**Eksamensdokument - Gruppe 28**

## **1. Valgt vanskelighetsgrad**

**Valgt kravnivå:** A

## **2. Studenter og GitHub-alias**

|  |  |
| --- | --- |
| **Navn** | **GitHub brukernavn** |
| Rama Rihawi Zanerni | RamaZanerni |

## **3. GitHub-lenke til prosjektet**

<https://github.com/RamaZanerni/billettlyst.git>

## **4. Redegjørelser, forutsetninger og utfordringer**

**Jobbet alene:**

Hele prosjektet ble gjennomført alene, noe som førte til økt kompleksitet i både utvikling, testing, og feilsøking.

### **Utfordringer med Ticketmaster API**

Under utviklingen støtte jeg på flere utfordringer knyttet til integrasjon med Ticketmaster API, spesielt:

Feil **429 Too Many Requests**, selv etter at jeg implementerte forsinkelser mellom forespørsler (await delay(500)).

Manglende eller ufullstendig informasjon i API-responsen for enkelte festivaler, inkludert **Skeikampen**, hvor ingen arrangementer ble returnert.

### **Fallback for tomme resultater (Skeikampen)**

I komponenten Hjem.jsx, implementerte jeg logikk som sjekker om resultatet fra API er tomt.

Et konkret eksempel er Skeikampen-festivalen, hvor Ticketmaster API ikke returnerer noen events.

I slike tilfeller viser applikasjonen følgende meldinger til brukeren:

* **"Couldn't load events"**
* **"No events found for this festival"**
* En knapp: **"Try Again"** for å forsøke å laste inn data på nytt

Dette forbedrer brukeropplevelsen ved å skille mellom tekniske feil og manglende innhold, og gir brukeren tydelig informasjon om hva som skjer.

### **Fallback for bilder**

For events, venues og artists som manglet bilde i API-responsen, brukte jeg AI for å finne en metode for fallback ved hjelp av onError-håndtering.

Jeg valgte å bruke en lokal backup (placeholder-bilde) i stedet for en ekstern URL, for å sikre stabilitet og kontroll over designet.

**Sanity CMS oppsett:**

Brukte AI som støtte for å sette opp event og user schema, men tilpasset feltene selv, inkluderte referanser og hotspot til bilder.

## **5.Kode med ekstern hjelp (med kilde)**

**Håndtering av API-feil 429 (inspirasjon fra AI):**

Kilde: ChatGPT og Deepseek

Hvor: Alle komponenter som henter fra Ticketmaster API (Hjem.jsx, EventPage.jsx, FestivalEventsPage.jsx, CategoryPage.jsx).

Hvorfor: For å implementere en enkel løsning for å forsinke API-forespørsler og prøve å redusere antall 429-feil, men løsningen viste seg å være delvis effektiv, og det ble lagt til fallback-meldinger til bruker.

## **6. Bruk av kunstig intelligens (AI) - dokumentasjon**

### **Bruk 1: Håndtering av 429 fra API**

**Prompt brukt:** Hvordan kan jeg unngå 429 Too Many Requests når jeg bruker Ticketmaster API i React?

**Resultat AI:**

const delay = (ms) => new Promise(resolve => setTimeout(resolve, ms));

await delay(300), await delay(500), await delay(800);

**Hvor i koden brukt:**

FestivalEventsPage.jsx, Hjem.jsx og CategoryPage.jsx

### **Bruk 2: Fallback for bilder**

**Prompt brukt:** Hvordan legge til fallback-bilde ved feil i bilde i React?

**Resultat AI:**

onError={(e) => {  
 e.target.src = '<https://via.placeholder.com/500>';  
}}

**Hvor i koden brukt:**

EventCard.jsx, FestivalEventCard.jsx, VenueCard.jsx og AttractionCard.jsx

### **Bruk 3: Best practice useEffect for API fetch**

**Prompt brukt:** Best practice for å bruke useEffect for å hente data fra API og håndtere loading og error.

**Resultat AI:**

useEffect(() => {  
 const fetchData = async () => {  
 try {  
 setLoading(true);  
 const res = await fetch(url);  
 const data = await res.json();  
 setData(data);  
 } catch (error) {  
 setError(error);  
 } finally {  
 setLoading(false);  
 }  
 };  
 fetchData();  
}, [dependency]);

**Hvor i koden brukt:**

EventPage.jsx, Hjem.jsx, Dashboard.jsx, FestivalEventsPage.jsx og CategoryPage.jsx

### **Bruk 4: Sanity schema oppsett**

**Prompt brukt:** Skriv Sanity schema for event med de nødvendige feltene og user med wishlist, venner og tidligere kjøp.

