Password Strength Checker

Software Requirements Specification (SRS)

Author: Shanmukha (Shaun)
Date: April 28, 2025

---

1. Introduction

1.1 Purpose

This SRS outlines the requirements for a Password Strength Checker web application. The tool evaluates password security in real-time, provides actionable feedback, estimates crack time using a custom algorithm, and checks for breaches using the HaveIBeenPwned API. It aims to enhance user security awareness and showcase my web development skills as a 2nd-year CSBS student at GITAM.

1.2 Scope

A primarily client-side web application built with HTML, CSS, and JavaScript, with potential for lightweight Node.js backend integration for API handling. Core features include strength analysis, crack time estimation, and breach detection. Future enhancements (e.g., Password Generator, Dark Mode) are noted. Target deployment: Glitch.me.

2. Overall Description

2.1 Product Perspective

A standalone web tool for casual users, developers, and recruiters, showcasing my full-stack skills (HTML, CSS, JS, Node.js). Integrates with the HaveIBeenPwned API for breach checks. Hosted on Glitch.me for quick deployment and sharing.

2.2 Product Functions

- Real-time password strength detection with visual meter (Weak, Fair, Strong).
- Detailed criteria analysis (length, case, digits, special chars).
- Personalized suggestions for improvement.
- Custom crack time estimation based on entropy and attempts per second.
- Breach detection via HaveIBeenPwned API.

- Password visibility toggle.

2.3 User Classes and Characteristics

- **End-users**: General users checking password strength.
- **Developers**: May extend or integrate the tool.
- **Recruiters**: Assess my technical and UI/UX skills.

2.4 Operating Environment

- Compatible with Chrome, Firefox, Edge, Safari (latest versions).
- Responsive for mobile (320px+) and desktop.
- Hosted on Glitch.me (supports Node.js, frontend assets, and API calls).

2.5 Design Constraints

- Primarily client-side; optional lightweight Node.js backend for API handling.
- Secure API calls (HTTPS) for breach checks.
- No password logging or tracking.
- Adheres to Glitch.me's resource limits (e.g., 200MB storage, API rate limits).

3. System Features

3.1 Real-Time Strength Meter

- **Description**: Updates strength as user types.
- **Criteria**: Length, variety (lowercase, uppercase, digits, special chars).
- **Output**: "Weak" (red), "Fair" (yellow), "Strong" (green).
- **Acceptance**: Strength changes within 100ms. This ensures a smooth user experience, as delays beyond 100ms are noticeable per Nielsen's usability studies, impacting perceived responsiveness.

3.2 Password Criteria Analysis

- **Checks**:
    - Length ≥ 8 (recommended, per NIST guidelines, as shorter passwords are exponentially easier to crack).
    - At least one lowercase, uppercase, digit, special character (increases entropy, reducing brute-force success).
- **Output**: Checkmark (✓) or cross (✗) for each criterion.
- **Acceptance**: Accurate reflection of input within 50ms, ensuring near-instant feedback for usability, aligning with Google's UX benchmarks for real-time interactions.

3.3 Suggestions Panel

- **Description**: Displays tips (e.g., "Add a digit", "Use special char").
- **Acceptance**: Suggestions appear instantly and are relevant to missing criteria.

3.4 Time to Crack Estimator

- **Description**: Estimates crack time using:
    - Entropy = (password.length * $\log_2$(charSetSize)) / scalingFactor.
    - charSetSize: 26 (lower), 26 (upper), 10 (digits), 32 (special).
    - scalingFactor: 3.5 (≤2 char types), 1.5 (>2 char types).
    - Attempts per second: 100,000,000 (based on modern GPU cracking speeds).
    - Seconds to crack = 2^entropy / attemptsPerSecond.
    - Formatted output (e.g., "Instantly", "3 hours").
- **Acceptance**: Matches calculated time within 10% accuracy.

3.5 Breach Detection

- **Description**: Checks password against HaveIBeenPwned API.
- **Output**: Warning with breach count (e.g., "Seen in 293 breaches").
- **Acceptance**: Returns result within 500ms with network latency, balancing speed and API response time.

3.6 Show/Hide Password Toggle

- **Description**: Toggles password visibility with an icon.
- **Acceptance**: Functionality works without affecting analysis.

3.7 Future Features

- **Password Generator**: Will generate passwords based on user-defined parameters (e.g., length 8-20, toggle for numbers, symbols, uppercase, lowercase). Includes a "Copy to Clipboard" feature for usability.
- **Dark Mode Toggle**: User-selectable theme with persistent preference (saved in localStorage). Includes color contrast adjustments for accessibility (WCAG 2.1 compliant).

4. External Interface Requirements

4.1 User Interfaces

- **Layout**: Centered card with input field, strength meter, feedback, suggestions.
- **Design**: Clean, minimal, color-coded (red/yellow/green).
- **Acceptance**: Usable on 320px screens, intuitive navigation.

4.2 Hardware Interfaces

- Runs on any browser-enabled device (PC, tablet, smartphone).

4.3 Software Interfaces

- **HaveIBeenPwned API**: For breach checks (secure HTTPS).
- **Libraries**: Optional Tailwind CSS for styling.
- **Node.js**: Optional for API handling on Glitch.me.

4.4 Communications Interfaces

- HTTPS for API calls; Glitch.me handles hosting and networking.

5. Non-Functional Requirements

5.1 Performance

- Feedback within 100ms.
- Page load < 2s on 1Mbps connection.
- Optimized for Glitch.me's free tier (e.g., 200MB storage).

5.2 Security

- No password storage or logging.
- Secure API key handling: API keys (if used) stored in Glitch.me environment variables (.env) to prevent client-side exposure, following OWASP best practices.
- API calls via HTTPS.

5.3 Maintainability

- Modular JS code (functions/components).
- Easy criterion updates.

5.4 Accessibility

- Keyboard navigable with proper focus management (e.g., tab order for input, toggle).
- Screen reader compatible with ARIA labels (e.g., aria-live for strength updates, aria-label for toggle button).

5.5 Testing Approach

- **Unit Testing**: Test core functions (e.g., entropy calculation, criteria checks) using Jest.
- **Integration Testing**: Verify API integration with HaveIBeenPwned for breach checks.
- **User Acceptance Testing**: Validate usability with 5+ users (e.g., ensure suggestions are clear, UI is intuitive).
- **Performance Testing**: Measure feedback latency (<100ms) using browser dev tools.

6. User Stories

- As a user, I want to see my password strength instantly to know if it's secure.
- As a developer, I want modular code to extend the tool.
- As a recruiter, I want a clean UI to evaluate Shaun's skills.

7. Assumptions and Dependencies

- User has a modern browser.
- Network required for breach checks.
- Glitch.me free tier limits are respected (e.g., storage, API calls).
- Optional Node.js backend for API proxy if needed.

8. Glossary

- **Entropy**: Measure of password unpredictability.
- **Brute-force**: Exhaustive password guessing.
- **Pwned**: Password compromised in a breach.