Password Strength Meter (PSM)

Relazione tecnica

Ingegneria del Software – A.A. 2025/2026

Prof.ssa Marina Mongiello

Referenti gruppo: Albore Tiziano, Lorè Alessio e Zambetti Camilla

Gruppo 74

Belviso, Vegliante, Didonna

Data: 2026-01-07

Indice

1 Introduzione

1.1 Ambito applicativo e obiettivo

La scelta di una password robusta è ancora oggi uno dei punti più critici nei processi di registrazione e autenticazione: l'utente tende a preferire password brevi, prevedibili e spesso correlate a informazioni personali (nome, cognome, email, anno di nascita), mentre i sistemi richiedono requisiti minimi che non sempre guidano verso scelte realmente sicure. In questo contesto si colloca **Password Strength Meter (PSM)**, un prototipo end-to-end che fornisce: una *valutazione* della forza della password su scala 0–100 in tempo reale, e una *validazione* finale basata su policy che impedisce l'uso di password deboli o facilmente riconducibili all'utente.

1.2 Problema e limiti delle soluzioni esistenti

Molte applicazioni adottano *password meters* basati su regole semplici (es. lunghezza e presenza di classi di caratteri) oppure su stime più sofisticate (pattern matching e dizionari). Linee guida recenti raccomandano di privilegiare password più lunghe e di controllare la presenza di password comuni/compromesse, riducendo l'enfasi su regole di composizione troppo rigide. Tuttavia, anche con un indicatore di forza, l'utente può essere indotto a credere che una password sia "accettabile" solo perché ottiene un punteggio discreto, pur contenendo elementi problematici (sequenze, ripetizioni, parole comuni "decorate" oppure informazioni personali).

1.3 Soluzione proposta

Il progetto PSM affronta il problema distinguendo esplicitamente **forza stimata** e **validazione**:

- **Forza stimata**: punteggio 0–100 e livello qualitativo per guidare l'utente mentre digita.
- **Validazione**: verifica finale dei vincoli minimi e blocco quando la password include dati personali (anche con sostituzioni tipo leet).

Il cuore del sistema è un **engine condiviso** (principio *Single Source of Truth*, SSOT): la stessa logica di valutazione viene riutilizzata da UI, API e pipeline sperimentale. La soluzione viene inoltre valutata confrontandola con una baseline nota (`zxcvbn`) [6], tramite esperimenti riproducibili, dashboard e export dei risultati.

2 Stato dell'arte

2.1 Password, attacchi e misure di robustezza

La robustezza di una password può essere analizzata da due prospettive complementari:

- **Proprietà sintattiche** (lunghezza, varietà di caratteri, assenza di pattern banali);
- **Resistenza alla *guessability***: quanto rapidamente un attaccante potrebbe indovinarla con dizionari, regole e modelli probabilistici.

In pratica, il secondo punto è quello più vicino alla sicurezza reale: password dall'aspetto "complesso" possono restare facilmente attaccabili se costruite con parole comuni e trasformazioni prevedibili (es. `Password123!`). Episodi di forte risonanza mediatica mostrano come credenziali banali (o esposte) possano compromettere anche sistemi critici: nelle ricostruzioni post-rapina al Louvre, la password del server di videosorveglianza risultava essere semplicemente `LOUVRE` [5, 4]; in ambito IoT, Mirai ha infettato centinaia di migliaia di dispositivi sfruttando combinazioni di credenziali predefinite molto note [2, 1]; in un caso molto discusso, l'account Twitter di Donald Trump è stato violato (secondo le autorità olandesi) indovinando una password estremamente prevedibile (`maga2020!`) [3].

2.2 Linee guida e raccomandazioni

Le linee guida moderne (NIST) suggeriscono di:

- favorire password/passphrase più lunghe;
- effettuare controlli su password comuni/compromesse;
- limitare regole di composizione eccessive che peggiorano l'esperienza utente senza benefici proporzionati.

OWASP fornisce indicazioni operative su policy, memorizzazione sicura e controlli comuni.

2.3 Password meters

I password meters hanno lo scopo di fornire un feedback immediato e di influenzare positivamente la scelta dell'utente. Diversi studi mostrano che il feedback può migliorare la complessità delle password, ma la qualità dipende molto dal *tipo* di feedback: indicatori troppo generici o incoerenti possono creare un falso senso di sicurezza [7]. Studi più recenti evidenziano inoltre che meter diversi possono produrre stime significativamente differenti a parità di password, rendendo importante documentare e validare le scelte di scoring [8].

2.4 zxcvbn come baseline di confronto

`zxcvbn` (Dropbox) è un estimatore di forza "low-budget" che combina pattern matching e liste di termini comuni per stimare il numero di tentativi necessari a indovinare una password [6]. Nel progetto PSM viene usato come baseline perché:

- è ampiamente conosciuto e riutilizzato;
- integra regole per parole comuni e trasformazioni tipiche;
- restituisce uno score discreto facilmente normalizzabile su 0–100.

2.5 Posizionamento del progetto

PSM si colloca in questo contesto proponendo:

- una valutazione in tempo reale con scala 0–100;

- una validazione finale che blocca password *formalmente* buone ma *praticamente* rischiose (in particolare per presenza di dati personali);
- una pipeline sperimentale riproducibile con export e dashboard, così da rendere verificabili le scelte progettuali.

3 Modello di processo adottato e valutazioni iniziali

3.1 Modello di processo

Il lavoro è stato organizzato con un approccio **iterativo-incrementale**: ad ogni iterazione sono stati prodotti artefatti verificabili (UML, componenti software, test, risultati sperimentali), integrati progressivamente fino ad ottenere una demo end-to-end. In particolare, le iterazioni hanno seguito una traccia coerente con i deliverable DS:

- **Analisi e requisiti**: definizione obiettivo del meter, vincoli e casi d'uso principali;
- **Progettazione**: documentazione UML e definizione dell'architettura modulare con engine SSOT;
- **Implementazione**: sviluppo incrementale di DS1–DS5 (UI, API, esperimenti, dashboard, export);
- **Verifica**: test automatici (unit/integration) e workflow CI per ridurre regressioni;
- **Validazione**: esecuzione di esperimenti e confronto con baseline (`zxcvbn`) per valutare l'efficacia.

3.2 Pianificazione (Gantt)

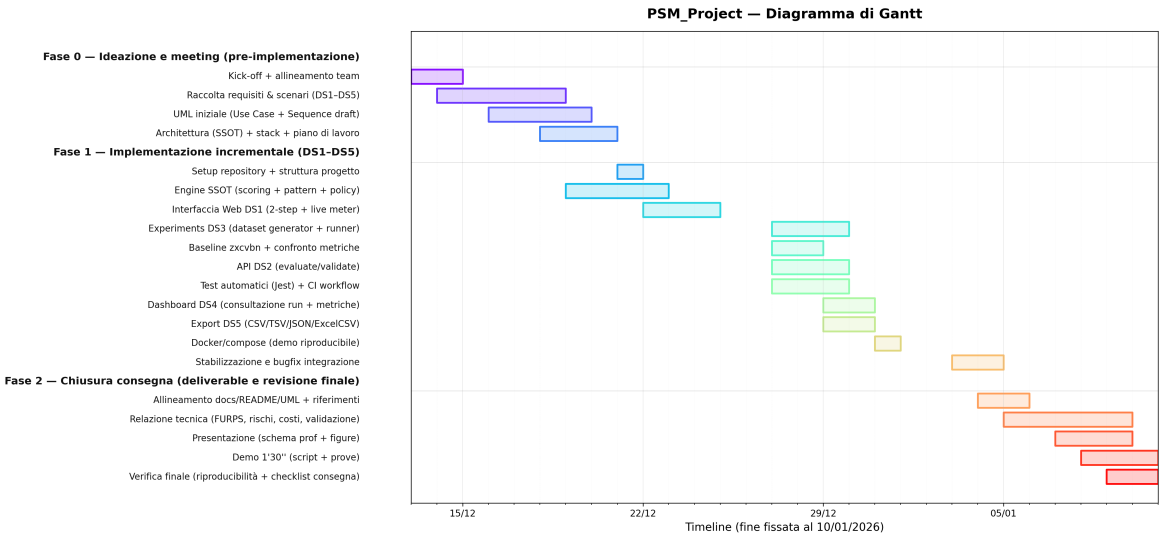


Figura 1: Gantt di progetto.

