Svendeprøve

Det Utrolige Teater

Frederik Skrubbeltrang

H1WE010122

Brugernavn til login: fres

Password til login: frederik

https://github.com/fredeskrubba/svendeproeve

Indhold

Vurdering af egen indsats

I løbet af prøven er jeg efter egen mening kommet godt i mål med opgaven. Jeg har fået opsat og stylet alle de givne undersider, og gjort dem funktionelle på diverse skærm størrelser. Jeg har lavet det vigtigste funktioner på siden, som login systemet, og har vidst at jeg kan bruge diverse metoder i http request, som vist ved at jeg har lavet GET, DELETE og POST metoder.

Kodemæssigt har jeg vist min kunnen indenfor html, css og React, ved at jeg har sat elementer semantisk op, ved brug af diverse html tags som section, article, nav og footer, i stedet for at bruge div. Stylingen er skrevet i scss hvilket opfylder kravet om prekompileret css. Her har jeg anvendt variabler og nesting. I forhold til react er projektet blevet inddelt i komponenter, bruger hooks og flere libraries, bl.a. zustand til state management og wouter til routing.

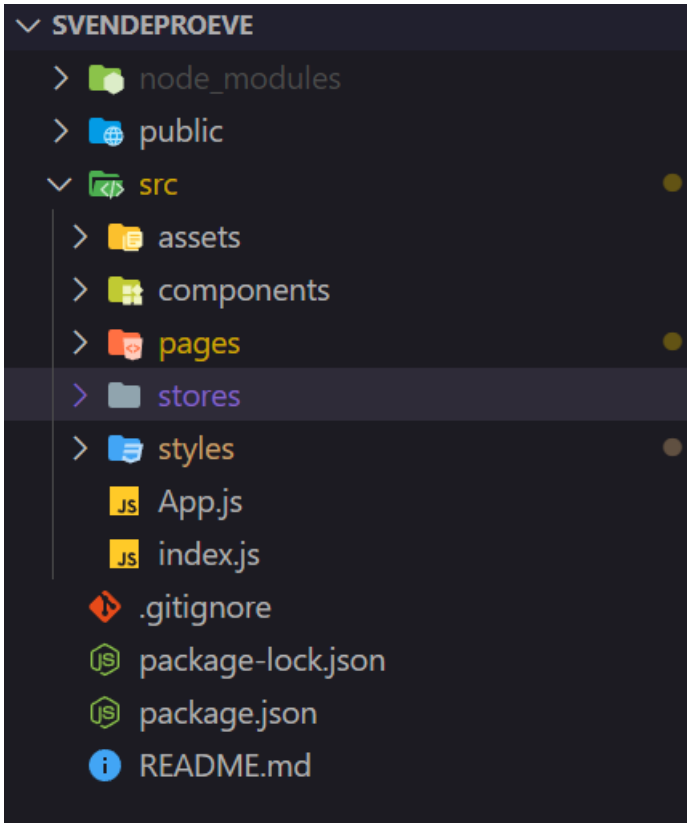
Jeg har valgt zustand til state managment da det er den løsning jeg har fundet med mindt boilerplate kode. På den måde formindsker jeg kompleksiteten af mit projekt, og har overskud over hvordan de forskellige dele virker. I samme dur har jeg valgt react wouter i stedet for den klassiske router, da jeg personligt finder den nemmere at læse og holde ren.

Tidsplan:

Min Tidsplan igennem ugen har været at bygge alle siderne færdig, med den korrekte styling, hvorefter funktionaliteten er blevet implementeret. Pga. enkelte problemer med styling og javascript logik, har nogle ting taget længere tid end de skulle, og jeg har derfor været nødt til at prioritere at lave alle sider færdige, over at alle funktionaliteter virkede 100%. Jeg synes dog jeg er noget i mål med at have lavet alle undersider, responsivitet, API kald, login og anmeldelses funktionalitet.

Redegørelse for kodeelementer

I løbet af opgaven har jeg fokuseret meget på at gøre koden overskuelig ved at dele koden så meget op i komponenter som muligt. Min fil struktur er bygget således op:



hvor nav og footer komponenter ligger i components mappen, og alle undersider har deres egne mapper med tilhørende komponenter i pages.

```
import { create } from 'zustand'

export const useActorStore = create((set) => ({
    actors: "",
    actorDetails: "",
    fetchActors: async (endpoint) => {
        const response = await fetch(endpoint, {
            method: "GET",
            headers: {
                'Content-Type': 'application/json'
            },
        })
        set({ actors: await response.json()})
    },
```

Eksemplet her viser hvordan jeg bruger zustand til state management. Actors og actorDetails variablerne bliver hentet fra API'et og opdateret i fetch metoden. Jeg har en stor for hver funktionalitet der inddrager API'et.

```
19      <Route path="/" component={Home}/>
20      <Route path="/events" component={Events}/>
21      <Route path="/actors" component={Actors}/>
22      <Route path="/login" component={Login}/>
23      <Route path="/myPage" component={MyPage}/>
24      <Route path="/actors/:id">
25        {(params) =>{
26          return <ActorDetails id={params.id}/>
27        }}
28      </Route>
29      <Route path="/events/:id">
30        {(params) =>{
31          return <EventDetails id={params.id}/>
32        }}
33      </Route>
34      <Route path="/tickets/:id">
35        {(params) =>{
36          return <BuyTicket id={params.id}/>
37        }}
```

Mine Routes ser sådan ud. På linje 34 ses et eksempel på en næstet route, som igennem params giver komponent et id med, som bruges i API kaldet.