is based on parameters set by the user, who specifies the number of characters for each type and the maximum password length. A dynamic counter updates in real-time to show how many characters are left, helping to avoid exceeding the allowed length. If the total number of selected characters is less than the maximum length allowed, the system displays a warning as shown in Figure 3.10 before automatically filling the remaining positions with random characters to achieve a balance between randomness and structure. On the other hand, if the selected characters exceed the maximum limit, the system warns the user.

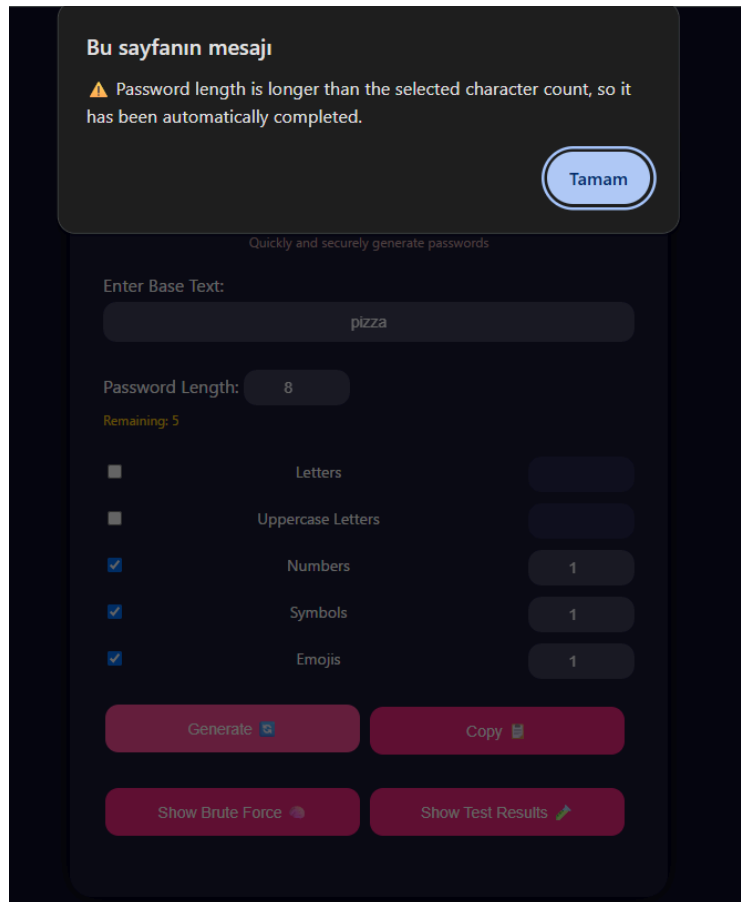


Figure 3.10: Frontend Alert Feature

The generated password is shown immediately on the screen and can be copied easily using a "Copy" button, which also gives a visual confirmation to the user. Additionally, the frontend is closely linked with the backend to support more advanced functions. After a password is created, it is converted into a bitstring and quickly tested in real-time using

several statistical tests, including Monobit, Runs, Cumulative Sums, and Serial tests. These tests are implemented in JavaScript and run instantly within the frontend. Their results are displayed in a clear, color-coded list showing which tests passed or failed, giving users immediate feedback about the quality of their password.

Additionally, the frontend supports a Brute Force Estimation feature. Once a password is created, the user can trigger a brute-force simulation that is handled by the backend but visualized by the frontend. The interface displays whether the password was successfully cracked, how many guesses it took, and how long the process ran. To maintain performance and realism, the simulation stops automatically after a set time (e.g., 30 seconds), and the frontend updates accordingly.

One of the important designs in the frontend is to clean up all the results (password, tests, brute force) when the user makes any changes to the properties needed when creating the password. This prevents stale data from being misinterpreted and reinforces the idea that every password is unique and must be analyzed fresh. The system also supports dual-mode password generation: when the total character count is less than the desired length, two versions of the password are created—one completed with emojis, and one without. Both passwords are presented side by side, allowing users to directly compare their complexity and structures.

The front end is more than just a visual interface; it embodies the concept of usable security by making advanced topics like entropy, semantic relevance, and brute force attack protection understandable and practical for everyday users. Through a JavaScript-powered layout, guided steps, and interactive features, the front end transforms password creation into a secure process while maintaining strong security.

3.3.11. Randomness Tests and Statistical Analysis

A key component of the system's backend is the use of four dynamic randomness tests: the Monobit Test, Runs Test, Cumulative Sums Test, and Serial Test. These tests, written in Python, run automatically on each password as soon as it is generated. Their main goal is to assess the statistical randomness and entropy of the passwords, making sure that the system creates outputs that are not only varied but also strong enough for cryptographic security.

