12/11-2025
Hold WebH125-1

Gittes Glamping

Skrevet af: Oleg Trojan

GitHub: Link

Login:
Guest Email: guest_test@example.com
Kode: guest123

Vurdering af egne indsats

Dette projekt, **"Gittes Glamping"**, er udviklet med React, hvilket har været en positiv erfaring for mig. Projektet er stylet med TailwindCSS. For en mere overskuelig struktur af klasserne brugte jeg clsx, hvilket gjorde det lettere at forstå, hvilke styles der blev anvendt på de enkelte elementer. Derudover anvendte jeg delvist styled-components — et populært bibliotek til React, som gør det muligt at skrive CSS direkte i JavaScript-filer og skabe "stylede komponenter".

For eksempel brugte jeg styled-components i BurgerMenu, da der her var komplekse CSS-animationer og transformationer, som er nemmere at beskrive direkte i komponenten. Det gjorde det også let at anvende dynamiske styles via props, såsom **$isOpen** eller **$scrolled**, hvilket ville være mere besværligt med Tailwinds utility-klasser. Denne tilgang gør stilen modulær, overskuelig og isoleret inden for komponenten.

Ved planlægning af mit arbejde brugte jeg Git Project, hvor opgaverne blev opdelt i kolonner: Backlog, In Progress, In Review, Ready, Done. Overordnet var der ingen større forsinkelser, men den største udfordring var LoginForm, specielt arbejdet med token i localStorage. Problemet opstod på grund af forkert adgang til token i serverens svar (data.data.token i stedet for data.token).

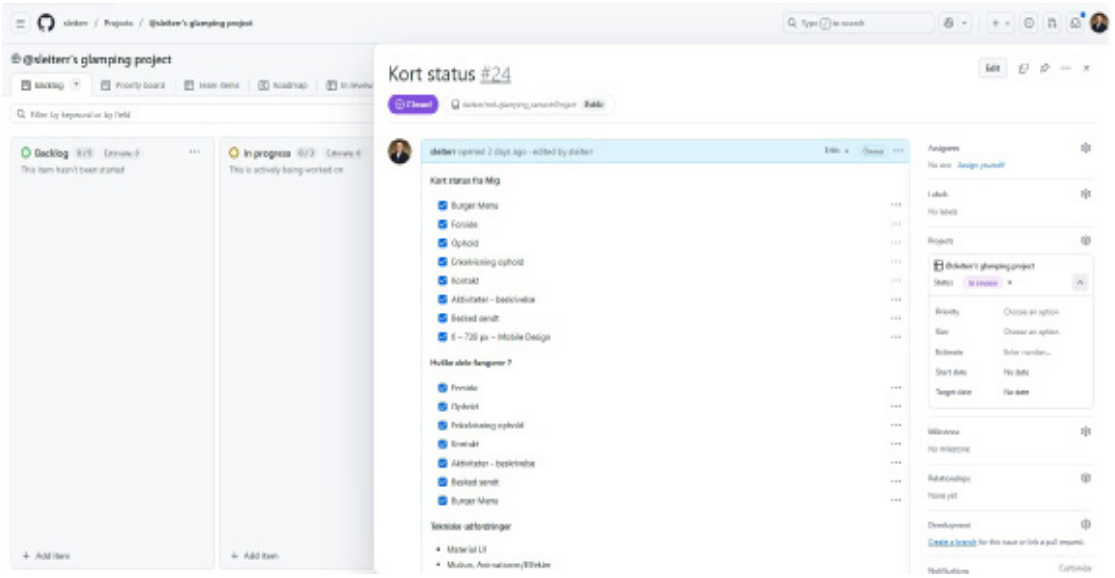
Projektet har en klar mappestruktur og følger en komponentbaseret tilgang. For eksempel har jeg lavet egendefinerede Section og Button komponenter for nem genbrug, hvilket gav mere tid til debugging og styling. Eneste bemærkning er, at jeg ikke brugte Motion eller UIMaterial — jeg fokuserede i stedet på designet for at overføre layoutet til en levende side og for at arbejde grundigt med data fra backend.

Alt i alt har projektet været interessant og lærerigt, da jeg har forbedret mine færdigheder med TailwindCSS og komponentstruktur, og vigtigst af alt, har jeg succesfuldt overført designet til en fungerende side.

