PROJEKTRAPPORT

Webbtjänster DA109A

Cyrus Shaerpour

Charbel Esdras

Ibrahim Ayoubi

Goran Slavnic

Om tjänsten

Gruppens tjänst är en webbaserad applikation som låter användaren skapa en personlig filmlista för att få en sammanfattande översikt i form av en så kallad wrapped – inspirerad av Spotify Wrapped. Tanken med tjänsten är att ge användaren insikter om sitt filmtittande genom enkel statistik och visualisering. Ett exempel på en sådan insikt är vilken genre som dominerar användarens lista och hur många filmer som har lagts till för tillfället.

Applikationen bygger på att användaren själv lägger till filmer som sedan berikas med extern data via filmrelaterade API:er. Resultatet presenteras i ett visuellt gränssnitt med stor fokus på enkelhet och underhållningsvärde.

Tekniska lösningen

Backend

Applikationen är uppdelad i en frontend och backend. Front är byggd med hjälp av HTML, CSS och JavaScript och ansvarar främst för interaktionen och visualiseringen av datan. Backend är byggd med ramverket FastAPI och exponerar ett REST-baserade API som frontend kommunicerar med via HTTP och JSON.

Backend hanterar även applikationens användardata och kommunikation med externa API:er. HTTP-anrop sker i Python för att göra anrop till externa datakällor, såsom TMDb och OMDb.

Kommunikationen med externa filmdatabaser görs med hjälp av biblioteket httpx. Med hjälp av detta kan flera API-anrop hanteras effektivt.

API-nycklar hanteras även via python-dotenv, eftersom att känslig information bör lagras säkert utanför källkoden.

Frontend

Frontend är skapad med hjälp av HTML och CSS för struktur och utseende. Javascript används för att hantera användarinteraktion, kommunikation med backendens API och en mer dynamisk respons. Javascript ansvarar också för presentationen av användarens data via tjänstens Wrapped-funktion. I denna funktion ingår animationer och visuell statistik.

Datakällor & externa API:er

Tjänsten använder två externa API:er för att hämta och kombinera information relaterad till filmer. TMDb API (The Movie Database) används som primär källa för filminformation. Via detta API hämtas grundläggande uppgifter om filmer, såsom titel, utgivningsdatum, poster, genre, speltid samt direktör. API:et används främst för sökning av filmer och när användaren lägger till filmer i sin "favoritlista".

OMDb (Open Movie Database) används som ett komplement till TMDb för att hämta filmbetyg från IMDb. Den används först i samband med TMDb när användaren söker efter film, men om en OMDb rating inte är tillgänglig används TMDb ratings. Utöver det används OMDb i Wrapped, för att ge användaren en sammanfattning och jämförande av deras filmsmak med en genomsnittlig rating av dessa filmer från IMDb.

Genom att kombinera data från dessa två API:er skapas en mashup där information från flera källor förs samman och presenteras via tjänstens egna API-endpoints.

Mashup – hur datan kombineras

- Filmdata hämtas från TMDb
- Ytterligare betyg hämtas från OMDb
- Filmer som sparas av användaren sparas som användargenererad data (favoritlista)
- Datan i favoritlistan bearbetas till ny statistic
    - Total tittartid
    - Vanligaste genre
    - X antal filmer med X antal ratings
- Presenterar resultatet via en Movie Wrapped: den visuella vyn

Databehandling

- Filmdatan från API:et filtreras och omstruktureras innan den visas i gränssnittet.
- Speltiden i minuter konverteras till ett format som är mer läsbart – timmar och minuter.
- För att få fram användarens vanligaste genre i Movie Wrapped så har genre data sammanställts.
- Alla användardata lagras lokalt i en JSON-fil.

Användarmanual

Installationsinstruktioner

För att köra applikationen lokalt krävs att Python 3.10 eller senare är installerat på datorn. Applikationen är byggd med FastAPI och använder externa APIer för filmdata.

- **Ladda ner projektet**
  Projektets filer laddas ner och placeras lokalt på din enhet.