To mitigate issues arising from variable-length characters (especially emojis), each password is first transformed into a uniform binary form using the SHA-256 hash function. This converts any input into a 256-bit binary string, which allows all passwords to be fairly evaluated, regardless of their character encoding. Once hashed, the password's binary sequence is subjected to the following tests:

Monobit Test

The Monobit Test checks whether the number of '1's and '0's in the binary sequence are approximately equal. This is considered the most fundamental test of randomness, since a major imbalance in bits suggests a predictable or biased source. In this project, Monobit Test is used to detect basic weaknesses in password diversity. It flags passwords that lean heavily towards one bit type, which could result from low-entropy sources or repetitive user inputs. Implementing this test ensures that passwords start with a fair bit-level balance before proceeding to more advanced evaluations.

Runs Test

The Runs Test examines sequences of consecutive identical bits (e.g., '11111' or '000') to determine whether such patterns occur with a statistically expected frequency. A truly random binary stream will contain short and long runs distributed in a way that doesn't deviate significantly from expected distributions. This test helps identify hidden

regularities or structural artifacts introduced by the password generation logic. Within our system, the Runs Test plays a critical role in ensuring that emoji placement and fallback completions don't accidentally introduce patterns that attackers could exploit.

Cumulative Sums Test

This test evaluates how the sum of bits (treated as +1 for '1' and -1 for '0') fluctuates over the sequence. It determines whether the cumulative sum drifts too far from zero in any direction, which could indicate subtle trends or long-range imbalances in the data. By analyzing how the bitstream "wanders," the test can detect weaknesses in long passwords where bias may accumulate. In the context of our system, this test is particularly important for analyzing longer emoji-rich passwords that might carry structural irregularities due to semantic groupings in the emoji data.

Serial Test

The Serial Test inspects overlapping pairs (or triplets) of bits, such as '00', '01', '10', and '11', and evaluates whether each appears with roughly equal probability. This test is designed to detect disproportionate representation of certain patterns, which could compromise entropy at the micro-structural level. It is especially useful in identifying repetitive suffixes or emoji combinations that, when hashed, manifest as repeated binary structures. Our implementation includes this test as a final check to ensure the full range of bit-level combinations is used uniformly in hashed passwords.

As a result, these four tests provide a comprehensive randomness assessment. The system applies these four tests to each generated password, allowing users to get real-time feedback on the reliability of their passwords. When the Show test result button is clicked, the test results are displayed on the front end, making the process transparent and informative. This approach also reinforces the system's goal of generating reliable passwords that are resistant to cryptographic attacks.

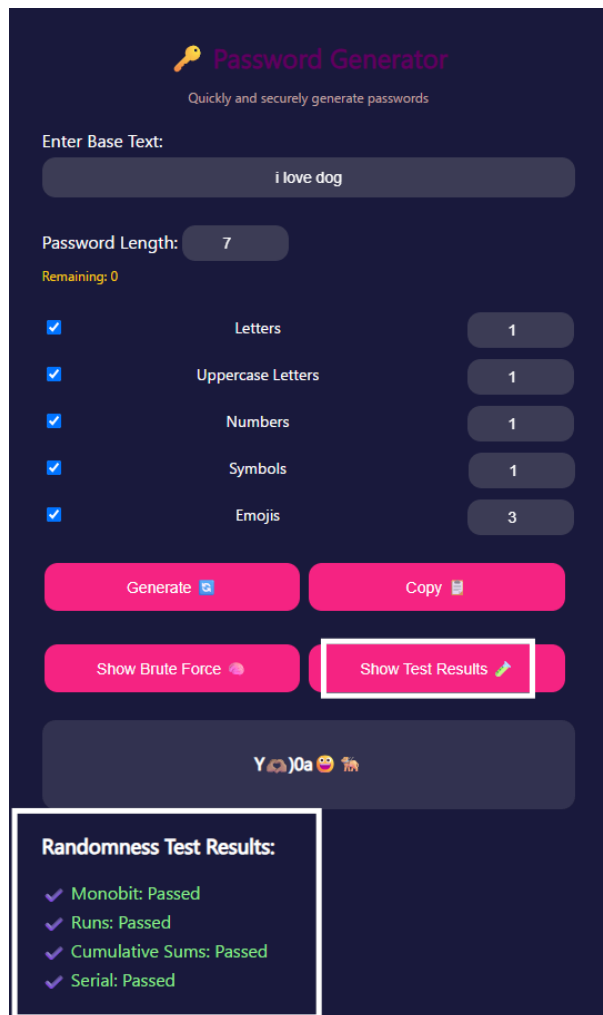


Figure 3.11: Test Features

3.4. Testing

3.4.1 Functional Testing and Manual Test Cases

The first phase of testing aimed to demonstrate that both the front-end interface and the back-end logic were working correctly. To this end, several manual checks were performed across various usage scenarios. These test scenarios included standard operations such as generating a password after a valid input, as well as conditions such as attempting to generate an emoji-based password without entering a sentence or when more character types than the specified password length were selected. Each scenario was manually tested

to ensure that it prevented invalid inputs and that the system responded with appropriate feedback in the face of unexpected user behaviors. For example, when the “Generate” button was clicked without selecting any character type, the system displayed a warning and stopped the process. Similarly, if the sum of the selected character types exceeded the desired password length, the interface prompted the user to reconsider their choices. Additionally, it was verified that the “Copy,” “Brute Force,” and “Test Results” buttons were only activated after a valid password was created, effectively preventing unwanted or invalid operations.

