

Eksamensdokument Gruppe 18

Vanskelighetsgrad:

Vi har valgt å gå for et C krav

Github-alias:

Markusol00 - Markus Olsen
Arshabba - Arshad Abba
Hedzpedz - Hedda Nilsen

Github lenke:

https://github.com/Hedzpedz/G18_eksamen

Redegjørelser

I oppgaveteksten sto det at all bruk av AI skal dokumenteres. Dette inkluderer prompter, resultater, hvor i koden det er lagt inn, samt hva koden gjør. I avsnittet "Bruk av kunstelig intelligens" har vi vedlagt vår bruk av AI under dette prosjektet. Vi har valgt å ha en struktur med skjermbilder av prompt og resultater, samt bruk av kode eller inspirasjon. Grunnen til dette er fordi vi var usikre på hvordan dette skulle dokumenteres, i og med at det ikke sto hvordan dette skulle gjøres i oppgaven.

Hva vi ikke rakk løse:

Vi rakk ikke å optimalisere UI i forhold til vår plan. Vi tenkte å designe vårt nettsted med fokus på universell utforming og ha et enda mer brukervennlig design for vår løsning. Styling av festivalkort på forsiden har endret seg og det har blitt introdusert en bug vi ikke rekker å finne ut av rett før innlevering.

Bruk av kunstelig intellegens:

1) *Bruk av AI: Kreve input før trykk.*

Prompt og resultater av AI generert svar:

<https://chatgpt.com/share/682a523c-ad4c-8000-b44f-6e4df26ee435>

Bruk i kode:

(Denne kodebiten brukes i komponenten CategoryPage)

```
const isValid = City !== "" && Country !== "";
```

Forklaring av kode og begrunnelse på bruk av AI:

Sjekk om 'city' og 'country' har blitt fylt ut. Vi ønsket at man skal vite om knappen skal enten være deaktivert eller aktivert. Hvis begge er true kan man da gjøre et søk.

2) Bruk av AI: Funksjon for klikkbare ikoner

Prompt:

Jeg skal lage en nettside i forhold til billettsalg. Jeg ønsker jeg å bytte mellom to ikoner fra jeg har lokalt når en bruker trykker på et button-element. Kan du gi meg ideer på hvordan jeg kan lage en funksjon for det? (forelderkomponenten heter CategoryPage og props blir sendt herfra evt).
Deler av koden min:

```
import regularHeartIcon from './website_images/heart-regular.svg';

export default function VenueCard({ name, image, country, city }) {
  return (
    <article className="event-card">
      <img src={image} alt={name} />
      <h2>{name}</h2>
      <button className="heart-button">
        <img
          src={regularHeartIcon}
          alt="Legg til i ønskeliste"
          className="heart-icon"
        />
      </button>
      <p>{country}</p>
      <p>{city}</p>
    </article>
  );
}
```

Resultater generert av AI:

Ja! Her er hvordan du kan gjøre det:

🔗 Legg til lokal state og funksjon for å bytte ikon

jsx

Kopier Rediger

```
const [isSaved, setIsSaved] = useState(false);

const handleClick = () => {
  setIsSaved(!isSaved);
};

const heartIcon = isSaved ? solidHeartIcon : regularHeartIcon;
```

Bruk i kode:

```
export default function VenueCard({ id, name, image, country, city,
  isSaved, solidHeartIcon, regularHeartIcon }) {

  const heartIcon = isSaved ? solidHeartIcon : regularHeartIcon;

  return (
    <article className='event-card'>
```

Forklaring av kode og begrunnelse på bruk av AI:

Inspirasjon fra koden brukes i komponenten VenueCard. Denne kodebiten sjekker om state variabelen isSaved er true. Dersom den er "true" returnerer den ikonet solidHeartIcon, om den ikke er det returnerer den et ikke-helt ikon. Nåværende kode inkluderer ikke const [isSaved, setIsSaved] = useState(false), pga endringer som ble gjort senere i eksamensprosessen. Disse endringene vises på bildet under 'Bruk i kode'. Selve variabelene og handleClick funksjonen bidro med inspirasjon og videreutvikling av koden. Bildet under viser hvordan vi inkluderte konstantene og pilfunksjon for klikk.

```
const handleClick = (id) => {
  let updated;

  if(wishlist.includes(id)){
    updated = wishlist.filter((item) => item !==id);
  } else {
    updated = [...wishlist, id]
  }

  setWishlist(updated);
  localStorage.setItem("wishlist", JSON.stringify(updated));
};
```

3) Bruk av AI: Aria-label for universell utforming.

