Introduktion

Denna handout presenterar en praktisk övning där du ska vidareutveckla en webbapplikation som använder ett externt API. Du arbetar vidare på en befintlig kodbas som du hittar på GitHub.

Länk till exempelkod

https://github.com/holgros/TE24D_webbutveckling/tree/master/vt%20v07

(Notera: Kopiera länken exakt inklusive mellanslag och specialtecken)

Viktigt: Ladda ner eller klona hela mappen för att få alla nödvändiga filer.

Övningens syfte

Du ska vidareutveckla en webbplats som hämtar filmdata från **Studio Ghibli API**. Målet är att förbereda dig för den kommande inlämningsuppgiften genom att:

1. Säkerställa förståelse för API-kommunikation
2. Träna på felhantering och användarvänlighet
3. Implementera delbara URL:er
4. Skapa både söksida och detaljsida

Del 1: Förberedelse (10 minuter)

1. Öppna GitHub-länken ovan
2. Ladda ner eller klona hela mappen `vt v07`
3. Öppna mappen i VS Code eller din favorit-editor
4. Studera filstrukturen:
   - `index.html` - Huvudsida med filmsökning
   - `details.html` - Detaljsida för enskilda filmer
   - `app.js` - JavaScript för huvudfunktionalitet
5. Testa att öppna `index.html` i webbläsaren
6. Klicka på en film för att se detaljsidan

Del 2: Problemformuleringar (40 minuter)

Du ska nu vidareutveckla webbplatsen genom att lösa följande problem:

Problem 1: Förbättrad sökfunktion

Nuvarande situation: Sökfältet filtrerar redan visade filmer på klientsidan.

Uppgift: Implementera **server-side sökning** genom att använda API:ets egna sökfunktion (om möjligt) eller genom att hantera filtrering mer effektivt.

Listing 1: Exempel på förbättrad sökning

```
// Istället för att hämta ALLA filmer och sedan filtrera:
async function searchMovies(searchTerm) {
    // Gör API-anrop med sökterm
    const response = await fetch(`https://ghibliapi.vercel.app/films?q=${
       searchTerm}`);
    // ... resten av koden
}
```

Problem 2: Paginering och "Ladda mer"

Nuvarande situation: Alla filmer laddas samtidigt, vilket kan vara långsamt.

Uppgift: Implementera en Ladda fler filmer-knapp eller oändlig scroll.

1. Visa först 5-10 filmer
2. Lägg till en knapp "Visa fler filmer"
3. När knappen klickas, ladda in nästa sätt med filmer
4. *Extra utmaning:* Implementera oändlig scroll

Problem 3: Caching för bättre prestanda

Nuvarande situation: Varje gång man går till detaljsidan görs ett nytt API-anrop.

Uppgift: Implementera enkel caching med `localStorage`.

Listing 2: Exempel på caching

```
// Spara filmer i localStorage
localStorage.setItem('ghibli_movies', JSON.stringify(movies));

// Hämta från localStorage om det finns
const cached = localStorage.getItem('ghibli_movies');
if (cached) {
    return JSON.parse(cached);
}
```

Problem 4: Responsiv bildhantering

Nuvarande situation: Bilder laddas med `loading=lazy"` men kan förbättras.

Uppgift: Implementera `srcset` eller JavaScript-baserad bildoptimering.

- Skapa olika storlekar för olika skärmar
- Visa lågupplöst version först, ladda sedan högupplöst
- Hantera bildfel elegant

Problem 5: Ytterligare API-information

Nuvarande situation: Endast grundläggande filminformation visas.

Uppgift: Lägg till information från relaterade API-endpoints.

- Visa karaktärer i varje film (använd `/people` endpoint)
- Visa relaterade filmer eller recommendations"
- Lägg till trailers eller ytterligare media

Del 3: Förberedelse för inlämningsuppgiften senare i vår

Följande krav kommer att gälla för att få höga betyg på den kommande inlämningsuppgiften senare i vår. Testa dig själv genom att kontrollera: **"Kan jag detta nu?"**

Krav 1: API-kommunikation

- Min sida kommunicerar med ett externt API
- Jag kan förklara skillnaden mellan GET, POST, PUT, DELETE
- Jag kan hantera API-nycklar säkert (inte i klientkod om möjligt)

Krav 2: Söksida med lista

- Det finns en sida för sökning
- Söksidan visar en lista av objekt från API:et
- Varje objekt har:
  - Namn/titel
  - Övrig relevant information
  - Bild om API:et tillhandahåller det
- Varje objekt länkar till en detaljsida
- Sökfält finns på söksidan (och ev. i header)
- Det finns ett menyalternativ till söksidan

Krav 3: Detaljsida

- Detaljsidan hämtar information via separat API-anrop
- URL:en är delbar (man kan skicka länken till någon annan)
- All relevant information visas
- Bild visas om tillgänglig

Krav 4: Felhantering

- Användaren får trevliga felmeddelanden vid API-fel
- Felmeddelandena är relevanta för användaren
- Fel hanteras med try-catch eller .catch()

Krav 5: Prestanda och tillgänglighet

- Bilder optimeras (lazy loading, srcset, eller motsvarande med JS)
- Sidan är responsiv
- Loading-indikatorer visas vid väntan

Rekommenderade API:er för inlämningsuppgiften

The Movie Database (TMDB)

- https://developers.themoviedb.org/3
- Kräver API-nyckel (gratis)
- Omfattande filmdatabas

Spoonacular (Recept)

- https://spoonacular.com/food-api
- Gratis nivå med begränsade anrop
- Perfekt för matlagningsapplikationer

Star Wars API (SWAPI)

- https://swapi.dev/
- Ingen API-nyckel krävs
- Begränsad men ren data

Andra förslag

- **NASA API:** https://api.nasa.gov/
- **OpenWeatherMap:** https://openweathermap.org/api
- **PokéAPI:** https://pokeapi.co/

- **Dog API:** https://dog.ceo/dog-api/