NTNU
Kunnskap for en bedre verden

Institutt for datateknologi og informatikk

IT2810 - Webutvikling

Prosjekt 1 - Gruppe 20

Authors:
Malene Lien Killi
Aaryan Neupane
Tobias Nygård
Ingrid Tandberg Ryan

Date: 20.09.2023

Innholdsfortegnelse

1 Funkjsonelle Krav

1.1 Krav 1

Innhenter api-data og viser ressurser en og en om gangen. Søkebaren gir muligheten til å hoppe til en spesifikk ressurs, pilene gjør det lett å bla seg frem og tilbake.

1.2 Krav 2

Ved bruk av søkebaren, får en opp forslag relatert til det du skriver inn. Med denne filtreringen får du presentert den valutaen du trykker på. I tillegg presenterer favorittsiden de valutaene du har favorisert. Valgene huskes selv om siden reloades.

1.3 Krav 3

En bruker kan gjøre valg ved å trykke på favorittknappen til valutaene de har lyst å følge med på. Disse valgene presenteres på favorittsiden. Disse kan enkelt legges til og fjernes. Valgene huskes selv om siden avsluttes og startes igjen.

1.4 Krav 4

Løsningen har et responsivt design innenfor gitte rammeverk. Det innebærer tilpasning av nettstedet slik at det ser bra ut og fungerer optimalt på forskjellige enheter og skjermstørrelser. Se mer utfyllende på tekniske krav.

1.5 Krav 5

Vi har valgt et gjennomgående design, slik at nettsiden er behagelig å se på. Det er ingen unødvendig elementer, noe som gir en ryddig utforming. Vi har tatt hensyn til universell utforming, der fargene og komponentene har sterk kontrast slik at alle skal ha mulighet til å lese siden best mulig. Samtidig skal det være intuitivt hvordan man skal bruke siden.

2 Tekniske Krav

2.1 Oppsett

Vi har fulgt filen "prosjektopprettelse" filen på blackboard for å opprette prosjektet og har basert den på Typescript og React.

2.2 React state og props

De fleste filene har benyttet React state og props.

2.3 Innhenting av informasjon fra REST API

REST API'et vi har benyttet i dette prosjektet heter "Exchange Rates API" og sender ut vekslingskurset mellom en euro og 170 forskjellige valutaer. Api'et er aktivt og oppdaterer verdiene på de forskjellige kursene veldig jevnt. Det vil si at vi er nødt til å oppdatere api-dataen vi innhenter

jevnlig for å kunne vise sann informasjon til brukerne. Løsningen vi har valgt å bruke involverer funksjonalitetene QueryClient, PersistQueryClientProvider og createSyncStoragePersister fra TanStack Query. QueryClient bruker vi i index filen for å opprette en queryclient for hele nettsiden, deretter bruker vi createSyncStoragePersister og PersistQueryClientProvider for å lagre og bruke innhentet informasjon i localstorage og hindre ekstra kall på api'et når en bruker navigerer på samme side.

For å hente inn data fra api'et har vi en egen api fil med informasjon om hvor man finner dataen og eksporterer dette i en async funksjon for å forsikre at innhentingen skjer i riktig rekkefølge. Denne fetchData funksjonen brukes da i både hjemme- og favorittsiden som query-funksjonen i UseQuery for innhenting av data fra api'et.

Til slutt er det verdt å nevne at vi har valgt at api'et skal kalles når brukeren først går innpå siden og hver gang den navigerer mellom de to sidene våre, samt når de kommer tilbake til siden fra andre applikasjoner og trykker på våre sider. Dette er for å forsørge at vekslingsdata'en vi tilbyr alltid er oppdatert i henhold til api'et i både hjemme- og favorittsiden, og om brukeren går tilbake nettsiden etter å brukt andre applikasjoner. Lagring av vekslingsdataen i localstorage for å vise på "mine favoritter" hadde medført gammel vekslingsrate data og dette ville vi unngå. Derfor valgte vi en løsning som lagret valuta navnet og en favoritt state på hver valuta som er favorisert i localstorage og deretter hentet inn oppdatert data fra api'et og filtrert på valuta navnet for å tilfredsstille behovet av ny data på favorittsiden.

2.4 Web- og localstorage

Ettersom vi vil lagre og huske valutaene brukeren har trykket favoritt på, har vi valgt å benytte localstorage. I HomepageCurrency filen, har vi benyttet en useEffect hook for innhenting av lagret informasjon om de bruker-valgte valutaene i localstoragen og implementert funksjonalitet for at favoritt knappen lagrer valutaen i localstorage om staten er true.

Sessionstorage har vi benyttet for å beholde søk-queryen uansett om man navigerer til forskjellige faner på nettsiden.

2.5 React Router DOM

I prosjektet benytter vi React Router DOM for å kunne navigere mellom de to sidene våre. I appen bruker vi Route og BrowserRouter til å opprette pathways og vise de to sidene våre som komponenter. I headeren benytter vi useNavigate til å kjapt bytte mellom sider ved å trykke på knapper og useLocation for å lettere implementere betingelser for knappen visningstilstand avhengig av hvilken pathway vi er på.

2.6 Krav 6

Vi oppnådde dette ved hjelp av CSS Media Queries, som tillater tilpasning av stil og layout avhengig av enhetens egenskaper. Da anvendte vi stort sett prosentandeler i stedet for faste piksler for å definere elementers bredde og plassering, som gjør at nettsiden kan tilpasse seg ulike skjermstørrelser. For å forhindre at de strekkes utover sin naturlige størrelse på mindre skjermer brukte vi 'max-width: 100Vi har tilpasset de enhetene vi har til rådighet som er fra iPhone12 til iPad air til Macbook. Foreløpig støtter vi ikke komponenter som er større enn 1500px, dette er noe vi skal gjøre i en senere anledning.

3 Testing

Testingen er basert på Vitest. Vi har testet at komponentene rendrer riktig innhold. Headerkomponenten er testet for navigering mellom hjemmesiden og favourite-siden. I HomepageCurrency

har vi testet at favourite-knappen fungerer som den skal, altså å legge til valutaen, med verdi, til local storage. Her har vi også testet dens "state" i local storage, altså at variabelen isFavourite er satt til true. FavouritepageCurrency er testet ved å se om knappen i komponenten fungerer. I tillegg til dette har vi tatt en Snapshot test av HomePage og FavouritePage.