Prompt og svar:

Jeg ønsker at nettsiden min er mere tilgjengelig i forhold til universell utforming. For å oppnå det ønsker jeg muligheten til å ha et aria-label som hjelper skjermleseren med å finne ut om en funksjon er aktivert eller ikke. Har du tips til hvordan dette kan gjøres?

Jal! For å hjelpe skjermlesere med å forstå om en funksjon er aktivert eller ikke, kan du bruke en kombinasjon av `aria-pressed` og et dynamisk `aria-label`. Her er et enkelt eksempel:

```
jsx
<button
  onClick={handleToggle}
  aria-pressed={isActive}
  aria-label={
    isActive
      ? 'Deaktiver varsler'
      : 'Aktiver varsler'
  }
>
</button>
```

Dette gjør følgende:

- `aria-pressed` sier til skjermleseren om knappen er aktiv (`true`) eller ikke (`false`).
- `aria-label` forklarer hva knappen gjør – og oppdateres etter tilstanden.

Skjermlesere vil da kunne si noe som: «Aktiver varsler, knapp» eller «Deaktiver varsler, knapp», alt etter hva som gjelder. Dette gir god støtte for universell utforming.

Kodebiten under sjekker om funksjonen isSaved er aktivert eller ikke, dersom den er det, så leser skjermleseren opp "Fjern \${name} fra ønskelisten". Dersom isSave ikke er aktivert så leser

den opp “Legg til $\{name\}$ i ønskelisten ”Koden ble brukt under ønskeliste-knappene i følgende komponenter, og ser like ut:

- EventCard.jsx
- VenueCard.jsx
- AttractionCard.jsx

4) *Bruk av AI: Lagring av ID i localStorage*

Prompt:

Jeg driver å lager en funksjon som setter et event i en ønskeliste dersom en bruker trykker på en like-knapp. Jeg er i gang og har laget følgende:
Ustate for wishlist - tom array
Funksjon for å toggle ønskeliste, følgende:

```
const handleWishlistClick = (id) => {  
  let updated;  
  
  if (wishlist.includes(id)) {  
    updated = wishlist.filter((item) => item !== id);  
  } else {  
    updated = [...wishlist, id]  
  }  
}
```

I komponenten VenueCard (kortkomponent) har jeg også denne for å vise riktig ikon i.
const heartIcon = isSaved ? solidHeartIcon : regularHeartIcon;

Jeg ønsker nå å lagre ID i localStorage når en bruker trykker på ikonet. Har du tips til hvordan det kan løses?

Resultater generert av AI:

Ja, absolutt! Når du ønsker å lagre ID-er i `localStorage` for å huske ønskelisten mellom sidebesøk, anbefaler jeg å bruke `useEffect`. Her er hvordan du løser det i to enkle steg:

✓ 1. Lagre ønskelisten i `localStorage` når den endres

Legg til denne `useEffect`-hooken i komponenten din (f.eks. `CategoryPage`):

```
js  
  
useEffect(() => {  
  localStorage.setItem("wishlist", JSON.stringify(wishlist));  
}, [wishlist]);
```

• Hva den gjør:

- Hver gang `wishlist` oppdateres (brukeren klikker hjerteknappen), kjøres denne effekten.
- Den konverterer arrayen til en streng med `JSON.stringify()` og lagrer den i `localStorage` under nøkkelen `"wishlist"`.