**Resultat AI:** Opprettet basis schemas for event og user.Jeg tilpasset feltene selv, la til hotspot for bilder, list i kategori, og referanser.

**Hvor i koden brukt:**Event.js, user.js og schema.js

## **bruk av AI**

AI ble brukt som støtte i spesifikke utfordringer knyttet til API-feil, fallback-mekanismer, og oppsett av schemas. All kode ble gjennomgått, testet, og tilpasset av meg. AI ble ikke brukt til å generere hele prosjektet, kun som støtte i enkelte deler som dokumentert. AI-en gjorde imidlertid mange feil og frustrerte meg i noen situasjoner, og jeg endret mye ting.

## **8.Redegjørelser og refleksjoner**

**Bruk av branches:**

Jeg oppdaget dessverre kravet om bruk av branches sent i prosjektet. For å forstå prinsippet opprettet jeg 2 test-branches og gjennomførte endringer i egne branches før jeg opprettet pull requests og flettet dem tilbake til main. men prosjektet ble i hovedsak utviklet på main.

Jeg ser verdien av å bruke branches for bedre kontroll og samarbeid, og vil prioritere dette i fremtidige prosjekter.

Jeg har dokumentert disse branchene i GitHub-repositoriet, og har lært at dette burde vært en del av arbeidsprosessen fra starten av prosjektet.

**Fokus på struktur og design:**  
 Jeg la stor vekt på brukervennlig design og strukturert CSS.  
 Jeg fokuserte på:

Semantisk HTML, gjenbrukbare komponenter, responsivitet, og Universell utforming.  
 Dette for å sikre at løsningen fungerer godt på alle enheter, og gir en god opplevelse til brukeren.

**Valg av løsninger med API og fallback:**  
 På grunn av de utfordringene jeg møtte med Ticketmaster API (spesielt 429 og manglende data), valgte jeg løsninger som prioriterte stabilitet og tydelig tilbakemelding til brukeren fremfor komplekse retry-logikker.

### **Bruk av proxy (forklaring)**

Under utviklingen brukte jeg en client-side proxy via vite.config.js:

server: {  
 proxy: {  
 '/discovery': {  
 target: '<https://app.ticketmaster.com>',  
 changeOrigin: true,  
 secure: false,  
 rewrite: path => path.replace(/^\/discovery/, '/discovery')  
 }  
 }  
 }

Dette ble brukt for å omgå CORS under utvikling, men det beskytter ikke API-nøkler og bør ikke brukes i produksjon.  
 I produksjon krever dette en server-side proxy, men i dette prosjektet fokuserte jeg på utviklingsproxy for enkelhet og rask testing.

## **Om StrictMode og dobbelt kjøring av useEffect**

Console.log ("✅ fetchFestivals ran (React StrictMode may cause double run in development)");

React.StrictMode er aktivert under utvikling for å hjelpe med å oppdage potensielle feil i koden. Den gjør blant annet at useEffect() kjøres to ganger med vilje for å teste om sideeffekter (f.eks. fetch-kall) er trygge.

I denne applikasjonen kan det føre til at API-kall gjentas midlertidig i utviklingsmodus og forårsake 429-feil fra eksterne API-er (Too Many Requests).

Derfor ble enten:

* StrictMode midlertidig fjernet **eller**
* det ble lagt til mekanismer for å forhindre dobbelt kjøring (f.eks. useRef() som sperre).

I produksjonsmodus (build) skjer dette ikke, og koden fungerer som forventet.

**Navn og bilder**

Jeg har brukt falske bilder fra denne nettsiden. Fra:

<https://pixabay.com/photos/portrait-woman-fashion-people-3190849/>

<https://www.pexels.com/photo/photography-of-a-guy-wearing-green-shirt-1222271/>

<https://pixabay.com/photos/girl-beauty-hat-cover-a-smile-4361977/>

De fiktive navnene som brukes i prosjektet er: Rama, Omar og Lana.

Kilder: pixabag

<https://pixabay.com/photos/portrait-woman-fashion-people-3190849/>