3.4.2 Real-Time Randomness Testing

To assess the statistical strength and unpredictability of generated passwords, the system implements a set of real-time randomness tests that run immediately after each password is created. These tests, which include Monobit Test, Runs Test, Cumulative Sums Test, and Serial Test, are based on bit-level statistical analyses adapted from NIST-style methodologies. Since emojis and special characters can vary in byte size, each generated password is first hashed using SHA-256 to ensure uniformity in the test input. The hashing process converts any input, regardless of length or encoding, into a 256-bit binary output, providing a standard structure for all randomness evaluations. The Monobit Test verifies whether the number of '0's and '1's in the hash are approximately equal. The Runs Test evaluates whether sequences of consecutive bits appear with expected frequency. The Cumulative Sums Test tracks directional drift over time in the bit sequence, while the Serial Test ensures balanced distribution of 2-bit patterns. These tests are implemented directly in JavaScript and are executed on-the-fly. Their results are presented in the user interface with clear visual indicators, enabling users to immediately assess whether the password meets statistical randomness expectations.

3.4.3 Brute-force Simulation Testing

The project includes a custom brute-force simulation engine developed using Flask, which aims to replicate a real-world offline attack scenario. Once a password is generated, users can activate the brute-force module, which analyzes the character types used and dynamically creates a search space accordingly. For instance, if the password includes only lowercase letters and numbers, the brute-force engine restricts guesses to that pool. Incorporating emojis into passwords substantially increases their complexity, as an attacker cannot ascertain whether emojis are included. This uncertainty significantly reduces the probability of a successful brute-force attack. The simulation performs random guesses until either the password is cracked or a time limit of 30 seconds is reached. If the password is cracked within that time, the guessed value, the number of attempts, and the elapsed time are shown on the frontend. Otherwise, a failure message with statistics is presented. This testing mechanism helps users understand the practical implications of their choices, especially how the inclusion of diverse character types—particularly emojis— affects brute-force resistance.

3.4.4 Integration and Regression Testing

As the system evolved, new features such as emoji fallback completion, dual password generation, and dynamic test result display were added. After each major addition, integration and regression tests were conducted to ensure that existing components continued to function as expected. For example, after implementing the automatic filling of missing characters in passwords, the behaviors of the "Generate," "Copy," and "Brute Force" buttons were reevaluated to confirm that their logic still aligned with the new output format. These tests prevented silent errors and maintained the reliability of the overall system. They also helped confirm that the interface dynamically adjusted to user changes without retaining outdated data or producing inconsistent results.

3.4.5 User Interface Feedback Testing

Finally, a dedicated round of testing was carried out to validate the clarity and responsiveness of the user interface. Each interactive element whether a button, text input, or checkbox was tested to ensure that it responded appropriately to valid and invalid actions. For example, result boxes were tested to appear only after valid generation; brute-force or test results were hidden when new parameters were selected; and alerts were verified forever tent, error-resistant, and informative. The frontend thus not only facilitated interaction but also reinforced secure behavior by enforcing correct usage flows and preventing ambiguous outcomes.

4. EXPERIMENTAL RESULTS

4.1 Brute-Force Test Results

Brute-force tests were performed to assess the core efficiency of the password cracking mechanism. These tests involved analyzing the time required to discover the password and the total number of attempts, using password samples of varying lengths and complexity levels. In some tests, the password was found successfully, i.e. when no emoji was used, while in some tests, the password could not be found due to the complexity of the character set and the use of emoji. Whether the password was found successfully, the number and duration of attempts, the best guess and similarity rates are given in Table 4.1.

Table 4.1: Comparison of Brute-Force Test Results

TEST NO	PASSWORD (TARGET)	LENGTH	CHARACTER SET (Charset)	PASSWORD FOUND?	NUMBER OF TRIALS	DURATION(s)	BEST GUESS	SIMILARITY SCORE
TN - 1	&K5	3	symbols,numbers ,uppercase letters	YES	506,082	0,93	&K5	100%
TN - 2	r<0	3	numbers,symbols ,letters	YES	132.665	1,12	r<0	100,00%
TN - 3	☀️ (5	3	numbers,symbols ,emojis	NO	636,056	1,14	a(5	66,07%
TN - 4	5G{u	4	letters,symbols, numbers, uppercase letters	YES	36,498,421	61,58	5G{u	100%
TN - 5	>6🚲p	4	letters,numbers, symbols, emojis	NO	54.700.816	72,75	a>6p	66,67%
TN - 6	^Lx6	4	letters,numbers, symbols,uppercase se letters	YES	42.891.441	72,75	^Lx6	100%
TN - 7	🦊z0*	4	letters,symbols, numbers,emojis	NO	54,700,816	94,43	az0*	60%
TN - 8	[jB0	4	letters,symbol, numbers, uppercase letters	YES	49.681.307	99.18	[jB0	100,00%
TN - 9	1C🦊<	4	symbol,numbers, uppercase letters,emojis	NO	54,700,816	99,18	a1C<	54,55%

Interpretation of Brute-Force Results: The results of the brute-force tests show that as the complexity of the password increases, the trial period and the number of attempts increase significantly. Especially in test scenarios that include emojis, the inability to find the password reveals a difficulty directly proportional to the expansion of the character set. In tests conducted with simpler character sets, the system was able to successfully decipher the password.