✓ 2. Hent ønskelisten fra `localStorage` når siden lastes

Legg til denne `useEffect`-hooken også:

```
js  
  
useEffect(() => {  
  const savedWishlist = localStorage.getItem("wishlist");  
  if (savedWishlist) {  
    setWishlist(JSON.parse(savedWishlist));  
  }  
}, []);
```

• Hva den gjør:

- Når komponenten rendres første gang (`[]` som dependency), sjekker den om noe er lagret.
- Hvis det finnes noe, parser den strengen tilbake til et array med `JSON.parse()` og setter det som din `wishlist`.

🔴 Resultat:

- Brukeren klikker på hjertet → `wishlist` oppdateres → `localStorage` oppdateres.
- Ved neste besøk hentes ID-ene automatisk fra lagringen.

Vil du jeg skal vise hele `CategoryPage`-komponenten `4in` med dette lagt inn også?

Bruk i kode:

Disse kodelinjene brukes i komponenten CategoryPage.jsx.

```
useEffect(() => {  
  const savedWishlist = localStorage.getItem("wishlist");  
  
  if (savedWishlist) {  
    setWishlist(JSON.parse(savedWishlist));  
  }  
}, []);
```

```
const [wishlist, setWishlist] = useState([]);  
  
useEffect(() => {  
  const savedWishlist = localStorage.getItem("wishlist");  
  
  if (savedWishlist) {  
    setWishlist(JSON.parse(savedWishlist));  
  }  
}, []);
```

Forklaring av kode og begrunnelse på bruk av AI:

En useEffect som oppdateres hver gang "wishlist" endrer seg. .json metoden gjør arrayen om til en string, slik at den kan brukes. Begrunnelse: Ønsket å lagre ønskelisten i localStorage for å ha varige ønskelister. Under utviklingsprosessen slet vi med og parsing av json, noe som resulterte i at vi spurte ChatGPT hva neste steget var.

5) Bruk av AI: Oversetting til engelske kategorinavn.

Prompt:

Jeg skal lage en nettside, hvor navigasjonsbaren har tre kategorier 'sport', 'musikk' og 'teater'. Vi følger en skisse som er laget på norsk, og ønsker å hente ut fra API'et som er på engelsk. I utviklinga trenger den engelske oversettelsen for å hente ut data. Vi trenger også segment'id for sjangeren. Er det en måte å kunne hente ut all den informasjonen på, uten å ha all informasjon i ulike variabler?

Resultat generert av AI:

Ja, du kan løse dette elegant ved å bruke et strukturert objekt som kobler sammen de norske kategorinavnene med nødvendig API-informasjon – både den engelske oversettelsen og `segmentId`. På den måten slipper du å ha alt spredd i separate variabler.

Her er et forslag til hvordan du kan strukturere det:

```
js                                                                    Kopier  Rediger

const classificationMap = {
  sport: {
    segmentId: "eksempelID",
    classificationName: "Sports",
  },
  musikk: {
    segmentId: "eksempelID",
    classificationName: "Music",
  },
  teater: {
    segmentId: "eksempelID",
    classificationName: "Theatre & Arts",
  },
};
```

🔗 Forslag til bruk:

Når brukeren velger en kategori i navigasjonen (f.eks. "musikk"), kan du enkelt hente ut både `segmentId` og riktig engelsk navn fra `classificationMap[musikk]`, og bruke det i API-kallet ditt.

Dette gjør det lett å holde frontend på norsk, samtidig som du henter ut riktige engelske verdier uten å duplisere data eller rote med oversettelser i flere steder i koden.

📄 🔍 🔄 ⌂ ↻

Bruk i kode:

```
const classificationMap = {
  sport: {
    segmentId: "KZFzniwnSyZfZ7v7nE",
    classificationName: "Sports",
  },
  musikk: {
    segmentId: "KZFzniwnSyZfZ7v7nJ",
    classificationName: "Music",
  },
  teater: {
    segmentId: "KZFzniwnSyZfZ7v7na",
    classificationName: "Theatre",
  },
};
```

Forklaring av kode og begrunnelse av AI:

Dette er et classification-map som er et JavaScript-objekt. Den har tre nøkler som gir oss muligheten til å hente ut ID'en og klassifiseringsnavnet på arrangementer, attraksjoner og artister. Den har tre nøkler. Vi ønsket å finne en måte for å oversette de norske nøklene til engelsk slik at vi kunne hente data ut fra API'et. Vi trengte også segmentID. Vedkommende som laget denne kode-biten har ikke hatt JavaScript tidligere.