Forklaring af arbejdsprocessen

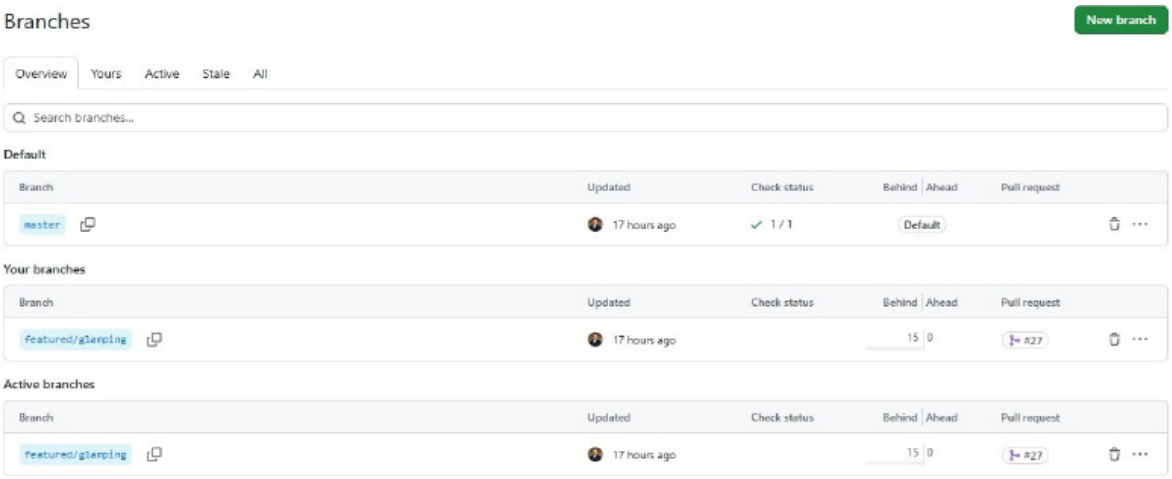
Min tilgang til rapporten adskiller sig lidt fra, hvordan jeg ideelt set gerne ville gøre det. I fremtiden planlægger jeg at dokumentere alle arbejdsfaser løbende, for eksempel i slutningen af hver arbejdsdag efter dagens arbejde. I denne rapport vil jeg forsøge at forklare min handlingsplan samt beskrive de metoder og principper, jeg har anvendt i udviklingen af dette projekt.

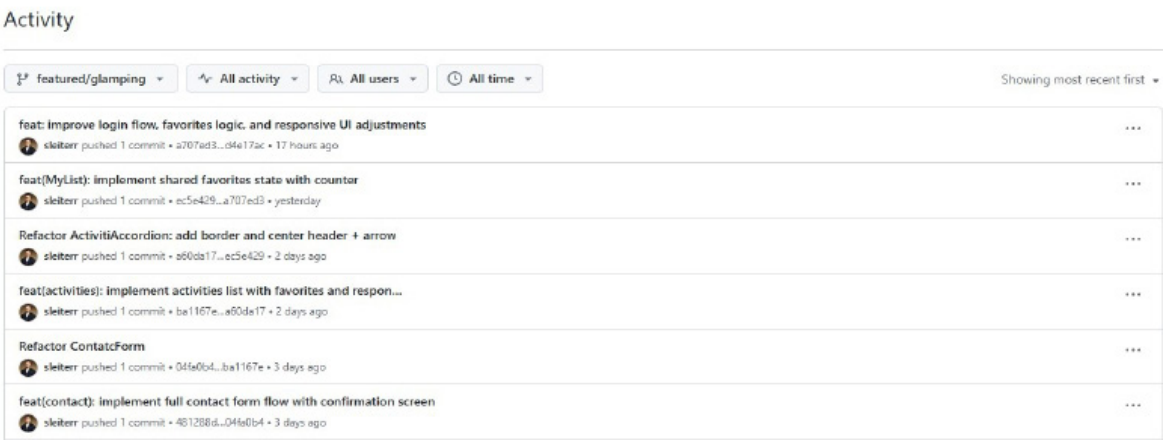
Jeg startede med at planlægge arbejdet i GitHub Project. Planlægningsprincippet bestod i at opdele arbejdsprocessen i daglige opgaver og maksimere tiden for at overholde tidsplanen. Jeg vurderer, at jeg lykkedes med dette, og jeg kunne følge planen med minimale forsinkelser.