- **Skapa och aktivera virtuellt python-miljö**
  I projektets mapp skapas ett virtuellt environment:
  - python -m venv venv

- **Aktivera sedan miljön:**
  MacOS/Linux:
  - source venv/bin/activate

  Windows:
  - venv/scripts/activate

- **Python-bibliotek installeras i environmentet:**
  - FastAPI
  - Uvicorn
  - python-dotenv
  - httpx

  Installera: pip install fastapi uvicorn python-dotenv httpx

- **Hantering av API-nycklar**
  - Skapa en .env-fil och lägg till giltiga API-nycklar enligt följande:
    API__KEY=<TMDBs API-nyckel>
    OMDB_API_KEY=<OMDBs API-nyckel>

- **Följande mappar/filer ska finnas:**
  - main.py
  - services.py
  - models.py
  - templates/
  - static/
  - users.json (skapas automatiskt efter första användning)

Körinstruktioner

- **Starta server**
  När det virtuella environmentet är aktivt startas applikationen med:
  - uvicorn main:app --reload

- **Öppna applikationen i webbläsare**
  Applikationen nås via http://127.0.0.1:8000

- **Användning av applikationen**
  - Användaren kan registrera ett konto och logga in
  - Söka filmer via titel eller regissör
  - Lägga till och ta bort filmer i sin personliga lista
  - Se en sammanfattning i form av Wrapped baserat på sparade filmer

- **Avsluta applikationen**
  - Servern stoppas genom att trycka på ctrl + c i terminalen
  - Det virtuella environmentet avslutas med: deactivate

API-dokumentation

Vi missförstod uppgiften lite i början och gjorde den första versionen utan realiserade endpoints som fungerar som riktiga api anrop. Det var helt HTML-baserat så vi fick strukturera om lite i slutet. En stor del av filerna som hade skapats var redan färdiga och kunde återanvändas innan vi skrev om python filerna i ett nytt repo med Fast-API istället för FLASK. Då verkade det lättare att använda för detta samt att det fanns lite guide från labbarna. APIt är nu designat med endpoints för det främsta funktionerna i programmet. Dvs ta fram och hantera data gällande filmer. Det viktigaste i en filmdatabas är ju att kunna söka efter film och få information kring dem. Detta sker via endpoint search. Register och login hanterar användare och sparar konton i programmets json fil. Favorites anropen hanterar my list som är huvudfunktionen i appen. Alltså att kunna spara filmer till sin egna lista för att sedan sammanställa en wrapped från den sammanlagda informationen från de filmerna. Favorite funktionerna är sparade till lista, visa lista och ta bort från lista. Listor är också sparade i json under sina respektive användare. Slutligen ville vi att wrapped också skulle inkluderat i apiet, då det är kärn iden i programmet och funktionen som sammanställer alla annan data som redan hanteras av övriga funktioner

APIet är designat enligt REST-principer för att följa en tydlig och logisk struktur som är lätt för användare att förstå. Vi använder standardiserade HTTP metoder som GET för att hämta data, exempelvis sökningar och listor, POST för att skapa resurser som att lägga något i favoriter och DELETE för att ta bort resurser. På detta sätt blir APIet lätt att förstå och följer populära mönster. Just de sju metoderna vi använder anser vi också räcker för att uppfylla all funktionalitet som programmet behöver i nuläget. Samt låter andra användare av APIet ta del av all relevant data vi bearbetar.

APIet är inte helt RESTful enligt Fieldings definition då vi inte använder oss av HATEOAS men borde uppnå kraven för att klassas som level 2 av RESTfulness enligt Richardsons maturity model. Med anledning att vi använder flera HTTP metoder och URIer.

Applikationen använder sig av ett eget internt API via fastapi-routes som hanterar:

- inloggning och användarsessioner för användare
- möjligheten att söka fram filmer
- hantering av favoritfilmer
- beräkning av statistik för Movie Wrapped

API-designen använder endpoints för de olika funktionerna.

Metod	Endpoints	Beskrivning
GET	**/api/search**	För att söka namn på filmer eller regissörer via TMDb som returnerar filmer med det namnet eller filmer regisserade av den regissören. Gör ett queryanrop till TMDb databasen via deras API.
POST	**/api/register**	Registrerar en ny användare med ett användarnamn och lösenord. Sparas i json.
POST	**/api/login**	Loggar in en användare med användarnamn och lösenord. Verifierar via json.
GET	**/api/favorites**	Hämtar listan med filmer som en användare har lagt in i favoriter (my list).

