**Zadanie 3**

Úvod do počítačovej bezpečnosti

1. **Kontrola zložitosti hesla pri registrácii**

Pri implementácii kontroly zložitosti hesla sme stanovili nasledujúce kritériá, aby sme zabezpečili, že používatelia používajú silné a bezpečné heslá:

* **Minimálna dĺžka hesla je 10 znakov**: Dlhšie heslá sú ťažšie uhádnuteľné a lepšie odolávajú útokom typu brute-force.
* **Obsah veľkých písmen**: Veľké písmená zvyšujú variabilitu hesla, čím pridávajú ďalšie vrstvy ochrany.
* **Obsah malých písmen**: Kombinácia malých a veľkých písmen zvyšuje bezpečnosť, pretože komplikuje jednoduché útoky na heslo.
* **Obsah aspoň jedného čísla**: Pridanie číslic zvyšuje zložitosť hesla, čo bráni jednoduchým útokom založeným na slovníkových heslách.
* **Obsah aspoň jedného špeciálneho znaku (napr. !@#$%^&\*)**: Špeciálne znaky ďalej zvyšujú variabilitu hesiel a znižujú šancu, že heslo bude ľahko uhádnuteľné.

Tieto kritériá zaručujú, že heslá používateľov nie sú len náhodné, ale aj dostatočne zložité na to, aby odolali bežným formám kybernetických útokov.

Vykonávanie validácie hesla na oboch úrovniach – frontendovej aj backendovej – je dôležité z viacerých dôvodov:

* **Frontendová validácia** poskytuje okamžitú spätnú väzbu používateľovi, čím zlepšuje používateľský zážitok a znižuje počet neúspešných pokusov o odoslanie formulára. Tým sa šetria zdroje servera, pretože zbytočné požiadavky s nesprávnym heslom sú zachytené už na strane klienta.
* **Backendová validácia** je nevyhnutná pre bezpečnosť. Keďže používateľ môže manipulovať s kódom na strane klienta (JavaScript), validácia na serveri zaisťuje, že všetky odoslané dáta budú dodatočne skontrolované a splnia stanovené bezpečnostné požiadavky. Tým sa predchádza obídeniu bezpečnostných pravidiel a neoprávnenému prístupu.

1. **Systém bezpečného ukladania hesiel**

Pri uložení používateľských hesiel do databázy je dôležité zabezpečiť, aby heslá neboli ukladané v čitateľnej podobe (plaintext). Namiesto toho sme implementovali systém, ktorý používa nasledujúce bezpečnostné postupy na ukladanie hesiel:

1. **Generovanie soli**: Pre každé heslo generujeme unikátnu **soľ** (salt) pomocou funkcie os.urandom(16). Soľ je náhodný reťazec, ktorý je pridaný k heslu pred jeho zahashovaním, čím sa zabezpečuje, že rovnaké heslá budú mať odlišné výsledné hashe. To bráni útokom na heslá založeným na predpripravených tabuľkách (rainbow tables).
2. **Hashovanie hesla**: Po pridaní soli k heslu používame kryptografickú hashovaciu funkciu PBKDF2-HMAC-SHA256 na jeho zahashovanie. Tento algoritmus je odolný voči útokom typu brute-force vďaka svojej opakovateľnosti – vykonávame ho 100 000-krát (parametrom iterácií). Tým sa výrazne spomaľuje čas potrebný na overenie jedného hesla, čo znemožňuje rýchle hádanie hesiel útočníkom. Funkcia je implementovaná nasledovne:

Tento systém zaručuje, že heslo bude silno zašifrované a bezpečne uložené v databáze.

1. **Ukladanie do databázy**: Po zahashovaní je heslo uložené spolu s príslušnou soľou v databáze. Keďže hash a soľ sú ukladané spolu, pri overovaní hesla dokážeme správne overiť zhodu bez nutnosti uložiť heslo v pôvodnej podobe.

**Zdôvodnenie implementovaného systému**

* **Soľ (salt)**: Použitie soli zabezpečuje, že aj rovnaké heslá majú odlišné hashe, čím sa chránia pred útokmi pomocou predpripravených tabuľkových hashov (rainbow tables).
* **PBKDF2-HMAC-SHA256**: Tento algoritmus zvyšuje bezpečnosť hashovania opakovaním procesu 100 000-krát, čím sa bráni útokom typu brute-force. Na implementáciu používame kryptografickú knižnicu **hashlib**, ktorá je štandardnou súčasťou Pythonu, takže nie je potrebné ju inštalovať samostatne.
* **Nepoužívanie hotových knižníc**: Implementácia využíva základné kryptografické funkcie z knižnice hashlib, čo umožňuje plnú kontrolu nad parametrami a správaním systému, čím sa zvyšuje flexibilita a bezpečnosť.