6) *Bruk av AI: Hjelp med Eventpage - fette ned festivalkort.*

Prompt: (Rekkefølge av skjermbilder går fra venstre til høyre)

Hei.

Vår gruppeeksamen hvor vi utvikler en nettside som skal ta i bruk Ticketmaster sitt developer V2 API. Jeg har endelig funnet ut av et problem med koden min.

Eventpage er ment for 4 festivaler i Home når man trykker på artikkelkortene i Home. Så i eventpage skal det ikke bli tatt inn noen events direkte, men attractions til festivalene. Altså det er helt feil logikk i nesten hele eventpage. En festival blir hentet inn fra API via attractions. Jeg trenger hjelp med å refactorere eventpage slik at den kan hente ut attractions istedenfor event. I en annen fil har vi categorypage, det er denne eventpage har tatt inn som event, noe som er til dels riktig. I kategoriene som tilhører categorypage skal inføen komme direkte ut under bildet, ikke som les mer(dette er feilen, da eventpage har blitt satt opp til å håndtere innholdet på categorypage, men som egentlig skal håndtere home).

Sett deg inn i dokumentasjonen fra Ticketmaster API: <https://developer.ticketmaster.com/products-and-docs/apis/discovery-api/v2/>

Kan du hjelpe meg med å endre funksjonaliteten i
EventPage slik at den tar inn attraction istedenfor event?
Her er koden jeg har nå:

```
export default function Home() {
  const [festivalItems, setFestivalItems] = useState(null);
  const [cityEvents, setCityEvents] = useState([]);
  const [selectedCity, setSelectedCity] =
    useState("Oslo");

  useEffect(() => {
    const wait = (ms) => new Promise(resolve =>
      setTimeout(resolve, ms));

    const fetchFestivalAttractions = async () => {
      const results = [];
      for (const term of FESTIVAL_SEARCH_WORDS) {
        await wait(100);
        try {
          const apiUrl = `https://app.ticketmaster.com/discovery/
v2/attractions.json?apikey=${API_KEY}
&preferredCountry=NO&keyword=${term}
&size=1&classificationName=Music,Festival`;

          const response = await fetch(apiUrl);
          const jsonData = await response.json();
          if (jsonData?.embedded?.attractions?.[0]) {
```

Kan du hjelpe meg med å endre funksjonaliteten i EventPage slik at den tar inn attraction istedenfor event? Her er koden jeg har nå:

```
home:
import { useEffect, useState } from "react";
import { Link } from "react-router-dom";
import EventCard from "../components/EventCard";
import CityEventCard from "../components/
CityEventCard";
import "../styles/CategoryPage.scss";
import "../styles/Home.scss";
```

```
const FESTIVAL_SEARCH_WORDS = [
  "Tons of Rock",
  "Findings festival",
  "Neon festival",
  "Skeikampenfestivalen"
];
const API_Key =
  "AnOGfh3JYmKpW5rJlqCetXQuRadlfUhp";
```

```
export default function Home() {
  const [festivalItems, setFestivalItems] = useState(null);
  const [cityEvents, setCityEvents] = useState([]);
  const [selectedCity, setSelectedCity] =
```

```
</>
);
}

eventpage:
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import "../styles/EventPage.scss";
import ArtistCard from "../components/ArtistCard";
```

```
export default function EventPage() {
  const [event, setEvent] = useState([]);
  const { id } = useParams();
  const [Start, setStart] = useState("Unknown");
  const [End, setEnd] = useState("Unknown");
  const [genre, setGenre] = useState(null);
  const [artistGenre, setArtistGenre] = useState(null);
  useEffect(() => {
    async function fetchEvent() {
      try {
        const response = await fetch(
          `https://app.ticketmaster.com/discovery/v2/events/${id}?
          apiKey=AnOGfh3JYmKpW5rJlqCetXQuRadlfUhp`
        );
        const json = await response.json();
        setEvent(json);
```

```
setEvent(json);
} catch (error) {
  console.error("Fetch error:", error);
}
}

fetchEvent();
}, [id]);

useEffect(() => {
  if (event?.classifications?.[0]?.genre?.name) {
    setGenre(event.classifications[0].genre.name);
  }
  const attraction = event?._embedded?.attractions?.[0];
  const artistGenre = attraction?.classifications?.
  [0]?.genre?.name;
  if (artistGenre) {
    setArtistGenre(artistGenre);
  }
}, [event]);
```

```
useEffect(() => {
  const start = event.sales?.public?.startDateTime;
  const end = event.sales?.public?.endDateTime;
  if (start) {
    setStart(new Date(start).toLocaleString("no-NO"));
```

```
name={event.name}
image={event?.images[0].url}
genre={artistGenre}
/>
)))
</>
):(
<p>Ingen arrangementer funnet.</p>
))
</article>
</section>
</>
);
}
```

