APLICAȚIE WEB PENTRU MANAGEMENTUL SUPEREROILOR

Profesor Coordonator: Bogdan Florea

Disciplina: Tehnologii Web

Studenti: Sandu Andreea-Maria & Cucu Teodora-Cristina

Grupa: 443B

CUPRINS

1. Descrierea generală a proiectului

2. Evoluția proiectului și etapele de implementare

3. Arhitectura aplicației (client–server)

4. Tehnologiile utilizate

5. Structura bazei de date și a setului de date (câmpuri)

6. Backend: design API, rutare, validare, CORS

7. Frontend: pagini, componente, flux de navigare

8. Căutare generală și filtrare pe câmpuri relevante

9. Autentificare și autorizare pe roluri (JWT)

10. Panou de control (admin): Insert / Update / Delete

11. Gestionarea imaginilor (Base64/Data URI)

12. Testare și debugging: erori întâlnite și soluții

13. Utilizarea inteligenței artificiale în realizarea proiectului

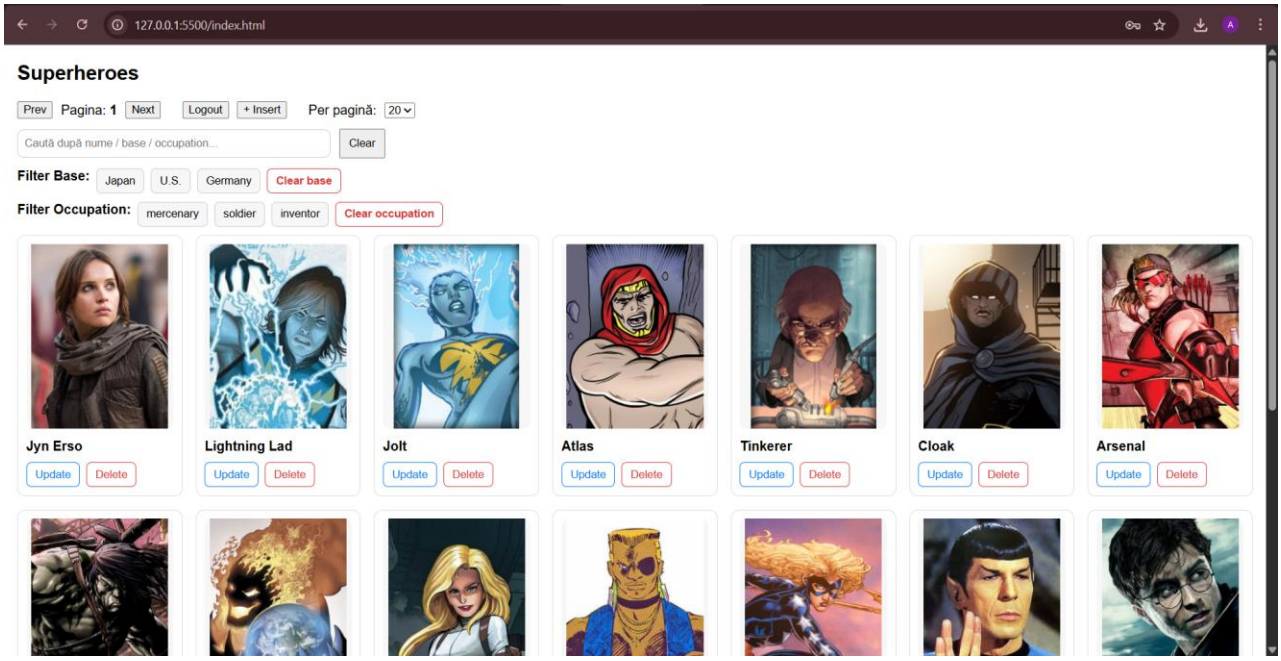
14. Concluzii personale și direcții de îmbunătățire

1. Descrierea generală a proiectului

Proiectul constă în dezvoltarea unei aplicații web care gestionează un set de date despre supereroi. Aplicația permite utilizatorilor să vizualizeze o listă de supereroi, să caute în mod global după text (ex. nume, locația bazei, ocupație) și să aplice filtre pe câmpuri relevante. În plus, proiectul implementează un mecanism de autentificare și autorizare bazat pe roluri (user/admin), astfel încât administratorii să poată executa operații de tip CRUD (Create, Read, Update, Delete) asupra înregistrărilor din baza de date.