3.3 Analisi dei rischi (Risk Matrix 3x3)

Questa analisi è stata impostata come valutazione preventiva: probabilità e impatto sono qualitativi (bassa/media/alta) e riflettono l'esperienza del team sul tipo di progetto e sulle attività svolte. La matrice in Figura 2 sintetizza i rischi principali; la Tabella 1 ne riporta una descrizione e le mitigazioni adottate.

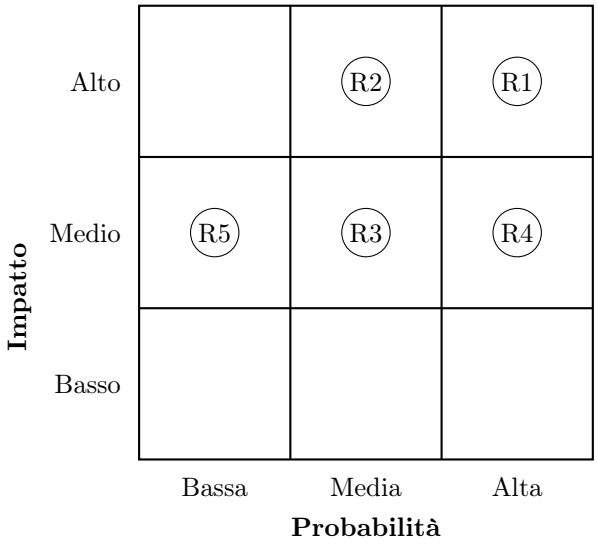


Figura 2: Risk matrix 3x3.

ID	Prob.	Imp.	Descrizione e mitigazione
R1	Alta	Alta	Dataset sperimentale non rappresentativo → generazione controllata, analisi per categoria, dashboard ed export risultati.
R2	Media	Alta	Regole dell'engine troppo permissive/severe → confronto con baseline + test automatici su casi limite.
R3	Media	Media	Integrazione UI/API/engine → SSOT nell'engine + validazioni input in API.
R4	Alta	Media	Regressioni dopo modifiche → CI con suite test (Jest) e workflow di smoketest.
R5	Bassa	Media	Problemi di ambiente/deploy → Docker compose e documentazione di esecuzione.

Tabella 1: Rischi principali e mitigazioni adottate.

3.4 Stima dei costi

La stima dei costi è stata effettuata come equivalente industriale di un progetto di dimensioni contenute, usando la formula:

$$Cost = \sum_{i} (ore_i) \times (tariffa\ oraria) \tag{1}$$

Si assume una tariffa oraria indicativa pari a 9€/h (valore convenzionale per attività didattiche), al solo scopo di quantificare lo sforzo profuso. Le attività sono state ripartite riflettendo l'impegno effettivo di sviluppo e documentazione. Con tre componenti, il carico medio pro capite risulta di circa 47 ore.

Attività (coerente con Gantt)	**Ore**	**Costo (€)**	**Output/note**
Kick-off e allineamento team	4	9	Definizione obiettivi.
Raccolta requisiti e scenari	6	54	Analisi vincoli e criteri.
UML iniziale (Use Case + Sequence)	8	72	Documentazione di base.
Architettura (SSOT) + Stack	8	72	Scelte tecnologiche.
Setup repository e struttura	3	27	Ambiente di lavoro.
Engine SSOT (core logic)	12	108	Sviluppo algoritmi scoring.
Interfaccia Web DS1	10	90	UI e feedback real-time.
Experiments Runner DS3	10	90	Pipeline sperimentale.
Baseline zxcvbn + metriche	4	36	Integrazione confronto.
API REST DS2	8	72	Sviluppo endpoint.
Test automatici (Jest) + CI	10	90	Suite di test e workflow.
Dashboard DS4	8	72	Visualizzazione run.
Export DS5 (CSV/JSON/Excel)	6	54	Formati di output.
Docker e containerizzazione	4	36	Demo riproducibile.
Stabilizzazione e bugfix	8	72	Raffinamento codice.
Allineamento README e UML	6	54	Rifinitura documenti.
Relazione tecnica	12	108	Stesura finale.
Presentazione (slide e figure)	6	54	Materiale espositivo.
Demo 1'30" (script e prove)	1	36	Preparazione video/live.
Verifica finale e checklist	7	63	Revisione pre-consegna.
Totale	**141**	**1269**	**Person-hours complessive**

Tabella 2: Stima dei costi aggiornata (tariffa 9€/h).

Con tre componenti di gruppo, il carico medio risulta pari a circa **47 ore/persona**, distribuite lungo le iterazioni di progetto.

4 Metodo proposto e requisiti

4.1 Metodo

Il progetto adotta un approccio a **motore centralizzato** (engine SSOT) che separa chiaramente:

- **Logica di valutazione**: calcolo punteggio, rilevamento pattern, generazione feedback, validazione finale;
- **Interfacce**: UI Web e API REST, che consumano l'engine senza duplicare regole;
- **Validazione sperimentale**: runner che confronta l'engine con una baseline e produce risultati persistiti.

Questa scelta permette di mantenere coerenza tra ciò che viene mostrato all'utente (DS1), ciò che viene esposto via servizio (DS2) e ciò che viene misurato negli esperimenti (DS3–DS5).

4.2 Requisiti funzionali

I requisiti funzionali principali (RF) sono:

- **RF1** – Valutare una password in tempo reale restituendo un punteggio 0–100 e un livello qualitativo.
- **RF2** – Identificare pattern deboli (sequenze, ripetizioni, parole comuni, anni/date, pop-culture, insiemi piccoli) e produrre un elenco strutturato di indicatori.
- **RF3** – Generare suggerimenti comprensibili per migliorare la password (feedback).
- **RF4** – Applicare una *validazione finale* che blocchi password non conformi (vincoli minimi e dati personali, anche in leet).
- **RF5** – Fornire un'interfaccia Web (registrazione a 2 step) che integri valutazione, feedback e conferma password.
- **RF6** – Esporre le funzionalità tramite API REST (`/api/evaluate`, `/api/validate`).
- **RF7** – Eseguire esperimenti su dataset e produrre output (JSON/CSV/TSV/ExcelCSV) confrontando PSM con una baseline.
- **RF8** – Fornire una dashboard per consultare run sperimentali, statistiche e download degli export.

4.3 Requisiti non funzionali (FURPS+)

Per i requisiti non funzionali è stato adottato il modello **FURPS+**: oltre alle cinque categorie classiche (Functionality, Usability, Reliability, Performance, Supportability), la componente "+" raccoglie *vincoli* e requisiti trasversali (implementazione, interfacce e operatività). La Tabella 3 collega gli aspetti FURPS+ a scelte implementative e a criteri di accettazione verificabili.

Categoria	Requisito (RNF)	Evidenza / criterio di accettazione
F	Coerenza della valutazione (SSOT).	Stessa logica riusata da UI/API/esperimenti tramite `src/engine/psmEngine.js`.
U	Feedback immediato e comprensibile.	UI a 2 step con punteggio, barra, suggerimenti e messaggi di validazione (Figg. 10–13).
R	Affidabilità e prevenzione regressioni.	Test automatici (Jest) + workflow CI che esegue test e smoketest su endpoint principali.
P	Reattività su input realistici e robustezza a input anomali.	Valutazione eseguita localmente in JS; lato API limiti su lunghezza password e token per evitare payload eccessivi.
S	Manutenibilità e riproducibilità.	Architettura modulare, Docker compose per demo ripetibile, export risultati e dashboard per ispezione.
+	Vincoli di progetto e interoperabilità.	Engine riusabile sia in browser sia in Node (SSOT); interfacce HTTP/JSON (API REST) con CORS; pipeline sperimentale deterministica (seed) e artefatti esportabili.

Tabella 3: Requisiti non funzionali secondo FURPS+: collegamento tra obiettivi e verifiche.