4.2 Brute-Force Similarity Score Test Results

As shown in Table 4.1, brute force tests were conducted to evaluate the performance of the password cracking system and compare the cracking time and similarity ratios between passwords with and without emojis. Additionally, Figure 4.1 shows the relationship between password length and similarity score during the password recovery process, while also comparing the impact of using and not using emojis on the similarity ratio. These results highlight the differences in security effectiveness between the two password types.

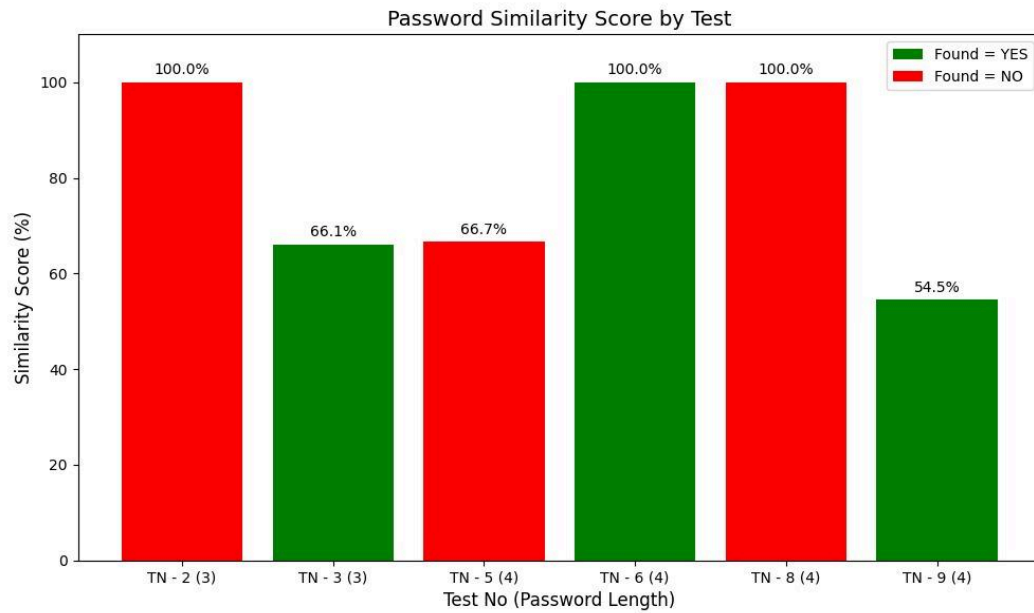


Figure 4.1: Comparison of password length and similarity score for finding the password

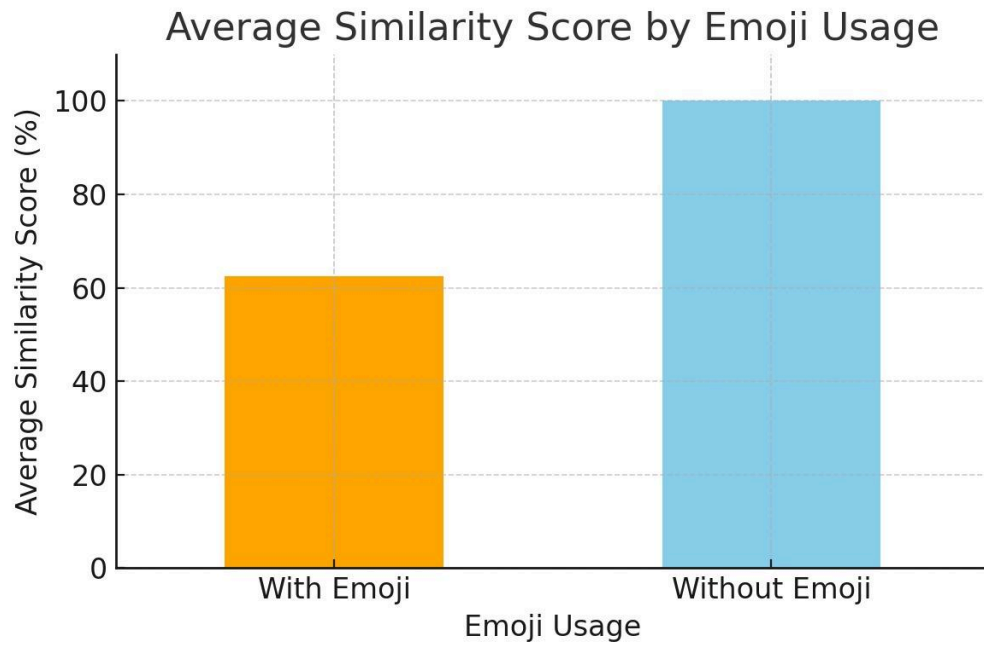


Figure 4.2: Comparison of the effect of using emoji and without emoji on similarity rate