Aplicația este construită pe arhitectura client–server. Frontend-ul este o interfață web simplă (HTML/CSS/JavaScript), care consumă un API REST oferit de backend (FastAPI). Backend-ul interacționează cu baza de date MySQL, unde datele despre supereroi sunt stocate într-un câmp JSON, oferind flexibilitate în structurarea informațiilor.

Din punct de vedere funcțional, proiectul acoperă: listare cu paginare, pagină de detalii pentru fiecare supererou, căutare globală, filtre (base și occupation), autentificare (login) și înregistrare (register) pentru utilizatori, precum și funcții administrative (Insert/Update/Delete) disponibile exclusiv pentru rolul admin



2. Evoluția proiectului și etapele de implementare

Dezvoltarea aplicației a urmat o evoluție incrementală, plecând de la conectarea la baza de date și expunerea unui endpoint simplu de listare, până la implementarea completă a autentificării, rolurilor și operațiilor CRUD.

2.1 Inițializare proiect și mediu de lucru

Proiectul a fost creat în PyCharm. Un pas esențial a fost configurarea mediului virtual (venv) și instalarea dependențelor necesare (FastAPI, Uvicorn, SQLAlchemy, jose, passlib etc.). După configurare, aplicația a putut fi pornită local cu Uvicorn.

2.2 Conectarea la MySQL și citirea datelor

În această etapă s-a realizat conexiunea la baza de date MySQL și s-a implementat un endpoint de tip GET pentru listarea înregistrărilor. Datele despre supereroi sunt stocate într-o coloană JSON, ceea ce a necesitat parse-ul corect în Python.

2.3 Paginare și pagină de detalii

Listarea a fost extinsă cu parametrii page și page_size. În frontend, s-au adăugat butoane Prev/Next și un select pentru numărul de elemente per pagină. În plus, fiecare card din listă deschide o pagină de detaliu (detail.html) care afișează numele, imaginea și câmpurile din work.

2.4 Căutare și filtre

S-a implementat o bară de căutare care trimite parametrul q către backend, iar backend-ul caută în câmpurile relevante din JSON (name, work.base, work.occupation). Ulterior, s-au adăugat filtre pe base și occupation prin butoane, inclusiv butoane de clear și evidențierea filtrului activ.

2.5 Autentificare, roluri și control acces

Etapa următoare a introdus un sistem de autentificare bazat pe JWT. Utilizatorii se autentifică printr-un endpoint /login, primesc un access token și un rol. Frontend-ul stochează token-ul în localStorage și trimite token-ul în header-ul Authorization. Accesul la endpoint-urile administrative este restricționat prin dependințe FastAPI (require_user / require_admin).

2.6 Panou de control (admin): Insert/Update/Delete)

În final, pentru rolul admin s-au adăugat funcții CRUD: insert (pagină insert.html), update (update.html) și delete direct din card.

3. Arhitectura aplicației (client–server)

Aplicația este structurată în trei componente principale: frontend, backend și baza de date. Frontend-ul rulează în browser și este responsabil de interfața vizuală și de interacțiunea cu utilizatorul. Backend-ul oferă un API REST (HTTP endpoints) care servește date către frontend, respectand validări și reguli de securitate. Baza de date MySQL asigură persistența datelor.

3.1 Flux de date

1) Utilizatorul se autentifică (login). Backend-ul validează credențialele și returnează token + rol.
2) Frontend-ul stochează token-ul și îl trimite la cereri către backend.
3) Pentru listare/căutare/filtrare, frontend-ul apelează endpoint-ul /items cu parametri (page, page_size, q, base, occupation).
4) Pentru admin, operațiile Insert/Update/Delete sunt disponibile prin endpoint-uri protejate.