Efter initialiseringen af projektet med Vite begyndte jeg arbejdet med dets filstruktur. I mappen components/ er placeret de vigtigste genanvendelige komponenter, som bruges i forskellige dele af websiden. Mappen pages/ indeholder de primære sider på sitet, hvor hver side svarer til en bestemt rute, der er defineret i filen App.jsx ved hjælp af React Router

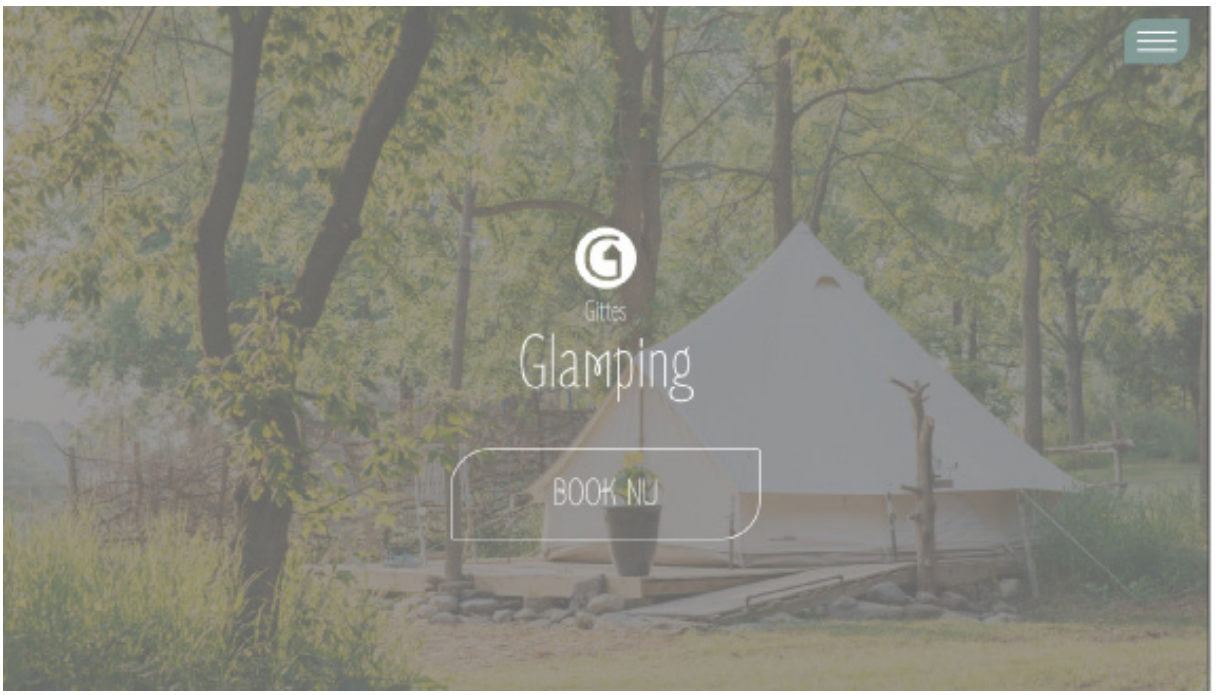
Efter hver fuldført opgave lavede jeg commit og push til det globale repository **featured/glamping**, hvor jeg detaljeret beskrev næsten hele arbejdsprocessen i commit-beskeden.





Beskrivelse af kodemetoden

I denne del af projektet vil jeg gerne fremhæve den valgte tilgang til strukturering af komponenter og arbejdet med indholdet inden i dem. Som eksempel tages **Hero-sektionen,** som er det primære visuelle element på forsiden (Forside). Denne sektion er ansvarlig for visning af logo, sidens navn og en knap til kontaktformularen. Komponenten er implementeret efter mobile-first-princippet med responsivt layout og baggrundsbillede, hvilket sikrer korrekt visning på forskellige enheder.



Valgt tilgang:

For opbygningen af indholdet i Hero har jeg oprettet en separat funktionskomponent kaldet HeroItem.

```
const HeroItem = () => {
  return (
    <div>
      {heroText.map((item) => (
        <div
          key={item.id}
          className="flex flex-col items-center justify-center"
        >
          {item.element}
          <h2 className={item.subtitleClass}>{item.subtitle}</h2>
          <h1 className={item.titleClass}>{item.title}</h1>
        </div>
      ))}
    </div>
  );
};
```

som mapper data fra arrayet **heroText**. Hvert element i arrayet indeholder information om en specifik tekstblok (logo, undertitel, titel) samt tilhørende style-klasser.

```
const heroText = [
  {
    id: 1,
    element: <img src={logo} alt="Logo" className="w-20 h-auto mb-2" />,
    subtitle: "Gittes",
    title: "Glamping",
    subtitleClass: "font-zen text-secondary text-4xl",
    titleClass: "font-zen text-primary text-8xl",
  },
];
```

Argumentation for valget:

1. Let opdatering af indhold:
Alt indhold er samlet i det strukturerede array heroText. Dette gør det hurtigt at ændre tekst, farver eller endda rækkefølgen af elementer uden at ændre JSX-strukturen. Hvis det i fremtiden bliver nødvendigt at hente disse data fra et API, er arrayets format allerede forberedt til dette.