✦ Vis tenking ▼

Ok, basert på det du beskrev, her er et forslag til hvordan `EventPage.js` kan refaktoreres for å hente en "attraction" (som en festival) og deretter hente relaterte "events" (som festivalpass eller spesifikke dager/konsserter på festivalen) og artister knyttet til disse eventene.

JavaScript

```
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import "../styles/EventPage.scss";
```

Resultat fra Ai:

JavaScript



```
import { useEffect, useState } from "react";
import { useParams } from "react-router-dom";
import "../styles/EventPage.scss";
import ArtistCard from "../components/ArtistCard";
import EventCard from "../components/EventCard"; // Antar du har en EventCard for å vise eventene

export default function EventPage() {
  const [event, setEvent] = useState([]); // Burde nok kalles attractionDetails eller event
  const { id } = useParams(); // Dette er attractionId fra Home linken
  // Start/End for selve festivalen (attraction) er kanskje mindre relevant enn for eventene
  // const [Start, setStart] = useState("Unknown");
  // const [End, setEnd] = useState("Unknown");
  const [genre, setGenre] = useState(null); // Sjanger for festivalen (attraction)
  const [artistGenre, setArtistGenre] = useState(null); // For å vise en felles sjanger for artistene
  const [tickets, setTickets] = useState([]); // For å lagre events (festivalpass, dager etc.)
  const [artists, setArtists] = useState([]); // For å lagre artister hentet fra eventene

  // Hent detaljer om selve festivalen (Attraction)
  useEffect(() => {
    async function fetchEvent() { // Bør kalles fetchAttraction
      try {
        const response = await fetch(
          `https://app.ticketmaster.com/discovery/v2/attractions/${id}.json?apikey=${API_KEY}`
        );
        const json = await response.json();
        setEvent(json); // Setter festival (attraction) detaljer
      } catch (error) {
        console.error("Fetch error:", error);
      }
    }
    fetchEvent();
  }, [id]);
```

```
    fetchEvent();
  }, [id]);

  // Hent events (f.eks. festivalpass) og deretter artister fra disse events
  useEffect(() => {
    async function fetchTickets() { // Bør kalles fetchAssociatedEventsAndArtists
      if (!id) return;
      try {
        // 1. Hent events knyttet til festivalen (attractionId)
        const response = await fetch(
          `https://app.ticketmaster.com/discovery/v2/events.json?apikey=${API_KEY}&attractionId=${id}`
        );
        const json = await response.json();
        const events = json?._embedded?.events || [];
        setTickets(events); // Lagrer listen over events (festivalpass, dager etc.)

        // 2. Hent artister fra det første eventet som eksempel
        // eller iterer gjennom alle events for å samle alle artister.
        // Her filtreres også hovedattraksjonen (selv festivalen) bort fra artistene
        if (events.length > 0) {
          // Samle artister fra alle events, unngå duplikater og ikke inkludere hovedattraksjonen
          const allArtists = [];
          const artistIds = new Set();

          events.forEach(singleEvent => {
            const eventAttractions = singleEvent?._embedded?.attractions || [];
            eventAttractions.forEach(a => {
              if (a.id !== id && !artistIds.has(a.id)) { // Ikke legg til festivalen
                allArtists.push(a);
                artistIds.add(a.id);
              }
            });
          });

          setArtists(allArtists);
        }
      } catch (error) {
        console.error("Fetch error:", error);
      }
    }
    fetchTickets();
  }, [id]);
```

```

    });
    setArtists(allArtists);

    // Sett en felles artistGenre basert på den første artisten funnet (kan
    if (allArtists.length > 0 && allArtists[0]?.classifications?.[0]?.genre?.name) {
      setArtistGenre(allArtists[0].classifications[0].genre.name);
    } else {
      setArtistGenre(null); // Tilbakestill hvis ingen artister eller sjanger
    }
  } else {
    setArtists([]); // Ingen events, så ingen artister
    setArtistGenre(null);
  }

} catch (error) {
  console.error("Ticket fetch error:", error);
  setTickets([]);
  setArtists([]);
}
}

fetchTickets();
}, [id]); // Kjører når attractionId (id) endres

// Effekt for å sette sjanger for festivalen (attraction)
useEffect(() => {
  if (event?.classifications?.[0]?.genre?.name) {
    setGenre(event.classifications[0].genre.name);
  }

  // Den opprinnelige logikken for artistGenre basert på event._embedded.attractions
  // er mindre relevant her hvis 'event' er selve festivalen.
  // Artist sjanger settes nå i fetchTickets effekten.
  // const attraction = event?._embedded?.attractions?.[0];
  // const currentArtistGenre = attraction?.classifications?.[0]?.genre?.name;

