Gruppe 23

**Vanskelighetsgrad:** A

**GitHub – navn:**

* DeOne313 = Mohamed Baker Alezairjawi
* Fwjfwe = Isa Adamovitsj Isajev
* adabaiz = Adam Mangoor

**GitHub – link:** <https://github.com/fwjfwe/eksamengruppe23>

**Redegjørelse / Forutsetninger**

**Oppgaveteksten sier:**

***«EventCard-komponentene i Home og Dashboard skal være klikkbare: Hele kortet, eller en dedikert knapp/lenke, skal kunne føre brukeren videre til en detaljvisning av eventet. Ved klikk skal brukeren navigeres til EventPage-komponenten, som viser detaljert informasjon om det valgte arrangementet. Dette gjelder kun der EventCard brukes til festivalkortene og på Dashboard-siden senere i oppgaven. Anbefalt rute er: /event/:id, basert på ID fra API-et.»***

Siden krav for karakter C hovedsakelig omhandler at man skal kunne vise en innloggingsside med tilhørende skjema, tolket vi det slik at dette kravet om EventCard og detaljvisning først og fremst er relevant for B-nivået. Derfor gjorde vi en forutsetning om at EventCard i Dashboard gjelder kun dersom man løser oppgaven på B-nivå og kobler opp Sanity.

**Potensielle utfordringer vi ikke rakk eller fikk løst**

En av utfordringene vi støtte på, var knyttet til Sanity-integrasjonen. Vi hadde implementert Sanity og koblet det til Dashboard-siden, og selve Sanity-prosjektet med innhold var ferdigstilt. Vi hadde også laget komponenten for detaljvisning (EventDetails).

Det som gjensto, var å hente detaljer for et enkelt event via Sanity-API-et og vise dette i EventPage. Vi manglet bare fungerende API-tilkobling fra frontend til Sanity. På grunn av tidsbegrensninger og manglende gjennomgang av API-bruken i forkant, valgte vi å legge dette til side og heller gå for C-nivået.

**Bruk av AI-verktøy – ChatGPT**

**Eksempel 1: Søkeside med filtrering og autocomplete**

Prompt:

*"Jeg utvikler en kategoriside i et React-basert frontend-prosjekt hvor brukere skal kunne filtrere eventer basert på ulike kriterier som nøkkelord, dato, by, landkode og kategori (f.eks. musikk, sport eller teater). Jeg bruker Ticketmaster Discovery API for å hente relevante events..."*

A. Generert API-funksjon (JavaScript)

javascript

CopyEdit

const API\_KEY = 'oUxracsATFhH1Gkva2GRyOTWWWR7uOgl';

const BASE\_URL = 'https://app.ticketmaster.com/discovery/v2';

export async function fetchCategoryContent(filters) {

const query = new URLSearchParams({ apikey: API\_KEY, size: 20 });

if (filters.keyword) query.append('keyword', filters.keyword);

if (filters.city) query.append('city', filters.city);

if (filters.countryCode) query.append('countryCode', filters.countryCode);

if (filters.date) query.append('startDateTime', `${filters.date}T00:00:00Z`);

if (filters.classificationName) query.append('classificationName', filters.classificationName);

const url = `${BASE\_URL}/events.json?${query.toString()}`;

console.log('[Ticketmaster Fetch] Requesting:', url);

try {

const res = await fetch(url);

console.log('[Ticketmaster Fetch] Status:', res.status);

const data = await res.json();

console.log('[Ticketmaster Fetch] Response JSON:', data);

const events = data.\_embedded?.events || [];

const attractions = [];

const venues = [];

events.forEach(event => {

const eventAttractions = event.\_embedded?.attractions || [];

const eventVenues = event.\_embedded?.venues || [];

eventAttractions.forEach(attr => {

if (!attractions.find(a => a.id === attr.id)) {

attractions.push(attr);

}

});

eventVenues.forEach(venue => {

if (!venues.find(v => v.id === venue.id)) {

venues.push(venue);

}

});

});

return { events, attractions, venues };

} catch (error) {

console.error('[Ticketmaster Fetch] Error fetching data:', error);

return { events: [], attractions: [], venues: [] };

}

}

B. Bruk i React-komponent

javascript

CopyEdit

useEffect(() => {

fetchCategoryContent({ ...filters, classificationName })

.then(setData)

.catch((err) => console.error('[CategoryPage] Fetch error:', err));

}, [filters, classificationName]);

const onFilter = (e) => {

e.preventDefault();

const f = e.target;

setFilters(fv => ({

...fv,

date: f.date.value,

countryCode: f.country.value,

city: f.city.value

}));

};

const onSearch = (e) => {

e.preventDefault();

const f = e.target;

setFilters(fv => ({ ...fv, keyword: f.keyword.value }));

};

Kommentar:

Hele fetchCategoryContent()-funksjonen ble generert av AI, sammen med forslagene til bruk av useEffect, onFilter og onSearch. Vi har tilpasset noe selv, men strukturen og logikken kommer fra AI-forslaget.

Hvorfor AI ble brukt:

* Ticketmaster-dokumentasjonen er omfattende og tidkrevende å sette seg inn i.
* AI hjalp oss raskt med en korrekt og funksjonell løsning.
* Det sparte oss tid og ga forslag til god struktur og feilhåndtering.

**Eksempel 2: Like/Lagre-funksjon med Font Awesome**

**Prompt:**

*"Jeg lager en komponent i React som viser en attraksjon... Jeg vil bruke Font Awesome for ikonene. Kan du vise hvordan jeg importerer riktige ikoner og bytter mellom dem basert på en boolean?"*

Resultat:

javascript

CopyEdit

import { FontAwesomeIcon } from "@fortawesome/react-fontawesome";

import { faHeart as solidHeart } from "@fortawesome/free-solid-svg-icons";

import { faHeart as regularHeart } from "@fortawesome/free-regular-svg-icons";

// I JSX:

<FontAwesomeIcon icon={isSaved ? solidHeart : regularHeart} />

Kommentar:

Importen av ikonene og logikken for å bytte mellom dem basert på isSaved-prop ble generert av AI. Dette ble brukt i AttractionCard-komponenten for å indikere lagring/liking.

Hvorfor AI ble brukt:

* For å få hjelp med korrekt import og bruk av ulike ikon-varianter (solid vs. regular).
* For å implementere visuell tilbakemelding raskt og korrekt i React.
* Det sikret at vi brukte riktige pakker og dokumenterte fremgangsmåter.

**Oppsummering:**Vi valgte å levere på C-nivå da vi hadde alt klart for innlogging og sanntidsvisning av arrangementer, men valgte bort Sanity-integrasjonen grunnet tidsmangel og API-problemer. AI-verktøyet ChatGPT ble brukt aktivt for å hente kodeforslag og forstå dokumentasjon, noe som sparte oss mye tid og økte kvaliteten på løsningen.