4.4 Use case

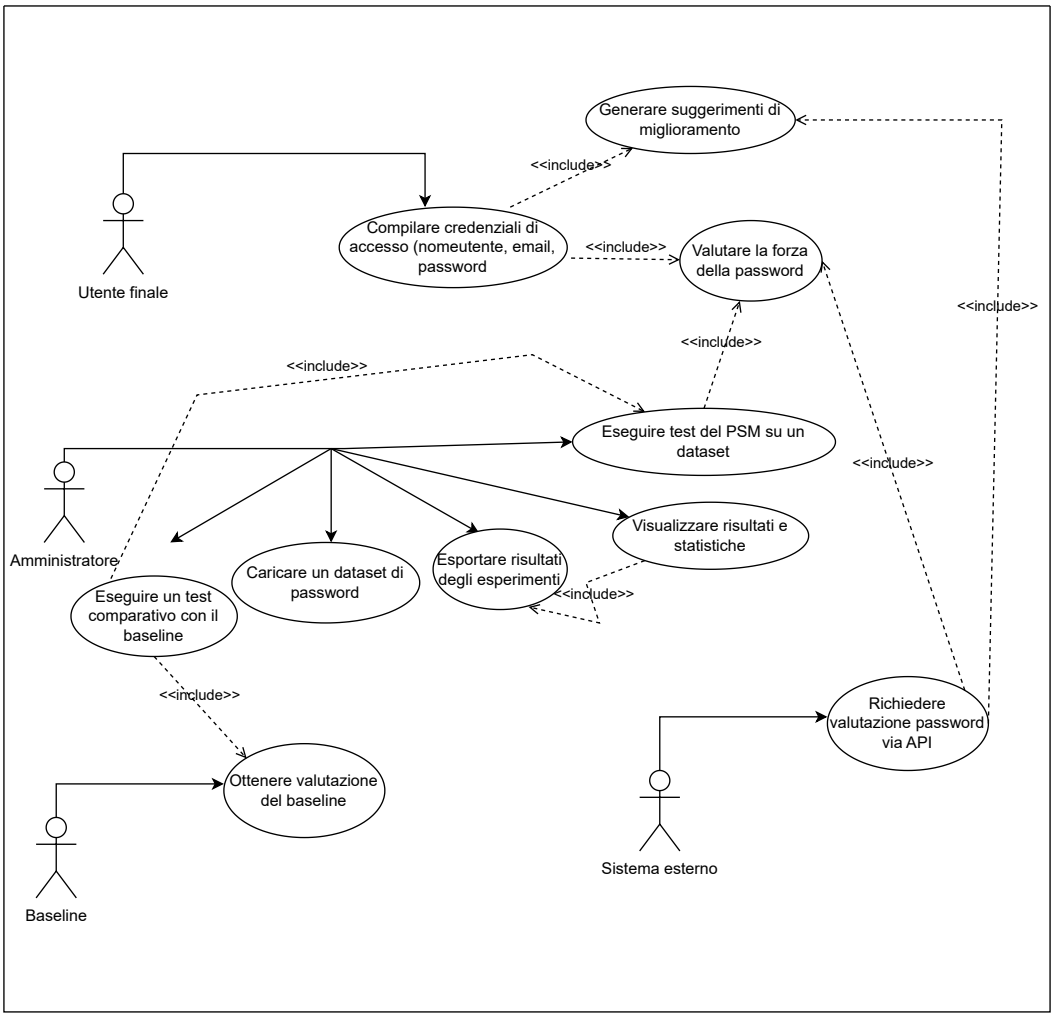


Figura 3: Diagramma dei casi d'uso.

5 Architettura e tech stack

5.1 Panoramica architetturale

L'architettura è stata progettata per isolare il **cuore del progetto** (engine) dalle interfacce e dagli strumenti di validazione, secondo il principio *Single Source of Truth:* tutte le regole di scoring, pattern detection, suggerimenti e validazione risiedono in un unico modulo condiviso.

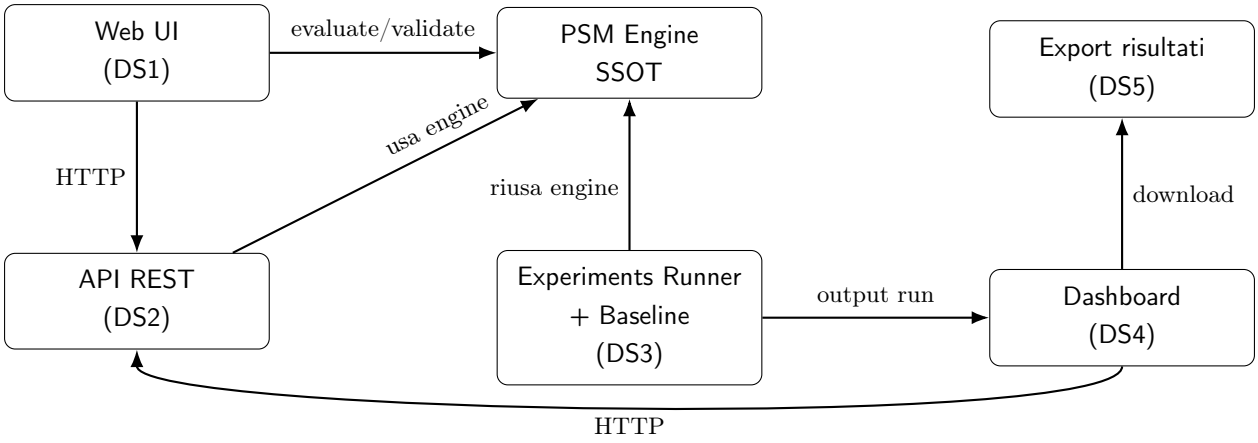


Figura 4: Architettura a componenti: l'engine è SSOT e viene riusato da UI, API ed esperimenti.

5.2 Componenti principali

- **Engine (SSOT)**: `src/engine/psmEngine.js`. Implementa scoring 0–100, rilevamento pattern, generazione suggerimenti e validazione finale.
- **UI Web (DS1)**: `src/web/`. Interfaccia a 2 step con valutazione live e messaggi di validazione.
- **API REST (DS2)**: `src/api/server.js`. Espone endpoint per valutazione/validazione e per consultare esportare run sperimentali.
- **Esperimenti (DS3)**: `src/experiments/`. Runner su dataset con baseline `zxcvbn`, produzione output e metriche.
- **Dashboard (DS4)**: `src/web/experiments.html` + `experiments.js`. Consultazione run e statistiche.
- **Export (DS5)**: endpoint API che serve risultati in diversi formati (JSON/CSV/TSV/ExcelCSV).

5.3 Tech stack

- **Front-end**: HTML/CSS/JavaScript (vanilla) per massima portabilità e rapidità di prototipazione.
- **Back-end**: Node.js + Express per API REST e gestione file run sperimentali.
- **Testing**: Jest per unit test e test di integrazione.

- **CI**: GitHub Actions (workflow di test, esperimenti e smoketest).
- **Containerizzazione**: Docker e `docker-compose` per una demo riproducibile (servizi Web + API).

5.4 Tracciamento DIAGRAMMI DELLE SEQUENZE (documentazione)

Per completezza, si riportano le schede DS prodotte (riassunti visivi della consegna incrementale).

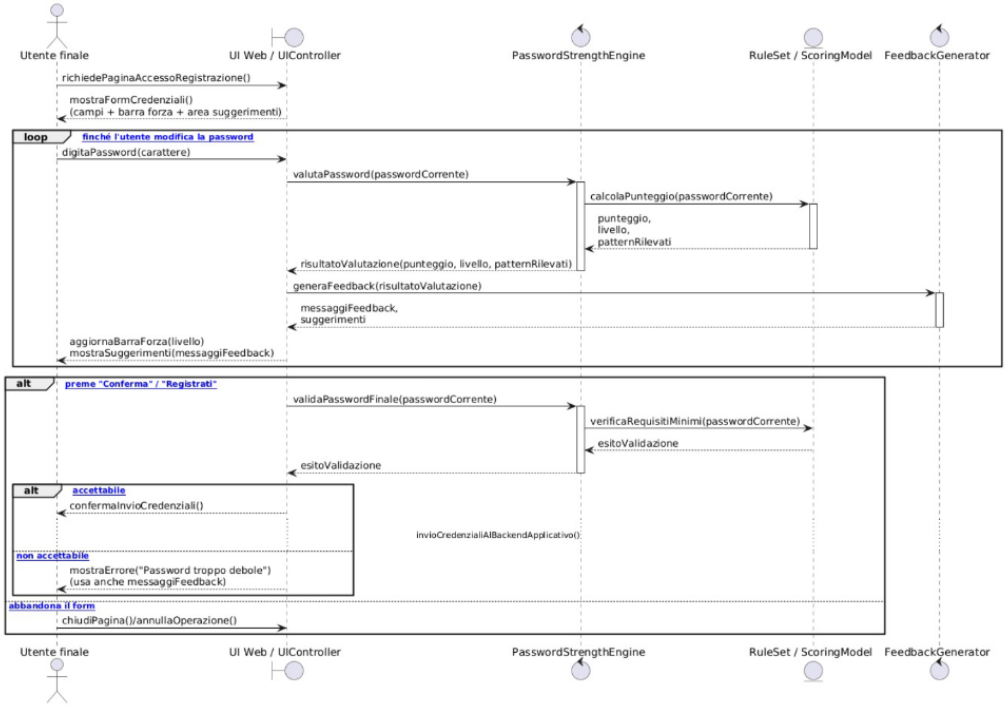


Figura 5: Documentazione DS1 (UI).