```

```

  // Den opprinnelige logikken for artistGenre basert på event._embedded.attractions
  // er mindre relevant her hvis 'event' er selve festivalen.
  // Artist sjanger settes nå i fetchTickets effekten.
  // const attraction = event?._embedded?.attractions?.[0];
  // const currentArtistGenre = attraction?.classifications?.[0]?.genre?.name;
  // if (currentArtistGenre) {
  //   setArtistGenre(currentArtistGenre);
  // }

}, [event]); // Kjører når festival (attraction) detaljer endres

// Denne effekten for Start/End datoer fra 'event' (som er attraction)
// er kanskje ikke det du vil vise for billettsalg, da attractions ikke alltid har
// Salgsdatoer er typisk på 'event'-objekter (festivalpass).
// Vurder å fjerne eller flytte denne logikken til der du viser individuelle 'tickets'
/*
useEffect(() => {
  const start = event.sales?.public?.startDateTime;
  const end = event.sales?.public?.endDateTime;

  if (start) {
    setStart(new Date(start).toLocaleString("no-NO"));
  }
  if (end) {
    setEnd(new Date(end).toLocaleString("no-NO"));
  }
}, [event]);
*/
// Fjerner duplikat useEffect for Start/End

return (
  <>
    <section className="festival-header"> { /* Endret klassenavn for klarhet */ }
    <img src={event?.images?.[0]?.url} alt={event?.name} />
  </>