Interpretation of Brute-Force Results: Table 4.3 clearly shows how the cracking time and similarity rates of passwords containing emojis differ compared to passwords without emojis. In particular: Looking at the TN - 3 (☀️) (5) and TN - 2 (r<0) tests, the cracking time (1.14 seconds) and similarity rate (66.07%) of the password containing emojis are significantly lower than the passwords without emojis. Similarly, in the TN - 5 (🚲p) and TN - 9 (1C 🦊 <) tests, the cracking time of passwords containing emojis is longer and the similarity rates are lower. These findings show that passwords containing emojis are more difficult to crack and require more time and trials for brute-force attacks.

4.3 Test Case Scenarios and Success Evaluation












This section provides a detailed description of the different test cases performed to check how well the password generation and security system works. Each test case demonstrates a specific situation, including the steps followed, what is expected to happen, and what actually happens. These tests ensure that the system works properly and meets the performance requirements in different scenarios. Table 4.3 below summarizes these test cases and their objectives.

Table 4.2: Comparison of Test Case Scenarios and Success Evaluation

TEST ID	DEFINITION	PRECONDITION	INPUT
TC - 01	Create text-only passwords with emojis	<ul style="list-style-type: none"> • Application open • “Emojis” selected • Base Text: “person” 	Text = “person”, EmojisCount = 8
TC - 02	Select only uppercase letters and symbols	<ul style="list-style-type: none"> • Application is open • No emojis are selected • Text field is hidden 	Text = “”, UppercaseCount = 8, SymbolsCount = 3
TC - 03	Warning when password length value is entered incompletely	<ul style="list-style-type: none"> • Application is open • At least one character type is marked 	Length = 3, LettersCount = 2
TC - 04	Emoji selected, Text field left blank	<ul style="list-style-type: none"> • “Emojis” selected • Text field visible 	Text = “”, EmojisCount = 5
TC - 05	Starting brute-force testing	<ul style="list-style-type: none"> • The password is generated and appears on the screen. 	Password = “random password is generated”
TC - 06	View randomness test results	<ul style="list-style-type: none"> • The password is generated and appears on the screen. 	Password = “random password is generated” Password Length = 8, Letters L. = 2, Uppercase L. = 2, Numbers L. = 2, Symbols L. = 2
TC - 07	Copy the password	<ul style="list-style-type: none"> • The password appears on the screen 	Password Length = 2 Text = “dog”, Emoji selected
TC - 08	UI visibility control	<ul style="list-style-type: none"> • The application must have been successfully installed and launched • The user must have completed the login screen and been directed to the master password generation screen 	The application home screen opens without any user input
TC - 09	Maximum length performance test	<ul style="list-style-type: none"> • All character types are selected 	Length = 32
TC - 10	Multiple productions with the same text	<ul style="list-style-type: none"> • “Emojis” selected • Text: “student” 	Password Length = 2, Text = “student”, Count = 5
TC - 11	Whether the system works properly when the input contains special characters, Turkish characters and spaces.	<ul style="list-style-type: none"> • Emoji and characters area selected 	Text = “queen”, Password Length = 8, emojis, letters and symbols selected
TC - 12	Automatically display text field when emoji checkbox is checked	<ul style="list-style-type: none"> • Page loaded properly • Emoji box not yet checked 	Check the emoji box
TC - 13	How does the system behave if the user tries to create a password without selecting any character type?	<ul style="list-style-type: none"> • The application interface must be properly loaded • The password generation area must be accessible • All character type boxes (letters, numbers, symbols, emojis) must be unchecked • A valid number (for example: 12) must be entered in the length field 	Length: 12 Character Types: None selected
TC - 14	Checking if the correct emoji is assigned even if the user enters a word with missing letters	<ul style="list-style-type: none"> • Application open • Emoji box checked • Text input field visible 	Text: “winer” (correctly: “winter”)