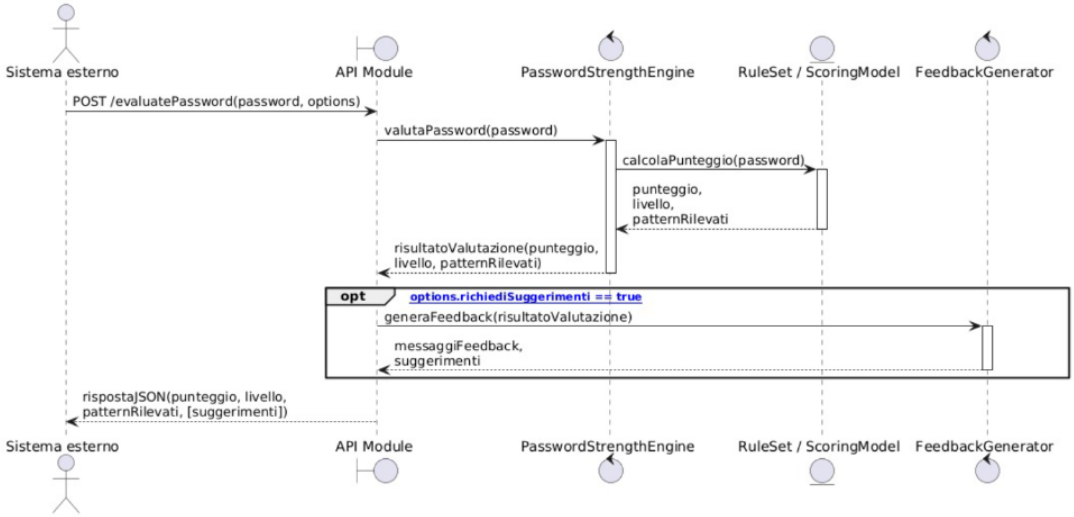


Figura 6: Documentazione DS2 (API).

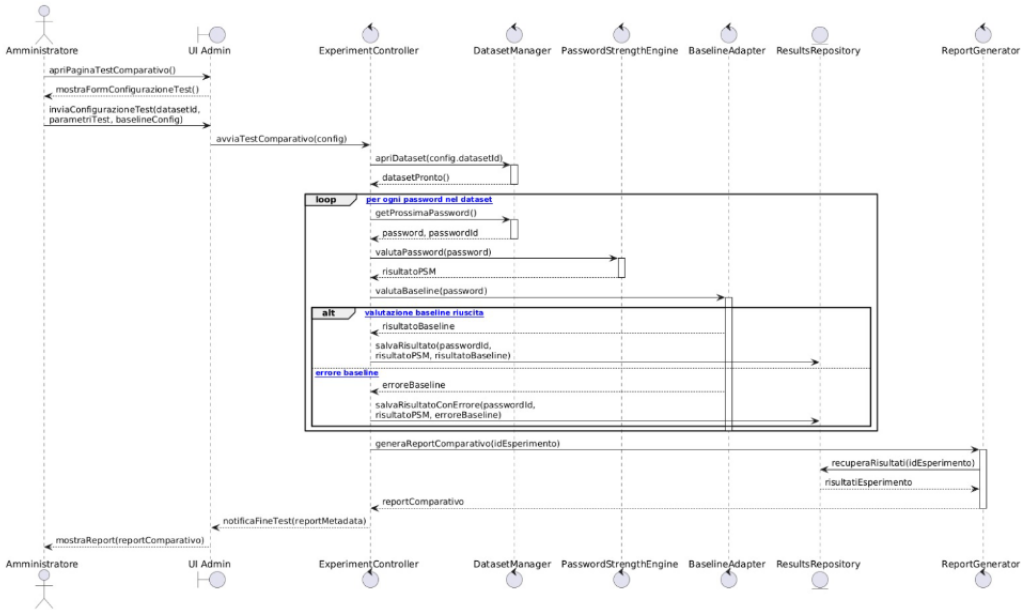


Figura 7: Documentazione DS3 (Esperimenti).

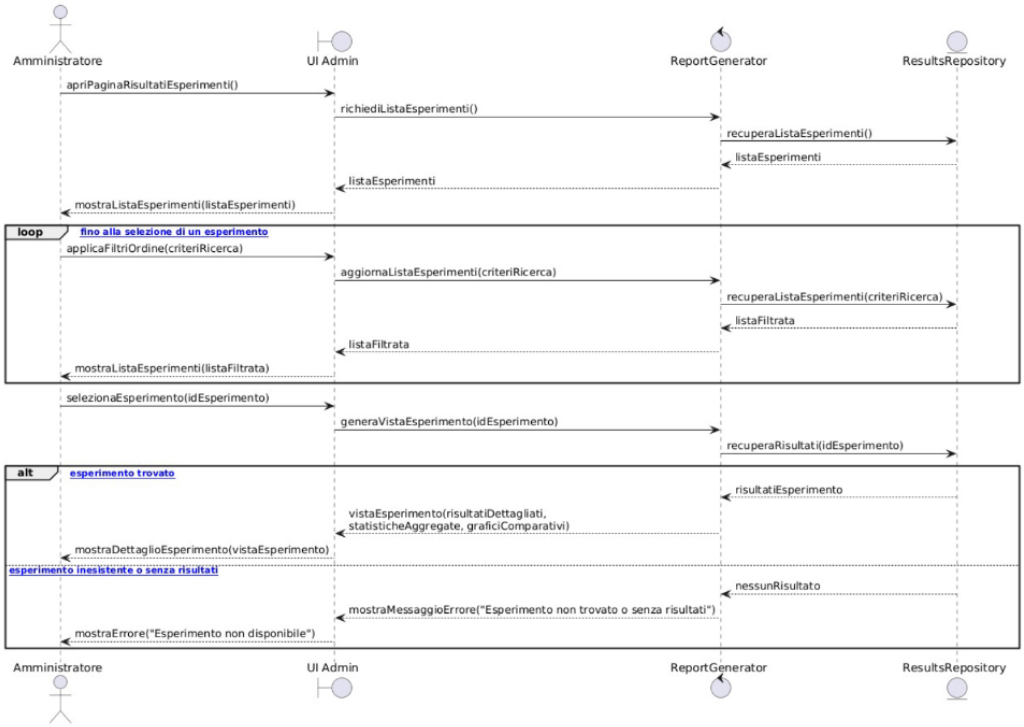


Figura 8: Documentazione DS4 (Dashboard).