POST	**/api/favorites**	Sparar en film till my list hos en inloggad användare. Sparas under en användare i json.
DELETE	**/api/favorites/{movie_id}**	Tar bort en film från my list hos en inloggad användare baserat på filmens TMDb id.
GET	**/api/wrapped**	Sammanställer och hämtar wrapped statistik för en inloggad användare baserat på användarens sparade filmer i sin lista.

GET /api/search

Query parametrar: query (string, required)

Exempel: /api/search?query=Sinners

Response: 200 OK

Returdata: Lista med filmobjekt som innehåller id, titel, poster, genre, speltid och betyg


```
{
  "movies": [
    {
      "id": 1233413,
      "title": "Sinners",
      "poster_url": "https://image.tmdb.org/t/p/w342/qTvFWCGeGXgBRaINLY1zqgTPSpn.jpg",
      "release_date": "2025-04-16",
      "rating": 7.5,
      "director": "Ryan Coogler",
      "runtime": 138,
      "genres": [
        "Horror",
        "Action",
        "Thriller"
      ],
      "imdbRating": null
    },
    {
      "id": 328358,
      "title": "Sinners",
      "poster_url": "https://image.tmdb.org/t/p/w342/zrnJMj4iTImAdnsAFN1Qom3JVR0.jpg",
      "release_date": "1990-01-29",
      "rating": 3.2,
      "director": "Charles T. Kanganis",
```


POST /api/register

Headers: Content-type -application/json

Request body (JSON):

- username (string) - användarnamn
- password (string) - lösenord

Response: 200 OK, 400 User already exists, 422 validation error

Returdata exempel:

```
{
  "status": "ok",
  "message": "User created"
}
```

POST /api/login

Headers: Content-Type: application/json

Request body (JSON):

- username (string) - användarnamn
- password (string) - lösenord

Response: 200 OK, 401 invalid credentials, 422 validation error

Returdata:

```
{
  "status": "ok",
  "username": "exempel"
}
```

GET /api/favorites

Authentication: Kräver inloggad användare

Request body: Ingen

Response: 200 OK, 401 not authenticated

Returdata: Lista med sparade filmobjekt. Exempel:

```
{
  "favorites": [
    {
      "id": 1233413,
      "title": "Exempel Film",
      "poster_url": "https://fastly.picsum.photos/id/17/2500/1667.jpg?hmac=HD-JrnNUZjFiP
      "release_date": "xxxx-xx-xx",
      "rating": 10,
      "director": "Jag",
      "runtime": 109,
      "genres": [
        "Exempel"
      ],
      "imdbRating": null
    },
    {
      "id": 283020,
      "title": "For Example, Argentina",
      "poster_url": "https://image.tmdb.org/t/p/w342/ejGVNW4xGohKXCdfByTw9hII07Y.jpg",
      "release_date": "1985-12-02",
      "rating": 7,
      "director": "Werner Schroeter",
      "runtime": 92,
      "genres": [
```


POST /api/favorites

Authentication: Kräver inloggad användare

Headers: Content-Type: application/json

Request body (JSON):

- movie_id (integer) - TMDb id för filmen som ska sparas

Response: 200 OK, 401 not authenticated, 422 validation error

Returdata exempel:


```
{
  "status": "ok",
  "message": "Added to favorites"
}
```


DELETE /api/favorites{movie_id}

Path parameters: movie_id (integer, obligatorisk) - TMDb id för filmen

Authentication: Kräver inloggad användare

Response: 200 OK, 401 not authenticated, 422 validation error

Returdata:

```
{
  "status": "ok",
  "message": "Removed"
}
```


GET /api/wrapped

Authentication: Kräver inloggad användare

Request body: Ingen

Response: 200 OK, 401 not authenticated

Returdata: wrappedobjekt med sammanställd statistik. Exempel:


```
{
  "hours": 1,
  "minutes": 49,
  "total_movies": 1,
  "average_rating": 10,
  "most_common_genre": "Exempel",
  "taste_label": "You are a true connoisseur, hats off good man!",
  "rated_movies": 1
}
```


Externa API-anrop

Applikationen använder sig av TMDb och OMDb API via HTTP GET-anrop.

- Söka film från TMDb
- Söka regissör via TMDb
- Hämta film detaljer primärt via TMDb
- Hämta regissör via credits via TMDb
- Hämta filmbetyg från OMDb primärt men använda TMDb som reserv om det inte skulle finnas något OMDb betyg. Dessa anrop kräver en giltig API-nyckel.