Table 4.3: Comparison of Test Case Scenarios and Success Evaluation Continued

TEST ID	STEPS	EXPECTED OUTPUT	ACTUAL RESULT	STATUS
TC - 01	1. The application interface opens and makes sure the page is fully loaded. 2. The “Emojis” box is checked. 3. The word person is written in the “Base Text” field. 4. The number of emojis is entered as 8. 5. The “Generate Password” button is clicked. 6. It is checked whether the emojis corresponding to the word person are included in the generated password.	An 8-character password that is text-based and includes emojis	A valid password like R&1V WJ was generated	✓ Passed
TC - 02	1. Open the application. 2. All character options are removed; Only the "Capital Letter" and "Symbol" boxes are checked. 3. The text field is left blank without being visible. 4. The "Password Length" value is not entered, and is marked as Uppercase: 8, Symbols: 3. 5. The "Generate Password" button is clicked. 6. It is verified that the generated password consists of only uppercase letters and symbols.	Password generation consisting of capital letters and symbols	A password like G#T8Q%RW was generated	✓ Passed
TC - 03	1. The application interface opens. 2. At least one character type (for example: Letters) is selected. 3. The value 3 is entered in the “Password Length” field. 4. The number of characters (Letters: 2) is entered. 5. The “Generate Password” button is clicked. 6. The minimum length warning is expected from the system.	“Remaining: 1” warning should be displayed	⚠ Password length is longer than the selected character count, so it has been automatically completed. "Notification is displayed.	✓ Passed
TC - 04	1. The application interface opens. 2. The “Emojis” box is checked. 3. The text field becomes visible but is left blank. 4. The number of emojis is entered as 5. 5. The “Generate Password” button is clicked. 6. The system expects a “Text field is empty” warning.	Warning: “Text input is required for emoji selection”	"Please enter base text for emojis! "Notification is displayed.	✓ Passed
TC - 05	1. The application interface opens 2. The password creation steps are completed and the password appears on the screen. 3. The “Start Brute-force Test” button is clicked. 4. The system starts the brute-force process to analyze the password. 5. Information such as duration, number of attempts and success status is displayed on the screen.	< Should not be broken in X seconds or not broken in time message	✗ Password could not be cracked within the time limit. ⌚ Attempts Made: 28.908.692 ⌚ Time Taken: 30 seconds	✓ Passed
TC - 06	1. The application interface opens 2. The “Generate Password” process is completed and the password appears on the screen. 3. The “Randomness Test” button is clicked. 4. The system examines the components of the password (letters, numbers, symbols, emojis). 5. Randomness scores and character distribution analysis are reflected on the screen.	All tests (Monobit, Runs, dll.) are successful	Randomness Test Results: ✓ Monobit: Passed ✓ Runs: Passed ✓ Cumulative Sums: Passed ✓ Serial: Passed	✓ Passed

TC - 07	<ol style="list-style-type: none"> 1. The application interface opens 2. After the password is generated, it is displayed on the screen. 3. The "Copy" button is clicked. 4. A feedback message indicating that it has been successfully copied to the clipboard (e.g. "Password copied") is expected. 5. The content on the clipboard can be checked manually. 	A "copied to clipboard" notification should appear	"  Password copied!" notification is displayed.	 Passed
TC - 08	<ol style="list-style-type: none"> 1. Open the application. 2. Make sure the page is fully loaded. 3. Make sure the following interface elements are visible: 4. Character selection boxes (Letters, Uppercase, Numbers, Symbols, Emojis) 5. Password length entry 6. Text field (hidden by default) 7. "Generate Password" and other functional buttons 	Text area, emoji box, buttons must be visible	All UI elements loaded successfully	 Passed
TC - 09	<ol style="list-style-type: none"> 1. The application interface opens. 2. All character types (Letters, Uppercase, Numbers, Symbols, Emojis) are selected. 3. The password length is entered as 36. 4. The "Generate Password" button is clicked. 5. The system response time and password format are observed. 	The application should generate passwords without freezing	No performance issues occurred	 Passed
TC - 10	<ol style="list-style-type: none"> 1. The application interface opens 2. The "Emojis" box is checked. 3. The word "student" is entered in the text field. 4. The password length is set to 2. 5. The "Generate Password" button is clicked five times in succession. 6. Each generated password is checked to see if it is different. 	5 different passwords should be created (differences depend on randomness)	Different codes were created in each production.	 Passed
TC - 11	<ol style="list-style-type: none"> 1. The application interface opens 2. The word ""queen"" is entered in the text field. 3. The Emojis, Letters and Symbols boxes are checked. 4. The password length is determined as 8. 5. The "Generate Password" button is clicked. 6. It is checked whether the system processes special characters and whether the password is generated correctly. 	A meaningful and valid password should be created from the Turkish character text.	Generated compatible password (system did not crash)	 Passed
TC - 12	<ol style="list-style-type: none"> 1. The application page opens properly. 2. When the page opens, it is verified that the text area is hidden by default. 3. The "Emoji" box is checked. 4. It is observed whether the text area becomes visible. 	The text input field should be visible automatically	The text field appeared dynamically on the screen	 Passed
TC - 13	<ol style="list-style-type: none"> 1. The application interface opens. 2. Write 12 in the "Password Length" field. 3. All character type boxes (Letters, Uppercase, Numbers, Symbols, Emojis) are left blank. 4. Click the "Generate Password" button. 5. Wait for a warning message from the system that no character type has been selected. 	A warning message should appear saying "You must select at least one character type"	 Please select at least one character type before generating a password!	 Passed
TC - 14	<ol style="list-style-type: none"> 1. The application is opened and the "Emoji" box is checked. 2. When the text field becomes visible, the text "winer" (correctly: "winter") is entered. 3. The "Generate Password" button is clicked. 4. The system matches the word "winer" with winter as the closest correct word. 5. The emoji corresponding to the word winter (e.g.  or ) is included in the password. 6. The password is checked to see if the relevant emoji is included. 	The app finds the closest match to the word "winter" and generates a password with the corresponding emoji (for example )	Password: 8    (fitting emoji found and added to password)	 Passed

4.4 Randomness Test Results

Table 4.5 shows the results of statistical tests performed to evaluate the randomness levels of generated passwords. Each row contains the results of four basic randomness tests (Monobits, Runs, Additive, Serial) for a password generated with different character sets and password lengths. As a result of these tests, it is seen that the tests are more successful and reliable when emojis are added or the password length is increased.