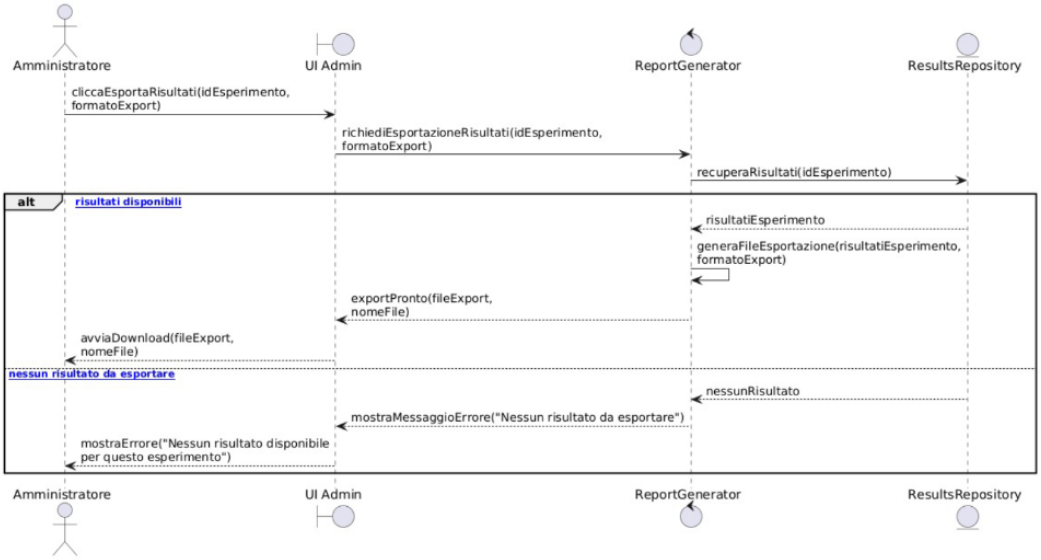


Figura 9: Documentazione DS5 (Export).

6 Prototipo

6.1 Flusso utente (registrazione a due step)

La UI guida l'utente in due step: inserimento credenziali e definizione password con valutazione in tempo reale. Nel primo step vengono raccolti *nome*, *cognome* ed *email*; tali dati vengono normalizzati e trasformati in `personalTokens` (es. parti dell'email e versioni normalizzate) che l'engine utilizza per individuare password riconducibili all'utente.

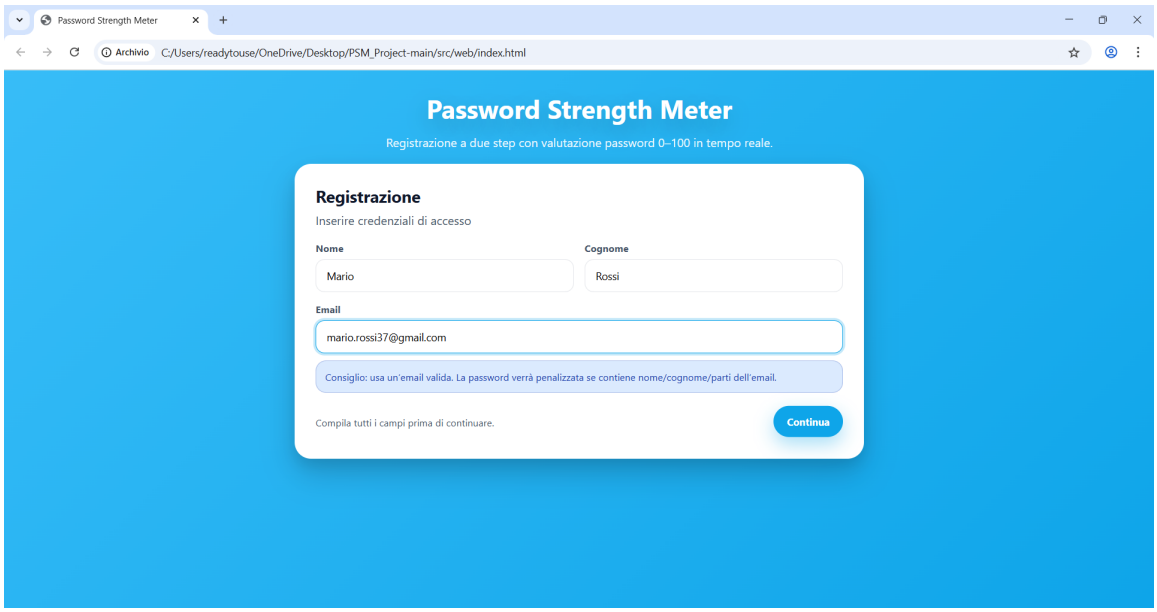


Figura 10: Step 1: inserimento credenziali (nome, cognome, email).

Nel secondo step l'utente inserisce la password: ad ogni modifica vengono aggiornati punteggio, livello e suggerimenti. La conferma password viene verificata in parallelo e il pulsante di

creazione account viene abilitato solo quando la validazione finale è soddisfatta.

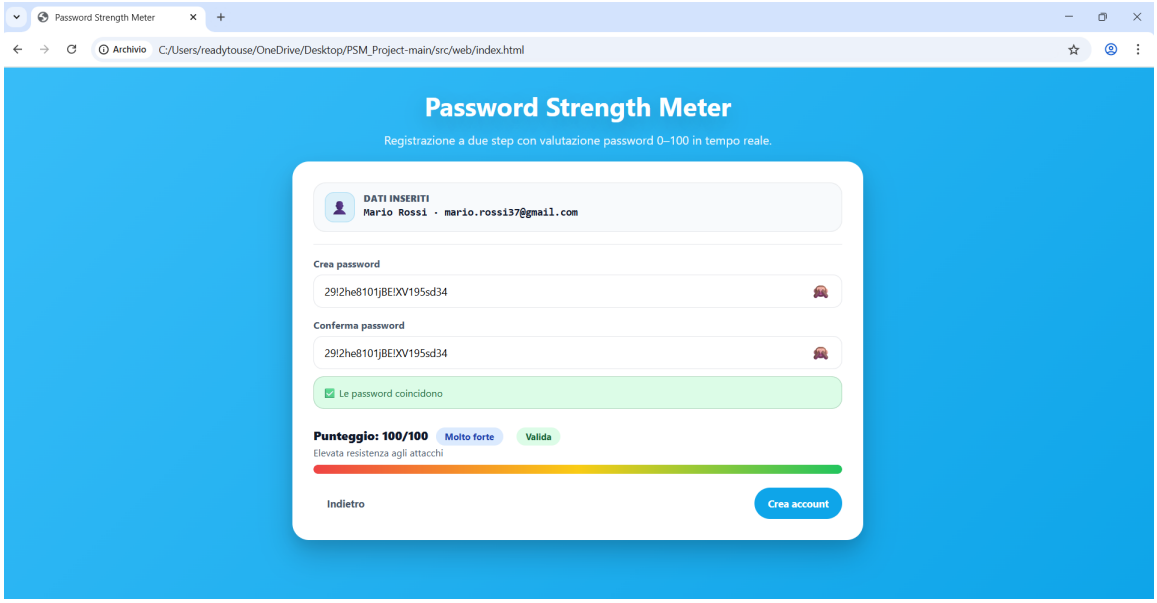


Figura 11: Step 2: esempio di password *validabile* con feedback e punteggio.

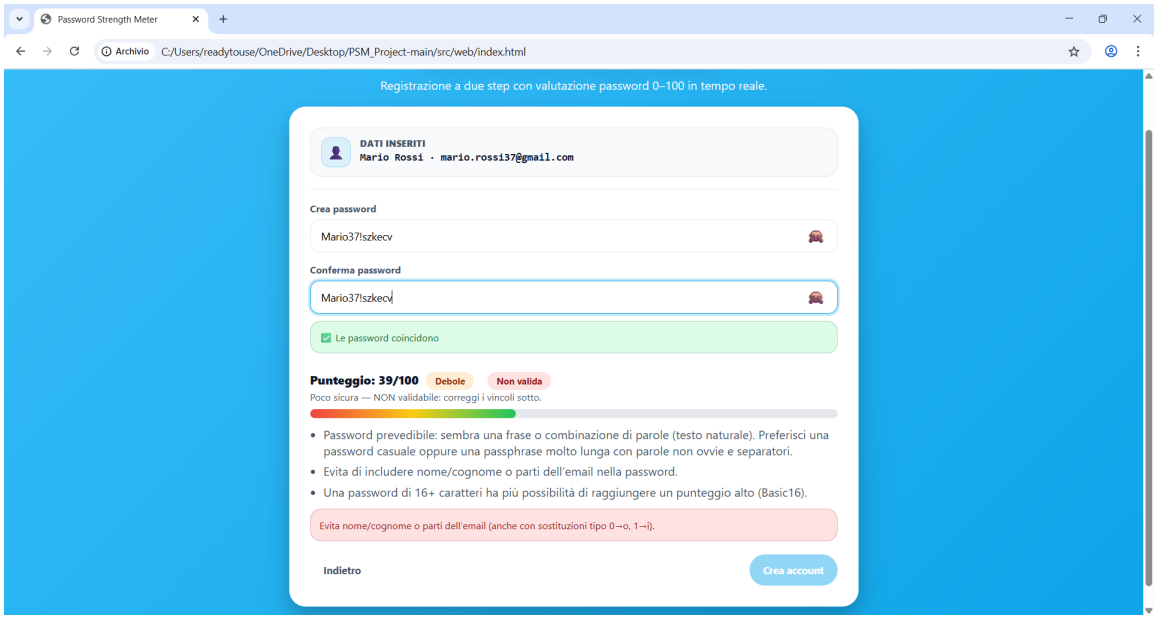


Figura 12: Step 2: esempio di password *non validabile* perché include dati personali (nome/e-mail).