4. Tehnologiile utilizate

4.1 Backend

• Python 3 – limbajul principal al backend-ului.
• FastAPI – framework web modern pentru API-uri REST.
• Uvicorn – server ASGI pentru rulare locală.
• SQLAlchemy (core) – pentru conexiune și execuție SQL.
• python-jose – pentru generarea și validarea token-urilor JWT.
• passlib (bcrypt) – pentru hashing-ul parolelor.
• CORS middleware – pentru permiterea cererilor din browser către API.

4.2 Bază de date

• MySQL – sistem de gestiune a bazelor de date relaționale.
• Coloană JSON pentru datele despre supereroi.
• Tabel separat pentru utilizatori, cu roluri.

4.3 Frontend

• HTML5 – structură pagini.
• CSS3 – stilizare (grid, carduri, butoane active).
• JavaScript – fetch API, DOM manipulation, paginare, căutare, filtrare, management token.

5. Structura bazei de date și a setului de date (câmpuri)

5.1 Tabelul users

Tabelul users stochează conturile utilizatorilor aplicației. Câmpurile utilizate au fost:
• id (INT, auto increment) – identificator unic.
• username (VARCHAR) – nume de utilizator.
• password (VARCHAR) – parola hash-uită (bcrypt), stocată ca string.
• role (ENUM sau VARCHAR) – rolul utilizatorului: admin sau user. Pentru utilizatorii obișnuiți se utilizează valoarea default 'user'.

5.2 Tabelul data (supereroi)

Tabelul data conține înregistrările despre supereroi. Structura:
• id (INT, auto increment) – identificator unic.
• data (JSON sau TEXT) – obiect JSON cu informațiile supereroului.

5.3 Structura JSON pentru un supererou

Câmpul data conține un JSON cu cheile relevante pentru aplicație. În proiect s-au folosit în special:
• name – numele supereroului.
• image – imagine (Data URI / Base64) pentru afișare în frontend.
• work – obiect cu subcâmpuri:
  - base – locație (ex. U.S., Japan, Germany etc.)
  - occupation – ocupație (ex. soldier, mercenary, inventor etc.)

Exemplu:

```
{
  "name": "Superman",
  "image": "...",
  "work": {
    "base": "Metropolis",
    "occupation": "Hero"
  }
}
```

6. Backend: design API, rutare, validare, CORS

6.1 Configurare CORS

Fiind o aplicație cu frontend separat (rulat din browser), a fost necesară configurarea CORS. Fără această configurare, browserul blochează cererile către API din motive de securitate.

```
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
```

6.2 Endpoint de listare cu paginare, căutare și filtre

Endpoint-ul /items acceptă parametri de query pentru paginare și filtrare. Paginarea este implementată prin limit și offset. Căutarea și filtrele sunt aplicate în SQL, folosind JSON_EXTRACT pentru a accesa câmpuri din obiectul JSON.

```
offset = (page - 1) * page_size
sql = "SELECT id, data FROM superhero_api.data ... LIMIT :limit OFFSET
:offset"
```

Un aspect important a fost identificarea câmpurilor relevante pentru căutare. În proiect s-au folosit: name, work.base și work.occupation.

7. Frontend: pagini, componente, flux de navigare

7.1 Pagina principală (index.html)

Pagina principală afișează lista de supereroi sub formă de grid cu carduri. Fiecare card include imaginea și numele. Utilizatorul poate naviga între pagini prin Prev/Next și poate schimba page size.