Table 4.4: Comparison of Randomness Test Results

TEST ID	PASSWORD	CHARACTER SET	LENGTH	MONOBIT	RUNS	CUMULATIVE	SERIAL
TR - 1]1📖T	Uppercase Letters Numbers Symbols Emojis	4	✓	✓	✓	✓
TR - 2	%1wC	Letters Uppercase Letters Numbers Symbols	4	✗	✓	✓	✓
TR - 3)br]IM24	Letters Uppercase Letters Numbers Symbols	8	✗	✓	✓	✓
TR - 4	J_11a%🔥d	Letters Uppercase Letters Numbers Symbols Emojis	8	✓	✓	✓	✓
TR - 5	5Q!teY7💙[o8#	Letters Uppercase Letters Numbers Symbols Emojis	12	✓	✓	✓	✓
TR - 6	_Yg5A4#X?zb2	Letters Uppercase Letters Numbers Symbols	12	✓	✓	✓	✓

DISCUSSION

The main problem addressed in this study was how to generate passwords that are not only secure and easy to remember, but also meaningful, considering the increasing risks associated with traditional password habits. As stated in the problem statement, users often choose weak or repetitive passwords based on personal information such as name, date of birth, or phone number, making them easy targets for brute force attacks. To mitigate this risk, our project uses a Natural Language Processing (NLP) approach to generate meaningful passwords that combine both randomness and emojis. Experimental results show that our system successfully analyzes the grammatical structure of the input sentence, extracts subject-verb-object roles using spaCy, and matches these keywords with appropriate emojis through a multi-step system that includes lemmatization, stemming, and fuzzy matching. The weighted emoji selection algorithm, which prioritizes emojis in the order of object - verb - subject according to the user's input text, significantly improves the visual distinctiveness and memorability of the generated passwords. Our design ensures variability by including randomness in emoji selection, thus preventing the same outputs for repeated inputs.

Furthermore, the system integrates statistical randomness tests such as Monobit, Runs, Cumulative Sums, and Serial Tests to verify the cryptographic strength of each output. These tests confirmed that most of the generated passwords exhibited high levels of entropy, demonstrating the transparency and scalability of our approach to the user. In addition, our dynamic brute force simulation demonstrated the increased resistance of emoji-rich passwords to attack patterns, particularly since attackers cannot reliably guess whether a password contains Unicode emojis or standard characters.

Despite this working model of the project, there are still sources of error. One of them is that the system can misinterpret user intent if the input sentence is ambiguous or grammatically incorrect. Similarly, while RapidFuzz provides effective fuzzy matching, it can occasionally select emojis that are semantically distant from the intended word, potentially leading the password away from its intended meaning. Another limitation is the reliance on a static emoji dataset; emerging language trends or newly introduced emojis may not be immediately reflected in the system unless the dataset is manually updated.

During testing, several anomalies were observed, especially when the input sentences contained abstract or idiomatic expressions. In such cases, the NLP model failed to extract a meaningful object component and generated emojis that were closest to the words in the text. These issues point to the difficulties of interpreting natural language when generating passwords, especially across dialects or everyday usage.

The broader significance of this project is that by making password generation a visually engaging and linguistically intuitive process, the system improves both usability and security.

Future work can explore the integration of multilingual NLP models to support password generation in various languages and expand the accessibility of the system. Additionally, adaptive learning algorithms can be used to personalize emoji suggestions based on user preferences or password history, increasing user satisfaction and memorability.

In conclusion, this project offers a creative, user-friendly, and secure alternative to traditional password generation approaches. By integrating linguistic understanding and semantic emoji matching into the password generation process, it successfully balances memorability with strong security. The results demonstrate that NLP-based security solutions have significant potential to improve both digital security and the overall user experience.

CONCLUSIONS

The main purpose of this project is to overcome the shortcomings of traditional password generation methods by combining natural language processing (NLP) and matching techniques and to help users create passwords that are both secure and easy to remember with emojis. The system designed in this project first examines the text structure of the user's input if they use emojis in password creation. It detects basic grammar elements such as subject, verb and object and creates meaningful, personalized passwords by matching these elements with the relevant emojis according to their semantic relationships. In addition, the system not only offers emoji-based solutions, but is also structured to meet different password preferences by offering password creation based on classic character groups (letters, capital letters, numbers and symbols). The technical tests and simulations performed played an important role in demonstrating the security potential of the system. Randomness evaluations conducted through Monobit, Runs, Additive Sums and Serial Tests showed that the generated passwords contained statistically sufficient randomness. In addition, due to the diversity offered by the Unicode character structure and the inclusion of emojis, significant increases were recorded in password cracking times against brute force attacks. For example, although a standard 4-character password can be cracked relatively quickly, it has been observed that using emojis in a password of the same length significantly increases the time required to crack it. This shows that the system not only strengthens security but also improves its resistance to various attack methods. In addition, when the character types specified by the user are missing, the system exhibits an adaptable and dynamic structure by automatically completing the missing part with emojis or classic characters. The user interface was designed as web-based and offered user-friendly features such as real-time password testing and security analysis. One of the most important takeaways from this project is that the system not only produced more memorable passwords by using the contextual meaning provided by natural language, but also improved overall user password generation through a more visually interactive interface.