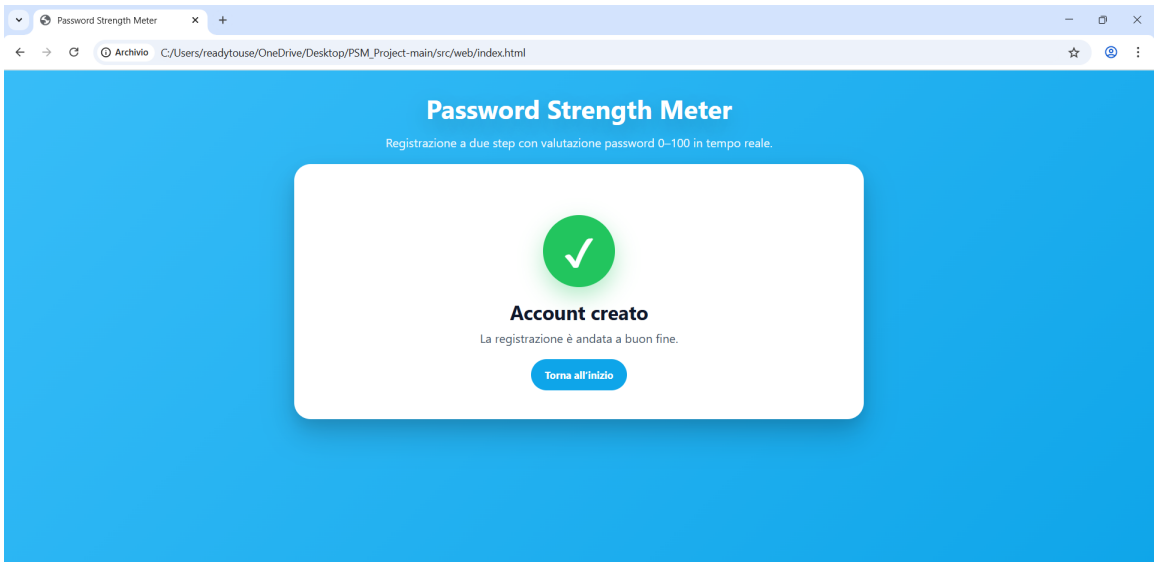


Figura 13: Completamento registrazione.

6.2 Distinzione tra forza stimata e validazione

L'engine restituisce due informazioni complementari:

- **Forza stimata** (`evaluate`): score e livello qualitativo servono a guidare l'utente, anche quando la password è ancora in fase di composizione.
- **Validazione** (`validateFinal`): applica i vincoli di policy (requisiti minimi e controllo su dati personali) e stabilisce se la password è accettabile per la creazione account.

La Figura 12 mostra un caso tipico: una password può raggiungere un punteggio discreto ("Discreta") ma essere comunque **non validabile** perché contiene token personali (anche con sostituzioni tipo `0`→`o`, `1`→`i`). Questa distinzione evita l'equivoco "punteggio sufficiente = password accettata" e rende il comportamento del prototipo coerente con l'obiettivo di sicurezza.

7 Validazione e verifica

7.1 Strategia di verifica

La verifica del sistema è stata condotta su tre livelli complementari:

- **Test automatici**: suite Jest per verificare engine e API (casi limite, validazione token personali, endpoint principali).
- **CI**: workflow GitHub Actions che esegue i test ad ogni push e include uno smoketest sugli endpoint di consultazione/ export delle run sperimentali.
- **Validazione sperimentale**: confronto tra PSM e baseline `zxcvbn` su dataset controllato, con risultati persistiti e consultabili.

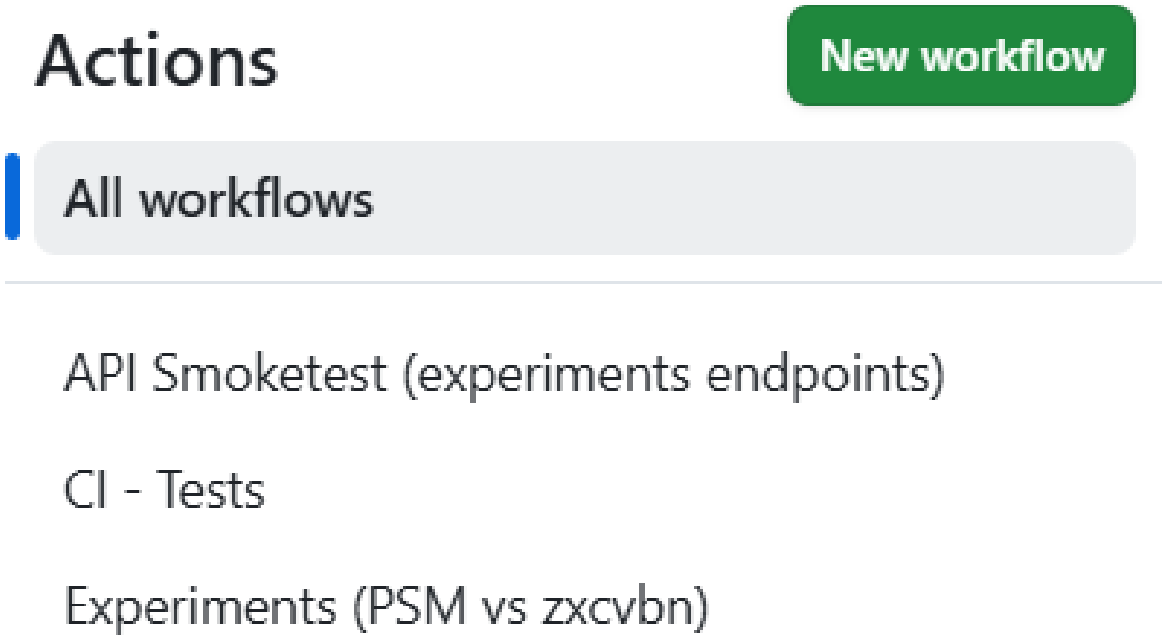


Figura 14: Workflow CI: esecuzione automatica dei test e degli smoketest.

7.2 Pipeline sperimentale

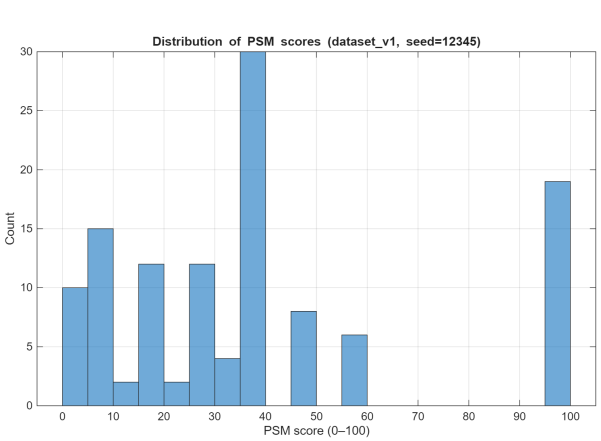
La pipeline sperimentale è stata progettata per essere riproducibile:

1. generazione (o caricamento) di un dataset di password etichettate per categoria;
2. valutazione con PSM e con baseline `zxcvbn`;
3. calcolo di statistiche e differenze (*delta*) tra i due punteggi;
4. salvataggio degli output su file in una cartella di run identificata da `runId`;
5. consultazione via dashboard ed export tramite API (CSV/TSV/JSON/ExcelCSV).