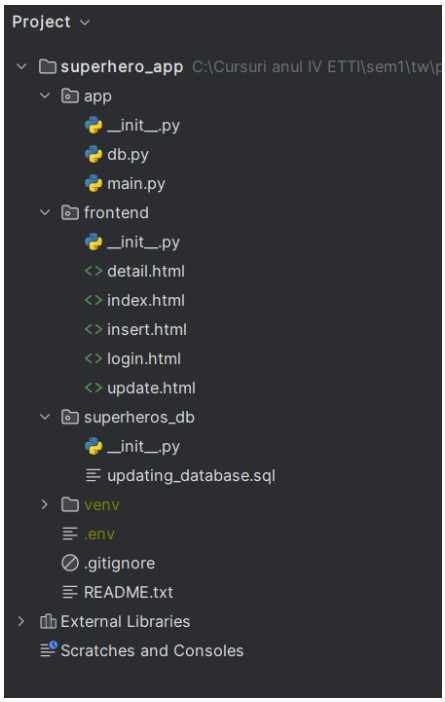
Căutarea se face printr-un input text care trimite parametrul q către backend, cu un mic debounce (setTimeout) pentru a nu trimite cereri la fiecare tastă în mod agresiv.

7.2 Pagina de detalii (detail.html)

La click pe un card, utilizatorul este redirecționat către detail.html?id=<id>. Pagina apelează /items/{id} și afișează câmpurile relevante din răspuns. Pentru o prezentare mai plăcută, work este afișat sub formă de text formatat, nu ca JSON brut.

7.3 Pagini administrative

Pentru rolul admin, aplicația oferă paginile insert.html și update.html. Insert permite crearea unui supererou nou. Update preia datele curente și permite modificarea lor, apoi trimite un PUT către backend.



8. Căutare generală și filtrare pe câmpuri relevante

8.1 Căutare generală (q)

Căutarea generală a fost implementată printr-un parametru q. Backend-ul caută textul introdus de utilizator în câmpurile considerate relevante. În mod concret, au fost incluse name, work.base și work.occupation.

8.2 Filtrare (base și occupation)

Filtrele au fost implementate ca butoane. La apăsare, frontend-ul setează baseFilter sau occFilter și reapelează funcția load(). În interfață, butonul selectat rămâne evidențiat (clasa active), iar butoanele de clear sunt stilizate diferit pentru a se diferenția.

9. Autentificare și autorizare pe roluri (JWT)

9.1 Fluxul de autentificare

1) Utilizatorul introduce username și password.
2) Frontend-ul trimite datele către /login.
3) Backend-ul verifică user-ul în baza de date și validează parola (bcrypt).
4) Backend-ul generează token JWT care conține sub (username) și role.
5) Frontend-ul stochează token-ul în localStorage și îl atașează în header-ul Authorization: Bearer <token>.

9.2 Protecția endpoint-urilor

În backend s-au implementat funcții de tip dependency (require_user și require_admin). Endpoint-urile generale (listare, detalii) pot fi accesate de orice utilizator autentificat, iar endpoint-urile administrative sunt limitate la rolul admin. Aceasta respectă principiul least privilege.

```
def require_admin(user=Depends(require_user)):
    if user['role'] != 'admin':
        raise HTTPException(status_code=403, detail='Admin only')
    return user
```

10. Panou de control (admin): Insert / Update / Delete

10.1 Insert (Create)

Insert-ul este realizat printr-o pagină dedicată (insert.html) care trimite un POST către backend. Administratorul introduce numele, imaginea (Data URI/Base64) și câmpurile work. Un aspect important a fost persistarea tranzacției în MySQL. S-a folosit engine.begin() pentru a asigura commit automat.

10.2 Update

Update-ul este implementat prin update.html?id=<id>. Pagina preia datele curente cu GET și, la salvare, trimite un PUT cu JSON-ul complet al supereroului. În listă (index.html), butonul Update apare doar pentru admin și redirecționează la update.html.