This approach allows the system to provide a personalized experience without compromising security.

The fact that the system currently only works with English language input is a limitation in terms of multilingual user accessibility. In addition, limited mobile compatibility and the fact that attack modeling is currently based only on brute force simulations highlight areas for future improvement. More detailed surveys can be conducted and user feedback can be collected to understand how user-friendly the system is. An AI-based system that provides password suggestions by taking into account users' previous password creation habits can be developed.

As a result, this project offers an approach that combines security, usability, and meaningful emojis for words. The integration of natural language processing techniques with cryptographic methods offers not only a new password creation method, but also a user-friendly, personalized structure. As a result, the study provides a strong foundation for future password systems with both its technical achievements and user-centered design, paving the way for a new generation of smart, secure, and flexible password creation.

REFERENCES

[1] Walid Ali Sulaiman Ali Alothman, Evaluating Passwords User Behavior and the Psychology of Password Management (2019)

https://www.researchgate.net/publication/334968445_Evaluating_Passwords_User_Behavior_and_the_Psychology_of_Password_Management

[2] Farhana Zaman Glory, Atif U1 Aftab, Olivier Tremblay-Savard, Noman Mohammed, Strong Password Generation Based On User Inputs (2019)

<https://ieeexplore.ieee.org/abstract/document/8936178>

[3] Tehreem Hussain*, Kiran Atta, N. Z. Bawany, Tehreem Qamar, Passwords and User Behavior (2018)

https://www.researchgate.net/publication/319815106_Passwords_and_User_Behavior

[4] Golla, M., & Dürmuth, M. (2017). EmojiAuth: Quantifying the Security of Emoji-based Authentication. USEC. (p. 4-6)

[5] SSRN (2020). Are Emojis Creating a New or Old Visual Language for New Generations?(p. 56-60)

[6] Rukhin, A. et al. (2010). A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications (SP 800-22 Rev1a). NIST.

[\[https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf\]](https://nvlpubs.nist.gov/nistpubs/legacy/sp/nistspecialpublication800-22r1a.pdf)

[7] Bishoi, A., et al. (2020). Xorshift Random Number Generators from Primitive Polynomials. Journal of Telecommunications and Information Technology.

[\[https://journals.pan.pl/Content/118539/PDF/bishoi_Xorshift%20random%20number.pdf\]](https://journals.pan.pl/Content/118539/PDF/bishoi_Xorshift%20random%20number.pdf)

[8] Fadhilah, W. et al. (2021). Implementation of the Fisher-Yates Shuffle Algorithm in Exam-Problem Randomization. ResearchGate.

[\[https://www.researchgate.net/publication/356781365\]](https://www.researchgate.net/publication/356781365)

[9] Vaithyasubramanian, S., Prakash, R., & Udhayan, J. (2014). An Analysis of Markov Password Against Brute Force Attack for Effective Web Applications. Middle-East Journal of Scientific Research, 22(4), 5825–5827.

- [10] Kjellevand, T., & Rauhut, R. (2018). Evaluating the Security and Usability of Emoji-Based Authentication. Master's Thesis, Norwegian University of Science and Technology (NTNU).
- [11] Chen, X., Howard, D., Guggeri, F., & Bosu, A. (2019). SEntiMoji: An Emoji-Powered Learning Approach for Sentiment Analysis in Software Engineering. Proceedings of the 2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR), 438–442. <https://doi.org/10.1109/MSR.2019.00077>
- [12] Strathmore University. (2021). BITREST Password Manager: Final Documentation Report for Information Systems Project 2. Nairobi, Kenya.
- [13] T. Seitz, F. Mathis, and H. Hussmann, “The bird is the word: A usability evaluation of emojis inside text passwords,” in Proc. 29th Australian Conf. Human-Computer Interaction (OzCHI), Brisbane, Australia, Nov. 2017, pp. 351–359.
- [14] N. K. Jha, “An approach towards text to emoticon conversion and vice-versa using NLTK and WordNet,” in 2018 2nd Int. Conf. Data Science and Business Analytics (ICDSBA), Changsha, China, Sept. 2018, pp. 161–165.
- [15] X. Nan, Password Security Reinforcement via Combining Unicode Character Set, M.Eng. thesis, Dept. of Computer Sci. and Communications Eng., Waseda Univ., Tokyo, Japan, Jul. (2024).