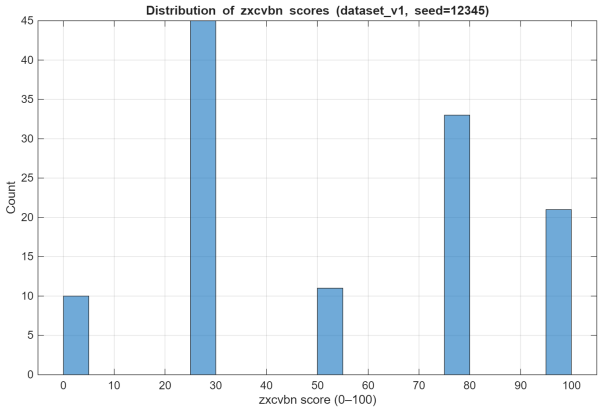
Il principale vantaggio di questa pipeline è la tracciabilità: ogni run conserva metadati e risultati, consentendo di replicare o confrontare diverse versioni dell'engine senza perdere lo storico.

7.3 Risultati (confronto PSM vs zxcvbn)

Le figure seguenti mostrano una sintesi dei risultati della valutazione sperimentale.



(a) Distribuzione punteggi PSM.



(b) Distribuzione punteggi zxcvbn (normalizzati 0–100).

Figura 15: Istogrammi dei punteggi sul dataset sperimentale.

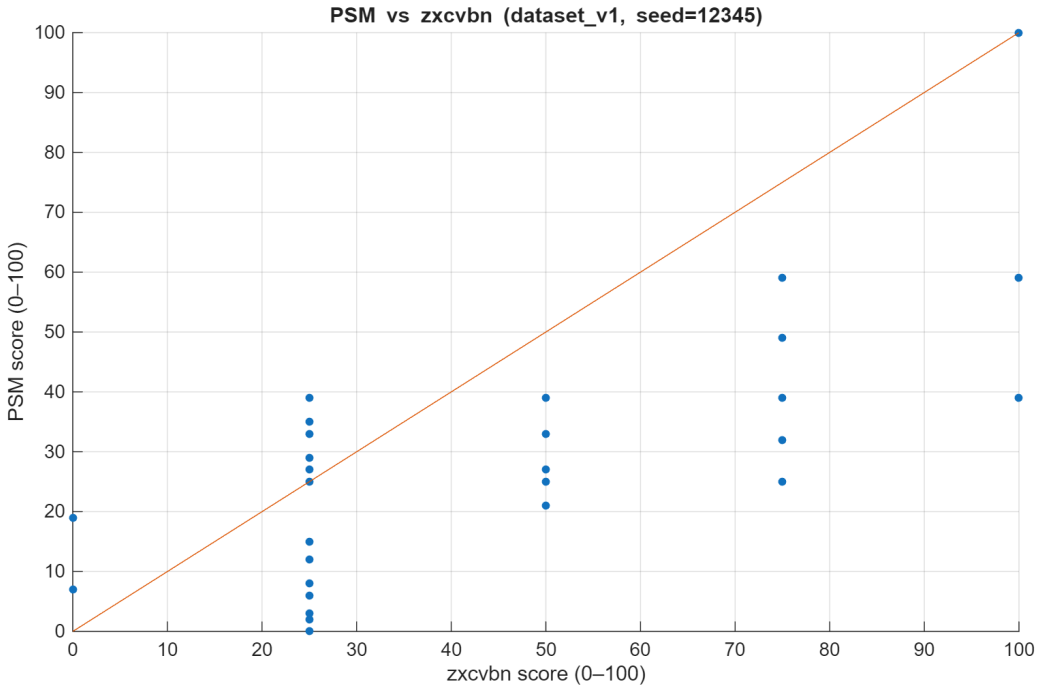


Figura 16: Scatter plot PSM vs zxcvbn: confronto diretto dei punteggi sui campioni del dataset.

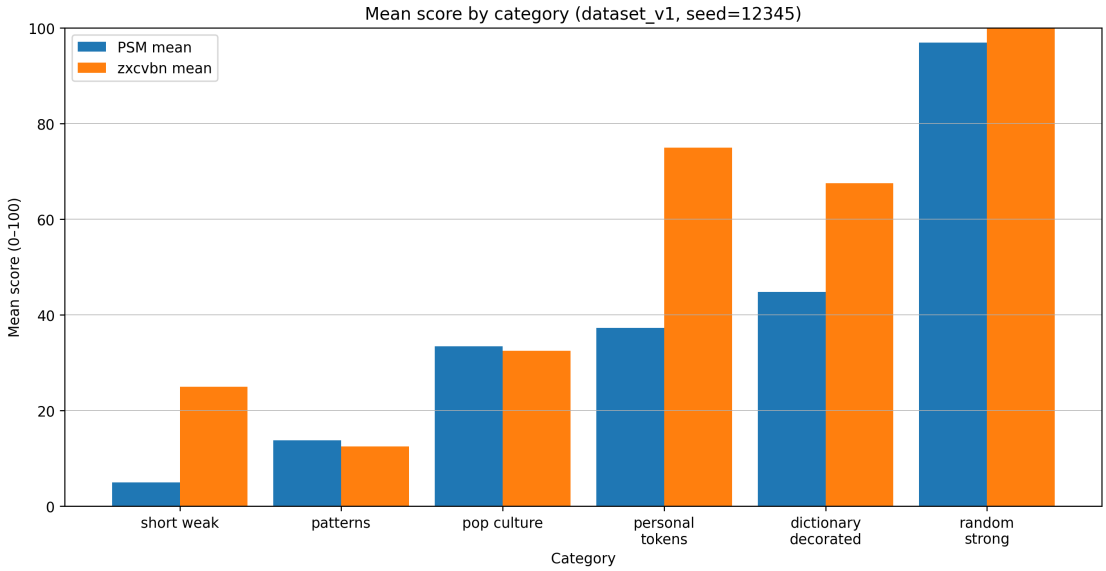


Figura 17: Media dei punteggi per categoria (PSM vs zxcvbn).

Dalla lettura combinata di Figura 16 e Figura 17 è possibile individuare categorie in cui PSM risulta più severo (es. password con pattern prevedibili o contenenti informazioni personali) e categorie in cui i due stimatori sono più allineati (es. password casuali lunghe).

7.4 Dashboard

La dashboard consente di esplorare una run sperimentale, filtrare/limitare il numero di record mostrati e scaricare gli export. Inoltre, gestisce scenari di errore di comunicazione con l'API rendendo evidente lo stato del sistema.

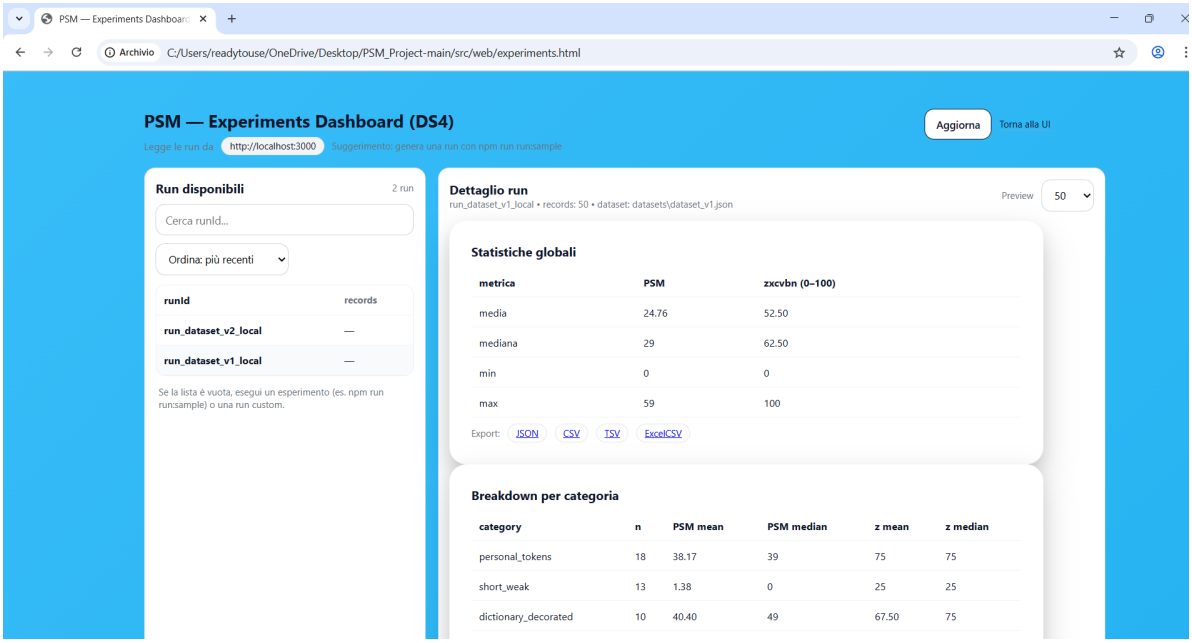


Figura 18: Dashboard: selezione ed esplorazione di una run sperimentale.