2. Nem skalering:
Hvis der skal tilføjes flere hero-blokke (f.eks. i en slider eller på forskellige sider), kan man blot tilføje nye objekter i heroText, og komponenten HeroItem vil automatisk rendre dem.

3. Adskillelse af logik og struktur:
Hovedkomponenten Hero har ansvar kun for sektionens struktur (layout, baggrund, overlay), mens HeroItem står for indholdet. Dette gør koden mere ren, logisk opdelt og lettere at vedligeholde.

4. Fleksibilitet i styling:
Tailwind-klasserne gives gennem objektets felter (subtitleClass, titleClass), hvilket gør det muligt at tilpasse stilen uden at duplikere komponenter. Dette følger principperne om reusability og separation of concerns.

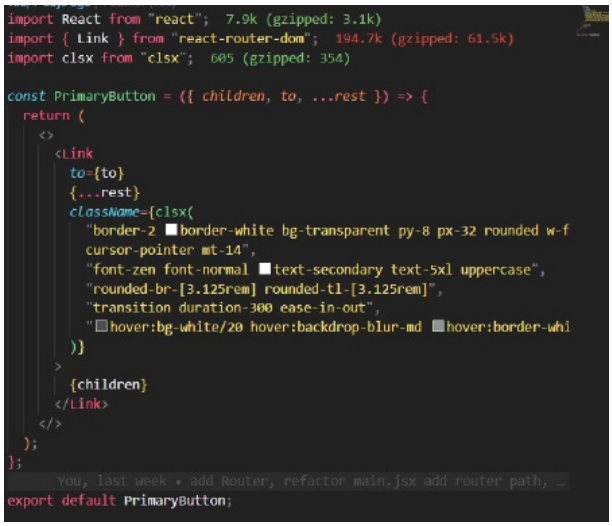
5. Forberedelse til dynamisk indhold:
Selvom indholdet i øjeblikket ikke hentes fra et API, gør datastrukturen og mappingen komponenten klar til fremtidig integration med backend, hvor heroText kan hentes dynamisk.

Tilpasning

Under udviklingen af projektet besluttede jeg at oprette flere tilpassede grund-komponenter, som kan genbruges i forskellige dele af siden.
Dette gør det muligt at bevare en ensartet stil, reducere kode-duplikation og forenkle fremtidig vedligeholdelse. Som eksempel kan man fremhæve to sådanne
komponenter: PrimaryButton og IntroSection.

Komponent PrimaryButton

Beskrivelse:
Komponenten PrimaryButton bruges som hovednavigationsknap på siden. Den er implementeret baseret på Link fra biblioteket react-router-dom til navigation mellem sider. Knappen har sit eget sæt Tailwind-klasser, som sikrer et genkendeligt udseende (afrundede hjørner, kant og hover-effekt).



Argumentation for valget:

1.Ensartet stil:
Én universel komponent sikrer, at alle knapper på siden ser ens ud, uanset hvor de bruges.

2.Let vedligeholdelse:
Hvis farve, størrelse eller hover-effekter skal ændres, er det nok at foretage ændringer ét sted.

3.Genanvendelighed:
Komponenten kan bruges i enhver sektion ved blot at give tekst eller et child-element via children og sti via to.

4.Fleksibilitet:
Brugen af clsx gør det nemt at kombinere eller udvide stilarter efter behov.

Komponent IntroSection

Beskrivelse: IntroSection er en sektion, som jeg blandt andet har brugt til introduktionsblokken på forsiden (Forside), i henhold til kravene for "Introduktion til **Gittes Glamping"**. Komponenten er ansvarlig for visning af indholdet i en fast stil med baggrundsbillede, afrundede hjørner og centreret justering.

Argumentation for valget:

1.Ensretning af sektionernes layout:
Komponenten gør det muligt at oprette forskellige sektioner på siden med ensartede strukturelle egenskaber (margener, positionering, hjørneradius, baggrund) uden at gentage kode.

2.Fleksibilitet:
Brugen af props className og innerClassName gør det muligt at tilpasse udseendet for hver enkelt side uden at ændre selve komponenten.

3.Understøttelse af "composition"-princippet:
Gennem children kan man indsætte enhver form for indhold, hvilket gør sektionen universel.

4.Modularitet:
Koden er logisk opdelt, hvilket forenkler projektstrukturen og gør komponenterne uafhængige af hinanden.