```

```

        tickets.map(ticket => ( // 'ticket' her er et 'event' objekt
        <EventCard // Bruk EventCard eller en dedikert komponent for å vise
            key={ticket.id}
            name={ticket.name} // F.eks. "Festivalpass" eller "Dagspass Lørdag"
            image={ticket?.images?.[0]?.url}
            date={ticket?.dates?.start?.localDate}
            // Du vil kanskje legge til mer info her: tid, sted (venue), pris
            // salesStart={ticket.sales?.public?.startDateTime}
            // venueName={ticket._embedded?.venues?.[0]?.name}
            link={ticket?.url} // Direkte link til dette eventet/billettypen
        />
        ))
    ) : (
        <p>Ingen festivalpass eller billetter funnet.</p>
    )
}
</article>
</section>

{/* Seksjon for Artister */}
<section className="festival-artists">
    <article>
        <h2>Artister</h2>
        {artists.length > 0 ? (
            artists.map((a) => ( // 'a' her er et 'attraction' objekt (en artist)
                <ArtistCard
                    key={a?.id}
                    name={a?.name}
                    image={a?.images?.[0]?.url}
                    // Sender med felles artistGenre, eller du kan hente spesifikk sjanger
                    // genre={a?.classifications?.[0]?.genre?.name || artistGenre}
                    genre={artistGenre || a?.classifications?.[0]?.genre?.name}
                />
            ))
        ) : (
            <p>Ingen artister funnet for denne festivalen.</p>
        )
    }
</section>

```

```

</section>

{/* Seksjon for Artister */}
<section className="festival-artists">
    <article>
        <h2>Artister</h2>
        {artists.length > 0 ? (
            artists.map((a) => ( // 'a' her er et 'attraction' objekt (en artist)
                <ArtistCard
                    key={a?.id}
                    name={a?.name}
                    image={a?.images?.[0]?.url}
                    // Sender med felles artistGenre, eller du kan hente spesifikk sjanger
                    // genre={a?.classifications?.[0]?.genre?.name || artistGenre}
                    genre={artistGenre || a?.classifications?.[0]?.genre?.name}
                />
            ))
        ) : (
            <p>Ingen artister funnet for denne festivalen.</p>
        )
    }
</article>
</section>
</>
);
}

```

Forklaring av kode og begrunnelse av AI:

Den opprinnelige EventPage-koden var laget for å hente ett enkelt event fra Ticketmaster API-et, noe som ikke fungerte for festivalsidene. En festival er en attraction som typisk har flere underliggende events (som dagspass eller enkeltkonserter) og et bredere artistprogram. Den gamle koden klarte ikke å vise dette da den var bygget opp til å kun vise et event. Etter flere dager hvor vi ikke forsto problemet, falt endelig brikkene på plass helt mot innspurten.

Med hjelp av AI ble koden derfor grunnleggende endret. Den nye versjonen henter nå først selve festivalen (som en attraction). Deretter gjør den et nytt API-kall for å finne alle tilknyttede events (billetter/dagspass) for denne festivalen. Til slutt hentes artistene ut fra disse events.

AI ble brukt i denne prosessen fordi den opprinnelige logikken var helt feil for det vi ønsket å oppnå med festivalsidene. Det krevdes en stor omstrukturering av både hvordan data hentes og hvordan siden er bygget opp. Gitt denne kompleksiteten, og at vi fant ut av problemet noen timer før innlevering, brukte vi AI som et verktøy. AI hjalp til med å generere et utkast til den nye logikken for datainnhenting og hvordan de ulike delene (festival, billetter, artister) kunne settes sammen og vises på siden.

7) Bruk av AI: Oppsett av knapp til form

I denne prompten ble AI brukt for å hjelpe meg sette opp en knapp til form. Brukte AI for jeg var for lat til å sjekke MDN sine hjelpesider for utviklere, og hadde glemt hva syntaksen for dette var:

<https://chatgpt.com/share/682a525c-bef8-8000-902d-cbeb08247446>

8) Bruk av AI: Kode for filtering eller HTML

Denne prompten ble opprinnelig brukt feil til å filtrere i eksisterende events (grensesnittet), men det meste ble fjernet. Promptene ble gjort for å få ut boilerplate kode for f.eks filtrering eller HTML.

<https://chatgpt.com/share/682a52a0-f144-8000-8764-3dcef632faa8>

9) Bruk av AI: Spørsmål til AI

Fikk litt panikk da jeg trodde jeg hadde pushet opp feil og merget feil i en pull request og måtte spørre AI

<https://chatgpt.com/share/682a52d8-3db8-8000-8b5c-ba8bb66bcff6>