**Príklad procesu pri registrácii používateľa**

1. Používateľ zadá heslo pri registrácii.
2. Systém vygeneruje náhodnú 16-bytovú soľ pre používateľské heslo.
3. Heslo je zahashované spolu so soľou pomocou PBKDF2-HMAC-SHA256 s 100 000 iteráciami.
4. Výsledný hash a soľ sú uložené v databáze spolu s ostatnými údajmi používateľa.

Týmto spôsobom sú používateľské heslá bezpečne uložené, a ani v prípade úniku údajov z databázy útočník nemá prístup k heslám v ich pôvodnej podobe.

1. **Ochrana voči brute-force útokom**

Na ochranu systému pred brute-force útokmi sme implementovali Google **reCAPTCHA**. Tento nástroj overuje, či je používateľom skutočná osoba, a nie automatizovaný skript (bot), ktorý by systematicky skúšal rôzne kombinácie hesiel pri prihlasovaní. reCAPTCHA pridáva ďalší krok, ktorý overuje používateľa predtým, než server spracuje jeho prihlasovacie údaje.

**Implementácia reCAPTCHA**

* **Vytvorenie reCAPTCHA projektu**: Na použitie reCAPTCHA je potrebné najskôr založiť projekt na [Google reCAPTCHA stránke](https://www.google.com/recaptcha" \t "_new). Po vytvorení projektu vám Google poskytne dva kľúče:
  + **Site Key**: Tento kľúč je verejný a používa sa na strane klienta (frontend). Je vložený do HTML formulára a slúži na zobrazenie reCAPTCHA widgetu.
  + **Secret Key**: Tento kľúč je súkromný a používa sa na overenie výsledku reCAPTCHA na strane servera (backend). Slúži na komunikáciu medzi vaším serverom a reCAPTCHA API.
* **Frontendová implementácia**: reCAPTCHA je integrovaná do prihlasovacieho formulára pomocou HTML widgetu, ktorý je aktivovaný pomocou **Site Key**.



* **Backendová implementácia**: Po odoslaní formulára server pošle overovaciu požiadavku na Google pomocou **Secret Key**. Ak Google potvrdí, že reCAPTCHA bola úspešne vyplnená, server pokračuje s overením prihlasovacích údajov.

A screen shot of a computer

Description automatically generated

**Zdôvodnenie implementovaného systému**

Google reCAPTCHA poskytuje silnú ochranu pred brute-force útokmi tým, že overuje, či je prihlasujúci používateľ človek, čím eliminuje riziko automatizovaných útokov. Použitie **Site Key** a **Secret Key** zabezpečuje bezpečnú komunikáciu medzi frontendom a backendom. Validácia na strane servera zabraňuje obchádzaniu kontroly na strane klienta, čo poskytuje dodatočnú vrstvu ochrany.

1. **Ochrana voči slovníkovým heslám**

Cieľom tejto implementácie je zabrániť používateľom registrovať sa s heslami, ktoré sú bežné alebo ľahko uhádnuteľné, ako napríklad "heslo" alebo "123456". Týmto spôsobom zvyšujeme bezpečnosť používateľských účtov a znižujeme riziko útokov na základe jednoduchých hesiel.

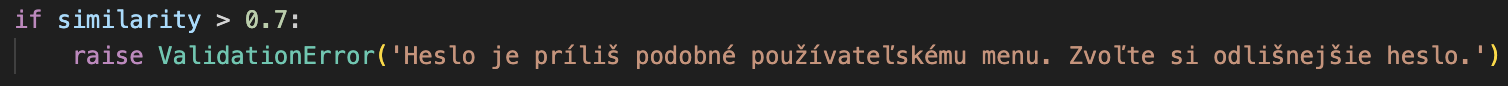
**Použité technológie**

Pri implementácii ochrany voči slovníkovým heslám bola použitá knižnica **difflib**, ktorá umožňuje porovnávanie podobnosti medzi heslami.

