🔐 Secure Password Generator API

Un'API REST completa per la generazione di password sicure con multiple opzioni di configurazione, controllo di sicurezza e conformità agli standard NIST/OWASP.

✨ Caratteristiche

- **Generazione password sicure** con configurazione personalizzabile
- **Multipli tipi di password**: tradizionali, leggibili, pronunciabili, passphrase
- **Controllo password compromesse** tramite HaveIBeenPwned API
- **Calcolo entropia** e valutazione forza password
- **Standard di sicurezza** NIST e OWASP
- **Rate limiting** per proteggere le API esterne
- **Generazione bulk** per multiple password
- **Esclusione caratteri ambigui** opzionale
- **Documentazione interattiva** con Swagger UI

🚀 Installazione Rapida

Requisiti

- Python 3.8+
- pip

Setup

bash

```
# Clona o scarica il progetto
git clone <your-repo-url>
cd secure-password-generator

# Crea ambiente virtuale
python -m venv venv

# Attiva ambiente virtuale
# Linux/Mac:
source venv/bin/activate
# Windows:
venv\Scripts\activate

# Installa dipendenze
pip install -r requirements.txt
```

Avvio

```bash
# Avvio semplice
python main.py

# Avvio con uvicorn (consigliato)
uvicorn main:app --host 0.0.0.0 --port 8001 --reload
```

L'API sarà disponibile su `http://localhost:8001`

📋 Dipendenze

```txt
fastapi==0.104.1
uvicorn==0.24.0
pydantic==2.5.0
requests==2.31.0
```

🔧 Utilizzo

Interfaccia Web

- **Swagger UI**: `http://localhost:8001/docs`
- **ReDoc**: `http://localhost:8001/redoc`

Endpoints Principali

1. Generazione Password Base

```bash
POST /api/generate
```

Parametri:

- `length`: Lunghezza password (8-128, default: 16)
- `include_uppercase`: Includere maiuscole (default: true)
- `include_lowercase`: Includere minuscole (default: true)
- `include_numbers`: Includere numeri (default: true)
- `include_symbols`: Includere simboli (default: true)
- `exclude_ambiguous`: Escludere caratteri ambigui (default: true)

- `security_standard`: Standard di sicurezza "NIST" o "OWASP" (default: "NIST")
- `check_compromised`: Verificare se compromessa (default: false)

Esempio:

bash

```
curl -X POST http://localhost:8001/api/generate \
  -H "Content-Type: application/json" \
  -d '{
    "length": 16,
    "include_uppercase": true,
    "include_lowercase": true,
    "include_numbers": true,
    "include_symbols": true,
    "exclude_ambiguous": true,
    "security_standard": "NIST",
    "check_compromised": false
  }'
```

Risposta:

json


```
{
  "password": "x7$nK2mP9@vL4sR",
  "length": 16,
  "entropy_bits": 95.27,
  "strength": "very_strong",
  "charset_size": 89,
  "is_compromised": false,
  "compromise_count": null,
  "security_standard": "NIST"
}
```


2. Generazione Password Multiple

bash

```
POST /api/generate/bulk
```

Parametri aggiuntivi:

- `count`: Numero di password da generare (1-100, default: 1)

Esempio:

bash

```
curl -X POST http://localhost:8001/api/generate/bulk \
  -H "Content-Type: application/json" \
  -d '{
    "count": 5,
    "length": 12,
    "include_symbols": false
  }'
```

3. Password Leggibili (Diceware-style)

bash

```
POST /api/generate/readable
```

Parametri:

- `word_count`: Numero di parole (2-8, default: 4)
- `separator`: Separatore tra parole (default: "-")
- `include_numbers`: Aggiungere numeri (default: true)
- `capitalize`: Capitalizzare parole (default: true)

Esempio:

bash

```
curl -X POST http://localhost:8001/api/generate/readable \
  -H "Content-Type: application/json" \
  -d '{
    "word_count": 4,
    "separator": "-",
    "include_numbers": true,
    "capitalize": true
  }'
```

Risposta:

json

```
{
  "password": "Correct-Horse-Battery-Staple42",
  "length": 32,
  "word_count": 4,
  "entropy_bits": 51.7,
  "strength": "very_strong",
  "is_compromised": false,
  "compromise_count": null,
  "type": "readable"
}
```


4. Password Pronunciabili

```
bash

POST /api/generate/pronounceable
```

Genera password con pattern consonante-vocale per facilità di pronuncia.