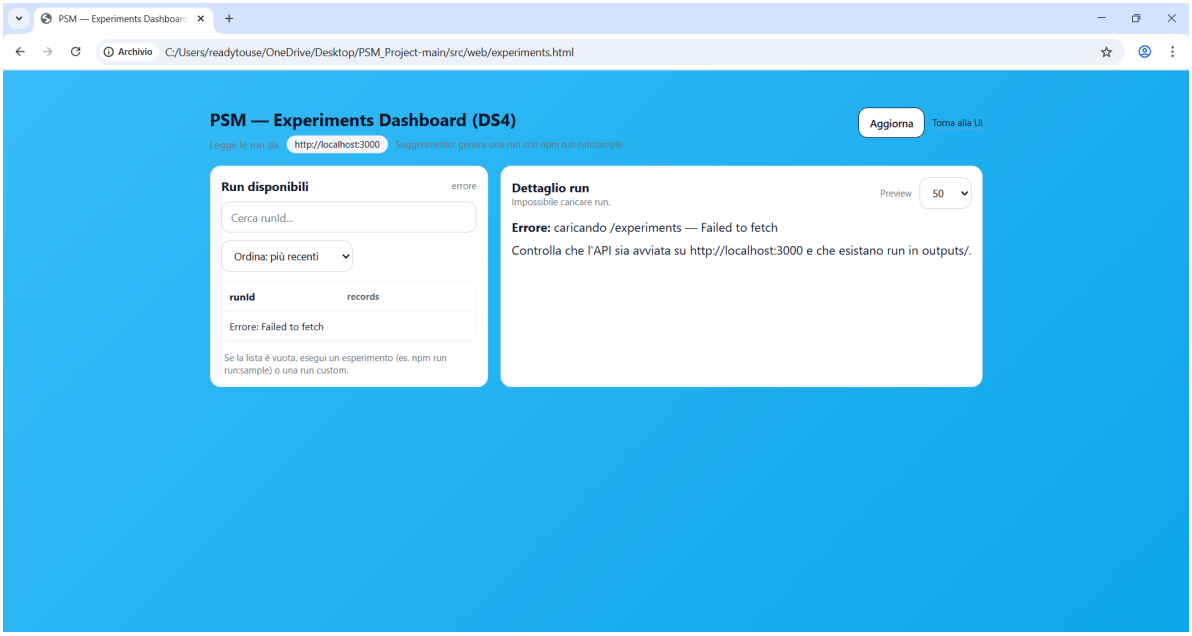


Figura 19: Dashboard: gestione errori di comunicazione con l'API.

8 Discussione

8.1 Punti di forza

- **Coerenza SSOT**: l'engine è l'unica sorgente di verità per scoring e validazione, riducendo duplicazioni e incoerenze tra UI, API ed esperimenti.
- **Feedback orientato all'utente**: la UI fornisce suggerimenti immediati e messaggi esplicativi, rendendo chiaro *perché* una password è debole o non validabile.
- **Validazione "sicurezza-first"**: la policy finale blocca password che includono informazioni personali, anche se ottengono un punteggio discreto durante la digitazione.
- **Verificabilità**: test automatici, CI e pipeline sperimentale con export e dashboard rendono il progetto ispezionabile e riproducibile.

8.2 Limiti e trade-off

- Lo scoring 0–100 è una sintesi: due password con punteggio simile possono avere rischi diversi in base al modello di attacco considerato.
- Le liste/dizionari incorporati nell'engine non possono essere esaustivi: un termine "comune" può cambiare nel tempo o variare per lingua/contesto.
- La baseline `zxcvbn` e PSM adottano assunzioni differenti; il confronto serve a evidenziare differenze qualitative, non a stabilire un "vincitore" assoluto.
- La validazione su dati personali migliora la sicurezza, ma può ridurre l'accettabilità per alcuni utenti; per questo è importante accompagnarla con suggerimenti chiari.

8.3 Minacce alla validità sperimentale

- **Validità interna**: il dataset potrebbe favorire alcune categorie; mitigazione: analisi per categoria (Figura 17) e run riproducibili.
- **Validità esterna**: risultati su dataset controllato potrebbero non generalizzare a popolazioni reali; mitigazione: ampliare dataset e includere password reali anonimizzate (ove possibile).
- **Validità costruttiva**: lo score 0–100 non coincide con il tempo reale di cracking; mitigazione: interpretare lo score come *indicatore* e non come garanzia.

9 Conclusioni e sviluppi futuri

9.1 Conclusioni

Il progetto PSM ha prodotto un prototipo completo che integra:

- una UI a due step con valutazione e feedback in tempo reale;
- un engine SSOT riutilizzato da tutti i componenti;
- un'API REST per valutazione/validazione e consultazione delle run;
- una pipeline sperimentale con confronto a baseline (`zxcvbn`), dashboard ed export.

L'elemento centrale è la distinzione tra **forza stimata** e **validazione** (Sez. 6): in questo modo il sistema evita che uno score discreto venga interpretato come "password accettata" quando la policy finale rileva condizioni non ammissibili (es. dati personali).

9.2 Sviluppi futuri

- Integrazione con un backend reale di registrazione/autenticazione (persistenza utenti, hashing sicuro, gestione sessioni).
- Estensione del controllo su password compromesse tramite consultazione di un servizio di breach-check (es. k-anonymity).
- Raffinamento del modello di scoring con metriche di guessability più vicine ai modelli di cracking moderni.
- Estensione della copertura linguistica dei dizionari e dei set "small" (lingue diverse e contesti specifici).
- Studio di usabilità con utenti (A/B test) per misurare l'impatto del feedback sulla qualità delle password create [8].

Riferimenti bibliografici e Link Utili

[1] Inside the infamous mirai iot botnet: a retrospective analysis. Cloudflare Blog, 2017. https://blog.cloudflare.com/inside-mirai-the-infamous-iot-botnet-a-retrospective-analysis/.

[2] Understanding the mirai botnet. Google Research (technical report / paper), 2017. https://research.google.com/pubs/archive/46301.pdf.

[3] Trump's twitter account was hacked, dutch ministry confirms. The Guardian, 2020. https://www.theguardian.com/us-news/2020/dec/16/trumps-twitter-account-was-hacked-dutch-ministry-affirms.

[4] Louvre heist reveals museum used 'louvre' as password for its video surveillance. Tom's Hardware, 2025. https://www.tomshardware.com/tech-industry/cyber-security/louvre-heist-reveals-glaring-security-weaknesses-previous-reports.

[5] Password to louvre's video surveillance system was 'louvre', according to employee. ABC News (International), 2025. https://abcnews.go.com/International/password-louvres-video-surveillance-system-louvre-employee/story?id=127236297.

[6] Dropbox. zxcvbn: Low-budget password strength estimation. https://github.com/dropbox/zxcvbn, 2012.

[7] Blase Ur, Patrick Gage Kelley, Saranga Komanduri, Joel Lee, Michelle Maass, Michelle L. Mazurek, Timothy Passaro, Richard Shay, Timothy Vidas, Lujo Bauer, Nicolas Christin, and Lorrie Faith Cranor. How does your password measure up? the effect of strength meters on password creation. In *Proceedings of the 21st USENIX Security Symposium*, 2012.

[8] Ding Wang, Xuan Shan, Qiying Dong, Yaosheng Shen, and Chunfu Jia. No single silver bullet: Measuring the accuracy of password strength meters. In *USENIX Security Symposium*, 2023.