10.3 Delete

Delete-ul este disponibil direct pe card pentru admin. La click se afișează un confirm dialog pentru a evita ștergerile accidentale. După ștergere, lista se reîncarcă. Endpoint-ul este protejat de require_admin.

```
<h2>Insert Superhero</h2>

<input id="name" placeholder="Name" />
<input id="image" placeholder="Image (Data URI / Base64)" />
<input id="base" placeholder="Work base" />
<input id="occupation" placeholder="Work occupation" />

<button id="save">Save</button>
<div id="msg" class="msg"></div>

<script>
const API = "http://127.0.0.1:8000";
const token = localStorage.getItem("token");
const role = localStorage.getItem("role");

if (!token) window.location.href = "login.html";
if (role !== "admin") {
  document.getElementById("msg").textContent = "Doar admin poate insera.";
  document.getElementById("save").disabled = true;
```

11. Gestionarea imaginilor (Base64/Data URI)

Un element specific proiectului este afișarea imaginilor supereroilor. În loc să fie folosite URL-uri externe sau fișiere locale, imaginile sunt stocate în baza de date ca Data URI (Base64). Această abordare permite transportarea imaginii direct în JSON.

În frontend, imaginea este afișată direct prin setarea atributului src al tag-ului <img> la valoarea Data URI. Exemplu: img.src = item.image;

Pentru a obține o imagine Base64 s-au folosit convertoare online.

```
import base64
with open('hero.jpg','rb') as f:
    b64 = base64.b64encode(f.read()).decode()
data_uri = '' + b64
```

12. Testare și debugging: erori întâlnite și soluții

12.1 Commit în tranzacții (INSERT/UPDATE/DELETE)

O problemă a fost situația în care insert-ul părea că rulează, dar datele nu apăreau în DB. Cauza principală: lipsa unui commit corect. Soluția: utilizarea engine.begin() în loc de engine.connect(), pentru a beneficia de tranzacții cu commit automat.

12.2 Mesaje și UX în cazul filtrelor fără rezultate

Atunci când filtrarea returna o listă goală, interfața nu era clară pentru utilizator. S-a introdus un mesaj informativ care explică faptul că nu există supereroi corespunzători filtrului selectat. A fost rezolvat și un bug în care mesajul nu dispărea după clear, prin resetarea explicită a zonei de mesaje la fiecare load().

13. Utilizarea inteligenței artificiale în realizarea proiectului

Inteligența artificială a fost utilizată ca instrument de suport în dezvoltare, în special pentru clarificarea conceptelor, generarea de schițe de cod și debugging.

13.1 Exemple concrete de utilizare a AI

• Structurarea proiectului: sugestii privind organizarea backend/frontend.
• Implementarea paginării și a query-urilor: explicații despre limit/offset.
• Implementarea căutării în câmpuri JSON în MySQL (JSON_EXTRACT).

• Autentificare și roluri: JWT, require_user/require_admin.
• Debugging: identificarea bug-urilor (de ex. handler de filtru cu variabile out of scope).
• DevOps minimal: soluții pentru probleme Git/GitHub (.gitignore, fișier >100MB, rebase conflicts)

14. Concluzii personale și direcții de îmbunătățire

Proiectul a oferit o experiență practică completă în dezvoltarea unei aplicații web cu backend și frontend separate. Au fost implementate funcționalități uzuale din aplicațiile reale: autentificare, roluri, listare cu paginare, căutare/filtrare și operații CRUD. În același timp, au fost întâlnite probleme reale de dezvoltare (tranzacții DB, bug-uri JavaScript), iar rezolvarea lor a consolidat înțelegerea tehnologiilor.

14.1 Dificultăți întâmpinate

• Configurarea mediului (venv, dependențe) și rularea corectă a serverului.
• CORS și accesarea API-ului din browser.
• Manipularea JSON în MySQL și construirea query-urilor.
• JWT și control acces (user vs admin).
• Persistența tranzacțiilor (commit) la operații de scriere.
• Probleme Git (fișiere mari, conflicte rebase).

14.2 Soluții implementate

• Folosirea engine.begin() pentru tranzacții.
• Organizarea clară a handler-elor JS și evitarea variabilelor în scope greșit.
• Introducerea mesajelor UX pentru rezultate goale la filtre.
• .gitignore corect și eliminarea fișierelor mari din repo.

În concluzie, aplicația îndeplinește cerințele funcționale cerute și oferă o bază solidă pentru extindere. Procesul de dezvoltare a fost unul incremental, cu testare continuă și îmbunătățiri bazate pe feedback și debugging.