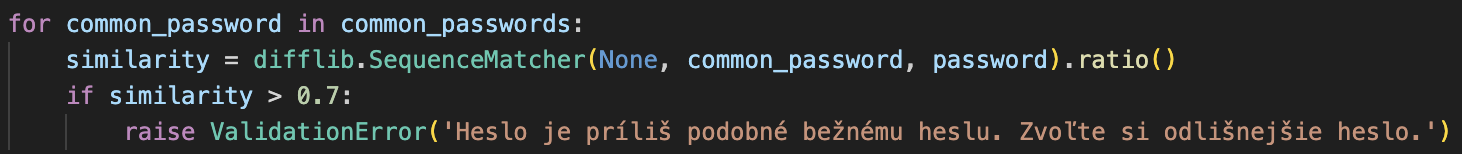
**Mechanizmus kontroly hesiel**

V rámci triedy RegisterForm je definovaná metóda validate\_password, ktorá vykonáva dve hlavné kontroly:

1. **Porovnanie hesla s používateľským menom**: Heslo sa najprv porovnáva s používateľským menom pomocou difflib.SequenceMatcher. Ak je podobnosť medzi heslom a menom väčšia ako 70 %, registrácia bude zamietnutá. Tento krok zabezpečuje, že heslo nie je príliš podobné používateľskému menu, čím sa zvyšuje bezpečnosť.



1. **Porovnanie hesla s bežnými heslami**: Následne sa heslo kontroluje voči databáze bežných hesiel, ktorú program načítava zo súboru. Pre každé bežné heslo sa opäť vypočíta podobnosť, a ak je táto podobnosť väčšia ako 70 %, registrácia je zamietnutá. Okrem toho sa overí, či je heslo priamo na zozname bežných hesiel.



1. **Databáza bežných hesiel**: Na overenie sa používa súbor 10k-most-common-passwords.txt, ktorý obsahuje zoznam bežných hesiel.

Implementovaná ochrana voči slovníkovým heslám zvyšuje úroveň bezpečnosti systému a podporuje používateľov, aby si vyberali silnejšie heslá. Týmto spôsobom sa znižuje riziko úspešných útokov na účty a zabezpečuje sa lepšia ochrana citlivých údajov.

**Pridané knižnice  
1. WTForms**

**Import:** from wtforms import StringField, PasswordField, SubmitField  
**Použitie:** Knižnica WTForms poskytuje validátory pre formuláre vo webovej aplikácii. ValidationError používame na vyvolanie chyby, ak validácia niektorého z polí vo formulári zlyhá (napríklad pri kontrole zložitosti hesla). Ak používateľ zadá nesprávny formát hesla alebo iné neplatné údaje, táto chyba zobrazí používateľovi príslušné chybové hlásenie.  
**Príkaz na inštaláciu:** pip install WTForms

**2. Requests**

**Import:** import requests  
**Použitie:** Knižnicu Requests využívame na odosielanie HTTP požiadaviek. Používame ju napríklad na komunikáciu s reCAPTCHA API od Googlu, aby sme overili, či používateľ úspešne prešiel reCAPTCHA kontrolou. Toto je dôležité pri zabezpečení proti botom alebo automatizovaným útokom.  
**Príkaz na inštaláciu:** pip install requests

**3. re**

**Import:** import re  
**Použitie:** Knižnicu re používame na prácu s regulárnymi výrazmi (regex). Pomocou regulárnych výrazov kontrolujeme formát hesla pri registrácii, napríklad overujeme prítomnosť veľkých a malých písmen, číslic a špeciálnych znakov.

**4. os**

**Import:** import os  
**Použitie:** Knižnicu os využívame na generovanie náhodného reťazca (tzv. salt) pre hashovanie hesiel. Funkcia os.urandom() slúži na generovanie náhodných bajtov, ktoré sú použité pri ochrane hesiel pred útokmi.

**5. hashlib**

**Import:** import hashlib  
**Použitie:** Knižnica hashlib poskytuje funkcie pre hashovanie údajov. Používame ju na implementáciu funkcie PBKDF2 (Password-Based Key Derivation Function 2) s hashovacou funkciou SHA-256 na bezpečné hashovanie používateľských hesiel. Heslá sa hashujú spolu s náhodným reťazcom (salt), aby sme zvýšili ich bezpečnosť.

**6. difflib**

**Import:** import difflib  
**Použitie:** Knižnica difflib nám umožňuje porovnávať reťazce a určiť ich podobnosť. Používame ju na kontrolu, či je zadané heslo príliš podobné používateľskému menu alebo iným bežným heslám, čo by znížilo bezpečnosť hesla.

**Inštalácia všetkých potrebných knižník**

pip install WTForms requests Flask-WTF Flask-SQLAlchemy Flask-Login