5. Passphrase per MFA/SSH

```
bash

POST /api/generate/passphrase
```

Parametri:

- `length`: Lunghezza passphrase (16-128, default: 32)
- `include_spaces`: Includere spazi (default: true)

6. Controllo Password Compromesse

```
bash

POST /api/check-compromised
```

Esempio:

```
bash

curl -X POST http://localhost:8001/api/check-compromised \
  -H "Content-Type: application/json" \
  -d '{"password": "password123"}'
```

7. Health Check

bash

```
GET /api/health
```

🧪 Testing

Script di Test Automatico

Linux/Mac:

bash

```
chmod +x test_api.sh
./test_api.sh
```

Windows:

cmd

```
# Con Git Bash
./test_api.sh

# Con curl manuale
curl http://localhost:8001/api/health
```

Opzioni di Test

bash

```
# Tutti i test
./test_api.sh

# Test con output dettagliato
./test_api.sh -v

# Test specifici
./test_api.sh basic readable passphrase

# Help
./test_api.sh -h
```

Test Manuali

Test di base:

bash

```
curl -X POST http://localhost:8001/api/generate \
  -H "Content-Type: application/json" \
  -d '{"length": 12, "include_symbols": true}'
```

Test password leggibile:

```
bash

curl -X POST http://localhost:8001/api/generate/readable \
  -H "Content-Type: application/json" \
  -d '{"word_count": 3, "separator": "_"}'
```

Test controllo compromissione:

```
bash

curl -X POST http://localhost:8001/api/check-compromised \
  -H "Content-Type: application/json" \
  -d '{"password": "123456"}'
```

🔒 Sicurezza

Caratteristiche di Sicurezza

- **Generazione crittograficamente sicura** con modulo `secrets`
- **Rate limiting** per API HaveIBeenPwned (1 richiesta/1.5s)
- **Controllo password compromesse** tramite hash SHA-1 parziale
- **Esclusione caratteri ambigui** (1, l, 0, O, i, I)
- **Standard conformi** NIST SP 800-63B e OWASP

Valutazione Forza Password

- **Weak**: < 40 bit entropia
- **Medium**: 40-60 bit entropia
- **Strong**: 60-80 bit entropia
- **Very Strong**: > 80 bit entropia

Character Sets

- **Minuscole**: a-z (26 caratteri)
- **Maiuscole**: A-Z (26 caratteri)
- **Numeri**: 0-9 (10 caratteri)
- **Simboli NIST**: !@#$%^&*()_+-=[]{}|;:,.<>? (32 caratteri)

- **Simboli OWASP**: !@#$%^&* (8 caratteri)

📊 Esempi di Risposta

Password Tradizionale

json


```
{
  "password": "K8#mN2pQ@vL4sR9x",
  "length": 16,
  "entropy_bits": 95.27,
  "strength": "very_strong",
  "charset_size": 89,
  "is_compromised": false,
  "compromise_count": null,
  "security_standard": "NIST"
}
```


Password Leggibile

json


```
{
  "password": "Magic-Wonder-Dream-Smile84",
  "length": 26,
  "word_count": 4,
  "entropy_bits": 51.7,
  "strength": "very_strong",
  "is_compromised": false,
  "compromise_count": null,
  "type": "readable"
}
```


Password Compromessa

json


```
{
  "is_compromised": true,
  "compromise_count": 3861493,
  "checked_at": "2024-01-12T15:30:45.123456"
}
```


🛠️ Configurazione

Variabili d'Ambiente

```bash
# Opzionali
export API_HOST=0.0.0.0
export API_PORT=8001
export DEBUG=false
```

Personalizzazione

- **Lista parole**: Modifica `COMMON_WORDS` in `main.py`
- **Character set**: Personalizza `SYMBOLS`, `AMBIGUOUS_CHARS`
- **Rate limiting**: Configura `RateLimiter` per HaveIBeenPwned

🔧 Risoluzione Problemi

Errori Comuni

API non raggiungibile:

```bash
# Verifica processo
ps aux | grep uvicorn

# Verifica porta
netstat -tuln | grep 8001
```

Errore dipendenze:

```bash
pip install --upgrade pip
pip install -r requirements.txt
```

Timeout HaveIBeenPwned:

- Il servizio include rate limiting automatico
- Timeout di 5 secondi per richieste esterne
- Fallback sicuro in caso di errore

Log e Debug

```bash
# Avvio con log dettagliati
uvicorn main:app --host 0.0.0.0 --port 8001 --log-level debug
```

📈 Performance

Benchmark Tipici

- **Generazione password**: ~0.1ms
- **Calcolo entropia**: ~0.01ms
- **Controllo HaveIBeenPwned**: ~200-500ms (limitato da rate limiting)
- **Generazione bulk (10 password)**: ~1ms

Limitazioni

- **Rate limiting HaveIBeenPwned**: 1 richiesta/1.5s
- **Generazione bulk**: Max 100 password per richiesta
- **Lunghezza password**: 8-128 caratteri

🤝 Contribuire

1. Fork del repository
2. Crea feature branch (`git checkout -b feature/amazing-feature`)
3. Commit delle modifiche (`git commit -m 'Add amazing feature'`)
4. Push al branch (`git push origin feature/amazing-feature`)
5. Apri una Pull Request

📄 Licenza

Distribuito sotto licenza MIT. Vedi `LICENSE` per maggiori informazioni.

🔗 Collegamenti Utili

- NIST SP 800-63B
- OWASP Authentication Cheat Sheet
- HaveIBeenPwned API
- FastAPI Documentation

📞 Supporto

Per supporto, apri un issue su GitHub o contatta [tuo-email@example.com]

⭐ Se questo progetto ti è stato utile, considera di dargli una stella su GitHub!